Université Mohammed V-Agdal
Faculté des Sciences
Département d'informatique

Cours La Programmation Web et J2EE : Licence SMI -S6 (Accréditation : 2009-2013)

Pr. EL BENANI

Année de production : 2013

Introduction à JEE 6

2013/2014

Département d'Informatique

elbenani@hotmail.com

elbenani@fsr.ac.ma

Introduction à JEE 6

- Rappel de Java
- Threads
- Applets
- Rappel de JEE 6
- Servlets
- JSP
- JDBC

Rappel de Java

Caractéristiques Principales de Java

Le langage Java est :

- « C-like » : Syntaxe familière aux programmeurs de C
- Orienté objet : Tout est objet, sauf les types primitifs (entiers, flottants, booléens, ...)
- Robuste : Typage fort, pas de pointeurs, etc.
- Code intermédiaire : Le compilateur ne produit que du bytecode indépendant de l'architecture de la machine où a été compilé le code source

"Write Once, Run Anywhere : écrire une fois, exécuter partout"

Quelques chiffres (2011)

- 97% des machines d'entreprises ont une JVM installée
- Java est téléchargé plus d'un milliards de fois chaque année
- Il y a plus de 9 millions de développeurs Java dans le monde
- Java est un des langages les plus utilisé dans le monde
- Tous les lecteurs de Blue-Ray utilisent Java
- Plus de 3 milliards d'appareils mobiles peuvent utiliser Java
- Plus de 1,4 milliards de cartes à puce utilisant Java sont produites chaque année

Editions Java 2

		Foundation Profile	Personal Profile	RMI Profile	...	PDA Profile	MID Profile	...	GSM Pr.	OP Pr.
J2EE	J2SE	CDC				CLDC			JavaCard	
		J2ME								
HotSpot VM	JVM	CVM				KVM			JavaCard VM	

10MB 1MB 512KB 32KB 1KB

µP 32 64 bits µP 32bits µP 16 32 bits µP 8 16 (32) bits

Didier Donsez, 2000-2008, Programmation J2ME

Comment obtenir JAVA

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Documentation:
http://docs.oracle.com/javase/7/docs/api/index.html

Java Development Kit de Sun

Le compilateur javac
La syntaxe est la suivante : javac [options] [fichiers] [@fichiers]

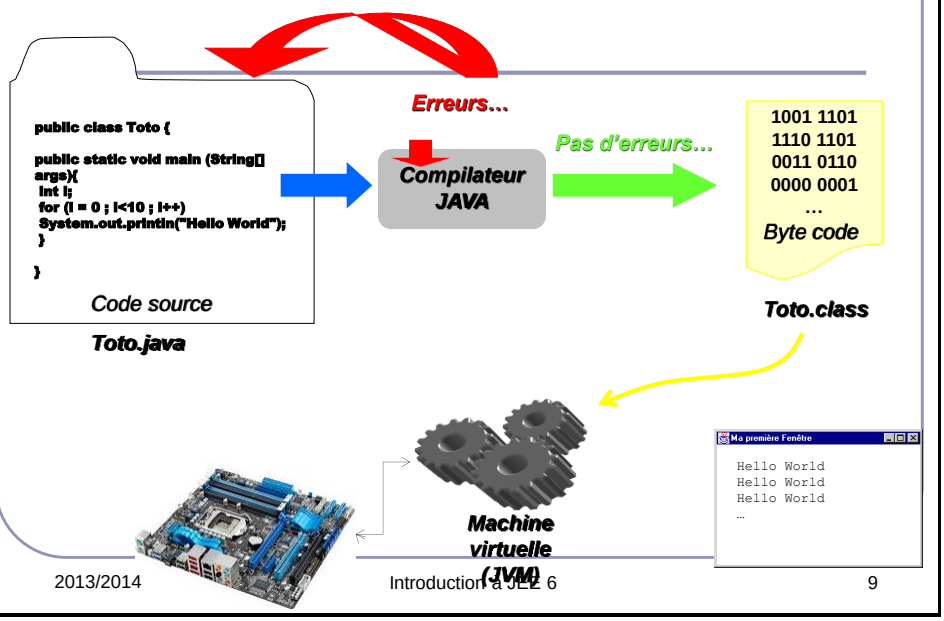
L'interpréteur java/javaw
Ces deux outils sont les interpréteurs de byte code : ils lancent le JRE, chargent les classes nécessaires et executent la méthode main de la classe.
java ouvre une console pour recevoir les messages de l'application alors que javaw n'en ouvre pas.

L'outil JAR
JAR est le diminutif de Java ARchive. C'est un format de fichier qui permet de regrouper des fichiers contenant du byte-code Java (fichier .class) ou des données utilisées en temps que ressources (images, son, ...). Ce format est compatible avec le format ZIP : les fichiers contenus dans un jar sont compressés de façon indépendante du système d'exploitation.

Pour tester les applets : l'outil appletviewer
Cet outil permet de tester une applet. L'intérêt de cet outil est qu'il permet de tester une applet avec la version courante du JDK. Un navigateur classique nécessite un plug-in pour utiliser une version particulière du JRE.

Pour générer la documentation : l'outil javadoc
Cet outil permet de générer une documentation à partir des données insérées dans le code source.

Exécution du code JAVA



Environnements de développement

Editeur de texte + lignes de commande
Les éditeurs simples : Jcreator, Emacs, JEdit
Les éditeurs avancés

- o Eclipse http://www.eclipse.org/
- o NetBeans http://www.netbeans.org/
- o IntelliJ IDEA
- o RAD
- o JDeveloper
- o JBuilder
- o BlueJ

La méthode main

- La méthode main est une méthode de classe publique, qui contient le « programme principal » à exécuter et qui a pour signature :

```
public static void main(String[] args)
```

- **Attention**
- La méthode main ne peut pas retourner d'entier comme en C.

```
public static int main(String[] args)
```


La méthode main

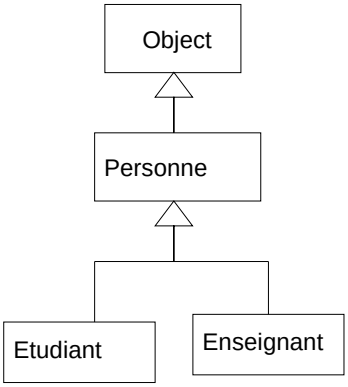
```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int age;
        Scanner sc = new Scanner(System.in);
        System.out.print("Saisissez votre âge : ");
        age = sc.nextInt();
        if (age > 20)
            System.out.println("vous êtes agé");
        else if (age > 0)
            System.out.println("vous êtes jeune");
        else
            System.out.println("vous êtes en devenir");
    }
}
```

Concepts POO

Héritage (Polymorphisme, Interface, Class abstraite, relation <<instance de>>, casting)
Visibilité et encapsulation
Attributs et Méthodes statiques

Héritage

- L'héritage est une relation entre deux classes.
- Ex. Etudiant hérite de Personne:
  - La classe étudiant et un sous-type (***sous-classe***) de la classe Personne (***superclasse***).
- Syntaxe: mot clé "extends"
  - public class Etudiant extends Personne { … }
- En Java, une classe ne peut pas hériter de plus d'une classe.
- Si l'héritage n'est pas déclaré, la classe hérite implicitement de **`java.lang.Object`**



une hiérarchie de classes

Sémantique d'héritage: attributs et méthodes

- La sous classe (ex. Etudiant) possède des attributs et des méthodes définis dans la superclasse (ex. Personne).
- Ex.

```
public class Personne {
  String nom;

  public void envoyerMail(
    String adr, String message)
     {  ….}

}
```

```
public class Etudiant extends Personne {
  String noEtudiant;

  public Etudiant(String _nom, _noEtudiant)   {
     nom = _nom;
     noEtudiant = _noEtudiant;
     envoyerMail("admin@fsr.ac.ma",
       "creation de dossier : " + nom);
    }
}
```

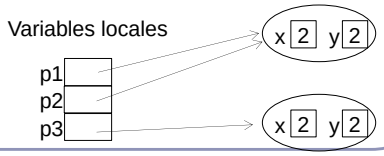

La classe Objet

- Classe mère de toutes les classes.
- Possède des méthodes de base qu'il est possible de redéfinir :
  - **toString()**
  - **equals() & hashCode()**
  - **getClass()**
  - **clone()**
  - **finalize()**

Tests d'égalité

- Les opérateurs de comparaison **==** et **!=** tests les valeurs des variables :
  - Pour les types primitifs, on test leurs valeurs
  - Pour les types objets, on test les valeurs de leurs références

```
Point p1;
p1=new Point(2,2);
Point p2;
p2=p1
Point p3;
p3=new Point(2,2);
p1==p2; // true
p1==p3; // false
p2==p3; // false
```



La méthode equals()

- Il existe déja une méthode equals(Object) dans Object
- Mais son implémentation test les références

```
Point p1=new Point(2,2);
Point p3=new Point(2,2);

p1==p3;         // false
p1.equals(p3); // false
```

- Pour comparer structurellement deux objet, il faut changer (on dit **redéfinir**) le code de la méthode equals()

constructeur

- Chaque classe a un ou plusieurs constructeurs
  - qui servent à créer les instances
  - initialiser l'état de ces instances
- Un constructeur :
  - a le même nom que la classe
  - n'a pas de type de retour
  - peut disposer d'un nombre quelconque d'arguments (éventuellement aucun).

Constructeur : exemple

```
public class Employe {
    private String nom, prenom;
    private double salaire;
    // Constructeur
    public Employe(String n, String p) {
            nom = n;
            prenom = p;
    }
    . . .
    public static void main(String[] args) {
            Employe e1;
            e1 = new Employe("Dupond", "Pierre");
            e1.setSalaire(12000);
            . . .
    }
}
```

Création d'une instance de Employe

Plusieurs constructeur : surcharge

```
public class Employe {
      private String nom, prenom;
      private double salaire;
      // 2 Constructeurs
      public Employe(String n, String p) {
                  nom = n;
                  prenom = p;
      }
      public Employe(String n, String p, double s) {
                  nom = n;
                  prenom = p;
                  salaire = s;
      }
      . . .
      e1 = new Employe("Dupond", "Pierre");
      e2 = new Employe("Durand", "Jacques", 15000);
```


Constructeur privé

- **Exemple 1:**

```
class ClasseA{
    private ClasseA() { … } // constructeur privée sans arguments
…}
ClasseA  a = new ClasseA(); // erreur,  car  ClasseA() est privé
```

- **Exemple 2:**

```
public class CacheSingleton {
  private static CacheSingleton instance = new CacheSingleton();
  private  Map<Long, Object> cache = new HashMap<Long, Object>();
  private  CacheSingleton() {  }
  public  static  synchronized  CacheSingleton  getInstance() {return
   instance; }
…
```

Les constructeurs en héritage

- La première instruction d'un constructeur peut être un appel :
  - à un constructeur de la classe mère : **super(...)**
  - ou à un autre constructeur de la classe : **this(...)**
- Interdit de placer **this()** ou **super()** ailleurs qu'en première instruction d'un constructeur
- Il n'est pas possible d'utiliser à la fois un autre constructeur de la classe et un constructeur de sa classe mère dans la définition d'un de ses constructeurs.

```
public class A {
  public A(int x) {
    super();
    this();
  }
}
```


Constructeur de la classe mère

```
public class Rectangle {
  private int x, y, largeur, hauteur;

  public Rectangle(int x, int y,
                   int largeur, int hauteur) {
    this.x = x;
    this.y = y;
    this.largeur = largeur;
    this.longueur = longueur;
  }
  . . .
}
```

Constructeurs de la classe fille

```
public class RectangleColore extends Rectangle {
  private Color couleur;
  public RectangleColore(int x, int y,
                         int largeur, int hauteur
                         Color couleur) {
    super(x, y, largeur, hauteur);
    this.couleur = couleur;
  }
  public RectangleColore(int x, int y,
                         int largeur, int hauteur) {
    this(x, y, largeur, hauteur, Color.black);
  }
  . . .
}
```


Appel implicite du constructeur de la classe mère

- Si la première instruction d'un constructeur n'est ni **super(...)**, ni **this(...)**, le compilateur ajoute au début un appel implicite au constructeur sans paramètre de la classe mère (**super()**) .
- Un constructeur de la classe mère est toujours exécuté avant les autres instructions du constructeur.
- La première instruction d'un constructeur de la classe mère est l'appel à un constructeur de la classe « grand-mère », et ainsi de suite...
- La première instruction exécutée par un constructeur est le constructeur (sans paramètre) de la classe **Object** !

Constructeur implicite : exemple 1

```
public class A {
  public A () {System.out.println("A");}}

public class  B extends A {
  public B () {System.out.println("B");}}
```

Le ligne de code suivante :

```
  B c = new B ();
```

produira l'affichage suivant :

```
java MonProg
  A
  B
```


Constructeurs : exemple 2

```
import java.awt.*; // pour classe Point
public class Cercle {
  // Constante
  public static final double PI = 3.14;
  // Variables
  private Point centre;
  private int rayon;
  // Constructeur
  public Cercle(Point c, int r) {
    centre = c;
    rayon = r;
  }
```

Plus de constructeur sans paramètre par défaut !

Appel implicite du constructeur `Object()`

Constructeurs : exemple 2

```
  // Méthodes
  public double surface() {
    return PI * rayon * rayon;
  }
  public Point getCentre() {
    return centre;
  }
  public static void main(String[] args) {
    Point p = new Point(1, 2);
    Cercle c = new Cercle(p, 5);
    System.out.println("Surface du cercle:"
                       + c.surface());
  }
}
```


Constructeurs : exemple 2

```
public class CercleColore extends Cercle {
  private String couleur;
  public CercleColore(Point p, int r, String c) {
    super(p, r);
    couleur = c;
  }
  public void setCouleur(String c) {
    couleur = c;
  }
  public String getCouleur() {
    return couleur;
  }
}
```

Que se passe-t-il si on enlève cette instruction ?

Constructeur implicite : erreur

Etape 1 :
Jusqu'ici tout va bien.

```
public class A {
  public int x;
}

public class  B extends A {
  public int y;
  public B (int x,int y) {
    this.x = x;
    this.y = y;
  }
}
```

Etape 2 :
Puis, on ajoute un constructeur.

```
public class A {
  public int x;
  public A (int x) {
   this.x =x;
  }
}

public class  B extends A {
  public int y;
  public B (int x,int y) {
    this.x = x;
    this.y = y;
  }
}
```


Constructeur implicite : correction

Solution 1 :
Ajout d'un constructeur vide.

```
public class A {
  public int x;
  public A () {}
  public A (int x) {
    this.x = x;
  }
}

public class  B extends A {
  public int y;
  public B (int x,int y) {
    this.x = x;
    this.y = y;
  }
}
```

Solution 2 :
Appel explicite au **super(int)**.

```
public class A {
  public int x;
  public A (int x) {
    this.x = x;
  }
}

public class B extends A {
  public int y;
  public B (int x,int y) {
    super(x);
    this.y = y;
  }
}
```

Redéfinir equals()

- Pourquoi equals ?
Car elle sert à cela.
- La plupart des classes de l'API redéfinissent la méthode equals

```
public class Point {
  private final int x,y;
  public Point(int x,int y) {
    this.x=x;
    this.y=y;
  }
  public boolean equals(Point p) {
    return x==p.x && y==p.y;
  }
} // est ce que c'est bon ?
```


Redéfinir equals()

```
Point p1=new Point(2,2);
Point p3=new Point(2,2);
p1==p3;         // false
p1.equals(p3); // true

ArrayList points=new ArrayList();
points.add(p1);
points.contains(p3); // false
// pourtant contains utilise Object.equals(Object) ??
```

- La VM ne fait pas la liaison entre Object.equals(Object) et Point.equals(Point)

Redéfinir equals()

- Ce n'est pas le même equals(), car il n'y a pas eut de redéfinition mais une autre définition (on dit **surcharge**)

```
Point p1=new Point(2,2);
Point p3=new Point(2,2);
p1.equals(p3); // true : Point.equals(Point)

Object o1=p1;
Object o3=p3;
o1.equals(o3); // false : Object.equals(Object)
```

- Point possède deux méthodes equals (equals(Object) et equals(Point))

Redéfinir equals()

- Il faut définir equals() dans Point de telle façon qu'elle remplace equals de Object
- Pour cela equals doit avoir la meme signature que equals(Object) de Object

```
public class Point {
  private final int x,y;
  public Point(int x,int y) {
    this.x=x;
    this.y=y;
  }
  public boolean equals(Object p) {
    return x==p.x && y==p.y; // ce code ne marche pas
  }
}
```

Utiliser @Override

- @Override est une annotation qui demande au compilateur de vérifier que l'on redéfinie bien une méthode

```
public class Point {
  private final int x,y;
  public Point(int x,int y) {
    this.x=x;
    this.y=y;
  }
  @Override public boolean equals(Point p) {
    ...
  } // the method equals(Point p) in Point must override
    // a superclass method
}
```


Redéfinir equals

- On demande dynamiquement à voir une référence à Object comme à Point
(car on le sait que c'est un Point)

```
public class Point {
  ...
  @Override public boolean equals(Object o) {
    Point p=(Point)o; // ClassCastException si pas un Point
    return x==p.x && y==p.y;
  }
}
```

- Mais equals doit renvoyer false si o n'est pas un Point et pas lever une ClassCastException

Redéfinir equals

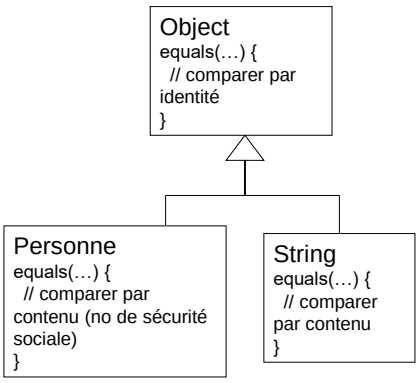
- On utilise instanceof qui renvoie vrai si une référence est d'un type particulier

```
public class Point {
  ...
  @Override public boolean equals(Object o) {
    if (!(o instanceof Point)) // marche aussi avec null
      return false;
    Point p=(Point)o;
    return x==p.x && y==p.y;
  }
}
```

- **null instanceof** WhatYouWant renvoie false

Exemple: redéfinition de la méthode equals(...)

```
public Personne {
  String noSecu;
  …
  public boolean equals(Object autre)
{
    if(autre instanceof Personne) {
       Personne p2 = (Personne) autre;
       if(noSecu.equals(p2.noSecu))
            return true;
    }
    return false;
  }
}
```



Accès aux méthodes redéfinies dans la superclasse

- Même si la sous classe redéfinit des méthodes de la superclasse, elle a le moyen d'accéder à ces méthodes en utilisant le mot clé <<super>>.

```
public class A {
  public A(String s)  {
     System.out.println("A : " + s);
  }
  public void m1() {
      System.out.println("A.m1");
  }
  public void m2() {
      System.out.println("A.m2");
  }
}
```

```
public class B extends A {
  public B(String s)  {
     super(s);
     System.out.println("B : " + s);
  }
  public void m1() {
      super.m1();
      m2(); // pas besoin de faire super.m2();
      System.out.println("B.m1");
  }
}
```

```
// main
B b= new B("hello");
b.m1();
```

Résultat :
A : hello
B : hello
A.m1
A.m2
B.m1

Modificateur final

- **final** appliquée à une classe, indique qu'on ne peut pas créer de classes dérivées pour cette classe.
- **final** appliqué à un champ, indique que la valeur de ce champ ne peut pas être modifiée après l'affectation initiale. Permet de définir des valeurs constantes.
- **final** appliqué à une méthode, indique que la méthode ne peut pas être redéfinie dans une sous-classe.
- Pour les paramètres d'une méthode, **final** indique que la valeur de ces paramètres ne peut pas être modifiée.
- **Remarque :** final permet au compilateur d'effectuer certaines optimisations.

super et this

- Soit une classe **B** qui hérite d'une classe **A**
- Dans une méthode d'instance **m** de **B**, « **super.** » sert à désigner un membre de **A**
- En particulier, **super.m(…)** désigne la méthode **m** de **A** qui est en train d'être redéfinie dans **B** :

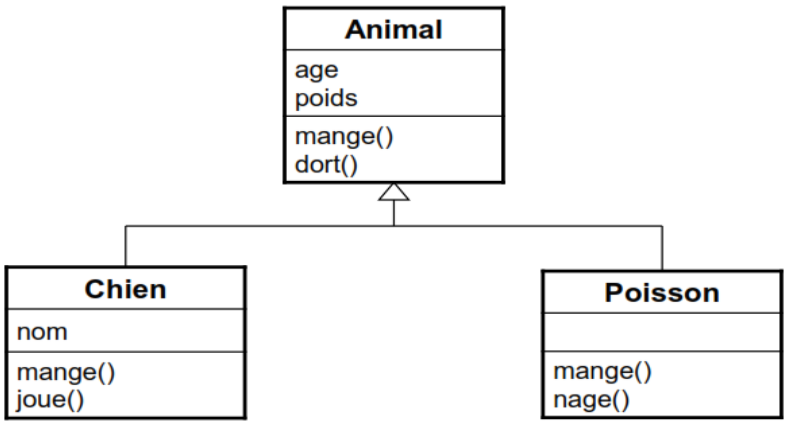
```
@Override
public int m(int i) {
  return 500 + super.m(i);
}
```


super

- une méthode **static,** ne peut être redéfinie.
- **super.m()** ne doit pas exister pour une méthode **static m()** ;
- On ne peut remonter plus haut que la classe mère pour récupérer une méthode redéfinie :
  - pas de *cast* « **(ClasseAncetre)m()** »
  - pas de « **super.super.m()** »

Polymorphisme

- Si l'on considère le diagramme de classes suivant :



Polymorphisme

- Définition : la capacité de choisir dynamiquement la méthode qui correspond au type réel de l'objet.
- Si la classe Poisson hérite de la classe Animal
  - classe Poisson "EST-UN" classe Animal
  - toute méthode m de Animal peut-être invoquée sur une instance de la classe poisson (héritage)
- Le polymorphisme consiste à exploiter cela en fournissant une sous-classe dans les expressions "qui attendent" une classe mère.
- une fille doit pouvoir faire tout ce que fait sa mère ;

Polymorphisme

```
Animal anim1 = new Poisson( ) ; //un poisson est un animal
Animal anim2 = new Chien() ; //un chien est aussi un animal

System.out.println(anim1.mange());
System.out.println(anim2.mange());

// un animal n'est pas nécessairement un poisson
Poisson p=anim1;
```


Polymorphisme

- Quelle méthode mange() sera invoquée dans ce cas ?
- C'est toujours la méthode mange() définie dans la sous-classe qui sera invoquée.
- Même si anim1 est une référence de type Animal, c'est le type de l'objet référencé qui détermine la méthode qui sera appelée.
- En Java, lorsqu'il y a héritage, la détermination de la méthode à invoquer n'est pas effectuée lors de la compilation.

Polymorphisme

- C'est seulement à l'exécution que la machine virtuelle déterminera la méthode à invoquer selon le type effectif de l'objet référencé à ce moment là.
- Ce mécanisme s'appelle ***"Recherche dynamique de méthode"*** (***Late Binding ou Dynamic Binding).***
- Ce mécanisme de recherche dynamique (durant l'exécution de l'application) sert de base à la mise en œuvre de la propriété appelée **polymorphisme.**

Polymorphisme

- On pourrait définir le polymorphisme comme la propriété permettant à un programme de réagir différemment à l'envoi d'un même message (invocation de méthode) en fonction des objets qui reçoivent ce message.
- Il s'agit donc d'une adaptation dynamique du comportement selon les objets en présence.
- Avec l'encapsulation et l'héritage, le polymorphisme est une des propriétés essentielles de la programmation orientée objet.

Polymorphisme

```
// Déclaration et création du tableau
Animal[ ] menagerie = new Animal[6];

// Alimentation du tableau
menagerie[0] = new Poisson(...);
menagerie[1] = new Chien(...);
menagerie[2] = new Chien(...);
menagerie[3] = new Animal(...);
menagerie[4] = new Poisson(...);
menagerie[5] = new Chien(...);
```


Polymorphisme

```
//-----------------------------------------------------
//  Appelle la méthode  mange() pour chaque animal contenu
//  dans le tableau passé en paramètre
//-----------------------------------------------------
public static void nourrir(Animal[] tabAnimaux) {

  if (tabAnimaux == null) return;

  for (int i=0; i<tabAnimaux.length; i++) {
    if (tabAnimaux[i] != null) {
      tabAnimaux[i].mange();
    }
  }
}
```

Upcasting : classe fille → classe mère

- surclassement ou upcasting est le fait d'enregistrer une référence d'une instance d'une classe B héritant d'une classe A dans une variable de type A.
- En java, cette opération est implicite et constitue la base du polymorphisme.

```
public class A { ... }

public class B extends A { ... }

A a = new B() // C'est de l'upcasting (surclassement).
```


Downcasting : classe mère → classe fille

- déclassement ou downcasting est le fait de convertir une référence « surclassée » pour «libérer» certaines fonctionnalités cachées par le surclassement.
- En java, cette conversion n'est pas implicite, elle doit être forcée par l'opérateur de cast : **(<nomClasse>)**.

```
public class A { ... }

public class B extends A { ... }

A a = new B(); // surclassement, upcasting
B b = (B) a; // downcasting
```

Downcasting

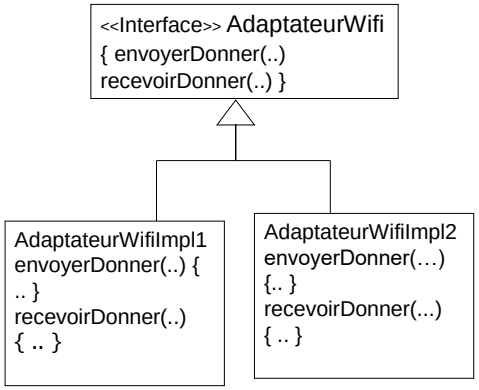
- Attention : Le downcasting ne permet pas de convertir une instance d'une classe donnée en une instance d'une sous-classe !

```
class A { A() {}}

class B extends A { B() {}}

A a = new A();
B b = (B) a ;
```

Ça compile, mais ça plantera à l'exécution :

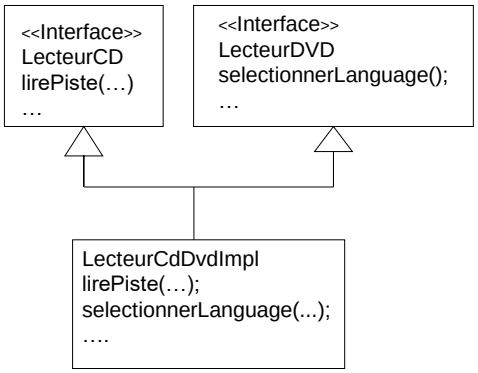
Interface: type abstrait

- Une interface est un type abstrait, censé être implémenté par des classes.
- Il ne contient que les méthodes sans corps.
- Ces méthodes sont à redéfinir dans les classes d'implémentation
- On ne peut pas créer une instance d'une interface mais on peut créer une instance d'une classe d'implémentation.



Interface: héritage multiple

- Une classe peut hériter de plusieurs interfaces.
  - Chaque interface représente un rôle d'une classe.
  - Ex. un Lecteur CD-DVD a comme rôles LecteurCD et LecteurDVD



Interface: syntaxe

```
interface nom_de_l'interface [ extends noms d'autres interfaces ] {
   public void methode1(String param1) ;
   public int methode2(int param1, int param2) ;

   // autres méthodes …
}
```

- Le mot clé "extends" permet à une interface d'hériter d'autres interfaces (une ou plusieurs)

Implémentation d'une interface

- Une classe implémentant une interface doit redéfinir toutes les méthodes déclarées dans l'interface.

```
public class LecteurCdDvdImpl
  implements LecteurCD, LecteurDVD  {

  public void lirePiste(int noPiste) {
     ….   // à ¨spécifier
  }
  public void selectionnerLangue(String lang) {
     ….  // à spécifier
  }

}
```


Les classes abstraites

- Une classe abstraite est incomplète
  - Elle ne peut être utilisée telle quelle.
  - Elle est censée être spécifiée par des sous classes.
  - On ne peut pas créer une instance d'une classe abstraite.
- Une classe abstraite peut avoir des méthodes vides (à redéfinir par des sous-classes).

Exemple d'une classe abstraite

```
public abstract Fenetre {
  // attributs génériques pour tout type de fenêtres.
  int posX;  int posY;
  int dimX; int dimY;
  // Les méthodes dont l'implémentation est générique
  // (pour tout type de fenêtres).
  public void deplacer() { … }
  public void redimensionner(int newDimX, int newDimY)  {
    dimX = newDimX; dimY = newDimY;
    rafraichirContenu();
   }
  // la méthode dont on ne connaît pas encore l'implémentation
  // (cela dépend de type de fenêtre). Ainsi on la laisse vide.
  public abstract rafraichirContenu();
}
```


Visibilité: modificateur public/private

- Le modificateur **public** marque que les attributs/ méthodes d'une classe sont accessibles partout (dans les classes).
- Le modificateur **private** marque que les attributs/ méthodes d'une classe ne sont accessibles que dans cette classe.

```
class C1 {
  public String a1;
  private String a2;
  public void m1();
  private void m2();
}
```

```
class C2 {
  void m3() {
    C1 c1 = new C1();
    c1.a1 = "hello";  // OK
    // c1.a2 = "hello";  illégal
    c1.m1();  // OK
    // c1.m2();  illégal
  }
}
```

Visibilité : autre modificateurs de visibilité

- Les autres modificateurs sont:
  - aucun modificateur (par défaut): les attributs/ méthodes sont accessibles
    - par les classes du même package
  - **protected**: les attributs/ méthodes sont accessibles
    - par les classes du même package, et
    - par les sous-classes de la classe courante.

package p1;

```
public class A {
   int a1;
   protected int a2;
}
```

```
public class B {
   // a1, a2 accessible
}
```

package p2;

```
public class C {
   // a1, a2 non accessible
}
```

```
public class D extends A {
  // a1 non accessible
  // a2 accessible
}
```


Encapsulation : principe

```
// pas d'encapsulation
public class Date {
  public int jour;
  public int mois;
  public int annee;
}
```

```
// modification illégale
Date d1 = new Date();
d1.jour = 23;
d1.mois = 2;
d1.annee = 2011;
```

- Principe d'encapsulation :
  - Une classe devrait cacher des attributs et exposer ses méthodes au monde extérieur (aux autres classes)
    - Protéger le contenu (les attributs) de la modification illégale
    - Masquer les détails internes de la classe
      → simplification

Encapsulation: Protection du contenu

```
public class Date {
  private int jour;
  private int mois;
  private int annee;
  public Date() {
     … // initialisé à la date aujourd'hui
  }
  public boolean setDate(int j, int m, int a) {
     if( estDateValide(j, m, a) ) {
       jour =j;   moi=m;  annee=a;  return true;
     } else {     return false;  }
  }
  …
}
```

- Seule la modification valide est permise

Encapsulation: Masque de détails internes

```
public class Image {
  // séquence d'octets au format propriétaire
  private byte[] donnees;
  public void chargerDeFichier(File ficher) {
     …  // charger et convertir au format propriétaire
  }
  public void afficher() {
     …. // interpréter le format propriétaire.
  }
}
```

- L'utilisateur n'a pas besoin de connaître le format de données pour utiliser cette classe

Membres statiques

- Le modificateur «***static***» peut s'appliquer à une méthode ou à un attribut d'une classe.
  - L'élément statique est partagé par toutes les instances de la classe.
  - Il est possible d'y accéder sans disposer d'une instance, mais directement par la classe.
- Ex : une méthode statique

```
public class Calculatrice {
  public static int valeurAbsolue(int i )
       {
       if(i < 0) return -1 * i;
       return i;
        }
}
```

```
// OK
System.out.println(
  Calculatrice.valeurAbsolue(-25) );

// pas besoin de faire cela:
Calculatrice c = new Calculatrice();
System.out.println(

c.valeurAbsolue(-25) );
```


Membres statiques: Exemple

```
public class Test {
  static int comptoir = 0; // initialiser
//lors du chargement de la classe
  int  id;
  public Test() {
     id = comptoir; // utiliser le comptoir
//comme identifiant d'objet
     comptoir ++; //incrémenter le
//comptoir à chaque instanciation
  }
  public imprimerId() {
     System.out.println(id);
  }
  public static imprimerComptoir() {
     System.out.println(comptoir);
  }
}
```

```
public static void main(String[] args) {
   Test t1 = Test();
   Test t2 = Test();
   Test t3 = Test();
   Test.imprimerComptoir();  // résultat?
   t1.imprimerId();  // résultat ?
  t2.imprimerId();  // résultat ?
  t3.imprimerId();  // résultat ?
}
```

- Les méthodes statiques ne peuvent accéder qu'aux attributs statiques.

Définition de constants

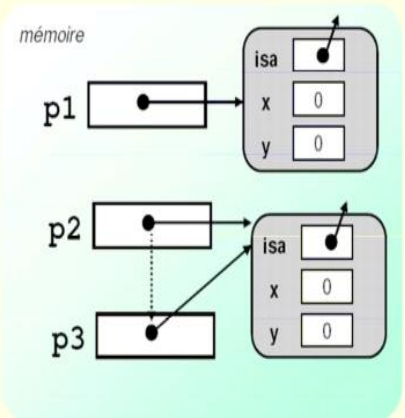
- Un constant peut être défini comme un attribut d'une classe, avec la déclaration <<static>> (partagée par toutes les instances) et <<final>> (sa valeur est non modifiable)
- Par convention le nom d'un constant est tout en majuscules.

```
public class MesConstants {
  public static final int LOAD = 0;
  public static final int SAVE = 0;
  public static final int MOVE = 2;
}
```


Gestion de la mémoire : Garbage Collector

- L'instanciation provoque une allocation dynamique de la mémoire.

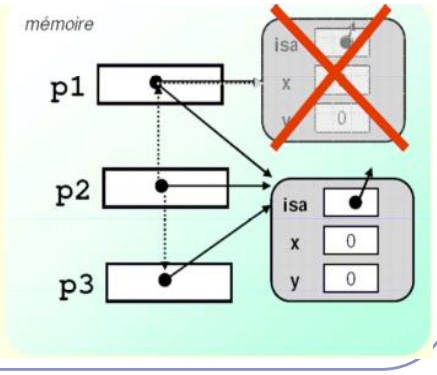
```
Point p1;
p1 = new Point();
Point p2 = new Point();
Point p3 = p2;
```



Gestion de la mémoire : Garbage Collector

- Objet non référencé : Le GC (Garbage collector) ou ramasse-miettes se charge de libérer la mémoire occupée (destruction asynchrone ou appel explicite: **System.gc())**

```
Point p1;
p1 = new Point();
Point p2 = new Point();
Point p3 = p2;
p1 = p2;
```



Garbage Collector

- Le GC (ramasse-miettes) est une tâche qui :
  - travaille en arrière plan
  - libère de la place occupée par les instances non référencées
- Il intervient
  - quand le système a besoin de mémoire
  - ou de temps en temps, avec une priorité faible

Atelier

```
public class Point{
    private int x,y;
    static int nbInstances = 0;
    final int CONSTANTE = 10;
    public Point (int x, int y){
        this.x = x; this.y = y;
        nbInstances ++;
        CONSTANTE ++;   //erreur ?
    }
    public void setX (int x){
        this.x = x;
        nbInstances ++;   // erreur ?
    }
```


Atelier

```
    public void setY (int y){
        this.y = y;
    }
    static void methodStatic (){
        nbInstances ++;
        x++; // erreur ?
    }
    public void finalize (){
        System.out.println("JE VIENS A L'INSTANT D'ETRE TUE !!!!");
    }
```

```
public static void main(String args[]){ //ligne 1
    Point p1 = new Point (0, 0); //ligne 3
    Point p2 = new Point (10, 15); //ligne 4
    Point p3; //ligne 5
    p3 = p1; //ligne 6
    System.out.println("Nbre d'instances:" + Point.nbInstances); // *2*
    System.out.println("Nbre d'instances:" + p1.nbInstances); // *2*
    System.out.println(p3 == p1); //ligne 10      // *true*
    p3.setX (100); //ligne 11
    System.out.println(p1.getX()); //ligne 12     // *100*
    p3 = null; //ligne 13
    System.gc(); //ligne 14     // Il reste une référence.
    p1 = null; //ligne 15
    System.gc(); //ligne 16     //*JE VIENS A L'INST …!*
}
}
```


Tableaux, collections et Map

Tableaux

- Un tableau est une séquence d'éléments
  - Séquence de données primitives
    - Exemple : int[], char[], byte[]
  - Séquence de références vers objets
    - Exemple : Object[], Personne[]
- On crée un tableau avec l'opérateur <<new>>
  - Exemple :
    - int[] tab1 = new int[10];
    - int dim = 20; Object[] tab2 = new Object[dim];
  - Pour un tableau d'objets, initialement ses éléments sont <<null>>
- Un tableau est lui-même un objet
- On accède aux éléments du tableau avec un index (entre 0 jusqu'à la dimension -1)
- On peut connaître la dimension du tableau par son attribut <<length>>

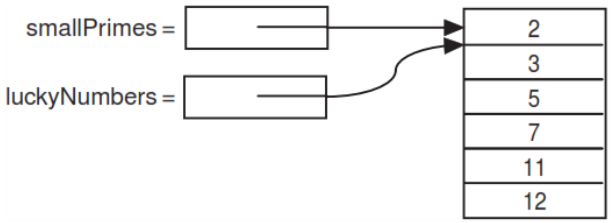
Un tableau est un objet

- On peut affecter un tableau au variable de type java.lang.Object
  - Object o = new Personne[20];
- On peut découvrir le type du tableau
  - System.out.println(o instanceof Personne[]); // résultat : true
  - System.out.println(o.getClass().isArray()); // résultat : true
  - System.out.prinln(o.getClass().getComponentType().getName());
    // résultat : Personne

Copie des tableaux

- Il est possible de copier une variable tableau dans une autre, mais les deux variables feront alors référence au même tableau :

- Exemple :
  - smallPrimes = new int[ ] { 2, 3, 5, 7, 11, 14 };
  - int[ ] luckyNumbers = smallPrimes;
  - luckyNumbers[5] = 12; // smallPrimes[5] vaut maintenant 12



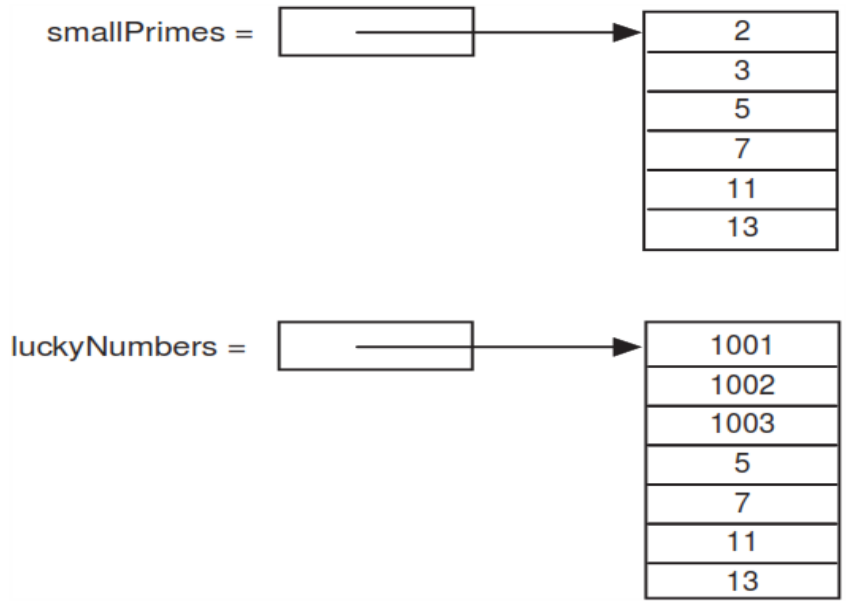
Copie des tableaux

- Pour effectivement copier toutes les valeurs d'un tableau dans un autre, il faut employer la méthode copyOf de la classe Arrays.
- Exemple :
  - Int [ ] copiedLuckyNumbers = Arrays.copyOf(luckyNumbers, luckyNumbers.length);
- Le second paramètre correspond à la longueur du nouveau tableau.
- Cette méthode est souvent employée pour augmenter la taille d'un tableau : luckyNumbers = Arrays.copyOf(luckyNumbers, 2 * luckyNumbers.length);
- Les autres éléments sont remplis de 0 si le tableau contient des nombres, false si le tableau contient des valeurs booléennes.

Copie des Tableaux

- Avant Java SE 6, la méthode arraycopy de la classe System servait à copier des éléments d'un tableau dans un autre.
- Sa syntaxe est la suivante :

  System.arraycopy(from, fromIndex, to, toIndex, count);
- Il s'agit de copier count éléments en commençant à partir de la position fromIndex du tableau from vers la position toIndex du tableau to
- Le tableau to doit disposer de suffisamment d'espace pour contenir les éléments copiés.
- Exemple :
  - int[] smallPrimes = {2, 3, 5, 7, 11, 13};
  - int[] luckyNumbers = {1001, 1002, 1003, 1004, 1005, 1006, 1007};
  - System.arraycopy(smallPrimes, 2, luckyNumbers, 3, 4);
  - Résultat ? Transparent suivant :



Tri d'un tableau

```
int [ ] a = new int[10000];
. . .
Arrays.sort(a);
```

- Trie le tableau en utilisant un algorithme QuickSort adapté.
- Exemple :

```
char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e'};
char[] copyTo = new char[5];
System.arraycopy(copyFrom, 1, copyTo, 0, 5);
Arrays.sort(copyTo);
System.out.println(new String(copyTo));
```

- Résultat : aceff

Tableaux : redimension

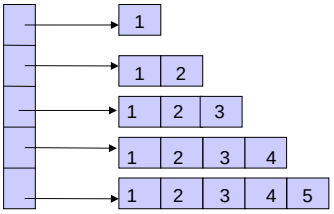
- La dimension du tableau est fixée lors de la création de l'objet (new)
- Pour redimensionner, il faut :
  - 1) Créer un nouveau tableau d'une nouvelle dimension
  - 2) Copier les éléments de l'ancien tableau vers le nouveau
    - Astuce: System.arraycopy(source, position, destination, position, nombreElements);
- Exemple :

```
public static Object[] redimentionner(Object[] tab, int nouvelleDim) {
  Object nouveauTab = new Object[nouvelleDim];
  int dimMin = (nouvelleDim> tab.length)? tab.length : nouvelleDim;
  System.arrayCopy(tab, 0, nouveauTab, 0, dimMin);
}
```

Un tableau de tableaux

- Quel est le tableau résultat de ce bloc de programme?

```
int[ ][ ] tab2dim = new int[5][ ]; // création de tableau de
//dimension 5 pour stocker des tableaux de <<int>>
for(int i=0; i<tab2dim.length; i++) {
   tab2dim[i] = new int[i+1];  // création d'un tableau des entiers
   for(int j=0; j< tab2dim[i].length; j++) {//remplir les tableaux
      tab2dim[i][j] = j+1;
   }
}
```



Opération sur les tableaux

- La classe **java.util.Arrays** définit des méthodes statiques de manipulation des tableaux :
  - equals(), hashCode(), toString()
  - deepEquals(), deepHashCode(), deepToString()
  - binarySearch(), sort(), fill()
- Pour la copie de tableau, on utilise :
  - Object.clone(),System.arraycopy(), Arrays.copyOf()

equals(), hashCode(), toString()

- Les méthodes equals(), hashCode(), toString() ne sont pas redéfinies sur les tableaux
- Arrays.equals(), Arrays.hashCode(),Arrays.toString() marchent pour Object et tous les types primitifs

```
int[] array=new int[]{2,3,5,6};
System.out.println(array);                          // [I@10b62c9
System.out.println(Arrays.toString(array));         // [2, 3, 5, 6]
System.out.println(array.hashCode());               // 17523401
System.out.println(Arrays.hashCode(array));         // 986147
int[] array2=new int[]{2,3,5,6};
System.out.println(array2.hashCode());              // 8567361
System.out.println(Arrays.hashCode(array2));        // 986147
System.out.println(array.equals(array2));           // false
System.out.println(Arrays.equals(array,array2)); // true
```


deepToString, deepEquals(), etc

- Algorithmes recursifs sur les tableaux d'objets
- Le calcul est relancé si un tableau contient lui-même un tableau etc.
- deepToString() détecte les circularités

```
Object[] array3=new Object[]{2,3,null};
array3[2]=array3;
System.out.println(Arrays.deepToString(array3));
// [2, 3, [...]]]
```

binarySearch, sort, fill

- Dichotomie (le tableau doit être trié)

```
binarySearch(byte[] a, byte key)
...
binarySearch(Object[] a, Object key)
<T> binarySearch(T[] a, T key, Comparator<? super T> c)
```

- Tri

```
sort(byte[] a)
sort(byte[] a, int fromIndex, int toIndex)
...
<T> sort(T[] a, Comparator<? super T> c)
<T> sort(T[] a, int fromIndex, int toIndex, Comparator<? super T> c)
```

- Remplissage

```
fill(boolean[] a, boolean val)
fill(boolean[] a, int fromIndex, int toIndex, boolean val)
...
fill(Object[] a, Object val)
fill(Object[] a, int fromIndex, int toIndex, Object val)
```


Ordre naturel

- Une classe peut spécifier un ordre naturel en implantant l'interface Comparable<T>

```
public interface Comparable<T>
{
   int  compareTo(T t);
}
```

- T doit être la classe spécifiant l'ordre

- Valeur de retour de **compareTo(T t)** :

<0 si this est **inférieur** à t
==0 si this est **égal** à t
>0 si this est **supérieur** à t

compareTo et equals

- L'implémentation de **compareTo** doit être compatible avec celle d'**equals** !!
- Si o1.equals(o2)==true alors
  o1.compareTo(o2)==0 (et vice versa)

```
public MyPoint implements Comparable<MyPoint> {
  public MyPoint(int x,int y) {
    this.x=x;
    this.y=y;
  }
  public int compareTo(MyPoint p) {
    int dx=x-p.x;
    if (dx!=0)
      return dx;
    return y-p.y;
  }
  private final int x,y;
```


Exemple

- Il faut donc redéfinir aussi **equals**

```
public MyPoint implements Comparable<MyPoint> {
  public MyPoint(int x,int y) {
    this.x=x;
    this.y=y;
  }
  @Override public boolean equals(Object o) {
    if (!(o instanceof MyPoint))
      return false;
    MyPoint p=(Mypoint)o;
    return x==p.x && y==p.y;
  }
  public int compareTo(MyPoint p) {
    int dx=x-p.x;
    if (dx!=0)
      return dx;
    return y-p.y;
  }
  private final int x,y;
}
```

A quoi cela sert ?

- Par exemple, java.util.Arrays.sort() demande à ce que le tableau contienne des éléments mutuellement comparables

```
public MyPoint implements Comparable<MyPoint> {
  public String toString() {
    return "("+x+','+y+')';
  }
  public static void main(String[] args) {
    MyPoint[] points=new MyPoint[] {
      new MyPoint(1,1), new MyPoint(3,3),
      new MyPoint(3,2), new MyPoint(1,9)
    };
    Arrays.sort(points);
    System.out.println(Arrays.toString(points));
    // affiche (1,1) (1,9) (3,2) (3,3)
  }
}
```


Comparaison externe

- L'interface java.util.Comparator permet de spécifier un ordre externe

```
public interface Comparator<T>
{
   int  compare(T o1, T o2);
}
```

- Un ordre externe est un ordre valable juste à un moment donné
(rien de naturel et d'évident)
- La valeur de retour de **compare** suit les mêmes règles que **compareTo**

Comparator

- L'implémentation de **compare** doit être compatible avec celle d'**equals** !!
- Si o1.equals(o2)==true alors
  compare(o1,o2)==0 (et vice versa)

```
Arrays.sort(points,new Comparator<MyPoint>() {
  public int compare(MyPoint p1,MyPoint p2) {
    int dx=x-p.x;
    int dy=y-p.y;
    return dx*dx+dy*dy;
  }
});
System.out.println(Arrays.toString(points));
// affiche (1,1) (3,2) (3,3) (1,9)
```


Comparator inverse

- Il existe deux méthodes **static** dans la classe java.util.Collections qui renvoie un comparator correspondant à :
- L'inverse de l'ordre naturel

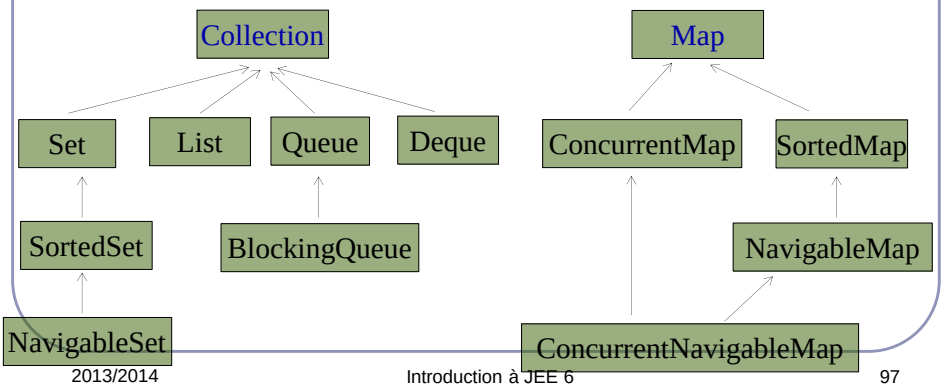
```
<T> Comparator<T> Collections.reverseOrder();
```

- L'inverse d'un ordre externe sur **T**

```
  <T> Comparator<T>
Collections.reverseOrder(Comparator<T> c);
```

L'API des collections

- 2 paquetages : java.util, java.util.concurrent
- 2 hiérarchies d'interfaces : Collection, Map



Copier une collection

- En général, les constructeurs des classes d'implémentation de collections (Vector, ArrayList, HashSet) prennent comme paramètre une collection source pour créer une copie.
  - Exemple :

```
List v = new Vector();
v.add("a");  v.add("ab");
List copie = new Vector(v);
copie.add("abc");
System.out.println(v);       // résultat : [ a, ab ]
System.out.println(copie);  // résultat : [ a, ab, abc ]
```

- Il est possible d'utiliser un tel constructeur pour convertir entre plusieurs variations de collections
  - un ensemble vers une liste : dans un ordre quelconque
  - une liste vers un ensemble : la duplication est ignorée

Conversion entre tableaux et collections

- Tableau vers collection
  - utiliser la classe java.util.Arrays et sa méthode :
    public static List asList(Object[] tab)
  - Attention la collection retournée dépond du tableau à convertir
  - Afin d'obtenir une collection indépendante du tableau, créer une copie de la liste retournée par Arrays.asList(…)
  - Exemple : Object[] tab = …; List l = new Vector(Arrays.asList(tab));
- Collection vers tableau
  - Collection c = …;
  - Object[] tab = c.toArray();
  - String[] tab2 = (String[]) c.toArray(new String[c.length])); // obtenir un //tableau d'un type spécifique

Trier une liste

- La classe java.util.Collections propose les méthodes pour trier une collection.
  - static void sort(List list) : trier une liste d'objet implémentant l'interface <<*Comparable*>>
  - static void sort(List list, Comparator c) : trier une liste d'objet en utilisant un comparateur personnalisé
- interface Comparable :
  - propose la méthode : *int compareTo(Object o);* // retourne un nombre négatif, 0, nombre positif si inférieur de, égal à, supérieur à
  - Les objets basiques (Integer, String, Byte, Char, …) implémentent Comparable
- interface Comparator :
  - propose la méthode : *int compare(Object o1, Object o2)*

Objet Class

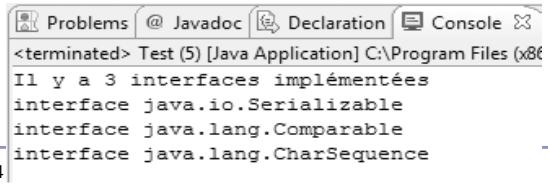
```
Class c1 = String.class;
Class c2 = new String().getClass();
System.out.println("La superclasse de la classe "+ c1.getName()+ " est : "
                   + c2.getSuperclass());
```

Affichage :

```
La superclasse de la classe java.lang.String est :
class java.lang.Object
```

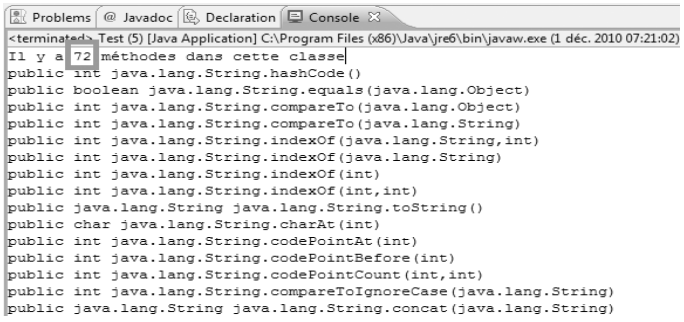
- Remarque : Object n'a pas de superclasse (null).

```
Class c =new String().getClass();  //équivalent à =String.class;
Class [ ] interfaces =c.getInterfaces(); //nombre d'interfaces implémentées
System.out.println("Il y a " + interfaces.length+"  interfaces implémentées ");
for(int i=0; i< interfaces.length; i++) System.out.println(interfaces[i]);
```



Objet Class

```
Class c = new String().getClass();
Method [ ] m =c.getMethods(); //Les méthodes de cette classe
System.out.println("Il y a " + m.length+"  méthodes de cette classe ");
for(int i=0; i< m.length; i++) System.out.println(m[i]);
```



Objet Class

```
Class c = new String().getClass();
Method [ ] m =c.getMethods(); //Les méthodes de cette classe
 System.out.println("Il y a " + m.length+"  méthodes de cette classe ");
 for(int i=0; i< m.length; i++)
 {
     System.out.println(m[i]);
     Class [ ] p=m[i].getParameterTypes();
      for(int j=0; i< p.length; j++)
        System.out.println(p[j].getName());
 }
```

Liste des champs

```
Class c = new String().getClass();
Field [ ] m =c.getDeclaredFields(); //Les champs de cette classe
```

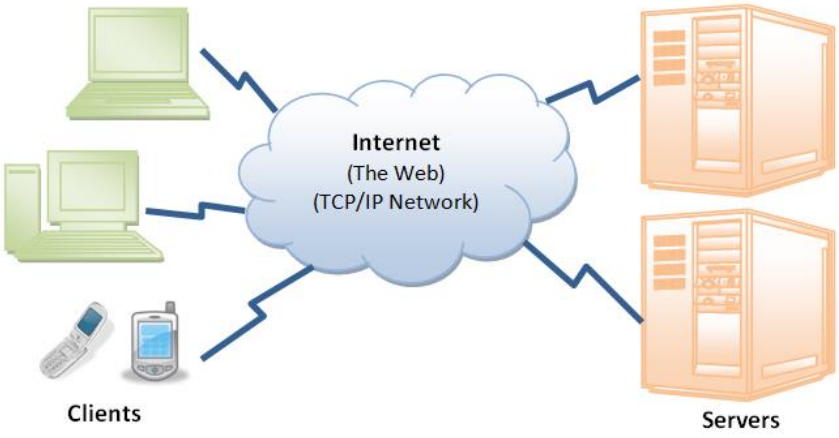

Java EE

Les bases de la plateforme Java EE

Plan

- Rappels
  - HTTP
- La plateforme Java Enterprise Edition
  - Introduction
  - L'architecture Java EE et les conteneurs
  - Les composants Java EE
  - Le services Java EE

Rappels



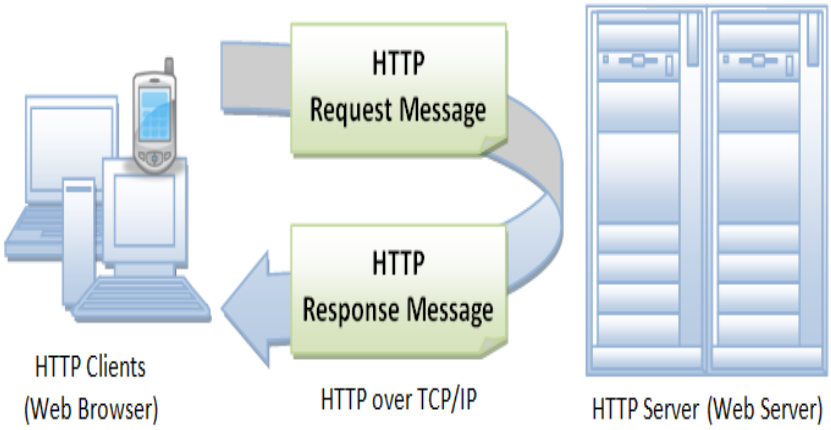
Rappels

- Internet : est un assemblage de multiples réseaux, tous connectés entre eux. Aussi, appelé le réseau des réseaux.
- Le Web : est un système de fichiers présents sur des serveurs transitant par des protocoles particuliers, consultable grâce à des navigateurs web.
- Pour une bonne communication entre le client et le serveur, un ensemble de protocoles de transfert de fichiers sont utilisés tels que : HTTP, FTP, SMTP, POP, etc.

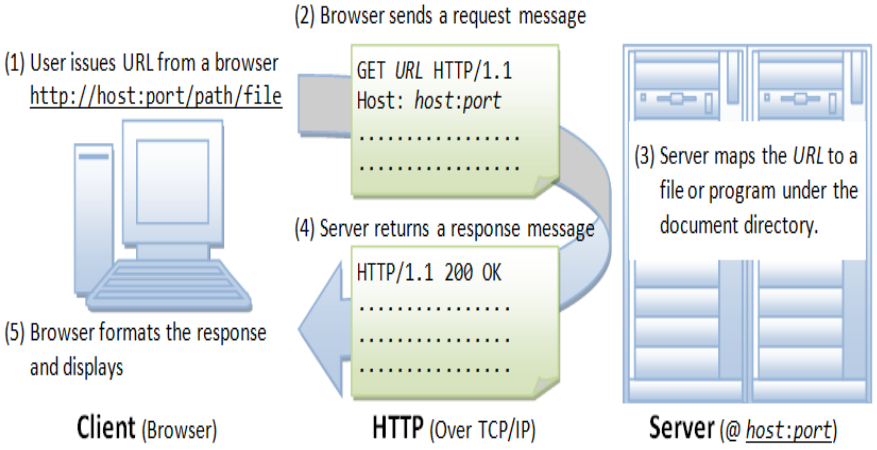
HyperText Transfer Protocol (HTTP)

- HTTP est le protocole d'applications le plus utilisé dans l'Internet (ou sur le Web).
- Un client HTTP envoie un message de requête à un serveur HTTP. Le serveur, à son tour, renvoie un message de réponse.
- HTTP est un protocole sans état. En d'autres termes, la demande actuelle ne sais pas ce qui a été fait dans les demandes précédentes.
- HTTP est basé sur des couples questions (requêtes) /réponses
- La requête contient au minimum des informations permettant d'identifier la ressource demandée par le client.
- La requête est simplement un bloc de texte transitant du client vers le serveur.

Protocole HTTP



Communication client serveur



L'URL

- URL : Uniform Resource Locator est associé au protocole HTTP.
- URL permet de localiser la ressource que l'on souhaite récupérer via une requête.
- Son format : protocole:// nom d'hôte: n° de port/ressource
- protocole : spécifie protocole utilisé HTTP, FTP, TELNET,…
- Nom d'hôte : nom de domaine DNS (www.test101.com) ou l'adresse IP (192.128.1.2) du serveur.
- numéro du port : numéro du port TCP sur lequel le serveur écoute les requêtes entrantes provenant des clients. Exemples : HTTP:port 80; FTP:port 20 ou 21; SMTP:port 25; …
  http://www.monsite.com:80/dossier/fichier.html

L'URL

- ressource : détermine l'emplacement de la ressource que l'on souhaite obtenir.
- Par exemple, dans l'URL http://www.test101.com/docs/index.html ,
- le protocole de communication est HTTP,
- le nom d'hôte est www.test101.com .
- Le numéro de port n'est pas spécifié dans l'URL, et prend le numéro par défaut, qui est le port TCP 80 pour HTTP.
- L'emplacement de la ressource est situé au " /docs/index.html ".
- D'autres exemples d'URL :
  - ftp://www.ftp.org/docs/test.txt
  - mailto: user@test101.com
  - telnet :/ / www.test101.com/

Protocole HTTP

- Soit une requête HTTP concernant la page d'accueil du site www.eni-ecole.fr

```
GET / HTTP/1.1
Host: www.eni-ecole.fr
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; fr;
rv:1.9.0.15) Gecko/2009101601 Firefox/3.0.15 (.NET CLR 3.5.30729)
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: fr,en;q=0.8,fr-fr;q=0.6,en-us;q=0.4,zh;q=0.2
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```


Protocole HTTP

- réponse HTTP faite par le serveur est (corps de la réponse tronqué) :

```
HTTP/1.1 200 OK
Date: Mon, 04 Jan 2010 11:20:50 GMT
Server: Apache/2.2.8 (Ubuntu) PHP/5.2.4-2ubuntu5.3 with Suhosin-Patch
X-Powered-By: PHP/5.2.4-2ubuntu5.3
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

<?xml version="1.0" encoding="iso-8859-1"?><!DOCTYPE html PUBLIC
"-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
.<META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-CACHE">
.<META HTTP-EQUIV="PRAGMA" CONTENT="NO-CACHE">
.<META HTTP-EQUIV="EXPIRES" CONTENT="0">
```

Les différents types de requêtes

- HTTP 1.0 dispose de 3 types de requêtes :
- Requête GET : pour obtenir une ressource disponible sur le serveur. Avec Get, les paramètres envoyés sont ajoutés à la suite de l'URL.
- Requête POST : pour envoyer vers le serveur des informations saisies dans un formulaire HTML. Les informations étant transférées via le corps de la requête et ne sont pas visibles dans l'URL.
- Requête HEAD : comme Get mais le serveur ne retourne rien dans le corps de la réponse HTTP. Elle utilisée pour la gestion de la mise en cache des informations.

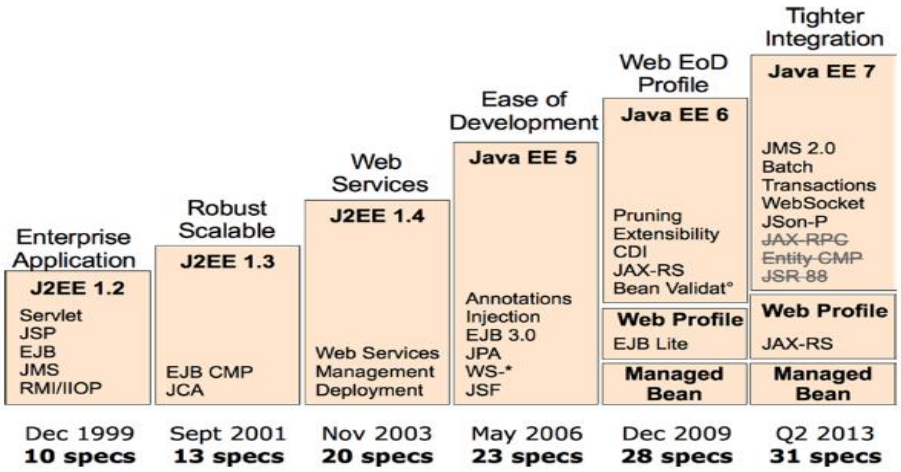
Les différents types de requêtes

- HTTP 1.1 ajoute de nouveaux types de requêtes :
- Requête PUT : permet d'envoyer vers le serveur une ressource pour l'enregistrement permanent.
- Requête DELETE : permet de demander au serveur la suppression d'une ressource indiquée.
- Requête TRACE : pour le test, permet de demander au serveur de retourner dans le corps de la réponse une copie de la requête qu'il vient de recevoir.
- Requête OPTIONS : permet d'obtenir les informations sur les options utilisables pour obtenir une ressource.

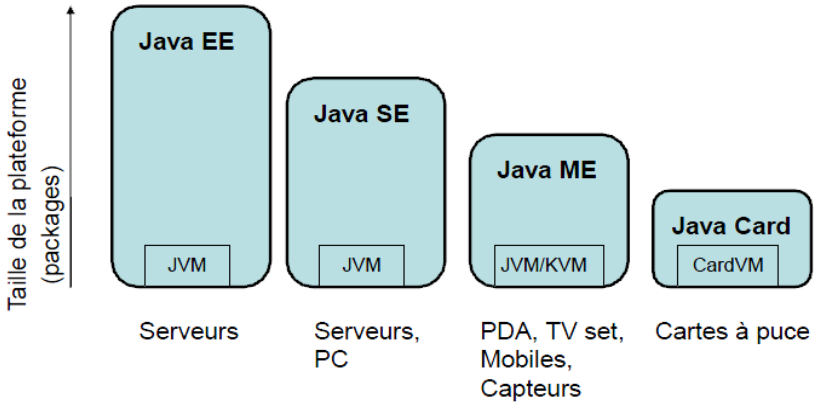
Les différents types de réponses

- Les réponses HTTP sont organisées en cinq catégories :
- 1XX : information. (100 continue : pour requêtes en plusieurs parties. dire que le début est bien reçu et j'attend la suite)
- 2XX : succès (200 OK : requête traitée correctement, 201 créé : ressource créée correctement en réponse à put)
- 3XX : redirection (301 déplacement permanent, 302 déplacement temporaire, 304 non modifié)
- 4XX : erreur liée au client (400 mauvaise requête, 401 client non autorisé, 403 accès interdit, 404 ressource non trouvée)
- 5XX : erreur liée au serveur (500 erreur interne, 503 service indisponible, 505 version HTTP non prise en charge)

JEE : Histoire



La galaxie java



Java EE

- Java EE : signifie Java Entreprise Edition et représente essentiellement des applications d'entreprise (réservation de vols,…).
- Java EE : Un ensemble de technologies pour construire des applications réparties
  - JSP/Servlet , JSF(i.e. *Web Component) : pages web dynamiques*
  - Composants EJB (i.e. Enterprise Java Bean) : composants JEE
  - JDBC : API d'accès à des SGBD, JPA
  - Et de nombreuses autres : JNDI, JTA, JCA, JMX, JAX, …
- Les applications d'entreprise peuvent avoir des interfaces utilisateurs multiples : interface web accessible sur Internet et une interface graphique sur le réseau local.

Java EE

- Les applications Java EE doivent posséder les qualités suivantes :
  - gérer les communications entre systèmes distants.
  - s'occuper automatiquement des différents protocoles de communication
  - synchroniser les sources avec éventuellement des technologies différentes
  - s'assurer que le système respecte en permanences les règles de l'activité de l'entreprise, appelés règles "métier".
  - s'occuper automatiquement de la base de données sans que le développeur est à intervenir.

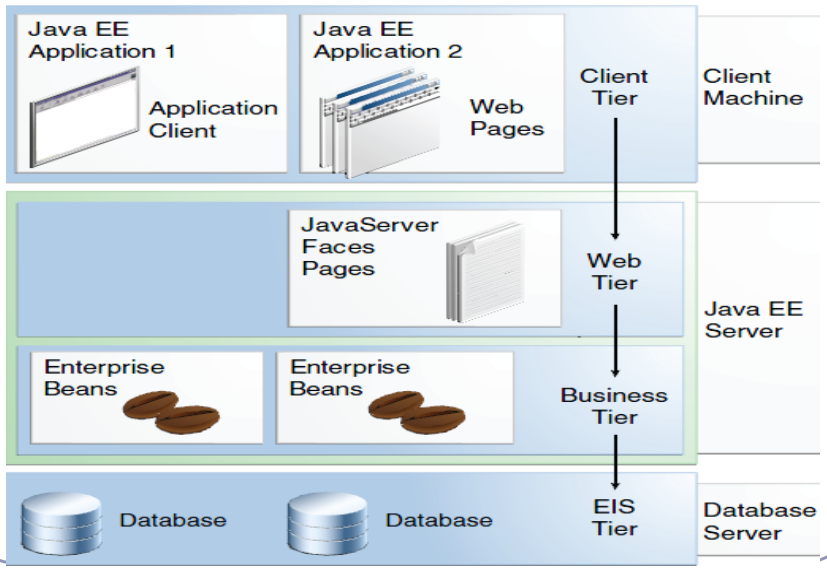
Serveurs d'applications

- Les serveurs d'applications mettent à disposition du programmeur les fonctionnalités permettant de réaliser des applications d'entreprise :
- communication entre ordinateurs
- mis en place de protocole adaptés
- gestion des connexions avec une base de données
- présentation de pages Web
- gestion des transactions
- etc.

Java EE : applications distribuées

- *Java EE est une collection de composants, de conteneurs et de services permettant de créer et de déployer des applications distribuées au sein d'une architecture standardisée.*
- Java EE est surtout destiné aux gros systèmes d'entreprise. Il ne fonctionne pas sur un simple PC mais requière une puissance beaucoup plus importante.
- Les applications JEE doivent être constituées de plusieurs composants pouvant être déployés sur des plateformes multiples afin de disposer de la puissance de calcul nécessaire. C'est la raison d'être des applications distribuées.

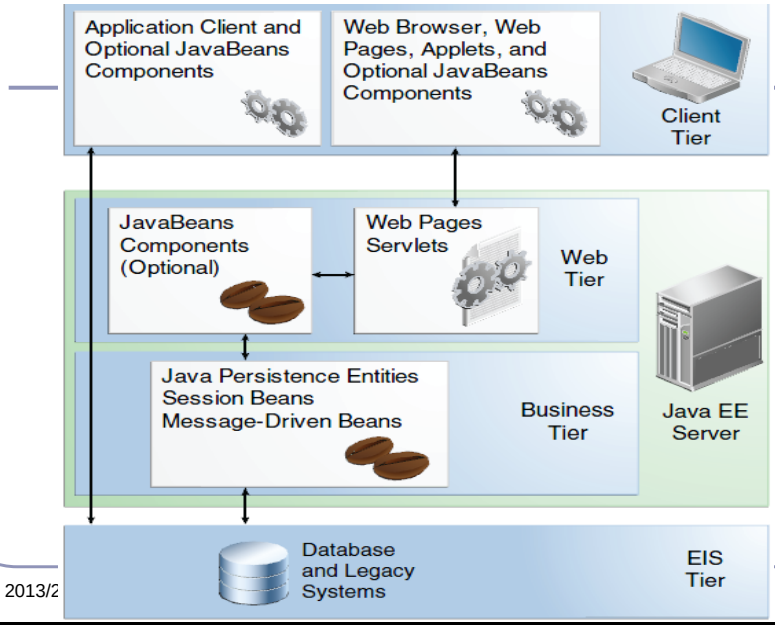
JEE : architecture multi-tiers



Les composants de Java EE

- Les applications Java EE sont formées de composants
- Un composant Java EE est une application fonctionnelle formée de classes et de fichiers qui peut communiquer avec d'autres composants.
- L'application client et les applets sont des composants coté client
- Les servlets, JSF et JSP sont des composants web côte serveur
- Les EJB sont des composants métiers côte serveur
- Java Bean n'est pas considéré comme composant

Architecture JEE

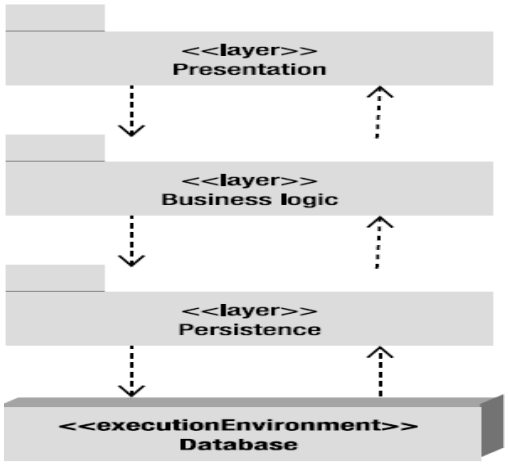


Architecture Java EE

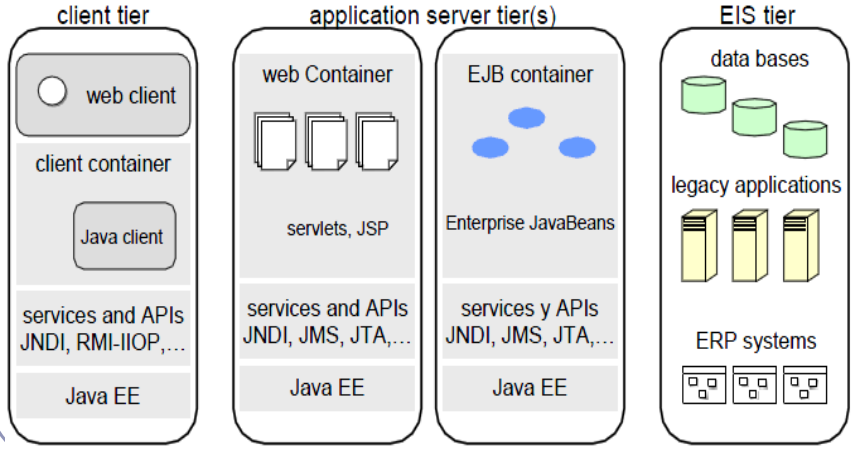
L'architecture d'une application Java EE se découpe idéalement en au moins trois couches :

- La couche cliente (présentation) : permet de dialoguer avec l'utilisateur. Elle est composée d'une application stand-alone, application web ou applet.
- La couche métier : encapsule les traitements (dans des composants EJB)
- La couche de données : stocke des données (Fichiers, Bases de données relationnelles ou XML, Annuaires d'entreprise, …)

Architecture d'une application Java EE



Architecture d'une application Java EE



Java EE : côté client

- Un client Java EE peut être :
  - une application console écrite en Java
  - une application dotée d'une interface graphique Swing. Ce type de client est appelé client lourd, en raison de la quantité importante de code qu'il met en œuvre.
- Un client Java EE peut également être conçu pour être utilisé à partir du Web.
  - Ce type de client fonctionne à l'intérieur d'un navigateur Web.
  - La plus grande partie du travail est reportée sur le serveur et le client ne comporte que très peu de code. Pour cette raison, on parle de client léger.
  - Un client léger peut être une simple interface HTML, une page contenant des scripts JavaScript, ou encore une applet Java si une interface un peu plus riche est nécessaire.

JEE : côté serveur

- Les API de Java EE peuvent se répartir en deux grandes catégories :
- Les **composants déployés sur le serveur**
  - Les composants web **qui sont réalisés à l'aide de servlets, de JavaServer Pages (JSP) ou de Java server face (JSF)**.
  - Les composants métier **sont des Entreprise JavaBeans (EJB).** Il s'agit de composants spécifiques chargés des traitements des données et de l'interfaçage avec les bases de données.
- Les **services**
  - Les services d'infrastructures : JDBC, JNDI, JTA, JCA, JMX
  - Les services de communication : RMI-IIOP, JavaMail, JAAS

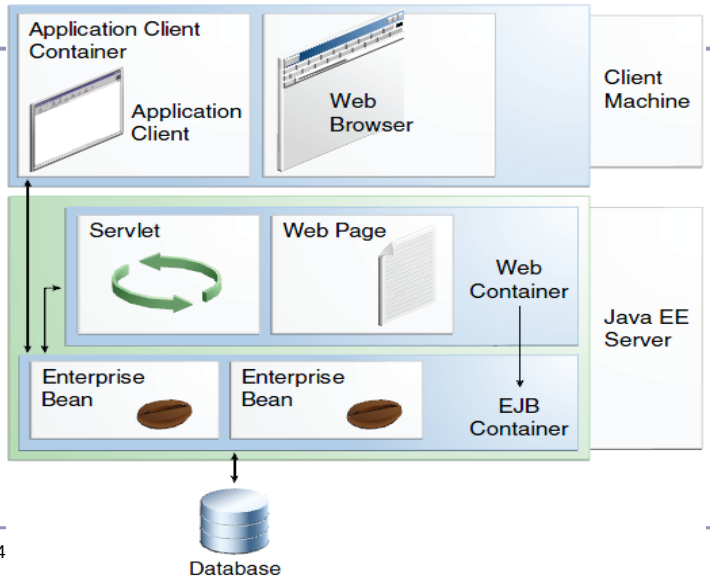
les conteneurs Java EE

- Les conteneurs sont les éléments fondamentaux de l'architecture Java EE.
- Ils fournissent un ensemble de services permettant aux développeurs d'applications de se concentrer sur la logique métier du problème à résoudre sans se préoccuper de toute l'infrastructure interne.
- Les conteneurs assurent la gestion du cycle de vie des composants qui s'exécutent en eux. Les conteneurs fournissent des services qui peuvent être utilisés par les applications lors de leur exécution.

les conteneurs Java EE

- La plate-forme Java EE disposent de conteneurs pour les composants Web et les composants métiers.
- Ces conteneurs possèdent des interfaces leur permettant de communiquer avec les composants qu'ils hébergent.
- Il existe plusieurs conteneurs définit par Java EE:
  - conteneur web : pour exécuter les servlets, les JSP et JSF
  - conteneur d'EJB : pour exécuter les EJB
  - conteneur client : pour exécuter des applications autonomes sur les postes qui utilisent des composants Java EE

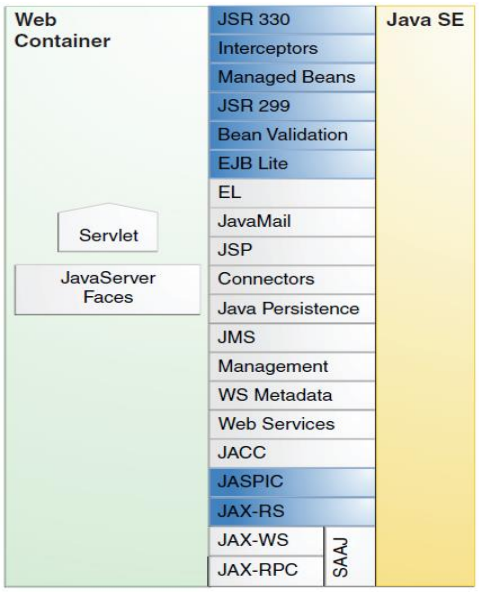
Les conteneurs Java EE



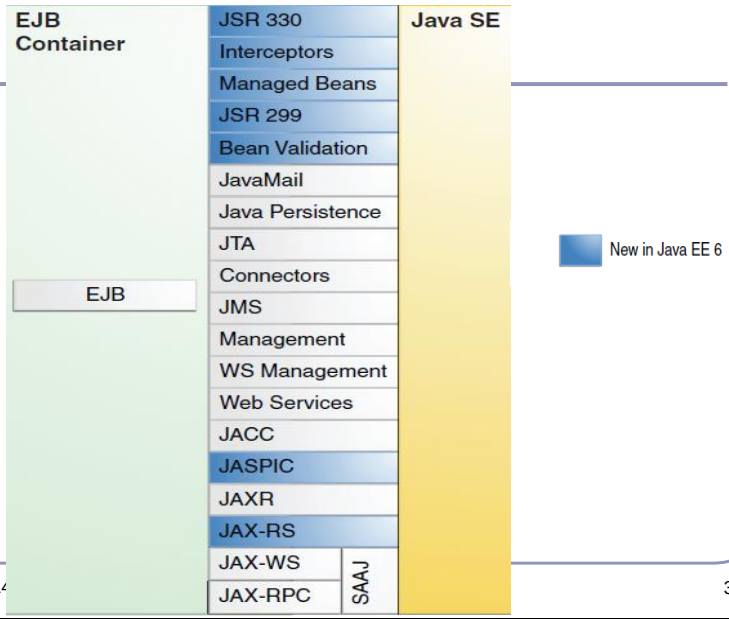
Les serveurs JEE

- Un serveur JEE est capable d'exécuter :
  - des servlets : classes Java exécutées côté serveur
  - des JSP, JSF :
    - Java Server Pages
    - Fichiers de script qui «mélangent» du Java et du HTML/Javascript
- Le développement avec les servlets/JSP nécessite des conteneurs de servlets simples : Tomcat (référence), Resin, Jetty,…
- Le développement de l'ensemble des spécifications JEE exige des conteneurs de servlets et des EJBs :
- Les serveurs d'applications peuvent fournir un conteneur web uniquement (Tomcat) ou un conteneur d'EJB uniquement (JBoss, Jonas, ...) ou les deux (GlassFish, Websphere, Weblogic, ...).

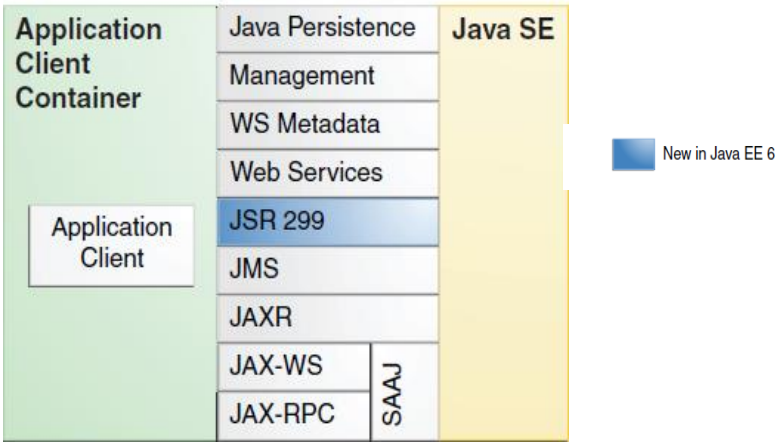
Java EE : Les APIs du conteneur web



Java EE : Les APIs du conteneur EJB



Java EE : Les APIs du conteneur client



déploiement d'une application

- Pour déployer une application dans un conteneur, il faut lui fournir deux éléments :
- l'application avec tous les composants (classes compilées, ressources ...) regroupée dans une archive ou module. Chaque conteneur possède son propre format d'archive.
- un fichier descripteur de déploiement contenu dans le module qui précise au conteneur des options pour exécuter l'application
- Une application est un regroupement d'un ou plusieurs modules dans un fichier EAR (Entreprise ARchive). L'application est décrite dans un fichier application.xml lui même contenu dans le fichier EAR

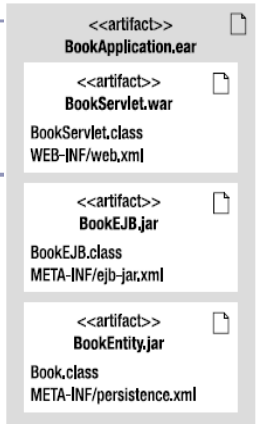
Les packages d'une application JEE

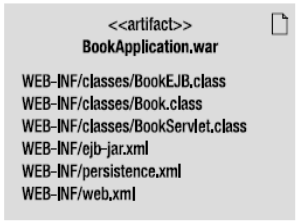
- Les composants web
  - Une application Web (*.xhtml, *.jsp, servlets, …) empaquetée dans un .jar (.war) et est paramétrée dans le fichier WEB-INF/web.xml
  - L'application est installée dans le répertoire webapps du serveur web JavaEE
- Structure d'une *Web Application Archive (.war)*
  - ***.xhtml, *.png, *.jsp, …, applets.jar, midlets.jar**
  - **WEB-INF/web.xml :** Fichier de déploiement pour paramétrer les servlets, types MIME additionnels, …
  - **WEB-INF/classes/ :** .class des servlets et des classes (JavaBean, …) associées ressources additionnelles (localstring.properties, …)
  - **WEB-INF/lib/ :** .jar additionnels provenant de tierce parties (comme des drivers JDBC,TagLib (jsf, …), …
  - **WEB-INF/tlds/ :** .tld décrivant les TagLibs

Assemblage

- Lorsque tous les modules sont assemblés dans un fichier jar, on peut les déployer directement dans un conteneur.
- On peut aussi intégrer le fichier **jar** dans un fichier **ear** (entreprise archive) et déployer ce dernier.
- Un fichier **ear** sert à assembler un ou plusieurs modules (des EJB ou des applications web) en une archive unique afin que leur déploiement sur un serveur d'applications soit simultané et cohérent.
- Pour déployer une application web on peut assembler les EJB et les entités dans des fichiers **jar** séparés, les servlets dans un fichier **war** et tout regrouper dans un fichier **ear**.
- Il suffit ensuite de déployer ce fichier sur le serveur d'applications pour pouvoir manipuler les entités à partir de la servlet en utilisant les EJB.

Assemblage





- Depuis EJB 3.1, les EJB peuvent également être assemblés directement dans un module web (fichier war).
- Dans la figure de droite, la servlet, l'EJB et l'entité sont tous assemblés dans le même fichier war, avec tous les descripteurs de déploiement.
- Le descripteur de déploiement est stocké dans META-INF/ejb-jar.xml dans le module EJB et dans WEB-INF/web.xml dans le module web.

Extraction

- Pour visualiser le ce contenu de l'archivage d'une application :

```
jar  tf  application.war
```

- Les fichiers peuvent être extraits de l'archive avec la commande suivante

```
jar  xvf  application.war
```

- Les fichiers dans l'archive sont recrées sur le disque dans le répertoire courant

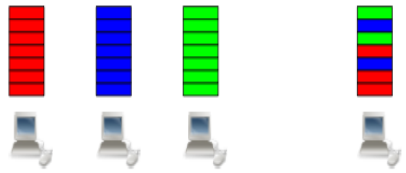
Les Threads

Les threads en java

- La programmation concurrente
- Threads et processus
- États d'un thread
- Priorités
- Synchronisation

Programmation concurrente

- Permet d'effectuer plusieurs traitements, spécifiés à être distincts les uns des autres, « **en même temps** »
- En général, dans la spécification d'un traitement, beaucoup de temps est passé en «attente»
  - Idée: exploiter ces temps d'attente pour réaliser d'autres traitements, en exécutant **en concurrence plusieurs traitements**
  - Sur mono-processeur, simulation du parallélisme

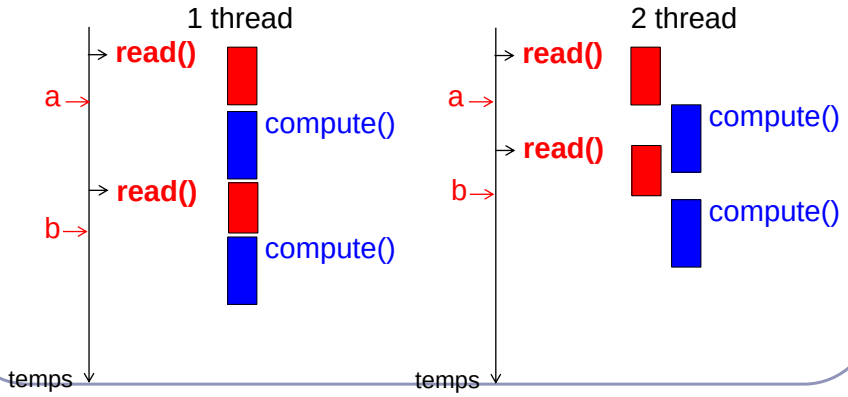


Application concurrente

Application non-concurrente

Entrelacer calcul/entrée/sortie

- Utilisation des temps d'attente de saisie



Programmation concurrente

- Une *application concurrente* est capable d'effectuer plusieurs tâches en même temps, en parallèle.
- Un programme est souvent constitué de plusieurs *threads d'exécution*. Il s'agit en fait de tâches dans un programme qui pourront être exécutées en parallèle.
- Le grand avantage des threads est qu'ils sont beaucoup plus facile à créer et nécessitent moins de ressources étant donné qu'ils vont pouvoir partager les ressources du programme qui les a lancé.
- Dans la programmation concurrente, il existe principalement deux unités d'exécution : les processus et les threads.
- L'exécution des processus et des threads est gérée par l'OS.
- Un processus possède son propre environnement d'exécution (ressources systèmes)

Threads et processus

- Multi-tâches de threads : il y a moins de surcharge que le multi-tâches de processus
- Processus :
  - Un ensemble de ressources ;

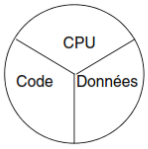
  Sous UNIX : segment de code, segment de données utilisateur, segment de données systèmes (répertoire de travail, descripteurs de fichiers, identificateurs de l'utilisateur ayant lancé le processus, du groupe dont il fait partie, infos. sur l'emplacement des données en mémoire, . . .).
  - Une unité d'exécution (sous UNIX, des informations nécessaires à l'ordonnanceur, la valeur des registres, le compteur d'instructions, la pile d'exécution, des informations relatives aux signaux)

Threads et processus

- Thread : on ne retient que l'aspect d'unité d'exécution.
- Contrairement aux processus, les threads sont légers :
  - ils partagent le même espace d'adressage,
  - ils existent au sein du même processus,
  - la communication inter-thread occasionne peu de surcharge,
  - le passage contextuel (context switching) d'un thread à l'autre est peu coûteux.
  - le multi-tâches de processus n'est pas sous le contrôle de l'environnement d'exécution java.
  - par contre, il y a un mécanisme interne de gestion multi-threads.
  - Par ex., un serveur multi-tâches (avec une tâche par client à servir)

Thread : définition

- Thread (ou contexte d'exécution) est considéré comme l'encapsulation d'une CPU virtuelle avec son propre code de programme et ses propres données.
- Un thread est composé de trois parties :
- Une CPU virtuelle
- Le code exécuté par la CPU
- Les données auxquelles est appliqué le code



Un thread ou contexte d'exécution

Les processus

- C'est un objet représentant une application qui s'exécute :
  - java.lang.Runtime
  - Contrôle l'environnement d'exécution. C'est une classe singleton obtenue par : Runtime rt=Runtime.getRuntime()
  - D'autres méthodes: [total/free/max]Memory(), gc(), exit(), halt(),...
  - exec() crée un nouveau processus
  - java.lang.Process et java.lang.ProcessBuilder
  - Contrôle un (ensemble de) processus, ou de commandes
  - Runtime.getRuntime().exec("cmd") crée un nouveau processus correspondant à l'exécution de la commande, et retourne un objet de la classe Process qui le représente

La classe Runtime

- Les différentes méthodes exec() créent un processus natif.
  - Exemple simple:

```
Runtime.getRuntime().exec("javac MonProg.java");
```

  - Avec un tableau d'arguments

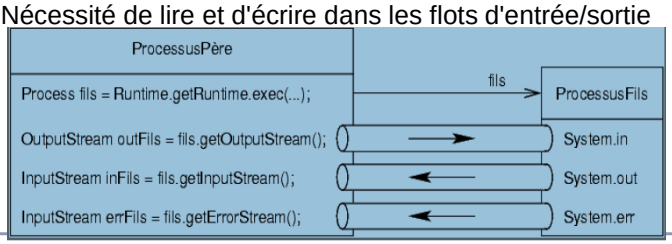
```
Runtime.getRuntime()
        .exec(new String[]{"javac", "MonProg.java"});
```

  - Exécute la commande cmd dans le répertoire /tmp avec comme variable d'environnement variable de valeur value.

```
Runtime.getRuntime().exec("cmd",
        new String[] {"VARIABLE=VALUE"}, new File("/tmp/"));
```


La classe Process

- Objet retourné par méthode exec() de Runtime
  - Process fils= Runtime.getRuntime().exec("commande");
    fils.waitFor(); // attend la terminaison du processus fils
    System.out.println(fils.exitValue());
  - Toutes les méthodes sont abstraites dans Process:
  – destroy(), getInputStream(), getOutputStream(), getErrorStream()
  - Nécessité de lire et d'écrire dans les flots d'entrée/sortie



Thread : processus léger

– Étant donnée une exécution de Java (une JVM)
- **un seul processus (au sens système d'exploitation)**
- disposer de **multiples fils d'exécution (threads) internes**
- possibilités de **contrôle plus fin (priorité, interruption...)**
- espace **mémoire commun entre les différents threads**

– Les processus légers s'exécutent en concurrence sur le nombre de processeurs disponibles

– Ils ont des piles différentes mais accède au même tas
- => problèmes d'accès concurrent

Threads de base

– Lorsqu'on exécute la commande % java Prog
- La JVM démarre plusieurs threads, dont le thread "main"
- Le thread "main" est d'exécuter le code de la méthode main()
- Le code peut demander la création d'autres threads
- Les autres threads servent à la gestion de la JVM (ramasse miettes, etc).
  – "Signal Dispatcher", "Finalizer", "Reference Handler", etc.
  – on peut avoir ces informations en envoyant un signal QUIT au programme (Ctrl-\ sous Unix ou Ctrl-Pause sous Windows).
  – La commande jconsole permet de « monitorer » les programmes Java

Créer un thread

```
public class CountDown extends Thread{
  public void run() {
       for (int i = 5; i >= 0; i--) {   System.out.println (i);  }
  }
}
```

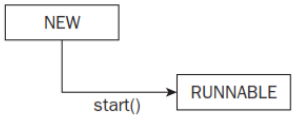
```
Thread thread = new CountDown();
thread.run();
```

- Ce programme est complètement séquentiel, il n'y a aucun thread qui a été créé.
- Pour créer un nouveau thread et avoir la tâche exécutée en parallèle, il faut appeler la méthode start() de la classe Thread.

```
Thread thread = new CountDown();
thread.start();
```


Caractéristiques d'un thread

- **Le Lancement d'un thread se fait par** l'appel de sa méthode start() (On appelle jamais directement run() start() le fait)
- **Remarques :**
- L'appel de start() se fait **une et une seule fois** pour chaque *thread*, sinon *java.lang.IllegalThreadStateException* si le *thread* n'est pas mort. Pas d'exception, mais rien n'est lancé, si le *thread* est mort
- Un *thread* meurt lorsque sa méthode run() se termine



Classe Thread et interface Runnable

- Système de multi-threads : autour de la classe Thread et de l'interface Runnable.
- La liste des méthodes les plus courantes :

Méthode	but
getPriority()	obtenir la priorité d'un thread
isAlive()	déterminer si un thread est toujours en cours d'exécution
join()	attente du thread en cours jusqu'à ce que le thread sur lequel est appelée join se termine
resume()	Poursuivre l'exécution d'un thread suspendu (obsolète)
run()	contient la tâche à exécuter. Il faut la redéfinir sans l'appeler
sleep()	Suspendre le thread courant pour une période de temps donnée
start()	permet de créer un nouveau thread et d'y exécuter la méthode run()
suspend()	Suspendre un thread (obsolète)

Thread : méthodes de gestion

Méthode	But
yield()	permet d'arrêter le thread en cours d'exécution et de donner la main à d'autres threads dont la priorité est au moins égale à celle du thread arrêté.
currentThread()	permet de récupérer l'instance du thread qui est en train d'exécuter le code.
getName()	récupérer le nom du thread
setName()	change le nom du thread via son constructeur. Par défaut, les threads sont successivement nommés *thread-0*, *thread-1*,…

Avant d'être exécutés, les threads doivent être créés:

```
Thread t = new Thread(...);
```

– Au démarrage du thread, par t.start();

- la JVM réserve et affecte l'espace mémoire nécessaire avant d'appeler la méthode run() de la cible.

Deux façons pour spécifier run()

– Redéfinir la méthode run() de la classe Thread

```
class MyThread extends Thread {
    @Override public void run() { // code }
    }
    ...
    Thread t=new MyThread();          // création
    t.start();                        // démarrage
```

– Implémenter l'interface Runnable

```
    class MyRunnable implements Runnable {
            public void run() { // code }
    }
    ...
    MyRunnable r=new MyRunnable();
    Thread t=new Thread(r); // création
    t.start();                  // démarrage
```


Comparaison des deux approches

- Par implémentation de l'interface
  - Usage : dans le package : **java.lang**
    ➔ public MaClasse implements Runnable
  - Avantages et inconvénients
    ☺ Meilleur sur le plan orienté objet
    ☺ La classe peut hériter d'une autre classe :
    public class MaClasse **extends MaGClasse** implements Runnable {...}
    ☺ Consistance
- Par héritage de la classe Thread elle-même
  - Usage : dans le package : **java.lang.thread**
    ➔ public MaClasse extends Thread
  - Avantages et inconvénients
    ☺ Code simple (l'objet est un Thread lui-même)
    ☹ La classe ne peut plus hériter d'une autre classe

Création de Thread : exemple

```
public class MonFil implements Runnable {
  public void run(){
    byte[] buffer=new byte[512];
    int i=0;
    while(true){
     if(i++%10==0)
       System.out.println(""+i+" est divisible par 10");
      if (i>101) break;
    }
  }
}
public class LanceFil {
  public static void main(String[]arg){
    Thread t=new Thread(new MonFil());
    t.start();
  }
}
```

Le constructeur de la classe Thread attend un objet Runnable en argument

Création de Thread : exemple

```
public class MyThread extends Thread {
  public void run(){
    byte[] buffer=new byte[512];
    int i=0;
    while(true){
      if(i++%10==0)
       System.out.println(""+i+" est divisible par 10");
      if(i>101) break;
    }
  }
}

public class LaunchThread{
  public static void main(String[]arg){
    MyThread t=new MyThread();
    t.start();
  }
}
```

Grâce à l'héritage, un objet de type MyThread est lui-même Runnable, on peut donc appeler un constructeur sans argument

Création de Thread : exemple

- utilisation d'une classe anonyme :

```
Thread monThread = new Thread() {
 public void run () {
 ...
 }
 };
 monThread.start();
```

Avantage :

- Pratique si le code comporte peu de lignes (sinon c'est vite illisible)

Thread et objet de contrôle

– À la fin de l'exécution de la méthode run() de la cible, le thread est terminé (mort):

- il n'est plus présent dans la JVM (en tant que thread)
- mais l'objet contrôleur (de classe Thread) existe encore
- sa méthode isAlive() retourne false
- il n'est pas possible d'en reprendre l'exécution
- l'objet contrôleur sera récupéré par le ramasse-miettes

– L'objet représentant le thread qui est actuellement en train d'exécuter du code peut être obtenu par la méthode statique Thread.currentThread()

passage des paramètres au thread

- **On ne peut pas changer la signature** de la méthode start() pour passer des paramètres au *thread*.
- Le passage des paramètres au Thread peut se faire comme suit :
  1. Définir des variables d'instance.
  2. Les initialiser dans le constructeur.
- Exemple

```
public class Test implements Runnable {
   int p1; Object p2;
   public Test(int p1,Object p2) {this.p1=p1; this.p2=p2; }
   public void run() { ... p1 ... p2 ... }
}
new Thread(new Test(12,aRef)).start();
```


Le thread courant

- Par défaut le thread courant s'appel **main.**

```
public class ThreadExample {
    public static void main(String[] args) throws InterruptedException {
    // Affiche les caractéristiques du thread courant
    Thread t = Thread.currentThread();
    System.out.println(t);
    // Donne un nouveau nom au thread
    t.setName("Fils");
    System.out.println(t);
    // Rend le thread courant
    // inactif pendant 1 seconde
    Thread.sleep(1000);  //millisecondes
    System.out.println("fin");
  } }
```

```
% java ThreadExample
Thread[main,5,main]
Thread[Fils,5,main]
fin
%
```

nom priorité groupe

L'accès au processeur

- Différents états possibles d'un thread
  - exécute son code cible (il a accès au processeur)
  - attend l'accès au processeur (mais pourrait s'exécuter)
  - attend un événement particulier (pour pouvoir s'exécuter)
- L'exécution de la cible peut libérer le processeur
  - si elle exécute un yield() (demande explicite)
  - si elle exécute une méthode bloquante (sleep(), wait()...)
- Sinon, c'est l'ordonnanceur de la JVM qui répartit l'accès des threads au processeur.
  - utilisation des éventuelles priorités

Différents états d'un processus léger

– Depuis la 1.5, il est possible de connaître l'état d'un processus léger *via la méthode getState(), exprimé par un* type énuméré de type Thread.State :

- NEW : pas encore démarré;
- RUNNABLE : s'exécute ou attend une ressource système, par exemple le processeur;
- BLOCKED : est bloqué en attente d'un moniteur;
- WAITING : attente indéfinie de quelque chose d'un autre thread;
- TIMED_WAITING : attente bornée de quelque chose d'un autre thread ou qu'une durée s'écoule;
- TERMINATED : a fini d'exécuter son code.

Arrêt d'un processus léger

- Les méthodes stop(), suspend(), resume() sont **déconseillées**
  - Risquent de laisser le programme dans un état incohérent!
- La méthode destroy() n'est pas implémentée
  - Spécification trop brutale: l'oublier
- Terminer de manière douce...
- Un thread se termine normalement lorsqu'il a terminé d'exécuter sa méthode run()
  - obliger proprement à terminer cette méthode

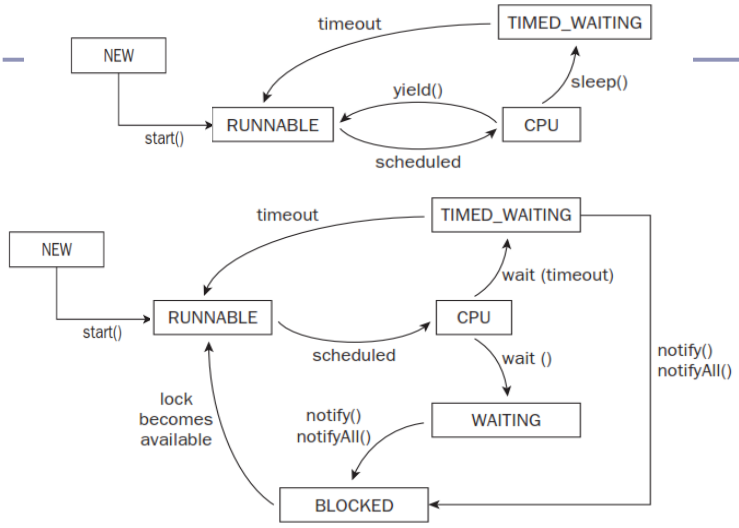
Interrompre un thread

- La méthode interrupt() appelée sur une thread t
  - Si t est en attente parce qu'elle exécute un wait(), un join() ou un sleep(), alors ce statut est réinitialisé et le thread reçoit une InterruptedException
  - Si t est en attente I/O sur un canal interruptible (java.nio.channels.InterruptibleChannel), alors ce canal est fermé, le statut reste positionné et le thread reçoit une ClosedByInterruptException
  - Sinon positionne un « statut d'interruption »
- Le statut d'interruption ne peut être consulté que par les méthodes interrupted() et isInterrupted()

Consulter le statut d'interruption

- public static boolean interrupted()
  - retourne true si le statut du thread actuellement exécuté a été positionné (méthode statique)
  - si tel est le cas, réinitialise ce statut à false
- public boolean isInterrupted()
  - retourne true si le statut du thread sur lequel est appelé la méthode a été positionné (méthode d'instance, non statique)
  - ne **modifie pas** la valeur du statut d'interruption

Les états d'un thread



Etats d'un thread

- Un thread de Java peut se trouver dans l'un des états qui suivent.
- nouveau (new) : Lorsqu'un nouveau thread est créé, par exemple avec Thread th = new Thread(a);
- Exécutable (runnable) : ou prêt à s'exécuter.
- En cours d'exécution(running : CPU) : Lorsque la méthode start() d'un thread est appelée, ce dernier passe dans l'état en cours d'exécution (running). Le système de planification accordent à chaque thread exécutable une tranche de temps pour effectuer sa tâche.
- Thread bloqué et en attente (blocked) : Un thread est bloqué ou en attente est temporairement inactif si :

  - tente d'obtenir un verrou d'objet détenu par un autre thread. Il se débloque lorsque le gestionnaire lui passe le verrou.

  - il entre dans un état de temporisation en appelant: Object.wait(), Thread.join(),Lock.tryLock() et Condition.await()

Etats d'un thread

- Thread terminé : Un thread peut se terminer pour l'une des deux raisons suivantes :

  - Il meurt naturellement lorsque la méthode run() s'est terminée normalement.

  - Il meurt soudainement parce qu'une exception non récupérée a mis fin à la méthode run().
- Remarque :
- Object.wait() : met en attente le *thread* en cours d'exécution
- Object.notify() : réactive un *thread* mis en attente par wait()
- Object.notifyAll() : réveille tous les threads en attente par wait(); le thread de priorité la plus élevée s'exécutera en premier.

  Ces méthodes nécessitent un accès exclusif à l'**objet exécutant** !!

  À utiliser avec méthode synchronized ou bloc synchronized

Thread : priorité

- Les threads peuvent avoir des priorités différentes :
- Un thread plus prioritaire a la main en priorité.
- Accès aux priorités, méthodes de la classe Thread :

- public int getPriority() : retourne le niveau de priorité du thread.
- public void setPriority(int priority) : change le niveau de priorité du thread

- Trois constantes de la classe Thread pour définir les priorités :
  - MAX_PRIORITY : niveau de priorité maximal possible (10)
  - MIN_PRIORITY : niveau de priorité minimal possible (1)
  - NORM_PRIORITY : niveau de priorité par défaut (5)

Multi-tâche : Priorité

```
package com.moi.test;
public class TestThread10 {
  public static void main(String[] args) {
    System.out.println("Thread.MIN_PRIORITY  = " +
     Thread.MIN_PRIORITY);
    System.out.println("Thread.NORM_PRIORITY = " +
     Thread.NORM_PRIORITY);
    System.out.println("Thread.MAX_PRIORITY  = " +
     Thread.MAX_PRIORITY);
  }
}
```

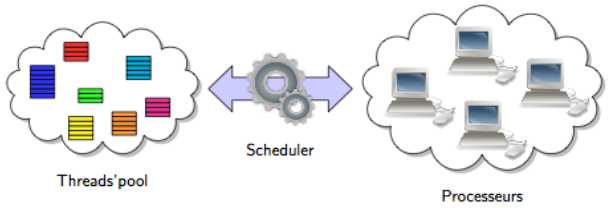
```
Résultat :
Thread.MIN_PRIORITY  = 1
Thread.NORM_PRIORITY = 5
Thread.MAX_PRIORITY  = 10
```

L'ordonnanceur

- Si le thread en cours d'exécution passe la main (via yield(), est suspendu ou bloqué, l'ordonnanceur choisit dans la file d'attente le thread de priorité la plus élevée (ou l'un d'entre eux si ils sont plusieurs) pour être exécuté.
- Un autre aspect est le partage du temps (time slicing). Un ordre de grandeur courant pour un quantum de temps est 100ms.
- En temps partagé, si un thread en cours d'exécution n'a pas été stoppé, bloqué, suspendu, ne s'est pas terminé, ou n'a pas passé la main, et s'il y a d'autres threads de priorité égale dans la file d'attente, le CPU est réalloué à l'un d'entre eux.

L'ordonnanceur

- Sous Solaris 2.x, la version 1.1 du JDK n'effectue pas de partage de temps entre les threads : un thread s'exécute jusqu'à ce qu'il se termine, soit stoppé, suspendu, bloqué, passe la main ou qu'un autre thread de plus haute priorité devienne prête.
- Le JDK 1.1 pour Windows XX partage le temps entre les threads.



L'ordonnanceur

```
public class ThreeThreadsTest {
      public static void main(String[] args) {
        new SimpleThread("Thread numéro 1").start();
        new SimpleThread("Thread numéro 2").start();
        new SimpleThread("Thread numéro 3").start();
      }
    }
    class SimpleThread  extends Thread {
      public SimpleThread(String str) {
        super(str);
      }
      public void run() {
        for (int i = 0; i < 10; i++) {
          System.out.println(i + " " + getName());
          try {
            sleep((long) (Math.random() * 1000));
          } catch (InterruptedException e) {
          }
        } // for
        System.out.println("DONE! " + getName());
      } // run()
} //SimpleThread
```

```
0 Thread numéro 2
0 Thread numéro 3
0 Thread numéro 1
1 Thread numéro 2
2 Thread numéro 2
3 Thread numéro 2
1 Thread numéro 1
4 Thread numéro 2
2 Thread numéro 1
3 Thread numéro 1
1 Thread numéro 3
4 Thread numéro 1
5 Thread numéro 2
2 Thread numéro 3
DONE! Thread numéro 2
5 Thread numéro 1
3 Thread numéro 3
DONE! Thread numéro 1
4 Thread numéro 3
5 Thread numéro 3
DONE! Thread numéro 3
```


L'ordonnanceur

- En ajoutant un sleep dans le code précédent, même de zéro millisecondes, on force en fait le thread à arrêter l'exécution est passer à l'état *Runnable*. Il s'agit donc juste d'une astuce nous permettant d'observer le travail du scheduler.
- La méthode de classe yield() permet d'arrêter l'exécution du thread courant. Après appel de la méthode, le thread courant va être suspendu, ce qui offre aux autres threads une chance d'être choisis par le scheduler pour pouvoir être exécutés.

Exemple : utilisation séquentielle

```
public class BankAccount {
    private int balance = 200;
    public int getBalance(){    return balance;}
    public void withdraw (int amount){
      if (amount <= balance){
         balance -= amount;
         System.out.println ("Compte débité de " + amount + " euros");
      }
   }
public static void main (String[] args){
   // Le compte bancaire partagé
   BankAccount account = new BankAccount();
   // Alice veut retirer 200 euros
   account.withdraw (200);
   System.out.println ("Il reste " + account.getBalance() + " euros sur le compte");
   // Bob veut retirer 200 euros
   account.withdraw (200);
   System.out.println ("Il reste " + account.getBalance() + " euros sur le compte");
}
}
```


Exemple : utilisation concurrente

- Jusque là, pas de surprises, tout se déroule comme on l'a toujours vu.
- Si on crée plusieurs threads qui vont accéder en parallèle à un même objet.
- On va donc créer deux threads, un pour Alice et un pour Bob. Ensuite, chacun de ces threads va tenter de retirer 200 euros sur le compte en même temps.

```
Compte débité de 200 euros
Il reste 0 euros sur le compte
Il reste 0 euros sur le compte
```

```
public static void main (String[] args) {   // Le compte bancaire partagé
      final BankAccount account = new BankAccount();
      class WithdrawMoney implements Runnable {
         public void run() {  account.withdraw (200);
      System.out.println ("Il reste " + account.getBalance() + " euros sur le compte");
         }
      }
   /*Alice et Bob veulent retirer 200 euros*/
   Thread t1= new Thread( new WithdrawMoney());   t1.setName("Alice");
   Thread t2=new Thread (new WithdrawMoney());   t2.setName("Bob"); t1.start();
   t2.start();
}
```

- Remplacer la méthode withdraw() de la classe BankAccount par :

```
public void withdraw (int amount) {
            if (amount <= balance){
            try {  Thread.sleep((long)(Math.random() * 1000));
                }catch (InterruptedException e){ }
          balance -= amount;
        System.out.println ("Compte débité de " + amount + " euros");
    } }
```


Exemple : utilisation concurrente

```
Compte débité de 200 euros
Il reste 0 euros sur le compte
Compte débité de 200 euros
Il reste -200 euros sur le compte
```

- Comment le solde du compte a-t-il pu devenir négatif malgré le test if (amount <= balance) qui est fait dans la méthode withdraw ?
- Ceci est dû à la méthode sleep appelée après le test if :
  - Alice exécute la méthode withdraw sur l'objet partagé, la condition du if est évaluée et vaut true ;
  - Le scheduler arrête d'exécuter Alice et passe à Bob ;
  - Bob exécute la méthode withdraw sur l'objet partagé, la condition du if est évaluée et vaut également true ;
- Les deux threads sont entrés dans la méthode withdraw et ont tous les deux passés le test du if. Ils vont donc tous les deux pouvoir retirer 200 euros sur le compte.

La synchronisation

- Lorsque 2 threads ou plus ont besoin d'une même ressource au même moment, il y a besoin de s'assurer que la ressource ne sera utilisée que par un thread en même temps : on utilise alors un procédé de synchronisation.
- Un moyen d'obtenir une synchronisation : les moniteurs.
- Moniteur : c'est une mini boîte ne pouvant contenir qu'un thread. Une fois qu'un thread y est entré, les autres doivent attendre qu'il en sorte.
- Les autres threads sont dites en attente du moniteur.
- Analogie avec des cabines d'essayage de vêtements.
- Deux moyens de synchronisation : les méthodes synchronisées et les blocs synchronisés.

Threads : Synchronisation

- Pour entrer dans le moniteur d'un objet, appeler une méthode modifiée avec le mot clé synchronized.
- Lorsqu'un thread est dans une méthode synchronisée, tout autre thread qui essaie de l'appeler (il ou tout autre méthode synchronisée) doit attendre.
- Pour sortir du moniteur, celui qui y est entré revient simplement de la méthode synchronisée.
- Un appel synchronisé peut être réalisé dans un bloc synchronisé :

```
synchronized(objet) { // instructions a synchroniser
}
```

où objet est une référence à l'objet à synchroniser.

- Un bloc synchronisé assure qu'un appel à une méthode quelconque d'objet ne pourra se faire qu'une fois que le thread courant sera entré dans le moniteur.

Synchronisation : exemple

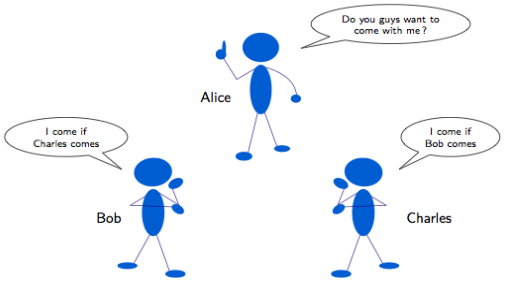
- Lorsqu'un thread est en train d'exécuter une méthode, empêcher l'accès à cette méthode à tous les autres threads.
- Exemple :
  public synchronized void withdraw (int amount){…}
- On obtient alors le résultat attendu suivant :

```
Compte débité de 200 euros
Il reste 0 euros sur le compte
Il reste 0 euros sur le compte
```


Synchronisation : Lock

- Tous les objets possèdent un *built-in lock*. Ce lock est nécessaire à un thread qui souhaite accéder à une méthode marquée synchronized.
- Un thread d'exécution va pouvoir prendre le lock d'un objet, et une fois cela fait, il va pouvoir entrer librement dans toutes les méthodes marquées synchronized. Une fois que le thread a terminé, il rend le lock de l'objet.
- Ce mécanisme permet donc de s'assurer qu'un seul thread à la fois est en train d'exécuter des méthodes d'un objet.
- Lorsqu'un thread tente d'exécuter une méthode marquée synchronized, deux cas peuvent se produire. Soit le lock de l'objet sur lequel la méthode est appelée est disponible et dans ce cas, le thread peut exécuter la méthode. Dans l'autre cas, le lock a déjà été pris et le thread est bloqué, en attente de la libération du lock.

Deadlock : Inter-blocage



- Alice désire inviter Bob et Charles à une super fête, et leurs demande donc si ils veulent venir. Bob ne vient que si Charles vient, et vice-versa.
- On est donc dans une situation où plus rien ne bouge, Bob attend la réponse de Charles qui attend celle de Bob. On est coincé, on est dans un *deadlock*.
- Si une variable est déclarée volatile, on a la garantie que si un thread modifie sa valeur, cette modification sera directement visible par les autres threads.
- Exemple : private volatile boolean cancelled;

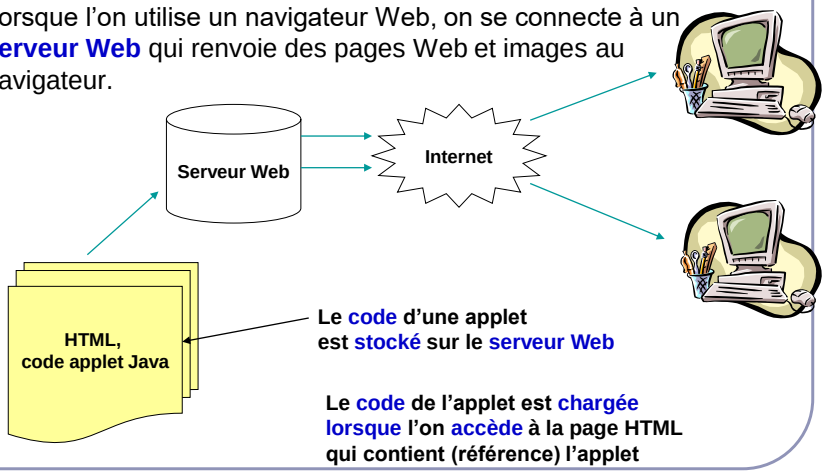
Les applets

Applet

- Plan
  - Généralités
    - Qu'est ce qu'une Applet
    - Différence entre Applet et Application
    - Applet et sécurité
  - Programmation d'une Applet
    - Le Cycle de vie d'une Applet
    - Dessin d'une Applet
    - Exemple
  - Inclusion d'une Applet dans une page WEB
  - Transmission de Paramètres à une Applet

Web et Applet

- Lorsque l'on utilise un navigateur Web, on se connecte à un **serveur Web** qui renvoie des pages Web et images au navigateur.



Exemple Applet

```
// AdditionApplet.java
import java.awt.Graphics; // importer la classe Graphics
import javax.swing.*; // importer le paquet javax.swing
public class AdditionApplet extends JApplet { double sum;
//somme des valeurs fournies par l'usager
public void init() { String firstNumber, secondNumber;
        double number1, number2;
     // lire les deux nombres
firstNumber = JOptionPane.showInputDialog("Entrer une première valeur en point flottant"
     );
secondNumber = JOptionPane.showInputDialog( "Entrer une deuxième valeur en point
     flottant" );
// convertir de chaîne de caractère à type double
number1 = Double.parseDouble( firstNumber );number2 = Double.parseDouble(
     secondNumber );
     // additionner les deux nombres
     sum = number1 + number2;
}
public void paint( Graphics g ){// écrire le résultat avec g.drawString
     g.drawRect( 15, 10, 270, 20 ); g.drawString( "The sum is " + sum, 25, 25 );
}}
```


Généralités

- Qu'est ce qu'une Applet ?
  - Une applet est un bout de code Java qui s'exécute dans un environnement d'un Navigateur
  - c'est une instance de la classe **java.awt.Panel**
- Différence entre Applet et Application
  - une applet **est chargée à travers le réseau** par l'intermédiaire d'un navigateur
  - une applet **ne réside pas sur le disque dur de la machine qui l'exécute**
  - le cycle de vie d'une applet est plus complexe que celui d'une Application. Pendant **qu'une applicaton est démarrée une et une fois**, une applet est initialisée(**init**), démarrée(start) ou arrêtée(stop) plusieurs fois
  - les applets **ont moins de droits** que les applications : sécurité oblige

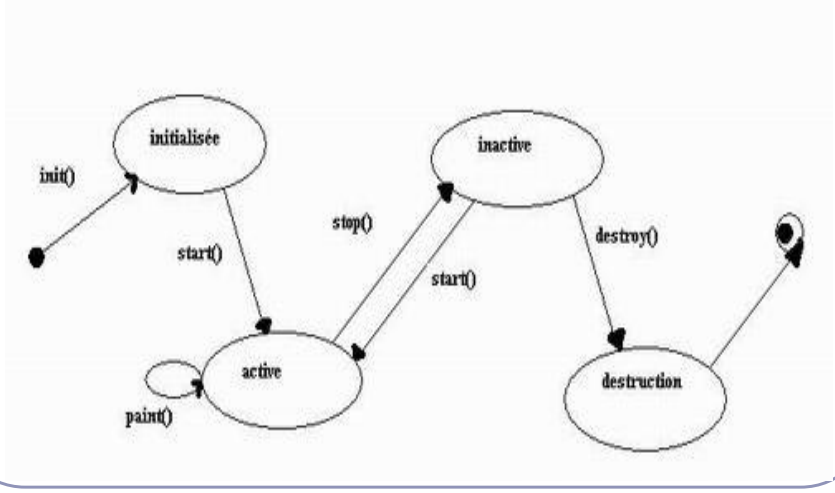
Généralités

- Applet et sécurité
  - Les applets **ne peuvent lire ni écrire sur le système de fichier de l'utilisateur** (sauf autorisation ou répertoire pré-désigné)
  - les applets **ne doivent communiquer à d'autres systèmes que ceux auxquels ils sont issues** (sauf paramétrage contraire du navigateur)
  - les applets **ne peuvent pas exécuter de programme dans le système** de fichier de leur utilisateur
  - Les applets **ne peuvent pas charger un programme natif sur le système de fichier de son utilisateur**

Programmation d'une Applet

- Le Cycle de vie d'une Applet
  - Une applet est créée en étendant la classe java.applet.Applet
  - Le Cycle de vie d'une applet comporte les points suivants :
    - Initialisation (**init**) :
    - Démarrage (**start**)
    - Affichage (**paint**)
    - Arrêt (**stop**)
    - Destruction(**destroy**)

Cycle de vie d'une applet



Programmation d'une Applet

- **init() :** Méthode appelée (une seule fois) par le navigateur lorsque l'applet est chargée. Cette méthode rassemble toutes les initialisations nécessaires à l'applet. Elle est comparable à un constructeur. En principe, cette méthode doit traiter les valeurs PARAM passée dans le code HTML et ajouter les composants de l'interface utilisateur.
- La **méthode init() de Applet** devra être redéfinie à cette occasion
  - **public void init(){...}**

Programmation d'une Applet

start() : Méthode appelée par le navigateur pour demander l'activation de l'applet après l'appel de la méthode ***init()*.** Elle est appelée chaque fois que l'utilisateur replace la page contenant l'applet au premier plan.

- une applet peut **être démarrée plusieurs fois** pendant sa durée de vie
- la méthode : **public void start(){…}** est à redéfinir pour y intégrer le code à exécuter
- les actions possibles dans la méthode **start()** sont :
  - lancement d'un Thread pour le contrôle de l'applet
  - l'envoi des messages appropriés aux objets de l'applet
  - ...

Programmation d'une Applet

- paint() : Méthode appelée par le navigateur pour demander l'exécution du processus d'affichage des composants de l'applet. Elle est appelée à chaque fois que l'applet devient active, ou lors d'une modification du contexte graphique lié à l'applet (redimensionnement de la fenêtre, par exemple).
- Cette méthode prend en paramètre un objet de type Graphics
  - public void paint(Graphics g){}

Programmation d'une Applet

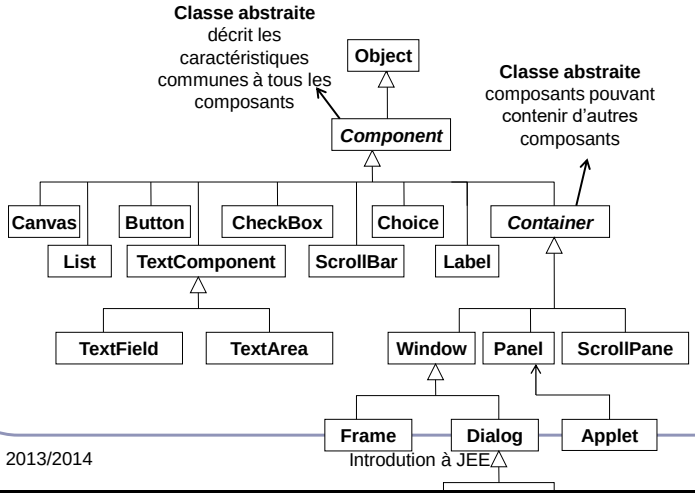
- stop() : arrêt d'une Applet
  - une applet peut **être arrêtée autant de fois qu'il y'a de redémarrage**
  - La méthode : **public void stop()** permet de stopper une applet lorsque la page qui la lancée est inactive
- destroy() : Destruction d'une applet
  - La méthode **public void destroy()** permet de libérer une applet et les ressources qu'elle a mobilisées
  - l'appel de *destroy* se produit si le navigateur est interrompu ou si l'applet doit être rechargée

Programmation d'une Applet

- **La méthode repaint()**
  - Cette méthode permet de demander explicitement au navigateur de redessiner l'applet (bouton Reload dans le navigateur)
  - Cette méthode appelle une autre méthode appelée **update()** qui elle en final appelle **paint()**
- **la méthode update**
  - Cette méthode est normalement appelée par **repaint()** et elle appelle ensuite **paint().** Sa tâche consiste, avant d'appeler **paint(),** de faire **le "ménage"** sur l'affichage courant de l'applet
  - Note : dans les cas courants, il n'est pas nécessaire de redéfinir **update()**
- **Le Thread AWT**
  - un Thread AWT est responsable du tracé de tous les composants de l'applet ainsi que de la répartition des événements en entrée. Il sera donc utile de répartir ce travail sur plusieurs THREAD pour le décharger

Applet ?

- Hiérarchie de classes couvrant les composants de awt

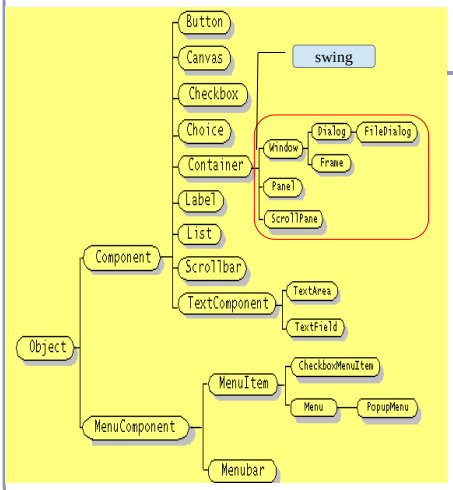


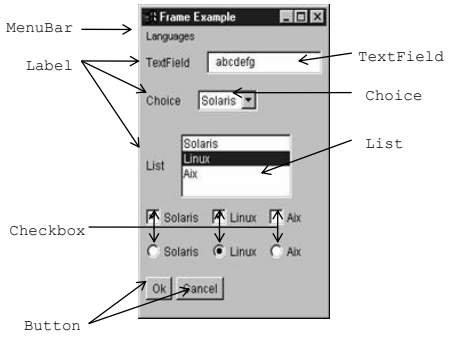
Interface graphiques

Abstract Window Toolkit (AWT)
Classe de base des awt : la classe abstraite Component.

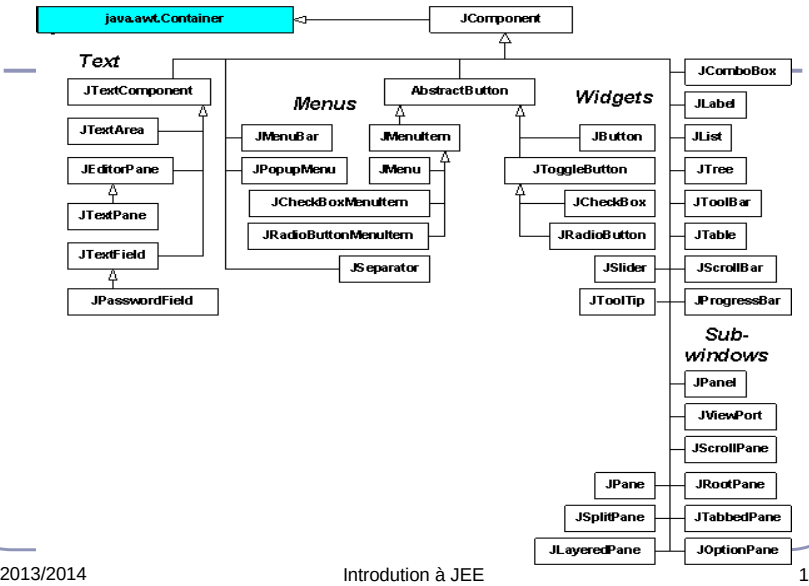
Swing
Classe de base des composants swing : JComponent.

AWT


Button
Canvas
Checkbox
Choice
Container
Label
List
Component
Scrollbar
TextComponent
TextArea
TextField
Object
MenuItem
CheckboxMenuItem
Menu
PopupMenu
MenuComponent
Menubar
swing
Window
Dialog
FileDialog
Frame
Panel
ScrollPane


MenuBar
Label
TextField
Choice
List
Checkbox
Button
Frame Example
Languages
TextField
abcdefg
Choice
Solaris
List
Solaris
Linux
Aix
Solaris
Linux
Aix
Solaris
Linux
Aix
Ok
Cancel

SWING


java.awt.Container
JComponent
Text
JTextComponent
JTextArea
JEditorPane
JTextPane
JTextField
JPasswordField
Menus
AbstractButton
JMenuBar
JMenuItem
JPopupMenu
JMenu
JCheckBoxMenuItem
JRadioButtonMenuItem
JSeparator
Widgets
JButton
JToggleButton
JCheckBox
JRadioButton
JSlider
JToolTip
JComboBox
JLabel
JList
JTree
JToolBar
JTable
JScrollBar
JProgressBar
Sub-windows
JPanel
JViewPort
JScrollPane
JPane
JRootPane
JSplitPane
JTabbedPane
JLayeredPane
JOptionPane

Positionnement des composants

Les instances dérivées de la classe **Applet** héritent (de la classe **Container**) de deux fonctions: **setLayout** et **getLayout**

```
public void setLayout(LayoutManager mgr)
public LayoutManager getLayout()
```

LayoutManager *(gestionnaires de géométrie)*

Les composants sont insérés dans l'Applet au moyen de sa méthode **add**

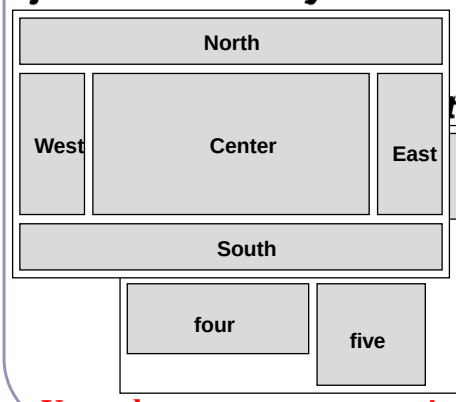
```
Component add(Component comp)
```

Exemple:

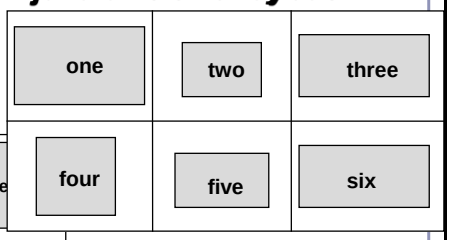
```
Button b = new Button("Test1");
add(b);
```


LayoutManager

java.awt.BorderLayout



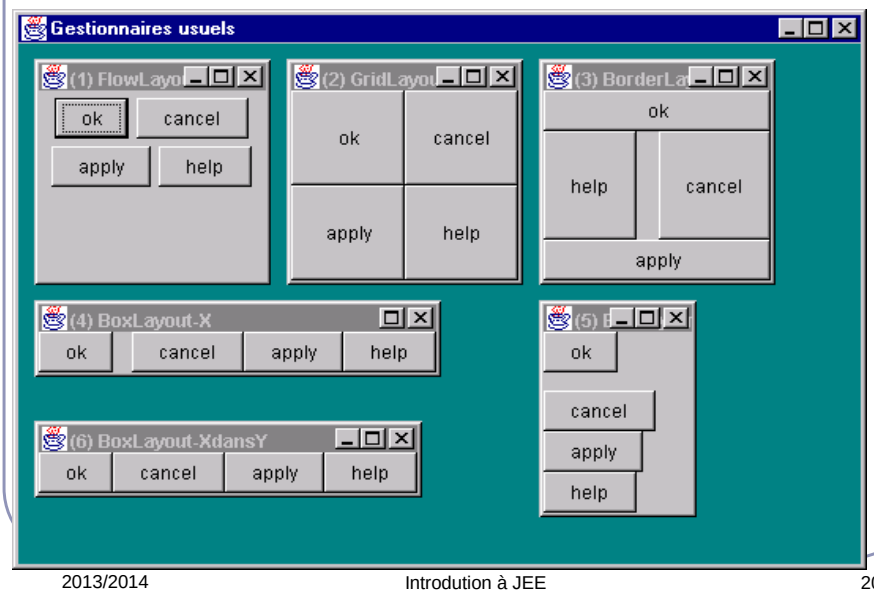
java.awt.GridLayout



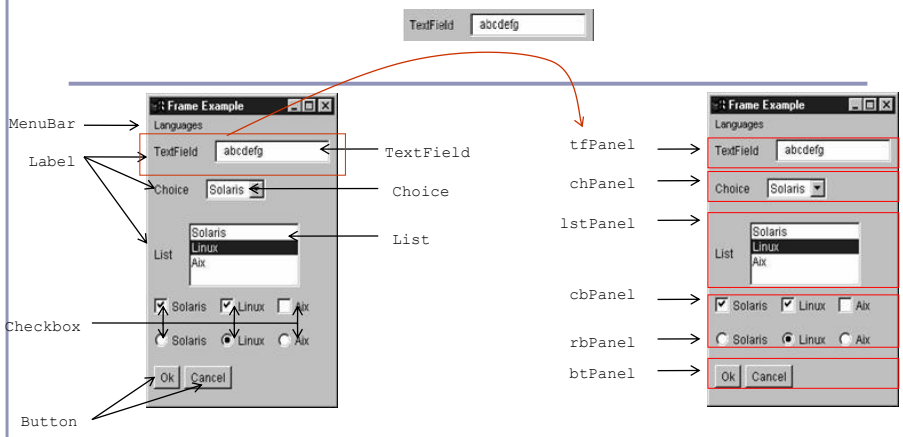
- *BoxLayout*
- *GridBagLayout*
- *CardLayout*

Un seul composant par zone !

LayoutManager



Panel pour regrouper

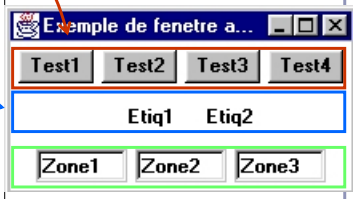


Dans une Applet

```
import java.awt.*;
import java.applet.Applet;

public class Essai extends Applet {

public void init(){
        setLayout(new BorderLayout());
        Panel p=new Panel();
        p.setLayout(new FlowLayout ());
        p.add(new Button("Test1"));
        p.add(new Button("Test2"));
        p.add(new Button("Test3"));
        p.add(new Button("Test4"));
        add("North",p);
        Panel p2=new Panel();
        p2.setLayout(new FlowLayout ());
        p2.add(new Label("Etiq1"));
        p2.add(new Label("Etiq2"));
        add("Center",p2);
        Panel p3=new Panel();
        p3.setLayout(new FlowLayout ());
        p3.add(new TextField("Zone1"));
        p3.add(new TextField("Zone2"));
        p3.add(new TextField("Zone3"));
        add("South",p3);
        }
}
```

Dans une application JAVA graphique

```
import java.awt.*;
class Fpanel {
 static public void main (String arg [ ]) {
 Frame w = new Frame("Exemple de fenetre ") ;
        w.setLayout(new BorderLayout());
        Panel p=new Panel();
        p.setLayout(new FlowLayout ());
        p.add(new Button("Test1"));
        p.add(new Button("Test2"));
        p.add(new Button("Test3"));
        p.add(new Button("Test4"));
        w.add("North",p);
        Panel p2=new Panel();
        p2.setLayout(new FlowLayout ());
        p2.add(new Label("Etiq1"));
        p2.add(new Label("Etiq2"));
        w.add("Center",p2);
        Panel p3=new Panel();
        p3.setLayout(new FlowLayout ());
        p3.add(new TextField("Zone1"));
        p3.add(new TextField("Zone2"));
        p3.add(new TextField("Zone3"));
        w.add("South",p3);
        w.show();
        w.pack(); w.addWindowListener(newWindowAdap
ter(){ public void windowClosing(WindowEvent e){
                System.exit(0);} });
```



Interface MouseListener

L'interface **MouseListener** permet à une classe de répondre aux événements souris de base. Elle est définie dans le package **java.awt.event** qu'il faut donc importer. Elle contient les méthodes suivantes :

```
public void mouseClicked(MouseEvent e)
public void mousePressed(MouseEvent e)
public void mouseReleased(MouseEvent e)
public void mouseEntered(MouseEvent e)
public void mouseExited(MouseEvent e)
```

```
import java.awt.*;
import java.awt.event.*;
public class drag extends java.applet.Applet
    implements MouseListener {
....

public void init() {
addMouseListener(this);
}
....
}
```


Exemple : MouseListener

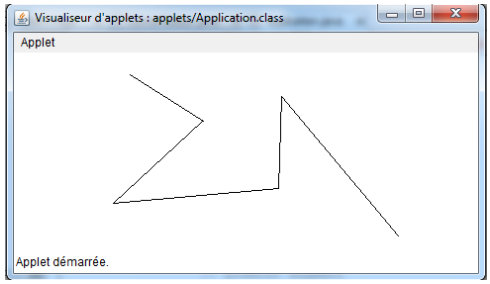
```
import java.applet.*;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class Application extends Applet implements MouseListener{

int xd[], yd[];                          // debuts des segments
int xf[], yf[];                          // fin des segments
static final int nsegments = 10;         // nbre de segments max
int n, ns;                           // pour compter les segments

public void init() {
    // création des tableaux
    xd = new int[nsegments];
    yd = new int[nsegments];
    xf = new int[nsegments];
    yf = new int[nsegments];
    // premier segment
    ns = n = 0;
    addMouseListener(this);
}
```

Exemple : MouseListener

```
public void mouseEntered(MouseEvent arg0) { }

public void mouseExited(MouseEvent arg0) { }

public void mousePressed(MouseEvent arg0) {
    xd[n] = arg0.getX();
    yd[n] = arg0.getY();
    }

public void mouseReleased(MouseEvent arg0) {
    xf[n] = arg0.getX();
    yf[n] = arg0.getY();
    n = (n+1)%nsegments;    // prochain point
    ns++;
    if (ns>=nsegments)ns = nsegments-1;
    repaint();          // forçage du dessin de l'applet
}

public void mouseClicked(MouseEvent arg0) {
    // récupération du click droit
    if (arg0.getButton()==3){   // click droit
        n = ns = 0;
        }
    }
```


Exemple : MouseListener

```
public void paint(Graphics g){
    // affichage de tous les segments
    for (int i=0 ; i<ns ; i++)
            g.drawLine(xd[i], yd[i], xf[i],  yf[i]);
}
```



Interface MouseMotionListener

L'interface **MouseMotionListener** permet à une classe de répondre aux événements liés au déplacement de la souris. Elle est définie dans le package java.awt.event qu'il faut donc importer.

Elle contient les méthodes suivantes :

```
public void mouseDragged(MouseEvent e)
public void mouseMoved(MouseEvent e)
```

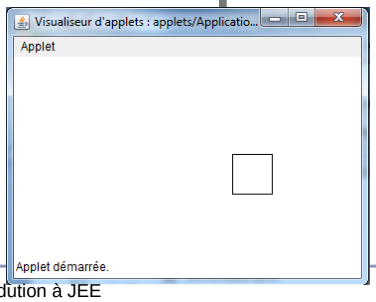
```
import java.awt.*;
import java.awt.event.*;
public class drag extends java.applet.Applet
    implements MouseListener, MouseMotionListener {
....

public void init() {
addMouseListener(this);
addMouseMotionListener(this);
}
....
}
```


Exemple : MouseMotionListener

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;

public class Application1 extends Applet implements MouseMotionListener {
int x, y;
int state;

public void init(){
    state = 0;
    addMouseMotionListener(this);
}

public void mouseDragged(MouseEvent arg0) {
    state = 2;
    x = arg0.getX();
    y = arg0.getY();
    repaint();

}
```

Exemple : MouseMotionListener

```
public void mouseMoved(MouseEvent arg0) {
state = 1;
x = arg0.getX();
y = arg0.getY();
repaint();
}

public void paint (Graphics g){
switch (state){
    case 1 : g.drawOval(x-20,  y-20, 20, 20);
break;
    case 2 : g.drawRect(x-20,  y-20, 40, 40);
break;
    }
}



}
```



Interface KeyListener

L'interface **KeyListener** permet à une classe de répondre aux événements liés au clavier (appui sur une touche, relâchement, ...).

Elle est définie dans le package java.awt.event qu'il faut donc importer.

Elle contient les méthodes suivantes :

```
public void keyTyped(KeyEvent e)
public void keyPressed(KeyEvent e)
public void keyReleased(KeyEvent e)
```

```
import java.awt.*;
import java.awt.event.*;
public class drag extends java.applet.Applet
    implements KeyListener {
....

public void init() {
addKeyListener(this);
}
....
}
```

Interface ActionListener

L'interface **ActionListener** permet à une classe de répondre à des actions faites sur des composants qu'elle contient (bouton, menus, .etc.). Elle est définie dans le package java.awt.event qu'il faut donc importer.

Elle contient les méthodes suivantes :

```
public void actionPerformed(ActionEvent evt)
```

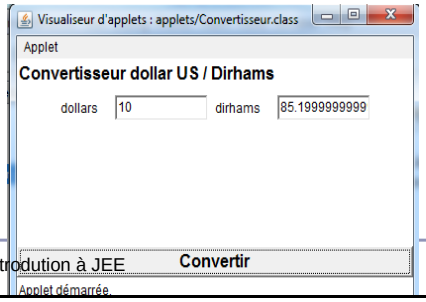
```
import java.awt.*;
import java.awt.event.*;
public class drag extends java.applet.Applet
    implements ActionListener {
....

public void init() {
addActionListener(this);
}
....
}
```


un convertisseur dollar/dirhams

```
import java.applet.Applet;
import java.awt.event.*;
import java.awt.*;

public class Convertisseur extends Applet implements ActionListener {
 static double tauxChangeDollarDirhams=8.52;
 TextField dollars;
 TextField dirhams;

public void init(){
     dollars = new TextField(10);
     dirhams = new TextField(10);
     this.setLayout(new BorderLayout());    // placement nord/sud/est/west

     Label l = new Label("Convertisseur dollar US / Dirhams");
     l.setFont(new Font("SansSerif",Font.BOLD,16));
     add("North", l);

     Panel p = new Panel();
     p.setLayout(new FlowLayout());
     p.add(new Label("dollars"));
     p.add(dollars);
     p.add(new Label("dirhams"));
     p.add(dirhams);
     add("Center", p);

     Button b = new Button("Convertir");
     b.setFont(new Font("SansSerif",Font.BOLD,16));
     add("South", b);
     b.addActionListener(this);
}
```

un convertisseur dollar/dirhams

```
public void actionPerformed(ActionEvent arg0) {
    if (arg0.getActionCommand()=="Convertir"){// le bon bouton
        try {
            double valeur = Double.valueOf(dollars.getText());
            valeur = valeur*tauxChangeDollarDirhams;
            dirhams.setText(String.valueOf(valeur));
            } catch (NumberFormatException nx) {
                dirhams.setText("impossible");
                }
    }
}

}
```



Programmation d'une Applet

- Exemple 1 : Applet simple

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;
import java.applet.Applet;

public class HelloWorldApplet extends java.applet.Applet{
    Font f=new Font("TimesRoman", Font.BOLD, 24);

    public void paint(Graphics g){
        g.setFont(f);
        g.setColor(Color.blue);
        g.drawString("HELLO WORLD", 10, 20);
    }
}
```

Programmation d'une Applet

- En jdk c'est l'outil appletviewer qui permet d'exécuter les applets
- Exemple 2 : Applet et image

```
import java.awt.*;
import java.applet.Applet;

public class TestImage    extends java.applet.Applet{
      Image digit;

      public void init(){
            // getDocumentBase() renvoie l'URL absolu de l'image
            digit = getImage(getDocumentBase(), "image.gif");
      }
      public void paint(Graphics g){
            g.drawImage(digit, 15, 15, this);
      }
}
```


Applet et application

Fonctionnement

- Une fois que le code de l'applet a été téléchargée et est en mémoire sur le poste client, comment s'exécute l'applet ?
- Le navigateur crée une instance de la classe de l'Applet en appelant le constructeur sans argument de la classe puis adresse à cet objet les méthodes init(), start() et paint() puis le cycle de vie continue en suivant les actions de l'utilisateur jusqu'à l'exécution de destroy lors de l'arrêt de la machine virtuelle
- Grande différence avec une application locale autonome :
  - On n'écrit pas de méthode main.
  - C'est le navigateur qui gère le cycle de vie de l'applet

Programmation d'une Applet

- Une applet ne peut donc être exécutée que si elle est intégrée dans une page HTML et elle s'exécute sous le contrôle d'un navigateur
- Une applet java fait toujours intervenir au moins deux fichiers
  - un fichier java compilé : prog.class
  - un fichier HTML (prog.html) qui contient la balise <APPLET> avec la référence au code prog.class
- Pour la mise au point d'une applet, on peut utiliser l'outil de jdk appletviewer

Inclusion d'une Applet dans une page WEB

- L'étiquette <APPLET...>
  - L'inclusion d'une Applet dans une page WEB se fait à partir de l'étiquette <APPLET > … </APPLET>
  - Cette extension permet d'indiquer le nom de l'applet java à activer (le fichier *.class contenant la méthode init())
  - Exemple d'une page simple

```
<HTML><HEAD>
<TITLE> Page avec APPLET </TITLE>
</HEAD>
<BODY>
<APPLET CODE ="MyApplet.class" WIDTH=200 HEIGHT = 100>
</APPLET>
</BODY>
</HTML>
```

Transmission de paramètres à une Applet

- La section <APPLET … /APPLET>
  - Elle comporte un certain nombre de paramètres permettant de transmettre des informations à l'applet

<Applet
code = **FichierApplet.class**
width= **pixels** height= **pixels**
[codebase= **codebaseURL**]
[alt= **alternateText**]
[name = **appletInstanceName**]
[align= **alignement**]
[vspace= **pixels**] [hspace= **pixels**]
>
[<param name = **AppletAttribute1** value= **value1** >]
...
[<param name = **AppletAttributeN** value= **valueN** >]
[alternateHTML]
</applet>

Transmission de paramètres à une Applet

- La section <APPLET … /APPLET>
  - code = **FichierApplet.class**
    - cet attribut permet de désigner le nom du fichier *.class java contenant la méthode init. Il est obligatoire
    - width= **pixels** height=**pixels** ces 2 paramètres permettent de dimensionner en pixel la zone d'affichage de l'applet. Ils sont obligatoires
    - [codebase= **codebaseURL**]. Ce paramètre permet d'indiquer l'URL de base de l'applet : répertoire contenant le code de l'applet. Facultatif.
    - [alt= **alternateText**]. Ce paramètre contient le texte que doit afficher si le navigateur ne réussit pas à exécuter l'applet java. Facultatif
    - [name = **appletInstanceName**] ce paramètre fournit un nom pour l'instance de l'applet. Plusieurs applets situer sur la même page peuvent communiquer à travers ce nom

Transmission de paramètres à une Applet

- La section <APPLET … /APPLET>
  - [align= **alignement**]
    - permet de préciser l'alignement de l'applet dans la page. Alignement peut valoir : left, middle, texttop, top, absmiddle, baseline, bottom, absbottom
      - Note : Facultatif
  - [vspace= **pixels**] [hspace= **pixels**]
    - ces paramètres permettent d'indiquer le nombre de pixels vertical (en dessus ou en dessous de l'applet) et horizontal (à droite et à gauche de l'applet)
    - [<param name = **AppletAttribute1** value= **value** >]
    - Paramètres de l'applets. La récupération des paramètres se fait via la méthode getParameter
    - Exemple :

```
<applet code = Essai.class width=100 hight=50>
<param name = font   VALUE="TimesRoman">
<param name=size     VALUE="24">
</applet>
```

      - **Récupération dans l'applet Java**

```
String nomFont = getParameter("font"); ...
```


Exemple de paramètres (2)

Fichier MonApplet.html

```
<HTML><HEAD>
<TITLE> Page avec APPLET </TITLE>
</HEAD>
<BODY>
<APPLET
CODE= "AppletAvecParam.class"
WIDTH=200 HEIGHT=100>
<PARAM NAME=nom VALUE= "Jacques Simard">
</ APPLET >
</BODY>
</HTML>
```

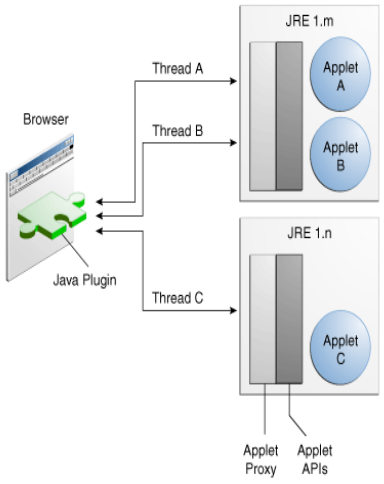
Commande : C:\ > appletviewer MonApplet.html

Exemple de paramètres (1)

```
import javax.swing.*;
import java.awt.*;
public class AppletBonjourParam extends JApplet {
 JLabel label;

public void init() {
    String nomPers= this.getParameter("nom");
    if (nomPers==null)
          label = new JLabel("Bonjour", JLabel.CENTER);
    else label = new JLabel("Bonjour "+nomPers ,JLabel.CENTER);
    Container c=this.getContentPane();
    c.add(label, BorderLayout.CENTER);
}
}
```


Applet et environnement d'exécution

- Le plug-in Java crée un thread pour chaque applet. Il lance une applet dans une instance de l'environnement d'exécution Java (JRE).
- Normalement, toutes les applets s'exécutent dans la même instance JRE. Le plug-in Java démarre une nouvelle instance de JRE si :
- 1- Une applet demande à être exécutée dans une version spécifique de JRE.
- 2-Une applet spécifie ses propres paramètres de démarrage JRE (par exemple, la taille du tas).



méthodes de JApplet

getCodeBase() : retourne l'URL du répertoire à partir duquel le code de l'applet a été chargé.

getDocumentBase() : retourne l'URL du répertoire où se trouve le document HTML qui contient l'applet

Exemple : si l'applet est dans le document http://java.sun.com/products/jdk/1.2/index.html

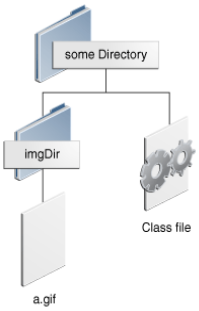
l'URL retournée est : http://java.sun.com/products/jdk/1.2/

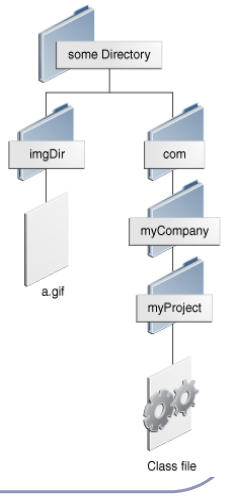
Ces 2 méthodes retournent la même URL (même serveur, même répertoire) s'il n'y a pas d'attribut CODEBASE

Autres méthodes de JApplet

Image image = getImage (getCodeBase(), "imgDir/a.gif");

retourne une instance d'image correspondant au fichier image sous imgDir par rapport à une base d'URL.





Transmission de paramètres à une Applet

- Exemple complet : insertion du son

```
import java.awt.*;
import java.applet.*;
import java.net.URL;
import java.awt.event.*;
public class TestSon extends Applet{
      boolean    boucle    = false;
      AudioClip audio=null;
      private Button sound, loop, stop;
      public void init(){
            this.add(sound=new Button("Sound"));
            this.add(loop=new Button("Loop"));
            this.add(stop=new Button("Stop"));
            this.setBackground(Color.blue);
            sound.addActionListener(new ActionL());
            loop.addActionListener(new ActionL());
            stop.addActionListener(new ActionL());
            try{AppletContext  ac =  getAppletContext();
                  if (ac != null)audio = ac.getAudioClip(new URL(getCodeBase(), "images/spacemusic.au"));
                  else System.out.println("Fichier sonore introuvable");
            }catch(java.net.MalformedURLException e){}
}
```


Transmission de paramètres à une Applet

- Exemple complet (suite)

```
      public void start(){
            if (audio != null )
                  if (boucle == false)  audio.play();
                  else  audio.loop();
      }
      public void stop(){
            if (audio != null )    audio.stop();
      }
      class ActionL implements ActionListener{
            public void actionPerformed(ActionEvent e){
                  if (e.getSource() == sound){boucle = false;   start();}
                  if (e.getSource() == loop){boucle = true; start(); }
                  if (e.getSource() == stop){stop();}
            }
      }
}
```

Animation d'un logo

Exemple :

```
import java.applet.*;
import java.awt.*;
public class AppletAnimation extends Applet implements Runnable
    {Thread thread;
     protected Image tabImage[];
     protected int index;
     public void init() {
       super.init();
       //chargement du tableau d'image
       index = 0;
       tabImage = new Image[2];
       for (int i = 0; i < tabImage.length; i++) {
       String fichier = new String(" images/monimage" + (i + 1) + ".gif ");
       tabImage[i] = getImage(getCodeBase(), fichier);
       }
     }
```

La surcharge de la méthode paint() permet d'éviter le tremblement de l'écran du à l'effacement de l'écran et à son rafraichissement.

Animation d'un logo (2)

```
public void paint(Graphics g) {
    super.paint(g);
      g.drawImage(tabImage[index], 10, 10, this); // affichage de l'image
    }
public void run() { //traitements exécuté par le thread
    while (true) { repaint(); index++;
      if (index >= tabImage.length) index = 0;
      try { Thread.sleep(500);} catch (InterruptedException e) {
                                        e.printStackTrace();}
    } }
public void start() { //demarrage du thread
if (thread == null) { thread = new Thread(this);thread.start();} }
public void stop() { // arret du thread
    if (thread != null) { //thread.stop();
                                thread = null;} }
public void update(Graphics g) {
    //la redéfinition de la méthode permet d'éviter les scintillements
    paint(g);} }
```

Applet et application

- Il faut rajouter une classe main à l'applet, définir une fenêtre qui recevra l'affichage de l'applet, appeler les méthodes init() et start() et afficher la fenêtre.

Exemple :

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class AppletApplication extends Applet implements WindowListener {
public static void main (java.lang.String[] args) {
    AppletApplication applet = new AppletApplication();
    Frame frame = new Frame("Applet");
    frame.addWindowListener(applet);
    frame.add("Center", applet);
    frame.setSize(350, 250);
    frame.setVisible(true);applet.init();applet.start();
}
public void paint(Graphics g) { super.paint(g);g.drawString("Bonjour", 10, 10);}
public void windowActivated(WindowEvent e) { }
public void windowClosed(WindowEvent e) { }
public void windowClosing(WindowEvent e) { System.exit(0);}
public void windowDeactivated(WindowEvent e) { }
public void windowDeiconified(WindowEvent e) { }
public void windowIconified(WindowEvent e) { }
public void windowOpened(WindowEvent e) { }
}
```

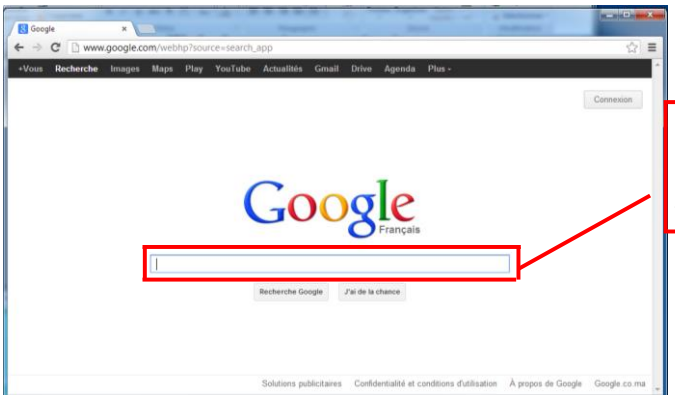

Les technologies côté serveur

- Il ya beaucoup de technologies disponibles côté serveur :
- Java (servlet, JSP, JSF, Struts, Spring, Hibernate),
- ASP,
- PHP,
- script CGI, et bien d'autres.
- Servlet Java est le *fondement* de la technologie Java côté serveur, JSP (JavaServer Pages), JSF (JavaServer Faces), Struts, Spring, Hibernate, et d'autres, sont des extensions de la technologie des servlets.

Les versions des Servlet

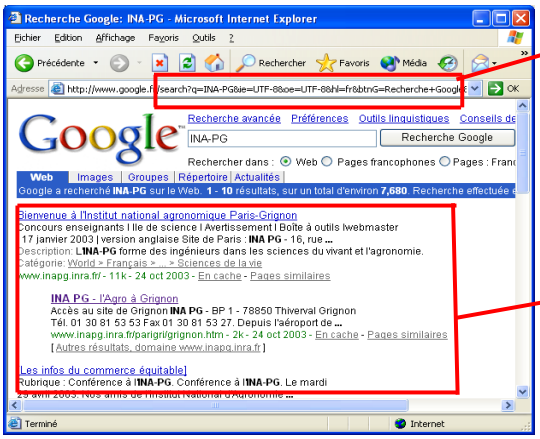
- J2EE 1.2 (12 Décembre 1999) (**Java Servlet 2.2** , JSP 1.1, EJB 1.1, JDBC 2.0)
- J2EE 1.3 (24 Septembre 2001) (**Java Servlet 2.3** , JSP 1.2, EJB 2.0, JDBC 2.1)
- J2EE 1.4 (11 Novembre 2003) (**Java Servlet 2.4** , JSP 2.0, EJB 2.1, JDBC 3.0)
- Java EE 5 (11 mai 2006) (**Java Servlet 2.5** , JSP 2.1, JSTL 1.2, JSF 1.2, EJB 3.0, JDBC 3.0)
- Java EE 6 (10 Décembre 2009) (**Java Servlet 3.0** , JSP 2.2/EL 2.2, JSTL 1.2, JSF 2.0, EJB 3.1, JDBC 4.0)
- Java EE 7: (septembre 2013) (**Java Servlet 3.1,** JSP 2.3/EL 3.0, JSF 2.2, HTML 5, EJB 3.2, JPA 2.1)

Web dynamique : principes

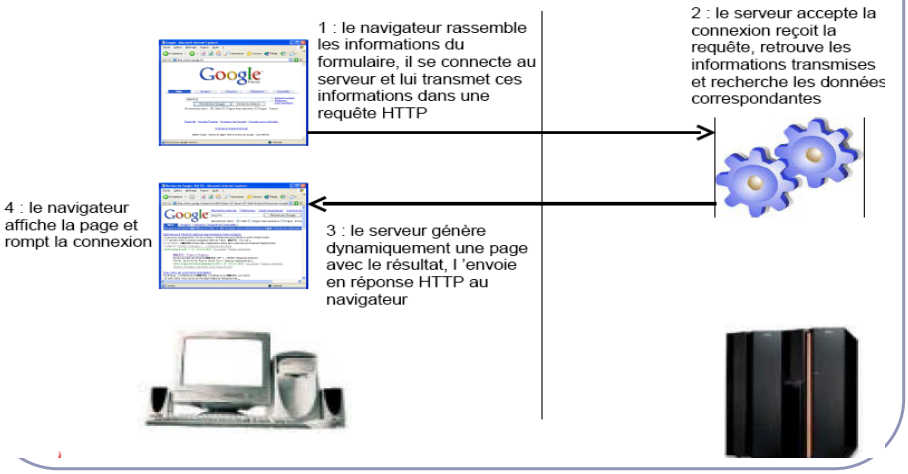


Formulaire :
-Champs de saisie
-Bouton de validation

Web dynamique : principes

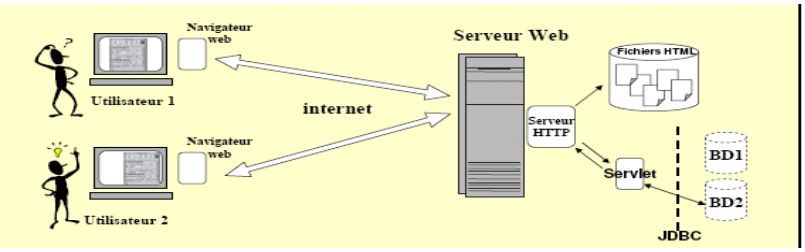


Web dynamique : principes

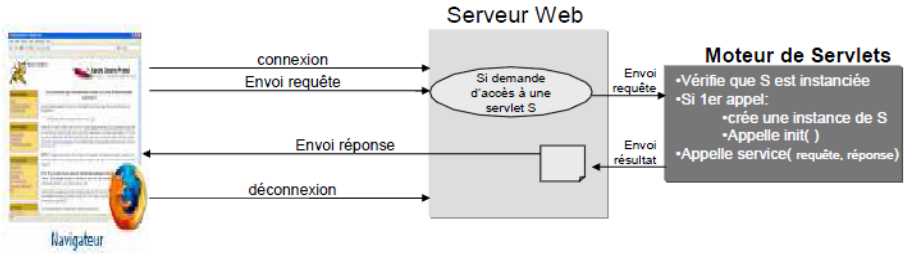


Les servlets : introduction

- Les servlets sont au serveur web ce que les applets sont aux navigateurs pour le client.
- Les servlets sont des applications java coté serveur, comme les CGI, ASP ou PHP.
- Les servlets permettent de gérer des requêtes HTTP et fournissent au client une réponse HTTP dynamique.



Les servlets : fonctionnement

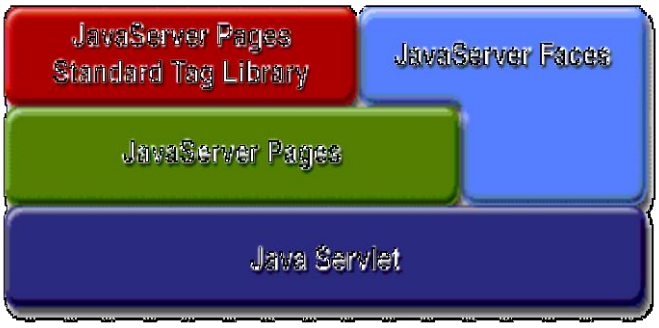


- Une Servlet est un programme Java qui lit les données envoyées par un client web et elle génère un résultat.
- Les données sont envoyées des formulaires ou des entêtes des requêtes.
- Lors de la création d'une réponse, une servlet :
  - peut utiliser toutes les fonctions du langage Java
  - peut communiquer avec des ressources externes (fichiers, BD, ...) et/ou d'autres applications

Avantage des servlets par rapport aux CGI

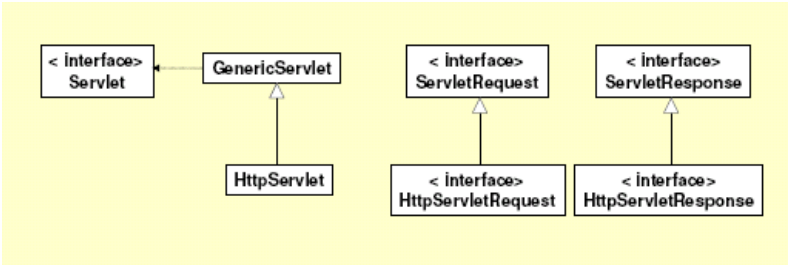
- Portabilité : indépendante système exploitation et des serveurs web
- Efficacité : semi compilée, multithread (plutôt que des processus pour les CGI), gestion du cache, connexions persistantes
- Puissance : communication bidirectionnelle avec le serveur web, partage de données entre servlets, chaînage de servlets, servlet modifiée peut être réactivée sans redémarrage du serveur
- Modularité : possibilité d'avoir plusieurs servlets, chacune pouvant accomplir une tâche spécifique
- Pratique : gestion des cookies, suivi des sessions, manipulation simple du protocole HTTP, peut dialoguer avec applets coté client via RMI

Les technologies

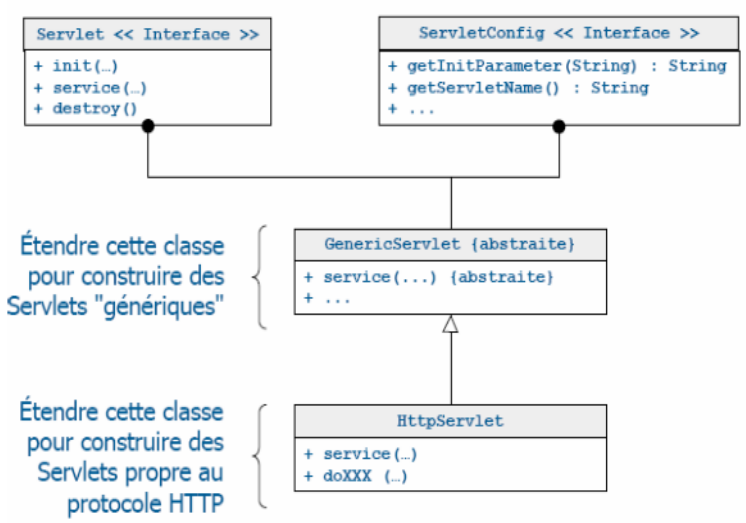


Package Servlet

- L'API pour les servlets est constituée de deux packages :
  - javax.servlet : package générique
  - javax.servlet.http : package spécifique pour les servlets HTTP

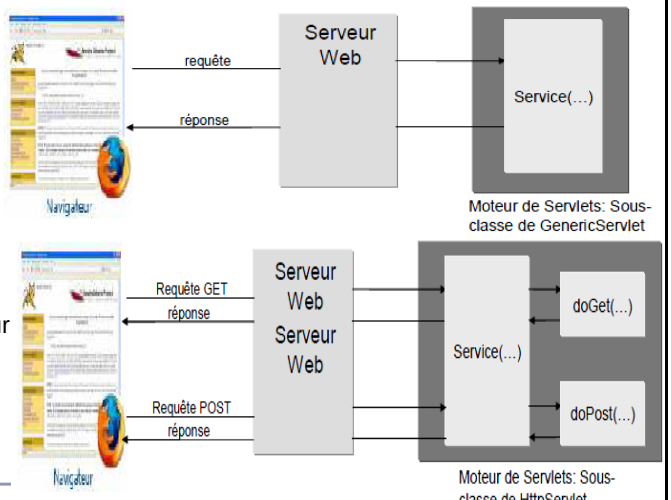


API des servlets



API servlet

- l'API Servlet fournit deux classes qui proposent déjà une implémentation:
  - GenericServlet : pour la conception de Servlets indépendantes du protocole
  - HttpServlet : pour la conception de Servlets spécifiques au protocole HTTP



Fonctionnement d'une servlet

- Lorsqu'une servlet est appelée par un client, la méthode *service()* est exécutée. Celle-ci est le principal point d'entrée de toute servlet et accepte deux objets en paramètres :
- l'objet *ServletRequest* encapsulant la requête du client, c'est-à-dire qu'il contient l'ensemble des paramètres passés à la servlet (informations sur l'environnement du client, cookies du client, URL demandée, ...)
- l'objet *ServletResponse* permettant de renvoyer une réponse au client (envoyer des informations au navigateur). Il est ainsi possible de créer des en-têtes HTTP (headers), d'envoyer des cookies au navigateur du client, ...

Développer une servlet

- Afin de développer une servlet fonctionnant avec le protocole HTTP, il suffit de créer une classe étendant *HttpServlet*.
- La classe *HttpServlet* (dérivant de *GenericServlet*) permet de fournir une implémentation de l'interface *Servlet* spécifique à HTTP.
- La classe *HttpServlet* surcharge la méthode *service* en lisant la méthode HTTP utilisée par le client, puis en redirigeant la requête vers une méthode appropriée.
- Les deux principales méthodes du protocole HTTP étant GET et POST, il suffit de surcharger la méthode adéquate afin de traiter la requête :
  - Si la méthode utilisée est GET, il suffit de redéfinir la méthode public void **doGet**(HttpServletRequest req, HttpServletResponse res);
  - Si la méthode utilisée est POST, il suffit de redéfinir la méthode public void **doPost**(HttpServletRequest req, HttpServletResponse res);

Servlet web : HttpServlet

- Une servlet HTTP doit hériter de la classe HttpServlet orientée développement Web.
- La classe HttpServlet possède des méthodes :
  - init(ServletConfig) : exécutée au chargement de la servlet
  - doGet(HttpServletRequest, HttpServletResponse) : exécutée à chaque connexion d'un navigateur (demande ressource)
  - doPost(HttpServletRequest, HttpServletResponse) : exécutée à chaque validation de formulaire (modifie la ressource)
  - destroy() : exécutée lors de l'arrêt du serveur
  - Les méthodes doGet(), doPost(), doPut(), doDelete(), doHead(), doOptions() et doTrace() utilisent des objets HttpServletRequest et HttpServletResponse passés en paramètres .
  - service.getLastModified() : vérifie si le contenu délivré par la servlet a changé

Lire la requête : *HttpServletRequest*

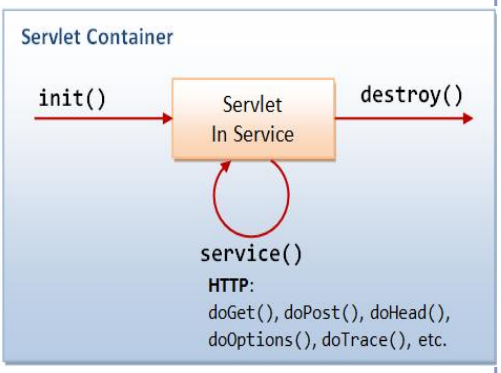
- Dans la méthode *doXXX()* (*doGet()* ou *doPost()* selon la méthode invoquée) la requête de l'utilisateur est passée en paramètres sous forme *HttpServletRequest*
- Les requêtes sont transmises du client au serveur par le protocole HTTP.
- les différentes méthodes de *HttpServletRequest* :
  - String getMethod() : récupère la méthode HTTP utilisée par le client
  - String getHeader(String Key) :récupère la valeur de l'attribut Key de l'en-tête
  - String getRemoteHost() : récupère le nom de domaine du client
  - String getRemoteAddr() : récupéré l'adresse IP du client
  - String getParameter(String Key) : récupère la valeur du paramètre Key (clé) d'un formulaire. Lorsque plusieurs valeurs sont présentes, la première est retournée
  - String[] getParameterValues(String Key) : récupère les valeurs correspondant au paramètre Key (clé) d'un formulaire, c'est-à-dire dans le cas d'une sélection multiple (cases à cocher, listes à choix multiples) les valeurs de toutes les entités sélectionnées
  - Enumeration getParameterNames() : retourne un objet *Enumération* contenant la liste des noms des paramètres passés à la requête
  - String getServerName() : récupère le nom du serveur
  - String getServerPort() : récupère le numéro de port du serveur

Créer la réponse : *HttpServletResponse*

- La réponse à fournir à l'utilisateur est représentée sous forme d'objet *HttpServletResponse*.
- Les différentes méthodes de l'objet *HttpServletResponse* sont :
  - String setStatus(int StatusCode) : définit le code de retour de la réponse
  - void setHeader(String Nom, String Valeur) : définit une paire clé/valeur dans les en-têtes
  - void setContentType(String type) : définit le type MIME de la réponse HTTP, c'est-à-dire le type de données envoyées au navigateur
  - void setContentLength(int len) : définit la taille de la réponse
  - PrintWriter getWriter() : retourne un objet *PrintWriter* permettant d'envoyer du texte au navigateur client. Il se charge de convertir au format approprié les caractères Unicode utilisés par Java
  - ServletOutputStream getOutputStream() : définit un flot de données à envoyer au client, par l'intermédiaire d'un objet *ServletOutputStream*, dérivé de la classe *java.io.OutputStream*
  - void sendredirect(String location) : permet de rediriger le client vers l'URL *location*

Servlet : cycle de vie

- Un client émet une requête destinée à une servlet
- Le serveur charge et exécute les classes Java qui génèrent des informations
- Le serveur retourne les informations au client
- Les états d'une servlet. Le passage d'un état à un autre est automatique et est fait par le conteneur de servlets.



Servlet : cycle de vie

- Une fois chargée en mémoire dans le moteur (conteneur) de servlets (i.e. la JVM), le conteneur lance la méthode service(…) à chaque requête.
- Les trois méthodes :
  - public void init(),
  - public void service(…),
  - public void destroy()

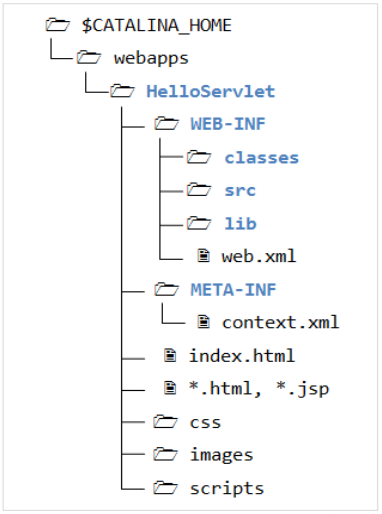
  sont définies dans la classe abstraite javax.servlet.GenericServlet.
- La méthode service(...) contient deux paramètres qui modélisent la requête et la réponse

HttpServlet : exemple 1

```
package mypkg;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Etape 1. Spécifier le type MIME du contenu de la réponse
response.setContentType("text/html");
// Etape 2. Récupère le PrintWriter pour envoyer des données au client
PrintWriter out = response.getWriter();
// Step 3. Envoyer l'information au client
out.println("<html>");
out.println("<head><title>Hello Servlet</title></head>");
out.println("<body>");
out.println("<h1> Hello à tous les étudiants SMI 6 !</h1>");
out.println("Il est : " + new java.util.Date());
out.println("</body></html>");
}
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException { doGet(request, response);
} }
```


Manipulation manuelle de Tomcat

- On lance le serveur web Tomcat
- La servlet compilée est rangée sous REP_INSTAL_TOMCAT\webapps\helloServlet\WEBINF\classes
- correspond à l'URL : http://localhost:8080/helloServlet/sayHello
- Mettre à jour web.xml

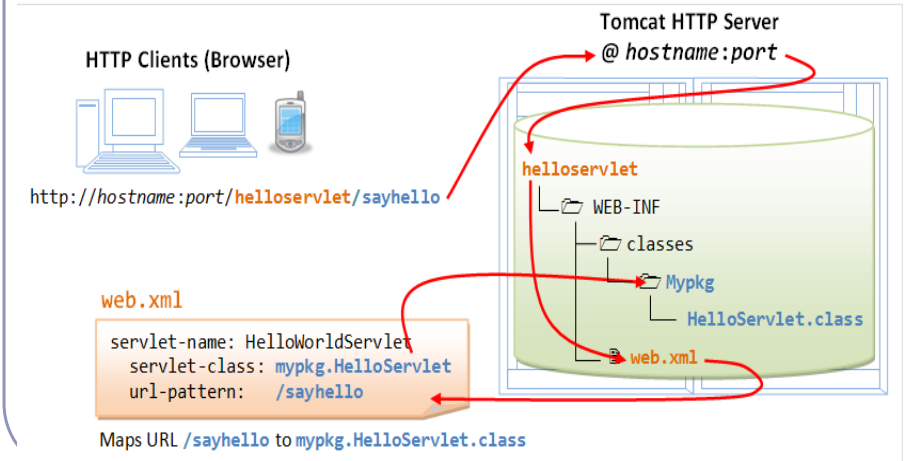


Web.xml

- Le fichier web.xml du répertoire WEB-INF de l'application web configure l'application web.
- L'élément <servlet> associe un nom à une classe Java servlet
- L'élément <servlet-mapping> associe à ce nom l'URL relative qui permet d'atteindre cette servlet.

```
<servlet>
    <servlet-name> HelloServlet </servlet-name>
   <servlet-class> mypkg.HelloServlet </servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name> HelloServlet </servlet-name>
    <url-pattern> /sayhello </url-pattern>
</servlet-mapping>
```


Fichier de description : web.xml



Entête du Web.xml

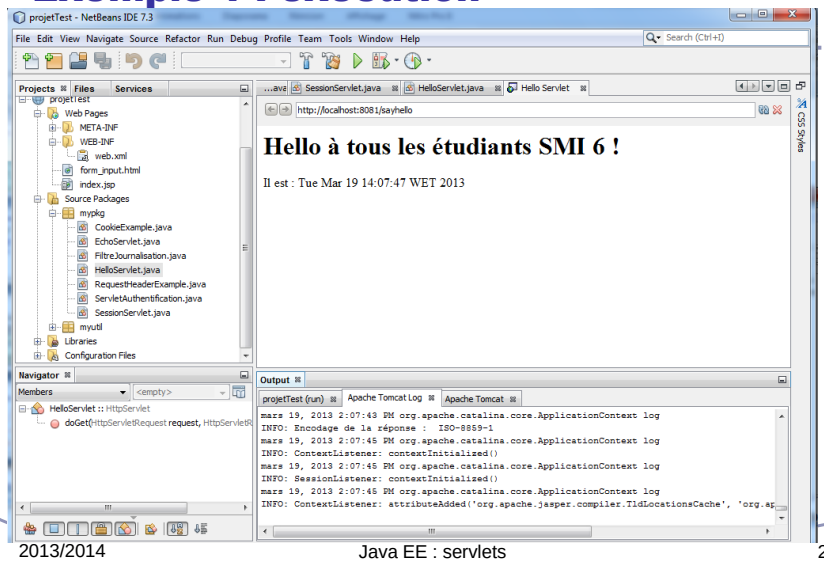
- **Syntaxe de " web.xml " en servlet 3.0 (**Tomcat 7 et Glassfish 3.1 **) :**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      metadata-complete="true">
  ......
</web-app>
```

- **Alors que sa syntaxe en servlet 2.5 (Tomcat 6 and Glassfish 3) :**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="2.5"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
   .......
</web-app>
```

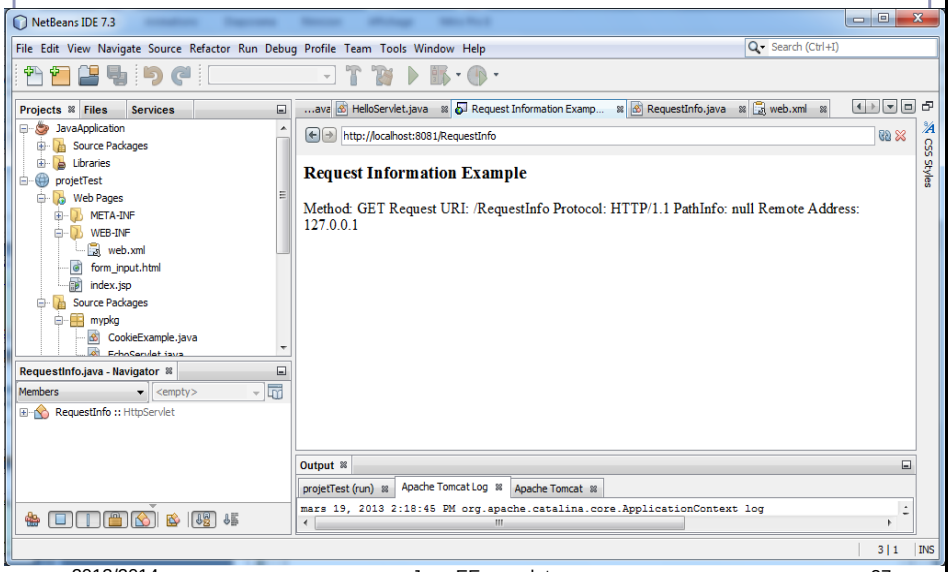

Exemple 1 : exécution



HttpServlet : exemple 2

```
//Source Code for RequestInfo Example
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RequestInfo extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException    {
        response.setContentType(" text/html ");
        PrintWriter out = response.getWriter();
        out.println(" <html> ");
        out.println(" <body> ");
        out.println(" <head> ");
        out.println(" <title>Request Information Example</title> ");
        out.println(" </head> ");
        out.println(" <body> ");
        out.println(" <h3>Request Information Example</h3> ");
        out.println(" Method: " + request.getMethod());
        out.println(" Request URI: " + request.getRequestURI());
        out.println(" Protocol: " + request.getProtocol());
        out.println(" PathInfo: " + request.getPathInfo());
        out.println(" Remote Address: " + request.getRemoteAddr());
        out.println(" </body> ");        out.println(" </html> ");
    }
/** We are going to perform the same operations for POST requests as for GET methods,
so this method just sends the request to the doGet method.     */
public void doPost(HttpServletRequest request, HttpServletResponse response)
                    throws IOException, ServletException { doGet(request, response);
}}
```


Exemple 2 :exécution

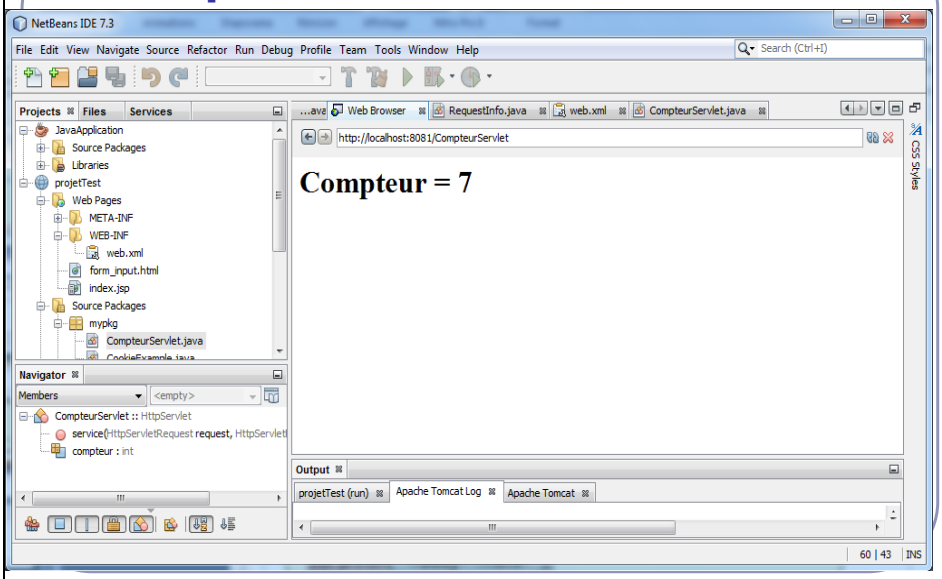


Servlet : exemple 3

- Chaque servlet n'est instanciée qu'une **seule fois**
- D'où la persistance de ses données entre 2 invocations

```
public class CompteurServlet extends HttpServlet {
int compteur = 0;
public void service( HttpServletRequest request,
HttpServletResponse response )
throws ServletException, IOException {
response.setContentType( "text/html" );
PrintWriter out = response.getWriter();
out.println( "<html><body>" );
out.println( "<h1> "+ " Compteur = "+compteur++ + "</h1>" );
out.println( "</body></html>" );
} }
```


Exemple 3 : exécution



Atelier 1 : servlet

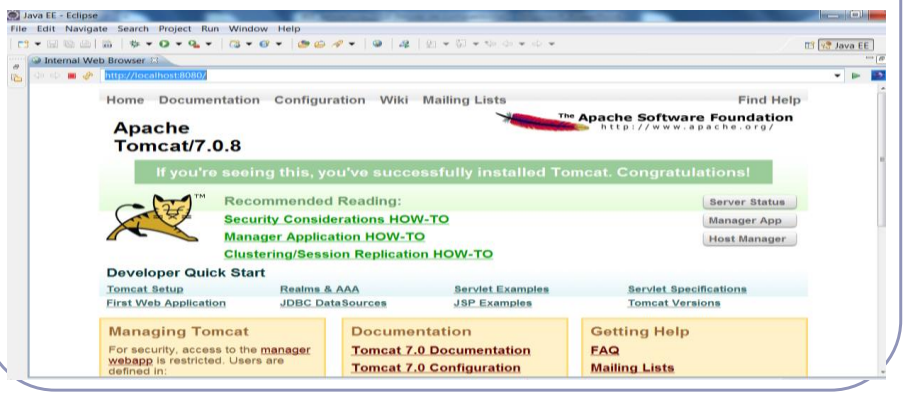
Les étapes pour construire un exemple de servlet en utilisant Eclipse et tomcat 7.x

Tomcat

- Un «conteneur web» (web container) J2EE
- Implémentation Java de référence
- Serveur open-source gratuit et multiplateformes(écrit en Java)
- Téléchargeable sur : http://jakarta.apache.org/tomcat
- Tomcat utilise la variable d'environnement JAVA_HOME qui désigne le dossier d'installation du JDK.
- Eclipse / window / préférences :
  - Tomcat: choisir version 7.x, répertoire d'installation de Tomcat, un fichier par contexte,
  - Rep_tomcat\conf\catalina\localhost

Vérification de l'installation

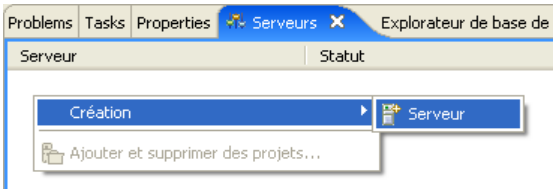
- Si tout s'est bien passé, en tapant http://localhost:8080/ dans le navigateur on obtient :



Pilotage de Tomcat à partir d'eclipse

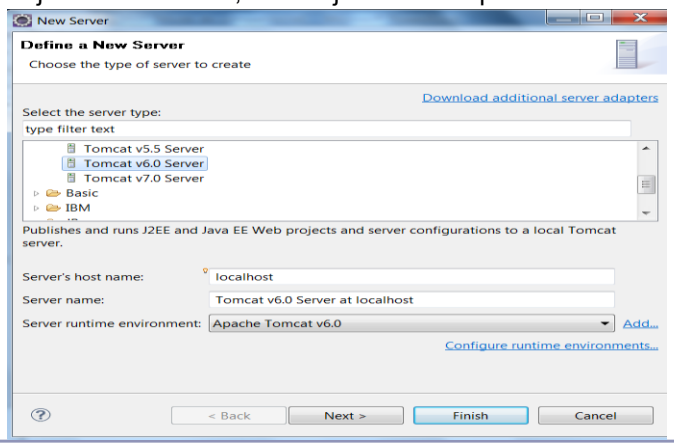
- Préciser à Eclipse le répertoire d'installation de Tomcat :
  - **Préférences->Server->Installed Runtimes** :

  (Par exemple C:\repInstal\Apache Software Foundation\Tomcat 7.0)
- Piloter tomcat à partir d'Eclipse :
  - Afficher la vue '**Serveurs**' : menu **Fenêtre->Afficher la vue->Autre...**, puis **Serveur->Serveurs.**
  - Dans la vue '**Serveurs**' menu contextuel :



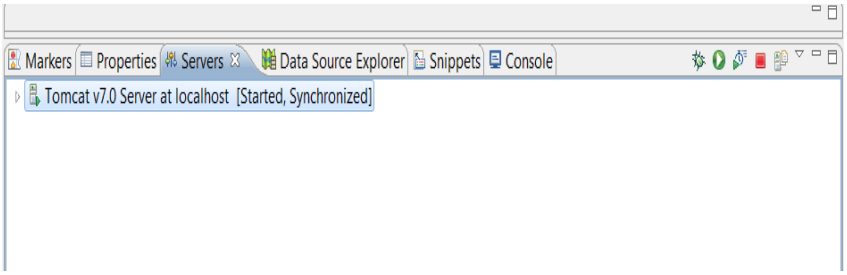
Ajout nouveau serveur

- Pour ajouter un serveur, il faut ajouter son répertoire d'installation.
-



Pilotage des serveurs à partir d'Eclipse

- La vue 'Serveurs' permet de piloter les serveurs déclarés, il est notamment possible de les démarrer en mode exécution ou débogage :

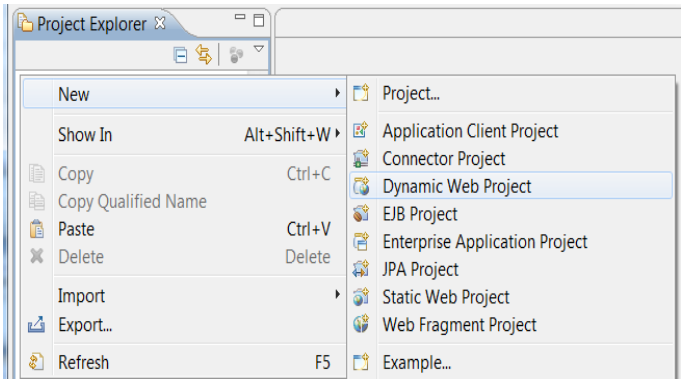


Vérification de l'installation dans navigateur interne à Eclipse

- Allez dans window de Eclipse, Show View, other…, General et Internal Web Browser, puis tapez http://localhost:8080/ dans le navigateur.
- En cas de **l'erreur 404** :
  - Allez dans le dossier : **votre-dossier-instal-tomcat\webapps** et copiez le dossier **root**
  - Allez dans : **votre-dossier-workspace\.metadata\.plugins\wst.server.core\tmp0\wtpwebapps** ou **[…\tmp1\wtpwebapps si vous avez un autre serveur déjà enregistré avec Eclipse**]
  - Collez dans ici le dossier **root** (dites yes pour fusionner et remplacer le dossier et les fichiers).
  - Copiez les dossiers **docs**, **examples** et **host-manager,** situé au même emplacement que root, dans : **votre-dossier-workspace\.metadata\.plugins\wst.server.core\tmp0\webapps**

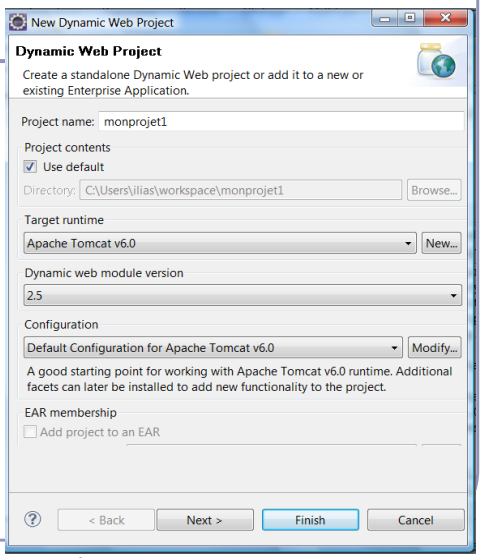
Création d'un projet web

- Menu contextuel de la vue 'Explorateur de projet', ensuite création de projet Web dynamique.
-



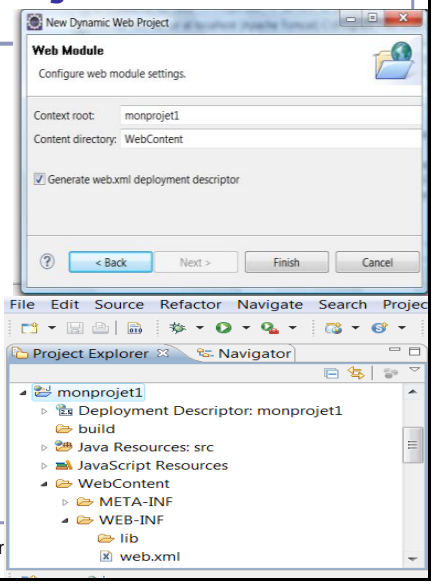
Création d'un projet web

- 1 - **Le nom du projet**.
  2 - **L'environnement d'exécution cible** : serveur associé au projet, pour ajouter les fichiers JAR nécessaires dans le chemin de compilation du projet.
  3 - **Appartenance à un EAR** : le projet Web dynamique correspond à un WAR, lors du déploiement un WAR puisse être stocké dans un fichier EAR. L'association d'un WAR avec un EAR est facultative. Tomcat ne supporte pas le format EAR.



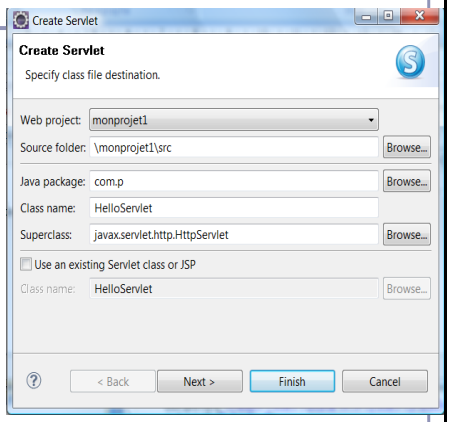
Création du projet web

1- Le nom de contexte : apparaît dans les URL permettant d'accéder à l'application. Par défaut, nom du projet.
2 - **Le nom du 'répertoire de contenu'** : un projet Web Dynamique n'a pas directement la structure d'un WAR. La structure du WAR se trouve dans un sous-répertoire du projet, (par défaut est 'WebContent'. Permet de stocker dans le projet des fichiers qui ne seront pas visibles par le serveur de test et qui ne seront pas exportés lors de la création du fichier WAR correspondant au projet. Par exemple, le code source n'est pas stocké dans le WAR, il est dans le sous-répertoire 'src'.



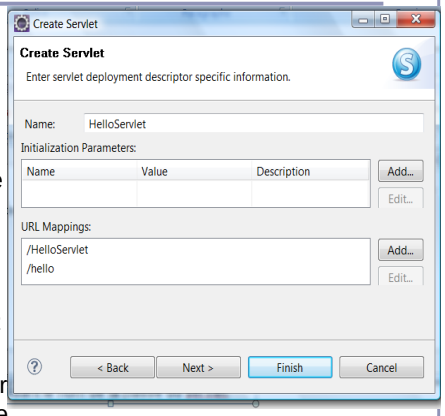
Développement d'une servlet

- Création d'une servlet : menu contextuel associé au projet Web, dans ce menu choisir 'Nouveau->Servlet'
- Saisir le nom du package et le nom de classe de la servlet
- Dans l'étape suivante



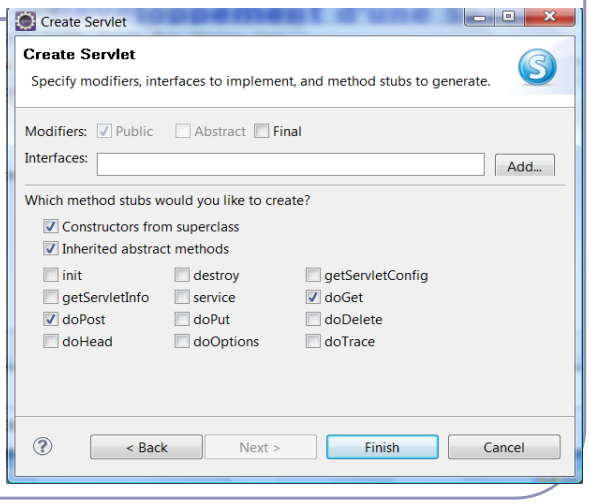
Développement d'une servlet

- La seconde étape permet de saisir des informations qui seront utilisées pour déclarer la servlet dans le descripteur de déploiement de l'application (fichier web.xml).
- La plus importante information est la liste des alias (champ 'Mappage d'URL'). Ces alias apparaîtront dans les URL permettant d'accéder à la servlet (la forme de ces URLs sera **http://nomDeServeur:port/nomDeContexte/Alias**).
- Une servlet peut avoir plusieurs alias, par défaut projet web définit nom de la classe de la servlet comme un alias.



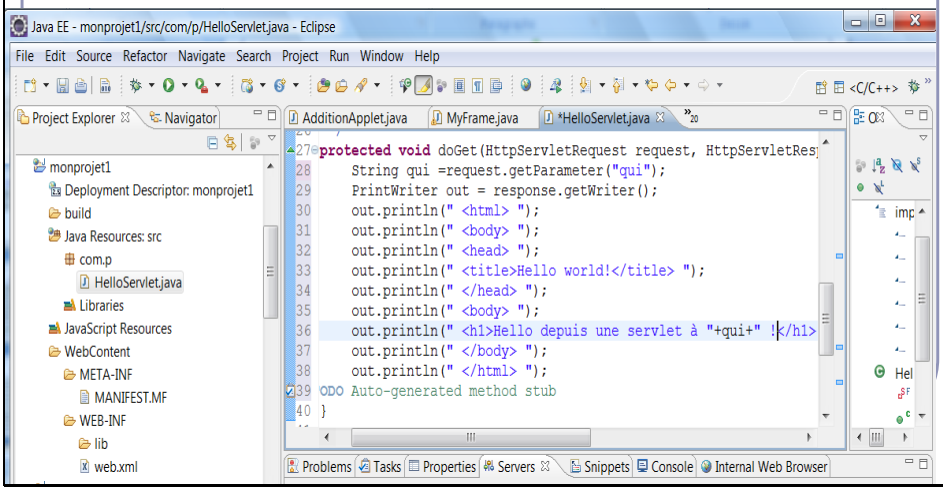
Développement d'une servlet

- La dernière étape permet de sélectionner les méthodes qui devront être générées lors de la création de la classe de la servlet :
- cliquer sur 'Terminer', la classe est crée. Cette nouvelle classe apparaît dans la vue 'Explorateur de projet' et est ouverte en édition.



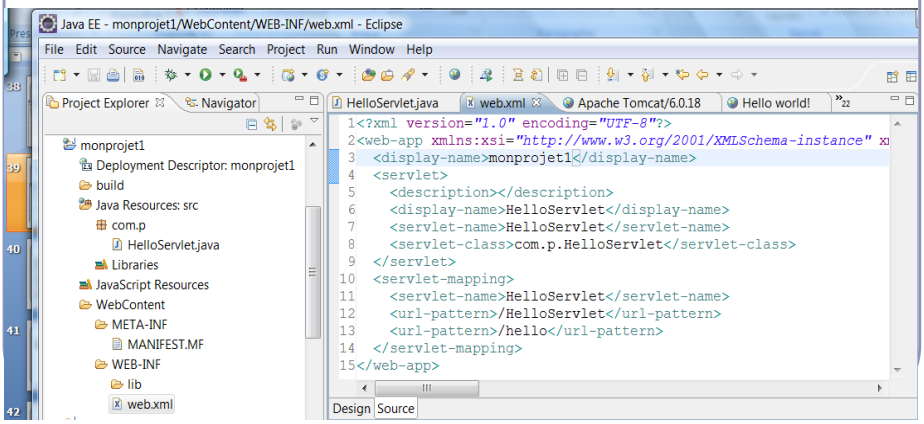
Développement d'une servlet

- Ajouter le code suivant dans la méthode **doGet** :



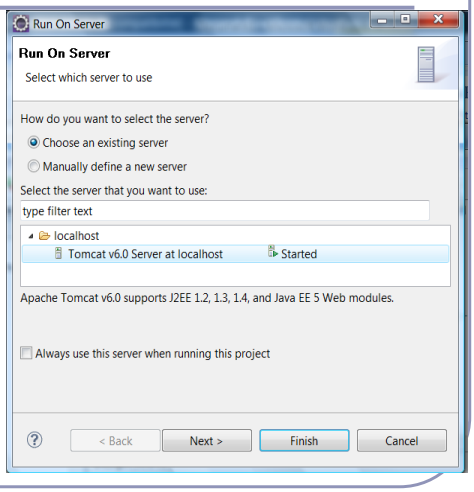
Développement d'une servlet

- Le fichier web.xml est mis à jour.



Test de la servlet

- Pour tester la servlet à partir de eclipse :
- menu contextuel associé à la classe de la servlet, dans ce menu choisir '**Exécuter en tant que->Exécuter sur le serveur**'.
- Le serveur peut aussi être lancé en mode debug en utilisant '**Deboguer en tant que->Deboguer sur le serveur**'.
- À la première utilisation, une boîte de dialogue permet d'indiquer le serveur sur lequel l'exécution doit se faire :



Test d'une servlet

- Un navigateur Web est ouvert dans la zone d'édition et invoque la servlet.
- Pour notre exemple, nous avons modifié l'URL déclenchée automatiquement pour ajouter le paramètre attendu par notre servlet :

Servlet et formulaire

- Un formulaire peut envoyer des informations à une servlet:
- Mettre le nom de la servlet qui va recevoir les informations en tant que valeur de l'attribut *Action* de la balise *Form*,
- Spécifier la méthode HTTP désirée grâce à l'attribut *Method*.
- Les données sont envoyées à l'aide de la méthode spécifiée (GET par défaut) après que l'utilisateur clique sur un bouton de type *Submit*
- Exemple : formulaire en HTML qui envoie des infos à la servlet *Parametres* :

```
<FORM Method="POST"
Action="http://localhost:8080/monprojet1/Parametres">
  Nom :          <INPUT type=text size=20 name=nom><BR>
  Prénom :       <INPUT type=text size=20 name=prenom><BR>
  Age :          <INPUT type=text size=2 name=age><BR>
                 <INPUT type=submit value=Envoyer>
</FORM>
```

Envoie par la méthode get

- Chaque élément du formulaire doit posséder un nom unique.
- La valeur et le nom de l'élément forme une paire nom/valeur du type : Nom_de_l_element = valeur_element
- L'ensemble des paires nom/valeur sont séparées par des *ET* ("&").
- champ1=valeur1&champ2=valeur2&champ3=valeur3
- L'envoi de cette chaîne se fera différemment selon que la méthode utilisée pour l'envoi du formulaire est GET ou POST.
- **La méthode GET :** envoie les éléments du formulaire au travers de l'URL, en ajoutant l'ensemble des paires nom/valeur à l'URL, séparé d'un point d'interrogation, ce qui donne une URL du type :
  http://serveur/cgi-bin/script.cgi?champ1=valeur1&champ2=valeur2...
- La longueur de l'URL étant limitée à 255 caractères, les informations situées au-delà de cette limite seront perdues.
- De plus, cela crée une URL surchargée dans la barre d'adresse d'un navigateur et peut dévoiler des informations sensibles comme un mot de passe...

Envoie par la méthode post

- **La méthode POST :** est une bonne alternative à la méthode GET.
- Cette méthode code les informations de la même façon que la méthode GET (encodage URL et paires nom/valeur)
- mais elle envoie les données à la suite des en-têtes HTTP, dans un champ appelé *corps de la requête*.
- De cette façon la quantité de données envoyées n'est plus limitée, et est connue du serveur grâce à l'en-tête permettant de connaître la taille du *corps de la requête*.

Lecture des paramètres avec une servlet

- L'un des points forts des servlets est la possibilité de traiter ("*parser*") les données en provenance de formulaires.
- L'objet *HttpServletRequest* possède de nombreuses méthodes (la plus courante *getParameter()*) permettant de retourner la valeur d'un champ du formulaire, qu'il s'agisse de données passées par POST ou GET.
- **La méthode getParameter()** : permet de retourner le valeur d'un champ dont on a passé le nom en argument :
- public String getParameter(String Key)
- Les noms des champs sont sensibles à la casse. Si le champ est vide, une chaîne vide est retournée. Si le champ n'existe pas, la valeur *null* est retournée.

Formulaire et servlet

- Un formulaire comportant une entrée comme suit :
- <Input type="text" name="NomDuChamp"> sera traité dans la servlet de cette façon : String Champ = req.getParameter("NomDuChamp")
- **La méthode getParameterValues() :** un champ ayant plusieurs valeurs (liste à choix multiples, cases à cocher, ...).
- public String[] getParameterValues(String Key) : retourne l'ensemble des valeurs affectées à la clé spécifiée en paramètre.
- **La méthode getParameterNames() :** donne l'ensemble des noms des champs du formulaire passé à la servlet.
- Enumeration getParameterNames() : L'objet *Enumeration*, contient la liste des champs du formulaire.
- Chaque entrée peut être traduite en chaîne, puis la traiter avec *getParameter()* afin de récupérer sa valeur.

Formulaire et servlet : exemple

```
package com.p;import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
import java.util.*;
public class Parametres extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, IOException { response.setContentType("text/html");
  PrintWriter out = response.getWriter();
   out.println("<html><body>\n" +"<h1>Tableau des paramètres</h1>\n" +
    "<table border=\"1\" cellspacing=\"0\" cellpadding=\"0\">\n" +"<tr>\n" +
                                  "<th>Nom</th><th>Valeur(s)</th>");
 Enumeration NomsParam = request.getParameterNames();
  while(NomsParam.hasMoreElements()) {
      String NomParam = (String)NomsParam.nextElement();
      out.println("<tr>"); out.print("<td>" + NomParam + "</td>");
       String[] ValeursParam = request.getParameterValues(NomParam);
       if (ValeursParam.length == 1) {
            String ValeurParam = ValeursParam[0];
```


Formulaire et servlet : exemple

```
    if (ValeurParam.length() == 0)
          out.println("<td><b>Aucune valeur</i></td>");
    else{  out.print("<td>"+ValeurParam+ "</td>"); out.println("</tr>");}
}
else {    out.println("<td><ul>");
          for(int i=0; i < ValeursParam.length; i++) {
               out.println("<tr>" + ValeursParam[i] + "</tr>"); }
              out.println("</ul></td>");}
      }
out.println("</table>\n</body></html>");
 }
  public void doPost(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {  doGet(request, response);  }
}
```

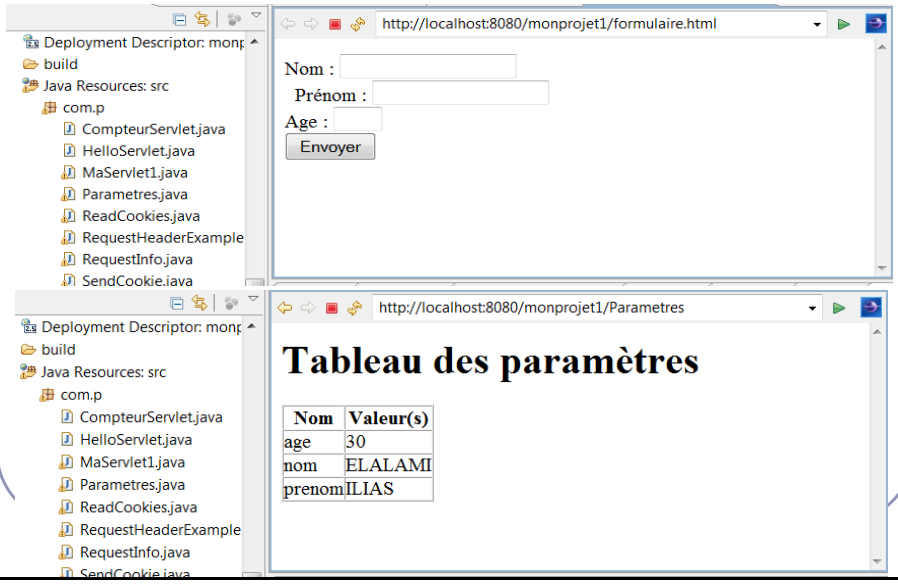
Servlet et formulaire

Dans monprojet1 créer new file formulaire.html et lancer
http://localhost:8080/monprojet1/formulaire.html

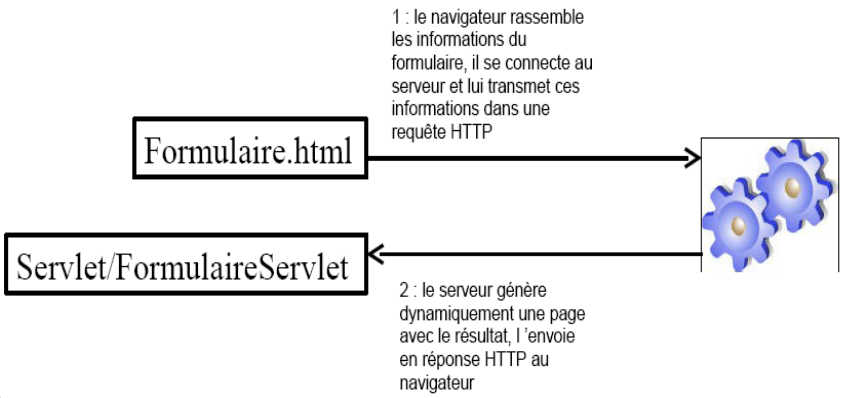
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Formulaire - saisie</title>
</head>
<body>
<FORM Method="POST" Action="Parametres">Nom : <INPUT type=text
size=20 name=nom><BR>
 Prénom : <INPUT type=text size=20 name=prenom><BR>
Age : <INPUT type=text size=2 name=age><BR>
<INPUT type=submit value=Envoyer></FORM>
</body>
</html>
```

54

Servlet et formulaire : resultat



Servlet : récapitulatif formulaire



Les cookies

- **Cookie** : Informations envoyées par le serveur, stockée sur le client et renvoyées par le client quand il revient visiter la même URL.
  - Durée de vie réglable
  - Permet d'avoir des données persistantes côté client
- **Utilisation** :
  - Identification des utilisateurs
  - Éviter la saisie d'informations à répétition (login, password, adresse,…)
  - Gérer des préférences utilisateur, profils
- **Sécurité** :
  - Jamais interprété ou exécuté : pas de virus
  - Pas partageable : le navigateur ne distribue la cookie qu'au serveur qui l'a émis
  - Une cookie est limitée à 4KB et les navigateurs se limitent à 300 cookies (20 par site) : pas de surcharge de disque
  - Bien pour rendre privées des données non sensibles mais ne constitue pas un traitement sérieux de la sécurité

Les cookies

- L'en-tête HTTP réservé à l'utilisation des cookies s'appelle Set-Cookie, il s'agit d'une simple ligne de texte de la forme:

  Set-Cookie : NOM=VALEUR; domain=NOM_DE_DOMAINE; expires=DATE

- Cette chaîne de caractères commence par *Set-Cookie* : suivie par des paires clés-valeur sous la forme CLE=VALEUR et séparées par des virgules.

- L'API servlet de Java propose un objet permettant de gérer de façon quasi-transparente l'usage des cookies, il s'agit de l'objet *Cookie*.

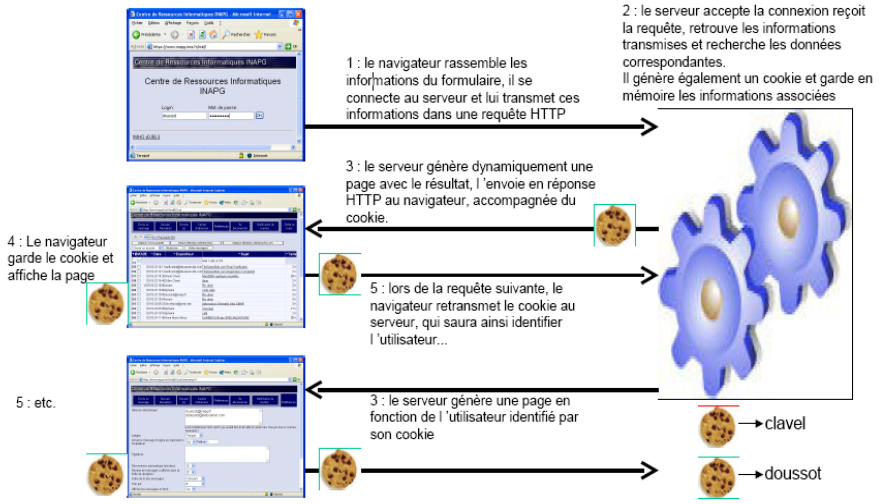
Classe Cookie

- La classe *javax.servlet.http.Cookie* permet de créer un objet *Cookie* encapsulant toutes les opérations nécessaires à la manipulation des cookies.
- **L'Envoi du cookie** vers le navigateur du client se fait grâce à la méthode *addCookie()* de l'objet *HttpServletResponse*.

  Exemple : Cookie MonCookie = new Cookie("nom", "valeur");
  response.addCookie(MonCookie);
- Pour récupérer les cookies provenant de la requête du client, on utilise la méthode *getCookies()* de l'objet *HttpServletRequest*
- Cookie[] getCookies() : retourne un tableau contenant l'ensemble des cookies présentes chez le client. *getName()* de *Cookie retrouve une cookie spécifique*.
- La récupération de la valeur d'une cookie se fait grâce à la méthode *getValue()* de *Cookie*

  Exemple : String Valeur = Cookie.getValue();

Web dynamique : cookies



Exemple de cookie

```
import javax.servlet.*; import javax.servlet.http.*;
public class ReadCookies extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, java.io.IOException {
  Cookie cookie = null;
   Cookie[] cookies = request.getCookies();
   boolean newCookie = false;
   if (cookies != null) {
     for (int i = 0; i < cookies.length; i++) {
                if (cookies[i].getName().equals("Cookies")) {
              cookie = cookies[i];}
    } }
   if (cookie == null) {
      newCookie = true;
       int maxAge;
      try {maxAge = new Integer(getServletContext().getInitParameter("cookie-age")).intValue();}
      catch (Exception e) {maxAge = -1; }
      cookie = new Cookie("Cookies", "" + getNextCookieValue());
      cookie.setPath(request.getContextPath());
       cookie.setMaxAge(100);
    response.addCookie(cookie);
  }  2013/2014                    Java EE : servlets                    61
```

Exemple de cookie(suite)

```
 response.setContentType("text/html");
  java.io.PrintWriter out = response.getWriter();
  out.println("<html>"); out.println("<head>");
  out.println("<title>Read Cookie</title>");out.println("</head>");
  out.println("<body>");
  out.println("<h2> Information sur notre Cookie appelé \"Cookies\"</h2>");
 out.println("Cookie value: " + cookie.getValue() + "<br>");
  if (newCookie) { out.println("Cookie Max-Age: " + cookie.getMaxAge() + "<br>");
                   out.println("Cookie Path: " + cookie.getPath() + "<br>");
  }
 out.println("</body>");out.println("</html>"); out.close();
 }
 private long getNextCookieValue() { return new java.util.Date().getTime();}
 public void doPost(HttpServletRequest request, HttpServletResponse response)
 throws ServletException, java.io.IOException {doGet(request, response);}
}
```


Cookie : résultat

HTTP: un protocole non connecté

- Le protocole HTTP est sans états (stateless protocol)
- Chaque requête est traitée indépendamment des autres et qu'aucun historique des différentes requêtes n'est conservé.
- Inconvénient dans le cas du e-commerce, pour lequel le serveur doit assurer mémoriser le suivi de session .
- **Les méthodes traditionnelles de suivi de session :**
- stocker les informations concernant l'utilisateur sur le serveur, et de les relier à chaque requête grâce à un identifiant présent dans la requête et sur le serveur.
- échanger des données d'identification du client entre chaque requête
  - en ajoutant à l'URL des variables d'identification
  - en passant en paramètre un champ de formulaire caché
  - en utilisant les cookies
- en demandant l'identification de l'utilisateur à chaque requête

HttpSession : introduction

- HttpSession est un objet persistant coté serveur qui permet de stocker les informations saisies au fur et à mesure par l'utilisateur et de les relier à chaque requête grâce à l'identifiant passé en paramètre.
- Un identificateur unique de session, permet de relier l'utilisateur aux données stockées sur le serveur et peut être transmis en :
  - L'URL : `<a href="http://serveur/servlet/exemple?id=674684641">…`
  - Champs de formulaire cachés : `<Input type="hidden" name="id" value="674684641">`
  - Utilisation de cookies :

```
Cookie C = new Cookie("id","674684641"); // Creation du cookie
C.setMaxAge(24*3600); // définition de la limite de validité
res.addCookie(C); // envoi du cookie dans la réponse HTTP
```

- Ce mécanisme de cookies pose quelques problèmes :
  - des utilisateurs désactivent les cookies par crainte
  - certains anciens navigateurs ne gèrent pas les cookies

HttpSession

- Avec HttpSession, une session est établie pour une durée préfixée, indépendamment du nombre de requêtes établies.
- Dans la classe HttpServletRequest, on a deux méthodes :
  - HttpSession getSession()
  - HttpSession getSession(boolean create) : retourne la session HttpSession courante associée à cette requête, ou bien une nouvelle session, dans le cas ou il n'y a pas de session courante et create est true.

HttpSession

- getRequestedSessionId() : retourne l'ID de la session.
- Dans la classe HttpServletResponse:
  - String encodeURL(String url) : code l'URL spécifiée en argument en lui incluant le ID de la session.
  - si l'encodage n'est pas nécessaire, elle se contente de retourner l'URL sans y introduire de modification.
- Dans la classe HttpSession:
  - setAttribute(String name, Object value) : associé un objet à un nom spécifié en argument, pour cette session.
  - getAttribute(String name) : retourne l'objet associé au nom spécifié en argument, ou bien null si aucun nom n'a été associé à l'objet.
  - removeAttribute(String name) : retire l'association de l'objet avec le nom spécifié en argument.

HttpSession

- String getId() : retourne le ID de la session.
- Enumeration getAttributeNames() : retourne un énumérateur d'objets string contenant les noms de tous les objets associés à cette session.
- long getCreationTime() : retourne la date de création de la session.
- long getLastAccessedTime() : retourne la date de la dernière requête envoyée par le client, associée à cette session.

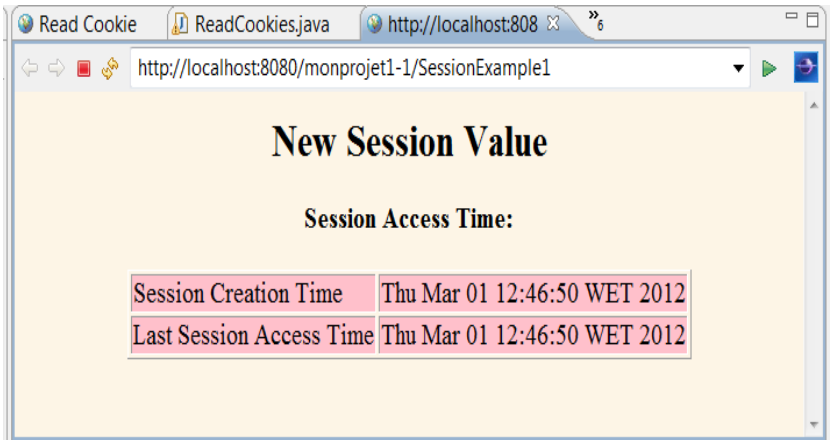
Exemple de Session

```
package com.p;import java.io.*;import javax.servlet.*;
import javax.servlet.http.*;import java.net.*;import java.util.*;
public class SessionExample1 extends HttpServlet {
  public void doGet(HttpServletRequest request,HttpServletResponse response)
    throws ServletException, IOException {
    HttpSession session = request.getSession(true);
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String head;
    Integer count = new Integer(0);
    if (session.isNew()) {  head = "New Session Value ";}
    else {
      head = "Old Session value";
      Integer oldcount =(Integer)session.getValue("count");
      if (oldcount != null) {
        count = new Integer(oldcount.intValue() + 1);
      }
    }
```

Code Source pour Session

```
session.putValue("count", count);
    out.println("<HTML><BODY BGCOLOR=\"#FDF5E6\">\n" +
      "<H2 ALIGN=\"CENTER\">" + head + "</H2>\n" +
      "<H4 ALIGN=\"CENTER\">Session Access Time:</H4>\n" +
      "<TABLE BORDER=1 ALIGN=CENTER>\n" + "<TR BGCOLOR=\"pink\">\n" +
      "  <TD>Session Creation Time\n" +"  <TD>" +
      new Date(session.getCreationTime()) + "\n" +
      "<TR BGCOLOR=\"pink\">\n" +"  <TD>Last Session Access Time\n" +
      "  <TD>" + new Date(session.getLastAccessedTime()) +
      "</TABLE>\n" +"</BODY></HTML>");
  }
  public void doPost(HttpServletRequest request,HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
  }
}
```


Résultat : session



Résultat : session

Interface ServletConfig

- javax.servlet.ServletConfig représente les informations de configuration d'une Servlet au sein d'une application Web.
- Le conteneur Web va créer un objet de type ServletConfig pour chaque élément <servlet> déclaré dans le fichier web.xml.
- Dans le fichier web.xml, on retrouve le nom de la Servlet ainsi qu'une éventuelle liste de paramètres associés.
- Ces informations de configuration peuvent ensuite être récupérées par la Servlet de préférence dans la méthode init(...).
- Le fichier de configuration peut avoir zéro ou plusieurs éléments <init-param> par Servlet. Chaque élément <initparam> correspond à un paramètre représenté par une paire nom/valeur avec <paramname> et <paramvalue>.

Interface ServletConfig

- La méthode init(...) est souvent chargée de récupérer des ressources utiles à la Servlet...
- L'interface ServletConfig permet de manipuler les paramètres du fichier de configuration de l'application web.xml avec :
- getInitParameter(...) : récupère le string associé à un paramètre dans le fichier web.xml (ou null si le paramètre n'existe pas).
- getInitParameterNames(…) : récupère sous forme d'une énumération l'ensemble des paramètres déclarés dans web.xml.
- getServletName(…) : récupère le nom de la Servlet déclarée au sein du descripteur de déploiement.

Initialisation d'une Servlet

```
<servlet>
    <servlet-name>servletauthentification</servlet-name>
    <servlet-class>mypkg.ServletAuthentification</servlet-class>
    <init-param>
      <param-name> defautIdentifiant</param-name>
      <param-value> monidentifiant</param-value>
    </init-param>
    <init-param>
      <param-name> defautMotDePasse </param-name>
      <param-value> monmotdepasse </param-value>
    </init-param>
  </servlet>
…
```

Exemple ServletConfig

```
package mypkg;
import java.io.IOException; import java.io.PrintWriter;
import javax.servlet.ServletConfig; import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet; import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ServletAuthentification extends HttpServlet {
    String ident=null;                          //variables de classe
     String mdp=null;
  public void init() {
 //récupération des paramètres d'initialisation de la Servlet dans web.xml
     ServletConfig config = getServletConfig();
     ident = (String) config.getInitParameter("defautIdentifiant");
     mdp = (String) config.getInitParameter("defautMotDePasse");
 }
```


Exemple ServletConfig

```
 public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException
   { //récupération de l'identifiant/login dans la requête
      String identifiant = request.getParameter("identifiant");
      //récupération du mot de passe dans la requête
      String motdepasse = request.getParameter("motdepasse");
      //flux de sortie
      PrintWriter out=response.getWriter();
      //pas d'identifiant
      if(identifiant==null)  {    out.println("Authentification incorrecte !");   }
      //pas de mot de passe
      if(motdepasse==null) {    out.println("Authentification incorrecte !");   }
```

Exemple ServletConfig

```
//vérifier l'égalité des valeurs
      if( (identifiant!=null && identifiant.equals(ident))
     && (motdepasse!=null && motdepasse.equals(mdp)) )
      {   out.println("Authentification correcte,
            bienvenue : " + identifiant);
      }
      else {   out.println("Authentification incorrecte, mauvaise
                  saisie des informations !");   }
   }
   public void doPost(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException
   {  doGet(request, response); }
}
```


Interface ServletContext

- ServletConfig gère des paramètres qui sont propres à une Servlet.
- javax.servlet.ServletContext représente les informations de configuration de toute l'application Web.
- Chaque Servlet de la même application Web aura donc accès à ces paramètres.
- javax.servlet.ServletContext peut également interagir avec le conteneur Web de l'application.
- Il y aura un accès en lecture et en écriture dans le fichier de configuration pour créer, lire et supprimer des attributs de façon dynamique, ce qui permet alors le partage de ressources entre Servlets d'une même application Web.

Exemple web.xml pour ServletContext

```
<web-app>
   <context-param>
      <param-name>  emailAdministrateur</param-name>
      <param-value>  admin@betaboutique.fr</param-value>
   </context-param>
    <servlet>
      <servlet-name> servletauthentification</servlet-name>
      <servlet-class> mypkg.ServletAuthentification</servlet-class>
      <init-param>
         <param-name> defautIdentifiant</param-name>
         <param-value> monidentifiant</param-value>
      </init-param>
        ….
    </servlet>
```


Exemple ServletContext

```
package mypkg; …
public class ServletAuthentification extends HttpServlet {
    …
   String emailadministrateur=null;
   public void init()
    {  //récupération des paramètres d'initialisation de la Servlet dans web.xml
       config=getServletConfig();
      ident=(String)config.getInitParameter("defautIdentifiant");
      mdp=(String)config.getInitParameter("defautMotDePasse");
      //récupérer l'email de l'administrateur
      ServletContext servletContext = getServletContext();
      emailadministrateur = servletContext.getInitParameter("emailAdministrateur");
   }
   public void doGet(){ …} }
```

ServletContext : Ajout attributs

- L'avantage de l'interface ServletContext est la possibilité d'ajouter des attributs à la volée.
- Le contexte est accessible par l'ensemble des Servlets/JSP et JSF de l'application Web et peut donc stocker, récupérer et détruire des attributs.
- Contrairement à l'interface ServletConfig, des objets divers peuvent être gérés et pas seulement des strings.
- Ce qui permet de déclarer au lancement de l'application un objet pour la connexion à la base de données et de le stocker à la volée dans le fichier de configuration comme variable globale.
- La connexion sera alors disponible pour l'ensemble des classes Java EE (Servlets et JSP).
- Ces attributs sont des paires clé/valeur, la clé étant un string et la valeur étant un objet de n'importe quel type.

ServletContext : Ajout attributs

- Les attributs d'une application Web sont donc des variables globales appelées également variables d'application des éléments participants à son fonctionnement et qui peuvent être partagées simultanément par plusieurs Servlets et pages JSP.
- La méthode setAttribute(nom, objet) : est utilisée pour positionner un attribut qui sera visible dans toute l'application (portée application). Si le nom de l'attribut existe déjà, la valeur existante est remplacée par la nouvelle.
- La méthode getAttribute(nom) : est utilisée pour récupérer la valeur d'un attribut de type quelconque dans l'application ou la valeur null si l'attribut n'existe pas.
- La méthode getAttributesNames() : permet de récupérer le nom de tous les attributs actuellement stockés dans le contexte de l'application Web.
- Enfin, la méthode removeAttribute(nom) : permet la suppression de l'attribut indiqué dans le contexte de l'application.

ServletContext : ajout attributs

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
  {  ...
     if( (identifiant!=null && identifiant.equals(ident))
    && (motdepasse!=null && motdepasse.equals(mdp)) )
     {  out.println("Authentification correcte, bienvenue : "+identifiant);
        //créer l'objet JavaBean Client
        Client client1 = new Client();
        client1.setIdentifiant(identifiant);
        client1.setMotdepasse(motdepasse);
        //sauvegarder l'objet client dans le contexte de l'application
        ServletContext servletContext = getServletContext();
        servletContext.setAttribute("client1", client1);
     } }
```


ServletContext : lecture

```
public class ServletLectureAuthentification extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse
response)throws ServletException, IOException
   { //flux de sortie
      PrintWriter out=response.getWriter();
      //lecture de l'objet dans le contexte de l'application
      ServletContext servletContext=getServletContext();
      Client client1 = (Client)servletContext.getAttribute( "client1" );
      if(client1!=null)
      {out.println("Le client dans le contexte est : ");
         out.println("identifiant : "+ client1.getIdentifiant() );
         out.println("mot de passe : "+ client1.getMotdepasse() );
      }
}
```

Les filtres

Les filtres permettent ainsi de traiter :

- Les requêtes qui viennent des clients avant qu'elles ne soient traitées par les Servlets.
- Les réponses venant des Servlets avant qu'elles ne soient envoyées aux clients.

Il est possible par exemple de :

- décrypter des requêtes envoyées aux Servlets, traiter les données avec les Servlets et crypter pour les clients ;
- gérer l'authentification des clients ;
- convertir des formats d'images, appliquer des transformations XSLT sur des données XML...

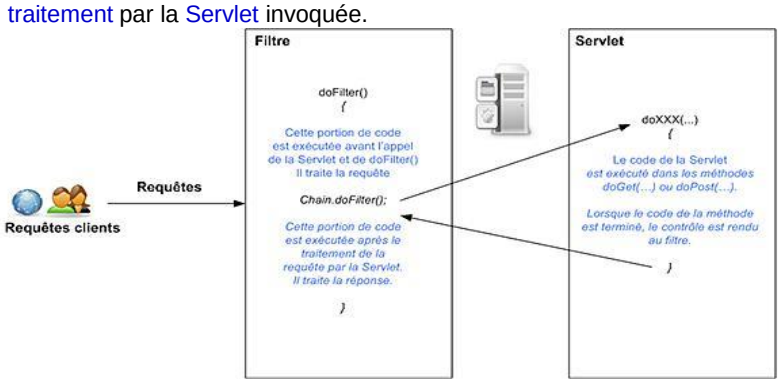
Les filtres

- Pour utiliser un filtre, il faut réaliser deux opérations :
- 1- écrire une classe qui implémente l'interface Filter.
- 2- modifier le descripteur de déploiement (web.xml) pour indiquer au conteneur d'utiliser le filtre.

- Lorsqu'un filtre est créé, le conteneur appelle sa méthode init(...). Dans cette méthode, nous pouvons accéder aux paramètres d'initialisation avec l'interface FilterConfig. Lors du traitement de la requête, le conteneur appelle la méthode doFilter(...).
- Avant de détruire le filtre, le conteneur appelle sa méthode destroy(...).
- Lorsque le filtre appelle chain.doFilter(), le filtre suivant dans la chaîne est exécuté.
- Le code placé avant chain.doFilter() est exécuté avant le traitement de la Servlet. Le code placé après cet appel est exécuté après le traitement de la Servlet. .

Les filtres

- Il est tout à fait possible de chaîner les filtres et d'utiliser un filtre par traitement spécifique.
- Le schéma suivant présente le fonctionnement d'un filtre avant et après traitement par la Servlet invoquée.



Déclaration du Filtre en web.xml

```
...
<filter>
   <filter-name>filtrejournalisation</filter-name>
   <filter-class>mypkg.FiltreJournalisation </filter-class>
</filter>
<filter-mapping>
   <filter-name> filtrejournalisation</filter-name>
   <servlet-name> servletauthentification</servlet-name> // filtre appliqué à une
                                                         // servlet
   <url-pattern>/*</url-pattern> // ou  filtre est appliqué pour l'accès à toutes les
                                 //      URL de l'application.
</filter-mapping>
...
```

Exemple du Filtre

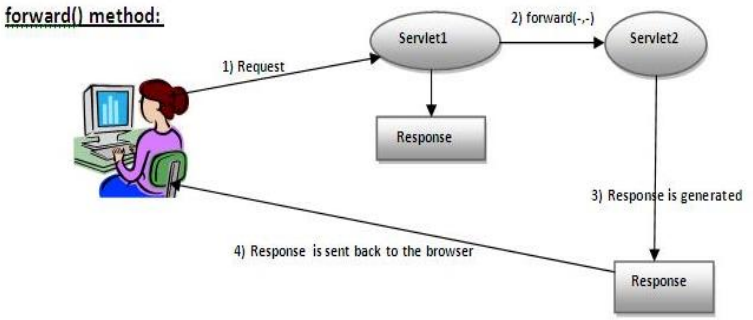
```
public class FiltreJournalisation implements Filter{
   private FilterConfig  filterConfig=null;
   // action d'initialisation du filtre
   public void init (FilterConfig filterConfig)
   {  this.filterConfig=filterConfig;
      System.out.println("Initialisation du filtre :"+this.filterConfig.getFilterName());
   }
   //action déclenchée lors du traitement d'une requête
 public void doFilter(ServletRequest request, ServletResponse response,
    FilterChain chain)
   { //traitement de la requête avant la Servlet
     …
```


Exemple du Filtre

```
  try  { //déclencher la servlet appropriée
          chain.doFilter(request, response);
       } catch(Exception e){ }
//Traitement de la réponse après le traitement de la requête par la Servlet
       ….
       //exemple d'écriture dans le fichier de log de l'application
       ServletContext context=this.filterConfig.getServletContext();
       context.log("Encodage de la réponse : "+response.getCharacterEncoding());
   }
   //action qui permet de détruire le filtre
   public void destroy()
   {   System.out.println("Destruction du filtre : "+this.filterConfig.getFilterName());}
```

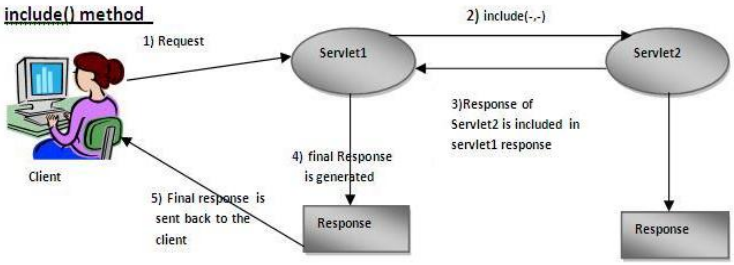
Interface RequestDispatcher

- Cette interface contient deux méthodes :
- **public void forward(ServletRequest request,ServletResponse response) :** transmet une requête d'une servlet vers une autre resource (servlet, page JSP, page HTML) sur le serveur.



Interface RequestDispatcher

- **public void include(ServletRequest request,ServletResponse response):** inclut le contenu d'une ressource (servlet, page JSP, ou page HTML) dans la réponse.

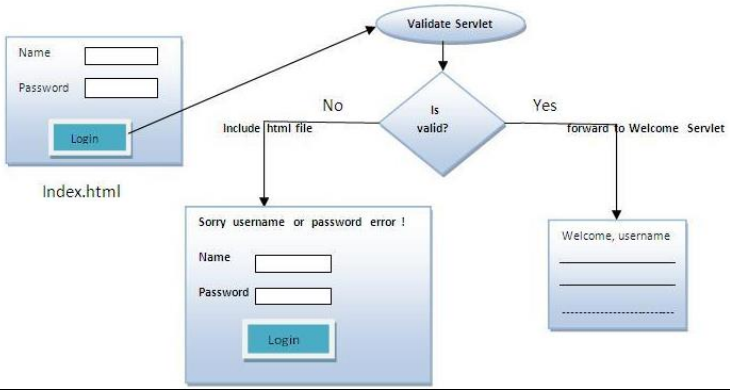


Exemple :

```
RequestDispatcher rd = request.getRequestDispatcher ("servlet2");
//servlet2 est l'url de la deuxième Servlet
rd.forward (request, response);//la méthode peut être include ou forward
```

Exemple RequestDispatcher

- index.html: pour saisir les informations de l'utilisateur.
- Login.java: une servlet pour traiter la réponse. Si le mot de passe est servet, elle transmet la requête à la servlet WelcomeSrvlet.
- WelcomeServlet.java: une servlet pour afficher le message de bienvenue.



94

Exemple RequestDispatcher

- **index.html**

```
<form action="servlet1" method="post">
    Name:      <input type="text" name="userName"/><br/>
    Password: <input type="password" name="userPass"/><br/>
              <input type="submit" value="login"/>
</form>
```

Exemple RequestDispatcher

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
public class Login extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
  PrintWriter out = response.getWriter();
  String n=request.getParameter("userName");
  String p=request.getParameter("userPass");
  if(p.equals("servlet"){
          RequestDispatcher   rd = request.getRequestDispatcher("servlet2");
           rd.forward(request, response);      }
  else{ out.print("Sorry UserName or Password Error!");
            RequestDispatcher rd=request.getRequestDispatcher("/index.html");
             rd.include(request, response);
                  }          } }
```


Exemple RequestDispatcher

- **WelcomeServlet.java**
-

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class WelcomeServlet extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse
  response)throws ServletException, IOException {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          String n=request.getParameter("userName");
          out.print(" We are in servlet Welcome "+n);
}
}
```

Plate forme Java EE : couche présentation

JSP, JSTL et EL

Plan

- Concepts de base pour la création des pages web
- Rappel de HTML, XHTML, CSS et JavaScript
- Les JavaServer Pages
- Le langage d'expression (EL)
- La bibliothèque JSTL

Les pages web

- Une couche présentation sert à interagir avec les utilisateurs. Elle doit être une interface web qui s'exécute sur un navigateur, plus riche, plus dynamique et plus simple à utiliser.
- Une application web, affiche généralement un contenu dynamique :
  - une liste de livres d'un catalogue
  - Un panier virtuel contenant les articles qu'on veut acheter.
- Le contenu statique change rarement : adresse d'un éditeur, dessin vidéo…
- Le but de la création d'une page est son affichage dans un navigateur.
- Elle doit alors utiliser les langages compris par les navigateurs (HTML, XHTML, CSS et JavaScript)

Rappel : HTML

- Hypertext Markup Language (HTML) est le langage qui domine dans les pages web.
- HTML utilise des balises, ou marqueurs, pour structurer le texte en paragraphes, listes, liens, boutons, zones de texte, etc.
- Une page HTML est un document texte utilisé par les navigateurs pour présenter du texte et des images : fichiers texte avec l'extension **.html** ou **.htm**.
- Une page web est formée d'un contenu, de balises permettant de changer certains aspects de ce contenu et d'objets externes comme des images, des vidéos, du code JavaScript ou des fichiers CSS.
- Une page HTML valide commence par une balise **<html>** qui agit comme conteneur du document. Elle est suivi des balises **<head>** et **<body>.**
- La balise **<body>** contient la partie visible. Dans l'exemple suivant, le corps est formé d'un tableau constitué de labels et de champs de saisie, et un bouton.

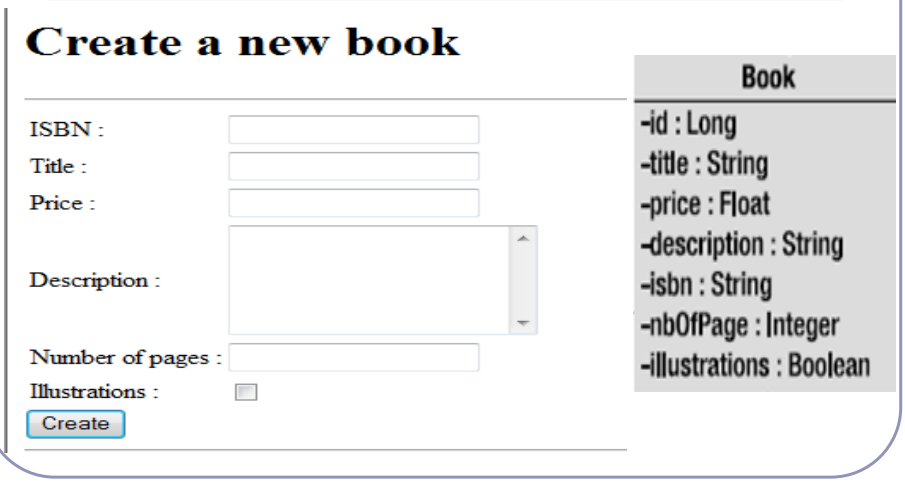
Rappel : exemple 1 page HTML

```
<h1>Create a new book</h1>
<hr>
<TABLE border=0>
   <TR>
      <TD>ISBN :</TD>
      <TD><input type=text/></td>
   </tr>
   <tr>
      <td>Title :</td>
      <TD><input type=text/></td>
   </tr>
   <tr>
      <td>Price :
      <TD><input type=text/>
   </tr>
   <tr>
      <td>Description :
      <td><textarea name=textarea
cols=20 rows=5></textarea>
   </tr>
   <TR>
      <TD>Number of pages :
      <td><input type=text/>
   </tr>
   <tr>
      <td>Illustrations :
      <td><input type="checkbox"/>
   </tr>
</table>
<input type=submit value=Create>
<hr>
<i> Java EE 7 </i>
```


Rappel : exemple page HTML

- Le fichier HTML n'est pas correctement formaté en termes de XML :
- La page n'a pas de marqueurs **<html>, <head>** ou **<body>.**
- Le marqueur **<input type=submit value=Create>** n'est pas fermé.
- Les marqueurs mélangent les majuscules et les minuscules (<TR> et </tr>) .
- La plupart des navigateurs autorisent ce type d'erreur et afficheront correctement le formulaire.
- Mais si on veut traiter ce document avec des parsers XML, le traitement échouera.
- Pour comprendre la raison, étudions la page web qui utilise une structure XML stricte avec XHTML (eXtensible Hypertext Markup Language)

Rappel : exemple de page HTML



Rappel : XHTML

- XHTML a été crée après HTML 4.01. Ses racines puisent dans HTML, mais avec une reformulation en XML strict.
- Un document XHTML est donc un document XML qui respecte un certain schéma et peut être représenté graphiquement par les navigateurs.
- Un fichier XHTML (d'extension .xhtml) peut être directement utilisé comme du XML ou être affiché dans un navigateur.
- Un document XHTML, par rapport à HTML, a l'avantage de permettre une validation et une manipulation du document à l'aide d'outils XML standard (XSL ou eXtensible Stylesheet Language; XSLT; etc.)

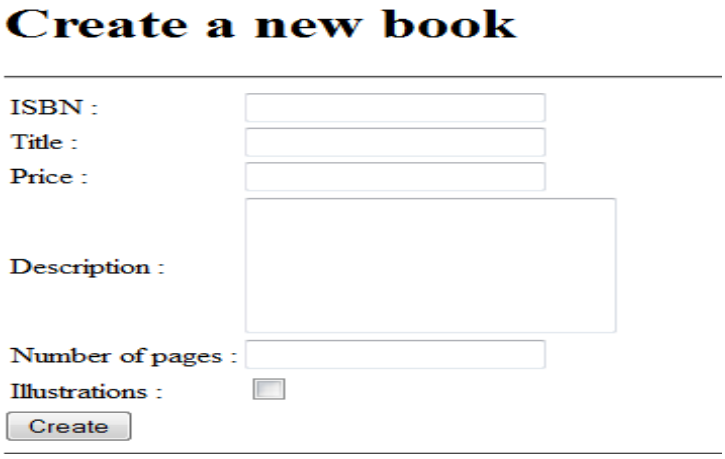
Rappel : exemple 2 de page en XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head> <title>Create a new book</title> </head>
<body> <h1>Create a new book</h1>
<hr/> <table border="0">
   <tr> <td>ISBN :</td>  <td><input type="text"/></td>  </tr>
   <tr>  <td>Title :</td>  <td><input type="text"/></td>   </tr>
   <tr> <td>Price :</td>    <td><input type="text"/></td>  </tr>
   <tr> <td>Description :</td>   <td><textarea name="textarea" cols="20"
     rows="5"></textarea></td></tr>
   <tr>  <td>Number of pages :</td>  <td><input type="text"/></td> </tr>
   <tr>  <td>Illustrations :</td> <td><input type="checkbox"/></td> </tr>    </table>
<input name="  " type="submit" value="Create"/>
<hr/> <i> Java EE 7 </i>          </body> </html>
```


Rappel : XHTML

- Les codes des exemples des pages en XHTML et en HTML sont différents.
- Le code en XHTML respecte une structure stricte et contient des marqueurs **<html>, <head>** et **<body>.**
- Tous les marqueurs sont fermés, même les vides (chaque <td> est fermé et on utilise <hr/> au lieu de <hr>)
- Les valeurs des attributs sont toujours entre apostrophes ou entre guillemets (<table border=**"0"**> ou <table border=**'0'**> mais pas <table border=**0**>).
- Tous les marqueurs sont en minuscules (<tr> au lieu de <TR>).
- Le respect strict des règles syntaxiques de XML et les contraintes de schéma rendent XHTML plus facile à maintenir et à traiter que HTML, et c'est pourquoi il est le langage préféré pour les pages web.

Rappel : exemple 2 de page XHTML



Rappel : CSS

- CSS (Cascading Style Sheets) sert à décrire la présentation d'un document écrit en HTML ou en XHTML
- Il permet de définir les couleurs, les polices, la disposition et les autres aspects de la présentation d'un document et, donc, de séparer son contenu (écrit en XHTML) de sa présentation (écrite en CSS).
- Par exemple, si on veut modifier les labels de l'exemple 2 pour qu'ils soient tous en italique (*font-style: italic;),*de couleur bleue (*color: #000099;*) et dans une taille de police plus grande (*font-size: 22px*;).
- Au lieu de répéter ces modifications pour chaque marqueur, il suffit de définir un style CSS (dans un marqueur <style type="text/css">) et de lui donner un alias (row, par exemple) : la page appliquera alors ce style pour tous les éléments qui utilisent cet alias (<td **class="row"**>).

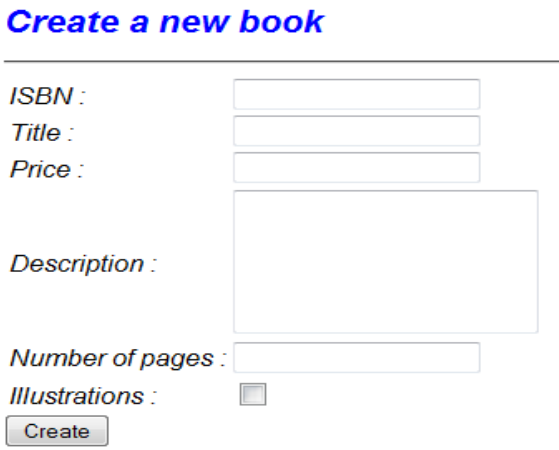
Rappel : exemple 3 de page CSS

```
<!DOCTYPE html  PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head> <title>Creates a new book</title>
   <style type="text/css">
      body { font-family: Arial, Helvetica, sans-serif;}
      table { border: 0; }
      h1 { font-size: 22px; color: blue; font-style: italic; }
      .row {  font-style: italic;  }
   </style>
</head>
<body> <h1>Create a new book</h1>
<hr/>
<table>
   <tr>  <td class="row">ISBN :</td> <td><input type="text"/></td></tr>
```


Rappel : exemple 3 de page CSS

```
<tr> <td class="row">Title :</td>  <td><input type="text"/></td> </tr>
 <tr> <td class="row">Price :</td> <td><input type="text"/></td> </tr>
 <tr> <td class="row">Description :</td>   <td><textarea name="textarea" cols="20"
     rows="5"></textarea></td> </tr>
  <tr>  <td class="row">Number of pages :</td>   <td><input type="text"/></td>  </tr>
   <tr>  <td class="row">Illustrations :</td>   <td><input type="checkbox"/></td> </tr>
</table>
<input name="  " type="submit" value="Create"/>
<hr/>
<i>Java EE 7</i>
</body>
</html>
```

Rappel : exemple 3 de page CSS

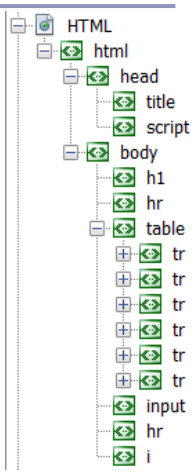


Rappel : CSS

- Dans l'exemple 3, le code CSS est intégré à la page XHTML mais, dans une vraie application, tous les styles seraient placés dans un fichier distinct qui serait importé par la page web.
- Le concepteur du web, peut ainsi créer un ou plusieurs fichiers CSS pour différents groupes de pages.
- L'interprétation du code de l'exemple 3 par un navigateur, affiche tous les labels en italique et le titre en beu.

Rappel : DOM

- Une page XHTML est un document XML et a donc une représentation DOM (Document Object Model).
- DOM est une représentation arborescente de la structure d'un document.
- La figure ci-joint montre une représentation DOM de la page XHTML : la racine est le marqueur html, ses deux fils sont head et body et ce dernier à lui-même un fils table.
- DOM fournit un moyen standard d'interaction avec les documents XML.
- On peut parcourir l'arbre d'un document et modifier le contenu d'un nœud.
- En utilisant JavaScript, il est possible d'ajouter un comportement dynamique à une page web.
- Note : installez DOM Inspector pour Firefox ou utilisez Netbeans 6.9 ou plus



Rappel : JavaScript

- Pour créer un contenu dynamique, on utilise les technologies côté serveur JSP ou JSF, mais on peut en produire côté client (navigateur) en exécutant du code JavaScript.
- JavaScript est un langage de script pour le développement web côté client.
- C'est un langage interprété et faiblement typé, il n'a rien à voir avec Java.
- Il est possible de créer des applications web dynamiques en écrivant des fonctions qui agissent sur le DOM d'une page.
- On ajoute du code JavaScript à l'exemple 3 précédent. Ainsi, le prix du livre doit être fourni par l'utilisateur côté client avant d'atteindre le serveur : une fonction JavaScript (*priceRequired()*) permet de valider ce champ en testant s'il est vide ou non.
- Cette fonction est intégrée dans la page au moyen d'un marqueur <script> et est appelée lorsque le champ de saisie perd le focus (représenté par l'événement **onblur**).

Rappel : exemple 4 en JavaScript

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html  PUBLIC  "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
   <title>Creates a new book</title>
   <script type="text/javascript">
      function priceRequired() {
         if (document.getElementById("price").value == "") {
            document.getElementById("priceError").innerHTML = "Please, fill the price !";
         }
      }
   </script>
</head>
```

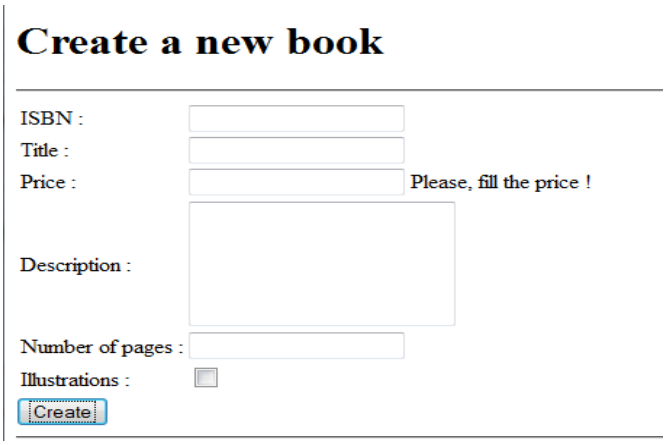

Rappel : exemple 4 en JavaScript

```
<body>
<h1>Create a new book</h1>
<hr/><table border="0">
   <tr> <td>ISBN :</td> <td><input type="text"/></td>  </tr>
   <tr>  <td>Title :</td>   <td><input type="text"/></td>   </tr>
   <tr>   <td>Price :</td>  <td><input id="price" type="text"
     onblur="javascript:priceRequired()"/>   <span id="priceError"/>    </td> </tr>
   <tr> <td>Description :</td>   <td><textarea name="textarea" cols="20"
     rows="5"></textarea></td> </tr>
   <tr>  <td>Number of pages :</td>   <td><input type="text"/></td>  </tr>
   <tr>   <td>Illustrations :</td>   <td><input type="checkbox"/></td> </tr>
</table>
<input name="" type="submit" value="Create"/>
<hr/>
<i> Java EE 7</i>
     </body>                    </html>
```

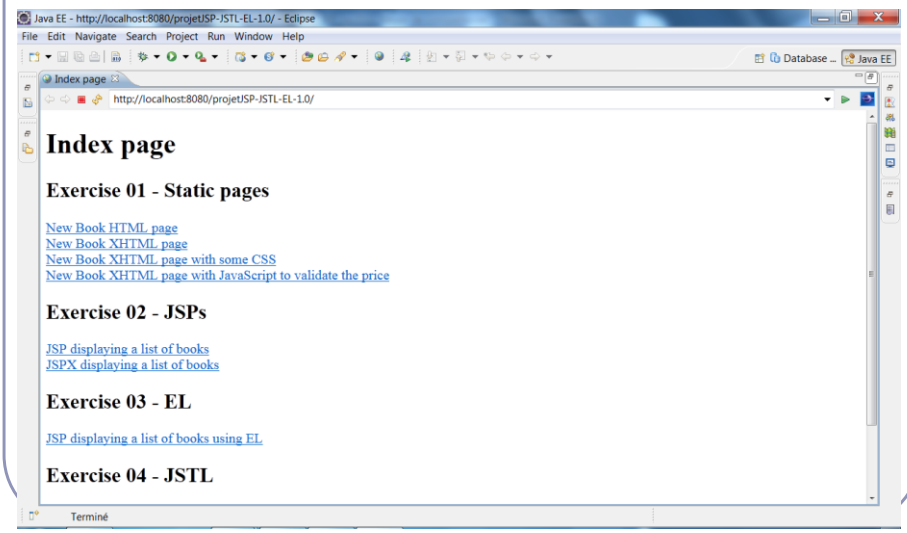
Rappel : exemple en JavaScript

- La fonction (*priceRequired()*) utilise l'objet *document* implicite qui représente le DOM du document XHTML.
- L'appel *getElementById("price")* recherche un élément ayant un identifiant *price(<input id="price">)* :on récupère sa valeur et l'on teste s'elle est vide. Si c'est le cas, la fonction recherche un autre élément appelé *priceError(getElementById("priceError"))* et fixe sa valeur à "Please, fill the price ! ".
- JavaScript est un langage puissant : nous n'en avons présenté qu'une partie pour montrer son interaction avec DOM.
- Une fonction JavaScript peut accéder à un nœud de la page et modifier dynamiquement son contenu côté client.
- La validation du formulaire vide de l'exemple 4 affichera le message suivant :

Rappel : exemple en JavaScript



Rappel : pages statique



Java server Pages

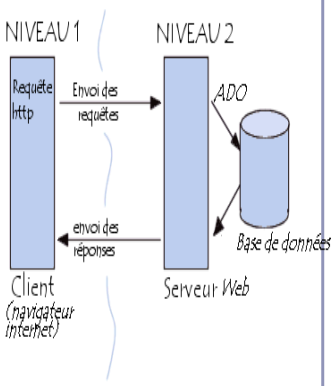
- JSP est né en 1999 avec J2EE 1.2 et permet de créer dynamiquement des pages web en réponse à une requête d'un client.
- Les pages JSP ressemblent à des pages HTML ou XHTML, sauf qu'elles contiennent des marqueurs spéciaux pour effectuer des traitements sur le serveur et appeler du code java côté serveur.
- Les pages sont traitées sur le serveur et compilées sous forme de servlets.
- Les servlets ont été crées pour permettre à un serveur d'accepter des requêtes HTTP des clients et de créer des réponses dynamiques.
- Les servlets et les JSP peuvent se servir de n'importe quelle ressource serveur comme les EJB, les BDs, les services web,…
- Les JSP sont dynamiques car elles exécutent du code Java pour former une réponse en fonction d'une requête.

Java server Pages

- Les JSP s'exécutent sur un serveur dans un conteneur de servlets et répondent aux requêtes des clients en utilisant le protocole HTTP.
- Le conteneur de servlets gère le cycle de vie d'une JSP en :
  - Compilant le code JSP dans une servlet;
  - Chargeant et initialisant la JSP;
  - Traitant les requêtes des clients et les faisant suivre à la JSP;
  - Renvoyant les réponses aux clients (ces réponses sont de type HTML ou XHTML);
  - Déchargeant la JSP et arrêtant de lui envoyer des requêtes.
- Une JSP peut soit produire du code HTML, elle doit donc utiliser l'extension **.jsp,** ou produire du code XHTML et elle doit porter l'extension **.jspx.**

Java Server Pages

- Servlet = du code Java contenant de l'HTML
- JSP = une page HTML contenant du code Java
- Plus concrètement :
  - toute la page JSP est convertie en une servlet
  - cette servlet est traitée par le moteur Java intégré au serveur Web et retourne la page HTML construite.
- JSP : les parties statiques sont écrites en HTML et les parties dynamiques sont écrites en Java



Java Server Pages : caractéristiques

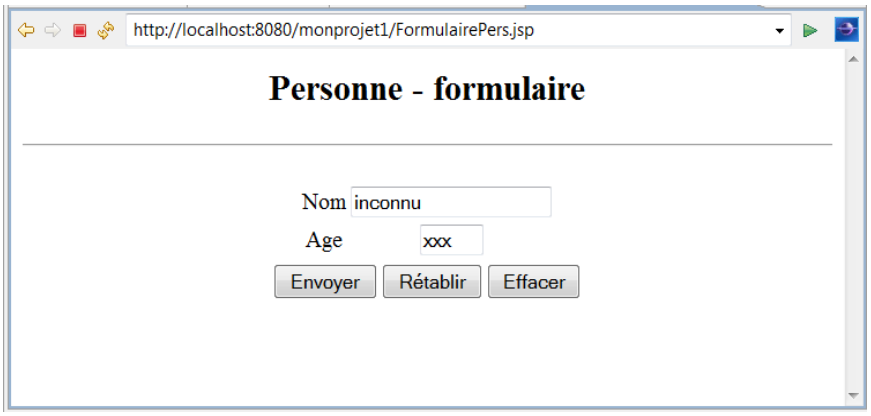
- les JSP sont multi-threadées,
- les JSP sont portables,
- les JSP sont orientées objet,
- les JSP sont sûres,
- Séparation entre données et logique applicative :
  - JSP intègre du code Java au sein du code HTML en utilisant des balises.
  - séparation entre les données (codées en HTML) et la logique applicative (traitements) fournie par Java.
- JSP doit ainsi être utilisée pour accéder à des composants réutilisables (servlets, JavaBeans, EJB (*Enterprise JavaBeans*)).

JSP : exemple

- Java Server Pages :
  - Fichier texte qui décrit comment créer une réponse à partir d'une requête particulière.
  - tags HTML +XML+ extensions + JAVA comme langage de script

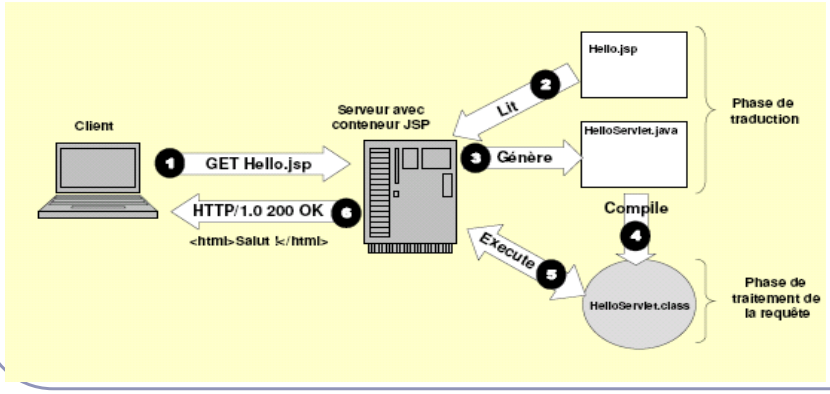
```
<% // on récupère les paramètres
String nom=request.getParameter("txtNom");
if(nom==null) nom="inconnu";
String age=request.getParameter("txtAge");
if(age==null) age="xxx";%>
<html> <head> <title>Personne - formulaire</title> </head>
<body> <center> <h2>Personne - formulaire</h2>
<hr> <form action="" method="post">
<table> <tr> <td>Nom</td> <td><input name="txtNom" value="<%= nom %>" type="text" size="20"></td>
      </tr>
<tr> <td>Age</td> <td><input name="txtAge" value="<%= age %>" type="text" size="3"></td></tr>
</table> <table> <tr> <td><input type="submit" value="Envoyer"></td>
                        <td><input type="reset" value="Rétablir"></td>
                       <td><input type="button" value="Effacer"></td>
</tr> </table> </form> </center></body> </html>
```

JSP : Exemple



Traitement des JSP

- **Les étapes 2, 3 et 4 effectuées uniquement à la première invocation de la page JSP ou au lancement de l'application**



Java Server Pages

- La spécification JSP définit les éléments suivants : directives, scripts et actions
- Il existe deux syntaxes pour chacun de ces éléments :
- la syntaxe XML pour les pages XHTML :

  <jsp:directive attributs />
- et la syntaxe JSP, qui n'est pas conforme à XML :

  <%@ attributs %>

JSP : directives

- Les directives fournissent des informations sur la JSP et ne produisent rien.
- Il existe trois directives :
  - **page**
  - **include**
  - **taglib**
- Deux syntaxes sont possibles:

  <%@ directive attributs %>

  ou **<jsp:directive attributs />**
- La directive **page** sert à indiquer les attributs de page : langage de programmation (java), le type MIME (Multipurpose Internet Mail Extension), l'encodage des caractères de la réponse,..

<%@ page contentType=" text/html;ISO-8859-1 " language= "java" %> ou
<jsp:directive.page contentType=" text/html;ISO-8859-1 " language= "java" />

La directive page

Principaux attributs de la directive page

- **<%@page import**=" java.util.*, java.sql.Connection" **%>**
  - même chose que les fichiers sources Java.
- **<%@page contentType**="text/html;charset=ISO-8859-1" **%>**
  - Indique le type MIME de la page et le jeu de caractères utilisés
- **<%@page session**="true|false" **%>**
  - Indique si la page est incluse ou non dans une session. Par défaut **true**, ce qui indique que la page fait partie d'une session et peut utiliser un objet de type **HttpSession** pour gérer des données de session
- **<%@page errorPage**="relativeURL" **%>**
  - Précise l'URL de la page JSP renvoyée au client au cas où une exception est levée
- **<%@page isErrorPage**=" true|false" **%>**
  - indique si la page JSP est une page de gestion d'erreur (dans ce cas l'objet exception peut être utilisée dans la page), false par défaut.
- **<%@page isThreadSafe**=" true|false" **%>**
  - Précise si la page peut être employée pour des accès simultanés. Par défaut true.
- …

La directive include

- La directive **include** sert à inclure une autre page (HTML, XHTML ou JSP) dans la page courante.
- On l'utilise pour insérer une page standard ou un élément commun à plusieurs pages (entête ou pied de page).
- Syntaxe : **<%@include file**="chemin relatif du fichier" **%>**
  - **ou** **<jsp:directive.include** file= "chemin relatif du fichier " **/>**
  - chemin relatif par rapport au répertoire qui contient la page JSP ou relatif par rapport au contexte de l'application Web si il débute par /
- Inclus le fichier dans le source JSP avant que celui-ci ne soit interprété (traduit en servlet) par le moteur JSP
- Le fichier peut être un fragment de code JSP, XHTML ou Java
- Insertion à la compilation et non pas à l'exécution un changement du fichier inclus ne provoque pas une régénération de la servlet.

```
<%@ include file= "header.jsp" %> ou
<jsp:directive.include file= " header.jsp " />
```

La directive taglib

- La directive **taglib** déclare qu'une page utilise l'une des bibliothèques standards en l'identifiant de façon unique par une **URI** et un préfixe.
- Avec la syntaxe XML, ces deux informations sont regroupées dans un espace de noms unique (xmlns).
- Dans l'exemple suivant, la bibliothèque de marqueurs **http://java.sun.com/jstl/core** est disponible pour la page via le préfixe **c** :

```
<%@ taglib uri= "http://java.sun.com/jstl/core " prefix= "c" % >
<jsp:root xmlns:c= "http://java.sun.com/jstl/core " >
```


JSP : scripts

- Les scripts contiennent du code java qui permet de manipuler des objets et d'effectuer des traitements affectant le contenu.
- Les scripts utilisent les deux syntaxes suivantes :
  - **<%! déclaration %>**
  - **<jsp:declaration>** ceci est une déclaration **</jsp:declaration>**
  - **<% scriptlet %>**
  - **<jsp:scriptlet>**ceci est un scriptlet**</jsp:scriptlet>**
  - **<%= expression %>**
  - **<jsp:expression>** ceci est une expression **</jsp:expression>**
- Les déclarations permettent de déclarer les variables ou les méthodes qui seront disponibles pour tous les autres scripts de la page.
- La déclaration n'apparait que dans la JSP traduite (c.-à-d. servlet), pas dans ce qui est envoyé au client.

JSP : scripts

- Exemple l'instance ArrayList sera globale à toute la page :

```
<%!  ArrayList  books=new ArrayList(); %>
<jsp:declaration> ArrayList  books=new ArrayList(); </jsp:declaration>
```

- Les scriptlets contiennent du code java pour décrire les actions à réaliser en réponse aux requêtes. Ils peuvent effectuer des itérations ou des conditions d'autres éléments de la JSP. Le code d'un scriptlet n'apparaît que dans la JSP traduite (la servlet).
- L'exemple suivant ajoute un objet book à l'ArrayList d'avant :

```
<% books.add(new Book("H2G2",12f, "Scifi IT  book", "1234-234",241,true )); %>
<jsp:scriptlet> books.add(new Book("H2G2",12f, "Scifi IT  book", "1234-
 234",241,true ));  </jsp:scriptlet>
```


JSP : scripts

- Les expressions servent à envoyer la valeur d'une expression java au client.
- Elles sont évaluées au moment de la réponse et leur résultat est converti en chaine de caractères puis inséré dans le flux affiché par le navigateur.
- L'exemple suivant affiche l'ISBN d'un livre :

  <%=book.getIsbn()%> ou <jsp:expression>book.getIsbn()</jsp:expression>
- Les déclarations, les scriptlets et les expressions doivent contenir du code java correct. La syntaxe XML doit être respectée aussi.
- **Exemple** : <%! ArrayList<Book> books=new ArrayList<Book>();%>

  déclare une ArrayList de livres en utilisant une classe générique.

  En XML : on doit pas utilisé les symboles < et > réservés à l'ouverture et la fermeture des marqueurs mais une section CDATA (character DATA) afin que le parser XML ne tente pas de l'analyser :

  < jsp:declaration ><**![CDATA[**

  ArrayList<Book> books=new ArrayList<Book>();**]]**></jsp:declaration>

Commentaires

- **<!-- ... -->**
  - Commentaires HTML
  - Intégralement reconduit dans le fichier HTML généré (transmis au client)
- **<%-- ... --%>**
  - Commentaires cachés
  - Contenu ignoré par le moteur JSP

```
<%@page contentType="text/html"%>
<%@page import="java.util.Date"%>
<html>
    <head><title>Commentaires dans une page JSP</title></head>
    <%-- Ce commentaire ne sera pas transmis au client --%>
    <!-- cette page a été générée le <%= new Date()%> -->
   <body> <H1>BONJOUR TOUT LE MONDE </H1>
   </body>
</html>
```

page jsp

Commentaire HTML

```
<html>
<head><title>Commentaires dans une page JSP</title></head>
<!-- cette page a été générée le Mardi 18 Mars 11:04:29 2012 -->
<body>
<H1>BONJOUR TOUT LE MONDE </H1>
</body>
</html>
```

HTML généré

Gestion des erreurs

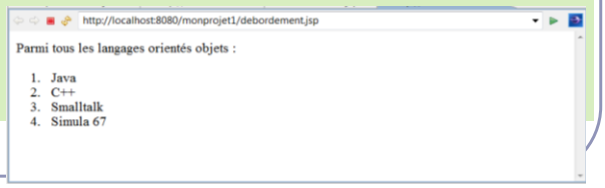
- Si une exception est levée dans une page JSP et n'est pas capturée
  - Si il n'y a pas de page d'erreur associée à la JSP alors on a l'affichage de la pile d'exécution
  - Si une page d'erreur est associée alors redirection vers cette page

Page erreur.jsp

```
<%@ page isErrorPage="true"%>
<html><body> exception levée <b> <%= exception %> </b>
<hr> <h3>trace de la pile</h3>
<pre>
<% java.io.PrintWriter myWriter = new java.io.PrintWriter(out);
    exception.printStackTrace(myWriter);
%>
</pre> </body></html>
```

Gestion des erreurs : page générant une erreur

```
<%@page errorPage="erreur.jsp" %>
<%! String[] langages = {"Java", "C++", "Smalltalk", "Simula 67"};
%>
<p>Parmi tous les langages orientés objets :</p>
<ol>
<%
    // levée d'une ArrayIndexOutOfBoundsException
    //  est effectuée si on fait 5 dans for au lieu de 4. 4 affiche le résultat si dessous :
    for (int i=0; i < 5; i++) {
    out.println("<li>" + langages[i] + "</li>");
    }
%>
</ol>
```



Gestion des erreurs

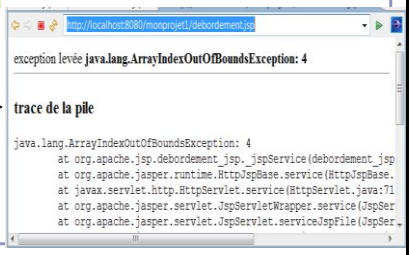
- Si une exception est levée dans une page JSP et n'est pas capturée
  - Si une page d'erreur est associée alors redirection vers cette page

erreur.jsp

```
<%@ page isErrorPage="true"%>
<html><body> exception levée <b> <%= exception %> </b>
<hr> <h3>trace de la pile</h3>
<pre>
<% java.io.PrintWriter myWriter = new java.io.PrintWriter(out);
    exception.printStackTrace(myWriter);
%>
</pre> </body></html>
```

debordement.jsp

```
<%@ page errorPage="erreur.jsp"%>
<%! String[] langages = {"Java", "C++", "Smalltalk", "Simula
67"};%>
<p>Parmi tous les langages orientés objets :</p>
<ol>
<% // levée d'une ArrayIndexOutOfBoundsException
for (int i=0; i < 5; i++) {
out.println("<li>" + langages[i] + "</li>");
}
%> </ol>
```



JSP : actions

- Les actions sont définies par la spécification JSP et forcent la page à effectuer certaines actions :inclure des ressources externes, faire suivre une requête vers une autre page, utiliser les propriétés d'objets java …
- Ces actions ressembles à des marqueurs HTML car elles sont représentées par des éléments XML préfixés par jsp : **<jsp:useBean>, <jsp:include>,** etc.
- Les actions disponibles sont :
  - **useBean** : associe une instance d'objets à une portée donnée et à un identifiant.
  - **setProperty** : fixe la valeur d'une propriété d'un bean.
  - **getProperty** : affiche la valeur d'une propriété d'un bean.
  - **include** : permet d'inclure des ressources statiques et dynamiques dans le même contexte que celui de la page courante.

JSP : actions

- **forward** : fait suivre la requête courante à une ressource statique ou une servlet dans le même contexte que celui de la page courante.
- **param** : utilisé avec les éléments include, forward et params. La page incluse ou transférée verra l'objet requête initial avec les paramètres originaux, plus les nouveaux.
- **plugin** : permet à une JSP de produire du HTML contenant des constructions spécifiques au navigateur (OBJECT ou EMBED), qui provoquera le téléchargement d'une extension.
- **params** : passe des paramètres. Fait partie de l'action plugin.
- **element** : définit dynamiquement la valeur du marqueur d'un élément XML.
- **attribute** : définit un attribut XML. Fait partie de l'action element.
- **body** : définit le corps d'un élément XML. Fait partie de l'action element

JSP et Java Beans

- Un des objectif des JSP par rapport aux Servlets c'est de ne pas mélanger du code HTML au code Java des Servlets
- D'un autre coté si c'est pour mettre du code Java dans le code JSP qu'est ce qu'on y gagne ?
- Un point important dans la conception de pages JSP est de minimiser le code Java embarqué dans les pages
- Déporter la logique métier dans des composants objets qui seront accédés depuis les pages JSP
  - Simplification des traitements inclus dans la JSP
  - Possibilité de réutilisation des composants depuis d'autres JSP ou d'autres composants
- Les spécifications JSP définissent une manière standard d'interagir avec des composants Java Beans

Java Beans : Rappel

- Les Java Beans sont des classes Java (POJO = Plain Old Java Objects) qui suivent certaines conventions :
  - Doivent avoir un constructeur vide (sans argument) :
    - On peut satisfaire cette contrainte soit en définissant explicitement un tel constructeur, soit en ne spécifiant aucun constructeur
  - Ne doivent pas avoir d'attributs publics
  - La valeur des attributs doit être manipulée à travers des méthodes (accesseurs) get*Xxx* et set*Xxx*
    - *Si une classe possède une méthode getTitle qui retourne un* String, on dit que le bean possède une propriété String nommée title
    - Les propriétés Boolean utilisent is*Xxx à la place de getXxx*

Java Beans : exemple

• Un exemple de bean

Propriété privée

Accesseur (getter)

Modifieur (setter)

```
package test;
public class Personne {
    private String nom;
    private String prenom;
    private int age = 0;
    private boolean marie;
    public Personne() {
            this.nom = "X"; // nom par défaut
            this.prenom = "x"; // prénom par défaut
        }
public String getNom() { return (this.nom);}
public void setNom(String nom) { this.nom = nom;}
...
public int getAge () { return (this.ge);}
public void setAge(int age) { this.age = age;}
public boolean isMarie() { return (this.marie);}
public void setMarie(boolean marie) {
 this.marie = marie;}

}
```


Utiliser un bean depuis une JSP

- Le tag <jsp:useBean>
  - Permet de localiser une instance ou bien d'instancier un bean pour l'utiliser dans la JSP
- Syntaxe

```
<jsp:useBean
    id="beanInstanceName"
    class="package.class"
    type="package.class"
    scope="page|request|session|application"
/>
```

Nom utilisé pour accéder au bean dans la page (doit etre unique)

Le nom de la classe du bean

Optionnel, le type de la variable référençant le bean. Classe du bean, sa classe parente etc.

Optionnel, détermine la portée durant laquelle le bean est défini et utilisable

• le bean demandé est déjà instancié pour la portée précisée :

→renvoi de sa référence

le bean demandé n'est pas déjà présent pour la portée précisée :

→instanciation du bean

→ajout de celui-ci à la portée spécifiée

Utiliser un bean depuis JSP

L'attribut scope

page **valeur par défaut**
bean utilisable dans toute la page JSP ainsi que dans les fichiers statiques inclus.

request bean accessible durant la durée de vie de la requête. La méthode **getAttribute()** de l'objet **request** permet d'obtenir une référence sur le bean.

session bean utilisable par toutes les JSP qui appartiennent à la même session que la JSP qui a instanciée le bean. Le bean est utilisable tout au long de la session par toutes les pages qui y participent. La JSP qui a créé le bean doit avoir l'attribut session = "true" dans sa directive de page.

application bean utilisable par toutes les JSP qui appartiennent à la même application que la JSP qui a instanciée le bean. Le bean n'est instancié que lors du rechargement de l'application.

Utiliser un bean depuis une JSP

- Exemple

```
<jsp:useBean
      id="utilisateur"
      class="test.Personne"
      scope="session"
/>
```

-Définition dans la session d'un java bean instance de la classe Personne du package test et désigné par la référence utilisateur.
-Les Beans doivent être déployés dans le même répertoire que les autres classes Java.
WEB-INF/classes/*folderPackage*
-LesBeans doivent *toujours être dans des* packages!

```
<HTML>
   <BODY>
      <UL>
        <LI>NOM : <%=utilisateur.getNom()%></LI>
        <LI>PRENOM : <%=utilisateur.getPrenom()%></LI>
       <LI>AGE : <%=utilisateur.getAge()%></LI>
      </UL>
  </BODY>
</HTML>
```

Fixer les propriétés d'un bean depuis une JSP

- Le tag **<jsp:setProperty>**
  - Permet de modifier la valeur de un ou plusieurs attributs d'un bean (appel de setXxx())
- Syntaxe

```
<jsp:setProperty name="beanInstanceName"
    property="propertyName"
    value="propertyValue" />
```

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<jsp:setProperty name="utilisateur" property="nom" value="Toto"/>
<jsp:setProperty name="utilisateur" property="age" value="34"/>
```

Quand le type de la propriété du Bean n'est pas String une conversion automatique est effectuée en utilisant la méthode valueOf de la classe enveloppe

```
<%utilisateur.setAge(Integer.valueOf("34"));%>
```


Fixer les propriétés d'un bean depuis une JSP

- Possibilité de fixer les propriétés du bean à l'aide des paramètres de la requête (la méthode la plus utilisée)
- Syntaxe

```
<jsp:setProperty name="beanInstanceName" property="propertyName"/>
```

Le nom de la propriété est le même que le nom du paramètre de la requête

```
<jsp:setProperty name="beanInstanceName" property="propertyName"
param="paramétrage"/>
```

Le nom de la propriété est différent du nom du paramètre de la requête

⇩ MaPageJSP?name="Toto"&age="24"

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<jsp:setProperty name="utilisateur" property="nom" param="name"/>
<jsp:setProperty name="utilisateur" property="age" />
```

```
<jsp:setProperty name="beanInstanceName" property="*"/>
```

Fixe toutes les propriétés correspondant à des paramètres de la requête

Accéder aux propriétés d'un bean depuis une JSP

- **Le tag <jsp:getProperty>**
  - Permet d'obtenir la valeur d'un attribut d'un bean
- Syntaxe

```
<jsp:getProperty name="beanInstanceName" property="propertyName" />
```

```
<jsp:useBean id="utilisateur" class="test.Personne" scope="session"/>
...
<body>
<UL>
<LI>Nom : <jsp:getProperty name="utilisateur" property="nom"/></LI>
<LI>Age : <jsp:setProperty name="utilisateur" property="age"/></LI>
</UL>
```


Tag de redirection

- **Le tag <jsp:forward>**
  - Permet de rediriger la requête vers un fichier HTML, une autre page JSP ou une Servlet
- Syntaxe

```
<jsp:forward page="relativeURL|<%=expression%>"/>
```

```
...
<jsp:forward page="uneAutrePage.jsp" />
```

Si URL commence par un / elle est absolue (contexte de l'application) sinon elle est relative à la JSP

Ce qui suit l'action forward est ignoré, et tout ce qui a été généré dans cette page JSP est perdu

- Possibilité de passer un ou plusieurs paramètres vers la ressource appelée

```
<jsp:forward page="relativeURL|<%=expression%>">
<jsp:param name="parametre" value="string|<%=expression%>">
...
</jsp:forward>
```

Tag d'inclusion

- **Le tag <jsp:include>**
- Permet d'intégrer **dynamiquement** un contenu généré par une autre page JSP ou une autre servlet.
- Syntaxe

```
<jsp:include page="relativeURL" flush="true|false"/>
```

spécifie si le tampon doit être envoyé au client et vidé

- Comme pour <jsp:forward> possibilité de passer un ou plusieurs paramètres vers la ressource incluse en utilisant le tag <jsp:param>

Les objets implicites

- Comme les pages JSP sont des servlets, il est normal de retrouver les mêmes objets. La particularité, c'est qu'ils existent implicitement.
- Le modèle JSP définit un certain nombre d'objets implicites. Ces objets sont appelés ainsi car nous pouvons y accéder sans jamais avoir à les déclarer ou à les initialiser. Les objets implicites sont accessibles dans les scriptlets et dans les expressions.
- request
- response
- out
- session
- config
- exception
- application

L'objet request

- Les pages JSP sont des composants Web dont le rôle est de répondre à des requêtes HTTP.
- L'objet implicite request représente la requête que doit traiter la page.
- Grâce à cet objet, il est possible de lire les en-têtes de la requête, ses paramètres, ainsi que de nombreuses autres informations.
- Le plus souvent, l'objet request est utilisé pour connaître les paramètres de la requête.
- String request.getParameter(String nom) ;
- La méthode getParameter(String) retourne la valeur du paramètre dont le nom correspond à la chaîne utilisée comme argument. Le nom utilisé comme argument est celui employé dans le formulaire.
- Exemple : <h3><%= request.getParameter("prenom") %></h3>

L'objet out

- L'objet implicite out est une référence au stream de sortie utilisée par la réponse.
- Il peut être employé dans une scriptlet pour écrire des données dans la réponse envoyée au client.
- Ainsi, le code suivant :
- <h3><%= request.getParameter("prénom") %></h3>
- Peut être remplacé par :
- <% out.println("<h3>"+request.getParameter("prénom")+"</h3>") ; %>
- Dans cet exemple, l'utilisation de l'objet implicite out ne procure pas un grand intérêt.
- Il est très utile lorsque nous sommes dans une grande scriptlet, et que nous ne désirons pas placer du code HTML au milieu de cet scriptlet.

L'objet session

- HTTP est un protocole sans état : chaque requête envoyée par un client est une nouvelle requête, sans aucune relation avec les précédentes.
- Or, une applications Web exige l'interaction d'un client avec l'application qui s'étend souvent sur plusieurs requêtes et réponses.
- Une session est l'ensemble d'une conversation entre un client et le serveur.
- Les composants JSP d'une application Web participent automatiquement à une session.
- Une page désirant ne pas participer à la session doit utiliser la directive page dans laquelle l'attribut session prend la valeur false.
- Seuls des objets, et non des primitives Java, sont placés dans une session. Les primitives doivent être enveloppées dans leurs classes : Integer, Double, ...

L'objet session

- Les méthodes pour placer et retrouver des objets dans la session sont :
- Object setAttribute(String nom, Object valeur) ;
  Object getAttribute(String nom) ;
  Enumeration getAttributeNames ( ) ;
  void removeAttribute(String nom) ;
- On peut placer un JavaBean dans la session mais il faut changer son scope, par défaut de la portée de la page JSP en cours.
- Pour stocker le JavaBean Personne dans la session, on doit apporter la modification suivante :
- <jsp:useBean id = "utilisateur" class = "bd.Personne" scope = "session" />
- Cet objet stocké dans la session, est retrouvé :
- Personne operateur = (Personne) session.getAttribute("utilisateur") ;

L'objet application

- Cet objet représente l'environnement de l'application Web.
- Il est utilisé pour lire les paramètres de configuration de l'application. Ces paramètres sont définis dans le descripteur de déploiement, dans l'élément <webapp> :

```
<webapp> <context-param>
<param-name>nom</param-name>
 <param-value>valeur</param-value> </context-param> </webapp>
```

- Dans la page JSP, le paramètre de configuration peut être récupéré au moyen de la méthode getInitParameter(String) de l'objet implicite application :
- application.getInitParameter(String nom) ;

L'objet config

- Cet objet lit les paramètres d'initialisation spécifiques aux pages JSP.
- Ces paramètres définis dans le descripteur de déploiement concernent une page particulière et non plus toute l'application Web comme pour l'objet application.
- Ces paramètres figurent dans l'élément <servlet> car la page une fois compilée est en fait une servlet.
- L'élément <servlet> peut contenir un ou plusieurs éléments <init-param>, comme suit :

```
<servlet> <servlet-name>NouveauMessage</servlet-name>
        <servlet-class>NouveauMessage.class</servlet-class>
        <init-param> <param-name>nom</param-name>
      <param-value>valeur</param-value> </init-param> </servlet>
```

- Les paramètres définis dans le descripteur de déploiement sont accessibles grâce à la méthode getInitParameter(String) de l'objet implicite config :
- config.getInitParameter(String nom) ;

JSP : application

- À titre d'application, on va créer une page qui affichera une liste de livres stockés dans une ArrayList (sans utiliser une base de données).
- L'ArrayList est initialisée avec un nombre donné d'objets Book, et parcourue pour afficher les attributs de chaque livre (isbn, titre, …) comme dans la figure suivante :

Lists all the books

ISBN	Title	Price	Description	Number of pages	Illustrations
1234-234	H2G2	12.0	Scifi IT book	241	true
564-694	Robots	18.5	Best seller	317	true
256-6-56	Dune	23.25	The trilogy	529	true

JSP : application

- Pour construire cette page, nous avons besoin de plusieurs éléments :
- Il faut importer les classes Arraylist et Book :

```
<%@ page import= " fsr.ac.Book" %>
<%@ page import="java.util.ArrayList" %>
```

- Déclarer un attribut books pour qu'il soit accessible à toute la page :

```
<%! ArrayList<Book> books = new ArrayList<Book>();%>
```

- Un scriptlet ajoute des livres dans books et un autre parcourt cette liste avec for. Pour afficher les attributs d'un livre, on utilise l'expression :**<%= book.getTitle()%>**
- Dans le code il y a du java, HTML et du texte. Tout ce qui est dans un scriptlet (entre <% et %>) est du code java qui sera exécuté sur le serveur. Le reste c'est du texte qui sera affiché dans la page de réponse.
- Une JSP devient vite difficile à lire avec un tel mélange. En plus, il n' y a pas de séparation entre la logique métier et la présentation.
- Ce mélange de java (développeur métier) et le XHTML (concepteur web) complique la maintenance

JSP : application

```
<%@ page import="fsr.ac.Book" %>
<%@ page import="java.util.ArrayList" %>
<%!  ArrayList<Book> books = new ArrayList<Book>(); %>
<html> <head> <title>List all the books</title> </head>
<body> <h1>Lists all the books</h1> <hr/>
    <% books.add(new Book("H2G2", 12f, "Scifi IT book", "1234-234", 241, true));
          books.add(new Book("Robots", 18.5f, "Best seller", "564-694", 317, true));
          books.add(new Book("Dune", 23.25f, "The trilogy", "256-6-56", 529, true));%>
    <table border="1">
     <tr>  <th>ISBN</th>  <th>Title</th>  <th>Price</th>   <th>Description</th>
          <th>Number of pages</th>   <th>Illustrations</th>  </tr>
    <%     Book book;
          for (int i = 0; i < books.size(); i++) {  book = books.get(i);    %>
    <tr>  <td><%= book.getIsbn()%> </td>   <td><%= book.getTitle()%>    </td>
          <td><%= book.getPrice()%> </td>   <td><%= book.getDescription()%> </td>
```


JSP : application

```
      <td><%= book.getNbOfPage()%> </td>
      <td><%= book.getIllustrations()%>  </td>
   </tr>
   <%
        }  // fin du for
     %>

</table>
    <hr/>
    <i> Partie du cours Java EE 6</i>
</body>
```

Langage d'expressions (EL)

- Depuis la version 2.0 des JSP, il est possible de placer **à n'importe quel endroit d'une page JSP des expressions qui sont évaluées et remplacées par le résultat de leur évaluation : les Expressions Languages (EL) .**
- Elles permettent de manipuler les données au sein d'une page JSP plus simplement qu'avec les **scriptlets Java (<% … %>).**
- Utilisées conjointement avec des librairies de tags (telles que les JSTL que nous verrons plus tard), elles permettent de se passer totalement des scriptlets.
- Une EL permet également d'accéder simplement aux **beans des différents scopes de l'application web (page, request, session et application).**
- Elles permettent également d'afficher les valeurs de variables et disposent de nombreux opérateurs mathématiques, logiques et relationnels.

Langage d'expressions (EL)

- La syntaxe de base d'une instruction EL est de la forme : ${expr} où *expr* est une expression valide qui peut être analysée et interprétée. À partir de JSP 2.1 on peut aussi utiliser la syntaxe : #{expr}
- La chaîne **expr correspond à l'expression à interpréter. Une expression peut être composée de plusieurs termes séparés par des opérateurs.**
- Exemple :
- ${ (5 + 2)*2 } => *14*
- ${ a && b } => *a ET b*
- Ou, avec les beans: ${bean.property}

Langage d'expressions (EL)

- Les expressions EL peuvent utiliser la plupart des opérateurs java habituels :
  - Arithmétiques :+, -, *, / (div), %(mod);
  - Relationnels : ==, !=, <, >, <=, >=;
  - Logiques :&&, | |, !
  - Autres : ( ), empty, [ ], …
- Note : certains opérateurs ont à la fois une forme symbolique et littérale : (< est équivalent à lt, / à div,…) ce qui permet de rendre une JSP conforme à XML (par exemple : ${2<3} peut être représentée par ${2 lt 3}).
- L'opérateur empty teste si un objet est null ou s'il référence un String, List, Map ou tableau null : **${empty book} ou ${empty book.isbn}.**
- L'opérateur **point** permet d'accéder à un attribut d'un objet. Il est aussi possible d'utiliser l'opérateur [ ] : **${book.isbn} ou ${book[isbn] }.**
- EL 2.2 permet d'appeler des méthodes : **${book.buy('EURO') }**

Langage d'expressions (EL)

- Les différents **termes peuvent être :**
- **Un type primaire *(null, Long, Double, String, Boolean, …)***
- **Un objet implicite *(requestScope, sessionScope, param, paramValues, header, headerValues, …)***
- **Un attribut d'un des scopes de l'application. Cet attribut pouvant être stocké dans n'importe quel scope (page, request, session ou application).**
- **L'EL cherchera dans tous les scopes dans cet ordre. On a donc :**
  ${name} ⇔<%= pageContext.findAttribute("name"); %>
- Il est conseillé de n'utiliser cette forme que pour accéder aux objets du scope **page, afin d'éviter d'éventuels conflits.**
- **Pour les autres scopes, il est préférable d'utiliser les objets implicites ${*requestScope.name*}, ${*sessionScope.name*}, ${*applicationScope.name*}, ...**
- **Une fonction EL (voir JSTL)**

Langage d'expressions (EL)

- Il est possible d'indiquer au conteneur JSP 2.0 de ne pas interpréter les **EL d'une page. Il suffit pour cela d'utiliser l'attribut isELIgnored de la directive page:**

  <%@ page isELIgnored="true" %>
- Il est également possible de ne pas interpréter une **EL en particulier en la protégeant avec un anti-slash:**

  \${ ceci ne sera pas interprété comme une EL }
- **Gestion des Exceptions : Les EL gèrent un certains nombres d'exceptions afin de ne pas avoir à les traiter dans les JSP.**
- – **NullPointerException :** les différentes propriétés d'un élément ne sont pas forcément renseignées et peuvent très bien être *null.*

  Exemple *:* #{ sessionScope['data'].information.date}
- Si l'*attribut data n'est pas présent dans la session, ou si la méthode getInformation() retourne null...* ***Dans tous les cas, l'expression prendra la valeur null mais aucune exception ne sera levée.***

EL : exceptions

- **ArrayOutOfBoundException:**

  L'accès à un index incorrect d'un élément d'un tableau ou d'une **List ne provoquera pas d'exception mais renverra une valeur null.**
- **ELException:**

  Certaines exceptions peuvent toujours être levées, mais elles ne concernent pas directement les données mais l'expression **EL.**

  Exemple : lorsque l'on tente d'accéder à une propriété sans accesseur publique, ou que l'on accède à une propriété indexée avec un index qui ne correspond pas à un nombre entier…

EL : avantages

- Une meilleure **lisibilité: le code se limite quasiment au nom du bean et de sa propriété.**
- Pas de **casting, et de ce fait pas d'import (on accède aux propriétés par réflexion).**
- **Gestion des Exceptions : Soit la donnée à afficher est valide alors on l'affiche, soit elle n'est pas présente et on n'affiche rien... Cela permet de se passer d'effectuer plusieurs vérifications au sein des JSP avant d'afficher des données...**

EL : résumé

- Les **EL permettent donc de simplifier le code des pages JSP tout en améliorant la sécurité du code grâce à la gestion de certaine exception de base.**
- La notion d'**EL a été introduite afin de faciliter la conception de pages JSP, en particulier afin de pouvoir accéder et utiliser des données sans devoir maîtriser un langage aussi complexe que Java...**
- En effet, la logique de conception des pages JSP se rapproche de la logique de conception d'une page HTML ou d'un fichier XML.
- Ainsi, une formation de base peut permettre à un web designer de concevoir des pages JSP dynamiques en utilisant des beans créées par un développeur Java...

La bibliothèque de marqueurs standard de JSP (JSTL)

- De nombreux frameworks facilitent le développement d'application Java EE. **JSTL propose une librairie standard pour la plupart des fonctionnalités de base d'une application Java EE.**
- Le but de la **JSTL est de simplifier le travail du web designer , responsable de la couche présentation d'une application web JEE.**
- La **JSTL permet de développer des pages JSP en utilisant des balises XML, donc avec une syntaxe proche des langages utilisés par les web designers, et leur permet donc de concevoir des pages dynamiques complexes sans connaissances du langage Java.**
- La **JSTL se base sur l'utilisation des EL en remplacement des scriptlets Java.**
- Il existe différentes versions des **JSTL(1.0, 1.1, 1.2). La plus récente se base sur les JSP 2.0 qui intègre un moteur d'EL.**

La bibliothèque de marqueurs standard de JSP (JSTL)

- JSTL est un ensemble de marqueurs standard permettant d'éviter le mélange du code Java et des marqueurs XHTML.
- Avec JSTL, on peut manipuler les données dynamiques contenues dans une JSP en utilisant des marqueurs XML au lieu de passer par java.
- Les actions possibles vont de l'affectation d'une valeur à un objet à la capture des exceptions en passant par le contrôle du flux avec des conditions et des itérateurs et par l'accès aux bases de données.
- Une bibliothèque de marqueurs est une collection de fonctions pouvant être utilisées dans une page JSP ou JSF.
- Ci-dessous, on énumère ces fonctions, les URI permettant de référencer les bibliothèques et les préfixes associés :

Domaine	URI	Préfixe
Noyau	http://java.sun.com/jsp/jstl/core	c
Traitement XML	http://java.sun.com/jsp/jstl/xml	x
I18N et formatage	http://java.sun.com/jsp/jstl/fmt	fmt
Accès BD	http://java.sun.com/jsp/jstl/sql	sql
Fonctions	http://java.sun.com/jsp/jstl/functions	fn

La bibliothèque de marqueurs standard de JSP (JSTL)

- Avant d'utiliser ces actions, la JSP doit importer l'URI de la bibliothèque et choisir un préfixe :
  - soit en utilisant une directive JSP avec le système de marqueurs JSP.
  - soit en utilisant une syntaxe XML.

```
<%@  taglib uri="http://java.sun.com/jsp/jstl/core"  prefix ="c" %> ou
<jsp:root        xmlns:jsp=" http://java.sun.com/JSP/Page "
                 xmlns:c= "http://java.sun.com/jstl/core"     version="1.2" >
```

- Avec cette déclaration, on peut utiliser toutes les actions de la bibliothèque des marqueurs fondamentaux en utilisant le préfixe c :
- <**c:set** var="upperLimit" value="20" /> : ce code initialise la variable upperLimit avec la valeur 20.

JSTL : actions fondamentales

- Les actions fondamentales ci-dessous, fournissent des marqueurs pour manipuler des variables, traiter des erreurs, effectuer des tests et exécuter des boucles et des itérations :

Action	Description
<c:out>	Évalue une expression et affiche son résultat
<c:set>	Initialise la valeur d'un objet
<c:remove>	Supprime une variable
<c:catch>	Capture une exception java.lang.Throwable lancée par l'une de ses actions imbriquées
<c:if>	Teste si une expression est vraie
<c:choose>	Fournit plusieurs alternatives exclusives
<c:when>	Représente une alternative dans une action <c:choose>
<c:otherwise>	Représente la dernière alternative d'une action <c:choose>

JSTL : actions fondamentales

Action	Description
<c:forEach>	Répète son corps pour chaque élément d'une collection ou un nombre fixé de fois
<c:forTokens>	Itère sur une liste de tokens séparés par des virgules
<c:import>	Importe une ressource
<c:url>	Encode une URL
<c:param>	Ajoute des paramètres de requête à une URL.
<c:redirect>	Redirige vers une URL précise

- Pour illustrer le fonctionnement de certains de ces marqueurs, l'exemple suivant présente une JSP qui boucle sur les nombres de 3 à 15 en affichant, pour chacun d'eux, s'il est pair ou impair.
- On peut remarquer que tout le traitement s'effectue grâce aux marqueurs et que cette page est conforme à XML et compréhensible même si on connait pas java.

JSTL : actions fondamentales

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <head>  <title>Checking numbers</title> </head>
<body> <h1>Checking numbers</h1>
<hr/>
< c:set var="upperLimit" value="20"/>
<c:forEach var="i" begin="3" end="${upperLimit - 5}">
   <c:choose>
      <c:when test="${i%2 == 0}">  <c:out value="${i} is even."/><br/>  </c:when>
      <c:otherwise>   <c:out value="${i} is odd."/><br/>    </c:otherwise>
   </c:choose>
</c:forEach>
</body>
</html>
```

JSTL : actions fondamentales

- l'exemple précédent affiche le résultat suivant :

Checking numbers

3 is odd.
4 is even.
5 is odd.
6 is even.
7 is odd.
8 is even.
9 is odd.
10 is even.
11 is odd.
12 is even.
13 is odd.
14 is even.
15 is odd.

JSTL : actions de formatage

- Les actions de formatages permettent de formater des dates, des nombres, des valeurs monétaires et des pourcentages. Elles reconnaissent aussi l'internationalisation (i18n) : on peut obtenir ou modifier les locales (variable de langue) et les zones horaires et obtenir l'encodage de la page web.

Action	Description
<fmt:message>	Internationalise une JSP en extrayant un message en fonction de la langue
<fmt:param>	Fournit un paramètre à <fmt:message>
<fmt:bundle>	Indique le paquetage contenant les message par langue
<fmt:seLocale>	Fixe la langue à utiliser
<fmt:requestEncoding>	Fixe l'encodage des caractères de la requête
<fmt:timeZone>	Précise la zone horaire du format de l'heure
<fmt:setTimeZone>	Stocke la zone horaire indiquée dans une zone

JSTL : actions de formatage

Action	Description
<fmt:formatNumber>	Formate une valeur numérique (nbre, monnaie,%) selon la locale
<fmt:formatDateEach>	Formate les dates et les heures selon la langue
<fmt:parseDate>	Analyse la représentation textuelle des dates et des heures

- L'exemple suivant montre comment utiliser ces marqueurs.
- La page importe les bibliothèques de marqueurs fondamentaux et de formatage.
- La variable now initialisée avec la date courante, et le marqueur la formate selon différents critères : uniquement l'heure, juste la date et les deux ensemble.
- On fixe la langue américaine et formate la valeur monétaire en $20.50.
- La locale est ensuite modifier pour la Grande-Bretagne afin que cette valeur soit exprimée en livres sterling.

JSTL : actions de formatage

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<html> <head> <title>Formatting data</title> </head>
<body> <h1>Formatting data</h1>
<hr/>  Dates<br/> <c:set var="now" value="<%=new java.util.Date()%>"/>
        <fmt:formatDate type="time" value="${now}"/><br/>
        <fmt:formatDate type="date" value="${now}"/><br/>
        <fmt:formatDate type="both" dateStyle="short" timeStyle="short"
                value=" ${now}"/><br/>
        <fmt:formatDate type="both" dateStyle="long" timeStyle="long"
                value=" ${now}"/><br/>
    Currency<br/> <fmt:setLocale value="en_us"/>
        <fmt:formatNumber value="20.50" type="currency"/><br/>
        <fmt:setLocale value="en_gb"/>
        <fmt:formatNumber value="20.50" type="currency"/><br/>
</body>  </html>
```

JSTL : actions de formatage

- L'exemple précédent produit le résultat suivant :

JSTL : actions SQL

- Les actions SQL de la JSTL permettent d'effectuer des requêtes sur une base de données (insertions, modifications et suppressions), d'accéder aux résultats de ces requêtes et même de mettre en place un contexte transactionnel.
- Normalement pour l'accès à une base de données se fait avec les entités JPA et les EJB mais, pour des applications spécifiques, on a besoin d'accéder à une base à partir d'une page web. Les marqueurs de la bibliothèque SQL sont :

Action	Description
<sql:query>	Interroge une base de données
<sql:update>	Exécute une instruction SQL INSERT, UPDATE ou DELETE.
<sql:transaction>	Établit un contexte transactionnel pour les marqueurs <sql:query> et <sql:update>
<sql:setDataSource>	Indique la source de données
<sql:param>	Fixe les valeurs des marqueurs d'emplacements (?) d'une instruction SQL
<sql:dataParam>	Fixe les valeurs des marqueurs d'emplacements (?) d'une instruction SQL pour les valeurs de type java.util.Date.

JSTL : actions SQL

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib  uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"  %>
<html> <head>  <title>Lists all the books</title>  </head>
<body>  <h1>Lists all the books</h1>
<hr/>       <sql:setDataSource dataSource="jdbc/__default"/>
            <sql:query var="books">   select * from book  </sql:query>
<table border="1">
   <tr>   <th>ISBN</th>    <th>Title</th>    <th>Price</th>   <th>Description</th>
          <th>Number of pages</th>  <th>Illustrations</th>                    </tr>
   <c:forEach var="row" items="${books.rows}">
    <tr> <td><c:out value="${row.isbn}"/></td>     <td><c:out value="${row.title}"/></td>
         <td><c:out value="${row.price}"/></td>   <td><c:out value="${row.description}"/></td>
         <td><c:out value="${row.nbOfPage}"/></td>
         <td><c:out value="${row.illustrations}"/></td>
      </tr>
    </c:forEach>
</table>                </body>                </html>
```


JSTL : actions SQL

- Cette JSP importe une bibliothèque sql et indique une source de données ici jdbc/__default fournie avec GlassFish.
- Le résultat de la requête select est stocké dans la variable books et toutes les lignes obtenues se trouvent dans la collection books.rows, que l'on parcourt avec le marqueur <c:forEach>.
- Le résultat d'affichage de l'exemple précédent est comme suit :

Lists all the books

ISBN	Title	Price	Description	Number of pages	Illustrations
1234-234	H2G2	12	Scifi IT book	241	true
564-694	Robots	18.5	Best seller	317	true
256-6-56	Dune	23.25	The trilogy	529	false

JSTL : actions XML

- La bibliothèque de marqueurs XML ressemble à la bibliothèque de marqueurs fondamentaux.
- Elle permet d'effectuer une analyse XML, d'itérer sur les éléments des collections, d'effectuer des opérations reposant sur les expressions Xpath et de réaliser des transformations à l'aide de documents XSL.

Action	Description
<x:parse>	Analyse un document XML
<x:out>	Évalue une expression XPATH et produit son résultat
<x:set>	Évalue une expression XPATH et stocke son résultat dans une variable
<x:if>	Évalue l'expression XPATH si l'expression est vraie
<x:choose>	Fournit plusieurs alternatives exclusives
<x:when>	Représente une alternative dans une action <x:choose>
<x:otherwise>	Représente la dernière alternative d'une action <x:choose>

JSTL : actions XML

Action	Description
<x:forEach>	Évalue une expression XPATH et répète son contenu sur le résultat
<x:transform>	Applique une feuille de style XSLT à un document XML
<x:param>	Ajoute les paramètres de la transformation <x:transform>

- Nous avons besoin d'un document XML pour effectuer les transformations réalisées par ces marqueurs.
- Soit un fichier *books.xml* qui contient une liste de livres.
- Soit un fichier JSP qui analyse et affiche tous les livres de *books.xml*
- Cette JSP importe la bibliothèque des marqueurs XML et charge le fichier *books.xml* dans une variable *bookUrl* qui contient le texte brut, qui doit être analysé par <x:parse>-le DOM résultat sera stocké dans la variable *doc*.
- Lorsque le document est analysé, on peut le parcourir et afficher les valeurs en utilisant des expressions XPATH avec <x:out>

JSTL : actions XML

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>
<html>    <head> <title>Lists all the books</title> </head>
<body>   <h1>Lists all the books</h1>
<hr/> <table border="1">
   <tr>  <th>ISBN</th>  <th>Title</th>  <th>Price</th>  <th>Description</th>
          <th>Number of pages</th>  <th>Illustrations</th>  </tr>
   <c:import url="books.xml" var="bookUrl"/>
   <x:parse xml="#{bookUrl}" var="doc"/>
   <x:forEach var="b" select="$doc/books/book">
    <tr> <td><x:out select="$b/@isbn"/></td>  <td><x:out select="$b/title"/></td>
        <td><x:out select="$b/@price"/></td>  <td><x:out select=" $b/description"/></td>
       <td><x:out select=" $b/@nbOfPage"/></td> <td><x:out select="$b/@illustrations"/></td>
     </tr>
   </x:forEach>
</table>
</body>  </html>
```


JSTL : actions XML

```
<?xml version='1.0' encoding='UTF-8'?>
<books>  <book isbn='1234-234' price='12' nbOfPage='241' illustrations='true'>
            <title>H2G2</title>  <description>Scifi IT book</description>  </book>
    <book isbn='564-694' price='18.5' nbOfPage='317' illustrations='true'>
            <title>Robots</title>   <description>Best seller</description>  </book>
     <book isbn='256-6-56' price='23.25' nbOfPage='529' illustrations='false'>
            <title>Dune</title> <description>The trilogy</description>  </book>
</books>
```

Lists all the books

ISBN	Title	Price	Description	Number of pages	Illustrations
1234-234	H2G2	12	Scifi IT book	241	true
564-694	Robots	18.5	Best seller	317	true
256-6-56	Dune	23.25	The trilogy	529	false

JSTL : Fonctions

- Les fonctions ne sont pas des marqueurs et sont quand même définies dans la spécification JSTL.
- Elles peuvent être utilisées avec EL et sont principalement employées pour traiter les chaines de caractères : **${fn:contains("H2G2", "H2")}**
- Ce code teste si une chaine contient une sous-chaine particulière : ici l'appel renverra true car H2G2 contient H2.
- **${fn:length("H2G2")}** : l'appel renvoie la longueur d'une chaine (ici 4)
- Une JSP peut afficher les résultats des fonctions (avec <c:out>), ou les utiliser dans un test ou une boucle.

Action	Description
<fn:contains>	Teste si une chaine contient la sous chaine indiquée
<fn:containsIgnoreCase>	Idem, sans tenir compte de la casse
<fn:endsWith>	Test si une chaîne se termine par le suffixe indiqué
<fn:escapeXml>	Protège les caractères pouvant être interprétés comme du XML

JSTL : Fonctions

Action	Description
<fn:indexOf>	Renvoie l'indice de la première occurrence d'une sous-chaîne dans une chaîne
<fn:join>	Joint tous les éléments d'un tableau pour former une chaîne
<fn:length>	Renvoie le nombre d'éléments d'une collection ou taille de string
<fn:replace>	Renvoie une chaîne où toutes les occurrences de la sous-chaîne indiquée ont été remplacées par une autre chaîne
<fn:split>	Découpe une chaîne pour obtenir un tableau de sous-chaînes
<fn:startsWith>	Teste si une chaîne commence par le préfixe indiqué
<fn:substring>	Renvoie une sous chaîne
<fn:substringAfter>	Renvoie la sous chaîne située après la sous chaîne indiquée
<fn:substringBefore>	Renvoie la sous chaîne située avant la sous chaîne indiquée
<fn:toLowerCase>	Convertit une chaîne en minuscules
<fn:toUpperCase>	Convertit une chaîne en majuscules
<fn:trim>	Supprime les espaces aux deux extrémités d'une chaîne

JSTL : Fonctions

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html> <head>   <title>Use JSTL Functions</title> </head>
<body> <h1>Use JSTL Functions</h1>
<hr/> Should be true : ${ fn:contains("H2G2", "H2") }<br/>
     Should be false : ${ fn:contains("H2G2", "h2") }<br/>
     Should be true : ${ fn:containsIgnoreCase("H2G2", "h2") }<br/>
     Should be false : ${ fn:endsWith("H2G2", "H2") }<br/>
     Should be true : ${ fn:endsWith("H2G2", "G2") }<br/>
     Should be 2 : ${fn:indexOf("H2G2", "G2")}<br/>
     Should be 4 : ${fn:length("H2G2")}<br/>
<c:if test="${fn:length('H2G2') == 4}">   It is four caracters long </c:if>
</body>
</html>
```


JSTL : Fonctions

- L'exemple précédent affiche le résultat suivant :

Use JSTL Functions

Should be true : true
Should be false : false
Should be true : true
Should be false : false
Should be true : true
Should be 2 : 2
Should be 4 : 4
It is four caracters long