

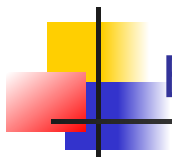
Librairie Standard,
Classes Enveloppes, chaines de
caractères et fichiers



Librairie Standard, Classes Enveloppes, chaines de caractères et fichiers

1

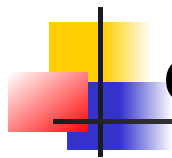
- l'API de la Librairie
- entetes de l'API
- La classe `Math`
- Classes enveloppes pour Types Primitifs
- La classe `String` (Les chaines de caractères)
- La classe `File` (Les fichiers texte)



Éléments de la librairie standard

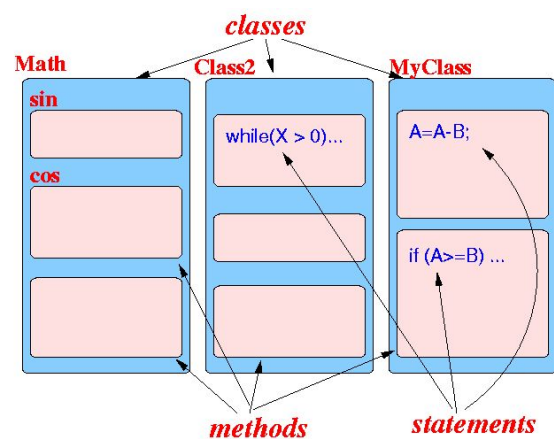
1

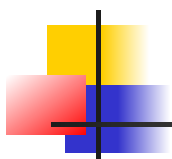
- l'API de la Librairie
- entetes de l'API
- La classe `Math`
- Classes enveloppes pour Types Primitifs



Classes et méthodes

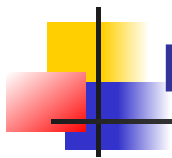
- **Méthode** = (fonction ou procédure) = une **collection d'instructions** qui effectue une **tâche complexe (utile)**
Une **méthode** est **identifiée** par son nom
- **Classe** = un **conteneur de méthodes**
Les méthodes qui servent un **but similaire** sont stockées dans **la même classe**
Une **classe** est **identifiée** par son nom





API de la Librairie

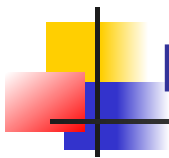
- Utiliser les méthodes prédéfinies pour votre programme.
("reinvent the wheel", **NON**!)
- Exemple:
 - Utiliser les méthodes de calcul mathématiques prédéfinies.
 - Utiliser les structures de données usuelles prédéfinies -- listes, piles, files, arbres.
(gain en temps, en performance, ...)
- Les méthodes sont définies dans des **classes** (types de données complexes).
- Les **classes** sont des enveloppes qui regroupent plusieurs méthodes relatives à un sujet donné. Classe **Math**, **String**, **Integer**, etc.



La Classe Math

7

- Quelques méthodes usuelles de la classe Math :
 - `public static int abs(int num)`
 - Retourne la valeur absolue de `num`.
 - `public static double abs(double num)`
 - Retourne la valeur absolue de `num`.
 - `public static int max(int x, int y)`
 - Retourne la plus grande valeur de `x` et `y`.
 - `public static double max(double x, double y)`
 - Retourne la plus grande valeur de `x` et `y`.



La Classe Math

8

- Quelques méthodes usuelles de la classe Math (suite):
 - `public static int min(int x, int y)`
 - Retourne la plus petite valeur de `x` et `y`.
 - `public static double min(double x, double y)`
 - Retourne la plus petite valeur de `x` et `y`.
 - *`public static double pow(double num, double power)`*
 - Retourne `num` élevée à la puissance `power`.
 - *`public static double random()`*
 - Retourne une valeur uniformément distribuée entre 0.0 et 1.0, mais n'incluant pas 1.0.
 - `public static double sqrt(double num)`
 - Retourne la racine carré `num`.



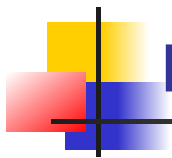
La Classe Math

- Les méthodes de la classe `Math` sont des méthodes "*static*" (méthodes de classe).
- Appel : Le nom de la classe `Math` suivi du point (`.`) suivi du nom de la méthode.

```
int position1 = 15, position2 = 18;  
int distanceApart = Math.abs(position1 - position2);
```

Appel de méthode: `Math.nomMéthode (Liste des arguments)`

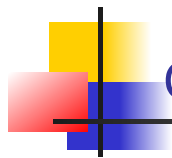
```
discriminant = Math.pow(b, 2) - (4 * a * c);  
X1 = ((-1 * b) + Math.sqrt(discriminant)) /  
      (2 * a);  
X2 = ((-1 * b) - Math.sqrt(discriminant)) /  
      (2 * a);
```

La Classe `Math`

11

- Les classes contiennent aussi des *variables*, des *constantes*, etc.
- La classe `Math` contient la constante mathématique de nom `PI`:
 - `Pi`, π = perimeter/diameter = 3.14159265358979323846
 - Type `double`
 - Constante (valeur fixe).
changement de valeur ==> erreur de compilation.
 - *variable de classe*, utilisation `(Math.PI)`.



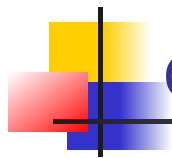
Classes enveloppes Pour Types Primitifs

- Une classe enveloppe est une classe relative à un type primitif lui ajoutant plus de fonctionnalités.
- Classes enveloppes de quelques types primitifs de Java:

<u>Classe enveloppe</u>	<u>Type Primitif</u>
Integer	int
Long	long
Float	float
Double	double
Character	char

- Exemple: Fonctions de conversions (String <--> Nombres) aux interfaces graphiques.

<u>Classe enveloppe</u>	<u>string → number</u>	<u>number → string</u>
Integer	Integer.parseInt(<string>)	Integer.toString(<#>)
Long	Long.parseLong(<string>)	Long.toString(<#>)
Float	Float.parseFloat(<string>)	Float.toString(<#>)
Double	Double.parseDouble(<string>)	Double.toString(<#>)



Classes enveloppes pour Types primitifs

String => valeur

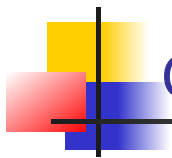
- La classe enveloppe de chaque *type de données* primitif a une méthode `parseType()` pour '*parser*' une représentation en "string" & retourne la valeur littérale.

```
Integer.parseInt("42")           => 42
```

```
Boolean.parseBoolean("true")     => true
```

```
Double.parseDouble("2.71")       => 2.71
```

```
//...
```



Classes enveloppes pour Types primitifs

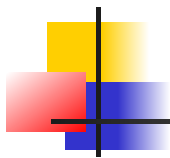
- Exemples de Conversions - strings aux nombres:

```
String anneeStr = "2002";  
String noteStr = "13.5";  
int year = Integer.parseInt(anneeStr);  
double note = Double.parseDouble(noteStr);
```

- Rappel - pour convertir une string à un type numérique, utiliser **XXX.parseXXX** tel que **XXX** est le nom de la classe enveloppe du type numérique utilisé.

- Exemples de Conversions - nombres aux strings:

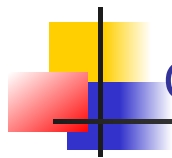
```
int annee = 2002;  
double score = 78.5;  
String yearStr = Integer.toString(annee);  
String noteStr = Double.toString(score);
```



Primitifs & enveloppes

- Java a une **Classe *enveloppe*** pour chacun des huit types de données primitives :

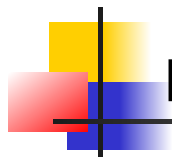
Type Primitif	Classe enveloppe	Type Primitif	Classe enveloppe
boolean	Boolean	float	Float
byte	Byte	int	Integer
char	Character	long	Long
double	Double	short	Short



Classes enveloppes pour Types primitifs

- Pour trouver la plus petite et la plus grande valeur possible pour un type primitif, utiliser la classe enveloppe du type et accéder les valeurs constantes nomées `MAX_VALUE` et `MIN_VALUE` de la classe enveloppe. Par exemple:

```
Integer.MAX_VALUE : 2147483647  
Double.MAX_VALUE : 1.7976931348623157E308  
Float.MIN_VALUE   : 3.4028235E38
```



Rappel: classes et méthodes

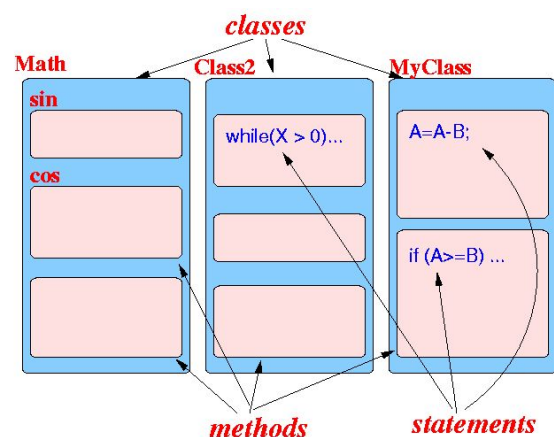
- **Méthode** = une **collection d'instructions** qui effectue une **tache complexe (utile)**

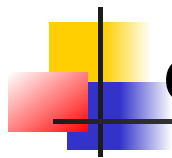
Une **méthode** est **identifiée** par son nom

- **Classe** = un **conteneur de méthodes**

Les méthodes that serves a **similar purpose** sont stored in **the same class**

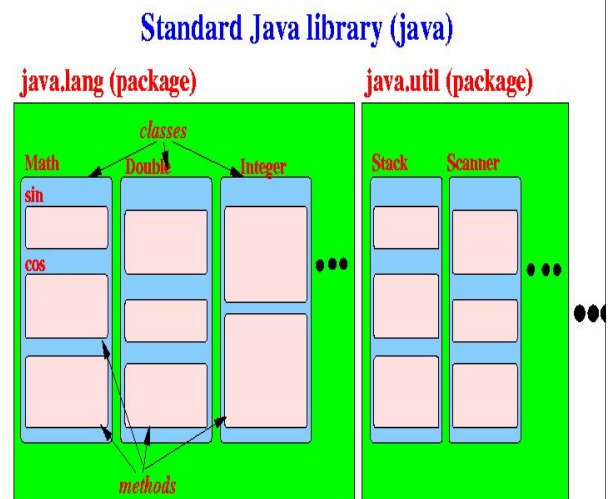
Une **classe** est **identifiée** par son nom

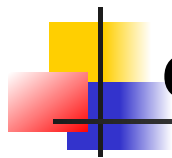




Organization de la librairie Java

- La *librairie Java* Standard consiste en un nombre de *packages*
Chaque *package* consiste d'un nombre de *classes* (qui fournissent une fonctionnalité similaire)
- La *librairie Java* standard est appelée *java*
- Un *package* nommé *xxx* dans la *librairie Java* standard est nommé *java.xxx*





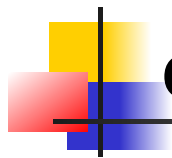
Organization de la librairie Java (suite.)

- Quelques packages couramment utilisées :

- **java.lang**: Fournit des classes qui sont fondamentales au design du langage de programmation Java.

Site web officiel :

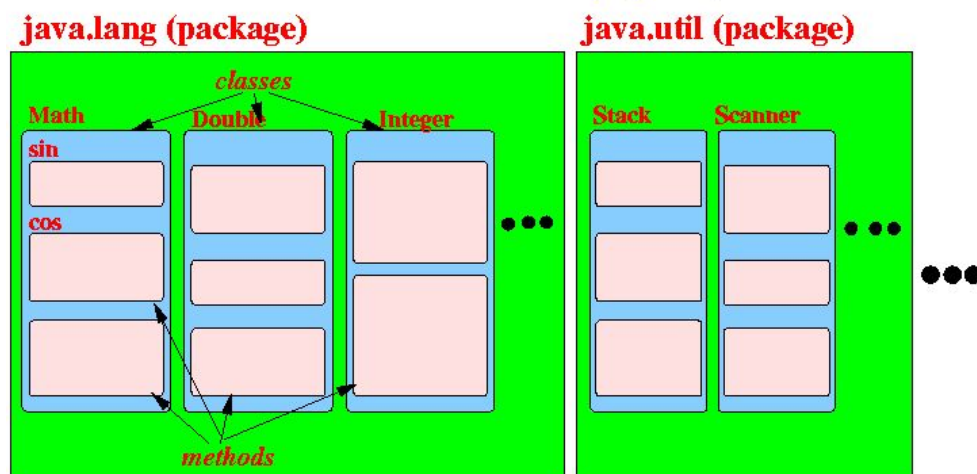
<http://download.oracle.com/javase/1.4.2/docs/api/java/lang/package-summary.html>

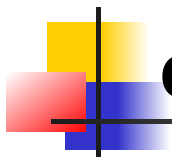


Organization de la librairie Java (suite.)

- Représentation schématique de la Librairie standard Java:

Standard Java library (java)



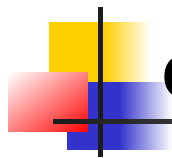


Organization de la librairie Java (suite.)

- Une **classe** nommée **yyy** dans le **package java.xxx** est nommée **java.xxx.yyy**

Exemple:

- La **classe Math** dans le package **java.lang** est connu en tant que **java.lang.Math**
- The **class Double** dans le package **java.lang** est connu en tant que **java.lang.Double**
- The **class Stack** dans le package **java.util** est connu en tant que **java.util.Stack**
- The **class Scanner** dans le package **java.util** est connu en tant que **java.util.Scanner**



Organization de la librairie Java (suite.)

- Note:

- C'est une convention Java que le **nom** d'une classe Java commence par une **lettre majuscule**
- C'est *aussi* une convention Java que le **nom** d'une méthode commence par une **lettre miniscule**



Utilisation des méthodes de la librairie standard de java: importation d'une classe

- Règle d'usage:

- Si un programme Java veut utiliser une **méthode** dans la librairie Java, le programme Java **doit en premier importer** la **classe** qui le contient

La **clause import** doit être **la première instruction** dans un programme (avant la définition de toute classe)

- Syntaxe pour **importer une classe** de la librairie Java :

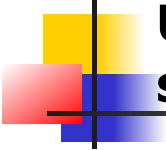
```
import nomClasse ;
```



Utilisation des méthodes de la librairie standard de java: importation d'une classe

- Exemples:

```
import java.lang.Math;  
import java.lang.Double;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
// Après la clause import , on peut écrire le code du  
//programme (la définition de la classe)  
  
// Ce programme peut maintenant utiliser toutes les  
//méthodes définies dans les classes Math, Double,  
//ArrayList et Scanner
```



Utilisation des méthodes de la librairie standard de java: importation d'une classe

```
class MonProgramme
{
    public static void main(String[] args)
    {
        double a;
        a = Math.sqrt(2.0);
        System.out.println(a);
    }
}
```



Importation de *toutes les classes* dans un package

- Quelques programmes Java complexes peuvent utiliser plusieurs méthodes différentes contenues dans plusieurs classes différentes dans le même package

Ce serait *dououreux* d'écrire une *longue liste de clauses "import"*

Exemple:

```
import java.lang.Math;  
import java.lang.Double;  
import java.lang.Integer; ...
```




Importation de *toutes les classes* dans un package (suite)

- Il existe un raccourci pour importer *toutes les classes* contenues dans un package:

```
import java.lang.* ; // importer toutes les classes dans  
//le package java.lang  
  
import java.util.* ; // importer toutes les classes dans le  
//package java.util
```



Méthodes Fréquemment utilisées : Le package *java.lang*

- Selon la Règle d'usage:

• Si un programme Java veut utiliser une **méthode** dans la librairie Java, le programme Java **doit en premier importer** la **classe** qui le contient

On doit **importer** **java.lang.Math** si on veut utiliser la méthode **Math.sqrt()**



Méthodes Fréquemment utilisées : Le package *java.lang*

On aurait du écrire:

```
import java.lang.Math; // On DOIT importer cette classe pour utiliser
//Math.sqrt
public class Abc
{
    double a, b, c, x1, x2; // Définit 5 variables
    a = 1.0;
    b = 0.0;
    c = -4.0;
```



Méthodes Fréquemment utilisées : Le package *java.lang*

```
x1 = ( -b - Math.sqrt( b*b - 4*a*c ) ) / (2*a);  
x2 = ( -b + Math.sqrt( b*b - 4*a*c ) ) / (2*a);  
System.out.print("a = "); System.out.println(a);  
System.out.print("b = "); System.out.println(b);  
System.out.print("c = "); System.out.println(c);  
System.out.print("x1 = ");  
System.out.println(x1);  
System.out.print("x2 = ");  
System.out.println(x2); }
```



Méthodes Fréquemment utilisées : Le package *java.lang*

- **Mais....** Parce que:

- Le package `java.lang` contient des classes qui sont **fondamentales** au **design** du langage de programmation Java.

Toutes les classes dans le package `java.lang` sont *automatiquement incluses* dans chaque programme Java (le compilateur Java est programmé à le faire)

C'est **pourquoi** **nous n'avons pas besoin** d'**importer** `java.lang.Math` dans notre program.



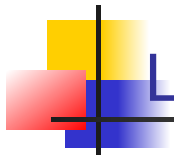
Utilisation des méthodes de la librairie standard de java

- Règle d'usage:

- Si un programme Java veut utiliser une **méthode** dans la librairie Java, le programme Java **doit en premier importer "import"** la **classe** qui le contient
- *Toutes* les **classes** dans le package **java.lang** ont déjà été importés dans un **programme Java** (On peut utiliser les méthodes dans ces classes **sans** la **clause import**)

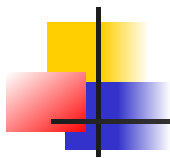
Les chaînes de caractères

La classe String



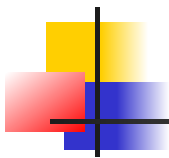
La classe String

- Un objet de la classe String représente une chaîne de caractères.
- String a deux opérateurs en outre, + and += (utilisés pour la concaténation).



Les littéraux (Strings)

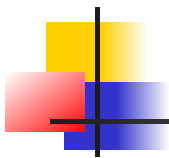
- Un "Littéral" string = objet anonyme de la classe String qui est une "chaîne constante" défini comme texte en double quotes.
- Création d'un Littéral string : pas besoin ils sont "juste là."



Les littéraux (Strings--suite)

- peuvent être affectés aux variables String.
- peuvent être passées aux méthodes en tant que paramètres.
- ont des méthodes qu'on peut appeler:

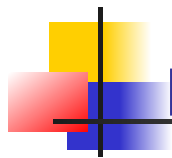
```
String nomFichier = "fil.dat";  
button = new JButton("écran suivant");  
if ("Start".equals(cmd)) ...
```



Les littéraux (Strings--suite)

- le texte de la string peut inclure les caractères "d'échappement" .
- Exemple:
 - \\ pour \
 - \n retour à la ligne

```
String s1 = "POO en Java";  
String s2 = "C:\\jdk1.4\\docs";  
String s3 = "Hello\n";
```

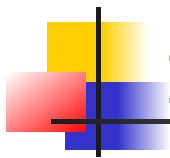


Exemples de "Littéral" String

```
//affecter un littéral à une variable String  
String nom = "Rachid";
```

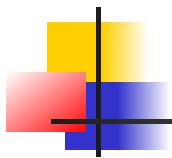
```
//Appel d'une méthode sur un littéral String  
char premierInitial = "Rachid".charAt(0);
```

```
//Appel d'une méthode sur une variable String  
char premierInitial = name.charAt(0);
```



Immuabilité

- Une fois créée, une string ne peut pas être modifiée: aucune de ses méthodes ne peut changer la string.
- De tels objets sont appelés *immuable*.
- Les objets immuables sont *pratiques* parce que plusieurs références de même type peuvent faire référence au même objet en toute sécurité: aucun danger de changer un objet à une référence sans que les autres ne le sachent.



Strings Vides

- Une string vide n'a aucun caractère; sa longueur est 0.

```
String s1 = "";  
String s2 = new String();
```

Strings vides

- Ne pas confondre avec une string non initialisée :

```
String msgErreur;
```

msgErreur
est **null**



Méthodes — length, charAt

`int length ();`

- Retourne le nombre de caractères dans la string

`char charAt (k);`

- Retourne le caractère à la k-ème position

Les positions des Caractères dans mes strings commencent de l'indice 0

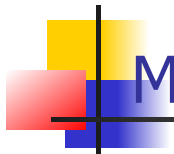
Retourne:

`"Oracle".length();`

6

`"Hind".charAt (2);`

'n'



Méthodes — Égalité

boolean b = mot1.**equals**(mot2);
retourne **true** si la string **mot1** est égale à **mot2**

boolean b = word1.**equalsIgnoreCase**(word2);
retourne **true** si la string **mot1** est égale à **mot2**,
ignorant la casse

```
b = "Java".equals("Java");//true  
b = "Java".equals("java");//false  
b = "Java".equalsIgnoreCase("java");//true
```

```
if(langage.equalsIgnoreCase("java"))  
    System.out.println("POO en " + langage);
```




Méthodes — Comparisons

```
int diff = mot1.compareTo(mot2);  
retourne la "différence" mot1 - mot2
```

```
int diff = mot1.compareToIgnoreCase(mot2);  
retourne la "différence" mot1 - mot2, ne tenant pas  
compte de la casse
```

Habituellement les programmeurs ne se préoccupent pas de la valeur numérique de la "différence" **mot1 - mot2**, mais juste de son signe, si la différence est négative (mot1 *précède* mot2), zéro (mot1 et mot2 sont égaux) ou positive (mot1 *devance* mot2). Souvent utilisée dans les instructions conditionnelles.

```
if(mot1.compareTo(mot2) > 0){  
    //mot1 vient après mot2...  
}
```

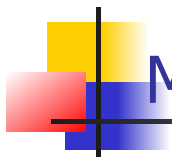


Comparison (Examples)

```
//différences négatives  
diff = "c".compareTo("java");//c avant j  
diff = "java".compareTo("java EE");//java est  
//plus courte que java EE
```

```
//différences zéro  
diff = "java".compareTo("java");//equal  
diff = "poo".compareToIgnoreCase("POO");//equal
```

```
//différences positives  
diff = "info".compareTo("INFO");//i après I  
diff = "FAC".compareTo("BAC");//F après B  
diff = "Java 9".compareTo("Java");// Java 9 est  
plus longue
```



Méthodes — Comparaisons

`boolean b = s1.equals(s2);`
retourne **true** si la string **s1** est égale à **s2**

`boolean b = s1.equalsIgnoreCase(s2);`
retourne **true** si la string **s1** est la même que **s2**, ignorant la casse

`int diff = s1.compareTo(s2);`
retourne la "différence" **s1** - **s2**

`int diff = s1.compareToIgnoreCase(s2);`
retourne la "différence" **s1** - **s2**, ignorant la casse



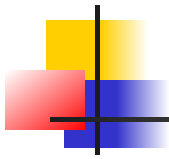
Méthodes — Concaténation

`String resultat = s1 + s2;`
concaténe s1 and s2

`String resultat = s1.concat(s2);`
meme chose que s1 + s2

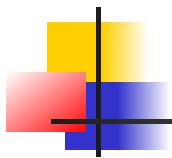
`resultat += s3;`
concaténe s3 à result

`resultat += num;`
convertit **num** à **String** et la concaténe à resultat



Traieiment de fichiers (texte)

La classe File



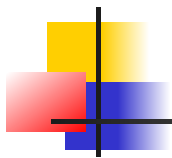
Entrée/Sortie (E/S)

```
import java.io.*;
```

- Créer un objet `File` pour obtenir des informations sur un fichier sur le disque dur. (Cela ne crée pas un nouveau fichier sur le disque dur.)

```
File f = new File("example.txt");  
if (f.exists() && f.length() > 1000) {  
    f.delete();  
}
```

Method name	Description
<code>canRead()</code>	retourne si le fichier peut etre lu
<code>delete()</code>	supprime le fichier du disque
<code>exists()</code>	si le fichier existe sur le disque
<code>getName()</code>	retourne le nom du fichier
<code>length()</code>	retourne le nombre d'octets dans le fichier
<code>renameTo(<i>file</i>)</code>	change le nom du fichier



Lecture des fichiers

- Pour lire un fichier, passer un `File` lors de la construction d'un `Scanner`.

```
Scanner nom = new Scanner(new File("nom fichier")) ;
```

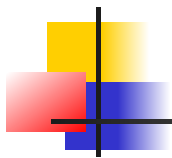
- Example: (en 2 temps)

```
File file = new File("données.txt") ;
```

```
Scanner input = new Scanner(file) ;
```

- ou (plus court):

```
Scanner input = new Scanner(new  
File("données.txt")) ;
```



Chemins de fichiers

- **chemin absolu:**

`C:\Users\az\Documents\TDtp1\Fic`

- **chemin relatif:**

`notes.txt`

`Module/notes.txt`

- **Supposé être relatif au répertoire courant:**

```
Scanner input = new Scanner(new File("data/readme.txt"));
```

Si notre programme est dans

`H:/rep` ,

Scanner cherchera dans

`H:/rep/data/readme.txt`



Erreurs Compilateur avec fichiers

```
import java.io.*;      // pour File
import java.util.*;    // pour Scanner

public class ReadFile {
    public static void main(String[] args) {
        Scanner input = new Scanner(new File("data.txt"));
        String text = input.next();
        System.out.println(text);
    }
}
```

- Le programme ne parvient pas à compiler avec l'erreur suivante:

```
ReadFile.java:6: unreported exception
    java.io.FileNotFoundException;
must be caught or declared to be thrown
        Scanner input = new Scanner(new File("data.txt"));
                                ^
```



La clause `throws`

- La clause ***throws*** : `throws FileNotFoundException`

- Ajoutée à l'entête de toute méthode qui traite les fichiers.

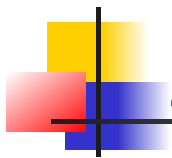
- Syntaxe:

```
public static type name(params) throws typeExcpt {
```

- Exemple: *clause **throws*** ajoutée à l'entête de la méthode main

```
public class ReadFile {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        ...  
        ...  
    }  
}
```

- ***throws Exception*** est aussi valable. Nous l'utiliserons aux travaux dirigés et aux travaux pratiques.



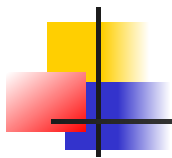
Jetons d'entrée

- **token:** Une unité d'entrée de l'utilisateur, séparés par des espaces.
 - Un `Scanner` divise le contenu d'un fichier en jetons (token).
- Si un fichier d'entrée contient ce qui suit:

23	3.14
"Zayd Reda"	

Le `Scanner` peut interpréter les tokens en tant que types comme suit:

<u>Token</u>	<u>Type(s)</u>
23	int, double, String
3.14	double, String
"Zayd	String
Reda"	String



Fichiers et curseur d'entr  e

- Considerons un fichier m  teo.txt qui contient ce texte:

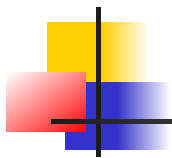
```
16.2    23.5
      19.1 7.4   22.8

18.5    -1.8 14.9
```

- Un `Scanner` voit tout le contenu comme un flux de caract  res:

```
16.2    23.5\n19.1 7.4   22.8\n\n18.5    -1.8 14.9\n^
```

- **curseur d'entr  e:** La position courante du `Scanner`.



Consommation des jetons

- **consommation des entrées:** Lire le contenu et avancer le curseur.
 - Appelant `nextInt` etc. déplace le curseur au-delà du token (jeton) courant.

```
16.2    23.5\n19.1 7.4    22.8\n\n18.5    -1.8 14.9\n
```

^

```
double d = input.nextDouble();    // 16.2
```

```
16.2    23.5\n19.1 7.4    22.8\n\n18.5    -1.8 14.9\n
```

^

```
String s = input.next();           // "23.5"
```

```
16.2    23.5\n19.1 7.4    22.8\n\n18.5    -1.8 14.9\n
```

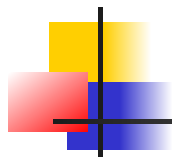
^



Tests du `Scanner` pour entrées valides

Méthode	Déscription
<code>hasNext()</code>	retourne <code>true</code> s'il existe un token suivant
<code>hasNextInt()</code>	retourne <code>true</code> s'il existe un token suivant et peut être lu en tant que <code>int</code>
<code>hasNextDouble()</code>	retourne <code>true</code> s'il existe un token suivant et peut être lu en tant que <code>double</code>

- Ces méthodes du `Scanner` ne consomment pas l'entrée;
Ils donnent l'information à propos du token suivant (le type du token suivant) .
 - Utile pour savoir ce qu'est l'entrée (input) à venir, et pour éviter les crashes.
 - Ces procédés peuvent être aussi bien utilisés avec un `Scanner` de console.
 - Lorsqu'elles sont appelées sur la console, elles bloquent parfois (en attente d'une entrée).



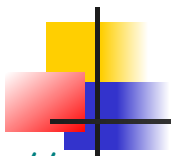
Fichier d'entrée (question)

- Rappelons le fichier d'entrée `méteo.txt`:

```
16.2    23.5
      19.1 7.4   22.8

18.5    -1.8 14.9
```

- Ecrire un programme qui compte le nombre de prises de température effectuées.

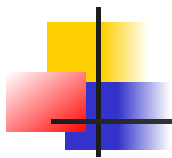


Fichier d'entrée (réponse 2)

// Compte le nombre de fois (de prises de températures) que les températures ont été enregistrées dans le fichier d'entrée.

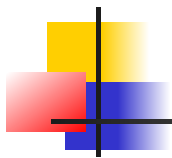
```
import java.io.*;    // pour File
import java.util.*;  // pour Scanner

public class Temperatures {
    public static void main(String[] args)
        throws FileNotFoundException {
        Scanner input = new Scanner(new File("météo.txt"));
        int c=0;
        while(in.hasNext())
            {in.next();c++;}
        system.out.println(c);
    }
}
```

Fichier d'entrée (question 3)

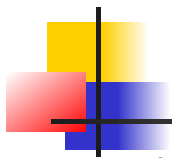
- Calculer le nombre de jours de prises de température.
- Les prises de températures d'un jour sont inscrites sur la même ligne.
- La fin d'une ligne est caractérisée par `\n`.
- Nous ne disposons d'aucun moyen pour savoir que nous avons consommé une ligne.



Line-based Scanners

Méthode	Déscription
<code>nextLine()</code>	retourne la ligne d'entrée suivante entièrement (du curseur au <code>\n</code>)
<code>hasNextLine()</code>	retourne <code>true</code> s'il existe plus de lignes d'entrées à lire (toujours <code>true</code> pour les entrées de la console)

```
Scanner input = new Scanner(new File("file name"));  
while (input.hasNextLine()) {  
    String line = input.nextLine();  
    traiter cette ligne;  
}
```



Consuming lines of input

```
23      3.14 Zayd Reda      "Hello" world
          45.2              19
```

- Le Scanner lit les lignes comme suit:

```
23\t3.14 Zayd Reda\t"Hello" world\n\t\t45.2  19\n^
```

- `String line = input.nextLine();`
23\t3.14 Zayd Reda\t"Hello" world\n\t\t45.2 19\n^

- `String line2 = input.nextLine();`
23\t3.14 Zayd Reda\t"Hello" world\n\t\t45.2 19\n^

- Chaque caractère `\n` est consommé mais non retourné.

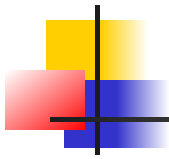


Fichier d'entrée (réponse 2)

```
// Compte le nombre de jour que les températures ont été  
enregistrées dans le fichier d'entrée (une ligne de prises de  
températures = 1 jour).
```

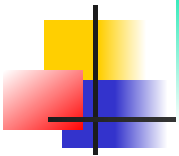
```
import java.io.*;    // pour File  
import java.util.*;  // pour Scanner
```

```
public class Temperatures {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        Scanner input = new Scanner(new File("météo.txt"));  
        int c=0;  
        while(in.hasNextLine())  
            {in.nextLine(); c++;}  
        system.out.println(c);  
    }  
}
```



Sortie des fichiers

La classe Printwriter



Utilisation de la classe *PrintWriter* pour écrire des données dans un fichier

La classe *PrintWriter* peut être utilisée pour l'écriture et la sauvegarde de données dans un fichier.

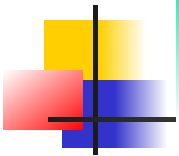
Exemple:

L'instruction suivante **crée**, **ouvre**, et **lie** le fichier référencé par *file* avec la variable *PrintWriter* nommée *outputFile*.

```
File file = new File("données.txt");  
PrintWriter outputFile = new PrintWriter(file);
```

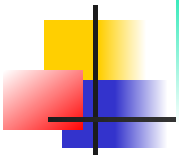
Passer la référence de
l'objet *file* au constructeur
de la classe *PrintWriter*.

**Warning: si le fichier existe déjà, il
sera effacé et remplacé par un
nouveau fichier.**



Utilisation de la classe *PrintWriter* pour écrire des données dans un fichier

- Après ouverture du fichier par un objet de la classe *PrintWriter*, les méthodes ***print***, ***println***, et/ou *printf* seront utilisées pour écrire les données dans ce fichier.
- ***print***, ***println*** seront utilisées de la même façon qu'elles le sont avec *System.out* pour l'affichage des données sur la fenêtre de la console.

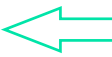


Utilisation de la classe *PrintWriter* pour écrire des données dans un fichier

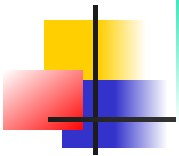
- Après avoir utilisé un fichier on doit le fermer utilisant la méthode *close*.

Exemple:

```
outputFile.close( );
```



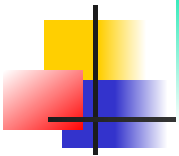
La méthode *close*
de l'objet
PrintWriter.



Utilisation de la classe *PrintWriter* pour écrire des données dans un fichier

Exemple:

```
public static void ecrire(String[] t,File f) throws IOException{  
    PrintWriter out=new PrintWriter(f);  
    for(String el:t) out.println(el);  
    out.close();  
}
```

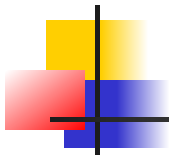


Utilisation de la classe *PrintWriter* pour écrire des données dans un fichier

Pour écrire les données dans un fichier texte:

1. Inclure l'instruction `import java.io.*;` avant les définitions des classe du fichier source.
2. Mettre la clause `throws IOException` dans l' entete de toute méthode qui crée un objet *PrintWriter* ou appelle une méthode qui crée un objet *PrintWriter*.

➤ `throws Exception` est aussi valable. Nous l'utiliserons aux travaux dirigés et aux travaux pratiques.



Programmation orientée objet

Librairie Standard, Classes Enveloppes, chaines de caractères et fichiers

Supports de présentation

<http://www.fatih.edu.tr/~moktay/2009/spring/ceng104/The.String.Class.ppt>

<http://www.comp.nus.edu.sg/~cs1101x/JohnDean/>

<http://www.buildingjavaprograms.com/slides/>

<http://www.utdallas.edu/~kkhan/CS1336/>