

BURKINA FASO

UNITÉ-PROGRÈS-JUSTICE

MASTER2: RÉSEAUX INFORMATIQUES ET MULTIMÉDIA

PROGRAMMATION MOBILE

Introduction au langage Java

COMPAORE MOCTAR

20 September 2022

LES ÉTAPES

- Écrire un programme dans un langage (e.g. Java)
- Compiler le programme
 - Traduire le programme dans un langage de bas niveau (machine)
 - [éventuellement optimisation]
 - Produire un programme (code) exécutable
- Exécution
 - Charger le programme en mémoire (typiquement en tapant le nom du programme exécutable)
 - Exécution

SOURCES

- IFT6800 – E 2008
- Pierre Poulin

TERMES

- Programme source, code source
 - Programme écrit dans un langage
- Code machine, code exécutable
 - Programme dans un langage de machine, directement exécutable par la machine
- Compilation (compilateur)
 - Traduire un code source en code exécutable
- Interpréteur
 - Certains langages n'ont pas besoin d'être traduit en code machine
 - La machine effectue la traduction sur la volée (*on the fly*), instruction par instruction, et l'exécute
 - E.g. Prolog, JavaScript

PROGRAMMATION

- Syntaxe d'un langage
 - Comment formuler une instruction correcte (grammaire)
- Sémantique
 - Ce que l'instruction réalise
- Erreur
 - de compilation: typiquement reliée à la syntaxe
 - d'exécution: sémantique (souvent plus difficile à détecter et corriger)

JAVA

- Langage orienté objet
 - Notions de classes, héritage, ...
- Beaucoup d'outils disponibles (packages)
 - JDK (*Java Development Kit*)
- Historique
 - Sun Microsystems
 - 1991: conception d'un langage indépendant du hardware
 - 1994: browser de HotJava, applets
 - 1996: Microsoft et Netscape commencent à soutenir
 - 1998: l'édition Java 2: plus stable, énorme librairie

INSTALLATION DE ANDROID STUDIO

- ❖ Installer le JDK (Java SE 8u231 ou supérieur)
 - ✓ Fournit sur support amovible
 - ✓ Lien :
www.oracle.com/technetwork/java/javase/downloads/index.html 2.
- ❖ Installer Android Studio
 - ✓ Fournit sur support amovible
 - ✓ Lien : <https://developer.android.com/studio/>

JAVA

- Compiler un programme en *Byte Code*
 - *Byte code*: indépendant de la machine
 - Interprété par la machine
- *javac programme.java*
 - Génère programme.class
- *java programme*
 - Lance le programme

ÉCRIRE UN PROGRAMME

```
public class Bonjour
{
    public static void main(String[] args)
    {
        // afficher une salutation
        System.out.println("Bonjour , le monde!");
    }
}
```

Nom de la classe

Une méthode

commentaire

Une instruction

- Stocker ce programme dans le fichier `Bonjour.java`

LANCER UN PROGRAMME

- Compilation
 - *Javac Bonjour.java*
 - *Ceci génère Bonjour.class*
- Lancer l'exécution
 - *java Bonjour*
- Résultat de l'exécution
 - *Bonjour, le monde!*

ÉLÉMENTS DE BASE DANS UN PROGRAMME

- mots réservés: `public class static void`
- identificateurs: `args Bonjour main String System out println`
 - `main String System out println`: ont une fonction prédéfinie
- littéral: "Bonjour, le monde!"
- ponctuation: { accolade } [crochet] (parenthèse)
- Commentaires
 - `// note importante pour comprendre cette partie du code`
 - `/* ... commentaires sur plusieurs lignes`
`*/`

CLASSE

- Un programme en Java est défini comme une classe
- Dans une classe:
 - attributs, méthodes
- L'en-tête de la classe
`public class NomDeClasse`
 - *public* = tout le monde peut utiliser cette classe
 - *class* = unité de base des programmes OO
- Une classe par fichier
- La classe NomDeClasse doit être dans le fichier NomDeClasse.java
- Si plus d'une classe dans un fichier *.java*, *javac* génère des fichiers *.class* séparés pour chaque classe

CLASSE

- Le corps

```
{
```

```
...
```

```
}
```

- Contient les attributs et les méthodes

- Attributs: pour stocker les informations de la classe
- Méthodes: pour définir ses comportements, ses traitements, ...

- Conventions et habitudes

- nom de classe: **NomDeClasse**
- indentation de { }
- indentation de ...

- Les indentations correctes ne seront pas toujours suivies dans ces notes pour des raisons de contraintes d'espace par PowerPoint...

MÉTHODE: EN-TÊTE

- L'en-tête:

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

- *main*: nom de méthode
- *void*: aucune sortie (ne retourne rien)
- *String[] args*: le paramètre (entrée)
 - *String[]*: le type du paramètre
 - *args*: le nom du paramètre

- Conventions

- nomDeParametre
- nomDeMethode
- nomDAttributs
- nomDObjet

MÉTHODE: CORPS

- Le corps:

```
{  
    // afficher une salutation  
    System.out.println("Bonjour, le monde!");  
}
```

- contient une séquence d'instructions, délimitée par { }
 - // afficher une salutation : commentaire
 - System.out.println("Bonjour, le monde!"): appel de méthode
- les instructions sont terminées par le caractère ;

MÉTHODE: CORPS

- En général:

`nomDObjet.nomDeMethode(<liste des paramètres>)`

- `System.out`: l'objet qui représente le terminal (l'écran)
- `println`: la méthode qui imprime son paramètre (+ une fin de ligne) sur un *stream* (écran)

`System.out.println("Bonjour, le monde!");`

- "Hello, World!": le paramètre de `println`
- La méthode `main`
 - "java Hello" exécute la méthode *main* dans la classe *Hello*
 - *main* est la méthode exécutée automatiquement à l'invocation du programme (avec le nom de la classe) qui la contient

VARIABLE

- Variable: contient une valeur
 - Nom de variable
 - Valeur contenue dans la variable
 - Type de valeur contenue
 - *int*: entier, *long*: entier avec plus de capacité
 - *Integer*: classe entier, avec des méthodes
 - *float*: nombre réel avec point flottant, *double*: double précision
 - *String*: chaîne de caractères (« Bonjour, le monde!»)
 - *char*: un caractère en Unicode ('a', '\$', 'é', ...)
 - *boolean*: true/false
- Définition générale

Type nomDeVariable;

Exemple: *int* age;

Type: int

Nom: age

0



MODIFIER LA VALEUR

- Affecter une valeur à une variable
- E.g. `age = 25;`

Type: int

Nom: age

25



- Erreur si: `age = "vingt cinq";`
 - Type de valeur incompatible avec la variable

CONDITION ET TEST

- Une condition correspond à vrai ou faux
- E.g. (`age < 50`)
- Tester une condition:

if condition A; else B;

- si *condition* est satisfaite, alors on fait *A*;
- sinon, on fait *B*
- E.g. `if (age < 65)`

`System.out.println("jeune");`

`else`

`System.out.println("vieux");`

TESTS

- Pour les valeurs primitives (int, double, ...)
 - $x == y$: x et y ont la même valeur?
 - $x > y$, $x \geq y$, $x \neq y$, ...
 - Attention: **(== != =)**
- Pour les références à un objet
 - $x == y$: x et y pointent vers le même objet?
 - `x.compareTo(y)`: retourne -1, 0 ou 1 selon l'ordre entre le contenu des objets référés par x et y

UN EXEMPLE DE TEST

```
public class Salutation
{
    public static void main(String[] args)
    {
        int age;
        age = Integer.parseInt(args[0]);
        // afficher une salutation selon l'age
        System.out.print("Salut, le ");
        if (age < 65)
            System.out.println("jeune!");
        else
            System.out.println("vieux!");
    }
}
```

args[0]: premier argument
après le nom

Integer.parseInt(args[0]): reconnaître et
transmettre la valeur
entière qu'il
représente

print: sans retour à la ligne
println: avec retour à la ligne

Utilisation:

java Salutation 20

Salut le jeune!

java Salutation 70

Salut le vieux!

// ici, args[0] = "20"

BOUCLE

Attention:

un ; après le for(), itère sur la condition, et 'somme' ne sera incrémentée qu'une seule fois

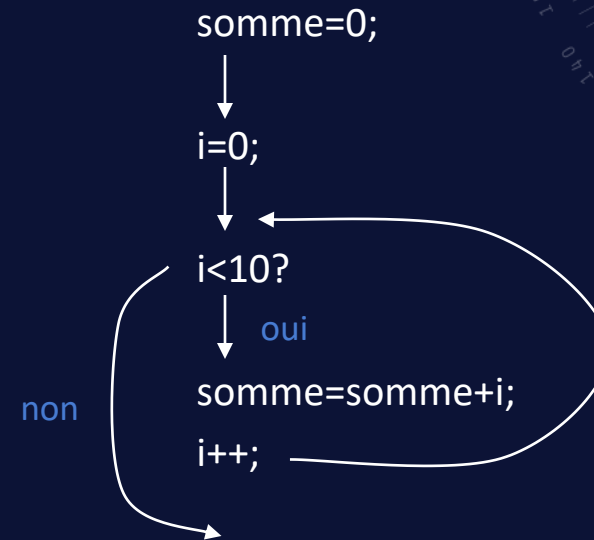
- Pour traiter beaucoup de données en série
- Schémas
 - Boucle *for*
`int somme = 0;`
`for (int i = 0; i<10; i++) somme = somme + i;`
 - Boucle *while*
`int somme = 0;`
`int i = 0;`
`while (i<10) { somme = somme + i;`
`i++;`
`}`
- Que font ces deux boucles?

i:		0,	1,	2,	3,	4,	5,	6,	7,	8,	9,	10			
		↓	↓									↓			
somme:	0	→	0,	→	1,	3,	6,	10,	15,	21,	28,	36,	45,	→	sortie

Attention:

'i' n'est déclarée ici qu'à l'intérieur de la boucle for

Schéma d'exécution



BOUCLE

do A while (condition)

- Faire A au moins une fois
- Tester la condition pour savoir s'il faut refaire A

```
int somme = 0;  
int i = 15;  
while (i<10) { somme = somme + i;  
               i++;  
            }
```

somme = 0

```
int somme = 0;  
int i = 15;  
do { somme = somme + i;  
     i++;  
    }  
while (i<10)
```

somme = 15

Schéma d'exécution

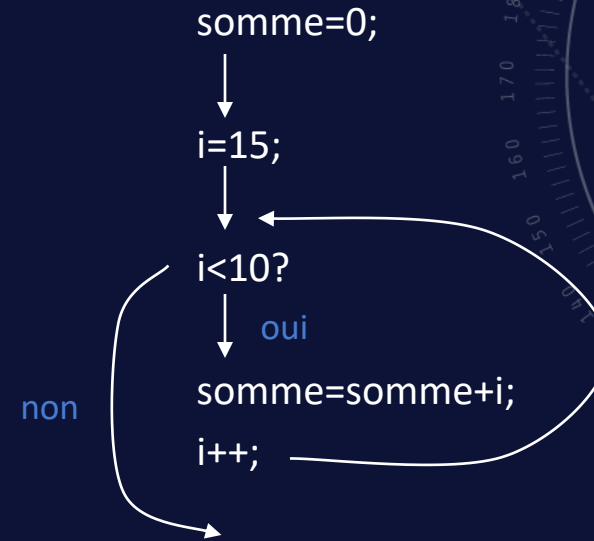
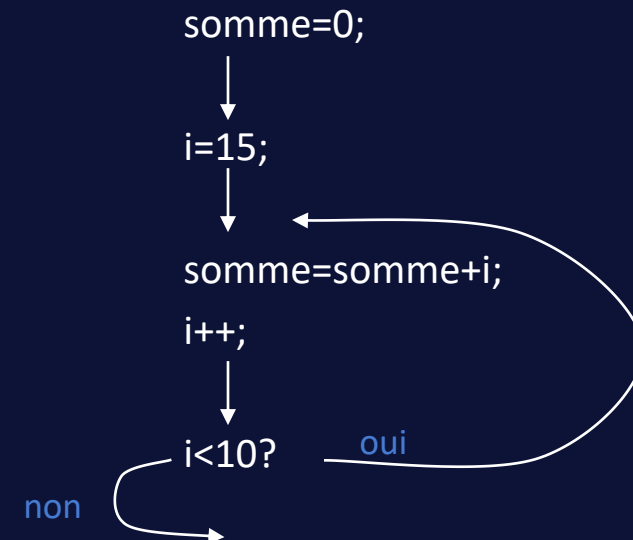


Schéma d'exécution

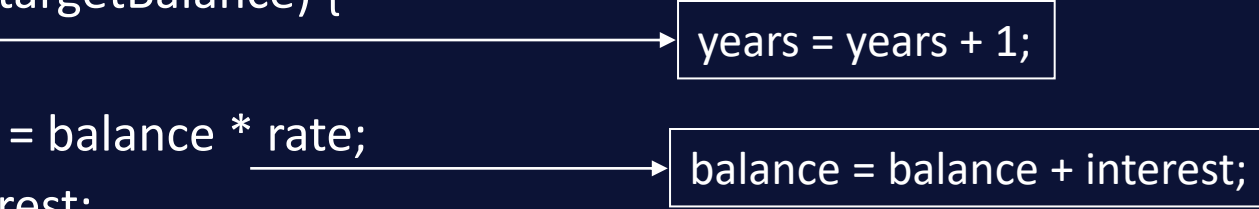


EXEMPLE

- Calcul des intérêts
- Étant donné le solde initial, le solde souhaité et le taux d'intérêt, combien d'années seront nécessaires pour atteindre le solde souhaité
 - au lieu d'utiliser une formule, on simule le calcul
- Algorithme (pseudocode):
 1. `ans = 0;`
 2. `WHILE` solde n'atteint pas le solde souhaité
 3. incrémenter ans
 4. ajouter l'intérêt au solde

PROGRAMME

```
public void nombreAnnees (double balance, double targetBalance, double
rate ) {
    int years = 0;
    while (balance < targetBalance) {
        years++;
        double interest = balance * rate;
        balance += interest;
    }
    System.out.println(years + " years are needed");
}
```



Appel de la méthode:

`nombreAnnees(1000, 1500, 0.05)`

Résultat:

`56 years are needed`

FACTORIELLE

```
public class Factorielle
{
    public static double factorielle(int x) {
        if (x < 0) return 0.0;
        double fact = 1.0;
        while (x > 1) {
            fact = fact * x;
            x = x - 1;
        }
        return fact;
    }

    public static void main(String[] args) {
        int entree = Integer.parseInt(args[0]);
        double resultat = factorielle(entree);
        System.out.println(resultat);
    }
}
```

Si une méthode (ou un attribut, une variable) de la classe est utilisée par la méthode main (*static*), il faut qu'il soit aussi *static*.

Attention:

Array

Array list

String

a.length

a.size()

a.length()

TABLEAU

- Pour stocker une série de données de même nature
- Déclaration

`int [] nombre; // une série de valeurs int dans le tableau nommé nombre`

`String [][] etiquette; // un tableau à deux dimensions de valeurs String`

- Création

`nombre = new int[10]; // crée les cases nombre[0] à nombre[9]`

`etiquette = new String[3][5]; // crée etiquette[0][0] à etiquette[2][4]`

`int[] primes = {1, 2, 3, 5, 7, 7+4}; // déclare, crée de la bonne taille et initialise`

- Utilisation

`nombre[0] = 4;`

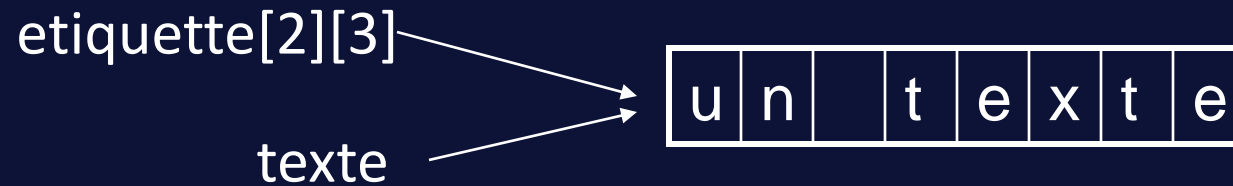
`for (int i=1; i<nombre.length; i++) nombre[i]=nombre[i]+1;`

`etiquette[2][3] = "un texte";`

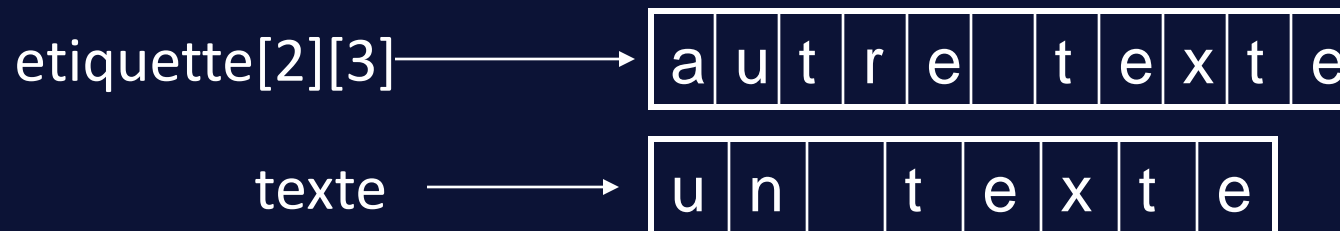
`String texte = etiquette[2][3];`

STRING

- Structure à deux parties:
 - En-tête: nom, longueur, ...
 - Corps: les caractères



- `String texte = etiquette[2][3];`
- Le contenu du corps ne peut pas être changé, une fois qu'il est créé (*String* est immuable)
- Par contre, on peut pointer/référencer `etiquette[2][3]` à un autre corps: `etiquette[2][3] = "autre texte";`



CLASSE ET OBJET

- Classe: moule pour fabriquer des objets
- Objet: élément concret produit par le moule
- Définition de classe:

```
class NomClasse {  
    Attributs;           String nom;  
    Méthodes;           int AnneeNaissance;  
    public int age() {...}  
}
```

- Une classe regroupe un ensemble d'objets (instances)

OBJET

- Structure à deux parties:
 - Référence
 - Corps
- Les étapes
 - Déclaration de la classe (e.g. *Personne*)
 - À l'endroit où on utilise:
 - Déclarer une référence du type de la classe
 - Créer une instance d'objet (*new*)
 - Manipuler l'objet

EXEMPLE

```
public class Personne {  
    public String nom;  
    public int anneeNaissance;  
    public int age() {return 2022 - anneeNaissance; }  
}
```

```
class Utilisation {  
    public static void main(String[] args) {  
        Personne qui;  
        qui = new Personne();  
        qui.nom = "Pierre";  
        qui.anneeNaissance = 1980;  
        System.out.println(qui.age());  
    }  
}
```

Déclaration de référence

Création d'une instance

Manipulation de l'instance
référée par la référence

qui

nom: "Pierre"

anneeNaissance: 1980

Personne:
age()

UN AUTRE EXEMPLE

```
class Circle {  
    public double x, y; // coordonnées du centre  
    private double r; // rayon du cercle  
    public Circle(double r) {  
        this.r = r;  
    }  
    public double area() {  
        return 3.14159 * r * r;  
    }  
}
```

'r' est inaccessible de
l'extérieur de la classe

constructeur

Math.PI

```
public class MonPremierProgramme {  
    public static void main(String[] args) {  
        Circle c; // c est une référence sur un objet de type Circle, pas encore un objet  
        c = new Circle(5.0); // c référence maintenant un objet alloué en mémoire  
        c.x = c.y = 10; // ces valeurs sont stockées dans le corps de l'objet  
        System.out.println("Aire de c :" + c.area());  
    }  
}
```


CONSTRUCTEURS D'UNE CLASSE

- Un constructeur est une façon de fabriquer une instance
- Une classe peut posséder plusieurs constructeurs
- Si aucun constructeur n'est déclaré par le programmeur, alors on a la version par défaut: `NomClasse()`
- Plusieurs versions d'une méthode: surcharge (*overloading*)

```
class Circle {  
    public double x, y; // coordonnées du centre  
    private double r; // rayon du cercle  
    public Circle(double r) {  
        this.r = r;  
    }  
    public Circle(double a, double b, double c) {  
        x = a; y = b; r = c;  
    }  
}  
  
public class Personne {  
    public String nom;  
    public int anneeNaissance;  
    public int age() { return 2008 - anneeNaissance; }  
}
```

this: réfère à l'objet courant

Le constructeur
`Personne()` est
déclaré par défaut

MANIPULATION DES RÉFÉRENCES

```
class Circle {  
    public double x, y; // coordonnées du centre  
    private double r; // rayon du cercle  
    public Circle(double r) {  
        this.r = r;  
    }  
    public Circle(double a, double b, double c) {  
        x = a; y = b; r = c;  
    }  
}
```

// Dans une méthode, par exemple, main:

```
Circle c1, c2;  
c1 = new Circle(2.0, 3.0, 4.0);  
c2 = c1; // c2 et c1 pointent vers le même objet  
c2.r = c2.r - 1; // l'objet a le rayon réduit  
c1 = new Circle(2.0); // c1 point vers un autre objet, mais c2  
ne change pas  
c1.x = 2.0; // on modifie le deuxième objet  
c2 = c1; // maintenant, c2 pointe vers le 2ième objet aussi
```

Que faire du premier objet?

Aucune référence ne pointe vers lui

L'objet est perdu et inutilisable

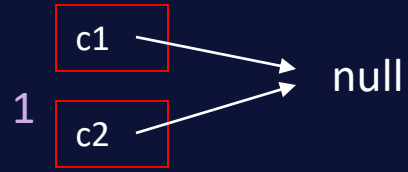
Ramasse miette (*garbage collector*) va récupérer l'espace occupé par l'objet

Comparaison des références

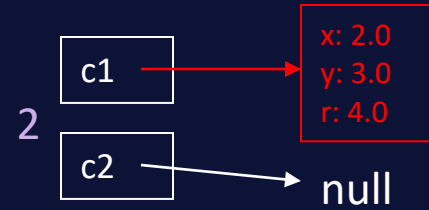
(c1 == c2): est-ce que c1 et c2 pointent vers le même objet?

ILLUSTRATION

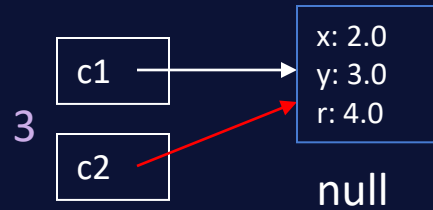
1. Cercle c1, c2;



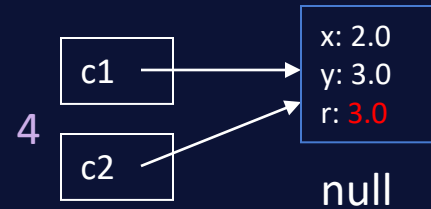
2. c1 = new Cercle(2.0, 3.0, 4.0);



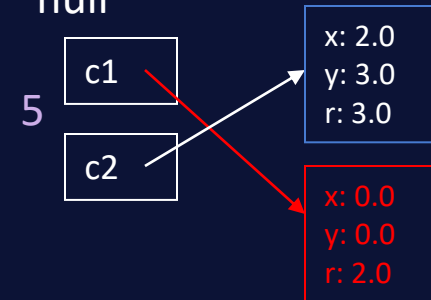
3. c2 = c1;



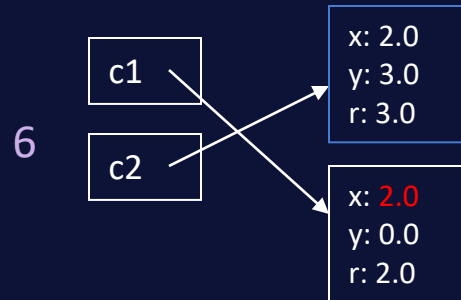
4. c2.r = c2.r - 1;



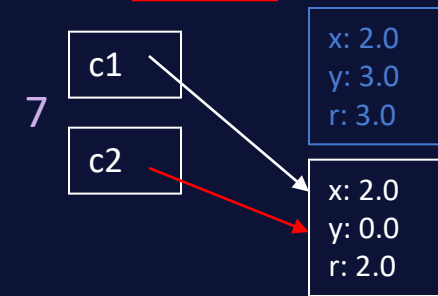
5. c1 = new Cercle(2.0);



6. c1.x = 2.0;



7. c2 = c1;



MANIPULATION DES OBJETS

- **Forme générale**

référence.attribut: réfère à un attribut de l'objet

référence.méthode(): réfère à une méthode de l'objet

- ***static***: associé à une classe

- Attribut (variable) statique: si on le change, ça change la valeur pour tous les objets de la classe
- Méthode statique: on la réfère à partir de la classe
 - *Classe.méthode*
 - E.g. *Math.sqrt(2.6)*: Appel à la méthode *sqrt* de la classe *Math*
 - Constante: *Math.PI*
 - Dans une classe: `static final float PI = 3.14159265358979;`
 - Une constante n'est pas modifiable

CLASSES ET HÉRITAGE

- Héritage
 - Les enfants héritent les propriétés du parent
 - Classe enfant (sous-classe) possède systématiquement les attributs et les méthodes de la classe parent (super-classe)
 - Héritage simple (une seule super-classe au plus)

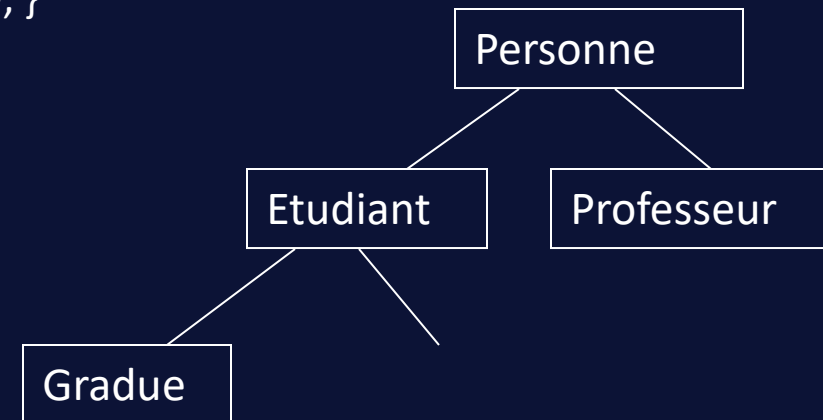
• E.g.

```
class Personne {  
    String nom;  
    int anneeNaissance;  
    public int age() { return 2008 - anneeNaissance; }  
}
```

```
class Etudiant extends Personne {  
    String [] cours;  
    String niveau;  
    String ecole;  
    ...  
}
```

• Ce qui est disponible dans Etudiant:

- nom, anneeNaissance, age(),
- cours, niveau, ecole, ...



PRINCIPE

- Définir les propriétés communes dans la classe supérieure
- Définir les propriétés spécifiques dans la sous-classe
- Regrouper les objets le plus possible
- Les objets d'une sous-classe sont aussi des objets de la super-classe
- La classe dont tous les objets appartiennent: *Object*
- Tester l'appartenance d'un objet dans une classe: *instanceof* (e.g. *instanceof Etudiant*)

EXEMPLE

```
public class Ellipse {  
    public double r1, r2;  
    public Ellipse(double r1, double r2) { this.r1 = r1; this.r2 = r2; }  
    public double area() {...}  
}
```

```
final class Circle extends Ellipse {  
    public Circle(double r) {super(r, r);}  
    public double getRadius() {return r1;}  
}
```

super(r,r): constructeur de la super-classe

final assure qu'aucune autre classe n'hériterait de Circle

// Dans une méthode

```
Ellipse e = new Ellipse(2.0, 4.0);
```

```
Circle c = new Circle(2.0);
```

```
System.out.println("Aire de e: " + e.area() + ", Aire de c: " + c.area());
```

```
System.out.println((e instanceof Circle)); // false
```

```
System.out.println((e instanceof Ellipse)); // true
```

```
System.out.println((c instanceof Circle)); // true
```

```
System.out.println((c instanceof Ellipse)); // true (car Circle dérive de Ellipse)
```

```
e = c;
```

```
System.out.println((e instanceof Circle)); // true
```

```
System.out.println((e instanceof Ellipse)); // true
```

```
int r = e.getRadius(); // erreur: méthode getRadius n'est pas trouvée dans la classe Ellipse
```

```
c = e; // erreur: type incompatible pour = Doit utiliser un cast explicite
```

CASTING

- La classe de la référence détermine ce qui est disponible (a priori)
- E.g.

```
class A {  
    public void meth() { System.out.println("Salut"); }  
}  
class B extends A {  
    int var;  
}
```

```
A a = new A();  
B b = new B();  
a.meth(); // OK  
b.meth(); // OK, héritée  
b.var = 1; // OK  
a.var = 2; // erreur  
a = b;  
a.var = 2; // erreur: var n'est a priori pas disponible pour une classe A  
((B) a).var = 2; // OK, casting
```

- *Casting*: transforme une référence d'une super-classe à celle d'une sous-classe
- Condition: l'objet référencé est bien de la sous-classe

SURCHARGE DE MÉTHODE

```
class A {  
    public void meth() {System.out.println("Salut"); }  
}  
class B extends A {  
    public void meth(String nom) {  
        System.out.println("Salut" +nom);  
    }  
}
```

- Dans la sous-classe: une version additionnelle
 - Signature de méthode: nom+type de paramètres
 - Surcharge: créer une méthode ayant une autre signature

OVERRIDING: ÉCRASEMENT

- Par défaut, une méthode est héritée par une sous-classe
- Mais on peut redéfinir la méthode dans la sous-classe (avec la même signature)
- Les objets de la sous-classe ne possèdent que la nouvelle version de la méthode
- E.g.

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

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

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

```
a.meth(); // Salut
```

```
b.meth(); // Hello
```

```
a = b; // a réfère à un objet de classe B
```

```
a.meth(); // Hello. Même si la référence est de classe A, l'objet est de classe B
```

CLASSE ABSTRAITE

- Certains éléments peuvent être manquants dans une classe, ou la classe peut être trop abstraite pour correspondre à un objet concret
- Classe abstraite
 - Une classe non complétée ou une classe conceptuellement trop abstraite
 - Classe *Shape*
 - on ne connaît pas la forme exacte, donc impossible de créer un objet
 - cependant, on peut savoir que chaque *Shape* peut être dessinée

```
abstract class Shape {  
    abstract void draw();  
}
```

INTERFACE

- Interface
 - Un ensemble de méthodes (comportements) exigées
 - Une classe peut se déclarer conforme à (implanter) une interface: dans ce cas, elle doit implanter toutes les méthodes exigées

- E.g.

```
public abstract interface Inter {  
    public abstract int carre(int a);  
    public abstract void imprimer();  
}  
  
class X implements Inter {  
    public int carre(int a) { return a*a; }  
    public void imprimer() {System.out.println("des informations"); }  
}
```

EXAMPLE

```
abstract class Shape { public abstract double perimeter(); }
interface Drawable { public void draw(); }
class Circle extends Shape implements Drawable, Serializable {
    public double perimeter() { return 2 * Math.PI * r ; }
    public void draw() {...}
}
class Rectangle extends Shape implements Drawable, Serializable {
    public double perimeter() { return 2 * (height + width); }
    public void draw() {...}
}
...
Drawable[] drawables = {new Circle(2), new Rectangle(2,3),
                        new Circle(5)};
for(int i=0; i<drawables.length; i++)
    drawables[i].draw();
```

UTILITÉ DE L'INTERFACE

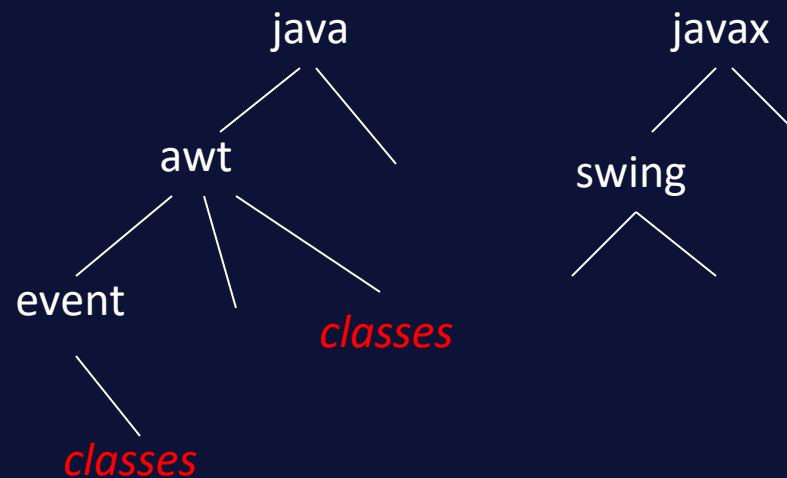
- Permet de savoir qu'une classe contient les implantations de certaines méthodes
- On peut utiliser ces méthodes sans connaître les détails de leur implantation
- Souvent utilisée pour des types abstraits de données (e.g. pile, queue, ...)

PACKAGE

- On organise les classes et les outils selon leurs fonctionnalités et les objets qu'elles manipulent
- Les classes qui traitent les mêmes objets: *package*
- Exemple:
 - Les classes pour traiter l'interface graphique sont dans le package *awt*
- Organisation des packages

- Hiérarchie

- java.awt
- java.awt.event
- javax.swing
- ...



UTILISER LES PACKAGES EXISTANTS

- Au début d'un fichier, importer les classes d'un package
- `import java.awt.*;`
 - Importer toutes les classes du package `java.awt` (reliées aux fenêtres)
- `import java.awt.event.*;`
 - Importer toutes les classes du package `java.awt.event` (reliées au traitement d'événements)

EXCEPTION

- Quand un cas non prévu survient, il est possible de le capter et le traiter par le mécanisme d'*Exception*
 - Si on capte et traite une exception, le programme peut continuer à se dérouler
 - Sinon, le programme sort de l'exécution avec un message d'erreur
 - Exemple d'exception: division par 0, ouvrir un fichier qui n'existe pas, ...
- Mécanisme de traitement d'exception
 - Définir des classes d'exception
 - *Exception*
 - *IOException*
 - *EOFException*, ...
 - Utiliser *try-catch* pour capter et traiter des exceptions

Hiérarchie des classes d'exceptions

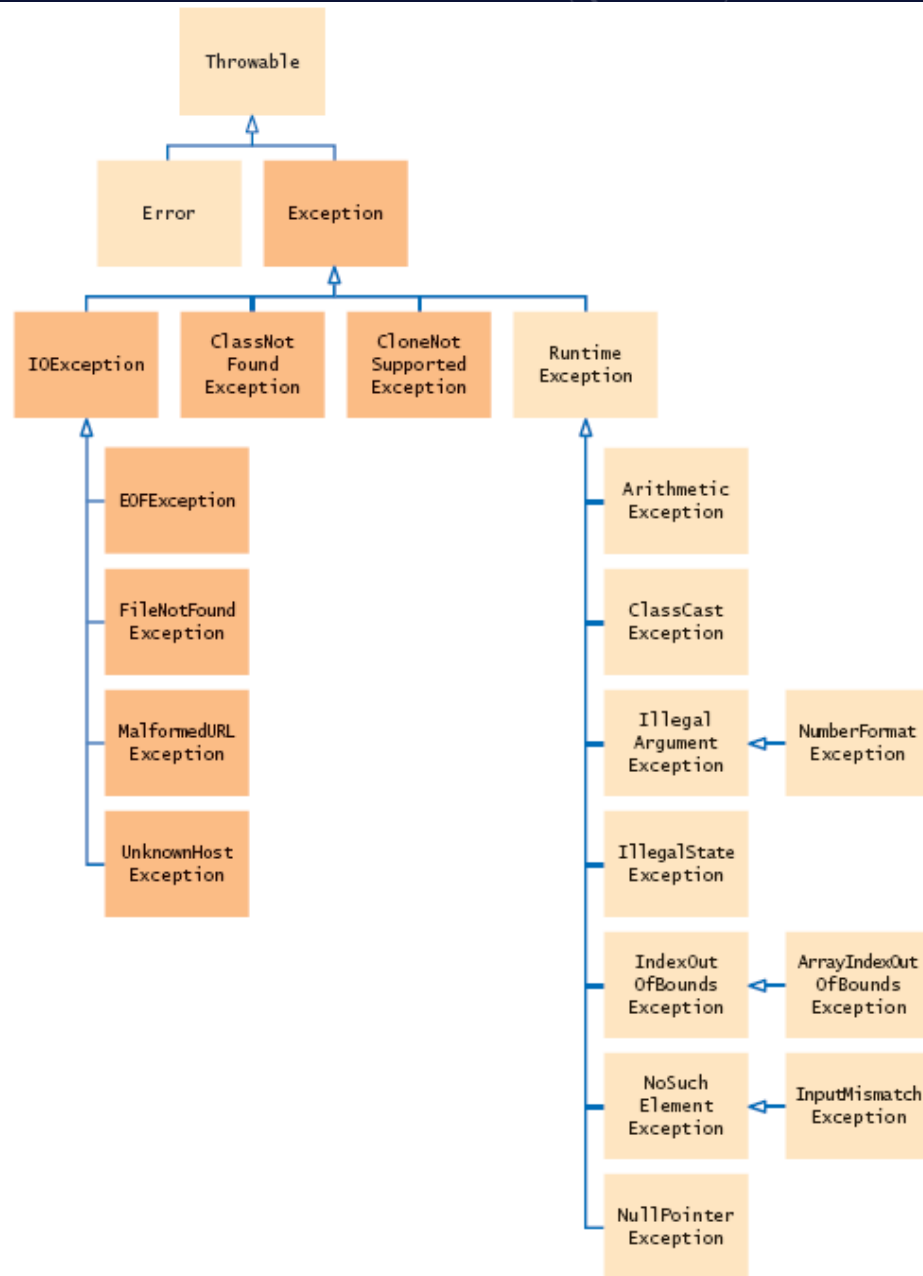


Figure 1 The Hierarchy of Exception Classes

ATTRAPER (*CATCH*) UNE EXCEPTION

- Attraper une exception pour la traiter

```
try {  
    statements  
    :  
    :  
} catch (ExceptionClass1 object) {  
    statements  
    :  
    :  
} catch (ExceptionClass2 object) {  
    statements  
    :  
    :  
} ...
```

Bloc où une exception
peut se générer

Blocs pour attraper
les exceptions

EXEMPLE

```
public static void ouvrir_fichier(String nom) {  
    try {  
        input = new BufferedReader(new FileReader(nom));  
    }  
    catch (IOException e) {  
        System.err.println("Impossible d'ouvrir le fichier d'entree.\n" +  
            e.toString());  
        System.exit(1);  
    }  
}
```

try: on tente d'effectuer des opérations

catch: si une exception de tel type survient au cours, on la traite de cette façon

ouverture
d'un fichier

FINALLY

Souvent combiné avec catch

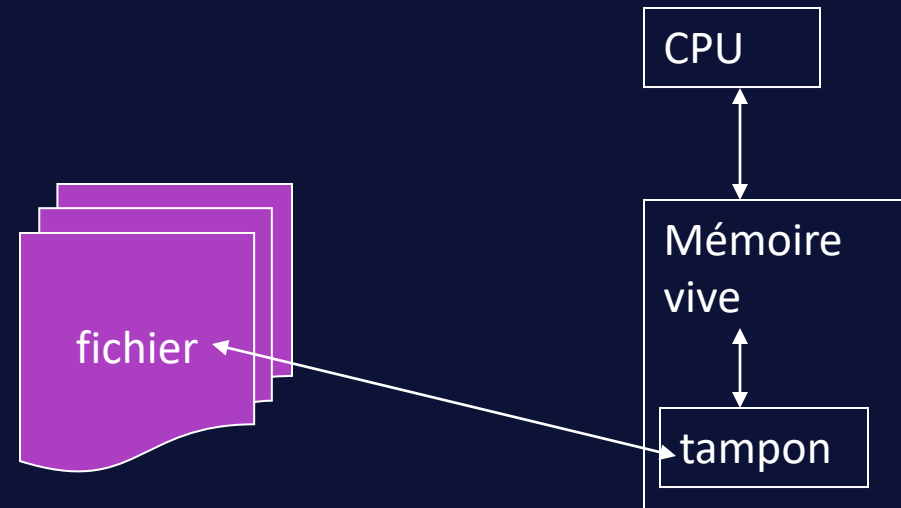
```
try
{
    statements
    ...
}
catch (ExceptionClass exceptionObject)
{
    statements
    ...
}
finally
{
    statements
    ...
}
```

Même si une exception est
attrapée, *finally* sera toujours
exécuté

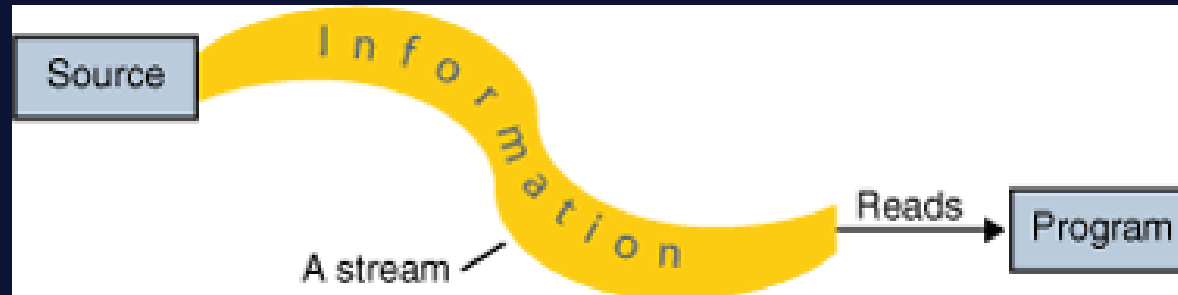
Utile pour s'assurer de certaine
sécurité (*cleanup*)

FICHER

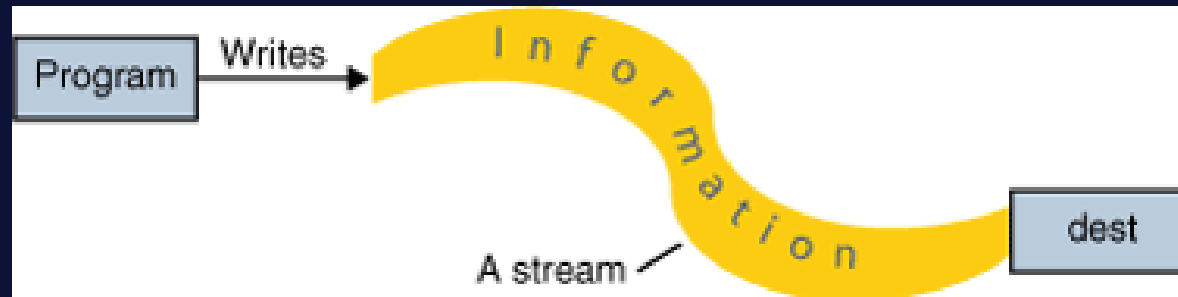
- Unité de stockage des données, sur disque dur
- Stockage permanent (vs. en mémoire vive)
- Un fichier contient un ensemble d'enregistrements
- Traitement



FICHER EN JAVA



- *Stream*: une suite de données (octets ou caractères)



OPÉRATIONS TYPIQUES

- Lecture:
 - Ouvrir un *stream*
 - Lire tant qu'il y a des données
 - Fermer le *stream*
- Écriture
 - Ouvrir un *stream* (ou créer un fichier)
 - Écrire des données tant qu'il y en a
 - Fermer le *stream*

Établir un canal de communication

Relâcher les ressources allouées

Écrire ce qu'il est dans le tampon, et relâcher les ressources allouées

EXEMPLE

```
public static void main(String[] args) {
    ouvrir_fichier("liste_mots");
    traiter_fichier();
    fermer_fichier();
}

public static void ouvrir_fichier(String nom) {
    try {
        input = new BufferedReader(
            new FileReader(nom));
    }
    catch (IOException e) {
        System.err.println("Impossible d'ouvrir
le fichier d'entree.\n" + e.toString());
        System.exit(1);
    }
}
```

```
public static void traiter_fichier() {
    String ligne;
    try { // catch EOFException
        ligne = input.readLine();
        while (ligne != null) {
            System.out.println(ligne);
            ligne = input.readLine();
        }
    }

    public static void fermer_fichier() {
        try {
            input.close();
            System.exit(0);
        }
        catch (IOException e) {
            System.err.println("Impossible de fermer les fichiers.\n" +
e.toString());
        }
    }
}
```

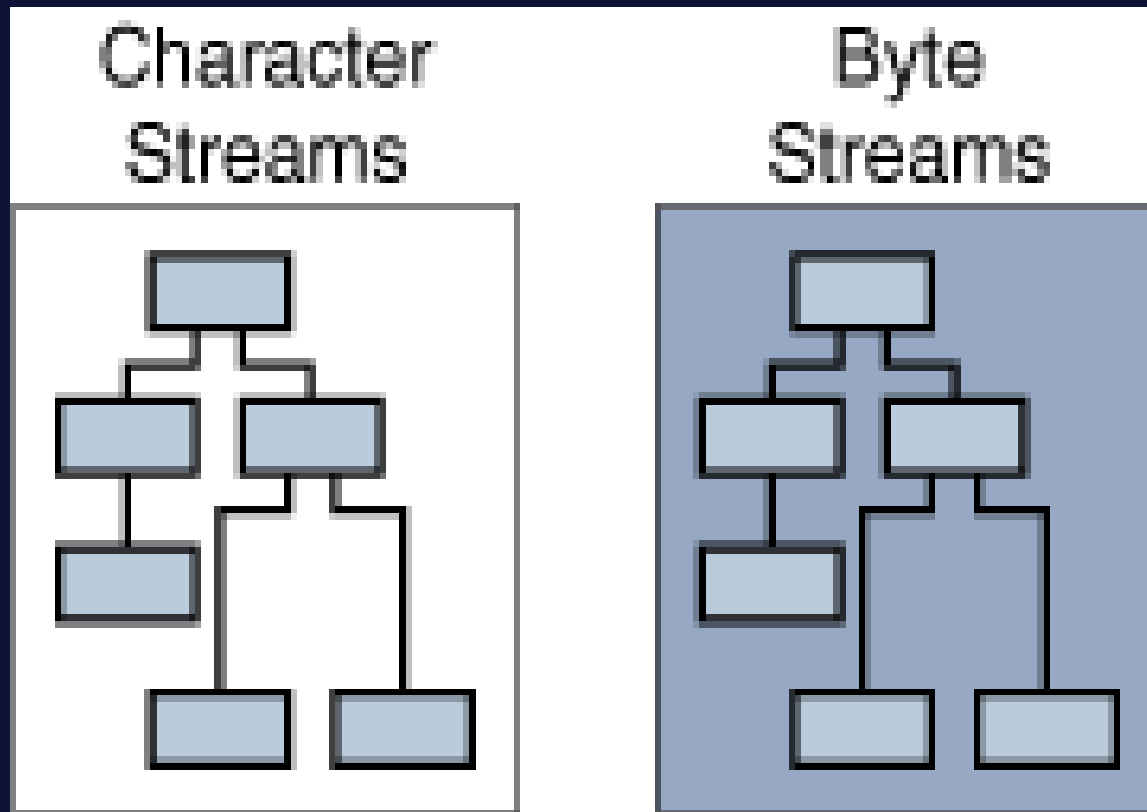
EXEMPLE

```
public static void traiter_fichier() {  
    String ligne;  
    try { // catch EOFException  
        ligne = input.readLine();  
        while (ligne != null) {  
            System.out.println(ligne);  
            ligne = input.readLine();  
        }  
    } catch (IOException e) {  
        System.err.println("Impossible de  
        traiter le fichier.\n" + e.toString());  
    }  
}
```

```
public static void fermer_fichier() {  
    try {  
        input.close();  
        System.exit(0);  
    }  
    catch (IOException e) {  
        System.err.println("Impossible  
        de fermer les fichiers.\n" +  
        e.toString());  
    }  
}
```

DEUX UNITÉS DE BASE

- Caractère (2 octets=16 bits) ou octet (8 bits)
- Deux hiérarchies de classes similaires (mais en parallèle)

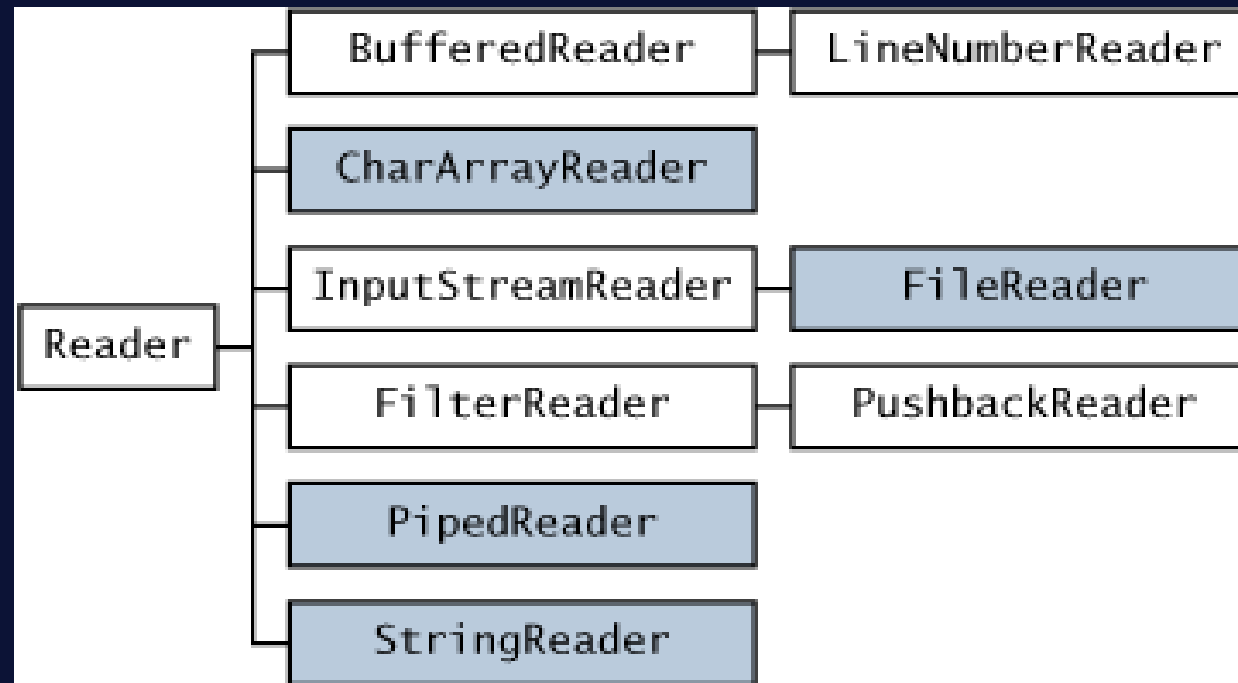


HIÉRARCHIES

- En haut des hiérarchies pour *stream* de caractères: 2 classes abstraites
- *Reader*
 - java.lang.Object
 - java.io.Reader
- *Writer*
 - java.lang.Object
 - java.io.Writer
- Implantent une partie des méthodes pour lire et écrire des caractères de 16 bits (2 octets)

HIÉRARCHIE DE *STREAM* DE CARACTÈRES

- Les sous-classes de *Reader*

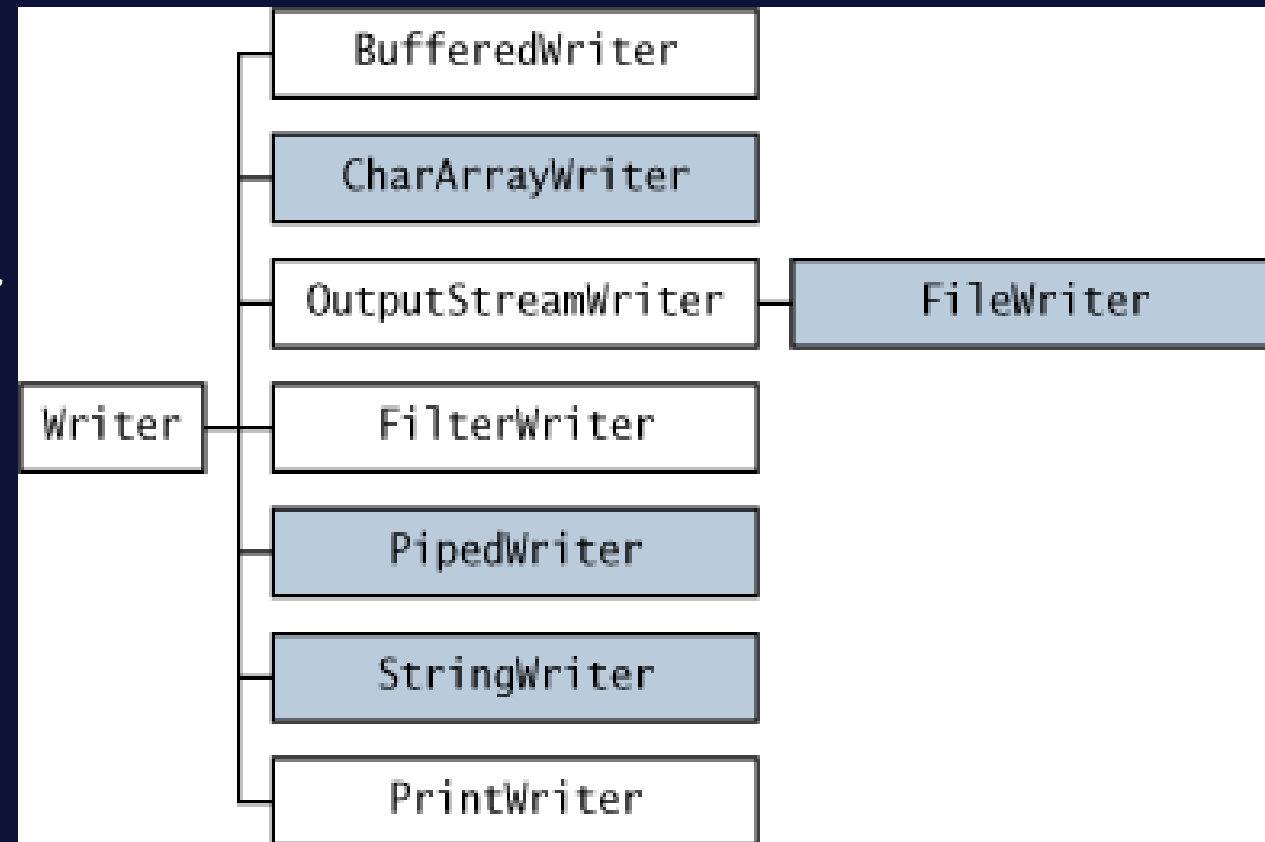


■ simple ■ pré-traitement

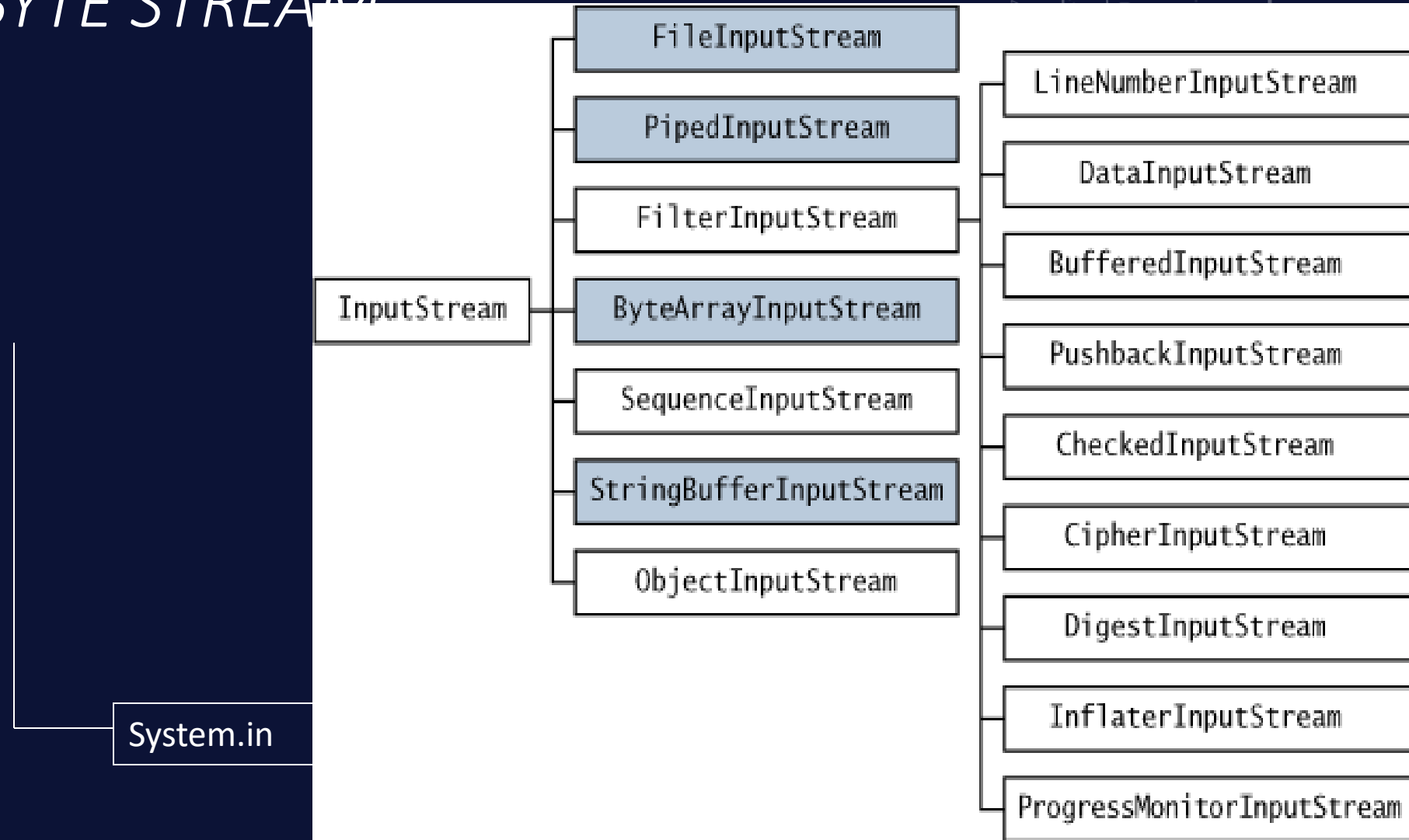
- Chaque sous-classe ajoute des méthodes

HIÉRARCHIE DE *STREAM* DE CARACTÈRES

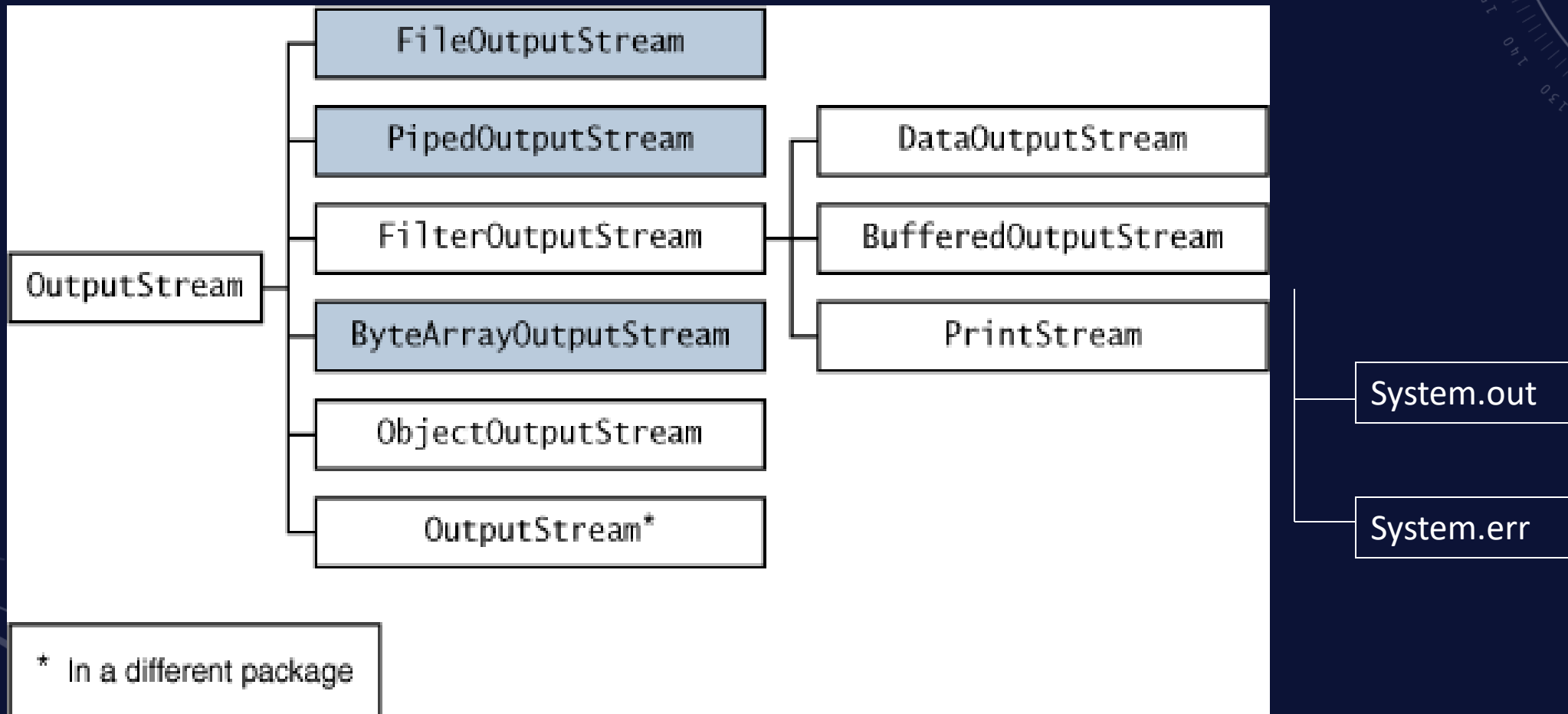
- Les sous-classes de *Writer*



HIÉRARCHIES *BYTE STREAM*



HIÉRARCHIE DE *BYTE STREAM*



EXEMPLE

- Utiliser FileReader et FileWriter
- Méthodes simples disponibles:
 - `int read()`, `int read(CharBuffer [])`, `write(int)`, ..
 - Exemple: copier un fichier caractère par caractère (comme un `int`)

```
import java.io.*;

public class Copy {
    public static void main(String[] args) throws IOException {
        File inputFile = new File("farrago.txt");
        File outputFile = new File("outagain.txt");

        FileReader in = new FileReader(inputFile);
        FileWriter out = new FileWriter(outputFile);
        int c;
        while ((c = in.read()) != -1) out.write(c);

        in.close();
        out.close();
    }
}
```

- Méthodes limitées

Fin de fichier: -1

AUGMENTER LES POSSIBILITÉS: *WRAP*

- Créer un *stream* en se basant sur un autre:

```
FileReader in = new FileReader(new File("farrago.txt"));
```

- **Avantage:**

- Obtenir plus de méthodes

- Dans *File*: les méthodes pour gérer les fichiers (`delete()`, `getPath()`, ...) mais pas de méthode pour la lecture
- Dans *FileReader*: les méthodes de base pour la lecture

- Un autre exemple:

```
DataOutputStream out = new DataOutputStream(  
    new FileOutputStream("invoice1.txt"));
```

- *FileOutputStream*: écrire des bytes
- *DataOutputStream*: méthodes pour les types de données de base: `write(int)`, `writeBoolean(boolean)`, `writeChar(int)`, `writeDouble(double)`, `writeFloat(float)`, ...

SÉRIALISER

- Convertir un objet (avec une structure) en une suite de données dans un fichier
- Reconvertir du fichier en un objet
- Utilisation: avec **ObjectOutputStream**

```
Employee[] staff = new Employee[3];  
ObjectOutputStream out = new ObjectOutputStream(new  
    FileOutputStream("test2.dat"));  
out.writeObject(staff);  
out.close();
```

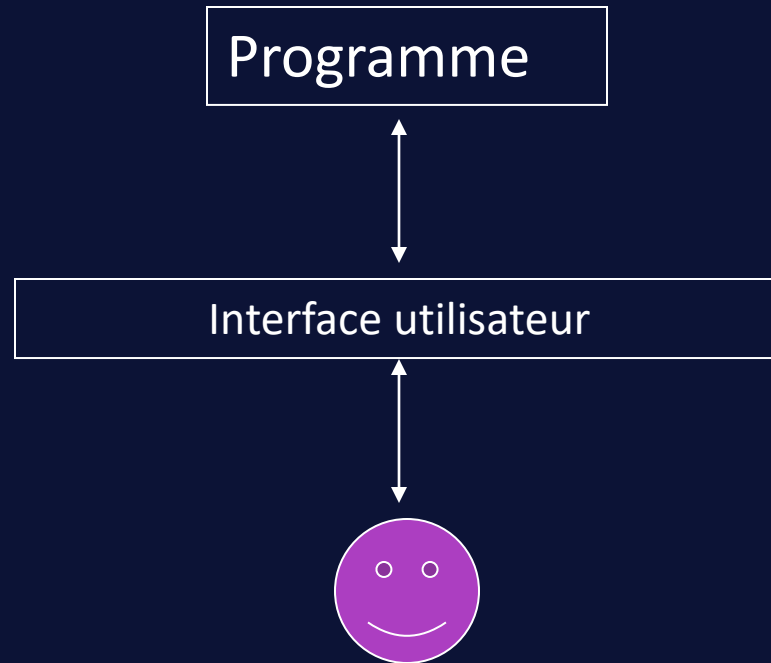
SÉRIALISER

- Utilité de sérialisation
 - Stocker un objet dans un fichier
 - Créer une copie d'objet en mémoire
 - Transmettre un objet à distance
 - Devient une transmission de *String*

INTERFACE GRAPHIQUE

- Comment créer des fenêtres?
- Comment gérer les interactions avec l'utilisateur?
 - Traiter des événements

GÉNÉRALITÉ



Rôles d'une interface utilisateur:

- montrer le résultat de l'exécution
- permettre à l'utilisateur d'interagir

EXEMPLE SIMPLE

```
import javax.swing.*;
public class DisplayFrame {
    public static void main (String[] args) {
        JFrame f = new JFrame("FrameDemo");
        //... components are added to its content frame.
        f.setSize(300,200);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

Importer le package

Créer un objet

Définir la taille

afficher



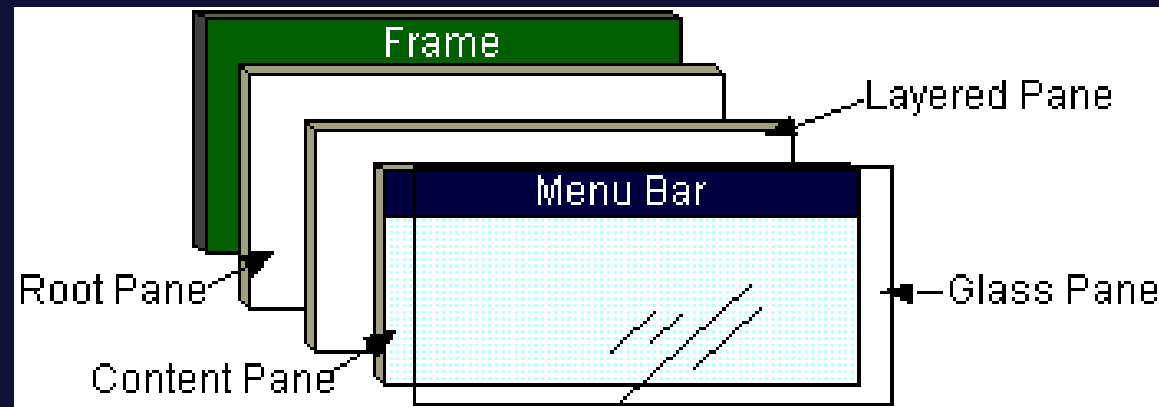
AFFICHER UNE INTERFACE

- Importer le package (les classes)
 - Les classes sont regroupées en package
 - Importer un package = importer toutes les classes du package
 - `import javax.swing.*;`
- Créer une fenêtre graphique (JFrame, ...)
- Mettre les paramètres (taille, ...)
- Afficher
- Différence:
 - `import java.awt.*;` les classes dans *awt*
 - `import java.awt.event.*;` les classes dans *event*

INSÉRER DES ÉLÉMENTS DANS LA FENÊTRE



- Composition d'une fenêtre JFrame



Structure interne de JFrame

Typiquement, on insère des éléments graphiques dans ContentPane

AJOUTER DES COMPOSANTS DANS UNE FENÊTRE

```
import javax.swing.*;

public class DisplayFrame {

    public static void main (String[] args) {
        JFrame f = new JFrame("FrameDemo");
        JLabel label = new JLabel("Hello World");
        JPanel p = (JPanel)f.getContentPane();
        p.add(label);

        f.setSize(300,200); //alternative: f.pack();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

Haut niveau

Composante
de base

Niveau
intermédiaire



COMPOSER UNE FENÊTRE

- Créer une fenêtre (1)
- Créer un ou des composants intermédiaires (2)
 - Pour *JFrame*, un *JPanel* est associé implicitement (ContentPane)
- Créer des composants de base (3)
- Insérer (3) dans (2)
- Insérer (2) dans (1)
- Afficher

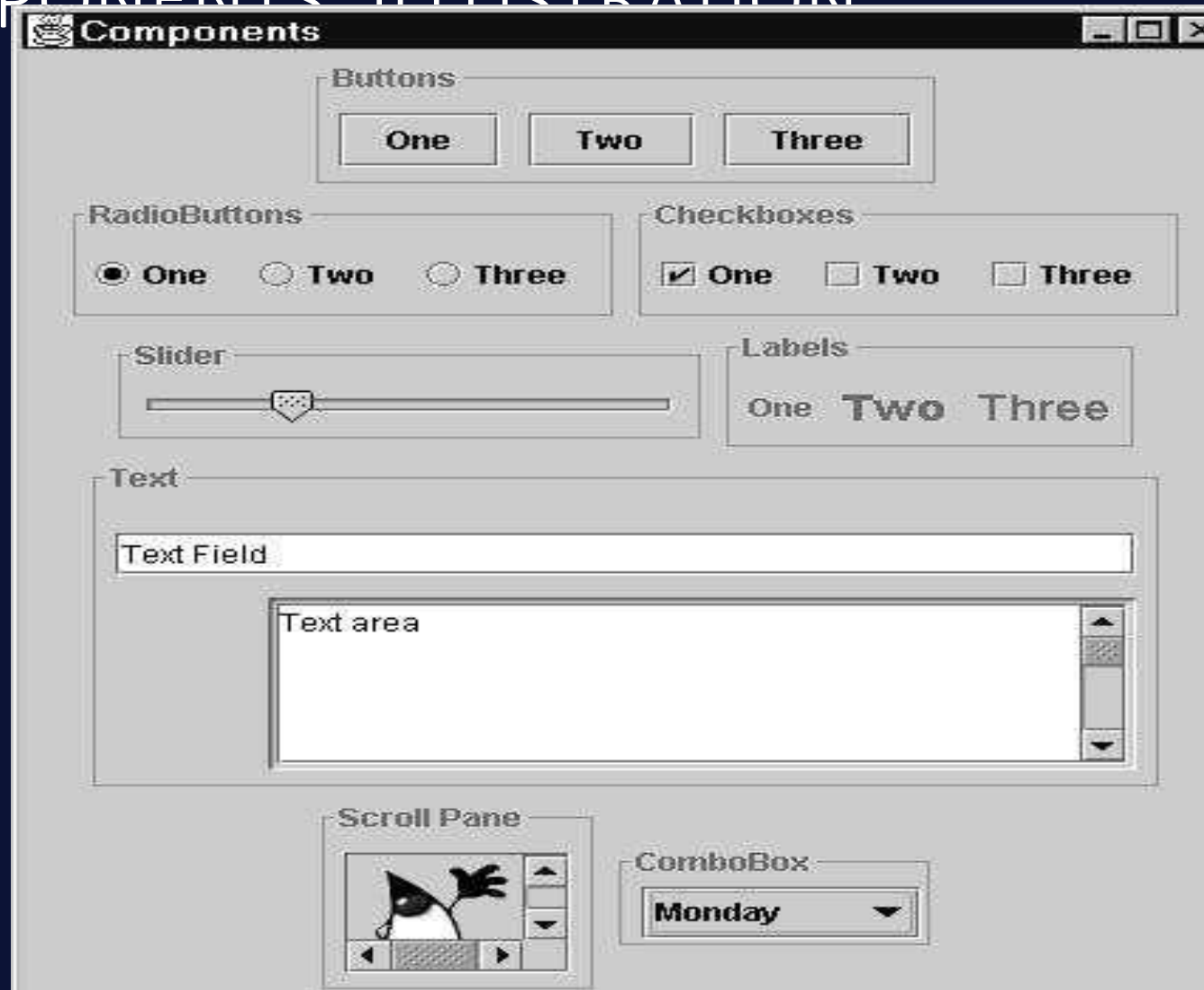
COMPOSANTS DE BASE POUR OBTENIR DES DONNÉES

- *JButton*
- *JCheckBox* a toggled on/off button displaying state to user.
- *JRadioButton* a toggled on/off button displaying its state to user.
- *JComboBox* a drop-down list with optional editable text field. The user can key in a value or select a value from drop-down list.
- *Jlist* allows a user to select one or more items from a list.
- *Jmenu* popup list of items from which the user can select.
- *Jslider* lets user select a value by sliding a knob.
- *JTextField* area for entering a single line of input.

COMPOSANTS DE BASE POUR AFFICHER L'INFORMATION

- *JLabel* contains text string, an image, or both.
- *JProgressBar* communicates progress of some work.
- *JToolTip* describes purpose of another component.
- *Jtree* a component that displays hierarchical data in outline form.
- *Jtable* a component user to edit and display data in a two-dimensional grid.
- *JTextArea*, *JTextPane*, *JEditorPane*
 - define multi-line areas for displaying, entering, and editing text.

SWING COMPONENTS. ILLUSTRATION



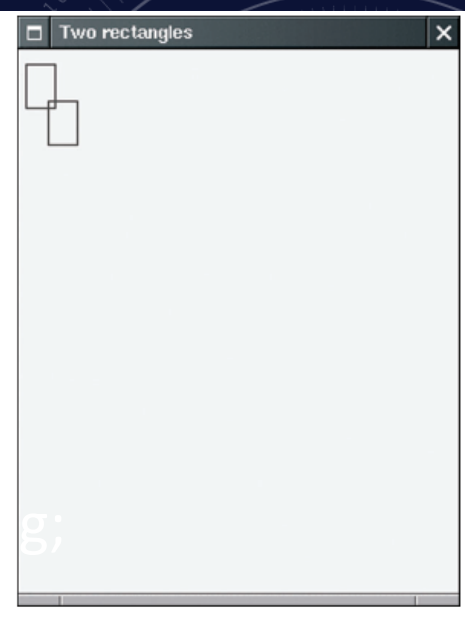
DÉFINIR SES PROPRES CLASSES

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JPanel;
05: import javax.swing.JComponent;
06:
07: /**
08:  A component that draws two rectangles.
09:  */
10: public class RectangleComponent extends JComponent
11: {
12:     public void paintComponent (Graphics g)
```

```
13:     {
14:         // Recover Graphics2D
15:         Graphics2D g2 = (Graphics2D) g;
16:
17:         // Construct a rectangle and draw it
18:         Rectangle box = new Rectangle(5, 10, 20, 30);
19:         g2.draw(box);
20:
21:         // Move rectangle 15 units to the right and 25 units
22:         box.translate(15, 25);
23:
24:         // Draw moved rectangle
25:         g2.draw(box);
26:     }
27: }
```

Figure 2

Drawing Rectangles



CRÉER ET VOIR L'OBJET

```
01: import javax.swing.JFrame;
02:
03: public class RectangleViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:
09:         final int FRAME_WIDTH = 300;
10:         final int FRAME_HEIGHT = 400;
11:
12:         frame.setSize(FRAME_WIDTH, FRAME_HEIGHT);
13:         frame.setTitle("Two rectangles");
14:         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15:
16:         RectangleComponent component = new RectangleComponent();
17:         frame.add(component);
18:
19:         frame.setVisible(true);
20:     }
21: }
```


ALTERNATIVE DE JFRAME: JAPPLET

- Applet = une fenêtre dans un navigateur
- Permet à n'importe quel utilisateur de lancer une application
- Plus de contrainte de sécurité (pas d'écriture)
- Programme englobé dans une page Web

AFFICHER DEUX RECTANGLES

```
01: import java.awt.Graphics;  
02: import java.awt.Graphics2D;  
  
03: import java.awt.Rectangle;  
  
04: import javax.swing.JApplet;  
  
05:  
06: /**  
07:  * An applet that draws two rectangles.  
08:  */  
09: public class RectangleApplet extends JApplet  
10: {  
11:     public void paint (Graphics g)  
12:     {  
13:         // Prepare for extended graphics  
14:         Graphics2D g2 = (Graphics2D) g;  
15:  
16:         // Construct a rectangle and draw it  
17:         Rectangle box = new Rectangle(5, 10, 20, 30);  
18:         g2.draw(box);  
19:  
20:         // Move rectangle 15 units to the right and 25 units down  
21:         box.translate(15, 25);  
22:  
23:         // Draw moved rectangle  
24:         g2.draw(box);  
25:     }  
26: }  
27:
```

Au lancement, paint(...) est
automatiquement exécutée
Pour ré-exécuter: repaint()

LANCER UN APPLET

À partir d'une page Web:

```
<html>
  <head>
    <title>Two rectangles</title>
  </head>
  <body>
    <p>Here is my <i>first applet</i>:</p>
    <applet code="RectangleApplet.class" width="300" height="400">
    </applet>
  </body>
</html>
```

DIFFÉRENCE

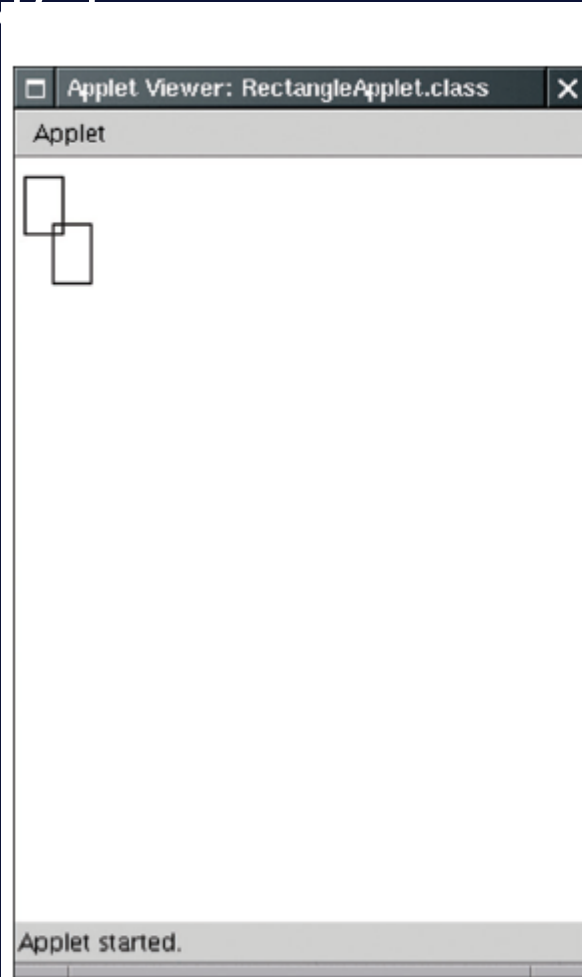


Figure 3
An Applet in the Applet Viewer

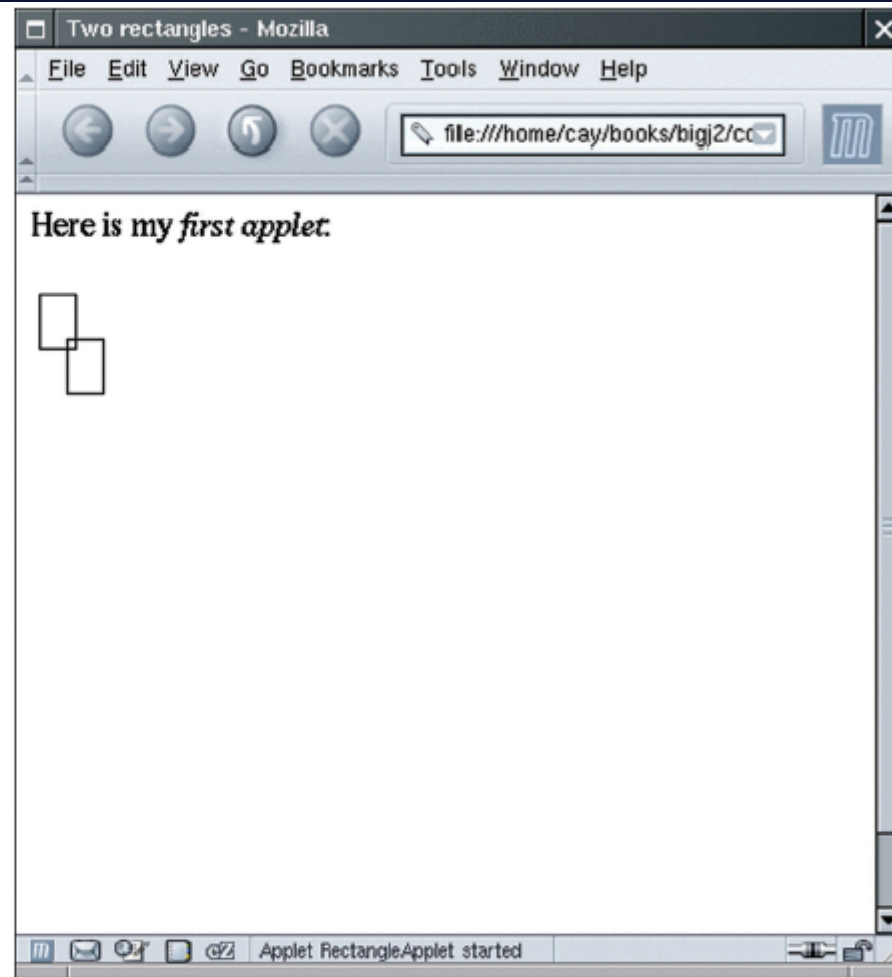


Figure 4
An Applet in a Web Browser

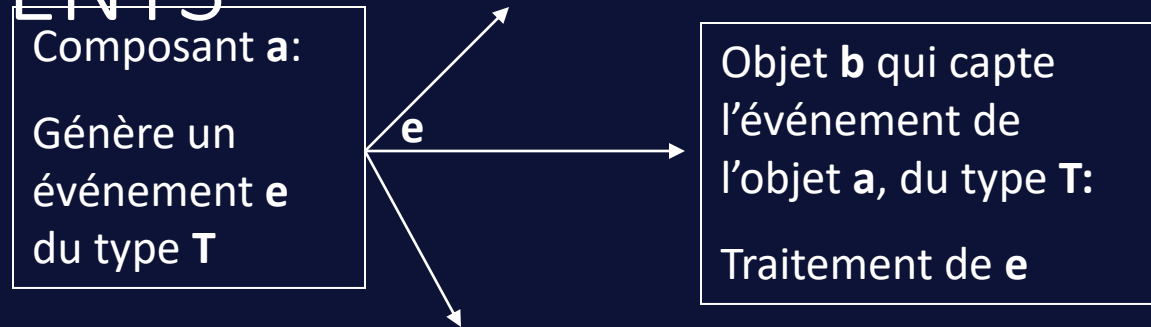
ÉVÉNEMENT

- Les événements sont des objets
- Sous-classes de la class abstraite `java.awt.AWTEvent`
- Les composants génèrent des événements
- Événements: chaque interaction de l'utilisateur sur un composant génère un événement
 - bouger la souris
 - cliquer sur un bouton
 - fermer une fenêtre
 - ...
- Un événement contient des informations: source, type d'événement, ...

```
public Object getSource();
```

- Utile pour détecter d'où provient l'événement

PROPAGATION ET TRAITEMENT DES ÉVÉNEMENTS



- Les événements sont générés et propagés
- Certains autres objets sont capables de capter des événements des types spécifiés, provenant de ces composants
 - **b** écoute les événements du type **T** venant de **a**
 - **b** est un ***listener*** de **a**
- On peut activer le traitement suite à la capture d'un événement
 - Le traitement lancé par l'objet **b**
- Programmation par événement
 - Le programme réagit aux événements

LISTENER ET EVENT HANDLER: DONNER LA CAPACITÉ D'ENTENDRE UN ÉVÉNEMENT

- *Listener*: Un objet est intéressé à *écouter* l'événement produit (être signalé quand il y a un événement)
- *Listener* doit implanter l'interface *event listener interface* associée à chaque type d'événement
- *Event Handler*: le programme qui lance un traitement suite à un événement
- Exemple

```
public class Capteur implements ActionListener
{
    public void actionPerformed(ActionEvent e) { ... }
}
```

ActionListener
classe

Type d'événement écouté

Action déclenchée

Exemple: changer la couleur

```
import java.awt.*;  
import javax.swing.*;  
import java.awt.event.*;
```

```
public class JAppletExample  
    extends JApplet  
    implements ActionListener  
{  
    private JButton rouge, bleu;  
    private Color couleur = Color.BLACK;
```

```
    public void init()  
    {  
        rouge = new JButton("Rouge");  
        bleu = new JButton("Bleu");
```

```
        Container content = getContentPane();  
        content.setLayout(new FlowLayout());  
        content.add(rouge);  
        content.add(bleu);
```

```
// Liens d'ecoute
```

```
    rouge.addActionListener(this);  
    bleu.addActionListener(this);  
}
```

```
// affichage
```

```
    public void paint(Graphics g)  
    {  
        super.paint(g);  
        g.setColor(couleur);  
        g.drawString("Choisir une couleur.", 100, 100);  
    }
```

```
// methode qui reagit aux evenements
```

```
    public void actionPerformed (ActionEvent e)  
    {  
        if (e.getSource() == rouge) couleur=Color.RED;  
        else if (e.getSource() == bleu) couleur =  
            Color.BLUE;  
        repaint(); //appeler paint(...) pour repindre  
    }  
}
```


TYPES D'ÉVÉNEMENTS ET ÉCOUTEUR

- **ActionEvent, ActionListener:**
 - Button, List, TextField, MenuItem, JButton, ...
 - `public void actionPerformed(ActionEvent)`
- **AdjustmentEvent, AdjustmentListener**
 - Scrollbar, ScrollPane, ...
 - `public void adjustmentValueChanged(AdjustmentEvent)`
- **ItemEvent, ItemListener**
 - Checkbox, CheckboxMenuItem, Choice, List
 - `public void itemStateChanged(ItemEvent)`

TYPES D'ÉVÉNEMENTS ET ÉCOUTEUR

- **MouseEvent**
 - Souris
 - **MouseListener**
 - `public void mouseDragged(MouseEvent)`
 - `public void mouseMoved(MouseEvent)`
 - **MouseMotionListener**
 - `public void mousePressed(MouseEvent)`
 - `public void mouseReleased(MouseEvent)`
 - `public void mouseEntered(MouseEvent)`
 - `public void mouseExited(MouseEvent)`
 - `public void mouseClicked(MouseEvent)`
- **TextEvent, TextListener**
 - `TextComponent` et ses sous-classes
 - `public void textValueChanged(TextEvent)`

TÉLÉCOMMUNICATION EN JAVA

- Communication sur l'Internet
- Connexion dans Java

INTERNET

- Stockage de données (informations)
 - Serveur
 - Client
- Connexion
 - Connexion entre un client et un serveur
 - Un canal de communication
- Transmission
 - Protocole:
 - définit les commandes
 - le format de données transmises

SCHÉMA DE COMMUNICATION TYPIQUE

- Serveur:
 - Il est lancé, en attente de recevoir un message (commande)
- Client
 - Demande à établir une connexion avec le serveur
 - Transmet une commande au serveur
- Serveur
 - Reçoit la commande
 - Traite la commande
 - Renvoie la réponse
- Client
 - Reçoit la réponse
 - Continue à traiter, transmet une autre commande, ...



ÉTABLIR UNE CONNEXION

- Identifier l'adresse du serveur à laquelle envoyer une requête de connexion
- Adresse:
 - Adresse IP (Internet Protocol): 4 octets (4 entiers 0-255)
 - 130.65.86.66
- *Domain Naming Service* (DNS): le nom correspondant à une adresse IP
 - Ss_domaine. sous_domaine . domaine
 - java.sun.com, www.iro.umontreal.ca
 - Traduction de DNS en adresse IP: par un serveur DNS
- Serveur
 - Prêt à recevoir des requêtes des types préétablis
 - E.g. GET

PROTOCOLE

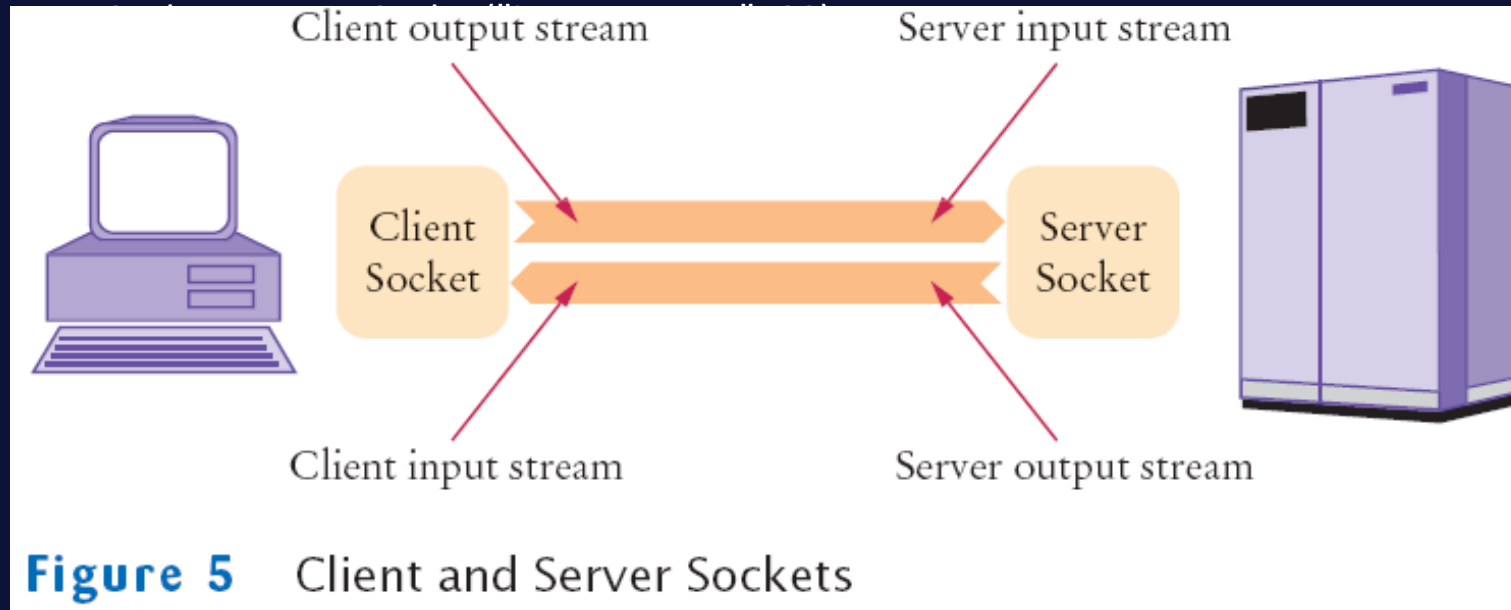
- Un serveur est établi pour communiquer selon un protocole
- Canal de communication (numéro de port)
 - 0 and 65,535
 - HTTP: par défaut: 80
- Serveur Web: prêt à recevoir les requêtes HTTP:
 - Adresse d'un document:
 - Uniform Resource Locator (URL)
 - `java.sun.com/index.html`
 - Commande
 - `GET /index.html HTTP/1.0` (suivie d'une ligne blanche)
 - `http://java.sun.com/index.html`

PROTOCOLE HTTP

Commande	Signification
• GET	Return the requested item
• HEAD	Request only the header information of an item
• OPTIONS	Request communications options of an item
• POST	Supply input to a server-side command and return the result
• PUT	Store an item on the server
• DELETE	Delete an item on the server
• TRACE	Trace server communication

EN JAVA

- Établir une connexion avec un serveur Web
 - Créer un socket entre Client et Serveur
 - `Socket s = new Socket(hostname, portnumber);`



EN JAVA

- Obtenir les *streams* du *socket*

```
InputStream instream = s.getInputStream();
```

```
OutputStream outstream = s.getOutputStream();
```

- *Cast* les *streams*

```
Scanner in = new Scanner(instream);
```

```
PrintWriter out = new PrintWriter(outstream);
```

- Fermer le *socket*

```
s.close();
```

EXEMPLE

- Un programme pour obtenir une page web d'un site
 - établir une connexion avec un serveur
 - envoyer une requête
 - recevoir la réponse
 - fermer

`java WebGet java.sun.com /`

- Lancer WebGet avec 2 paramètres:
 - `java.sun.com`: DNS
 - `/`: page racine
 - Port par défaut: 80

```
01: import java.io.InputStream;
02: import java.io.IOException;
03: import java.io.OutputStream;
04: import java.io.PrintWriter;
05: import java.net.Socket;
06: import java.util.Scanner;
07:
14: public class WebGet
15: {
16:     public static void main(String[] args) throws IOException
17:     {
18:
19:
20:         String host;
21:         String resource;
22:
23:         if (args.length == 2)
24:         {
25:             host = args[0];
26:             resource = args[1];
27:         }
28:         else
29:         {
30:             System.out.println("Getting / from java.sun.com");
31:             host = "java.sun.com";
32:             resource = "/";
33:         }
34:
```

```
37:     final int HTTP_PORT = 80;
38:     Socket s = new Socket(host, HTTP_PORT);
39:
42:     InputStream instream = s.getInputStream();
43:     OutputStream outstream = s.getOutputStream();
44:
47:     Scanner in = new Scanner(instream);
48:     PrintWriter out = new PrintWriter(outstream);
49:
52:     String command = "GET " + resource + "
HTTP/1.0\n\n";
53:     out.print(command);
54:     out.flush();
55:
58:     while (in.hasNextLine())
59:     {
60:         String input = in.nextLine();
61:         System.out.println(input);
62:     }
63:
66:     s.close();
67: }
68: }
```

RÉSVLTAT: JAVA WEBGET

HTTP/1.1 200 OK

Server: Sun-Java-System-Web-Server/6.1

Date: Tue, 28 Mar 2006 20:07:26 GMT

Content-type: text/html; charset=ISO-8859-1

Set-Cookie: SUN_ID=132.204.24.63:218361143576446; EXPIRES=Wednesday, 31-Dec-2025

23:59:59 GMT; DOMAIN=.sun.com; PATH=/

Set-cookie: JSESSIONID=519A024C45B4C300DA868D076CA33448; Path=/

Connection: close

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<title>Java Technology</title>
```

```
<meta name="keywords" content="Java, platform" />
```

```
<meta name="collection" content="reference">
```

```
<meta name="description" content="Java technology is a portfolio of products that are based on the power of networks and the idea that the same software should run on many different kinds of systems and devices." />
```

```
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
```

```
<meta name="date" content="2006-03-23" />
```

...

RÉSUMÉ SUR JAVA

- Programmation classique: traiter des données des types primitifs
- Programmation OO: regrouper les données (attributs) et leurs traitements (méthodes)
- Outils disponibles
 - Classes regroupées dans des packages
 - Interface graphique
 - Communication à travers l'Internet
 - Package pour interagir avec un SGBD (Système de gestion de base de données)
 - ...

À RETENIR

- Programme = ?
- Comment un programme est traduit en code exécutable? (compilation ou interprétation)
- Environnement de programmation
- Quels sont les opérations qu'on peut mettre dans un programme?
- Concept de base: variable, type, classe, objet, héritage (OO), ...
- Utilisation des packages existants
- Principe de gestion d'interface graphique
- Principe de télécommunication en Java