

Cours 2

(30 septembre)



D) Un exemple...

- Pile abstraite et diverses implémentations

Type abstrait de données

NOM

pile[X]

FONCTIONS

vide : pile[X] -> Boolean

nouvelle : -> pile[X]

empiler : X x pile[X] -> pile[X]

dépiler : pile[X] -> X x pile[X]

PRECONDITIONS

dépiler(s: pile[X]) \Leftrightarrow (not vide(s))

AXIOMES

forall x in X, s in pile[X]

vide(nouvelle())

not vide(empiler(x,s))

dépiler(empiler(x,s))=(x,s)

Remarques

- Le type est paramétré par un autre type
- Les axiomes correspondent aux pré-conditions
- Il n'y pas de représentation
- Il faudrait vérifier que cette définition caractérise bien un pile au sens usuel du terme (c'est possible)



Pile abstraite en java

```
package pile;
```

```
abstract class Pile <T>{  
    abstract public T empiler(T v) ;  
    abstract public T dépiler() ;  
    abstract public Boolean estVide() ;  
}
```

Divers

- `package`: regroupement de diverses classes
- `abstract`: signifie qu'il n'y a pas d'implémentation
- `public`: accessible de l'extérieur
- La classe est paramétrée par un type (java 1.5)

Implémentations

- On va implémenter la pile:
 - avec un objet de classe `Vector` (classe définie dans `java.util.package`) en fait il s'agit d'un `ArrayList`
 - Avec un objet de classe `LinkedList`
 - Avec `Integer` pour obtenir une pile de `Integer`

Une implémentation

```
package pile;
import java.util.EmptyStackException;
import java.util.Vector;
public class MaPile<T> extends Pile<T>{
    private Vector<T> items;
    // Vector devrait être remplacé par ArrayList
    public MaPile() {
        items =new Vector<T>(10);
    }
    public Boolean estVide(){
        return items.size()==0;
    }
    public T empiler(T item){
        items.addElement(item);
        return item;
    }
    //...
```


Suite

```
//...
public synchronized T dépiler() {
    int len = items.size();
    T item = null;
    if (len == 0)
        throw new EmptyStackException();
    item = items.elementAt(len - 1);
    items.removeElementAt(len - 1);
    return item;
}
}
```

Autre implémentation avec listes

```
package pile;
import java.util.LinkedList;
public class SaPile<T> extends Pile<T> {
    private LinkedList<T> items;
    public SaPile(){
        items = new LinkedList<T>();
    }
    public Boolean estVide(){
        return items.isEmpty();
    }
    public T empiler(T item){
        items.addFirst(item);
        return item;
    }
    public T dépiler(){
        return items.removeFirst();
    }
}
```

Une pile de Integer

```
public class PileInteger extends Pile<Integer>{
    private Integer[] items;
    private int top=0;
    private int max=100;
    public PileInteger(){
        items = new Integer[max];
    }
    public Integer empiler(Integer item){
        if (this.estPleine())
            throw new EmptyStackException();
        items[top++] = item;
        return item;
    }
    //...
```

Suite...

```
public synchronized Integer dépiler() {
    Integer item = null;
    if (this.estVide())
        throw new EmptyStackException();
    item = items[--top];
    return item;
}
public Boolean estVide() {
    return (top == 0);
}
public boolean estPleine() {
    return (top == max - 1);
}
protected void finalize() throws Throwable {
    items = null; super.finalize();
}
}
```



Comment utiliser ces classes?

- Le but est de pouvoir écrire du code utilisant la classe Pile abstraite
- Au moment de l'exécution, bien sûr, ce code s'appliquera à un objet concret (qui a une implémentation)
- Mais ce code doit s'appliquer à toute implémentation de Pile

Un main

```
package pile;
public class Main {
    public static void vider(Pile p){
        while(!p.estVide()){
            System.out.println(p.dépiler());
        }
    }
    public static void main(String[] args) {
        MaPile<Integer> p1= new MaPile<Integer>();
        for(int i=0;i<10;i++)
            p1.empiler(i);
        vider(p1);
        SaPile<String> p2= new SaPile<String>();
        p2.empiler("un");
        p2.empiler("deux");
        p2.empiler("trois");
        vider(p2);
    }
}
```

E) java: quelques rappels...

- Un source avec le suffixe `.java`
- Une classe par fichier source (en principe) même nom pour la classe et le fichier source (sans le suffixe `.java`)
- Méthode
 - ```
public static void main(String[]);
```
  - `main` est le point d'entrée
- Compilation génère un `.class`
- Exécution en lançant la machine java

# Généralités...

---

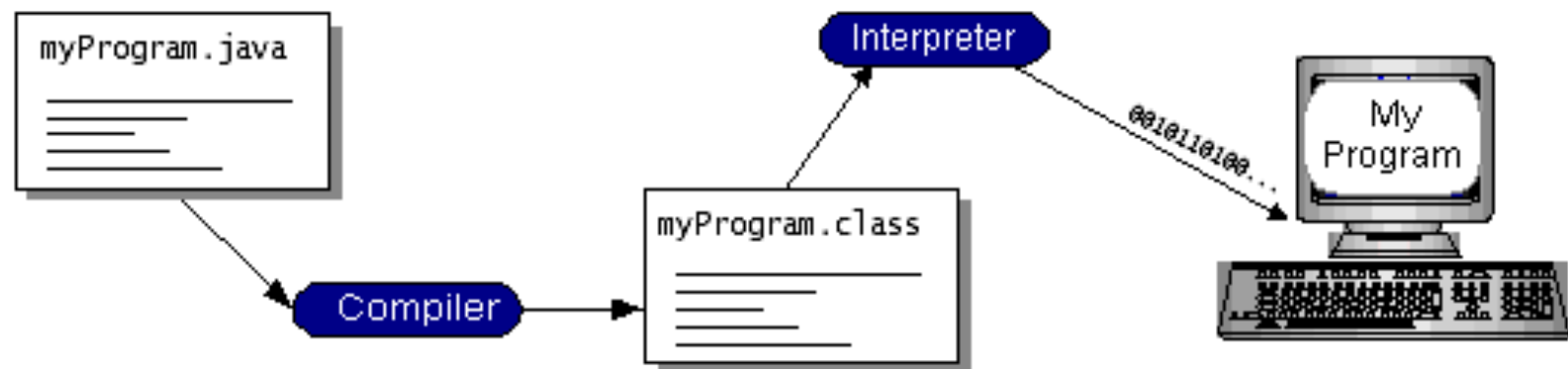
- Un peu plus qu'un langage de programmation:
  - "gratuit"!
  - Indépendant de la plateforme
  - *Langage interprété et byte code*
    - Portable
  - Syntaxe à la C
  - Orienté objet (classes héritage)
    - Nombreuses bibliothèques
  - Pas de pointeurs! (ou que des pointeurs!)
    - Ramasse-miettes
  
  - Multi-thread
  - Distribué (WEB) applet, servlet etc...
  - url: <http://java.sun.com>
    - <http://java.sun.com/docs/>



# Plateforme Java

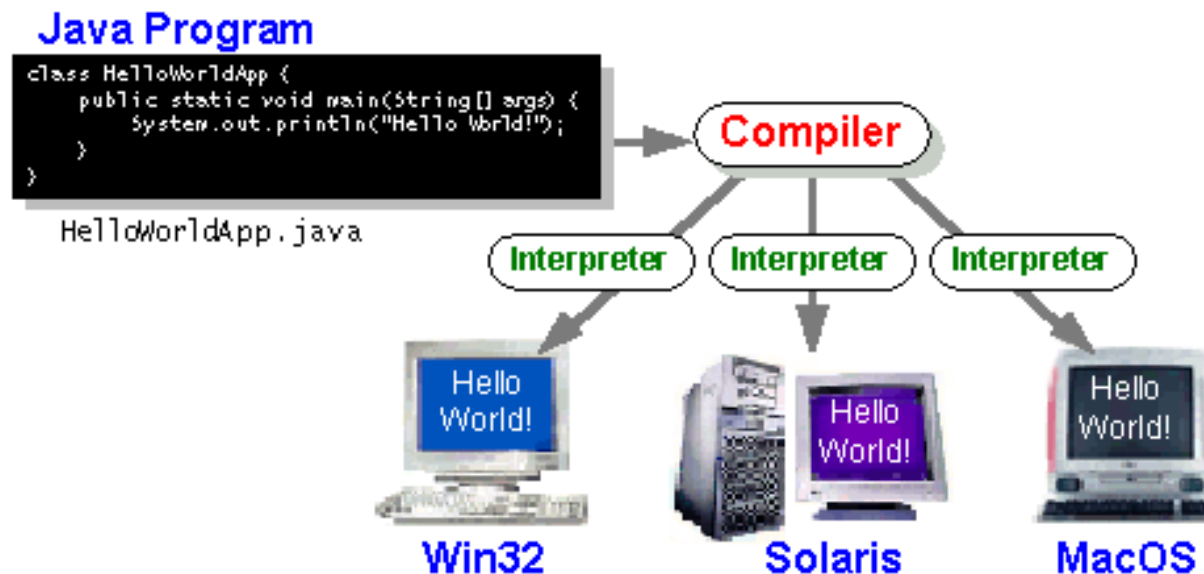
- La compilation génère un .class en bytecode (langage intermédiaire indépendant de la plateforme).
- Le bytecode est interprété par un interpréteur Java JVM

Compilation `javac`  
interprétation `java`



# Langage intermédiaire et Interpréteur...

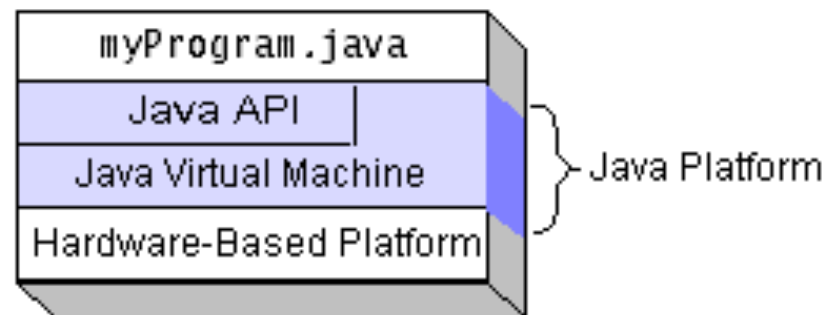
- **Avantage: indépendance de la plateforme**
  - Échange de byte-code (applet)
- **Inconvénient: efficacité**



# Plateforme Java

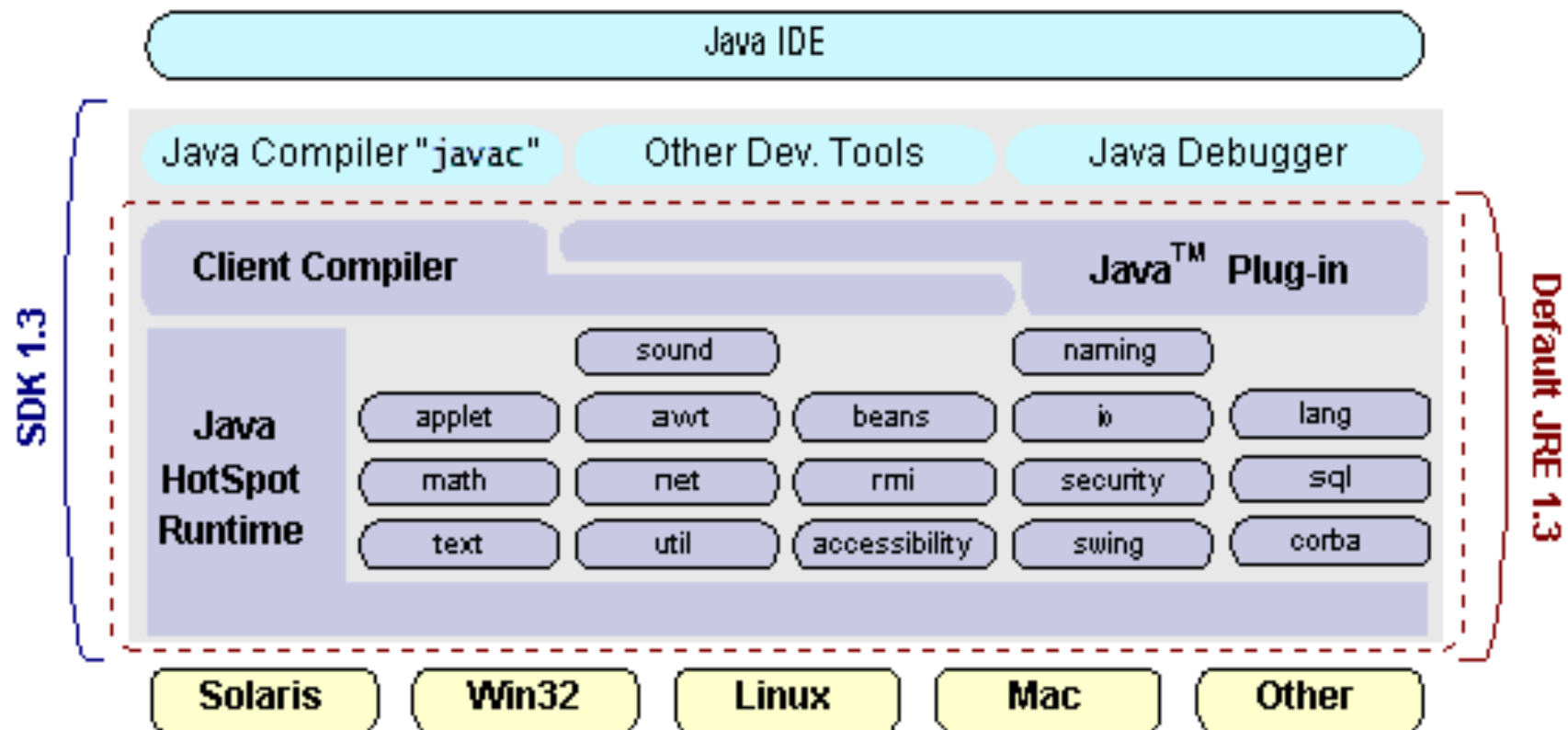
---

- La plateforme java: software au-dessus d'une plateforme exécutable sur un hardware (exemple MacOS, linux ...)
- Java VM
- Java application Programming Interface (Java API):

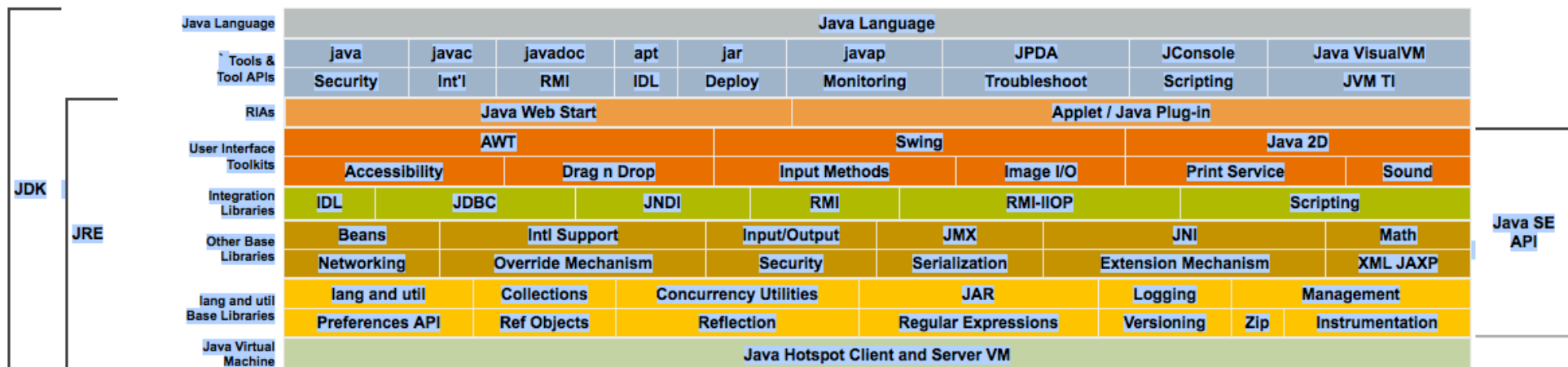


# Tout un environnement...


- Java 2 sdk: JRE (java runtime environment + outils de développements compilateur, débogueurs etc...)



# Tout un environnement...



[Java SE 6 API Documentation](#)



# Trois exemples de base

---

- Une application
- Une applet
- Une application avec interface graphique

# Application:

---

- Fichier Appli.java:

```
/**
 * Une application basique...
 */
class Appli {
 public static void main(String[] args) {
 System.out.println("Bienvenue en L3...");
 //affichage
 }
}
```

# Compiler, exécuter...

---

- Créer un fichier `App1.java`
  - Compilation:
    - `javac App1.java`
  - Création de `App1.class` (bytecode)
  - Interpréter le byte code:
    - `java App1`
  - Attention aux suffixes!!!
    - (il faut que `javac` et `java` soient dans `$PATH`)
- Exception in thread "main" java.lang.NoClassDefFoundError:
- Il ne trouve pas le main -> vérifier le nom!
  - Variable `CLASSPATH` ou option `-classpath`



# Remarques

---

- Commentaires `/* ... */` et `//`
- Définition de classe
  - une classe contient des méthodes (=fonctions) et des variables
  - Pas de fonctions ou de variables globales (uniquement dans des classes ou des instances)
- Méthode `main`:
  - `public static void main(String[] arg)`
    - `public`
    - `static`
    - `Void`
    - `String`
  - Point d'entrée

# Remarques

---

- Classe System
  - out est une variable de la classe System
  - println méthode de System.out
  - out est une variable de classe qui fait référence à une instance de la classe PrintStream qui implémente un flot de sortie.
    - Cette instance a une méthode println

# Remarques...

---

- Classe: définit des méthodes et des variables (déclaration)
- Instance d'une classe (objet)
  - Méthode de classe: fonction associée à (toute la) classe.
  - Méthode d'instance: fonction associée à une instance particulière.
  - Variable de classe: associée à une classe (globale et partagée par toutes les instances)
  - Variable d'instance: associée à un objet (instancié)
- Patience...

# Applet:

---

- Applet et WEB
  - Client (navigateur) et serveur WEB
  - Le client fait des requêtes html, le serveur répond par des pages html
  - Applet:
    - Le serveur répond par une page contenant des applets
    - Applet: byte code
    - Code exécuté par le client
    - Permet de faire des animations avec interfaces graphiques sur le client.
    - Une des causes du succès de java.

# Exemple applet

---

- Fichier MonApplet.java:

```
/**
 * Une applet basique...
 */
import java.applet.Applet;
import java.awt.Graphics;
public class MonApplet extends Applet {
 public void paint(Graphics g){
 g.drawString("Bienvenue en en L3...", 50,25);
 }
}
```

# Remarques:

---

- import et package:
  - Un package est un regroupement de classes.
  - Toute classe est dans un package
  - Package par défaut (sans nom)
  - classpath
- `import java.applet.*;`
  - Importe le package `java.applet`
    - Applet est une classe de ce package,
    - Sans importation il faudrait `java.applet.Applet`

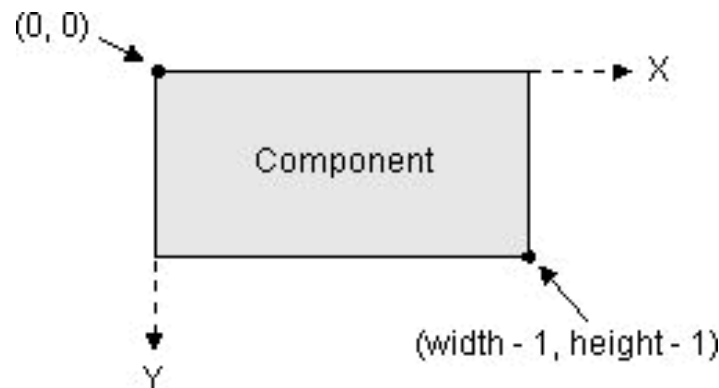
# Remarques:

---

- La classe Applet contient ce qu'il faut pour écrire une applet
- ... extends Applet:
  - La classe définie est une extension de la classe Applet:
    - Elle contient tout ce que contient la classe Applet
    - (et peut redéfinir certaines méthodes (paint))
  - Patience!!

# Remarques...

- Une Applet contient les méthodes `paint`, `start` et `init`. En redéfinissant `paint`, l'applet une fois lancée exécutera ce code redéfini.
- `Graphics g` argument de `paint` est un objet qui représente le contexte graphique de l'applet.
  - `drawString` est une méthode (d'instance) qui affiche une chaîne,
  - `50, 25`: affichage à partir de la position  $(x,y)$  à partir du point  $(0,0)$  coin en haut à gauche de l'applet.







# Pour exécuter l'applet

---

- L'applet doit être exécutée dans un navigateur capable d'interpréter du bytecode correspondant à des applet.
- Il faut créer un fichier HTML pour le navigateur.

# Html pour l'applet

---

- Fichier Bienvenu.html:

```
<HTML>
<HEAD>
<TITLE> Une petite applet </TITLE>
<BODY>
<APPLET CODE='MonApplet.class' WIDTH=200
 Height=50>
</APPLET>
</BODY>
</HTML>
```

# Html

---

□ Structure avec balises:

□ Exemples:

- `<HTML> </HTML>`

- url:

- `<a target="_blank" href="http://www.liafa.jussieu.f/~hf">page de hf</a>`

□ Ici:

```
<APPLET CODE='MonApplet.class' WIDTH=200
 Height=50>
</APPLET>
```

# Exemple interface graphique

Fichier MonSwing.java:

```
/**
 * Une application basique... avec interface graphique
 */
import javax.swing.*;
public class MonSwing {
 private static void creerFrame() {
 //Une formule magique...
 JFrame.setDefaultLookAndFeelDecorated(true);
 //Creation d'une Frame
 JFrame frame = new JFrame("MonSwing");
 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 //Afficher un message
 JLabel label = new JLabel("Bienvenue en L3...");
 frame.getContentPane().add(label);
 //Afficher la fenetre
 frame.pack();
 frame.setVisible(true);
 }
 public static void main(String[] args) {
 creerFrame();
 }
}
```



# Remarques

---

- ❑ Importation de packages
- ❑ Définition d'un conteneur top-level JFrame, implémenté comme instance de la classe JFrame
- ❑ Affichage de ce conteneur
- ❑ Définition d'un composant JLabel, implémenté comme instance de JLabel
- ❑ Ajout du composant JLabel dans la JFrame
- ❑ Définition du comportement de la JFrame sur un click du bouton de fermeture
- ❑ Une méthode main qui crée la JFrame



# Pour finir...

---

- Java 1.5 et 6 annotations, types méthodes paramétrés par des types
- Très nombreux packages
- Nombreux outils de développement (gratuits)
  - eclipse, netbeans..



# En plus...

---

- Entrées-sorties

# Entrée-sortie

---

```
public static void main(String[] args) {
 // sortie avec printf ou
 double a = 5.6d ;
 double b = 2d ;
 String mul = "multiplié par" ;
 String eq="égal";
 System.out.printf(Locale.ENGLISH,
 "%3.2f x %3.2f = %6.4f \n", a ,b , a*b);
 System.out.printf(Locale.FRENCH,
 "%3.2f %s %3.2f %s %6.4f \n", a, mul,b eq,a*b);
 System.out.format(
 "Aujourd'hui %1$tA, %1$te %1$tB,"+
 " il est: %1$tH h %1$tM min %1$ts \n",
 Calendar.getInstance());
 // System.out.flush();
}
```





# Sortie

---

$$5.60 \times 2.00 = 11.2000$$

5,60 multiplié par 2,00 égal 11,2000

Aujourd'hui mardi, 10 octobre, il est: 15 h  
31 min 01

# Scanner

---

```
Scanner sc = new Scanner(System.in);
for(boolean fait=false; fait==false;){
 try {
 System.out.println("Répondre o ou 0:");
 String s1 =sc.next(Pattern.compile("[0o]"));
 fait=true;
 } catch(InputMismatchException e) {
 sc.next();
 }
}
if (sc.hasNextInt()){
 int i= sc.nextInt();
 System.out.println("entier lu "+i);
}
System.out.println("next token :"+sc.next());
sc.close();
```

# Scanner

---

```
if (sc.hasNextInt()){
 int i= sc.nextInt();
 System.out.println("entier lu "+i);
}
System.out.println("next token :"+sc.next()); sc.close();
String input = "1 stop 2 stop éléphant gris stop rien";
Scanner s = new(Scanner(input).useDelimiter("\\s*stop\\s*"));
 System.out.println(s.nextInt());
 System.out.println(s.nextInt());
 System.out.println(s.next());
 System.out.println(s.next());
 s.close();
}
```



# Sortie

---

- next token :o
- 1
- 2
- éléphant gris
- rien

# Les classes...

---

- System
  - System.out variable (static) de classe  
PrintStream
    - PrintStream contient print (et printf)
  - System.in variable (static) de classe  
InputStream
- Scanner

# Chapitre II

## Classes et objets (rappels)

---

(mais pas d'héritage)



# Classes et objets

---

- I) Introduction
- II) Classe: membres et modificateurs
- III) Champs: modificateurs
- IV) Vie et mort des objets,  
Constructeurs
- V) Méthodes
- VI) Exemple

# I) Introduction

---

- Classe
  - Regrouper des données et des méthodes
    - Variables de classe
    - Méthodes de classe
  - Classes $\leftrightarrow$ type
- Objet (ou instance)
  - Résultat de la création d'un objet
    - Variables d'instance
    - Variables de classe
- Toute classe hérite de la classe Object





## II) Classes

---

- Membres d'une classe sont:
  - Champs = données
  - Méthodes = fonctions
  - Classes imbriquées



# Modificateur de classe

---

- Précède la déclaration de la classe
  - Annotations (plus tard...)
  - `public` (par défaut package)
  - `abstract`(incomplète, pas d'instance)
  - `final`(pas d'extension)
  - `strictfp` (technique...)

# III) Champs

---

- Modificateurs
  - annotations
  - Contrôle d'accès
    - private
    - protected
    - public
    - package
  - static (variables de classe)
  - final (constantes)
  - transient
  - volatile
- Initialisations
- Création par opérateur new

## IV) Vie et mort des objets, constructeurs

---

- Création d'une instance: opérateur new
- Objet mort = plus de référence à cet objet -> garbage collector
  - on peut exécuter du code spécifique quand un objet est détruit :  
`protected void finalize() throws Throwable`

# Références

---

- Une variable est (en général) une référence à un objet
  - Type primitif: directement une valeur
  - Type référence : une référence à un objet (existant ou créé par new)
    - null : référence universelle
    - conséquences:
      - dans le passage par valeur un type référence correspond à un passage par référence
      - 'a == b' teste si les a et b référencent le *même* objet
      - Méthode equals qui peut être redéfinie (défaut this==obj)

# Exemple

---

```
int i=0;
int j=0;
(i==j) // vrai
class A{
 int i=0;
}
A a;
A b=new A();
a=b;
(a==b) // vrai
b=new A();
(a==b) // faux
```



# Constructeurs

---

- Appelés par l'opérateur new pour créer un objet
  - Peuvent avoir des paramètres (avec surcharge)
  - Initialisent les objets
  - Constructeur par défaut (si aucun constructeur n'est défini)
  - Constructeur de copie

# Exemple:

---

```
public class Astre {
 private long idNum;
 private String nom = "<pasdenom>";
 private Astre orbite = null;
 private static long nextId = 0;
 /** Creation d'une nouvelle instance of Astre */
 private Astre() {
 idNum = nextId ++;
 }
 public Astre(String nom, Astre enOrbite){
 this();
 this.nom=nom;
 orbite=enOrbite;
 }
 public Astre(String nom){
 this(nom,null);
 }//...
```





# Exemples...

---

## □ Copie

```
public Astre(Astre a){
 idNum = a.idNum;
 nom=a.nom;
 orbite=a.orbite;
}
```

# Statique - dynamique

---

- Statique  $\leftrightarrow$  à la compilation
- Dynamique  $\leftrightarrow$  à l'exécution
- Le type d'une variable est déterminé à la compilation (déclaration et portée)
- Avec la possibilité de l'héritage une variable peut être une référence sur un objet d'un autre type que le type de sa déclaration

# Static

---

- Une variable (une méthode) déclarée `static` est une variable (méthode) de classe: elle est associée à la classe (pas à une instance particulière).
- Statique parce qu'elle peut être créée au moment de la compilation (pas de `new()`).
- Statique -> les initialisations doivent avoir lieu à la compilation.

# Initialisations

---

```
private static long nextId = 0;
```

□ Bloc d'initialisation

```
private static long nextId = 0;
```

```
{
```

```
 idNum = nextId++;
```

```
}
```

# Initialisation static

---

```
public class Puissancedeux {
 static int[] tab = new int[12];
 static{
 tab[0]=1;
 for(int i=0; i< tab.length-1;i++)
 tab[i+1]= suivant(tab[i]);
 }
 static int suivant(int i){
 return i*2;
 }
}
```

# V) Méthodes

---

- Modificateurs:
  - Annotations
  - Contrôle d'accès (comme pour les variables)
  - abstract
  - static n'a pas accès aux variables d'instances
  - final ne peut pas être remplacée
  - synchronized
  - native (utilisation de fonctions « native »)
  - strictfp

# Passage par valeur

---

```
public class ParamParVal {
 public static void parVal(int i){
 i=0;
 System.out.println("dans parVal i="+i);
 }
}
//...
int i =100;
System.out.println("Avant i="+i);
ParamParVal.parVal(i);
System.out.println("Avant i="+i);
```

```

Avant i=100
dans parVal i=0
Avant i=100
```

# Mais...

---

- Comme les variables sont de références (sauf les types primitifs)...

```
public static void bidon(Astre a){
 a=new Astre("bidon", null);
 System.out.println("bidon a="+a);
}
public static void bidonbis(Astre a){
 a.setNom("bidon");
 a.setOrbite(null);
 System.out.println("bidonbis a="+a);
}
```



# Méthodes...

---

## □ Contrôler l'accès:

//...


```
public void setNom(String n){
 nom=n;
}
public void setOrbite(Astre a){
 orbite=a;
}
public String getNom(){
 return nom;
}
public Astre getOrbite(){
 return orbite;
}
```

# Méthodes, remplacement...

---

```
public String toString(){
 String st=idNum + "("+nom+")";
 if (orbite != null)
 st += "en orbite "+ orbite;
 return st;
}
```

Remplace la méthode toString de la classe Object



# Nombre variable d'arguments...

---

```
public static void affiche(String ... list){
 for(int i=0;i<list.length;i++)
 System.out.print(list[i]+" ");
}
```

```
//...
```

```
affiche("un", "deux", "trois");
```



# Méthodes main

---

```
public static void main(String[] args) {
 for(int j =0; j<args.length;j++){
 System.out.print(args[j] + " ");
 }
}
```

Le main est le point d'accès et peut avoir des arguments:

# VI) exemple: Les astres...

```
package exempleClasses;

/**
 *
 * @author sans
 */
public class Astre {
 private long idNum;
 private String nom = "<pasdenom>";
 private Astre orbite = null;
 private static long nextId = 0;
 /** Creates a new instance of Astre */
 private Astre() {
 idNum = nextId ++;
 }
}
```


□



# Suite

---

```
public Astre(String nom, Astre enOrbite){
 this();
 this.nom=nom;
 orbite=enOrbite;
}
public Astre(String nom){
 this(nom,null);
}
public Astre(Astre a){
 idNum = a.idNum;
 nom=a.nom;
 orbite=a.orbite;
}//...
```



---

```
public void setNom(String n){
 nom=n;
}
public void setOrbite(Astre a){
 orbite=a;
}
public String getNom(){
 return nom;
}
public Astre getOrbite(){
 return orbite;
}
public String toString(){
 String st=idNum + "("+nom+")";
 if (orbite != null)
 st += "en orbite "+ orbite;
 return st;
}
```