Cours programmation-orientée objet en Java

Licence d'informatique

Hugues Fauconnier

hf@liafa.jussieu.fr

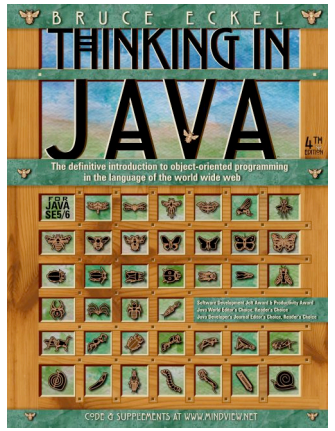
Plan du cours

- Introduction:
  - programmation objet pourquoi? Comment? Un exemple en Java
- Classes et objets (révision)
  - Méthodes et variables, constructeurs, contrôle d'accès, constructeurs
- Héritage: liaison dynamique et typage
  - Extension de classe, méthode et héritage, variables et héritage, constructeurs et héritage
- Héritage: compléments
  - classes abstraites et interfaces, classes internes et emboîtées, classe Object, clonage,
- Exceptions
  - Exceptions, assertions
- Divers
  - Enumeration, tableaux, conversions, noms
- Généricité
  - Généralités, types génériques imbriqués, types paramètres bornés, méthodes génériques
- Types de données
  - String et expressions régulières, Collections, Conteneurs, itérations
- Entrées-sorties
- Introduction à Swing
- Threads
- Compléments
  - Reflections, annotations, documentation…

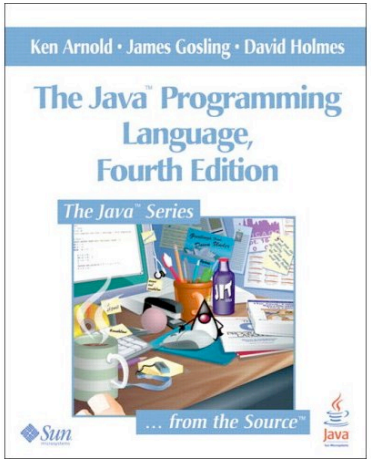
Le site du cours: http://www.liafa.jussieu.fr/~hf/verif/ens/an09-10/poo/L3.POO.html

Bibliographie

- De nombreux livres sur java (attention java >= 1.5)
- En ligne:
  - http://mindview.net/Books/TIJ4
  - Thinking in Java, 4th edition Bruce Eckel
  - http://java.sun.com/docs/index.html



- Livre conseillé:
  - The Java Programming language fourth edition AW Ken Arnold, James Gosling, David Holmes



Chapitre I

Introduction

A) Généralités

- Problème du logiciel:
  - Taille
  - Coût : développement et maintenance
  - Fiabilité
- Solutions :
  - Modularité
  - **Réutiliser le logiciel**
  - Certification

- Comment?

Typage…

- Histoire:
  - Fonctions et procédures (60 Fortran)
  - Typage des données (70) Pascal Algol
  - Modules: données + fonctions regroupées (80) ada
  - Programmation objet: classes, objets et héritage

B) Principes de base de la POO

- Objet et classe:
  - Classe = définitions pour des données (variables) + fonctions (méthodes) agissant sur ces données
  - Objet = élément d'une classe (instance) avec un état
  - (une méthode ou une variable peut être
    - de classe = commune à la classe ou
    - d'instance = dépendant de l'instance

)

Principes de bases (suite)

- Encapsulation et séparation de la spécification et de l'implémentation
  - Séparer l'implémentation de la spécification.
    - Ne doit être visible de l'extérieur que ce qui est nécessaire, les détails d'implémentation sont « cachés »
- Héritage:
  - Une classe peut hériter des propriétés d'une autre classe: un classe peut être une extension d'une autre classe.

Principes de bases de la POO

- Mais surtout notion de *polymorphisme*:
  - Si une classe A est une extension d'une classe B:
    - A doit pouvoir *redéfinir* certaines méthodes (disons f())
    - Un objet a de classe A doit pouvoir être considéré comme un objet de classe B
    - On doit donc accepter :
      - B b;
      - b=a; (a a toutes les propriétés d'un B)
      - b.f()
        - Doit appeler la méthode redéfinie dans A!
      - C'est le *transtypage*
    - (exemple: méthode paint des interfaces graphiques)

Principes de bases

- Polymorphisme:
  - Ici l'association entre le nom 'f()' et le code (code de A ou code de B) a lieu dynamiquement (=à l'exécution)

    Liaison dynamique
  - On peut aussi vouloir « paramétrer » une classe (ou une méthode) par une autre classe.

    Exemple: Pile *d'entiers*

    Dans ce cas aussi un nom peut correspondre à plusieurs codes, mais ici l'association peut avoir lieu de façon statique (au moment de la compilation)

C) Comment assurer la réutilisation du logiciel?

- Type abstrait de données
  - définir le type par ses propriétés (spécification)
- Interface, spécification et implémentation
  - Une interface et une spécification (=les propriétés à assurer) pour définir un type
  - Une (ou plusieurs) implémentation du type abstrait de données
    - Ces implémentations doivent vérifier la spécification

Comment assurer la réutilisation du logiciel?

- Pour l'utilisateur du type abstrait de données
  - Accès uniquement à l'interface (pas d'accès à l'implémentation)
  - Utilisation des propriétés du type abstrait telles que définies dans la spécification.
  - (L'utilisateur est lui-même un type abstrait avec une interface et une spécification)

Comment assurer la réutilisation du logiciel?

- Mais en utilisant un type *abstrait* l'utilisateur n'en connaît pas l'implémentation
  - il sait uniquement que la spécification du type abstrait est supposée être vérifiée par l'implémentation.
- Pour la réalisation *concrète*, une implémentation particulière est choisie
- Il y a naturellement polymorphisme

Notion de contrat (Eiffel)

- Un *client* et un *vendeur*
- Un *contrat* lie le vendeur et le client (*spécification*)
- Le client ne peut utiliser l'objet que par son *interface*
- La réalisation de l'objet est cachée au client
- Le contrat est conditionné par l'utilisation correcte de l'objet (*pré-condition*)
- Sous réserve de la pré-condition le vendeur s'engage à ce que l'objet vérifie sa spécification (*post-condition*)
- Le vendeur peut déléguer: l'objet délégué doit vérifier au moins le contrat (*héritage*)

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]) <=> (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 `ListArray`
  - 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

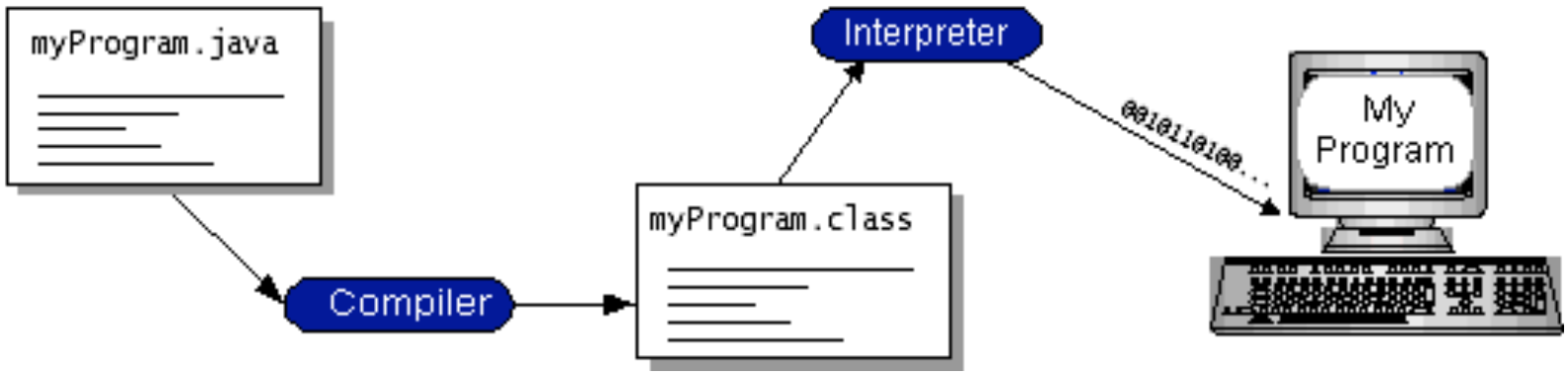
Généralités…

- 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/books/tutorial/index.html

Plateforme Java

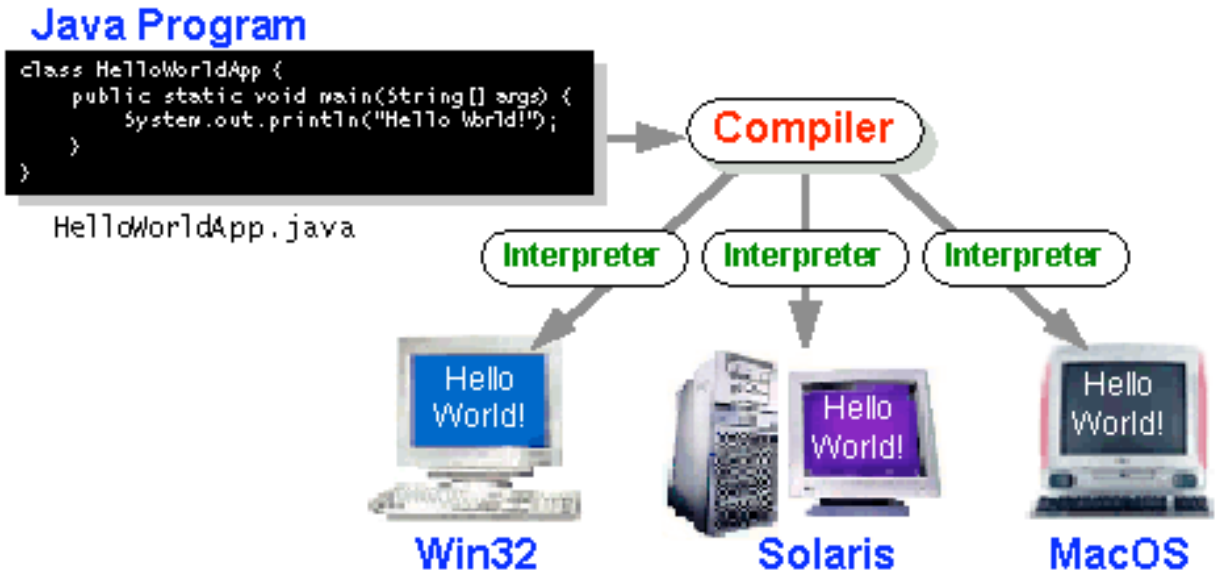
- La compilation génère un .class en bytecode (langage intermédiaire indépendant de la plateforme).
- Le bytecode est interprété par un interpréteur Java JVM

Compilation `javac`
interprétation `java`



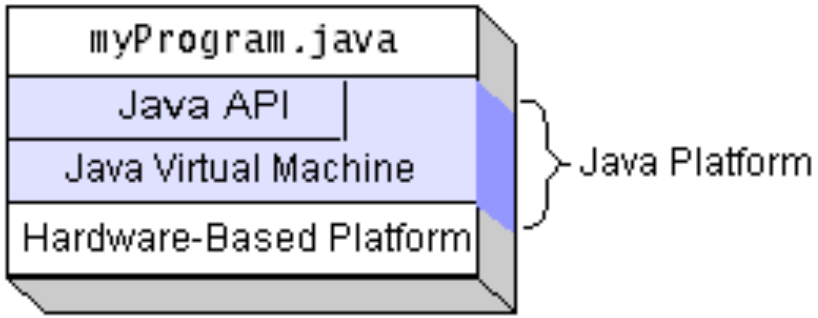
Langage intermédiaire et Interpréteur…

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



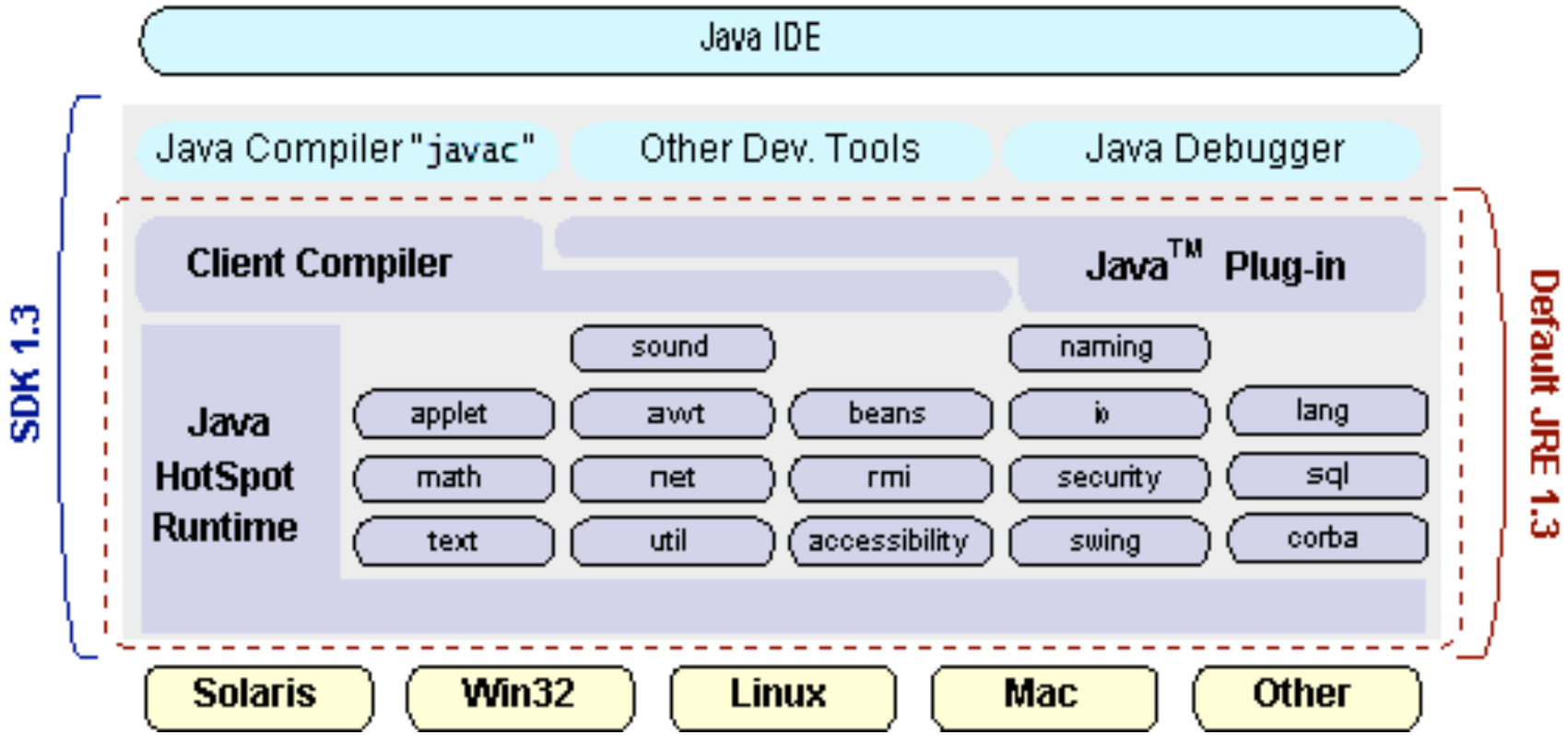
Plateforme Java

- 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, debogueurs etc…)



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 `Appli.java`
- Compilation:
  - `javac Appli.java`
- Création de `Appli.class` (bytecode)
- Interpréter le byte code:
  - `java Appli`
- 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:

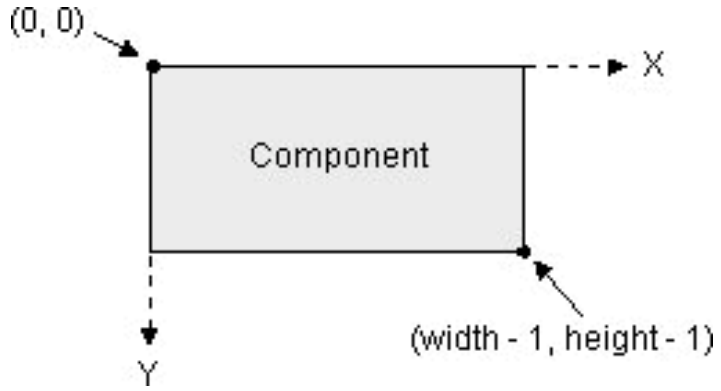
- import et package:
  - Un package est un regroupement de classes.
  - Toute classe est dans un package
  - Package par défaut (sans nom)
  - classpath
- import java.applet.*;
  - Importe le package java.applet
    - Applet est une classe de ce package,
    - Sans importation il faudrait java.applet.Applet

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:
    - page de hf
- 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 fenêtre
        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 fremeture
- 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…

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 X 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 O:");
       String s1 =sc.next(Pattern.compile("[Oo]"));
              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<->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