Programmation-orientée objet et interfaces graphiques en Java

Licence d'informatique

Hugues Fauconnier

hf@liafa.univ-paris-diderot.fr

Bibliographie

- De nombreux livres sur java (attention aux versions de java la version actuelle et la version 8)
- En ligne:
  - API: http://docs.oracle.com/javase/8/docs/api/
  - Java tutorial: http://docs.oracle.com/javase/tutorial

Chapitre I

Introduction

A) Pourquoi?

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

Comment?

B) Typage…

- Histoire
  - Fonctions et procédures (60 Fortran)
  - Typage des données

C) 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)

statique-dynamique

D) Comment la programmation objet permet d'assurer la réutilisation du logiciel?

- Spécification/ Implémentation
  - Type abstrait de données
    - définir le type par son interface (exemple empiler, dépiler pour une pile) et ses propriétés la spécification (dépiler(empiler(v)) retourne v et la pile est dans l'état précédent empiler(v))
  - des implémentations
    - l'implémentation décrit un code qui doit satisfaire le type de données

Comment la programmation objet permet d'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 la programmation objet permet d'assurer la réutilisation du logiciel?

- Mais en utilisant un type abstrait l'utilisateur n'en connaît pas l'implémentation
  - il ne peut utiliser que l'interface
  - il sait uniquement que la spécification du type abstrait est supposée être vérifiée par l'implémentation
- Bien sûr, pour la réalisation concrète, l'utilisateur choisira une implémentation particulière
- Il y a naturellement polymorphisme:
  - empiler(v) dans le code de l'utilisateur sera associé à une implémentation particulière de la méthode empiler

Réutiliser?

- l'interface et la spécification définissent le type utilisé
- on peut sans modifier le code de l'utilisateur changer l'implémentation de ce type
  - si le code change et vérifie la spécification du type de données, les propriétés du programme de l'utilisateur sont inchangées (le programme vérifie bien sa spécification)

Notion de contrat

- 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)

E) 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())

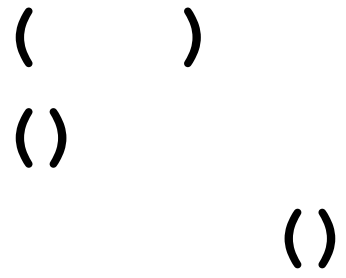
not vide(empiler(x,s))

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



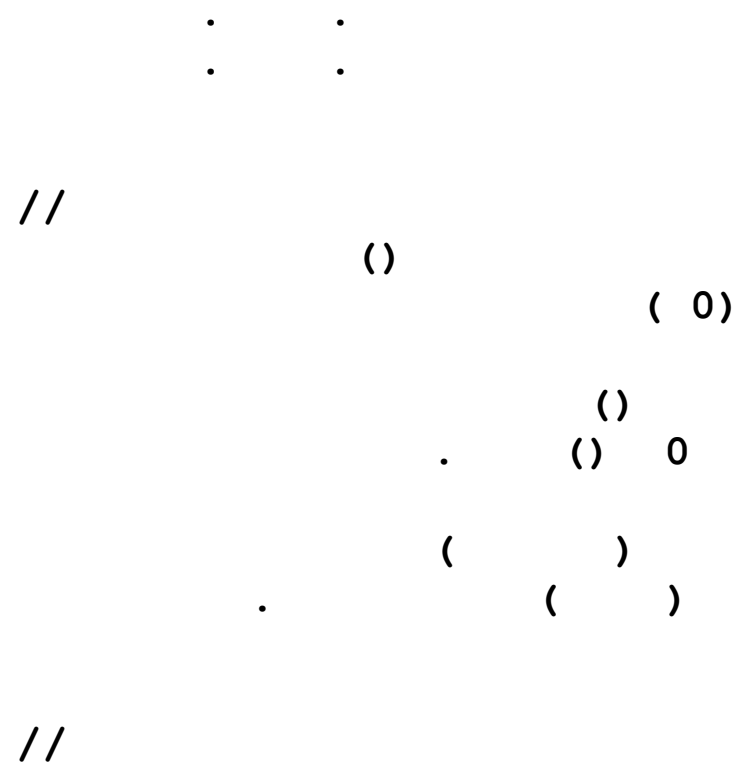
Divers

- `package`: regroupement de diverses classes
- `abstract`: signifie qu'il n'y a pas d'implémentation (on aurait aussi pu utiliser une `interface`)
- `public`: accessible de l'extérieur (par l'utilisateur de le classe)
- La classe est paramétrée par un type (T)

Implémentations

- On va implémenter la pile:
  - Comme pile sur un type `T`
    - 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`
  - Comme pile de `Integer` avec un tableau de `Integer`

Une implémentation

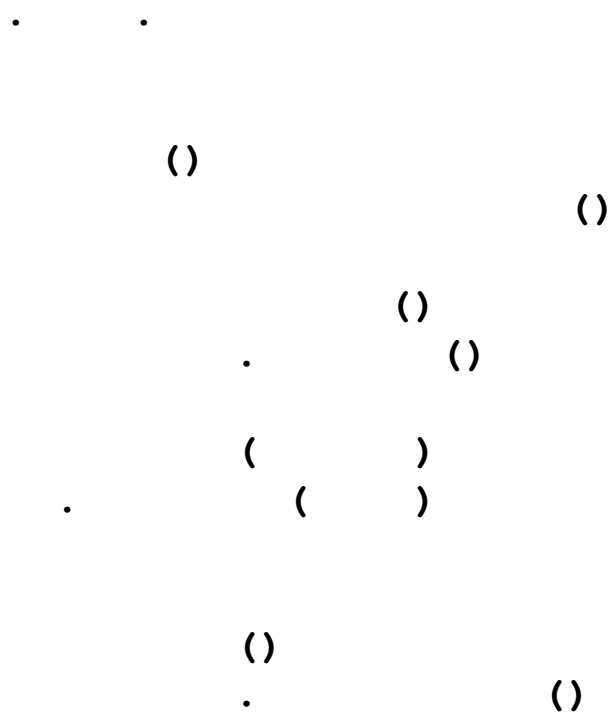


Suite

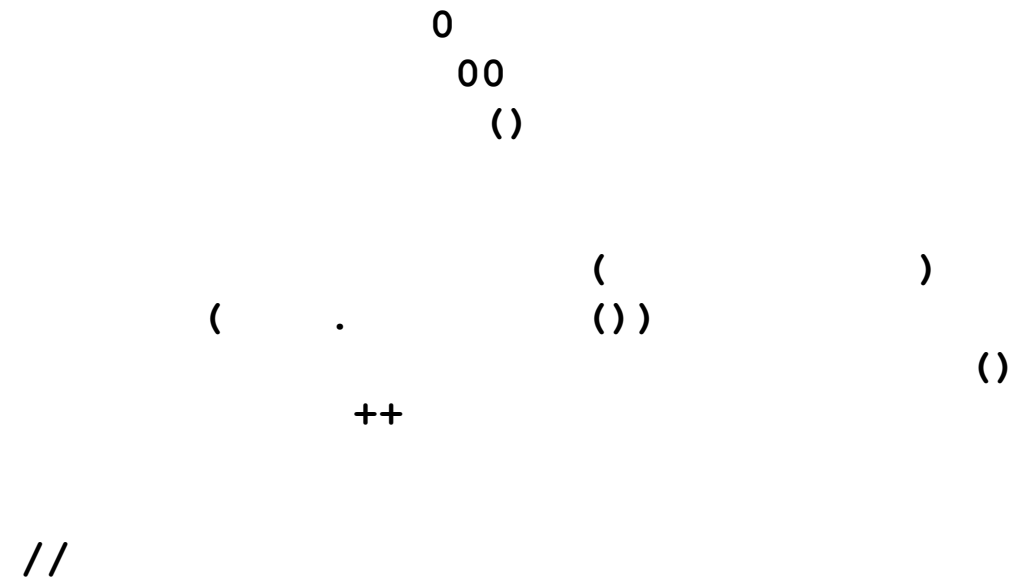
```
//
                                    ()
                               .    ()

                    (        0)
                                                 ()
                          .            (     -   )
                       .               (     -   )
```

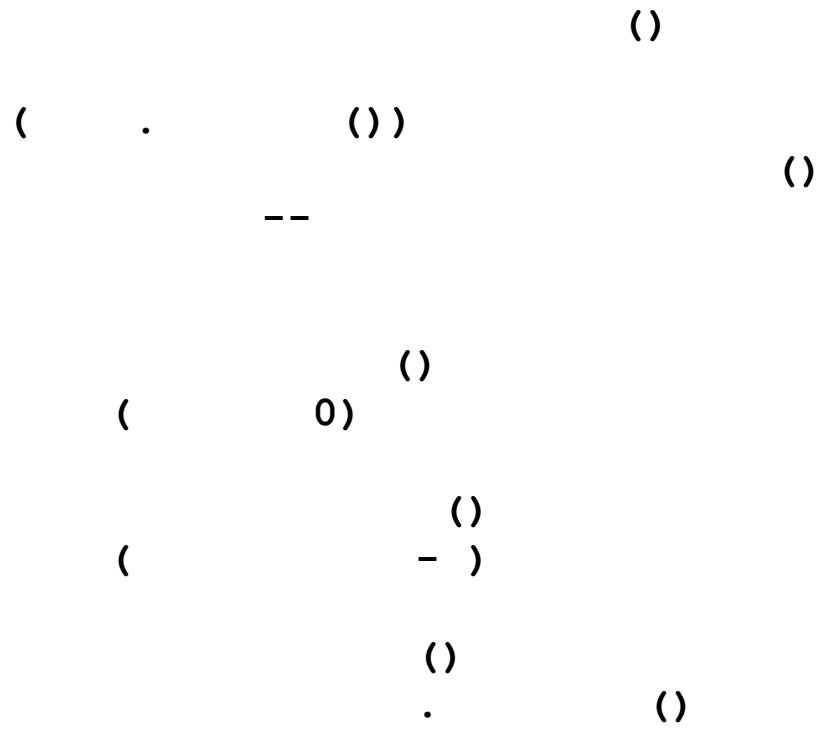
Autre implémentation avec listes



Une pile de Integer



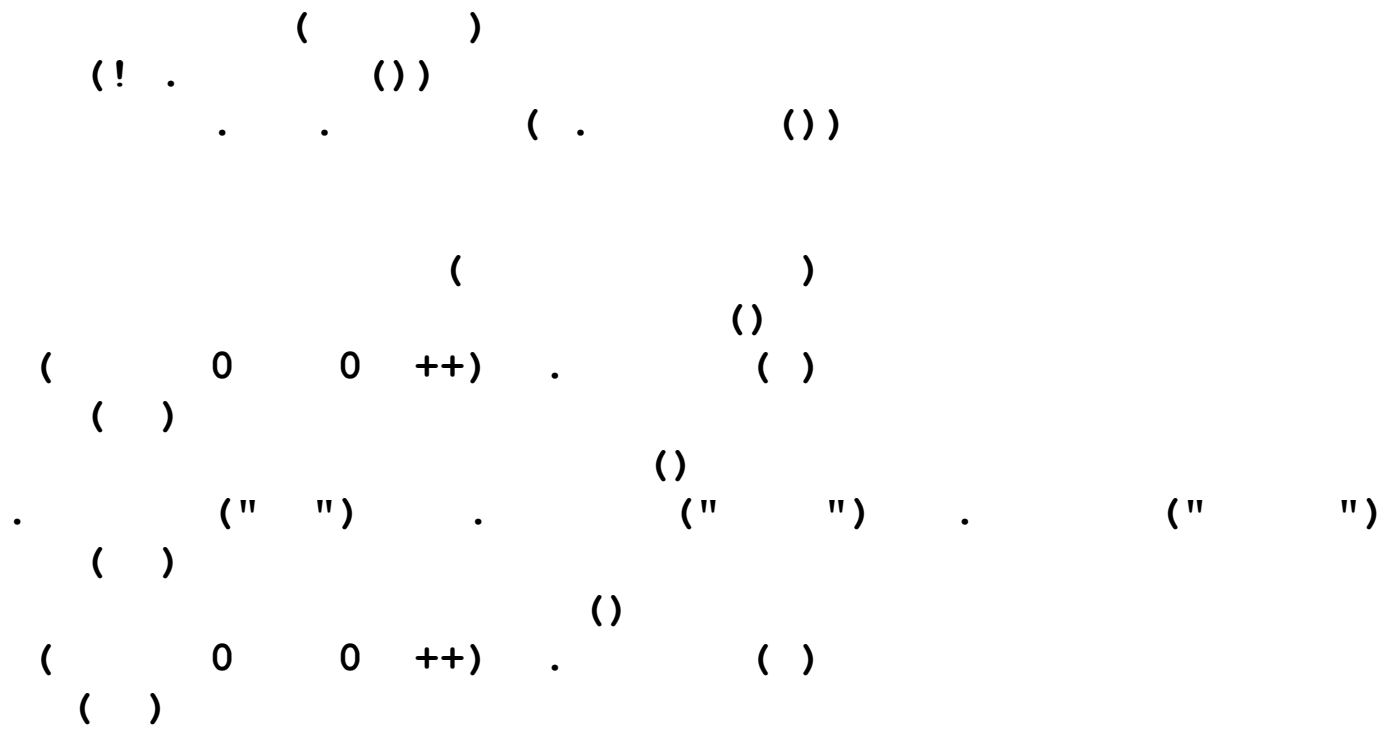
Suite…



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 pouvoir s'appliquer à toute implémentation de Pile

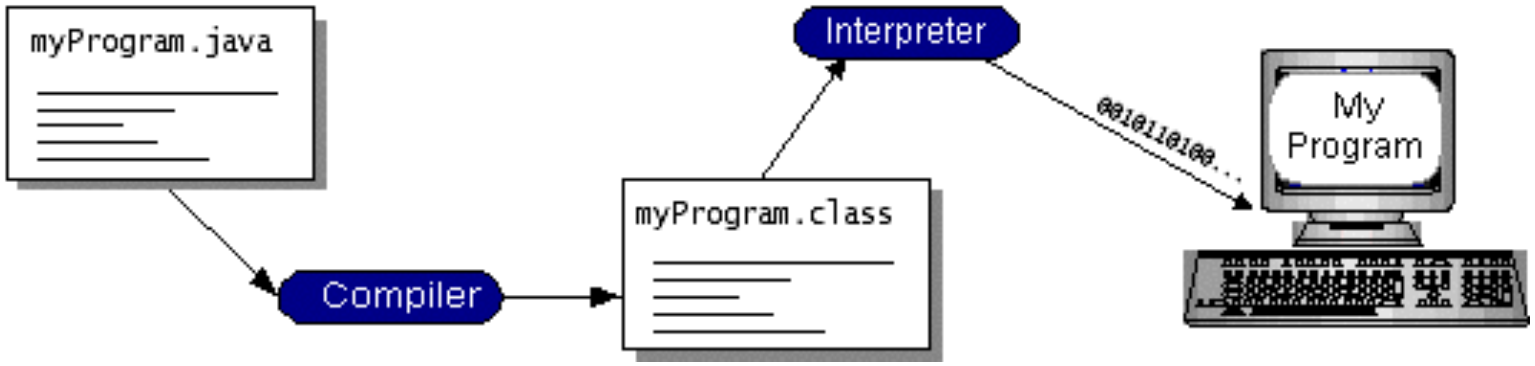
Un main



E) java

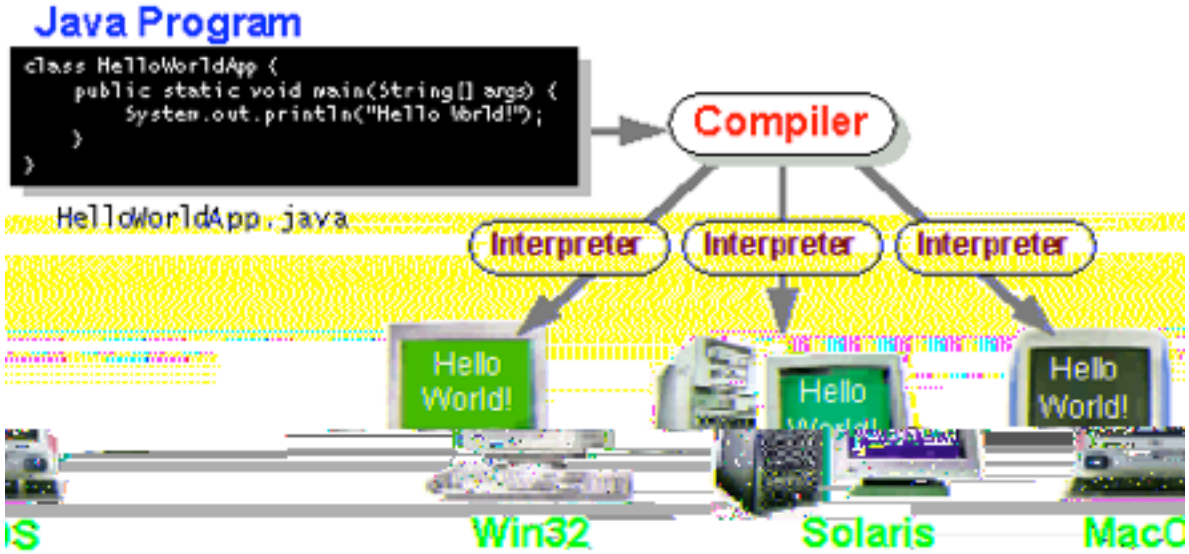
- La compilation génère un fichier `.class` en « bytecode » (langage intermédiaire indépendant de la plateforme).
- Le bytecode est interprété par un interpréteur Java JVM

Compilation commande `javac`
Interprétation commande `java`



Langage intermédiaire et Interpréteur…

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

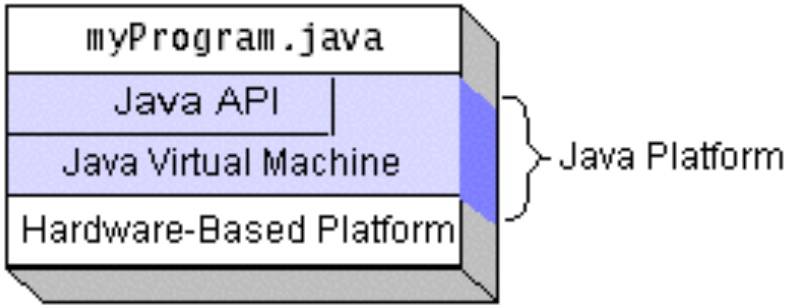


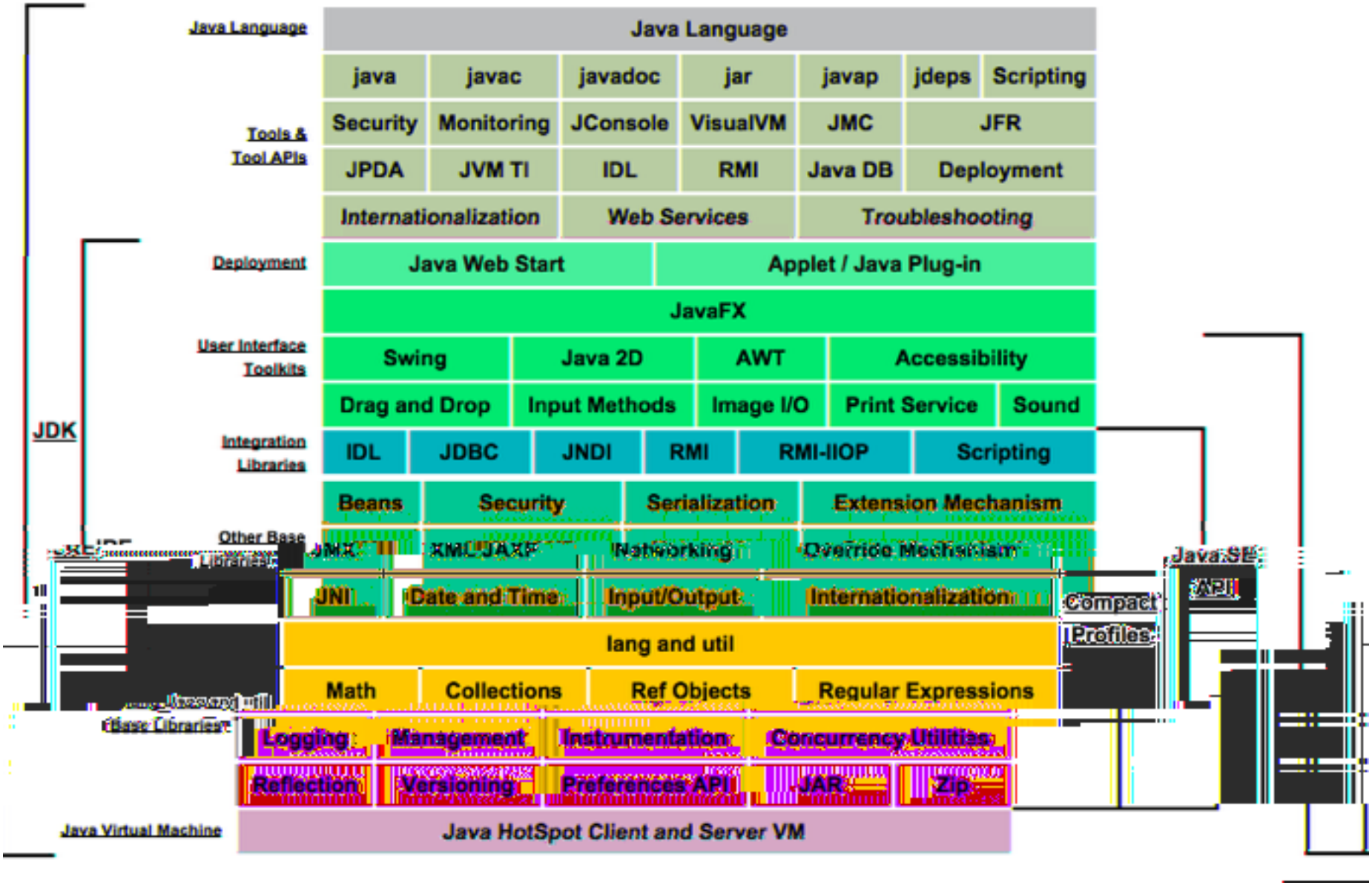
Généralités…

- Un peu plus qu'un langage de programmation:
  - "gratuit"! (licence GPL)
  - Indépendant de la plateforme
  - Langage interprété et byte code
  - 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, …
  - Dernière version Java SE 7 (GPL)
  - Site:http://www.java.com/fr

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):




Java Language
Java Language
java
javac
javadoc
jar
javap
jdeps
Scripting
Tools & Tool APIs
Security
Monitoring
JConsole
VisualVM
JMC
JFR
JPDA
JVM TI
IDL
RMI
Java DB
Deployment
Internationalization
Web Services
Troubleshooting
Deployment
Java Web Start
Applet / Java Plug-in
JavaFX
User Interface Toolkits
Swing
Java 2D
AWT
Accessibility
Drag and Drop
Input Methods
Image I/O
Print Service
Sound
JDK
Integration Libraries
IDL
JDBC
JNDI
RMI
RMI-IIOP
Scripting
Beans
Security
Serialization
Extension Mechanism
Other Base Libraries
JMX
XML JAXP
Networking
Override Mechanism
JNI
Date and Time
Input/Output
Internationalization
Java SE
Compact Profiles
API
lang and util
Math
Collections
Ref Objects
Regular Expressions
Base Libraries
Logging
Management
Instrumentation
Concurrency Utilities
Reflection
Versioning
Preferences API
JAR
Zip
Java Virtual Machine
Java HotSpot Client and Server VM

Créer une application

- Fichier `Appli.java`:

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

- Compilation: `javac Appli.java`
- Crée de `Appli.class` (bytecode)
- Interpréter le bytecode: `java Appli`
- Si: `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 du programme: l'interpréteur java exécute le code de main

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…

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

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 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