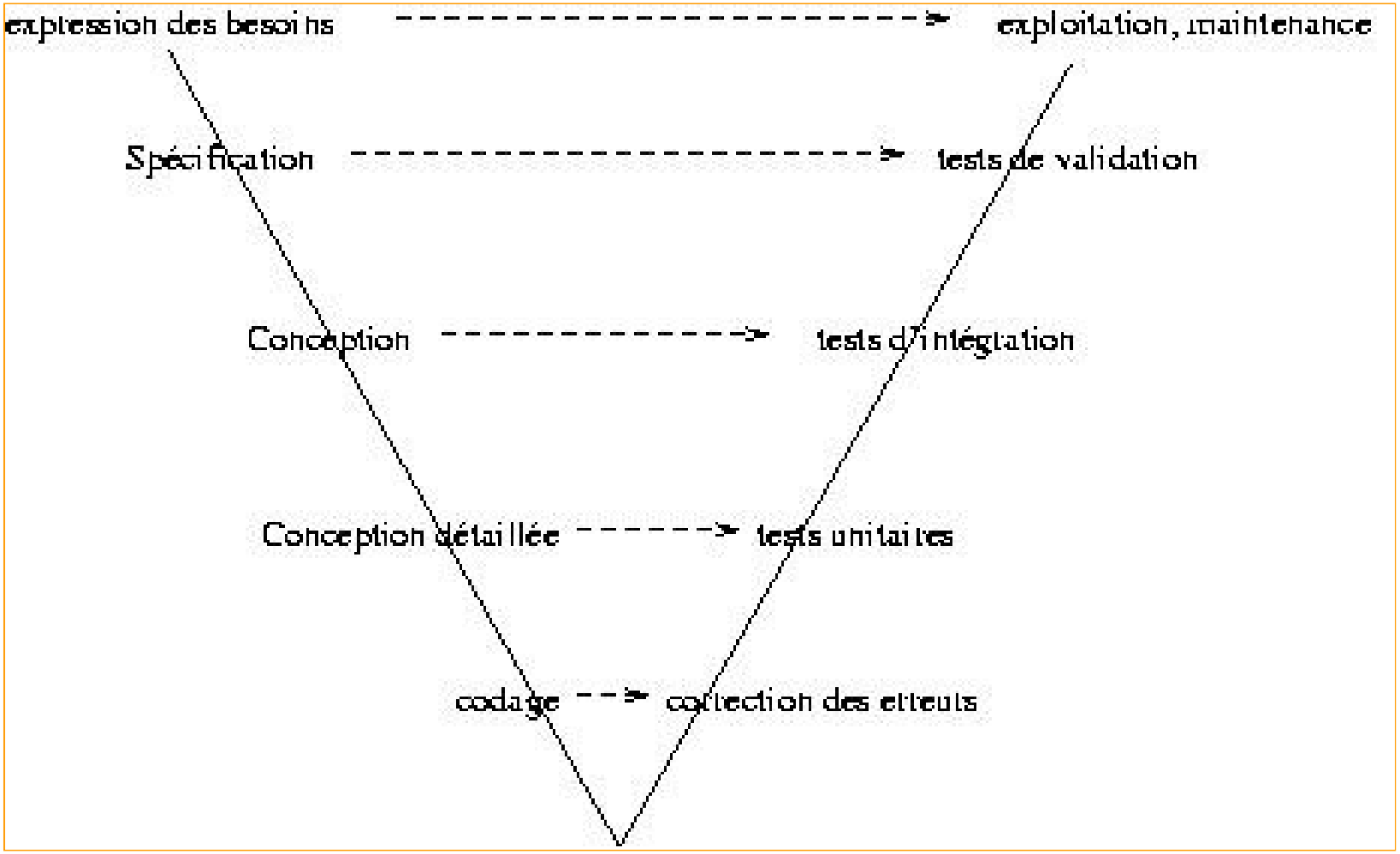
Chapitre 1

Premiers pas

Développement d'application

- Transformation progressive d'une suite de modèles :
  - du 1er modèle, description des besoins du client
  - au dernier, programme complètement testé
- Etapes :
  - analyse, conception, implémentation, tests
- Analyse : spécification des fonctionnalités du système
- Conception : architecture en terme de composants
- Tests : unitaires et d'intégration

Cycle de vie du logiciel



Les étapes du cycle de vie
- Analyse -

- Dans cette étape, on s'attache à répondre à trois grandes questions :
  - comprendre le problème, identifier les données et les résultats attendus
  - dégager les grandes fonctionnalités du système (spécification fonctionnelle)
  - identifier les ressources nécessaires (matérielles et humaines)

- Cette phase est indépendante de tout langage d'implantation.

Les étapes du cycle de vie
- Conception -

- On décompose le problème en sous-problèmes plus simples :
  - Ceci donne naissance à un ensemble d'unités informatiques (composants) qui constituent le logiciel d'application.
  - C'est dans cette phase que l'architecture du logiciel est élaborée. Les composants (modules ou unités) et leurs relations sont spécifiés.
- Il s'agit d'un processus itératif qui peut conduire à un retour vers l'analyse.
- Java peut être utilisé comme langage de conception.

Les étapes du cycle de vie
- Codage -

- Il s'agit, dans cette étape, d'implanter (coder) la conception, c'est à dire l'ensemble des composants de l'application.

- Plus précisément, il s'agit de traduire les traitements en terme de structures de contrôle et les données en termes des structures de données du LP

- On utilise un langage de programmation (Ada, C, Java, C++, Eiffel, Cobol, Fortran, Pascal, ...) pour exprimer le code.

Les étapes du cycle de vie
- validation/vérification -

- ✓ tests unitaires (niveau composant) :
  - chaque composant fait, individuellement, l'objet de tests
- ✓ tests d'intégration (niveau système)

- L'évolution du système est contrôlée à travers le processus conception/codage/test.

Crise du logiciel

- complexité croissante
- coût
- qualité

- En 1976, une étude met en lumière le coût prohibitif du logiciel dû à la complexité croissante :

  - 75 $ par instruction développée
  - 4000$ par instruction modifiée (après livraison)

Une étude (1)

- Un autre étude menée en 1979 par le gouvernement américain et portant sur 487 projets illustre le coût exorbitant de la maintenance :

- 41,8% est dû à un changement de spécification
- 17,4% est dû à un changement dans le format des données
- 12,4% à des sorties d'urgence
- 9% à des défaillances nécessitant un débogage

Une étude (2)

La même étude indique que :

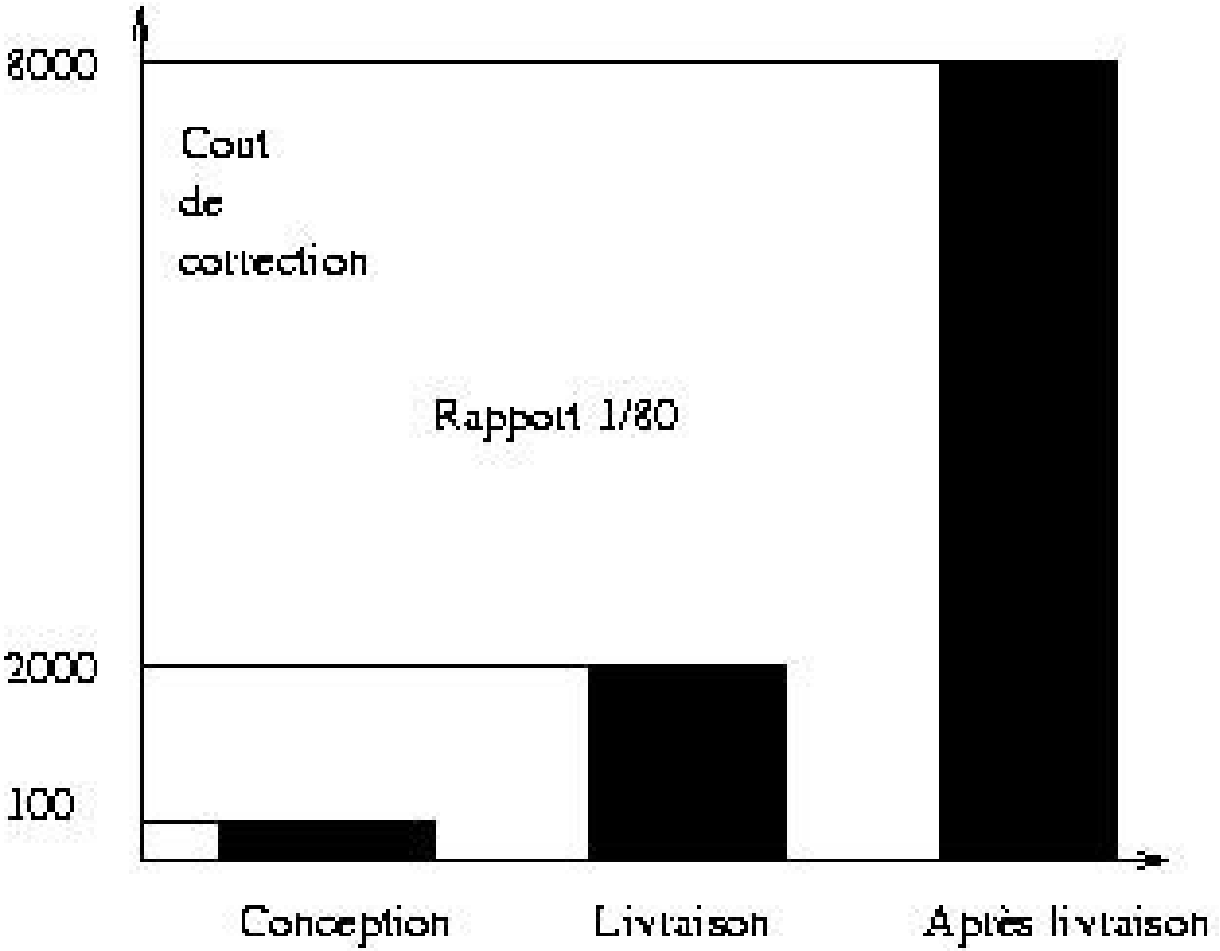
- 47% des applications délivrées ne sont jamais utilisées
- 29% payées mais non terminées
- 19% abandonnées ou réécrites
- 3% utilisées après modifications
- 2% utilisées sans changement

Une autre étude

Une enquête effectuée aux USA en 1986 auprès de 55 entreprises révèle que 53% du budget total d'un logiciel est affecté à la maintenance :

- 34% à la maintenance évolutive : modification des spécifications initiales
- 10% à la maintenance adaptative : nouvel environnement, nouveaux utilisateurs ;
- 17% à la maintenance corrective : correction des bogues ;
- 16% à la maintenance perfective : améliorer les performances sans changer les spécifications ;
- 6% à l'assistance aux utilisateurs ;
- 6% au contrôle qualité ;
- 7% l'organisation et au suivi ;
- 4% divers.

Qualités d'un logiciel

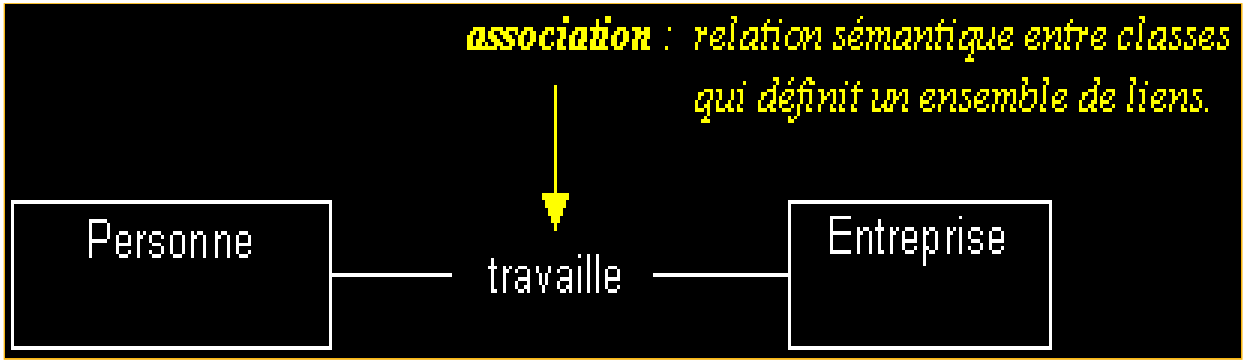


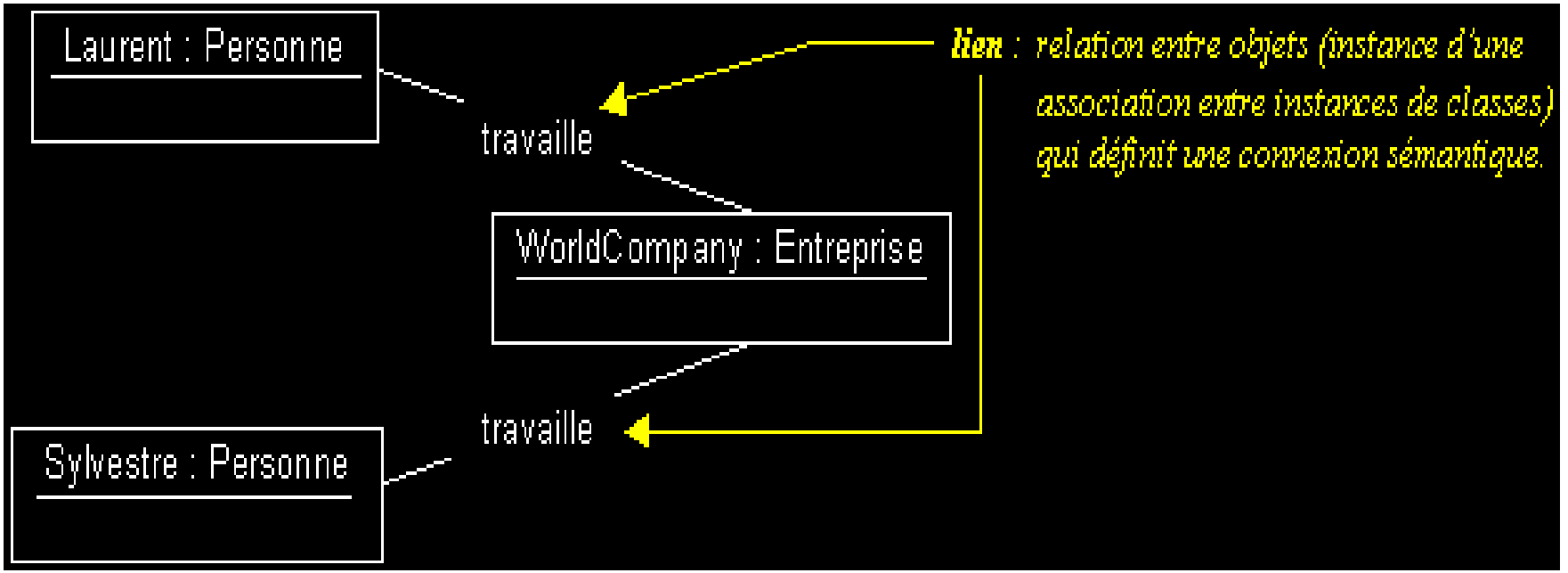
coût de la non qualité

Développement de composants logiciels

- Développement
  - Spécification => UML, interfaces
  - Implantation => Java
- Développement par spécialisation
  - Héritage
- Qualités attendues
  - Réutilisabilité
  - Modularité
  - Extensibilité

UML : Diagrammes de classe





Spécification d'un composant

```
interface IRectangle {
 static final int XMAX=800, YMAX=600;
 /**
  * sémantique
  * @param x abscisse de la position finale
  * @param y ordonnée de la position finale
  * preconditions
  *    deplacer(x,y) => 0<=x<=XMAX et 0<=y<=YMAX
  */
 void deplacer(int x, int y);
 /**
  * sémantique
  * @return la surface du rectangle
  */
 int surface();
}
```

Implantation d'un composant

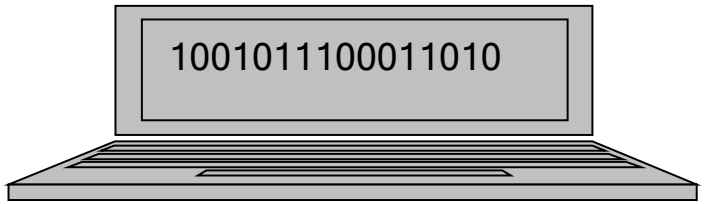
```
public class Rectangle implements IRectangle{
 public int largeur = 0; public int hauteur = 0;
 public Point origine;
 public Rectangle() {
      origine = new Point(0, 0); }
 public Rectangle(Point p) {
      origine = p; }
 public Rectangle(int w, int h){
      this(new Point(0, 0), w, h);}
 public Rectangle(Point p,int w,int h)
            { origine = p; largeur = w;hauteur = h; }
 public void deplacer(int x, int y){
      origine.x=x; origine.y=y;}
 public int surface() {
      return largeur * hauteur; }
}
```

```
public class CreateObjectDemo {

 public static void main(String[] args) {
  Point origine_1  = new Point(23, 94);
  Rectangle rect_1 = new Rectangle(origine_1, 100, 200);
  Rectangle rect_2 = new Rectangle(50, 100);
  System.out.println("Largeur of rect_1: "+ rect_1.largeur);
  System.out.println("Hauteur of rect_1: "+ rect_1.hauteur);
  System.out.println("Surface of rect_1: "+ rect_1.surface());
  rect_2.origine = origine_1;
  System.out.println("X Position of rect_2: "+ rect_2.origine.x);
  System.out.println("Y Position of rect_2: "+ rect_2.origine.y);
  rect_2.deplacer(40, 72);

  System.out.println("X Position of rect_2: "+ rect_2.origine.x);

  System.out.println("Y Position of rect_2: "+ rect_2.origine.y);       }

}
```

Langages et programmes (1/3)

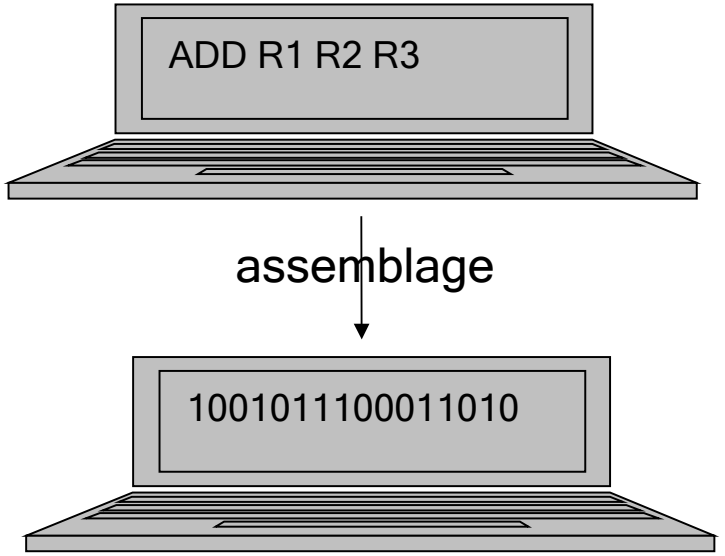
langage machine/
langage natif

ensemble d'instructions
primitives intégrées à une machine
instructions sous forme binaire



langage d'assemblage

langage bas niveau
instructions machine symboliques
→
nécessité d'une traduction par un assembleur

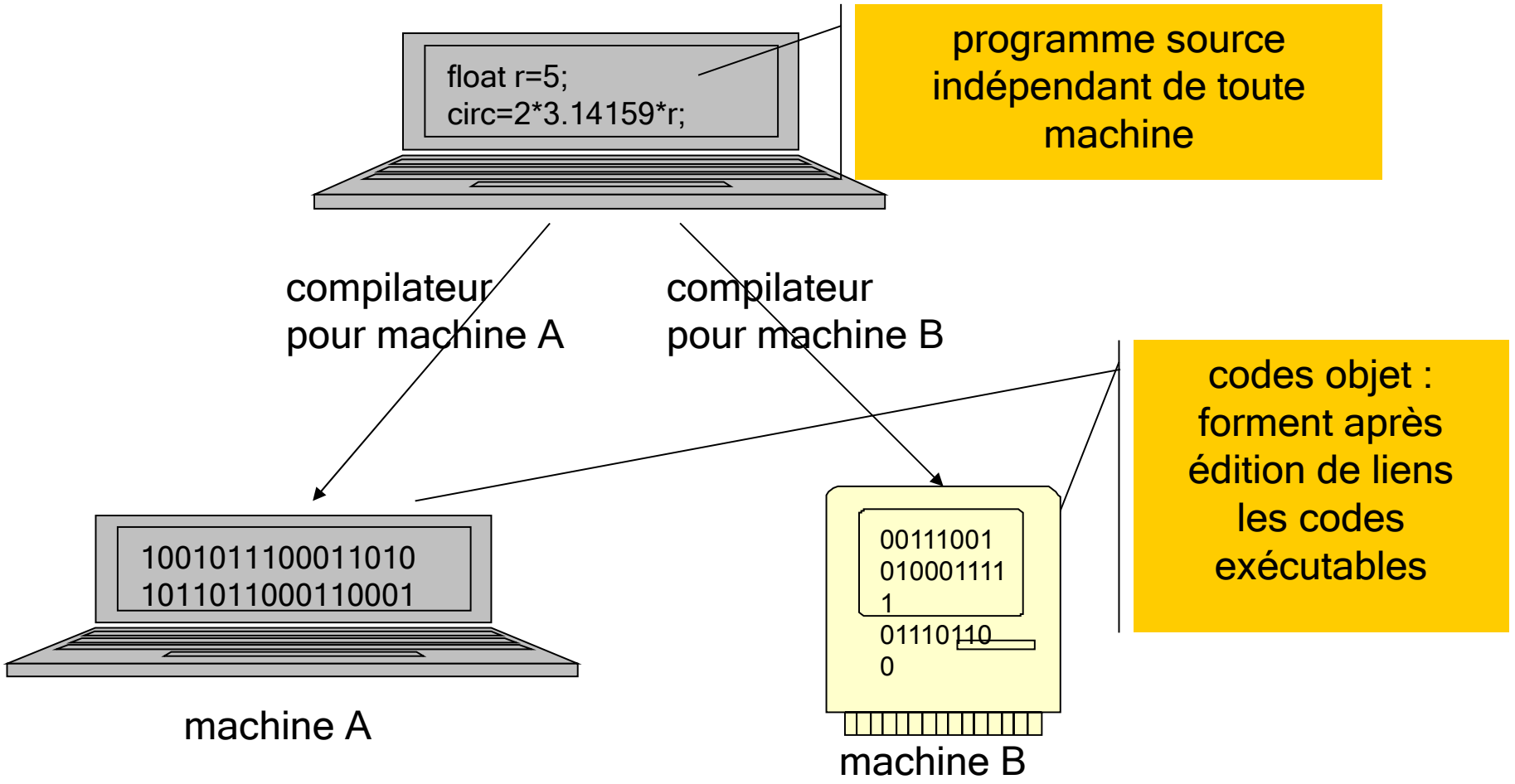


langage dépendant du processeur

Langages et programmes (2/3)

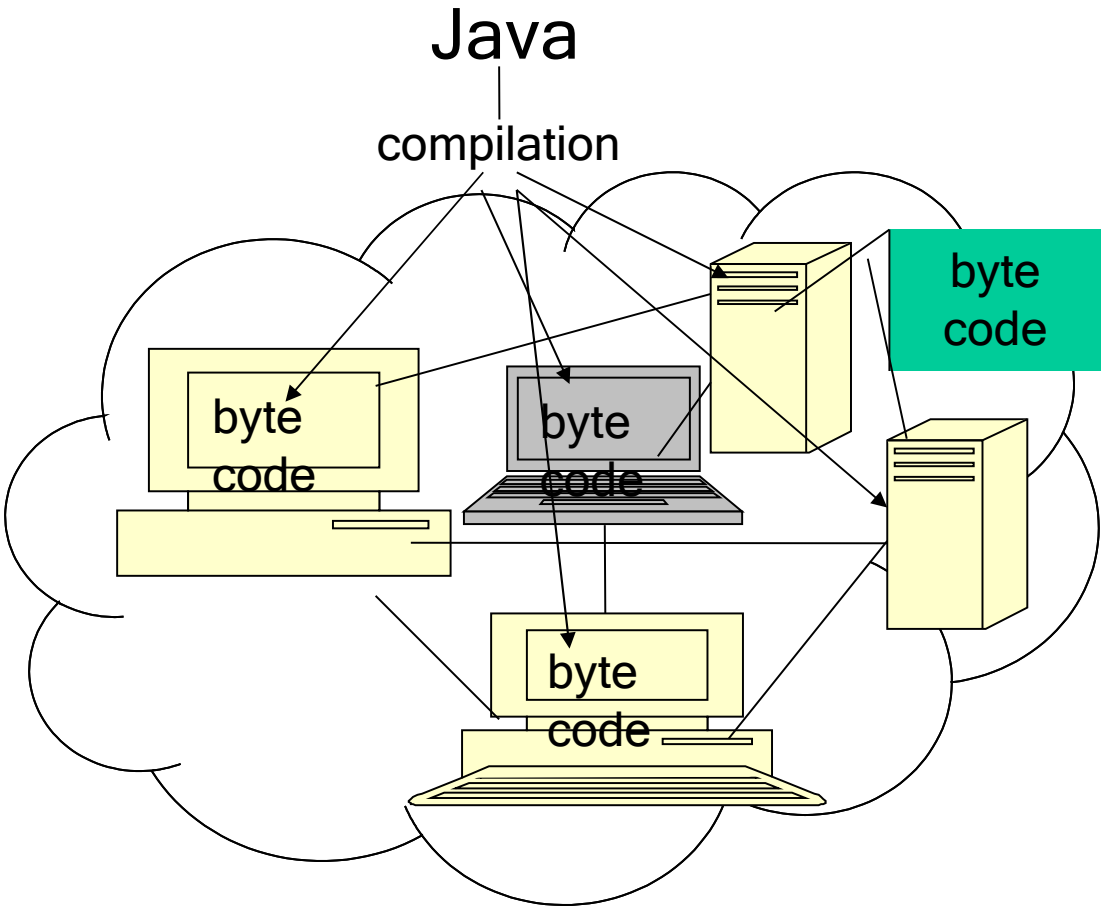
langage de haut niveau

plus proche du langage naturel



Langages et programmes (3/3)

Machines en réseau => créer des programmes exécutables sur n'importe quelle plate-forme sans recompilation



Choix de Java

- Multithreading
- Web et Applettes
- Programmation évènementielle
- Programmation réseau type client-serveur

Caractéristiques de Java

- C'est un langage simple qui hérite des constructions de C et C++
- C'est un langage **orienté objet** qui permet la conception et la réalisation d'applications complexes avec une architecture modulaire.
- C'est un langage **robuste** muni d'un mécanisme de gestion des exceptions qui permet de déceler et de traiter des erreurs pendant l'exécution du programme.
- Java est **indépendant de la plate forme** d'exécution, donc indépendant de la machine et de son système d'exploitation.
- Java est un langage **distribué**. Un programme peut être déployé sur plusieurs machines d'un réseau d'ordinateurs et exécuté sur celui-ci.
- C'est un langage **sûr**. Il possède des caractéristiques qui permettent une protection contre du code non fiable (virus). Il impose des contraintes aux applications Web dès que le téléchargement dans un navigateur est effectué.

Java API

- Application Program Interface (API) est un ensemble de classes et interfaces prédéfinies
- 3 éditions d'API :
  - J2SE pour le développement d'applications coté client ou d'applettes
  - J2EE pour le développement d'applications coté serveur
  - J2ME développements pour mobiles

Outils de développement

Ces outils fournissent un environnement de développement intégré dans une interface graphique (Integrated Development Environment)

Principaux IDE :

- JBuilder (Borland)
- NetBeans Open Source (Sun)
- Sun One (version commerciale de NetBeans)
- Eclipse Open Source (IBM)

Environnement Java

- J2SE (Java 2 Standard Edition) disponible sur java.sun.com (actuellement J2SE 6)
- comprend un JDK (Java Development Toolkit), ensemble de programmes (javac, java, javadoc, jar, appletviewer, ...) invocables à partir d'une ligne de commande
- Rassemble :
  - Environnement d'exécution
  - Langage
  - Application Programming Interfaces
  - Bibliothèques

Compilation

Un compilateur traduit un programme source en langage machine

Inconvénient : le résultat dépend de l'ordinateur visé

L'idée de java est de placer un intermédiaire dans la phase de compilation

Au lieu de compiler dans un langage machine spécifique à un ordinateur particulier, le compilateur Java transforme le programme source en un programme dans le langage machine de la machine virtuelle Java (JVM)

Ce langage est un langage de bas niveau appelé **`bytecode.`**

Il est indépendant de tout ordinateur

Interprétation

Avec un compilateur normal, le programme est exécuté directement par l'ordinateur. Le processeur lit les instructions et les exécute.

En Java, le programme compilé n'est pas directement compréhensible par un ordinateur.

Un programme, appelé interpréteur, traduit au vol (pendant l'exécution même du programme) le bytecode en instructions pour l'ordinateur.

Compilation vs interprétation

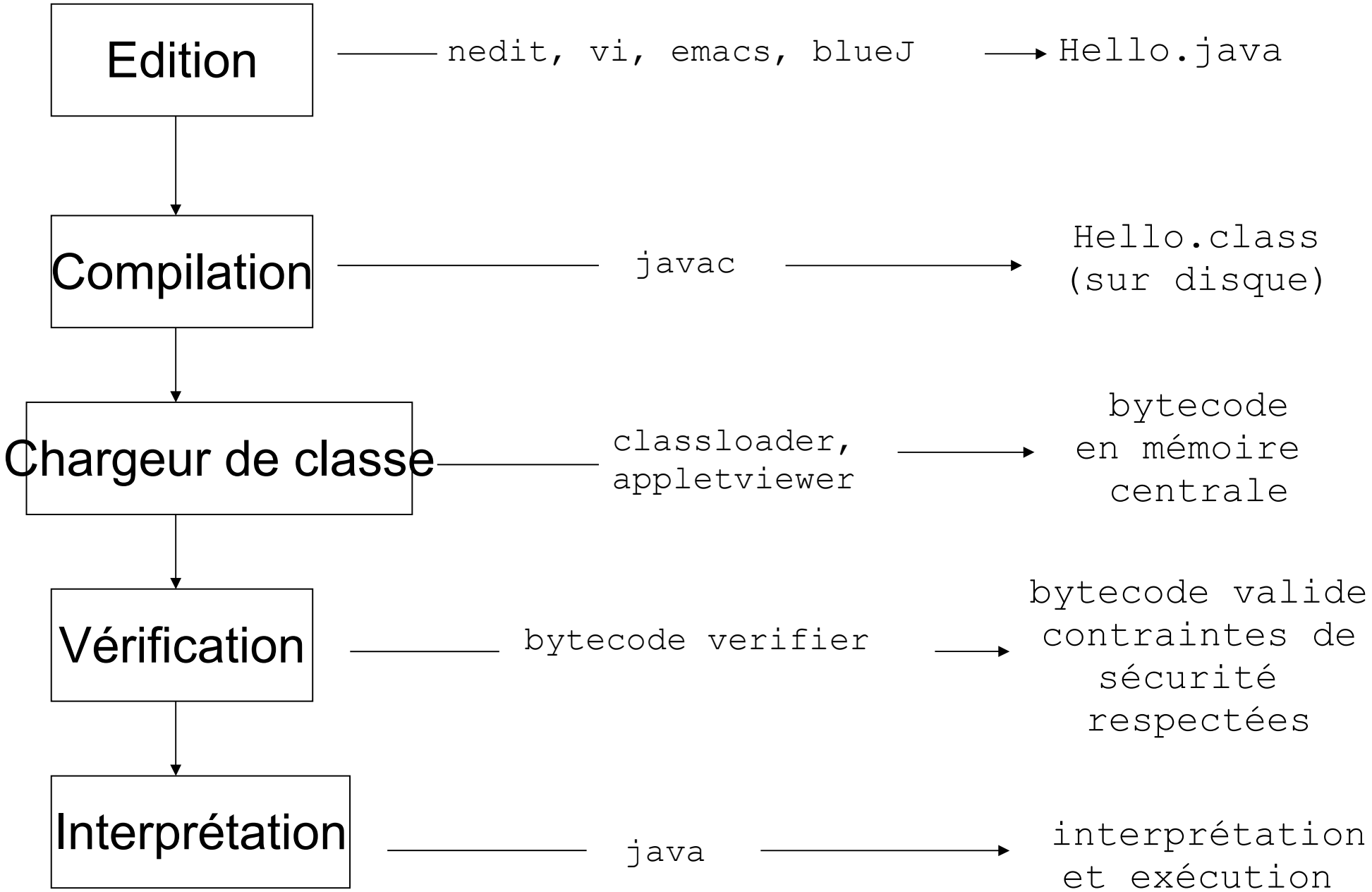
Chaque interprétation produit sa traduction. Le programme produit par l'interpréteur n'est pas conservé.

Le programme produit par le compilateur est conservé

L'interprétation est moins efficace (en temps) que la compilation

Un programme Java compilé peut être interprété sur tout ordinateur (la JVM est partout présente)

Pour exécuter un programme écrit dans un autre langage, il faut le recompiler


Edition
nedit, vi, emacs, blueJ
Hello.java
Compilation
javac
Hello.class
(sur disque)
Chargeur de classe
classloader,
appletviewer
bytecode
en mémoire
centrale
Vérification
bytecode verifier
bytecode valide
contraintes de
sécurité
respectées
Interprétation
java
interprétation
et exécution
Du code

Un premier programme

Un programme Java est constitué de classes, au moins une appelée classe principale.

Une des classes doit contenir une méthode `main` chargée de recevoir les arguments de la commande de lancement du programme.

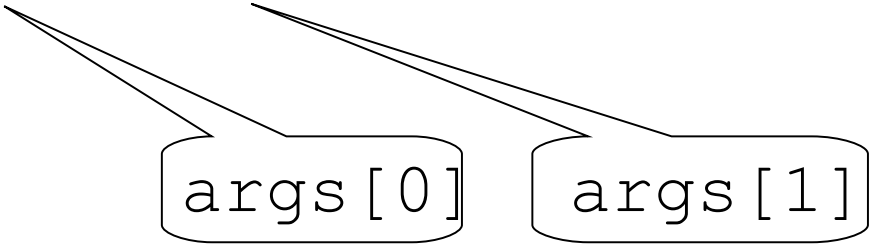
Le traditionnel premier programme est enregistré dans un fichier `Hello.java` contenant le code source suivant :

```
public class Hello{
  public static void main( String[] args ){
       System.out.println( "Hello!"+args[0]+args[1]+2007 );
  }
}
```

Production de programme

```
javac Hello.java

java Hello nouvelle année
```

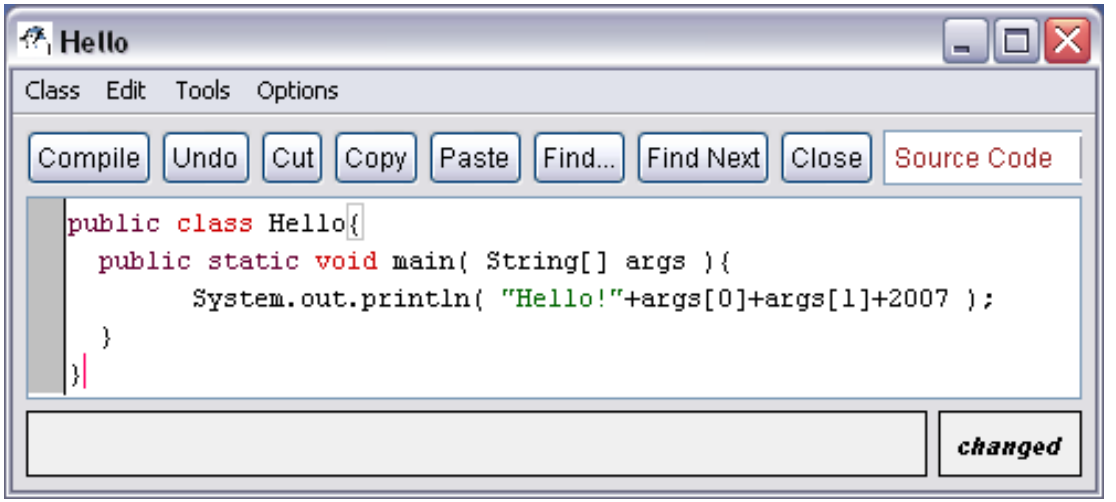
résultat affiché




```
Hello!nouvelleannée2007
```

Sous bluej

édition du programme



exécution du programme

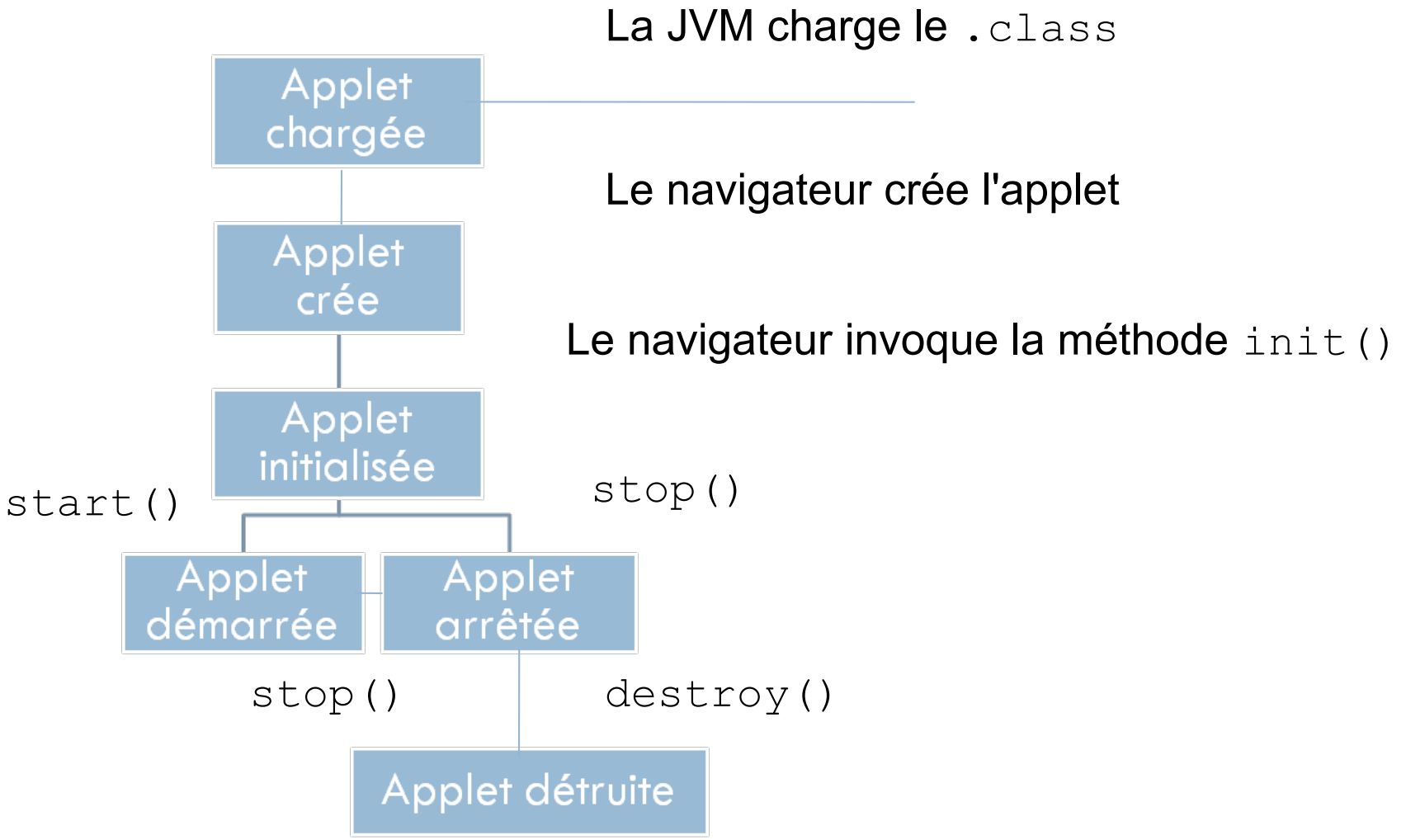
résultat

Autre version

```
import javax.swing.JOptionPane;
public class Hello{
  public static void main( String[] args ){
       JOptionPane.showMessageDialog(null,
                              "Hello!"+args[0]+args[1]+2007 );
  }
}
```


```
javac Hello.java
java Hello nouvelle année
```

Applet : flot de contrôle



Applet : Hello.java

```
import java.awt.*;
import javax.swing.*;
public class Hello extends JApplet{
  String msg=null;
  public void init(){
    msg=JoptionPane.showInputDialog("Votre message");
  }
  public void paint(Graphics g){
    super.paint(g);
    g.drawString(msg+2009,25,25 );
  }
  public void start(){}
  public void stop(){}
  public void destroy(){}
}
```

Applet : Hello.html

```
<html>
<head> applet Hello world</head>
<body>
<applet code = "Hello.class"
                width = 200 height = 300>
</applet>
</body>
</html>
```

Applet : Hello.html

```
C: > appletviewer  Hello.html
```

Lecture à partir du clavier

Lire un fichier ou lire les données tapées sur un clavier sont deux opérations équivalentes.
On importe la classe `java.util.Scanner` :

- lecture à partir clavier

```
Scanner in = new Scanner( System.in );
String s = in.next();
```

ou bien

```
int i = in.nextInt();
```

ou encore

```
boolean b = in.nextBoolean();
double d = in.nextDouble();
```

Quelques sites utiles

- Bibliographie
  - ✓ Programmer en Java, C.Delannoy, Eyrolles
  - ✓ Introduction to Java programming, Y.Daniel Liang, Pearson Prentice Hall
  - ✓ Java how to program Deitel&Deitel, Prentice hall
  - ✓ Java by Dissection, Ira Pohl & Charlie McDowell, Addison Wesley

- Compilateurs, bibliothèques, tutoriels, documentation

  http://java.sun.com

- IDE

  http://www.bluej.org

  http://www.eclipse.org

  http://www.netbeans.org