UNIVERITÉ HASSAN II
DE CASABLANCA
ECOLE SUPÉRIEURE DE TECHNOLOGIE

PROGRAMMATION ORIENTÉE OBJET AVEC JAVA

PRÉPARÉ PAR :
LARBI HASSOUNI

Sources principales :

Sites :

1. **www.oracle.com (le plus complet et le plus à jour : c'est la principale source)**
2. **http://deptinfo.unice.fr/~grin/**

Livres :

1. **Au cœur de Java 2 : Notions fondamentales (Cay S.Horstmann & Gary Cornell)**
2. **Au cœur de Java2 : Foctions avancées(Cay S.Horstmann & Gary Cornell)**
3. **BIG Java (Cay S.Horstmann)**
4. **Java How to program (Deitel)**
5. **Beginning Java 2 (Ivor Horton)**
6. **Practical Database Programming with Java (Ying Bai)**
7. **Data Structures & Algorithms in JAVA (MICHAEL T. GOODRICH, ROBERTO TAMASSIA)**
8. **Objetc-Oriented Data Structures Using Java (NELL DALE, DANIEL T.JOYCE, CHIP WEEM)**
9. **Conception et programmation orientée objet (Bertrand Meyer)**
10. **Java En Concentré (David Flanagan)**

Technologie JAVA

• **La technologie Java est développée par SUN Microsystems™ en 1995 et comporte principalement de trois composantes:**

- ✓ Un langage de programmation
- ✓ Une plateforme , environnement logiciel dans lequel les programmes java s'exécutent.
- ✓ Une API (Application Programming Interface)

• La technologie Java est utilisée **dans de nombreux domaines d'application et en particulier dans:**

- ✓ **Les serveurs d'applications (Java EE)**
- ✓ **Les téléphones portables**
- ✓ **Les cartes à puces(JME)**

Exemple de programme en JAVA

Le code de la classe doit être enregistré dans un fichier de même nom (casse comprise) que la classe

Tout code java doit être défini à l'intérieur d'une classe

HelloWorld.java

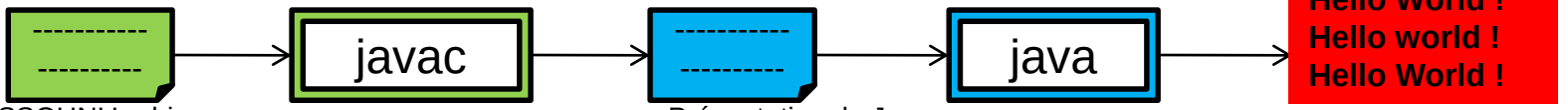
```
public class HelloWorld {

        public static void main(String[ ] args) {

          for(int i = 1; i <= 5; i++) {
              if(i%2 == 1){
                    System.out.println("Hello World !");
              }
          }
          }
}
```

La description de la classe est effectuée à l'intérieur d'un bloc { }

Le point d'entrée pour l'exécution est la méthode main()

Compilation : javac HelloWorld.java

Exécution : java HelloWorld.



HelloWorld.java

HelloWorld.class

Compilation d’un code source

♠ Un code source ne peut pas être exécuté directement par un ordinateur

♠ Il faut traduire ce code source dans un langage que l’ordinateur (le processeur de l'ordinateur) peut comprendre (langage *natif)*

♠ Un compilateur est un programme qui effectue cette traduction

Compilation en Java → *bytecode*

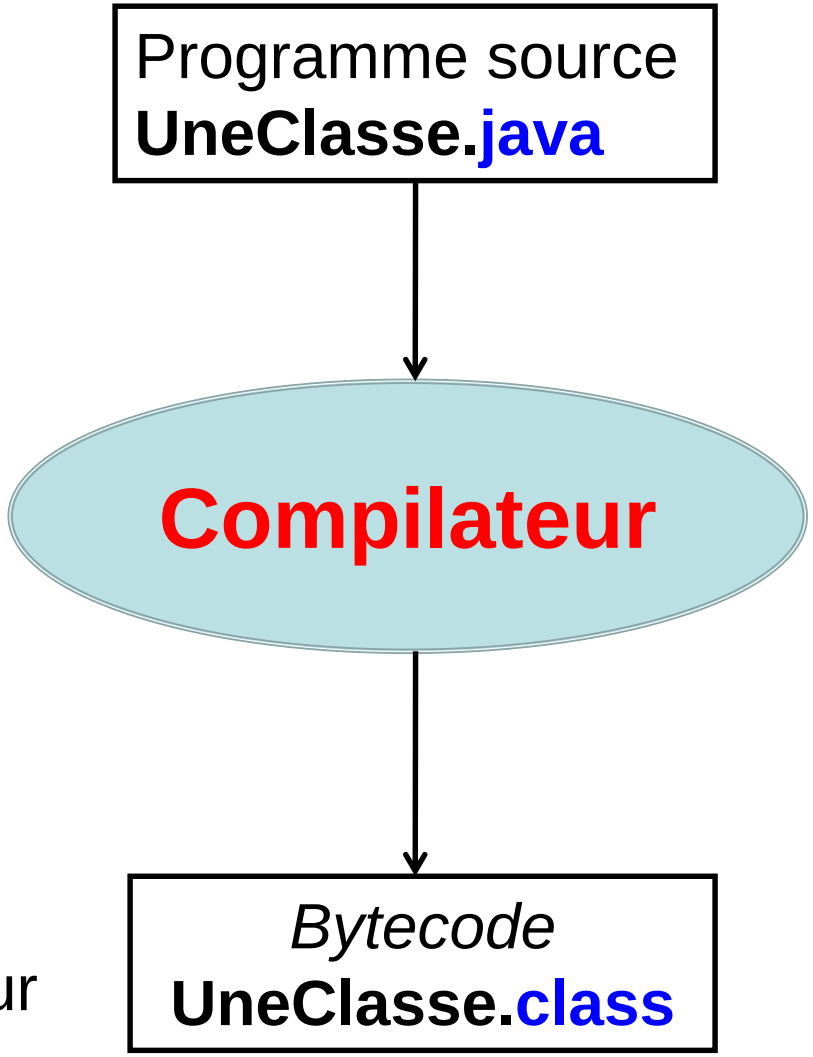
♠ En Java, le code source n'est pas traduit directement dans le langage de l'ordinateur

♠ Il est d'abord traduit dans un langage appelé «*bytecode», langage d'une machine virtuelle* (JVM ; *Java Virtual Machine) définie par Sun*

♠ Ce langage (bytecode) est indépendant de l'ordinateur qui va exécuter le programme

La compilation fournit du *bytecode*

Programme écrit en Java



Programme en *bytecode*, indépendant de l'ordinateur

Compilation avec *javac*

Sun fournit le compilateur javac avec le JDK

javac HelloWorld.java

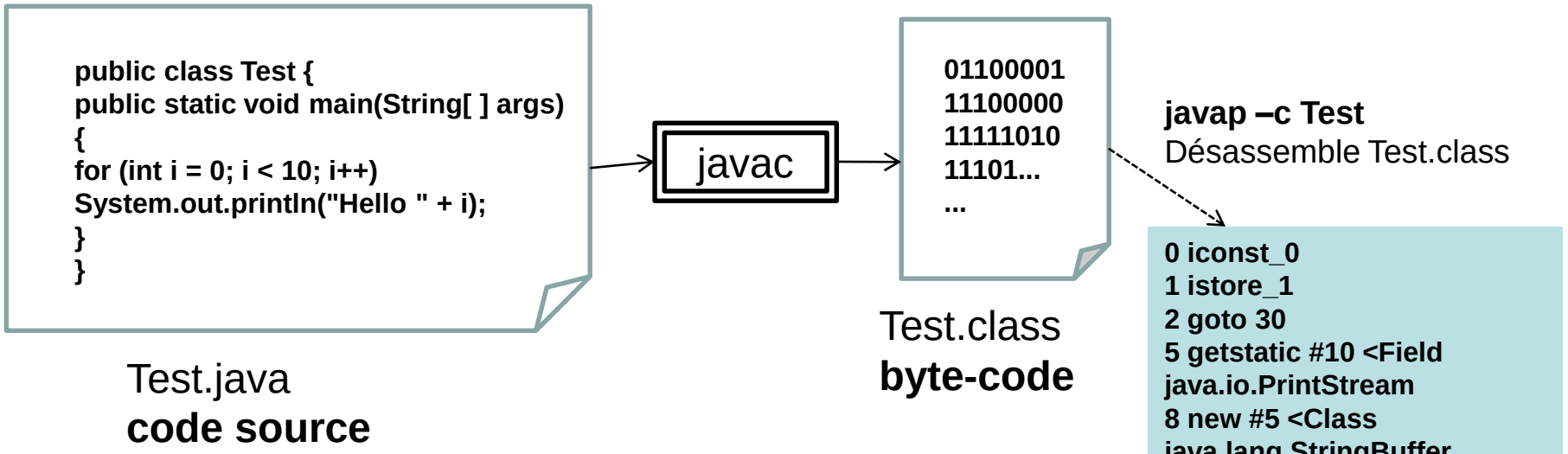
crée un fichier « **HelloWorld.class » qui** contient le *bytecode, situé dans le même* répertoire que le fichier « **.java »**

On peut désigner le fichier à compiler par un chemin absolu ou relatif :

javac SousRep/HelloWorld.java

Le langage Java
Un langage compilé & interprété

- **Compilation d'un programme JAVA : génération de byte-code**



Test.java
code source

Test.class
byte-code

- **Le byte-code est :**
  - proche d'un langage machine
  - indépendant de la platforme d'exécution (matériel + OS)

Exécution du *bytecode*

Le *bytecode doit être exécuté par une JVM*

Cette JVM n'existe pas ; elle est simulée par un programme qui :

- lit les instructions (en *bytecode) du* programme **.class,**
- les traduit dans le langage natif du processeur de l'ordinateur
- lance leur exécution

Exécution avec *java*

♠*Sun fournit le programme java qui simule une* JVM

♠ **java HelloWorld**

Nom d'une classe
(pas d'un fichier) ;
pas de suffixe .class !

interprète le *bytecode de la méthode* ***main()*** de la classe **HelloWorld**

♠ **HelloWorld.class** doit être dans le répertoire courant ou dans un des emplacements indiqués par une option **-classpath** ou par la variable **CLASSPATH**

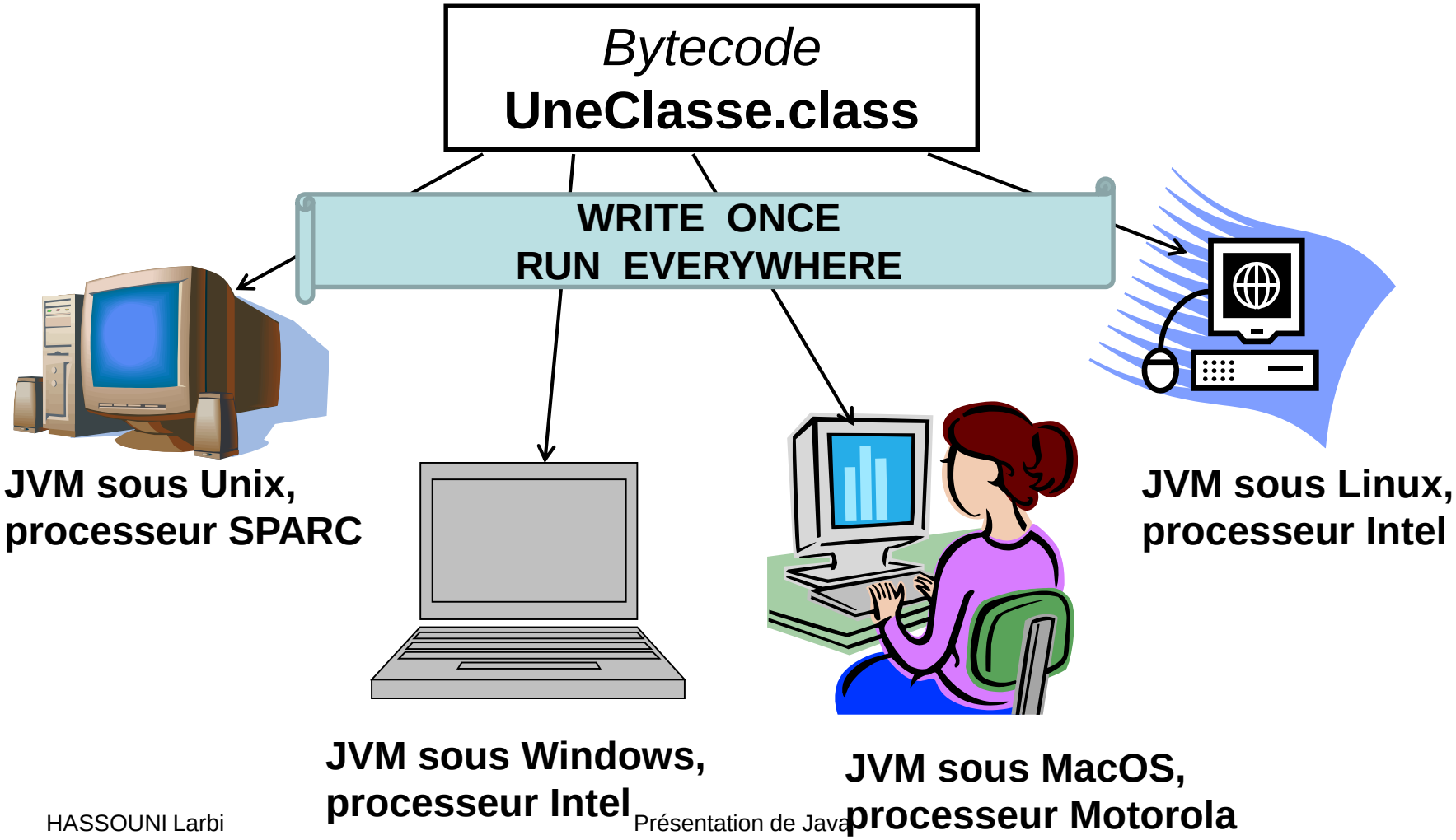
Les JVM

♠ Les systèmes qui veulent pouvoir exécuter un programme Java doivent fournir une JVM

♠ A l'heure actuelle, tous les systèmes ont une JVM (Linux, Windows, MacOs,…)

♠ Il existe aussi depuis peu quelques JVM « en dur », sous forme de processeurs dont le langage natif est le *bytecode ; elles sont* rarement utilisées

Le *bytecode peut être exécuté par* n'importe quelle JVM



Avantages de la JVM pour Internet

♠ Grâce à sa portabilité, le *bytecode d'une classe* peut être chargé depuis une machine distante du réseau, et exécutée par une JVM locale

♠ La JVM fait de nombreuses vérifications sur le *bytecode avant son exécution pour s'assurer qu'il* ne va effectuer aucune action dangereuse

♠ La JVM apporte donc
– de la souplesse pour le chargement du code à exécuter
– mais aussi de la sécurité pour l'exécution de ce code

Une certaine lenteur...

♠ Les vérifications effectuées sur le bytecode et l'étape d'interprétation de ce bytecode (dans le langage natif du processeur) ralentissent l'exécution des classes Java

♠ Mais les techniques « *Just In Time (JIT) » ou « Hotspot » réduisent ce problème :*
Elles permettent de ne traduire qu'une seule fois en code natif les instructions qui sont exécutées

Java et les autres langages

♠ Java est devenu en quelques années un des langages de développement les plus utilisés, surtout pour les applications qui ont besoin d'une grande portabilité ou d'une grande souplesse sur Internet

♠ Pour les applications qui nécessitent une très grande rapidité d'exécution, on préfère encore les langages C, C++, ou le bon vieux Fortran
(qui a des bibliothèques très utilisées pour le calcul scientifique)

Spécifications de Java

♠ Java, c'est en fait

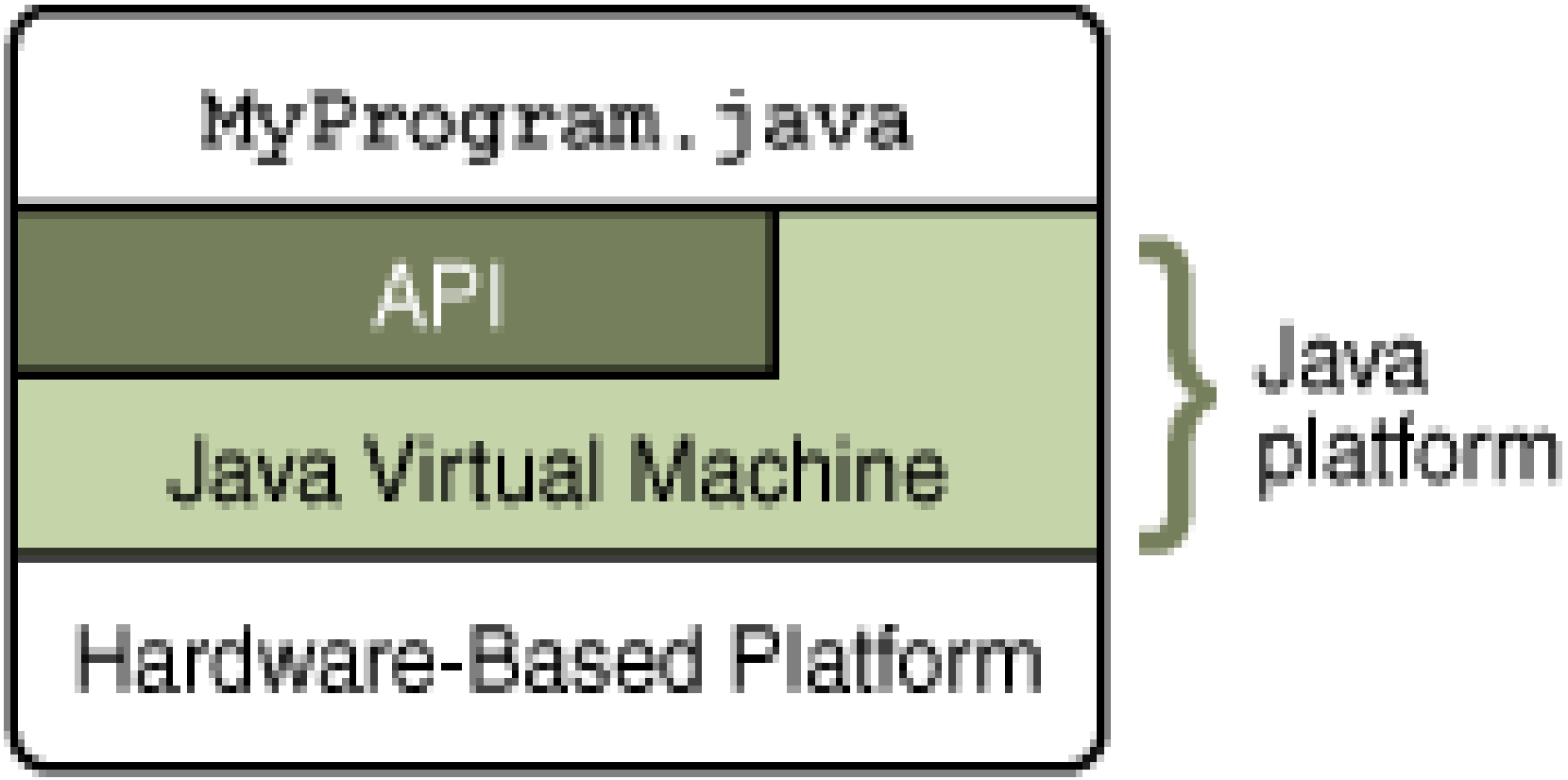
– le langage Java : http://java.sun.com/docs/books/jls/

– une JVM : http://java.sun.com/docs/books/vmspec/

– les API : Ensemble de classes prédéfinies et réparties sur plusieurs packages

♠ Java n'est pas normalisé ; son évolution est gérée par le JCP (Java Community Process ; http://www.jcp.org/) dans lequel Oracle tient une place prépondérante

Plate-forme Java



API (*Application Programming Interface) :*
bibliothèques de classes standard

3 éditions de Java

♠ Java SE : Java Standard Edition ; JDK = *J2SE Development Kit, aussi appelé SDK (Software Development Kit) pour certaines versions*
Fournit les compilateurs, outils, runtimes, et APIs pour écrire, déployer, et exécuter des applets et applications

♠ JavaEE : Enterprise Edition qui ajoute les API pour écrire des applications installées sur les serveurs dans des applications distribuées : servlet, JSP, EJB,…

♠ JavaME : Micro Edition, version allégée de Java pour écrire des programmes embarqués (cartes à puce/Java card, téléphones portables,…)

Votre environnement de développement

- Éditeur de texte NotePad
- Compilateur (*javac)*
- Interpréteur de *bytecode (java)*
- Aide en ligne sur le JDK (sous navigateur Web)
- Générateur automatique de documentation (*javadoc)*
- Testeur pour applet (*appletviewer)*
- Débogueur (*jdb)*

Votre environnement de développement

♠ Nombreux IDE (Integrated Development Environment)

Java Studio Creator
Sunsoft

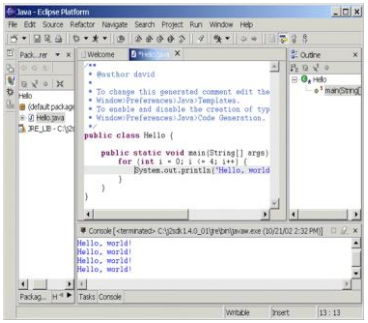
WebSphere Studio Application Developper (VisualAge) IBM

JBuilder
Borland

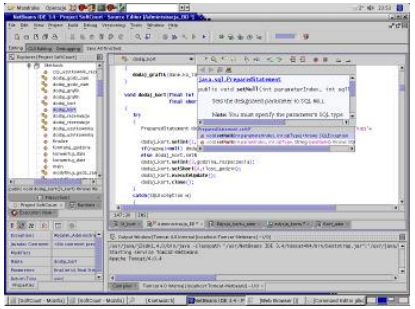
ORACLE JDEVELOPER 10g

JDeveloper
Oracle

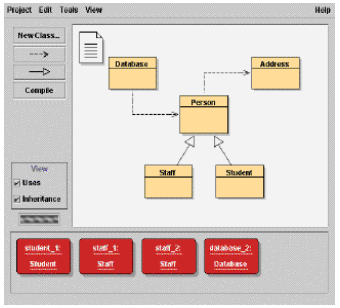
♠ Des environnements open-source ou freeware



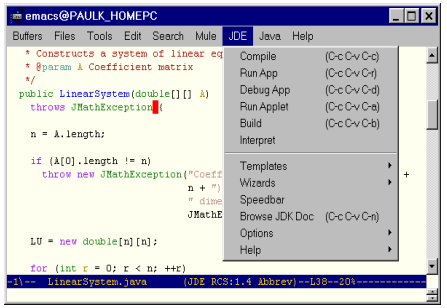
Eclipse
www.eclipse.org



NetBeans
www.netbeans.org



BlueJ
www.bluej.org



Emacs + JDE
http://sunsite.auc.dk/jde

Variables d'environnement

♠ **PATH** : doit inclure le répertoire qui contient les utilitaires Java (**javac, java, javadoc,…)**

♠ **CLASSPATH** : indique le chemin de recherche des classes de l'utilisateur

♠ **Le débutant ne doit pas avoir de variable CLASSPATH**

Une classe Point

```
/** Modélise un point de coordonnées x, y */
public class Point {
        private int x, y;
        public Point(int x, int y) { // un constructeur
                this.x = x;
                this.y = y;
        }

        public double distance(Point p) { // une méthode
                return Math.sqrt((this.x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
        }

        public static void main(String[] args) {
                Point p1 = new Point(1, 2);
                Point p2 = new Point(5, 1);
                System.out.println("Distance : " + p1.distance(p2));
        }
}
```

2 classes dans 1 fichier

```
/** Modélise un point de coordonnées x, y */
public class Point {
        private int x, y;
        public Point(int x1, int y1) {
                x = x1; y = y1;
        }

        public double distance(Point p) {
                return Math.sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
        }
}
```

Fichier **Point.java**

```
/** Teste la classe Point */
class TestPoint {
        public static void main(String[] args) {
                Point p1 = new Point(1, 2);
                Point p2 = new Point(5, 1);
                System.out.println("Distance : " + p1.distance(p2));
        }
}
```

Compilation et exécution de la classe Point

La compilation du fichier **Point.java**

```
javac Point.java
```

fournit 2 fichiers classes : **Point.class et TestPoint.class**

On lance l'exécution de la classe **TestPoint** qui a une méthode **main()**

```
java TestPoint
```

2 classes dans 2 fichiers

Fichier **Point.java**

```
/** Modélise un point de coordonnées x, y */
public class Point {
        private int x, y;
        public Point(int x1, int y1) {
                x = x1; y = y1;
        }

        public double distance(Point p) {
                return Math.sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
        }
}
```

Fichier **TestPoint.java**

```
/** Pour tester la classe Point */
class TestPoint {
        public static void main(String[] args) {
                Point p1 = new Point(1, 2);
                Point p2 = new Point(5, 1);
                System.out.println("Distance : " + p1.distance(p2));
        }
}
```

Architecture d'un programme source Java

♠ Programme source Java = ensemble de fichiers « **.java »**

♠ Chaque fichier « **.java » contient une ou** *plusieurs définitions de classes*

♠ Au plus une définition de classe **public par** fichier « **.java » (avec nom du fichier = nom de la classe publique**)

Chargement dynamique des classes

♠ Durant l'exécution d'un code Java, les classes (leur *bytecode) sont chargées dans la JVM au* fur et à mesure des besoins

♠ Une classe peut être chargée:

– depuis la machine locale (le cas le plus fréquent)

– depuis une autre machine, par le réseau

– par tout autre moyen (base de données,…)

Types de programmes Java

On peut développer trois types de programmes avec java

1- Applications indépendantes (ou stand alone)

2- *Applets* exécutées dans l'environnement/JVM d'un navigateur Web et chargées par une page HTML

3- **Allications Client/Serveur** qui s'exécutent sur un serveur d'application

Application indépendante

♠ **Application doit posséder une classe principale**
– classe possédant une méthode de signature

public static void main(String[] args)

Tableau de chaînes de caractères (équivalent à argc, argv du C)

♠ **Cette méthode sert de point d'entrée pour l'exécution de l'application**

♠ Lancement de l'application s'effectue en exécutant la méthode main de la classe principale de l'application*; par exemple :*

java TestPoint

lance l'interprétation du code de la méthode **main()** de la classe principale **TestPoint** qui se trouve dans le fichier **TestPoint.class**

Applet

♠ **Classe ne possédant pas de méthode main()**

• **Hérite de java.awt.Applet ou javax.swing.JApplet**

• **Son bytecode réside sur un serveur http**

• **Elle est véhiculée vers un client http (navigateur Web) via une page HTML qui contient son url**

• **Lorsqu'un navigateur compatible Java (avec sa propre machine virtuelle java (JVM)) reçoit cette page HTML, il télécharge le code de la classe et l'exécute sur le poste client**

– l'applet doit posséder un certain nombre de méthodes pour permettre cette exécution :

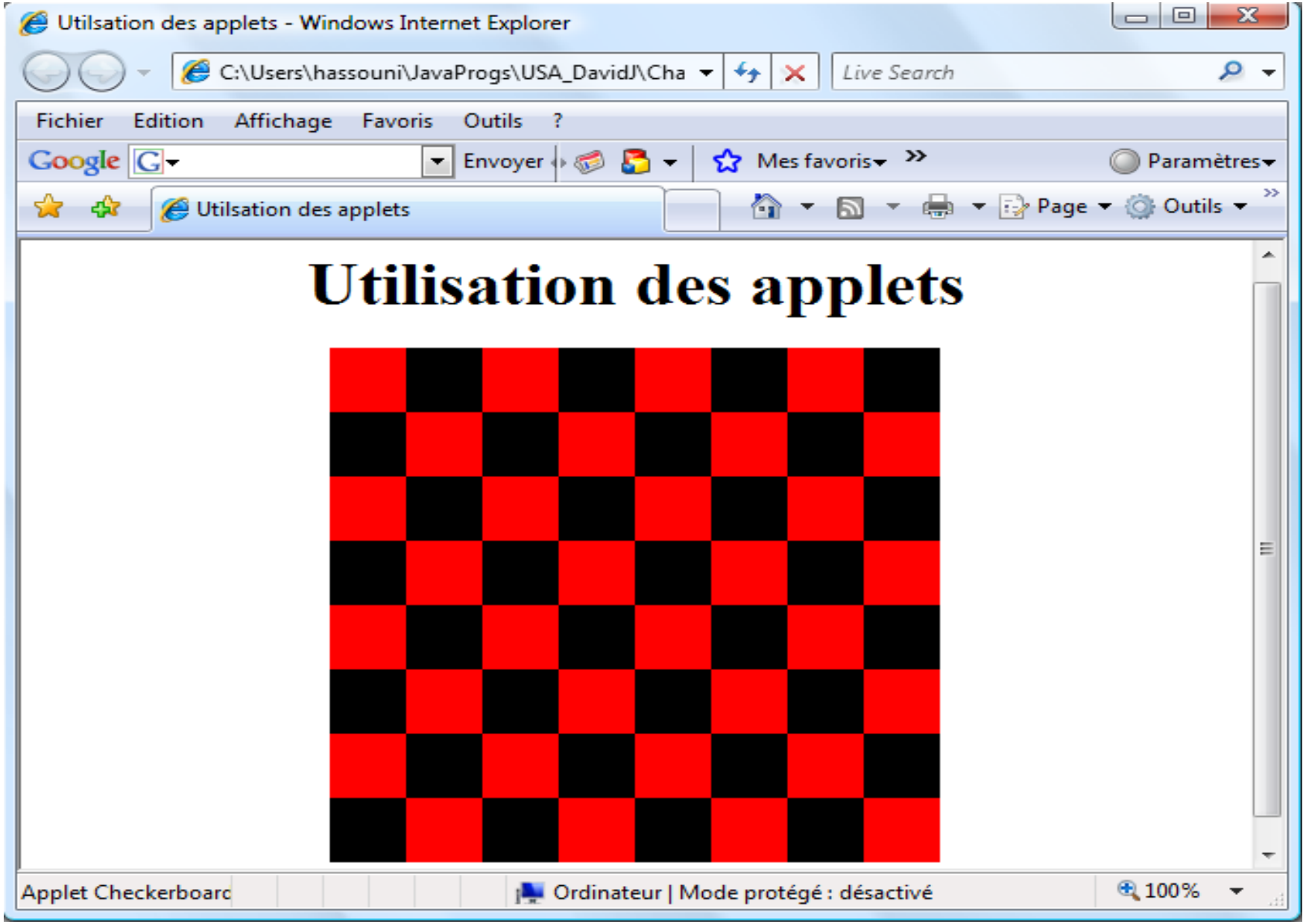
init(), start(), stop(), paint(), destroy()

Exemple d'applet

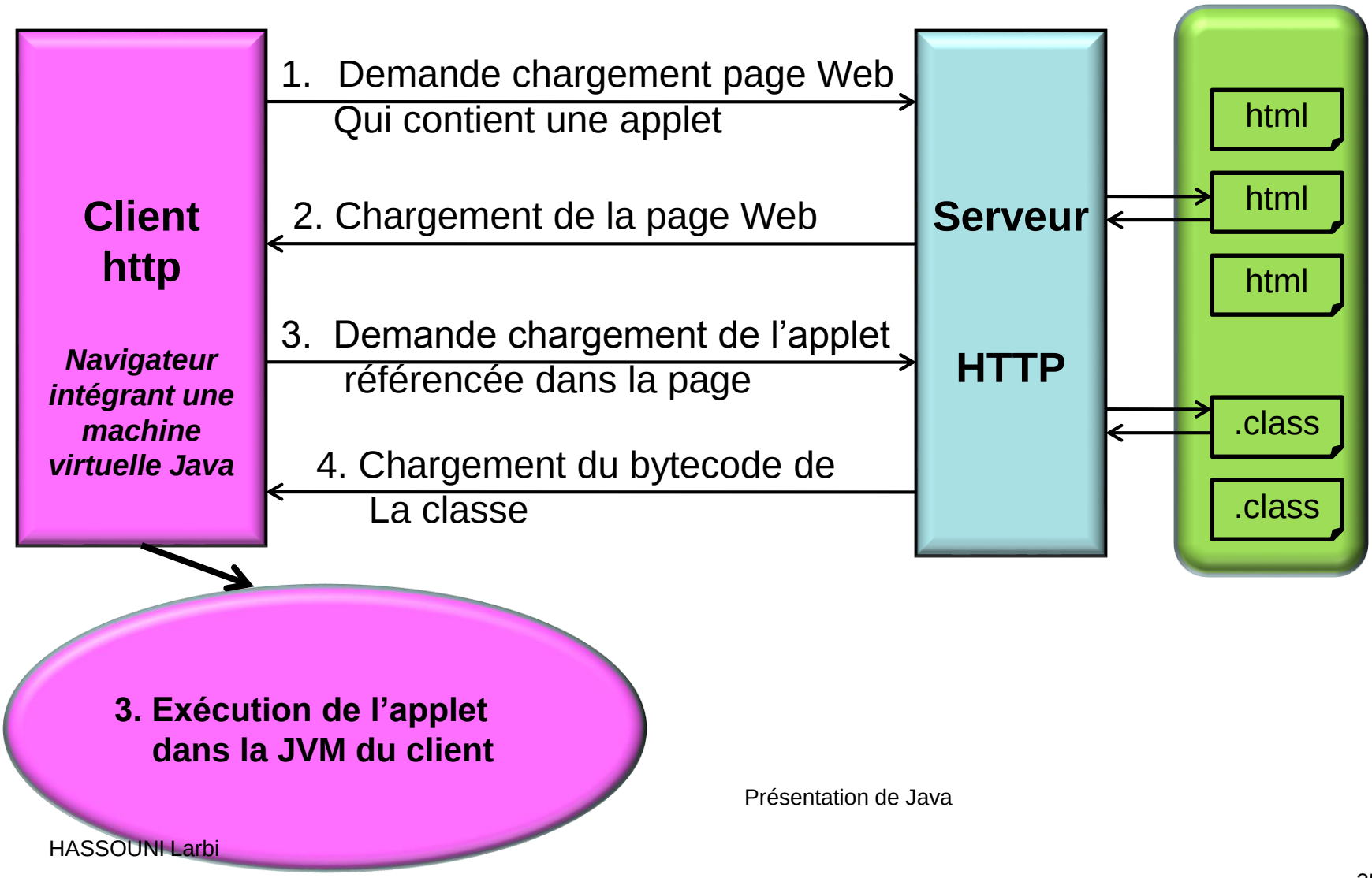
```
import java.awt.*;
import java.applet.*;
public class Checkerboard extends Applet {
   /*  This applet draws a red-and-black checkerboard.
       It is assumed that the size of the applet is 160  by 160 pixels.
   */
   public void paint(Graphics g) {
      int row;   // Row number, from 0 to 7
      int col;   // Column number, from 0 to 7
      int x,y;   // Top-left corner of square
      for ( row = 0;  row < 8;  row++ ) {
         for ( col = 0;  col < 8;  col++) {
            x = col * 80;            y = row * 80;
            if ( (row % 2) == (col % 2) )
               g.setColor(Color.red);
            else
               g.setColor(Color.black);
            g.fillRect(x, y, 80, 80);
         }
      } // end for row
   }  // end paint()
}  // end class Checkerboard
```

Exemple de page Web qui contient une applet

```
<html>
<head>
<title>Utilsation des applets</title>
</head>
<body>
<h1 align = "center">Utilisation des applets</h1>
<p align=center>
<applet code="Checkerboard.class"
        height=640  width=640>
</applet>
</p>
</body>
<html>
```


Utilsation des applets - Windows Internet Explorer
C:\Users\hassouni\JavaProgs\USA_DavidJ\Cha
Live Search
Fichier Edition Affichage Favoris Outils ?
Google
Envoyer
Mes favoris
Paramètres
Utilsation des applets
Page
Outils
Utilisation des applets
Applet Checkerboarc
Ordinateur | Mode protégé : désactivé
100%

Étapes pour l'exécution d'une applet



Exécution de l'applet

♠ Le navigateur a sa propre machine virtuelle

♠ Un programme Java spécial démarré par le navigateur va lancer certaines méthodes de la classe **Applet :**
init(), start(), stop(), destroy(), paint()

♠ **init()** est exécuté seulement quand l'applet
est lancée pour la première fois

♠ **paint()** dessine l'applet dans la page Web

Utilité des *applets*

♠ Les applets permettent de faire des pages Web plus riches (grâce aux possibilités offertes par Java)

♠ La page Web peut contenir

- des animations ou des mises en forme complexes pour mettre en valeur certaines informations
- des résultats de calculs complexes
- des informations « dynamiques » (pas connues au moment où la page Web statique est créée) trouvées en interrogeant une base de données
- ….