Programmation par Objets - Printemps 2010
Frédéric Mallet & Jean-Paul Roy
L1 Info&Math - Faculté des Sciences de Nice

http://deptinfo.unice.fr/~roy

Cours n°8

Université Nice SOPHIA ANTIPOLIS

Le langage Java

Java™

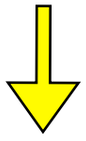
avec

Les langages Processing et Java

• **Java** est un langage de programmation généraliste, orienté objets, né vers 1995, et qui permet à la fois d'écrire des programmes industriels et de petites applications (***applets***) à inclure dans des pages Web.

• Il permet aussi de programmer des téléphones portables, des appareils électro-ménagers, etc. Il comporte d'énormes bibliothèques mais demande une certaine expertise.

http://java.sun.com



http://www.processing.org

• **Processing** est un logiciel de création de programmes d'animation graphique, notamment pour inclusion dans des pages Web, avec un effort beaucoup moins important que si l'on utilisait seulement Java.

• Il est notamment destiné aux étudiants et aux artistes numériques. De nombreux chercheurs ont étendu son champ d'applications vers de nouvelles zones : musique, électronique, etc.

Pourquoi ne pas étudier que Processing ?

• Parce que **Processing** n'est qu'un système de programmation particulier, plutôt réservé à un usage bien délimité. Pour un futur professionnel de l'informatique, il est insuffisant. Peu de livres sont disponibles, récents, et en anglais seulement jusqu'à présent...

Pourquoi ne pas étudier que Java ?

• Parce que **Java** est un langage complexe, pour lequel une petite application graphique devient tout de suite une tâche très compliquée. Et le graphisme (le multimédia en général) fait partie de notre contexte informatique quotidien !

• Parce qu'un simple calcul mathématique requiert inutilement l'écriture d'une classe en Java, ce qui encombre l'approche du débutant.

• Parce que Processing est basé sur Java. La syntaxe est simplifiée mais on peut y inclure du véritable code Java. Il peut délivrer une véritable application Java, ou une applet pour une page Web.

Pourquoi cette orientation vers Java ?

- Pour toutes raisons que l'on a citées plus haut. Il est moderne, très utilisé, orienté objets, permet de générer des applications pour Internet, pour la téléphonie mobile, il est inscrit dans la technologie la plus récente.

- Il est possible de programmer gratuitement avec le JDK Java pour toutes les plate-formes (Linux, MacOS-X, Windows). La même application tournera en principe à l'identique sur n'importe quelle machine, gage d'économie de développement.

- Ses principaux concurrents sont **C#** sur Windows (Microsoft), **Objective-C** sur MacOS-X (Apple), et **C++** partout.

- A notre échelle, sa syntaxe reste relativement simple, tant qu'on n'aborde pas le graphisme. Les problèmes graphiques resteront traitables par Processing, quitte à programmer en Processing de véritables classes Java.

Simplifications arithmétiques de Processing

• Pour accélérer le graphisme, Processing utilise des nombres approchés en **simple précision** : c'est le type primitif `float`.

```
println(PI);
```

Run →

```
3.1415927
```

• Bien qu'il admette aussi le type `float`, Java opte pour une **double précision** des nombres approchés : c'est le type primitif `double`.

```
System.out.println(Math.PI);
```

→

```
3.141592653589793
```

• OUI, vous pouvez taper cette phrase Java en Processing ! Mais ne mélangez pas les deux, `println` ne connait pas le type `double` !

```
println(3.141592653589793);
```

→

```
3.1415927
```

```
println(Math.PI);
```

→

```
ERROR
```

• Donc `println` pour Processing et `System.out.println` pour Java.
Notez au passage la notation : `println` de Java est un message envoyé à l'objet `System.out` (l'écran du poste de travail) ! La **notation pointée** a donc la même signification en Processing et en Java...

• Les **fonctions arithmétiques** de Processing sur les nombres approchés fonctionnent en Java avec le type `double` et appartiennent à une classe spécialisée `Math` qui est cachée dans Processing :

en Processing		*en Java*
`println("x = " + x);`		`System.out.println("x = " + x);`
type `float` *par défaut pour les nombres approchés*		*type* `double` *par défaut pour les nombres approchés*
`sqrt(x)`	$\sqrt{x}$	`Math.sqrt(x)`
`pow(x,y)`	x^y	`Math.pow(x,y)`
`PI`	π	`Math.PI`

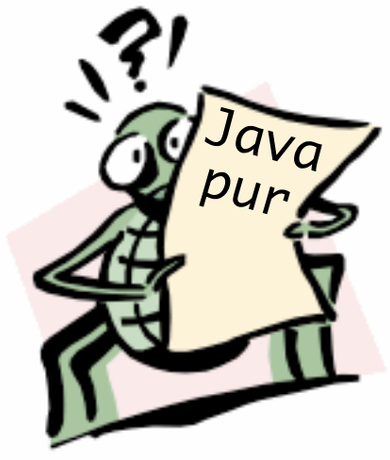
Le graphisme de Processing

• Il est extraordinairement **plus simple que celui de Java** ! Une animation fluide de Processing nécessiterait en Java des classes compliquées de l'API, des techniques de *double buffer* pour éviter le scintillement à chaque effacement du canvas, etc.

• N'essayez donc pas de transposer en Java pur (construit en-dehors de Processing) vos anciens programmes graphiques ! Vous verrez comment faire en L3-Info !...

• Dans un programme Java pur, ne pensez plus au canvas, aux fonctions `setup`, `draw`, `ellipse`, `stroke`, etc.

Faire du pur Java ?
Mais comment ?...



Le logiciel DrJava

http://www.drjava.org

Le logiciel DrJava

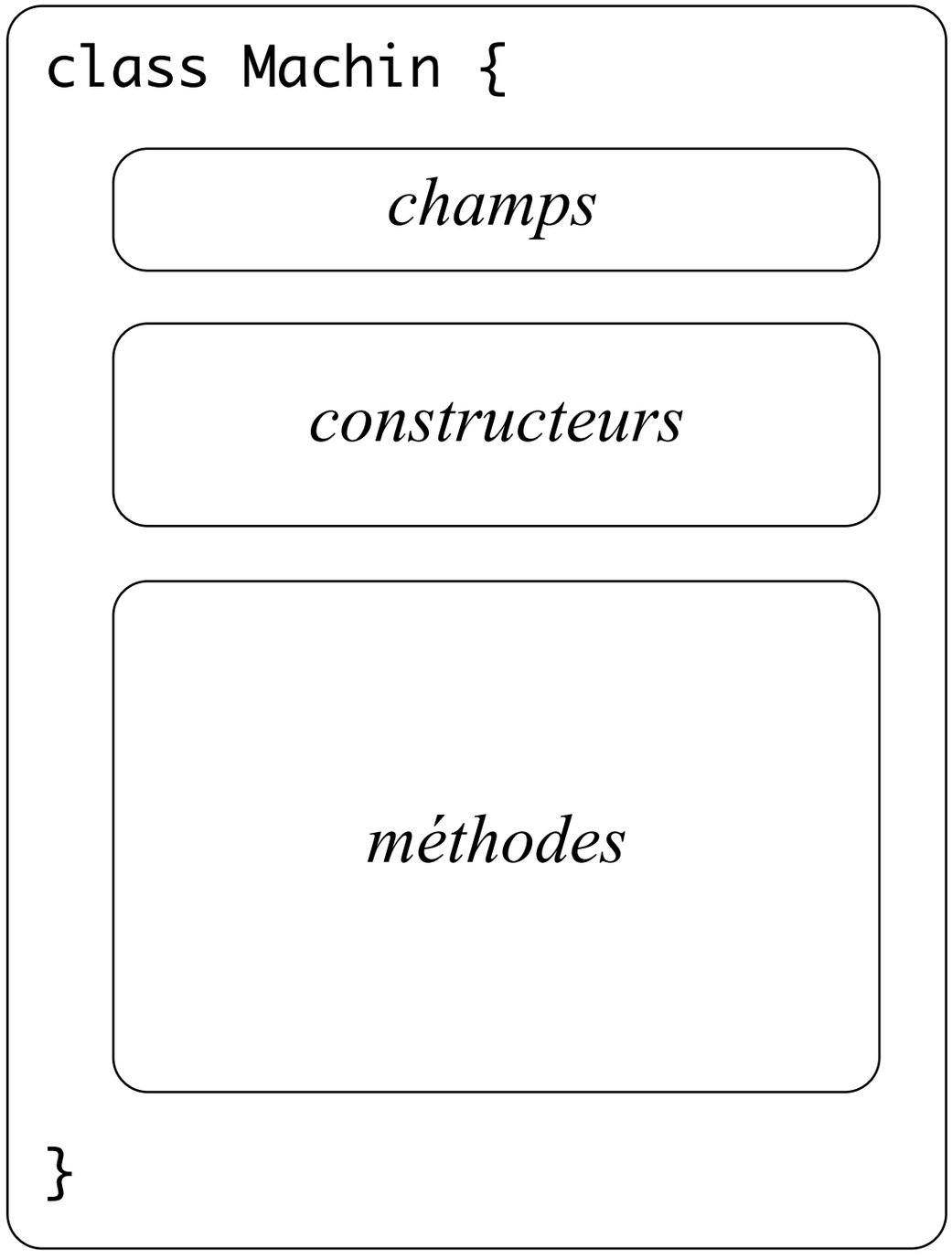
• Il s'agit là aussi d'un logiciel **gratuit téléchargeable** sur tous les systèmes (Linux, MacOS-X, Windows). Bien entendu, comme pour Processing, il faut avoir installé le **JDK** (1.5) de Sun qui fournit le compilateur et l'API.

• Comme Processing, DrJava est un environnement de développement intégré (IDE) contenant un éditeur de textes Java et une fenêtre pour les résultats. Non, ne me parlez pas de graphisme !

• DrJava est un système universitaire dédié à l'enseignement. Vous utiliserez plus tard des environnements beaucoup plus professionnels (Eclipse, JBuilder, Sun One, Visual Age) mais aussi bien plus compliqués.

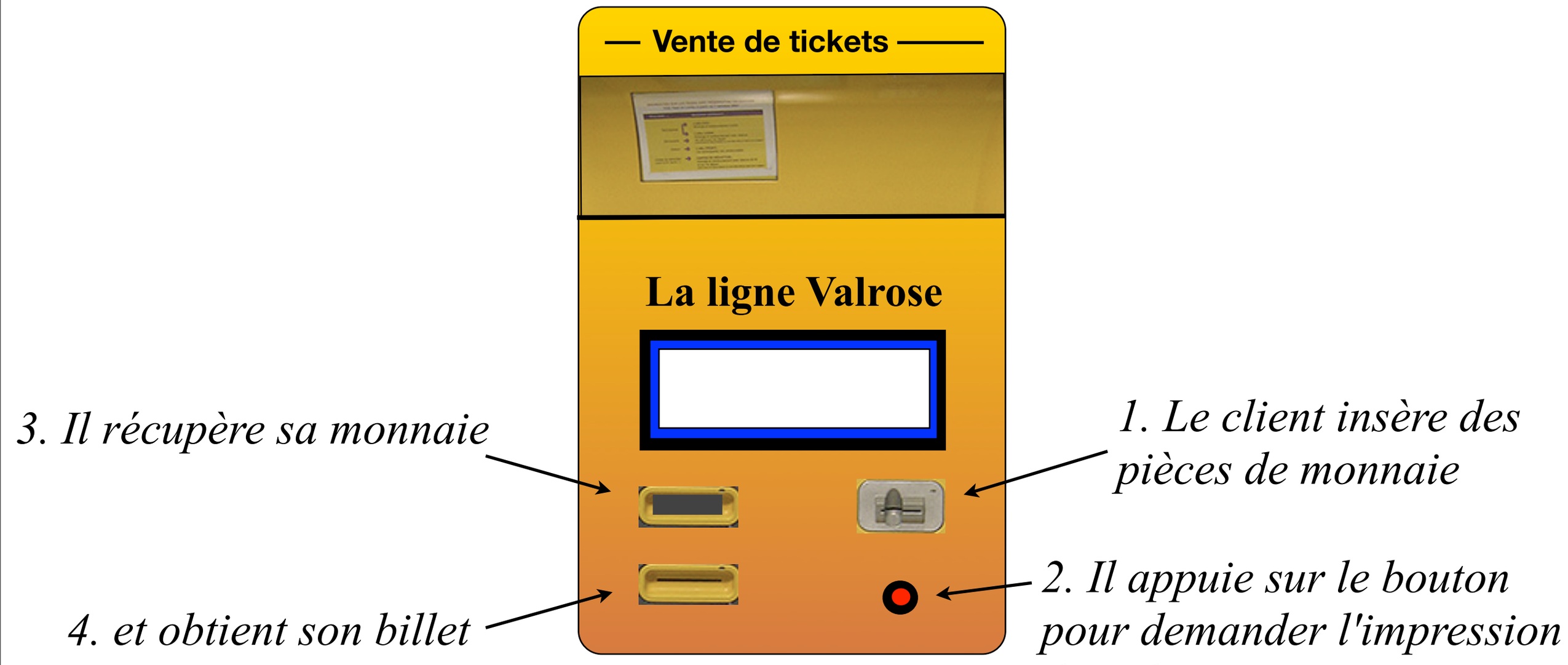
Programmer une classe d'objets en Java

- Nous avons déjà vu l'architecture d'une classe en Processing, nous conservons cette architecture en Java :



Exemple : la classe `TicketMachine`

- Modélisation naïve d'un **distributeur de billets de train à prix unique**.
- Les clients insèrent de l'argent et demandent l'impression d'un ticket.



• A tout moment **l'état** d'un distributeur est caractérisé par : le prix de son billet, la somme d'argent qui a déjà été insérée par un client, et la monnaie qui a été rendue. Prenons un seul **constructeur** :

champs

constructeur

```
class TicketMachine {
  int ticketCost;    // prix du billet
  int balance = 0;   // somme déjà insérée

  TicketMachine(int cost) {
    ticketCost = cost;
  }

  <méthodes>
}
```

• Notez qu'un champ est initialisé par le constructeur avec une valeur constante, l'autre par le constructeur en fonction d'un paramètre.

- METHODES. Que doit savoir faire un distributeur d ?

  - **accepter de l'argent** du client. Cette méthode va modifier la somme déjà insérée.

    `d.insertMoney : int → void`

  - **imprimer le billet** et rendre la monnaie. Il effectue bien entendu les vérifications nécessaires quant à la somme insérée...

    `d.printTicket : void → void`

  - à tout moment **donner son état** :

    `d.toString : void → String`

- Dans une modélisation par objets, il est essentiel de bien réfléchir à ce qui caractérise l'état d'un objet, et à son savoir-faire !

- Les méthodes de la classe `TicketMachine` :

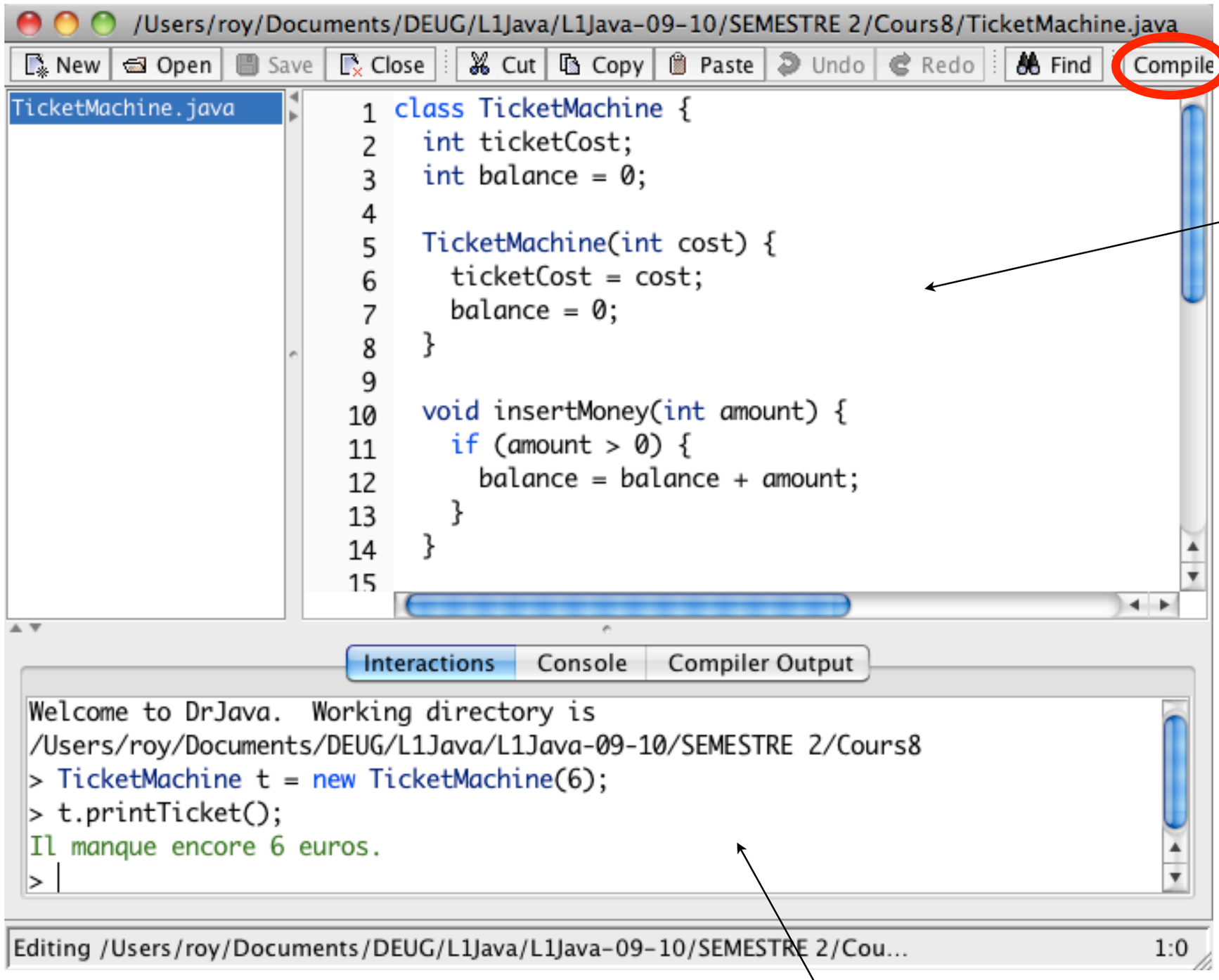
```
void insertMoney(int amount) {
    if (amount > 0) {
      balance = balance + amount;
    }
  }

void printTicket() {
  if (balance >= ticketCost) {
    System.out.println("*** BILLET LIGNE VALROSE : " + ticketCost + " EUROS ***");
    balance = balance - ticketCost;
    if (balance > 0) {
      int refund = balance;
      balance = 0;
      System.out.println("Veuillez récupérer " + refund + " euro(s).");
    }
  } else {
    System.out.println("Il manque encore " + (ticketCost - balance) + " euros.");
  }
}

public String toString() {
    return "TicketMachine[ticketCost=" + ticketCost + ",balance=" + balance + "]";
}
```

Utilisation de drjava

le compilateur

la classe en cours d'édition

l'éditeur



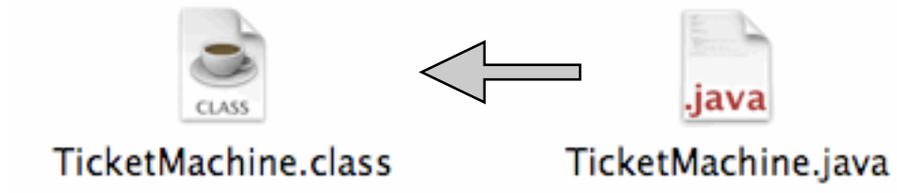
le toplevel interactif pour parler aux objets et tester ses programmes

• Vous avez **entré votre programme dans l'éditeur**, vous le sauvegardez sur votre clef USB. ATTENTION : le nom du fichier doit être exactement le nom de la classe avec l'extension `.java`. Ici :

`TicketMachine.java`

• Soyez attentif aux majuscules/minuscules : erreur fréquente !...

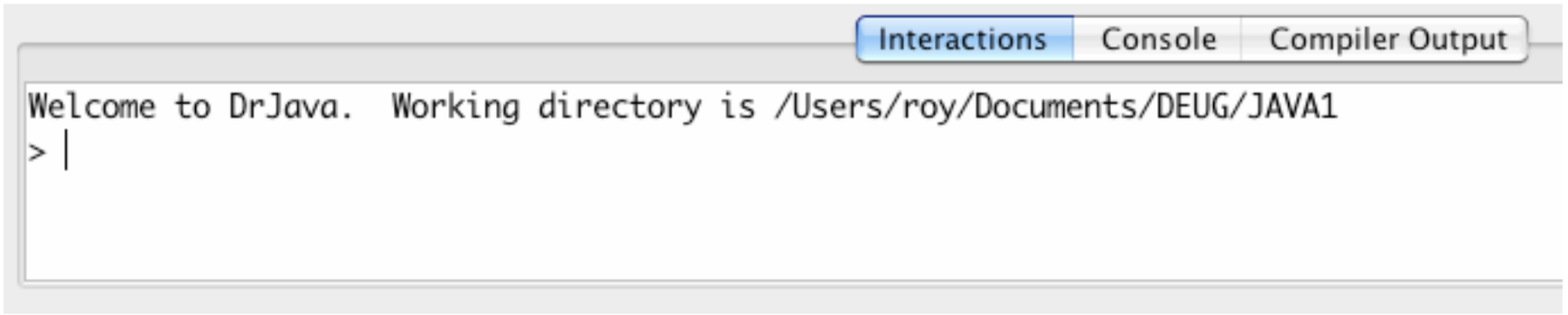
• Ensuite vous **compilez la classe** : traduction en langage-machine. Le compilateur Java (inclus dans le JDK de Sun®, et utilisé par Processing et DrJava) va traduire votre fichier `TicketMachine.java` en un fichier binaire `TicketMachine.class`. C'est ce dernier qui sera exécuté et qui voyagera sans problème entre Linux, MacOS-X, et Windows.



le fichier compilé

le fichier source

- Si le compilateur a montré des erreurs, vous corrigez ! S'il a dit `Compilation completed`, vous allez cliquer dans l'onglet ***Interactions*** pour arriver au ***toplevel*** de DrJava :



- Au toplevel, vous allez pouvoir créer des objets et leur envoyer des messages. **Si votre message est sans résultat, c'est une instruction**, terminez-le par un point-virgule.

```
Welcome to DrJava.
> TicketMachine tm = new TicketMachine(6);
> tm.insertMoney(4);
>
```

- Si vous demandez la **valeur d'une expression**, ne la terminez PAS par un point-virgule !

```
> tm
TicketMachine[ticketCost=6,balance=4]
> tm.balance
4
> tm.balance;
>
```

- Utilisons le toplevel pour tester le fonctionnement d'un distributeur :

```
Welcome to DrJava.
> TicketMachine tm = new TicketMachine(6);
> tm.insertMoney(4);
> tm.printTicket();
Il manque encore 2 euros.
> tm.insertMoney(3);
> tm
TicketMachine[ticketCost=6,balance=7]
> tm.printTicket();
*** BILLET LIGNE VALROSE : 6 EUROS ***
Veuillez récupérer 1 euro(s).
> tm
TicketMachine[ticketCost=6,balance=7]
```

- Le toplevel sert aussi pour tester des primitives Java :

```
> int x = 2009;
> Math.sqrt(x) + 1       // type double !
45.82186966202994
```

- Le bouton Reset de DrJava permet de **ré-initialiser le toplevel** et d'oublier toutes variables définies. Les touches fléchées permettent de récupérer les anciennes commandes entrées au toplevel (*historique*).

Tester une classe d'objets

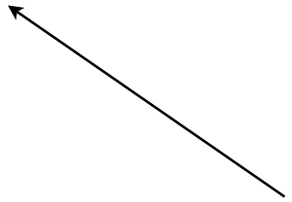
- On peut programmer une autre classe sans constructeur, dont les objets sont des tests, par exemple `TicketMachineTest` :

```
class TicketMachineTest {
  void go() {
    System.out.println("Je construis un distributeur à 6 euros");
    TicketMachine tm = new TicketMachine(6);
    System.out.println("J'insère 4 euros");
    tm.insertMoney(4);
    System.out.println("Je demande à imprimer le billet");
    tm.printTicket();
    System.out.println("J'insère 3 euros");
    tm.insertMoney(3);
    System.out.println("Je demande à imprimer le billet");
    tm.printTicket();
    System.out.println("Je demande à voir l'état de la machine");
    System.out.println(tm);
  }
}
```

- **Lorsqu'une classe n'a pas besoin de constructeur, on utilise le constructeur par défaut sans paramètre !**

- Je compile tout, puis je construis un test au toplevel, et le lance :

```
Welcome to DrJava.
> TicketMachineTest test = new TicketMachineTest();
> test.go();
Je construis un distributeur à 6 euros
J'insère 4 euros
Je demande à imprimer le billet
Il manque encore 2 euros.
J'insère 3 euros
Je demande à imprimer le billet
*** BILLET LIGNE VALROSE : 6 EUROS ***
Veuillez récupérer 1 euro(s).
Je demande à voir l'état de la machine
TicketMachine[ticketCost=6,balance=0]
>
```



le constructeur par défaut

- Il pourrait y avoir plusieurs méthodes de test.
- Une classe de test peut tester plusieurs classes d'objets.
- Les objets d'une classe de test n'ont pas vraiment d'intérêt. Nous verrons plus tard des méthodes *qui appartiennent à la classe*...