



Programmation OO

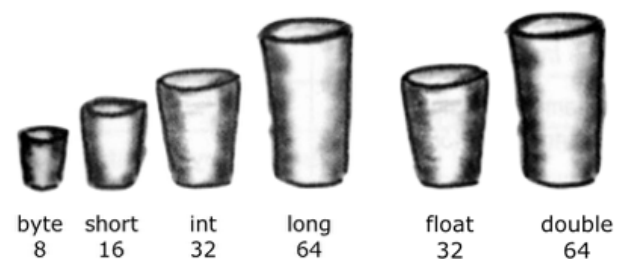
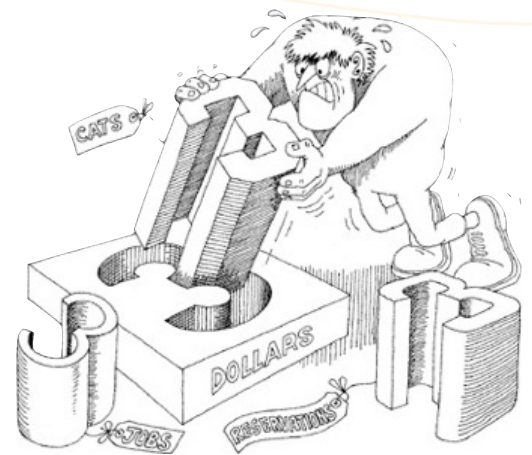
Programmation Orientée Objet en Java

POO Java

Éléments de syntaxe avancés

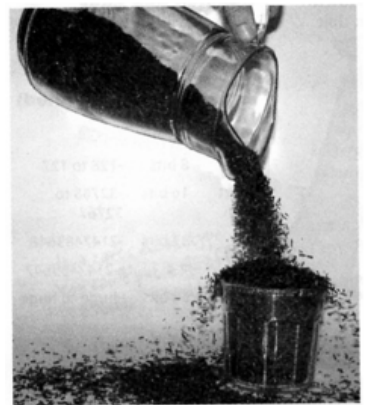
Types primitifs

- ☐ boolean **true** ou **false**
- ☐ char 16 bits (caractères)
- ☐ Numériques (signés)
 - ☐ Entiers
 - ☐ byte 8 bits -128 à 127
 - ☐ short 16 bits -32 768 à 32 767
 - ☐ int 32 bits -2 147 483 648 à 2 147 483 647
 - ☐ long 64 bits immense
 - ☐ Décimaux
 - ☐ float 32 bits
 - ☐ double 64 bits



□ Les pièges !











- Vérifier que la valeur peut entrer dans la variable
- Le compilateur vous empêche de faire :
`int x = 24;`
`byte b = x;`
- Les valeurs littérales sont par défaut de type **int** sauf si on précise le type
`long var = 234L;`



```
public class LongDivision {  
    private static final long MILLIS_PER_DAY = 24 * 60 * 60 * 1000;  
    private static final long MICROS_PER_DAY = 24 * 60 * 60 * 1000 * 1000;  
    public static void main(String[] args) {  
        System.out.println(MICROS_PER_DAY / MILLIS_PER_DAY);  
    }  
}
```

↪ Tester et corriger le programme

Classes d'encapsulation (ou classes d'emballage)

-  Permettent d'utiliser un type primitif sous forme d'objet
 -  Boolean
 -  Number, Byte, Short, Integer, Long, Float, Double
 -  Character
-  Ces classes définissent des constantes
 -  MIN_VALUE et MAX_VALUE pour les types numériques
 -  NaN, POSITIVE_INFINITY, NEGATIVE_INFINITY pour les types Float et Double
-  Elles fournissent également des méthodes pour manipuler les données
 -  Character : isLetter(), isLowerCase(), isDigit(), isWhiteSpace(), ...
 -  Elles fournissent des méthodes "parse" pour la conversion de chaîne de caractères

☐ Classes d'encapsulation (ou classes d'emballage)

☐ auto-boxing / auto-unboxing

☐ Auto-boxing

- ☐ `Integer myInt = 13;`

- ☐ remplacé par `Integer myInt = new Integer(13);` par le compilateur

☐ Auto-unboxing

- ☐ `int x = myInt;`

- ☐ remplacé par `int x = myInt.intValue();` par le compilateur

☐ Classes d'encapsulation (ou classes d'emballage)

- ☐ auto-boxing / auto-unboxing

- ☐ Les pièges

```
Long sum = 0L;  
for (long i = 0; i < Integer.MAX_VALUE; i++) {  
    sum += i;  
}
```

- ☐ Cette boucle s'exécute en 7 secondes sur ma machine alors qu'en la corrigeant elle passe à 1,1 secondes
- ☐ Quelle correction faut-il apporter ?
- ☐ Tester (en mesurant le temps d'exécution)
- ☐ Autre piège : ne pas comparer des instances de classes d'encapsulation avec == mais plutôt avec equals()

❑ Les pièges des flottants

- ❑ Les types float et double ne permettent pas de représenter des valeurs exactes

❑ Exemple

```
public class Monnaie {  
    public static void main(String[] args) {  
        System.out.println(2.00 - 1.1);  
    }  
}
```

- ❑ Les types double et float sont inadaptés là où un résultat exact est attendu

❑ Solutions

- ❑ Utiliser des entiers représentant des centimes ($200 - 110 = 90$)
- ❑ Utiliser la classe `BigDecimal` qui permet de représenter des grands nombres et d'interagir avec le type SQL `DECIMAL`

□ Les pièges des flottants

□ Classe BigDecimal

- Utiliser systématiquement le constructeur `BigDecimal(String)` et non pas `BigDecimal(double)` car sinon la précision est perdue avant la représentation

```
import java.math.BigDecimal;
```

```
public class Monnaie {  
    public static void main(String[] args) {  
        System.out.println(new BigDecimal("2.0").subtract(new BigDecimal("1.1")));  
    }  
}
```

- Pour cet exemple, la solution n'est pas particulièrement élégante et est un peu plus lente
- Pour des applications financières dans lesquelles on veut des résultats exacts et la maîtrise des détails des méthodes utilisées pour arrondir notamment, ces classes peuvent être utilisées

☐ Les pièges des flottants

- ☐ Ecrire un petit programme qui permet de résoudre le problème suivant
 - ☐ Vous avez un euro en poche et vous arrivez devant un étal avec des boîtes de bonbons à 10 c€, 20c€, 30 c€, et ainsi de suite jusqu'à 1 €. Vous en achetez un de chaque en commençant par ceux qui coûtent 10 c€ jusqu'à ce que vous n'ayez plus assez d'argent pour acheter le prochain bonbon de l'étal.
 - ☐ Combien de bonbons avez-vous acheté ?
 - ☐ Combien vous reste-t-il en poche ?
 - ☐ Tester dans un premier temps en commettant l'erreur de choisir le type double, puis écrivez deux solutions (une avec les centimes et l'autre avec BigDecimal)

□ Les pièges des flottants

- Ne jamais faire des tests d'égalité sur des flottants

```
public class FloatingPointsEquality {  
    public static void main(String[] args) {  
        double a = 6.6 / 3.0;  
  
        if (a == 2.2) {  
            System.out.println("Normal !");  
        } else {  
            System.out.println("WTF !");  
        }  
  
        System.out.println(a);  
    }  
}
```

□ Les pièges des flottants

- Ne jamais faire des tests d'égalité sur des flottants
- Solution

```
public class FloatingPointsEquality {  
    private static final double EPSILON = 1e-6;  
  
    public static void main(String[] args) {  
        double a = 6.6 / 3.0;  
  
        if (a >= (2.2 - EPSILON) && a <= (2.2 + EPSILON)) {  
            System.out.println("Normal !");  
        } else {  
            System.out.println("WTF !");  
        }  
  
        System.out.println(a);  
    }  
}
```

Types énumérées

- enum
- Possèdent des méthodes :
 - values()
 - name()

```
import javax.swing.JOptionPane;

public class EnumTest {

    public enum Jour {
        DIMANCHE, LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI}

    public static void main(String[] args) {
        Jour[] tabJours = Jour.values();
        Jour jour = tabJours[(int)(Math.random()*tabJours.length)];
        String msg;

        switch (jour) {
            case SAMEDI:
            case DIMANCHE:
                msg = "Vive le week-end !";
                break;
            case LUNDI :
                msg = "Les lundis sont nuls";
                break;
            default:
                msg="Vivement le week-end !";
                break;
        }

        JOptionPane.showMessageDialog(null, jour.name() + " : " + msg);
    }
}
```

☐ Types énumérées

- ☐ Les enums sont implémentés comme des classes
- ☐ Ils peuvent posséder
 - ☐ un constructeur
 - ☐ des méthodes
- ☐ Ecrire une classe PoidsSurPlanetes qui demande la masse sur terre et qui affiche le poids sur toutes les planètes du système solaire (ou pour avoir un résultat plus parlant la masse équivalente sur terre)

```
public enum Planet {
    MERCURE (3.303e+23, 2.4397e6),
    VENUS   (4.869e+24, 6.0518e6),
    TERRE   (5.976e+24, 6.37814e6),
    MARS    (6.421e+23, 3.3972e6),
    JUPITER (1.9e+27, 7.1492e7),
    SATURN  (5.688e+26, 6.0268e7),
    URANUS  (8.686e+25, 2.5559e7),
    NEPTUNE (1.024e+26, 2.4746e7);

    private final double mass; // en kg
    private final double radius; // en mètres
    Planet(double mass, double radius) {
        this.mass = mass;
        this.radius = radius;
    }
    private double mass() { return mass; }
    private double radius() { return radius; }

    // Constante gravitationnelle universelle (m3 kg-1 s-2)
    public static final double G = 6.67300E-11;

    public double surfaceGravity() {
        return G * mass / (radius * radius);
    }

    public double surfaceWeight(double otherMass) {
        return otherMass * surfaceGravity();
    }
}
```

□ Classes anonymes et listeners





□ Programmation événementielle

- Comment associer une action à un appui sur le bouton ?
- Swing met en place (de façon *presque* transparente pour l'utilisateur) un thread qui récupère les événements de la souris et du clavier en provenance du système d'exploitation
- Les événements sont ensuite transmis au composant concerné
- L'utilisateur peut configurer les différents composants pour être notifié quand un événement se produit et pour déclencher les traitements associés

cf. quelques transparents plus loin pour le lancement d'un programme swing



Classes anonymes et listeners

-  Les objets qui veulent être prévenus lorsqu'une action sur un élément d'IHM se produit doivent s'enregistrer auprès de ces composants
 -  c'est le concept d'événement listener (observateur d'événement)
-  Pour que les composants d'IHM soient indépendants de leurs utilisateurs, c'est aux utilisateurs de s'adapter et de présenter un type défini par les composants
 -  Les observateurs doivent implémenter une interface

☐ Classes anonymes et listeners

- ☐ Associer une action à un bouton
 - ☐ La classe JButton définit une méthode addActionListener qui prend en paramètre un objet de type ActionListener
 - ☐ ActionListener est une interface qui définit la méthode
void **actionPerformed**([ActionEvent](#) e)
 - ☐ L'objet e de type(ActionEvent) permet de connaître la source de l'événement ou les modificateurs (touches Shift, Ctrl, ...)
 - ☐ Il existe deux grandes alternatives pour associer une action à un bouton
 - ☐ Implémenter l'interface ActionListener dans la classe de gestion de l'IHM
 - ☐ Créer une instance d'une classe anonyme implémentant l'interface ActionListener

Java

```
public class PremierBouton implements ActionListener {
    JLabel monLabel;

    public void go(){
        JFrame cadre = new JFrame();
        cadre.setLayout(new FlowLayout(FlowLayout.LEFT));

        JButton monBouton = new JButton("Mon bouton");
        monBouton.addActionListener(this);
        monLabel = new JLabel();

        cadre.add(monBouton);
        cadre.add(monLabel);
        cadre.setSize(300,60);
        cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        cadre.setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        monLabel.setText("Hello !");
    }

    public static void main(String[] args) {
        new PremierBouton().go();
    }
}
```

Ajout de l'objet courant à la liste des "listeners" du bouton

Implémentation de l'interface
ActionListener

Il faut une instance de la classe
PremierBouton pour recevoir les
événements

```
public class BoutonClasseAnonyme {

    public static void main(String[] args) {
        JFrame cadre = new JFrame();
        cadre.setLayout(new FlowLayout(FlowLayout.LEFT));

        JButton monBouton = new JButton("Mon bouton");
        final JLabel monLabel = new JLabel();
        monBouton.addActionListener(new ActionListener(){
            @Override
            public void actionPerformed(ActionEvent e) {
                monLabel.setText("Hello !");
            }
        });

        cadre.add(monBouton);
        cadre.add(monLabel);

        cadre.setSize(300,60);
        cadre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        cadre.setVisible(true);
    }
}
```

La variable monLabel doit être déclarée **final** pour être accessible depuis la classe anonyme

Classe anonyme : le corps de la classe est déclaré au moment de l'instanciation

☐ Classe anonyme

On crée une nouvelle instance d'une classe anonyme (elle n'a pas de nom) qui implémente l'interface ActionListener

```
final JLabel monLabel = new JLabel();  
  
monBouton.addActionListener(new ActionListener(){  
    @Override  
    public void actionPerformed(ActionEvent e) {  
        monLabel.setText("Hello !");  
    }  
});
```




final car monLabel doit rester lié à l'instance de la classe anonyme même quand on sort de la méthode courante (la variable perd son statut de "variable locale")

□ Classe anonyme - avantages

- Les méthodes d'une classe anonyme peuvent utiliser les variables et les méthodes d'instance de la classe englobante (y compris les variables privées)
- Les méthodes d'une classe anonyme peuvent utiliser les variables locales et les paramètres de la méthode dans laquelle la classe anonyme est définie s'ils sont déclarés **final**
- Le code du comportement associé à un composant est regroupé avec son instantiation
- Il n'y a pas besoin de trouver un nom pour la classe anonyme (dans une application complexe cela pourrait conduire à un grand nombre de classes avec des noms semblables)



Exercice

-  Modifier le programme précédent en utilisant une variable d'instance pour le label
-  Utiliser une classe anonyme pour écouter les événements du bouton
-  Le bouton contiendra le texte "Quelle heure est-il ?" et le label affichera l'heure au format 23:12:46 (utiliser la méthode `String.format()`)

□ Lancement d'un programme Swing

```
public class PremierFrame {

    private static void createAndShowGUI() {
        JFrame cadre = new JFrame("Titre du cadre");
        cadre.add(new JButton("Mon bouton"));
        cadre.add(new JLabel("Mon Label"));
        cadre.setSize(300,60);
        cadre.setVisible(true);
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

Classe anonyme qui implémente l'interface Runnable dont la méthode run() sera appelée depuis le thread qui distribue les événements aux composants

