Développement orienté objets
R2.01

Isabelle Borne
Régis Fleurquin
Lucie Naert
Brendan Le Trionnaire

Organisation pratique

- La ressource R2.01 se déroule sur 2 périodes : semaines 4 à 12 ; puis semaines 14 à 22
- Un contrôle terminal écrit par période (semaines 13 et 24)
- Un contrôle continu par période
- Remise des TPs individuellement chaque semaine sur la zone Moodle.
- Chaque semaine :
  - 1 cours en amphi de 45 minutes
  - 1 TD de conception
  - 1 TD de programmation
  - 1 TP de conception-programmation

Objectifs de la ressource R2.01

- Initier à la programmation objets :
  - Introduire les concepts de la programmation par objets
  - Introduire les principes de modélisation, de programmation et de documentation des classes
  - Mettre en œuvre ces concepts et principes avec le langage de programmation Java
- Comprendre le paradigme des objets

Programmer avec des objets

La programmation par objets c'est

- Programmer par simulation
- Une façon de voir le monde, différente des approches procédurale ou algorithmique.
- Définir des objets qui communiquent par envoi de messages.

Cours n°1

1.Origines de la programmation par objets

2. Introduction aux 4 concepts fondamentaux

- Encapsulation des objets
- Abstraction et instanciation des classes
- Envoi de messages
- Héritage

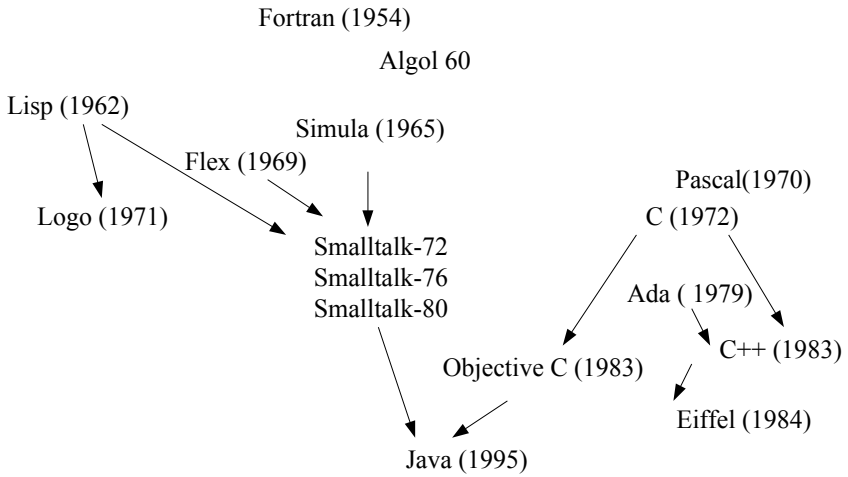
3. Tableau d'objets, à plusieurs dimensions et rappel conditionnelle

4. La machine virtuelle Java

1. Origines de la programmation par objets

- Dans les années 60 Alan Kay entreprend la conception d'un ordinateur portable multimédia.
- Inspiré par les travaux des norvégiens sur le langage Simula, la première version du langage Smalltalk sort au début des années 70 du « Learning Research Group » de Xerox à Palo Alto .
- Dan Ingalls en 1974 met au point le système de multi-fenêtrage chez Xerox
- Apple est créé en 1976, le Macintosh sort en 1980 avec un système de multi-fenêtrage ….

Arbre généalogique des langages à objets



Les origines du langage Java

- Le projet démarre en 1991 dans une équipe de Sun Microsystems dirigée par James Gosling (Green project)
- 1993 : arrivée du protocole http et du navigateur Mosaic
- L'arrivée d'Internet pour le marché grand public ré-oriente le projet vers le développement d'applications télé-chargeables et exécutables par les navigateurs web.
- 1995 : sortie de Java 1.0 (Oak) avec pour cible les PDA. Java devient rapidement populaire avec les Applets.
- Sun et Netscape intègre la technologie Java dans leur navigateur.

Les versions se succèdent

- 1998 Java 2
- 2004 : Java 5
- La société Oracle a racheté Sun en 2009.
- 2011 : Java 7
- 2014 : Java 8 ; 2016 : Java 9 ; 2018 : Java SE 11, ...

Les aspects fondamentaux ont peu changé

Les bibliothèques standards ont beaucoup évolué

Qualités recherchées par Java

- Simplicité
  - Par rapport à des langages comme C ou C++
- Sécurité
  - Exécution dans des navigateurs
- Robustesse
  - Résistance aux changements de conditions
- Portabilité
  - Fonctionnement dans différents environnements d'exécution

2- Les 4 concepts fondamentaux de la programmation par objets

- L'encapsulation
- L'abstraction et l'instanciation
- L'envoi de messages et le polymorphisme
- L'héritage (sera vu plus tard)

Encapsulation des objets

- Les objets sont caractérisés par leur état et les opérations qu'ils peuvent effectuer sur cet état.
- L'objet peut posséder des composants et l'état correspond alors à l'état de ses composants.
- L'ensemble des opérations applicables à un objet définit son protocole de communication.

Un objet encapsule un état et des opérations

Un premier objet

- Un objet qui représente un compte bancaire a comme état :
  - Un numéro
  - Un solde
  - Un découvert autorisé
- Et pour protocole de communication :
  - déposer un montant
  - retirer un montant
  - obtenir son solde
  - transférer un montant sur un autre compte

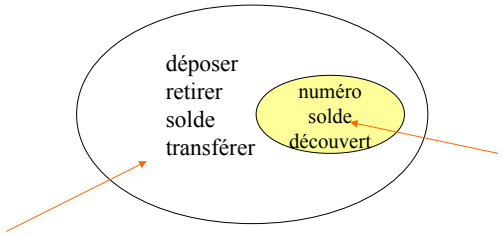
Abstraction et type abstrait

- Pour que les modifications d'un système complexe puissent se faire en toute sécurité, aucune partie de ce système ne doit dépendre des détails internes d'une autre partie.

> Principe (*information hiding*) :
> cacher l'implémentation des objets en ne divulguant que leur interface avec le monde extérieur.

- Les utilisateurs d'un objet n'ont pas besoin d'avoir accès à la représentation ou à l'implémentation des opérations.

Une vue des comptes bancaires



Structure privée d'un compte

Interface publique d'un compte

Classes et instances

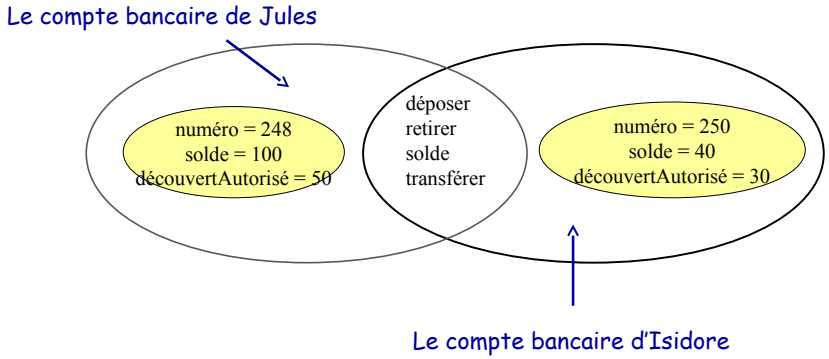
- Classe (type) : une description d'un ensemble d'objets avec des caractéristiques, attributs et comportements similaires.
- La classe est l'abstraction qui capture les attributs et les opérations communs à un ensemble d'objets
- Instance : un objet individuel qui est décrit par une classe particulière et qui est un membre de cette classe.
- Les opérations définies par une classe s'appellent des méthodes.

Classes et instances : principe

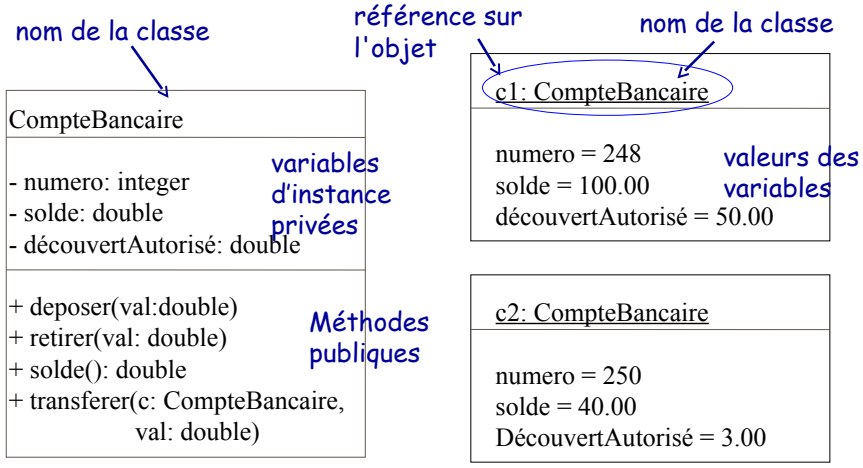
Tout objet est l'instance d'une classe

- La classe définit le savoir-faire de ses instances.
- Tous les objets d'une classe donnée partagent le même savoir-faire, c'est-à-dire le même ensemble de méthodes.
- Il est important de décrire le comportement des classes en termes de responsabilités.

Instances d'une classe `CompteBancaire`



Notation UML des classes et des instances



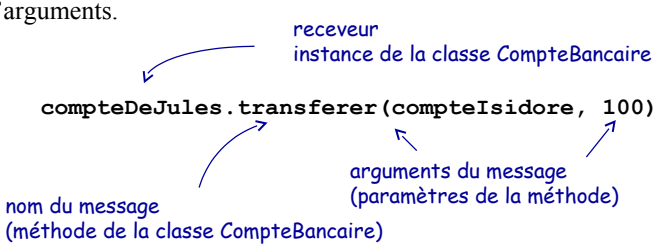
Envoi de message : principe

Les objets communiquent par envoi de messages.

- Pour demander une information à un objet on lui envoie un message
- Pour demander qu'un objet effectue une opération on lui envoie un message.
- Pour réaliser une opération, un objet peut envoyer un message à un autre objet : il délègue une partie de l'opération à cet objet.
- Une méthode est exécutée quand l'objet reçoit le message correspondant.

Structure d'un envoi de message

- Un envoi de message est composé d'un objet, le receveur, et d'un message
- Syntaxe Java : `receveur.message`
- Le message est composé d'un nom correspondant à un nom de méthode défini dans la classe du receveur et d'un ensemble d'arguments.



Autres exemples de messages

```
compteJules.deposer(100)

compteIsidore.retirer(20)


System.out.println(compteDeJules.solde())
```


Résolution de l'envoi de messages

Quand un objet reçoit un message il y a :

- Recherche de la méthode correspondant au nom du message dans la classe de l'objet receveur ;
- Évaluation des arguments du message ;
- Application de cette méthode avec les arguments du message dans le contexte de l'objet receveur.

```
compteJules.transferer(compteIsidore, 100)
```

La méthode correspondante dans la classe CompteBancaire est :

```
public void transferer(CompteBancaire destinataire, double
   val){…}
```


Retour d'un envoi de messages

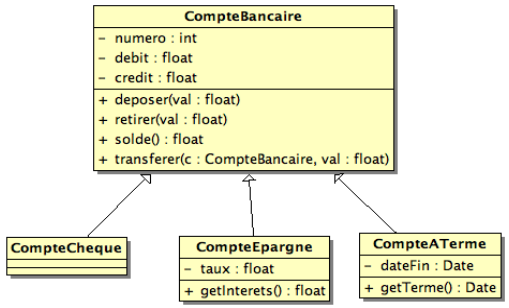
- L'exécution d'un envoi de message peut produire ou non une valeur de retour :
  - Les messages qui modifient un objet ne produisent en général pas de valeur de retour, on dit qu'ils font un *effet de bord*.
  - Les messages qui « calculent » une valeur retournent en général cette valeur en retour de l'envoi de message.
  - Dans un langage purement objet, la valeur de retour est un objet.
  - En Java la valeur de retour peut être un type primitif ou un objet.

Héritage (sera étudié en détail plus tard)

- Programmer en objets c'est programmer avec héritage.
- Toute classe est définie par rapport à une classe parente dont elle hérite des attributs et des méthodes.

superclasse

sousclasses



En résumé

- Programmer par objets c'est :
  - simuler une situation réelle (réifier = rendre objet)
  - définir des types abstraits de données (abstraire)
  - créer les objets concrets de l'application (instancier)
  - définir des opérations polymorphes (communiquer)
  - spécialiser - généraliser des classes d'objets (hériter)

3. Les tableaux

- Les tableaux font partie intégrante de la plupart des langages de programmation et possèdent leur propre syntaxe.
- En Java les tableaux ne sont pas véritablement des objets : instances d'une classe `Array` spéciale et cachée.
- Un tableau d'objets est une collection d'objets de taille fixe et dont les éléments sont tous du même type et sont des références aux objets.
- La taille d'un tableau ne peut ni être augmentée, ni être réduite.

Initialisation implicite des tableaux en Java

- Un des principaux objectifs de Java est la sécurité.
- Quand on crée un tableau d'objets, on crée en réalité un tableau de références, et chacune de ces références est automatiquement initialisée à une valeur particulière avec son propre mot clé : **`null`**.
- Il faut affecter un objet à chaque référence avant de l'utiliser et si on essaie d'utiliser une référence encore **`null`**, le problème sera signalé lors de l'exécution.
- On peut aussi créer des tableaux de variables de type primitif. À nouveau, le compilateur garantit l'initialisation car il met à zéro la mémoire utilisée par ces tableaux.

Danger des initialisations implicites

- Le développeur n'a pas une maîtrise parfaite de son programme et de ce qui se passe réellement.
- Cela conduit à produire du code qui n'est pas fiable.

D'où la règle à respecter :

Toujours initialiser explicitement les objets et les variables utilisés dans son programme

Déclaration et création d'un tableau
(exemple de tableau de primitifs)

1) Déclarer une variable de type tableau

```
int[] tableauDeNombres
```

2) Définir la taille du tableau et allocation de l'espace

```
tableauDeNombres = new int[4]
```

3) Affecter des valeurs aux éléments d'un tableau

```
tableauDeNombres[0]= ...
```


Obtenir la taille d'un tableau

- La taille d'un tableau = son nombre d'éléments.
- On peut l'obtenir en utilisant une sorte d'attribut :

```
tableauDeNombres.length
```

- **Attention : on ne peut pas changer la taille d'un tableau.**
- Exemple d'initialisation d'un tableau :

```
for (int i = 0; i < tableauDeNombres.length; i++){
        tableauDeNombres[i] = i;
}
```

Si la taille du tableau est 4 quels sont les valeurs des éléments ?

Exemple de déclaration et création d'un tableau d'objets

```
String[] monTableau ;
```

//On a créé un tableau de chaînes de caractères mais ce tableau n'est

// pas encore utilisable car il n'a pas de taille !

```
monTableau = new String[nombreDObjets] ;
```

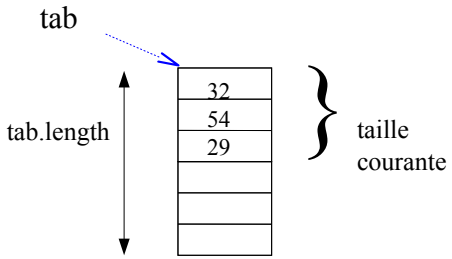
//Il est maintenant utilisable et indexé de 0 à nombreDObjets-1.

Que fait cet exemple ?

```
int n = 10 ;
CompteBancaire[] listeDeComptes ;
listeDeComptes = new CompteBancaire[n] ;
listedeComptes[0]= new CompteBancaire(42);
listedeComptes[1]= new CompteBancaire(44);
```

Tableau partiellement rempli

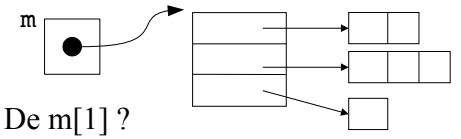
- Un tableau ne peut pas changer de taille à l'exécution.
- Pour un tableau partiellement rempli, une variable doit contenir la « taille courante » du tableau.



Tableaux à plusieurs dimensions

- Java considère les tableaux à plusieurs dimensions comme des tableaux de tableaux.
- Les sous-tableaux peuvent être de tailles différentes.

```
int[][] m;
m = new int[3][];   crée un tableau de 3 tableaux d'entiers
m[0]= new int[2];   crée le premier vecteur de m de taille 2
m[1]= new int[3];   crée le deuxième vecteur de m de taille 3
m[2]= new int[1];   crée le troisième vecteur de m de taille 1
```



Taille de m ? De m[1] ?

Exemple du tutoriel Java

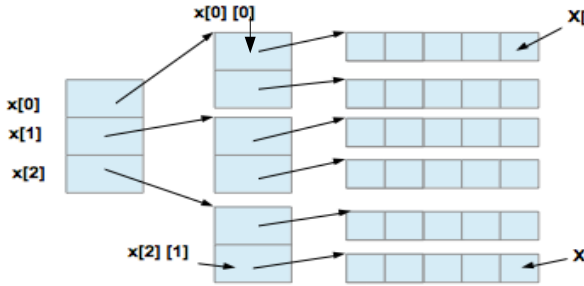
```
public class TwoDimArrayDemo {
  public static void main(String[] args){
    String[][] names = {{"Mr. ", "Mrs. ", "Ms. "},
                        {"Smith", "Jones"}};
    System.out.println(names[0][0] + names[1][0]);
    System.out.println(names[0][2] + names[1][1]);
    }
}
```

Qu'est-ce que l'on obtient à l'exécution ?

Tableau à 3 dimensions

- Un tableau à 3 dimensions est un tableau à 1 dimension dont chaque élément est lui-même un tableau à 2 dimensions.

```
int[][][] x = new int[3][2][5] ;
```




```
x.length =
x[0].length =
x[2][1].length =
```

Structures de contrôle : conditionnelle

```
if (condition) {
      instructionV1 ;
        …
      instructionVN ;
 }
 else {
      instructionF1 ;
        …
      instructionFN ;
}
```

Plusieurs expressions ou Messages en séquence

Conditionnelles imbriquées

```
if (condition1) {
   les messages et opérations si condition1 est vrai
}
else if (condition2) {
    les messages et opérations si condition2 est vrai
}
else if (condition3) {
    les messages et opérations si condition3 est vrai
}
else {
     les messages et opérations si conditions 1,2 et 3
  sont faux
}
```


4. Machine virtuelle et utilisation de Java

(http://download.oracle.com/javase/tutorial/getStarted/intro/definition.html)

Le code source est écrit comme du texte dans un éditeur de texte (JEdit par exemple) et sauvé dans un fichier d'extension « .java »

Notion de machine virtuelle

- La **machine virtuelle Java** (anglais *Java virtual machine* abr. JVM) est un environnement d'exécution des programmes compilés sous forme de bytecode Java.
- Elle assure l'indépendance du matériel et du système d'exploitation.
- La machine virtuelle interprète le bytecode produit par le compilateur Java.
- La machine virtuelle comporte un ramasse-miettes (anglais garbage collector) qui est responsable du recyclage de la mémoire allouée puis inutilisée.

La plate-forme Java