BPOO - Sujet TD 2

Dut/Info-S2/M2103

Table des matières

PreReq	Cours 1 : approche objet. S1.
ObjTD	**Utiliser des objets en Java**.
Durée	**1 séance** de 1,5h

1. Préliminaires et rappels

1.1. Déclaration de sous-programmes en Java (principes)

1.1.1. En-tête de sous-programme

Un en-tête de sous-programme définit :

1. une spécification pour le programmeur qui doit développer le sous-programme,
2. une "mode d'emploi" pour le programmeur qui doit utiliser ce sous-programme.

Cet en-tête définit :

- l'identificateur du sous-programme,
- le nombre ET le type ET l'ordre des paramètres formels : une suite de couples `type`

`identificateur` séparés par des virgules,

- un éventuel type de valeur retournée, void si rien n'est retourné.

  Notez que "void", du point de vue du compilateur, ne veut pas dire "rien" mais est un type qui n'a aucune valeur et veut donc dire "pas de valeur de retour".

La **signature ou prototype** du sous-programme est ce qui permet de le distinguer ET de le reconnaître. Elle est formé de :

- l'identificateur du sous-programme,
- le nombre ET le type ET l'ordre des paramètres formels.

Exemple :

```
/**
* Calcul de la puissance puiss de la valeur v (calcul de v puissance puiss).
* Ne fonctionne que si puiss >= 0
*
* @param    v      valeur dont on cherche la puissance
* @param    puiss  puissance à laquelle on va élever v
* @return   v élevé à la puissance puiss
**/
double  puissance (double v, int puiss) {
        double resultat ;
        ...
        return resultat;
}
```

1.1.2. Appel d'un sous-programme

L'appel d'un sous-programme fait intervenir :

- l'identificateur du sous-programme,
- des paramètres effectifs en nombre ET ordre ET type correspondant au prototype du sous-programme.

Les paramètres effectifs peuvent être toute expression donnant une valeur du type attendu.

Exercice Pour les appels suivants,expliquez si : i) l'appel est correct, ii) les calculs effectués pour réaliser l'appel. On considérera les appels corrects même si nous reviendrons plus tard sur la syntaxe exacte d'appel.

1. Q : `double puissance (double v, int puiss)`

```
double x;
x = puissance (10.5, 1+1);
```

- Correct ?
- Calculs ?

2. Q : `double puissance (double v, int puiss)`

```
double x;
x = puissance (10.5, 2, 2+2);
x = puissance (10.5);
```

- Correct ?
- Calculs ?

3. Q : `double puissance (double v, int puiss)`

```
double x;
x = puissance (10, 2);
```

- Correct ?
- Calculs ?

4. Q : `double puissance (double v, int puiss)`

```
double x;
x = puissance (10.5, 2.5);
```

- Correct ?
- Calculs ?

5. Q : `double puissance (double v, int puiss)`

```
int val;
val = puissance (10, 2);
```

- Correct ?

- Calculs ?

1.2. Méthodes static en Java

Il existe des sous-programmes qui existent "en eux-mêmes". Ont peut les décrire et on souhaite pouvoir les programmer pour les appeler. Exemple : sinus, cosinus, calcul de puissance, obtenir l'heure du système, …

En java :

1. Tout identificateur déclaré ne peut l'être qu'à l'intérieur d'une unité de définition appelée classe.
2. On peut déclarer un sous-programme rattaché à une classe, appelée méthode **de classe** ou méthode **statique**.
3. Elle s'écrit précédée du mot-clef **`static`**
4. Pour l'appeler, il faut utiliser le nom de la classe devant l'appel : `NomDeClasse . identificateurmethode(…)`

Exemple :

```
public class ClasseLibrairie {
        /**
        * Calcul de la puisssance puiss de la valeur v (calcul de v puissance
puiss).
        * Ne fonctionne que si puiss >= 0
        *
        * @param    v      valeur dont on cherche la puissance
        * @param    puiss  puissance à laquelle on va élever v
        * @return   v      elévé à la puissance puiss
        **/
        public static double  puissance (double v, int puiss) {
                double resultat=0 ;
                ...
                return resultat;
        }
        public static void main(String argv[]) throws Exception  {
                double x;
                x = ClasseLibrairie.puissance (10.5, 2);
        }
}
```

Par la suite, lorsque nous écrirons des classes, on séparera clairement :

- des classes "librairies" qui définissent uniquement des méthodes statiques. Ex Math et System ou

une classe contenant une fonction main().

- des classes décrivant des objets définissant uniquement des méthodes non statiques.

Cette dichotomie que l'on recherchera ne sera pas tous toujours possible ou souhaitable. Nous l'appliquerons le plus possible. Exception : les sujets de TD ou TP (pour diminuer le volume des exemples).

Notons que les librairies Java mélangent souvent les deux type de méthodes et d'attributs. Faire attention en lisant la documentation.

Autre exemple :

```
public static void main(String argv[])  {
        double x, y, xpowy;
        long debut, lg;

        debut = System.currentTimeMillis();  // instant de démarrage en milisecondes
                                             // calculé depuis January 1, 1970 UTC
- 0h.)
        x = 100 * Math.random();    // Tirage au sort d'un nombre entre 0 et 100
        y = 10 * Math.random();     // Tirage au sort d'un nombre entre 0 et 10
        xpowy = Math.pow(x, y);     // Calcul de x puissance y
        System.out.println("Résultat : " + x + " pow " + y + " => " + xpowy);
        lg = System.currentTimeMillis() - debut; // Calcul durée du programme
        System.out.println("Durée : " + lg);
}

Exécution :
 Résultat : 80.6991463652636 pow 1.7813938064916623 => 2493.931501272706
 Durée : 3
 Exit code: 0
```

1.3. Constantes (attributs static)

De la même manière, on peut aussi définir des attributs de classe (static) : attributs accessibles à partir de la classe.

Ils sont très souvent déclarés **public** (accessibles), **static** (de classe), **final** (non modifiables).

Exemple : Math.PI → Valeur de approchée de pi (double) : 3,141592…

Exemple : Math.E → Valeur de approchée de e (double) : e = 2,718281…

1.4. Autres cas particuliers (attributs static)

Certaines classes définissent des attributs de classe (static) "modifiables", mais à ne modifier qu'avec précaution.

Ils donnent accès à des ressources systèmes.

Exemple :

- System.out → accès au dispositif de sortie standard par défaut du programme (objet de la classe PrintStream).
- System.in → accès au dispositif d'entrée standard du programme (objet de la classe BufferedInputStream).
- System.err → accès au dispositif de sortie d'erreur standard du programme (objet de la classe PrintStream).

```
System.in
|-----|     Une classe (un indicateur -> majuscule)
|---------| Un attribut de la classe (indicateur -> pas de parenthèses après le
"in").
```

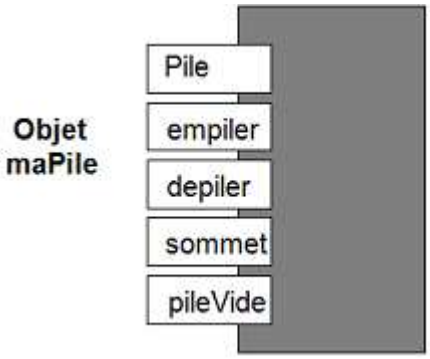
---

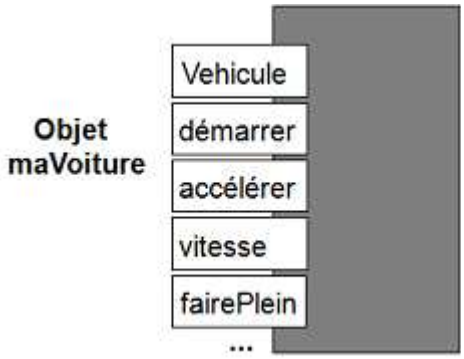
2. Utiliser des objets en Java

2.1. Rappel objets

Objet = identité + état + comportement

- Identité : Un objet forme un tout identifiable
- Un état : ensemble de propriétés/caractéristiques (attributs) ayant une valeur
- Comportement : ensemble des traitements (méthodes) que peut accomplir l'objet si on le lui « demande »





Par principe :

- Un objet **encapsule** son état
  - Etat : accessible que par l'objet lui-même (méthodes)
  - Principe objet majeur à respecter ⇒ **abstraction**
- Un objet reçoit des **messages** ⇒ appel de méthode appliqué à l'objet
- Un objet est créé à partir d'une classe qui fournit sa description.

Un objet reçoit des messages ⇒ appel de méthode appliqué à l'objet

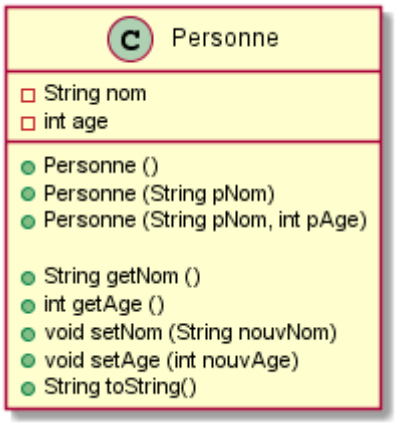
- Idée : c'est l'objet qui est "actif" et qui réagit (exécute la méthode)
- On peut "penser" objet comme cela …
- … mais la mise en oeuvre est différente.

2.2. Principe des classes en Java

Pour décrire une classe d'objet en Java, on définit une classe qui déclare :

- des attributs pour les objets (private),
- des méthodes pour les objets (public) + éventuellement des méthodes internes (private),
  - des méthodes observateurs (appelées aussi pour certaines getters),
  - des méthodes transformateurs (appelées aussi pour certaines setters),
- un/des constructeurs (public) pour construire des objets.

Exemple :




```
class Personne
{
                // Attributs
                private  String nom;
                private  int age;

        // Constructeurs
        public  Personne ()  { ... }
        public  Personne (String pNom)   { ... }
        public  Personne (String pNom, int pAge) { ... }

        // Méthodes
        public  String getNom () { ... }
        public  int getAge ()   { ... }
        public  void setNom (String nouvNom) { ... }
        public  void setAge (int nouvAge)  { ... }
        public  String toString() { ... }
} // Fin Classe Personne
```

Principes de syntaxe :

1. Attributs : **toujours** déclarés private. Règle à appliquer pour construire des objets (et non des enregistrements).
2. Constructeur :
   - Porte le nom de la classe.
   - Pas de valeur de retour.
   - Plusieurs constructeurs possibles (surcharge) : différenciés par leur signature.

La classe définit un type :

1. On peut déclarer des variables de ce type.
2. Ces variables contiennent des **références** vers des objets de ce type (une référence est typée).

Utiliser l'opérateur **new** : Exemple `new Personne ("Jean", 39);`

2.3. Création d'objet

- réserve la mémoire nécessaire : environ le nombre d'octets pour stocker les attributs définis par la classe,
- initialise chaque attribut (`0` pour les nombres, `false` pour les booleans, `null` pour les objets et les tableaux),
- applique le constructeur spécifié (signature à partir des paramètres),
- **renvoie la référence** de l'objet créé dans le tas.

```
 new  Personne  ("Jean", 39)
|-------------| ..................> L'objet à créer -> avec les attributs définis
par la classe

 new  Personne  ("Jean", 39)
     |----------------------| ....> Signature du constructeur à appliquer ->
Personne (String, int)
```

Autre façon de créer des objets ? oui, méthode clone() appliquée à un objet … mais sera vu plus tard.

2.4. Identité d'objet en Java

L'identité d'objet en Java est automatique → c'est la référence de l'objet créé par **new**.

Elle est unique et permet de désigner l'objet indépendamment des valeurs des attributs.

Les variables objet contiennent la référence vers l'objet … donc son identité.

2.5. Envoi de message

Syntaxe : `expressionReferenceObjet . identificateurMethode ( [parametreEffectif1, …] )`

2.6. Exemple

```
class ClasseEssai
{
        public static  void main (String  argv[]) {
                int i;
                Personne  p1, p2, p3;
                Personne tabPers [];

                p1 = new Personne ("Jean", 39);
                p2 = new Personne ("Jean", 39);
                p3 = new Personne ("Pierre");

                p3.setAge (45);
                p3.setNom ("Durand");

                System.out.println (p1.toString());
                System.out.println (p1.getNom () );
                System.out.println (p2);   // Implicitement fera appel à
p2.toString()
                System.out.println (p3);   // Idem

                if  (p1 != p2)
                        System.out.println ("Non Egalité des références");

                if (p1.getNom().equals(p2.getNom()))
                        System.out.println ("Egalité des noms des Personnes");

                tabPers = new Personne [3];
                tabPers[0] = p1;
                tabPers[1] = p2;
                tabPers[2] = new Personne ("Albert", 30);

                for (i=0; i<tabPers.length; i++) {
                        System.out.println (tabPers[i]);
                }
        }// Fin Main
}// Fin Classe ClasseEssai
```

3. Utilisation de classe

3.1. Exercice

Nous allons utiliser une classe Devinette pour jouer, cf. ci-après.

Au jeu de la "devinette" :

- On suppose une valeur entière à trouver, tirée au sort par le programme, entre une borne basse (10<>20) et haute (40<>50).
- L'utilisateur chercher la valeur par essais successifs, la machine répondant à chaque coup si le nombre recherché est supérieur ou inférieur au nombre soumis.
- Le jeu s'arrête quand la valeur est trouvée.

Exemple d'exécution :

```
Coup : (13-49) : 33 // ❶
Plus Haut ... Coup : (13-49) : 40 // ❷
Plus Haut ... Coup : (13-49) : 49 // ❸
Plus Bas ... Coup : (13-49) : 48
Plus Bas ... Coup : (13-49) : 41
Plus Haut ... Coup : (13-49) : 47
Plus Bas ... Coup : (13-49) : 44
Plus Haut ... Coup : (13-49) : 46
Bravo, en 8 coups.
```

❶ Saisie utilisateur de 33

❷ Saisie utilisateur de 40

❸ …

A faire :

1. Étudier la documentation de la classe Devinette (constructeurs/méthodes).
2. Un objet Devinette, instance de la classe Devinette, est un objet qui :
   - lors de sa création, détermine des bornes min et max au jeu et choisit la valeur à chercher,
   - dispose de méthodes permettant de jouer et d'interroger l'état de la partie (nombre de coups joués, résultat d'un coup, …).
3. Écrire un programme main permettant de faire une partie de devinette en utilisant la classe Devinette : tirage au sort des valeurs, demander une valeur comprise entre les deux bornes et répondre à l'utilisateur sur la validité de son coup, afficher « partie gagnée » à la fin ainsi que le nombre de coups joués (essais).
4. Écrire un autre programme main permettant de jouer successivement 3 parties de devinette. Écrire d'abord un sous-programme `void jouer (Devinette d)` permettant de "dérouler" une partie de devinette avec l'objet d en paramètre. Puis écrire le programme demandé (quelques lignes).

Dernière mise à jour 2014-02-10 01:23:26 CET