Algorithmique et programmation par objets

Inf F3
Licence 2 MIASHS
Université Grenoble Alpes

Jerome.David@univ-grenoble-alpes.fr

2023-2024

http://miashs-www.u-ga.fr/~davidjer/inff3/

Cours 5
Constructeurs, Packages et Modificateurs d'accès

- Sommaire
  - Initialisation des objets
    - Notions de constructeurs, surcharge, mot-clé this
  - Organisation du code en package
    - Notion d'espace de noms
    - Utilisation et déclaration de packages
  - Les modificateurs d'accès
    - Notions de contrôle d'accès, encapsulation
    - Modificateurs public, protected, private

Initialisation des objets

- Beaucoup d'erreurs de programmation viennent d'oublis :
  - d'initialisation : valeurs inattendues (ou valeur nulles)
  - De nettoyage : fuites mémoires
- Java propose
  - Les constructeurs : méthode automatiquement appelée lors de la création d'instances
  - Le ramasse-miette : mécanisme de récupération de la mémoire utilisée par des objets non utilisés

Initialisation via les constructeurs

- un constructeur est une méthode spéciale
  - Elle est automatiquement appelée à la création de l'objet (`new`), après allocation de la mémoire
  - Elle a exactement le même nom que la classe
    - La casse est prise en compte
  - Elle n'a pas de type de retour
    - Même pas de void
    - `new` retourne la référence vers l'objet nouvellement créé mais le constructeur lui même ne retourne aucune valeur

Exemple de constructeur

- Une classe Point avec un constructeur par défaut qui initialise toute nouvelle instance de point avec une abscisse et ordonnée à 1.0

```
class Point {
    double x;
    double y;

    /*
     * Constructeur qui initialise tout
     * nouveau point avec les coordonnées
     * (1.0,1.0)
     */
    Point() {
        x=1.0;
        y=1.0;
    }
}
```

Exercice

- Schématisez la mémoire utilisée par le code suivant :

```java
class Test {
    public static void main(String[] args) {
        Cercle c = new Cercle();
    }
}
```

```java
class Point {
    double x;
    double y;

    Point() {
        x=1.0;
        y=1.0;
    }
}
```

```java
class Cercle {
    Point centre;
    int rayon;

    Cercle() {
        centre=new Point();
        rayon=1;
    }

    double aire() {
        return Math.PI*rayon*rayon;
    }
}
```

Constructeurs avec paramètres

- Le constructeur sans paramètre est appelé « constructeur par défaut »
- On peut définir des constructeurs avec paramètres
  - Par contre, si c'est le seul défini, on est obligé de l'utiliser

```
class Point {
    double x;
    double y;

    Point(double abcisse, double ordonnee) {
        x=abcisse;
        y=ordonnee;
    }
}
```

~~`new Point();`~~

`new Point(3, 2.1);`

Surcharge de constructeurs

- Peut-on avoir plusieurs constructeurs ?
  - Exemple : je veux pouvoir créer un cercle de plusieurs manières :
    - Sans paramètre : centre (0;0) et de rayon 1.0
    - Avec seulement le centre ou le rayon de spécifié
    - Avec les deux
- Java permet de définir plusieurs constructeurs
  - C'est ce que l'on appelle **la surcharge**
  - Il faut juste que le nombre de paramètres ou les types des paramètres soient différents
    - **Question :** A votre avis pourquoi cette limitation ?

Exemple de surcharge de constructeurs

```java
class Cercle {

    Point centre;
    int rayon;

    Cercle() {
        centre=new Point(0,0);
        rayon=1;
    }
    Cercle(int r) {
        centre=new Point(0,0);
        rayon=r;
    }
    Cercle(Point c) {
        centre=c;
        rayon=1;
    }
    Cercle(Point c, int r) {
        centre=c;
        rayon=r;
    }
    double aire() {
        return Math.PI*rayon*rayon;
    }
}
```

```java
class Test {
    public static void main(String[] args) {
        Cercle c1 = new Cercle();
        Cercle c2 = new Cercle(4);
        Cercle c3 = new Cercle(new Point(1,2));
        Cercle c4 = new Cercle(new Point(1,2),3);
    }
}
```

Autres Exemples

- Ce que l'on ne peut pas faire
  - Avoir 2 constructeurs avec les mêmes paramètres
    - Même nombre, mêmes types dans le même ordre

```
class Point {
     double x;
     double y;

     Point() {
          x=1.0;
          y=1.0;
     }

     Point(double abcisse, double ordonnee) {
          x=abcisse;
          y=ordonnee;
     }

     Point(double rayon, double angle) {
          x=rayon*Math.cos(angle);
          y=rayon*Math.sin(angle);;
     }
}
```

Surcharge de méthode
Cas général

- En Java, on peut surcharger n'importe quelle méthode dans une classe donnée
- Par contre, on ne peut pas surcharger le type de retour….
  - Pourquoi ?

Les constructeurs par défaut

- Jusqu'à maintenant, nous avons défini des classes sans constructeur et nous avons tout de même instanciées…
- Si il n'y a pas de constructeur défini, Java en génère un automatiquement.
  - Il utilise celui de la super classe
  - On verra cela plus tard

Le mot clé this

- Il a deux usages :
  - C'est une référence vers l'instance « courante » sur laquelle le message que l'on traite est envoyé
    - Permet de lever l'ambiguité quand il y en a une
    - Permet de passer l'instance courante en paramètre
  - A la première ligne d'un constructeur, il permet d'appeler un constructeur de la même classe mais avec des paramètres différents

La référence this

- Pour lever les ambiguïtés
  - `this.x` désigne l'attribut
  - `x` désigne le paramètre

```java
class Point {
   double x;
   double y;

   Point(double x, double y) {
      this.x=x;
      this.y=y;
   }
}
```

Que se passe-t-il si j'écris dans le constructeur x=x ?

- Pour passer une référence sur l'instance courante en paramètre d'une autre méthode

```java
class Point {
   ...
   void dessinerSur(ZoneDessin d) {
     d.dessiner(this) ;
   }
   ...
}
```

```java
class ZoneDessin {
      ...
      void dessiner(Point p) {
           ...
      }
      ...
}
```

Appeler un constructeur dans un constructeur

- L'utilisation de `this(...)` permet d'appeler un constructeur avec des paramètres différents
  - S'utilise UNIQUEMENT dans un constructeur
  - Et UNIQUEMENT comme première instruction du constructeur

```
class Point {
   double x;
   double y;

   Point() {
      this(0.0,0.0); // appelle le constructeur Point
                     // avec deux paramètres
   }
   Point(double x, double y) {
      this.x=x;
      this.y=y;
   }
}
```

Ordre des initialisations

- On peut initialiser les attributs :
  - Au niveau de leur déclaration
  - Dans les constructeurs
- L'initialisation au niveau de la déclaration est faite avant l'appel au code du constructeur

```java
class Cercle {
   Point centre=new Point(0,0);
   int rayon=1;

   Cercle(Point c, int r) {
      centre=c;
      rayon=r;
   }
}
```

```java
class Test {
   public static void main(String[] args) {
      Cercle c = new Cercle(new Point(1,2),3);
   }
}
```

Dans le programme Test, 2 points vont être successivement instanciés

Contrôle d'accès et encapsulation

- La programmation par objets met l'accent sur la réutilisation du code par d'autre programmeurs
- On distingue :
  - Le concepteur d'une classe donnée
  - L'utilisateur de cette classe
- Le concepteur veut contrôler l'accès :
  - Aux attributs : afin de contrôler les valeurs qui sont mise dedans, pré-traiter les données, etc.
  - Aux méthodes : certaines méthodes n'ont pas besoin d'être ouvertes sur l'extérieur (i.e. en dehors de l'objet)
- Il faut aussi permettre d'organiser les classes de grande librairies
  - Ca facilite la maintenance, et aussi l'utilisation

Les mots clés du contrôle d'accès

- Java permet de spécifier la « visibilité » des attributs et méthodes
  - private, protected, RIEN, public
- Java permet d'organiser les classes en packages
  - Un package est un ensemble de classe rangées dans un même espace de noms
    - Pour simplifier dans un même dossier

Les espaces de noms

- Un espace de noms permet :
  - De regrouper des choses de la même famille
  - De lever les ambiguïtés de noms
- Exemples :
  - Le nom de famille est un espace de noms permettant de discriminer les personnes ayant le même prénom
  - Les domaines des sites web sont des espaces de noms permettant de différencier les pages html qui ont le même nom

Le package

- Il regroupe des classes sous le même espace de noms
  - Exemple : package `java.util` : Il contient des classes comme `Arrays`, `ArrayList` (tableau dynamique), `GregorianCalendar`, `Currency`
- Pour utiliser une classe d'un package :
  - On utilise son nom complet

```
class Test {
   public static void main(String[] args) {
      java.util.ArrayList tab = new java.util.ArrayList();

   }
}
```

  - On importe la classe, puis on utilise son nom simple

```
import java.util.ArrayList;
class Test {
   public static void main(String[] args) {
      ArrayList tab = new ArrayList();
   }
}
```

Organisation des classes en packages

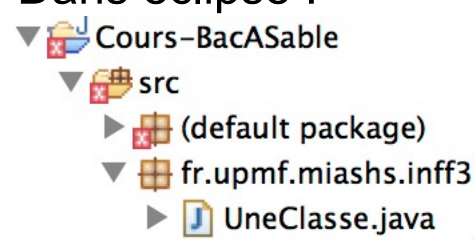
- On peut organiser ses classes en package
  - Il faut déclarer le package d'appartenance
    - Via `package nompackage;` placé en première ligne du fichier .java
  - Placer le fichier .java dans une hiérarchie de dossiers portant le nom du package

UneClasse.java

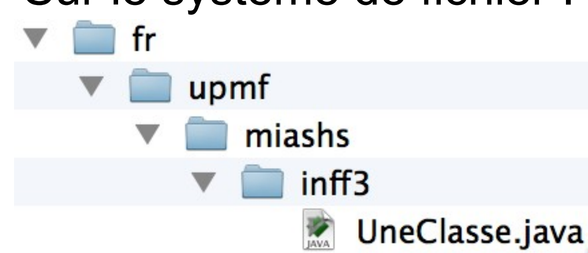
```java
package fr.upmf.miashs.inff3;

public class UneClasse {

}
```

Il faut que la classe soit publique pour être visible en dehors du package

Dans eclipse :



Sur le système de fichier :



Règles pour les noms de package

- Le nom de package sert d'espace de noms
  - Il doit être unique (si possible)
    - Utiliser quelque chose qui vous identifie
      - Exemples : fr.upmf.miashs.inff3, le nom de domaine de votre organisation à l'envers
  - Par convention, il ne contient que des lettres minuscules
- Qu'est ce qu'il se passe si deux classes on le même nom?
  - Si elles ont des packages différents, j'utilise leur nom complet
  - Si elles ont le même nom complet, je ne peux pas les utiliser ensemble dans le même projet

Les modificateurs d'accès

- Ils peuvent être placés devant une déclaration
  - De classe
  - De méthode
  - D'attribut (d'instance ou de classe)
- Il y a 4 modificateurs d'accès en Java

	classe	package	Sous-classe	Le monde
public	Oui	Oui	Oui	Oui
protected	Oui	Oui	Oui	Non
(rien)	Oui	Oui	Non	Non
private	Oui	Non	Non	Non

Règles pour choisir un modificateur

- Quand on a le choix, choisir le modificateur le plus restrictif
  - Utiliser private en priorité
  - Eviter d'utiliser public sur les attributs (sauf pour les constantes)
    - Pour donner un accès en lecture et/ou modification, on ajoute des méthodes → c'est ce qu'on appelle l'ENCAPSULATION
- Les membres publics définissent l'interface exposée de votre classe
  - Vous ne pourrez plus les modifier par la suite sous peine de rendre les codes qui utilisent votre classe non compilables...

Modificateurs sur les classes

- Les modificateurs `private` et `protected` ne sont pas possibles pour une classe*
- Une classe publique est accessible à tous
- Une classe sans modificateur n'est accessible que dans le package auquel elle fait partie
- On ne peut avoir qu'une classe publique par fichier .java
  - De toute façon, généralement, on ne déclare qu'une classe par fichier .java

* Sauf cas particulier non abordé dans le cours

Exercice

- Comment faire pour que l'on ne puisse
  - Pas changer le nom et le prénom d'une personne après création
    - Mais y avoir accès
  - Pas mettre un âge négatif
  - Mais assurer que si une instance x à un conjoint y alors y aura comme conjoint x (et nul autre)

```java
public class Personne {
    String nom;
    String prenom;
    int age;
    Personne conjoint;
}
```

Exercice

- Ajoutez les modificateurs d'accès adéquat sur la classe `DictionnaireStringInt` du TD3

```
public class DictionnaireStringInt {
      String[] cles = new  String[10];
      int[] valeurs = new int[10];
      int nbElements = 0;

      void  ajouterModifier(String cle, int valeur ) {...}

      int  rechercherValeur(String cle) {...}

      int  rechercherIdx(String cle) {...}

      void agrandir() {...}

      int  supprimer(String cle) {...}

      String toString() {...}

      int getNbElements() {...}

}
```