CÉGEP
MARIE-VICTORIN

420-210-MV
Programmation orientée objet

Samuel Fostiné

Exercices

- On vous charge de créer un système de gestion de notes pour les cours du département d'informatique.
- On vous donne les requis suivants:
- On veut commencer par ajouter le cours de programmation orientée Objet dont le code est 420-210-MV
  - Le professeur de ce cours est
  - Samuel Fostiné
  - Son identifiant est 09814
  - Le cours contient 5 étudiants.
  - Chaque étudiant a
    - un nom
    - un prénom
    - 4 notes pour le cours et chaque note est sur 25

Exercices - Continuation

- On veut pouvoir calculer la moyenne des notes des étudiants du cours programmation orientée objet
- On veut pouvoir associé une cote à chaque étudiant du cours, en se basant sur le format suivant:
  - A+ pour une note supérieure à 90
  - A pour une note inférieure à 90, mais supérieure à 80
  - B+ pour une note supérieure à 75, inférieur à 80
  - B pour une note entre 70 et 75
  - C+ pour une note entre 65 et 70
  - C pour une note entre 60 et 65
  - D pour une note en dessous de 60

new

- L’opérateur ***new*** permet d’instancier une classe, c’est-à-dire de créer une instance de cette classe.
- Ex: Etudiant etudiant = new Etudiant();
- Professeur samuel = new Professeur();

static

- Un élément déclaré **static** appartient à une classe et non à ses instances.
- Un élément statique est accessible sans créer au préalable un objet de la classe
- Les objets instanciés à partir d'une classe ne possèdent pas les éléments de cette classe qui ont été déclaré static.
- Un seul élément existe pour la classe et il est partagé par toutes les instances.
- Cela ne limite en aucune façon l'accessibilité mais conditionne le résultat obtenu lors des accès.
- Les primitives, les objets et les méthodes peuvent être déclaré static.

La visibilité en Java

- L'un des avantages des classes est que les classes peuvent protéger leurs variables et méthodes membres contre l'accès par d'autres objets.

- Pourquoi est-ce important? Eh bien, considérez ceci. Vous écrivez une classe qui représente une requête sur une base de données contenant toutes sortes d'informations secrètes, par exemple des dossiers d'employés ou des déclarations de revenus pour votre entreprise en démarrage.

- Certaines informations et requêtes contenues dans la classe, celles prises en charge par les méthodes et variables accessibles au public dans votre objet de requête, sont acceptables pour la consommation de tout autre objet du système. Les autres requêtes contenues dans la classe sont là simplement pour l'usage personnel de la classe. Ils prennent en charge le fonctionnement de la classe mais ne doivent pas être utilisés par des objets d'un autre type. Vous avez des informations secrètes à protéger. Vous aimeriez pouvoir protéger ces variables et méthodes personnelles au niveau du langage et interdire l'accès aux objets d'un autre type.

- En Java, vous pouvez utiliser des spécificateurs d'accès pour protéger à la fois les variables d'une classe et ses méthodes lorsque vous les déclarez. Le langage Java prend en charge quatre niveaux d'accès distincts pour les variables membres et les méthodes : privé, protégé, public et, s'il n'est pas spécifié, package.

Mot clé	Classe	Classe enfant	Package	Partout
private	✓			
protected	✓	✓*	✓	
public	✓	✓	✓	✓
package	✓		✓	

class	attribut	method
• public • final • abstract • default	• public • final • private • protected • default	• public • final • private • protected • default • abstract • static

Quelques modificateurs

- Le mot-clé ***public*** est un modificateur d'accès utilisé pour les classes, les attributs, les méthodes et les constructeurs, les rendant accessibles à toute autre classe.
- Le mot-clé **private** est un modificateur d'accès utilisé pour les attributs, les méthodes et les constructeurs, les rendant accessibles uniquement au sein de la classe déclarée.
- Le mot-clé ***protected*** est un modificateur d'accès utilisé pour les attributs, les méthodes et les constructeurs, les rendant accessibles dans le même package et les mêmes sous-classes.

Héritage

- L'héritage est un principe clé de la programmation orientée objet. Cela implique le transfert de la structure existante d'une classe, y compris son constructeur, ses variables et ses méthodes, vers une classe différente.
- La nouvelle classe est appelée classe enfant (ou sous-classe), tandis que celle dont elle hérite, est appelée classe parent (ou superclasse).
- On dit que la classe enfant étend la classe parent dans le sens où non seulement elle hérite des structures définies par le parent, mais elle crée également de nouvelles structures
- Pour hériter d'une classe, utilisez le mot-clé ***extends***.

Syntaxe de l’héritage

Syntaxe de déclaration de l’héritage d’une classe

```
[visibilité] [final] class NomSousClasse extends NomSuperClasse {
   //...
}
```

C’est le mot-clé *extends* qui précède le nom de la superclasse.

Exemple de classe héritière

```
class ClasseEnfant extends ClasseParent {

    //...
}
```

super

- Le mot-clé super fait référence aux objets de superclasse (parents).
- Il est utilisé pour appeler des méthodes de superclasse et pour accéder au constructeur de superclasse.
- L'utilisation la plus courante du mot-clé super consiste à éliminer la confusion entre les superclasses et les sous-classes qui ont des méthodes portant le même nom.

```java
class Animal { // Superclass (parent)
  protected String name;

  public Animal(String nom){
    this.name = nom;
  }

  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Dog extends Animal { // Subclass (child)

  public Dog(String name){
    super(name);
  }

  public void animalSound() {
    super.animalSound(); // Call the superclass method
    System.out.println("The dog " + this.name + " says: bow wow");
  }
}

public class Main {
   public static void main(String[] args) {
      Dog myDog = new Dog("Médor"); // Create a Dog object
      myDog.animalSound(); // Call the method on the Dog object
   }
}
```