M2103 – Bases de la Programmation Orientée Objets

Java – Cours 8

Interfaces & Classes Abstraites

Plan du Cours

- Classes Abstraites et Méthodes
- Interfaces
- Héritage Multiple
- Mécanisme de rappel

Instanciation d'Objets

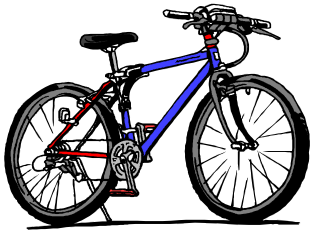
- Type de la variable doit être le même que celui de l'objet référencé, ou celui d'une classe de l'ascendance (classe parente…)
- Variable peut se référer à n'importe quelle instance qui est de même type classe ou d'une classe dérivée de ce type
- Exemples:

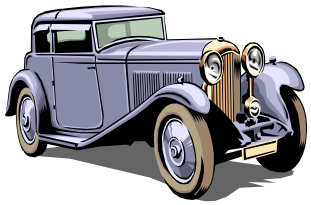
```
Voiture maVoit = new Voiture(…);
Velo tonVelo = new Velo(…);

Vehicule vehicTransport = maVoit;
Vehicule vehicPasCher = tonVelo;
```

Instanciation d'Objets (2)

- Certaines classes ne devraient pas être instanciés.
- Cf. exemple précédent :
  - Peut-on instancier un 'Vehicule' générique ?
  - Est-ce qu'on en a besoin ?
  - A quoi ressemblerait un 'Vehicule' ?
- `Vehicule` est un concept abstrait. Les classes dérivées plus spécifiques comme `Voiture` et `Velo` caractérisent elles des entités concrètes.





Classe Vehicule

```
public class Vehicule {
    int roues;
    int engrenages;
    Vehicule (int roues, int engrenages) {
       this.roues = roues;
       this.engrenages = engrenages;
    }
    public void demarrer() {…}
    public void stopper() {…}
    public void tourner() {…}

    public String toString() {.. }

}
```

Implémentation des méthodes ?

Méthodes Abstraites

- Méthodes pour lesquelles on ne désire pas produire du code peuvent ne pas être implémentées
- Utilisation du mot-clé **abstract**
- Syntaxe:
  public abstract void demarrer();
  public abstract void tourner();

Classes Abstraites

- Si au moins une méthode de la classe est déclarée abstraite, alors la classe devient une ***Classe Abstraite***
- Utilisation du mot-clé **abstract**
- On ne peut instancier d'objets à partir des classes abstraites
  - On peut le faire à partir d'une sous-classe
- On peut toutefois toujours mettre en place attributs et méthodes que l'on sait implémenter (par ex. constructeur)

Classe Vehicule – Version Classe Abstraite

```
public abstract class Vehicule {
    int rouges;
    int engrenages;
    Vehicule (int roues, int engrenages) {
       this.roues = roues;
       this.engrenages = engrenages;
    }
    public abstract void demarrer() ;
    public abstract void stopper() ;
    public abstract void tourner() ;

    public String toString() {.. }

}
```

Classes Filles Concrètes

- Une sous-classe d'une classe abstraite doit normalement hériter des méthodes abstraites et donc être une classe abstraite.
- Elle doit donc proposer une ***implémentation concrète*** pour toutes les méthodes abstraites héritées
  - Elle devient une ***classe concrète*** à partir de laquelle on peut instancier des objets
  - Les signatures de méthodes doivent correspondre, exception faite du mot-clé 'abstract'

Classe Velo – Sous-classe Concrète de Vehicule

```
public class Velo extends Vehicule {
    String guidon;
    String pedales;
    Velo (int roues, int engrenages, String guidon, String pedales) {
          super(roues, engrenages);
          this.guidon = guidon;
          this.pedales = pedales;

    }
    public double pedaler () {.. }
    public String toString() {   return (super.toString() + " " +
      guidon + " " + pedales);  }

    public void demarrer () { … }
    public void stopper () { … }
    public void tourner () { … }

}
```

Le constructeur dans Vehicule doit être implémenté

Méthodes devront être implémentées dans cette classe

Héritage Multiple

- Comment peut-on matérialiser le fait que des classes concrètes puissent être mises en oeuvre dans différentes classifications ?
- Exemple : un `Velo` est un `Vehicule` et est également un `MaterielDeSport`
- Problème : Java ne permet à une classe que d'avoir une classe parente directe

Héritage Multiple en Java : Interfaces

- Consiste en une liste de signatures de méthodes, sans implémentation
  - comme des méthodes abstraites, mais sans le mot-clé abstract
- Permettent l'héritage multiple, i.e. une classe résultante hérite de caractéristiques provenant de différentes sources
- A la base de la technique de ***programmation par contrat***, i.e. une classe qui se conforme au format spécifié par une interface doit *implémenter* cette interface
  - Implémentation concrète pour chaque méthode de l'interface

Interfaces - Exemple

```
public interface MaterielDeSport {
    int prix;
    String nom;
    ArrayList<String> sports;
    public void sportsCorrespondants ();
}
class Velo extends Vehicule implements MaterielDeSport  {
    public void sportsCorrespondants ()
    {
        .....
    }
}
```

Compléments sur les Interfaces

- Parfois une interface sert à indiquer que les objets instanciés à partir d'une classe implémentant l'interface ont un rôle ou une fonctionnalité spécifique.
  - implements Comparable
    - indique que la classe présente une méthode permettant de comparer ses instances (ordre total)
- Variable référence peut être déclarée de type interface
  - Peut indiquer tout objet dont la classe implémente l'interface

Application - Mécanisme de Rappel

- Est basé sur 2 classes avec 2 rôles spécifiques
  - **Classe Emettrice** – annonce les événements qui se sont déroulés
  - **Classe Souscriptrice** – désire être informée à propos des événements qui se sont déroulés
- Un événement est quelque chose d'observable, qui se déroule à un certain moment, et pour lequel il peut y avoir de l'information
  - La porte a été fermée/ouverte
  - Le bouton gauche de la souris a été cliqué
  - Un `DocLibrairie` emprunté a été rendu
- Exemple de la file d'attente de la banque

  Vous êtes le prochain client à être servi. Vous patientez jusqu'à ce qu'un guichetier dise 'suivant'. Vous répondez à cette ***notification*** parce que vous êtes le premier de la file. Vous êtes le ***souscripteur***. Le guichetier est l'***émetteur***.

Processus pour la Mise en Oeuvre du Mécanisme de Rappel

1. Souscripteur doit informer l'émetteur qu'il désire recevoir une notification à propos d'un événement futur
   - ***Souscription***
2. Emetteur surveille la situation
3. Quand l'émetteur a un fait intéressant à communiquer, il notifie les souscripteurs
   - ***Emission/Publication***
4. Souscripteur peut informer l'émetteur de ne plus envoyer de notification d'événements futurs
   - ***Arrêt de la souscription***

Implémentation des Mécanismes de Rappel

1. Implémenter une interface souscriptrice
2. Implémenter une classe émettrice
3. Implémenter des classes souscriptrices concrètes
4. Ecriture du code créant l'émetteur et les souscripteurs, puis qui inscrit les souscripteurs au niveau de l'émetteur

Mise en place d'une Interface Souscriptrice

- Décision sur les notifications transmises par l'émetteur
- Ecriture d'une interface afin de spécifier les méthodes de mise à jour pour chaque notification

```
public interface Souscripteur {
  //Méthode(s) de mise à jour suite à une notification, utilisée(s) par l'émetteur
    public void maj1();
    public void maj2();
   (…)
  //Mise en relation avec l'émettteur
    public void setEmetteur (Emetteur e); }
```

- Exemples de telles méthodes :

public void stockDispoTropFaible(int stockDispo) – réaction à la notification indiquant que le stock disponible n'est pas suffisant

public void docDisponible(DocLibrairie d) – réaction à la notification indiquant que le `DocLibrairie` *d* est désormais disponible

Mise en place d'une Classe Emettrice

- Nécessite des méthodes permettant la souscription et l'arrêt de celle-ci
- Nécessite une structure pour (éventuellement) se rappeler de tous les souscripteurs
  - Les éléments doivent être de même type (type interface)
- Nécessite une méthode pour générer les notifications d'événements aux souscripteurs

```
public class Emetteur {
  // Attrbuts privés avec éventuellement structure stockant tous les souscripteurs
    (…)
  // Méthodes
    public Emetteur () {…};
    // Souscription et arrêt de la souscription
    public void souscription(Souscripteur s);
    public void arretSouscription(Souscripteur s);
    // Notification pour les souscripteurs du changement
    public void notificationSouscripteurs();
    // Obtention d'information de mise à jour de la part de l'émettteur
    public Object getInfoMaJ(Souscripteur s); }
```

Classes Souscriptrices Concrètes

- Classes recevant les notifications de la part de l'émetteur
- Doivent être déclarées comme implémentant l'interface souscriptrice
  - Pour que l'émetteur appelle les méthodes de notification adéquates
- Implémentent les méthodes de l'interface
  - I.e. fournissent les réponses spécifiques aux messages

Inscription des Souscripteurs

- Dans une classe de coordination (test), création d'une instance de la classe émettrice et instance(s) souscriptrice(s)
- Souscripteur(s) doivent être 'inscrits' au niveau de l'émetteur
- Exemple:

```
// Emetteur :
ClasseSurveilleNiveauStock csns = new ClasseSurveilleNiveauStock();

// Souscripteur :
Comptabilite cpt = new Comptabilite();

csns.ajoutStockSurveillance(cpt);          // inscription du souscripteur
```

Lorsque le niveau du stock se situe en-dessous d'un niveau prédéterminé, l'objet ***cpt*** *aura sa méthode* ***stockDispoTropFaible*** appelée par ***csns***