Interface Homme-Machine (IHM)

INFO0202

janvier 2019

Contact : Jean-Charles.Boisson@univ-reims.fr

ORGANISATION

Organisation du module

- Cours Magistraux (CMs) : 6h ⇔ 3 séances

  ➔ semaines 5, 7 et 11.

Organisation du module

- Cours Magistraux (CMs) : 6h ⇔ 3 séances

  ➔ semaines 5, 7 et 11.

- Travaux Dirigés (TDs) : 4h ⇔ 2 séances

  ➔ semaines 7 et 11.

Organisation du module

- Cours Magistraux (CMs) : 6h ⇔ 3 séances
  ➔ semaines 5, 7 et 11.

- Travaux Dirigés (TDs) : 4h ⇔ 2 séances
  ➔ semaines 7 et 11.

- Travaux Pratiques (TPs) : 16h ⇔ 8 séances
  ➔ semaines 9-14, 17-18

Organisation du module

- Evaluation :
  - 1ère session ⇔ 100% CC* :
    - ITP : 50%
    - Projet : 50%

Organisation du module

- Evaluation :
  - 1ère session ⇔ 100% CC* :
    - ITP : 50%
    - Projet : 50%

OBJECTIFS

Objectifs

- Renfort en Programmation Orienté Objet (POO)
- Maîtrise du patron de conception Observateur
- Connaissance du principe
  Modèle Vue Contrôleur (MVC)
- Mise en œuvre en JAVA SWING et FX

AVANT DE COMMENCER …

Supports de cours

- Ressources disponibles sur Moodle

  ➔ INFO0202 Interface Homme-Machine

- Inscription par une clé :
  - INFO202_1819_NG
    - Avec N valant 3, 4 ou 5
    - Avec G valant a, b ou c
    - Par exemple, pour un étudiant du groupe S2A3A

      ➔ INFO202_1819_3a

l'API JAVA ?

- API ⇔ Application Programming Interface
- Appelée communément JAVADOC (du nom de la commande qui sert à sa production)
- Comprendre la différence entre JRE et JDK
- Quelle version utiliser ? 8, (9,10,) ou 11

Aide en JAVA

- 2 ressources à utiliser :
  - La JAVADOC brute (disponible localement après téléchargement) :
    - JAVADOC :
      - 1.8
      - 1.9
      - 1.10
      - 1.11
  - Le centre d'aide Oracle (incluant les APIs) :

documentation Oracle en ligne

Utilisation de l'API JAVA

- Utilisez une classe ⇔ 2 méthodes

- Méthode 1:
  - Inclusion du paquetage (package) qui contient la classe.
  - Utiliser le nom de la classe.

- Méthode 2 : utilisation du nom complet de la classe ⇔ paquetage + classe

Utilisation de l'API JAVA

- Exemple de création d'un objet Vector localisé dans java.util.
- Exemple méthode 1 :

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

Utilisation de l'API JAVA

- Exemple de création d'un objet Vector localisé dans java.util.
- Exemple méthode 2 :

```
public class Test{
  public static void main(String[] args){
    java.util.Vector vecteur = new java.util.Vector()
    ... } }
```

UN PEU DE POO ?

La POO

- Principe de programmation se basant sur l'intéraction d'entités ayant une structure et des capacités clairement définies.
- Ces entités ⇔ les objets.
- Tout est objet en JAVA.
- Un objet ⇔ un concept ➔ Conception OO

Objet ???

- Un objet est une entité possédant :
  - Une structure interne (visible ou non)
    - Attributs, champs, caractéristiques
  - Des capacités (visible ou non)
    - Fonctions, procédures

Attributs d'un objet

- Les attributs sont des variables/constantes typées et nommées.

- Accessibilité aux attributs :
  - Privée ⇔ seule l'objet a accès à ses attributs
  - (protégée) ➔ cf héritage
  - Publique ⇔ tous ont accès au attribut de l'objet.

Attributs d'un objet

- Exemples :

  public int number;

  ➔Variable entière publique non initialisée.

  private boolean activated;

  ➔ Variable booléen privée non initialisée.

La construction d'objet

- Appel à un constructeur

  ➔ création d'une instance de cet objet.

- Durée de vie d'une instance :

  ➔ tant qu'une variable du type de l'objet contient une référence vers cette instance.

- « Destruction » d'une instance :

  ⇔ plus aucune référence vers elle

➔ nettoyage la mémoire par le garbage collector

Le constructeur

- Constructeur ⇔ méthode particulière portant le même nom que la classe.

- But du constructeur ➔ initialiser les attributs *(si leur valeur par défaut n'est pas acceptable)*

- Comme toute méthode, un constructeur peut être public, protected ou private.

Le constructeur

- Tout objet possède un constructeur sans paramètre par défaut ⇔ constructeur vide

- Un constructeur non vide présent

  ➔ le constructeur vide disparaît

- Pour dupliquer logiquement un un objet

  ➔ constructeur par copie

Le constructeur

- Situation : un objet a (entre autres) d'autres objets comme attributs.

- Constructeur de copie superficielle :

  ➔ Dans le nouvel objet les attributs « objets » sont une copie de référence (et non des « objets » créés)

- Constructeur de copie profonde :

  ➔ Chaque attribut « objet » est réellement instancié (constructeur de copie de l'objet)

Contenu d'une instance

- Une instance contient :
  - Des valeurs spécifiques pour les attributs qui sont liées à elle ⇔ initialisé par le constructeur.
  - Ces attributs sont aussi appelés variables d'instance.
  - Les méthodes qui lui sont liées ⇔ méthodes d'instance.
- Variables et méthodes non d'instance ???

Les variables de classe

- Si une variable dépend forcément de la construction d'un objet ➔ variable d'instance.

- Si une variable est indépendant d'une instanciation ⇔ sa vie est liée directement à sa définition dans la classe elle-même.

  ➔ variable de classe

Méthodes de classe

- Si une méthode se base sur des variables d'instance ➔ méthode d'instance
- Si une méthode ne dépend pas de variables d'instance mais est liée à la vie d'une instance

  ➔ méthode d'instance.

- Si une méthode est tout à fait indépendante d'une instanciation

  ➔ méthode de classe

Durée de vie

- Des attributs :
  - Variable d'instance : tant que l'instance existe l'attribut existe.
  - Variable de classe : tant que la définition de l'objet existe en mémoire.

- Des méthodes :
  - Méthodes d'instance : utilisables tant que l'instance existe.
  - Méthodes de classe : utilisables tant que la définition de l'objet existe en mémoire.

Le mot clé « static »

- Définit une variable/méthode qui ne dépend pas d'une instanciation.

- Définit les :
  - Variables de classe
  - Méthodes de classe
  - Les constantes de classe avec le mot clé « final »
    - Exemple :

```
public static final double PI = 3.14
```

Utilisation pratique

- Variable/méthode d'instance :

MaClasse nomClasse = new MaClasse(param …)

nomClasse.methode(…)

nomClasse.attribut //si public

- Variable/méthode de classe :

MaClasse.methodeDeClasse(…)

MaClasse.attributDeClasse //si public

Exemple applicatif

Définissez une manière de pouvoir compter le nombre d'instances d'un objet qui ont été créées.

Exemple d'objet

- Création d'un objet représentant un carré.
- Un carré est défini uniquement la longueur d'un de ses côtés.
- On ne veut pas que quelqu'un puisse changer le longueur du côté d'un carré ➔ variable privée.

Classe Square minimale

```
public class Square {
        private double longueurCote;

        public Square(double l) {
                longueurCote = l;
        }
}
```

L'objet « this »

- Dans une classe, « this » fait référence à l'instance d'objet sur laquelle une action est lancée.

- « this » permet de nommer sans ambiguïté une variable ou une méthode la classe.

- « this » ne peut être utilisé dans une méthode de classe ⇔ méthode déclarée avec le mot clef static

Classe Square avec this

```
public class Square {
        private double longueurCote;

        public Square(double longueurCote) {
                this.longueurCote = longueurCote;
        }
}
```

Contrôle d'accès

- private ⇔ accès interdit depuis l'extérieur

- Si nécessaire, comment contrôler depuis l'extérieur la:
  - lecture d'une variable privée ?
  - modification d'une variable privée ?

  ➔La rendre publique ?

Contrôle d'accès

- private ⇔ accès interdit depuis l'extérieur

- Si nécessaire, comment contrôler depuis l'extérieur la:
  - lecture d'une variable privée ?
  - modification d'une variable privée ?

  ➔La rendre publique ? **mauvaise idée** car aucun contrôle sur qui et quand accède à la variable.

Contrôle d'accès

- private ⇔ accès interdit depuis l'extérieur

- Si nécessaire, comment contrôler depuis l'extérieur la:
  - lecture d'une variable privée ?
  - modification d'une variable privée ?

  ➔La rendre publique ? mauvaise idée car aucun contrôle sur qui et quand accède à la variable.

  ➔Ajouter des méthodes d'accès ?

Contrôle d'accès

- private ⇔ accès interdit depuis l'extérieur

- Si nécessaire, comment contrôler depuis l'extérieur la:
  - lecture d'une variable privée ?
  - modification d'une variable privée ?

  ➔La rendre publique ? mauvaise idée car aucun contrôle sur qui et quand accède à la variable.

  ➔Ajouter des méthodes d'accès ? **OUI**

  les accesseurs (ou getter/setter)

Contrôle d'accès

- Pouvoir lire le contenu d'une variable

  ➔ « getter »

```
…
private type var;
…
public type getVar() {
    return this.var;
}
…
```

Contrôle d'accès

- Pouvoir modifier le contenu d'une variable

  ➔ « setter »

```
…
private type var;
…
public void setVar(type newValue) {
    this.var = newValue;
}
…
```

Classe Square avec accesseurs

```
public class Square {
        private double longueurCote;
        public Square(double longueurCote) {
                this.longueurCote = longueurCote;  }
        public double getLongueurCote() {
                return this.longueurCote; }
        public void setLongueurCote(double l) {
                this.longueurCote = l; }
}
```

L'héritage

- Permet d'obtenir une classe plus spécialisée à partir d'une autre.
- Permet d'accéder aux méthodes/attributs de la classe mère sauf si définis « private ».
- Est unique (en JAVA*).

L'héritage

- Utilisation du mot clef extends

- Impose l'appel au constructeur de la classe mère :
  - 1ère instruction du constructeur de la classe fille.

  - Si constructeur mère vide ⇔ appel implicite
  - Sinon appel explicite obligatoire

L'héritage

- Relation classe mère/ classe fille:
  - this ⇔ accès non ambigü aux caractéristiques de l'instance en cours
  - super ⇔ accès non ambigü aux caractéristiques de classe mère de l'instance en cours

    ➔ Par exemple : redéfinition du toString de la classe fille en utilisant celui de la classe mère

L'objet « Object »

- Par défaut toute classe hérite de java.lang.Object
- Toute classe possède les mêmes capacités que la classe Object.
- Comparaison JAVA 8 et 11.

L'objet « Object »

- Les méthodes basiques :
  - public String toString() : fonction appelée automatiquement dès qu'un objet est affiché sur un flux.
  - public final Class<?> getClass() : fonction qui permet d'obtenir l'objet « Class » associée à une instance. Cela permet notamment d'accéder aux informations d'une classe à la volée.

L'objet « Object »

- Les méthodes basiques :
  - public boolean equals(Object o) : méthode appelée dès pour prouver que deux objets sont logiquement équivalents.
  - public int hashCode() : méthode qui décrit sous forme d'un entier l'instance en cours. Elle va de pair avec la méthode equals pour l'utilisation des *tables de hashage*.

L'objet « Object »

- Les méthodes d'« Object » sont génériques et non dédiées.

- Par héritage, on peut redéfinir une méthode ⇔ donner un autre contenu à la même méthode.
  - Redéfinition de toString pour permettre un affichage plus aisé des objets.
  - Redéfinition de equals (et hashCode) pour permettre une comparaison efficace des objets.

Le polymorphisme

- Lorsque plusieurs définitions d'une méthode existent dans l'arbre d'héritage d'un objet, comment choisir la bonne ?

  ➔ celle associée au type de l'objet lors de sa création ⇔ polymorphisme

- On peut forcer le choix de la méthode en [up/down]castant explicitement vers le type désiré.

Classe Square

- Le carré est-il un type de figure en particuliers ?
- Si oui peut-on généraliser certains traitement ?
- Peut-on aller plus loin ?

Les classes génériques

- Toute classe est un sous-type de la classe

- Toute classe définit un comportement pour un type précis.

- Comment définir un comportement indépendamment d'un type ?

Les classes génériques

- Exemple : création d'une classe liste de voiture
  - Classe Voiture
  - Classe ListeDeVoitures
  - Classe TestListeDeVoitures

Les classes génériques

```
public class Voiture {
    private String nom;
    public Voiture(String nom)  {
        this.nom = new String(nom);
   }
    public String toString()  {
        return "Je suis la voiture "+ this.nom;
    }
}
```

Les classes génériques

```
public class ListeDeVoitures {
   private Voiture courant;
   private ListeDeVoitures prochain;

   public ListeDeVoitures(Voiture voiture)    {
            this.courant = new Voiture(voiture);
            this.prochain = null;
   }

   public void ajouteVoiture(Voiture voiture)     {
      ListeDeVoitures encours = this;
      while(encours.prochain != null)  {
            encours = encours.prochain;
      }

     encours.prochain = new
                        ListeDeVoitures(voiture);
   }
```

```
   public String toString()     {
      String message = "Je suis une liste de
voitures\n";
      ListeDeVoitures encours = this;
      do  {
        message+="\t"
              +encours.courant.toString()+"\n";
        encours=encours.prochain;  }
      while(encours != null);
      return message;
   }
}
```

Les classes génériques

```
public class TestListeDeVoitures {
    public static void main(String[] args)  {
        ListeDeVoitures liste =
                new ListeDeVoitures(new Voiture("toto"));
        System.out.println(liste);

        liste.ajouteVoiture(new Voiture("titi"));
        System.out.println(liste);
    }
}
```

Les classes génériques

- Résultat de l'exécution :

Je suis une liste de voitures

Je suis la voiture toto

Je suis une liste de voitures

Je suis la voiture toto

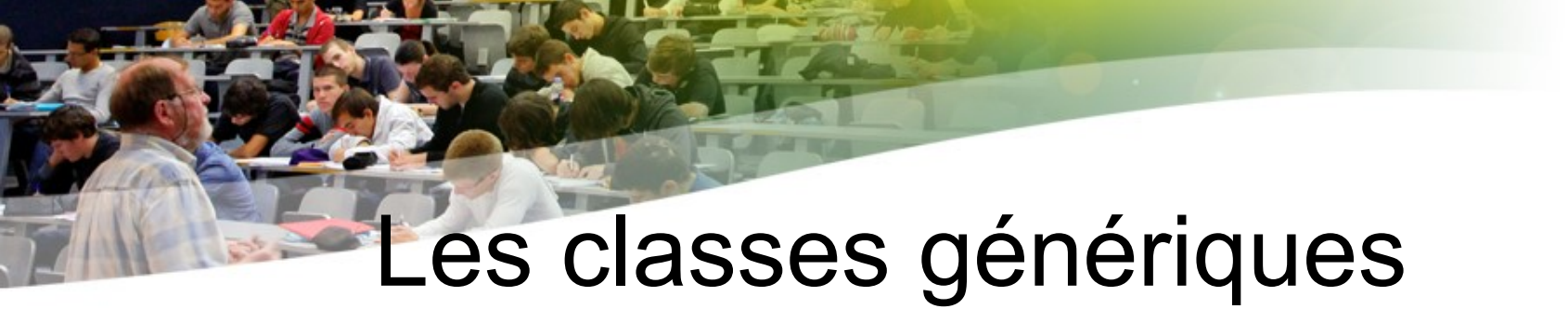
Je suis la voiture titi

Les classes génériques

- Exemple : création d'une classe liste de motos ?
  - Classe Moto ?
  - Classe ListeDeMotos ?
  - Classe TestListeDeMotos ?

Les classes génériques

- Création d'une classe générique ListeDe ⇔ non dédié à un type spécifique.
- La classe sera typée à l'instanciation.
- Classe générique ⇔ class ListeDe<T> avec T un type quelconque.



Les classes génériques

```
public class ListeDe<T> {
   private T courant;
   private ListeDe<T> prochain;

   public ListeDe(T unType)  {
     // Attention copie superficielle
    this.courant = unType;
    this.prochain = null;
   }

   public void ajoute(T unType)
   {
      ListeDe<T> encours = this;
      while(encours.prochain != null) {
           encours = encours.prochain;
      }
     encours.prochain = new ListeDe<T>(unType);
   }
```

```
   public String toString()   {
      String message = "Je suis une liste de " +
                courant.getClass().getName()+ " \n ";

      ListeDe<T> encours = this;

      do {
        message+="\t"+
                  encours.courant.toString()+"\n";
        encours=encours.prochain;
       }
      while(encours != null);

      return message;
   }
}
```

Les classes génériques

```
public class TestListeDe {
    public static void main(String[] args)   {
        ListeDe<Voiture> liste = new
                    ListeDe<Voiture>(new Voiture("toto"));
        System.out.println(liste);

        liste.ajoute(new Voiture("titi"));
        System.out.println(liste);
    }
}
```

Les classes génériques

- Résultat de l'exécution :

Je suis une liste de

Je suis la voiture toto

Je suis une liste de

Je suis la voiture toto

Je suis la voiture titi

Les classes génériques

```
public class TestListeDe {
    public static void main(String[] args)   {
        ListeDe<Moto> liste = new
                    ListeDe<Moto>(new Moto("toto"));
        System.out.println(liste);

        liste.ajoute(new Moto("titi"));
        System.out.println(liste);
    }
}
```

Les classes génériques

- Résultat de l'exécution :

Je suis une liste de

Je suis la moto toto

Je suis une liste de

Je suis la moto toto

Je suis la moto titi

Les classes génériques

- Toutes les structures liste, pile, tableau existent en JAVA
- Ces structures sont génériques ⇔ gère des Objects

- Mais peuvent être typées dynamiquement

- Exemple :
  - java.util.LinkedList list = new java.util.LinkedList();
  - java.util.LinkedList<Voiture> list = new
    java.util.LinkedList<Voiture>();

L'interfaçage

- Permet d'ajouter une/des nouvelle(s) spécialisation(s) à une classe.
- Doit être complet (tout ou rien).
- Peut être multiple.

L'interfaçage

- Interface ⇔ simple énumération de signatures de méthodes.
- Utilisation du mot clef implements
- Interfaçage multiple
  - ➔ implements classe1, classe2, …

L'interfaçage : exemple

- Avoir la capacité de « crier » ⇔ pas de comportement par défaut.

- Interface « SaitCrier » :

```
public interface SaitCrier
        {
                public String crie();
        }
```

L'interfaçage : exemple

- Exemple d'implémentation :

```
public class Chien implements SaitCrier
        {
                public String crie()
                        {
                                return « Wouaff »
                        }
        }
```

Les interfaces de JAVA

- L'interface java.lang.Comparable <T>

- Définit la relation d'ordre entre objets :

  public int compareTo(T)

- Permet le tri automatique dans la plupart des structures ⇔ utilisation de la méthode statique sort de la classe java.util.Collections

Les interfaces de JAVA

- L'interface java.lang.Iterable<T>

- Permet l'accès à un itérateur sur l'objet :

  public Iterator<T> iterator()

- Tous les structures types tableau, liste, file, … peuvent fournir un itérateur.

Les interfaces de JAVA

- L'interface java.lang.Cloneable
- Interface particulière ⇔ sans signatures
- Redéfinition de « clone » ➔ autorisation de son utilisation.

L'abstraction

- Permet de garder une/des méthode(s) non encore définie(s) pour un objet

  ➔ l'objet n'est plus instanciable.

- Obtenir une classe instanciable depuis une classe abstraite ≈ héritage + interfaçage

- Une classe « fille » devra donner corps à toutes les méthodes « abstraites » pour être instanciable.

L'abstraction

- Utilisation du mot clef abstract
- Seules les méthodes peuvent être abstraites.
- Méthode(s) abstraite(s) ➔ classe abstraite
- Une classe abstraite ⇏ méthode(s) abstraite(s)
  - *On peut juste vouloir empêcher l'instanciation*