Programmation Orientée Objet

Master 1 CS

DR. Sofiane AOUAG
Université De Batna II
Faculté des Mathématique et de l'Informatique
Département d'Informatique

2017/2018

Plan du cours

Programmation
- Classes et Héritage -

1. Quelques spécificateurs
   - `static`
   - `final`
2. Types énumérés
   - Présentation
   - Exemples
3. Héritage
   - Réutilisation
   - En Java
   - Polymorphisme
   - La classe `Object`

Attributs et Méthodes de classe

Propriétés

Les membres `static` :
- ne sont pas attachés à une instance
- ont une existence avant instanciation
- ne peuvent accéder aux membres non static
- sont partagés par toutes les instances

Syntaxe

```
class NomClasse {

  accès static type attribut;

  accès static type methode(...);

}
```


Initialisateur Static

Attributs de classe
- initialisés dans la classe (au chargement)
- accès par la classe ou une instance

Utilisation : information commune à des instances (constantes), compteur de références, ...

Méthodes de classe
- ne peuvent accéder qu'aux membres `static`
- accès par la classe ou une instance

Utilisation : boite à outils (`Math`), manipulation d'instances de la classe, ...

Initialisateur static

Permet l'initialisation complexe de membres static

Syntaxe

```
class NomClasse {
  static {
    // instructions
  }
}
```


Exemples

ClasseCompteur.java

```
public class ClasseCompteur {

  private static int compteur=0;

  public ClasseCompteur() {
    compteur++;
  }

  public static int getNbObjetConstruits() {
    return compteur;
  }

  public static void main(String[] args) {
    System.out.println("Nb␣objets␣construits␣:␣"+ClasseCompteur.getNbObjetConstruits());
    ClasseCompteur[] cc=new ClasseCompteur[10];
    for(int i=0;i<10;i++)
      cc[i]=new ClasseCompteur();
    System.out.println("Nb␣objets␣construits␣:␣"+ClasseCompteur.getNbObjetConstruits());
  }
}
```


Hasard.java

```
import java.util.Random;

public class Hasard {

  private static final int NB_VAL = 20;
  private static int[] sequenceAleatoire = new int[NB_VAL];

  private Random aleatoire;

  static {
    Random alea=new Random();
    for(int i=0; i<NB_VAL; i++)
      sequenceAleatoire[i]=alea.nextInt()%100;
  }

  public Hasard() {
    aleatoire = new Random(System.currentTimeMillis());
  }

  public int getNombre() {
    return sequenceAleatoire[aleatoire.nextInt()%NB_VAL];
  }
}
```


Les significations de final

`final` = non modifiable

En fonction du contexte

- `final type attribut`
  type simple : valeur constante
  type objet : référence constante
- `type nomMethode(final type param, ...)`
  le paramètre est constant (cf. ci-dessus)
- `final type nomMethode(...)`
  la méthode ne peut être redéfinie lors d'un héritage
  private → final
- `final class NomClasse ...`
  la classe ne peut être dérivée

Constantes

Initialisation

Toute constante doit être initialisée :

- soit à la déclaration
- soit dans le constructeur (ou initialisateur `static`)

l'initialisation peut être :

- à la compilation : la valeur est disponible et elle est placée en dur
- à l'exécution : la constante est initialisée à l'exécution

Deux "types" de constantes

- constante d'instance
  initialisée à l'instanciation, une constante par instance
- constante de classe
  déclarée `static` (init au chargement), une constante unique

Exemples

Constantes.java

```
public class Constantes {
  public final int valeurentiere=42;
  public final int valeurAleatoire=(int)(Math.random()*100);
  public static final int VALEUR_PLANCHER=200;
  public final int[] tableau={1,2,3};
  public final int valeurentiere2;

  public Constantes(int valent) {
    valeurentiere2=valent;
    //   valeurentiere=valent;
    //   VALEUR_PLANCHER=13;
  }
  public static void main(String[] args) {
    Constantes cnsts=new Constantes(4);
    //   cnsts.valeurentiere2=3;
    //   cnsts.tableau=new int[5];
    cnsts.tableau[0]=5;
  }
}
```


Définition et syntaxe

Définition

Un type énuméré permet de définir explicitement un ensemble ordonné de valeurs.

Syntaxe

```
enum NomTypeEnum {VAL1,VAL2, ..., VALn};
```

Définition interne à une classe (inline) ou externe (public ou non)

Syntaxe et fonctionnalités

- un enum est une classe (dérive de `java.lang.Enum`)
- les valeurs sont `public static final`
- comparaison via == ou `equals()`
- implémente l'interface `java.lang.Comparable`
- dispose des méthodes `toString()` et `valueOf()`[a]
- `ordinal()` ⇒ position, `values()` ⇒ itération

[a] à la redéfinition de l'une, l'autre doit toujours avoir la fonctionnalité inverse

Genre.java

```
public enum Genre {MASCULIN, FEMININ};
```

Informaticien.java

```
public class Informaticien {
  // un enum dans une classe est static
  public enum Type {NOUVEAU, DEBUTANT, CONFIRME, GOUROU};
  private Type type;
  private Genre genre;

  public Informaticien(Type _type, Genre _genre) {
    type=_type;
    genre=_genre;
  }
  public Informaticien(String strType, String strGenre) {
    type=Type.valueOf(strType);
    genre=Genre.valueOf(strGenre);
  }
  public String toString() {
    return "Informaticien␣"+genre.toString().toLowerCase()+"␣"+type.toString().toLowerCase();
  }
}
```


EnumTest.java

```
public class EnumTest {
  public static void main(String[] args) {
    Informaticien inf1 = new Informaticien(Informaticien.Type.NOUVEAU,
                                           Genre.MASCULIN);
    Informaticien inf2 = new Informaticien("GOUROU","FEMININ");
    System.out.println(inf1);
    System.out.println(inf2);
  }
}
```