Classes internes, Classes locales, Classes anonymes

Victor Marsault
Aldric Degorre

CPOO 2015

Enum (1)

- Quand les utiliser:
  - disjonctions de cas
  - "type" au sens courant (eg. type de messages d'erreur, type de pièces dans un jeu)
- Il n'existe qu'un nombre fini d'instances différentes d'un `enum` (ici 7);
- Toujours implicitement statique

```
public enum Jour {
    LUNDI, MARDI, MERCREDI, JEUDI,
    VENDREDI, SAMEDI, DIMANCHE ;
}
```

Enum (2)

- Utilisation de `==` et de `switch`;
- Garde-fou

```
public enum Jour {
    LUNDI, MARDI, MERCREDI, JEUDI,
    VENDREDI, SAMEDI, DIMANCHE ;

    public Jour next() {
        switch (this) {
          case LUNDI: return MARDI;
          case MARDI: return MERCREDI;
            ...
          case DIMANCHE: return LUNDI;
        }
    }
}
```

Enum (3)


```
public enum Jour {
    LUNDI(0,"Mon."), MARDI(1,"Tue."), MERCREDI(2,"Wed."),
    JEUDI(3,"Thu."), VENDREDI(4,"Mon."),
    SAMEDI(5,"Sat."), DIMANCHE(6,"Sun.") ;

    final public int id;
    final public String nom;
    Jour (int id, String nom) { // Constructeur
        this.nom=nom; this.id=id;
    }
    public static Jour ofInt (int i) {
        for(Jour j : Jour.values())
            if (j.id == i) return j;
        return null; // return ofInt(i%7);
    }
}
```

Enum (4)


```
public class Test {
  public static void main (String args) {
    Jour j1 = Jour.Mardi;
    Jour j2 = Jour.ofInt(3);  // vaut Jour.Mercredi
    Jour j3 = j1.next().next();  // vaut Jour.Vendredi

    for (Jour j : Jour.values())
      if (j.id < j3.id)
        System.out.println(j.nom);
    // affiche  Mon. Tue. Wed. Thu.
  }
}
```

Classe interne statique (1)

- Membre statique d'une autre classe
  $\rightarrow$ mot clef `static`
- N'a accès qu'aux membres statiques (de la classe externe).
- Utilisation principale: hiérarchisation.

```
class Externe {
    class static Interne {
      ...
    }
  ...
}
```

Classe interne statique (2)


```
public class PileExecutable {
    public static abstract class Chainable {
        private Chainable suivant;
        public Chainable getSuivant() { ... }
        public void setSuivant(Chainable noeud) { ... }
        public abstract void execute() ;
    }
    Chainable tete;
    public void empiler(Chainable c){ ... }
    public Chainable depiler(){ ... }
    public void execute() {
        Chainable tmp = tete;
        while (tmp!= null) {
            tmp.execute();
            tmp = tmp.getSuivant();
        }
    }
}
```

Classe interne statique (3)


```
class EntierAutoIncrem extends PileExecutable.Chainable {
    int valeur;
    public void execute()  { valeur = valeur + 1; }
    EntierAutoIncrem(int valeur)  { this.valeur= valeur;}
} // Dans EntierAutoIncrem.java

class Copieur extends PileExecutable.Chainable {
    public void execute()  {
        if (suivant!= null) suivant.execute(); }
} // Copieur.java

class StrAutoPrint extends PileExecutable.Chainable {
    String valeur;
    public void execute()  { System.out.println(valeur); }
    StrAutoPrint(String valeur)  { this.valeur= valeur; }
} // StringAutoPrint.java
```

Classe interne statique (4)


```
static void main (String args) {
    PileExecutable l = new PileExecutable();
    l.empiler( new EntierAutoIncrem(1) );
    l.empiler( new StrAutoPrint("Hello") );
    l.empiler( new StrAutoPrint("World") );
    l.empiler( new Copieur() );
    l.execute();  //   -> affiche
                  //  World
                  //  World
                  //  Hello
                  //   -> l'entier vaut 2
}
```

Classe interne non-statique

- Accède aux membres non-statiques de la classe.
- Est une partie de la classe externe:
- Ne peut avoir de membre static.

```
class Externe {
    class Interne {
       ...
    }
    Interne att;
  ...
}
```

Classe locale

- Définie à l'intérieur d'un bloc de code.
- Accède aux attributs de la classe Externes.
- Accède aux variables locales déclarées `final`.
- Ne peut avoir de membre statique

```
class Externe {
    public void TypeDeRetour mamethode () {
        class Interne extends TypeDeRetour {
          ...
        }
        return new Interne()
    }
}
```

Classe locale – Premier exemple


```
public class ClasseExterne {
    public int i = 1;
    public afficheToto() { System.out.println("toto"); }

    public void mamethode() {
        final int j=10;
        int k=100;
```

Classe locale – Premier exemple


```
public class ClasseExterne {
    public int i = 1; // accessible car membre externe
    public afficheToto() { System.out.println("toto"); }//

    public void mamethode() {
        final int j=10; // accessible car final
        int k=100; // inaccessible
        class ClasseLocale {
```

Classe locale – Premier exemple


```
public class ClasseExterne {
    public int i = 1; // accessible car membre externe
    public afficheToto() { System.out.println("toto"); }//

    public void mamethode() {
        final int j=10; // accessible car final
        int k=100; // inaccessible
        class ClasseLocale {
            int x = i;
```

Classe locale – Premier exemple


```
public class ClasseExterne {
    public int i = 1; // accessible car membre externe
    public afficheToto() { System.out.println("toto"); }//

    public void mamethode() {
        final int j=10; // accessible car final
        int k=100; // inaccessible
        class ClasseLocale {
            int x = i;
            int y = j;
```

Classe locale – Premier exemple


```
public class ClasseExterne {
    public int i = 1; // accessible car membre externe
    public afficheToto() { System.out.println("toto"); }//

    public void mamethode() {
        final int j=10; // accessible car final
        int k=100; // inaccessible
        class ClasseLocale {
            int x = i;
            int y = j;
            public void affiche() { afficheToto() }
        }
```

Classe locale – Premier exemple


```
public class ClasseExterne {
    public int i = 1; // accessible car membre externe
    public afficheToto() { System.out.println("toto"); }//

    public void mamethode() {
        final int j=10; // accessible car final
        int k=100; // inaccessible
        class ClasseLocale {
            int x = i;
            int y = j;
            public void affiche() { afficheToto() }
        }
        ClasseLocale loc1 = new ClasseLocale();
        System.out.println(loc1.x);  // affiche  1
        System.out.println(loc1.y);  // affiche  10
        loc1.affiche;  // affiche  toto
}   }
```

Problème d'Itérateur et Itérable


```
// Java Standard
public interface Iterator<E>{
    boolean hasNext();
    E next() throws NoSuchElementException;
    void remove() throws UnsupportedOperationException,
                         IllegalStateException;
}

public interface Iterable<E> {
    Iterator<E> iterator()
}
```

Si une classe `C` implémente `Iterable<E>` alors toute variable `liste` de type `C` peut être utilisée comme `for(E element: liste)`

Problème d'Itérateur et Itérable (2)


```
public class MaCelluleDEntier {
    private int valeur;
    private int suivant;
    ...
}

public class MaListeDentier {
    MaCellule tete;
    ...
}
```

But

Faire en sorte que `MaListeDentier` implémente `Iterable<int>`

Solution sans classe locale (1)


```
public class MonIterateur implements Iterator<int> {
    MaCelluleDEntier courante;
    MonIterateur(MaCelluleDEntier courante) { ... }

// méthodes requises pour implémenter Iterator<int>
    boolean hasNext()   {  return (courante != null);  }

    int next() throws NoSuchElementException {
        int i = courante.getValeur();
        courante = courante.getSuivante();
        return i;
    }
```

Solution sans classe locale (1)


```
public class MonIterateur implements Iterator<int> {
    MaCelluleDEntier courante;
    MonIterateur(MaCelluleDEntier courante) { ... }

// méthodes requises pour implémenter Iterator<int>
    boolean hasNext()   {  return (courante != null);  }

    int next() throws NoSuchElementException {
        int i = courante.getValeur();
        courante = courante.getSuivante();
        return i;
    }

    void remove() throws UnsupportedOperationException,
                         IllegalStateException {
        throw new UnsupportedOperationException();
    }
}
```

Solution sans classe locale (2)


```
public class MaListeDentier implements Iterable<int> {
    MaCellule tete;
    ...

// méthode requise pour implémenter Iterator<int>
    Iterator<int> iterator() {
        return (new MonIterator(tete));
    }
}
```

Solution sans classe locale (2)


```
public class MaListeDentier implements Iterable<int> {
    MaCellule tete;
    ...

// méthode requise pour implémenter Iterator<int>
    Iterator<int> iterator() {
        return (new MonIterator(tete));
    }
}
```

Cette méthode est lourde: elle oblige l'ajout d'une nouvelle classe.

Avec classe locale


```
public class MaListeDentier implements Iterable<int>{
    MaCellule tete;
    ...
// méthode requise pour implémenter Iterator<int>
  Iterator<int> iterator() {
        class MonIterateur implements Iterator<int> {
            MaCelluleDEntier courante=tete;

            public boolean hasNext()   { ... }
            public int next() throws NoSuchElementExceptio
            void remove() throws UnsupportedOperationExcep
                                    IllegalStateException { .
        }
        return (new MonIterateur());
    }
}
```

Et avec une classe anonyme


```
public class MaListeDentier implements Iterable<int>{
    MaCellule tete;
    ...
// méthode requise pour implémenter Iterator<int>
    Iterator<int> iterator() {
        return (new Iterator<int> {
            MaCelluleDEntier courante=tete;

            public boolean hasNext()   { ... }
            public int next() throws NoSuchElementExceptio
            void remove() throws UnsupportedOperationExcept
                                    IllegalStateException { .
        })
    }
}
```

Les classes anonymes sont utiles pour les IG


```
class MaFenetre extends JFrame {
    ...

    JButton bouton;
     void actionDuBouton(ActionEvent evt) { ... };

    MaFenetre ( ... ) {  // Constructeur
        ...
        bouton.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent evt){
                actionDuBouton(evt);
            }
        })
    }
}
```

Parce que dans la réalité...


```
class MaFenetre extends JFrame {
    ...

    JButton bouton1;
    JButton bouton2;
    JButton bouton3;
    JButton bouton4;
    JButton bouton5;

    ...
}
```