Java Avancé

Chapitre1:
Classe Interne, Anonyme

Pr: A.IDALENE

Objectifs

- Les *inner-class*: Les classes internes (non statiques) de classe
- Les classes interne statique de classe
- Les classes internes de méthode
- Les classes anonymes de méthode

Introduction

Le langage Java autorise la définition d'une classe à l'intérieur d'une autre classe;

- Un des intérêts de cette notation est qu'une classe interne peut avoir accès aux méthodes est attributs de la classe englobante.
- Comme les attributs et les méthodes, les classes internes ont une visibilité.
- La classe interne est une classe distincte, donc un fichier .class distinct est généré pour elle.

Après la compilation de ce programme, il y aura deux classes:

- Englobante.class
- et Interne.class

```
public class Englobante {
int a;
class Interne{
      int k = a; }
}
```

Classe interne

```
public class Englobante {
        private int x = 7;

        // définition de la inner-class
        class Interne {
                public void voirEnglobante() {
                System.out.println("Englobante -> x est " + x);
                        }
        } // fermeture de la classe Interne
    } // fermeture de la classe Englobante
```

Notez que la classe interne accède à tout membre de la classe englobante, car la classe interne est également membre de la classe englobante.

Instantiation de Classe internes à l'intérieure de la classe Englobante

```
public class Englobante {

    public Interne creerNouvelleInstanceClasseInterne() {
        Interne interne = new Interne();
        return interne;
    }

    class Interne {
    // corps de la classe interne
    }
}
```

Cette syntaxe fonctionne car la méthode de la classe englobante effectue l'instanciation, il existe déjà une instance de la classe englobante - l'instance exécutant la méthode creerNouvelleInstanceClasseInterne().

Instantiation de Classe internes à l'extérieure de la classe Englobante

À l'extérieure de la classe englobante, la classe interne ne vous est pas accessible de la manière habituelle. cous ne pouvez pas écrire:

```
Interne interne = new Interne();
```

Cette instruction n'est pas correcte, car la seule façon d'accéder à la classe interne (non-static) est via une instance de la classe englobante.

```
Englobante englobante = new Englobante();
Englobante.Interne interne = englobante.new Interne();
```

Une Classe interne (non static) a toujours besoin d'un objet de la classe englobante pour exister.

Classe interne : Exemple

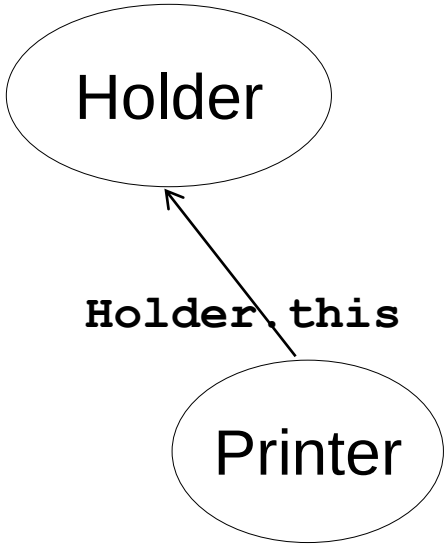
```
public class Sequence {
private int[] array;

public class SubSequence {
    public SubSequence(int lenght) {
    array = new int[lenght];
    for (int i=0; i<lenght; i++)
    array[i]=(int)(Math.random()*100);}

public int charAt(int index) {
      if (index<0 || index>=array.length)
        throw new IllegalArgumentException();
      return array[index];}
  }
public static void main(String[] args) {
       Sequence sequence = new Sequence();
       Sequence.SubSequence subSequence = sequence.new
SubSequence(5);
       System.out.println(subSequence.charAt(2));}
}
```

Référence sur la classe englobante et sur la classe interne

- Dans la classe interne, la référence « this » fait référence à l'instance de la classe interne. « this » fait toujours référence à l'objet en cours d'exécution.
- Que faire si la classe interne veut une référence explicite à l'instance de classe englobante à laquelle l'instance interne est liée?
- Remarques importantes:
- Pour référencer l'instance de classe interne, à partir du code de classe interne (elle-même), utilisez « this ».
- Pour référencer la classe englobante à partir du code de classe interne, utilisez NomClassEnglobante.this (par exemple, Englobante.this).

Référence sur la classe englobante

```
public class Holder {
public void print() {
    System.out.println(this);
  }
  public class Printer {
    public void print() {
      System.out.println(this);
      System.out.println(Holder.this);}
  }
  public static void main(String[] args) {
  Holder holder = new Holder();
  Holder.Printer printer = holder.new Printer();
  holder.print();
  printer.print();  }
}
```



Classe interne et membre statique

- Il est interdit de déclarer des membres statiques à l'intérieur d'une inner-class

```
public class A {
   public class B {
   static void m() { // erreur de compilation }
  }
}
```

- Il est possible de déclarer ce membre dans la classe englobante.

```
public class A {
   public class B { }
   static void m() {  }
}
```

Classe interne statique

Une classe interne statique ne peut accéder qu'aux membres statiques de la classe englobante :

```
public class ClasseExterne {
private int compteur = 0;
    private static String nom = "ClasseExterne";

    static class ClasseInterne
    {
        private int index = 0;

        public ClasseInterne()
        {
            System.out.println("Création d'un objet dans
"+nom);
        }
    }
```

Classe interne statique

Classe interne statique n'a pas besoin d'un objet de la classe englobante pour exister.

On peut alors créer une instance de la classe statique, sans qu'il existe d'instance de la classe englobante;

Dans ce cas, on ne peut pas accéder aux attributs ou méthodes autres que *static*, de la classe englobante.

Le nom de la classe interne est ClasseExterne.ClasseInterne

Lors de la compilation, le compilateur génère un fichier constitué du nom de la classe englobante puis un ‘$’, puis le nom de la classe interne, exemple :

ClasseExterne$ClasseInterne.class

Classe interne statique

```
public class Coords {

public static class Pair {
    private int x,y;
    Pair(int x1, int y1){
        x=x1;
        y=y1;}
        }
    public static Pair nouveauPair(int x,int y) {
        return new Pair(x,y);}

    public static void main(String[] args) {
        Coords.Pair pair=Coords.nouveauPair(1,2);  }
 }
```

Le compilateur génère deux classes différentes :
Coords.class et Coords$Pair.class

Interface interne

- Il est possible de définir des interfaces à l'intérieure de classe ou d'interface;
- Une interface interne est toujours statique;
- Une classe interne d'interface est toujours statique;

```
class MyClass {
  public interface I { // interface interne statique
  }
}

interface MyInterface {
  public class A {    // classe interne statique
  }
}
```

Visibilité

- Les classes internes ont une visibilité, elles sont des membres de la classe englobante;
- Le constructeur par défaut de cette classe possède la même visibilité;

```
public class Coords {
  private static class Pair {
    // le compilateur génère un constructeur privé
  }
}
```

Accès et visibilité

- Une classe englobante à accès à **tous** les membres de sa classe interne (même privés)
- Une classe interne **statique** a accès à **tous** les membres **statiques** de la classe englobante
- Une classe interne non statique a accès à **tous** les membres de la classe englobante

Exemple d'accès

- La classe englobante **Coords** à accès au champs privée de la classe interne statique **Pair** car ceux-ci sont situé dans le même fichier.

```
public class Coords {
  private Pair[] array;

  public int getX(int index) {
    return array[index].x; // accès à x
  }
  private static class Pair {
    private int x,y;

  }
```

Les classes internes de méthode

Une classe interne est définie à l'intérieur d'une autre classe, au même niveau que les autres membres. Vous pouvez également définir une classe interne au sein d'une méthode:

```
public class Englobante2 {
    void faireTraitement() {
    /* définition d’une classe interne
    à l’intérieure d’une méthode*/
    class Interne {
        public void voirEnglobante2() {
        System.out.println("Englobante2 " + Englobante2.this);
        }
        }
    }
}
```

Les classes internes de méthode

Une classe interne est définie à l'intérieur d'une autre classe, au même niveau que les autres membres. Vous pouvez également définir une classe interne au sein d'une méthode:

```
public class Englobante {
    void faireTraitement() {
    /* définition d’une classe interne
    à l’intérieure d’une méthode*/
    class Interne {
        public void voirEnglobante() {
        System.out.println("Englobante " + Englobante.this);
        }
        }
    }
}
```

instanciation d'une classe interne de méthode

```
class Englobante {

    void faireTraitement() {
        class Interne {
            public void voirEnglobante() {
                System.out.println("Englobante2 : " +
                Englobante.this);
            }
        }
        // instanciation de classe interne
        Interne interne = new Interne();
        interne.voirEnglobante();

    } // fermeture de la méthode faireTraitement
} // fermeture de la classe Englobante
```

Classe interne de méthode

- Une classe interne de méthode ne peut être instanciée que dans la méthode où la classe interne est définie.
- La classe interne de méthode peut accéder à tous les membres de la classe englobante.
- Pour les modificateurs au sein d'une méthode: les mêmes règles s'appliquent aux classes internes de méthode qu'aux déclarations de variables locales. Vous ne pouvez pas, par exemple, marquer une classe interne de méthode locale avec un modificateur d’accès. Seuls les modificateurs que vous pouvez appliquer à une classe interne de méthode sont abstraits et final, mais jamais les deux en même temps.

Classe interne de méthode

- La classe interne de méthode peut utiliser les variables locales de la méthode dans laquelle elle se trouve.

```
public class Englobante {
    String x="attribut";

    void faireTraitement() {
        class Interne {
        String z = "variable locale";
        public void voirEnglobante() {
            System.out.println("x : " + x);
            System.out.println("z : " + z);}
        }
        Interne interne = new Interne();
        interne.voirEnglobante();
    }
}
```

Classe anonyme

- Classe anonyme est classes internes déclarées sans aucun nom de classe. Vous pouvez définir ces classes non seulement dans une méthode, mais même dans un argument à une méthode.
- La classe anonyme est un sous-type d'une interface ou d'une classe abstraite ou concrete;
- Les classes anonymes sont utilisées pour implémenter les méthodes d’une interface;

Classe anonyme

```
class Popcorn {
    public void pop() {
    System.out.println("popcorn"); }
}

public class Food {
    Popcorn p = new Popcorn() {
    public void pop(){ System.out.println( "popcorn for food");}
    };


    public static void main(String[] args) {

        Food food = new Food();
        Popcorn popcorn1= new Popcorn();
        Popcorn popcorn2 = food.p;
        popcorn1.pop();
        popcorn2.pop();
    }
}
```

Classe anonyme

Dans l’exemple précédent:

- Nous définissons deux classes, Popcorn et Food.
- Popcorn a une méthode, pop ().
- Food a un attribut de type Popcorn, p

L’ attribut p ne fait pas référence à une instance de Popcorn, mais à une instance d'une sous-classe anonyme (sans nom) de type Popcorn.

Classe anonyme

```
Popcorn p = new Popcorn() {
   public void pop(){ System.out.println( "popcorn for food");}
};
```

Dans ce code, nous déclarons une variable de référence, p, de type Popcorn. Ensuite, nous déclarons une nouvelle classe qui n'a pas de nom, mais qui est une sous-classe de Popcorn. L'accolade ouvre la définition de classe anonyme. La classe anonyme redéfinit la méthode pop de la superclasse Popcorn. L'intérêt de créer une classe interne anonyme est soit pour remplacer une ou plusieurs méthodes de la superclasse!, ou pour implémenter des méthodes d'une interface.

Classe anonyme de classe

```
class Popcorn {
    public void pop() {
    System.out.println("popcorn"); }
}

class Food {
Popcorn p = new Popcorn() {
    public void do() {
    System.out.println("anonymous do popcorn"); }

    public void pop() {
    System.out.println("anonymous popcorn");}
};

public void popIt() {
p.pop(); // OK,
p.do(); // Erreur }
}
```

Classe anonyme d'interface

```
interface Cookable {
        public void cook();
}

class Food {
    Cookable c = new Cookable() {
        public void cook() {
        System.out.println("anonymous cookable implementer");
        }
    };
}
```

Remarque: c'est la seule manière où vous verrez la syntaxe new Cookable () où Cookable est une interface plutôt qu'un type de classe non abstrait. Là, il ne s'agit pas d'instancier un objet Cookable, mais plutôt de créer une instance d'une nouvelle classe anonyme qui implémente une interface:

Classe anonyme d'interface

```
class MyWonderfulClass {
void go() {
    Bar b = new Bar();
    b.doStuff(new Foo() {
                  public void foof() {
                  System.out.println("foofy");
                  }
                  });
    }
}

interface Foo {
void foof();
}

class Bar {
void doStuff(Foo f) {}
}
```

Classe anonyme d'interface

```
interface Operation {
  int eval();
}

public class OperatorFactory {
public static Operation plus(int e1,int e2) {
    return new Operation() {
                                @Override
                                public int eval() {
                                  return e1+e2;
                                }
                              };
  }
}
```

Classe anonyme et Inner-class

```
public class Sequence {
private char[] array;

public void setArray(char[] array) {
this.array = array;
}

public int length() { return array.length; }

public int get(int index) { return array[index]; }

public Sequence sub(int offset) {
    return new Sequence() {
        public int length() { return array.length - offset; }

        public int get(int index) { return array[offset + index]; }
        };
    }
}
```

Classes anonymes & Limitation

- Au vu de la syntaxe, il est impossible de créer une classe anonyme :
  - Implantant plusieurs interfaces
  - Héritant d'une classe et implantant une interface
- Dans ces cas, il est toujours possible de déclarer une classe interne à une méthode