Cours 4 : Classes internes, anonymes

Rappel

- Deux formes de types abstraits en Java :
  1. Les interfaces.
  2. Les classes abstraites.

Rappel

- Deux formes de types abstraits en Java :
  1. Les interfaces.
  2. Les classes abstraites.
- Ce cours : moyen de définir rapidement des types concrets implémentant des types abstraits.

Classes internes

- Quatre sortes de classes internes !
  1. Classes internes **statiques** de classe.
  2. Les **inner-class** de classe.
  3. Les classes internes **de méthode**.
  4. Les classes **anonymes** de méthode.

Classes internes

- Qui est quoi ?

```
public class A {
  public static class B{
  }
}
```

```
public class A {
  public class B {
  }
}
```

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

```
public class A {
  public void m() {
    new Object() {
      ...
    }
  }
}
```

Classes internes

- Et ça ?

```
public class A {

}

public class B {

}
```

Classes internes

- Et ça ?

```
public class A {

}

public class B {

}
```

- Compile pas ! (une seule publique au max, déconseillé de toute façon).

Cours 4 : Classes internes, anonymes et énumérations

Classes internes statiques

- Classe interne qui n'a **pas** besoin d'**instance** de la classe englobante pour exister.
- Son nom est `ClasseEnglobante.ClasseInterne`.
- Utile pour cacher des détails d'implémentations.
- Accès uniquement aux champs et méthodes statiques de l'englobante.

```
public class Coords {
  private final ArrayList<Pair> pairs = new ArrayList<>() ;

  public void add(int x, int y) {
    Coords.Pair p = new Coords.Pair(x,y) ;
    pairs.add(p) ;
  }
  static class Pair { //visibilité package
    private final int x,y ;
    ...
  }
}
```

Cours 4 : Classes internes, anonymes et énumérations

Classes internes de classe (inner-class)

- Classe interne qui a besoin d'une instance de la classe englobante pour exister.
- Accès aux **champs** de l'objet englobant ! (contrairement aux classes internes statiques !)

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

  public class Sub {
    private final int offset ;
    private final int length ;

    public char charAt(int index) {
      if (index<0 || index>=length)
        throw new IllegalArgumentException(...) ;
      return array[offset+index] ; //champ de la classe englobante !
    }
    ...
  }
  ...
}
```

inner-class - instanciation

- Doit être construite **sur** un objet de la classe englobante !

```
Sub sub = new Sub() ; //compile pas ! (sinon quels champs ?)
Sequence.Sub sub2 = new Sequence.Sub() ; //compile pas !

Sequence seq = new Sequence("svink") ;
Sub sub3 = seq.new Sub(1,2) ; //ok
```

- Dans une instance de la classe englobante.

```
public class Sequence {
  ...

  public Sub subseq(int off) {
    return new Sub(...) ; //this par defaut
    //return this.new Sub(...) ;
  }

  public class Sub {
    ...
  }
}
```

inner-class - membres statiques

- Interdit de déclarer un membre statique dans une classe interne non statique !
- Car la classe interne est créée pour une instance spécifique de la classe englobante.

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

inner-class - membres statiques

- Interdit de déclarer un membre statique dans une classe interne non statique !
- Car la classe interne est créée pour une instance spécifique de la classe englobante.

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

- Mais bien sûr toujours possible dans la classe englobante !

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

Classes internes - résumé

- Manière rapide d'écrire une classe ayant une **référence** sur une classe englobante.
- L'inner-class a accès à tous les membres de la classe englobante (champs, méthodes, autres classes internes...), même privés.
- L'inverse est vrai : la classe englobante a accès à tous les membres de ses classes internes.

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

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

Classes internes

- Le compilateur génère deux classes différentes !

```
-rw-r--r-- 1 florian florian 851 oct. 1 15 :19 Coords.class
-rw-r--r-- 1 florian florian 448 oct. 1 15 :19 Coords$Pair.class
```

- La VM ne fait pas la différence entre une classe et une classe interne.

Cours 4 : Classes internes, anonymes et énumérations

Classe interne de méthode

- Déclaration d'une classe dans une méthode, sans modificateur de visibilité (peu utilisé).
- Classe visible que dans la méthode.
- Accès aux champs de la classe englobante.
- Jusqu'à Java 7 : accès aux variables locales **final** et paramètres **final** de la méthode.
- Pourquoi ?

Classe interne de méthode

- Déclaration d'une classe dans une méthode, sans modificateur de visibilité (peu utilisé).
- Classe visible que dans la méthode.
- Accès aux champs de la classe englobante.
- Jusqu'à Java 7 : accès aux variables locales **final** et paramètres **final** de la méthode.
- Pourquoi ?
  - Valeur des variables et paramètres stockés dans la classe au moment de son instanciation.
  - Imposer final le rappelle au développeur.

Classe interne de méthode

- Déclaration d'une classe dans une méthode, sans modificateur de visibilité (peu utilisé).
- Classe visible que dans la méthode.
- Accès aux champs de la classe englobante.
- Jusqu'à Java 7 : accès aux variables locales **final** et paramètres **final** de la méthode.
- Pourquoi ?
  - Valeur des variables et paramètres stockés dans la classe au moment de son instanciation.
  - Imposer final le rappelle au développeur.
- Depuis Java 8 : lecture possible (mais modification impossible).

Classe interne de méthode

```
public class Bar {
  private int aa=2 ;

  public void foo(int a, final int b) {
    int v = 1 ;
    final int v2 = 2 ;

    class A {
      void m() {
        System.out.println(a) ; //ok que JAVA 8
        System.out.println(v) ; //ok que JAVA 8
        System.out.println(v2) ; //ok
        System.out.println(b) ; //ok
        System.out.println(aa) ; //ok
        aa = 20 ; //ok
        v = 11 ; //compile pas
      }
    }
    new A().m() ;
  }
}
```

Cours 4 : Classes internes, anonymes et énumérations

Classes anonymes

Implémenter une interface **sans donner de nom** à la classe.

```
public File[] subDirectories(File dir) {
  return dir.listFiles(new FileFilter() {
    @Override
    public boolean accept(File pathname) {
      return pathname.isDirectory() ;
    }
  }) ;
}
```

- `FileFilter` : interface du JDK avec une méthode `accept`
- `listFiles` : méthode pouvant prendre 1 `Filter` en argument

Classes anonymes

Implémenter une interface **sans donner de nom** à la classe.

```
public File[] subDirectories(File dir) {
  return dir.listFiles(new FileFilter() {
    @Override
    public boolean accept(File pathname) {
      return pathname.isDirectory() ;
    }
  }) ;
}
```

- `FileFilter` : interface du JDK avec une méthode `accept`
- `listFiles` : méthode pouvant prendre 1 `Filter` en argument

Equivalent à :

```
public class DirectoryFilter implements FileFilter {
  @Override
  public boolean accept(File pathname) {
    return pathname.isDirectory() ;
  }
}
  public File[] subDirectories(File dir) {
    return dir.listFiles(new DirectoryFilter()) ;
  }
```

Syntaxe

- Création possible depuis :
  - Interface.
  - Classe abstraite.
  - Classe concrète.
- Syntaxe :

```
Type var = new Type(param1,param2...) {
  //définition de membres
  //(méthode/champs/classe)
};
```

Variables

- Comme inner-classes, accès aux champs de l'englobante.

Variables

- Comme inner-classes, accès aux champs de l'englobante.
- Comme pour les classes internes de méthodes : valeur des variables locales constantes (final) visibles.
- Depuis Java 8 : final pas obligatoire (mais modification impossible).
- Copie des variables à la création.

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

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

Limitations

- Vu la syntaxe, impossible d' :
  - Implémenter plusieurs interfaces.
  - Hériter d'une classe et implémenter une interface.
- Utiliser alors une classe interne de méthode.

Classes anonymes - intérêt

- Manière rapide d'implémenter une interface/classe sans créer de nouveau fichier.
  - Beaucoup utilisé pour les interfaces graphiques, les threads...

Résumé

	inner static	inner class	inner meth	anon
Modif visib	Oui	Oui		
Accès champs objet englob		Oui	Oui	Oui
Acces variables locales			Oui	Oui
Sous-type de plusieurs type	Oui	Oui	Oui	

Quiz

Quiz