# Programmation Orientée Objet – Java

Bertrand Estellon

Département Informatique et Interactions
Aix-Marseille Université

12 novembre 2014

# Le mot-clé final

Première utilisation : interdire l'extension d'une classe

```
final public class Integer {

}
```

Deuxième utilisation : interdire la redéfinition d'une méthode

```
public class VariadicOperator {
  /* ... */
  final public double value() { /* ... */ }
  /* ... */
}
```

# Le mot-clé final

Troisième utilisation : interdire la modification d'un attribut

```
final public class Integer {
  private final int value;

  public Integer(int value) { this.value = value; }
}
```

- L'attribut doit être initialisé après la construction de l'instance ;
- La valeur de l'attribut ne peut plus être modifiée ensuite.

# Le mot-clé final

Quatrième utilisation : interdire la modification d'une variable

```
public class Stack<T> {
  /* ... */
  public T pop() {
    final T top = array[size-1];
    array[size-1] = null; size--;
    return top;
  }
  /* ... */
}
```

# Les classes anonymes

Supposons que nous ayons l'interface suivante :

```
public interface ActionListener {
  public void actionPerformed(ActionEvent event);
}
```

Il est possible de :

- définir une classe anonyme qui implémente cette interface ;
- d'obtenir immédiatement une instance de cette classe

```
ActionListener listener = new ActionListener() {
  public void actionPerformed(ActionEvent event) {
    counter++;
  }
});
```

# Les classes anonymes

```
public class Window {
  private int counter;

  public Window() {
    Button button = new Button("count");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        counter++;
      }
    });
  }
}
```

# Les classes anonymes

Il est possible d'utiliser des attributs de la classe "externe" :

```
public class Window {
  private Counter counter = new Counter();

  public Window() {
    Button button = new Button("count");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        counter.count();
      }
    });
  }
}
```

# Les classes anonymes

Il est possible d'utiliser des variables **finales** de la méthode :

```
public class Window {

  public Window() {
    final Counter counter = new Counter();
    Button button = new Button("count");
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent event) {
        counter.count();
      }
    });
  }
}
```

# Java 8 : Lambda expressions

```
public interface ActionListener {
  public void actionPerformed(ActionEvent event);
}
```

Avec Java 8, il est possible d'écrire directement :

```
public class Window {
  public Window() {
    Button button = new Button("button");
    button.addActionListener(
        event -> System.out.println(event)
    );
  }
}
```

**Explication** : `ActionListener` possède une seule méthode donc on peut affecter une lambda expression à une variable de type `ActionListener`.

# Java 8 : Lambda expressions

Les trois interfaces suivantes sont définies :

```
public interface Predicate<T> {
  public boolean test(T t);
}

public interface Function<T,R> {
  public R apply(T t);
}

public interface Consumer<T> {
  void accept(T t);
}
```

# Java 8 : Lambda expressions

On peut écrire directement :

```
persons
   .stream()
   .filter(person -> person.getAge() >= 18)
   .map(person -> person.getName())
   .forEach(name -> System.out.println(name));
```

Types des paramètres et retours des méthodes :

- `stream()` → `Stream<Person>`
- `filter(Predicate<Person>)` → `Stream<Person>`
- `map(Function<Person, String>)` → `Stream<String>`
- `forEach(Consumer<String>)`

# Java 8 : Faire référence à une méthode

On peut faire référence à une méthode :

```
persons
   .stream()
   .map(Person::getName) /* person->person.getName() */
   .forEach(name -> System.out.println(name));
```

Un autre exemple :

```
String[] strings = { "Truc", "Machin", "Bidule" };
Arrays.sort(strings, String::compareToIgnoreCase);
                  /* (s1,s2)->s1.compareToIgnoreCase(s2)); */
```

Il est également possible de (Java Tutorials) :

- faire référence à une méthode d'une instance ;
- faire référence à un construction.