Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

Programmation et TP

T1A

Réseau et Sécurité

S20 Lambdas, classe anonymes et généricité

Roten Marc

2017/2018

Table des matières

1ª Implémenter les classes `Entity` et `MyListener` en accord avec ce qui est décrit ci-contre : spécification, programme principal et effet attendu.

Les instances de la classe `Entity` peuvent être *observées* : l'observateur (listener) s'abonne auprès d'un objet en vue de recevoir une notification à chaque invocation de setValue() (principe du pattern « observer »).

```
public class Entity {
  public Entity();
  public String getValue();
  public void setValue(String value);
  public void setListener(
      ValueChangedListener listener);
}
```

```
pub
public interfac
  void onValueC
}
```

```
public class Ex1Er
  public static vo
    Entity entity
    ValueChangedLi
    listener = new
    entity.setValu
    entity.setList
    entity.setValu
    entity.setValu
  }
}
```

Affichage lors

```
An entity cha
An entity cha
```

1ᵇ Adapter le `main()` pour définir l'observateur via une classe anonyme.

1ᶜ Adapter le `main()` pour définir l'observateur via une expression lambda.

1ᵈ Pourrait-on définir un observateur "intrusif et têtu" qui annule toute mise à jour systématiquement l'entité avec la valeur "`Apple`" ?

1 Exercice 1

1.1 Ex1a

```
package s20_1a;

public class Entity {
   public String entityValue;
   ValueChangedListener listener;
   public Entity() {
     entityValue = "vide";
   }
   public String getValue() {
     return entityValue;
   }
   public void setValue(String value){
     entityValue = value;
     if(listener != null)
       listener.onValueChanged(this);
```

```
    }
  public void setListener(ValueChangedListener listener) {
    this.listener = listener;
  }
}
```

```
package s20_1a;

public class Ex1EntityDemo extends MyListener {
  public static void main(String[] args) {

    Entity entity = new Entity();
    ValueChangedListener listener;
    listener = new MyListener();
    entity.setValue("abcd");
    entity.setListener(listener);
    entity.setValue("toto");
    entity.setValue("titi");
  }
}
```

```
package s20_1a;

public class MyListener implements ValueChangedListener {
   @Override
  public void onValueChanged(Entity e){
    System.out.println("An entity changed his value to:
       "+e.entityValue);
  }
}
```

```
package s20_1a;

public interface ValueChangedListener {
  void onValueChanged(Entity e);
}
```

1.2 Ex1b

```
package s20_1b;

import s20_1a.Entity;
import s20_1a.MyListener;
import s20_1a.ValueChangedListener;
```

```
public class Ex1EntityDemo extends MyListener{
  public static void main(String[] args) {
    Entity entity = new Entity();
    ValueChangedListener listener;
    listener= new ValueChangedListener(){
      @Override
      public void onValueChanged(Entity e){
        System.out.println("An entity changed his value to: "
          +e.getValue());
      }
    };
    entity.setValue("abcd");
    entity.setListener(listener);
    entity.setValue("toto");
    entity.setValue("titi");
  }
}
```

1.3 Ex1c

```
package s20_1c;

import s20_1a.Entity;
import s20_1a.MyListener;
import s20_1a.ValueChangedListener;


public class Ex1EntityDemo extends MyListener{
  public static void main(String[] args) {
    Entity entity = new Entity();
    ValueChangedListener listener;
    listener =(e)->{
      System.out.println("An entity changed his value to: "
        +e.getValue());
    };
    entity.setValue("abcd");
    entity.setListener(listener);
    entity.setValue("toto");
    entity.setValue("titi");
  }
}
```

1.4 Ex1d

```
package s20_1d;
```

```
import s20_1a.Entity;
import s20_1a.ValueChangedListener;

public class MyListener implements ValueChangedListener {
    @Override
   public void onValueChanged(Entity e){
     System.out.println("An entity changed his value to: APPLE");
   }
}
```

2 Ex2

```
package s20_2;

public class Ex2Shoes {
  public static class Shoes
  {
    private final String type;
    private final String color;
    private final int size;
    public Shoes( String type,String color,int size)
    {
      super();
      this.type = type;
      this.color = color;
      this.size = size;
    }
    public String getType() {
      return type;
    }
    public String getColor() {
      return color;
    }
    public int getSize() {
      return size;
    }
    public String toString() {
      return " \nla chaussure est de type : " +this.getType() +
          " \nla couleur est : " + this.getColor() +
          " \nla taille est de : " +this.getSize();
    }
  }
  //================================================
  public interface ShoesSelector<T> {
    public boolean isOk(Shoes s, T wantedValue);
```

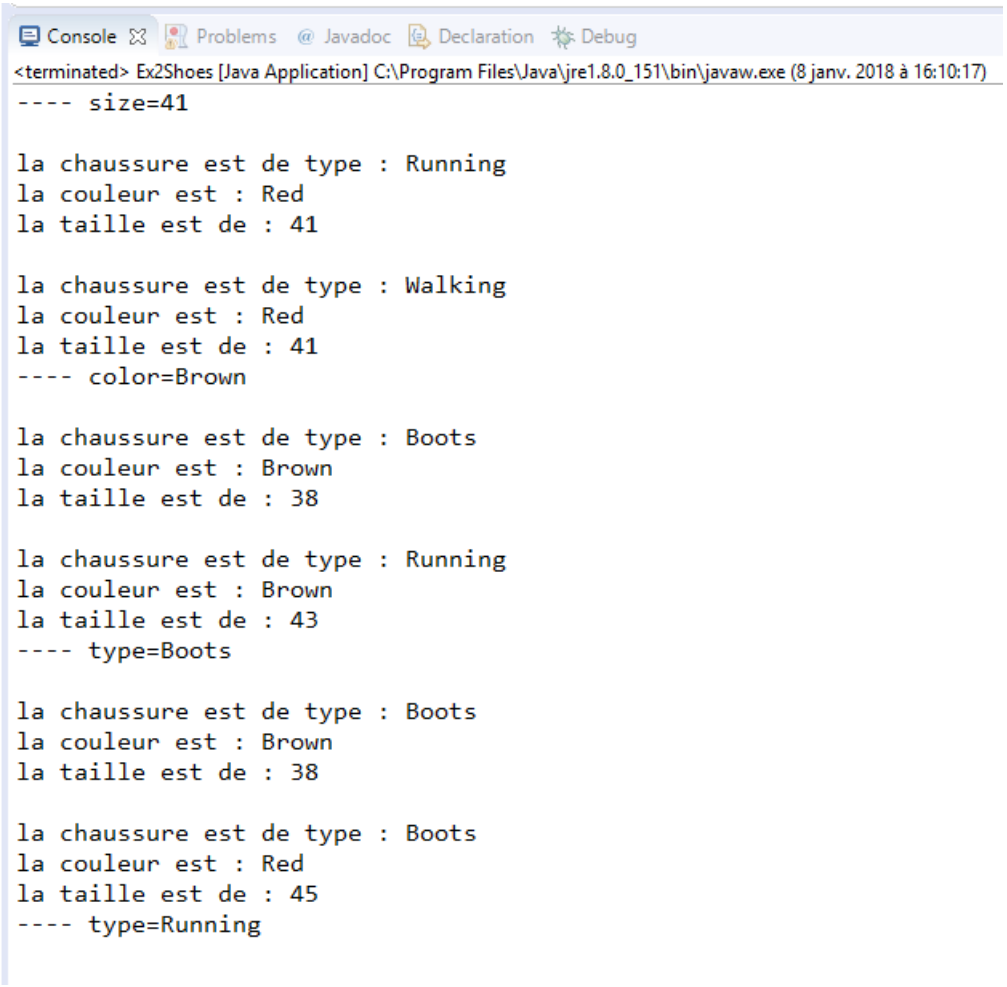
```
}
static class BySizeSelector implements ShoesSelector<Integer>
{
  @Override
  public boolean isOk(Shoes s, Integer wantedValue) {

    return s.getSize() == wantedValue;
  }
}

public static void main(String[] args) {

  Shoes[] shoesArray = new Shoes[]
      {
          new Shoes("Running", "Red", 41),
          new Shoes("Boots", "Brown", 38),
          new Shoes("Running", "Brown", 43),
          new Shoes("Running", "Yellow", 40),
          new Shoes("Walking", "Red", 41),
          new Shoes("Boots", "Red", 45)
      };
  //Print by size [with a separate class]
  System.out.println("---- size=41");
  printShoes(shoesArray, 41, new BySizeSelector());
  // Print by color [with an anonymous class]
  System.out.println("---- color=Brown");
  printShoes(shoesArray, "Brown", new ShoesSelector<String>()
  {
    @Override
    public boolean isOk(Shoes s, String wantedValue)
    {
      return s.getColor()==wantedValue;
    }
  });
  // Print by type [with a lambda expression]
  System.out.println("---- type=Boots");
  printShoes(shoesArray, "Boots",(s, wantedValue) ->{
    return s.getType()==wantedValue;
  });
  // Print by type [with a lambda expression of a standard
     "predicate" type]
  //... shoesSelector = (...) -> ...;
  System.out.println("---- type=Running");
  //printShoes1(shoesArray, "Running", shoesSelector);
}
public static <T> void printShoes(Shoes[] t, T wantedValue,
   ShoesSelector<T> c) {
  for(Shoes shoes : t){
    if (c.isOk(shoes, wantedValue))
```

```
            System.out.println(shoes.toString());
        }
    }
}
```



3 Ex3

3 Indiquer/expliquer lesquelles de ces définitions correspondent à des *interfaces fonctionnelles* Java.

```
@FunctionalInterface
public interface F1 {
  void d();
  void e();
}
```

```
public functional
interface F2 {
  void d();
}
```

```
public interface F3 {
  void d();
  default void e() {
    System.out.print("hi!");
  }
}
```

```
public interface F4 {
  void d();
  static void e() {
    System.out.print("hi!");
  }
}
```

Seule la fonction F2 est une interface fonctionnelle.

4 Ex4

4 Voici un extrait de la librairie standard.

```
package java.io;
public class File {
  ...
  public File(String pathname);
  /* Returns an array of strings naming the
     files and directories in the directory
     denoted by this abstract pathname. */
  public String[] list();
  /* The behavior of this method is the same
     as that of the list() method, except that
     the strings in the returned array must
     satisfy the filter.                  */
  public String[] list(FilenameFilter filter);
}
```

```
package java.io;
@FunctionalInterface
public interface FilenameFilter {
  /* Parameters:
  dir - the directory in which the file
        was found.
  name - the name of the file.
     Returns:
  true if and only if the name should be
  included in the file list.          */
  boolean accept(File dir, String name);
}
```

```
package java;
public class String { ...
  public boolean endsWith(String s);
}
```

Créer une classe `MyFileDemo` pour illustrer l'utilisation de ces composants, en affichant tous les fichiers images (fichiers avec l'extension `.jpg`, `.gif` ou `.png`) trouvés dans le répertoire courant ("."). Pour le filtre, vous êtes libres de coder une classe anonyme, une expression lambda, ou une classe interne.

package s20$_4$;

import java.io.File ; import java.io.FilenameFilter ; public class MyFileDemo implements FilenameFilter private String extension ; public MyFileDemo(String extension) this.extension=extension ; @Override public boolean accept(File directory, String filename) if(extension != null) return filename.endsWith('.'+extension) ; else return false ;

5 Ex5

5 Le programme ci-contre a un défaut : on calcule tout un tableau de résultats alors qu'un seul est finalement utilisé...

Pour corriger le problème, modifier la méthode `main()` en utilisant des *références de méthodes*.

Ceci illustre la notion d'*exécution différée*.

```
package java.util.function;
@FunctionalInterface
public interface DoubleSupplier {
  double getAsDouble();
}
```

```
public class Ex5MethodRef {
  private static Random rnd=new Random();

  static double randomValue(double[] t) {
    return t[rnd.nextInt(t.length)];
  }

  static double randomValue(DoubleSupplier[] t) {
    return t[rnd.nextInt(t.length)].getAsDouble();
  }

  static double hardComputation1() {...}
  static double hardComputation2() {...}
  static double hardComputation3() {...}

  public static void main(String[] args) {
    double[] t = {
        hardComputation1(),
        hardComputation2(),
        hardComputation3()
    };
    double d=randomValue(t2);
    System.out.println(d);
  }
}
```

```
package s20_5;

import java.util.Random;
import java.util.function.DoubleSupplier;
```

```
public class Ex5 {
  private static Random rnd=new Random();
  static double randomValue(double[] t) {
    return t[rnd.nextInt(t.length)];
  }
  static double randomValue(DoubleSupplier[] t) {
    return t[rnd.nextInt(t.length)].getAsDouble();
  }
  static double hardComputation1() {
    double res=0.0;
    for(int i=1; i<10_000_000; i++) {
      res+= Math.sin(Math.sqrt(Math.sqrt(Math.PI/i)));
    }
    return res;
  }
  static double hardComputation2() {
    double res=0.0;
    for(int i=1; i<10_000_000; i++) {
      res+= Math.atan(Math.pow(Math.PI/i, Math.sqrt(i)));
    }
    return res;
  }
  static double hardComputation3() {
    return Math.log(hardComputation1()+hardComputation2());
  }
  public static void main(String[] args) {
    System.out.println("start");
    double tt[] = {
       hardComputation1(),
       hardComputation2(),
       hardComputation3()
    };
    double d=randomValue(tt);
    System.out.println(d);
  }
}
```