M1 informatique – module GL et environnement pro. 1

TP2 : IHM avec MVC, framework Spring IoC et tests

Exercice de modélisation d'un panier de fruits (suite …)

Repartez du **projet Java** (basé sur maven) généré au moyen de Netbeans au **TP1 (Partie 3)** ou sinon, suivez le **tutoriel** : https://openclassrooms.com/fr/courses/4503526-organisez-et-packagez-une-application-java-avec-apache-maven/4608897-creez-votre-premier-projet-maven

I) IHM basée sur MVC

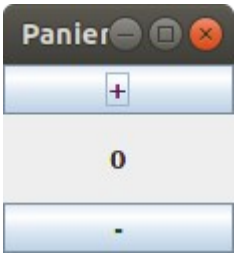
1) Inspirez-vous du projet https://github.com/roudetUb/Compteur.git pour créer une fenêtre graphique toute simple (avec la bibliothèque graphique **Swing**) pour manipuler votre Panier de fruits, en respectant le **patron MVC**. Cette fenêtre graphique permettra :

- d'afficher **combien** le panier en cours contient d'oranges,

- d'**ajouter** une orange (grâce à un simple bouton) : concrètement ceci ajoutera une orange (celle de votre choix) à la fin de la liste qui représente le panier, s'il n'est pas plein,

- ou de **retirer** une orange du panier (grâce à un simple bouton) : concrètement ceci retirera la dernière orange ajoutée au panier, s'il n'est pas vide,

- enfin pour les **cas exceptionnels** (lorsque le panier est plein ou vide) utilisez la méthode de votre choix (il ne se passe rien ou on affiche un message d'erreur).



Créer ainsi une classe **`VueGSwing.java`** qui hérite de `javax.swing.Jframe`, à l'aide d'un clic droit sur le projet, puis **New > JFrame Form...**

Voici un exemple de la fenêtre graphique que l'on doit obtenir :

Créer également une classe **`VueConsole.java`**, pour tracer textuellement les étapes réalisées par l'IHM.

2) Changement de framework graphique

Pour la vue créée précédemment, changer de framework graphique pour la faire passer en **AWT**. Créer donc une nouvelle classe `VueGAWT.java` (héritant de `java.awt.Frame`) pour remplacer la classe précédente et remarquer que grâce à l'architecture **MVC** adoptée, seule la partie présentation a été modifiée.

Voici à quoi ressemble la fenêtre précédente avec **AWT** :

Rappel : il est nécessaire d'ajouter les lignes suivantes dans le constructeur de la classe `VueGAWT.java` pour afficher correctement la fenêtre et pouvoir la fermer en cliquant sur « la croix » :

```
this.addWindowListener(new WindowAdapter(){
   public void windowClosing(WindowEvent e){
      System.exit(0);
   }
});
this.pack();              // à ajouter aussi avec Swing
this.setVisible(true);    // à ajouter aussi avec Swing
```

II) Spring IoC

Nous avons maintenant deux classes pour représenter notre fenêtre graphique (IHM). Le choix de la fenêtre à afficher se fait “en dur” dans la méthode “**main**” de la classe principale de notre application, appelée par la suite **TestMVC**.

Or il est possible, grâce au framework **Spring**, de configurer la fenêtre à afficher dans un **fichier XML**, de la façon suivante :

1) Ajout de la dépendance maven associée, dans le fichier pom.xml

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>2.0.5</version>
</dependency>
```

2) Réécriture de la méthode « main », de la classe TestMVC.java

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class TestMVC {
    private VueGraphique vueg;
    public VueGraphique getVueg() {   //pour java bean
        return vueg; }
    public void setVueg(VueGraphique vueg) {  //pour java bean
        this.vueg = vueg; }
    public static void main(String[] args){
      ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");
      TestMVC test = (TestMVC)context.getBean("MVC");
       … }
}
```

3) Création d’un fichier « beans.xml » dans src/main/resources/

Créer un fichier nommé par exemple « **beans.xml** » dans **src/main/resources/** dans lequel on indiquera ceci ("`VueGSwing.java`" étant la classe de la fenêtre graphique réalisée avec Swing et située dans le répertoire **src/main/java/fr/ufrsciencestech/panier/View**) :

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
       http://www.springframework.org/schema/beans/spring-beans.xsd">
  <!-- ===== Déclaration des références ===== -->
  <bean id="VueG" class="fr.ufrsciencestech.panier.View.VueGSwing"></bean>
  <!-- ===== Utilisation des références ===== -->
  <bean id="MVC" class="fr.ufrsciencestech.panier.TestMVC">
        <property name="vueg" ref="VueG"/>
  </bean>
</beans>
```

Plus de détails ici : https://openclassrooms.com/fr/courses/4504771-simplifiez-le-developpement-dapplications-java-avec-spring/4758013-linversion-de-controle-avec-spring

III) Tests unitaires, d'intégration et d'acceptation pour les vues

1) Écrire de nouvelles classes de test pour tester les **vues** :

- d'abord en **isolation**,

- puis via des **tests d'intégration** (en intégrant un « vrai » panier d'oranges).

Sachant qu'on peut simuler l'appui sur les boutons « + » et « - » de l'IHM, il est maintenant possible de réaliser des **tests d'acceptation** (ou de recette) pour vérifier par exemple les scenarii suivants :

- l'appui sur le bouton « + » ne rajoute plus d'orange dans le panier à partir du moment où celui-ci est plein,
- l'appui sur le bouton « - » ne supprime plus d'orange dans le panier à partir du moment où celui-ci est vide.

2) Écrire quelques **tests d'acceptation**, dans la classe de test de la vue graphique (IHM), pour s'assurer de la conformité de certains scenarii d'utilisation aux besoins des utilisateurs.

Pour simuler l'appui sur les boutons de l'IHM, il faut d'abord créer la classe **TestUtils.java** :

```
public class TestUtils {
    public static Component getChildNamed(Component parent, String name) {
         if (name.equals(parent.getName())) { return parent; }
         if (parent instanceof Container) {
            Component[] children = ((Container)parent).getComponents();
            for (int i = 0; i < children.length; ++i) {
               Component child = getChildNamed(children[i], name);
               if (child != null) { return child; }
            }
         }
         return null;
    }
}    // https://www.javaworld.com/article/2073056/swing-gui-programming/automate-gui-tests-for-swing-applications.html
```

Sa méthode **`getChildNamed`**`(Component parent, String` **`name`**`)` retourne une référence au composant (bouton, zone de texte, …) se trouvant dans la fenêtre graphique et portant le nom **name** (ce nom doit être spécifié au moment de la création du composant dans la classe de la fenêtre).

Pour simuler l'appui sur :

- un **JButton** (Swing), il faut appeler la méthode **`doClick()`** du bouton,
- un **Button** (AWT), il faut appeler la méthode **`dispatchEvent(ae)`** du bouton, juste après cette instruction :

```
ActionEvent ae = new ActionEvent((Object)plus, ActionEvent.ACTION_PERFORMED,"");
```

Voici un **exemple** de test d'acceptation pour le scenario suivant (réalisé au moyen de l'IHM) :
- on part d'un panier vide (de contenance maximum = 2),
- on simule l'appui sur le bouton « + » : il y a maintenant une orange dans le panier,
- puis on simule l'appui sur le bouton « - » : le panier est de nouveau vide.

```
public class VueGraphiqueTest {       //classe JUnit 4
  private static VueGraphique vueg;
  private Controleur c;               //contrôleur associé à la vue
  private Panier p;
  @Before
  public void setUp() {
     vueg = new VueGraphique();
```

```
    p = new Panier(2);
    c = new Controleur();
    c.setPanier(p);
    p.addObserver(vueg);        //la vue observe le modèle (panier)
    vueg.ajoutControleur(c);    //on associe le contrôleur à la vue
  }
  @Test
  public void testStoryAjouteRetire() {              //test d'acceptation
    assertNotNull(vueg);  // Instantiated?
    JLabel res = (JLabel)TestUtils.getChildNamed(vueg, "Affichage");
    assertNotNull(res); // Component found?
    final JButton plus = (JButton)TestUtils.getChildNamed(vueg, "Plus");
    assertNotNull(plus);
    final JButton moins = (JButton)TestUtils.getChildNamed(vueg, "Moins");
    assertNotNull(moins);
    plus.doClick();
    assertEquals(res.getText(), "1");
    moins.doClick();
    assertEquals(res.getText(), "0");
  }
…
}
```

Sans oublier, pour un **bouton** géré avec **Swing** (tels que les boutons « + » et « - » ci-dessus) :

```
//classe de l'IHM
public class VueGraphique extends JFrame implements Observer {
    private JButton inc, dec;
    public VueGraphique(){
        super("Panier");
        inc = new Jbutton("+");
        inc.setName("Plus");
        dec = new Jbutton("-");
        dec.setName("Moins");
        …
   }
}
```