NSY102

Conception de logiciels Intranet : Patrons et Canevas.

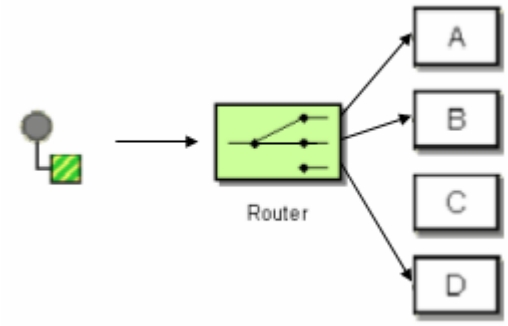
Session de Juin 2015-durée : 2 heures

Tous documents papiers autorisés

Cnam-CEP/HTT et FOD Nationale

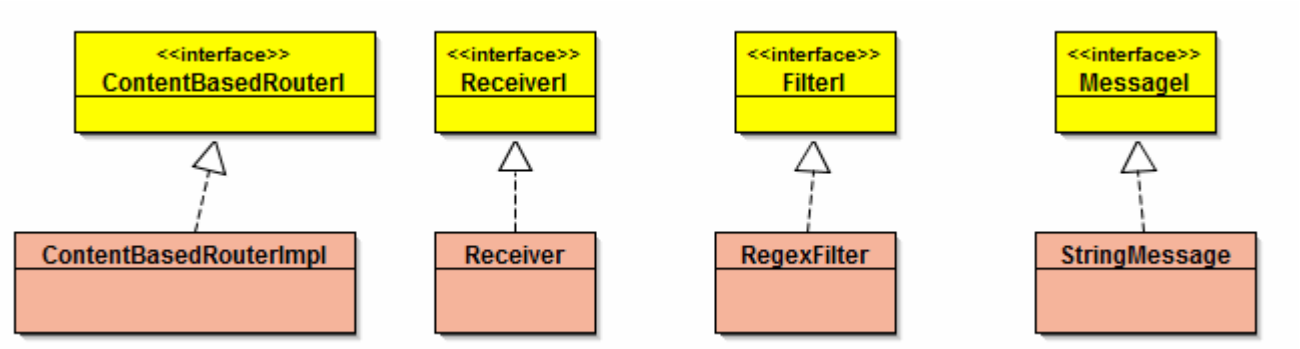
Le Patron *Content-Based Router*

Ce patron permet d'adresser un message à certains receveurs. Le choix des receveurs est effectué en fonction du contenu du message. Un filtre, associé à chaque receveur, au sein du routeur, précise le contenu souhaité du message. Ce filtre peut être mis à jour en cours d'exécution et par défaut un receveur réceptionne tous les messages émis. Un même receveur peut être présent dans plusieurs routeurs.



Source : http://www.enterpriseintegrationpatterns.com/

Ci dessous, une architecture de classes Java en notation BlueJ/UML, reflétant le Patron ***ContentBasedRouter***, les receveurs, les filtres et les messages,



L'interface **ContentBasedRouterI** ci-dessous**,** contient les opérations :

- d'installation d'un receveur (**`installReceiver`**),
- d'installation d'un filtre pour un receveur de ce routeur (**`setFilter`**),
- d'envoi d'un message aux receveurs (**`sendMessage`**),
- d'ajout d'un *listener*, déclenché à chaque exception levée par un receveur (**addExceptionListener**),
- d'obtention de la liste des receveurs installés (**`getReceivers`**),
- d'obtention du nom donné au routeur par l'utilisateur (**`getName`**).

```
public interface ContentBasedRouterI{

  /** Interface de notification lors d'une exception.
   * L'exception se produit lors d'un envoi de message et est levée par un receveur
   */
  public interface ExceptionListener{
    /** Méthode déclenchée à chaque exception le vée par un receveur.
     * @param router le routeur source de l'envoi
     * @param cause l'exception levée par le receveur
     */
    public void onException(ContentBasedRouterI router, ReceiverI r, Exception cause);
  }

  /** Installation d'un receveur de message.
   * Par défaut le receveur installé reçoit tous les messages émis.
   * @param receiver le receveur.
   */
  public void installReceiver(ReceiverI receiver);

  /** Installation d'un filtre de message auprès d'un receveur déjà présent.
   * @param receiver le receveur.
   * @param filter le filtre à appliquer sur le contenu du message.
   * @throws Exception si le receveur n'existe pas(n'a pas été installé).
   */
  public void setFilter(ReceiverI receiver, FilterI filter) throws
                                                    ReceiverNotInstalledException;

  /** Envoi d'un message aux receveurs installés.
   * Cet envoi est conditionné par la réussite du filtre.
   * A chaque exception d'un receveur, le listener préalablement
   * installé est déclenché, dans un thread séparé.
   * @param msg le message.
   * @return le nombre de receveurs auxquels ce message a été envoyé
   */
  public int sendMessage(MessageI msg);

  /** Ajout d'un listener. Appelé à la suite d'une exception levée,
   * par un receveur lors d'un envoi de messages.
   * L'unique listener est déclenché dans un thread distinct.
   * @param nl le listener
   */
  public void addExceptionListener(ExceptionListener nl);

  /** Obtention de la liste, un ensemble des receveurs installés.
   * @return l'ensemble des receveurs.
   */
  public Set<ReceiverI> getReceivers();

  /** Obtention du nom du routeur transmis à la création.
   * @return le nom de ce routeur.
   */
  public String getName();

}
```

L'interface **ReceiverI** est à implémenter par les receveurs du message :

```
public interface ReceiverI{

 /** Réception du msg à ce receveur.
   * @param msg le message.
   * @param router le routeur source de l'envoi
   * @Exception si ce receveur lève une exception.
   */
  public void receive(MessageI msg, ContentBasedRouterI router) throws Exception;

  /** Ajout du ContentBasedRouter auquel est abonné ce receveur.
```

```
    * @param router le routeur
    */
   public void addRouter(ContentBasedRouterI router);

   /** Obtention des routeurs auxquels est abonné de receveur.
    * @return l'ensemble des routeurs.
    */
   public Set<ContentBasedRouterI> getRouters();

   /** Obtention du nom du receveur.
    * @return son nom.
    */
   public String getName();
}
```

L'interface **FilterI** représente le filtre associé à un receveur, il détermine l'envoi du message au receveur selon son contenu.

```
public interface FilterI{

   public final FilterI ALL_MESSAGES = new FilterI(){
      public  boolean accept(MessageI msg){
        return true;
      }
     };
   public final FilterI NO_MESSAGES = new FilterI(){
      public  boolean accept(MessageI msg){
        return false;
      }
     };
   /** Filtre sur les messages.
    * @param msg le message.
    * @return true si le filtre accepte le message, false autrement.
    */
   public  boolean accept(MessageI msg);
}
```

L'interface **MessageI** représente le type du message.

```
public interface MessageI{

   public Object getContent();
}
```

Une classe de tests unitaires à lire attentivement:

```
public class TestContentBasedRouter extends junit.framework.TestCase{

  public static class ReceiverMock extends question1.Receiver/* cf.question 1-2*/{
    private MessageI msgReceived=null;
    public ReceiverMock(String name){super(name);}

    public void receive(MessageI msg, ContentBasedRouterI router) throws Exception{
      this.msgReceived = msg;
      if(msg.getContent().equals("EXCEPTION")){ // Test des exceptions
        this.msgReceived = null;
        throw new Exception("EXCEPTION");
      }
    }

    public MessageI getMsgReceived(){
      MessageI msg = msgReceived;
      msgReceived=null;
      return msg;
    }
  }
```

```
    public void testNormalStringMessage(){
     try{
       ContentBasedRouterI routerA = new ContentBasedRouterImpl("router_A");
       ReceiverMock a = new ReceiverMock("a");
       ReceiverMock b = new ReceiverMock("b");

       routerA.installReceiver(a);
       routerA.setFilter(a, new RegexFilter("[A-Z]+")); // toutes les lettres du message
                                                        // sont en majuscule
       routerA.installReceiver(b); // par défaut sans filtre, b reçoit tous les messages

       ContentBasedRouterI routerB = new ContentBasedRouterImpl("router_B");
       routerB.installReceiver(b);

       // envoi d'un message vers le router_A
       int number = routerA.sendMessage(new StringMessage("CNAM"));
       assertEquals("received number ? ", 2, number);
       String message = (String)a.getMsgReceived().getContent();
       assertEquals("message received ? ","CNAM",message);
       message = (String)b.getMsgReceived().getContent();
       assertEquals("message received ? ", "CNAM", message);

       // envoi d'un nouveau message vers le router_A
       number = routerA.sendMessage(new StringMessage("NSY102"));
       assertEquals(" received number ? ", 1, number); // seul b a reçu ce message

       // envoi d'un message vers le router_B
       number = routerB.sendMessage(new StringMessage("nsy102"));
       assertEquals(" received number ? ", 1, number);

       assertTrue(b.getRouters().contains(routerA));
       assertTrue(b.getRouters().contains(routerB));

     }catch(Exception e){
       fail(e.getMessage());
     }
    }

     public void testWithExceptionStringMessage() throws Exception{
       ContentBasedRouterI routerA = new ContentBasedRouterImpl("router_A");
       ReceiverMock a = new ReceiverMock("a");
       ReceiverMock b = new ReceiverMock("b");

       routerA.installReceiver(a);
       routerA.installReceiver(b);
       routerA.addExceptionListener(new OnException());

       int number = routerA.sendMessage(new StringMessage("EXCEPTION"));
       assertEquals(" received number ? ", 0, number); // aucun receveur
     }

    private static class OnException
     implements ContentBasedRouterI.ExceptionListener{

     public void onException(ContentBasedRouterI router, ReceiverI r, Exception e){
       System.out.println( e.getMessage() + " levée par : " + r + " depuis : " + router);
     }
    }
  }
```

Question1-1) 10 points)

Précisez quelles sont les structures de données utilisées, le fonctionnement de la méthode sendMessage, Ecrivez une implémentation **complète de la classe *ContentBasedRouterImpl***.

Vous pouvez compléter les dernières pages de cet énoncé afin de répondre aux questions, sans oublier de reporter le numéro de votre copie sur celles-ci.

Question1-2) 4 points)

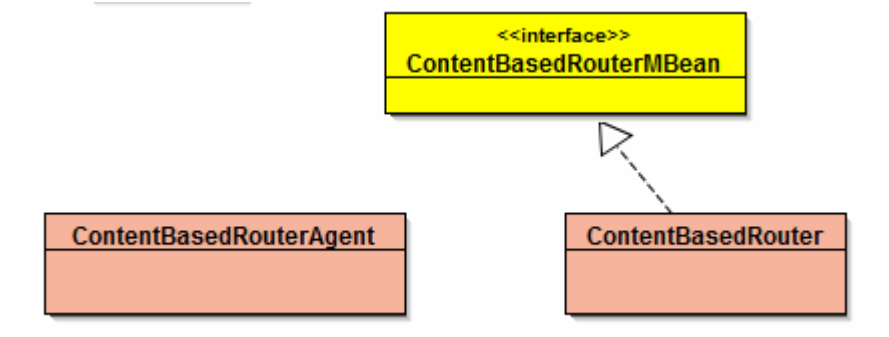
Ecrivez une implémentation **complète de la classe *Receiver***.

Question2 (6 points) : Patron *Content-Based Router* / JMX

Un routeur, une instance de la classe ContentBasedRouter devient un composant MBean et est accessible via la technologie JMX, les notifications correspondent aux exceptions levées lors des envois de messages par les receveurs, c'est un cumul de comportement qui est attendu : l'exception est traitée par le listener en place et une notification est émise aux souscripteurs JMX.

Une nouvelle interface est définie ***ContentBasedRouterMBean***, son implémentation pour JMX ***ContentBasedRouter*** et un agent ***ContentBasedRouterAgent*** dont le code source est ci-dessous

Une architecture de type UML/BlueJ



Le source de l'agent JMX:

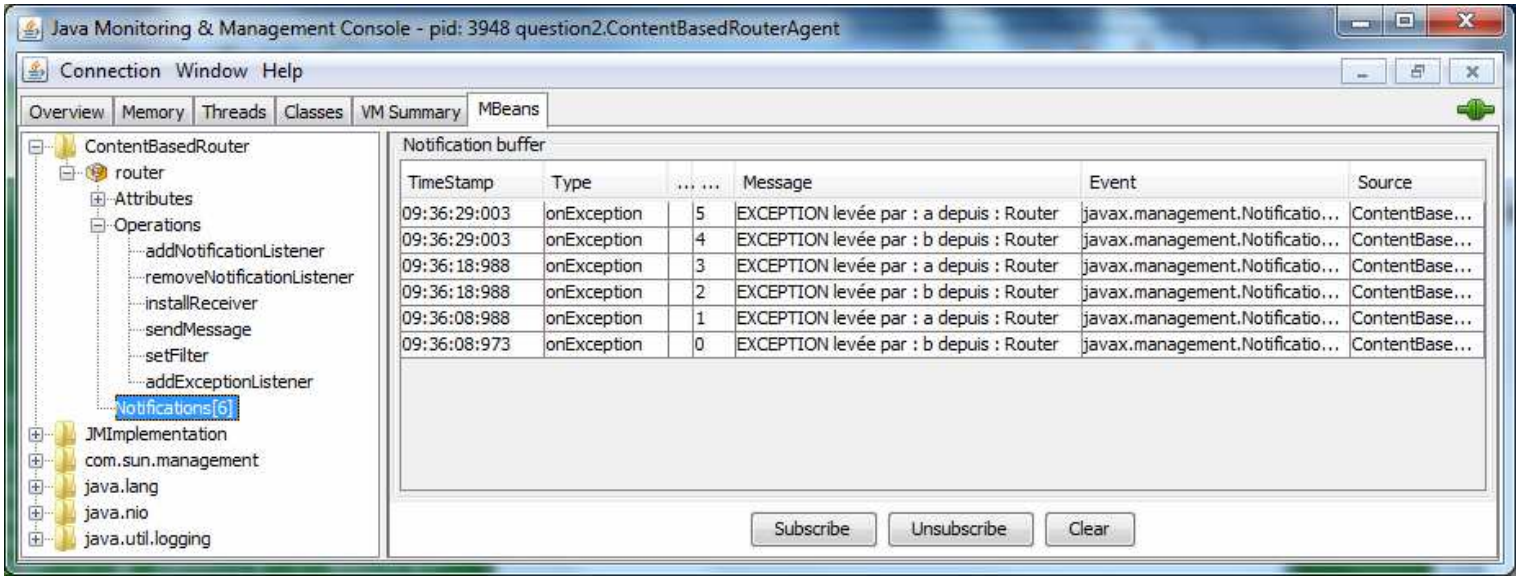
```
import question1.*;
import static question1.TestContentBasedRouter.ReceiverTest;

public class ContentBasedRouterAgent{

  private MBeanServer mbs;
  private ContentBasedRouterMBean router;

  public ContentBasedRouterAgent(){
    try{
      this.mbs = ManagementFactory.getPlatformMBeanServer();
      ObjectName name = new ObjectName("ContentBasedRouter:name=router");
      this.router = new ContentBasedRouter(new ContentBasedRouterImpl("Router"));
      mbs.registerMBean(router, name);    // enregistrement


      ReceiverTest a = new ReceiverTest("a");
      ReceiverTest b = new ReceiverTest("b");
      router.installReceiver(a);
      router.installReceiver(b);
      router.addExceptionListener(new OnException());

      Thread.sleep(40000);
      router.sendMessage(new StringMessage("EXCEPTION"));
      Thread.sleep(10000);
      router.sendMessage(new StringMessage("EXCEPTION"));
      Thread.sleep(10000);
      router.sendMessage(new StringMessage("EXCEPTION"));


    }catch(Exception e){
        //e.printStackTrace();
```

```
        System.out.println(" Exception : " + e.getMessage());
      }
    }

     private static class OnException
      implements ContentBasedRouterI.ExceptionListener{

      public void onException(ContentBasedRouterI router, ReceiverI r, Exception e){
        System.out.println( e.getMessage() + " levée par : " + r + " depuis : " + router);
      }
    }

    public static void main(String[] args)throws Exception{
      ContentBasedRouterAgent agent = new ContentBasedRouterAgent();
      Thread.sleep(5*60*1000);
    }

}
```

Trace sur la console, à la suite de l'exécution de la méthode main de ContentBasedRouterAgent

```
G:\FREECOM\NSY102\examen_juin_2015>java -cp . question2.ContentBasedRouterAgent
EXCEPTION levée par : b depuis : Router
EXCEPTION levée par : a depuis : Router
EXCEPTION levée par : b depuis : Router
EXCEPTION levée par : a depuis : Router
EXCEPTION levée par : b depuis : Router
EXCEPTION levée par : a depuis : Router
```

Avec l'outil JConsole : les six exceptions levées par les deux receveurs ont engendré six notifications



Question2-1) Ecrivez **l'interface ContentBasedRouterMBean**

Question2-2) Proposez une solution, avec éventuellement l'usage d'un patron pour la réalisation de la classe ***ContentBasedRouter***

Question2-3) Ecrivez la méthode ***addExceptionListener***, de la classe ***ContentBasedRouter***

Question 1 **Numéro de copie :**

```
public class ContentBasedRouterImpl implements ContentBasedRouterI{
  private String name;
  private ExceptionListener listener;

  private









  public ContentBasedRouterImpl(String name){
    this.name      = name;



  }









  public void installReceiver(ReceiverI receiver){



  }



  public void setFilter(ReceiverI receiver, FilterI filter) throws
ReceiverNotInstalledException{



  }
```

```
 public int sendMessage(MessageI msg){
    int number = 0;









    return number;
  }

  public void addExceptionListener(ExceptionListener nl){
    this.listener = nl;
  }



  public Set<ReceiverI> getReceivers(){

    return


  }

  public String getName(){return this.name; }
  public String toString(){return getName();}
}
```

Question 1 Numéro de copie :

```
public class Receiver implements ReceiverI{
  private String   name;

  private




  public Receiver(String name){
    this.name = name;



  }

  public Receiver(String name, ContentBasedRouterI router){
    this(name);



  }

  public void addRouter(ContentBasedRouterI router){



  }

  public void receive(MessageI msg, ContentBasedRouterI router) throws Exception{



  }



   public Set<ContentBasedRouterI> getRouters(){


     return


   }

   public String getName(){return this.name; }
   public String toString(){return getName();}
}
```

Question 2 Numéro de copie :

```
public final class ContentBasedRouter extends _____________________
 implements ContentBasedRouterMBean{



  private



  public void addExceptionListener(ExceptionListener nl){
```