NFP121

Programmation avancée

Session de février 2018-durée : 3 heures

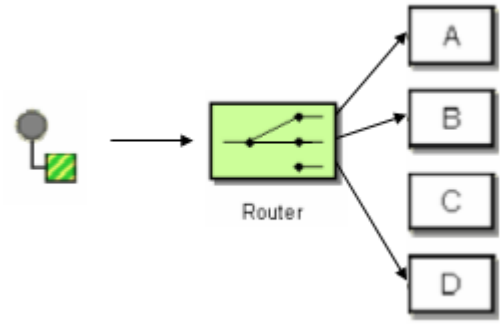
Tous documents papiers autorisés

Cnam-CEP/FOD Nationale

Sommaire :

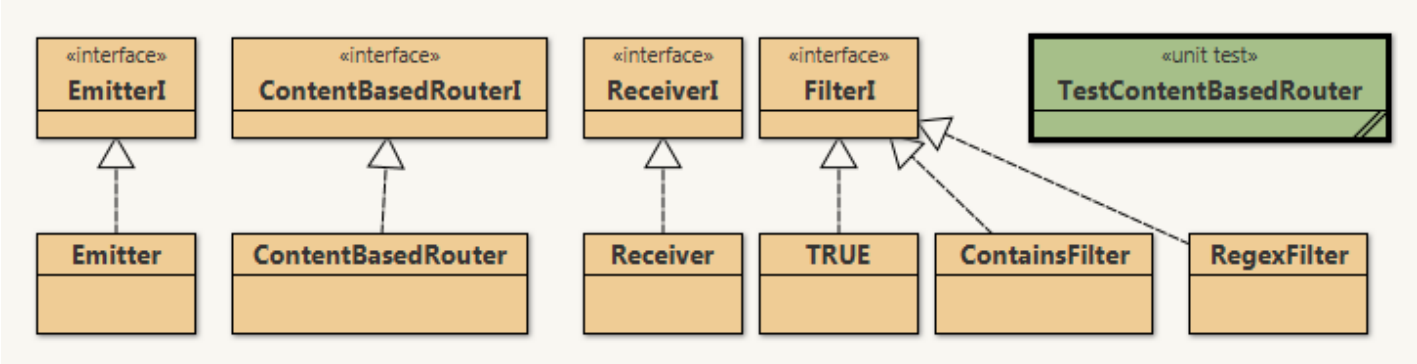
Question1) Le Patron *Content-Based Router*

Ce patron permet d'adresser un message à certains receveurs. Le choix des receveurs est effectué en fonction du contenu du message. Une liste de filtres associée à chaque receveur précise le contenu souhaité. Cette liste de filtres peut être enrichie en cours d'exécution et un receveur peut être inhibé temporairement.



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

Ci dessous, une architecture de classes Java en notation BlueJ/UML, reflétant le Patron *Content-Based Router*, les receveurs, les filtres, les messages émis sont uniquement de type String.



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

- d'ajout d'un receveur (`addReceiver`) , et de retrait d'un receveur (`removeReceiver`),
- d'ajout d'un filtre pour un receveur (`addFilter`),
- d'envoi d'un message (`sendMessage`),
- de validation (`setEnabled`) ou d'invalidation (`setDisabled`) d'un receveur,
- d'obtention d'un itérateur sur les receveurs installés (`getReceivers`),
- d'obtention de la liste des receveurs ayant levé une exception (`getReceiversException`),

```java
/** Patron ContentBasedRouter.
 * Ce patron permet d'adresser un message à certains receveurs.
 * Le choix des receveurs est effectué en fonction du contenu du message.
 * Une liste de filtres précise le contenu souhaité du message pour un receveur.
 * Cette liste peut être enrichie en cours d'exécution et un receveur être inhibé temporairement.
 * Le type de message est uniquement de type String.
 */
      public interface ContentBasedRouterI extends Iterable<ReceiverI>{

        /** Installation d'un receveur de message.
         * Par défaut, ce receveur ne peut recevoir de message(cf. setEnabled),
         * @param receiver le receveur.
         * @throws RuntimeException en cas de doublons
         * @return this
         */
        public ContentBasedRouterI addReceiver(ReceiverI receiver);

        /** Ajout d'un filtre pour un receveur. Les filtres sont cumulés en ET,
         * (en ET: tous les filtres doivent accepter le message)
         * @param receiver le receveur.
         * @param filter le filtre à appliquer sur le contenu du message.
         * @throws RuntimeException si le receveur n'existe pas, n'a pas été ajouté.
         * @return this
         */
        public ContentBasedRouterI addFilter(ReceiverI receiver, FilterI filter);

        /** Retrait d'un receveur de message.
         * @param receiver le receveur.
         * @throws RuntimeException si le receveur n'existe pas, n'a pas été ajouté.
         * @return this
         */
        public ContentBasedRouterI removeReceiver(ReceiverI receiver);

        /** Envoi d'un message à tous les receveurs.
         * Cet envoi est conditionné par la réussite de tous les filtres
         * et l'autorisation à recevoir. Les receveurs ayant levé une exception
         * sont inhibés et accessibles par la méthode getReceiversException.
         * @param msg le message.
         * @return le nombre de receveurs auxquels ce message a été envoyé et reçu
         */
        public int sendMessage(String msg);

        /** Autorisation de recevoir un message.
         * Par défaut, dès son ajout au routeur, le receveur est inhibé.
         * @param receiver le receveur à autoriser.
         * @throws RuntimeException si le receveur n'existe pas, n'a pas été ajouté.
         * @return this
         */
        public ContentBasedRouterI setEnabled(ReceiverI receiver) ;

        /** Blocage, inhibition de l'envoi d'un message vers ce récepteur.
         * @param receiver le receveur à inhiber.
         * @throws RuntimeException si le receveur n'existe pas, n'a pas été ajouté.
         * @return this
         */
        public ContentBasedRouterI setDisabled(ReceiverI receiver) ;

        /** Obtention d'un itérateur, sur les receveurs installés.
         * @return un itérateur.
         */
        public Iterator<ReceiverI> iterator();

         /** Obtention de l'ensemble des receveurs ayant levé une exception.
         * @return un ensemble des receveurs ayant levé une exception.
         */
        public Set<ReceiverI> getReceiversException();
      }
```

L'interface ***ReceiverI,*** est fournie :

```
public interface ReceiverI{

   /** Réception d'un message via le routeur.
    * @param msg le message reçu
    */
   public void receive(String msg) throws Exception;

   /** Réception d'un message via le routeur.
    * @return le nom du receveur
    */

   public String getName();

}
```

L'interface ***EmitterI,*** est fournie :

```
public interface EmitterI{

   /** Affectation du router.
    * @param router le routeur
    */
   public void setRouter(ContentBasedRouterI router);

   /** Envoi d'un message.
    * @param msg le message à envoyer au(x) routeur(s)
    * @return le nombre de receveurs ayant reçu le message
    */
   public int sendMessage(String msg);

   /** Obtention du nom de l'émetteur.
    * @return le nom de cet émetteur
    */
   public String getName();

}
```

L'interface ***FilterI*** représente le filtre en fonction du contenu du message, il détermine l'envoi effectif ou non du message au receveur.

```
   public interface FilterI{
    /** Filtrage du message.
     * @param msg le message en entrée
     * @return true si le filtre réussit, false autrement.
     */
   public boolean accept(String msg);
}
```

Une classe de tests unitaires, à lire attentivement avant de répondre aux questions :

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

          private static class ReceiverTest extends Receiver{
              private String msgReceived=null;

              public ReceiverTest(String name){
                  super(name);
              }

              public void receive(String msg){
                  this.msgReceived = msg;
                  if(msg.contains("exception"))throw new RuntimeException();
              }

              public String getMsgReceived(){
```

```
            String msg = msgReceived;
            msgReceived=null;
            return msg;
        }
    }

    public void testStringMessage(){
        try{
            ContentBasedRouterI router = new ContentBasedRouter("router");
            ReceiverTest a = new ReceiverTest("a");
            ReceiverTest b = new ReceiverTest("b");
            ReceiverTest c = new ReceiverTest("c");
            ReceiverTest d = new ReceiverTest("d");
            router.addReceiver(a).addReceiver(b).addReceiver(c).addReceiver(d);

            FilterI f = new TRUE();// filtre à succès
            router.addFilter(a,f).addFilter(b,f).addFilter(c,f).addFilter(d,f);
            EmitterI emitter = new Emitter("e1");
            emitter.setRouter(router);

            int number = emitter.sendMessage("meteo_pluie"); // il pleut...
            assertEquals(" received number ? ", 0, number); // aucun receveur autorisé
            router.setEnabled(a).setEnabled(b).setEnabled(c).setEnabled(d);
            number = emitter.sendMessage("meteo_pluie");
            assertEquals(" received number ? ", 4, number); // 4 receveurs autorisés

            router.addFilter(a, new ContainsFilter("meteo"));
            router.addFilter(b, new RegexFilter("[a-z]+"));
            router.addFilter(c, new RegexFilter("[A-Z]+"));
            number = emitter.sendMessage("meteo_pluie");
            assertEquals(" received number ? ", 2, number); // seuls a et d (ont reçu)

            router.setDisabled(a);
            number = emitter.sendMessage("meteo_soleil");
            assertEquals(" received number ? ", 1, number); // seul d (a reçu)

            router.removeReceiver(c);
            router.addReceiver(c).setEnabled(c);
            router.addFilter(c,new RegexFilter("[a-z_]+"));
            number = emitter.sendMessage("meteo_soleil");
            assertEquals(" received number ? ", 2, number); // c et d (ont reçu)

            number = emitter.sendMessage("meteo_exception");
            assertEquals(" received number ? ", 0, number);
            // pour les tests avec une exception silmulée
            int numberExceptions = router.getReceiversException().size();
            assertEquals(" receiver number with exception ? ", 2, numberExceptions);
            assertTrue(router.getReceiversException().contains(c));
            assertTrue(router.getReceiversException().contains(d));
            router.setDisabled(d);
            number = emitter.sendMessage("meteo_grêle");
            assertEquals(" received number ? ", 0, number);

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

}
```

Question1)

1-1) Ecrivez une implémentation **complète de la classe *ContentBasedRouter***. Certaines descriptions d'interfaces extraites du paquetage java.util sont en annexe. Vous pouvez utiliser la dernière page de cet énoncé pour votre réponse, **veillez à reporter sur celle-ci votre numéro de copie d'examen.**

1-2) Proposez la classe ***ContainsFilter***, qui permet de vérifier si le contenu du message contient un terme transmis lors de sa création.

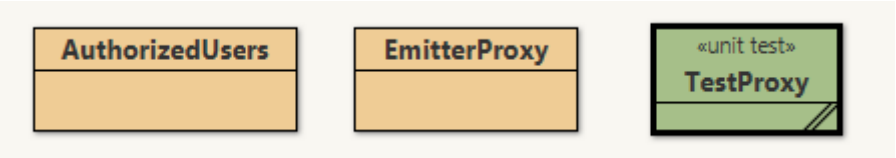
1-3) Ecrivez une implémentation de la classe ***Emitter***, laquelle implémente l'interface ***EmitterI***.

1-4) Que pourrait apporter à cette application l'utilisation d'un outil logiciel permettant l'injection de dépendances, un outil analogue à ***femtoContainer*** vu et utilisé en cours.

Question2) Les patrons Singleton et Proxy

Seuls certains émetteurs sont autorisés à envoyer un message au routeur, une liste de noms d'émetteurs issue de la classe ***AuthorizedUsers*** permet au mandataire ***EmitterProxy*** de contrôler les envois de messages vers le routeur de messages

La classe ***AuthorizedUsers*** est un Singleton, (une seule instance de cette classe est garantie) et la classe ***EmitterProxy*** reflète le patron Proxy.



Une classe de tests unitaires, à lire attentivement avant de répondre aux questions :

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

    public void testSimple(){
        ContentBasedRouterI router = new ContentBasedRouter("router");
        ReceiverI a = new Receiver("a");
        ReceiverI b = new Receiver("b");
        router.addReceiver(a).addReceiver(b);
        FilterI f = new TRUE();
        router.addFilter(a,f).setEnabled(a).addFilter(b,f).setEnabled(b);

        EmitterI emitter = new Emitter("inconnu"); // sans proxy
        emitter.setRouter(router);
        int nombre=emitter.sendMessage("meteo_pluie");
        assertEquals(2,nombre);                    // a et b ont reçu le message


        emitter = new EmitterProxy("inconnu"); // avec proxy, (contrôle du nom)
        emitter.setRouter(router);
        nombre=emitter.sendMessage("meteo_pluie");
        assertEquals(0,nombre);                 // aucun n'a reçu le message
    }
}
```

2-1) Ecrivez une implémentation **complète de la classe *AuthorizedUsers***. vous devez garantir une et une seule instance de la liste des utilisateurs agréés. Seuls les émetteurs dont les noms sont dans cette liste sont autorisés à envoyer des messages vers le routeur.

```
final public class AuthorizedUsers{
  ...
```

à compléter sur votre copie ,

mentionnez code pour singleton q2-1

```
...

public List<String> users(){
   return Collections.unmodifiableList(usersList);
}

private static List<String> usersList;

static{
     usersList = new ArrayList<String>(); // e1, e2, e3, e4
     usersList.add("e1");
     usersList.add("e2");
     usersList.add("e3");
     usersList.add("e4");
 }
}
```

2-2) Proposez la classe ***EmitterProxy***, qui permet de contrôler les émetteurs agréés de message vers le routeur.

Question3) Introspection et interface graphique

Une interface graphique en Swing a été développée dans le but de "tester" une instance du "routeur", et la réception des messages par 4 receveurs a, b, c d.

Test de l'envoi de messages : L'utilisateur/testeur entre un contenu de message, au clic sur le bouton **« envoyer »**, le message est envoyé au routeur, la fenêtre de résultat mentionne simplement le nombre de receveurs ayant reçu un message et la liste des receveurs installés ainsi que les receveurs injoignables.

Test, ajout d'un filtre : L'utilisateur/testeur entre le nom d'un filtre **(le nom d'une classe java),** de son paramètre et le nom du receveur cible. L'utilisateur clique sur le bouton **« charger »** le filtre ainsi créé, est ajouté à la liste des filtres du receveur.

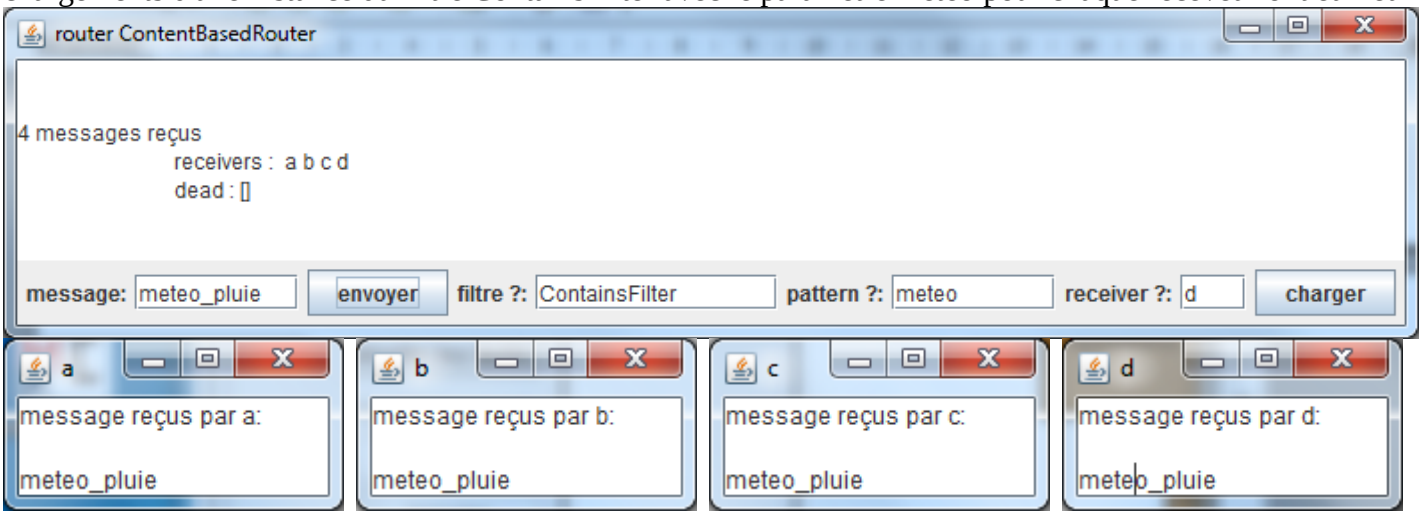
L'interface pour ce test utilise 5 fenêtres graphiques : une fenêtre pour le test du router et une pour chaque receveur (a, b, c, d). La fenêtre du routeur contient les fonctions d'envoi et de chargement du filtre. La fenêtre de chaque receveur affiche ce qu'il reçoit.

Toutes les exceptions sont affichées dans la fenêtre résultante (usage de getMessage(), issue de la classe Exception)

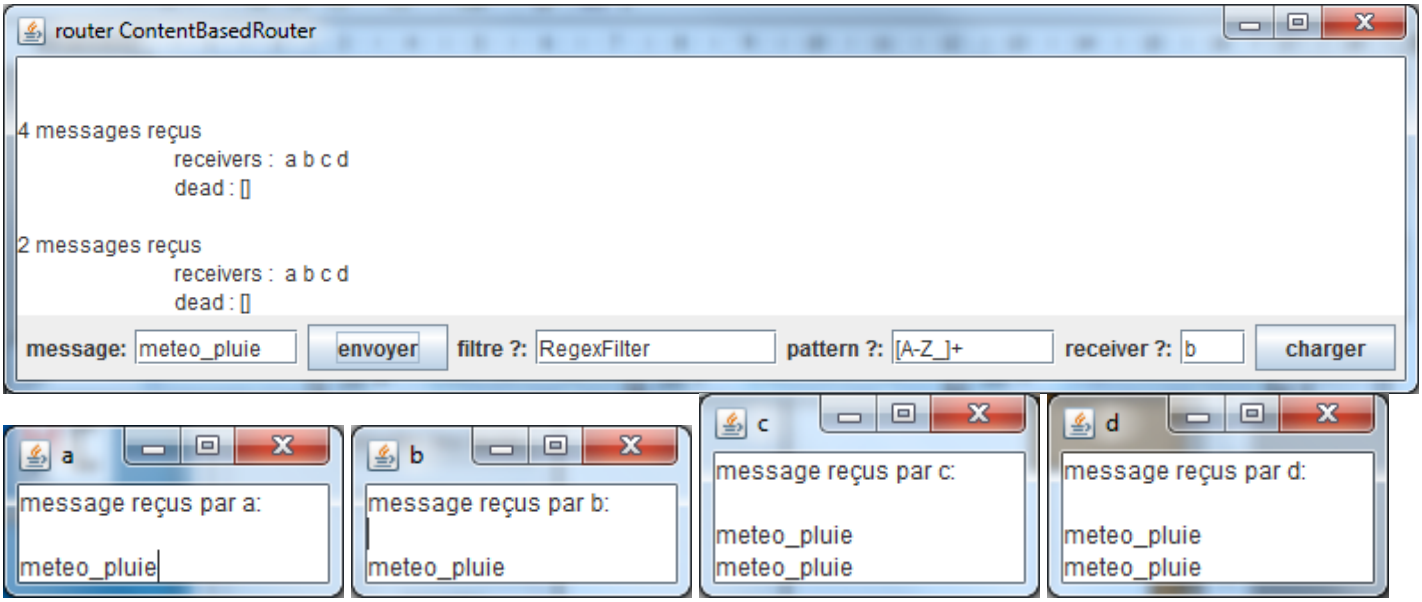
3-1) Complétez cette IHM, en écrivant le comportement des deux boutons envoyer, charger. **Le chargement d'un nouveau filtre pour un receveur se fera uniquement par introspection,** avec l'hypothèse que le constructeur de tout filtre a un seul paramètre de type String.

Un scénario :

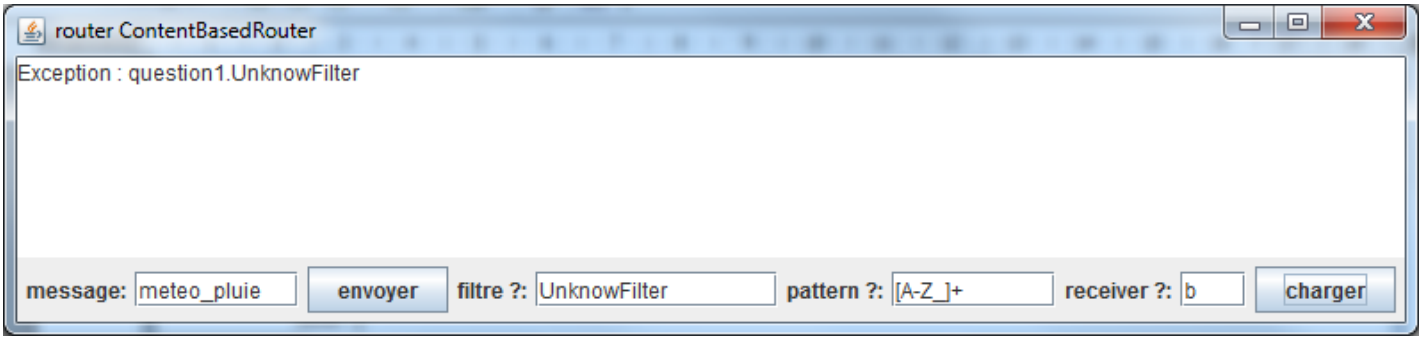
1) Envoi du message *meteo_pluie*, chaque receveur reçoit ce message. Avec préalablement 4 chargements d'une instance du filtre ContainsFilter avec le paramètre meteo pour chaque receveur ont eu lieu



Scénario, suite, 2) Installation, ajout d'un nouveau filtre avec une expression régulière [A-Z_]+ pour les receveurs a et b, à l'envoi du message meteo_pluie, **seuls c et d reçoivent ce message**



Scénario, suite, 3) Tentative d'ajout d'un filtre dont la classe n'existe pas, un message mentionnant l'exception est affichée dans la fenêtre résultante



```
public class IHM {
  private class IHMRouter extends JFrame {

  private JTextArea      resultat = new JTextArea("", 7,60);
  private JTextField     message = new JTextField("meteo_pluie",8);
  private JButton        envoi = new JButton("envoyer");
  private JTextField     filtre = new JTextField("ContainsFilter",12);
  private JTextField     parametre = new JTextField("meteo",8);
  private JTextField     receveur = new JTextField("a",3);
  private JButton        charger = new JButton("charger");

  private  ContentBasedRouterI router;


  public IHMRouter(ContentBasedRouterI router) {
     this.setTitle("router " + router.getClass().getSimpleName());
     this.router = router;
     this.setAlwaysOnTop(true);
     Container container = this.getContentPane();
     container.setLayout(new BorderLayout());
     container.add(resultat, BorderLayout.NORTH);
     JPanel panel = new JPanel();
     panel.setLayout(new FlowLayout());
     panel.add(new JLabel("message:"));
     panel.add(message);
     panel.add(envoi);
     panel.add(new JLabel("filtre ?:"));
     panel.add(filtre);
     panel.add(new JLabel("pattern ?:"));
     panel.add(parametre);
     panel.add(new JLabel("receiver ?:"));
     panel.add(receveur);
     panel.add(charger);

     container.add(panel, BorderLayout.SOUTH);


       this.envoi.addActionListener(
```

à compléter sur votre copie , mentionnez code pour envoi q3

```
        );

        this.charger.addActionListener(
```

à compléter sur votre copie , mentionnez code pour charger q3

```
        );

       this.pack();
     this.setVisible(true);
  }
}

 private static class IHMReceiver extends JFrame {
  private JTextArea  resultat;

  public IHMReceiver(String name) {
      this.setTitle(name);
      Container container = this.getContentPane();
      container.setLayout(new BorderLayout());
      resultat = new JTextArea("message reçus par "+name+":\n", 3,14);
      container.add(resultat, BorderLayout.NORTH);
```

```
        this.pack();
        this.setVisible(true);
    }
    public void println(String str){
      resultat.setText(resultat.getText()+ "\n" + str);
      this.pack();
    }
  }

  private class LocalReceiver extends Receiver{
    private IHMReceiver ihm;

    public LocalReceiver(String name ){
      super(name);
      this.ihm = new IHMReceiver(name);
    }
    public void receive(String msg) throws Exception{
      ihm.println(msg);
    }
    public String toString(){
      return "Receiver: " + getName();
    }


  }
  private  ContentBasedRouterI router;
  private Map<String,LocalReceiver> map;

    public void config() throws Exception{
        this.router = new ContentBasedRouter("router");
        this.map = new TreeMap<>();
        this.map.put("a", new LocalReceiver("a"));
        this.map.put("b", new LocalReceiver("b"));
        this.map.put("c", new LocalReceiver("c"));
        this.map.put("d", new LocalReceiver("d"));

        for(String name : map.keySet()){
          router.addReceiver(map.get(name));
          router.setEnabled(map.get(name));
        }
        new IHMRouter(router);
    }

    public static void main(String[] args) throws Exception{
       new IHM().config();
    }

}
```

3-2) Que pourrait apporter à cette application constituée de cette IHM de tests, l'utilisation d'un outil logiciel permettant l'injection de dépendances, un outil analogue à ***femtoContainer*** vu et utilisé en cours.

Un corrigé sera en ligne cette semaine http://jfod.cnam.fr/NFP121/annales/2018_fevrier/

Annexes

java.util
Interface Collection<E>

All Superinterfaces:
Iterable<E>
All Known Subinterfaces:
…List<E>, …
All Known Implementing Classes:
… ArrayList, …, LinkedList…, Vector

```
public interface Collection<E>
extends Iterable<E>
```

Method Summary	
boolean	**add**(E e) Ensures that this collection contains the specified element (optional operation).
void	**clear**() Removes all of the elements from this collection (optional operation).
boolean	**contains**(Object o) Returns true if this collection contains the specified element.
boolean	**equals**(Object o) Compares the specified object with this collection for equality.
int	**hashCode**() Returns the hash code value for this collection.
boolean	**isEmpty**() Returns true if this collection contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this collection.
boolean	**remove**(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
int	**size**() Returns the number of elements in this collection.

java.util
Interface List<E> ***partielle***

All Superinterfaces:
Collection<E>, Iterable<E>
All Known Implementing Classes:
LinkedList, … ArrayList

```
public interface List<E>
extends Collection<E>
```

Method Summary	
boolean	**add**(E e) Appends the specified element to the end of this list (optional operation).
void	**add**(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
void	**clear**() Removes all of the elements from this list (optional operation).
boolean	**contains**(Object o) Returns true if this list contains the specified element.
boolean	**equals**(Object o) Compares the specified object with this list for equality.
E	**get**(int index) Returns the element at the specified position in this list.
boolean	**isEmpty**()

	Returns `true` if this list contains no elements.
`Iterator<E>`	**`iterator`**`()` Returns an iterator over the elements in this list in proper sequence.
`boolean`	**`remove`**`(Object o)` Removes the first occurrence of the specified element from this list, if it is present (optional operation).
`E`	**`set`**`(int index, E element)` Replaces the element at the specified position in this list with the specified element (optional operation).
`int`	**`size`**`()` Returns the number of elements in this list.

---

java.util
Interface Map<K,V> ***partielle***

Type Parameters:

`K` - the type of keys maintained by this map
`V` - the type of mapped values

All Known Implementing Classes:

HashMap, **Hashtable** **TreeMap**, **WeakHashMap**

`public interface` **`Map<K,V>`**

Method Summary	
`void`	**`clear`**`()` Removes all of the mappings from this map (optional operation).
`boolean`	**`equals`**`(Object o)` Compares the specified object with this map for equality.
`V`	**`get`**`(Object key)` Returns the value to which the specified key is mapped, or `null` if this map contains no mapping for the key.
`int`	**`hashCode`**`()` Returns the hash code value for this map.
`boolean`	**`isEmpty`**`()` Returns `true` if this map contains no key-value mappings.
`Set<K>`	**`keySet`**`()` Returns a `Set` (Set extends Collection) view of the keys contained in this map.
`V`	**`put`**`(K key, V value)` Associates the specified value with the specified key in this map (optional operation).
`V`	**`remove`**`(Object key)` Removes the mapping for a key from this map if it is present (optional operation).
`int`	**`size`**`()` Returns the number of key-value mappings in this map.

---

java.lang
Class String ***partielle***

Method Summary	
`boolean`	**`contains`**`(CharSequence s)` Returns true if and only if this string contains the specified sequence of char values.

```
public class ContentBasedRouter implements ContentBasedRouterI{
```

N° copie :

```
  private   String name;



  public ContentBasedRouter(String name){



  }
  public ContentBasedRouter addReceiver(ReceiverI receiver) {



    return this;
  }
  public ContentBasedRouter addFilter(ReceiverI receiver, FilterI filter){



    return this;
  }
  public ContentBasedRouter removeReceiver(ReceiverI receiver){



    return this;
  }
  public ContentBasedRouter setEnabled(ReceiverI receiver) {



    return this;
  }
  public ContentBasedRouter setDisabled(ReceiverI receiver) {



    return this;
  }
```

N° copie :

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









    return number;
  }



  public Iterator<ReceiverI> iterator(){

      return
  }

  public Set<ReceiverI> getReceiversException(){

    return

  }

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

  public String toString(){
    return
  }
}
```