NFP121

Programmation avancée.

Session de Février 2016-durée : 3 heures

Tous documents papiers autorisés

Cnam / Paris-HTO & FOD/Nationale

Sommaire :

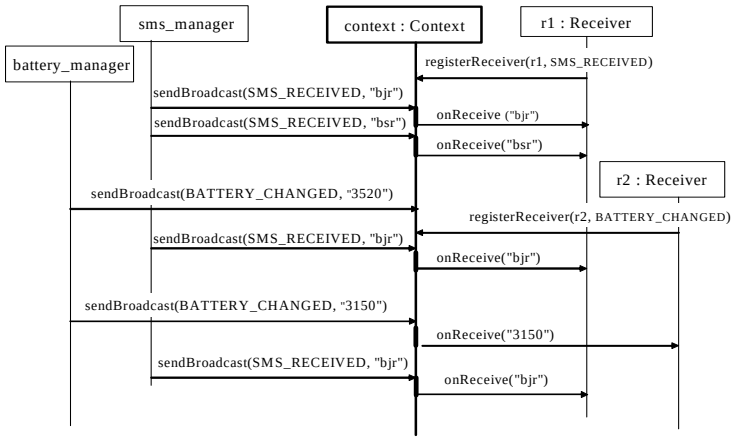
Question1 : Patron *Publish-Subscribe*

Ce patron permet aux abonnés ayant souscrit à un type d'événement, d'être prévenus lorsque ceux-ci se produisent. Un médiateur se charge de l'inscription des abonnés aux différents thèmes de souscription et assure la notification des évènements aux abonnés concernés. La notification peut être effectuée selon la priorité des souscripteurs, l'un des souscripteurs peut arrêter la propagation.

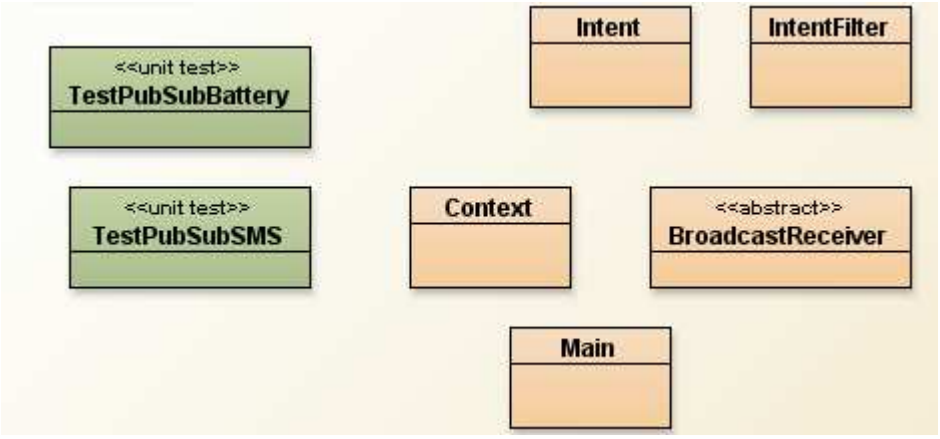
Ce patron est inclus dans Android, qui est un système d'exploitation pour mobile. Les noms de classes, de méthodes et le fonctionnement ont été réutilisés pour cet énoncé.

Le diagramme de séquence ci-dessous présente un scénario possible dans lequel interviennent:

- Une notification d'évènements « **SMS_RECEIVED** » et « **ACTION_BATTERY_CHANGED**» ,
- Un médiateur de gestion des souscriptions et de notifications (*context : Context*),
- Un souscripteur r1 au thème « **SMS_RECEIVED** »,
- Un souscripteur r2 au thème **« ACTION_BATTERY_CHANGED**».



L'architecture retenue des classes pour ce patron est la suivante (notation BlueJ/UML) :



La classe ***Context*** contient les opérations

- d'abonnement d'un souscripteur (**`registerReceiver`**) pour un thème de publication,
  - Plusieurs thèmes de souscription par receveur sont possibles,
- de retrait d'un souscripteur (**`unregisterReceiver`**) ,
- de publication (**`sendBroadcast`**) aux souscripteurs concernés,
  - Tous les souscripteurs reçoivent une notification,
- de publication selon la priorité des souscripteurs (**`sendOrderedBroadcast`**),
  - Un arrêt de la publication peut être décidé par l'un des souscripteurs.

```
public class Context{

/** Enregistrement d'un souscripteur.
 *  @param receiver le souscripteur
 *  @param filter le ou les thèmes de souscription
 */
public void registerReceiver(BroadcastReceiver receiver, IntentFilter filter){...}


/** Retrait d'un souscripteur.
 *  @param receiver le souscripteur à retirer
 */
public void unregisterReceiver(BroadcastReceiver receiver){...}


/** Publication sur ce thème.
 *  Les souscripteurs abonnés à ce thème sont tous notifiés.
 *  Note : l'appel de la méthode abortBroadcast par un souscripteur
 *         est sans effet, cf. sendOrderedBroadcast
 *  @param intent le thème de publication
 */
public void sendBroadcast(Intent intent){...}

/** Publication ordonnée sur ce thème.
 *  Les souscripteurs sont notifiés selon leur priorité.
 *  Un souscripteur notifié peut interrompre la notification par
 *  l'appel de la méthode abortBroadcast( cf. classe BroadcastReceiver).
 *  @param intent le thème de publication
 */
public void sendOrderedBroadcast(Intent intent){...}

}
```

La classe ***Intent*** correspond à la notification, l'action transmise précise son type, la classe est complète.

```
public class Intent{
  public static final String ACTION_BATTERY_CHANGED    = "ACTION_BATTERY_CHANGED";
  public static final String BATTERY_STATUS_FULL       = "BATTERY_STATUS_FULL";
  public static final String ACTION_BATTERY_LOW        = "ACTION_BATTERY_LOW";
  public static final String ACTION_POWER_CONNECTED    = "ACTION_POWER_CONNECTED";
  public static final String ACTION_POWER_DISCONNECTED = "ACTION_POWER_DISCONNECTED";
  public static final String SMS_RECEIVED              = "SMS_RECEIVED";

  private Map<String,Object> extras;
  private String action;

  public Intent(){
    this.extras = new HashMap<String,Object>();
    this.action = null;
  }

  public Intent setAction(String action){
    this.action = action;
    return this;
  }

  public String getAction(){return action;}

  public Intent putExtra(String key, Object value){
    this.extras.put(key, value);
    return this;
  }

  public Map<String,Object> getExtras(){return this.extras;}
  public Object getExtra(String key){ return this.extras.get(key); }
  public String toString(){
    return "<"+action + "," + extras + ">";
  }
}
```

La classe abstraite ***BroadcastReceiver*** implémente les méthodes permettant un abonnement à un ou plusieurs thèmes de publication (Ces thèmes sont précisés dans une instance de la classe ***IntentFilter***). Cette classe abstraite laisse à ses sous classes la responsabilité d'implémenter la réception et le traitement d'une notification (méthode ***onReceive*****)**.

```
public abstract class BroadcastReceiver{
  private boolean abort;
  private IntentFilter filter;

  public void setIntentFilter(IntentFilter filter){
    this.filter = filter;
  }
  public IntentFilter getIntentFilter(){
    return this.filter;
  }
  public final void abortBroadcast(){
    this.abort = true;
  }
  public final boolean getAbortBroadcast(){
    boolean res = this.abort;
    if(this.abort)this.abort=false;
    return res;
  }
  public abstract void onReceive(Context context, Intent intent);
}
```

La classe ***IntentFilter*** précise les thèmes de souscription et la priorité associée

```
public class IntentFilter{
```

```
  public static final int MAX_PRIORITY = 1000;
  public static final int MIN_PRIORITY = 0;

  public IntentFilter(){...}

  public Iterator<String> actionsIterator(){...}

  public void addAction(String action){...}

  public void setPriority(int priority){ {...}
  public int getPriority(){...}
}
```

Ci-dessous, le source des deux classes de tests unitaires, ***TestPubSubBattery*** et ***TestPubSubSMS***, Ces deux classes sont **à lire attentivement** avant de répondre à la question.

```
public class TestPubSubBattery extends junit.framework.TestCase{
  public static final int BATTERY_MIN_LEVEL = 2000;
  public static class Receiver extends BroadcastReceiver{
    private boolean notified; // pour les tests : verification

   public void onReceive(Context context, Intent intent){
      notified =true;
      Integer level = (Integer)intent.getExtra("level");
      if(level!=null && level<=BATTERY_MIN_LEVEL){
        abortBroadcast(); // rappel: pris en compte uniquement
                          // lors de sendOrderedBroadcast
        System.out.println("abortBroadcast !!");
      }
      System.out.println("onReceive: " + intent);
    }
    public boolean notified(){  // test de la notification
      boolean temp = notified;
      notified = false;
      return temp;
    }
  }
  private Context context;
  private Receiver r1, r2, r3;

  protected void setUp(){
        this.context = new Context();

        IntentFilter filter1 = new IntentFilter();
        filter1.addAction(Intent.ACTION_BATTERY_CHANGED);
        filter1.addAction(Intent.ACTION_BATTERY_LOW);
        filter1.setPriority(IntentFilter.MAX_PRIORITY-1);
        this.r1 = new Receiver();r1.setIntentFilter(filter1);
        context.registerReceiver(r1, filter1);

        IntentFilter filter2 = new IntentFilter();
        filter2.addAction(Intent.ACTION_POWER_CONNECTED);
        filter2.addAction(Intent.ACTION_POWER_DISCONNECTED);
        filter2.addAction(Intent.ACTION_BATTERY_CHANGED);
        filter2.setPriority(IntentFilter.MAX_PRIORITY);
        this.r2 = new Receiver();r2.setIntentFilter(filter2);
        context.registerReceiver(r2, filter2);

        IntentFilter filter3 = new IntentFilter();
        filter3.addAction(Intent.ACTION_BATTERY_CHANGED);
        filter3.addAction(Intent.ACTION_BATTERY_LOW);
        filter3.setPriority(IntentFilter.MIN_PRIORITY);
        this.r3 = new Receiver();r3.setIntentFilter(filter3);
        context.registerReceiver(r3, filter3);
  }
```

```
        public void test_sendBroadcast(){
          Intent batteryIntent = new Intent();
          batteryIntent.setAction(Intent.ACTION_BATTERY_CHANGED);
          batteryIntent.putExtra("level", BATTERY_MIN_LEVEL-500);
          context.sendBroadcast(batteryIntent);

          assertTrue("r1 non notifié ? ", r1.notified());
          assertTrue("r2 non notifié ? ", r2.notified());
          assertTrue("r3 non notifié ? ", r3.notified());
        }

        public void test_sendOrderedBroadcast(){

          batteryIntent = new Intent();
          batteryIntent.setAction(Intent.ACTION_BATTERY_CHANGED);
          batteryIntent.putExtra("level",4120);
          context.sendOrderedBroadcast(batteryIntent);
          assertTrue("r1 non notifié ? ", r1.notified());
          assertTrue("r2 non notifié ? ", r2.notified());
          assertTrue("r3 non notifié ? ", r3.notified());

          batteryIntent = new Intent();
          batteryIntent.setAction(Intent.ACTION_BATTERY_CHANGED);
          batteryIntent.putExtra("level", BATTERY_MIN_LEVEL-500);
          context.sendOrderedBroadcast(batteryIntent);
          assertTrue("r2 non notifié ? ", r2.notified());
          assertFalse("r1 est-il notifié ?, curieux... ", r1.notified());
          assertFalse("r3 est-il notifié ?, curieux... ", r3.notified());
        }
}

public class TestPubSubSMS extends junit.framework.TestCase{

  public static class Receiver extends BroadcastReceiver{
    private boolean notified; // pour les tests : verification

   public void onReceive(Context context, Intent intent){
      notified =true;
      String body = (String)intent.getExtra("sms_body");
      if(body!=null && body.startsWith("abort")){
        abortBroadcast(); // pris en compte uniquement
                          // lors de sendOrderedBroadcast
      }
      System.out.println("onReceive: " + intent);
    }
    public boolean notified(){  // test de la notification
      boolean temp = notified;
      notified = false;
      return temp;
    }
  }
  private Context context;
  private Receiver r1, r2, r3;

  protected void setUp(){
       this.context = new Context();

       IntentFilter filter1 = new IntentFilter();
       filter1.addAction(Intent.SMS_RECEIVED);
       filter1.setPriority(IntentFilter.MAX_PRIORITY-1);
       this.r1 = new Receiver();r1.setIntentFilter(filter1);
       context.registerReceiver(r1, filter1);

       IntentFilter filter2 = new IntentFilter();
       filter2.addAction(Intent.SMS_RECEIVED);
       filter2.setPriority(IntentFilter.MAX_PRIORITY);
       this.r2 = new Receiver();r2.setIntentFilter(filter2);
       context.registerReceiver(r2, filter2);
```

```
        IntentFilter filter3 = new IntentFilter();
        filter3.addAction(Intent.SMS_RECEIVED);
        filter3.setPriority(IntentFilter.MIN_PRIORITY);
        this.r3 = new Receiver();r3.setIntentFilter(filter3);
        context.registerReceiver(r3, filter3);
    }

    public void test_sendBroadcast(){
        Intent smsIntent = new Intent();
        smsIntent.setAction(Intent.SMS_RECEIVED);
        smsIntent.putExtra("sms_body","bjr");
        context.sendBroadcast(smsIntent);
        assertTrue("r1 non notifié ??? ", r1.notified());
        assertTrue("r2 non notifié ??? ", r2.notified());
        assertTrue("r3 non notifié ??? ", r3.notified());
    }

    public void test_sendOrderedBroadcast(){
        Intent smsIntent = new Intent();
        smsIntent.setAction(Intent.SMS_RECEIVED);
        smsIntent.putExtra("sms_body","bjr");
        context.sendOrderedBroadcast(smsIntent);
        assertTrue("r1 non notifié ??? ", r1.notified());
        assertTrue("r2 non notifié ??? ", r2.notified());
        assertTrue("r3 non notifié ??? ", r3.notified());

        smsIntent.putExtra("sms_body","abort");
        context.sendOrderedBroadcast(smsIntent);
        assertTrue("r2 non notifié ??? ", r2.notified());
        assertFalse("r1 est-il notifié ?, curieux... ", r1.notified());
        assertFalse("r3 est-il notifié ?, curieux... ", r3.notified());
    }
}
```

Question1-1)

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

Question1-2)

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

Notes :

- Vous pouvez joindre les dernières pages de cet énoncé à votre copie, sans oublier de reporter le numéro de celle-ci.
- Plusieurs descriptions d'interfaces extraites du paquetage java.util sont en annexe.

Question2: Une interface graphique

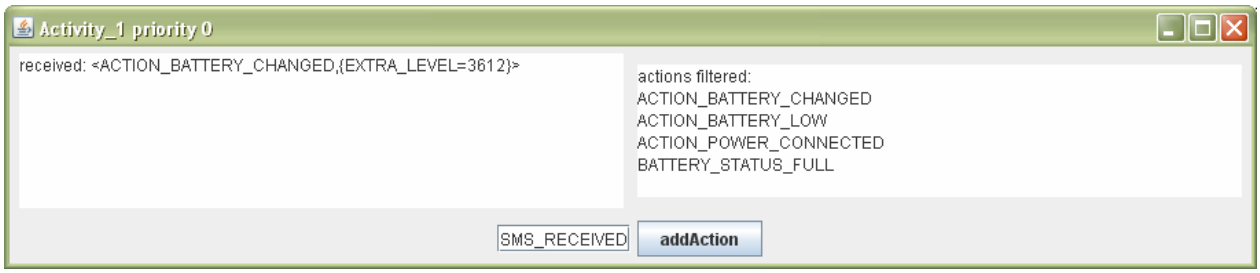
Une interface graphique en Swing a été développée dans le but de "tester" une instance de la classe *Context*, ainsi que la génération de notifications et de souscriptions, sur des thèmes indiqués par l'utilisateur/testeur.

Les envois de notifications sont effectués par deux interfaces, associées aux services *BatteryManager et SMSManager* les envois s'effectuent en cliquant sur le bouton correspondant. Au clic, un message contenant le type d'envoi, l'*intent* émise et ses paramètres sont affichés.

Emission d'une « *intent* » (ACTION_BATTERY_CHANGED) par l'appel de **sendBroadcast**

Emission d'une « *intent* » (SMS_RECEIVED) par l'appel de **sendBroadcast**

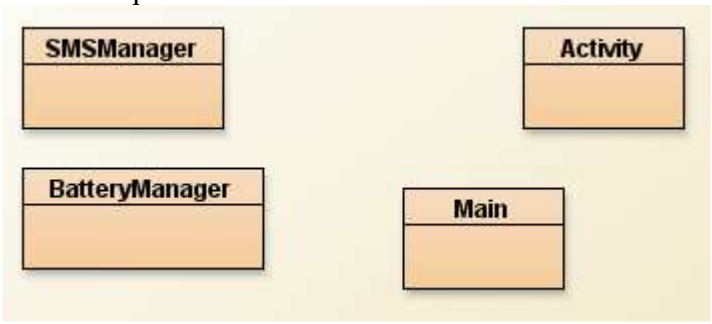
Les notifications reçues apparaissent dans une interface utilisateur, issue de la classe *Activity*, celle-ci se contente d'afficher le type et le contenu des messages reçus, une zone de texte mentionne toutes les actions auxquelles est abonnée cette activité. Au clic sur ***addAction*** un nouveau thème d'abonnement est ajouté, la liste est réactualisée.



Une première notification ACTION_BATTERY_CHANGED a eu lieu.

Après avoir ajouté la souscription aux sms reçus (SMS_RECEIVED), la fenêtre des résultats mentionne le contenu du texto, envoyé par l'interface d'émission.

Architecture des classes pour cette question



Les classes *SMSManager* et *BatteryManager* sont des interfaces graphiques permettant l'émission de notifications via le contexte transmis à la création de ces deux services. La classe *Activity* permet de souscrire à des thèmes d'abonnement et reçoit en conséquence, via le contexte, les *intents* concernés.

Ci-dessous la classe Main, installant les interfaces et autorisant ces tests, au total deux services d'émission et cinq souscripteurs :

```
public class Main{
  public static void main(String[] args){
    // Création du contexte
    Context context = new Context();
    // Création des services
    SMSManager serviceSMS = new SMSManager(context);
    BatteryManager serviceBattery = new BatteryManager(context);

    // Souscription et creation d'une Activité (activity1)
    IntentFilter filter1 = new IntentFilter();
    filter1.addAction(Intent.ACTION_BATTERY_CHANGED);
    filter1.addAction(Intent.ACTION_BATTERY_LOW);
    filter1.addAction(Intent.ACTION_POWER_CONNECTED);
    filter1.addAction(Intent.BATTERY_STATUS_FULL);
    Activity activity1 = new Activity(context, "Activity_1", filter1);

    // Souscription et creation d'une Activité (activity2)
    IntentFilter filter2 = new IntentFilter();
    filter2.addAction(Intent.ACTION_BATTERY_LOW);
    Activity activity2 = new Activity(context, "Activity_2", filter2,
                                      IntentFilter.MAX_PRIORITY-100);

    // Souscription et creation d'une Activité (activity3)
    IntentFilter filter3 = new IntentFilter();
    filter3.addAction(Intent.ACTION_BATTERY_LOW);
    Activity activity3 = new Activity(context, "Activity_3", filter3,
                                      IntentFilter.MAX_PRIORITY);

    // Souscription et creation d'une Activité (activity4)
    IntentFilter filter4 = new IntentFilter();
    filter4.addAction(Intent.SMS_RECEIVED);
    Activity activity4 = new Activity(context, "Activity_4", filter4, 100);

    // Souscription et creation d'une Activité (activity5)
    IntentFilter filter5 = new IntentFilter();
    filter5.addAction(Intent.SMS_RECEIVED);
    Activity activity5 = new Activity(context, "Activity_5", filter5, 200);
  }
}
```

La classe Activity est à compléter : son exécution présente une interface graphique, et se contente d'afficher le type et le contenu des messages (instances de la classe *Intent*) reçus. Son interface autorise l'ajout par l'utilisateur de nouveaux thèmes de souscription. Ces ajouts sont mentionnés dans l'interface. Lors d'une réception de sms, si le contenu du message (paramètre *sms_body*) commence par « abort », la notification n'est pas propagée.

```
public class Activity extends JFrame{

    private JTextField   action;
    private JButton      add;
    private JTextArea    resultat, actionsFiltered;
    private JCheckBox    abort;

    private Context  context;
    private Receiver receiver;
```

```
    private class Receiver extends BroadcastReceiver{

      public void onReceive(Context context, Intent intent){
        resultat.append("received: " + intent.toString()+"\n");
        String body = (String)intent.getExtra("sms_body");
        if(body!=null && body.startsWith("abort")){
          abortBroadcast();
        }
      }
      public String toString(){
        return "Receiver: priority: " + getIntentFilter().getPriority();
      }
    }

    public Activity(Context context, String name, IntentFilter filter){
      this(context, name, filter, 0);
    }

    public Activity(Context context, String name, IntentFilter filter, int priority){
      super(name + " priority " + priority);
      this.receiver = new Receiver();
      this.context = context;
      this.context.registerReceiver(receiver, filter);
      this.receiver.getIntentFilter().setPriority(priority);

      this.resultat = new JTextArea("", 7,40);
      this.resultat.setLineWrap(true);
      this.actionsFiltered = new JTextArea("", 4,40);
      this.actionsFiltered.setLineWrap(true);
      Container container = this.getContentPane();
      container.setLayout(new BorderLayout());
      JPanel panel = new JPanel();
      panel.setLayout(new FlowLayout());
      panel.add(this.resultat);
      panel.add(new JSeparator(SwingConstants.VERTICAL));
      panel.add(this.actionsFiltered);
      container.add(panel, BorderLayout.NORTH);

      JPanel panelButtons = new JPanel();
      this.add = new JButton("addAction");
      this.action = new JTextField(Intent.SMS_RECEIVED);
      panelButtons.add(action);
      panelButtons.add(add);
      container.add(panelButtons, BorderLayout.SOUTH);

       printActionsFiltered(receiver.getIntentFilter());

       this.add.addActionListener(new ActionListener(){
         public void actionPerformed(ActionEvent ae){
```

à compléter, sur votre copie, « indiquez code pour add, question2»

```
         }});

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.pack();
      this.setVisible(true);
    }


   private void printActionsFiltered (…
```

à compléter, sur votre copie, « indiquez code pour printActionFiltered, question2»

```
 }
```

Question 2)

Compléter **l'implémentation de la classe *Activity*** afin de produire le fonctionnement décrit ci-dessus.

Question3: Configuration des tests en XML

Nous souhaitons **remplacer complètement la classe Main** de la question 2, soit une configuration en XML, suivie d'une interprétation des balises générant une exécution équivalente.

Exemple : la balise activity ci-dessous

```
<activity name="Activity_2" class="question2.Activity">
   <intent-filter priority="900">
           <action name="ACTION_BATTERY_LOW" />
   </intent-filter>
</activity>
```

engendre après la lecture, le même comportement que l'exécution de ces instructions :

```
// Souscription et creation d'une Activité (activity2)  cf. Classe Main (Question 2)
IntentFilter filter2 = new IntentFilter();
filter2.addAction(Intent.ACTION_BATTERY_LOW);
Activity activity2 = new Activity(context, "Activity_2", filter2,
                                  IntentFilter.MAX_PRIORITY-100);
```

Soit le fichier XML de configuration:

```
<application>
    <service name="batteryManager" class="question2.BatteryManager" />
    <service name="smsManager" class="question2.SMSManager" />

    <activity name="Activity_1" class="question2.Activity">
           <intent-filter>
                  <action name="ACTION_BATTERY_CHANGED" />
           </intent-filter>
           <intent-filter>
                  <action name="ACTION_BATTERY_LOW" />
           </intent-filter>
           <intent-filter>
                  <action name="ACTION_POWER_CONNECTED" />
           </intent-filter>
           <intent-filter>
                  <action name="BATTERY_STATUS_FULL" />
           </intent-filter>
    </activity>
    <activity name="Activity_2" class="question2.Activity">
           <intent-filter priority="900">
                   <action name="ACTION_BATTERY_LOW" />
           </intent-filter>
    </activity>
    <activity name="Activity_3" class="question2.Activity">
           <intent-filter priority="1000">
                  <action name="ACTION_BATTERY_LOW" />
           </intent-filter>
    </activity>
    <activity name="Activity_4" class="question2.Activity">
           <intent-filter priority="100">
                  <action name="SMS_RECEIVED" />
           </intent-filter>
    </activity>
    <activity name="Activity_5" class="question2.Activity">
           <intent-filter priority="200">
                  <action name="SMS_RECEIVED" />
           </intent-filter>
    </activity>
</application>
```

Question 3)

Proposer une classe Configuration dont l'appel du constructeur permet la lecture d'un fichier en XML et qui engendre l'exécution des instructions correspondantes.

Si vous utilisez SAX pour votre analyse du fichier XML, un squelette de programme à compléter est en annexe.

Question4 : Question de cours

Injection de dépendances : Rappelez le principe et les bénéfices escomptés. Que permettrait l'usage de ce principe à la question 1 de cet examen, proposez un schéma d'architecture logicielle.

Fin

Un corrigé sera en ligne demain http://jfod.cnam.fr/NFP121/annales/2016_fevrier/

Annexes

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.
int	**hashCode**() Returns the hash code value for 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 Subinterfaces:
SortedMap<K,V>
All Known Implementing Classes:
HashMap, **Hashtable** **TreeMap**, **WeakHashMap**

---

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

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value.

Method Summary	
void	**clear**() Removes all of the mappings from this map (optional operation).
boolean	**containsKey**(Object key) Returns true if this map contains a mapping for the specified key.
boolean	**containsValue**(Object value) Returns true if this map maps one or more keys to the specified value.
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.util
Class ArrayList<E>

public class **ArrayList<E>**

extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable

Constructor Summary

ArrayList() Constructs an empty list with an initial capacity of ten.
ArrayList(Collection<? extends E> c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity) Constructs an empty list with the specified initial capacity.

Method Summary

boolean	**add**(E e) Appends the specified element to the end of this list.
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.

java.util
Interface Iterator<E>

public interface **Iterator<E>**

Method Summary	
boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

java.util
Class Collections

java.lang.Object
  └ **java.util.Collections**

Method Summary

static <T> void	**sort**(List<T> list, Comparator<? super T> c) Sorts the specified list according to the order induced by the specified

	comparator.

---

java.util

Interface Comparator<T>

Type Parameters:

`T` - the type of objects that may be compared by this comparator

All Known Implementing Classes:

Collator, RuleBasedCollator

```
public interface Comparator<T>
```

Method Summary

`int`	**`compare`**`(T o1, T o2)` Compares its two arguments for order. Return a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.

```
public class Configuration {

  public Configuration() throws SAXException, IOException, Exception{
    this("question3/README.TXT");
  }

  public Configuration(String fileName) throws SAXException, IOException, Exception{



    this.parse(fileName);


    Context context = new Context();



  }







  private void parse(String fileName) throws SAXException, IOException{
    XMLReader saxReader = XMLReaderFactory.createXMLReader();
    saxReader.setContentHandler(new DefaultHandler(){

    @Override
    public void startElement(String uri, String name, String qualif, Attributes at)
throws SAXException{

      if(name.equals("service")){


        for( int i = 0; i < at.getLength(); i++ ){
          if(at.getLocalName(i).equals("name"))



          if(at.getLocalName(i).equals("class"))



        }

      }


      if(name.equals("activity")){


        for( int i = 0; i < at.getLength(); i++ ){
          if(at.getLocalName(i).equals("name"))



          if(at.getLocalName(i).equals("class"))



        }
```

```
      }



      if(name.equals("intent-filter")){

        for( int i = 0; i < at.getLength(); i++ ){
          if(at.getLocalName(i).equals("priority"))


        }


      }

      if(name.equals("action")){
        for( int i = 0; i < at.getLength(); i++ ){
          if(at.getLocalName(i).equals("name"))







        }
      }
    }

    @Override
    public void endElement(String uri, String localName, String qName) throws
SAXException{


    }


  });

    saxReader.parse(fileName);
  }

}
```

```
public class IntentFilter{
  public static final int MAX_PRIORITY = 1000;
  public static final int MIN_PRIORITY = 0;

  public IntentFilter(){




  }
  public Iterator<String> actionsIterator(){









  }
  public void addAction(String action){









  }
  public void setPriority(int priority){









  }
  public int getPriority(){




  }
}
```

```
public class Context{
  public Context(){

  }
  public void sendBroadcast(Intent intent){









  }




  public void sendOrderedBroadcast(Intent intent){









  }
  public void registerReceiver(BroadcastReceiver receiver, IntentFilter filter){









  }
  public void unregisterReceiver(BroadcastReceiver receiver){



  }
}
```