NFP121

Programmation avancée.

Session de Février 2020-durée : 3 heures

Tous documents papiers autorisés

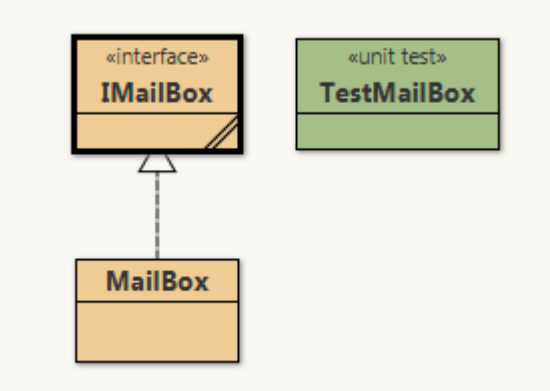
Cnam / Paris-HTO & FOD/Nationale

Sommaire :

Question1 : Patron *MailBox*

L' *E-mail* est un exemple classique de l'usage de ce patron. Ce patron permet de délivrer des messages, de les mémoriser, jusqu'à ce que leur(s) destinataire(s) les relève.

Ci-dessous, en syntaxe UML/BlueJ, une architecture possible de classes Java reflétant ce Patron :



Cette architecture, simplifiée, est constituée de l'interface ***IMailBox***, *(dont les commentaires (javadoc) sont à lire attentivement)*, d'une implémentation ***MailBox***, (*ce qui est demandé)* et d'une classe de tests unitaires ***TestMailBox*** fournie (*à lire tout aussi attentivement…).* Les messages sont de type *String*.

L'interface ***IMailBox*** ci-dessous**,** propose les opérations :

- d'inscription d'un destinataire de messages (**`add`**),
- d'envoi d'un message à un seul destinataire (**`send`**),
- d'envoi d'un message à plusieurs destinataires (**`send`**),
- de relevé les nouveaux messages pour un destinataire (**`read`**),
- d'obtention de tous les messages pour un destinataire (**`getAllMessages`**),
- d'obtention de tous les messages non lus pour un destinataire (**`getAllUnreadMessages`**),
- d'obtention d'un itérateur sur la liste des destinataires inscrits (**`iterator`**).

```
public interface IMailBox extends Iterable<String>, Serializable{

   /** Inscription d'un destinataire.
    * L'inscription est obligatoire afin de recevoir des messages.
    * Attention Pas d'homonyme possible.
    * @param dest le destinataire le nom de l'utilisateur
    * @throws Exception si le destinataire est déjà inscrit ou un homonyme est en place
    */
   public void add(String dest) throws Exception;
```

```
    /** Envoi d'un message pour un destinataire.
     * @param message le message envoyé.
     * @param dest le destinataire du message
     * @throws Exception si le destinataire est inconnu
     */
    public void send(String message, String dest) throws Exception;

    /** Envoi d'un message pour plusieurs destinataires.
     * @param message le message envoyé.
     * @param dest les destinataires du message.
     * @param message le message envoyé.
     * @throws Exception si un des destinataires (au moins) est inconnu.
     */
    public void send(String message, String[] dest) throws Exception;

    /** Relevé des messages pour ce destinataire.
     * @param dest le nom du destinataire.
     * @return la liste contenant les nouveaux messages, depuis le dernier relevé.
     * @throws Exception si le nom du destinataire est inconnu
     */
    public List<String> read(String dest) throws Exception;

    /** Obtention de la liste des destinataires.
     * @return un itérateur, parcours des inscrits
     */
    public Iterator<String> iterator();

    /** Relevé de tous les messages pour ce destinataire.
     * @param dest le nom du destinataire.
     * @return la liste contenant tous les messages, lus ou non.
     * @throws Exception si le nom du destinataire est inconnu
     */
    public List<String> getAllMessages(String dest) throws Exception;

     /** Relevé de tous les messages non lus pour ce destinataire.
     * @param dest le nom du destinataire.
     * @return la liste contenant tous les messages non lus.
     * @throws Exception si le nom du destinataire est inconnu
     */
    public List<String> getAllUnreadMessages(String dest) throws Exception;
}
```

Ci-dessous, un extrait de la classe ***TestMailBox*** : une classe de tests unitaires, à lire attentivement avant de répondre à cette question.

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

  public void testScenario1() throws Exception{
    IMailBox mb = new MailBox();
    mb.add("a");mb.add("b");mb.add("c");mb.add("d"); // 4 destinataires : a,b,c,d

    String msg1 = new String("src1_msg1"); //
    mb.send(msg1,"a");                              // envoi d'un message pour "a"
    List<String> list = mb.read("a");
    assertEquals(1,list.size());                    // "a" a reçu 1 message
    assertEquals("src1_msg1",list.get(0));          // "a" a reçu "src1_msg1"
    list = mb.read("a");
    assertEquals(0,list.size());                    // "a" n'a plus de message à lire

    String msg2 = new String("src2_mail_2");
    mb.send(msg2,"a");
    list = mb.read("a");

    String msg3 = new String("src3_mail_3");
    mb.send(msg3,new String[]{"a","b","c","d"});    // un message pour a,b,c,d
```

```
    String msg4 = new String("src4_mail_4");
    mb.send(msg4,new String[]{"a","b","c"});       // un message pour a,b,c

    list = mb.read("b");
    assertEquals(2,list.size());                   // 2 messages lus pour b
    assertEquals("src3_mail_3",list.get(0));       // "b" a reçu "src3_msg3"
    assertEquals("src4_mail_4",list.get(1));       // "b" a reçu "src4_msg4"

    list = mb.getAllMessages("a");
    assertEquals(4,list.size());            // au total 4 messages reçus pour "a"
    list = mb.getAllUnreadMessages("a");
    assertEquals(2,list.size());            // 2 messages non lus pour "a"
  }

  public void testScenario2_Cas_Exception() throws Exception{
    IMailBox mb = new MailBox();
    mb.add("a");mb.add("b");mb.add("c");mb.add("d"); // 4 destinataires : a,b,c,d

    String msg1 = new String("src1_mail_1");
    try{
      mb.send(msg1,"x");              // "x" est inconnu, alors une exception est attendue
      fail("une exception destinataire inconnu est attendue !");
    }catch(Exception e){
      try{
        mb.send(msg1,new String[]{"a","b","x","c"});
        fail(" x est inconnu,  une exception doit être levée");
      }catch(Exception e1){
        assertEquals("src1_mail_1",mb.read("a").get(0)); // "a","b","c" ont reçu
        assertEquals("src1_mail_1",mb.read("b").get(0));
        assertEquals("src1_mail_1",mb.read("c").get(0));
      }
    }
  }

  public void testScenario3_Equals() throws Exception{
    MailBox mb = new MailBox();
    mb.add("a");mb.add("b");mb.add("c");mb.add("d"); // 4 destinataires : a,b,c,d
    String msg1 = new String("src1_msg1"); // source : src1, message msg1
    mb.send(msg1,"a");
    List<String> list = mb.read("a");
    list = mb.read("a");

    IMailBox mbBis = new MailBox(mb);  // une copie de la boite aux lettres est possible
    assertEquals(mb,mbBis);            // elles sont (doivent être) égales
  }
}
```

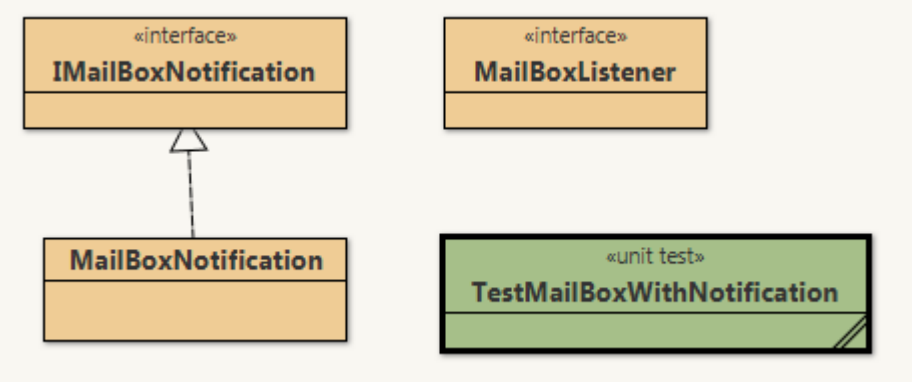
Question)

Ecrivez une implémentation **complète de la classe *MailBox***. Certaines documentations d'interfaces et de classes sont en annexe. Vous pouvez détacher la dernière page de cet énoncé, la joindre à votre copie en n'oubliant pas de reporter le numéro de celle-ci.

Question2 : Patron Observateur

Un destinataire préalablement inscrit souhaite maintenant être notifié dès qu'un message à son attention est déposé dans la boîte aux lettres.

L'architecture de la question précédente s'est enrichie de deux nouvelles interfaces *MailBoxListener* et *IMailBoxNotification*, soit en syntaxe UML/BlueJ:



L'interface ***IMailBoxNotification***, propose une méthode permettant d'inscrire les clients en vue d'une notification.

```
public interface IMailBoxNotification extends IMailBox{

  /** Inscription d'un observateur(Listener) auprès d'un destinataire.
   * Rappel: pas d'homonyme possible.
   * Plusieurs observateurs sont possibles par destinataire.
   * @param listener l'observateur, est notifié à chaque réception de courrier
   * @param dest  le nom de l'utilisateur
   * @throws Exception si le destinataire n'est pas inscrit
   */
  public void addMailBoxListener(String dest, MailBoxListener listener) throws
Exception;
}
```

L'interface ***MailBoxListener,*** implémentée par les destinataires souhaitant être notifiés, est donnée ci-dessous, cette interface contient la méthode ***onResceive***, exécutée à chaque réception d'un message :

```
public interface MailBoxListener{

  public void onReceive(String msg) throws Exception;

}
```

Ci-dessous un extrait de la classe **TestMailBoxWithNotification** : une classe de tests unitaires (*à lire attentivement avant de répondre à cette question).*

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

  public class TestMailBoxListener implements MailBoxListener{
    private List<String> notifications = new ArrayList<>();

    public void onReceive(String msg) throws Exception{
      notifications.add(msg); // à chaque notification, le message est ajouté à la liste
    }
    public List<String> getNotifications(){// accesseur pour la liste notifications
      return notifications;
    }
  }

  public void testScenario1() throws Exception{
    IMailBoxNotification mb = new MailBoxNotification();
    TestMailBoxListener listenerA = new TestMailBoxListener();
```

```
    mb.add("a");
    mb.addMailBoxListener("a", listenerA);  // un destinataire : "a"
    String msg = new String("src_mail_1");
    mb.send(msg,"a");
    assertEquals("src_mail_1",listenerA.getNotifications().get(0));// notification effective
  }

  public void testScenario2()  throws Exception{
    IMailBoxNotification mb = new MailBoxNotification();
    mb.add("a");mb.add("b");mb.add("c");mb.add("d");

    TestMailBoxListener listenerA = new TestMailBoxListener();
    mb.addMailBoxListener("a", listenerA);

    TestMailBoxListener listenerB = new TestMailBoxListener();
    mb.addMailBoxListener("b", listenerB);

    TestMailBoxListener listenerC1 = new TestMailBoxListener();
    mb.addMailBoxListener("c", listenerC1);

    TestMailBoxListener listenerC2 = new TestMailBoxListener();
    mb.addMailBoxListener("c", listenerC2);  // "c" possède une second "écouteur"

    String msg = new String("src_mail_1");
    mb.send(msg,new String[]{"a","b","c"});
                                                    // Tous les écouteurs ont été notifiés
    assertEquals("src_mail_1",listenerA.getNotifications().get(0));
    assertEquals("src_mail_1",listenerB.getNotifications().get(0));
    assertEquals("src_mail_1",listenerC1.getNotifications().get(0));
    assertEquals("src_mail_1",listenerC2.getNotifications().get(0));
  }
```

Question)

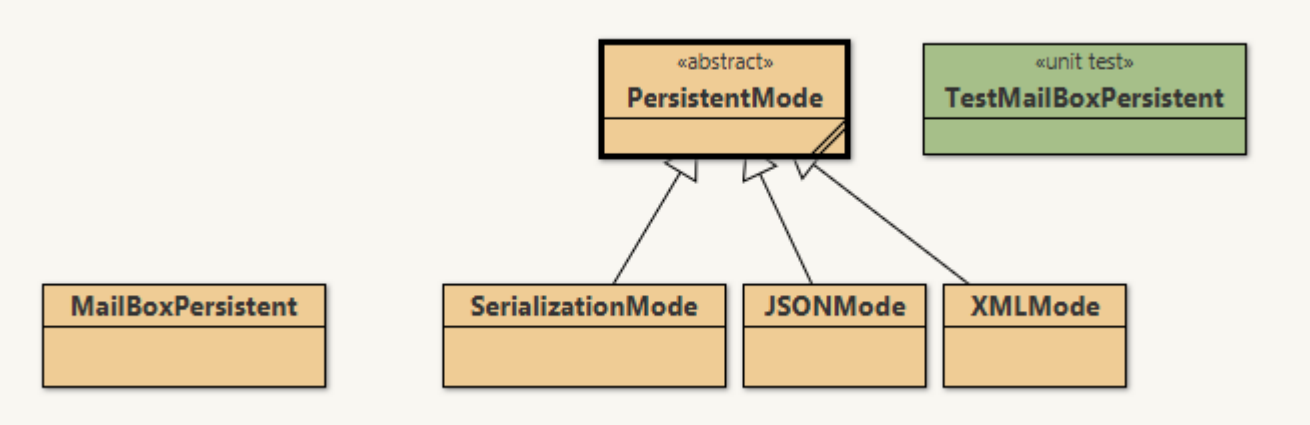
Ecrivez une implémentation **complète de la classe *MailBoxNotification***. Certaines documentations d'interfaces et de classes sont en annexe.

Question3 : Persistance

La persistance de la boîte aux lettres est assurée par un fichier selon un mode choisi à l'exécution par l'utilisateur.

Plusieurs modes sont possibles, **seule la persistance à l'aide de la sérialisation en java (cf. classe SerializationMode) est demandée**.

L'architecture proposée en syntaxe UML/BlueJ:



La classe abstraite ***PersistentMode*** propose les méthodes de sauvegarde et de restitution partielle d'une instance de boîte aux lettres**, attention, seuls les messages non lus sont restitués**.

```
public abstract class PersistentMode implements java.io.Serializable{
```

```
    /** Sauvegarde d'une instance de boîte aux lettres.
     * @param mb la boîte mail
     * @param fileName le nom du fichier
     * @throws Exception levée d'une exception pour toute anomalie
     */
    public abstract void writeMailBox(IMailBox mb, String fileName) throws Exception;

    /** restitution d'une instance de boîte aux lettres.
     * Attention, Seuls les messages non lus sont restitués.
     * @param fileName le nom du fichier
     * @throws Exception levée d'une exception pour toute anomalie
     * @return la boîte mail
     */
    public abstract IMailBox readMailBox(String fileName) throws Exception;

}
```

Ci-dessous un extrait de la classe **TestMailBoxPersistent,** l'enchaînement des appels des méthodes de test est le suivant :

1. Une sauvegarde (**`testSauvegarde`**) est effectuée puis,
2. une première restitution (**`testRestitution1`**) suivie,
3. d'une seconde restitution (**`testRestitution2`**).

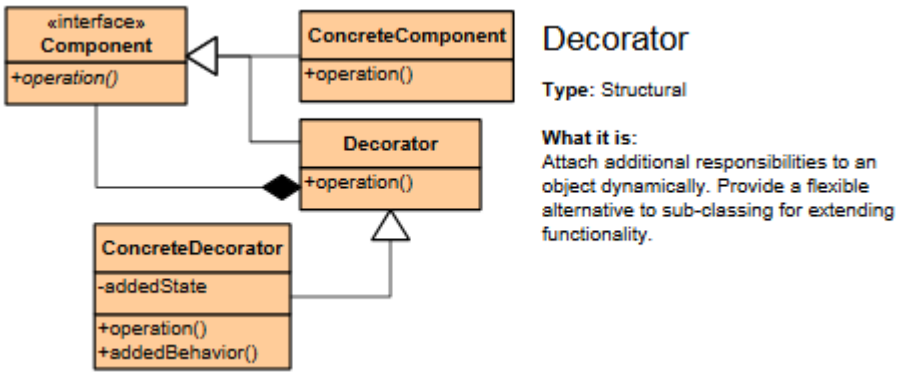
```
public void testSauvegarde() throws Exception{
  new File("./mailbox.ser").delete(); // suppression du fichier

  IMailBox mb = new MailBoxPersistent("mailbox.ser",new SerializationMode());
  mb.add("a");mb.add("b");mb.add("c");mb.add("d");
  mb.send("msg_1",new String[]{"a","b","c"});
  for(int i=2; i<4; i++){
    mb.send("msg_"+i,new String[]{"a","b","c","d"});
  }
  List<String> list = mb.read("a"); // lecture des messages pour "a"
  assertEquals(3,list.size());
  list = mb.read("a");
  assertEquals(0,list.size());
}

public void testRestitution1() throws Exception{
  IMailBox mb = new MailBoxPersistent("mailbox.ser",new SerializationMode());
  List<String> list = mb.read("a");
  assertEquals(0,list.size());
  list = mb.read("b");              // lecture des messages pour "b"
  assertEquals(3,list.size());
  list = mb.read("d");              // lecture des messages pour "d"
  assertEquals(2,list.size());
  mb.send("msg_4",new String[]{"a","b","d"});
}

public void testRestitution2() throws Exception{
  IMailBox mb = new MailBoxPersistent("mailbox.ser",new SerializationMode());
  List<String> list = mb.read("a");
  assertEquals(1,list.size());
  list = mb.read("b");              // lecture des messages pour "b"
  assertEquals(1,list.size());
  list = mb.read("d");              // lecture des messages pour "d"
  assertEquals(1,list.size());
}
```

Question3-1) Ecrivez complètement les classes ***SerializationMode*** et ***MailBoxPersistent***

Question3-2) Quels sont les patrons de conception utilisés pour cette question ?

Question3-3) En deux ou trois phrases, rappelez le principe de l'injection de dépendances et, ce que pourrait apporter à cette question l'utilisation d'un outil comme femtoContainer étudié dans cette unité.

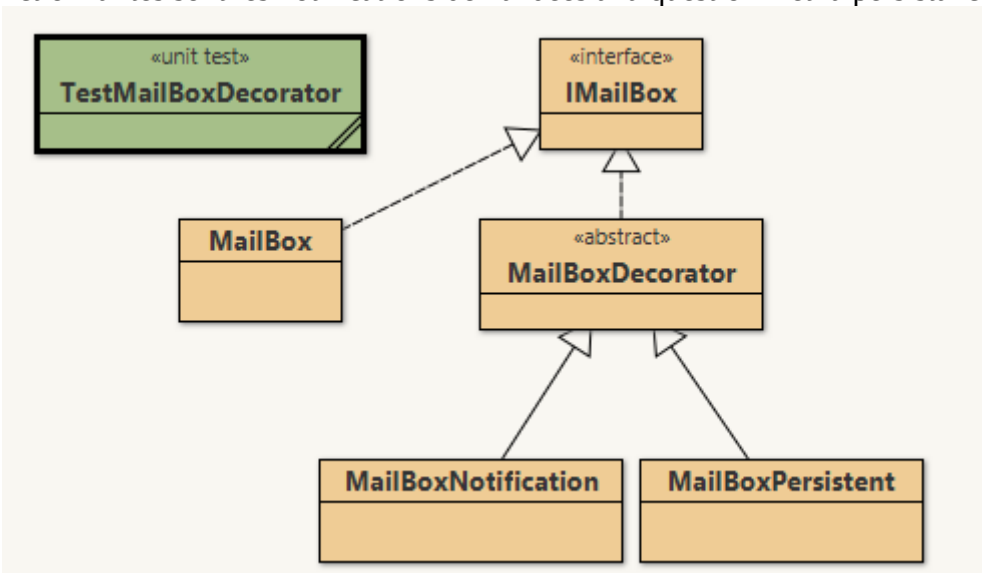
Question4 : Le patron Décorateur



Selon Wikipedia <https://fr.wikipedia.org/wiki/Patron_de_conception>
<<Le patron décorateur (en anglais decorator) permet d'attacher dynamiquement des responsabilités à un objet. … Un objet peut être caché à l'intérieur d'un autre objet décorateur qui lui rajoutera des fonctionnalités, l'ensemble peut être décoré avec un autre objet qui lui ajoute des fonctionnalités et ainsi de suite. Cette technique nécessite que l'objet décoré et ses décorateurs implémentent la même interface, qui est typiquement définie par une classe abstraite>>

Question4-1)

Le patron décorateur est maintenant utilisé pour les fonctionnalités supplémentaires de la boite aux lettres initiale. Ces fonctionnalités sont les notifications demandées à la question 2 et la persistance issue de la question 3.



Un extrait de la classe de test TestMailDecorator :

```
public void testSauvegarde() throws Exception{
    public void testDecorateur() throws Exception{
      IMailBox mb = new MailBoxNotification(new MailBoxPersistent(new MailBox(),"mailbox.ser"));
      ....
```

Proposer uniquement l'interface IMailBox et la classe **MailBoxDecorator**, les décorateurs concrets **MailBoxNotification** et **MailBoxPersistent** **ne sont pas demandés**.

Question4-2)

Que pourrait apporter à cette question l'utilisation d'un outil permettant l'injection de dépendances ?.

N.B.Un corrigé sera en ligne dans quelques jours http://jfod.cnam.fr/NFP121/annales/2020_fevrier/

Annexes

java.util
Interface Collection<E>

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

Method Summary	
boolean	**add**(E e) Ensures that this collection contains the specified element (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (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	**containsAll**(Collection<?> c) Returns true if this collection contains all of the elements in the specified collection.
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).
boolean	**removeAll**(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	**retainAll**(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this collection.
Object[]	**toArray**() Returns an array containing all of the elements in this collection.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

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

All Superinterfaces:

Collection<E>, Iterable<E>

All Known Implementing Classes:

LinkedList, … ArrayList, Vector

---

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	**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	**isEmpty**() Returns true if this list contains no elements.
E	**get**(int index) Returns the element at the specified position in this list.
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).
int	**size**() Returns the number of elements in this list.

---

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

See Also:

HashMap, TreeMap, Hashtable

Nested Class Summary	
static interface	**Map.Entry<K,V>** A map entry (key-value pair).

Method Summary	
void	**clear**() Removes all 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.
Set<Map.Entry<K,V>>	**entrySet**() Returns a set view of the mappings contained in this map.
boolean	**equals**(Object o) Compares the specified object with this map for equality.
V	**get**(Object key) Returns the value to which this map maps the specified 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 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).
void	**putAll**(Map<? extends K,? extends V> t) Copies all of the mappings from the specified map to this map (optional operation).
V	**remove**(Object key) Removes the mapping for this key from this map if it is present (optional operation).
int	**size**() Returns the number of key-value mappings in this map.
Collection<V>	**values**() Returns a collection view of the values contained in this map.

java.lang
Class String

Constructor Summary
String() Initializes a newly created `String` object so that it represents an empty character sequence.
String(String original) Initializes a newly created `String` object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string.

java.io
Class ObjectOutputStream

Constructor Summary	
	ObjectOutputStream(OutputStream out) Creates an ObjectOutputStream that writes to the specified OutputStream.

Method Summary	
void	**writeObject**(Object obj) Write the specified object to the ObjectOutputStream.

java.io
Class ObjectInputStream

Constructor Summary	
	ObjectInputStream(InputStream out)

Method Summary	
Object	**readObject**(Object obj)

```
public MailBox(){

public MailBox(MailBox mb){

public void add(String dest) throws Exception{

public void send(String message, String dest) throws Exception{

public void send(String message, String[] destinataires) throws Exception{

public List<String>  read(String dest) throws Exception{

public Iterator<String> iterator(){

public List<String> getAllMessages(String dest) throws Exception{

public List<String> getAllUnreadMessages(String dest) throws Exception{

public boolean equals(Object o){
    if(!(o instanceof MailBox)) return false;

public int hashCode(){ return
```