Nom et prénom : **……**

Contrôle terminal (1h45min)

Exercice 1 :

Soit la hiérarchie de classes suivante :

```
public class MyException extends Exception { }
public class MyException1 extends MyException { }
public class MyException3 extends MyException { }
public class MyException2 extends MyException1 { }
```

Question 1. On veut appeler la méthode dont la signature est donnée ci-dessous et gérer les trois erreurs séparément, dans quel ordre doit-on mettre les clauses catch ? Justifier votre réponse.

```
public static void compute() throws MyException1, MyException3, MyException2;
```

……
……
……
……

Question 2. Considérons la même hiérarchie de classes ci-dessus.

```
public class Test {
   public static void main (String[] args)
   {
      try
      {
         m1();
        System.out.print ("C");
      }
      catch (MyException e)
      {
        System.out.print ("D");
      }
      System.out.print ("E");
   }
   private static void m1() throws MyException
   {
      try
      {
         m2();
      }
      catch (MyException1 e)
      {
         System.out.print ("A");
         throw new MyException();
      }
   }
   private static void m2() throws MyException
   {
      throw new MyException2();
   }
}
```

Qu'affiche le programme ci-contre à la console ? Justifier votre réponse.

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Nom et prénom : **……**

Exercice 2 :

Nous allons dans cet exercice utiliser la classe ArrayList afin de simuler le comportement des ensembles. Afin de simplifier cet exercice, nous considérerons des ensembles contenant uniquement des nombres entiers (mais il est aisé de créer des ensembles d'objets divers).

Pour créer notre classe Ensemble, nous utiliserons comme attribut (privé) une "ArrayList" d'éléments de type Integer : `private ArrayList<Integer> elements ;`

Vous définirez pour cette classe Ensemble, les différentes méthodes suivantes :

`public Ensemble()`	Constructeur créant un ensemble vide.
`public Ensemble(Ensemble e)`	Constructeur créant un ensemble non vide.
`public void ajoute(Integer a)`	Permettant d'ajouter un élément à notre ensemble.
`public int taille()`	Permettant de retourner la taille de l'ensemble.
`Public boolean estVide(Ensemble e)`	Renvoie true si l'ensemble donné en paramètre est vide.
`public Integer getElement(int i)`	Permettant de retourner le $i^{ème}$ élément de notre ensemble en supposant bien sûr qu'il existe.
`public String toString()`	Permettant l'affichage de l'ensemble, on retournera ensemble vide si l'ensemble ne contient aucun élément.
`public void remove(Integer a)`	Supprime l'entier donné en paramètre de l'ensemble.
`public boolean appartient(Integer a)`	Renvoie true si l'entier en paramètre appartient à l'ensemble.
`public Ensemble union(Ensemble e)`	Permettant de réaliser une union de l'ensemble courant avec l'ensemble passé en paramètre.
`public Ensemble intersection(Ensemble x)`	Permettant de réaliser l'intersection entre l'ensemble courant et un ensemble passé en paramètre.
`public boolean inclusion(Ensemble e)`	renvoie true si l'ensemble est inclus dans un autre passé en paramètre.

Question 1. Créer la classe Ensemble, en définissant les méthodes ci-dessus.

Question 2. Réaliser la classe qui va tester l'application en utilisant tous les méthodes de la classe Ensemble.

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

Nom et prénom : ……