EPF – Lausanne *COURS DE MATHEMATIQUES SPECIALES*

15 avril 2015

Contrôle d'informatique no 3

Solution

Durée : 1 heure 45'

Nom : …… Groupe : ☐

Prénom : ……

No	1	2
Total points	points (4+20+55+18)	points 55

Remarque générale : toutes les questions qui suivent se réfèrent au langage de programmation Java (à partir du JDK 5.0) et les réponses doivent être rédigées à l'encre et d'une manière propre sur ces feuilles agrafées.

Remarques initiales :

- Il est conseillé de lire les sujets jusqu'à la fin, avant de commencer la rédaction de la solution.
- Les sujets peuvent être résolus (éventuellement) séparément, mais après avoir lu et compris l'ensemble des énoncés.
- Si vous ne savez pas implémenter une méthode, écrivez son en-tête, réduisez son corps à un simple commentaire et passez à la suite.
- La classe ***Object*** est une classe prédéfinie (ou standard) du package ***java.lang*** ; elle est la classe "racine", ancêtre de toute autre classe Java.
- La classe ***String*** est une classe prédéfinies du package ***java.lang***; elle permet notamment la manipulation des chaînes de caractères en tant qu'objets.
- Si dans l'en-tête de la définition d'une classe on ne précise pas explicitement quelle est sa classe mère (ou sa classe de base), cette nouvelle classe définie dérive implicitement de la classe "racine" ***java.lang.Object***.
- Faites attention aux éventuels **casts** obligatoires ainsi qu'aux accès aux champs privés.

Sujet no 1.

On considère un projet Java qui contient une interface **publique** et trois classes **publiques** regroupées dans un package nommé ***cms_ctr3***, à savoir :

- l'interface ***IComparable*** ;
- la classe ***Couple*** qui implémente l'interface mentionnée ci-dessus ;
- la classe **C_String** qui dérive de la classe de base ***Couple*** ;
- la classe principale ***CP_Ctr3Exo1*** qui utilise la classe ***C_String***.

Le but de cet exercice est de réaliser une application autonome interactive qui aide l'utilisateur à travailler avec des couples de chaînes de caractères. L'interface et chaque classe sont **publiques** et définies dans un fichier à part. On vous demande d'écrire les codes complets de l'interface ***IComparable*** et de la classe ***C_String***, en respectant les consignes précisées. En outre, le code source de la classe principale ***CP_Ctr3Exo1*** est donné et vous devrez indiquer quels seront les résultats affichés dans la fenêtre console suite à son exécution

1.1 L'interface **publique** ***IComparable*** fait partie du package ***cms_ctr3*** et contient l'en-tête d'une seule méthode qui est nommée ***comparer***, qui a un seul argument de type ***Object*** et qui retourne un résultat de type numérique entier.

Indiquer ci-dessous le code complet de l'interface ***IComparable***.

.....

```
package cms_ctr3;

public interface IComparable
{
    int comparer(Object o);
}
```

.....

1.2

La classe ***Couple*** est **publique**, fait partie du package ***cms_ctr3*** et implémente l'interface ***IComparable*** sans redéfinir la méthode ***comparer***.

De plus, la classe ***Couple*** définit :

- **a)** un champ (d'instance) nommé ***first***, de type ***Object***, déclaré **privé**, sans valeur initiale explicite et qui sert à stocker l'adresse d'un objet qui est le premier élément d'un couple d'objets ;
- **b)** un champ (d'instance) nommé ***second***, de type ***Object***, déclaré **privé**, sans valeur initiale explicite et qui sert à stocker l'adresse d'un objet qui est le deuxième élément d'un couple d'objets;
- **c)** une méthode **publique** (d'instance) "**getter**" nommée en respectant la convention Java pour le nommage des getters et qui retourne (sans aucune vérification) la valeur du champ privé **first** ;
- **d)** une méthode **publique** (d'instance) "**setter**" nommée en respectant la convention Java pour le nommage des setters et qui permet la modification de la valeur du champ privé ***first*** ; plus précisément, la valeur de son argument, noté lui aussi ***first***, est copiée (sans aucune vérification) dans le champ ***first*** ;
- **e)** une méthode **publique** (d'instance) "**getter**" similaire à celle précisée au point **c)** mais pour le champ privé ***second*** ;
- **f)** une méthode **publique** (d'instance) "**setter**" similaire à celle précisée au point **d)** mais pour le champ privé ***second*** ;
- **g)** un constructeur **public** sans argument qui affiche le message ***Couple d'objets !***

Indiquer à la page suivante le code source complet de la classe ***Couple***.

X

X

```
package cms_ctr3;

public abstract class Couple implements IComparable
{
     private Object first;
     private Object second;

     public Object getFirst()
     {
          return first;
     }

     public void setFirst(Object first)
     {
          this.first = first;
     }

     public Object getSecond()
     {
          return second;
     }

     public void setSecond(Object second)
     {
          this.second = second;
     }

     //abstract public  Couple() FAUX
     public Couple()
     {
          System.out.println("Couple d'objets !");
     }
}
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

1.3 Une instance de la classe ***C_String*** est un objet de type ***C_String*** qui encapsule les valeurs des deux chaînes de caractères (par exemple un prénom et un nom de famille) qui peuvent être considérées comme "un couple de chaînes de caractères".

La classe ***C_String*** est **publique**, fait partie du package ***cms_ctr3*** et dérive de la classe ***Couple***. Autrement dit, la classe ***C_String*** est la fille (ou la classe dérivée) de la classe mère (ou classe de base) ***Couple***.

De plus, la classe ***C_String*** :

a) redéfinit la méthode **publique** (d'instance) "**setter**" associée au champ hérité ***first*** ; plus précisément, cette méthode doit respecter les consignes suivantes :

- **i.** si la valeur de son argument est la valeur spéciale ***null*** ou si la classe correspondant au type de l'objet argument n'est pas (exactement) la classe prédéfinie ***String***, la méthode stocke dans le champ ***first*** la valeur spéciale ***null*** ;
- **ii.** autrement, si la chaîne de caractères correspondant à son argument commence par une lettre majuscule, la méthode stocke la valeur de son argument dans le champ ***first*** ;
- **iii.** autrement, la méthode stocke la chaîne de caractères ***First Name*** dans le champ ***first***;

b) redéfinit la méthode **publique** (d'instance) "**setter**" associée au champ hérité ***second*** ; plus précisément, cette méthode doit respecter les consignes suivantes :

- **i.** si la valeur de son argument est la valeur spéciale ***null*** ou si la classe correspondant au type de l'objet argument n'est pas (exactement) la classe prédéfinie ***String***, la méthode stocke dans le champ ***second*** la valeur spéciale ***null*** ;
- **ii.** autrement, si la chaîne de caractères correspondant à son argument commence par une lettre majuscule et si elle ne contient aucun caractère espace, la méthode stocke la valeur de son argument dans le champ ***second*** ;
- **iii.** autrement, la méthode stocke dans le champ ***second*** la chaîne de caractères ***Surname*** ;

c) un constructeur **public** avec deux arguments de type chaîne de caractères et qui permet la création d'un objet correspondant à un nouveau couple de chaînes de caractères ; ce constructeur doit respecter les consignes suivantes :

- **i.** par un appel à la méthode "setter" adéquate, il "stocke" la valeur du premier argument dans le champ (hérité) ***first*** du nouvel objet ;

ii. par un appel à la méthode "setter" adéquate, il "stocke" la valeur du deuxième argument dans le champ (hérité) ***second*** du nouvel objet ;

iii. il affiche à l'écran le message ***Couple de String !***

Ensuite, la classe ***C_String*** redéfinit les méthodes ("héritées") suivantes :

d) la méthode ***comparer*** qui, normalement, doit réaliser une comparaison lexicographique entre la chaîne de caractères obtenue par la concaténation des champs ***first*** et ***second*** de l'objet appelant et de la chaîne de caractères obtenue par la concaténation des champs ***first*** et ***second*** de l'objet argument ; plus précisément, cette méthode doit respecter les consignes suivantes :

i. si le champ hérité ***first*** de l'objet appelant a la valeur spéciale ***null*** ou si le champ hérité ***second*** de l'objet appelant a la valeur spéciale ***null***, la méthode retourne la valeur entière ***-11*** ;

ii. (autrement) si la valeur de son argument est la valeur spéciale ***null*** ou si la classe correspondant au type de l'objet argument n'est pas (exactement) la classe de l'objet appelant, la méthode retourne la valeur entière ***666*** ;

iii. (autrement) si le champ hérité ***first*** de l'objet argument a la valeur spéciale ***null*** ou si le champ hérité ***second*** de l'objet argument a la valeur spéciale ***null***, la méthode retourne la valeur entière ***22*** ;

iv. (autrement) on procède ainsi :

i. on définit un variable locale de type chaîne de caractères nommée ***sAppelant*** et on y stocke le résultat obtenu par la concaténation du champ **first** (transformé en chaîne de caractères) et du champ **second** (transformé en chaîne de caractères) de l'objet appelant ;

ii. on définit une variable locale de type chaîne de caractères nommée ***sArgument*** et on y stocke le résultat obtenu par la concaténation du champ **first** (transformé en chaîne de caractères) et du champ **second** (transformé en chaîne de caractères) de l'objet argument ;

iii. on définit une variable locale de type numérique entier nommée ***ent*** et on y stocke le résultat de la comparaison lexicographique des deux chaînes de caractères mentionnées (comparaison réalisée par un appel de la méthode appropriée de la classe ***String*** en utilisant comme objet appelant la chaîne référencée par la variable ***sAppelant*** précisée ci-dessus) ;

iv. si la valeur de le variable ***ent*** est strictement négative, la méthode ***comparer*** retourne la valeur entière ***-1*** ;

v. (autrement) si la valeur de la variable ***ent*** est strictement positive, la méthode ***comparer*** retourne la valeur entière ***1*** ;

vi. (autrement) la méthode ***comparer*** retourne la valeur entière ***0*** ;

e) la méthode ***toString*** qui doit retourner la chaîne de caractères ***Un couple de String XXX YYY.*** où ***XXX*** et ***YYY*** sont les chaînes de caractères qui correspondent, respectivement, aux champs hérités ***first*** et ***second*** de l'objet appelant.

Indiquer ci-dessous le code source complet de la classe ***C_String***.

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

```
package cms_ctr3;

public class C_String  extends Couple
{    @Override
     public void setFirst(Object first)
     {    //if(premier == null || premier.getClass() !=
                                       String.class)
          //if(premier == null || premier.getClass() !=
                                   "abc".getClass())
          if(first == null ||
   !first.getClass().getName().equals("java.lang.String"))
          {
               super.setFirst(null);
          }else
          {
               String p = (String)first;
               if(p.charAt(0)>='A' && p.charAt(0) <='Z')

                    super.setFirst(first);
               else
                    super.setFirst("First Name");
          }
     }

     @Override
     public void setSecond(Object second)
     {    if(second == null || second.getClass() !=
                                  "abc".getClass())
          {
               super.setSecond(null);
          }else
          {    String p = (String)second;
               if(p.charAt(0)>='A' && p.charAt(0) <='Z' &&
                                        p.indexOf(' ')<0)
               {
                    super.setSecond(second);
               }else
               {
                    super.setSecond("Surname");
               }
          }
     }
```

```
    public C_String(String first, String second)
    {
        setFirst(first);
        setSecond(second);
        System.out.println("Couple de String !");
    }

    @Override
    public int comparer(Object o)
    {
        if(this.getFirst()==null ||
                            this.getSecond()==null)
            return -11;

        if(o == null || o.getClass() != this.getClass())
            return 666;

        C_String cs = (C_String)o;
        if(cs.getFirst()==null || cs.getSecond()==null)
            return 22;

        String sAppelant = this.getFirst().toString() +
                            this.getSecond().toString();
        String sArgument = cs.getFirst().toString() +
                           cs.getSecond().toString();

        int ent = sAppelant.compareTo(sArgument);
        if(ent < 0)
            return -1;
        if(ent > 0)
            return 1;
        return 0;
    }

    @Override
    public String toString()
    {
        return "Un couple de String : " +
                  getFirst() + " " + getSecond() + ".";
    }
}
```

```
package cms_ctr3;

public class C_String  extends Couple
{    @Override
     public void setFirst(Object first)
     {    //if(first == null || first.getClass() !=
                                       String.class)
          //if(first == null || first.getClass() !=
                                       "abc".getClass())
          if(first == null ||
   !first.getClass().getName().equals("java.lang.String"))
          {
               super.setFirst(null);
          }else
          {
               if(((String)this.getFirst()).charAt(0)>='A'
               && ((String)this.getFirst()).charAt(0) <='Z')
               {
                    super.setFirst(first);
               }else
               {
                    super.setFirst("First Name");
               }
          }
     }

     @Override
     public void setSecond(Object second)
     {    if(second == null || second.getClass() !=
                                  "abc".getClass())
          {
               super.setSecond(null);
          }else
          {    if(((String)this.getSecond()).charAt(0)>='A'
              && ((String)this.getSecond()).charAt(0) <='Z'
              && ((String)this.getSecond()).indexOf(' ')<0)
               {
                    super.setSecond(second);
               }else
               {
                    super.setSecond("Surname");
               }
          }
     }
```

```
    public C_String(String first, String second)
    {
        setFirst(first);
        setSecond(second);
        System.out.println("Couple de String !");
    }

    @Override
    public int comparer(Object o)
    {
        if(this.getFirst()==null ||
                            this.getSecond()==null)
            return -11;

        if(o == null || o.getClass() != this.getClass())
            return 666;

        C_String cs = (C_String)o;
        if(cs.getFirst()==null || cs.getSecond()==null)
            return 22;

        String sAppelant = this.getFirst().toString() +
                            this.getSecond().toString();
        String sArgument = cs.getFirst().toString() +
                           cs.getSecond().toString();

        int ent = sAppelant.compareTo(sArgument);
        if(ent < 0)
            return -1;
        if(ent > 0)
            return 1;
        return 0;
    }

    @Override
    public String toString()
    {
        return "Un couple de String : " +
                  getFirst() + " " + getSecond() + ".";
    }
}
```

1.4 On donne ci-dessous le code source de la classe principale ***CP_Ctr3Exo1*** qui fait partie du package ***cms_ctr3*** et qui utilise les classes présentées auparavant.

```
package cms_ctr3;

public class CP_Ctr3Exo1
{
      public static void main(String[] args)
      {
            Couple couples[] = new Couple[3];
            Object prenoms[] = {"Jean Claude", "Bonnet",
                                          new Character('1')};
            Object noms[] = {"Van Damme", "Blanc", "Bonaparte"};

            for(int i=0; i<couples.length; i++)
            {
                  couples[i] = new C_String(prenoms[i].toString(),
                                          noms[i].toString());
                  System.out.println(couples[i]);
                  System.out.println("----------------------------");
            }

            for(int i=0; i<couples.length; i++)
            {
                  System.out.println(couples[i].
                        comparer(couples[(i+1) % couples.length]));
                  System.out.println("----------------------------");
            }
      }//fin de la méthode main
}//fin de la classe principale
```

Préciser <u>à la page suivante</u> quels sont les résultats affichés dans la fenêtre console suite à l'exécution de la classe principale ***CP_Ctr3Exo1***.

```
Couple d'objets !
Couple de String !
Un couple de String : Jean Claude Surname.
------------------------------
Couple d'objets !
Couple de String !
Un couple de String : Bonnet Blanc.
------------------------------
Couple d'objets !
Couple de String !
Un couple de String : First Name Bonaparte.
------------------------------
1
------------------------------
-1
------------------------------
-1
------------------------------
```

Sujet no 2.

Préciser les messages qui seront affichés à l'écran suite à l'exécution du projet correspondant au code source suivant (qui est contenu dans un même fichier **CP_Ctr3Exo2.java**).

```
package cms_ctr3;

class Test
{     int m = 10;

      int tester(int n) throws ImpairException
      {     try{
                  if(n > 1000 && n %2 == 1)
                        throw new TropGrandImpairException(n);
                  if(n % 2 == 1 || n % 2 == -1)
                        throw new ImpairException(n);
                  if(n < 0)
                        throw new NegatifException();
            }catch(TropGrandImpairException tgie){
                  if(tgie.nb < 10_000){
                        System.out.println("Traitement local !");
                        return 10_000 - tgie.nb;
                  }else
                  {     System.out.println("Traitement raté !");
                        throw tgie;
                  }
            }catch(NegatifException ne){
                  System.out.println(ne);
                  return m;
            }finally{
                  m = -m;
            }
            System.out.println("Pas de soucis !");
            return n*m;
      }//fin de la méthode test
}//fin de la classe Test
```

```
class ImpairException extends Exception
{     int nb;
      ImpairException(int i){
            super("Pas pair !");
            nb = i;
            System.out.println("Impair !");
      }
}//fin de la classe ImpairException

class TropGrandImpairException extends ImpairException
{     TropGrandImpairException(int i){
            super(i);
            System.out.println("Trop impair !");
      }
}//fin de la classe TropGrandImpairException

class NegatifException extends RuntimeException
{     public String toString(){
            return "Au-dessous du zéro !";
      }
}

public class CP_Ctr3Exo2 {
      public static void main(String args[ ])
      {     Test t  = new Test();
            int tab[] = {66, -33, -66, 33, 100_111, 8_999};

            for(int i =0; i<tab.length; i++){
                  try{
                        System.out.println(t.tester(tab[i]));
                  }catch(TropGrandImpairException tgie){
                        System.out.println(tgie.getMessage());
                        System.out.println("Rien à faire !");
                  }catch(ImpairException ie){
                        System.out.println(ie.getMessage());
                        System.out.println("Problème: " + ie.nb + "!");
                  }
                  System.out.println("t.m vaut : " + t.m + ".");
                  System.out.println("-----------------------------");
            }

            System.out.println("Fin de la méthode main !");
      }//fin de la méthode main
}//fin de la classe principale
```

```
Pas de soucis !
-660
t.m vaut : -10.
------------------------------
Impair !
Pas pair !
Problème: -33!
t.m vaut : 10.
------------------------------
Au-dessous du zéro !
10
t.m vaut : -10.
------------------------------
Impair !
Pas pair !
Problème: 33!
t.m vaut : 10.
------------------------------
Impair !
Trop impair !
Traitement raté !
Pas pair !
Rien à faire !
t.m vaut : -10.
------------------------------
Impair !
Trop impair !
Traitement local !
1001
t.m vaut : 10.
------------------------------
Fin de la méthode main !
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

A part ce texte, cette dernière page doit rester vide, mais vous pouvez l'utiliser si la place prévue pour vos réponses n'a pas été suffisante.