Université de Montpellier / Faculté Des Sciences

Correction des exercices sur la généricité en Java (introduction)

1er février 2020

QUESTION 1 *Transformer la classe* `FileAttente` *en classe générique, puis écrivez un programme qui crée et remplit une file d'attente de personnes, puis une file d'attente de rectangles.*

```
public class FileAttente
{
     private String nomFile;
     private static int nbPersonnesEntreesTotal = 0;
     private ArrayList<Personne> contenu;
     public FileAttente(){contenu=new ArrayList<Personne>();}
     public void entre(Personne p){contenu.add(p); nbPersonnesEntreesTotal++;}
     public Personne sort()
          {
               Personne p=null;
               if (!contenu.isEmpty())
                    {p=contenu.get(0);
                    contenu.remove(0);}
               return p;
          }
     public boolean estVide(){return contenu.isEmpty();}
     public String toString(){return ""+descriptionContenu();}
     private String descriptionContenu()
     {
          String resultat = "";
          for (Personne p:this.contenu)
               resultat += p + " ";
          return resultat;
     }
}
```

RÉPONSE 1

```
public class FileAttente<E>
{
private String nomFile;
private static int nbElementsEntresTotal = 0;
private ArrayList<E> contenu;
```

```
public FileAttente(){contenu=new ArrayList<E>();}

public int nbElements(){return contenu.size();}

public void entre(E p){contenu.add(p); nbElementsEntresTotal++;}

public E sort()
{
E p=null;
if (!contenu.isEmpty())
{p=contenu.get(0);contenu.remove(0);}
return p;
}

public boolean estVide(){return contenu.isEmpty();}

public String toString(){return ""+descriptionContenu();}

private String descriptionContenu()
{
String resultat = "";
for (E p:this.contenu)
resultat += p + " ";
return resultat;
}

public static void main(String[] args) {
FileAttente<Personne> fp = new FileAttente<>();
fp.entre(new Personne("Jeanne","Larivière"));
FileAttente<Rectangle> fr = new FileAttente<>();
fr.entre(new Rectangle(15,18));
}

}
```

QUESTION 2

- *Ecrivez pour la classe générique* `File d'attente` *une méthode statique permettant de retourner le nombre d'éléments entrés dans toutes les files d'attente (utilisez l'attribut existant* `nbPersonnesEntreesTotal` *en le renommant* `nbElementsEntresTotal`*).*

- *Ecrivez pour la classe générique* `File d'attente` *une méthode statique prenant en paramètre deux files d'attente contenant des objets de même type et retournant vrai si elles contiennent les mêmes objets.*

- *Ecrivez pour la classe générique* `File d'attente` *une méthode non statique prenant en paramètre une file d'attente contenant des objets du même type que la file receveur et retournant vrai si les deux files contiennent les mêmes objets.*

— *Ecrivez un programme qui montre comment appeler ces méthodes.*

Réponse 2
La première méthode statique demandée n'a pas besoin d'être paramétrée car le type du paramètre de généricité n'intervient pas dans le calcul.

```
public static int getNbPersonnesEntreesTotal() //
  {
    return FileAttente.nbElementsEntresTotal;
  }
```

La seconde méthode statique demande de réintroduire un paramètre, ici pour vérifier que f1 et f2 contiennent des éléments de même type.

```
public static <T> boolean memeContenu(FileAttente<T> f1, FileAttente<T> f2)
  {
    if (f1.nbElements() != f2.nbElements()) return false;
    for (int i=0; i< f1.nbElements(); i++)
    if (! f1.contenu.get(i).equals(f2.contenu.get(i)))
      return false;
    return true;
  }
```

La méthode non statique ne nécessite pas de réintroduire un nouveau paramètre de généricité.

```
public  boolean memeContenu(FileAttente<E> autreFile)
{
  if (this.nbElements() != autreFile.nbElements()) return false;
  for (int i=0; i< this.nbElements(); i++)
  if (! this.contenu.get(i).equals(autreFile.contenu.get(i)))
    return false;
  return true;
}

public static void main(String[] args) {
  FileAttente<Personne> fp = new FileAttente<>();
  Personne p = new Personne("Jeanne","Larivière");
  fp.entre(p);
  FileAttente<Personne> fp2 = new FileAttente<>();
  fp2.entre(p);
  System.out.println(memeContenu(fp,fp2));
  System.out.println(fp.memeContenu(fp2));
  FileAttente<Rectangle> fr = new FileAttente<>();
  fr.entre(new Rectangle(15,18));
  //System.out.println(memeContenu(fp,fr)); ne peut pas compiler
}
```

Question 3 *Ecrivez pour la classe générique* ***File d'attente*** *une méthode non statique prenant en paramètre une file d'attente avec des éléments qui ne sont pas forcément du même type que la file receveur et retournant vrai si les deux files sont de la même longueur.*

Ecrivez un programme qui montre comment l'appeler sur une file de rectangles et sur une file de personnes.

RÉPONSE 3 *Pour écrire cette méthode, même si elle est non statique, on doit ajouter un nouveau paramètre de généricité.*

```
public <T> boolean memeLongueur(FileAttente<T> autreFile)
{return this.nbElements() == autreFile.nbElements();}
```

A la suite du programme de la question précédente, on peut à présent écrire

```
System.out.println(fp.memeLongueur(fr));
```

QUESTION 4 *Ecrivez un programme créant une file d'attente de personnes et une file d'attente de rectangles. Mettez une personne dans la première et un rectangle dans la seconde. A votre avis, quelle est la valeur de l'attribut static* `nbPersonnesEntreesTotal` *après ces deux opérations ? Comment l'expliquez-vous ?*

RÉPONSE 4 *L'attribut statique stocke le nombre de personnes entrées dans toutes les files de tous les types, c'est une des conséquences du principe de l'effacement de type à la compilation. Il y a une seule variable statique* `nbPersonnesEntreesTotal` *pour tous les paramétrages.*

QUESTION 5 *Pleins de bonnes intentions, nous voudrions transformer la classe implémentant la liste avec un tableau en classe générique, et nous écrivons le code (partiel) ci-dessous. Malheureusement, cela va mal se passer. Essayez de comprendre quelle est la partie du code en cause.*

```
public class listeTabGenerique<E> {
private int nbrElements=0;
private int tailleInitiale=10;
private E[] contenu;

public listeTabGenerique()
{this.contenu = new E[this.tailleInitiale];}

public listeTabGenerique(int tailleInitiale)
{
      this.tailleInitiale=tailleInitiale;
      this.contenu = new E[this.tailleInitiale];
 }
 //.............
}
```

RÉPONSE 5 *On ne peut pas compiler une expression comme* `new E[this.tailleInitiale];` *car le type* `E` *n'est pas connu (effacement de type).*

QUESTION 6 *Les tableaux vont se révéler produire d'autres ennuis. Le code suivant compile et provoque une erreur à l'exécution. Identifiez le problème.*

```
Object[] t = new Object[3];
t[0] = "Alice";
t[1] = new Personne();
t[2] = new Rectangle_tab();

t = new String[3];
t[0] = "Alice";
t[1] = new Personne();
t[2] = new Rectangle_tab();
```

Déduisez-en la raison pour laquelle ce qui suit (la deuxième affectation) vous sera interdit (ne compilera pas) :

```
ArrayList<Object> a = new ArrayList<Object>();
a = new ArrayList<String>();
```

RÉPONSE 6 *Le typage des tableaux n'est pas aussi sûr en Java que le typage des classes paramètrées. Après avoir déclaré le tableau t comme étant un tableau d'objets, on l'initialise ensuite avec une structure tableau de String. Au moment où on cherche, à l'exécution, à mettre autre chose que des String, une erreur se produit.*
On interdit :

```
ArrayList<Object> a = new ArrayList<Object>();
a = new ArrayList<String>();
```

car si c'était autorisé, on pourrait compiler une expression comme `a.add(new Rectangle());`*, les rectangles étant des objets. Mais à l'exécution, on aurait des erreurs puisque l'on s'attendrait à ce que* `a` *ne contienne que des String.*

QUESTION 7 *Ecrivez une classe générique* `PaireEtiquetee` *à partir de la définition de la classe générique* `Paire`*. Une paire étiquetée est une paire à laquelle on attache une étiquette. Le type de l'étiquette est aussi un paramètre de généricité. Ecrivez un programme créant des paires étiquetées avec des étiquettes de type* `String`*, puis des paires étiquetées avec des étiquettes de type* `Integer`*.*

RÉPONSE 7 *Cette question illustre la manière de dériver une classe générique à partir d'une autre classe générique et en ajoutant un paramètre de généricité de plus.*

```
public class PaireEtiquetee<A,B,TypeEtiquette> extends Paire<A,B> {
   private TypeEtiquette etiquette;
   public PaireEtiquetee() {}
   public PaireEtiquetee(A f, B s, TypeEtiquette etiquette) {
      super(f, s);
      this.etiquette = etiquette;
   }
   public TypeEtiquette getEtiquette() {return etiquette;}
   public void setEtiquette(TypeEtiquette etiquette) {this.etiquette = etiquette;}

   public static void main(String[] args) {
      PaireEtiquetee<String,Personne,String> p1=
         new PaireEtiquetee<String,Personne,String>("sarah",new Personne(),"prenom");
      PaireEtiquetee<String,Personne,Integer> p2=
```

```
        new PaireEtiquetee<String,Personne,Integer>("sarah",new Personne(),20021004);
   }
}
```

QUESTION 8 *Ecrivez une classe* ***EntreeAgenda*** *à partir de la définition de la classe générique* ***Paire****. Une entrée d'agenda est une paire dont le premier membre est une date et le deuxième membre est une chaîne de caractères. Ecrivez un programme créant des entrées d'agenda.*

RÉPONSE 8 *Cette question montre que l'on peut en même temps dériver par héritage en remplaçant les paramètres de généricité par des types réels (pour réaliser une invocation).*

```
public class EntreeAgenda extends Paire<Date,String> {
   public EntreeAgenda() { }
   public EntreeAgenda(Date f, String s) {super(f, s);}

   public static void main(String[] a){
      EntreeAgenda e1 = new EntreeAgenda(new Date(), "cours 4 FMIN220");
      System.out.println(e1);
   }
}
```

QUESTION 9 *Ecrivez une classe générique* ***BouteilleEtiquetee*** *à partir de la définition de la classe* ***Bouteille****. Une bouteille étiquetée est une bouteille à laquelle on attache une étiquette. Le type de l'étiquette est aussi un paramètre. Ecrivez un programme créant des bouteilles étiquetées avec des étiquettes de type* ***String****, puis des bouteilles étiquetées avec des étiquettes d'une classe* ***EtiquetteBouteille*** *que vous définirez. Une étiquette de bouteille comprend plusieurs informations : le degré d'alcool, le nom du producteur, son adresse, un descriptif du contenu, la quantité contenue (en litres).*

RÉPONSE 9 *Cette question montre que l'on peut dériver une classe générique (en ajoutant un paramètre) d'une classe ordinaire.*

```
public class BouteilleEtiquetee<TypeEtiquette> extends Bouteille {
   private TypeEtiquette etiquette;
   public BouteilleEtiquetee() {}
   public BouteilleEtiquetee(TypeEtiquette etiquette) {this.etiquette = etiquette;}
   public TypeEtiquette getEtiquette() {return etiquette;}
   public void setEtiquette(TypeEtiquette etiquette) {this.etiquette = etiquette;}

   public static void main (String[] a)
   {
     BouteilleEtiquetee<Etiquette> b1 = new BouteilleEtiquetee<Etiquette>(
           new Etiquette(0,"LaPetiteFerme","chemin des Alpes","sirop de fraise",1));
     BouteilleEtiquetee<String> b2 = new BouteilleEtiquetee<String>("sirop de fraise");

   }
}
```

QUESTION 10 *Ecrivez une classe générique* `PaireHomogène` *à partir de la définition de la classe générique* `Paire`*. Une paire homogène est une paire dont les deux membres sont de même type. Ecrivez un programme créant des paires homogènes.*

RÉPONSE 10 *Dans cette question on montre que l'on peut réduire le nombre de paramètres de généricité lors de la dérivation par héritage.*

```
public class PaireHomogene<A> extends Paire<A,A> {
   public PaireHomogene() {}
   public PaireHomogene(A f, A s) {super(f, s);}
}
```