Polymorphisme paramétrique (ou généricité)

Exercice 1 *Classe générique (implémentant une interface générique)*

Voici une interface définissant un type abstrait "Pile de <A>" avec les fonctionnalités classiques d'une pile :

```
public interface IPile<A>
{
  boolean estVide();
  void empile(A a);
  A depile(); // retourne l'element en sommet de pile et depile
  int nbElements();
  A sommet(); // retourne le sommet de pile mais ne le depile pas
}
```

Question 1. Écrivez une classe générique CPile qui implémente l'interface IPile. Vous stockerez les éléments de la pile dans une liste chaînée (instance de java.util.LinkedList, voir en annexe quelques méthodes publiques de cette classe).

Question 2. Écrivez un petit programme qui crée et manipule des piles en instanciant la classe générique de différentes façons (par exemple pile de String, pile de Integer, ...).

Exercice 2 *Une autre classe générique*

La classe suivante Tableau encapsule un tableau d'entiers et comporte deux méthodes permettant de trier le tableau et d'afficher son contenu. La méthode main de cette classe fournit un exemple d'utilisation. On voudrait disposer d'une méthode de tri pour n'importe quel type de tableau.

```
public class Tableau  {
        private int T [];

        public Tableau (int T []){
        this.T = T;} // on fait ici une recopie "superficielle"
        public void triBulles (){
                int i = T.length -2;
                boolean ech = true;
                while (i >=2 && ech){
                        ech = false;
                        for (int j = 0; j <= i; j ++)
                         if (T[j] > T[j+1]){
                                int aux = T[j];
                                T[j] = T[j+1];
                                T[j+1] = aux;
                                ech = true;
                         }
                 i--;
                }
        }

        public void affiche ()  {
                for(int i = 0; i < T.length; i++)
                        System.out.print(T[i]+" ");
                System.out.println();
        }

        public static void main(String[] args) {
                int T[] = {10,2,6,11, 7, 2, -1, 0, 9};
                Tableau obj = new Tableau(T);
                obj.triBulles();
                obj.affiche();
        }
}
```

Question 3. Proposez une solution basée sur la généricité.

Question 4. Testez votre solution. Trouvez et corrigez l'erreur dans l'algorithme de tri.

Exercice 3

Question 5. On considère une classe Personne (qui sera définie plus loin, sa définition n'a pas d'importance ici). Soit la classe suivante représentant une file d'attente de personnes :

```
public class FileAttente
{
        protected ArrayList<Personne> contenu;
        public FileAttente(){contenu=new ArrayList<Personne>();}
        public void entre(Personne p)
                {contenu.add(p);}
        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 ""+contenu;}
}
```

Proposez une classe générique représentant les files d'attente contenant des objets de n'importe quel type (personnes, voitures, etc.).

Question 6. Nous considérons à présent l'interface décrivant les objets munis d'une priorité.

```
public interface ElementAvecPriorite
{    int priorite();    }
```

Ecrivez une classe générique représentant les files d'attente avec priorité contenant des objets de n'importe quel type à condition qu'ils soient munis d'une priorité. Les éléments sortent de la file en favorisant ceux qui ont la plus petite priorité.

Question 7. La classe Personne est définie de la façon suivante :

```
public class Personne
{
        private String nom;
        private int age;
        public Personne(){}
        public Personne(String n, int a){nom=n;age=a;}
        public String getNom(){return nom;}
        public void setNom(String n){nom=n;}
        public int getAge(){return age;}
        public void setAge(int a){age=a;}
        public String toString(){return nom+" "+age;}
}
```

Modifier la classe Personne pour pouvoir stocker des personnes dans une file d'attente avec priorité. Vous pouvez utiliser l'âge pour déterminer trois niveaux de priorité (priorité 1 pour un âge entre 0 et 12, priorité 2 pour un âge de 60 et plus, priorité 3 pour les autres valeurs).

Question 8. Écrivez un programme dans lequel on déclare une file d'attente (avec priorité) de personnes et rangez-y quelques personnes.

Exercice 4 *Généricité contrainte, un peu d'exceptions et de typage*

Une médiathèque met en place des "pochettes" de documents à emprunter. Les pochettes ont un thème donné, et le personnel de la médiathèque conçoit des assortiments de documents autour de ce thème, l'assortiment reste une surprise pour le lecteur qui emprunte la pochette. On peut créer des pochettes avec un seul type de documents (par exemple des pochettes de documents audio) ou plusieurs (livres documentaires, BD, livre de recettes, etc). Les classes de documents sont déjà existantes dans la bibliothèque. Ici, on s'intéresse aux éléments suivants :

- interface Document qui définit l'interface commune à tous les documents. On pourra la laisser vide ici.
- interface DocumentJeunesse qui définit l'interface commune aux documents jeunesse. Elle étend Document et introduit une méthode permettant d'obtenir l'âge minimum conseillé pour le document.
- une classe Livre qui est une implémentation de Document. On pourra la laisser vide ici.
- une classe LivreJeunesse qui est une implémentation de DocumentJeunesse et une extension de Livre.
- une classe GuideVoyage qui est une extension de Livre. On pourra la laisser vide ici.

Question 9. Introduisez les classes et interfaces de documents sous leur forme minimale.

Question 10. Définissez une classe paramétrée pour les pochettes de documents, disposant d'une chaîne de caractère décrivant le thème de la pochette, et d'une collection permettant de stocker les documents ainsi que de quelques méthodes : une méthode d'ajout de documents (cette méthode peut jeter une exception si l'ajout viole une règle régissant les pochettes, nous verrons de telles règles plus loin), une méthode toString (la chaîne retournée contiendra le thème et le nombre de documents de la pochette), une méthode retournant le nombre de documents de la pochette, et un constructeur paramétré créant une pochette de contenu vide mais avec un thème.

Question 11. Commencez à mettre en place une classe de test. Préparez quelques instances de documents, et de pochettes de différents types. Vérifiez que, dès la compilation, vous ne pouvez pas par exemple ajouter un livre jeunesse dans une pochette de guides de voyage, ou un guide de voyage dans une pochette de documents jeunesse. Vérifiez en revanche que vous pouvez ajouter différents types de documents dans une pochette de documents.

Question 12. Mettez maintenant en place une classe spécifique pour les pochettes de documents jeunesse. Cette classe sera générique (mais ne pourra contenir que des documents jeunesse). Elle sera munie d'un âge conseillé pour emprunter la pochette. Lors de l'ajout de documents dans la pochette, on vérifiera que l'âge conseillé pour le document ajouté est inférieur ou égal à l'âge conseillé de la pochette. Dans le cas contraire, l'ajout n'aura pas lieu et lèvera une exception. Complétez votre classe de test.

Question 13. Munissez la classe Pochette d'une méthode permettant de transférer les documents de la pochette courante vers une autre pochette de documents passée en paramètre. On n'acceptera pas n'importe quelle pochette en paramètre : son type de documents devra être suceptible d'accueillir des documents du type de documents de l'instance courante. Par exemple, on ne devra pas accepter de transférer une pochette de documents vers une pochette de guides de voyages, mais par contre on peut transférer les documents d'une pochette de livres de voyage vers une pochette de documents.

Question 14. Peut-on ranger une Pochette<LivreJeunesse> dans une variable de type Pochette<Livre> ? Peut-on ranger une PochetteJeunesse<LivreJeunesse> dans une variable de type Pochette<Livre> ?

Exercice 5 *Paramétrage contraint (Des couples de toutes sortes)*

Question 15. Définissez une interface Mâle et Femelle vides. Définissez des classes d'animaux mâles et femelles implémentant l'interface appropriée (par exemple, Taureau, Vache, Dauphin, Dauphine, ...). Munissez ces classes d'une méthode toString(), retournant par exemple un nom correspondant à la classe.

Définissez, en dérivant la classe Paire vue en cours, la classe générique CoupleConventionnel qui représente les couples constitués d'un mâle et d'une femelle.

Instanciez-la pour créer des couples d'animaux (taureau et vache, dauphin et dauphine etc...). Remarquez que l'on peut ainsi créer des couples d'espèces différentes (par exemple, constitués d'un dauphin et d'une vache).

Question 16. Proposez, en dérivant la classe Paire vue en cours, une classe générique CoupleEspèce pour représenter les couples constitués de deux membres de la même espèce, mais pas forcément de sexes opposés. Réfléchissez à la façon de représenter la notion de membre d'une espèce. Instanciez-la. Vous ne devez plus pouvoir créer des couples constitués d'animaux d'espèce différente (un dauphin et une vache).

Question 17. Proposez une classe CoupleFertile pour représenter les couples constitués d'un mâle et d'une femelle de la même espèce. Réfléchissez aux différentes solutions envisageables.

Instanciez la classe CoupleFertile. Vous ne devez plus pouvoir créer des couples constitués de deux mâles ou deux femelles, même s'ils sont de la même espèce, ni des couples constitués d'animaux d'espèces différentes, même s'ils sont de sexe différent.