Université Montpellier II **2012/2013**
Programmation par objets en Java (GLIN 505)

TD/TP 4
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'élément en sommet de pile et dépile
  int nbElements();
  A sommet(); // retourne le sommet de pile mais ne le dépile pas
}
```

1. Ecrivez 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).

- **Si vous ne connaissez pas encore les interfaces,** vous pouvez remplacer cette interface IPile par une classe abstraite APile dont toutes les méthodes sont publiques et abstraites. La classe CPile *hérite* alors de la classe abstraite APile (au lieu d'*implémenter* l'interface IPile).

2. Ecrivez 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*. Proposez une solution basée sur la généricité.

```
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.trieBulles();
		obj.affiche();
	}
}
```

Exercice 3 (extrait examen 2007/2008)

Question 1. 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(contenu.size()-1);
			 contenu.remove(contenu.size()-1);}
			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 2. 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 3. 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).
Ne recopiez pas la classe Personne : il vous suffit d'indiquer les modifications apportées.

Question 4. Ecrivez un programme dans lequel on déclare une file d'attente (avec priorité) de personnes et rangez-y quelques personnes.

Exercice 4* : paramétrage contraint (Des couples de toutes sortes)

Question 1.

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 2.

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 3. 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.

Annexe : la classe LinkedList

java.util

Class LinkedList<E>

java.lang.Object
  └ java.util.AbstractCollection<E>
    └ java.util.AbstractList<E>
      └ java.util.AbstractSequentialList<E>
        └ **java.util.LinkedList<E>**

Type Parameters:

E - the type of elements held in this collection

Quelques méthodes publiques :

`LinkedList()`
Constructs an empty list.

`void addFirst(E o)`
Inserts the given element at the beginning of this list.

`E element()`
Retrieves, but does not remove, the head (first element) of this list.
Throws:
`NoSuchElementException` - if this queue is empty.

`E getFirst()`
Returns the first element in this list.
Throws:
`NoSuchElementException` - if this list is empty.

`E peek()`
Retrieves, but does not remove, the head (first element) of this list.
Returns `null` if this queue is empty.

`E removeFirst()`
Removes and returns the first element from this list.
Throws:
`NoSuchElementException` - if this list is empty.

`int size()`
Returns the number of elements in this list.