IUT d'Orsay - DUT Informatique - Jean-Claude MARTIN

JEAN-CLAUDE.MARTIN@U-PSUD.FR

Programmation Orientée Objet (POO)

EX 11 : Programmation Générique

Ex 1

Ecrire une classe générique Triplet permettant de manipuler des triplets d'objets d'un même type avec :

- constructeur avec trois arguments
- 3 méthodes getPremier(), getSecond(), getTroisieme()
- une méthode afficher() qui affiche les 3 éléments du triplet

Vous testerez cette classe dans le main d'une classe TestTriplet en créant un triplet de 3 entiers puis un triplet de 3 String et en appelant toutes les méthodes sur ces triplets.

Essayez dans le main de créer un même triplet avec comme éléments : un Integer, un Double et une String. Que se passe-t-il ?

……
……...

Ex 2

Créer une autre classe TripletX qui permet cette fois de gérer trois éléments qui peuvent être de types différents. Tester aussi cette classe en y mettant un main().

Ex 3

Dans le programme suivant repérer les erreurs de compilation et/ou d'exécution.
Tapez et tester ce programme. Mettez en commentaire les lignes provoquant des erreurs et ajouter un commentaire pour expliquer donner et expliquer le message d'erreur.

```
class C<T> {
```

```
        T x ;
        T[] t1 ;
        T[] t2 ;
        public static T inf ;
        public static int compte ;
        void f () {
                x = new T() ;
                t2 = t1 ;
                t2 = new T [5] ;
        }
}
```

Ex 4

Ecrire une méthode générique fournissant en retour un objet tiré au hasard dans un tableau fourni en argument. Testez le main() suivant qui utilisera votre méthode hasard avec un tableau de Integer puis un tableau de String.

```
  public static void main(String args[])
  {
        Integer[] tabi = {1,  7, 8, 4, 9} ;  // ici boxing automatique
        System.out.println ("hasard sur tabi = " + hasard (tabi) ) ;

        String[] tabs = {"Java", "C", "C++", "C#", "Visual Basic"} ;
        System.out.println ("hasard sur tabs = " + hasard (tabs) ) ;
  }
```

Ex 5

A l'exécution, les paramètres de type sont effacés et sont remplacés par Object.
Cela ne permet plus de contraindre par exemple que deux paramètres sont du même type générique dès la compilation.

Par exemple, la méthode générique suivante qui retourne au hasard un de ses deux arguments qui *doivent être du même type* :

```
      public static <T> T hasard (T x, T y) {
                  double v = Math.random () ;
                  if (v < 0.5) return x ;
                  else return y ;
            }
```

Est en fait traduite par le compilateur :

```
       public static Object hasard (Object x, Object y) {
```

```
        double v = Math.random () ;
        if (v < 0.5) return x ;
        else return y ;
    }
```

Vous pouvez dès la compilation, imposer que les deux arguments soient de même type en précisant le nom de la classe où la méthode générique est définie et appliquée :

nomClasse<Type>.nomMethode(arguments)

Par exemple : MethGen2arg.<Integer> hasard (i1, i2)

Complétez la méthode main() suivante :

```
public class MethGen2arg
{
    public static void main (String args[]) {
        Integer i1 = 3 ; Integer i2 = 5 ;
        String s1 = "Salut" ; String s2 = "bonjour" ;

        // Appel et affichage du résultat de la méthode hasard avec 2 Integer
        ...

        // Appel et affichage du résultat de la méthode hasard avec 2 String
        ...

        // Appel et affichage du résultat de hasard avec 1 String et 1 Integer
        // L'appel suivant est ……..à la compilation
        System.out.println ("hasard (i1, s1) = " + hasard (i1, s1)) ;

        // Les appels suivants seront …….à la compilation
        MethGen2arg.<Integer> hasard (i1, s1) ;
        MethGen2arg.<String>  hasard (i1, s1) ;
        MethGen2arg.<Integer> hasard (i1, i2) ;
        MethGen2arg.<Number>  hasard (i1, i2) ;
}

    public static <T> T hasard (T x, T y) {
        double v = Math.random () ;
        if (v < 0.5) return x ;
        else return y ;
    }
}
```

Ex 6

On dispose de la classe générique suivante :

```
public class Couple<T> {
    private T x, y ;
```

```
	public Couple (T premier, T second) {
		x = premier ;
		y = second ;
	}
	public void affiche () {
		System.out.println("1ere valeur : " + x + " - 2eme
valeur : " + y);
	}
}
```

1) Créer une sous-classe CoupleNomme permettant de manipuler des couples analogues à ceux de la classe Couple<T>, mais possédant en plus un nom de type String. On redéfinira les méthodes de cette nouvelle classe en réutilisant les méthodes de la classe Couple.
2) Créer une autre sous-classe nommée PointNomme (qui ne sera pas générique) dans laquelle les éléments du couple sont de type Integer et le nom de type String.
3) Ecrire une méthode main() testant ces classes