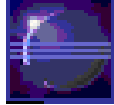
	TP Généricité en JAVA	
---	--	---

Objectifs	Temps alloué	Outils
- S'initier au concept de généricité.	3h00	Eclipse

Exercice 1 : introduction à la généricité

- 1) Définir la classe générique Point dont les attributs abs et ord sont de type générique T1. (Constructeur, Getters et Setters et toString)
- 2) Construire dans la classe Test 2 Points : p1 manipule des données de type entier et p2 manipule des données de type double. (Faire l'affichage de chacun des points)
- 3) Définir la classe non générique PointColoréNG qui hérite de la classe Point les attributs abs et ord de type entier et qui ajoute l'attribut « couleur » de type String.
- 4) Créer un Point coloré non générique nommé « pcng1 » et afficher ses caractéristiques.
- 5) Définir la classe générique PointColoré qui hérite de la classe générique Point et qui définit en plus des attributs génériques hérités, un attribut générique nommé « couleur » de type T2. (Constructeur, Getters et Setters et toString)
- 6) Créer un Point coloré pc1 qui manipule des attributs (abs et ord) de type entier et la couleur de type String. Afficher ses caractéristiques.
- 7) Créer un deuxième point coloré pc2 qui manipule des données (abs et ord) de type double et dont la couleur est défini grâce à la classe « CouleurRVB » que vous devez implémentez.
 - a. La classe « CouleurRVB » définit 3 attribut de types entiers : rouge, vert et bleu. (Constructeur, Getters et Setters et toString)

- 8) Définir une classe Cercle caractérisées par son rayon de type entier et son Centre de type Point générique. (Constructeur, Getters et Setters et toString)
- 9) Créer un cercle c1 de rayon 50 et de centre le point p1.
- 10) Définir la classe « CercleAvecRestriction » qui définit le rayon du cercle de type entier et le centre du cercle dont le type peut être n'importe que classe qui hérite de la classe Point.
- 11) Créez le cercleAvecRestriction car1 de rayon 100 et de centre p2
- 12) Créez le cercleAvecRestriction car2 de rayon 30 et de centre pc2

Exercice 2 : Pile générique en représentation chaînée

Vous disposez du code suivant représentant l'implémentation d'une pile

```
// interface Pile
interface Pile {
    public boolean estVide();
    public Object dernier();
    public void depiler();
    public void empiler(Object o);
}
/* Le recours à une interface peut s'expliquer par la possibilité après
de faire
le choix de la représentation physique voulue (chaînée ou contigüe) */
// Représentation chaînée : listes de Object
class Noeud {
    Object info;
    Noeud suivant;
}
// implémenter les Pile avec des listes (représentation chaînée)
class PileListe implements Pile{
    private Noeud sommet;
    public PileListe(){
        sommet = null;
    }
    public boolean estVide(){
        return (sommet == null);
    }
    public Object dernier(){
        return sommet.info;
    }
    public void empiler(Object o){
        Noeud n = new Noeud();
        n.info = o;
        n.suivant = sommet;
        sommet = n;
    }
    public void depiler(){
        sommet = sommet.suivant;
    }
}
```

```

class TestPile{
    public static void main(String[] args){
        PileListe p = new PileListe();
        for(int i = 0 ; i < 10 ; i++)
            p.empiler(new Integer(i));
        while(!p.estVide()){
            System.out.println((Integer) p.dernier());
            p.depiler();
        }
        // Tester aussi p.empiler("L'entier " + i);
        /* System.out.println((Integer) p.dernier());
=> Erreur à l'exécution de casting
=> System.out.println(p.dernier());
=> Ouverture sur les exceptions */
/* Remplacer while(!p.estVide()) par for(int i = 0 ; i < 20 ; i++)
=> Erreur à l'exécution lors du dépilement d'une pile vide
=> Ouverture sur les exceptions mais surtout sur la généricité */
    }
}

```

Donnez la version générique de l'implémentation de la pile