

Tutoriel Généricité en Java

1) Sans généricité

```
public class CoupleSansGen {

    private Object a;
    private Object b;

    public CoupleSansGen(Object a, Object b) {
        super();
        this.a = a;
        this.b = b;
    }

    public String toString() {
        return "CoupleSansGen [a=" + a + ", b=" + b + "]";
    }

    public Object getA() {
        return a;
    }

    public void setA(Object a) {
        this.a = a;
    }

    public Object getB() {
        return b;
    }

    public void setB(Object b) {
        this.b = b;
    }

    public static void main(String[] args) {
        CoupleSansGen c =new CoupleSansGen("généricité","java");

        String ch=(String) c.getA();//Obligation de faire un transtypage

        double d=(double) c.getB(); //Pas de contrôle de type
        //l'erreur n'apparaît qu'à l'exécution sous forme d'une exception

    }
}
```

2) Déclaration d'une classe générique

```
public class Couple <T>{
    private T a;
    private T b;
    public Couple() {
        super();
    }
    public Couple(T a, T b) {
        super();
        this.a = a;
        this.b = b;
    }
    public T getA() {
        return a;
    }
    public void setA(T a) {
        this.a = a;
    }
    public T getB() {
        return b;
    }
    public void setB(T b) {
        this.b = b;
    }
}
```

Utilisation d'une classe générique

```
import java.util.Date;

public class Test {

    public static void main(String[] args) {
        Couple <Integer> ci=new Couple<Integer>(3,5);
        Couple <Date> cd=new Couple<Date>(new Date(),new Date());
        System.out.println(ci.getA());
        System.out.println(ci.getB());

        System.out.println(cd.getA());
        System.out.println(cd.getB());
    }
}
```

3) Classe paramétrée par plusieurs Type

```
public class Couple2Types<T,U> {

    private T a;
    private U b;

    public Couple2Types(T a, U b) {
        this.a = a;
        this.b = b;
    }

    public T getA() {
        return a;
    }

    public void setA(T a) {
        this.a = a;
    }

    public U getB() {
        return b;
    }

    public String toString() {
        return "Couple2Types [a=" + a + ", b=" + b + "]";
    }

    public void setB(U b) {
        this.b = b;
    }

    public static void main(String[] args) {
        Couple2Types<String, Integer> c1=new Couple2Types<String, Integer>("abc", 5);
        System.out.println(c1);
        Personne p1=new Personne("ahmed");
        Voiture v1=new Voiture(123456, "citroen", "rouge");
        Couple2Types<Personne, Voiture> c2=new Couple2Types<Personne, Voiture>(p1, v1);

        System.out.println(c2);

    }

}
```

4) Conséquences de l'effacement de type

- ▷ *Il n'existe qu'une classe (le type brut) partagée par toutes les instanciations du type générique*

```
public static void main(String[] args) {
    Couple <Integer> ci=new Couple<Integer>(3,5);
    Couple <Date> cd=new Couple<Date>(new Date(),new Date());

    System.out.println(ci.getClass()==cd.getClass()); //true
}
```

- ▷ *les membres statiques d'une classe paramétrée sont partagés par toutes les instanciations de celle-ci*

```
public static void main(String[] args) {

    Couple <Integer> ci=new Couple<Integer>(3,5);
    Couple <Date> cd=new Couple<Date>(new Date(),new Date());
    Couple<Double> cdb=new Couple<Double>(2.5, 7.5);

    System.out.println("Nombre de couple =" +Couple.nbCouple); //3
}
```

- ▷ *Les membres statiques d'une classe paramétrée ne peuvent pas utiliser ses paramètres de type*

```
public class Couple <T>{
    private T a;
    private T b;
    static int nbCouple=0;

    ✗ static T attStat;
    ✗ static T methode(T param) { }
```

- ▷ *au sein d'une classe paramétrée on ne peut pas utiliser les paramètre de type pour instancier un objet (simple ou tableau)*

```
public class Couple <T>{
    private T a;
    private T b;

    ✓ T x1;
    T tab[];

    void methode() {

        ✗ x1=new T();
        ✗ tab=new T[10]; }
}
```

▷ *seul le type brut est connu à l'exécution*

```
public static void main(String[] args) {  
    Couple <Integer> ci=new Couple<Integer>(3,5);  
    Couple <Date> cd=new Couple<Date>(new Date(),new Date());  
    Couple<Double> cdb=new Couple<Double>(2.5, 7.5);  
  
    System.out.println(ci instanceof Couple); //true  
    System.out.println(cd instanceof Couple); //true  
    System.out.println(cdb instanceof Couple); //true  
}
```

▷ *on ne peut utiliser un type générique instancié dans un contexte de vérification de type*

```
public static void main(String[] args) {  
    Couple <Integer> ci=new Couple<Integer>(3,5);  
    Couple <Date> cd=new Couple<Date>(new Date(),new Date());  
  
    ✗ if ( ci instanceof Couple<Doule>)
```

▷ *on ne peut pas instancier de tableaux d'un type générique*

```
public static void main(String[] args) {  
    Couple <Integer> ci=new Couple<Integer>(3,5);  
    Couple <Integer> tab;  
  
    ✗ tab=new Couple <Integer> [100];
```

5) Généricité & Héritage

Une classe dérivée d'une classe générique peut elle-même être générique ou pas.

a) Classe dérivée générique

```
public class Triplet <T,T2>extends Couple<T> {
    private T2 c;

    public T2 getC() {
        return c;
    }

    public void setC(T2 c) {
        this.c = c;
    }

    public Triplet() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Triplet(T a, T b, T2 c) {
        super(a, b);
        this.c = c;
    }
}
```

```
import java.util.Date;

public class Test2 {

    public static void main(String[] args) {
        Triplet<String, Double> t =new Triplet<>("a", "b", 5.5);
        System.out.println(t.getA());
        System.out.println(t.getB());
        System.out.println(t.getC());

        Triplet<Date, Float> t2=new Triplet<Date, Float>(new Date(),
new Date(), 7.0f);
        System.out.println(t2.getA());
        System.out.println(t2.getB());
        System.out.println(t2.getC());
    }
}
```

b) Classe dérivée NON générique

```
import java.util.Date;
public class TripletNG extends Couple<Date>{

}
```

6) Méthodes génériques

```
public class Calcul<U> {  
    private U x;  
    public <T>boolean comparer(T a, T b){  
        return a.equals(b);  
    }  
}
```

```
public class Personne {  
    private Long id;  
    String nom;  
    //String prenom;  
    public boolean equals(Object p){  
        return (this.nom==((Personne)p).nom);  
    }  
    public Personne(String nom) {  
        super();  
        this.nom = nom;  
    }  
    @Override  
    public String toString() {  
        return "Personne [nom=" + nom + " ]";  
    }  
}
```

```
import java.util.Date;  
public class Test3 {  
  
    public static void main(String[] args) {  
        Calcul<String> cal=new Calcul<String>();  
        double a=5.0;  
        double b=7.5;  
        System.out.println(cal.comparer(a, b));  
  
        String c="azerty";  
        String d="azerty";  
        System.out.println(cal.comparer(c, d));  
  
        Personne n1=new Personne("ahmed");  
        Personne n2=new Personne("ahmed");  
        System.out.println(cal.comparer(n1, n2));  
    }  
}
```

```
false  
true  
true
```

7) Restriction sur les types

```
public class Traitement <T extends Personne> {
    private T a;
    private T b;
    public Traitement(T a, T b) {
        super();
        this.a = a;
        this.b = b;
    }
    public Traitement() {
        super();
    }
    public T getA() {
        return a;
    }
    public void setA(T a) {
        this.a = a;
    }
    public T getB() {
        return b;
    }
    public void setB(T b) {
        this.b = b;
    }
}
```

```
public class Test4 {
    public static void main(String[] args) {
        //Traitement<Date> t =new Traitement<>();
        //faux car la classe traitement n'accepte pas n importe quel
type.
        //Elle accepte seulement les types dérivés de Personne.

        Traitement<Etudiant> t =new Traitement<>(new
Etudiant("ahmed", "DSI"),new Etudiant("aymen", "RSI"));
        System.out.println(t.getA());
        System.out.println(t.getB());
    }
}
```

8) Interface Métier

```
import java.util.List;
public interface IMetier <T,U>{
    public T save(T o);
    public List<T> getAll();
    public T findOne(U id);
    public void update(T o);
    public void delete(U id);
}
```



```
import java.util.List;
public class MetierPersonneImpl implements IMetier<Personne, Long> {

    @Override
    public Personne save(Personne o) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public List<Personne> getAll() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Personne findOne(Long id) {
        // TODO Auto-generated method stub
        return null;
    }
1
    @Override
    public void update(Personne o) {
        // TODO Auto-generated method stub

    }

    @Override
    public void delete(Long id) {
        // TODO Auto-generated method stub

    }

}
```