Méthodes génériques et classe générique

```
public class GenMetod {
     public static  void displayInt(int[] elt,int size){
       for (int i=0; i<size;i++) {
          System.out.println(elt[i]);
       }
   }

   public static  void displayString(String[] elt,int
size){
       for (int i=0; i<size;i++) {
          System.out.println(elt[i]);
       }
   }

   public static void main(String args []) {
       int s=5;
       int[] t;
       String[] c;
       t = new int[s];
       c = new String[s];
       for (int i=0; i<s;i++) {
          t[i]=i+1;
       }

       for (int i=0; i<s;i++) {
          c[i]="a" + i;
       }

      displayInt(t,s);
      displayString(c,s);
   }

}
```

```
public class Gen {

   public static <T> void display(T[] elt,int size){
      for (int i=0; i<size;i++) {
         System.out.println(elt[i]);
      }
   }

   public static void main(String args []) {
      int s=5;
      Integer[] t;
      String[] c;
      t = new Integer[s];
      c = new String[s];
      for (int i=0; i<s;i++) {
         t[i]=i+1;
      }

      for (int i=0; i<s;i++) {
         c[i]="a" + i;
      }

     display(t,s);
     display(c,s);
   }
}
```

```
class StackFullException extends RuntimeException {

  public StackFullException(){
     super();
  }

  public StackFullException(String message){
     super(message);
  }
}

/**Exception to indicate that Stack is empty */

class StackEmptyException extends RuntimeException {

  public StackEmptyException(){
     super();
  }
  public StackEmptyException(String message){
     super(message);
  }
}

/** Stack class(generic type)*/

class Stack<T> {
  private int size;
  private T[] stackAr;
  private int top; // top of stack

  @SuppressWarnings("unchecked")
  public Stack(int size) {
       this.size = size;
       stackAr = (T[])new Object[size]; //Creation of Generic Stack Array
       top = -1; // initialize Stack to with -1
  }

  public void push(T value){
       if(isFull()){
            throw new StackFullException("Cannot push "+value+", Stack is full");
       }
       stackAr[++top] = value;
  }

  public T pop() {
       if(isEmpty()){
            throw new StackEmptyException("Stack is empty");
       }
       return stackAr[top--]; // remove item and decrement top as well.
  }

  public boolean isEmpty(){
```

```
        return (top == -1);
    }


    public boolean isFull(){
        return (top == size - 1);
    }
}

public class StackAppGeneric {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<Integer>(10); // Creation of Generic Stack
        stack.push(11);
        stack.push(21);
        stack.push(31);
        stack.push(41);
        stack.push(51);

        System.out.print("Popped items: ");
        System.out.print(stack.pop()+" ");
        System.out.print(stack.pop()+" ");
        System.out.print(stack.pop()+" ");
        System.out.print(stack.pop()+" ");
        System.out.print(stack.pop()+" ");

    }
}
```

Design Pattern

Iterator and Factory Method

1- Iterator in Java

Classe1

```
public interface Iterator {
  public boolean hasNext();
   public Object next();
 }
```

Classe2

```
public interface Container {
public Iterator getIterator();
}
```

Classe3

```
public class NameRepository implements Container {
      public String names[] = {"Ali" , "Mohamed" ,"Samir" , "Salah"};
           @Override
           public Iterator getIterator() {
           return new NameIterator();
           }

private class NameIterator implements Iterator {
   int index;
           @Override
           public boolean hasNext() {
           if(index < names.length){
                    return true; }
                    return false;
           }

      @Override
           public Object next() {
           if(this.hasNext()){
                    return names[index++]; }
         return null; }
           }
}
```

Classe4

```
public class IteratorPatternDemo {

  public static void main(String[] args) {
    NameRepository namesRepository = new NameRepository();

    for(Iterator iter = namesRepository.getIterator(); iter.hasNext();){
```

```
        String name = (String)iter.next();
        System.out.println("Name : " + name);
      }
    }
}
```

2- Collection in java

```
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Set;
import java.util.HashSet;
import java.util.TreeSet;

public class IterTest {
   public static void main(String[] args) {
      List <Integer> lis1 = new ArrayList<Integer>();
      List <Integer> lis2 = new LinkedList<Integer>();
      Set <Integer> set1 = new HashSet<Integer>();
      Set <Integer> set2 = new TreeSet<Integer>();

      for (int i=0; i<6; i++)
         lis1.add(i);

      for (int i=0; i<6; i++)
         lis2.add(i);

      for (int i=0; i<6; i++)
         set1.add(i);

      for (int i=0; i<6; i++)
         set2.add(i);

      Iterator <Integer> itl1 = lis1.iterator();
      Iterator <Integer> itl2 = lis2.iterator();
      Iterator <Integer> its1 = set1.iterator();
      Iterator <Integer> its2 = set2.iterator();

      System.out.println("ArrayList");
      while(itl1.hasNext()){
         System.out.print(itl1.next() + "  ");}

      System.out.println("\nLinkedList");
      while(itl2.hasNext()){
         System.out.print(itl2.next() + "  ");}
```

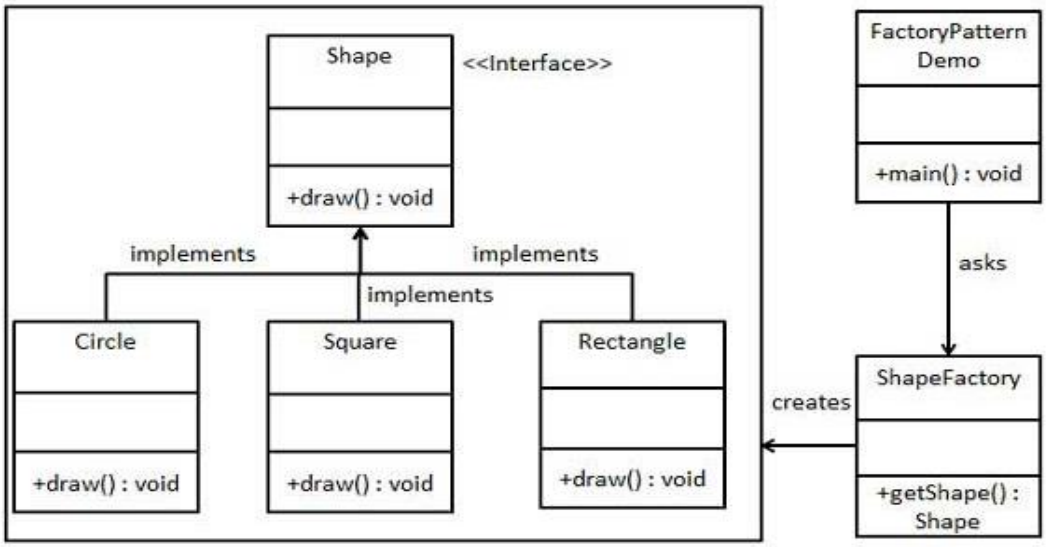
```
        System.out.println("\nHashSet");
        while(its1.hasNext()){
           System.out.print(its1.next() + "  ");}

        System.out.println("\nTreeSet");
        while(its2.hasNext()){
           System.out.print(its2.next() + "  ");}

        System.out.println("\n");
        }
}
```

3-Factory Method

Usine de formes géometriques



Shape.java

```
public interface Shape {
   void draw();
}
```

Rectangle.java

```
public class Rectangle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Square.java

```
public class Square implements Shape {
```

```
   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

Circle.java

```
public class Circle implements Shape {

   @Override
   public void draw() {
      System.out.println("Inside Circle::draw() method.");
   }
}
```

L'usine (Factory) pour la fabrication d'objects

ShapeFactory.java

```
public class ShapeFactory {

   //use getShape method to get object of type shape
   public Shape getShape(String shapeType){
      if(shapeType == null){
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE")){
         return new Circle();
      } else if(shapeType.equalsIgnoreCase("RECTANGLE")){
         return new Rectangle();
      } else if(shapeType.equalsIgnoreCase("SQUARE")){
         return new Square();
      }
      return null;
   }
}
```

Le client

FactoryPatternDemo.java

```
public class FactoryPatternDemo {

   public static void main(String[] args) {
      ShapeFactory shapeFactory = new ShapeFactory();

      //get an object of Circle and call its draw method.
      Shape shape1 = shapeFactory.getShape("CIRCLE");

      //call draw method of Circle
      shape1.draw();

      //get an object of Rectangle and call its draw method.
      Shape shape2 = shapeFactory.getShape("RECTANGLE");

      //call draw method of Rectangle
      shape2.draw();

      //get an object of Square and call its draw method.
      Shape shape3 = shapeFactory.getShape("SQUARE");

      //call draw method of circle
      shape3.draw();
   }
```

```
    }
```

Verifier le resultat

```
    Inside Circle::draw() method.
    Inside Rectangle::draw() method.
    Inside Square::draw() method.
```