为什么使用泛型：

当类型不使用泛型时，如果有N多种类型，存在类型不对应的问题。
泛型只存在于编译阶段。

泛型的使用：
①泛型类
②泛型方法
③泛型接口

①

```
class A<T> {          // 自命名，一般为T
    private T key;

    public void setKey(T key){
      this.key = key;
    }
    public T getKey(){
        return this.key;
    }
}
```

```
A<String> a1 = new A<String>();  //不指定泛型时，变量默认类型是Object
a1.setKey("xxxx");
String s = a1.getKey();
//相当类模版
//同样的类，但是在new对象时用泛型指定不同的数据类型，这些对象不能相互赋值
```

③

```
interface TB<T>{
    T test(T t);
}

class B1<T> implements TB<T>{

    @Override
    public T test(T t){
        ......
    }
}
```

```
class B2<String> implements TB<String>{
    @Override
    public String test(T, t){
        ...
    }
}
B1<String> b1 = new B1<String>();
B2 b2 = new B2();
```

②
```
class C<E>{
    private E e; // 可以被实例使用

    public <T> void test(T s){
        T t = s;
        System.out.println(this.e);
    }
    public <T> T test1(T s){
        return s;
    }
    public <T> void test2(T... strs){
      for(T s: strs){
          System.out.println(s);
      }
    }
}

C<Object> c = new C<Object>();
c.test("xxx");

Integer i = c.test1(2); // 这里调用时确定具体类型
Boolean b = c.test1(true);
```

boolean b = c.test(ls);

泛型通配符

```
package Test;

import java.util.ArrayList;
import java.util.List;

public class Fanxing {
    public static void main(String[] args) {
        Dd d =new Dd();
        List<String> l1 = new ArrayList<String>();
        d.test(l1);

        List<Integer> l2 = new ArrayList<Integer>();
        d.test(l2);

    }
}

class Dd{
    public void test(List<?> list){  // 这里的<?>是泛型通配符
                            // 不使用Object，Object仍是类型不安全
                           // 没有使用泛型类
                          // 也不是泛型方法
    }
}
```

// 假使对参数类型的要求是List，不对List的
// 泛型不作要求，只能使用<?>达到效果

<? > 任意类型

<? extends Person> Person及其子类

<? super Person> Person及其父类

<? extends Comparable> Comparable接口的实现类