1、课程名称：泛型

在 JDK 1.5 之后提供了泛型的技术支持，也是一个 Java 的新特性。

2、知识点

2.1、上次课程的主要知识点

1、 包的定义及使用
2、 访问控制权限

2.2、本次预计讲解的知识点

1、 掌握泛型的基本作用
2、 掌握泛型的通配符的使用
3、 掌握泛型接口的和泛型方法的操作

3、具体内容

3.1、泛型的引出

要求设计一个可以表示出坐标的类，（X、Y）

但是此坐标可以同时满足以下几种要求：

- x=10、y=100
- x=10.3、y=50.2
- x="东经 180 度"、y="北纬 210 度"

问，这样的坐标类该如何设计？

分析：

因为现在的程序中可以接收三种数据类型的数据，所以为了保证程序的正确性，最好使用 Object 完成，因为 Object 可以接收任意的引用数据类型：

- 数字 → 自动装箱操作 → Object
- 字符串 → Object

按照以上的特点，完成程序：

```
public class Point { // 表示坐标
    private Object x;
    private Object y;
    public Object getX() {
        return x;
    }
    public void setX(Object x) {
        this.x = x;
    }
    public Object getY() {
        return y;
    }
    public void setY(Object y) {
        this.y = y;
    }
}
```

此时，Point 类完成了。

此时，设置一个整型数字，那么来观察是否可以操作：

```
package org.lxh.genericsdemo01;
public class GenDemo01 {
    public static void main(String[] args) {
        Point p = new Point();
        p.setX(11); // int --> Integer --> Object
        p.setY(20); // int --> Integer --> Object
        int x = (Integer) p.getX(); // 取出x坐标
        int y = (Integer) p.getY();// 取出y坐标
```

```
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
}
```

此时，达到了设置整数的目的，那么下面继续完成设置小数的操作。

```
package org.lxh.genericsdemo01;
public class GenDemo02 {
    public static void main(String[] args) {
        Point p = new Point();
        p.setX(11.3f);
        p.setY(20.3f);
        float x = (Float) p.getX(); // 取出x坐标
        float y = (Float) p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
}
```

下面继续设置字符串作为 x 和 y 的坐标。

```
package org.lxh.genericsdemo01;
public class GenDemo03 {
    public static void main(String[] args) {
        Point p = new Point();
        p.setX("东经 1 8 0 度");
        p.setY("北纬 2 2 0 度");
        String x = (String) p.getX(); // 取出x坐标
        String y = (String) p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
}
```

此时，基本的功能已经完成了，但是此种操作是否存在问题呢？

在此操作之中，可以发现所有的内容都是以 Object 进行操作的，那么就意味着，可以设置任意的类型，即：X 可以是整型，Y 可以是字符串。

```
package org.lxh.genericsdemo01;
public class GenDemo04 {
    public static void main(String[] args) {
        Point p = new Point();
        p.setX(10);
        p.setY("北纬 2 2 0 度");
        int x = (Integer) p.getX(); // 取出x坐标
        int y = (Integer) p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
```

```
}
```

此时程序在编译的时候没有任何的问题，但是在执行的时候出现了以下的错误提示：

```
Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot be cast to java.lang.Integer
    at org.lxh.genericsdemo01.GenDemo04.main(GenDemo04.java:9)
```

之所以会这样，因为在程序中的所有的属性都可以向 Object 进行转换，那么此时程序的入口就显得不那么规范了，而且存在安全的漏洞。

但是，从之前所学习过的全部代码来看，此处只能应用到这里了。没有更好的方法了。

3.2、JDK 1.5 的新特性 —— 泛型

JDK 1.5 之后出现了新的技术 —— 泛型，此技术的最大特点是类中的属性的类型可以由外部决定，而且在声明类的时候应该采用如下的形式：

```
class 类名称<泛型类型,泛型类型,...>{

}
```

那么现在使用如上的操作格式来修改之前的操作类。

```
package org.lxh.genericsdemo02;
public class Point<T> { // 表示坐标
    private T x; // x属性的类型由外部决定
    private T y; // y属性的类型由外部决定
    public T getX() {
        return x;
    }
    public void setX(T x) {
        this.x = x;
    }
    public T getY() {
        return y;
    }
    public void setY(T y) {
        this.y = y;
    }
}
```

此时，程序中加入了泛型操作之后，可以发现一切的操作类型此时都不再由程序固定设置，而是由实例化对象的时候在外部进行了指定。

```
package org.lxh.genericsdemo02;
public class GenDemo05 {
    public static void main(String[] args) {
        Point<Integer> p = new Point<Integer>();
        p.setX(11);
        p.setY(20);
        int x = p.getX(); // 取出x坐标
```

```
        int y = p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
}
```

发现，此时在使用 Point 类的时候，需要加入一个属性类型的声明，而且加入之后再取出属性的时候本身也变得非常容易，不用再使用向下转型了。

而且，使用上面的操作有一点最方便之处，如果此时设置的内容不是整型，那么程序中将出现错误。

```
package org.lxh.genericsdemo02;
public class GenDemo06 {
    public static void main(String[] args) {
        Point<Integer> p = new Point<Integer>();
        p.setX(11);
        p.setY("北纬 2 2 0 度");   // 错误，不能设置String类型
        int x = p.getX(); // 取出x坐标
        int y = p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
}
```

加入泛型之后，可以对程序的操作起到更加安全的目的。

3.3、泛型的其他注意点

在使用泛型操作的时候，实际上有很多小的注意点，例如：构造方法上依然可以使用泛型或者有一种称为泛型的擦除。

3.3.1、在构造方法上引用泛型

一般开发中，经常使用构造方法设置属性的内容。那么此时实际上构造方法上依然可以使用泛型的类型。

```
package org.lxh.genericsdemo03;

public class Point<T> { // 表示坐标
    private T x; // x属性的类型由外部决定
    private T y; // y属性的类型由外部决定
    public Point(T x, T y) {
        this.setX(x);
        this.setY(y);
    }
    public T getX() {
        return x;
    }
    public void setX(T x) {
```

```
        this.x = x;
    }
    public T getY() {
        return y;
    }
    public void setY(T y) {
        this.y = y;
    }
}
```

那么此时在调用的时候就需要使用构造方法设置内容，当然，设置的内容本身依然由泛型指定。

```
package org.lxh.genericsdemo03;
public class GenDemo07 {
    public static void main(String[] args) {
        Point<Integer> p = new Point<Integer>(10, 20);
        int x = p.getX(); // 取出x坐标
        int y = p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
}
```

3.3.2、擦除泛型

如果在使用的时候没有指定泛型的话，则表示擦除泛型。

泛型一旦擦出之后，将按照 Object 进行接收，以保证程序不出现任何的错误。

```
package org.lxh.genericsdemo03;
public class GenDemo08 {
    public static void main(String[] args) {
        Point p = new Point(10, 20);
        int x = (Integer) p.getX(); // 取出x坐标
        int y = (Integer) p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
        System.out.println("y的坐标是：" + y);
    }
}
```

但是在以上的操作代码中依然会存在警告信息，那么该如何去掉警告信息呢？

```
package org.lxh.genericsdemo03;
public class GenDemo08 {
    public static void main(String[] args) {
        Point<Object> p = new Point<Object>(10, 20);
        int x = (Integer) p.getX(); // 取出x坐标
        int y = (Integer) p.getY();// 取出y坐标
        System.out.println("x的坐标是：" + x);
```

```
        System.out.println("y的坐标是：" + y);
    }
}
```

但是，以上的操作虽然去掉了警告信息，但是有些多余了，而且有些有搞笑。

3.4、通配符

在泛型的操作中通配符使用较多，而且在日后的系统类库中有很多的地方都要使用这些操作

例如，现在有如下的操作代码：

```
package org.lxh.genericsdemo04;
public class Test {
    public static void main(String[] args) {
        Object obj = "hello" ;
    }
}
```

以上的语法实际上是表示进行了向上的转型操作，因为 String 是 Object 的子类，但是现在在泛型中却没有此概念。

3.4.1、?

在进行对象转型的时候可以使用自动的向转型，但是在使用泛型的时候却没有此种操作。

```
package org.lxh.genericsdemo04;
public class Point<T> { // 表示坐标
    private T x; // x属性的类型由外部决定
    private T y; // y属性的类型由外部决定
    public T getX() {
        return x;
    }
    public void setX(T x) {
        this.x = x;
    }
    public T getY() {
        return y;
    }
    public void setY(T y) {
        this.y = y;
    }
}
```

那么下面定义两个 Point 类的对象。

```
package org.lxh.genericsdemo04;
public class GenDemo09 {
    public static void main(String[] args) {
        Point<Object> p1 = new Point<Object>();
        Point<Integer> p2 = new Point<Integer>() ;
```

```
        p1 = p2 ;   // 此时无法转换
    }
}
```

此时的程序发现，根本就无法进行转换的操作。

此时的程序实际上已经不完全属于对象的转型操作了，属于一个大的类型和小的类型的划分。

例如：将“Point<Object> p1 = new Point<Object>();”表示为整个商场的全部商品，而“Point<Integer> p2 = new Point<Integer>() ;”表示每一个顾客购买的商品。如果现在执行“p1 = p2 ;”那么就意味着，本顾客所购买的商品就是商场中的全部商品。这样肯定说不同，所以不能接收。

不能使用以上的方式接收最大的影响在于方法的参数接收上。

```
package org.lxh.genericsdemo04;
public class GenDemo10 {
    public static void main(String[] args) {
        Point<Object> p1 = new Point<Object>();
        Point<Integer> p2 = new Point<Integer>();
        fun(p1) ;
        fun(p2) ;
    }
    public static void fun(Point<?> po) { // 表示，此时可以接收任意的类型
        System.out.println(po.getX());
        System.out.println(po.getY());
    }
}
```

程序中的“?”表示的是可以接收任意的泛型类型，但是只是接收输出，并不能修改。

3.4.2、泛型上限

上限就指一个的操作泛型最大的操作父类，例如，现在最大的上限设置成“Number”类型，那么此时，所能够接收到的类型只能是 Number 及其子类（Integer）。

泛型的上限通过以下的语法完成：

```
? extends 类
```

例如：在 Point 类中只能设置数字的坐标

```
public class Point<T extends Number> { // 表示坐标，最高只能是Number
    private T x; // x属性的类型由外部决定
    private T y; // y属性的类型由外部决定
    public T getX() {
        return x;
    }
    public void setX(T x) {
        this.x = x;
    }
    public T getY() {
        return y;
    }
```

```
    public void setY(T y) {
        this.y = y;
    }
}
```

以上的泛型类型明确的指出，最大的操作父类是 Number，能设置的内容只能是其子类 Integer、Float 等等。

```
package org.lxh.genericsdemo05;
public class GenDemo11 {
    public static void main(String[] args) {
        Point<Integer> p1 = new Point<Integer>() ; // 设置的是Number的子类
    }
}
```

如果，此时设置的泛型类型是字符串的话，则也会出现错误。

而且，使用泛型的上限也可以在方法上使用，例如：接受参数。

```
package org.lxh.genericsdemo05;
public class GenDemo13 {
    public static void main(String[] args) {
        Point<Integer> p2 = new Point<Integer>();
        fun(p2) ;
    }
    public static void fun(Point<? extends Number> po) { // 表示，此时可以Number的子类
        System.out.println(po.getX());
        System.out.println(po.getY());
    }
}
```

3.4.3、泛型下限

泛型的下限指的是只能设置其具体的类或者父类。设置的语法如下：

```
? super 类
```

例如，定义一个方法，此方法只能接收 String 或 Object 类型的泛型对象。

```
package org.lxh.genericsdemo06;
public class Point<T> { // 表示坐标，最高只能是Number
    private T x; // x属性的类型由外部决定
    private T y; // y属性的类型由外部决定
    public T getX() {
        return x;
    }
    public void setX(T x) {
        this.x = x;
    }
    public T getY() {
        return y;
    }
```

```
    public void setY(T y) {
        this.y = y;
    }
}
```

在方法中设置泛型的下限：

```
package org.lxh.genericsdemo06;
public class GenDemo14 {
    public static void main(String[] args) {
        Point<String> p1 = new Point<String>();
        Point<Object> p2 = new Point<Object>();
        fun(p1) ;
        fun(p2) ;
    }
    public static void fun(Point<? super String> po) { // 表示，此时可以Number的子类
        System.out.println(po.getX());
        System.out.println(po.getY());
    }
}
```

3.5、泛型接口

泛型不光可以在类上使用，还可以在接口中进行定义。操作的语法如下：

```
interface 接口名称<泛型类型,泛型类型,...>{}
```

范例：定义泛型接口

```
package org.lxh.genericsdemo07;
public interface Demo<T> { // 定义泛型接口
    public void print(T param);//   此抽象方法中使用了泛型类型
}
```

泛型接口定义完成之后，下面就需要定义子类实现此接口，实现的方法有两种

范例：第一种实现手段

```
package org.lxh.genericsdemo07;
public class DemoImpl1<T> implements Demo<T> {
    public void print(T param) {
        System.out.println("param = " + param);
    }
}
```

下面对以上的程序进行测试：

```
package org.lxh.genericsdemo07;
public class GenDemo15 {
    public static void main(String[] args) {
        Demo<String> demo = new DemoImpl1<String>() ;
        demo.print("hello") ;
    }
```

```
}
```

范例：在实现接口的时候还有第二种做法

```
package org.lxh.genericsdemo07;
public class DemoImpl2 implements Demo<DemoImpl2> {    //  设置具体类型
    public void print(DemoImpl2 param) {
        System.out.println("param = " + param);
    }
}
```

此时 print()方法中只能接收 DemoImpl2 对象实例。

```
package org.lxh.genericsdemo07;
public class GenDemo16 {
    public static void main(String[] args) {
        Demo<DemoImpl2> demo = new DemoImpl2() ;
        demo.print(new DemoImpl2()) ;
    }
}
```

3.6、泛型方法

泛型除了在类中定义之外，还可以在方法上定义，而且在方法上使用泛型，此方法所在的类不一定是泛型的操作类。

```
package org.lxh.genericsdemo08;
public class Demo {
    public <T> T print(T param){    // 定义泛型方法
        return param ;
    }
}
```

Demo 类中的 print()方法里面接收泛型的类型，而且此方法的返回值也是指定的泛型类型。下面使用以上的类进行操作。

```
package org.lxh.genericsdemo08;
public class GenDemo17 {
    public static void main(String[] args) {
        Demo d = new Demo() ;
        System.out.println(d.print(1));// 如果输入 1 表示类型是Integer
    }
}
```

当然，也可以将方法的返回值定义成一个泛型的数组。

```
package org.lxh.genericsdemo08;
public class GenDemo18 {
    public static void main(String[] args) {
        Integer i[] = fun(1, 2, 3, 4, 5, 6, 7, 8, 9);
        for (int x : i) {
            System.out.println(x);
        }
```

```
    }
    public static <T> T[] fun(T... param) {
        return param; // 返回数组
    }
}
```

3.7、泛型的嵌套的设置

现在只是突出语法，具体的操作意义需要等待后面的类库之中才能够更加明白。

```
package org.lxh.genericsdemo09;
public class Info<T> {
    private T param ;
    public T getParam() {
        return param;
    }
    public void setParam(T param) {
        this.param = param;
    }
}
```

之后定义一个 Person 类型。

```
package org.lxh.genericsdemo09;
public class Person<T> {
    private T info;
    public T getInfo() {
        return info;
    }
    public void setInfo(T info) {
        this.info = info;
    }
}
```

此时如果要将 Info 的类型设置到 Person 之中，那么同时即要指定 Person 的泛型类型，又要指定 Info 中的泛型类型。

```
package org.lxh.genericsdemo09;
public class Test {
    public static void main(String[] args) {
        Person<Info<String>> per = new Person<Info<String>>() ;
        per.setInfo(new Info<String>()) ;
        per.getInfo().setParam("mldnjava") ;
        System.out.println(per.getInfo().getParam()) ;
    }
}
```

以上的操作在后面将会有所应用。

3.8、泛型的操作范例

现在有如下的题目要求：

要求设计一个程序，定义一个 Person 类，Person 类中要存放具体的信息，但是信息分为基本信息或联系方式等等，那么此时该如何设计呢？

此时最好的设计是需要定义一个表示信息的操作标准。但是此时这个标准肯定使用接口实现，但是现在在接口中并不编写任何的操作。

```
package org.lxh.genericsdemo10;
public interface Info {
}
```

此接口没有任何的操作代码，所以，此种接口在设计上称为标示接口，表示一种能力。

之后定义 Person 类，Person 类中的信息只能由 Info 的子类决定，所以此时指定了上限。

```
package org.lxh.genericsdemo10;
public class Person<T extends Info> {
    private T info ;
    public T getInfo() {
        return info;
    }
    public void setInfo(T info) {
        this.info = info;
    }
}
```

以上的操作类中，能设置的内容只能是 Info 的子类。

```
package org.lxh.genericsdemo10;
public class Basic implements Info {
    private String name;
    private int age;
    public Basic() {
        super();
    }
    public Basic(String name, int age) {
        super();
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
```

```
    }
    public void setAge(int age) {
        this.age = age;
    }
    public String toString() {
        return "人的信息：" + "\n" + "\t|- 姓名：" + this.getName() + "\n" + "\t|- 年龄："
                + this.getAge();
    }
}
```

以上只是基本信息，但是在人中还有联系方式的子类。

```
package org.lxh.genericsdemo10;
public class Contact implements Info {
    private String address;
    private String zipcode;
    public Contact() {
        super();
    }
    public Contact(String address, String zipcode) {
        super();
        this.address = address;
        this.zipcode = zipcode;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public String getZipcode() {
        return zipcode;
    }
    public void setZipcode(String zipcode) {
        this.zipcode = zipcode;
    }
    public String toString() {
        return "地址信息：" + "\n" + "\t|- 地址：" + this.getAddress() + "\n"
                + "\t|- 邮编：" + this.getZipcode();
    }
}
```

下面，使用基本信息完成操作

```
package org.lxh.genericsdemo10;
public class TestPerson1 {
    public static void main(String[] args) {
        Person<Basic> per = new Person<Basic>();
```

```
        per.setInfo(new Basic("张三", 30));
        System.out.println(per.getInfo());
    }
}
```

下面，使用地址信息完成操作

```
package org.lxh.genericsdemo10;
public class TestPerson2 {
    public static void main(String[] args) {
        Person<Contact> per = new Person<Contact>();
        per.setInfo(new Contact("北京市", "100088"));
        System.out.println(per.getInfo());
    }
}
```

以上的 Person 中的信息属性只能是 Info 的子类，从而保证了操作的正确性。

4、总结

1、 本章只是阐述了泛型的基本操作及基本语法
2、 可以通过“?”、“? extends 类”、“? super 类”指定泛型的操作界限
3、 泛型如果没有设置的话，则将会进行擦除，擦除之后按照 Object 进行接收
4、 泛型可以在接口上设置，指定泛型接口的子类需要明确的给出操作类型

5、预习任务

1、 Java 的常用类库
- StringBuffer、System、Runtime、Date、SimpleDateFormat

6、作业

1、 全面的复写面向对象的各个知识
2、 泛型的操作语法及目的要掌握