java泛型（Generic）

一、什么叫泛型

泛型：

字面意思：广泛的类型，各种各样的类型。

专业定义：参数化的类型。

参数：方法的参数仅仅是一个声明，可以接收指定类型的任意数据。

参数化的类型：将类型变成参数，可以传不同的类型。

二、java中的泛型

2.1 泛型的概念：

泛型是jdk1.5的新特性。

> 泛型，即“参数化类型”。一提到参数，最熟悉的就是定义方法时有形参，然后调用此方法时传递实参。那么参数化类型怎么理解呢？顾名思义，就是将类型由原来的具体的类型参数化，类似于方法中的变量参数，此时类型也定义成参数形式（可以称之为类型形参），然后在使用/调用时传入具体的类型（类型实参）。

泛型的本质是为了参数化类型（在不创建新的类型的情况下，通过泛型指定的不同类型来控制形参具体限制的类型）。也就是说在泛型使用过程中，操作的数据类型被指定为一个参数。

泛型可以使用在类、接口和方法中，分别被称为泛型类、泛型接口、泛型方法。

2.2 泛型的作用

1、 类型安全。

泛型的主要目标是提高 Java 程序的类型安全。通过知道使用泛型定义的变量的类型限制，编译器可以在一个

高得多的程度上验证类型假设。

2、 消除强制类型转换。

泛型的一个附带好处是，消除源代码中的许多强制类型转换。这使得代码更加可读，并且减少了出错机会。

3、 潜在的性能收益。

泛型为较大的优化带来可能。在泛型的初始实现中，编译器将强制类型转换（没有泛型的话，程序员会指定

这些强制类型转换）插入生成的字节码中。

三、泛型类

语法：

```
class 类名称 <泛型标识：可以随便写任意标识号，标识指定的泛型的类型>{
  private 泛型标识 /*（成员变量类型）*/ var;
  .....

  }
}
```

实例：

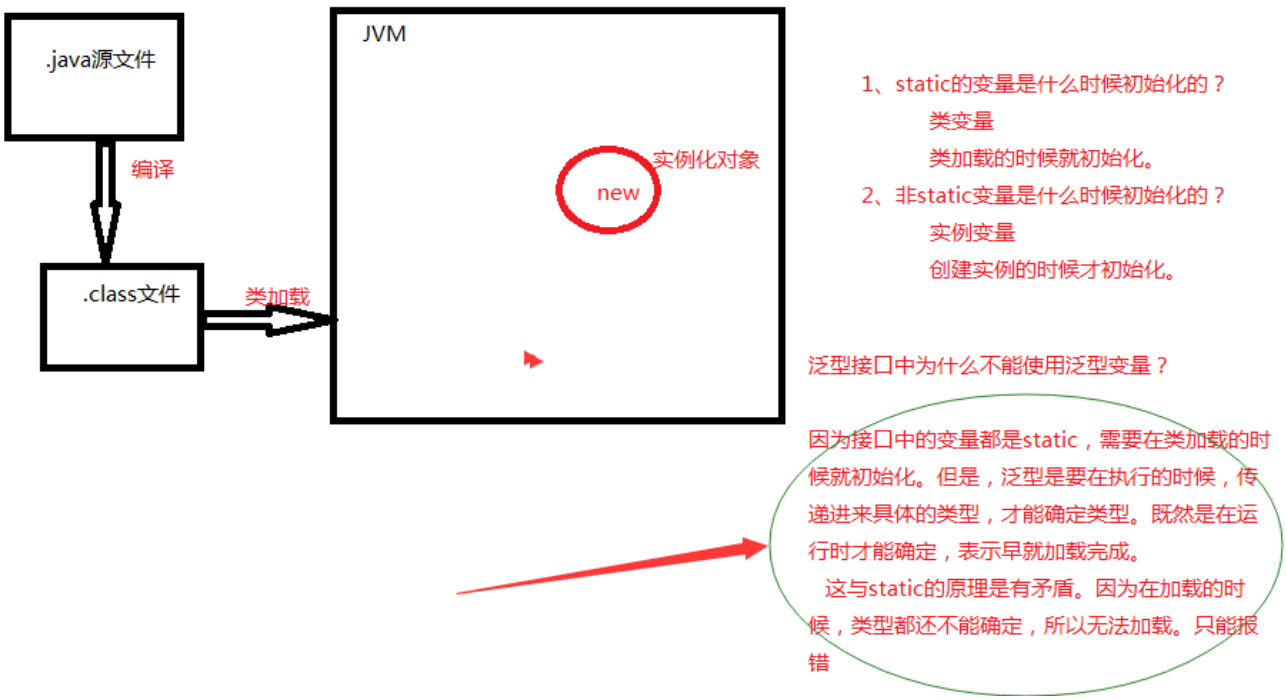
```
//此处T可以随便写为任意标识，常见的如T、E、K、V等形式的参数常用于表示泛型
//T：Type，类型
//E: Element 元素
//K：key
//V：Value
//在实例化泛型类时，必须指定T的具体类型
public class Generic<T>{
    //key这个成员变量的类型为T,T的类型由外部指定
    private T key;

    public Generic(T key) { //泛型构造方法形参key的类型也为T，T的类型由外部指定
        this.key = key;
    }

    public T getKey(){ //泛型方法getKey的返回值类型为T，T的类型由外部指定
        return key;
    }
}
```

四、泛型接口

在定义泛型接口的时候，其内不能定义泛型属性。



实例：

```
//接口
public interface IGenericDemo<T> {

    T fun(T var);

}
```

```
//实现类
public class GenericDemoImpl<E> implements IGenericDemo<String> {

    @Override
    public String fun(String var) {
        return null;
    }

}
```

五、泛型的使用细节

5.1 多泛型变量

```
//当需要使用多个泛型变量的时候，使用,隔开即可
public class GenericDemo<T, A, B> {
```

```
    private T var;
    private A a;
    private B b;

    public T getVar() {
        return var;
    }

    public void setVar(T var) {
        this.var = var;
    }

    public A getA() {
        return a;
    }

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

    public B getB() {
        return b;
    }

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

5.2 泛型方法

顾名思义：使用了泛型的方法

第一种形式：使用类定义的泛型变量

```
public void add(T entiy){

}
```

这是一种不标准的泛型方法。甚至有些资料也认为这不能叫做泛型方法。

第二种方式：方法自己定义泛型变量

```
public class Demo {
    public static void main(String[] args) {
        Student stu = new Student();
        stu.setName("mazi");

        //泛型方法的调用
        //也可以不加<String>，如果不加，默认为Object
```

```
        Demo.<Student>add(stu);

    }

    //泛型方法
    public static <E> E name(E e) {
        System.out.println(e);
        return e;
    }

    //泛型方法
    //list中可以使用泛型方法定义的泛型变量
    public static <E> List<E> query(){
        return null;
    }

    public static <E> void add(E entiy) {

    }
}
```

六、泛型通配符

```
public void showKeyValue1(Generic<Number> obj){
    System.out.println("泛型测试","key value is " + obj.getKey());
}

上述方法，接收一个Generic<Number>类型的参数。
Integer 是 Number类型的子类。

当我们调用这个方法，传入参数：Generic<Integer>
编译会报错。

原因是：
        Generic<Number>   此时Number不是形参，是一个实参。
    就像我们使用List<Number>一样，这个Number是给List定义的时候的泛型变量E传入的实参。

    所以，在泛型中，与面向对象的多态是有区别的。多态的时候，是父类类型，可以接受子类类型。
    道理是一样，但是，此处：是实参，而不是形参。
```

要解决这个问题：

```
使用通配符
public void showKeyValue1(Generic<?> obj){
   System.out.println("泛型测试","key value is " + obj.getKey());
}
```

List<?> 表示可以接收任意类型的List变量。

通配符的上界和下界：

<? extends Number> 上界，限制只能接受Number类型以及他的子类

<? super Number> 下届，限制只能接收Number类型以及他的父类。

七、作业

学生管理系统

学生

ID、姓名、年龄、性别、所在班级（班级编号）、手机号码、地址

班级

ID、班级名称、班主任、专业

用户

ID、用户名、密码

功能：

1、注册和登陆

2、班级和学生各自的CURD

3、统计各班级的人员信息

要求显示：

班级名称 总人数 男生 女生