泛型接口、泛型类

Java增加泛型支持是为了让集合记住其元素的数据类型。在没有泛型之前，如果把一个对象放进集合，集合会忘记对象的数据类型，而统一当成Object类型处理，当程序从集合中取出来就需要进行强制类型转换。

增加了泛型后的集合，可以记住其元素的类型，并且可以在编译的时候检查集合中元素的类型。

泛型入门

如果定义一个只想保存String类型的List集合，使用泛型后，如果向集合添加Integer对象，则编译时会出现错误。

Java引入了参数化类型的概念，允许程序在创建集合时指定集合元素的类型，Java的参数化类型被称为泛型。

例如:`List<String> str = new ArrayList<String>();`，就创建了一个只可以保存字符串类型的List集合。

我们还可以去掉后面ArrayList构造器中的指定类型，即`List<String> str = new ArrayList<>();`和上述例子是一样的，因为Java会推断后面尖括号里是什么类型。这种语法被称为菱形语法。

需要说明的是，如果使用var定义变量，那么就不能使用菱形语法。

定义泛型接口、类

泛型，就是允许在定义类、接口和方法时使用类型形参，这个类型形参(或者叫做泛型)将在声明变量，创建对象、调用方法时动态指定(即传入实际类型参数，称为类型实参）。

例子

```
public interface List<E> {
    //E可做为类型使用
    //下面方法可以使用E作为参数类型
   void add(E x);
    Iterator<E> iterator();
}

public interface Iterator<E> {
    E next();
    boolean hasNext();
}
//定义该接口时使用了两个泛型形参
public interface Map<K, V> {
    //在该接口中K,V完全可以作为类型使用
    Set<K> keySet();
    V put(K key, V value);
}
```

上面定义了三个带泛型的接口，这就是泛型实质：允许在定义接口、类时声明泛型形参，泛型形参在整个接口、整个类体中可以被当成类型使用，几乎所有可使用普通类型的地方都可以使用泛型。

注意：包含泛型声明的类型可以在定义变量、创建对象时传入类型实参，从而可以动态产生无数个逻辑子类，但这种子类在物理上并不存在。

可以为任何类、接口定义泛型声明，并不是只有集合类才可以使用泛型，虽然集合类是泛型的重要使用场所。

注意：当创建带泛型的自定义类时，为该类定义构造器时，构造器名还是原来类名，不要增加泛型声明，例如定义了一个Apple类，构造器名仍为Apple，但是调用该构造器时要使用Apple的形式。

从泛型类派生子类

当创建带泛型声明接口、父类之后，可以为接口创建实现类，或子类继承父类，此时，接口和父类不应该包含泛型形参，而应该是为形参赋一个类型实参，或者直接写接口、父类的名字，没有后面的尖括号。

例子

```
class A<T> {

        private T info;

        public A(T info) {

                this.info = info;
        }

        public void setInfo(T info) {
                this.info = info;
        }

        public T getInfo() {
                return this.info;
        }
}

class B extends A<String> {
        public B(String info) {

                super(info);
        }

        @Override
        public void setInfo(String info) {
                super.setInfo(info);
        }

        @Override
        public String getInfo() {
                return super.getInfo();
        }
}

class C extends A {

        public C(Object info) {
                super(info);
```

```
        }

        @Override
        public void setInfo(Object info) {
                super.setInfo(info);
        }

        @Override
        public Object getInfo() {
                return super.getInfo();
        }
}
```

创建B、C两个类的写法都是正确的，创建B类时为泛型形参赋了String的类型实参，则子类重写父类方法所有T的地方都替换成String，创建C类时，没有传类型实参，这称为原始类型(raw type)，Java编译器可能会发出警告，并且会被T形参全部当成Object处理。

而`class D extends A<T> {}`这样的写法是错误的。

注意：并不存在泛型类，例如系统并不会为ArrayList生成新的class文件，也不会把它当成新类处理，对于Java来说，他们都被当成同一个类处理，在内存中，占用一块内存空间。因此在静态方法、静态初始化块中或者静态变量的声明和初始化中不允许使用泛型形参。

同时instanceof运算符后不允许使用泛型类。