包装类

Java 官方提供的一组类，这组类的作用是将基本数据类型的数据封装成引用类型。

Byte、Integer、Short、Long、Float、Double、Boolean、Character

装箱与拆箱

装箱是指将基本数据类型转换为包装类对象。

拆箱是指将包装类对象转换为基本数据类型。

- 装箱

1、public Type(type value)

2、public Type(String value)/public Type(char value)

3、valueOf(type value) 静态方法，参数是基本数据类型的数据

每一个包装类都有一个 valueOf(type value) 方法

```
byte b = 1;
Byte byt = Byte.valueOf(b);
short s = 2;
Short shot = Short.valueOf(s);
int i = 3;
Integer integer = Integer.valueOf(i);
long l = 4L;
Long lon = Long.valueOf(l);
float f = 5.5f;
Float floa = Float.valueOf(f);
double d = 6.6;
Double doub = Double.valueOf(d);
boolean boo = true;
Boolean bool = Boolean.valueOf(boo);
char ch = 'J';
Character cha = Character.valueOf(ch);
```

4、valueOf(String value)/valueOf(char value) 专门为 Character 转换使用的，其他的 7 个包装类都可以使用 valueOf(String value)。

```
Byte byt = Byte.valueOf("1");
Short sho = Short.valueOf("2");
Integer integer = Integer.valueOf("3");
Long lon = Long.valueOf("4");
Float flo = Float.valueOf("5.5f");
Double dou = Double.valueOf("6.6");
Boolean boo = Boolean.valueOf("true");
Character cha = Character.valueOf('J');
```

需要注意的是 Boolean.valueOf(String value) 方法中，当 value 为 "true" 时，Boolean 的值为 true，否则，Boolean 的值为 false。

- 拆箱

1、*Value()

每个包装类都有一个 *Value() 方法，通过该方法可以将包装类转为基本数据类型。

```
Byte byt = Byte.valueOf("1");
Short sho = Short.valueOf("2");
Integer integer = Integer.valueOf("3");
Long lon = Long.valueOf("4");
Float flo = Float.valueOf("5.5f");
Double dou = Double.valueOf("6.6");
Boolean boo = Boolean.valueOf("true");
Character cha = Character.valueOf('J');

byte b = byt.byteValue();
short sh = sho.shortValue();
int i = integer.intValue();
long l = lon.longValue();
float f = flo.floatValue();
double d = dou.doubleValue();
boolean bo = boo.booleanValue();
char c = cha.charValue();
```

2、parse*(String value)

除了 Character 类以外的每一个包装类都有一个静态方法可以将字符串类型转为基本数据类型。

```
byte b = Byte.parseByte("1");
short s = Short.parseShort("2");
int i = Integer.parseInt("3");
long l = Long.parseLong("4");
float f = Float.parseFloat("5.5");
double d = Double.parseDouble("6.6");
boolean bo = Boolean.parseBoolean("true");
```

3、toString(type value)

每个包装类都有该方法，作用是将基本数据类型转为 String 类型。

```
byte b = 1;
String bstr = Byte.toString(b);
short s = 2;
String sstr = Short.toString(s);
String i = Integer.toString(3);
long l = 4L;
String lstr = Long.toString(l);
float f = 5.5f;
String fstr = Float.toString(f);
double d = 6.6;
String dstr = Double.toString(d);
boolean bo = true;
String bostr = Boolean.toString(bo);
String chstr = Character.toString('J');
```

接口

- 什么是接口?

接口是由抽象类衍生出来的一个概念，并由此产生了一种编程方式：面向接口编程。

面向接口编程就是将程序中的业务模块进行分离，以接口的形式去对接不同的业务模块。

面向接口编程的优点：当用户需求变更时，只需要切换不同的实现类，而不需要修改串联模块的接口，减少对系统的影响。

1、能够最大限度实现解耦合，降低程序的耦合性。

2、使程序易于扩展。

3、有利于程序的后期维护。

- 如何使用接口

接口在 Java 中时独立存在的一种结构，和类相似，我们需要创建一个接口文件，Java 中用 class 关键字来标识类，用 interface 来标识接口，基本语法：

```
public interface 接口名{
  public 返回值 方法名(参数列表)
}
```

接口其实就是一个抽象类，极度抽象的抽象类。

抽象类：一个类中一旦存在没有具体实现的抽象方法时，那么该类就必须定义为抽象类，同时抽象类允许存在非抽象方法。

但是接口完全不同，接口中不能存在非抽象方法，接口中必须全部是抽象方法。

因为接口中必须全部都是抽象方法，所以修饰抽象方法的关键字 abstract 可以省略。

接口中允许定义成员变量，但是有如下要求：

1、不能定义 private 和 protected 修饰的成员变量，只能定义 public 和默认访问权限修饰符修饰的成员变量。

2、接口中的成员变量在定义时就必须完成初始化。

3、接口中的成员变量都是静态常量，即可以直接通过接口访问，同时值不能被修改。

```
package com.southwind.test;

public interface MyInterface {
  public int ID = 0;
  String NAME = "张三";
  public void test();
}
```

使用接口时，不能直接实例化接口对象，而必须实例化其实现类对象，实现类本身就是一个普通的 Java 类，创建实现类的代码如下所示。

```
package com.southwind.test;

public class MyInterfaceImpl implements MyInterface {

  @Override
  public void test() {
    // TODO Auto-generated method stub

  }

}
```

通过 implements 关键字来指定实现类具体要实现的接口，在实现类的内部需要对接口的所有抽象方法进行实现，同时要求访问权限修饰符、返回值类型、方法名和参数列表必须完全一致。

接口和继承，Java 只支持单继承，但是接口可以多实现（一个实现类可以同时实现多个接口）

```
package com.southwind.test;

public interface MyInterface {
  public int ID = 0;
  String NAME = "张三";
  public void fly();
}
```

```java
package com.southwind.test;

public interface MyInterface2 {
  public void run();
}
```

```java
package com.southwind.test;

public class MyInterfaceImpl implements MyInterface,MyInterface2 {

  @Override
  public void run() {
    // TODO Auto-generated method stub
    System.out.println("实现了跑步的方法");
  }

  @Override
  public void fly() {
    // TODO Auto-generated method stub
    System.out.println("实现了飞行的方法");
  }

}
```

```java
package com.southwind.test;

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

public class Test {
  public static void main(String[] args) {
    MyInterfaceImpl myInterfaceImpl = new MyInterfaceImpl();
    myInterfaceImpl.fly();
    myInterfaceImpl.run();
  }
}
```