
# 第八章 包装器类

# 讲授思路

- 包装器类的概念
- 包装器类型与基本数据类型的转换
- 自动装箱的概念
- 自动装箱引发的一系列问题

# 包装器类

- Java中的八种基本数据类型本身只能表示一种数值，为了能将基本类型视为对象来处理,并能连接相关的方法，Java为每个基本类型提供了包装类。
- Java的八种基本数据类型对应的包装器类分别为：Byte，Short，Character，Integer，Long，Float，Double， Boolean。

# 创建包装器类型对象

- Java可以直接处理基本数据类型，但在有些情况下需要将其作为对象来处理,这时就需要将其转化为包装器类型，在一定的场合，运用Java包装器类来解决问题，能大大提高编程效率。
- 创建包装器类型对象的两种方式:
  - 构造方法：new;
    - Integer i = new Integer(1);
  - 调用包装器类型的valueOf方法。
    - Double d = Double.valueOf(3.14);
- 包装器类型对象共同的特点:
  - 对象一旦赋值，其值不能再改变。

# 包装器类型与基本数据类型的转换

- 有时也需要将包装器类型对象转换为基本数据类型

| 方法 | 返回类型 |
| --- | --- |
| byteValue() | byte |
| shortValue() | short |
| intValue() | int |
| longValue() | long |
| floatValue() | float |
| doubleValue() | double |

# 装箱、拆箱

- 在Java中一些运算或程序对数据类型是有限制的，比如：++/--操作符只能操作基本类型数据，集合中只能存放包装器类型对象等等。
- 思考：如何对Integer类型的数据进行++/--操作？

```
Integer y = new Integer (567);
int x = y.intValue();
x++;
y = new Integer(x);
```

# 自动装箱

- 手动装箱和拆箱的操作是很繁琐的，JDK1.5版本之后为了方便程序员的开发，Java提供了自动装箱机制来解决类似的麻烦。
- 实现Integer类型数据++/--操作。

```
Integer y = new Integer (567);
y++;
```

```
Integer y = new Integer (567);
int x = y.intValue();
x++;
y = new Integer(x);
```

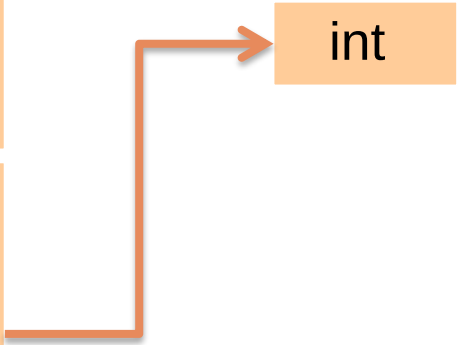
注意：y++之前和之后所引用的内存地址不同

# 基本数据类型的加宽带来的方法重载问题

- 基本数据类型会被加宽到更宽泛的基本数据类型
  - 基本数据类型和对应的包装器类型是不同的数据类型

```
public class TestOverLoad {
        void go(int x){
                System.out.println("int");
        }
        void go(Short s){
                System.out.println("Short");
        }
}
```

```
public static void main(String[] args) {
        TestOverLoad test = new TestOverLoad();
        short s = 5;
        test.go(s);
 }
```

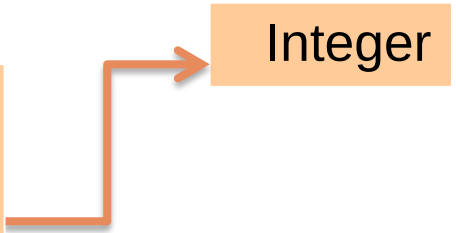


# 自动装箱带来的方法重载问题

- 自动装箱给方法的重载带来了一定的难题。

```
public class TestOverLoad{
        void go(short x){
                System.out.println("short");
        }
        void go(Integer x){
                System.out.println("Integer");
        }
}
```

```
public static void main(String[] args) {
        TestOverLoad test = new TestOverLoad();
        int i = 5;
        test.go(i);
 }
```

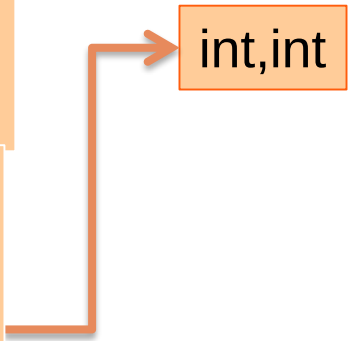


# 可变元参数列表带来的方法重载问题

- 可变元参数列表给方法的重载带来了一定的难题。

```
public class TestOverLoad {
        void go(int x, int y){
                System.out.println("int,int");
        }
       void go(short  …x){
                System.out.println("short…");
        }
        void go(Short x, Short y){
                System.out.println("Short,Short");
        }
}
```

```
 public static void main(String[] args) {
        TestOverLoad test = new TestOverLoad();
        short s1 = 5;
        short s2 = 6;
        test.go(s1,s2);
  }
```



# 重载方法调用规则

- 选择重载时匹配哪个方法时遵循以下原则：
  - 是否有直接匹配的
  - 是否加宽后直接能够匹配的
  - 是否装箱后能够匹配的
  - 先装箱后加宽后能否匹配
    - 可以先装箱，后加宽（int可以通过Integer编程Object）
  - 是否有不定长参数能够匹配的

# 总结

- 包装器类的概念
- 包装器类型与基本数据类型的转换
- 自动装箱的概念
- 自动装箱引发的一系列问题

# Thank You