2、具体内容

包装类的主要功能是针对于基本数据类型的对象转换而实现的，并且随着JDK版本的更新，包装类的功能也在放生着改变，有着更多的支持。

■认识包装类

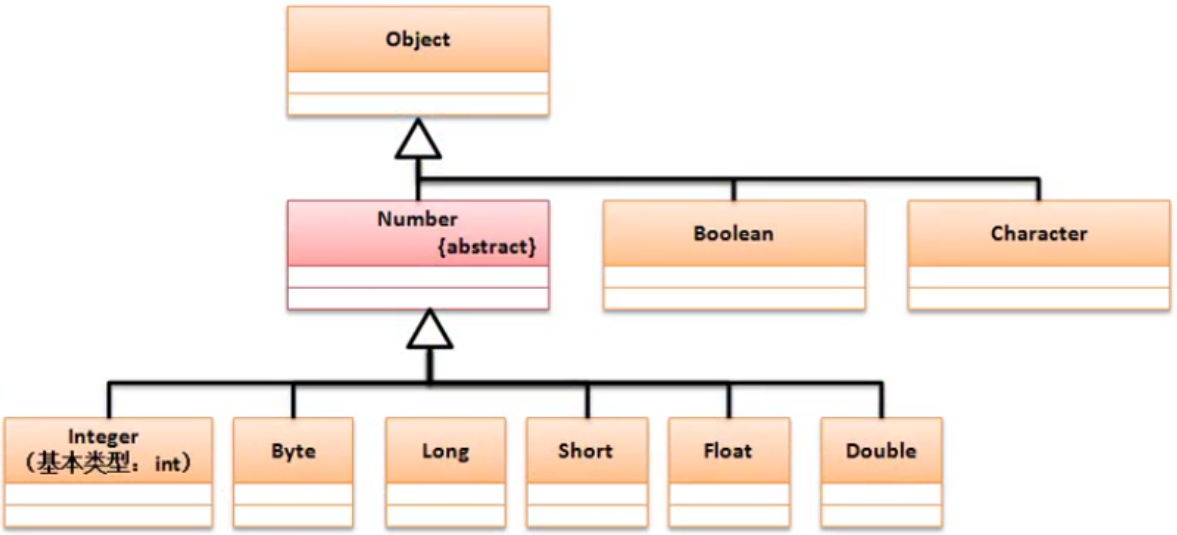
Object类最大的特点是所有类的父类，并且可以接收所有的数据类型，但是在这个过程之中就存在有个问题：基本数据类型并不是一个类，所以现在如果想要将基本数据类型以类的形式进行处理，那么就需要对其进行包装。

范例：以int数据为例实现一个包装处理的定义

```
class Int {
	private int data ; // 包装了一个基本数据类型
	public Int(int data) {
		this.data = data ;
	}
	public int intValue() {
		return this.data ;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Object obj = new Int(10) ;	// 装箱：将基本数据类型保存在包装类之中
		int x = ((Int)obj).intValue() ; // 拆箱：从包装类对象中获取基本数据类型
		System.out.println(x * 2) ;
	}
}
```

基本数据类型进行包装处理后可以像对象一样进行引用传递，同时也可以使用Object类来进行接收，但是如果我们都可以发现基本数据类型与Object类型之间的缺陷，那么JDK也早就发现了同样的问题，所以在JDK1.0的时候就提供有了包装类的概念，基本数据类型一共有八种，所以提供有八种包装类，那么这八种包装类的基本定义如下：

包装类



现在可以发现在Java中包装类一共提供有两种类型：

- 对象型包装类（Object直接子类）：Boolean、Character；
- 数值型包装类（Number直接子类）：Byte、Short、Integer、Float、Double；

Number是一个抽象类，以后只要是看见了Number都应该明确的表示它是一个数字的概念，这个类定义有如下方法：

NO.	方法名称	类型	描述
01	public byte byteValue()	普通	从包装类中获取byte数据
02	public short shortValue()	普通	从包装类中获取short数据
03	public abstract int intValue()	普通	从包装类中获取int数据
04	public abstract long longValue()	普通	从包装类中获取long数据
05	public abstract float floatValue()	普通	从包装类中获取float数据
06	public abstract double doubleValue()	普通	从包装类中获取double数据

Number类中的方法就是直接提供有获取包装类中基本数据类型的功能，一共只定义有六个方法。

■装箱与拆箱操作

基本数据类型的包装类都是为了基本数据类型转为对象提供的，这样对于基本类型与包装类之间就有了如下的操作关系：

- 数据装箱：将基本数据类型保存到包装类之中，一般可以利用构造方法完成；

|- Integer类：public Integer(int value)

|- Double类：public Double(double value)

|- Boolean类：public Boolean(boolean value)

- 数据拆箱：从包装类中获取基本数据类型：

|- 数值型包装类已经有Number类定义了拆箱的方法；

|- Boolean型：public boolean booleanValue()

范例：以int和Integer为例

```
public class JavaDemo {
	public static void main(String args[]) {
		Integer obj = new Integer(10) ;	// 装箱
		int num = obj.intValue() ; // 拆箱
		System.out.println(num * num) ;
	}
}
```

范例：以double与Double为例

```
public class JavaDemo {
	public static void main(String args[]) {
		Double obj = new Double(10.1) ;	// 装箱
		double num = obj.doubleValue() ; // 拆箱
		System.out.println(num * num) ;
	}
}
```

范例：以boolean与Boolean为例

```
public class JavaDemo {
	public static void main(String args[]) {
		Boolean obj = new Boolean(true) ; // 装箱
		Boolean num = obj.booleanValue() ; // 拆箱
		System.out.println(num) ;
	}
}
```

可以发现JDK1.9之后，对于所有包装类之中提供的构造方法都变为了过期处理，不建议用户继续使用了，这是因为从JDK1.5之后为了方便处理提供了自动的装箱与拆箱操作，所以这种手工的模式基本没人用了。

范例：观察自动装箱与拆箱

```
public class JavaDemo {
	public static void main(String args[]) {
		Integer obj = 10 ; // 自动装箱，此时不再关心构造方法了
		int num = obj ; // 自动拆箱
		obj ++ ; // 包装类对象可以直接参与数学运算
		System.out.println(num * obj) ;	// 直接参与数学运算
	}
}
```

除了提供有这种自动的数学运算支持之外，使用自动装箱最大的好处是可以实现Object接收基本数据类型的操作。

范例：Object接收小数

```
public class JavaDemo {
	public static void main(String args[]) {
		Object obj = 19.2 ; // double自动装箱为Double，向上转型为Object
		double num = (Double) obj ; // 向下转型为包装类，再自动拆箱
		System.out.println(num * 2) ;
```

```
    }
}
```

JDK1.5之后提供的自动支持功能，到了JDK1.9之后为了巩固此概念，所以将包装类的构造方法都设置为过期定义。

但是对于包装类需要注意一些问题了，关于相等判断上。

范例：观察相等判断

```
public class JavaDemo {
    public static void main(String args[]) {
        Integer x = 99 ;
        Integer y = 99 ;
        System.out.println(x == y) ;    // true
    }
}
```

```
public class JavaDemo {
    public static void main(String args[]) {
        Integer x = 127 ;
        Integer y = 127 ;
        System.out.println(x == y) ;    // false
        System.out.println(x.equals(y)) ;// true
    }
}
```

以后进行包装类相等判断的时候一定要使用equals()完成，而保障类本身也需要考虑占位的长度，如果超过了一字节的内容那么就需要使用equals()比较，如果不超过则使用“==”即可判断。