非贷款，0元入学，不1万就业不给1分钱学费，我们已干四年了！

笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

4、继承

4.10 异常

4.10.1 异常的体系

异常：是在运行时期发生的不正常情况。

在java中用类的形式对不正常情况进行了描述和封装对象。描述不正常的情况的类，就称为异常类。

1. 以前正常流程代码和问题处理代码相结合，现在将正常流程代码和问题处理代码分离，提高阅读性。
2. 其实异常就是java通过面向对象的思想将问题封装成了对象，用异常类对其进行描述。
3. 不同的问题用不同的类进行具体的描述。比如角标越界、空指针异常等等。
4. 问题很多，意味着描述的类也很多，将其共性进行向上抽取，形成了异常体系。

不正常情况分成了两大类：

Throwable：无论是error，还是异常、问题，问题发生就应该可以抛出，让调用者知道并处理。
该体系的特点就在于Throwable及其所有的子类都具有可抛性。

可抛性到底指的是什么呢？怎么体现可抛性呢？
其实是通过两个关键字来体现的：throws throw，凡是可以被这两个关键字所操作的类和对象都具备可抛性。

1. 一般不可处理的：Error
   特点：是由jvm抛出的严重性问题。
   这种问题发生，一般不针对性处理，直接修改程序。
2. 可以处理的：Exception

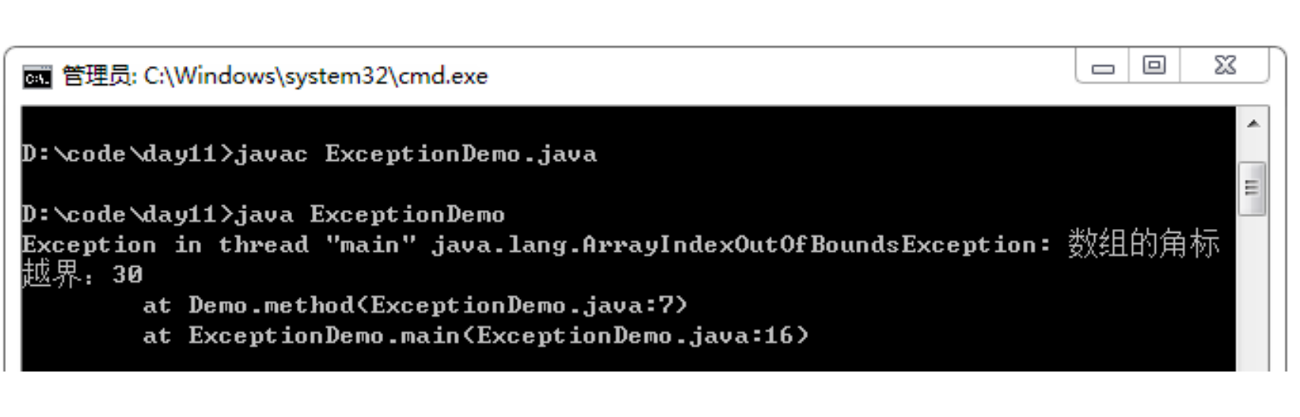
该体系的特点：
子类的后缀名都是用其父类名作为后缀，阅读性很强。

Throwable中的方法：
1. getMessage()：获取异常信息，返回字符串。
2. toString()：获取异常类名和异常信息，返回字符串。
3. printStackTrace()：获取异常类名和异常信息，以及异常出现在程序中的位置，返回值void。
4. printStackTrace(PrintStream s)：通常用该方法将异常内容保存在日志文件中，以便查阅。

示例：

```
class Demo{
        public static int method(int[] arr, int index){
              if(arr == null){
                    throw new NullPointerException("数组的引用不能为空！");
              }
              if(index >= arr.length ){
                    throw new ArrayIndexOutOfBoundsException("数组的角标越界：" + index);
              }
              return arr[index];
        }
}
class ExceptionDemo{
        public static void main(String[] args){
             int[] arr = new int[3];
             Demo.method(arr,30);
        }
}
```
复制代码

运行结果：



```
D:\code\day11>javac ExceptionDemo.java

D:\code\day11>java ExceptionDemo
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 数组的角标越界: 30
        at Demo.method(ExceptionDemo.java:7)
        at ExceptionDemo.main(ExceptionDemo.java:16)
```

4.10.2 自定义异常

可以自定义出的问题称为自定义异常。
对于角标为负数的情况，可以用负数角标异常来表示，负数角标这种异常在java中并没有定义过。
那就按照java异常的创建思想，面向对象，将负数角标进行自定义描述，并封装成对象。
这种自定义的问题描述称为自定义异常。

P.S.
如果让一个类成为异常类，必须要继承异常体系，因为只有成为异常体系的子类才有资格具备可抛性，才可以被两个关键字所操作：throws、throw。

自定义类继承Exception或者其子类，通过构造函数定义异常信息。

示例：

```
Class DemoException extends Exception
{
    DemoException(String message)
    {
        super(message);
    }
}
```
复制代码

通过throw将自定义异常抛出。

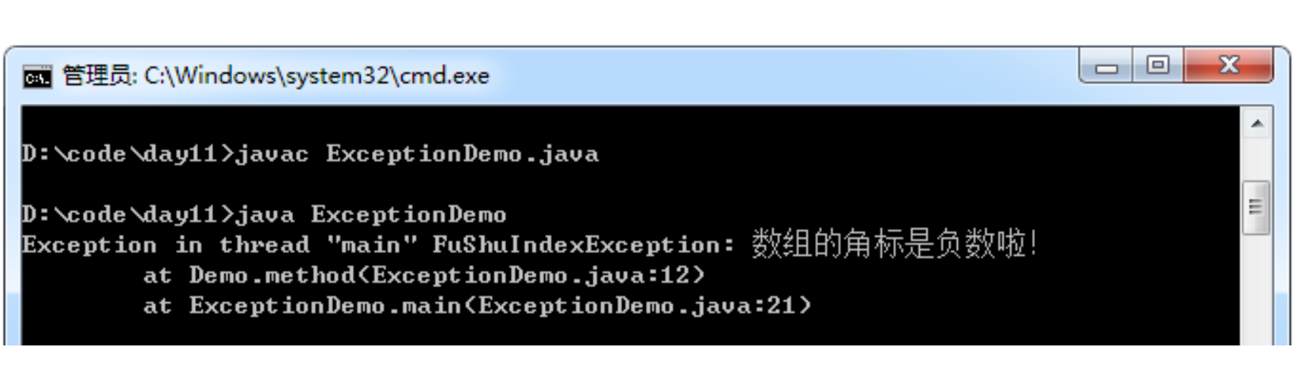
throws和throw的区别：
1. throws用于标识函数暴露出的异常类，并且可以抛出多个，用逗号分隔。throw用于抛出异常对象。
2. thorws用在函数上，后面跟异常类名。throw用在函数内，后面跟异常对象。

定义功能方法时，需要把出现的问题暴露出来让调用者去处理，那么就通过throws在函数上标识。
在功能方法内部出现某种情况，程序不能继续运行，需要进行跳转时，就用throw把异常对象抛出。

示例：

```
class FuShuIndexException extends Exception{
     FuShuIndexException(){}

     FuShuIndexException(String msg){
          super(msg);
     }
}

class Demo{
      public static int method(int[] arr, int index) throws FuShuIndexException{
           if(index < 0){
                throw new FuShuIndexException("数组的角标是负数啦！");
           }
           return arr[index];
      }
}

class ExceptionDemo{
      public static void main(String[] args) throws FuShuIndexException{
          int[] arr = new int[3];
          Demo.method(arr,-30);
      }
}
```
复制代码

运行结果：



```
D:\code\day11>javac ExceptionDemo.java

D:\code\day11>java ExceptionDemo
Exception in thread "main" FuShuIndexException: 数组的角标是负数啦!
        at Demo.method(ExceptionDemo.java:12)
        at ExceptionDemo.main(ExceptionDemo.java:21)
```

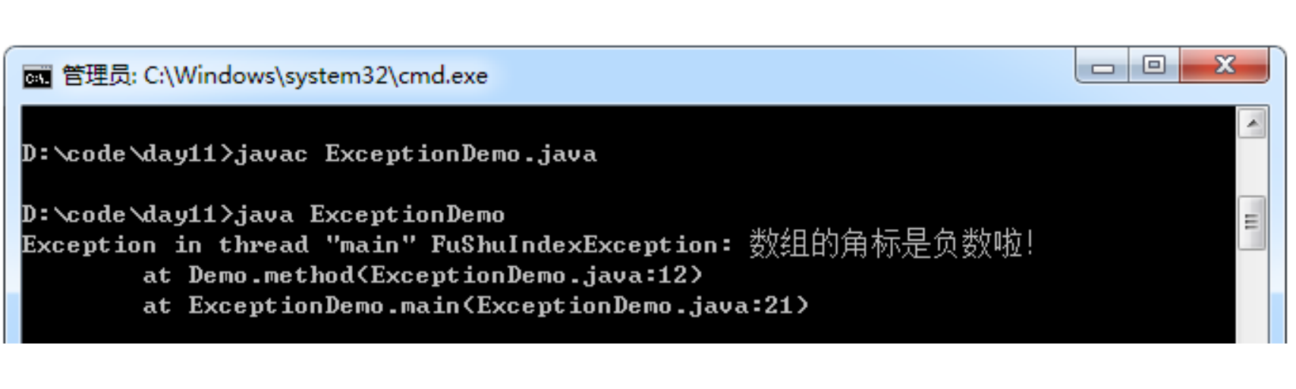
异常的分类：
1. 编译时被检测异常：只要是Exception和其子类都是，除了特殊子类RuntimeException体系。
   这种问题一旦出现，希望在编译时就进行检测，让这种问题有对应的处理方式。
   这样的问题都可以针对性的处理。

2. 编译时不检测异常（运行时异常）：就是Exception中的RuntimeException和其子类。
   这种问题的发生，无法让功能继续，运算无法运行，更多是因为调用的原因导致的或者引发了内部状态的改变导致的。
   那么这种问题一般不处理，直接编译通过，在运行时，让调用者调用时的程序强制停止，让调用者对代码进行调整。
   所以自定义异常时，要么继承Exception，要么继承RuntimeException。

示例：

```
class FuShuIndexException extends RuntimeException{
     FuShuIndexException(){}

     FuShuIndexException(String msg){
          super(msg);
     }
}

class Demo{
      public static int method(int[] arr, int index){//RuntimeException没有必要用throws抛出，并不是必须要处理
           if(index < 0){
                throw new FuShuIndexException("数组的角标是负数啦！");
           }
           return arr[index];
      }
}
```
复制代码

运行结果：



```
D:\code\day11>javac ExceptionDemo.java

D:\code\day11>java ExceptionDemo
Exception in thread "main" FuShuIndexException: 数组的角标是负数啦!
        at Demo.method(ExceptionDemo.java:12)
        at ExceptionDemo.main(ExceptionDemo.java:21)
```

P.S.
RuntimeException是那些可能在Java虚拟机正常运行期间抛出的异常的超类。
可能在执行方法期间抛出但未被捕获的RuntimeException的任何子类都无需在throws子句中进行声明。

异常处理的捕捉形式：
可以对异常进行针对性处理的方式。

具体格式是：
```
try{
    //需要被检测异常的代码。
}
catch(异常类 变量) //该变量用于接收发生的异常对象
{
    //处理异常的代码。
}
finally{
    //一定会执行的代码；
}
```

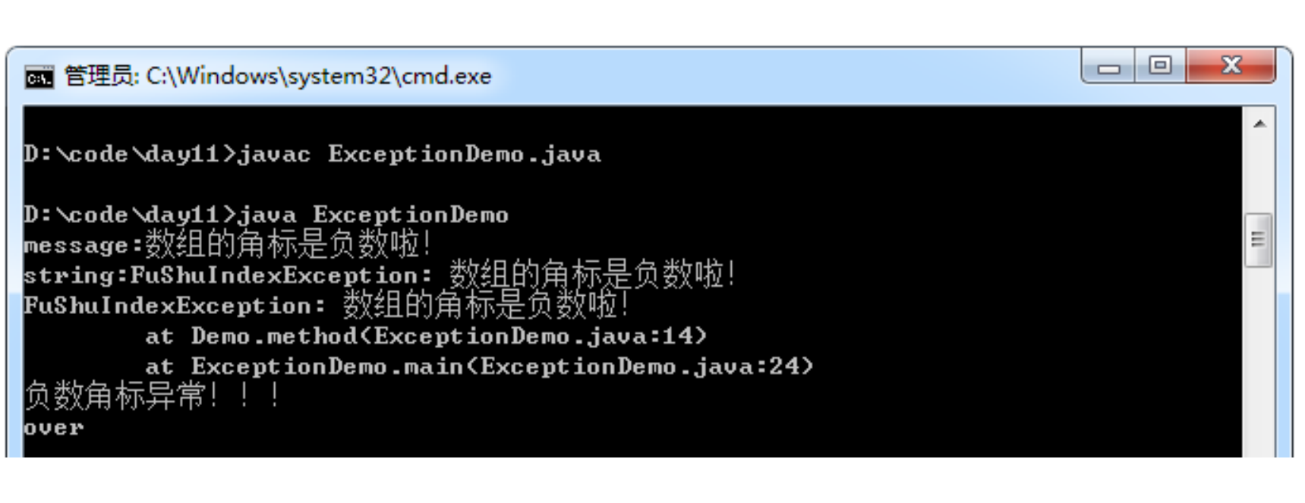
P.S.
finally代码块只有一种情况不会被执行，就是在之前执行了System.exit(0)。

处理过程：
try中检测到异常会将异常对象传递给catch，catch捕获到异常进行处理。
finally里通常用来关闭资源。比如：数据库资源，IO资源等。
需要注意：try是一个独立的代码块，在其中定义的变量只在该变量块中有效。
如果在try以外继续使用，需要在try外建立引用，在try中对其进行初始化。IO，Socket就会遇到。

示例：

```
class FuShuIndexException extends RuntimeException{
        FuShuIndexException(){}

        FuShuIndexException(String msg){
               super(msg);
        }
}

class Demo{
         public static int method(int[] arr, int index) throws NullPointerException,FuShuIndexException{
               if(arr == null)
                     throw new NullPointerException("没有任何数组实体");
               if(index < 0){
                     throw new FuShuIndexException("数组的角标是负数啦！");
              }
               return arr[index];
        }
}

class ExceptionDemo{
         public static void main(String[] args){
              int[] arr = new int[3];
              try{
                    int num = Demo.method(arr,-30);
                    System.out.println("num:" + num);
              } catch(NullPointerException e){
                    System.out.println(e);
              } catch(FuShuIndexException e){
                    System. out.println("message:" + e.getMessage());
                    System.out.println("string:" + e);
                    e.printStackTrace(); //jvm 默认的异常处理机制就是调用异常对象的这个方法。
                    System.out.println("负数角标异常！！！");
              } catch(Exception e){//Exception的catch放在最下面，先处理有针对性的异常
                    System.out.println(e);
              }
              System.out.println("over" );
        }
}
```
复制代码

运行结果：



```
D:\code\day11>javac ExceptionDemo.java

D:\code\day11>java ExceptionDemo
message:数组的角标是负数啦!
string:FuShuIndexException: 数组的角标是负数啦!
FuShuIndexException: 数组的角标是负数啦!
        at Demo.method(ExceptionDemo.java:14)
        at ExceptionDemo.main(ExceptionDemo.java:24)
负数角标异常！！！
over
```

异常处理的原则：
1. 函数内容如果抛出需要检测的异常，那么函数上必须要声明。
   否则，必须在函数内用try/catch捕捉，否则编译失败。

2. 如果调用到了声明异常的函数，要么try/catch，要么throws，否则编译失败。

3. 什么时候catch，什么时候throws呢？
   功能内容可以解决，用catch。
   解决不了，用throws告诉调用者，由调用者解决。

4. 一个功能如果抛出了多个异常，那么调用时，必须有对应多个catch进行针对性处理。
   内部有几个需要检测的异常，就抛几个异常，抛出几个，就catch几个。

示例：

```
class Demo{
      public int show(int index) throws ArrayIndexOutOfBoundsException{
           if(index < 0)
                throw new ArrayIndexOutOfBoundsException("越界啦！");
           int[] arr = new int[3];
           return arr[index];
      }
}

class ExceptionDemo{
      public static void main(String[] args){
          Demo d = new Demo();
          try{
               int num = d.show(-3);
               System.out.println("num = " + num);
           } catch(ArrayIndexOutOfBoundsException e){
               System.out.println(e.toString());
               System.exit(0);//退出jvm
           } finally{//通常用于关闭（释放）资源
               System.out.println("finally");//由于前面执行了System.exit(0);，故不会执行此语句。
           }
          System.out.println("over");
      }
}
```
复制代码

运行结果：

```
D:\code\day11>javac ExceptionDemo.java

D:\code\day11>java ExceptionDemo
java.lang.ArrayIndexOutOfBoundsException: 越界啦!
```

try catch finally 代码块组合特点：
1. try catch finally
2. try catch(多个)：当没有资源需要释放时，可以不用定义finally。
3. try finally：异常无法直接catch处理，但是资源必须关闭。

示例：

```
void show() throws Exception{
    try{
        //开启资源
        throw new Exception();
     }finally{
        //关闭资源
     }
}
```
复制代码

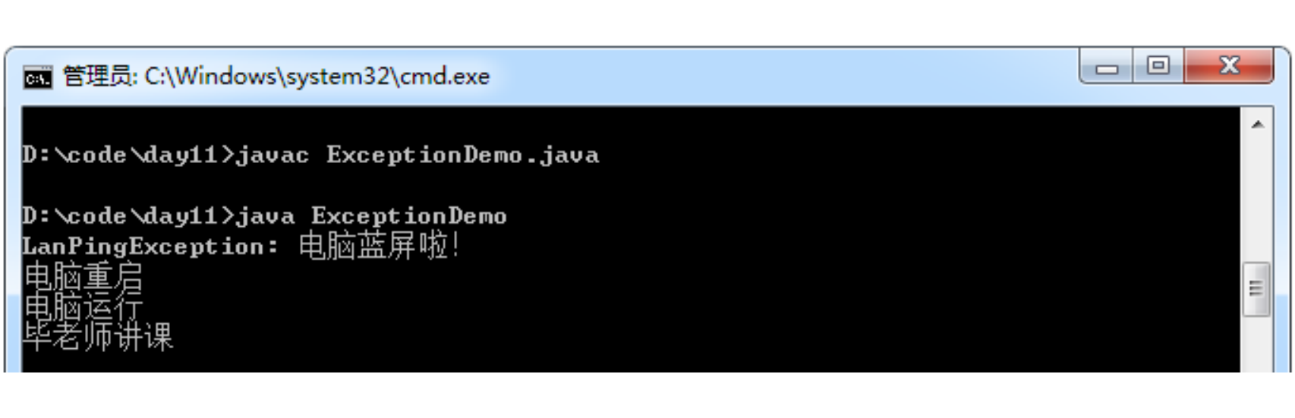
异常综合案例：

```
/*
毕老师用电脑上课。

问题领域中涉及两个对象。
毕老师，电脑。

分析其中的问题。
比如电脑蓝屏，冒烟等。
*/
class LanPingException extends Exception{
     LanPingException(String msg){
          super(msg);
     }
}

class MaoYanException extends Exception{
     MaoYanException(String msg){
          super(msg);
     }
}

class NoPlanException extends Exception{
     NoPlanException(String msg){
          super(msg);
     }
}

class Computer{
      private int state = 1;//0 2
      public void run() throws LanPingException,MaoYanException{
          if(state == 1)
               throw new LanPingException("电脑蓝屏啦！");
          if(state == 2)
               throw new MaoYanException("电脑冒烟啦！");
          System. out.println("电脑运行");
      }
```

```
    public void reset(){
        state = 0;
        System.out.println("电脑重启");
    }
}

class Teacher{
    private String name ;
    private Computer comp ;

    Teacher(String name){
         this.name = name;
         comp = new Computer();
    }

    public void prelect() throws NoPlanException{
         try{
             comp.run();
             System. out.println(name + "讲课");
         } catch(LanPingException e){
             System.out.println(e.toString());
             comp.reset();
             prelect();
         } catch(MaoYanException e){
             System. out.println(e.toString());
             test();
              //可以对电脑进行维修
              throw new NoPlanException("课时进度无法完成，原因：" + e.getMessage());
         }
    }
    public void test(){
        System.out.println("大家练习");
    }
}

class ExceptionDemo{
    public static void main(String[] args){
        Teacher t = new Teacher("毕老师");
         try{
             t.prelect();
         } catch(NoPlanException e){
             System.out.println(e.toString() + "......." );
             System.out.println("换人");
         }
    }
}
```

复制代码

运行结果：



```
D:\code\day11>javac ExceptionDemo.java

D:\code\day11>java ExceptionDemo
LanPingException: 电脑蓝屏啦!
电脑重启
电脑运行
毕老师讲课
```

异常的注意事项：

1. RuntimeException以及其子类如果在函数中被throw抛出，可以不用在函数上声明。

2. 子类在覆盖父类方法时，父类的方法如果抛出了异常，那么子类的方法只能抛出父类的异常或者该异常的子类。

3. 如果父类抛出多个异常，那么子类只能抛出父类异常的子集。

简单说：子类覆盖父类只能抛出父类的异常或者子类的子集。

P.S.

如果父类的方法没有抛出异常，那么子类覆盖时绝对不能抛，就只能try。

4.11 Object类

Object：所有类的根类。

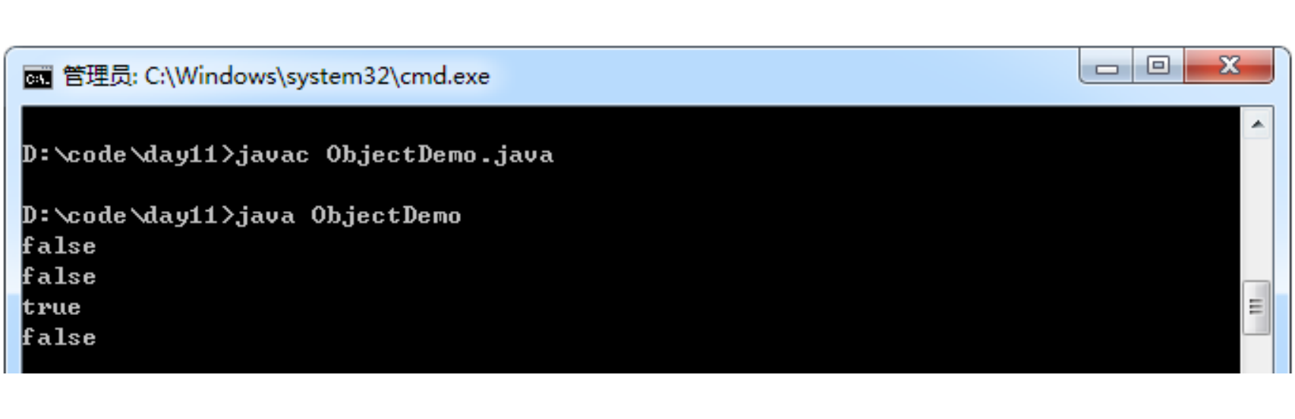
Object是不断抽取而来，具备着所有对象都具备的共性内容。

示例：

```
class Person{
    private int age ;
    Person(int age){
         this.age = age;
    }
}

class Demo{
}

class ObjectDemo{
     public static void main(String[] args){
         Person p1 = new Person(20);
         Person p2 = new Person(20);
         Person p3 = p1;

         Demo d = new Demo();

         System. out.println(p1 == p2);//false
         System. out.println(p1.equals(p2));//false
         System. out.println(p1.equals(p3));//true
         System. out.println(p1.equals(d));//false
    }
}
```

复制代码

运行结果：



```
D:\code\day11>javac ObjectDemo.java

D:\code\day11>java ObjectDemo
false
false
true
false
```

P.S.

==以及Object类的equals方法默认都是根据对象的哈希值判断两个对象是否相等。

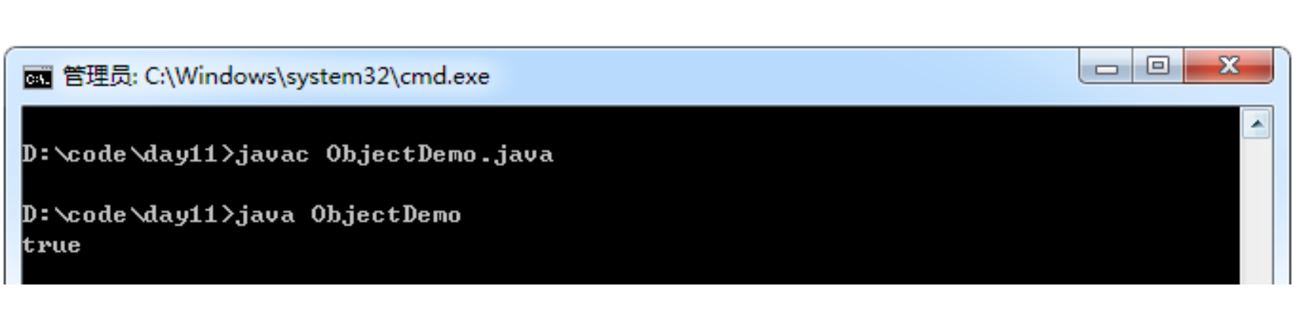
可以通过覆盖Object的equals方法来重写比较规则。

示例：

```
class Person{
       private int age ;
       Person( int age){
            this.age = age;
       }
       //比较Person的年龄，是否是同龄人
       //一般都会覆盖此方法，根据对象的特有内容，建立判断对象是否相同的依据。
       public boolean equals(Object obj){
            if(!(obj instanceof Person))
                 throw new ClassCastException("类型错误");
            Person p = (Person)obj;
            return this .age == p.age;
      }
}

class ObjectDemo{
       public static void main(String[] args){
            Person p1 = new Person(20);
            Person p2 = new Person(20);

            System. out.println(p1.equals(p2));
      }
}
```

复制代码

运行结果：



```
D:\code\day11>javac ObjectDemo.java

D:\code\day11>java ObjectDemo
true
```

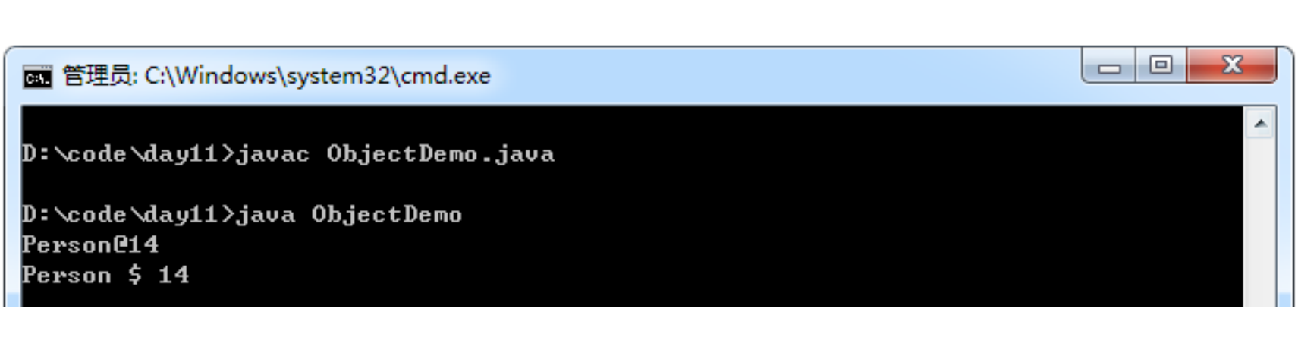
Object类的toString方法默认返回的内容是“对象所属的类名+@+对象的哈希值（十六进制）”。

示例：

```
class Person{
    private int age ;
    Person(int age){
         this.age = age;
    }
    public int hashCode(){
         return age ;
    }
}

class ObjectDemo{
     public static void main(String[] args){
         Person p1 = new Person(20);

         System. out.println(p1);
         System. out.println(p1.getClass().getName() + " $ " +
Integer.toHexString(p1.hashCode()));
    }
}
```

复制代码

运行结果：



```
D:\code\day11>javac ObjectDemo.java

D:\code\day11>java ObjectDemo
Person@14
Person $ 14
```

~END~

~爱上海，爱黑马~