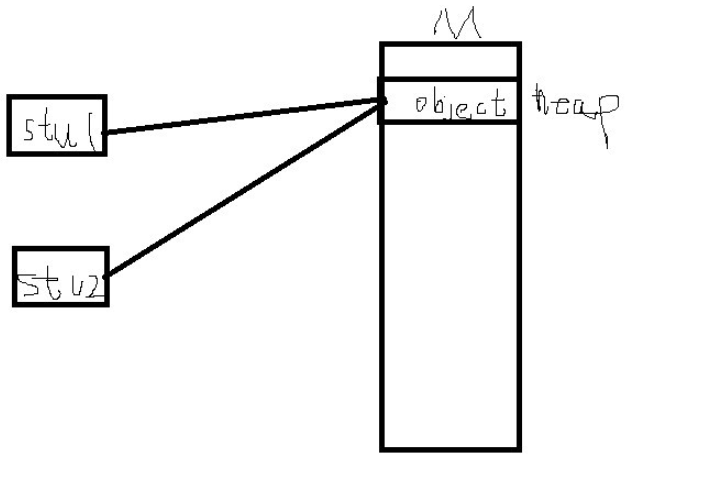
深复制和浅复制

# 1、先看看我们之前的复制

## 1、基本数据类型和对象的赋值复制

### 1、理解：基本数据类型，它复制之后，不会改变，如果是 对象，不管有没有new，新生成的改变，旧的自然会改变。 原因，原因出在(stu2 = stu1) 这一句。该语句的作用是将stu1的引用赋值给stu2，

这样，stu1和stu2指向内存堆中同一个对象。如图：



|  |
| --- |
| /\*\*  1、 = 复制  基本数据类型 不需要new，所以它复制之后，不会改变  如果是 对象，没有使用new的话，复制之后的对象如果改变，则之前的代码也会改变  \*/  @Test  **public** **void** copeObject(){    **int** a = 2;  **int** b = a;  b = 4;  System.*out*.println("a="+a+"b="+b);    Person personOne = **new** Person();  personOne.setId(123L);    //原因出在(personTwo = personOne) 这一句。该语句的作用是将personOne的引用赋值给personTwo，  //这样，personOne和personTwo指向内存堆中同一个对象。如图：  Person personTwo = personOne;      personTwo.setId(234l);;  personTwo.setName("HealerJean");  System.*out*.println("personTwo:"+personTwo.getId()+"、Name:"+personTwo.getName()); //234  System.*out*.println("personOne:"+personOne.getId()+"、Name:"+personOne.getName()); //234    System.*out*.println();  Person personThree = **new** Person();  personThree = personOne;  personThree.setId(456L);  System.*out*.println("personThree:"+personThree.getId()+"、Name:"+personThree.getName()); //456  System.*out*.println("personOne:"+personOne.getId()+"、Name:"+personOne.getName()); //456  } |

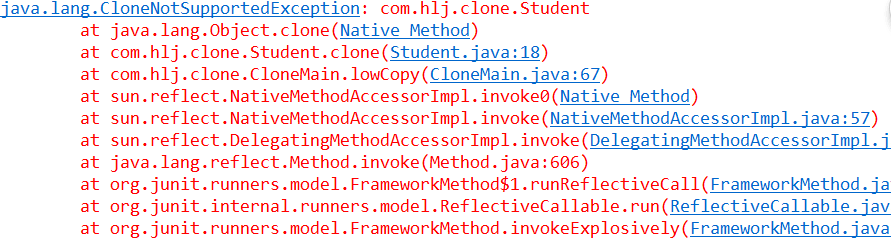
# 2、浅复制

## 1、先观察Object中的clone方法，里面的方法是Protected ，为了我们的类可以用，需要重写这个clone方法

|  |
| --- |
| **protected** **native** Object clone() **throws** CloneNotSupportedException; |

## 2、被复制的类，必须实现Cloneable接口，不实现的话在调用clone方法会抛出CloneNotSupportedException异常) 该接口为标记接口(不含任何方法)

### 举例：我没有实现Cloneable接口，在clone的时候抛出了如下异常



# 3、开始实现浅复制

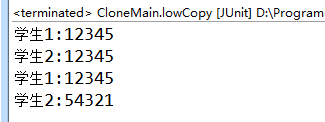
## 1、被复制的类Student

|  |
| --- |
| **class** Student **implements** Cloneable{  **private** **int** number;    **public** **int** getNumber() {  **return** number;  }    **public** **void** setNumber(**int** number) {  **this**.number = number;  }    @Override  **public** Object clone() {  Student stu = **null**;  **try**{  stu = (Student)**super**.clone();  }**catch**(CloneNotSupportedException e) {  e.printStackTrace();  }  **return** stu;  }  } |

## 2、直接开始测试

|  |
| --- |
| @Test  **public** **void** lowCopy(){  Student stu1 = **new** Student();  stu1.setNumber(12345);  Student stu2 = (Student)stu1.clone();    System.*out*.println("学生1:" + stu1.getNumber());  System.*out*.println("学生2:" + stu2.getNumber());    stu2.setNumber(54321); //这个是复制过来的，学生2改变了，学生1是不会变得，正确复制哈哈    System.*out*.println("学生1:" + stu1.getNumber());  System.*out*.println("学生2:" + stu2.getNumber());  } |

## 3.、控制台，可以看到了改了复制出来的，也不会改变之前的



# 4、浅复制 对象中引入对象，观察浅复制和深复制的区别

## 理解：

### 1、浅复制中加入对象，测试观察引出深复制

**2、浅拷贝是指在拷贝对象时，对于基本数据类型的变量会重新复制一份，而对于引用类型的变量只是对引用进行拷贝，没有对引用指向的对象进行拷贝。**

### 2、浅复制只是复制了对象addr变量的引用（也就是地址），并没有真正的开辟另一块空间，将值复制后再将引用返回给新对象。，所以当原来的addr改变的时候，复制出来的对象也会改变

### 3、为了达到真正的复制对象，而不是纯粹引用复制。需要使用深复制，观察4

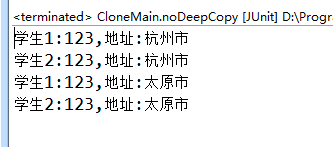
## 1、被复制的student类和address类

|  |
| --- |
| **class** AddressNo {  **private** String add;    **public** String getAdd() {  **return** add;  }    **public** **void** setAdd(String add) {  **this**.add = add;  }  }    **class** StudentNoDeep **implements** Cloneable{  **private** **int** number;    **private** AddressNo addr;    **public** AddressNo getAddr() {  **return** addr;  }    **public** **void** setAddr(AddressNo addr) {  **this**.addr = addr;  }    **public** **int** getNumber() {  **return** number;  }    **public** **void** setNumber(**int** number) {  **this**.number = number;  }    @Override  **public** Object clone() {  StudentNoDeep stu = **null**;  **try**{  stu = (StudentNoDeep)**super**.clone(); //浅复制  }**catch**(CloneNotSupportedException e) {  e.printStackTrace();  }  **return** stu;  }  } |

## 2、直接开始测试：

|  |
| --- |
| /\*\*  \*  3、浅复制中加入对象，测试观察引出深复制  浅复制只是复制了addr变量的引用，并没有真正的开辟另一块空间，将值复制后再将引用返回给新对象。  所以，为了达到真正的复制对象，而不是纯粹引用复制。  我们需要将Address类可复制化，并且修改Sudent中的clone方法 具体观察4  \*/  @Test  **public** **void** noDeepCopy(){    AddressNo addr = **new** AddressNo();  addr.setAdd("杭州市");  StudentNoDeep stu1 = **new** StudentNoDeep();  stu1.setNumber(123);  stu1.setAddr(addr);    StudentNoDeep stu2 = (StudentNoDeep)stu1.clone();    System.*out*.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());  System.*out*.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());      //同一个地址addr 改变了内容，下面的stu1和stu2都会改变  addr.setAdd("太原市");    //下面是新开辟的地址，所以会改变，但是都会各自改变自己的东西。不会互相影响  // stu2.setAddr(getAddressNo());  // stu1.setAddr(getAddressNo());  System.*out*.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());  System.*out*.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());  }  **public** AddressNo getAddressNo(){  AddressNo addr = **new** AddressNo();  addr.setAdd("太原市");  **return** addr;  } |

## 3、控制台



# 5、深复制

## 理解：

### 而深拷贝是指在拷贝对象时，同时会对引用指向的对象进行拷贝。深复制和浅复制区别就在于是否对  对象中的引用变量所指向的对象进行拷贝。

## 方法：被引入对象Address中也添加Clone复制方法和实现Cloneable接口，然后再Student的clone中克隆Address

## 1、被复制的对象

|  |
| --- |
| **class** Address **implements** Cloneable {  **private** String add;    **public** String getAdd() {  **return** add;  }    **public** **void** setAdd(String add) {  **this**.add = add;  }    @Override  **public** Object clone() {  Address addr = **null**;  **try**{  addr = (Address)**super**.clone();  }**catch**(CloneNotSupportedException e) {  e.printStackTrace();  }  **return** addr;  }  }    **class** StudentHaveDeep **implements** Cloneable{  **private** **int** number;    **private** Address addr;    **public** Address getAddr() {  **return** addr;  }    **public** **void** setAddr(Address addr) {  **this**.addr = addr;  }    **public** **int** getNumber() {  **return** number;  }    **public** **void** setNumber(**int** number) {  **this**.number = number;  }    @Override  **public** Object clone() {  StudentHaveDeep stu = **null**;  **try**{  stu = (StudentHaveDeep)**super**.clone(); //浅复制  }**catch**(CloneNotSupportedException e) {  e.printStackTrace();  }  stu.addr = (Address)addr.clone(); //深度复制  **return** stu;  }  } |

## 2、测试对象

|  |
| --- |
| /\*\*  深复制  \*/  @Test  **public** **void** deepCopy(){    Address addr = **new** Address();  addr.setAdd("杭州市");  StudentHaveDeep stu1 = **new** StudentHaveDeep();  stu1.setNumber(123);  stu1.setAddr(addr);    StudentHaveDeep stu2 = (StudentHaveDeep)stu1.clone();    System.*out*.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());  System.*out*.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());    addr.setAdd("太原市");    System.*out*.println("学生1:" + stu1.getNumber() + ",地址:" + stu1.getAddr().getAdd());  System.*out*.println("学生2:" + stu2.getNumber() + ",地址:" + stu2.getAddr().getAdd());  } |

## 3、控制台

