

类与对象3

1.1 嵌套类

○ 嵌套类

嵌套类允许一个类定义被放到另一个类定义里、一个语句块里或一个表达式内部。嵌套类是一个有用的特征，因为它们允许将逻辑上同属性的类组合到一起，并在另一个类中控制一个类的可视性。

○ 内部类（inner class）

嵌套类可以声明为static，我们将非static嵌套类称为内部类。

【例5.13】 在一个类中定义类（内部类）。

```
class RectDemo6 {  
    public static void main(String args[])  
    { double ar;  
      class RectangleR  
      { double length;  
        double width;  
        double getArea()  
        { return length * width; //返回面积      }  
        void setDim(double w, double l)  
        { width = w;    //设置长方形的大小  
          length = l;    }  
      }  
    }  
}
```

```
RectangleR rect1 = new RectangleR();  
RectangleR rect2 = new RectangleR();  
rect1.setDim(10, 20);  // 初始化每个长方形  
rect2.setDim(3, 6);  
// 调用area方法得到第一个长方形的面积  
ar = rect1.getArea();  
System.out.println("第一个面积是: " + ar);  
// 调用area方法得到第二个长方形的面积  
ar = rect2.getArea();  
System.out.println("第二个面积是: " + ar);  
}  
}
```

1.2 内部类

让宿主类提供一个
能够返回内部类
引用的方法

```
public class Parcel2 {
    class Contents {
        private int i = 11;
        public int value() { return i; }
    }
    class Destination {
        private String label;
        Destination(String whereTo) {
            label = whereTo;
        }
        String readLabel() { return label; }
    }
    public Destination to(String s) {
        return new Destination(s);
    }
    public Contents cont() {
        return new Contents();
    }
    public void ship(String dest) {
        Contents c = cont();
        Destination d = to(dest);
        System.out.println(d.readLabel());
    }
    public static void main(String[] args) {
        Parcel2 p = new Parcel2();
        p.ship("Tanzania");
        Parcel2 q = new Parcel2();
        // Defining references to inner classes:
        Parcel2.Contents c = q.cont();
        Parcel2.Destination d = q.to("Borneo");
    }
} //:~
```

1.3 内部匿名类

```
public class InnerClass
{   public void f()
    {   //必须是final
        final String str="hello";
        //内部类
        Object o=new Object()
        {   public String toString()
            {   return str;   }   };
        System.out.println(o);
    }
    public static void main(String args[])
    {   InnerClass a=new InnerClass();
        a.f();   }
}
```

注意：如果要在内部匿名类中使用外部的局部变量，变量在宣告时必须为 **"final"**

1.4 访问内部类的方法1

```
public class InnerClass
{   public class Inner
    {   public void f()
        {       System.out.println("访问非静态内部类的方法!");    }
    }
    public void visitInnerMethod()
    {   Inner i=new Inner();           在InnerClass中使用Inner类
        i.f();  }
    public static void main(String args[])
    {   InnerClass a=new InnerClass();
        a.visitInnerMethod();
        //在静态方法中访问
        Inner i=a.new Inner();
        i.f();  }
}
```

1.4 访问内部类的方法2

```
//InnerClass.java
```

```
public class InnerClass
```

```
{ public class Inner
```

```
    { public void f()
```

```
        { System.out.println("访问非静态内部类的方法!");    } }
```

```
}
```

```
//TestInnerClass.java
```

```
public class TestInnerClass
```

```
{ public static void main(String args[])
```

```
    { InnerClass ic=new InnerClass();
```

```
        InnerClass.Inner myInner=ic.new Inner(); //在InnerClass外部使用
```

```
        myInner.f();
```

```
    }
```

```
}
```


1.5 访问嵌套类的方法1

```
public class InnerClass
{   public static class Inner
    {   public void f()
        {       System.out.println("访问静态内部类的方法!");    }
    }
    public void visitInnerMethod()
    {   Inner i=new Inner();    //在InnerClass中使用Inner
        i.f();    }
    public static void main(String args[])
    {   InnerClass a=new InnerClass();
        a.visitInnerMethod();
        //在静态方法中访问
        Inner i=new Inner();
        i.f();    }
}
```

1.5 访问嵌套类的方法2

//InnerClass.java

public class InnerClass

{ public static class Inner

{ public void f()

{ System.out.println("访问静态内部类的方法!"); }

}

}

//TestInnerClass.java

public class TestInnerClass

{ public static void main(String args[])

{ InnerClass ic=new InnerClass();

InnerClass.Inner myInner=new InnerClass.Inner(); //在外部使用
myInner.f();

}

}

1.6 嵌套类中如何如何识别同名变量

```
public class Outer
{ private int size;
  public class Nested
  { private int size;
    public void doStuff(int size)
    { sieze++;
      this.size++;
      Outer.this.size++;    }
  }
}
```

1.7 嵌套类特性

- 在定义它的作用域内可以使用其类名对它进行引用；
- 在此作用域之外应使用全称（如：**Outer.Nested**）；
- 可以在方法内定义嵌套类，这种类型称局部类；
- 局部类可以访问其所在块内标记为**final**的局部变量，包括方法的参数。
- 嵌套类可以是抽象类；
- 接口也可以嵌套。这种接口可以用通常的方法实现，如果有必要的话，也可以由另一个嵌套类实现；
- 嵌套类可以访问所在类的静态成员。

2 对象克隆(clone)

○目的

为了获取对象的一份拷贝。

○标识接口：Cloneable

接口没有定义任何抽象方法，实现该接口只是通知系统该对象支持clone。

○为什么在派生类中覆盖Object的clone()方法时，一定要调用super.clone()呢？

在运行时刻，Object中的clone()识别出你要复制的是哪一个对象，然后为此对象分配空间，并进行对象的复制，将原始对象的内容一一复制到新对象的存储空间中。

2 对象克隆(clone)

java中clone的含义:

假设x是一个非空对象,应该有:

- `x.clone()`与x不相等, 说明它们不是同一个对象;
- `x.clone().getClass()`与`x.getClass()`相等, 说明它们是同一个类型class;
- `x.equals(x.clone())`为true, 逻辑上应该相当。

2 对象克隆(clone)

○实现方法:

- 1.在类定义中实现Cloneable接口;
- 2.在类中覆盖基类的clone()方法, 声明为public, 注意clone()返回为Object类型;
- 3.在类的clone()方法中, 调用super.clone();
- 4.使用clone方法时要进行强制转换。

○注意区分“浅克隆”和“深克隆”

- 1.“浅克隆”仅是对象中基本数据类型的clone;
- 2.“深克隆”还涉及对象中引用类型的clone。

2.1 浅克隆

○浅克隆：

复制对象的所有变量都含有与原来的对象相同的值，而所有的对其他对象的引用仍然指向原来的对象。换言之，浅复制仅仅复制所考虑的对象，而不复制它所引用的对象。

○浅克隆例：

2.1 浅克隆-Student.java

```
public class Student implements Cloneable  
{ String name;  
  int age;  
  Teacher teacher;  
  public Student(String name, int age, Teacher teacher)  
  {  
    this.name=name;  
    this.age=age;  
    this.teacher=teacher;  
  }  
  public Object clone() throws CloneNotSupportedException  
  {  return super.clone();    }  
}
```

2.1 浅克隆-Teacher.java

```
public class Teacher
{
    String name;
    String major;

    public Teacher(String name, String major)
    {
        this.name=name;
        this.major=major;
    }
}
```

2.1 浅克隆-TestSTClone.java

```
public class TestSTClone
{ public static void main(String[] args)
  { Teacher t=new Teacher("王选","排版");
    Student s=new Student("张三",20,t);
    try
    { Student s1=(Student)s.clone();
      s1.name="李四";
      s1.age=19;
      s1.teacher.name="萨师煊";
      s1.teacher.major="数据库";
      System.out.println(s.name);
      System.out.println(s1.name);
      System.out.println(s.teacher.name);
      System.out.println(s1.teacher.name);    }
    catch(CloneNotSupportedException e)
    { e.printStackTrace();    }
  }
}
```

2.2 深克隆

○深克隆：

被复制对象的所有变量都含有与原来的对象相同的值，除去那些引用其他对象的变量。那些引用其他对象的变量将指向被复制过的新对象，而不再是原有的那些被引用的对象。换言之，深复制把要复制的对象所引用的对象都复制了一遍。

○方法：

1.对对象中的引用类型再次克隆

2.使用序列化

○深克隆例：

2.2 深克隆-Student1.java

```
public class Student1 implements Cloneable  
{ String name;  
  int age;  
  Teacher1 teacher;  
  public Student1(String name, int age, Teacher1 teacher)  
  { this.name=name;  
    this.age=age;  
    this.teacher=teacher;  }  
  public Object clone() throws CloneNotSupportedException  
  { Student1 o=(Student1)super.clone();  
    o.teacher=(Teacher1)teacher.clone();  
    return o;  }  
}
```

2.2 深克隆-Teacher1.java

```
public class Teacher1 implements Cloneable
{
    String name;
    String major;
    public Teacher1(String name,String major)
    {
        this.name=name;
        this.major=major;
    }
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

2.2 深克隆-TestSTClone1.java

```
public class TestSTClone1
{   public static void main(String[] args)
    {   Teacher1 t=new Teacher1("王选","排版");
        Student1 s=new Student1("张三",20,t);
        try
        {   Student1 s1=(Student1)s.clone();
            s1.name="李四";
            s1.age=19;
            s1.teacher.name="萨师煊";
            s1.teacher.major="数据库";
            System.out.println(s.name);
            System.out.println(s1.name);
            System.out.println(s.teacher.name);
            System.out.println(s1.teacher.name);        }
        catch(CloneNotSupportedException e)
        {   e.printStackTrace();                        }
    }
}
```

3 单例

- 单例（Singleton）设计模式：

保证整个软件体系结构中，只存在给定类的一个实例，并且只能通过唯一的点访问该对象。

- 单例例子：

3 单例-ChinaMobile.java

```
public class ChinaMobile
{ private static ChinaMobile instance=new ChinaMobile();
  private String name;
  private String tel;
  private ChinaMobile()
  { name="中国移动";
    tel="1861";      }
  public static ChinaMobile getInstance()
  { return instance;  }
  public String getName()
  { return name;      }
  public String getTel()
  { return tel;       }
}
```

3 单例-TestSingleton.java

```
public class TestSingleton
{
    public static void main(String[] args)
    {
        ChinaMobile cm1=ChinaMobile.getInstance();
        System.out.println(cm1.getName());
        System.out.println(cm1.getTel());
    }
}
```

4 BigInteger、BigDecimal

位逻辑通常需要将一个二进制数进行左移或者右移等位运算。若需要处理64位以下的二进制数，只需要使用long或者int类型即可。

若需要处理超过64位的二进制数，那么将需要使用Java.math.BigInteger类。

```
BigInteger bi = new BigInteger("FFFFFFFFFFFFFFFF", 16);
```

BigInteger与BigDecimal类具有所有的基本算术运算方法。如加、减、乘、除，以及可能会用到的位运算如或、异或、非、左移、右移等。

4 BigInteger、BigDecimal

```
BigInteger bi=new BigInteger("888");  
BigInteger result=bi.multiply(new BigInteger("2"));  
result=bi.divide(new BigInteger("2"));  
result=bi.add(new BigInteger("232"));  
result=bi.subtract(new BigInteger("23122"));  
result=bi.shiftRight(10);  
System.out.println(result);  
System.out.println(result.intValue());  
System.out.println(result.floatValue());  
System.out.println(result.doubleValue());  
注：BigDecimal为对实数操作，方法同BigInteger。
```

4 BigInteger、BigDecimal-BigIntegerTest.java

```
import java.math.*;
public class BigIntegerTest
{ public static void testBigInteger()
    { BigInteger bi=new BigInteger("888");
      BigInteger result=bi.multiply(new BigInteger("2"));
      System.out.println(result);
      result=bi.divide(new BigInteger("2"));
      System.out.println(result);
      result=bi.add(new BigInteger("232"));
      System.out.println(result);
      result=bi.subtract(new BigInteger("23122"));
      System.out.println(result);
      result=bi.shiftRight(10);
      System.out.println(result);    }
```

4 BigInteger、BigDecimal-BigIntegerTest.java(续)

```
public static void testBigDecimal()
{   BigDecimal bi=new BigDecimal("888.888");
    BigDecimal result=bi.multiply(new BigDecimal("2.3"));
    System.out.println(result);
    result=bi.divide(new BigDecimal("90.4"), 2);
    System.out.println(result);
    result=bi.add(new BigDecimal("232.34"));
    System.out.println(result);
    result=bi.subtract(new BigDecimal("343.3434"));
    System.out.println(result);      }
public static void main(String[] args)
{   testBigInteger();
    testBigDecimal();                }
}
```

下课! 😊

Thank you!