11 特质

11.1 java.awt.Rectangle 类有两个很有用的方法 translate 和 grow,但可惜的是像 java.awt.geom.Ellipse2D 这样的类没有。在 Scala 中，你可以解决掉这个问题。定义一个 RenctangleLike 特质,加入具体的 translate 和 grow 方法。提供任何你需要用来实现的抽象方法,以便你可以像如下代码这样混入该特质:

```
val egg = new java.awt.geom.Ellipse2D.Double(5,10,20,30) with RectangleLike

egg.translate(10,-10)

egg.grow(10,20)
```

使用自身类型使得 trait 可以操作 x,y

```
import java.awt.geom.Ellipse2D




trait RectangleLike{

  this:Ellipse2D.Double=>

  def translate(x:Double,y:Double){

    this.x = x

    this.y = y

  }

  def grow(x:Double,y:Double){

    this.x += x

    this.y += y

  }
```

```
}



object Test extends App{

  val egg = new Ellipse2D.Double(5,10,20,30) with RectangleLike

  println("x = " + egg.getX + " y = " + egg.getY)

  egg.translate(10,-10)

  println("x = " + egg.getX + " y = " + egg.getY)

  egg.grow(10,20)

  println("x = " + egg.getX + " y = " + egg.getY)

}
```

11.2 通过把 scala.math.Ordered[Point]混入 java.awt.Point 的方式，定义 OrderedPoint 类。按辞典编辑方式排序，也就是说，如果 x<x'或者 x=x'且 y<y'则(x,y)<(x',y')

```
import java.awt.Point



class OrderedPoint extends Point with Ordered[Point]{

  def compare(that: Point): Int = if (this.x <= that.x && this.y < that.y) -1

                                  else if(this.x == that.x && this.y == that.y) 0

                                  else 1

}
```

11.3 查看 BitSet 类,将它的所有超类和特质绘制成一张图。忽略类型参数([…]中的所有内容)。然后给出该特质的线性化规格说明

这个略

11.4 提供一个 CryptoLogger 类，将日志消息以凯撒密码加密。缺省情况下密匙为 3，不过使用者也可以重写它。提供缺省密匙和-3 作为密匙是的使用示例

```
trait Logger{

  def log(str:String,key:Int = 3):String

}



class CryptoLogger extends Logger{



  def log(str: String, key:Int): String = {

    for ( i <- str) yield if (key >= 0) (97 + ((i - 97 + key)%26)).toChar else (97 + ((i - 97 +
26 + key)%26)).toChar

  }

}



object Test extends App{

    val plain = "chenzhen";

    println("明文为：" + plain);

    println("加密后为：" + new CryptoLogger().log(plain));

    println("加密后为：" + new CryptoLogger().log(plain,-3));

}
```

11.5 JavaBean 规范里有一种提法叫做属性变更监听器(property change listener)，这是 bean 用来通知其属性变更的标准方式。PropertyChangeSupport 类对于任何想要支持属性变更通知其属性变更监听器的 bean 而言是个便捷的超类。但可惜已有其他超类的类—比如 JComponent—必须重新实现相应的方法。将 PropertyChangeSupport 重新实现为一个特质,然后将它混入到 java.awt.Point 类中

```
import java.awt.Point

import java.beans.PropertyChangeSupport



trait PropertyChange extends PropertyChangeSupport



val p = new Point() with PropertyChange
```

11.6 在 Java AWT 类库中,我们有一个 Container 类，一个可以用于各种组件的 Component 子类。举例来说,Button 是一个 Component,但 Panel 是 Container。这是一个运转中的组合模式。Swing 有 JComponent 和 JContainer,但如果你仔细看的话，你会发现一些奇怪的细节。尽管把其他组件添加到比如 JButton 中毫无意义,JComponent 依然扩展自 Container。Swing 的设计者们理想情况下应该会更倾向于图 10-4 中的设计。但在 Java 中那是不可能的。请解释这是为什么？Scala 中如何用特质来设计出这样的效果?

Java 只能单继承,JContainer 不能同时继承自 Container 和 JComponent。Scala 可以通过特质解决这个问题.

11.7 市面上有不下数十种关于 Scala 特质的教程,用的都是些"在叫的狗"啦，"讲哲学的青蛙"啦之类的傻乎乎的例子。阅读和理解这些机巧的继承层级很乏味且对于理解问题没什么帮助,但自己设计一套继承层级就不同了,会很有启发。做一个你自己的关于特质的继承层级，要求体现出叠加在一起的特质,具体的和抽象的方法，以及具体的和抽象的字段

```
trait Fly{

  def fly(){

    println("flying")

  }
```

```
  def flywithnowing()

}



trait Walk{

  def walk(){

    println("walk")

  }

}



class Bird{

  var name:String = _

}



class BlueBird extends Bird with Fly with Walk{

  def flywithnowing() {

    println("BlueBird flywithnowing")

  }

}
```

```
object Test extends App{

  val b = new BlueBird()

  b.walk()

  b.flywithnowing()

  b.fly()

}
```

11.8 在 java.io 类库中，你可以通过 BufferedInputStream 修饰器来给输入流增加缓冲机制。用特质来重新实现缓冲。简单起见，重写 read 方法

后续 JavaIO 详细讨论

11.9 使用本章的日志生成器特质,给前一个练习中的方案增加日志功能，要求体现缓冲的效果

同上

11.10 实现一个 IterableInputStream 类，扩展 java.io.InputStream 并混入 Iterable[Byte]特质

```
import java.io.InputStream



class IterableInputStream extends InputStream with Iterable[Byte]{

  def read(): Int = 0



  def iterator: Iterator[Byte] = null

}
```