OMD – Övning vecka 4

Du förväntas göra lösningsförslag till uppgifterna före övningen, gärna tillsammans med andra kursdeltagare, och får bonus för detta om du är beredd att presentera din lösning inför klassen.

Lösningar till dessa uppgifter presenteras under övningen i läsvecka 4, dvs. 25-28 september.

Uppgifter

1 Studera javadoc-sidan för `java.util.Stack`. Kritisera designen och gör en bättre. Om du vill titta på implementeringen så finns den på `http://developer.classpath.org/doc/java/util/Stack-source.html`. Källkoden för alla paketen finns hos `http://hg.openjdk.java.net/`.

2 Klassen `java.util.Random` representerar en slumptalsgenerator som tillhandahåller metoder för att generera slumptal enligt olika fördelningar. Gör en design där varje sorts fördelning representeras av en egen klass, `UniformRandom`, `NormalRandom`, `PoissonRandom` etc. Fundera över vilka konstruerare och metoder som skall finnas.

3 Följande klass agerar både modell, vy och ”control”.

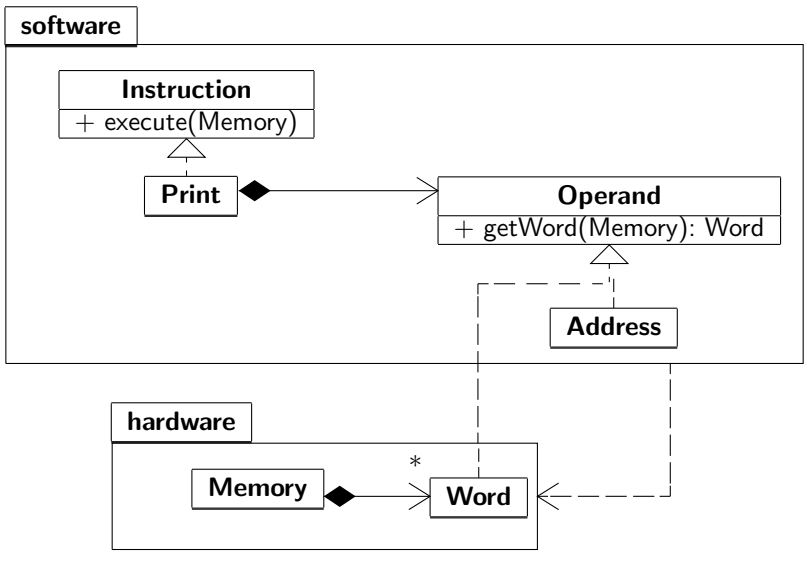
```
public class Switch extends JButton implements ActionListener {
  private boolean on = false;

  public Switch() {
    super("OFF");
    addActionListener(this);
  }

  public void actionPerformed(ActionEvent e) {
    on = !on;
    setLabel(on ? "ON" : "OFF");
  }
}
```

Gör om designen så att MVC-mönstret används med tre separata klasser med användning av *Observer*-ramverket. Lösningen redovisas med Java-kod.

4 Följande Computer-design har ett cykliskt beroende mellan paketen. Gör en ny design utan att flytta några givna klasser. Det är tillåtet att lägga till klasser och gränssnitt. Lösningen ges med klassdiagram.



5 En testklass till en simulator för en liten dator innehåller

```
public class Factorial extends Program {
    public Factorial() {
        Address n = new Address(0),
                fac = new Address(1);
        add(new Copy(new LongWord(5), n));
        // omissions
    }
}
```

Det första argumentet i konstrueraren `Copy(Word, Address)` skall ha ett argument som implementerar `Word` och är av den typ som datorns minne har. Testklassen går alltså inte att använda för en dator vars minne innehåller `VeryLongWord`. Använd mönstret *Factory method* så att man kan använda samma `Factorial`-klass för alla minnestyper som implementerar `Word`. Det argument som anger vilket tal som skall representeras av ett `Word` skall vara av typen `String`. Lösningen redovisas med ett Java-gränssnitt som alla `Word`-fabriker skall implementera och en modifierad version av `Factorial` som använder en godtycklig sådan fabrik.

6 Gör en design av ett paket för att hantera belopp i olika valutor. Uppdragsgivaren kräver att det skall vara möjligt att

- addera, subtrahera och jämföra belopp i en och samma valuta med exakt aritmetik utan att använda andra aritmetiska operationer än addition och subtraktion och utan att behöva skapa nya objekt för varje operation.
- givet ett belopp i en valuta kunna skapa ett belopp i en annan valuta med samma värde. För att kunna göra detta skall varje valuta ha en omräkningsfaktor av typ `double`.
- för varje valuta konstruera belopp baserat på de ”mynt”-slag som finns, dvs kr och ören för svensk valuta, pund, shilling och pence för en gammal brittisk valuta och lire för den gamla italienska valutan.
- konvertera ett belopp till en sträng på decimalform.

Addition, subtraktion och jämförelser av två belopp i olika valutor skall ge kompileringsfel eller kasta undantag. Lösningen presenteras med ett klassdiagram med ovannämnda valutor och alla generaliseringar, realiseringar, metoder och attribut samt en full Java-implementering av den svenska valutan.