Tentamen i Objektorienterad modellering och design

Tentamen består av 5 uppgifter med totalt 25 poäng. För godkänt betyg kommer att krävas högst 13 poäng. Vid bedömningen kommer hänsyn att tas till lösningens kvalitet. UML-diagram skall ritas i enlighet med UML-häftet. Man får förutsätta att det finns standardkonstruerare i alla klasser. De behöver ej redovisas i lösningar.

Hjälpmedel: Martin: Agile Software Development
Andersson: UML–syntax
Föreläsningsbilderna F01-06.pdf
Holm: Java snabbreferens
Java snabbreferens och UML-häftet finns att låna hos skrivningsvakten.

---

1 I en liten modell av en dator finns följande klasser.

```
public interface Instruction {
  public void execute();
}

public class Add implements Instruction {
  private IntWord word1, word2, word3;
  public void execute() {
    word3.setValue(word1.getValue() + word2.getValue());
  }
}

public class IntWord {
  private int value;
  public int getValue() {
    return value;
  }
  public void setValue(int value) {
    this.value = value;
  }
}
```

Gör om designen så att man kan använda modellen även för `LongWord` och `VeryLongWord`. I den första klassen representeras värdet med ett och i den andra med två attribut av typen `long`. Lösningen redovisas med Java-kod för klasserna `Add` och `LongWord` och eventuella tillkommande klasser och gränssnitt. Du behöver inte göra någon kod för att stödja andra aritmetiska instruktioner. (5p)

2 Konstruera ett Java-program för att representera ett aritmetiskt heltalsuttryck och beräkna värdet av det med användning av *Strategy*-mönstret. Det finns bara två sorters uttryck, heltal och binära uttryck. Ett binärt uttryck har två operander som är uttryck och en strategi som bestämmer vilken operation som skall användas på operandernas värden. Du behöver bara implementera strategier för addition och multiplikation.

Visa också koden för att konstruera uttrycket $(1 + 2) * 3$. (5p)

3 I ett kalkylprogram finns två klasser som bara skiljer sig åt i fyra rader. Klassen `LoadMenuItem` finns nedan med skillnaderna till klassen `SaveMenuItem` som kommentarer.

```
1  public class LoadMenuItem extends JMenuItem implements ActionListener {
2  //public class SaveMenuItem extends JMenuItem implements ActionListener {
3      private XL xl;
4      private Sheet sheet;
5
6      public LoadMenuItem(XL xl, Sheet sheet) {
7  //  public SaveMenuItem(XL xl, Sheet sheet) {
8          super("Load");
9  //      super("Save");
10         this.xl = xl;
11         this.sheet = sheet;
12         addActionListener(this);
13     }
14
15     public void actionPerformed(ActionEvent event) {
16         JFileChooser fileChooser = new JFileChooser(".");
17         int option = fileChooser.showOpenDialog(xl);
18 //      int option = fileChooser.showSaveDialog(xl);
19         if (option == JFileChooser.APPROVE_OPTION) {
20             File file = fileChooser.getSelectedFile();
21             sheet.load(new XLBufferedReader(file));
22 //          sheet.save(new XLPrintStream(file));
23         }
24     }
25 }
```

Använd *Template Method*-mönstret för att eliminera duplicerad kod. Lösningen redovisas med Java-kod för den gemensamma klassen och den modifierade `LoadMenuItem`-klassen. Det är tillåtet att i lösningen skriva till exempel `rad 19-21` i stället för koden på de angivna raderna. (5p)

4 I en kravspecifikation finns det något som kallas 'egenskaper' (på engelska 'features'). Det skall finnas en metod returnerar en egenskap i specifikationen om man ger dess namn. En egenskap innehåller en beskrivning som är en sträng och namnen på noll eller flera egenskaper som kallas för 'delegenskaper' ('subfeatures'). Det skall också finnas metoder för att lägga till en egenskap till specifikationen, att lägga till en delegenskap till en egenskap, och en metod `toString(String name)` som returnerar en sträng sammansatt av beskrivningen i egenskapen `name` och beskrivningarna i alla underordnade delegenskaper.

Det är tillåtet att använda klasser i `java.util` även om man inte vet exakt vad klasserna och metoderna heter.

a. Gör ett klassdiagram för en kravspecifikation med alla klasser som behövs för att representera den. Diagrammet skall visa alla relationer, attribut och de metoder med parametrar som behövs för att kunna implementera den beskrivna funktionaliteten. (3p)

b. Implementera `toString(String name)` och de metoder som den använder. Metoden skall bara användas för felsökning och behöver inte innehålla några radbyten eller andra tecken för att separera delbeskrivningarna. Man får förutsätta att alla egenskaper som efterfrågas finns. (2p)

5 Implementera den metod i kravspecifikationsklassen i uppgift **4** som lägger till en delegenskap till en egenskap. Båda egenskaperna skall anges med sina namn. En egenskap får inte innehålla sig själv som delegenskap, varken direkt eller indirekt. Om man försöker göra det skall metoden kasta ett undantag av typen

```
public class CircularException extends RuntimeException {}
```

Använd samma design-idé som i XL-projektet för att åstadkomma detta. (5p)