Tentamen i Objektorienterad modellering och design

Tentamen består av 3 uppgifter om totalt 25 poäng. För godkänt betyg kommer att krävas högst 13 poäng. Vid bedömningen kommer hänsyn att tas till lösningens kvalitet. Onödigt komplicerade lösningar kan ge poängavdrag. UML-diagram skall ritas i enlighet med UML-häftet.

Hjälpmedel: Martin: Agile Software Development
Andersson: UML–syntax
Föreläsningsbilderna F01-06,14.pdf
Holm: Java snabbreferens
Java snabbreferens och UML-häftet finns att låna hos skrivningsvakten.

---

1 I en objektorienterad modell av en dator finns bl a maskininstruktioner för att addera och multiplicera tal. En utskrift av några sådana instruktioner:

```
ADD 1 2 [1]
ADD 1.2 3.4 [2]
MUL [2] 3.4 [2]
MUL 1 2.3 [3]
```

Instruktionerna har tre operander. Operationen utföres på de två första och resultatet placeras på den adress som anges i den tredje. Det två första operanderna är heltal, flyttal eller minnesadresser och den sista är alltid en minnesadress. I exemplet skrives flyttal med decimalpunkt och minnesadresser inom hakparenteser. En minnescell kan antingen innehålla ett heltal eller ett flyttal. I modellen representeras värdet av ett heltal med `int` och flyttal med `float`. Instruktionerna skall ge exekveringsfel genom att kasta ett undantag om de två första operanderna innehåller olika typer av värden som är fallet i den sista instruktionen ovan.

a. Konstruera modeller för minnet samt de olika typerna av operander och instruktioner. Lösningen skall uppfylla *open/closed*-principen och redovisas med ett klassdiagram med alla klasser med attribut och gränssnitt. Det skall alltså vara möjligt att lägga till ytterligare instruktioner och operandtyper utan att ändra i befintlig kod. Inga metoder eller konstruerare behöver anges. (5p)

b. Utvidga modellen med metoder så att det går att exekvera instruktionerna. Eftersom resultatet av en aritmetisk operation kan ha olika representation är det tillåtet att skapa ett nytt objekt för resultatet. Använd *Template method*-mönstret för att undvika duplicerade attribut och duplicerad kod. Lösningen redovisas med Java-kod. Klasser som är snarlika behöver bara redovisas i en version. (5p)

2 En telekommunikationsleverantör erbjuder följande mobiltelefonabonnemang:

	Prata på	Till vänner	Storpratare	Max 25	Full koll
månadsavgift	29	49	599	49	189
minutavgift, eget nät	0,29	0	0	0	0,69
minutavgift, annat nät	0,29	0,69	0	0,49	0,69
öppningsavgift	0,79	0,69	0	0,99	0,99
sms	0,29	0,69	0	0	0
mms	1,99	1,99	0	1,69	1,99
surf, kr/Mb	20	20	20	20	20

Om det bara funnes en abonnemangsform skulle debiteringen för en månad kunna beräknas med metoden `debit`:

```
public class Account {
  private ArrayList<Connection> connections;
  public double debit() {
    double sum = 29;
    for (Connection connection : connections) {
      sum += connection.debit();
    }
    return sum;
  }
}
```

Ett `Connection`-objekt antas tillhandahålla information om tidpunkten för uppkopplingen, typ av uppkoppling, tidpunkt för nedkoppling när så är relevant samt antalet Mbyte vid en surf-uppkoppling.

a. Gör en design för uppkopplingar och använd *Strategy*-mönstret för att hantera olika abonnemangsformer. Lösningen presenteras med ett klassdiagram och skall redovisa alla klasser, gränssnitt, attribut och metoder som behövs för att kunna exekvera metoden `debit` i `Account`. (5p)

b. Implementera den uppkoppling som representerar ett samtal i eget nät och den strategi som beskriver abonnemanget *Prata på*. Detaljer i debiteringen som ej framgår av tabellen hanteras på rimligt sätt. (5p)

Använd klassen `GregorianCalendar` för att representera tidpunkter. Den enda metod som behövs är `public long getTimeInMillis()` som returnerar antalet millisekunder sedan början av år 1970.

3 Följande klasser ingår i ett röstningsprogram med ett grafiskt användargränssnitt. Modifiera programmet så att *Observer*-mönstret används för att uppdatera vyn och att modellen inte känner till vyn. Lösningen redovisas med Java-kod. (5p)

```
public class Counters {
  private int yesCounter, noCounter;
  private View view;
  public Counters(View view) {
    this.view = view;
  }
  public void incrementYes() {
    yesCounter++;
    updateView();
  }
  public void incrementNo() {
    noCounter++;
    updateView();
  }
  private void updateView() {
    view.update(yesCounter, noCounter);
  }
}
public class View extends JPanel {
  public void update(int yesCounter, int noCounter) {
    // omissions
  }
}
```

I `java.util` finns

```
public interface Observer {
  public void update(Observable observable, Object object);
}

public class Observable {
  public void addObserver(Observer observer) ...
  protected void setChanged() ...
  public void notifyObservers(Object object) ...
}
```