Tentamen i Objektorienterad modellering och diskreta strukturer

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: Diskreta strukturer
Andersson: UML–syntax
Föreläsningsbilderna F01-06.pdf
Holm: Java snabbreferens

---

1 Nedan utdrag ur ett Javaprogram som implementerar ett kalkylark.

```
public class Sheet extends Observable implements Environment {
    private SlotFactory slotFactory = new SlotFactory();
    private Map<String, Slot> map = new HashMap<String, Slot>();

    public void clear(String address) {
        //code omitted
    }

    public void set(String address, String string) {
        Slot newSlot = slotFactory.build(string);
        map.put(address, newSlot);
        setChanged();
        notifyObservers();
    }

    public String toString(String address) {
        Slot slot = map.get(address);
        return (slot == null) ? "" : slot.toString();
    }
}

public class Editor extends JTextField implements Observer {
    private String currentAddress;
    private Sheet sheet;

    public Editor(Sheet sheet) {
        this.sheet = sheet;
        sheet.addObserver(this);
    }

    public void update(Observable observable, Object object) {
        setText(sheet.toString(currentAddress));
    }
}
```

`Environment` är ett gränssnitt med metoden `toString(String)`. `JTextField` har en metod `setText`. *Observable* och *Observer* enligt nedan. Ni ska i uppgifterna utnyttja er kunskap om *Observer*-mönstret och det ska framgå av era diagram hur *Observer*-mönstret fungerar, t ex hur Editorn uppdateras.

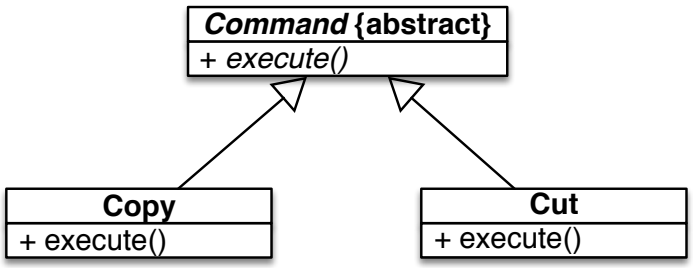
```
public interface Observer {
  public void update(Observable observable, Object object);
}

public abstract class Observable {
  public void addObserver(Observer observer)
  public void setChanged()
  public void notifyObservers()
}
```

a. Rita ett fullständigt klassdiagram inkluderande bl a associationer, relationer, attribut, metoder och synlighet.

b. Rita ett sekvensdiagram som visar hela exekveringen av metoden `set` (i `Sheet`). De aktuella parametrarna är "A1", respektive "A2+3".

(8p)

2 I nedanstående klassdiagram så är `execute`-metoden i `Command` abstrakt och implementationen av metoden i `Copy` respektive `Cut` nästan identiska. De visar sig skilja sig åt endast på ett ställe i koden. Man bestämmer sig för att ta bort den duplicerade koden genom att



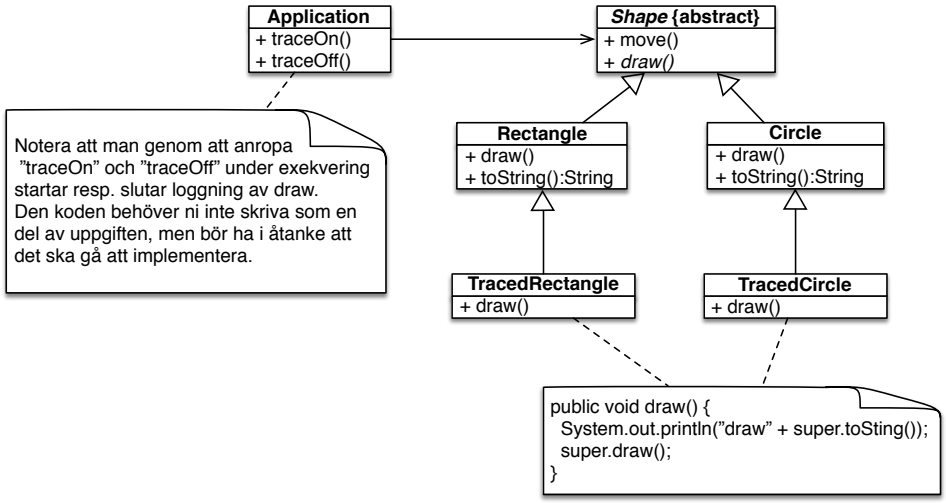
använda *Template Method*-mönstret.

a. Rita om klassdiagrammet så som det ser ut efter förändringen. Du får själv namnge ev. nya metoder. Inga nya klasser eller gränssnitt behövs.

b. Förklara vad du har gjort.

(6p)

3 Nedan ett klassdiagram som visar en design gjord för att Application ska kunna hantera figurer (`Shape`). Två typer av `Shape` finns, `Rectangle` och `Circle`. Man har även lagt till subklasser till `Rectangle` och `Circle` vilka beter sig precis som sin respektive superklass, men dessutom ger trace-utskrift vid anrop av `draw`. Tanken är att man i `Application` ska kunna slå på och av "trace" genom att antingen ha objekt av Trace-klasserna eller av de "vanliga" klasserna (`Rectangle` och `Circle`).



a. Detta är ingen bra design! Man bör i stället använda sig av *Decorator*-mönstret. Förklara varför.

b. Rita ett klassdiagram som visar hur lösningen ser ut när *Decorator* används i stället för nuvarande lösning. Visa även hur Java-koden blir för metoden `draw` i dekoratorn (med en notisruta i klassdiagrammet på samma sätt som i uppgiften).

(5p)

4 I följande klass går det inte att förändra formeln för att beräkna avkastningen (`revenue`) utan att kompilera om klassen.

```
public class Account {
    private float balance;
    public float revenue(int days) {
        float interest = 4.0;
        return balance*days*interest/365;
    }
// other methods omitted
}
```

Gör om designen med användning av *Strategy*-mönstret så att man under exekveringen kan byta algoritm för att beräkna avkastningen. Man får förutsätta att algoritmen bara använder beloppet (`balance`) och antalet dagar (`days`). Metoden `revenue` används av andra klasser som man inte kan ändra på. Lösningen redovisas med en modifierad `Account`-klass, den klass som implementerar algoritmen ovan samt övriga klasser som används i den nya versionen av Account. (3p)

5 Rita ett tillståndsdiagram för en fönsterhiss till en bilruta. En motor som öppnar resp. stänger fönstret ska styras (motor=*Up/Down/Stop*). Insignaler till styrlogiken är:

- en fjädrande vippbrytare vilken kan tryckas ner i läge *Upp* eller *Ner*, men som fjädrar tillbaka till *Noll* när man släpper den.
- två givare: en som ger signal när rutan kommit helt upp (*Uppe*) och en när rutan är helt ner (*Nere*).

När man trycker *Upp* eller *Ner* ska rutan stängas resp. öppnas tills man släpper knappen eller rutan stängts resp. öppnats helt. Starttillståndet är att fönsterrutan är stängd (dvs helt uppe). (4p)

6 En av de Morgans lagar säger att $P \vee Q \dashv\vdash \neg(\neg P \wedge \neg Q)$.

För att visa detta måste man visa härledningen i båda riktningarna.

a. Visa, med naturlig härleding, $P \vee Q \vdash \neg(\neg P \wedge \neg Q\ )$
Ange vilken härledningsregel som använts i varje led
Ledning: gör en indirekt härledning med hjälp av $\vee_E$

b. Gör härledningen åt andra hållet, $\neg(\neg P \wedge \neg Q) \vdash P \vee Q$,
genom att komplettera nedanstående bevisträd

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[\cancel{P}]}}{\dfrac{[\neg(\cancel{P \vee Q})]}{\neg P}} \qquad \dfrac{\dfrac{}{[\cancel{Q}]}}{\dfrac{[\neg(\cancel{P \vee Q})]}{\neg Q}}}{\neg P \wedge \neg Q}}{\neg(\neg P \wedge \neg Q)}$$

Ange vilken härledningsregel som använts i varje led

c. Härledningen $\forall x \,.\, \exists y \,.\, P(x,y) \vdash \exists x \,.\, P(x,x)$,

$$\dfrac{\dfrac{\dfrac{\forall x \,.\, \exists y \,.\, P(x,y)}{\exists y \,.\, P(a,y)}\,[\forall_E] \qquad \dfrac{[\cancel{P(a,a)}]}{\exists x \,.\, P(x,x)}\,[\exists_I]}{\exists x \,.\, P(x,x)}\,[\exists_E]$$

är inte giltig, vilket visas av följande motexempel:
Låt $x \in \mathbb{N}, y \in \mathbb{N}$ och $P(x,y) \triangleq x > y$

Vilken regel är det som används fel, och varför leder detta till en felaktig slutsats?

(6p)

7 För två relationer, $\rho_1 \subseteq M_1 \times M_2$ och $\rho_2 \subseteq M_2 \times M_3$ definieras sammansättningen
$\rho_1 \,;\, \rho_2 \triangleq \{(a,b) \in M_1 \times M_3 \mid \exists x \in M_2 \,.\, (a,x) \in \rho_1 \wedge (x,b) \in \rho_2)\}$

Givet mängderna $A \triangleq \{1,2,3,4,5,6\}, B \triangleq \{2,4,6\}$ och $C \triangleq \{1,2,3\}$
och relationerna $\rho \subseteq A \times B$ och $\mu \subseteq B \times C$ där

$\rho \triangleq \{(1,2),(2,2),(3,4),(4,4),(5,6),(6,6)\}$

$\mu \triangleq \{(2,1),(4,2),(6,3)\},$

beräkna sammansättningen $\rho \,;\, \mu$. Svara med en mängd. (2p)

8 POSIX syntax för utökade reguljära uttryck kan sammanfattas

```
symbol   matchar
------   ----------------------------------------------------------------
.        ett godtyckligt tecken
[ ]      ett av tecknen inom klamrarna. T ex matchar [ab] 'a' eller 'b'
[^ ]     ett tecken som inte finns mellan klamrarna. [^ab] matchar t ex 'c'
         men  inte 'a' eller 'b'
^        början av en sträng. I radbaserade verktyg, första tecknet på en rad.
$        slutet av en sträng eller eller (tecknet före) en radbrytning.
*        föregående element noll eller fler gånger.
+        föregående element en eller fler gånger.
{n}      föregående element n gånger
{n,m}    föregående element minst n, max m gånger
\x       om x är en specialsymbol betecknar \x tecknet x i stället för specialsymbolen
         T ex matchar \{ tecknet '{', och \* tecknet '*'
(S|T)    uttrycket S eller uttrycket T. t ex matchar (b+|c)at bland annat
         strängarna "bat", "bbat" och "cat"
```

Ange ett reguljärt uttryck för språket av alla Java-kommentarer av formen
`/* kommentartext ... */`

Du behöver inte ta någon särskild hänsyn till radbrytningar,
men tänk på att tecknet '*' kan förekomma inuti kommentarstexten. (2p)

Härledningsregler

Regler för $\wedge$

$$\frac{P \qquad Q}{P \wedge Q}\,[\wedge_I] \qquad \frac{P \wedge Q}{P}\,[\wedge_{E_1}] \qquad \frac{P \wedge Q}{Q}\,[\wedge_{E_2}]$$

Regler för $\rightarrow$

$$\frac{\begin{array}{c}[\cancel{P}]\\ \vdots\\ Q\end{array}}{P \rightarrow Q}\,[\rightarrow_I] \qquad \frac{P \qquad P \rightarrow Q}{Q}\,[\rightarrow_E]$$

Regler för $\vee$

$$\frac{P}{P \vee Q}\,[\vee_{I_1}] \qquad \frac{Q}{P \vee Q}\,[\vee_{I_2}] \qquad \frac{P \vee Q \qquad \begin{array}{c}[\cancel{P}]\\ \vdots\\ R\end{array} \qquad \begin{array}{c}[\cancel{Q}]\\ \vdots\\ R\end{array}}{R}\,[\vee_E]$$

Regler för $\neg$

$$\frac{\begin{array}{c}[\cancel{P}]\\ \vdots\\ R\end{array} \qquad \begin{array}{c}[\cancel{P}]\\ \vdots\\ \neg R\end{array}}{\neg P}\,[\neg_I] \qquad \frac{\begin{array}{c}[\cancel{P}]\\ \vdots\\ R \wedge \neg R\end{array}}{\neg P}\,[\neg_I] \qquad \frac{\neg\neg P}{P}\,[\neg_E]$$

Regler för $\forall$

$$\frac{P}{\forall x\,.\,P}\,[\forall_I] \qquad \frac{\forall x\,.\,P}{P[x\backslash t]}\,[\forall_E]$$

Regler för $\exists$

$$\frac{P[x\backslash t]}{\exists x\,.\,P}\,[\exists_I] \qquad \frac{\exists x\,.\,P \qquad \begin{array}{c}[\cancel{P[x\backslash y]}]\\ \vdots\\ Q\end{array}}{Q}\,[\exists_E]$$

$\begin{array}{c}[\cancel{P}]\\ \vdots\\ Q\end{array}$ i en regel betyder att man har gjort ett (hypotetiskt) antagande P för att härleda Q och att antagandet upphävs (strykes) när man använder regeln. Det hypotetiska antagandet får finnas noll eller flera gånger i beviset av Q.

När man använder $\forall_I$ så får beviset av P inte innehålla några icke upphävda antaganden om x.

I $\exists_E$ får y inte vara en fri variabel i Q eller förekomma i ett icke upphävt antagande i $\begin{array}{c}[\cancel{P[x\backslash y]}]\\ \vdots\\ Q\end{array}$