Tentamen i Objektorienterad modellering och design

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

---

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.

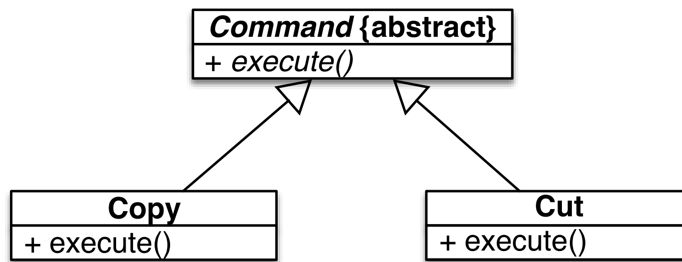
```
public interface Observer {
  public void update(Observable observable, Object object);
}

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



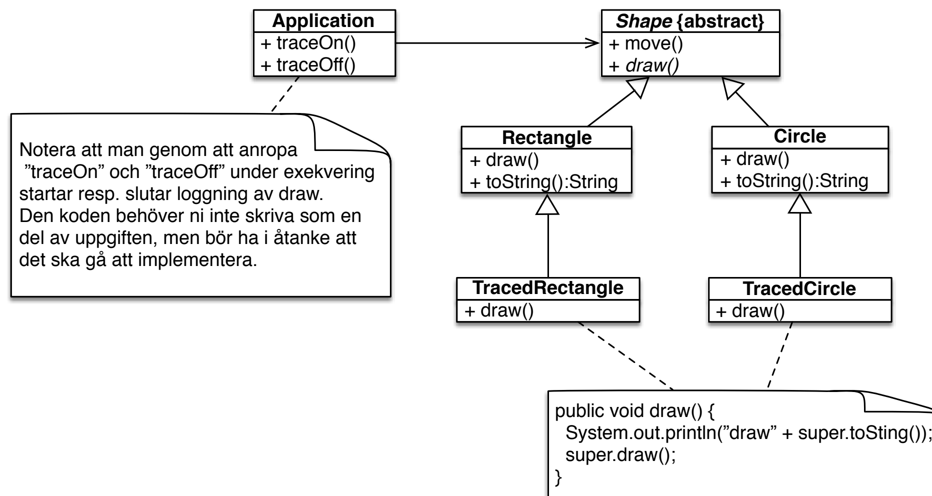
använda *Template Method*-mönstret.

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)