OMD – Övning vecka 3

Du förväntas göra lösningsförslag till uppgifterna före övningen, gärna tillsammans med andra kursdeltagare.

Lösningar till dessa uppgifter presenteras under övningen i läsvecka 3.

---

1 En mobiltelefonoperatör erbjuder olika slags abonnemang. Läraren i OMD har sagt att när beskrivningen av verkligheten använder "olika slags ..." skall man använda arv. Den uppmärksamme teknologen gör följande design på sommarjobbet på Comviq:

```
public abstract class Tariff {
    private String phoneNumber;
    protected List<Call> call = new ArrayList<Call>();
    public abstract Money debit(Call call);
    // omissions
}

public class CashCard extends Tariff {
    public Money debit(Call call) {
    // omissions
    }
}
public class KnockOut extends Tariff {
    public Money debit(Call call) {
    // omissions
    }
}
```

Projektledaren har invändningar; när en abonnent vill byta abonnemangsform måste man skapa ett nytt `Tariff`-objekt och kopiera telefonnummer mm från det gamla abonnemanget. Gör en ny design utan denna olägenhet som utnyttjar designmönstret *Strategy*. Redovisa designen med ett klassdiagram.

2 En tidigare upplaga av en lärobok i objektorienterad programmering och Java innehåller följande kod:

```
...
private Application application;
private JButton b1, b2, b3;
class ActionHandler implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == b1) {
            application.action1();
        } else if (e.getSource() == b2) {
            application.action2();
        } else {
            application.action3();
        }
    }
}
```

Designen strider mot *Open-Closed Principle*; man kan inte lägga till ytterligare `JButton`s utan att modifiera klassen. Gör om designen så att detta blir möjligt. Designen skall innehålla de tre knapparna och händelsehanteringen. Undvik duplicerad kod. Redovisa designen med ett klassdiagram.

3 Syftet med designmönstret *Singleton* är att förhindra att mer än en instans av klassen kan skapas. Nedan två olika implementeringar som båda begränsar antalet instanser av klassen.

Förklara för- och nackdelar med de två olika implementeringarna.

```
public class Singleton {
        private static Singleton instance; // attributes omitted
        private Singleton() {
                // omissions
        }
        public static Singleton instance() {
                if (instance == null) {
                        instance = new Singleton();
                }
                return instance;
        }
        // other methods omitted
}
```

```
public class Singleton {
    private static int bound = 1;
    public Singleton() {
        if (bound == 0) {
            throw new RuntimeException(
               "Singleton: No more objects must be created");
        }
        bound--;
    }
}
```

4 Följande två klasser innehåller duplicerad kod. Elimenera den genom att använda *Template Method*-mönstret. Lösningen redovisas som Java-kod.

```
public class StarList extends java.util.ArrayList {
    public String toString() {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < size(); i++) {
            builder.append("* ");
            builder.append(get(i));
            builder.append('\n');
        }
        return builder.toString();
    }
```

```
public class EnumList extends java.util.ArrayList {
    public String toString() {
        StringBuilder builder = new StringBuilder();
        for (int i = 0; i < size(); i++) {
            builder.append(i + 1).append(". ");
            builder.append(get(i));
            builder.append('\n');
        }
        return builder.toString();
    }
```

5 I följande exempel används `null` för att representera okända föräldrar. Använd *Null Object*-mönstret för att få en bättre design där det inte behövs några if-satser. Lösningen

redovisas med ett klassdiagram med generaliseringar, attribut och metoder samt Java-kod.

```
public class Person {
    private String name;
    private Person father, mother;
    public void printAncestors() {
        System.out.println(name);
        if (father != null) {
            father.printAncestors();
        }
        if (mother != null) {
            mother.printAncestors();
        }
    }
}
```

6 Med institutionens kaffeautomat kan man välja olika drycker och flera tillbehör genom att trycka på knappar. Använd *Decorator*-mönstret för att modellera möjligheten att välja kaffe eller choklad, med eller utan tillbehören mjölk och socker. Om man trycker flera på tillbehörsknapparna ökar dosen på tillbehöret.

Modellen skall testas med

```
public static void main(String[] args) {
     Drink drink = new Sugar(new Milk(new Milk(new Coffee())));
     System.out.println(drink);
     System.out.println(drink.cost());
}
```

med den förväntade utskriften

```
kaffe mjölk mjölk socker
7
```

Kostnaden för en dos kaffe är 5 kronor, en skvätt mjölk kostar 1 kr och sockret är gratis.

Gör ett klassdiagram för hela modellen och implementera klasserna Coffee och Milk och alla abstrakta klasser.