Instuderingsuppgifter läsvecka 4

1.

Vilka egenskaper skall en equals-metod uppfylla?

2.

Antag att klassen Dog är definierad på följande sätt:

```
public class Dog {
   private String breed;
   private String name;
   private String gender;
   public Dog(String aBreed, String aName, String aGender) { … }
   public String getBreed() { … }
   public String getName() { … }
   public String getGender() { … }
   //Two dogs are equal if they have equal breeds, genders and names.
   public boolean equals(Object o) {…)
   public int hashCode() { … }
   …
}//Dog
```

Skriv kod för metoderna equals och hashCode.

3.

a) Antag att du av någon anledning vill förbjuda att objekt av en viss klass kopieras med clone. Vad behöver du addera till klassen? Visa hur det skall se ut!

b) Överskugga metoden clone() i klassen C nedan:

```
public class B implements Cloneable {
   public B clone() { . . . }
}

public class C {
   private int x;
   private B b;
   . . .
}
```

c) Överskugga metoden clone() i klassen D nedan:

```
import java.util.ArrayList;
final public class D {
   private int x;
   private ArrayList<C> cs;
   . . .
}
```

Använd tekniken med *kopieringskonstruktor*, vilken givetvis då också skall implementeras, samt utnyttja möjligheten till *kovariant returtyp*. Tips: Standard klassen ArrayList överskuggar clone().

d) När är det lämpligt att använda tekniken med *kopieringskonstruktor* för att implementera clone() och när är det olämpligt.

4.

Förklara kortfattat vad ett designmönster är, samt vilka fördelar det finns med att använda designmönster.

5.

Ge ett exempel på där designmönstret *Singleton* kan vara användbart.

6.

Varför kan man inte skapa en subklass till en klass som är singleton?

7.

Klasserna OpenButton och CloseButton är nästan identiska.

```
public class OpenButton extends JButton implements ActionListener {
    private Door door;
    public OpenButton(Door door) {
        super("Open");
        this.door = door;
        addActionListener(this);
    }
    public void actionPerformed(ActionEvent event) {
        door.open();
    }
}
public class CloseButton extends JButton implements ActionListener {
    private Door door;
    public CloseButton(Door door) {
        super("Close");
        this.door = door;
        addActionListener(this);
    }
    public void actionPerformed(ActionEvent event) {
        door.close();
    }
}
```

Gör om designen med användning av mönstret *Template method* för att eliminera duplicerad kod. Lösningen redovisas som Java-kod.

8.

Betrakta följande klass:

```
public class Printer {
    private int format;
    public Printer(int format) {
        this.format = format;
    }
    public void output(String s) {
        if (format == 1) {
            System.out.println(s);
        }
        else if (format == 2) {
            System.out.println("<p>" + s + "</p>");
        }
        else if (format == 3) {
            System.out.println(s + "//");
        }
    }//output
}//Printer
```

Designen bryter mot *Open/Closed-principen*; man kan inte lägga till ytterligare utskriftsformat utan att behöva modifiera klassen Printer. Eliminiera detta problem genom att göra om designen med användning av *Strategy*-mönstret.

9.

Betrakta klassen Book nedan:

```
public class Book {
   private String title;
   private double price;

   public Book(String title, double price) {
      this. title = title;
      this. price = price;
   }

   public double getPrice() {
      return price;
   }

   public String getTitle() {
      return title;
   }
}//Book
```

a) Gör nödvändiga tillägg i klassen så att det införs en naturlig ordning på böckerna, vilken skall definieras efter alfabetisk växande ordning på böckernas titel. Överskugga samtidigt equals-metoden.

b) Det finns också ett behov av att kunna sortera böckerna efter växande ordning på priset (utan att förstöra den naturliga sorteringsordningen efter böckernas titel). Visa hur detta går till.

10.

Vad är syftet med esignmönstren *State*?