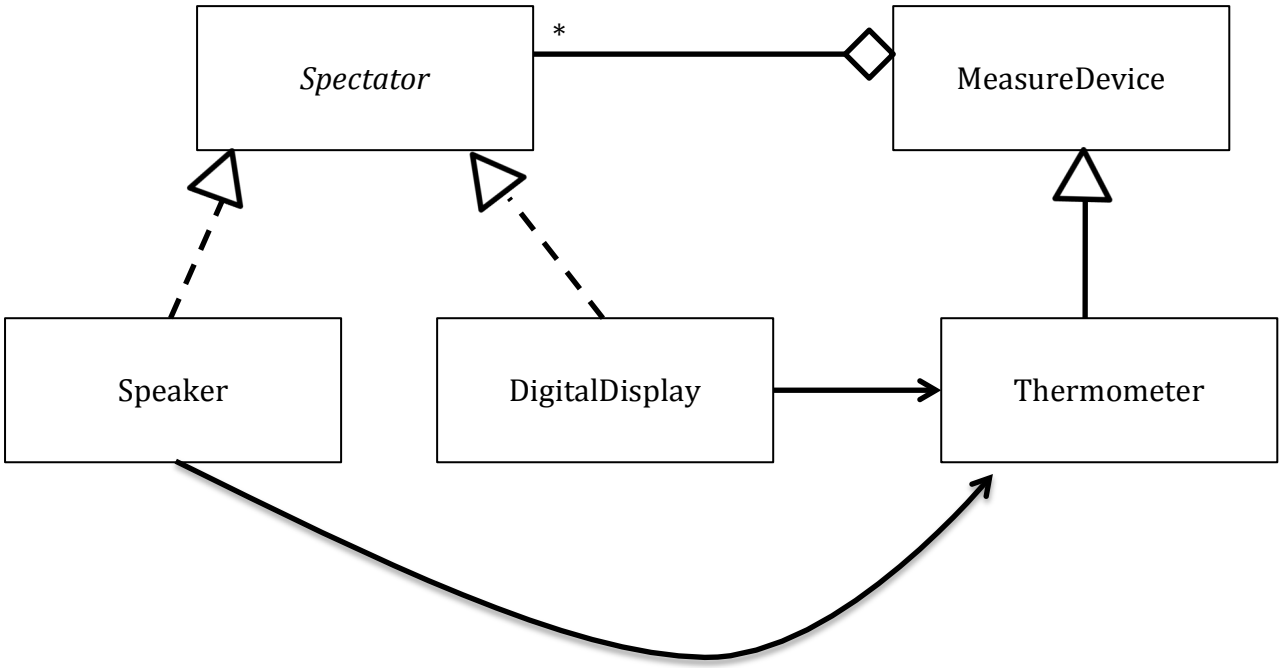
Lösning till tentamen 7:e juni 2016

1.
   a. Template Metod
   b. State
   c. Proxy
   d. Iterator
   e. Observer

2.



3. Observer
4.
   a. Tesla:2 Tesla:2
   b. mark har typen String och är instansvariabel vilket innebär att variabeln är unik för alla instanser av klassen Car.

      amount har typen int och är klassvariabel vilket innebär att variabeln hör till klassen Car och delas av alla instanser av klassen Car.
5.

   a. I abstrakta klasser kan man ha konstruktorer och konkreta metoddeklarationer men interface kan inte innehålla konkreta metoddeklarationer. I abstrakta klasser kan man definiera instansvariabler och klassvariabler med i interface kan man endast ha konstanter.
      Med abstrakta klasser kan man undvika upprepning av kod mellan olika klasser i olika hierarkinivåer men interface används för att lämna konkretisering av funktioner till implementerande klasser.

b. Enum används för att skapa egna uppräkningstyper med egna konstanta värden. Exempelvis: Vardagar, Årstider.

```
public enum Vardag{måndag, tisdag, onsdag,
torsdag, fredag} ;
Vardag veckoDag2 = Vardag.tisdag;
```

6.

a. metoden activity i klassen Person har deklarerats med åtkomstnivå modifieraren private vilket gör att variabler av typen Person som är definierad utanför klassen Person som miranda inte kommer åt metoden, lösningen är att ta bort Private. Instansering av klassen Athlete i klassen Main, `new Athlete("Miranada")`, kräver att det finns en konstruktor med en parameter av typen String som anropar på superklassens konstruktor i klassen Athlete.

b.

```
class Person{
…
     private void activity(){
         System.out.println("REST");
…}
class Athlete extends Person{
    public Athlete(String name){
         super(name);
    }
…}
```

7. b, d, h, j

8.

a. En tråd som är i död tillstånd har förbrukats och kan inte startas igen. Tråden kommer inte kunna hamna i annat tillstånd än död.
b. Färdig för körning
c. Koden som finns i metoden `run` kommer att köras men den kör inte som en tråd, d.v.s. att den kommer inte kunna köra parallell med resten av programmet.

9. Se föreläsning 12

10. Kraven som ställs Singletonklass är andra klasser ska inte kunna skapa instanser av klassen genom att använda `new`. Samt att när väl en instans av klassen har skapats så ska inte flera instanser av klassen ska kunna skapas.

```
class CentralFuseBox{
```

```
    static CentralFuseBox theOnlyOne;
      private CentralFuseBox(){...}
      …
  }
```

11.

```
import java.util.*;
class Bag extends PackItem {
    ArrayList<PackItem> things = new
                        ArrayList<PackItem>();

    Bag(String n) {
      super(n);
    }

    void add(PackItem p) {
          things.add(p);
          p.parent = this;
    }

    void remove(PackItem p) {
          things.remove(p);
          p.parent = null;
    }


    void print(String off) {
          System.out.println(off + name);
          for (PackItem p:things)
               p.print(off + "  ");
    }

    int countThings() {
        int n = 0;
        for (PackItem p:things)
            if (p instanceof Thing)
               n += 1;
        return n;
    }

    void putInBag() {
         if (countThings() == 1 && things.size()==1) {
               System.out.println("No extra bag needed
                             for " + things.get(0).name);
               return;
         }
         for (PackItem p:things)
```

```
                p.putInBag();
            }

    }



    abstract class PackItem {
        String name;
        Bag parent;


        PackItem (String n) {
           name = n;
        }

       abstract void print(String off);
       abstract void putInBag();
    }



class Thing extends PackItem {

     Thing(String n) {
      super(n);
     }

     void print(String off) {
      System.out.println(off + name);
     }

     void putInBag(){
      Bag thebag = new Bag("Safetybag");
      int i = parent.things.indexOf(this);
      parent.things.set(i,(thebag));
      thebag.add(this);
     }
}
```