Tentamen för TDA550 **Objektorienterad programvaruutveckling IT, fk**

DAG: 12-04-10 TID: 8:30 – 12:30

Ansvarig: Christer Carlsson, ankn 1038

Förfrågningar: Christer Carlsson

Resultat: erhålls via Ladok

Betygsgränser:

3:a	24 poäng
4:a	36 poäng
5:a	48 poäng
maxpoäng	60 poäng

Siffror inom parentes: anger maximal poäng på uppgiften.

Granskning: Måndag 7/5 kl 12-13 och torsdag 8/5 kl 12-13, rum 6128 i EDIT-huset.

Hjälpmedel: Inga hjälpmedel är tillåtna förutom bilagan till tesen.

Var vänlig och: Skriv tydligt och disponera papperet på lämpligt sätt.

Börja varje uppgift på nytt blad. Skriv ej på baksidan av papperet.

Observera: Uppgifterna är ej ordnade efter svårighetsgrad. Titta därför igenom hela tentamen innan du börjar skriva.

Alla program skall vara välstruktruerade, lätta att överskåda samt enkla att förstå.

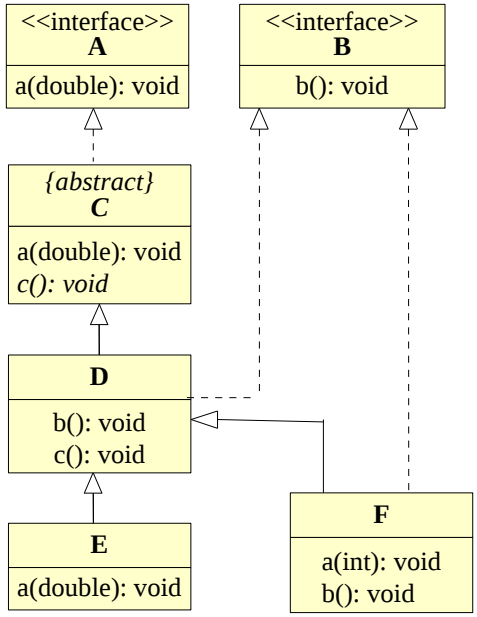
Vid rättning av uppgifter där programkod ingår bedöms principella fel allvarligare än smärre språkfel.

LYCKA TILL!!!!

Uppgift 1.

Betrakta nedanstående klasser och interface:

```
public interface A {
    public void a(double x);
}//A

public interface B {
    public void b();
}//B

public abstract class C implements A {
    public void a(double x) {
        System.out.println( "a() in C" );
    }//a
    public abstract void c();
}//C

public class D extends C implements B {
    public void b() {
        System.out.println( "b() in D" );
    }//b
    public void c() {
        System.out.println( "c() in D" );
    }//c
}//D

public class E extends D {
    public void a(double x) {
        System.out.println( "a() in E" );
    }//a
}//E

public class F extends D implements B {
    public void a(int i) {
        System.out.println( "a() in F" );
    }//a
    public void b() {
        System.out.println( "b() in F" );
    }//b
}//F
```



Vad blir resultatet för var och en av följande satser (ger kompileringsfel, ger exekveringsfel, skriver ut xxx, etc)?

a)
```
A x = new C();
x.a(5);
```

b)
```
D x = new F();
x.a(5);
```

c)
```
F x = new F();
x.a(5);
```

d)
```
B x = new F( );
D y = (D) x;
y.b();
```

e)
```
B x = new D( );
F y = (F) x;
y.b();
```

f)
```
B x = new E( );
D y = (D) x;
y.c();
```

(6 poäng)

Uppgift 2.

a) Vad blir utskriften av nedanstående program?

```
public class ExceptionStuff {
    private static boolean throwException = true;
    public static void main(String[ ] args) {
        try {
            methodA();
        }
        catch (Exception e) {
            System.out.println("Caught Exception in main");
        }
    } //main

    private static void methodA( ) throws Exception {
        System.out.println( "Enters methodA" );
        try {
            methodB();
        }
        catch (Exception e) {
            System.out.println("Caught Exception in methodA");
        }
        finally{
            System.out.println("finally in methodA");
        }
        System.out.println("Exits methodA");
    } // methodA

    private static void methodB( ) throws Exception {
        System.out.println("Enters methodB");
        if (throwException == true) {
            throw new Exception();
        }
        System.out.println("Exits methodB");
    } // methodB
} // ExceptionStuff
```

b) Vad blir utskriften av ovanstående program om vi initierar variabeln throwException till **false**?

(4 poäng)

Uppgift 3.

Är nedanstående klass trådsäker? Motivera ditt svar!

```
public class Utilities {
    public final int extra;

    public Utilities(int extra) {
        this.extra = extra;
    }//constructor

    public int sum(int x, int y) {
        return x + y + extra;
    }//sum

    public int product(int x, int y) {
        return x * y * extra;
    }//product
}// Utilities
```

(2 poäng)

Uppgift 4.

a) Skriv specifikationen för nedanstående metod

```
/**
    @pre    . . .
    @post   . . .
    @returns  . . .
**/
public static double mean (int[] a, int n) {
    double sum = 0;
    for (int i = 0; i < n; i++)
        sum = sum + a[i];
    return sum / n;
}//mean
```

(2 poäng)

b) Betrakta följande specfikation för metoden getMostFrequent:

```
/*
    @returns    returns the character that appears most frequently in str
    @throws     throws NullPointerException if str == null
*/
 public char getMostFrequent(String str) { . . . }
```

Denna specifikation är inte väldefinierad för alla möjliga indatavärden. Det är inte uppenbart vad metoden skall returnera när str är en tom sträng.

Ge ytterligare ett exempel på en indatasträng för vilken beteendet av metoden inte är väldefinierat.

(2 poäng)

Uppgift 5.

Betrakta nedanstående klass för att simulerar händelser:

```
public class Event {
    public final static int ARRIVAL = 0, DEPARTURE = 1, MEASUREMENT = 2;
    private int kind, time;
    public void execute(Context context) {
        switch (kind) {
            case ARRIVAL:
                context.arrival(time);
                break;
            case DEPARTURE:
                context.departure(time);
                break;
            default:
                context.measurement(time);
        }
    }//execute
    public String toString() {
        switch (kind) {
            case ARRIVAL:
                return "ARRIVAL" + " " + time;
            case DEPARTURE:
                return "DEPARTURE" + " " + time;
            default:
                return "MEASUREMENT" + " " + time;
        }
    }//toString
}//Event
```

Klassen bryter mot Open/Closed-principen. Man kan inte lägga till fler sorters händelser utan att ändra i metoderna execute och toString. Använd designmönstret Template Method för att göra en bättre design som följer Open/Closed-principen. Lösningen redovisas med Java-kod.

(6 poäng)

Uppgift 6.

Anta att standardklassen Float, som representerar reella tal som objekt, är implementerad enligt följande:

```
public class Float {
    private float f;
    public Float(float f) {
        this.f = f;
    }//constructor
    public float floatValue() {
        return f;
    }//floatValue
    public boolean equals(Object o) {
        if (! (o instanceof Float))
            return false;
        float f1 = this.floatValue();
        float f2 = ((Float) o).floatValue();
        return (f1 == f2);
    }//equal
    // more methods here that don't interest us now
}//Float
```

Pelle Hacker beslutar att skriva en förbättrad variant av Float-klassen genom att göra en subklass TolerantFloat som tolererar små avvikelser när man jämför huruvida två reella värden är lika. Pelles kod har följande utseende:

```
public class TolerantFloat extends Float {
    public static final float TOLERANCE = 0.01;
    public TolerantFloat(float f) {
        super (f);
    }//constructor
    public boolean equals(Object o) {
        if (!(o instanceof Float))
            return false;
        float f1 = this.floatValue();
        float f2 = ((Float) o).floatValue();
        return (Math.abs(f1 – f2) <= TOLERANCE);
    }//equals
}// TolerantFloat
```

I *The Java Language Specification* anges att en equals-metod skall vara reflexiv, symmetrisk och transitiv.

i) Är TolerantFloat.equals() reflexiv? Om inte, ge ett exempel som visar detta.

ii) Är TolerantFloat.equals() symmetrisk? Om inte, ge ett exempel som visar detta.

iii) Är TolerantFloat.equals() transitiv? Om inte, ge ett exempel som visar detta.

(6 poäng)

Uppgift 7.

Skriv om nedanstående klass så att den blir generisk och därmed kan användas för att handha andra klasser än bara String.

```
public class Col {
    private ArrayList<String> c;
    public String get() {
        return c.remove(0);
    }//get
    public void insert(String value) {
        c.add(value);
    }//insert
}//Col
```

(2 poäng)

Uppgift 8.

I ett röstprogram med grafiskt användargränssnitt ingår bl.a. följande klasser:

```
public class Counter {
    private int yesCounter, noCounter;
    private View view;
    public Counter(View view) {
        this.view = view;
    }
    public void incrementYes() {
        yesCounter++;
        updateView();
    }
    public void incrementNo() {
        noCounter++;
        updateView();
    }
    private void updateView() {
        view.update(yesCounter, noCounter);
    }
}//Counter

import javax.swing.*;
public class View extends JPanel {
    public void update(int yesCounter, int noCounter) {
        // visar, på ett eller annat sätt, upp värdena av yesCounter och noCounter på panelen
    }
}//View

public class Main {
    public static void main(String[] args) {
        . . .
        View view = new View();
        Counter counter = new Counter(view);
        . . .
    }
}//Main
```

Designen har är bristfällig eftersom modellklassen Counter är beroende av vyklassen View. Modifiera designen med användning av *Observer-mönstret* så att vyn kan uppdateras utan att modellen känner till vyn. Lösningen redovisas med Java-kod.

(6 poäng)

Uppgift 9.

Betrakta nedanstående klasser:

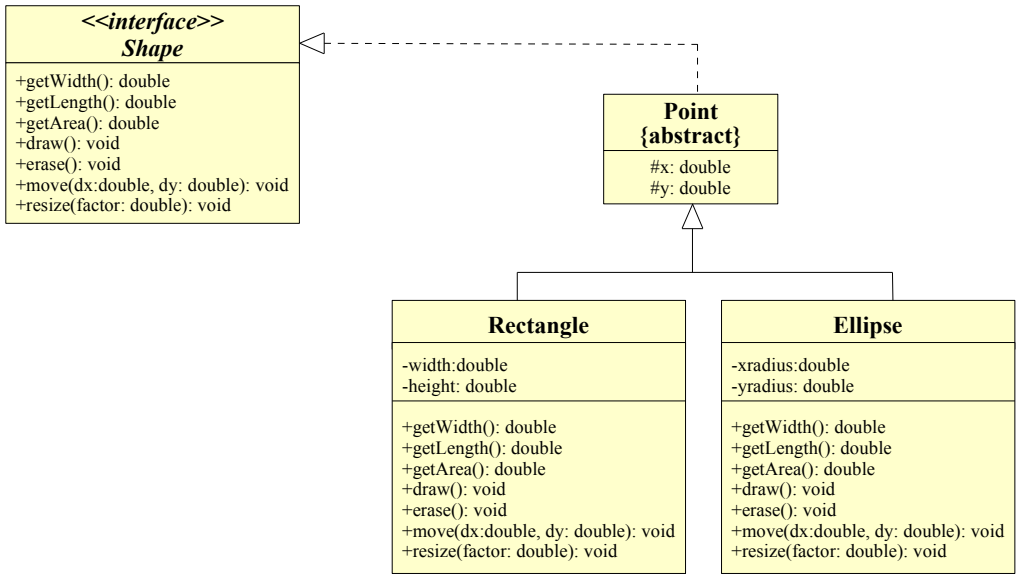
```
public class Course implements Cloneable, Serializable {
    //... implementation are omitted for the sake of brevity
}//Course

public class Student {
    private List<Course> courses;
    private String name;
    private int year;
    private double gpa;
    //... constructors and methods are omitted for the sake of brevity
}//Student
```

a) Överskugga (implementera) metoden clone() i klassen Student. Djup kloning skall användas. (5 poäng)

b) Gör de tillägg som behövs för att kunna skriva ut objekt av klassen Student på en fil. (1 poäng)

Uppgift 10.

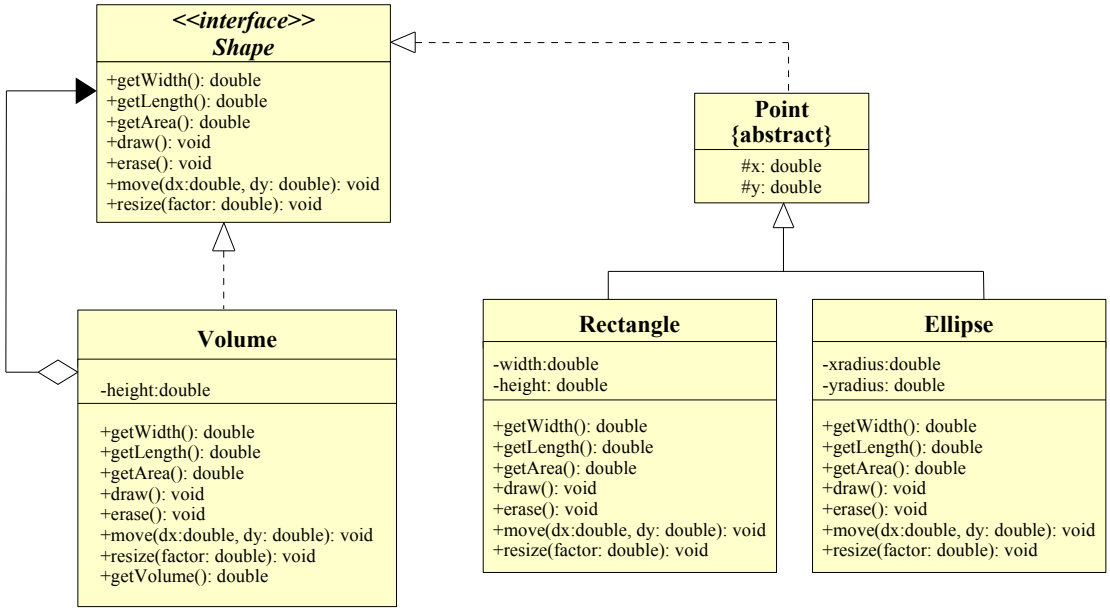
Antag att man har tillgång till följande klasser för att representera geometriska figurer



Klasserna är givna och får inte ändras. Antag att man vill kunna addera en tredje dimension till rektangel- och ellipsobjekt så att de får en volym. De skall alltså även ha en höjd. Inför klassen Volume. Klassen skall bl.a ha metoden

public double getVolume()

För att göra detta skall du använda *designmönstret Decorator* enligt figuren nedan:



a) Implementera klassen Volume i Java. (5 poäng)

b) Ge ett kodexempel som skapar en kub och skriver ut dess volym. (1 poäng)

Uppgift 11.

Klassen Assistants används för att hålla reda på vilka övningsassistenter som deltar på vilka kurser. Klassen använder sig av en Map där kurserna (som representeras av strängar) utgör nycklar och där övningsassistenterna på kursen (som också representeras som strängar) lagras i ett Set.

```
public class Assistants {
    private Map<String, Set<String>> map;
    public Assistants() {
        // skall implementeras i deluppgift a)
    }
    public void addTA(String course, String taName) {
        // skall implementeras i deluppgift b)
    }
    public void displayTAsPerCourse() {
        // skall implementeras i deluppgift c)
    }
}// Assistants
```

a) Implementera konstruktorn som skapar en tom Map. Deluppgift c) ger nödvändig information för att du ska kunna bestämma vilken konkret implementation av Map som skall användas.

b) Implementera metoden addTA som lägger till en övningsassistent på en specifik kurs. Om kursen inte redan finns i Map:en skall en ”map entry” för kursen skapas. Deluppgift c) ger nödvändig information för att du ska kunna bestämma vilken konkret implementation av Set som skall användas.

c) Implementera metoden displayTAsPerCourse som, på System.out, skriver ut namnet på samtliga kurser samt de övningsassistenter som finns på kurserna. Kurserna och övningsassistenterna skall skrivas ut i *alfabetisk ordning* enligt:

```
Diskret matematik
   Anne Andersson
   Bo Bertilsson
   Pelle Persson
Objektorienterad programmering
   Lisa Larsson
   Pelle Persson
   Sara Svensson
```

(6 poäng)

Uppgift 12.

Betrakta nedanstående klass:

```
public class Shutdown {

    /** Print msg and shut down JVM */
    public static void shutdownNow(String msg) {
        System.out.println(msg);
        System.exit(1);              //cause entire JVM to shut down
    }// shutdownNow
}//Shutdown
```

Klassen innehåller endast den statiska metoden shutdownNow, som har en sträng msg som parameter. Metoden skriver ut msg varefter programmet skjuts ner.

Din uppgift är att utöka klassen med en metod

```
public static void delayedShutdown(int seconds, String msg)
```

som skall fungera som metoden shutdownNow förutom att det skall dröja seconds sekunder innan strängen mgs skrivs ut och programmet skjuts ned.

Tips: Låt klassen Shutdown *antingen* implementera interfacet Runnable *eller* utöka klassen Thread.

(6 poäng)

Tentamen 120410- LÖSNINGSFÖRSLAG

Uppgift 1.

a) Tilldelningen A x = **new** C() ger kompileringsfel eftersom klassen C är abstrakt.

b) Ger utskriften ”a() in C”

c) Ger utskriften ”a() in F”

d) Ger utskriften ”b() in F”

e) Ger exekveringsfel, eftersom den dynamiska typen på x är D och typen D kan inte typomvandlas till typen F.

f) Ger utskriften ”c() in D”

Uppgift 2.

a) Utskriften blir:

```
Enters methodA
Enters methodB
Caught Exception in methodA
finally in methodA
Exits methodA
```

b) Utskriften blir:

```
Enters methodA
Enters methodB
Exits methodB
finally in methodA
Exits methodA
```

Uppgift 3.

Ja, klassen är trådsäker eftersom den är icke-muterbar.

Uppgift 4.

a)

```
/**
    @pre      a != null and 0 < n <= a.length
    @post     none
    @returns  returns the mean of the first n numbers in a
**/
```

b) Till exempel är beteendet inte väldefinierat för strängen ”ab”. Eller mer generellt, varje sträng där en eller flera tecken förekommer med samma frekvens, så är “most frequently” inte tillräckligt definierat.

Uppgift 5.

```
public abstract class Event {
    protected int time;
    public abstract void execute(Context context);

    public abstract String kind();

    public String toString() {
        return kind() + " " + time;
    }//toString
}//Event
```

```
public class Arrival extends Event {

    public void execute(Context context) {
        context.arrival(time);
    }//execute

    public String kind() {
        return "ARRIVAL";
    }//kind
}// Arrival
```

```
public class Departure extends Event {

    public void execute(Context context) {
        context.departure (time);
    }//execute

    public String kind() {
        return "DEPARTURE";
    }//kind
}// Departure
```

```
public class Measurement extends Event {

    public void execute(Context context) {
        context. measurement (time);
    }//execute

    public String kind() {
        return "MEASUREMENT";
    }//kind
}// Measurement
```

Uppgift 6.

i) Ja, equals() är reflexiv. x.equals(x) ger **true** för alla objekt x av type Float och för alla objekt x av typen TolerantFloats.

ii) Nej, equals() är inte symmetrisk. Om vi skapar objekten

```
Float f = new Float(1.0f);
TolerantFloat tf = new TolerantFloat(1.01f);
```

erhåller vi att

```
tf.equals(f)  == true
```

men att

```
f.equals(tf)  == false
```

iii) Nej, equals() är inte transitiv. Om vi skapar objekten

```
TolerantFloat tf1 = new TolerantFloat(1.0f);
TolerantFloat tf2 = new TolerantFloat(1.01f);
TolerantFloat tf3 = new TolerantFloat(1.02f);
```

erhåller vi att

```
tf1.equals(tf2) == true
tf2.equals(tf3) == true
```

men att

```
tf1.equals(tf3) == false
```

Uppgift 7.

```
public class Col<E> {
    private ArrayList<E> c;
    public E get() {
        return c.remove(0);
    }//get
    public void insert(E value) {
        c.add(value);
    }//insert
}//Col
```

Uppgift 8.

```
import java.util.Observable;
public class Counter extends Observable {
    private int yesCounter, noCounter;
    public Counter() {}
    public void incrementYes() {
        yesCounter++;
        setChanged();
        notifyObservers();
    }// incrementYes
    public void incrementNo() {
        noCounter++;
        setChanged();
        notifyObservers();
    }// incrementNo
    public int getYesCounter() {
        return yesCounter;
    } // getYesCounter
    public int getNoCounter() {
        return noCounter;
    } // getNoCounter
}//Counter

import javax.swing.*;
import java.util.*;
public class View extends JPanel implements Observer {
    public void update(Observrable o, Object obj) {
        if (o instanceof Counters) {
            Counter counter = (Counter) o;
            int yesCounter = counter.getYesCounter();
            int noCounter = counter.getNoCounter();
            // visar, på ett eller annat sätt, upp värdena av yesCounter och noCounter på panelen
        }
    }
}//View

public class Main {
    public static void main(String[] args) {
        . . .
        View view = new View();
        Counter counter = new Counter();
        counter.addObserver(view);
        . . .
    }
}//Main
```

Med användning av PropertyChangeSupport och PropertyChangeListener

```
import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;
public class Counter {
    private PropertyChangeSupport pcs = new PropertyChangeSupport(this);
    private int yesCounter, noCounter;
    public Counter() {}
    public void addObserver(PropertyChangeListener observer) {
        pcs.addPropertyChangeListener(observer);
    }
    public void removeObserver(PropertyChangeListener observer) {
        pcs.removePropertyChangeListener(observer);
    }
    public void incrementYes() {
        yesCounter++;
        pcs.firePropertyChange("yesCounter",new Integer(0),this);
    }
    public void incrementNo() {
        noCounter++;
        pcs.firePropertyChange("noCounter", new Integer(0), this);
    }
    public int getYesCounter() {
        return yesCounter;
    }
    public int getNoCounter() {
        return noCounter;
    }
}//Counter

import javax.swing.*;
import java.util.*;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
public class View extends JPanel implements PropertyChangeListener {
    public void propertyChange(PropertyChangeEvent ev) {
        if (ev.getSource() instanceof CounterPC) {
          Counter c = (Counter) ev.getSource();
          int yesCounter = c.getYesCounter();
          int noCounter = c.getNoCounter();
          // visar, på ett eller annat sätt, upp värdena av yesCounter och noCounter på panelen
        }
    }
}//View


public class Main {
    public static void main(String[] args) {
        . . .
        View view = new View();
        Counter counter = new Counter();
        counter.addObserver(view);
        . . .
    }
}//Main
```

Uppgift 9.

```
public class Student implements Cloneable, Serializable {
   private List<Course> courses;
   private String name;
   private int year;
   private double gpa;

     public Student clone() {
          try {
              Student copy = (Student) super.clone();
              copy.courses = new ArrayList<Course>();
              for (Course course : courses) {
                   copy.courses.add(course.clone());
              }
              return copy;
          } catch (CloneNotSupportedException cnse) {
              return null;
          }
     }//clone
}//Student
```

Med användning av kopieringskonstruktor:

```
public class Student implements Cloneable, Serializable {
   private List<Course> courses;
   private String name;
   private int year;
   private double gpa;

     public Student(Student other) {
          this.name = other.name;
          this.year = other.year;
          this.gpa = other.gpa;
          this..courses = new ArrayList<Course>();
          for (Course course : other.courses) {
                   this.courses.add(course.clone());
          }
     }
     public Student clone() {
          return new Student(this);
     }//clone
}//Student
```

Observera att i tesen fanns inga konstruktorer för klassen Student angivna, varför lösningar som använder icke-definierade konstruktorer inte accepteras.

Uppgift 10.

a)

```
public class Volume implements Shape {
    private Shape obj;
    private double height;
    public Volume(Shape obj, double height) {
        this.obj = obj;
        this.height = height;
    }
    public double getWidth() {
        return obj.getWidth();
    }
    public double getLength() {
        return obj.getLength();
    }
    public double getArea() {
        return obj.getArea();
    }
    public void draw() {
        obj.draw();
    }
    public void erase() {
        obj.erase();
    }
    public void move(double dx, double dy) {
        obj.move(dx, dy);
    }
    public void resize(double factor) {
        height = height * factor;
        obj.resize(factor);
    }
    public double getVolume() {
        return obj.getArea()*height;
    }
}
```

b)

```
Volume cube = new Volume( new Rectangle(0 ,0,10, 10), 10);
System.out.println(cube.getVolume());
```

Uppgift 11.

```
import java.util.*;
public class Assistants {
    private Map<String, Set<String>> map;
    public Assistants() {
        map = new TreeMap<String, Set<String>>();
    }
    public void addTA(String course, String taName) {
        Set<String> tas = map.get(course);
        if (tas == null) {
            tas = new TreeSet<String>();
            map.put(course, tas);
        }
        tas.add(taName);
    }//addTA
    public void displayTAsPerCourse() {
        for (String c : map.keySet()) {
            System.out.println(c);
            for (String taName: map.get(c))
                System.out.println(taName);
        }
    }//displayTAsPerCourse
}// Assistants
```

```
    public void displayTAsPerCourse() {
        Iterator<Map.Entry<String, Set<String>>> itr1 = map.entrySet().iterator();
        while (itr1.hasNext()) {
            Map.Entry<String, Set<String>> me = itr1.next();
            System.out.println(me.getKey());
            Iterator<String> itr2 = me.getValue().iterator();
            while (itr2.hasNext()) {
                System.out.println("   " + itr2.next());
            }
        }
    }//displayTAsPerCourse

    public void displayTAsPerCourse() {
        Set<String> keys = map.keySet();
        Iterator<String> itr1 = keys.iterator();
        while (itr1.hasNext()) {
            String course = itr1.next();
            System.out.println(course);
            Set<String> names = map.get(course);
            Iterator<String> itr2 = names.iterator();
            while (itr2.hasNext()) {
                System.out.println("   " + itr2.next());
            }
        }
    }//displayTAsPerCourse
```

Uppgift 12.

Klassen Shutdown implementerar interfacet Runnable:

```
public class Shutdown implements Runnable {
    private int secs;
    private String msg;

    public Shutdown(int secs, String msg) {
        this.secs = secs;
        this.msg = msg;
    }//construktor

    public static void shutdownNow(String msg) {
        System.out.println(msg);
        System.exit(1); // cause entire JVM to shut down
    }// shutdownNow

    public static void delayedShutdown(int seconds, String msg) {
        Thread t = new Thread(new Shutdown(seconds, msg));
        t.start();
    }// delayedShutdown

    public void run() {
        try {
        Thread.sleep(secs*1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        shutdownNow(msg);
    }//run
}//Shutdown
```

Alternativ lösning som där klassen Shutdown utökar klassen Thread:

```
public class Shutdown extends Thread {
    private int secs;

    private String msg;
        public Shutdown(int secs, String msg) {
        this.secs = secs;
        this.msg = msg;
    }//construktor

    public static void shutdownNow(String msg) {
        System.out.println(msg);
        System.exit(1); // cause entire JVM to shut down
    }// shutdownNow

    public static void delayedShutdown(int seconds, String msg) {
        Thread t = new Shutdown(seconds, msg);
        t.start();
    }// delayedShutdown

    public void run() {
        try {
        Thread.sleep(secs*1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        shutdownNow(msg);
    }//run
}//Shutdown
```