Tentamen för TDA550 **Objektorienterad programvaruutveckling, fk**

DAG: 17-01-13 TID: 14:00 – 18:00

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:		Tisdag 14/2 kl 12-13 och torsdag 16/2 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 svar 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.

a) Vad skriver programmet ut? Motivera!

```
public class A {
    public void f() {
        System.out.println("A.f");
    }
    public void h() {
        f();
    }
}//A
```

```
public class B extends A {
    public void g() {
        h();
    }
}//B
```

```
public class C extends B {
    public void g(A obj) {
        obj.h();
        g();
    }
    public void f() {
        System.out.println("C.f");
    }
}//C
```

```
public class Main {
    public static void main(String[] args) {
        C c = new C();
        c.g( new B() );
    }
}//Main
```

b) Studera klasserna nedan.

```
public interface Int {
    public void h();
    public void f();
}

public class Base {
    public void f() {}
    public void g() {}
}

public class Sub1 extends Base {
    public void h() {}
}

public class Sub2 extends Base implements Int {
    public void h() {}
}
```

Vilka av metodanropen nedan är tillåtna och vilka ger kompileringsfel? Motivera!

i)
```
Base obj1 = new Sub1();
obj1.f();
obj1.g();
obj1.h();
```

ii)
```
Base obj2 = new Sub2();
obj2.f();
obj2.g();
obj2.h();
```

iii)
```
Int obj3 = new Sub2();
obj3.f();
obj3.g();
obj3.h();
```

(5 poäng)

Uppgift 2.

Betrakta klasserna:

```
public class USBStick {
   public USBStick(int x) { ... }
   public int occupiedSpace() { ... }
}

public class DVDMemory {
   public DVDMemory(int x) { ... }
   public int getMemoryUsage() { ... }
}

public class SSDUnit {
   public SSDUnit(int x) { ... }
   public int memoryConsumption() { ... }
}
```

samt följande klientkod:

```
public class Main {
   public static long getMemoryUsage(List<Object> units) {
      long usage = 0;
      for (Object o : units) {
         if (o instanceof USBStick)
            usage += ((USBStick)o).occupiedSpace();
         else if ( o instanceof DVDMemory )
            usage += ((DVDMemory)o).getMemoryUsage();
         else if (o instanceof SSDUnit)
            usage += ((SSDUnit)o).memoryConsumption();
      }
      return usage;
   }

   public static void main(String[] args) {
      List<Object> units = new LinkedList<>();
      units.add(new USBStick(1000));
      units.add(new DVDMemory(80000));
      units.add(new SSDUnit(20000));
      System.out.println(getMemoryUsage(units));
   }
}
```

a) Koden ovan bryter mot en viktig designprincip, vilken?

b) Refaktorera koden så att den baseras på polymorfism.

(5 poäng)

Uppgift 3.

En Brittisk webshop prissätter sina varor i valutan USD. Kunder inom Storbrittanien betalar i valutan GBP, övriga europeiska kunder i EURO och resten av världen i USD. En PriceCalculator är ett objekt som räknar ut det totalpris inklusive frakt som kunden skall betala i den valda valutan:

```
public abstract class PriceCalculator {
   public static boolean isBrittish(String country) { ... }
   public static boolean isEuropean(String country) { ... }
   public abstract float calculateTotalPrice(float subTotal, float weight, String country);
}

public class GBPCalculator extends PriceCalculator {
   public GBPCalculator(String country) { ... }
   public float calculateTotalPrice(float subTotal, float weight, String country) { ... }
}

public class EUROCalculator extends PriceCalculator {
   public EUROCalculator(String country) { ... }
   public float calculateTotalPrice(float subTotal, float weight, String country) { ... }
}

public class USDCalculator extends PriceCalculator {
   public USDCalculator(String country) { ... }
   public float calculateTotalPrice(float subTotal, float weight, String country) { ... }
}

public class BadMain {
   public static void main(String[] arg) {
      String country = arg[0];
      PriceCalculator priceCalculator;
      float subTotal = 12345.0F;  // USD (priset antas vara satt i dollar)
      float weight = 123.3F;
      if (PriceCalculator.isBrittish(country))
         priceCalculator = new GBPCalculator(country);
      else if (PriceCalculator.isEuropean(country))
         priceCalculator = new EUROCalculator(country);
      else
         priceCalculator = new USDCalculator(country);

      System.out.println(priceCalculator.calculateTotalPrice(subTotal, weight, country));
   }
}
```

a) Klassen BadMain bryter mot en viktig designprincip, vilken?

b) Refaktorera koden enligt designmönstret *Factory*. (De fyra första klasserna skall ej ändras.)

(5 poäng)

Uppgift 4.

Antag att följande klasser är givna:

```
public class Furniture {...}
public  class CoffeeTable extends Table {...}
public class Table extends Furniture{...}
public class DinnerTable extends Table {...}
```

Antag vidare att vi I ett program har gjort följande deklarationer:

```
Object o;
Furniture f;
CoffeeTable ct;
Table t;
DinnerTable dt;
List<? extends Furniture> lef;
List<? extends Table> let;
List<? super Table> lst;
```

Ange för var och en av nedanstående satser om satsen är korrekt eller ger kompileringsfel. Motivera!

a) lef.add(t);
b) lef.add(o);
c) lst.add(t);
d) lst.add(o);
e) let.add(dt);
f) o = lef.get(1);
g) f = let.get(1);
h) ct = let.get(1);
i) t = lst.get(1);
j) o = lst.get(1);

(5 poäng)

Uppgift 5.

a) Rangordna nedanstående villkoren efter avtagande logisk styrka (A => B):
   i) i < 20
   ii) true
   iii) 40 < i || i < 30
   iv) false
   v) 0 > i && i < 10

b) Uppfyller någon av implementationerna ArraySearchImpl1 och ArraySearchImpl2 kontraktet för ArraySearch? Är de utbytbara mot varandra? Motivera!

```
public interface ArraySearch {
   //@Pre a != null
   //@Post returns the index of an arbitrary occurrence of x if such an element exists in a, otherwise -1
   int find(int[] a,int x);
}

public class ArraySearchImpl1 implements ArraySearch {
   //@Pre True
   //@Post returns the index of the first occurrence of x if such an element exists in a, otherwise -1
   public int find(int[] a,int x) { ... }
}

public class ArraySearchImpl2 implements ArraySearch {
   //@Pre a != null && a.size > 0
   //@Post returns the index of an arbitrary occurrence of x if such an element exists in a
   public int find(int[] a,int x) { ... }
}
```

(4 poäng)

Uppgift 6.

En gles vektor är en vektor där de flesta elementen har värdet 0. Exempel på en gles vektor med 14 element varav 2 är skilda från 0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0.0	0.0	0.0	0.0	0.0	0.0	3.12	0.0	0.0	0.0	0.0	4.21	0.0	0.0

Att lagra glesa vektor i ett fält av typen **double**[] är ineffektivt avseende minnesåtgång, eftersom även alla nollor måste lagras. En betydligt bättre metod är att endast lagra element som inte är 0 i en Map, där nyckel/värde-paret är elementets index respektive elementets värde.

En gles matris specificeras av nedanstående interface:

```
public interface  SparseVector {
    /**
     * Puts val in element with index i.
     * @param i index
     * @param val the new value
     * @throws IllegalArgumentException if i < 0 or i >= the size of this vector
    */
    public void put(int i, double val);

    /**

     * Gets the value in the element with index i.
     * @param i index
     * @throws IllegalArgumentException if i < 0 or i >= the size of this vector
    */
    public double get(int i);

     /**
     * Returns the size of this vector.
     * @return the size of this vector
    */
    public int size();

    /**
     * Computes the dot product of this vector and v.
     * @param v the other vector
     * @return the dot product
     * @throws IllegalArgumentException if the two vectors have different sizes
    */
    public double dot(SparseVector v);
}//SparseVector
```

Skriv en klass SparseVectorMap som implementerar interfacet SparseVector. Internt skall klassen SparseVectorMap representera den glesa matrisen med hjälp av en Map. Klassen ska ha två konstruktorer:

public SparseVectorMap(**int** size)	som skapar en instans av SparseVectorMap med storleken size, i vilken alla element är 0.
public SparseVectorMap(**double**[] v)	som skapar en instans av SparseVectorMap med samma element som i fälet v.

Tips: Metoden dot(SparseVector v) ska beräkna skalärprodukten (eng. dot product) på följande sätt:

$$a \cdot b = a_0 b_0 + a_1 b_1 + ... + a_{n-1} b_{n-1}$$

Tänk på att parametern v har typen SparseVector och inte SparseVectorMap.

(10 poäng)

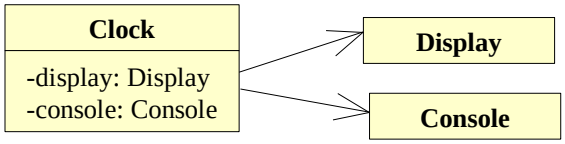
Uppgift 7.

Nedan har vi de tre klasserna Clock, Display och Console. Klassen Clock används för att avbilda en klocka och innehåller bl.a. metoden tick som räknar fram tiden. Klasserna Display och Console har båda metoder för att visa aktuell tid, metoderna show respektive print.

```
public class Clock {
    private Time time;
    private Display display;
    private Console console;
    public Clock(Display display, Console console) {
        this.display = display;
        this.console = console;
    }//constructor
    public void tick() {
        time.increase();
        display.show(time);
        console.print(time);
    }//tick
    //more methods and constructors not shown here
}//Clock

public class Display {
    public void show(Time time) { ... }
    ...
}//Display

public class Console {
    public void print(Time time) { ... }
    ...
}//Console
```



Designen har bristen att Clock är beroende av klasserna Display och Console. Det borde vara tvärtom.

Ändra designen så att designmönstret Observer realiseras (genom att använda Observable och Observer eller genom att använda PropertyChangeSupport och PropertyChangeListener). Det räcker om du visar koden för klasserna Clock och Display (Console blir analog med Display).

(6 poäng)

Uppgift 8.

Följande interface och klass är givna:

```
public interface EnemyAttacker {
    public void fireWeapon();
    public void driveForward();
    public void assignDriver(String driver);
}

public class EnemyRobot {
    public void smashWithHand() {
        //code omitted
    }
    public void walkForward() {
        //code omitted
    }
    public void reactToHuman(String human) {
        //code omitted
    }
}
```

Skriv en klass EnemyRobotAdapter som implementerar desigmönstret *Adapter* för att anpassa gränssnittet för typen EnemyRobot till typen EnemyAttacker.

(5 poäng)

Uppgift 9.

Ett rationellt tal $\frac{p}{q}$ består av en täljare p och en nämnare q (där både p och q är heltal samt $q \neq 0$). Betrakta klassen Rational nedan som avbildar rationella tal:

```
//Class invariant: q > 0
public class Rational {
    private int p;
    private int q;
    public Rational(int p, int q) {
        this.p = p;
        this.q = q;
    }
    public Rational add(Rational other) {
        return new Rational(p*other.q + other.p*q, q*other.q);
    }
     // other artimethic operations omitted
    public boolean equals(Object other) {
        if (this == other)
            return true;
        if (other == null || getClass() != other.getClass())
            return false;
        Rational o = (Rational) other;
        return p == o.p && q == o.q;
    }
    private int gcd(int x,int y) {
        // returns the greatest common divisor of x and y.
        //code not shown here
    }
}//Rational
```

a) Rent matematiskt gäller förstås att t.ex. 1/3 + 1/3 = 2/3, men givet

```
Rational a = new Rational(1, 3);
Rational b = new Rational(2, 3);
```

kommer b.equals(a.add(a)) att returnera **false**! Varför?

Modifiera klassen på så sätt att x.equals(y) ger **true** om x och y avbildar två rationella tal som är matematiskt lika, dvs att anropet b.equals(a.add(a)) ovan skall ge värdet **true**. Metoderna equals och gcd får inte förändras.

Tips 1: När skall klassinvarianten vara uppfylld? Tips 2: Talet noll kan representeras som 0/1.

b) Är det lämpligt att lagra objekt av klassen Rational i ett HashSet? Om inte, komplettera klassen med det som saknas?

c) Gör klassen Membership kopierbar. Du kan förutsätta att SomeClass är kopierbar.

```
public class Membership {
    private String name;
    private int age;
    private LinkedList<SomeClass> someObjects;

    public Membership() {
        someObjects = new LinkedList<>();
    }
     // other members omitted
}//Membership
```

(7 poäng)

Uppgift 10.

Betrakta deklarationerna:

```
public class E1 extends Exception { ... }
public class E2 extends E1 { ... }
public class E3 extends E2 { ... }
public class E4 extends E1 { ... }

public class Base {
    public void f() throws E2 { ... }
    public void g() throws E1 {
        if (some condition)
            System.out.print("g succeeded");
        else throw new ... ;
    }
}

public class Sub1 extends Base{
    public void f() throws E1 { ... }
}

public class Sub2 extends Base{
    public void f() throws E3 { ... }
}

public class Sub3 extends Base{
    public void f() throws E2 {
        throw new E3();
    }
}

public class Sub4 extends Base{
    public void f() throws E3 {
        throw new E2();
    }
}
```

a) Är deklarationen av Sub1, Sub2, Sub3 respektive Sub4 typkorrekt? Motivera!

b) Vilka utskrifter är möjliga vid exekvering av kodavsnittet nedan? Motivera?

```
Base obj = new Base();
try {
   try {
      obj.g();
   }
   catch ( E2 e ) {
        System.out.print("catched E2");

   }
   catch ( E4 e ) {
        System.out.print("catched E4");
   }
}
catch ( E1 e ) {
    System.out.print("catched E1");
}
```

(4 poäng)

Uppgift 11.

Betrakta klassen ClothingStore nedan:

```
public class  ClothingStore {
    private int  sweatersInStock = 0;
    private int  outwardConnectionCount;
    public void  addToStock(int  quantity) {
        sweatersInStock = sweatersInStock + quantity;
    }// addToStock

    public void  removeFromStock(int  quantity) {
        if  (quantity > sweatersInStock)
            throw new IllegalArgumentExecption("To big quantity");
        sweatersInStock = sweatersInStock - quantity;
    }// addToStock

    public int  sweatersInStock() {
        return  sweatersInStock;
    }// numberInStock

    //more code that is not rellevant for this assignment
}//ClothingStore
```

a) Förklara varför klassen inte är trådsäker och visa med ett exempel hur en instans av klassen kan hamna i ett inkonsistent tillstånd.

b) Gör klassen trådsäker. Din lösning skall så långt som möjligt bibehålla parallella bearbetning.

(4 poäng)

Tentamen 170113 - LÖSNINGSFÖRSLAG

Uppgift 1.

a) Utskriften blir:

A.f
C.f

b)

i) anropet obj1.h() ger kompileringsfel eftersom den statiska typen Base inte har metoden h.

ii) anropet obj2.h() ger kompileringsfel eftersom den statiska typen Base inte har metoden h.

iii) anropet obj3.g() ger kompileringsfel eftersom den statiska typen Int inte har metoden g.

Uppgift 2.

a) Koden bryter mor *Dependency Inversion Principle* genom att den beror av specifika klasser och metodnamn snarare än ett gemensamt gränssnitt. Ett symptom är fallanalysen i metoden getMemoryUsage som måste skrivas om vid tillägg av nya minnesenheter.

b)

```
public abstract class MemoryUnit {
   public abstract long getMemoryUsage();
}//MemoryUnit

public class USBStick extends MemoryUnit {
   public USBStick(long x) { ... }
   public long getMemoryUsage() { ... }
}//USBStick

public class DVDMemory extends MemoryUnit {
   public DVDMemory(long x) { ... }
   public long getMemoryUsage() { ... }
}//DVDMemory

public class SSDUnit extends MemoryUnit {
   public SSDUnit (long x) { ... }
   public long getMemoryUsage() { ... }
}//SSDUnit

public class Main {
   public static long getMemoryUsage(List<MemoryUnit> units) {
      long usage = 0;
      for (MemoryUnit u : units)
         usage += u.getMemoryUsage();
      return usage;
   }

   public static void main(String[] args) {
      List<MemoryUnit> units = new LinkedList<>();
      units.add(new USBStick(1000));
      units.add(new DVDMemory(80000));
      units.add(new SSDUnit(20000));
      System.out.println(getMemoryUsage(units));
   }
}//Main
```

Uppgift 3.

a) Klassen BadMain bryter mot *Open/closed-principen* som säger att en klass skall vara öppen för tillägg men stängd för ändringar.

b) Inför först fabriksklassen

```
public class PriceCalculatorFactory {
    public static PriceCalculator createPriceCalculator(String country) {
        if ( PriceCalculator.isBrittish(country) )
            return new GBPCalculator(country);
        else if ( PriceCalculator.isEuropean(country) )
            return new EUROCalculator(country);
        else
            return new USDCalculator(country);
    }
}//PriceCalculatorFactory
```

och skriv om klassen BadMain till GoodMain:

```
public class GoodMain {
    public static void main(String[] arg) {
        String country = arg[0];
        PriceCalculator priceCalculator =  PriceCalculatorFactory.createPriceCalculator(country);
        float subTotal = 12345.0F;  // USD (priset antas vara satt i dollar)
        float weight = 123.3F;
        System.out.println(priceCalculator.calculateTotalPrice(subTotal,weight,country));
    }
}//GoodMain
```

Vi ser att koden i main-metoden frikopplats från de valutaspecifika klasserna. Objekt av dem kan skapas utan att namnge deras klasser. Vid tillägg av nya valutor eller ändringar i implementeringsklasserna isoleras förändringarna till fabriksklassen.

Uppgift 4.

a) Kompileringsfel! Listan lef kan t.ex. vara av typen List<DinnerTable> och ett objekt av typen Table är ingen subtyp till Dinnertable.

b) Kompileringsfel! Listan lef kan t.ex. vara av typen List<DinnerTable> och ett objekt av typen Object är ingen subtyp till Dinnertable.

c) Ok! Listan lst kan vara av typen List<Table> eller List<Object>, och typen Table är subtyp till typen Object.

d) Kompileringsfel! Listan lst kan vara av typen List<Table> eller List<Object>, och typen Object är ingen subtyp till typen Table.

e) Kompileringsfel! Listan let kan vara av typen List<CoffeeTable>, och typen DinnerTable är ingen subtyp till typen CoffeeTable.

f) Ok! Typen Object är supertyp till alla andra typer.

g) Ok! Typen Furniture är supertyp till typen Table (och alla subtyper till Table).

h) Kompileringsfel! Listan let kan t.ex. vara av typen List<Table> och typen CoffeeTable är ingen subtyp till typen Table.

i) Kompileringsfel! Listan lst kan vara av typen List<Object> och typen Table är ingen supertyp till Object.

j) Ok! Typen Object är supertyp till alla andra typer.

Uppgift 5.

a) false => 0 > i && i < 10 => i < 20 => 40 < i || i < 30 => true

b) ArraySearchImpl1 har starkare specifikation än ArraySearch eftersom ArraySearchImpl1 har svagare förvillkor och starkare eftervillkor. Vi kan därför använda ArraySearchImpl1 i klienter som är kompatibla med ArraySearch.

ArraySearchImpl2 däremot har svagare specifikation än ArraySearch eftersom ArraySearchImpl2 har starkare förvillkor och svagare eftervillkor. ArraySearchImpl2 kan därför inte användas i en klient som förväntar sig en implementering kompatibel med ArraySearch.

Uppgift 6.

```
public class SparseVectorMap implements SparseVector {
  private Map<Integer, Double> map;
  private int size;

  public SparseVectorMap(int size) {
      this.size = size;
      map = new HashMap<Integer, Double>();
  }

  public SparseVectorMap(double[] v) {
      this.size = v.length;
      map = new HashMap<Integer, Double>();
      for (int i = 0; i < v.length; i++) {
          if (v[i] != 0) {
              map.put(i, v[i]);
          }
      }
  }

  public void put(int index, double val) {
      if (index < 0 || index >= size)
          throw new IllegalArgumentException();
      if (val == 0) {
          map.remove(index);
      } else {
          map.put(index, val);
      }
  }

  public double get(int index) {
      if (index < 0 || index >= size)
          throw new IllegalArgumentException();
      Double d = map.get(index);
      if (d == null) {
          return 0.0;
      } else {
          return d;
      }
  }

  public int size() {
      return size;
  }

  public double dot(SparseVector v) {
      if (size != v.size())
          throw new IllegalArgumentException("Vector lengths are different");
      double sum = 0;
      for (Map.Entry<Integer, Double> e : map.entrySet()) {
          sum += e.getValue() * v.get(e.getKey());
      }
      return sum;
  }
}//SparseVectorMap
```

Uppgift 7.

Med användning av Observer och Observable:

```
import java.util.Observable;
public class Clock extends Observable {
    private Time time;

    public void tick() {
        time.increase();
        setChanged();
        notifyObservers();
    }//constructor

    public Time getTime() {
        return time;
    }//getTime

    //more methods and constructors not shown here

}//Clock
```

```
import java.util.Observer;
import java.util.Observable;
public class Display implements Observer {
    private Clock clock;

    public Display(Clock clock) {
        this.clock = clock;
        clock.addObserver( this );
    }//constructor

    private void show(Time time) {
        ...
    }//show

    public void update(Observable obs, Object o) {
        show(clock.getTime());
    }//update
}//Display
```

Klassen Console redovisas här med en alternativ lösning:

```
import java.util.Observer;
import java.util.Observable;
public class Console implements Observer {

    public Console(Clock clock) {
        clock.addObserver(this);
    }//constructor

    private void print(Time time) {
        ...
    }//print

    public void update(Observable obs, Object o) {
        print(((Clock) obs).getTime());
    }//update
}//Console
```

Med användning av PropertyChangeSupport och PropertyChangeListener:

```
import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;
public class Clock {
  private Time time;
  private final PropertyChangeSupport pcs = new PropertyChangeSupport(this);

  public void addPropertyChangeListener(PropertyChangeListener listener) {
      this.pcs.addPropertyChangeListener(listener);
  }

  public void removePropertyChangeListener(PropertyChangeListener listener) {
      this.pcs.removePropertyChangeListener(listener);
  }

  public void tick() {
      time.increase();
      pcs.firePropertyChange("", true, false);
  }//constructor

  public Time getTime() {
      return time;
  }//getTime

  //more methods and constructors not shown here

}//Clock


import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
public class Display implements PropertyChangeListener {
  private Clock clock;

  public Display(Clock clock) {
      this.clock = clock;
      clock.addPropertyChangeListener(this);
  }//constructor

  private void show(Time time) {
      ...
  }//show

  public void propertyChange(PropertyChangeEvent e) {
      show(clock.getTime());
  }//propertyChange
}//Display
```

Klassen Console redovisas här med en alternativ lösning:

```
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeEvent;
public class Console implements PropertyChangeListener {

  public Console(Clock clock) {
      clock.addPropertyChangeListener(this);
  }//constructor

  private void print(Time time) {
      ...
  }//print

  public void propertyChange(PropertyChangeEvent ev) {
      print(((Clock2) ev.getSource()).getTime());
  }//propertyChange
}//Console
```

Uppgift 8.

```
public class EnemyRobotAdapter implements EnemyAttacker {
  private EnemyRobot theRobot = new EnemyRobot();
  public EnemyRobotAdapter(EnemyRobot newRobot){
      theRobot = newRobot;
  }

  @Override
  public void fireWeapon() {
      theRobot.smashWithHand();
  }
  public void driveForward() {
      theRobot.walkForward();
  }
  public void assignDriver(String driver) {
      theRobot.reactToHuman(driver);
  }
}//EnemyRobotAdapter
```

Uppgift 9.

a) Metoden equals är korrekt skriven.

Det första problemet med implementeringen av klassen är att klassinvarianten inte är uppfylld. För det första bör undantaget IllegalStateException kastas i konstruktorn om q är noll eftersom nämnaren inte kan vara noll i ett rationellt tal.

För det andra tillåter klassen oändligt många olika representationer av samma rationella tal. T.ex. kan 1/3 lagras som 2/6, 15/45, -1/-3, o.s.v. En lösning är att låta konstruktorn reducera bråket genom att förkorta det så långt det går, samt bestämma att negativa tal alltid representeras med negativ täljare och positiv nämnare. Då uppfylls dessutom klassinvarianten.

Skriv alltså om konstruktorn:

```
public Rational(int p, int q) {
    this.p = p;
    if ( q == 0 )
        throw new IllegalStateException("Zero denominator");
    this.q = q;
    normalize();
}
```

och utför förkortningen av bråket med gcd, samt justera tecknet:

```
private void normalize() {
    int gcd = gcd(p,q);
    p /= gcd;
    q /= gcd;
    if ( q < 0 ) {
        p = -p;
        q = -q;
    }
}
```

Ex gcd(20,28) är 4 och bråket kan förkortas till 5/7.

b) Klassen måste överskugga hashCode-metod:

```
    public int hashCode() {
        return 31*p + q;
    }
```

c) Deklarera först

public class Membership **implements** Cloneable

och inför en clone-metod

```
public Membership clone() {
    Membership result = null;
    try {
        result = (Membership)super.clone();
        result.someObjects = (LinkedList<SomeClass>)someObjects.clone();
        for (int i = 0; i < someObjects.size(); i++ )
            result.someObjects.set(i, someObjects.get(i).clone());
    }
    catch ( CloneNotSupportedException e) {
        throw new InternalError();
    }
    return result;
}
```

Uppgift 10.

a) Sub2 och Sub3 är typkorrekta.

I Sub1 deklarerar f ett undantag av vidare typ än basklassens f. En sådan överskuggning är inte tillåten.

I Sub4 kastar f ett undantag av vidare typ än den deklarerar och det är förstås inte tillåtet.

b) Man kan få vilken som helst av utskrifterna (men bara en):

- g succeeded
- catched E2
- catched E4
- catched E1

Base.g måste inte kasta ett undantag men kan om den vill, kasta vilket som helst av E1, E2, E3 och E4.

Undantagen E2 och E3 fångas av **catch** (E2 e), E4 av **catch** (E4 e), samt E1 av den yttre **catch** (E1 e).

Uppgift 11.

a) I klassen ClothingStore är inte metoderna addToStock och removeFromStock trådsäkra. I och med detta kan följande inträffa:

Instansvariabeln sweatersInStock anger antalet produkter i lager. Anta att den har värdet 150 när två trådar tråd1 och tråd2 samtidigt exekverar metoden addToStock respektive removeFromStock. I beräkningen av satserna:

```
sweatersInStock = sweatersInStock + quantity;
```

respektive

```
sweatersInStock = sweatersInStock - quantity;
```

läser både tråd1 och tråd2 av värdet 150 på instansvariabeln sweatersInStock.

tråd1 adderar 70 till värdet 150 och får resultatet 220.
tråd2 subtraherar 55 från värdet 150 och får resultatet 95.
tråd2 lagrar värdet 95 i instansvariabel sweatersInStock.
tråd1 lagrar värdet 220 i instansvariabeln sweatersInStock.

Effekten blir att instansvariabeln sweatersInStock nu har värdet 220, men det faktiska antalet produkter i lager är 165.

b)

```
public class  ClothingStore {
   private int  sweatersInStock = 0;
   private int  outwardConnectionCount;
   public synchronized void  addToStock(int  quantity) {
        sweatersInStock = sweatersInStock + quantity;
   }// addToStock

   public synchronized void  removeFromStock(int  quantity) {
       if  (quantity > sweatersInStock)
           throw new IllegalArgumentExecption("To big quantity");
       sweatersInStock = sweatersInStock - quantity;
   }// addToStock

   public int  sweatersInStock() {
       return  sweatersInStock;
   }// numberInStock

   //more code that is not rellevant for this assignment
}//ClothingStore
```