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

DAG: 17-08-16 TID: 8:30 – 12:30

Ansvarig: Christer Carlsson, ankn 1038

Förfrågningar: Alex Gerdes, ankn 6154

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 12/9 kl 12-13 och måndag 18/9 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 klasserna Bird och Penguin som finns i paketet aviator:

```
package aviator;
public abstract class Bird {
    protected int a = 1;
    public abstract void identify();
    protected void shout(){
        System.out.println("Kaw-Kaw");
    }//identify

    public void action() {
        System.out.println("Fear me" + this.a + " times");
    }//action

    public void flyAround() {
        System.out.println("Wooooosh, I am a");
        identify();
        shout();
        System.out.println("You should");
        action();
    }//flyAround

    protected int getA() {
        return a;
    }//getA

    protected void addToA() {
        a++;
    }//addToA
}//Bird
```

```
package aviator;
public class Penguin extends Bird {
    private int a;
    public Penguin() {
        a = 2;
    }//constructor

    @Override
    public void identify() {
        System.out.println("Penguin #" + a +
                           " reporting for duty!");
    }//identify

    public void shout() {
        System.out.println("Shouting is uncivilized...");
    }//shout

    public void action(int times) {
        System.out.println("Swim "+  (times + getA()) +
                           " times!");
    }//action

    public void flyAround() {
        System.out.println("Sometimes I dream" +
                           " I can fly like this: ");
        super.flyAround();
    }//flyAround
}//Penguin
```

I paketet bungler finns klassen Program:

```
package bungler;
public class Program{
    public static void act1() {
        Bird b = new Bird();
        b.flyAround();
    }//act1

    public static void act2() {
        Bird b = new Penguin();
        act3(b);
    }//act2

    public static void act3(Penguin p) {
        Bird b    = new Penguin();
        b.shout();
    }//act3

    public static void act4() {
        Bird b    = new Penguin();
        b.action(15);
    }//act4

    public static void act5() {
        Bird b = new Penguin();
        b.flyAround();
    }//act5

    public static void act6() {
        Bird b    = new Penguin();
        b.identify();
    }//act6

    public static void act7() {
        Bird b = new Penguin();
        b.addToA();
        ((Penguin)b).action(4);
    }//act7
}//Program
```

Vad inträffar för vart och ett av nedanstående anrop (ger kompileringsfel, ger exekveringsfel, ger utskriften ”xxxx”, etc). Svaren skall motiveras!

a) Program.act1() b) Program.act2() c) Program.act3(**new** Penguin()) d) Program.act4()

e) Program.act5(5) f) Program.act6() g) Program.act7()

Anm: Klassen Program går eventuellt inte att kompilera, men bortse från detta när du besvarar frågorna. Tänk dej att både kompilering och exekvering görs när respektive metod anropas.

(7 poäng)

Uppgift 2.

Den driftiga småföretagaren flitiga Lisa har en verksamhet som omfattar ett gym, en skönhetssalong, samt en cykelverkstad i källaren. Lisa har ärvt en gammal programvara som bl.a. räknar ut momsen (*eng.* VAT) som skall redovisas för dagskassan. De olika tjänsterna har olika momssatser (6% för gymmet, 25% för skönhetssalongen och 12% för cyckelverkstaden), vilket framgår av koden nedan:

```
public class GymCard {
    private float price;
    public GymCard(float x) {
        price = x;
    }
    public float getPrice() {
        return price;
    }
    // Other code omitted
}//GymCard
```

```
public class BeautyCare {
    private float charge;
    public BeautyCare(float x) {
        charge = x;
    }
    public float getCharge() {
        return charge;
    }
    // Other code omitted
}//BeautyCare
```

```
public class BikeRepair {
    private float cost;
    public BikeRepair(float x) {
        cost = x;
    }
    public float getCost() {
        return cost;
    }
    // Other code omitted
}//BikeRepair
```

```
import java.util.ArrayList;
public class SmallBuiseness {
    private ArrayList receipts = new ArrayList();
    public void addReceipt(Object receipt) {
        receipts.add(receipt);
    }
    public float computeVAT() {
        float totalVAT = 0;
        for (Object o : receipts) {
            if (o instanceof GymCard)
                totalVAT += ((GymCard)o).getPrice() * 0.06f;
            else if (o instanceof BeautyCare)
                totalVAT += ((BeautyCare)o).getCharge() * 0.25f;
            else if (o instanceof BikeRepair)
                totalVAT += ((BikeRepair)o).getCost() * 0.12f;
        }
        return totalVAT;
    }//computeVAT
    // Other code omitted
}//SmallBuiseness
```

a) Programvaran lämnar en del att önska. Förklara vad som är det huvudsakliga problemet. (1 poäng)

b) Skriv om koden så att detta problem elimineras, genom att nyttja polymorfism! (5 poäng)

Uppgift 3.

Betrakta nedanstående interface och klasser (utlämnad kod är betydelselös för uppgiften):

```
public interface PortableDevice {…}
public class Computer {…}
public class Laptop extends Computer implements PortableDevice {…}
```

Antag också att följande deklarationer är gjorda:

```
Object obj;
PortableDevice portable;
Computer computer;
Laptop laptop;
List<? extends PortableDevice> peList;
List<? extends Computer> ceList;
List<? super Laptop> lsList;
```

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

a) peList.add(portable);

b) peList.add(laptop);

c) portable = peList.get(0);

d) lsList.add(laptop);

e) portable = lsList.get(0);

f) obj = lsList.get(0);

g) peList = **new** ArrayList<Laptop>();

h) ceList = **new** ArrayList<Laptop>();

i) pelist = ceList;

j) List<?> list = ceList;

(5 poäng)

Uppgift 4.

a) Antag att vi har nedanstående specifikationer för metoden

public double average(List<Things> list)

som beräknar och returnerar genomsnittliga vikten av objekten i listan list.

S1) @return average weight of Things in the list or 0.0 if the list is empty

S2) @requires list is not empty
@return average weight of Things in the list

S3) @return average weight of Things in the list
@throws IllegalArgumentException if the list is empty

S4) @return average weight of Things in the list
@throws RuntimeException if the list is empty

OBS: Klassen IllegalArgumentException är en subklass till RuntimeException.

i) Vilka av ovanstående specifikationer är starkare än specifikation S1. Motivera ditt svar!

ii) Vilka av ovanstående specifikationer är starkare än specifikation S2. Motivera ditt svar!

iii) Vilka av ovanstående specifikationer är starkare än specifikation S3. Motivera ditt svar!

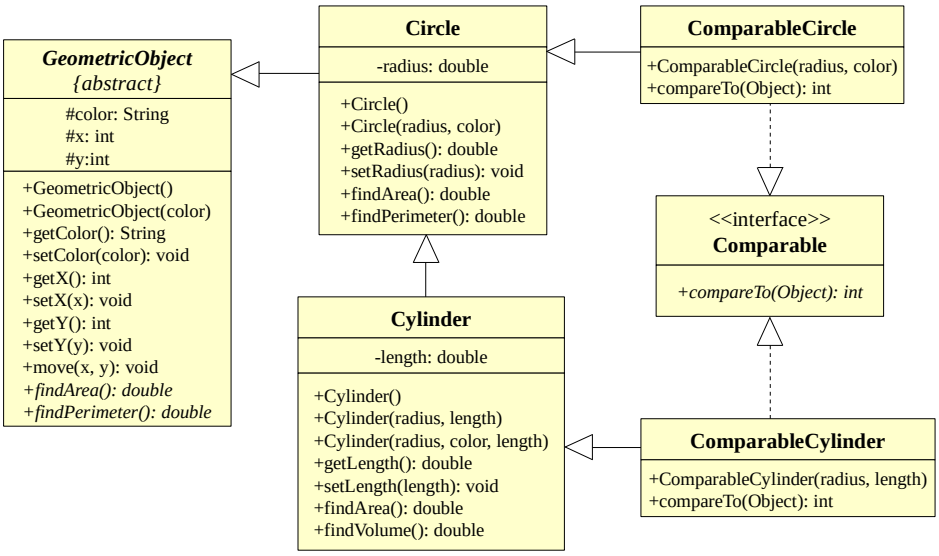
iv) Vilka av ovanstående specifikationer är starkare än specifikation S4. Motivera ditt svar!

(4 poäng)

b) Förklara hur starkare/svagare specifikationer hänger ihop med *Liskov Substitution Principle*. (2 poäng)

Uppgift 5.

På en nyligen avslutad grundkurs i objektorienterad programutveckling illustrerades arv med följande klasser och interface (för att avbildar geometriska objekt som har en viss färg och en viss placering i det två-dimensionella rummet):



Det är i och för sig något suspekt att kunna positionera tre-dimensionella objekt i ett två-dimensionellt rum, men bortse från detta när du besvarar nedanstående frågor.

a) Identifiera och förklara de brister som finns i ovanstående design med avseende på de arvsrelationer som finns. (2 poäng)

b) Visa med ett (minimalt) UML-diagram en alternativ design, som inte lider av de brister du identifierat i deluppgift a). (2 poäng)

c) Betrakta synligheten av instansvariablerna i respektive klass. Kommentera! (2 poäng)

d) Betrakta uppsättningen konstruktorer i respektive klass. Kommentera! (2 poäng)

e) Finns det något att sägas om hur tillstånden i klassen GeometricObject har avbildats (dvs vilka instansvariabler som har specificerats och typerna på dessa). (1 poäng)

Uppgift 6.

Betrakta nedanstående interface och klass:

```
public interface ImperialSensor {
   // returns: distance traveled in miles
   public double getDistance();

   // returns: speed in miles per hour
   public double getSpeed();
}//ImperialSensor
```

```
public class Client {
    private ImperialSensor sensor;

    public Client(ImperialSensor sensor) {
        this.sensor = sensor;
    }//constructor

    public void calculate() {
        double distance = sensor.getDistance();
        double speed = sensor.getSpeed();
        . . .
    }//calculate
    . . .
}//Client
```

Klassen Client använder en sensor av typen ImperialSensor, problemet är dock att den enda konkreta sensor som finns tillgänglig är av typen MetricSensor med följande utseende:

```
public class MetricSensor {
   // returns: distance traveled in meters
   public double readDistance() { . . . }

   // returns: speed in meters per second
   public double readSpeed() { . . . }
}
```

För att avhjälpa detta problem är din uppgift att skriva en klass MetricSensorToImperialSensor som implementerar *designmönstret Adapter* för att anpassa gränssnittet för typen MetricSensor till gränssnittet av typen ImperialSensor.

Tips: Följande konstanter kan behövas

MILES_PER_METER = 0.000621371192
HOURS_PER_SEC = 0.000277777778

(6 poäng)

Uppgift 7.

a) Vad innebär det att en klass är *icke-muterbar* (*immutable*)? Redogör för vilka fördelar det finns med icke-muterbara klasser.

(2 poäng)

b) Klassen Transaction nedan är inte icke-muterbar. Förklara varför och gör den icke-muterbar.

```
import java.util.Calendar;
public class Transaction {
    private int amount;
    private Calendar date;

    public Transaction(int amount, Calendar date) {
        this.amount = amount;
        this.date = date;
    }//constructor

    public int getAmount () {
        return this.amount ;
    }//getAmount

    public Calendar getDate() {
        return this.date;
    }//getDate
}//Transaction
```

(3 poäng)

Uppgift 8.

En sändlista är ett kommunikationsmedel som bygger på e-post. Sändlistan är en lista av e-postadresser som representeras av en enda e-postadress. Ett brev som skickas till sändlistans e-postadress, vidarebefordras till samtliga adresser som finns i sändlistan. För att få breven som skickas till en sändlista måste man prenumera på sändlistan och en användare kan premunera på flera sändlistor.

Inom företag och organisationer är det vanligt att sändlistor används inom en begränsad krets, d.v.s. endast de som är medlemmar i organisationen eller de som arbetar på arbetsplatsen får prenumera på sändlistorna. Din uppgift är att färdigställa klassen MailListHandler nedan:

```
public class MailListHandler {
    private Set<String> authorizedMembers;
    private Map<String, Set<String>> listMap;

    public MailListHandler() {
        authorizedMembers = new HashSet<String>();
        listMap = new HashMap<String, Set<String>>();
    }//constructor

    public boolean addMember(String member) {
        //skall implementeras i deluppgift a)
    }//addMember

    public void removeMember(String member) {
        //skall implementeras i deluppgift b)
    }//removeMember

    public boolean add(String listName, Set<String> subscribers) {
        //skall implementeras i deluppgift c)
    }//add

    public Map<String, Set<String>> subscriberMap() {
        //skall implementeras i deluppgift d)
    }//subscriberMap
}//MailListHandler
```

Klassen håller reda på sändlistor och deras prenumeranter, där prenumeranterna tillhöra en begränsad krets.

Instansvariabeln authorizedMembers representerar mängden möjliga prenumeranter, d.v.s. de som ingår i organisationen eller tillhör arbetsplatsen. Dessa identifieras entydigt av sina e-postadresser (en String).

Instansvariabeln listMap är en avbildning (Map) där nycklarna utgörs av sändlistornas namn (en String). Värdet som hör ihop med en nyckel är en mängd av strängar (Set<String>). Strängarna i mängden är e-postadresserna till sändlistans prenumeranter. En e-postadress i denna mängd måste finnas med i mängden authorizedMembers.

a) Implementera metoden

   public boolean addMember(String member)

   Metoden skall lägga till en ny medlem member i mängden authorizedMembers. Om member redan finns i mängden authorizedMembers skall **false** returneras, annars **true**.

   (1 poäng)

b) Implementera metoden

   public void removeMember(String member)

   Metoden anropas när någon utgår ur organisationen eller slutar på arbetsplatsen. Medlemmen skall tas bort från mängden authorizedMembers och från samtliga sändlistor där hon eller han är prenumerant. Om member inte finns i mängden authorizedMembers skall IllegalArgumentException kastas, annars utförs borttagningarna.

   (3 poäng)

c) Implementera metoden

public boolean add(String listName, Set<String> subscribers)

Metoden skall försöka lägga in samtliga mailaddresser i mängden subscribers som prenumeranter till sändlistan med namnet listName. Följande krav gäller:

Om någon av mailadresserna i subscribers inte tillhör en medlem i organisationen eller en anställd i företaget (d.v.s. inte finns i mängden authorizedMembers) utförs ingen insättning alls och false returneras.

Annars finns det två fall:

- Om det inte finns någon sändlista med namnet listName så sätts en sådan in med alla adresser i subscribers som prenumeranter och **true** returneras.
- Om sändlistan redan finns läggs alla adresser i subscribers till i dess prenumerantmängd och **true** returneras.

(3 poäng)

d) Implementera metoden

public Map<String, Set<String>> subscriberMap()

Metoden skall returnera en avbildning (Map) där nycklarna utgörs av samtliga medlemmar som är prenumerant på någon sändlista och värdena utgörs av en mängd bestående av namnen på de sändlistor på vilka respektive medlem prenumererar. Om alla sändlistor saknar prenumeranter skall en tom avbildning returneras.

Tips: Utgå från vilka sändlistor som finns, inte från prenumeranterna.

(4 poäng)

Uppgift 9.

I ett programsystem som Pelle Hacker skall vidareutveckla finns klassen Account nedan:

```
public class Account {
    private int balance;

    public Account(int balance) {
        this.balance = balance;
    }//constructor

    //requires amount >= 0
    public void deposit(int amount) {
        balance = balance + amount;
    }//deposit

    //requires amount <= 0
    public void withdrawal(int amount) {
        if (balance >=  amount)
            balance = balance - amount;
    }//withdrawal
}//Account
```

Det nya programsystemet skall vara fleratrådat varför klassen Account måste göras trådsäker. Dessutom skall metoden withdrawal i det nya programsystemet fungera på så sätt att metoden returnera först när det finns ett tillräckligt stort belopp på kontot för att genomföra uttaget (d.v.s.tråden som gör uttaget måste eventuellt vänta tills andra trådar gjort insättningar på kontot). Då Pelle har begränsade kunskaper om trådar och trådsäkerhet blir det din uppgift att skriva om klassen Account (enligt vad som angivits ovan).

(5 poäng)

Tentamen 170816 - LÖSNINGSFÖRSLAG

Uppgift 1.

a) Kompileringsfel! En abstrakt klass kan inte instaisieras.

b) Kompileringsfel! Metoden act3(Penguin p) kan inte anropas med en aktuell parameter av typen Bird.

c) Kompileringsfel! Metoden shout() är protected och anropas utanför paketet.

d) Kompileringsfel! Metoden action(**int** i) finns inte för den statiska typen Bird.

e) Kompileringsfel! Metoden act5 har ingen parameter.

Eftersom man från frågeställningen kunde få uppfattningen om att eventuella fel låg i klassen Program har jag även godkänt att anropet av act5 var felaktigt. Utskriften blir då:

```
Some times I dream I can fly like this:
Wooooosh, I am
Penguin #2 reporting for duty!
Shouting is uncivilzed...
You should
Fear me 1 times
```

f) Penguin #2 reporting for duty!

g) Kompileringsfel! Metoden addToA() är **protected** och anropas utanför paketet.

Uppgift 2.

a) Koden strider mot Open/Closed Principel (och DependencyInversion Principle).

b)

```
public abstract class Service {
    private float price;
    public Service(float price) {
        this.price = price;
    }//constructor
    public float getPrice() {
        return price;
    }//getPrice
    public abstract float getVAT();
}//Service
```

```
public class BikeRepair extends Service {
   public BikeRepair(float price) {
        super(price);
    }//constructor
    @Override
    public float getVAT() {
        return getPrice() * 0.12f;
    }//getPrice
    // Other code omitted
}//BikeRepair
```

```
public class GymCard extends Service {
   public GymCard(float price) {
        super(price);
    }//constructor
    @Override
    public float getVAT() {
        return getPrice() * 0.06f;
    }//getPrice
    // Other code omitted
}//GymCard
```

```
public class BeautyCare {
    public BeautyCare(float price) {
        super(price);
    }//constructor
    public float getVAT() {
        return getPrice() * 0.25f;
    }//getPrice
    // Other code omitted
}//BeautyCare
```

```
import java.util.ArrayList;
public class SmallBuiseness {
    private ArrayList<Service> receipts = new ArrayList<>();

    public void addReceipt(Service receipt) {
        receipts.add(receipt);
    }//addReceipt

   public float computeVAT() {
      float totalVAT = 0;
      for (Service s : receipts)
          totalVAT += s.getVAT();
      return totalVAT;
   }//computeVAT
}//SmallBuiseness
```

Uppgift 3.

a) Kompileringsfel! Listan peList kan vara av typen List<PortableDevice>, List<Laptop> eller List<SomeType> där SomeType är en subtyp till typen PortableDevice. Ett objekt av typen PortableDevice kan inte läggas in i en lista av typen List<Laptop> eller typen List<SomeType> eftersom typerna LapTop och SomeType inte är supertyper till typen PortableDevice.

b) Kompileringsfel! Listan peList kan vara av typen List<PortableDevice>, List<Laptop> eller List<SomeType> där SomeType är en subtyp till typen PortableDevice. Ett objekt av typen Laptop kan inte läggas in i en lista av typen List<SomeType> eftersom typen Laptop inte är en subtyp till typen SomeType.

c) Korrekt! Listan peList kan vara av typen List<PortableDevice>, List<Laptop> eller List<SomeType> där SomeType är en subtyp till typen PortableDevice, och således är typen PortableDevice supertyp till typerna Laptop och SomeType.

d) Korrekt! Listan lsList kan vara av typen List<Laptop>, List<Computer>, List<PortableDevice> eller List<Object>. Ett objekt av typen Laptop kan läggas in i samtliga dessa typer av listor eftersom typen Laptop är en subtyp till typerna Computer, PortableDevice och Object.

e) Kompileringsfel! Listan lsList kan vara av typen List<Laptop>, List<Computer>, List<PortableDevice> eller List<Object>. Ett objekt av typen PortableDevice är inte supertyp till typerna Computer och Object.

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

g) Korrekt!

h) Korrekt!

i) Kompileringsfel!

j) Korrekt! List<?> är supertyp alla listor List<T>.

Uppgift 4.

a)

i) Ingen. S2 har ett förvillkor. S3 och S4 kastar exceptions om listan är tom (inte 0.0 som S1 specificerar).

ii) S1,S3 och S4.

iii) Ingen

iv) S3

Observera att en specifikation S är starkare än en specifikation W om varje implementation som uppfyller S också uppfyller W.

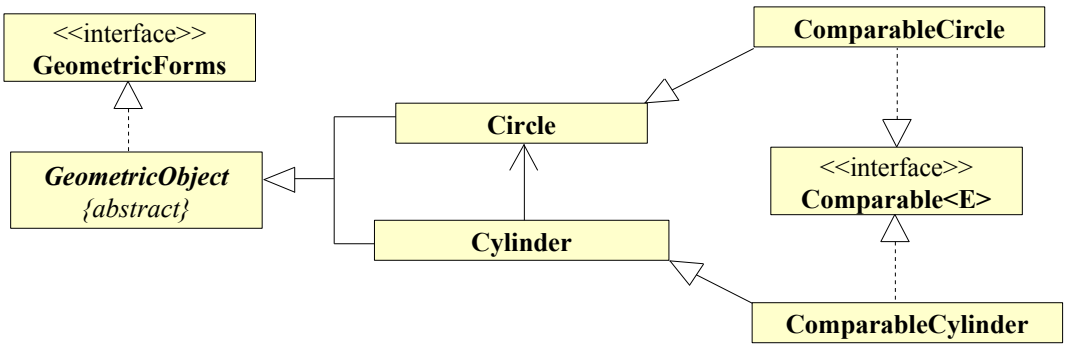
b) *Liskov Substitution Principle* säger att det är acceptabelt att göra klassen S till en subklass av klassen T, om och endast om det för varje publik metod som finns både i T och S gäller att S's metod som indata godtar alla värden som T's metod godtar samt att S gör alla bearbetningar på denna indata som T gör. Detta innbeär att samtliga metoder som överskuggas i en subklass måste ha lika stark eller starkare specifikation än motsvarande metod i superklassen.

Uppgift 5.

a) *Liskov Substitution Principle* säger att en supertyp är utbytbar med sina subtyper. Med den design som gjorts innebär detta alltså att var som helst där vi förväntar oss en cirkel går det lika bra med en cylinder. Eller med andra ord gäller det att *en cylinder är en cirkel*, vilket naturligtvis är helt galet. Den som gjort designen verkar ha infört implementationsarvet mellan klasserna Cylinder och Circle enbart med återanvändning av kod som motiv, vilket inte är ett tillräckligt motiv.

Klasserna ComparableCircle och ComparableCylinder implementerar det råa interfacet Comparable. Klassen borde implementera det generiska gränssnittet Comparable<E> iställtet för att erhålla typsäkerhet.

b) Delegering är ett alternativ implementationsarv för att återanvända kod.



c) Synligheten i GeometricObject är **protected**. Synligheten bör dock vara **private**. Man skall alltid eftersträva att exponera så lite som möjligt av superklassens interna representation till dess subklasser. Orsaken är givetvis att om den interna representationen är exponerad och används av subklasserna, måste subklasserna förändras ifall superklassen byter representation. I vårt specifika fall tillhandahåller dessutom klassen GeometricObject publika access-metoder för samtliga instansvariabler, varför inga ytterligare förändringar av klassen behöver göras.

d) Det saknas konstruktorer för att ange var det skapade objektet skall placeras i det två-dimensionella rummet. Är det möjligt skall en instans ges sitt fullständiga tillstånd när instansen skapas.

Uppsättningen konstruktorer är inkonsekvent mellan de olika klasserna. Exempelvis borde ComparableCylinder ha samma uppsättning konstruktorer som Cylinder.

Ur konsekvenssynpunkt kan även ordningen på parametrarna till konstruktorerna ifrågasättas. Exempelvis har klassen Cylinder konstruktorerna Cylinder(radius, length) och Cylinder(radius, color, length), det hade varit lämpligare att den sistnämnda konstruktorn ersatts med Cylinder(radius, length, color).

e) Det hade varit lämpligare att avbilda en färg med exempelvis klassen java.awt.Color (eller en egendefinierd klass) än en String. Likaledes hade det varit lämpligare att avbilda en position med exempelvis klassen java.awt.Point (eller en egendefinierad klass) än som två **int** x och y.

Uppgift 6.

```
public class MetricToSensorToImperialSensor implements ImperialSensor {
  public final static double MILES_PER_METER = 0.000621371192;
  public final static double HOURS_PER_SEC = 0.000277777778;
  private final MetricSensor ms;

  public MetricToSensorToImperialSensor() {
     ms = new MetricSensor();
  }//constructor

  // returns: distance traveled in miles
  public double getDistance() {
     return ms.readDistance() * MILES_PER_METER;
  }//getDistance

  // returns: speed in miles per hour
  public double getSpeed() {
     return ms.readSpeed() * MILES_PER_METER * HOURS_PER_SEC;
  }//getSpeed
}//MetricToSensorToImperialSensor
```

Uppgift 7.

a) Objekten som tillhör en icke-muterbar klass är oförändliga, dvs de behåller under hela sin livstid det tillstånd som de fick när de skapades. Icke-muterbara objekt har många fördelar:

- De är lätta att förstå och resonera om.
- De är alltid trådsäkra.
- Det är ofarligt att lämna ut referenser till dem. Det gör inget om en massa olika objekt råkar använda ett och samma immutable object.
- Deras interna data kan återanvändas. Om ett nytt immutable objekt skall skapas där vissa fält inte skiljer sig från de i ett befintligt objekt av samma klass kan alla oförändrade fält referera till exakt samma instanser som i det befintliga objektet.
- Det är lätt att använda immutable objects som tillstånd i andra objekt.
- Det är ofarligt att skicka dem till andra processer i distribuerade system.

b) Klassen exponerar sin interna representation, eftersom objekt av klassen Calendar är muterbara. I konstruktorn föreligger en risk. , exponeras den inre representationen till klient-objektet som har access till det initiala date-objekt. Det föreligger risk även i metoderna getDate(), eftersom klienten som anropar denna metod erhåller en referens till date-objektet.

I konstruktorn är det enklaste sättet att åtgärda exponeringsriskerna genom att skapa och lagra en kopia av objektet som fås som parametrar. I metoderna getDate() är det enklaste sättet att returnera en kopia av instansvariabeln date.

```
import java.util.Calendar;
public class Transaction {
    private final int amount;
    private final Calendar date;

    public Transaction(int amount, Calendar date) {
        this.amount = amount;
        this.date = (Calendar) date.clone();
    }//constructor

    public int getAmount () {
        return this.amount ;
    }//getAmount

    public Calendar getDate() {
        return (Calendar) this.date.clone();
    }//getDate
}//Transaction
```

Uppgift 8.

a)

```
public boolean addMember(String member) {
   return authorizedMembers.add(member);
}//addMember
```

b)

```
public void removeMember(String member) {
   if (!authorizedMembers.contains(member)) {
       throw new IllegalArgumentException();
   }
   authorizedMembers.remove(member);
   for (Set<String> s : listMap.values()) {
       s.remove(member);
   }
}//removeMember
```

c)

```
public boolean add(String listName, Set<String> subscribers) {
   if (!authorizedMembers.containsAll(subscribers)) {
       return false;
   } else {
       if (listMap.containsKey(listName)) {
           listMap.get(listName).addAll(subscribers);
       } else {
           listMap.put(listName, new HashSet<String>(subscribers));
       }
       return true;
   }
}//add
```

d)

```
public Map<String, Set<String>> subscriberMap() {
   Map<String, Set<String>> result = new HashMap<String, Set<String>>();
   for (Map.Entry<String, Set<String>> e : listMap.entrySet()) {
       String list = e.getKey();
       for (String s : e.getValue()) {
           if (!result.containsKey(s)) {
               result.put(s, new HashSet<String>());
           }
           result.get(s).add(list);
       }
   }
   return result;
}//subscriberMap
```

Uppgift 9.

```
public class Account {
   private int balance;
   public Account(int balance) {
       this.balance = balance;
   }//constructor

   //requires amount >= 0
   public synchronized void deposit(int amount) {
       balance += amount;
       notify();
   }//desposit

   //requires amount <= 0
   public synchronized void withdrawal(int amount) {
       while (balance < amount) {
           try {
               wait();
           } catch (InterruptedException e) {
               return;
           }
       }
       balance = balance - amount;
       notify();
   }//withdrawal
}//Account
```