Tentamen för TDA540 Objektorienterad programmering

DAG: 14-08-28 TID: 14:00 – 18:00

Ansvarig: Joachim von Hacht och Christer Carlsson

Förfrågningar: Joachim von Hacht, 0707/311066
Christer Carlsson, ankn 1038

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: Tentamen kan granskas på studieexpeditionen. Vi eventuella åsikter om rättningen eposta och ange noggrant vad du anser är fel så återkommer vi.

Hjälpmedel: *Cay Horstmann: Java for everyone* eller
Jan Skansholm: Java direkt med Swing.
Understrykningar och smärre förtydligande noteringar får finnas.

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 väl strukturerade, lätta att överskåda samt enkla att förstå. ***Indentera programkoden och skriv inte längre programrader än maximalt 80 tecken!!***

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

LYCKA TILL!!!!

Uppgift 1.

I mappen Uppgift1 finns filerna MyMax.java, Square.java och Uppgift1c.java.

Det är dessa filer du skall uppdatera för respektive deluppgift.

a) Nedanstående program innehåller ett antal fel (kompileringsfel och/eller logiska fel). Rätta felen!

```
public class MyMax {
    public static void main(String args) {
        System.out.println("Maximum is: ", maximum(5, 7));
        System.out.println("Maximum is: ", maximum(7, 5));
        System.out.println("Maximum is: ", maximum(3, 4, 9));
        System.out.println("Maximum is: ", maximum(3, 9, 4));
        System.out.println("Maximum is: ", maximum(9, 4, 3));
    }//main
    // Returns the maximun value of the parameters x, y and z
    public int maximum (int x, int y, int z) {
        int max = x;
        if (y >  max);
            max = y;
        if (z > max)
            max = z;
        return max;
    }//maximum
    // Returns the maximun value of the parameters x and y
    public static int maximum (int x, y) {
        if (x > y)
            return x;
    }//maximum
}//MyMax
```

Ett korrekt fungerande program skall ge utskriften:

```
Maximum is: 7
Maximum is: 7
Maximum is: 9
Maximum is: 9
Maximum is: 9
```

(5 poäng)

b) Betrakta nedanstående klass

```
public class Square {
    private int sideLength;
    private int area;

    public Square(int initiallength) {
        sideLength = initiallength;
        area = sideLength * sideLength;
    }//constructor

    public int getArea() {
        return area;
    }//getArea

    public void grow() {
        sideLength = 2 * sideLength;
    }//grow
}//Square
```

Klassen innehåller ett logiskt fel! Lokalisera och korrigera felet. (2 poäng)

c) Betrakta nedanstående program

```
import java.util.Scanner;
public class Uppgift1c {
    public static void main(String[] args) {
        if (proceed())
            System.out.println("I'm glad you will continue!");
        else
            System.out.println("Good-bye.");
    }//main

    public static boolean proceed() {
        Scanner scannerObject = new Scanner(System.in);
        while (true) {
            System.out.println("Answer yes to continue, and no to stop.");
            String answer = scannerObject.next();
            if (answer == "yes" || answer == "y")
                return true;
            if (answer == "no" || answer == "n")
                return false;
            else
                System.out.println("Your answer should be yes or no!");
        }
    }//proceed
}//Uppgift1c
```

Syftet med metoden proceed är att upprepa en fråga tills användaren besvarar frågan med ”yes” eller ”y” alternativt ”no” eller ”n”. Besvaras frågan ned ”yes” eller ”y” returnerar metoden värdet **true**, och om frågan besvaras med ”no” eller ”n” returnerar metoden **false**. Men metoden fungerar inte som avsett! Förklara vad som är fel och ge en korrekt lösning.

(2 poäng)

Uppgift 2.

Ytan Y av en triangel med sidorna a, b och c kan beräknas med hjälp av Herons formel

$$Y = \sqrt{p(p-a)(p-b)(p-c)} \quad \text{, där} \quad p = \frac{1}{2}(a+b+c)$$

Skriv ett program som läser in längden av sidorna i triangeln samt beräknar och skriver ut ytan av triangeln.

Du får själv välja om du vill göra in- och utmatning via dialogrutor eller använda System.in respektive System.out (se exemplen nedan)..

För att få full poäng på uppgiften:

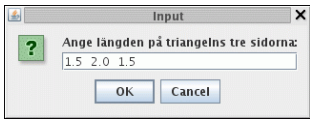
- skall programmet utformas på så sätt att inläsningen upprepas tills användaren avbryter exekveringen (vid användning av dialogrutor genom att användaren trycker på Cancel-knappen och vid användning av System.in genom att användaren lämpligen ger ctrl z)
- skall programmet innehålla en metod

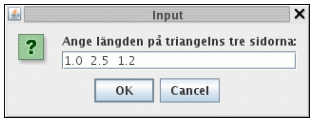
  public static double heron(**double** a, **double** b, **double** c)

  som beräknar och returnerar ytan av en triangel enligt Herons formel. Längden av sidorna på triangeln, vars yta skall beräknas, ges som argument till metoden.
- skall längden av de tre sidorna läsas med en inläsningssats (dvs ett Scanner-objekt skall användas).
- skall programmet kontrollera att de inlästa längderna verkligen kan utgöra sidor i en triangel. Om felaktiga indatavärden ges skall utskriften DETTA ÄR INGEN TRIANGEL! skrivas ut.
- skall utskriften av triangelns ytan anges med *exakt* 3 decimaler.

Tips: För att sidorna skall kunna bilda en triangel måste gälla att ingen sida får vara längre eller lika lång med den sammanlagda längden av de två övriga sidorna.

Med användning av dialogrutor





Med användning av System.in resp System.out

```
Ange längden på triangelns tre sidor: 1.5 2.0 1.5
Ytan av triangeln är 1.118

Ange längden på triangelns tre sidor: 1.0 2.5 1.2
DETTA ÄR INGEN TRIANGEL!

Ange längden på triangelns tre sidor:
```

(10 poäng)

I mappen Uppgift2 finns filen Heron.java som utgör skelettet till den klass i vilket du skall skriva din lösning

Uppgift 3.

Skriv en metod

public static int[] append(**int**[] vektA, **int**[] vektB)

som som tar två heltalsfält vektA och vektB och returnera ett nytt heltalsfält, där det nya fältet bildas genom att slå samman fälten vektA och vektB. I det nya fältet skall elementen i fältet vektA komma före elementen i fältet vektB och den inbördes ordningen av elementen i dessa båda fält skall bibehållas i det nya fältet. Om båda parametrarna till metoden är **null** skall en IllegalArgumentException kastas, metoden skall dock kunna hantera att en av argumenten är **null**. Metoden skall skrivas i klassen Uppgift3.

Nedanstående testprogram skall resultera i utskriften:

```
Illegal call! Both parameters are null!
[7, 3, 9, 1, 5]
[7, 3, 9, 1, 5]
[7, 3, 9, 1, 5, 12, 4, 14, 8, 10, 2, 6]
```

```
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        int[] vekt1 = null;
        int[] vekt2 = null;
        try {
            System.out.println(Arrays.toString(Uppgift3.append(vekt1, vekt2)));
        } catch (IllegalArgumentException e) {
            System.out.println("Illegal call! Both parameters are null!");
        }
        int[] vekt3 = {7, 3, 9, 1, 5};
        System.out.println(Arrays.toString(Uppgift3.append(vekt1, vekt3)));
        System.out.println(Arrays.toString(Uppgift3.append(vekt3, vekt1)));
        int[] vekt4 = {12, 4, 14, 8, 10, 2, 6};
        System.out.println(Arrays.toString(Uppgift3.append(vekt3, vekt4)));
    }//main
}//Main
```

(6 poäng)

I mappen Uppgift3 finns filen Uppgift3.java som innehåller ett skelett för metoden append. Det är i filen Uppgift3.java som du skall skriva din lösning.

I mappen Uppgift3 finns också filen Main.java som innehåller ovanstående main-metod för att testa metoden append.

Uppgift 4.

En digital färgbild lagras, som bekant, på RGB-format. Varje bildpunkt (pixel) utgörs av *tre* heltalsvärden i intervallet [0,255], där de enskilda värdena representerar intensiteten av färgerna rött, grönt respektive blått. En färgbild kan således avbildas med ett tredimensionellt fält av typen **int**[][][], är den första dimensionen definierar bildens höjd, den andra dimensionen bildens bredd och den tredje dimensionen de tre färgerna.

En teknik som får nya färgbilder bilder att se ut som gamla åldrade gråskalebilder är att göra *sepia filtrering*. Sepia filtrering beskrivs lättast som en process i två steg.

- Det första steget är att skapa en gråskalebild från färgbilden. För att få bästa kvalitén på gråskalebilden beräknas bildpunkternas *luminans*. Luminansen L är den för ögat upplevda ljusheten hos en yta och definieras som

  $$L = 0.299\, r + 0.587 g + 0.114 b.$$

  där r är intensiteten av rött, g är intensiteten av grönt och b är intensiteten av blått. I den gråskalebild som beräknas från en färgbild sätts intensiteten av färgerna rött, grön och blått i varje pixel till värdet L i motsvarande pixel i färgbilden.

- Det andra steget är att, för varje bildpunkt, göra den gröna och den blå färgkomponenten mörkare genom att multiplicera dessa färgkomponenter med 0.6 respektive 0.4.

Slutresultatet blir att bilden i stället för en grå färgton får en rödbrun färgton (som kallas sepia brun).

Din uppgift är att skriva en metod

public static int[][][] sepiaFilter(**int**[][][] sample)

som tar en färgbild sample och returnerar en ny bild som är en sepia filtrerad kopia av sample (enligt beskrivningen ovan).


Färgbild


Gråskalebild


Sepiabild

(7 poäng)

I mappen Uppgift4 finns filerna image.jpeg, ColorImage.class, ColorImagePanel.class, ColorImageWindow.class, Sepia.class, Main.java och Uppgift4.java. I filen Uppgift4.java finns ett skelett för metoden sepiaFilter och det är i denna fil du skall ge din lösning. Detta är den enda av ovanstående filer som du skall editera. När du har en kompilerad version av din lösning kan du testköra lösningen genom att kompilera och exekvera Main. Om exekveringen av Main visar ett fönster som innehåller originalbilden och den nya bilden enligt figurerna ovan fungerar din metod.

Uppgift 5.

I tärningsspelet Mexico deltar minst två spelare och spelet är ett elimineringsspel. När spelet börjar har alla spelare lika mycket pengar att spela med (= startavgift, dvs de pengar som spelaren riskerar att förlora).

En omgång går till på så sätt att varje spelare kastar två 6-sidiga tärningar. Spelaren som får lägst *poängvärde* förlorar omgången och måste lägga en i förväg fastställd insatsen i potten. Detta upprepas tills endast en av spelarna har pengar kvar. Denne är vinnaren och får hela potten.

För att beräkna *poängvärdet*, beräknas först den sammanlagda *tärningssumman*. Detta görs genom att först multiplicera värdet på tärningen med högst poäng med 10 och sedan addera värdet på tärningen med lägst poäng. Har tärningarna poängen 3 och 4 erhålls tärningssumman 43, har tärningarna poängen 5 och 5 erhålls tärningssumman 55, osv.

Tärningssumman ligger till grund för att beräkna poängvärdet. Tärningssummornas poängvärde från högsta till lägsta är enligt:

21, 66, 55, 44, 33, 22, 11, 65, .., 61, 54, .., 51, 43, 42, 41, 32, 31

Högsta poängvärdet är alltså tärningsvärdet 21 (som kallas ”Mexico”), följt av ”paren” (i numerisk ordning), följt av övriga tärningsvärden (i numerisk ordning).

I denna uppgift skall du skriva kompletterande kod till en kommandobaserad datorversion av tärningsspelet Mexico. Koden du skall komplettera finns i Bilaga A. Klasserna MexicoOptions och Player är helt klara. Du skall få spelat att fungera som exempelutskriften i Bilaga B (eller bättre).

a) Gör klart metoden roll() i klassen MexicoDice. Metoden skall returnera tärningssumman enligt beräkningen ovan.

(4 poäng)

b) Gör klart konstruktor i klassen Mexico. Konstruktor skapar, givet spelarnamnen, alla spelare och ger dem tillgång till en tärning. Alla spelare läggs i listan players.

(4 poäng)

c) Gör klart metoden finished() i klassen Mexico. Metoden skall returnera **true** om endast en spelare har pengar kvar, annars **false**.

(4 poäng)

d) Gör klart metoden potToWinner() i klassen Mexico. Metoden tilldelar vinnaren pengarna i potten.

(4 poäng)

e) Gör klart metoden lowestGrade() i klassen Mexico. Metoden returnerar spelaren med lägst poängvärde beräknat enligt ovan (förloraren). Om flera spelare har lägst poängvärde, så skall metoden returnera någon av dessa.

TIPS: Uppgiften innebär alltså att från spelarnas tärningssummor skapa en ordningsrelation som överensstämmer med tärningssummornas poängvärde. Ett sätt att skapa värdena i denna ordningsrelation är enligt följande:

- tärningssumman 21 får ett maxvärde (t.ex 1000)
- tärningssummor som är ett ”par” får ett värde som är tärningssumman multiplicerad med 10
- övriga tärningssummor får ett värde som motsvarar tärningssumman.

Ordningsrelationen som då erhålls blir

1000, 660, 550, 440, 330, 220, 110, 65, .., 61, 54, .., 51, 43, 42, 41, 32, 31

För att undvika att metoden returnerar en utslagen spelare (= en spelare som förlorat alla sina pengar och inte längre deltar i spelet) ges alla utslagna spelarna maxvärdet som poängvärde.

(8 poäng)

f) Gör klart de markerade delarna i main-metoden i klassen CommandLineMexico.

(4 poäng)

I mappen Uppgift5 finns filerna MexicoOptions.java, Player.java, MexicoDice, Mexico.java och CommandLineMexico.java. Det är i dessa filer du skall skriva dina lösningar.

Uppgifter (de ställen du skall göra klart är märkta med TODO i koden)

```
import java.util.Random;
public class MexicoDice {
    public  MexicoDice() {
    }//constructor

    public int roll() {
        // TODO  deluppgift a)
    }//roll
}//MexicoDice

import java.util.ArrayList;
import java.util.List;
public class Mexico {
    private final int MEXICO = 21;
    private final List<Player> players = new ArrayList<>();
    private int pot = 0;

    public Mexico(List<String> playerNames) {
        // TODO  deluppgift b)
    }//constructor

    public Player lowestGrade() {
        // TODO  deluppgift e)
    }//lowestGrade

    public List<Player> getPlayers() {
        return players;
    }//getPlayer

    public boolean finished() {
            // TODO  deluppgift c)
    }//finished

    public void addToPot(int i) {
        pot += i;
    }//addToPot

    public void potToWinner() {
        // TODO  deluppgift d)
    }//potToWinner
}//Mexico
```

Uppgifter (de ställen du skall göra klart är märkta med TODO i koden)

```
import java.util.Scanner;
public class CommandLineMexico {
    public static void main(String[] args) {
        boolean done = false;
        Mexico mexico = new Mexico(MexicoOptions.getPlayersNames());
        System.out.println("Mexico started. Players are: ");
        dump(mexico);
        Scanner s = new Scanner(System.in);
        while (!done) {
            for (Player p : mexico.getPlayers()) {
                if (p.hasMoney()) {
                    System.out.println("Player is " + p.getName() + ". Press enter to roll!");
                    s.nextLine();
                    p.roll();
                    System.out.println(p.getName() + " got " + p.getScore());
                } else {
                    p.clearScore();
                }
            }
            // TODO Do grading
            // And move money
            if (mexico.finished()) {
                mexico.potToWinner();
                done = true;
            }
            dump(mexico);
        }
        System.out.println("Round finished");
    }//main

    private static void dump(Mexico mexico) {
        System.out.println("-----------------------");
        for (Player p : mexico.getPlayers()) {
            System.out.print(p + " ");
        }
        System.out.println();
        System.out.println("-----------------------");
    }//dump
}//CommandLineMexico
```

```
// This class is finished, don't touch
import java.util.Arrays;
import java.util.List;
public class MexicoOptions {
    public static final int STARTING_FEE = 2;       //spelarnas startkapital
    public static final int PAYOFF = 1;             //kostnaden för en spelare som förlora en omgång
    private static final List<String> names = Arrays.asList("Anna", "Urban", "Fia");
    public static List<String> getPlayersNames() {
        return names;
    }//getPlayer
}//MexicoOptions

// This class is finished, don't touch
public class Player {
    private final String name;
    private final MexicoDice dice;
    private int score;
    private int balance;
    public Player(String name, int balance, MexicoDice dice) {
        this.name = name;
        this.balance = balance;
        this.dice = dice;
    }//constructor
    public String getName() {
        return name;
    }//getName
    public void roll() {
        score = dice.roll();
    }//roll
    public void withdraw(int i) {
        balance -= i;
    }//withdraw
    public void add(int i) {
        balance += i;
    }//add
    public boolean hasMoney() {
        return balance > 0;
    }//hasMoney
    public int getScore() {
        return score;
    }//getScore
    public void clearScore() {
        score = 0;
    }//clearScore
    @Override
    public String toString() {
        return "{" + name + " : " + score + ":" + balance + "}";
    }//toString
}//Player
```

Bilaga B

```
Mexico started. Players are:
{Anna : 0:2} {Urban : 0:2} {Fia : 0:2}
Player is Anna. Press enter to roll!

Anna got 22              <--- Par är alltid högre än icke-par
Player is Urban. Press enter to roll!

Urban got 21                    <--- Högsta möjliga poäng
Player is Fia. Press enter to roll!

Fia got 41
Fia must put in pot
----------------------
{Anna : 22:2} {Urban : 21:2} {Fia : 41:1}
----------------------
Player is Anna. Press enter to roll!

Anna got 53
Player is Urban. Press enter to roll!

Urban got 33             <--- Par är alltid högre än icke-par
Player is Fia. Press enter to roll!

Fia got 52
Fia must put in pot
----------------------
{Anna : 53:2} {Urban : 33:2} {Fia : 52:0}  <!-- Fia har förlorat alla sina pengar och får inte delta i spelet längre
----------------------
Player is Anna. Press enter to roll!

Anna got 63
Player is Urban. Press enter to roll!

Urban got 52
Urban must put in pot
----------------------
{Anna : 63:2} {Urban : 52:1} {Fia : 0:0}
----------------------
Player is Anna. Press enter to roll!

Anna got 52
Player is Urban. Press enter to roll!

Urban got 62
Anna must put in pot
----------------------
{Anna : 52:1} {Urban : 62:1} {Fia : 0:0}
----------------------
Player is Anna. Press enter to roll!

Anna got 11                     <--- Par är alltid högre än icke-par
Player is Urban. Press enter to roll!

Urban got 42
Urban must put in pot
----------------------
{Anna : 11:6} {Urban : 42:0} {Fia : 0:0} <!-- Anna fick hela potten
----------------------
Round finished
```