Tentamen för TDA540 Objektorienterad programmering

DAG: 14-04-23 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 Math.java, Uppgift1b.java och Main.java. Det är dessa filer du skall uppdatera för respektive deluppgift.

a) Betrakta nedanstående program, vars syfte är att läsa in ett tal och skriva ut talets kvadratrot:

```
import javax.swing.*;
public class Math {
    public static void main (String[] arg) {
        String indata = JOptionPane.showInputDialog("Ange ett tal: ");
        double tal = Double.parseDouble(indata);
        double root = Math.sqrt(tal);
        JOptionPane.showMessageDialog(null, "Roten ur talet " + tal + " är " + root);
    }//main
}//Math
```

När programmet kompileras fås följande felmeddelande

```
Math.java:6: error:cannot resolve symbol
     double root = Math.sqrt(tal);
                           ^
symbol  : method sqrt (double)
location: class Math
```

Förklara vad som är felaktigt i programmet och rätta till i koden. (2 poäng)

b) Betrakta nedanstående metod

```
public static boolean  sorted(int[] vekt){
    boolean  okey = true ;
    for  (int  index = 0; index < vekt.length; index = index + 1)
        if (vekt[index] > vekt[index+1])
            okey = false ;
    return  okey;
}//sorted
```

Avsikten med metod är att den skall returnera värdet **true** om fältet vekt är sorterat i stigande ordning (dvs att vekt[0] ≤ vekt[1] ≤ vekt[2] ≤ . .) och värdet **false** om så inte är fallet. Metoden är dock inte korrekt utan ger upphov till ett exekveringsfel! Förklara vad som är fel och korrigera felet. (2 poäng)

c) Betrakta nedanstående program, vars syfte är att skriva ut 100 slumptal i intervallet [1, 10].

```
import java.util.Random;
public class Main {
    public static void main (String[]  arg) {
        for (int i = 1; i <= 100; i = i + 1) {
            int val = Random.nextInt(10) + 1;
            System.out.println("Random number between 1 and 10 is " + val);
        }
    }//main
}//Main
```

När man kompilerar programmet får man felutskriften

```
Main.java:5: error:  non-static method nextInt(int) cannot be referenced from a static context
                     int val = Random.nextInt(10) + 1;
                                      ^
```

Förklara vad felet beror på och rätta till i koden. (2 poäng)

Uppgift 2.

Det månatliga återbetalningsbeloppet för ett lån beräknas med formel:

$$belopp = \frac{l * c(1+c)^n}{(1+c)^n - 1}$$

där *l* är lånebeloppet, *c* är månadsräntan (en tolftedel av årsräntan) och *n* är antalet månader för att avbetala lånet.

Din uppgift är att skriva ett program som läser in lånebelopp (i kronor), årsränta (i %) och återbetalningstid (i år), samt beräknar och skriva ut det månatliga återbetalningsloppet.

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

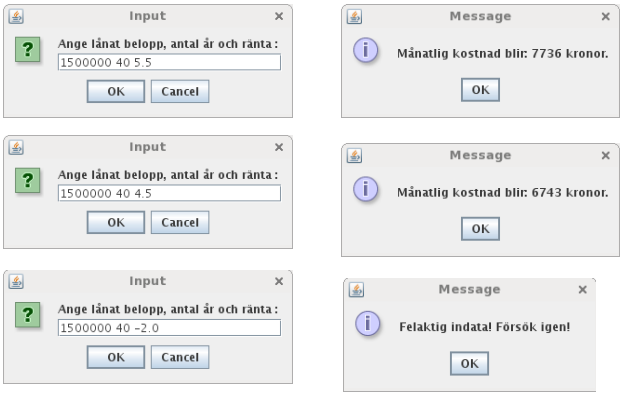
För att kunna erhålla full poäng på uppgiften:

- 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 lånebelopp, ränta och återbetalningstid läsas i en inläsningssats (dvs ett Scanner-objekt skall användas)
- skall en felutskrift ges om någon av indatan som läses in inte är positiv
- skall beloppet i utskriften anges som ett heltal
- skall programmet innehålla en metod

  public static double calculatePayment(**double** loan, **double** rate, **int** years)

  som beräknar det månatliga återbetalningsbeloppet.

Med användning av dialogrutor



Med användning av System.in resp System.out

```
Ange lånat belopp, antal år och ränta : 1500000 40 5.5
Månatlig kostnad blir: 7736 kronor

Ange lånat belopp, antal år och ränta : 1500000 40 4.5
Månatlig kostnad blir: 6743 kronor

Ange lånat belopp, antal år och ränta : 1500000 40 -2.0
Felaktig indata! Försök igen!

Ange lånat belopp, antal år och ränta :
```

(10 poäng)

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

Uppgift 3.

För att ett kreditkortsnummer skall vara giltigt måste det gälla att:

- det är 16 siffror långt
- varje siffra är mellan 0 och 9
- checksumman är giltig (se nedan).

Checksumman beräknas på följande sätt:

1) för varje siffra som finns på en jämn position, multiplicera siffran med 2 och beräkna siffersumman av det erhållna resultatet. Observera att första siffran har position 0!
2) addera *de ingående siffrorna* som erhålls i steg 1 samt siffrorna som finns i de udda positionerna i kontonumret.
3) om summan som erhålls i steg 2 är en multipel av 10 är checksumman giltig, annars ogiltig.

Exempel:

Anta att kreditkortsnummer är 1234567812345678. Beräkningen av checksumman tillgår på följande sätt:

1)

position	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
siffra	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
dubblera jämna	**2**	2	**6**	4	**10**	6	**14**	8	**2**	2	**6**	4	**10**	6	**14**	8

2) Summan blir:

$$2 + 2 + 6 + 4 + (1 + 0) + 6 + (1 + 4) + 8 + 2 + 2 + 6 + 4 + (1 + 0) + 6 + (1 + 4) + 8 = 68$$

3) Eftersom summan i steg 2 blir 68 och 68 inte är en multipel av 10 är 234567812345678 ett ogiltigt kreditkortsnummer.

Din uppgift är att skriva en metod

public static boolean verifyCreditCardChecksum(**int**[] digits)

som tar ett heltalsfält digits som representerar ett 16 siffror långt kreditkortsnummer, och avgör om det är ett giltigt kreditkortsnummer eller inte. Du får anta att heltalsfältet digits har 16 element och elementen har värden i intervallet [0, 9].

(5 poäng)

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

I mappen Uppgift3 finns också filen Main.java som innehåller en main-metod för att testa metoden verifyCreditCardChecksum. Två kreditkortsnummer testas, där det första numret är felaktigt och det andra numret är korrekt. Om metoden verifyCreditCardChecksum fungerar som avsett skall utskriften

```
false
true
```

erhållas.

Uppgift 4.

En digital bild kan representeras som ett tvådimensionellt fält av bildpunkter. I en digital färgbild utgörs varje bildpunkt av *tre* heltalsvärden i intervallet 0-255, där de enskilda värdena representerar intensiteten av färgerna rött, grönt och blått. En färgbild kan avbildas med ett tredimensionellt fält av typen **int**[][][], är den första dimensionen definierar bildens höjd, den andra dimensionen definierar bildens bredd och den tredje dimensionen representerar färgerna rött, grönt och blått.

Din uppgift är att skriva en metod

public static int[][][] fourCopies(**int**[][][] samples)

som tar en bild samples och returnerar en ny bild som består av fyra förminskade kopior av bilden samples. Bilden som returneras skall ha samma storlek som bilden samples (se figuren nedan), vilket innebär att var och en av de fyra förminskade kopiorna har har en höjd respektive en bredd som är hälften av höjden respektive bredden av bilden samples.

Eftersom originalbilden har 4 gånger så många bildpunkter som en förminskad kopia, beräknas värdet av en bildpunkt i en förminskad koipa som medelvärdet av fyra närliggande bildpunkter i originalbilden. Exempelvis beräknas bildpunkt (0, 0) i en förminskad kopia från bildpunkterna (0, 0), (0, 1), (1, 0) och (1, 1) i originalbilden. Allmänt gäller att bildpunkt (row, col) i kopian beräknas från bildpunkterna (2*row, 2*col), (2*row, 2*col + 1), (2*row+1, 2*col) och (2*row+1, 2*col+1) i originalbilden.

Original bild

Ny bild

(7 poäng)

I mappen Uppgift4 finns filerna mushroom.jpeg, ColorImage.class, ColorImagePanel.class, ColorImageWindow.class, FourCopies.class, Main.java och Uppgift4.java. I filen Uppgift4.java finns ett skelett för metoden fourCopies 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 orginalbilden och den nya bilden fungerar din metod.

Uppgift 5.

I denna uppgift skall du skriva kompletterande kod till en kommandobaserad datorversion av tärningsspelet LCR. (Kod du skall komplettera finns i bilaga A.)

I LCR deltar minst tre spelare och tre tärningar används. Tärningarna är en speciell typ av 6-sidiga tärningar, där tre av sidorna är märkta med "." och de övriga sidorna med "L", "C" respektive "R".

När spelet startas tilldelas varje spelare tre brickor. Underspelets gång kan brickor omfördelas bland spelarna och brickor kan plockas bort ur spelet. Spelet pågår tills endast en spelare har brickor kvar, denna spelare har då vunnit.

Själva spelandet tillgår på följande sätt:

Aktuell spelare kastar så många tärningar som brickor han/hon har, dock max tre tärningar. Beroende på utfallet på tärningarna sker följande:

"L"	spelaren ger en bricka till spelaren på sin vänstra sida
"R"	spelaren ger en bricka till spelaren på sin högra sida
"C"	spelaren lägger en bricka i en gemensam hög och brickan försvinner ur spelet
"."	spelaren gör inget

Om flera spelare har brickor kvar, är det spelaren till vänster om den aktuella spelaren som är nästa spelare att kasta tärningarna. Om endast en spelare har brickor kvar är spelet över och spelaren med brickor är vinnare.

Inga spelare slås ut under spelets gång. En spelare med 0 brickor kan få nya brickor under spelets gång. Om den aktuella spelaren har 0 brickor kastar han/hon 0 tärningar.

a) Gör klart metoden roll() i klassen LCRDice. Lämpligen används klassen java.util.Random för att slumpa fram vilken sida som kommer upp när tärningen kastas. (3 poäng)

b) Gör klart de private metoderna getPlayerLeft(), getPlayerRight() och getNRolls() i klassen LCRGame. Se kommentarer i koden hur de skall fungera. (8 poäng)

c) Gör klart den publika metoden gameOver() i klassen LCRGame. (3 poäng)

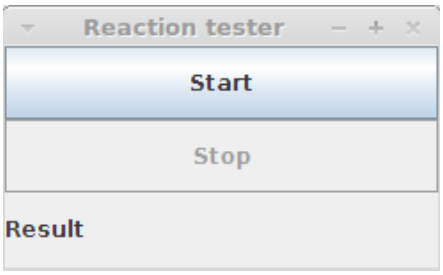
d) Gör med hjälp av de private metoderna klart metoden roll() i LCRGame. Observera att i vår implementation av LCR används endast en tärning och inte tre. Denna tärning kastas alltså flera gånger. (6 poäng)

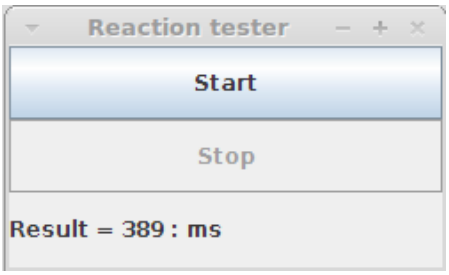
e) I klassen CommandLineLCR; Skapa en instans av LCRGame. Gör klart så att vi får ett fungerande spel (jämför med körningsexemplet i bilaga B). (6 poäng)

I mappen Uppgift5 finns filerna LCRDice.java, LCRGame.java, CommandLineCR.java och Player.java. Det är i dessa filer du skall skriva dina lösningar.

Uppgift 6.

I denna uppgift skall du skriva en liten händelsebaserad applikation som mäter din ”reaktionstid”. Applikationen innehåller två knappar och en etikett (enligt bilderna nedan).





När applikationen startas visas ett fönster enligt bilden till vänster ovan. Start-knappen är tryckbar medan Stop-knappen är otryckbar. När användaren trycker på Start-knappen görs Stop-knappen tryckbar.

Reaktionstiden definieras som den tid som det förflyter mellan användaren har tryckt på Start-knappen tills det användaren trycker på Stop-knappen. När användaren tryckt på Stop-knappen görs Stop-knappen otryckbar och reaktionstiden i millisekunder skrivs ut i etiketten (se bilden till höger ovan).

För att beräkna reaktionstiden skall klassmetoden currentTimeMillis() i klassen System användas. Metoden currentTimeMillis() returnerar ett heltal av typen **long** (som används på precis samma sätt som typen **int**). Värdet som returneras anger antalet millisekunder som förflutit sedan midnatt 1 januari 1970 fram tills anropet av metoden.

(6 poäng)

I mappen Uppgift6 finns filen MainFrame.java som innehåller ett skelett i vilket du skall skriva din lösning.

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

```
public class LCRDice {
     // Method should return L, C, R or . (dot) with  corresponding probabilities 1/6, 1/6, 1/6, 3/6
    public String roll() {
        // TODO  deluppgift a)
   }
}//LCRDice

import java.util.ArrayList;
import java.util.List;
public class LCRGame {
    private final LCRDice dice;  //it is simpler to use one dice and throw three times
    private final List<Player> players;
    private Player actual;
    private final List<String> result = new ArrayList<>();

    public LCRGame(List<Player> players, LCRDice dice) {
        this.players = players;
        this.dice = dice;
    }

    public void start() {
        actual = players.get(0);
    }

    public Player getActualPlayer() {
        return actual;
    }

    public List<String> getResult() {
        return result;
    }

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

    // Get player to the left of actual
    private Player getPlayerLeft() {
        // TODO  deluppgift b)
    }

    // Get player to the right of actual
    private Player getPlayerRight() {
        // TODO  deluppgift b)
    }

    // Decide how many rolls depending on players number of chips
    private int getNRolls() {
        // TODO  deluppgift b)
    }

    // Checks if there is only one player that still have chips
    public boolean gameOver() {
        // TODO  deluppgift c)
    }

    // Roll dice and distribute chips according to result. Set new actual player clockwise
    public void roll() {
        result.clear();
        // TODO deluppgift d)
    }
}//LCRGame
```

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

```
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
public class CommandLineLCR {

    public static void main(String[] args) {
        boolean done = false;
        //The list players decides the  playing order amonge the players.
        //Pelle is the first player.  Fia is to the left of Pelle and Sven is to the right of Pelle
        List<Player> players = Arrays.asList(new Player("Pelle", 3), new Player("Fia", 3), new Player("Sven", 3));
        LCRGame lcr = ...  /*TODO deluppgift e): Create a game here!!*/ ;
        lcr.start();
        System.out.println("LCR started");
        System.out.print("Players are ");
        dump(lcr);
        Scanner s = new Scanner(System.in);
        while (!done) {
            System.out.println("Player is " + lcr.getActualPlayer());
            System.out.print("> ");
            String cmd = s.nextLine();
            // TODO deluppgift e): use cmd to call methods on LCRGame or quite
        }
         // TODO deluppgift e): Print out winner or game aborted
    }

    private static void dump(LCRGame lcr) {
        for (String s : lcr.getResult()) {
            System.out.print(s + "  ");
        }
        System.out.println();
        for (Player p : lcr.getPlayers()) {
            System.out.print(p + " ");
        }
        System.out.println();
    }
}
```

```
// This class is finished, don't touch
public class Player {
    private final String name;
    private int nChips;
    public Player(String name, int nChips) {
        this.name = name;
        this.nChips = nChips;
    }
    public void addChip() {
        nChips++;
    }
    public void removeChip() {
        if (nChips > 0) {
            nChips--;
        }
    }
    public int getNChips() {
        return nChips;
    }

    public String getName() {
        return name;
    }
    @Override
    public String toString() {
        return "{" + name + "," + nChips + '}';
    }
}//Player
```

Bilaga B

```
LCR started
Players are
{Pelle,3} {Fia,3} {Sven,3}
Player is {Pelle,3}
> r
R . .
{Pelle,2} {Fia,3} {Sven,4}
Player is {Fia,3}
> r
L C R
{Pelle,3} {Fia,0} {Sven,5}
Player is {Sven,5}
> r
C C .
{Pelle,3} {Fia,0} {Sven,3}
Player is {Pelle,3}
> r
C . R
{Pelle,1} {Fia,0} {Sven,4}
Player is {Fia,0}
> r

{Pelle,1} {Fia,0} {Sven,4}
Player is {Sven,4}
> r
R C .
{Pelle,1} {Fia,1} {Sven,2}
Player is {Pelle,1}
> r
.
{Pelle,1} {Fia,1} {Sven,2}
Player is {Fia,1}
> r
L
{Pelle,1} {Fia,0} {Sven,3}
Player is {Sven,3}
> r
R C C
{Pelle,1} {Fia,1} {Sven,0}
Player is {Pelle,1}
> r
L
{Pelle,0} {Fia,2} {Sven,0}
Game over! Winner is {Fia,2}
```