# Tentamen för TDA540 Objektorienterad programmering

DAG: 13-12-20 TID: 8:30 – 12:30

---

**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.

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

På de programmeringsuppgifter som ingår i tentamenstesen kan vissa poäng erhållas även om ett fullständigt program inte redovisas, detta kräver dock att en algoritm som löser problemet presenteras.

# LYCKA TILL!!!!

**Uppgift 1.**

a) Följande klass innehåller två fel.

```
public class Trubbel{
    private int aNumber;
    public Trubel(int n){
        aNumber = n;
    }
    public int getNumber(){
        return n;
    }
}
```

Vid första kompileringsförsöket fås följande felmeddelande

```
Trubbel.java:3: invalid method declaration; return type required
public Trubel(int n){
              ^
1 error
```

Efter att ha rättat det första felet görs ett nytt kompileringsförsök, vilket resulterar i ett annat felmeddelande.

```
Trubbel.java:7: cannot find symbol
symbol : variable n
location: class Trubbel
return n;
       ^
1 error
```

Förklara orsaken till felen och rätta till i koden! (3 poäng)

b) Betrakta nedanstående metod:

```
public class Uppgift1b {
    public static void main(String[] args) {
        String[] animalism = {new String("four"), new String("legs"), new String("two"),
                              new String("legs"), new String("bad")};
        int count = countOccurrences(animalism, "legs");
        System.out.println("Number of occurrences: " + count);
    }
    public static int countOccurrences(String[] wordArray, String word) {
        int result = 0;
        for (int i = 0; i < wordArray.length; i = i + 1) {
            if (wordArray[i] == word) {
                result = result + 1;
            }
        }
        return result;
    }
}
```

Meningen är att metoden skall skall returnera hur många gånger strängen word förekommer i fältet wordArray. Metoden innehåller dock en fel. När vi exekverar main-metoden erhålls utskriften

Number of occurrences: 0

och inte den förväntade utskriften

Number of occurrences: 2

Förklara vad som är fel och rätta till felet. (3 poäng)

I mappen Uppgift1 finns filerna Trubbel.java och Uppgift1b.java. Det är dessa filer du skall uppdatera för respektive deluppgift.

**Uppgift 2.**

Skriv ett program som beräknar och skriver ut medelhastigheten utifrån avverkad sträcka och använd tid. Den avverkade sträckan skall anges i meter och förbrukad tid i sekunder. Medelhastigheten skall beräknas både i meter/sekund och kilometer/timma.

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

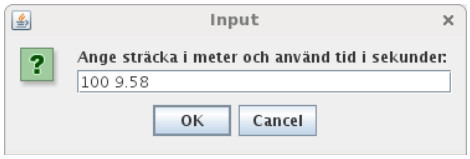
För att få full poäng på uppgiften:

- 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

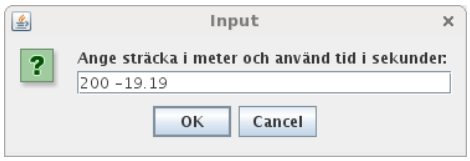
  **public static double** toKmPerHours(**double** mPerSec)

  som tar en hastighet angiven i meter/sekund och returnerar motsvarande hastighet i kilometer/timma
- skall sträcka och tid läsas med en inläsningssats (dvs ett Scanner-objekt skall användas)
- skall programmet kontrollera att de inlästa värdena är rimliga, dvs att sträcka och tid har positiva värden. Om felaktiga indatavärden ges skall utskriften FELAKTIG INDATA! skrivas ut

**Med användning av dialogrutor**









**Med användning av System.in resp System.out**

Ange sträcka i meter och använd tid i sekunder: 100 9.77
Medelhastigheten är 10.24 meter per sekund
vilket motsvarar 38.85 kilometer per timme

Ange sträcka i meter och använd tid i sekunder: 200 -19.19
FELAKTIG INDATA!

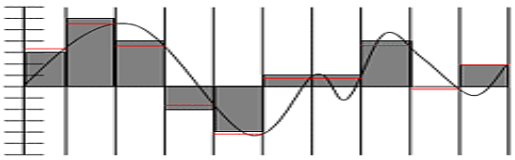
Ange sträcka i meter och använd tid i sekunder:

(10 poäng)

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

**Uppgift 3.**

Ett ljud kan digitaliseras genom att den analoga ljudsignalen samplas vid diskreta tidpunkter. En vanlig standard för digitaliserad musik är en samplingsfrekvens på 441 kHz (alltså 44100 avläsningar per sekund) .



Värdet 0 representerar tystnad. Skriv en metod

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

som tar ett heltalsfält samples som innehåller digitaliserat ljud och skapar ett nytt heltalsfält som är en kopia av samples, men där inledande tystnad (d.v.s. 0:or) är borttagen.

Exempel:

Körs nedanstående main-metod

```
public static void main(String[] args) {
    int[] sound = {0, 0 , 0, 0, -14, 120, 67, -234, -15, -389, 289, 178, -437, 33, 15,  -32, 230, 368};
    int[] result = trimSilenceFromFront(sound);
    System.out.println(Arrays.toString(result));
}
```

skall utskriften

[-14, 120, 67, -234, -15, -389, 289, 178, -437, 33, 15, -32, 230, 368]

erhållas.

(6 poäng)

I mappen Uppgift3 finns filen Uppgift3.java som innehåller den main-metod som anges i tentamenstesen, samt ett skelett för metoden trimSilenceFromFront. Det är i filen Uppgift3.java som du skall skriva din lösning.

**Uppgift 4.**

Steganografi handlar om att skriva dolda budskap på ett sådant sätt att ingen - bortsett från avsändaren och den avsedda mottagaren - misstänker att det finns ett meddelande. I denna uppgift skall ni gömma ett meddelande i en digital färgbild.

En digital färgbild lagras, som bekant, på RGB-format. Varje bildpunkt 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.

Hur ett meddelande göms i en digital färgbild beskrivs nedan:

I ASCII-koden (American Standard Code for Information Interchange) representeras varje tecken som ett heltal bestående av tre siffror. För alla bokstäver, siffror och andra tecken såsom ?, $ och @, kan man i Java erhållas ASCII-koden genom att typomvandla tecknet till ett heltal, t.ex. evalueras (**int**) 'B' till 66 (eller 066 som ett 3-siffrigt heltal), och (**int**) 'k' till 107.

Ett tecken kan gömmas i en bildpunkt genom att utnyttja att tecknet i ASCII-koden utgörs av ett 3-siffrigt heltal och att en bildpunkten utgörs av de tre komponenterna rött, grön och blått. Den första siffran i ASCII-koden döljas i den minst signifikanta siffran i komponenten som representerar rött, den andra siffran i ASCII-koden döljas i den minst signifikanta siffran i komponenten som representerar grönt och den tredje siffran i ASCII-koden döljas i den minst signifikanta siffran i komponenten som representerar grönt.

Exempel:

Tecknet 'B', som har ASCII-koden 107, göms i en bildpunkt vars RGB-värden är [159, 223, 163] enligt:

| **Ursprunglig bildpunkt** | | | | **Bildpunkten med 'k' gömd** | | |
|---|---|---|---|---|---|---|
| Red | Green | Blue | göm 'k', vars ACSII-kod är 107 | Red | Green | Blue |
| 159 | 223 | 163 | → | 15**1** | 22**0** | 16**7** |

Modifikationen i bildpunkten är så liten att den inte kan upptäckas med blotta ögat.

För att avkoda ett meddelande i en bild utför den omvända processen. Beräkna den minst signifikanta siffran i respektive färgkomponent, bilda ASCII-koden för tecknet från dessa tre siffror och typomvandla det erhållna heltalet till ett tecken (t.ex. evalueras (**char**) 107 till 'k'). Avkodningen av meddelandet *förändrar inte bilden*, utan meddelandet kommer att finnas i bilden för alltid.

Din uppgift är att skriva en metod

**public static int**[][][] hideMessage(**int**[][][] sample, String msg)

som gömmer meddelandet msg i bilden sample enligt ovan beskrivna tillvägagångssätt. Tecknen i meddelandet läggs in i bilden radvis, med start i första kolumnen. Du får anta att meddelandet ryms i bilden, dvs inte är längre än antalet bildpunkter i bilden.

För att den beskrivna metoden ovan skall fungera måste bilden lagras på ett bildformat där ingen information går förlorad vid komprimeringen. Detta gäller t.ex. för formatet png, men inte för jpg.

(7 poäng)

I mappen Uppgift4 finns filerna cat.png, ColorImage.class, Steganografi.class, Main.java och Uppgift4.java. I filen Uppgift4.java finns ett skelett för metoden hideMessage 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 ger utskriften

Detta är ett test!!

fungerar din metod.

**Uppgift 5.**

I denna uppgift skall du skriva en kommandoradsbaserad datorversion av tärningsspelet Gris.

I Gris deltar två eller flera spelare. Man använder sig av en 6-sidig tärning och spelet går ut på att först nå eller komma över totalpoängen 100 poäng. Spelarna kastar tärningen i turordning i ronder enligt följande:

- Om 1 kommer upp på tärningen erhåller spelaren 0 poäng i denna rond och spelet övergår till nästa spelare som står på tur.
- Om 2, 3, 4, 5 eller 6 kommer upp på tärningen erhålls lika många poäng som tärningen anger till rondens poäng. Spelaren kan sedan välja mellan att kasta tärningen igen, och därmed riskera de poäng som spelaren hittills erhållit i ronden (genom att få 1 i nästa kast), eller att passa.
- Om spelaren passar adderas rondens poäng till totalpoängen och nästa spelare som står på tur påbörjar sin rond.

a) Skapa en klass Dice för tärningen. Tärningen skall hålla reda på summan som en spelare erhåller under en rond (ackumulerar poängen för kasten om poängen är > 1). I klassen Dice skall följande finnas:

```
// Kontruktor. Parametern face anger antalet sidor på tärningen.
public Dice(int faces)

// Kastar tärningen och returnerar tärningens poäng. Ackumulerar poängen om poängen > 1
  //annars sätts ackumulerad poäng till 0.
public int roll()

// Returnerar den ackumulerade poängen.
public int getTotal()

// Returnerar poängen för senaste tärningskastet, om inget kast har gjorts skall IllegalStateException resas.
public int getLastResult()

// Nollställer ackumulerad poäng
public void clear()
```

(6 poäng)

b) Skapa en klass Pig som representerar spelet Gris. Klassen använder de givna klasserna PigOptions och Player (se Bilaga A), samt klassen Dice för att skapa tärningen. I klassen Pig skall följande finnas:

```
// Konstruktor som tar med en lista med de spelare som deltar.
public Pig(List<Player> players)

// Kastar tärningen och returnerar tärningens poäng
public int roll()

// Returnera aktuell spelare.
public Player getActualPlayer()

// Returnera en lista med alla spelare
public List<Player> getPlayers()

// Sätter aktuell spelare och startar spelet.
public void start()

// Byter aktuell spelare till den spelare som står näst i tur.
public void next()

//Adderar poängen som aktuell spelare erhållit under ronden till den aktuella spelarens totalpoäng.
public void setTotal()

// Returnerar true om den aktuella spelaren vunnit, annars returneras false.
public boolean gameOver()
```

(6 poäng)

c) Implementera en kommandoradsbaserad version av Gris. Detta skall göras genom att komplettera den bifogade klassen CommandLinePig. Låt hela spelet köras i main-metoden. Följande kommandon skall finnas

| | |
|---|---|
| r | roll - aktuell spelare kastar tärningen |
| p | pass – aktuell spelare pasasar och nästa spelare i tur blir aktuell spelare |
| q | quite – aktuell spelare avbryter spelet. Namn och uppnådd totalpoäng skrivs ut för samtliga spelare. |

I bilaga B finns ett körningsexempel som visar hur är tänkt att fungera. (8 poäng)

d) Vi vill kunna återanvända den grundläggande funktionaliteten för tärningen. Detta gör vi genom att lägga basfunktionaliteten i den abstrakta klassen AbstractDice, och den specifika Gris-funktionaliteten (att ackumulera summan av successiva kast) i en subklass PigDice. Implementera dessa båda klasser. Ändra sedan i klassen Pig så att PigDice används istället för Dice.

(6 poäng)

I mappen Uppgift5 finns filerna Player.java samt PigOptions.java för de givna klasserna Player och PigOptions. Vidare finns filerna Dice.java, Pig.java och CommandLinePig.java. Det är också i denna mapp som du skall spara dina filer AbstractDice.java samt PigDice.java.

**Uppgift 6.**

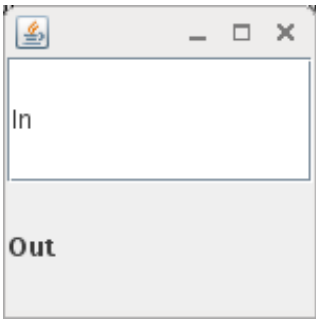
I denna uppgift skall du skriva en liten händelsebaserad applikation (enligt bilderna nedan).

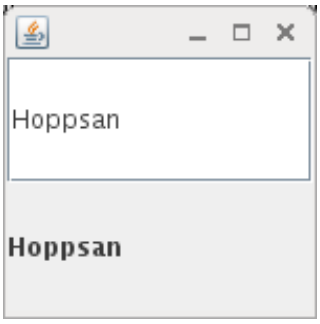
När programmet startas visas ett fönster enligt den första bilden nedan. Användaren kan skriva in text i en textruta (där det står In i första bilden). Då användaren trycker enter (med markören i textrutan) skrivs den aktuella texten ut till etiketten (label) nedanför (där det står Out i första bilden). Om användaren skriver in texten Hoppsan och gör ett enterslag skall resultatet bli enligt den andra bilden nedan.

Du skall komplettera den givna klassen MainFrame, se kommentarer i denna.

Att notera :

- Både textrutor och etiketter har metoder för att avläsa och sätta texten (setText()/getText()) .
- Klassen MainFrame skall fungera som lyssnare för textrutan, den skall alltså ha en lyssnarmetod.
- Alla händelseobjekt har en metod getSource() som ger en referens till komponenten som genererat händelsen. Typen för referensen är Object.




(5 poäng)

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

**Bilaga A:**

```
public class Player {
    private final String name;
    private int total;

    public Player(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public int getTotal() {
        return total;
    }

    public void addPoints(int points) {
        total += points;
    }

    @Override
    public String toString() {
        return "{" + name + " : " + total + "}";
    }
}//Player


import java.util.ArrayList;
import java.util.List;
public class PigOptions {
    private PigOptions() {}
    //private static final int POINTS_TO_WIN = 100;
    private static final int POINTS_TO_WIN = 20;  //During development
    private static final List<Player> players = new ArrayList<>();

    public static List<Player> getPlayers() {
        // Default players
        players.add(new Player("Anna"));
        players.add(new Player("Urban"));
        players.add(new Player("Fia"));
        return players;
    }

    public static int getPointsToWin() {
        return POINTS_TO_WIN;
    }
}// PigOptions
```

**Bilaga B:**

```
// En omgång av spelet Gris där vinst utgör vid 20 poäng
Pig started
Players are {Anna : 0} {Urban : 0} {Fia : 0}
Player is {Fia : 0}
> r
5
> n
{Anna : 0} {Urban : 0} {Fia : 5}        <-------  Fias total ökas med summa för runda
Player is {Anna : 0}
> r
1
{Anna : 0} {Urban : 0} {Fia : 5}        <-------  Anna fick inga poäng (slog en 1:a)
Player is {Urban : 0}
> r
5
> r
2
> r
6
> n
{Anna : 0} {Urban : 13} {Fia : 5}       <-------  Urbans total ökas med summa för runda
Player is {Fia : 5}
> r
6
> r
2
> r
3
> n
{Anna : 0} {Urban : 13} {Fia : 16}
Player is {Anna : 0}
> r
2
> r
1
{Anna : 0} {Urban : 13} {Fia : 16}
Player is {Urban : 13}
> r
3
> r
2
> r
4
> n
{Anna : 0} {Urban : 22} {Fia : 16}      <-------  Vinst då man byter till nästa spelare
Game over! Winner is {Urban : 22}
// Om spelet avbrutet skrivs följande ut (kommandot q)
Game aborted! {Anna : ...} {Urban : ...} {Fia : ...}
```