DAT043 – Objektorienterad Programmering
Tentamen 2018-08-22
Tid: 08.30-12.30

Ansvarig lärare: Moa Johansson
Tfn: 031 772 10 78

Ansvarig lärare besöker tentamenssalarna ca klockan 9.30 samt 11.00.

Tentamensregler

Tentamen består av två delar. För att få **godkänt på tentan (betyg 3) måste man lösa minst fem av sju uppgifter i Del 1** (frågor numrerade 1–7). För att får högre betyg krävs att man utöver detta även löser uppgifter i Del 2 (frågor 8–9). För att få **betyg 4 ska, utöver godkänt på Del 1, även en fråga från Del 2 lösas**. För **betyg 5 skall, utöver godkänt på Del 1, dessutom båda frågorna i del 2 lösas**. Förutsatt att man klarat Del 1 får man självklart testa att lösa båda uppgifterna i Del 2. Lyckas man inte med någon har man ändå säkrat betyg 3, lyckas man bara med den ena så får man betyg 4. Minuspoäng ges ej.

Poäng på Del 2 kan *eventuellt* räknas mot godkänt om det skulle behövas (t.ex. fyra godkända lösningar på Del 1 och en godkänd lösning på Del 2 kan ge en trea i betyg), men det är osannolikt att man klarar frågorna i Del 2 om man inte samtidigt klarat Del 1.

Tillåtna hjälpmedel: Den tresidiga lathund som finns tillgänglig på kurssidan och upptryckt i ett exemplar till varje student vid tentamen (man ska alltså inte ta med sig en egen kopia).

Tentamensgranskning: Efter rättning finns tentamen tillgänglig på expeditionen på plan 4 i EDIT huset och kan granskas där. Önskar man diskutera rättningen kommer man kunna boka tid för detta genom att fylla i det formulär som kommer finnas länkat till från kurshemsidan efter att tentan är rättad. Notera att tentan i så fall ska lämnas kvar på expeditionen.

- Implementeringar ska skrivas i Java. Oväsentliga syntax- och namnfel eller liknande, betyder inte att lösningen underkänns. Det viktiga är att lösningar bedöms vara tillfredställande, d.v.s. att studenten med sitt svar tydligt visar förståelse för problemet i uppgiften och dess lösning.
- Skriv tydligt och välstrukturerat. Svårförståeliga lösningar kan underkännas.
- Lösningar som är onödigt krångliga eller inte följer god programmeringsstil, dels allmänt och dels med tanke på idéerna med objektorienterad programmering, kan underkännas.
- Om det inte uttryckligen står motsatsen i uppgiften kan du använda klasser och metoder i Java:s API.
- Om det inte uttryckligen står motsatsen i uppgiften får du definiera egna hjälpmetoder.
- Importeringar av klasser i Java:s API behöver inte skrivas ut.

Lycka till!

DEL 1

För att få godkänt på tentan (betyg 3) måste du **lösa minst fem av sju uppgifte**r i denna del.

Uppgift 1

Implementera en klass `Person` som ska representera följande persondata:

- `Förnamn`
- `Efternamn`
- `Ålder`
- `Längd`
- `Vikt`

Välj lämpliga typer för värdena ovan (från de typer som finns inbyggda i Java). Ålder ska representeras av ett heltal, medan längd och vikt ska anges som ett decimaltal.

Förnamn, efternamn samt ålder **ska vara tillgängliga** utanför klassen `Person`, medan värdena för längd och vikt **inte skall vara tillgängliga** utanför klassen. **Använd lämpliga tillgänglighetsmodifierare.**

Klassen ska ha **två konstruktorer**: En som tar initiala värden för **samtliga fem värden** ovan som argument, och en som enbart tar **tre argument**: för- och efternamn samt ålder (längd och vikt ska då få s.k. default-värden för den valda typen).

Uppgift 2

Skriv en metod som **transponerar en matris**. Att transponera en matris betyder att raderna och kolumnerna ”byter plats”. Som exempel, om vi tar en matris:

```
1  2  3  4
5  6  7  8
```

och transponerar den, så får vi resultatet:

```
1   5
2   6
3   7
4   8
```

Notera att om man transponerar matrisen en gång till, så får man tillbaka den ursprungliga matrisen.

Du ska alltså implementera en metod:

```
public static double[][] transponera(double[][] a)
```

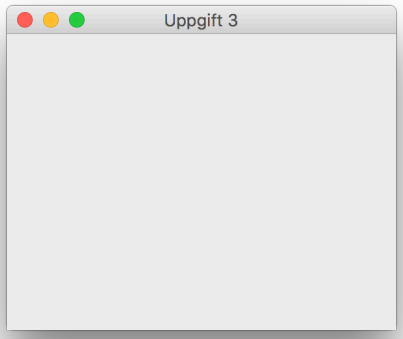
Matrisen representeras vårt fall av en tvådimensionell array i Java, där första index anger rad och andra index anger kolumn. Metoden ska fungera oavsett matrisens dimensioner.

Uppgift 3

Studera programmet i rutan nedan. Det ska visa en GUI-applikation med en knapp längst ner i fönstret som skriver ut texten ”Löst uppgiften” när man trycker på den.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class Uppgift3 implements Runnable {
6
7    public static void main(String[] args) {
8        SwingUtilities.invokeLater(new Uppgift3());
9    }
10
11   private class ButtonActionListener implements ActionListener {
12      @Override
13      public void actionPerformed(ActionEvent e) {
14          System.out.println("Löst uppgiften!");
15      }
16   }
17
18     public void run() {
19         JFrame frame = new JFrame("Uppgift 3");
20         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21         JButton button = new JButton("Skriv 'Löst uppgiften!'");
22         button.addActionListener(new ButtonActionListener());
23         frame.setPreferredSize(new Dimension(300, 250));
24         frame.pack();
25         frame.setVisible(true);
26     }
27   }
```

Men något är fel och när man kör programmet visas bara ett tomt fönster:



Vad är felet? Beskriv kortfattat i en mening eller två. Skriv även rättad kod och ange var i programmet du skulle lägga in din rättelse (radnummer). Du behöver alltså inte skriva in hela programmet igen i ditt svar, enbart det du vill ändra.

Uppgift 4

Följande enkla lilla program ska skriva ut värden i en array till terminalfönstret.

```
1  import java.util.*;
2
3  public class Uppgift4{
4
5    public static void main(String[] args) {
6      int [] a = {1,2,3,4};
7      System.out.println(a);
8    }
9  }
```

Dock får vi följande kryptiska output när vi kör programmet:

```
> java Uppgift4
[I@6073f712
```

Förklara kortfattat vad detta betyder och beror på i en eller några få meningar. Rätta även programmet så det skriver ut de förväntade värdena istället:

```
[1, 2, 3, 4]
```

Uppgift 5

Studera följande interface och klasser i Java.

```
interface MyInterface{
  String method1();
  int method2(int arg);
  default boolean method3(int arg){
        return arg > 0;
   }
 }
```

```
 interface AnotherInterface{
   String method4();
  }
```

```
class A implements MyInterface, AnotherInterface{ }
```

```
class B implements MyInterface{ }
```

Vilka av följande påståenden **är sanna**?

A. `MyInterface` är felaktigt eftersom `method3` har en implementation. Inga metoder i interfaces får ha implementationer. Om man tar bort implementationen är `MyInterface` korrekt, inte annars.
B. `MyInterface` är redan korrekt eftersom `method3` har deklarerats med nyckelordet `default`. Då är det OK att ha en implementation.
C. Klass A är felaktig eftersom den inte kan implementera både `MyInterface` och `AnotherInterface`.
D. Klass B måste implementera `method1` och `method2`.
E. Klass B får implementera `method1` och `method2`, men den måste inte.
F. Klass B får inte implementera `method3`.
G. Klass B måste implementera `method3`.
H. Klass B får implementera `method3` men den måste inte.

Uppgift 6

En programmerare har fått i uppgift att skriva en Javaklass för att representera värden från olika termometrar som är placerade på olika ställen. Varje termometer-objekt ska hålla reda på sin senaste avläsning i en instansvariabel `currentTemp`. Klassen ska även hålla reda på de **maximala och minimala temperaturerna som uppmätts globalt, dvs. av någon av alla termometer-objekt**. Dessa ska representeras av två klassvariabler `max` och `min`.

Nedan är programmerarens första försök, som innehåller misstag.

```
1  class Termometer{
2    public double max;
3    public double min;
4    private double currentTemp;
5
6    // Make a new reading of the current temperature
7    public void setTemp(double newTemp){
8      if (newTemp > max)
9        max = newTemp;
10     if (newTemp < min)
11       min = newTemp;
12     currentTemp = newTemp;
13   }
14   // Read off the latest recorded temperature
15   public double readTermometer(){return currentTemp;};
16 }
```

Koden ovan kompilerar, men när en testare försöker skriva ett testprogram kompilerar inte testprogrammet:

```
class TestTermometer{
  public static void main(String[] args) {
    Termometer t1 = new Termometer();
    Termometer t2 = new Termometer();
    t1.setTemp(23.3);
    t2.setTemp(-4);
    t2.setTemp(-12.2);
    System.out.println("Förväntat Max: 23.3. Max uppmätt: " + Termometer.max);
    System.out.println("Förväntat Min: -12.2. Min uppmätt: " + Termometer.min);
  }
}
```

Hen får följande felmeddelande:

```
> javac TestTermometer.java
TestTermometer.java:30: error: non-static variable max cannot be referenced from a
static context
    System.out.println("Förväntat Max: 23.3. Max uppmätt: " + Termometer.max);
                                                                        ^
TestTermomenter.java:31: error: non-static variable min cannot be referenced from a
static context
    System.out.println("Förväntat Min: -12.2. Min uppmätt: " + Termometer.min);
                                                                        ^
2 errors
```

Aha! tänker testaren och skriver en bugrapport till programmeraren. Förklara kortfattat vad felet är och rätta även den/de rader i programmet som orsakar problemet.

Uppgift 7

Klassen `Box` är tänkt att representera en låda i vilken man kan stoppa ner objekt av godtycklig typ. En gammal implementation har därför använt typen `Object` för att representera lådans innehåll. Detta har dock nackdelen att man lätt kan göra misstag som inte upptäcks vid kompilering. I `main`-metoden nedan är det exempelvis tänkt att `b1` ska innehålla en `String`, medan `b2` ska innehålla en `Integer`. Koden kompilerar, men när programmet körs får man ett `ClassCastException` på rad 22, eftersom man av misstag lagt en `Integer` i `b1`.

```
1  class Box{
2    private Object content;
3
4    public void put(Object thing){
5      content = thing;
6    }
7    public Object remove(){
8      Object thing = content;
9      content = null;
10     return thing;
11   }
12
13   public static void main(String[] args) {
14     Box b1 = new Box(); //Tänkt att innehålla en String
15     Box b2 = new Box(); //Tänkt att innehålla en Integer
16     b1.put("hello");
17     b2.put(0);
18     b1.put("world");
19     b1.put(1);
20     String s = (String) b1.remove(); //ClassCastException
21     Integer i = (Integer) b2.remove();
22   }
23 }
```

Detta skulle ha upptäckts redan vid kompilering om generics hade använts. Skriv därför om klassen `Box` så att den använder generics istället! Inkludera även en `main`-metod där du skapar två Box-objekt, en för `String` och en för `Integer`. Lägg in och ta ut minst en sak från något av Box-objekten.

DEL 2

Du behöver **bara svara på dessa frågor om du aspirerar på betyg 4 eller 5**. Vill du ha fyra ska du lösa en uppgift (välj själv) och för femma ska du lösa båda uppgifterna i den här delen, utöver de uppgifter du löst i Del 1.

Uppgift 8

Urvalssortering (selection sort) är en enkel sökalgoritm som vi stött på under kursens gång. Vi påminner oss om algoritmen som kan beskrivas med följande steg:

1. Sök igenom arrayen för att hitta det minsta elementet.
2. Flytta detta till den första positionen.
3. Sök efter det näst största elementet.
4. Flytta detta till den andra positionen.
5. … och så vidare tills vi sorterat hela arrayen.

Algoritmen kan alltså sägas hålla reda på en sorterad och en osorterad ”sektion” av arrayen i varje steg. I början är den sorterade delen tom och hela arrayen ligger i den osorterade sektionen. Efter steg 1-2 ovan innehåller den sorterade delen ett element (det första och minsta) och den osorterad delen resten. Efter steg 3-4 innehåller den sorterade delen två element och resten ligger i den osorterade delen. När algorithmen är klar hör hela arrayen till den sorterade sektionen, och den osorterade är tom. Notera att algoritmen för urvalssortering sorterar arrayen ”in-place” dvs. inget extra utrymme behövs.

Du ska implementera en metod `selectionSort` som sorterar en array:

- Metoden ska vara *generisk* och fungera för typer vilka implementerar interfacet `Comparable` (du ska alltså *inte* använda typen `Object` för argumenten utan ge argumenten generiska typer).
- Metoden ska en ta array med element av generisk typ som argument.
- Metoden ska ha returtyp void.

Vi påminner om att metoden `int compareTo(T arg)` returnerar:

- `0` om elementet som jämförs är lika med `arg`,
- ett heltal (`int`) mindre än 0 om elementet är mindre än `arg`,
- ett heltal större än 0 om elementet är större än `arg`.

OBS: Att använda sig av en färdig sorteringsmetod från Javas bibliotek ger naturligtvis **inte** rätt svar på denna fråga. *Du ska implementera algoritmen för urvalssortering själv.*

Exempel på användning:

```
String[] myStrArr = {”b”, ”a”, ”d”, ”c”};
selectionSort(myStrArr);  //Ska nu innehålla {”a”, ”b”, ”c”, ”d”}

Integer[] myIntArr = {2, 4, 3, 1};
selectionSort(myStrArr);  //Ska nu innehålla {1, 2, 3, 4}.
```

Tips: Studera interfacet `Comparable` i lathunden.

Uppgift 9

Anta att du arbetar med ett program för att läsa in och hantera information om bilar. Du kan anta att du har tillgång till en klass Car för att representera bilar:

```
class Car{
  public String brand;
  public String model;
  public int year;

  public Car(String brand, String model, int year){…}
  public String toString(){…}
}
```

Din uppgift är att skriva en metod `getSelectedCarList` som ska:

- Läsa in en fil som innehåller information om olika bilar (filnamnet ges som argument). Filen innehåller text, och på varje rad finns information om en bil i följande format: *Märke, Modell, År.* De tre värdena är separerade med ett kommatecken. En inputfil `Uppgift9.txt` kan alltså antas se ut t.ex. så här:

  ```
  Volvo,Amazon,1971
  Volvo,V40,2017
  Saab,900,1997
  Tesla,Model3,2018
  Volkswagen,Mini,2007
  Fiat,500,2016
  Volvo,V40,2015
  ```

  Om filen inte kan hittas ska metoden hantera det undantag som då kastas på ett lämpligt sätt (t.ex. skriva ut ett felmeddelande och returnera en tom lista). Om någon rad inte innehåller tre värden ska detta också hanteras (t.ex. skriva ut ett felmeddelande och returnera en tom lista)
- Metoden ska skapa ett objekt av klassen Car för varje bil i inputfilen.
- Metoden ska därefter filtrera dessa objekt enligt ett godtyckligt villkor (tips: detta ska ges som ett argument till metoden). Metoden ska alltså kunna användas för att filtrera bilarna i inputfilen på många olika sätt.
- Metoden ska sedan returnera en lista av de Car-objekt som uppfyller det givna villkoret som en `List<Car>`.
- Skriv ner hur du anropar metoden för att **skapa en lista som innehåller samtliga Volvobilar av årsmodell från 2000 eller senare** från exempelfilen `Uppgift9.txt`. Denna lista ska alltså innehålla bilarna:

  ```
  Volvo V40, (2017)
  Volvo V40, (2015)
  ```