

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 få 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 namnfejl 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!

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 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 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);
                                   ^
TestTermometer.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.

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 osorterade delen resten. Efter steg 3-4 innehåller den sorterade delen två element och resten ligger i den osorterade delen. När algoritmen är klar har 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 ta en 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(myIntArr); //Ska nu innehålla {1, 2, 3, 4}.
```

Tips: Studera interfacet `Comparable` i lathunden.

Typer

Primitiva typer

prim. type	motsv. klass	ex. literaler	default
boolean	Boolean	true, false	false
char	Character	'A', '3', '\n'	'\u0000'
int	Integer	37, -3, 12345	0
double	Double	3.1416, 1E-10	0.0

Referensyper

- Array/fält:
Exempel: `int[], Ball[], double[] []`
Skapa objekt: `int[] a = new int[10];`
Initiering: `double[] [] data = {{1,3},{4,8}};`
Indexering: `a[i], 0 <= i < a.length.`
- Klasser:
Skapas med konstruerare:
`Ball b = new Ball(10,20,Color.RED);`
`LifeModel model = new LifeModel(50,50);`
- Interfaces/gränssnitt:
Deklarerar metoder med resultattyp, namn, parametrar, undantag. Objekt kan inte skapas, men klasser kan implementera ett interface.

Uppräkningstyper

`enum E {VAL1, VAL2, ...}`

Variabler

Variabler måste deklaras: `int x; double[] ys;`

Initiering

Instansvariabler and array-element initieras till default-värdet för primitiva typer och null för referenstyper. Lokala variabler måste initieras explicit.

Uttryck

Uttryck byggs av variabler, literaler, operatorer och metoder.

Binära operatorer i precedensordning

operator	argtyp	restyp
<code>*</code> , <code>/</code> , <code>%</code>	number	number
<code>+</code> , <code>-</code>	number*	number*
<code><</code> , <code><=</code> , <code>></code> , <code>>=</code>	number	boolean
<code>==</code> , <code>!=</code>	any	boolean
<code>&&</code>	boolean	boolean
<code> </code>	boolean	boolean
<code>=</code> , <code>+=</code> , <code>-=</code>	var, t/number	t/number

`*`) + kan också ha String som argtyp och restyp.
number betyder numerisk primitiv typ.

Andra operatorer

operator	argtyp	restyp
<code>expr++</code> , <code>expr--</code>	number	number
<code>++expr</code> , <code>--expr</code> , <code>-expr</code>	number	number
<code>!expr</code>	boolean	boolean
<code>expr?expr:expr</code>	boolean, <i>t</i> , <i>t</i>	<i>t</i>

Typomvandling

Omvandling av ett värde av typen `int` eller `double` till `double` eller `String` sker implicit vid behov i uttryck. I omvänd riktning krävs explicit typomvandling, t. ex. `(int)`. Typomvandling från en klass till en superklass sker automatiskt. I omvänd riktning krävs explicit typomvandling. Operatorm *instanceof* kan användas för att avgöra om omvandling är möjlig.

Omvandling mellan primitiva typer och dess motsvarande klass sker automatiskt i båda riktningarna.

Generics

```
class C<T1, T2, ...> {...}
interface I<T1, T2, ...> {...}
<T1, T2, ...> rettyp m(argtyp1 argnamn1, ...)
```

Modifierare och andra nyckelord

`abstract`, `static`, `final`, `private`, `protected`,

`public`
`extends`, `implements`, `super`, `this`, `throw`,
`throws`, `void`

Satser

<i>expr</i> ;	<i>typ var</i> ;
	<i>typ var</i> = <i>init</i> ;
<code>break</code> ;	<code>continue</code> ;
<code>return</code> ;	<code>return expr</code> ;
<code>if (test) {</code> <i>statements</i> <code>}</code> <code>else {</code> <i>statements</i> <code>}</code>	<code>if (test) {</code> <i>statements</i> <code>}</code> <code>else {</code> <i>statements</i> <code>}</code>
<code>while (test) {</code> <i>statements</i> <code>}</code>	<code>do {</code> <i>statements</i> <code>}</code> <code>while (test);</code>
<code>for (init; test; upd) {</code> <i>statements</i> <code>}</code>	<code>for (type var : expr) {</code> <i>statements</i> <code>}</code>
<code>switch (expr) {</code> case <i>lit1</i> : <i>stmt</i> ... case <i>litN</i> : <i>stmt</i> <code>}</code>	<code>try {</code> <i>statements</i> <code>} catch (exc-type var) {</code> <i>statements</i> <code>}</code>
lambda-uttryck: <i>params</i> -> <i>funktionskropp</i>	

Klasser

- Funktionsbibliotek (Ex: `Math`, `Arrays`). Innehåller bara statiska funktioner/subrutiner.
- Mallar från vilka objekt skapas (de flesta klasser). Innehåller oftast inte statiska metoder. I stället instansvariabler, konstruere, metoder.
- Huvudklassen i en applikation. Innehåller `public static void main(String[] args)`

java.util.function

Interfacen nedan är s.k. *funktions-interface*.

interface Predicate<T>

Representerar ett predikat (*funktion som returnerar en boolean*) med ett argument.

boolean test(T t)

default Predicate<T> and

(Predicate<? super T> otherP)

default Predicate<T> or

(Predicate<? super T> otherP)

interface Consumer<T>

Representerar en operation som tar ett argument och inle returnerar något resultat.

void accept(T t)

interface Function<T,R>

Representerar en funktion som tar ett argument och producerar ett resultat av typ R.

R apply (T t)

AWT/Swing (förenklat)

class java.awt.Component

void addMouseListener(MouseListener l)

void repaint()

void setBackground(Color c)

void setPreferredSize(Dimension d)

void setVisible(boolean b)

class java.awt.Container
extends Component

Component add(Component comp)

void setLayout(LayoutManager mgr)

javax.swing.JFrame extends Container

JFrame()
JFrame(String title)
Container getContentPane()
void pack()

javax.swing.JPanel
extends Container

JPanel()

void paintComponent(Graphics g)

javax.swing.JButton
extends Container

JButton(String text)

void addActionListener(
ActionListener l)

interface java.awt.event.MouseListener

voidMouseClicked(MouseEvent e)

interface java.awt.event.ActionListener

void actionPerformed(ActionEvent e)

java.awt.event.MouseEvent

int getX()

int getY()

java.awt.BorderLayout
implements LayoutManager

static String CENTER, WEST, SOUTH, ...

java.awt.Graphics

void setColor(Color c)

void drawLine(int x1,int y1,
int x2,int y2)

void drawOval(int x,int y,int w,int h)

void drawRect(int x,int y,int w,int h)

void fillOval(int x,int y,int w,int h)

void fillRect(int x,int y,int w,int h)

void drawString(String str,int x,int y)