Lösningsförslag: Instuderingsfrågor, del 1

Uppgift 1.

Följande regler gäller för namngivning av identifierare i Java:

1. Ett identifierarnamn består endast av bokstäver, siffror, understrykningstecken('_') och dollartecken ('$').
2. Ett identifierarnamn får vara godtyckligt långt.
3. Ett identifierarnamn får ej börja med en siffra.
4. De reserverade orden i Java är otillåtna identifierarnamn.

Följande identifierare är således ogiltiga:

any.time	innehåller tecket '.'
1stStreet	inleds med en siffra
now&then	innehåller tecket '&'
float	**float** är ett reserverat ord
tooHot?	innehåller tecket '?'
final score	innehåller ett blanktecken

Uppgift 2.

a) 4	b) 13	c) 4
d) 1	e) 2	f) 4
g) 5	h) 2	i) 1

Uppgift 3.

a) Antalet är 17 styck.
b) Antalet är + myNumber + styck.
c) Antalet är 1712 styck.
d) Antalet är 29 styck.

Uppgift 4.

a) 1	b) 3	c) 5.1
d) 28	e) 1.275	f) 13.4
g) 25.6	h) 1.0999999999999996	i) 1.3000000000000007

Uppgift 5.

a) **int**	b) **double**	c) String
d) **boolean**	e) String	f) **char**

Uppgift 6.

Uttrycken x + y * z och x + (y * z) är ekvivalenta, eftersom * har högre prioritet än +.

Uppgift 7.

Följande satser är fel:

int c = 3, d = 4.5;	variabeln d är av typen **int**, men 4.5 är av typen **double**
double x, a;	variabeln a är redan deklarerad som en **int**
char teck = "y";	variabeln teck är av typen **char**, men "y" av typen String
String str = 'en text';	variabeln str refererar till ett objekt av typen String, men 'en text' är ingen sträng, utan ett otillåtet uttryck. En strängliteral anges av "-tecken.

Uppgift 8.

a) alt = alt + base + col4 + sum;
b) pressure:= (temp + entropy) * spec2;
c) gradient:= gradient - (hgt - slope);
d) eff = eff + full * Math.exp(h3 * Math.log(loss));
e) x = -b + Math.sqrt(b * b - 4 * a * c);

Uppgift 9.

Följande tilldelningar är felaktiga:

c) i + j = k;

Vänsterledet i tilldelningssatsen skall vara den variabel som skall tilldelas uttrycket i högerledet av tilldelningssatsen. Här står uttrycket i vänsterledet och variabeln i högerledet.

h) y = a + b;

Variablerna a och b är av datatypen **boolean** och operatorn + finns ej för denna datatyp.

i) i = x + y;

Variabeln i är av datatypen **int**, medan uttrycket x + y är av datatypen **double**. Således uppstår en typkonflikt, eftersom ett värde av typen **double** inte kan lagras i en variabel av typen **int**.

Uppgift 10.

Implicit typomvandling innebär att en variabel av en "större" typ automatiskt kan tilldelas ett värde av en "mindre" typ, eftersom det "mindre" värdet ryms i den "större" typen. Exempelvis kan en variabel av typen **double** tilldelas ett värde av typen **int** (datatypen **double** inkluderar alla heltal som finns i **int**).

Explicit typomvandling innebär att man med hjälp av en speciell sats kan begära att ett värde av en viss typ görs om till ett värde av en annan typ. Detta är nödvändigt om man vill tilldela ett värde en "större" typ till ett värde av en "mindre" typ. Typomvandlingen innebär i detta fall ofta att det nya värdet endast blir ett närmevärde till det gamla värdet. Exempelvis kan en värde av **double** omvandlas till ett värde av **int** med explicit typomvandling på följande sätt

```
int intTal = (int) 10.1234567;          //värdet blir 10
```

Uppgift 11.

Tilldelningssatsen

```
MAX_SIZE = 50;
```

ger ett kompileringsfel, eftersom MAX_SIZE är en konstant och således inte får tilldelas ett nytt värde.

Uppgift 12.

a) 81.0 b) 4.0 c) 26 d) 13.0

Uppgift 13.

a) 1 / (2 * Math.PI * Math.sqrt(L * C))
b) C * h / Math.pow(Math.pow(h, 2) + Math.pow(d, 2), 3)
c) Math.sin(Math.pow(x, 2) + Math.pow(y, 2))
d) Math.sqrt((a+b)*(a-b)) / (Math.pow(a,2) + 1 / (Math.pow(a, 3) - 2))

Uppgift 14.

För att kunna använda dialogrutan, dvs anropa metoden JOptionPane.showMessageDialog, måste vi ha tillgång till paketet javax.Swing. Därför måste vi först i programmet ge satsen

```
import javax.swing.*;
```

Uppgift 15.

a) index = index + 1;
b) factor = shoeSize + shoulderBreadth;
c) factor = Math.sqrt(factor / weight);
d) start = **true**;
e) flag = size > index;
f) finished = (ch1 == ch2) || (ch1 == 'n') || (ch1 = 'N');

Uppgift 16.

```
year = date / 10000;
month = date % 10000 / 100;
day = date % 100;
```

Uppgift 17.

Följande tilldelningar är felaktiga:

b) i = x / k;

Variabeln i är av datatypen **int**, medan uttrycket x /k är av datatypen **double**. Således uppstår en typkonflikt, eftersom ett värde av typen **double** inte kan lagras i en variabel av typen **int**.

c) i = Math.pow(k , 2);

Variabeln i är av datatypen **int**, medan uttrycket Math.pow(k ,2) är av datatypen **double**. Således uppstår en typkonflikt, eftersom ett värde av typen **double** inte kan lagras i en variabel av typen **int**.

Uppgift 18.

Genom att använda någon av metoderna System.out.printf eller String.format:

```
System.out.printf("%.3f", size);
```
eller
```
System.out.print(String.format("%.3f", size));
```

Uppgift 19.

Genom att använda någon av metoderna System.out.printf eller String.format:

```
System.out.printf("%02d:%02d:%02d", tim, min, sek);
```
eller
```
System.out.println(String.format("%02d:%02d:%02d", tim, min, sek));
```

Uppgift 20.

Man använder sig av ett objekt av klassen Scanner:

```
import java.util.*; //Scanner finns i detta paket och måste
. . .
    String indata = JOptionPane.showInputDialog("Ange önskade tre värden: ");
    Scanner sc = new Scanner(indata);
    int number = sc.nextInt();
    double freq = sc.nextDouble();
    int count = sc.nextInt();
    . . .
```

Uppgift 21.

Programmet blir mycket lättare att läsa och dess logiska uppbyggnad lättare att förstå, vilket t.ex. underlättar när man söker fel i ett program.

Uppgift 22.

Informellt kan man säga att en algoritm är en beskrivning av hur ett problem skall lösas. En algoritm består av en ordnad, ändlig följd av elementära och entydiga instruktioner.

Uppgift 23.

Sekvens, selektion och iteration.

Uppgift 24.

- Definition av problemet (uppgiftsformulering).
- Analys av problemet.
- Framtagning av lösningsskiss.
- Val och representation av algoritmer.
- Kodning.
- Testning och validering (försäkra sig om att programmet löser uppgiften).
- Dokumentation.
- Programunderhåll.

Lösningsförslag: Instuderingsfrågor, del 2

Uppgift 25.

a) x > y && y > z

b) x < 0 && y < 0

c) !(x < 0 && y < 0)

d) x == y && x != z

Uppgift 26.

c

Uppgift 27.

```
!(z == x) || (x > 2) && (y > 3)
= !false || true && false          (! har högre prioritet än || och &&)
= true || true && false            (&& har högre prioritet än ||)
=  true || false
 =  true
```

Uppgift 28.

```
delbar = tal % 7 == 0;
```

Uppgift 29.

a) x > 3

b) y >= 2 && y <= 5

c) r < 0 && z >=0

d) (a < 0 && b < 0) || (a>= 0 && B >= 0)

e) p == q && q != r

Uppgift 30.

a) **true** b) **false** c) **true** d) **true**

Uppgift 31.

a) **false** b) **false** c) **false** d) **true**

Uppgift 32.

a) **false** b) **true** c) **true** d) **false**

Uppgift 33.

number >= 80 && number <=90

Uppgift 34.

Problemet med reella tal är att dessa inte kan lagras exakt och att det beräkningar uppstår trunkeringsfel och kancellationsfel. Därför kan man inte jämföra på exakthet när det gäller reella tal utan man får istället jämföra på tillräcklig noggrannhet. Vad tillräcklig noggrannhet är i de aktuella frågeställningarna kan väl diskuteras, men en avvikelse mellan vikterna på några tiondels procent kan väl vara acceptabelt.

a) Math.abs(weight1 - weight2) < 0.1,

b) Math.abs(weight1 -weight2) < 0.00001

c) Math.abs(weight1 -weight2) < 50

Uppgift 35.

a) value >= 1.0

b) value < 0.0 || value >= 1.0

c) smallNumber != 0 && bigNumber != 10000

d) value > 1.0

Uppgift 36.

Tabellen nedan visar att de tre uttrycken är ekvivalenta

x < 10	(x < 10) == **true**	x >= 10	(x >= 10) == **false**
true	**true**	**false**	**true**
false	**false**	**true**	**false**

Det enklaste av dessa villkor att förstå torde vara x < 10

Uppgift 37.

a) Always

b) OK

Uppgift 38.

Utskriften blir:

```
z = 10.0
```

Orsaken till detta är att **else**-satsen inte hör till den yttre **if**-satsen, som indenteringen av programsatserna kan antyda, utan till den inre **if**-satsen. Rätt indenterat ser programsekvensen ut enligt följande:

```
int x = -1, y = 4, z = 10;
if (x > 0)
    if (y > 0)
        z = Math.sqrt(x) + Math.sqrt(y);
    else
        z = 0;
System.out.println("z = " + z);
```

Uppgift 39.

Orsaken är att det står ett semikolon efter villkoret i **if**-satsen!! Således får vi en **if**-sats med en tom sats och utan **else**-del. Alltså hör inte **else** till någon **if**-sats! Om vi indenterar programmet som det logiskt tolkas syns detta tydligare:

```
if (x == 0)
    ;
x = 100;
else
    x = x + 50;
```

Uppgift 40.

Tabellen nedan visar att b och !test har samma värde

test	!test	b
true	**false**	**false**
false	**true**	**true**

Uppgift 41.

a) 12.5

b) 30.0

Uppgift 42.

När man har en **if**-sats inne i en annan **if**-sats säger man att man har nästlade **if**-satser.

Uppgift 43.

a)
```
if (a > c && b > c)
    x = y;
else
    x = z;
```

b)
```
if (a == b || a > c)
    x = y;
else
    x = z;
```

Uppgift 44.

I programsekvens a) har vi en **if**-sats med ett tvåvägsalternativ, medan vi i programsekvens b) har två **if**-satser som båda har envägsvalsalternativ. Så i programsekvens a) kan högst en av tilldelningssatserna genomlöpas, medan i programsekvens b) kan båda tilldelningssatserna genomlöpas.

Om x har värdet -1 innan respektive programsekvens kommer x att ha värdet -1 efter att sekvens a) genomlöpts och värdet -1 efter att sekvens b) genomlöpts.

Om x har värdet 1 innan respektive programsekvens kommer x att ha värdet 2 efter att sekvens a) genomlöpts och värdet 4 efter att sekvens b) genomlöpts.

Uppgift 45.

```
finished = answer =='Q';
```

Uppgift 46.

Variabeln y deklareras i programblocket som hör till **if**-satsen och är därför en lokal variabel i detta programblock, medan satsen System.out.println(y) ligger utanför detta block. Variabeln y är således okänd i denna sats.

Uppgift 47.

```
if (x < 0 && y < 3 && z = 6)
    a = x + y + z;
```

Uppgift 48.

"Algoritm" b) är felaktig. Vi betalar med check även om vi redan har betalt med kontanter.

Uppgift 49.

Följande uttryck

```
(finished && !dead) || (dead && !finished)
```

och

```
finished != dead
```

är logiskt ekvivalenta.

Lösningsförslag: Instuderingsfrågor, del 3

Uppgift 50.

a) Utskriften blir:

```
10 minutes has passed.
30 minutes has passed.
50 minutes has passed.
```

b) Utskriften blir:

```
30
```

c) Utskriften blir:

```
Resultat: 1
Resultat: 2
Resultat: 0
Resultat: 1
Resultat: 2
```

d) Utskriften blir:

```
a = 1 b = 2
a = 2 b = 2
a = 3 b = 2
a = 4 b = 4
a = 5 b = 6
a = 6 b = 8
```

e) Utskriften blir:

```
Sum is 20
Count is 5
```

Uppgift 51.

a)
```
double i = 0.25;
while (i <= 5.0) {
   System.out.print(i + " ");
   i = i + 0.25;
}
System.out.println();
```

b)
```
int i = 1, sum = 0;;
while (i <= 8) {
   sum = sum + i;
   System.out.print(sum + " ");
   i = i + 1;
}
System.out.println();
```

c)
```
while (!finished) {
   //
}
```

d)
```
while (counter > LOWER && counter < UPPER) {
   //
}
```

Uppgift 52.

b) Terminerar inte, detta pga att det uppstår avrundningsfel när vi gör beräkningar på reella tal.

c) Terminerar inte, eftersom `x` får aldrig värdet 55.

d) Terminerar inte. Villkoret säger att loopen skall fortsätta så länge som `i < 10` eller `sum != 15`, dvs loopen bryts när `x>=10` och `sum == 15`. Detta inträffar dock aldrig. Visserligen kommer `sum` att under exekveringen ha värdet 15, men dock inte samtidigt som `x >= 10`.

e) Terminerar inte. Det står nämligen ett semikolon (;) efter villkoret i **while**-satsen, vilket med all säkerhet är misstag av programmeraren. **while**-satsen består alltså endast av en tom sats och i en tom sats görs inget, varför utfallet av villkoret som styr **while**-satsen inte kommer att förändras.

Om vi indenterar programmet som det logiskt tolkas syns detta tydligare:

```
int k = 1;
while (k != 10)
      ;
{
      System.out.println(k);
      k = k +1;
}
```

Uppgift 53.

Det uppkommer en division med 0 i uttrycket 1.0/i, eftersom **while**-satsen går ett varv längre än vi tänkt oss. Detta kan givetvis åtgärdas genom att ändra villkoret i **while**-satsen så den löper ett varv mindre, men det egentliga felet är inte villkoret i **while**-satsen utan startvärdet på variabeln i! Varför sätta denna till 11 när den borde vara 10? Denna felaktiga sätta att initiera startvärdet innebär att det måste korrigeras inne i **while**-satsen genom att räknas ner med 1 innan själva beräkningarna utförs. En korrekt programsekvens har följande utseende:

```
int i = 10;
double sum = 0;
while (i >= 1) {
    sum = sum + 1.0 / i;
    i = i-1;
}
```

Uppgift 54.

a)	b)	c)	d)	e)	f)
1	4	-8	1	5	15
2	8	-4	2	7	12
3	12	0	4	9	9
0	16	4	8	11	6
1	20	8	16	13	

Uppgift 55.

Om man använder en for-sats som en generell villkorsloop är det mycket lätt att få obegriplig kod, en while-sats är att föredra vid villkorsloopar då dess syntax är mycket enklare att förstå.

Uppgift 56.

Den styrande variabeln i förändras inuti **for**-satsen! När man läser initieringen av for-satsen

```
for (int i = 1; i <= 10; i = i + 1)
```

förväntar man sig att satsen skall genomlöpas då styrande variabeln i har värdena 1,2,3,...,10. MEN, satsen genomlöps då i har värdena 1,3,5,7 och 9!

Vi har här ett bra exempel på hur man inte skall använda en **for**-sats. **for**-satsen skall enbart nyttjas som en räkneloop, vilket innebär att man aldrig inom **for**-satsen skall påverka den styrande variabeln i **for**-satsen eller villkoret i **for**-satsen. För att erhålla det önskade resultatet borde man istället använt följande sats:

```
for (int i = 1; i <= 9; i = i + 2) {
    System.out.println(i);
}
```

Uppgift 57.

a) **for**-satsen har skrivits på ett felaktigt sätt! Delarna i **for**-satsen skall separeras med semikolon (;) och inte med komma (,). Satsen skall se ut på följande sätt:

```
for (int k = 1; k <= 10; k = k +1)
    System.out.println(k);
```

b) Variabeln i deklareras två gånger inom samma programenhet. Hur detta skall rättas till beror på om variabeln i enbart skall existera inom **for**-satsen eller om den även skall existera utanför **for**-satsen.

Fall 1: variabeln i är definierad enbart inom **for**-satsen

```
for (int i = 1; i <= 10; i = i +1) {
    //här kommer en eller flera programsatser
}
```

Fall 2: variabeln i är definierad inom hela det aktuella programblocket.

```
int i;
for (i = 1; i <= 10; i = i +1) {
    //här kommer en eller flera programsatser
}
```

c) Variabeln j är odefinierad i utskriftssatsen! Det står ett semikolon efter själva **for**-satsen, vilket innebär att **for**-satsen enbart innehåller en tom sats. Eftersom variabeln j är lokal inom **for**-satsen och utskriftssatsen inte ligger i **for**-satsen är således j odefinierad i utskriftssatsen. Om vi indenterar programmet som det logiskt tolkas syns detta tydligare:

```
for (int j = 1; j <= 10; i = j +1)
    ;
{
 System.out.println(j);
      //här kommer eventuellt flera programsatser
}
```

Ett programblock får lägg in varsomhelst i ett program där en sats får förekomma.

d) Kompileringsfelet uppkommer pga att variabeln i redan är deklarerad när den deklareras i den inre **for**- satsen. Avsikten med kodavsnittet är troligen att den yttre **for**-satsen skall löpa 10 gånger och den inre **for**-satsen också skall löpa 10 gånger, varför koden borde ha följande utseende:

```
int antalVarv = 0;;
for (int i = 1; i <=10; i = i + 1)
    for (int j = 1; j <=10; j = j + 1)
        antalVarv = antalVarv + 1;
```

Uppgift 58.

a)
```
******
******
******
```

b)
```
*
**
***
```

c)
```
5670
5671
5672
5673
```

Uppgift 59.

a)
```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

b)
```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

c)
```
1 1 1 1 1
2 2 2 2
3 3 3
4 4
5
```

Uppgift 60.

a) 27 gånger

b) 18 gånger

Uppgift 61.

a)
```
for (int i = 1; i <= 15; i = i +2)
    System.out.println(i);
```

b)
```
for (double i = 0.25; i <= 5.0; i = i + 0.25) {
    System.out.println(i);
```

c)
```
for (int i = 10; i >= -10; i = i -2) {
    System.out.println(i);
```

d)
```
for (int i = 0; i <= 9; i = i + 1) {
    System.out.println(i * i);
```

Uppgift 62.

a) En **for**-sats är lämpligast, eftersom vi har en räkneloop.

b) En **while**-sats är lämpligast, eftersom vi har en villkorsloop.

c) En **for**-sats är lämpligast, eftersom vi har en räkneloop.

d) En **while**-sats är lämpligast, eftersom vi har en villkorsloop.

Uppgift 63.

Utskriften blir:

```
i = 4
```

Uppgift 64.

Variabeln more är okänd utanför programblocket, i vilket variabeln är deklarerad. Variablen more måste således deklareras utanför **do**-**while**-blocket:

```
boolean more;
do {
    i = i + 1;
    sum = sum * i;
    if (i > 6)
        more = false;
    else
        more = true;
} while(more);
```

Uppgift 65.

Lämpligen används en **do**-sats enligt:

```
do {
    String indata = JOptionPane.showInputDialog("Ge talet: ");
    int tal = Integer.parseInt(indata);
    if (tal < 1 || tal > 7)
        JOptionPane.showMessageDialog(null, "Fel indata! Försök igen");
}while ( tal < 1 || tal > 7);
```

En alternativ lösning är att lägga inläsningen i en evighetssats och hoppa ut när ett korrekt värde blivit inläst:

```
while(true) {
    String indata = JOptionPane.showInputDialog("Ge talet: ");
    int tal = Integer.parseInt(indata);
    if (tal >= 1 && tal <= 7)
        break;
    JOptionPane.showMessageDialog(null, "Fel indata! Försök igen");
}
```

Denna lösning är sämre eftersom vi lämnar loopen via en *bakdörr*. En **while**-loop är det normalt (≈alltid) villkoret som skall bestämma när loopen skall lämnas.

Lösningsförslag: Instuderingsfrågor, del 4

Uppgift 66.

Top-down-design innebär att man betraktar det ursprungliga problemet på en hög abstraktionsnivå och bryter ner det ursprungliga problemet i ett antal delproblem. Var och en av dessa delproblem betraktas sedan på en lägre abstraktionsnivå. Om nödvändigt bryts delproblemen ner i mindre och mer detaljerade delproblem. Denna process upprepas till man har delproblem som är triviala att lösa.

Fördelen med top-down-design är att man lösa det delproblemen i det ursprungliga problemet steg för steg istället för att direkt göra en fullständig lösning.

Uppgift 67.

Resultattypen **void** betyder att metoden inte returnerar något värde.

Uppgift 68.

De formella parametrarna definieras i metodens parameterlista och är lokala inom metoden. De formella parametrarna får sina värden när metoden anropas via de parametrar som anges vid anropet. De parametrar som anges vid anropet kallas för aktuella parametrar. När en metod anropas sker en bindning mellan de aktuella och formella parametern. Detta sker genom att den formella parametern tilldelas värdet av motsvarande aktuell parameter. Således gäller vid anrop till en metod att de de aktuella parametrarna måste överensstämma med de formella parametrarna i antal, typ och ordning.

Uppgift 69.

a) Falskt.

b) Falskt. En **void**-metod returnerar inget värde och behöver således inte någon **return**-sats

c) Falskt.

d) Falskt.

e) Sant.

f) Falskt.

Uppgift 70.

a) Kompileringsfel! Metoden methodOne är en **void**-metod. Det går inte att tilldela variabeln d returvärdet från metoden, eftersom det inte finns något returvärde.

b) Korrekt. Den formella parametern one till metoden har typen **double** och den aktuella parametern a har typen **int**, men **int** är typkompatibel med **double**.

c) Kompileringsfel! Den aktuella parametern c av typen **double** är inte typkompatibel med den formella parametern one av typen **int**.

d) Korrekt. Den formella parametern two till metoden har typen **double** och den aktuella parametern b har typen **int**, men **int** är typkompatibel med **double**.

d) Korrekt. Metoden returnerar ett värde av **int** som tilldelas variabeln d av typen **double**, men **int** är typkompatibel med **double**.

e) Kompileringsfel! Antalet aktuella parametrar överensstämmer inte med antalet formella parametrar.

Uppgift 71.

48

Uppgift 72.

103

Uppgift 73.

a) Metoden är deklarerad som **void**, dvs att metoden inte skall returnera något värde, dock returneras en **int**! En korrekt metod har följande utseende:

```
public int methodOne (int one) {
        return one + one;
}//methodOne
```

b) Metoden skall returnera en **boolean**, men om villkoret är falskt returneras inget värde!. En korrekt metod har följande utseende:

```
public boolean methodTwo (int a, int b) {
    if (a >2*b)
       return true;
    else
       return false;
}//methodTwo
```

Uppgift 74.

Metoden returnerar **true** om parametern x är ett jämt heltal och false om parametern x är ett udda heltal. Ett lämpligare namn är isEven.

Uppgift 75.

Metoden beräknar n^k. Ett lämpligare namn på metoden är power.

Uppgift 76.

6

Uppgift 77.

a)
```
public static int lastDigit(int number) {
   return Math.abs(number % 10);
}//lastDigit
```

b)
```
public static int firstDigit(int number) {
   int temp = Math.abs(number);
   while (temp > 10) {
      temp = temp / 10;
   }
   return temp;
}//firstDigit
```

c)
```
public static int nrOfDigits(int number) {
   int temp = Math.abs(number);
   int nrDigits = 1;
   while (temp > 10) {
      nrDigits = nrDigits +1;
      temp = temp / 10;
   }
   return nrDigits;
}//nrOfDigits
```

d)
```
public static int largestDigit(int number){
   int temp = Math.abs(number);
   int max = 0;
   while (temp > 0) {
      int digit = temp % 10;
      if (digit > max)
         max = digit;
      temp = temp / 10;
   }
   return max;
}// largestDigit
```

Uppgift 78.

a)
```
public static int squareSum(int from, int to) {
    int sum = 0;
    for (int i = from; i <= to; i = i + 1) {
        sum = sum + i*i;
    }
    return sum;
}//squareSum
```

b)
```
public static double powerSum(double x, int n) {
    double term = x;
    double sum = 1;
    for (int i = 1; i <= n; i = i + 1) {
        sum = sum + term;
        term = term * x;
    }
    return sum;
}//powerSum
```

Uppgift 79.

Skillnaden mellan en klass och ett objekt är den att klasser fungerar som en ritning/mall utifrån vilken objekt skapas när ett program exekveras. Alla objekt som skapas från en viss klass får samma egenskaper och beteenden. En klass är således en beskrivning av en mängd objekt med gemensamma egenskaper och beteenden. Klassen specificerar den information/data som kan lagras i ett objekt samt de metoder som ett objekt kan utföra.

Ett objekt är en unik och konkret realisering av en klass. Alla objekt har ett tillstånd, ett beteende och en identitet.

Uppgift 80.

Det är ingen skillnad, utan två olika sätt att uttrycka samma sak.

Uppgift 81.

En konstruktor är den metod som anropas när man skapar ett objekt av en klass. Anropet sker med användning av operatorn **new** enligt:

KlassNamn nyttObjekt = **new** KlassNamn(parameterlista);

En konstruktor deklareras inuti en klass och den har samma namn som klassen själv. Det är möjligt att överlagra konstruktorer, dvs en klass kan ha flera konstruktorer, var och en med sin unika parameterprofil.

Uppgift 82.

Nej! Deklareras ingen konstruktor i en klass har man ändock tillgång till den parameterlösa konstruktorn. Men om man deklarerar en eller flera egna konstruktorer måste man också själv deklarera den parameterlösa konstruktorn.

Uppgift 83.

Attribut är en gemensam beteckning för de variabler (instansvariabler och klassvariabler) och konstanter som finns i en klass.

Uppgift 84.

Inkapsling är en grundprincip i objektorienterad programmering som innebär att objektets inre uppbyggnad är dold för den som använder klassen. Allt programmeraren behöver veta är vad objektet kan göra och hur man ber objektet att göra det. Inkapsling innebär att man skiljer mellan objektets implementation och dess gränssnitt (specifikation). Ett objekt kan endast påverkas via de metoder som finns specificerade för objektet. Inkapsling innebär också att objektet kan isoleras mot fel i andra objekt.

Uppgift 85.

a) En enkel variabel lagrar data som tillhör någon av de enkla datatyperna, t.ex **int**, **double**, **boolean** och **char**.

b) En referensvariabel är en variabel som refererar till ett objekt. En referensvariabel av en viss klass kan endast kan referera till objekt av denna klass.

c) En lokal variabel är en variabel som deklareras inuti ett block eller inuti en metod . En lokal variabel har begränsad livslängd, den skapas när man börjar exekvera blocket och den är inte längre åtkomlig när man lämnat blocket.

d) En instansvariabel är en variabel som håller information om ett objekts tillstånd. En instansvariabel är specifik för varje instans av en klass.

e) En klassvariabel är en variabel som är gemensam för alla objekt av en viss klass. En klassvariabel är en resurs som delas av alla objekt av en klass.

Uppgift 86.

Orsaken är att variablerna value1 och value2 är deklarerad som en instansvariabler men vi har inget objekt av klassen Strange! Det finns tre lösningar på detta problem. Den första, och den som absolut är att föredra, är att göra variablerna lokala i main():

```
public class Strange {
    public static void main(String[] arg) {
        int value1 = 2, value2 4;
        System.out.println((2 * value1 + 5 * value2 - 4));
    }//main
}//Strange
```

Ett annat sätt som är möjlig, men just i detta exempel ganska onaturlig)t, är att skapa ett objekt av klassen Strange:

```
public class Strange {
    public int value1 = 2, value2 = 4;
    public static void main(String[] arg) {
        Strange objekt = new Strange();

        System.out.println((2 * objekt.value1 + 5 * objekt.value2 - 4));
    }//main
}//Strange
```

Ett tredje sätt som är möjlig, men oerhört ”fult”, är att skapa göra variablerna till klassvariabler:

```
public class Strange {
    static int value1 = 2, value2 = 4;
    public static void main(String[] arg) {
        System.out.println((2 * value1 + 5 * value2 - 4));
    }//main
}//Strange
```

Uppgift 87.

En statisk metod eller klassmetod är en metod som kan anropas utan att använda en referens till en instans av dess klass. Eftersom ett sådant metodanrop inte är kopplat till något objekt så kan klassmetoder inte referera till klassens instansvariabler.

Uppgift 88.

När en metod anropas sker en bindning mellan de aktuella och formella parametrarna. I Java sker bindningen via ett så kallat värdeanrop, vilket innebär att den formella parametern tilldelas värdet av motsvarande aktuell parameter. Således gäller vid anrop till en metod att de de aktuella parametrarna måste överensstämma med de formella parametrarna i antal, typ och ordning. Efter anropet har den aktuella parametern samma värde som innan anropet.

Uppgift 89.

En synlighetsmodifierare kan anges för en klass samt för attributen och metoderna som finns i en klass. Synlighetsmodifieraren specificerar vem som ser (= har access) till entiteten i fråga. Följande tre synlighetsmodifierare finns

public - anger att entiteten är synlig för alla klasser

protected - anger att entiteten är synlig i för klasser inom samma paket och för subklasser

private - anger att entiteten är synlig endast inom klassen själv

utelämnad – anger att entiteten är synlig i för klasser inom samma paket

Uppgift 90.

Överlagring innebär att det inom en och samma klass finns flera metoder (eller konstruktorer) med samma namn, men med olika parameterprofiler.

Uppgift 91.

Det finns två konstruktorer med samma parameterprofiler:

```
public ClassA(int x) { . . .}
public ClassA(int y) { . . .}
```

Uppgift 92.

Utskriften blir:

```
0
3
```

Uppgift 93.

Utskriften blir:

```
The name is: First
The name is: Second
Second Second
```

Uppgift 94.

Följande satser är felaktiga:

obj.b = 10;	instansvariabeln b är deklarerad **private**
obj.m1(7.8);	metoden m1 skall ha en parameter av typen **int**
double x = obj.m2();	metoden m2 är deklarerad som **void**
System.out.println(obj.m2(12.8));	metoden m3 är deklarerad som **private**

Uppgift 95.

Utskriften blir:

```
x = 20
y = 30
z = 20
```

Uppgift 96.

Orsaken är att metoden compute är deklarerad som en *instansmetod* men vi har inget objekt av klassen Uppgift1A. Det finns två lösningar på detta problem. Den första, och den som är att föredra, är att göra metoden compute till en *klassmetod* genom att ändra metodhuvudet till

```
public static int compute(int a, int b)
```

Den andra metoden som är möjlig, men onaturlig, är att skapa ett objekt av klassen Uppgift1A och anropa metoden compute för detta objekt. Metoden main skulle då få följande utseende:

```
public static void main(String[] arg) {
    Uppgift1A obj = new Uppgift1A();
    int tal1 = 2, tal2 = 4;
    int res = compute(tal1, tal2);
}
```

Lösningsförslag: Instuderingsfrågor, del 5

Uppgift 97.

a) Inget fält behövs. Man kan läsa in ett tal i taget och addera dessa till summan.

b) Här behövs ett fält.

c) Här behövs inget fält. Talen kan läsas ett i taget och hela tiden sparar man undan det hittills största talet.

d) Här behövs ett fält.

e) Här behövs inget fält. För att beräkna medelvärdet räcker att beräkna summan av talen.

f) Här behövs ett fält.

g) Här behövs ett fält

h) Här behövs ett fält.

Uppgift 98.

Fältet rymmer 4 element. Elementen i fältet är av datatypen **int**. Fältet är indexerat från 0 till 3.

Uppgift 99.

a) **char** vokaler = **new** char[5];	**char** i högerledet och **char**[5] i vänsterledet
b) **double**[] m = **double**[10];	**new** saknas i högerledet
c) **int**[]odd = **new int**{1, 3, 5, 7 , 9};	{ och } istället för [och] i vänsterledet

Uppgift 100.

a) **int**[] talen = {3, 19, 7, 9, 4};

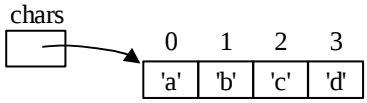
b) **int**[] talen = {'0' , '1' , '2' , '3' , '4' , '5', '6' , '7' , '8' , '9' };

c) Point[] punkter = **new** Point[25];

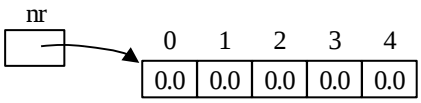
Uppgift 101.

a)

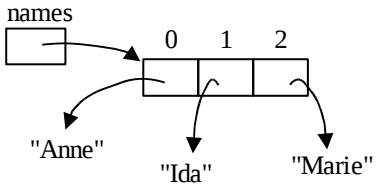
b)



c)



d)



Uppgift 102.

Utskrifterna blir:

a) Olika

b) Lika

Orsaken till att utskriften i deluppgift a) blir "Olika" beror på att ett fält är ett objekt och att därför är en fältvariabel är en referensvariabel. Således är det referenserna fälten som jämförs och inte innehållet i de båda fälten. För att jämföra innehållet i fälten kan, som i deluppgift b), klassmetoden equals i klassen Arrays användas.

Uppgift 103.

a) 25 b) 13 c) 12 d) 6 e) 1

Uppgift 104.

a) [1, 3, 5, 7, 9, 9, 9, 9, 9]

b) [2, 3, 4, 5, 4, 3, 2, 1, 1]

c) [1, 1, 1, 1, 1, 1, 1, 1, 1]

d) [1, 1, 1, 1, 1, -1, -1, -1, -1]

e) [5, 4, 3, 2, 5, 4, 3, 2, 1]

Uppgift 105.

a) 3

b) Vi får ett exekveringsfel! Detta beroende på att talen i serien är sorterade i växande ordning. Således kommer **while**-satsen att löpa i genom samtliga element i fältet. När de två siste elementen jämförs, dvs elementen med värdena 49 respektive 53, evalueras villkoret i **while**-satsen till **true** och **while**-satsen genomlöps ytterligare en gång. Nu inträffar emellertid följande: variabeln i har värdet 5 och villkoret i **while**-satsen innebär alltså att fältelementen vekt[5] och vekt[6] (skall jämföras. Men det finns inget fältelement vekt[6]!!!

Uppgift 106.

[1, -1, -2, -1, 1, 2, 1, -1, -2]

Uppgift 107.

a) **for**-satsen kommer att genomlöpas då k har värdet 10, men fältet values har ingen element med index 10. Ett exekveringsfel kommer att inträffa. **for**-satsen bör ha följande utseende:

```
for (int k = 0; k < values.length; k = k + 1)
```

b) Det finns inget fält att genomlöpa. Deklarationen skapar en variabel values som kan referera till ett heltalsfält, men inget fält har skapas varför values refererar till **null**.

c) **for**-satsen kommer att genomlöpas då k har värdet 7. Inuti **for**-satsen finns en refereras till elementet values[k + 1], men fältet values har ingen element med index 8. Ett exekveringsfel kommer att inträffa. **for**-satsen bör ha följande utseende:

```
for (int k = 0; k < values.length - 1; k = k + 1)
```

Uppgift 108.

Utskriften blir:

7

Uppgift 109.

[1, 1, 2, 3, 5, 8, 13]

Uppgift 110.

[1, 5, 6, 9]

Uppgift 111.

Utskriften blir:

```
x= 5
v[0] = 3
v[1] = 2
v[2] = 3
```

Förklaring:

Värdet av de aktuella parametrarna kopieras in i de formella parametrarna, vilket betyder att en aktuell parameter som är en enkel datatyp kommer att ha samma värde efter metodanropet som innan, dvs x kommer att ha kvar värdet 5. Är den aktuella parametern ett objekt innebär det att den aktuella parametern och den formella parametern refererar till samma objekt, och att de förändringar som utförs i metoden på objektet som refereras av den formella parametern också utförs på objektet som refereras av den aktuella parametern (är ju ett och samma objekt).

Uppgift 112.

```
111
2
666
```

Uppgift 113.

```
for (int i = tal.length - 1; i >= 0; i = i - 1)
    System.out.println(tal[i]);
```

Uppgift 114.

Enklast är nog att använda sig av två **for**-satser enligt följande:

```
for (int i = 0; i <= vekt.length; i = i + 2)
    vekt[i] = 1;
for (int i = 1; i <= vekt.length; i = i + 2)
    vekt[i] = 0;
```

Alternativt kan man använda en **for**-sats och en **if**-sats enligt nedan:

```
for (int i = 0; i <= vekt.length; i = i + 1) {
    if (i%2 == 0)
        vekt[i] = 1;
    else
        vekt[i] = 0;
}
```

Uppgift 115.

Orsaken till felet är att vi endast har deklarerat fältet triangle, men inte skapat fältet. När vi refererar till fältelementet triangle[0] fås därför ett fel, eftersom fältet inte ännu finns.

```
Point[] triangle = new Point[3];
triangle[0] = new Point(3.0, 0.0);
triangle[1] = new Point(0.0, 4.0);
triangle[2] = new Point(0.0, 0.0);
```

Uppgift 116.

```
public static int countOdd(int[] vekt) {
    int nrOfOdd = 0;
    for (int i = 0; i < vekt.length; i = i + 1) {
        if (vekt[i] % 2 == 1) {
            nrOfOdd = nrOfOdd + 1;
        }
    }
    return nrOfOdd;
}// countOdd
```

Uppgift 117.

```
public static int computeOddSum(int[] vekt) {
    int sumOfOdd = 0;
    for (int i = 0; i < vekt.length; i = i + 1) {
        if (vekt[i] % 2 == 1) {
            sumOfOdd = sumOfOdd + vekt[i];
        }
    }
    return sumOfOdd;
}//computeOddSum
```

Uppgift 118.

```
public static int countCloseTo(double[] values, double target, double tolerance) {
    int count = 0;
    for (int i = 0; i < values.length; i = i + 1) {
        if ( Math.abs(target - values[i]) < tolerance) {
            count = count + 1;
        }
    }
    return count;
}// countCloseTo
```

Uppgift 119.

```
public static double[] doubleValues(double[] original) {
    double[] doubled = new double[original.length];
    for (int i = 0; i < original.length; i = i + 1) {
        doubled[i] = original[i] * 2;
    }
     return doubled;
}// doubleValues
```

Uppgift 120.

```
public static boolean[] reverse(boolean[] original) {
    boolean[] reversed = new boolean[original.length];
     for (int i=0 ; i < reversed.length; i = i + 1) {
        reversed[i] = original[original.length - i - 1];
    }
     return reversed;
}// reverse
```

Uppgift 121.

```
public static double[] removeFirst(double[] original) {
    double[] shorter = new double[original.length - 1];
    for (int i = 1; i < original.length; i = i + 1) {
        shorter[i - 1] = original[i];
    }
    return shorter;
}// removeFirst
```

Uppgift 122.

```
public static int findFirstTarget(int[] values, int target) {
    for (int i=0; i < array.length; i = i + 1) {
        if (array[i] == target) {
            return i;
        }
    }
    return -1;
}// findFirstTarget
```

Uppgift 123.

```
public static int[] removeAllTargets(int[] original, int target) {
    int location = findFirstTarget(original, target);
    while (location >= 0 ) {
        original = removeAt(original, location);
        location = findFirstTarget(original, target);
    }
    return original;
}//removeAllTargets
```

Uppgift 124.

a) 27	b) h	c) 2
d) 10	e) three	f) THE THREE DID FEED THE DEER
g) -1	h) 25	i) Tha thraa did faad tha daar

Uppgift 125.

a) ett tal större än 0	b) ett tal mindre än 0	c) 0

Uppgift 126.

a) **false**	b) **false**	c) **true**
d) **true**	e) **false**	f) **true**

Uppgift 127.

Utskriften blir :

snusfabrik
fabrikör

Uppgift 128.

Utskriften blir:

optr

Uppgift 129.

Utskriften blir:

utercom

Uppgift 130.

Lämpligast är att införa en metod:

```
String str = "Detta ÄR en STRÄNG med BÅDE små OcH STORA bOkStÄvEr!";
str = exchangeCapitalAndLower(str);
```

Metoden får följande utseende:

```
public static String exchangeCapitalAndLower(String str) {
    String strCopy = "";
    for (int i = 0; i < str.length(); i = i + 1) {
        if ( Character.isLowerCase(str.charAt(i)))
            strCopy = strCopy + Character.toUpperCase(str.charAt(i));
        else if ( Character.isUpperCase(str.charAt(i)))
            strCopy = strCopy + Character.toLowerCase(str.charAt(i));
        else
            strCopy = strCopy + str.charAt(i);
    }
    return strCopy;
}// exchangeCapitalAndLower
```

Uppgift 131.

```
public static String fixEmail(String oldEmail) {
    int  k = s.indexOf('_');
    if  (k < 0)
        k = s.indexOf('.');
    int  at= s.indexOf('@');
    return  s.substring(k+1,at) + "." + s.charAt(0) + "@" + s.substring(at+1,s.length()-3) + "net";
}//fixEmail
```

Uppgift 132.

Utskriften blir:

```
world  6
world  6
```

Uppgift 133.

```
public static String middle(String str) {
    if (str.length() % 2 != 0)
        return str.substring(str.length() / 2, str.length() / 2 + 1);
    else
        return str.substring(str.length() / 2 - 1, str.length() / 2 + 1);
}//middle
```

Lösningsförslag: Instuderingsfrågor, del 6

Uppgift 134.

a) **double**[][] matris = **new double**(5)(4); (och) istället för [och] i vänsterledet

b) **double**[][] temp= **new double**[3][9]; korrekt

c) **int**[][] speed = {{1, 3, 5}, {7 , 9}}; korrekt

d) **char**[][] letters = **new** letters[2][]; korrekt

Uppgift 135.

a) 'b' b) 'd' c) ger ett exekveringsfel

e) {'a', 'b', 'c', 'd'}

Uppgift 136.

Utskriften blir:

4
3
5
2

Uppgift 137.

Fältet mat har följande utseende:

2 1 1 1
3 2 1 1
3 3 2 1

Uppgift 138.

Utskriften blir

33

Uppgift 139.

Utskriften blir

1 3 4 5
1 2 6 33

Uppgift 140.

[1, 4, 7]
[2, 5, 8]
[3, 6, 9]

Uppgift 141.

Utskriften blir

5 33

Uppgift 142.

Utskriften blir

4

Uppgift 143.

Utskriften blir

5

Metoden calculate returnerar längden på den längsta raden i fältet m.

Uppgift 144.

Utskriften blir

1

Metoden calculate returnerar index på den längsta raden i fältet m.

Uppgift 145.

Utskriften blir

[1, 2, 3, 4, 5]

Metoden calculate returnerar en kopia av den längsta raden i fältet m.

Uppgift 146.

```
public static boolean exist(int[][] m) {
    if (m == null || m.length == 0)
        return false;
    else
        return true;
} //exist
```

Uppgift 147.

```
public static int largestSumOfRow(int[][] m) {
    int largestSoFar = Integer.MIN_VALUE;
    for (int row = 0; row < m.length; row = row + 1) {
        int sum = 0;
        for (int col = 0; col < m[row].length; col = col + 1) {
            sum = sum + m[row][col];
        }
        if (sum > largestSoFar) {
            largestSoFar = sum;
        }
    }
    return largestSoFar;
}//  largestSumOfRow
```

Uppgift 148.

```
public static int indexOfLargestRow(int[][] m) {
    int largestSoFar = 0;
    int indexOfLargest = 0;
    for (int row = 0; row < m.length; row = row + 1) {
        int sum = 0;
        for (int col = 0; col < m[row].length; col = col + 1) {
            sum = sum + m[row][col];
        }
        if (sum > largestSoFar) {
            largestSoFar = sum;
            indexOfLargest = row;
        }
    }
    return indexOfLargest;
}//indexOfLargestRow
```

Uppgift 149.

```
public static int[] getLargestRowt(int[][] m) {
    int largestSoFar = 0;
    int indexOfLargest = 0;
    for (int row = 0; row < m.length; row = row + 1) {
        int sum = 0;
        for (int col = 0; col < m[row].length; col = col + 1) {
            sum = sum + m[row][col];
        }
        if (sum > largestSoFar) {
            largestSoFar = sum;
            indexOfLargest = row;
        }
    }
    return java.util.Arrays.copyOf(m[indexOfLargest], m[indexOfLargest].length);
}//getLargestRow
```

Uppgift 150.

```
public double[][] identityMatrix(int size) {
    double[][] dArray = new double[size][size]; // allocate the two-dimensional array at once
    for (int row = 0; row < size; row = row + 1) {
        for (int column = 0; column < size; column = column + 1) {
            dArray[row][column] = 0;
        }
        dArray[row][row] = 1.0;
    }
    return dArray;
}// identityMatrix
```

Uppgift 151.

```
public static int[][] transponat(int[][] mat) {
    int[][] tran = new int[mat.length][mat.length];
    for (int i = 0; i < mat.length; i = i + 1) {
        for (int j = 0; j < mat.length; j = j + 1) {
            tran[i][j] =  mat[j][i];
        }
    }
    return tran;

}//transponat
```

Uppgift 152.

a) Det går inte att lagra primitiva datatyper i en ArrayList. Objekt av tillhörande omslagsklass måste lagras. Deklarationen skall alltså vara:

```
ArrayList<Integer> values = new ArrayList<Integer>();
```

b) Vilken typ av värden som skall lagras i listan måste anges vid anropet av konstruktorn. Deklarationen skall alltså vara:

```
ArrayList<Double> values = new ArrayList<Double>();
```

c) Parameterlistan måste anges vid anrop till konstruktorn. Deklarationen skall alltså vara:

```
ArrayList<String> values = new ArrayList<String>();
```

d) Ingen insatns av ArrayList har skapats, varför referensvariabeln values har värdet **null**. Innan for-satsen måste listan skapas med satsen

```
values = new ArrayList<Integer>();
```

e) values refererar till en ArrayList som innehåller 0 element varför det inte går att ändra värdet på de 10 första elementen i listan, eftersom dessa inte finns. Troligen skall elementen inte ändras utan läggas in i listan, vilket innebär att metoden det är metoden add som skall anropas och inte metoden set. Detta går bra eftersom elementen position 0, 1, 2, .., 9 läggs in i tur och ordning.

Uppgift 153.

Utskriften blir:

[4, 7, 12, 19, 28, 39]

Uppgift 154.

Utskriften blir:

A
B
C
D

Uppgift 155.

Utskriften blir:

[1, 2, 5, 1, 2, 5]

Uppgift 156.

Utskriften blir:

[green, red, blue]

Uppgift 157.

Utskriften blir:

[green, black]

Uppgift 158.

Utskriften blir:

[yellow, green, yellow, black]

Uppgift 159.

```
public static ArrayList<Integer> append(ArrayList<Integer> a, ArrayList<Integer> b) {
    ArrayList<Integer> res= new ArrayList<Integer>();
    res.addAll(a);
    res.addAll(b);
    return res;
} //append
```

Uppgift 160.

```
public static int nrOfTargets(ArrayList<Integer> list, Integer target) {
    int nrOf = 0;
    for (int i = 0 ; i < list.size(); i = i + 1) {
        if (target.equals(list.get(i)))
            nrOf = nrOf + 1;
    }
    return nrOf;
}//nrOfTargets
```

Uppgift 161.

```
public static ArrayList<Integer> reverse(ArrayList<Integer> original) {
    ArrayList<Integer> reversed = new ArrayList<Integer>();
    for (int i = original.size() - 1 ; i >= 0; i = i - 1) {
        reversed.add(original.get(i));
    }
     return reversed;
}// reverse
```

Lösningsförslag: Instuderingsfrågor, del 7

Uppgift 162.

Med arv menas att man vid definitionen av en klass kan utgå ifrån en annan redan existerande klass och ärva de egenskaper som denna existerande klass har. Den nya klassen kan sedan utöka eller omdefiniera de egenskaper den ärvt, men aldrig ta bort ärvda egenskaper. En klass som ärver kallas för *subklass* till den klass den ärver ifrån och den klass en subklass ärver ifrån kallas *superklass*.

Fördelen med arv är att det blir enkelt att återanvända klasser och det därför krävs små arbetsinsatser för att skapa nya specialiserade klasser.

Uppgift 163.

En subklass kan vara superklass till andra klasser, vilket innebär att man kan skapa flera "generationer" av arv. En arvshierarki är ett "släktträd" som visar vilka superklasser och subklasser en viss klass har.

Uppgift 164.

Att en subklass överskuggar en metod i superklassen innebär att subklassen omdefinierar denna metod, dvs i subklassen implementeras en ny version av metoden.

Uppgift 165.

Med operatorn **instanceof** kan man avgöra om ett objekt är av en viss klass.

Uppgift 166.

Klassen Object är superklass till alla andra klasser.

Uppgift 167.

Ett grafiskt användargränssnitt är den miljö som utgör mellanskiktet mellan själva programmet och använd- aren, dvs att det är via det grafiska användargränssnitt som användaren kommunicerar med programmet. I ett grafiskt användargränssnitt sker kommunikationen genom att flytta datormusen (eller något annat pekdon) och t.ex. klicka på knappar och menyer, fylla i frågeformulär eller dra ikoner mellan olika fönster.

Uppgift 168.

Program med grafiska användargränssnitt är händelsestyrda, dvs exekveringen styrs av användaren genom de olika aktiviteter som denne gör i det grafiska gränssnittet (t.ex. trycker på knappar, drag musen eller drar i sliders). I ett program med textbaserad in- och utmatning är det programmet som styr i vilken ordning de olika programsegmenten skall exekveras och det är programmet som avgör när indata användaren skall ge indata och vilka typer av indatasekevenser denne skall ge.

Uppgift 169.

I Java finns två grafiska standardklasser - awt och Swing.

Uppgift 170.

En behållare är ett grafiskt objekt som kan innehålla andra grafiska objekt.

Uppgift 171.

Ett fönster är en grafisk enhet som kan visas på skärmen. Fönstret utgör programmets gränssnitt mot användaren och innehåller således de komponenter som bygger upp det grafiska gränssnittet.

Vanligtvis använder man klassen JFrame för att skapa fönster (Swing innehåller dock även klasserna JWindow och JDialog för att skapa fönster).

Uppgift 172.

Koordinatsystemet i Java har origo i övre vänstra hörnet, x-axeln har ökande koordinater åt höger i fönstret och y-axeln har ökande koordinater nedåt i fönstret.

Uppgift 173.

Ett JPanel-objekt är ett delfönster som används för att gruppera samman olika grafiska komponenter. För att kunna visas måste ett JPanel-objekt placeras i ett fönster (vanligtvis i ett JFrame-objekt):

Uppgift 174.

a) myWindow.setBackground(Color.green)

b) myWindow.setVisible(**true**)

c) myWindow.setSize(200, 400)

d) myWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE))

Uppgift 175.

Ett JLabel-objekt kan ses som en etikett eller skylt som används för att visa information (texter och/eller bilder). Ett JButton-objekt kan förvisso visa information, men dess huvudsakliga syfte är att vara en inmatningsenhet med vilken användaren (genom att trycka på) kan kommunicera med programmet.

Uppgift 176.

a) etikett.getText()

b) etikett.setHorizontalAlignment(JLabel.CENTER)

c) etikett.setBackground(Collor.yellow)

d) etikett.setOpaque(**true**)

Uppgift 177.

a) knapp.setText("Tryck")

b) knapp.setForeground(Color.blue)

c) knapp.isEnabled()

d) knapp.setEnabled(**false**)

Uppgift 178.

När man trycker på ett JButton-objekt genereras en händelse (= skapas ett objekt) av typen ActionEvent.

Uppgift 179.

För att fånga en händelse av typen ActionEvent måste det finnas en lyssnare (= ett objekt) av klassen ActionListener, dvs man måste implementera interfacet ActionListener.

Ett objekt som genererar händelser av typen ActionEvent måste registrera sig hos en lyssnare av typen ActionListener, vilket görs med metoden addActionListener.

När en händelse av typen ActionEvent inträffar fångas händelsen upp av metoden actionPerformed hos den lyssnare där objekt som genererade händelsen finns registrerad.

Uppgift 180.

a) Instansmetod getSource() returnerar en referens till det objekt som genererade händelsen.

b) Instansmetod getActionCommand() returnerar texten som finns på objekt som genererade händelsen.

Uppgift 181.

De klasser som handhar händelsehantering finns i paketet java.awt.event.

Uppgift 182.

Ett JTextField-objekt kan endast visa en rad, medan ett JTextArea-objekt kan visa många rader.

Uppgift 183.

En layout manager är ett objekt som placerar ut komponenterna i en behållare enligt en viss given strategi.

Uppgift 184.

Ett FlowLayout-objekt lägger ut komponenterna på en rad från vänster till höger. Om behållaren inte rymmer alla komponenter på en rad sker utplaceringen på nästa rad.

Uppgift 185.

Ett GridLayout-objekt lägger ut komponenterna radvis från vänster till höger i en tabell med r rader och k kolumner. Antalet rader och kolumner specificeras när objekt skapas.

Uppgift 186.

Ett BorderLayout-objekt lägger ut komponenterna i en av fem specificerade ytor - north, south, west, east och center.

Uppgift 187.

Paketet java.awt innehåller layout manager klasserna.

Uppgift 188.

JFrame har BorderLayout som default.

Uppgift 189.

JPanel har FlowLayout som default.

Uppgift 190.

Färger beskrivs av klassen Color. En färg (= ett objekt av klassen Color) innehåller tre komponenter, som är heltal mellan 0 och 255. Dessa tre heltal anger "mängden" av grundfärgerna rött, grönt och blått, vilket kallas RGB-talet, för den aktuella färgen. Ett färg med "mängderna" 150 röd färg, 200 grön färg och 250 blå färg skapas enligt

```
Color niceBlue = new Color(150, 200, 250);
```

Uppgift 191.

Typsnitt handhas av klassen Font.

Uppgift 192.

Utritning av komponenter som tillhör subklasser till klassen JComponent sker med metoden paintComponent. Att förändra utseendet hos en sådan komponent, innebär således att vi skapar en klass som ärver egenskaperna från den aktuella JComponent-klassen och omdefinierar metoden paintComponent.

Uppgift 193.

För att rita figurer och skriva texter i en grafisk komponent används ett objekt av klassen Graphics. Klassen Graphics har instansmetoder för att t.ex. rita linjer, rita ovaler, rita rektanglar, skriva texter och ange vilken färg en utritad figur eller text skall ha.

Uppgift 194.

För att rita om en komponent från ett program används metoden repaint().