Primitiva datatyper och selektering

Föreläsning 2

TDA540 - Objektorienterad Programmering

CHALMERS

Extra föreläsning imorgon!

HC4 15:15 - 17:00

Sammanfattning föreläsning 1

- Vad är en dator?
- Vad är ett program?
- Kompilera
- Data vs. information
- Algoritmer
- Första Java program

Första laborationstillfälle

- Svenska bokstavar: använd UTF-8
- Försvinnande fönster:
    - hamnar bakom IntelliJs huvudfönster
    - använd ALT-TAB för att växla fönster
- `null`: antyda att något finns inte
- Lite trångt i labsalar
- Kolla hemsidans senaste version (googla inte)

Datatyper

Det finns tre viktiga anledningar till att använda sig av ett typsystem:

1. typer hjälper oss att förstå och organisera våra tankar rörande objekt
2. ett typsystem hjälper oss att se och resonera om unika egenskaper hos specifika typer
3. typsystemet hjälper oss att upptäcka fel, t ex felaktig användning av operationer.

Omslagsklassen Double

Omslagsklassen Double innehåller bl.a följande konstanter och metoder:

static final int MAX_VALUE	det största värdet som kan lagras i en **double**
static final int MIN_VALUE	det minsta värdet som kan lagras i en **double**
static String toString(**double** n)	ger det reella talet n som en sträng
static double parseDouble(String str)	ger strängen str som en **double**

Exempel:

Anropet Double.parseDouble("12.34") returnerar det reella talet 12.34

Anropet Double.toString(56.78) returnerar strängen "56.78"

Satserna

```
System.out.println("Största reella talet: " + Double.MAX_VALUE);
System.out.println("Minsta reella talet: " + Double.MIN_VALUE);
```

ger utskriften:

Största reella talet: 1.7976931348623157E308

Minsta reella talet: 4.9E-324

Problemexempel: In- och utmatning av reella tal

Problem: Skriv ett program som läser två reella och skriver ut summan av talen.

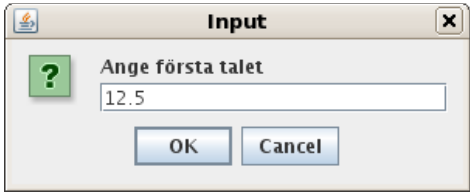
Analys:

Indata: De två reella talen som skall adderas.

Utdata: Summan av de inlästa talen.

Exempel på körning:

Känner du igen problemet?



Problemexempel: In- och utmatning av reella tal

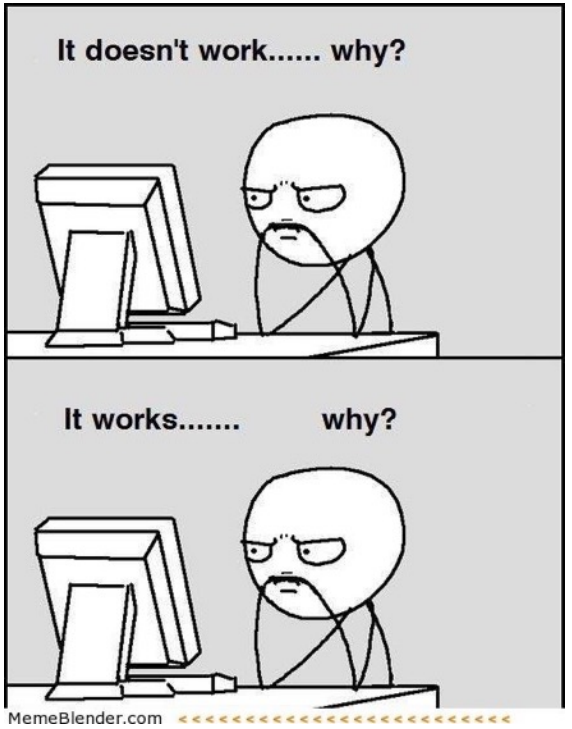
Design:

Algoritm:

1. Skriv texten *”Ange första talet: ”*.
2. Läs *tal1*.
3. Skriv texten *”Ange andra talet: ”*.
4. Läs *tal2*.
5. Addera *tal1* och *tal2*, samt lagra resultatet i *summa*.
6. Skriv texten *”Summan av talen är: ”*.
7. Skriv ut *summa*.

Datarepresentation:

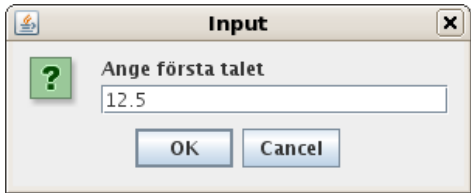
tal1, *tal2* och *summa* är av datatypen **double**.

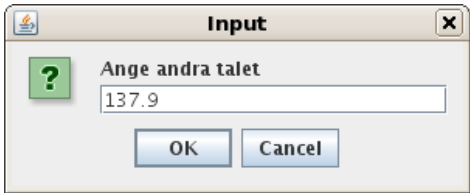


Problemexempel: In- och utmatning av reella tal

Implementation:

```
/* Programmet läser in och adderar två reella samt skriver ut resultatet. */
import javax.swing.*;
public class AddTwoRealNumbers {
   public static void main (String[] arg) {
      String input = JOptionPane.showInputDialog("Ange första talet");
      double number1 = Double.parseDouble(input);
      input = JOptionPane.showInputDialog("Ange andra talet");
      double number2 = Double.parseDouble(input);
      double sum = number1 + number2;
      JOptionPane.showMessageDialog(null, "Summan av talen är " + sum);
   } //main
} //AddTwoRealNumbers
```





Regler för namngivning i Java

Alla storheter, t.ex. klasser, variabler och konstanter, som används i ett program måste ges unika *identifierarnamn*.

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.
5. Java skiljer på små och stora bokstäver (*case senstive*). Exempelvis är hej, Hej och HEJ tre olika identifierare.
6. *Namnkonvention* i Java är att identifierare på klasser inleds med stor bokstav, identifierare på konstanter enbart innehåller stora bokstäver, medan identifierare på andra entiteter i programmet inleds med liten bokstav.

Regler för namngivning i Java

Exempel:

Hello, String, JOptionPane namn på klasser

PI, MAXIMUM, SIZE namn på konstanter

name, greeting, println namn på entiteter som ej är klasser eller konstanter, t.ex metoder och variabler

Välj alltid namnet på en identifierare på så sätt att namnet anger vad identifierarna används till!!

Undvik att använda de svenska bokstäverna å, ä och ö.

Använd helst engelska namn.

Reserverade ord

abstract	boolean	break	byte	byvalue*
case	cast*	catch	char	class
const*	continue	default	do	double
else	extends	false	final	finally
float	for	future*	generic*	goto*
if	implements	import	inner*	instanceof
int	interface	long	native	new
null	operator*	outer*	package	private
protected	public	rest*	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	transient	true
tryvar*	void	volatile	while	

* används ej ännu, men är reserverade för framtiden

Mer om variabler

För att handha objekt som avbildas i ett Javaprogram används *variabler* i vilka man kan lagra data.

I Java finns olika slag av variabler och de variabler som används för att lagra primitiva datatyper kallas *enkla variabler*.

En variabel har ett *namn*, är av en viss *typ* och har ett *värde*.

En variabel kan ha olika värden under sin livstid.

En variabel kan ses som en *namngiven behållare* i vilken man kan lagra ett värde av en viss typ.

Alla variabler som används i ett program *måste deklareras*. Vid deklarationen anges variabels *namn* och *typ*.

Sitt värde får en enkel variabel via en tilldelningssats.

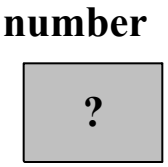
En variabel kan tilldelas ett initialt värde direkt i deklarationssatsen.

Variabler

Exempel:

Deklarationen
int number;

innebär att variabeln antal skapas och
dess värde blir odefinierat:



Tilldelningssatserna
number = 10;
double price = 24.75;

ger respektive variabel ett specifikt värde:

Deklaration av variabler

type namn;
type name1, name2, name3;
type namn = value;

Exempel:

```
int number;
int value1, value2, value3;
double radius = 10.5;
String text = "Texten är detta";
```

Deklaration av konstanter

final type NAME = value;

Exempel:

```
final int MAXSIZE = 100;
final double PI = 3.1415;
final String PHRASE = "Välkommen";
```

Obs: Praxis är att använda endast STORA bokstäver för att namnge konstanter

Konstanter kan t.ex. användas för att undvika *magiska tal*, d.vs. tal som dyker upp i koden utan någon förklarning.

Exempel:

I satsen

```
totalPrice = price * 1.25;
```

finns ingen förklarning till talet 1.25. Används en konstant blir koden mer informativ:

```
final double TAX = 0.25;
totalPrice = price * (1 + TAX);
```

Indentering

För att göra källkoden lättare att läsa och förstå för en mänsklig betraktare skall man *indentera* programmet. *Indentering* innebär att man skjuter in programsatserna med extra mellanslag på sådant sätt att de satser som hör ihop hamnar på samma avstånd från vänstermarginalen:

```
import javax.swing.*;
public class Hello2 {
   /* Detta program ger en hälsning från datorn */
   public static void main (String[] args) {
      JOptionPane.showMessageDialog(null,"Hello world! \n" +
                                    "This is a message from the computer.");
   } // main
} // Hello2
```

Indentering

Kompilatorn bryr sig inte om hur källkoden skrivs. Därför skulle man egentligen inte behöva indentera sina program, men koden blir då mer eller mindre oläsligt:

```
import javax.swing.*; public class Hello2 {/* Detta program ger
en hälsning från datorn */ public static void main (String[] args
{ JOptionPane.showMessageDialog(null,
"Hello world! \n" +       "This is a message from the computer."); } //
main
} // Hello2
```

Indentera alltid era program!

Kommentarer i programmet

För att öka förståelsen av ett program är det möjligt att ge *kommentarer* i programmet. Kommentarerna ignoreras av kompilatorn (är avsedda endast för den mänskliga läsare).

Java stödjer tre olika typer av kommentarer:

• för att markera att *resten av en rad* utgörs av en kommentar

```
//Denna rad är en kommentar
System.exit(0);   //avslutar exekveringen
```

• för att markera att en kommentar sträcker sig över *flera rader*

```
/* Detta är en kommentar som sträcker sig över
ett flertal rader!! */
```

• för att kunna generera dokumentation med javadoc

```
/** Detta är en dokumentationskommentar
 *@author Janne Java
 *@version 1.0
*/
```

Med javadoc fås dokumentationen i form av HTML-filer med samma format som dokumentationen av Javas API.

Klassen Math – matematiska standardfunktioner

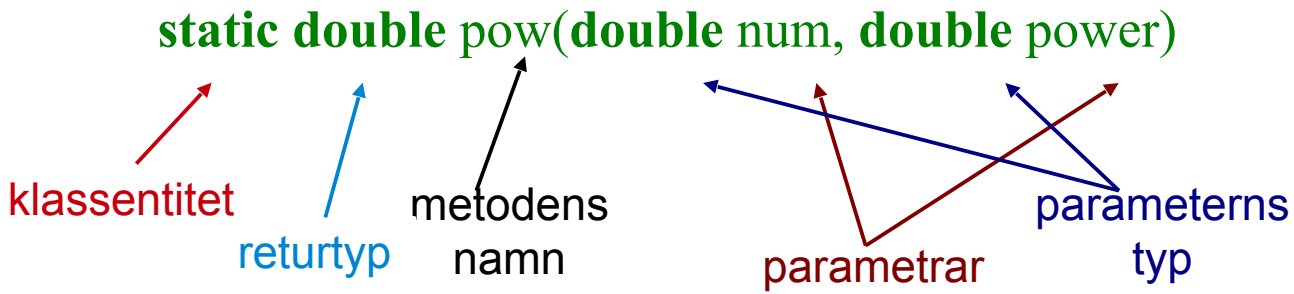
I tekniska tillämpningar är det vanligt att man behöver göra numeriska beräkningar. Java tillhandahåller standardklassen java.lang.Math, som innehåller en del vanliga matematiska beräkningsfunktioner.

I Math finns bland annat följande:

Konstant	Beskrivning
static double PI	konstanten π
static double E	konstanten e (basen för den naturliga logaritmen)
Metod	**Beskrivning**
static int abs(**int** x)	ger absolutbeloppet av heltalet x
static double exp(**double** p)	ger värdet av e^p
static double log(**double** x)	ger den naturliga logaritmen (ln) av x.
static int max(**int** a, **int** b)	ger det största av heltalen a och b.
static int min(**int** a, **int** b)	ger det minsta av heltalen a och b.

Metoderna abs, min och max finns också för **long**, **float** och **double**.

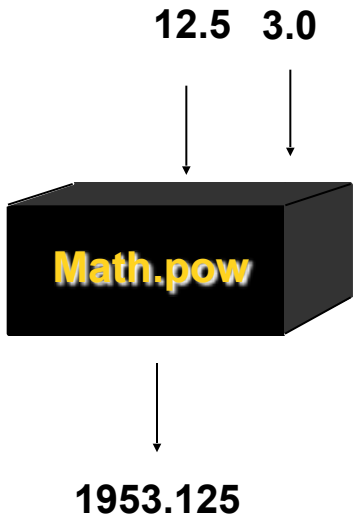
Matematiska standardfunktioner



Metodens namn, returtyp, parametrar och parametertyper utgör metodens *gränssnitt*. När man använder en metod måste anropet av metoden överensstämma med metodens gränssnitt.

Ett korrekt anrop av metoden pow har följande utseende:

```
double x = Math.pow(12.5, 3.0);
```



- variabeln x har samma typ som metodens returtyp
- fullständiga namnet för en klassmetod är klassens namn följt av '.' följt av metodens namn
- parametrarna till metoden vid anropet överensstämmer till *antal*, *typ* och *ordning*.

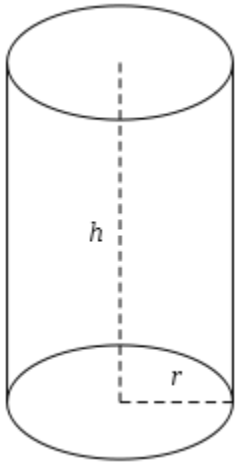
Problemexempel

Skriv ett program som läser in radien och höjden av en cylinder och beräknar cylinderns volym.

Volymen V av en cylinder definieras av formeln

$$V = \pi r^2 h$$

där *r* och *h* är radien respektive höjden av cylindern.



Råd på vägen

Tänk först, koda sedan!

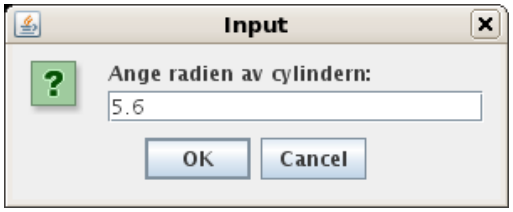
Ju tidigare du börjar koda, ju längre tid kommer det att ta innan du har ett fungerande program!

Analys:

Indata: Cylinderns radie och cylinderns höjd.

Utdata: Cylinderns volym.

Exempel på körning:



Design:

Algoritm:

1. Skriv texten "Ange cylinderns radie:"
2. Läs cylinderns radie *r*.
3. Skriv texten "Ange cylinderns höjd:"
4. Läs cylinders höjd *h*.
5. Beräkna cylinderns volym *V* med hjälp av formeln $V = \pi r^2 h$.
6. Skriv ut cylinderns volym *V*.

Datarepresentation:

Cylinderns radie *r* är ett reellt tal.

Cylinderns höjd *h* är ett reellt tal.

Cylinderns volym *V* är ett reellt tal.

I standardklassen `Math` finns konstanten `PI` att tillgå för att representera π.

Implementation:

```
/*Programmet läser in radien och höjden av en cylinder, samt beräknar och skriver ut
volymen av cylindren */
import javax.swing.*;
public class Cylinder {
    public static void main (String[] arg) {
        String input = JOptionPane.showInputDialog("Ange radien av cylindern:");
        double radius  = Double.parseDouble(input);
        input = JOptionPane.showInputDialog("Ange höjden av cylinden:");
        double height = Double.parseDouble(input);
        double volume = Math.PI * Math.pow(radius, 2) * height;
        JOptionPane.showMessageDialog(null, "Volymen av en cylinder med höjden "
                                            + height + " och radien " + radius + " är " + volume);
    } //main
} //Cylinder
```

Iakttagelse 1: Om vi t.ex ger indatavärdena 2.4 och 6.7 får vi resultatet 121.2403436873373. Är verkligen precisionen i resultatet riktig?

Iakttagelse 2: Vore det inte enklare att läsa in de båda indatavärdena i samma dialogruta?

Redigering av utskrifter

För att redigera utskrifter kan man använda klassmetoderna

System.out.printf	vid utskrift i terminalfönstret
String.format	vid utskrift i en dialogrutor

Dessa båda metoder fungerar på likartat sätt. Man anger en formatmall som specificerar hur utskriften skall se ut, samt de utskrifter som skall göras

Formatmallen har följande uppbyggnad:

%[*flagga*][*vidd*][.*precision*] *argumenttyp*

De viktigaste flaggorna och argumenttyperna är:

Flagga	Betydelse
-	utskriften vänsterställs (utfyllnad med mellanslag sker till höger)
+	talets tecken (+ eller -) skrivs alltid ut
0	utfyllnad sker med 0 istället för mellanslag
Argumenttyp	**Betydelse**
s	sträng
d	heltal på decimal form
f	reella tal på decimal form med heltalsdel och decimaler
e	reella tal på exponentform
o	heltal på oktal form
x	heltal på hexadecimal form

Redigering av utskrifter: Exempel

Nedanstående kod illustrera hur utskrifts formateringen går till:

```
System.out.printf("Min: är %3.2f och max är %e\n", 123.123, 567.567);
System.out.printf("%5s%-6s%s\n", "AB", "CDE", "FH");
System.out.printf("%2d:%2d:%2d\n", 1, 2, 3);
System.out.printf("%02d:%02d:%02d\n", 1, 2, 3);
final String HEADER_FORMAT = "%-10s%10s\n";
final String DATA_FORMAT = "%5.2f%10d\n";
System.out.printf(HEADER_FORMAT, "Pris", "Kvantitet");
System.out.printf(DATA_FORMAT, 12.345, 10);
```

Utskrifterna blir:

```
Min: är 123.12 och max är 5.675670e+02
   ABCDE   FH
 1: 2: 3
01:02:03
Pris       Kvantitet
12.35        10
```

Vid utskrift I ett dialogfönster används metoden String.format för att erhålla den formaterade strängen som skall skrivas ut

```
String output = String.format("Min: är %3.2f och max är %e\n", 123.123, 567.567);
JOptionPane.showMessageDialog(null, output);
```

Klassen Scanner

I ett program som behöver flera indatavärden kan det ibland vara klumpigt att läsa in varje enskilt indatavärde via en egen dialogruta. Ofta vill man kunna ge flera indatavärden i samma dialogruta.

För att möjliggöra detta kan man använda sig av klassen Scanner som finns i paketet java.util.

Exempel:

```
import java.util.*;
import javax.swing.*
. . .
   String input = JOptionPane.showInputDialog("Ange det första heltalet och"
                                              + " det första reella talet: ");
   Scanner scan = new Scanner(input);
   int integerValue = scan.nextInt();
   double realValue = scan.nextDouble();
```

Klassen Scanner

Ett objekt av klassen Scanner kan dela upp strängen den får i delar, s.k *tokens*.

Tokens avskiljs (by default) av *vita tecken* (blanktecken, '\t' och '\n').

I klassen Scanner finns bl.a. följande användbara instansmetoder

Metod	Beskrivning
int nextInt()	returnerar nästa token som en **int**
double nextDouble()	returnerar nästa token som en **double**
boolean nextBoolean()	returnerar nästa token som en **boolean**
String next()	returnerar nästa token som en String
boolean hasNextInt()	returnerar värdet **true** om nästa token är en **int**, annars returneras **false**
boolean hasNextDouble()	returnerar värdet **true** om nästa token är en **double**, annars returneras **false**
boolean hasNextBoolean()	returnerar värdet **true** om nästa token är en **boolean**, annars returneras **false**
boolean hasNext()	returnerar värdet **true** om det finns fler tokens, annars returneras **false**

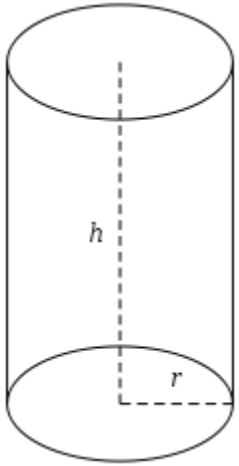
Problemexempel

Skriv ett program som läser in radien och höjden av en cylinder och beräknar cylinderns volym.

Volymen V av en cylinder fås av formeln

$V = \pi r^2 h$, där *r* är radien och *h* är höjden av cylindern.

Inmatningen av cylinderns radie och höjd skall göras i samma dialogruta och cylinderns volym skall skrivas ut med exakt 2 decimaler.



Analys:

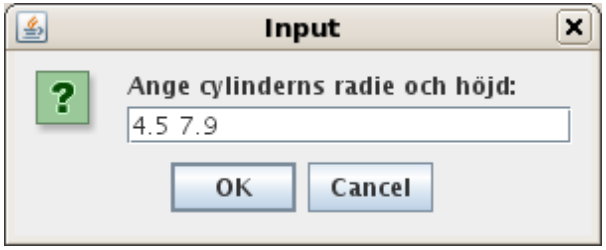
Detta problem är exakt som föregående problem förutom att en precisering har gjorts avseende hur indata skall läsa och hur utdata skall skrivas.

I analyserna av de båda problemen är det således enbart exemplifieringen av hur en körning ser som skiljer sig.

Indata: Cylinderns radie och cylinderns höjd.

Utdata: Cylinderns volym.

Exempel på körning:



Design:

Algoritm:

1. Skriv texten "Ange cylinderns radie och höjd:"
2. Läs cylinderns radie *r*.
3. Läs cylinders höjd *h*.
4. Beräkna cylinderns volym *V* mha formeln $V = \pi r^2 h$.
5. Skriv ut cylinderns volym *V*.

Datarepresentation:

Cylinderns radie *r* är ett reellt tal.

Cylinderns höjd *h* är ett reellt tal.

Cylinderns volym *V* är ett reellt tal.

I standardklassen Math finns konstanten PI att tillgå för att representera π.

Implementation:

```
/*Programmet läser in radien och höjden av en cylinder, samt beräknar och
   skriver ut volymen av cylindren */
import javax.swing.*;
import java.util.*;
public class Cylinder2 {
   public static void main (String[] arg) {
      String input = JOptionPane.showInputDialog("Ange cylinderns radie och höjd:");
      Scanner sc = new Scanner(input);
      double radius  = sc.nextDouble();
      double height = sc.nextDouble();
      double volume = Math.PI * Math.pow(radius, 2) * height;
      String output = String.format("%s %.2f", "Volymen av cylinden är", volume);
      JOptionPane.showMessageDialog(null, output);
   } //main
} //Cylinder2
```

Inläsning från kommandofönstret

Allt som skrivs in via kommandofönstret hamnar i en s.k. ström med namnet System.in.

Genom att koppla System.in till ett objekt av klassen Scanner kan man på ett enkelt sätt läsa data från kommandofönstret.

Exempel:

I nedanstående program görs inläsning från System.in och skrivning till System.out.

```
import java.util.*;
public class Cylinder3 {
   public static void main(String[] arg) {
      System.out.print("Ange cylinderns radie och höjd: ");
      Scanner sc = new Scanner(System.in); //kopplar kommandofönstret till Scanner-
objektet
      double radius  = sc.nextDouble();
      double height = sc.nextDouble();
      double volume = Math.PI * Math.pow(radius, 2) * height;
      System.out.printf("%s %.2f", "Volymen av cylinden är", volume);
   } //main
} //Cylinder3
```

Typkonvertering

Java är ett *typat språk*, vilket innebär att man *inte* kan blanda olika datatyper i uttryck hur som helst.

Exempel:

Tilldelningen

int value = 5 + 10.5;

är ej tillåten, eftersom variabeln value som skall tilldelas resultatet av uttrycket är av typen **int** medan uttrycket innehåller en *litteral* 10.5 som är av typen **double**.

Automatisk typomvandling utförs av kompilatorn då en typomvandling kan göras som inte innebär risk att information går förlorad.

Exempel:

Tilldelningen

double value = 5 + 10.5;

är tillåten, eftersom heltalet 5 kan omvandlas till det reella talet 5.0 utan att precisionen i talet förloras.

Automatisk typomvandling sker enligt:

byte => **short** => **int** => **long** => **float** => **double**

Typkonvertering

Om ett uttryck innehåller flera datatyper och det finns risk för att information måste en *explicit typomvandling* göras.

Exempel:

```
int value = 5 + (int) 10.5;
int value2 = 5 + (int) (10.5 + 13.6);
```

Vid explicit typomvandling används *avhuggning*, dvs. efter ovanstående tilldelningar har variabeln value värdet 15 och variabeln value2 värdet 29.

För att avrunda ett reellt tal till ett heltal används metoden round i klassen Math

```
long round(double x)
```

Exempel:

```
int value3 = 5 + (int) Math.round(10.5);
```

Varför behöver typomvandling göras i ovanstående och nedanstående exempel?

```
int sum = 10 + 16 + 24 + 32;
int numbers = 4;
double mean = (double) sum / numbers;
```

Komponenterna i Java

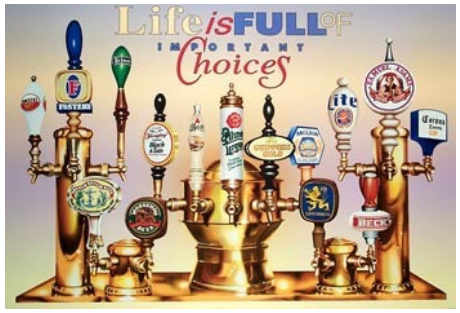
Styrstrukturer:

De konstruktioner som finns i språket för att instruera datorn om vilka operationer som skall utföras.

Sekvens:
- tilldelningssatsen
- selektionssatser
- iterationssatser
- **return**-satsen
- anrop av **void**-metod
- programblock
- exception

Selektion:
- **if**-satsen
- **switch**-satsen

Iteration:
- **while**-satsen
- **do-while**-satsen
- **for**-satsen

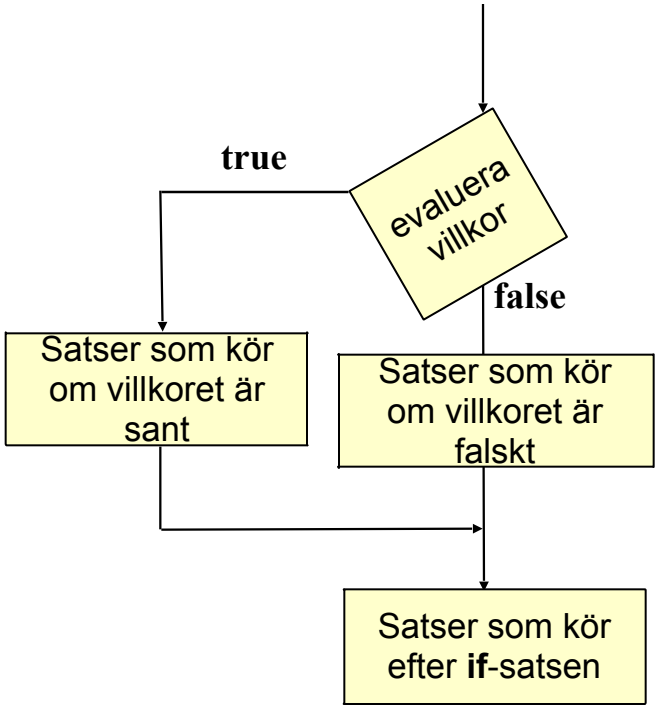


Selektering: if-satsen

Den generella **if**-satsen har följande utseende:

```
if (villkor)
    sats1;
else
    sats2;
```

och är en *tvåvägs-selektering*.



Semantiskt innebär ovanstående **if**-sats:

beräknas villkor *till sant utförs* sats1 *annars utförs* sats2

Satserna sats1 och sats2 är godtyckliga satser, t.ex.
en **if**-sats (vi får då *nästlade* ***if****-satser*)
ett *programblock* (satser omslutna mellan { och })
en tom sats

Selektering: if-satsen

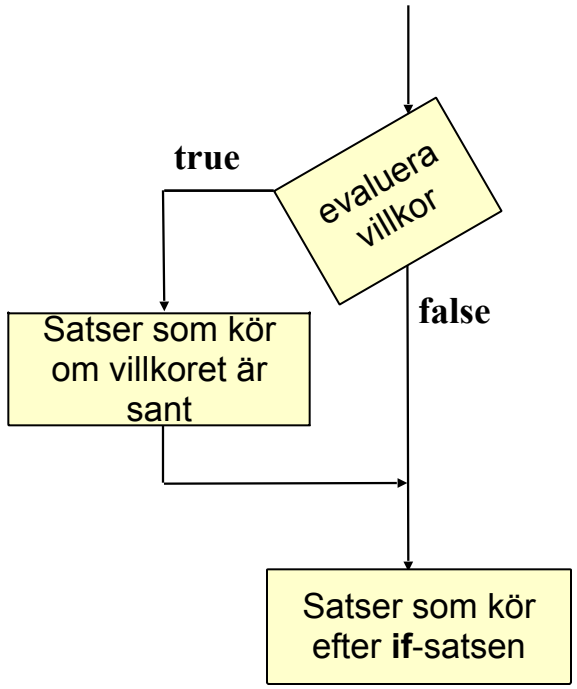
Om satsen i **else**-delen av en **if**-sats är en tom sats, som i exemplet nedan

```
if (villkor)
    sats1;
else
    ; //tom sats
```

kan man utesluta **else**-delen helt och hållet

```
if (villkor)
    sats1;
```

Detta betyder att vi har en *envägs-selektering*.



Semantiskt innebär ovanstående **if**-sats:

beräknas villkoret villkor *till sant utförs* sats1 *annars görs ingenting*

Selektering: if-satsen

Skriv aldrig en **if**-sats enligt nedan

```
if (villkor)
    ;
else
    sats;
```

Negera villkoret och skriv istället en envägs-selektering enligt:

```
if (!villkor)
   sats;
```

Anledningen är att koden blir enklare att läsa och förstå!

Programblock

Ett *programblock* grupperar ihop ett antal programsatser till en *sammansatt sats*. Ett block kan användas överallt där en enda programsats är tillåten och får innehålla alla typer av satser.

Görs deklarationer i ett programblock blir dessa *lokala deklarationer*, dvs de blir endast tillgängliga inom det programblock där de har deklarerats.

Ett programblock påbörjas med en vänsterklammer { och avslutas med en högerklammer }.

```
if (villkor1) {    //tvåvägs-selektering
   sats1A;
   . . .
   sats1N;
}
else {
   sats2A;
   . . .
   sats2N;
}
```

```
if (villkor2) {    //envägs-selektering
   sats3A;
   . . .
   sats3N;
}
```

Nästlade if-satser: Flervägs-selektering

```
if (villkor1) {
    sekvens av satser
}
else if (villkor2) {
    sekvens av satser
}
    . . .
else if (villkorN) {
    sekvens av satser
else {
    sekvens av satser
}
```

Logiska operatorer i Java

När ett villkor evalueras fås ett värde av typen **boolean**.

Typen **boolean** har bara två värden, **true** och **false**.

På datatypen **boolean** finns följande operationer:

Notation	Betydelse
==	lika med
!=	inte lika med
&&	logiskt och
||	logiskt eller
!	logiskt icke
^	exklusivt eller

Med dessa operationer kan man bilda *logiska uttryck* eller *villkorsuttryck*:

```
(year % 4 == 0 && year % 100 != 0) || year % 400 == 0
```

Vad beräknas i ovanstående villkor?

Logiska operatorer i Java

För de logiska operatorerna && och || använder Java *lat evaluering*. Vid lat evaluering evalueras ett uttryck bara så långt som det är absolut nödvändigt. Detta innebär att

· om den vänstra operanden till operatorn && blir **false** så beräknas aldrig den högra operanden

· om den vänstra operanden till operatorn || blir **true** så beräknas aldrig den högra operanden.

Lat evaluering är praktiskt i många sammanhang när den högra operanden kan generera exekveringsfel.

Betrakta nedanstående villkor:

```
(denom != 0) && (num / denom > 10.5)
```

Här vill man undvika att göra division med 0. Vi kontrollerar därför att nämnaren denom inte är 0 *innan* divisionen denom/num skall utföras.

Operatorernas prioritetsordning

Varje operator har en *prioritet* som avgör i vilken ordning operatorerna i ett uttryck skall utföras. Har två operatorer samma prioritet utförs de *från vänster till höger*. Följande *prioritetsordning* gäller i Java:

Prioritet	Operator	Notation
högst	postfix inkrement och dekrement	a++, a--
	prefix inkrement och dekrement unära + och + logisk icke	++a, --a +a, -a !
	multiplikation, division, modulus	*, /, %
	addition, subtraktion	+, -
	mindre än, större än, mindre eller lika med, större eller lika med	<, >, <=, >=
	likhet och olikhet	==, !=
	exklusiv eller	^
	logiskt och	&&
lägst	logiskt eller	||

Operatorernas prioritetsordning

För att upphäva den fördefinierade prioritetsordningen kan man använda parenteser.

Exempel:

Enligt den fastställda prioritetsordningen kommer uttrycket

2 + 3 * 5 - 4

att beräknas som

2 + (3 * 5) - 4

Vill man beräkna uttrycket i annan ordning måste parenteser användas, t.ex

(2 + 3) * (5 – 4)

Problemexempel

Skriv ett program som beräknar porto för brev enligt följande taxa:

Vikt högst (gram)	Porto (kronor)
20	5.50
100	11.00
250	22.00
500	33.00

Brev över 500 gram räknas som paket.

Analys:

Indata: Brevets vikt i gram.

Utdata: Brevets porto i kronor.

Design:

Algoritm:

1. Läs brevets vikt till variabeln *vikt*.
2. Om *vikt* $\leq$ 0.0, så skriv att angiven vikt är ogiltig.
3. Om 0.0 < *vikt* $\leq$ 20.0, så skriv att portot är 5.50 kr.
4. Om 20.0 < *vikt* $\leq$ 100.0, så skriv att portot är 11.00 kr.
5. Om 100.0 < *vikt* $\leq$ 250.0, så skriv att portot är 22.00 kr.
6. Om 250.0 < *vikt* $\leq$ 500.0, så skriv att portot är 33.00 kr.
7. Om *vikt* > 500.0, så skriv att brevet måste sändas som paket.

Datarepresentation:

Variabeln *vikt* är av datatypen **double**.

Implementation 1:

```
/* Programmet läser in vikten på ett brev och skriver ut brevets porto    */
import javax.swing.*;
public class  Postage {
   public static void main( String[] arg) {
      String input = JOptionPane.showInputDialog("Ange vikten:");
      double weight = Double.parseDouble(input);
      if (weight <= 0.0)
         JOptionPane.showMessageDialog(null, "Du har angivit en ogiltig vikt!!");
      if ((weight > 0.0) && (weight <= 20.0))
         JOptionPane.showMessageDialog(null, "Portot är 5.50 kronor.");
      if ((weight > 20.0) && (weight <= 100.0))
         JOptionPane.showMessageDialog(null, "Portot är 11.00 kronor.");
      if ((weight > 100.0) && (weight <= 250.0))
         JOptionPane.showMessageDialog(null, "Portot är 22.00 kronor.");
      if ((weight > 250.0) && (weight <= 500.0))
         JOptionPane.showMessageDialog(null,"Portot är 33.00 kronor.");
      if (weight > 500.0)
         JOptionPane.showMessageDialog(null, "Måste gå som paket.");
   } // main
} // Postage
```

Alternativ implementation:

I ovanstående implementation skapas en dialogruta i varje **if**-sats. I nedanstående implementation skapas istället en dialogruta efter alla **if**-satser och i **if**-satsen skapas den utskrift som skall stå i dialogrutan för respektive utfall.

```
import javax.swing.*;
public class  Postage2 {
   public static void main( String[] arg) {
      String input = JOptionPane.showInputDialog("Ange vikten:");
      double weight = Double.parseDouble(input);
      String output = "";
      if (weight <= 0.0)
         output = "Du har angivit en ogiltig vikt!!";
      if ((weight > 0.0) && (weight <= 20.0))
         output = "Portot är 5.50 kronor.";
      if ((weight > 20.0) && (weight <= 100.0))
         output = "Portot är 11.00 kronor.";
      if ((weight > 100.0) && (weight <= 250.0))
         output = "Portot är 22.00 kronor.";
      if ((weight > 250.0) && (weight <= 500.0))
         output = "Portot är 33.00 kronor.";
      if (weight > 500.0)
         output = "Måste gå som paket.";
      JOptionPane.showMessageDialog(null, output);
   } // main
} // Postage2
```

Måste initieras! Varför?

Alternativ design:

Algoritm:

1. Läs brevets vikt *vikt*.
2. Om *vikt* ≤ 0 så
   skriv att brevets vikt är ogiltig (negativ).
   annars om *vikt* ≤ 20.0 så
   skriv att portot är 5.50 kr
   annars om *vikt* ≤ 100.0 så
   skriv att portot är 11.00 kr
   annars om *vikt* ≤ 250.0 så
   skriv att portot är 22.00 kr
   annars om *vikt* ≤ 500.0 så
   skriv att portot är 33.00 kr
   annars
   skriv att brevet måste sändas som paket

Implementation:

```
/* Programmet läser in vikten på ett brev och skriver ut brevets porto */
import javax.swing.*;
public class Postage3 {
   public static void main( String[] arg) {
      String input = JOptionPane.showInputDialog("Ange vikten:");
      double weight = Double.parseDouble(input);
      String output;
      if (weight <=0.0)
         output = "Du har angivit en ogiltig vikt!!";
      else if (weight <=20.0)
         output = "Portot är 5.50 kronor.";
      else if (weight <= 100.0)
         output = "Portot är 11.00 kronor.";
      else if (weight <=250.0)
         output = "Portot är 22.00 kronor.";
      else if (weight <=500.0)
         output = "Portot är 33.00 kronor.";
      else
         output = "Måste gå som paket.";
      JOptionPane.showMessageDialog(null, output);
   } // main
} // Postage3
```

Behöver inte initieras!
Varför?
Bör variabeln initieras?