EDAA20
Programmering och databaser

Föreläsning 1, 2023-08-28, Niklas Fors

Baserat på föreläsningsbilder av Anna Axelsson

Kursmål

Programmering (6 hp):

- Lösa problem med hjälp av dator
- Grundläggande programmering
- Grundläggande objektorientering och programspråket Java

Databaser (1.5 hp):

- Lagra data i relationsdatabaser
- Använda SQL för att hämta data från relationsdatabaser

Viktiga länkar

Kurshemsida:

http://cs.lth.se/edaa20/

Moodle:

https://moodle.cs.lth.se/course/view.php?id=998 (anmälningskod: edaa20)

Kort om Niklas Fors

- Civilingenjör inom Datateknik (LTH), 2010
- Teknisk doktor inom datavetenskap (LTH), 2016
- Biträdande universitetslektor, datavetenskap (LTH), 2021-

- Undervisar EDAA20 och EDAA70 (Inledande programmering i Python)
- Forskar inom programanalys, programspråk för automationsteknik och kompilatorteknik

Kursstruktur

Programmering

- 14 föreläsningar
- 11 obligatoriska datorlaborationer över 14 tillfällen
- Övningsuppgifter i Moodle
- Resurstider

Databaser

- 2 inspelade föreläsningar (se Moodle)
- 2 obligatoriska laborationer (läsvecka 2). Anmäl er i Moodle i slutet av veckan
- 1 obligatorisk seminarieövning (läsvecka 3)

Från **Kursprogrammet** (se Moodle-sida)

Programmering

	Läsvecka 1	Läsvecka 2	Läsvecka 3	Läsvecka 4	Läsvecka 5	Läsvecka 6	Läsvecka 7
Mån	F1	F3	F5	F7	F9	F11	F13
Tis	F2	F4	F6	F8	F10	F12	F14
Ons	L1	L3	L4	L6	L8	L	L11
Fre	L2	L	L5	L7	L9	L10	L
Ons	R	R	R	R	R	R	R

F föreläsningar, L datorlaborationer (obligatoriska), R resurstid

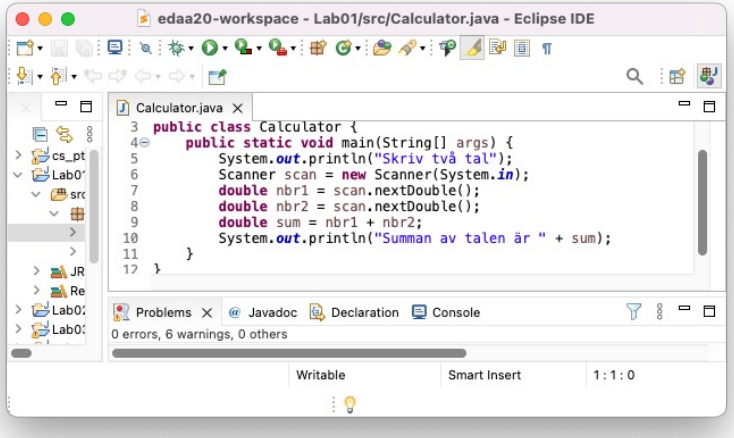
Databaser

	Läsvecka 1	Läsvecka 2	Läsvecka 3
Tis		LD1	
Tors		LD2	S

LD datorlaborationer databaser (obligatoriska)
S seminarieövning (obligatorisk)
Se inspelade två föreläsningar

Laborationer i programmering

- 11 obligatoriska laborationsuppgifter
- Uppgifterna löses parvis
  - Arbeta med någon med liknande programmeringsbakgrund
- På egen dator eller Linuxdator (i E-huset)
- Labbarna är kluriga och man fastnar ofta
  - Koncept från föreläsningarna appliceras på större problem
  - Uppgifterna tvingar er att tänka själva



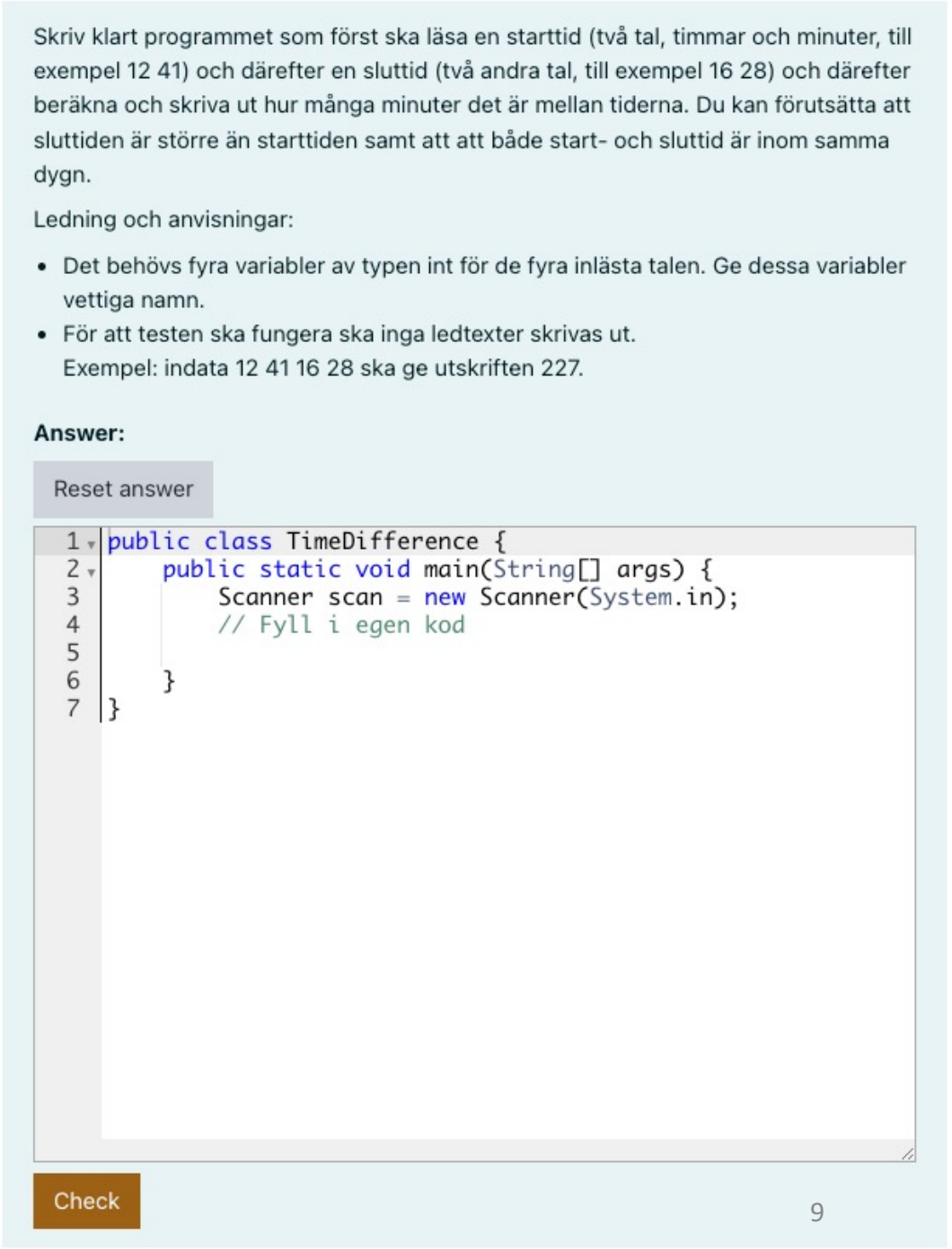
Labbtillfällen

- Få hjälp och redovisa laborationer vid labbtillfällen
  - **Anmäl er till labbtillfälle via länk i Moodle!**
  - Sök labbpartner i pausen av föreläsningen om ni inte har
- Labbledarna är där för att hjälpa er
  - Var inte rädda för att fråga om hjälp – men försök själva först
  - Fråga om labbarna, Moodle-uppgifter, osv
- OK att ligga någon uppgift efter,
  men man kan ***inte*** komma i slutet av kursen och redovisa alla uppgifter

Uppgifter i Moodle

- Moodle-uppgifter löses i webbläsaren
- Mindre uppgifter än labbarna
- Fokuserar på ett koncept i taget
- Lös gärna uppgifterna enskilt
  - Du måste lära dig att koda själv också

- Bonuspoäng fås genom att lösa minst 50% eller 90% av uppgifterna i Moodle och ger 0,5 eller 1,0 poäng, respektive. På tentamen kan man få 40 poäng, varav 20 är godkänt.



Tentamen

- På egen dator i sal
  - Låna dator? Säg till i god tid. Det finns några lånedatorer.
- I Moodle och med Respondus LockDown Browser
  - Kräver Windows/Mac
- Förståelseinriktad och testar programmeringsfärdigheter
- Hjälpmedel: Java Snabbreferens
- Betyget (U, 3-5) på kursen ges av tentamen
- Ordinarie tenta: måndag 23 okt kl 8-13

Java snabbreferens (se Moodle)

Bekanta er med den under kursens gång

Java snabbreferens @ LTH

Vertikalstreck | används mellan olika alternativ. Parenteser ( ) används för att gruppera en mängd alternativ. Hakparenteser [ ] markerar valfria delar. En sats betecknas `stmt` medan `x`, `i`, `s`, `ch` är variabler, `expr` är ett uttryck, `cond` är ett logiskt uttryck. Med `...` avses valfri, extra kod.

Java 1(4) Java 2(4)

Satser

Block	`{stmt1; stmt2; ...}`	fungerar "utifrån" som **en** sats
Tilldelning	`x = expr;`	variabeln och uttrycket av kompatibel typ
Förkortade	`x += expr;` `x++;`	x = x + expr; även −=, *=, /= x = x + 1; även x − −
if-sats	`if (cond) {stmt; ...}` `[else { stmt; ...} ]`	utförs om cond är true utförs om false

switch-sats	`switch (expr) {`
	`  case A: stmt1; break;`
	`  ...`
	`  default: stmtN; break;`
	`}`
		expr är ett heltalsuttryck
		utförs om expr = A (A konstant)
		"faller igenom" om break saknas
		sats efter default: utförs om inget case passar

for-sats	`for (int i = a; i < b; i++) {`
	`  stmt; ...`
	`}`
		satserna görs för i = a, a+1, ..., b-1
		Görs ingen gång om a >= b
		i++ kan ersättas med i = i + step

for-each-sats	`for (int x: xs) {`
	`  stmt; ...`
	`}`
		xs är en samling, här med heltal
		x blir ett element i taget ur xs
		fungerar även med array

while-sats	`while (cond) {stmt; ...}`	utförs så länge cond är true

do-while-sats	`do {`
	`  stmt; ...`
	`} while (cond);`
		utförs minst en gång,
		så länge cond är true

return-sats	`return expr;`	returnerar funktionsresultat

Uttryck

Aritmetiskt uttryck	(x + 2) * i / 2 + i % 2	för heltal är / heltalsdivision, % "rest"
Objektuttryck	new Classname(...) | ref-var | null | function-call | this | super	
Logiskt uttryck	! cond | cond && cond | cond || cond | relationsuttryck | true | false	
Relationsuttryck	expr (< | <= | == | >= | > | !=) expr	för objektuttryck bara == och !=, också typtest med expr instanceof Classname
Funktionsanrop	obj-expr.method(...) Classname.method(...)	anropa "vanlig metod" (utför operation) anropa statisk metod
Array	new int[size] vname[i] vname.length	skapar int-array med size element elementet med index i, 0..length−1 antalet element

Matris	new int[r][c]
	m.length
	m[i].length
		//Skapar matris med r rader och c kolonner
		//Ger matrisens längd (d.v.s. antalet rader)
		//Ger antalet element (längden) på raden i

Typkonvertering	(newtype) expr (int) real-expr (Square) aShape	konverterar expr till typen newtype – avkortar genom att stryka decimaler – ger ClassCastException om aShape inte är ett Square-objekt

Deklarationer

Allmänt	[<protection>] [static] [final] <type> name1, name2, ...;	
<type>	byte | short | int | long | float | double | boolean | char | Classname	
<protection>	public | private | protected	för attribut och metoder i klasser (paketskydd om inget anges)
Startvärde	int x = 5;	startvärde bör alltid anges
Konstant	final int N = 20;	konstantnamn med stora bokstäver
Array	<type>[] vname = new <type>[10];	deklarerar och skapar array
Matris	<type>[][] m = new <type>[4][5];	// deklarerar och skapar 4x5 matrisen m

Klasser

Deklaration	[public] [abstract] class Classname
	[extends Classname1] [implements Interface1, Interface2, ...] {
	<deklaration av attribut>
	<deklaration av konstruktorer>
	<deklaration av metoder>
	}	

Attribut	Som vanliga deklarationer. Attribut får implicita startvärden, 0, 0.0, false, null.	

Konstruktor	<prot> Classname(param, ...) {
	stmt; ...
	}
		Parametrarna är de parametrar som ges vid new Classname(...). Satserna ska ge attributen startvärden

Metod	<prot> <type> name(param, ...) {
	stmt; ...
	}
		om typen inte är void måste en return-sats exekveras i metoden

Huvudprogram	public static void main(String[] args) { ... }	
Abstrakt metod	Som vanlig metod, men abstract före typnamnet och {...} ersätts med semikolon. Metoden måste implementeras i subklasserna.	

Standardklasser, java.lang, behöver inte importeras

Object	Superklass till alla klasser.	

	boolean equals(Object other);
	int hashCode();
	String toString();
		ger true om objektet är lika med other
		ger objektets hashkod
		ger en läsbar representation av objektet

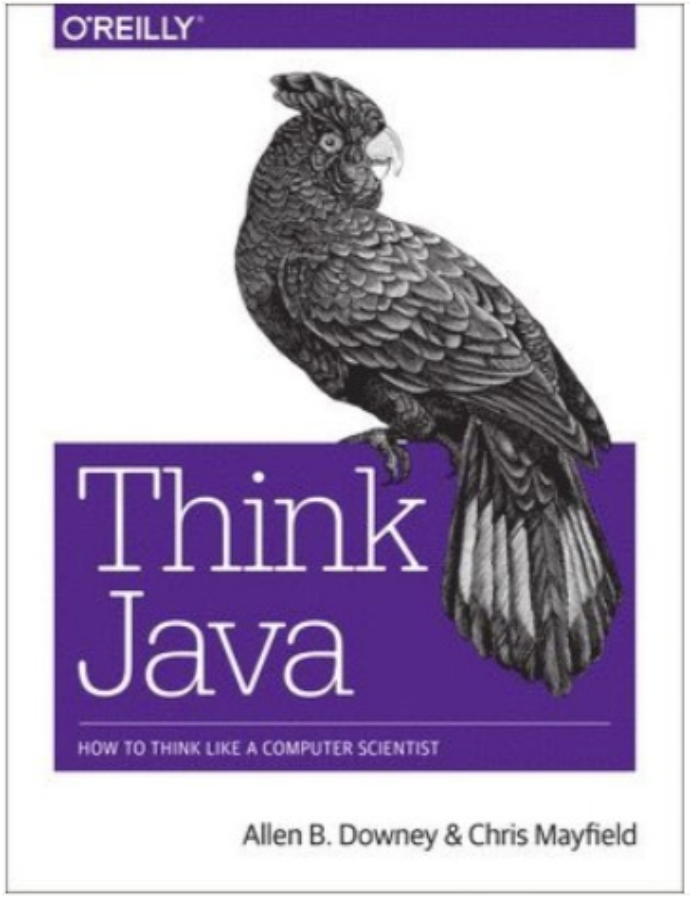
Math	Statiska konstanter Math.PI och Math.E. Metoderna är statiska (anropas med t ex Math.round(x)):	

	long round(double x);
	int abs(int x);
	double hypot(double x, double y);
	double sin(double x);
	double exp(double x);
	double pow(double x, double y);
	double log(double x);
	double sqrt(double x);
	double toRadians(double deg);
		avrundning, även float → int
		$\|x\|$, även double, ...
		$\sqrt{x^2+y^2}$
		$\sin x$, liknande: cos, tan, asin, acos, atan
		e^x
		x^y
		$\ln x$
		$\sqrt{x}$
		$deg \cdot \pi/180$

System	void System.out.print(String s);
	void System.out.println(String s);
	void System.exit(int status);
	Parametern till print och println kan vara av godtycklig typ: int, double, ...
		skriv ut strängen s
		som print men avsluta med ny rad
		avsluta exekveringen, status != 0 om fel

Kursbok och kursmaterial

- Allen B. Downey & Chris Mayfield: **Think Java** https://greenteapress.com/wp/think-java/
  - Kan läsas på nätet (interaktiv version), laddas ner (pdf, gratis) eller köpas (bok)
- Övrigt kursmaterial:
  - Föreläsningsbilderna
  - Laborationsuppgifterna
  - PM om programutvecklingsmiljön Eclipse
  - Övningsuppgifter i Moodle
  - Java snabbreferens



Sista kursomgång

EDAA20 ges för sista gången HT 2023

Ersätts av EDAA90 Programmeringsteknik och databaser, grundkurs

- Python istället för Java

Tentor i EDAA20: oktober 2023, januari 2023, april 2024, augusti 2024

Övergångsregler: https://cs.lth.se/javagrundkurser/

Hur klarar man kursen?

- **Studera 20 timmar/veckan** (inkl. föreläsningar, labbar, självstudier, osv)
- Följ föreläsningarna (koncept introduceras)
- Lös övningsuppgifter i Moodle (fokuserar på enskilda koncept)
- Gör labbarna för att förstå (koncept kombineras)
- **Programmera mycket!**
- Häng med från början och försök vara i fas
  - Grunderna är väldigt viktiga
  - Senare delar bygger vidare på grunderna

Varför lära sig programmera?

- Det är kul!
- Träning i problemlösning
- Lösa ingenjörsproblem (beräkningsprogrammering, data science, osv)
- Automatisera småsaker *(scripts)*

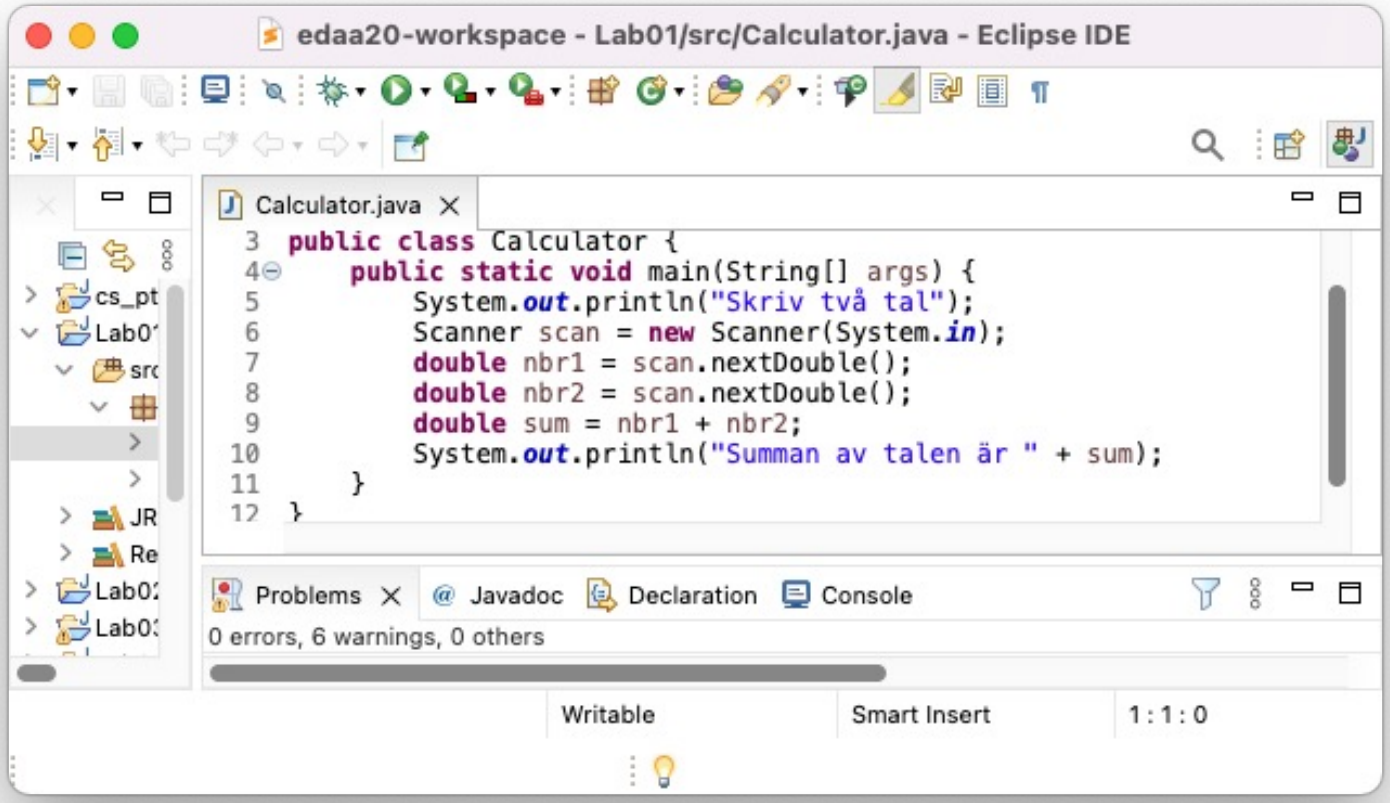
- Programmering av IT-system
- Delaktig i utveckling av IT-system (projektledare, beställare, etc)
  - Beställarkompetens
  - Digitalisering. Hur förändrar nya IT-system samhället/verksamheter?

Java

- Java är ett programmeringsspråk från 1995
- Ett av de mest använda språken (Android-appar, …)
- Objektorienterat språk
  - Program organiseras i ***objekt***
  - Objekt är instanser av ***klasser***
- Statisk typning
  - Typer behöver anges (heltal, decimaltal, osv)

(Java ska ej förväxlas med JavaScript)

Eclipse



Eclipse är en *utvecklingsmiljö* som man programmerar i

Används på labbarna

Se Moodle för installations-instruktioner på egen dator

Datorprogram

Ett ***datorprogram*** är stegvisa instruktioner till datorn som beskriver hur ett problem ska lösas.

I instruktioner manipuleras information (data) och skrivs i ett programspråk som exempelvis Java

Ett exempel på ett Java-program

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

Programmet skriver ut "Hello World!" i ett terminalfönster.

Programkoden sparas i en fil som heter **`Hello.java`**

Ett exempel på ett Java-program

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

OBS! Allt detta behövs för ett huvudprogram.

Senare i kursen kommer ni att förstå vad det betyder.

Lite terminologi

```
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello world!");
  }
}
```

En ***klass*** med namnet `Hello`

En ***metod*** med namnet `main`

Ett ***metodanrop*** till `println`

Programkod

Programmet på förra bilden är skrivet i Java

- Java är ett högnivåspråk (skapat för människor)
- Denna kod kallas *källkod* och lagras i en fil med filändelsen `.java`

Innan programmet körs (*exekveras*) översätts det till enklare instruktioner som går att köra i datorns processor (CPU)

- Java-programmet översätts (*kompileras*) till *bytekod* som sedan interpreteras (tolkas) i processorn.
- Bytekoden lagras i en fil med filändelsen `.class`

Editera – Kompilera – Exekvera

Editera

- Skriv källkoden i en fil med filändelsen `.java`

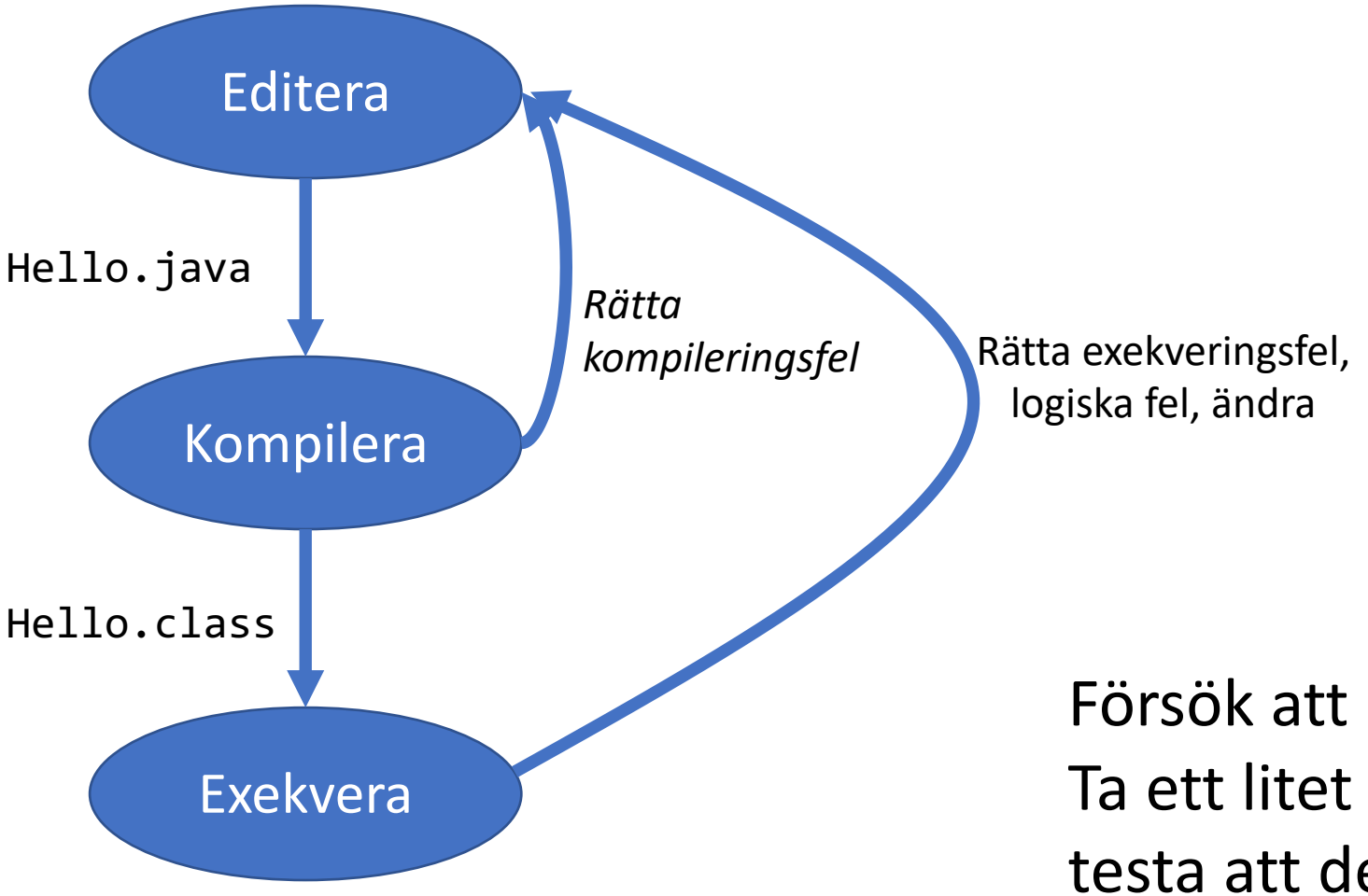
Kompilera

- Programkoden kontrolleras. Om koden är korrekt (dvs, inte bryter mot språkets regler) översätts den till bytekod, som lagras i en fil med filändelsen `.class`

Exekvera

- Programmet körs, dvs, bytekodinstruktionerna i programmet utförs

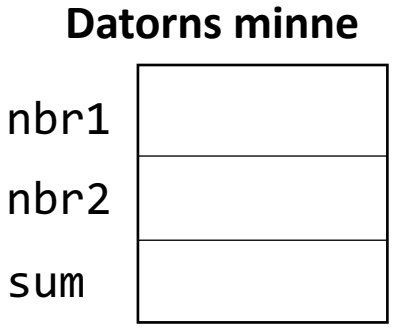
Editera – Kompilera – Exekvera



Försök att lösa en del av problemet. Ta ett litet steg i taget, exekvera och testa att det fungerar

Ett till exempel

```
public class Sum {
  public static void main(String[] args) {
    double nbr1 = 12.5;
    double nbr2 = 0.7;
    double sum = nbr1 + nbr2;
    System.out.println("Summan av talen är " + sum);
  }
}
```



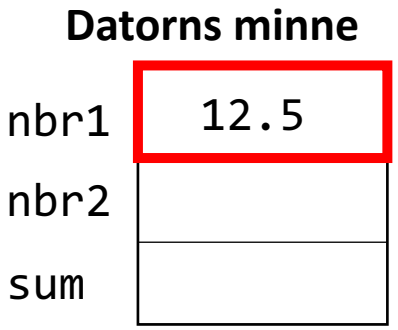
Programmet exekverar stegvis. Det summerar två värden och skriver ut summan.

`nbr1`, `nbr2`, `sum` är ***variabler***
`double` är en ***typ*** och anger vilka värden som är tillåtna (flyttal med decimaldel)

= är tilldelning (ej lika med!). Värdet till höger sparas i variabeln till vänster

Ett till exempel – Exekvering (1/4)

```
public class Sum {
  public static void main(String[] args) {
    double nbr1 = 12.5;
    double nbr2 = 0.7;
    double sum = nbr1 + nbr2;
    System.out.println("Summan av talen är " + sum);
  }
}
```

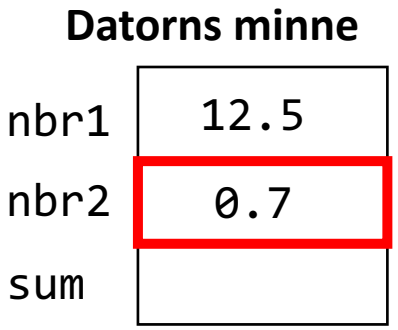


Den röda pricken representerar att vi har exekverat den raden

Värdet `12.5` sparas i variabeln `nbr1`

Ett till exempel – Exekvering (2/4)

```
public class Sum {
  public static void main(String[] args) {
    double nbr1 = 12.5;
    double nbr2 = 0.7;
    double sum = nbr1 + nbr2;
    System.out.println("Summan av talen är " + sum);
  }
}
```

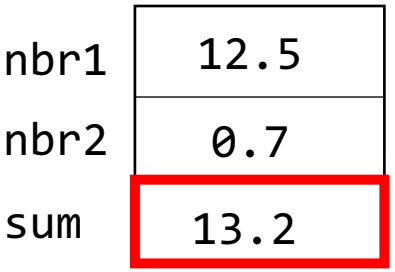


Värdet `0.7` sparas i variabeln `nbr2`

Ett till exempel – Exekvering (3/4)

```
public class Sum {
  public static void main(String[] args) {
    double nbr1 = 12.5;
    double nbr2 = 0.7;
  ●double sum = nbr1 + nbr2;
    System.out.println("Summan av talen är " + sum);
  }
}
```

Datorns minne



Värdet som finns i `nbr1` och `nbr2` adderas, och resultatet sparas i variabeln `sum`

Ett till exempel – Exekvering (4/4)

```
public class Sum {
  public static void main(String[] args) {
    double nbr1 = 12.5;
    double nbr2 = 0.7;
    double sum = nbr1 + nbr2;
  System.out.println("Summan av talen är " + sum);
  }
}
```

Datorns minne

nbr1	12.5
nbr2	0.7
sum	13.2

Skriver ut:

```
Summan av talen är 13.2
```

`System.out.println` är en färdigskriven metod för utskrift.
I kursen används färdigskriven programkod för exempelvis inläsning, rita osv.

Variabel

Variabler används för att lagra värden som programmet måste komma ihåg

- Variabler är ett namngivet utrymme i datorns minne
- Variabler måste deklareras och ha:
  - Ett **namn** (exempelvis nbr1)
  - En **typ** (exempelvis double) som anger vilka värden man kan spara och vilka operationer man kan utföra
- Exempel:

```
double nbr1;
```

Tilldelningssats

I en ***tilldelningssats*** ger man en variabel ett nytt värde.
Exempel: `nbr1 = 12.5;`

Kan vara ett godtyckligt uttryck

Deklaration och tilldelning kan göras i samma sats. Exempel:

```
double nbr1 = 12.5;
double nbr2 = 0.7;
double sum = nbr1 + nbr2;
```

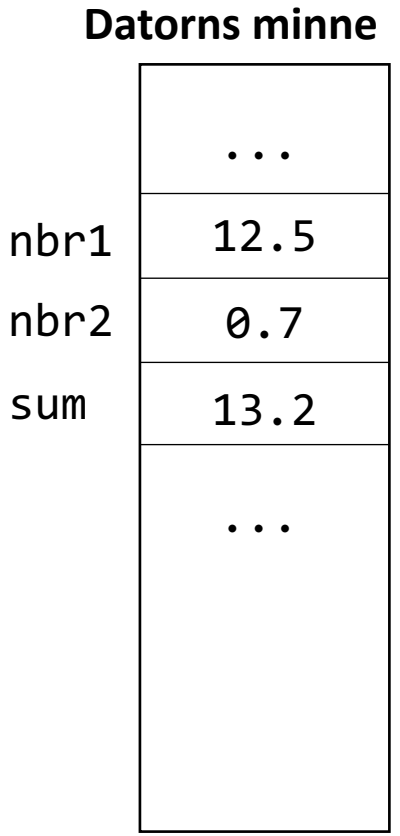
Variabeln och uttrycket måste ha samma typ (några undantag finns)

I Java betyder = tilldelning (ej lika med!)

Variabler i datorns minne

Variablernas värden lagras i datorns *primärminne*. Minnet är uppdelat i olika minnesceller.

- Vid deklaration av en variabel reserveras det plats för variabeln i minnet.
- Vid tilldelning läggs variabelns värde i motsvarande minnescell(er)
- När en variabel används hämtas dess värde från minnet.



Typer

- Varje variabel och varje värde har en ***typ***
  - Exempel: värdet 10 är av typen *heltal*

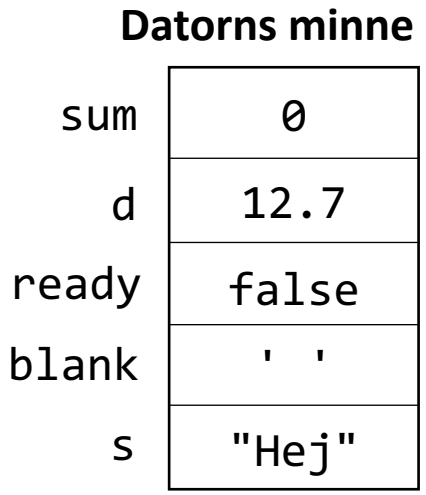
- En typ anger:
  - Vilka värden som är möjliga (exempelvis 1, 2, 3, osv, för heltal)
  - Vilka operationer som man kan utföra och vad de gör

- Exempel: typen anger hur operationen utförs
  - `5 + 5` beräknas till `10` *(addition av heltal)*
  - `"5" + "5"` beräknas till `"55"` *(ihopslagning av strängar)*

Datatyper

Det finns många olika datatyper, exempelvis:

```
int sum = 0;
double d = 12.7;
boolean ready = false;
char blank = ' ';
String s = "Hej";
```



Primitiva datatyper

Primitiva datatyper

- Börjar med liten bokstav
- Innehåller ett enda värde, exempelvis, ett heltal, ett flyttal, ett tecken eller ett logiskt värdet

byte short **int** long	heltal (-128 … 127) heltal (-32768 … 32767) heltal (-2 147 483 648 ... 2 147 483 647) heltal (ca $-9*10^{18}$ ... $+9*10^{18}$)
float **double**	flyttal (tal med decimaldel) flyttal (tal med decimaldel)
boolean	logiskt värde (`true` eller `false`)
char	tecken

Man brukar använda:
`int`, `double`, `boolean`, `char`

Aritmetiska uttryck

Aritmetiska operatorer:

+	addition
-	subtraktion
*	multiplikation
/	division
%	rest vid heltalsdivision / modulo

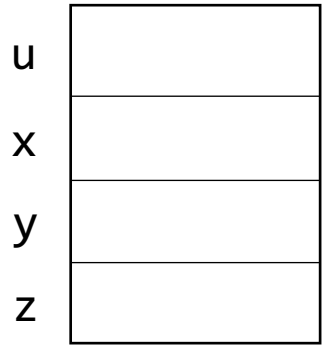
Exempel:

```
2 * (x + y) - 10
```

Övning

Vilka värden har variablerna när följande satser exekverats:

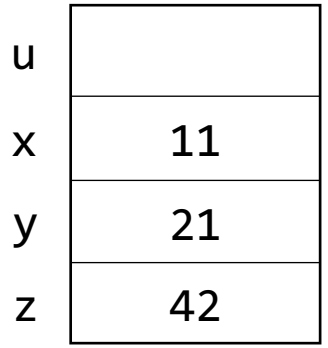
```
int u, x, y, z;
x = 10;
y = 2 * x + 1;
z = (y + x) + (y - x);
x = x + 1;
```



Övning

Vilka värden har variablerna när följande satser exekverats:

```
int u, x, y, z;
x = 10;
y = 2 * x + 1;
z = (y + x) + (y - x);
x = x + 1;
```



Satserna utförs i sekvens och i samma ordning som de är skrivna i (exekvering sker alltså rad för rad)

Heltalsdivision och rest

a / b ger ett heltal som resultat om både a och b är heltal

Exempel:

17 / 3 är lika med 5 (heltalsdivision)

17 % 3 är lika med 2 (rest vid heltalsdivision)

Exempel: 3 personer ska rättvist dela på 17 kakor. Alla får 5 kakor var och 2 kakor blir över.

Övning

Komplettera följande programkod så att variabeln `lastDigit` tilldelas sista siffran i `nbr`.

```
int nbr = 2547;
int lastDigit =
```

Övning

Komplettera följande programkod så att variabeln `lastDigit` tilldelas sista siffran i `nbr`.

```
int nbr = 2547;
int lastDigit = nbr % 10;
```

Typkonvertering mellan `int` och `double`

Ett `int`-värde "får plats" i en variabel av typen `double`:

```
double d = 42;
```

Tvärtom går inte utan konvertering eller avrundning

- Från double till `int`:

```
double d = 42.76;
int a = (int) d;    // a får värdet 42
```

- Avrundning till närmaste heltal:

```
double d = 42.76;
int b = (int) Math.round(d); // avrunda till 43
```

Metoden `round` ger ett resultat av typen `long` som måste konverteras till typen `int`

Enkel miniräknare

```
import java.util.Scanner;

public class Calculator {
  public static void main(String[] args) {
    System.out.println("Skriv in två tal");
    Scanner scan = new Scanner(System.in);
    double nbr1 = scan.nextDouble();
    double nbr2 = scan.nextDouble();
    double sum = nbr1 + nbr2;
    System.out.println("Summan av talen är " + sum);
  }
}
```

Det här programmet läser istället in två godtyckliga flyttal från användaren, beräknar summan av dem och skriver ut den.

Inläsning från användaren

Ett `Scanner`-**objekt** skapas som används för att läsa från tangentbordet:

```
Scanner scan = new Scanner(System.in);
```

Objektet lagras i ***referensvariabeln*** `scan`

Därefter kan man läsa in två flyttal (av typen `double`):

```
double nbr1 = scan.nextDouble();
double nbr2 = scan.nextDouble();
```

Import-satser

Man använder ofta programkod skriven av andra utvecklare. I Java finns det mycket programkod i standardbiblioteket.

I vårt program `Calculator` använder vi den färdiga *klassen* Scanner för inläsning. För att använda klasser i bibliotek behöver vi importera dem:

```
import java.util.Scanner;
```

Utskrift i terminal/konsolfönstret

Utskrift av en teckensträng (omges av citattecken):

```
System.out.println("Skriv två tal");
```

Utskrift av en variabels värde (inga citattecken):

```
System.out.println(sum);
```

Utskrift av flera värden (läggs ihop med +):

```
System.out.println("Summan av talen är " + sum);
```

Algoritm

En ***algoritm*** är en följd av instruktioner som beskriver hur man ska göra för att stegvis lösa ett problem.

Exempel:

- Matrecept
- Beskrivning (på svenska eller pseudokod) hur man löser ett problem
  - Hitta minsta talet bland många tal
  - Hitta talet x bland många tal (sökning)
- Programkod

Exempel på en väldigt enkel algoritm (pseudokod)

nbr1 = läs in tal från användaren
nbr2 = läs in tal från användaren
sum = nbr1 + nbr2
skriv ut sum

Objektorientering

- Java är ett objektorienterat språk, vilket är en teknik att dela upp ett stort program i mindre delar.
- Ett *objekt* beskriver en sak (exempelvis en person) och finns när programmet exekveras. Ett objekt skapas när man skriver `new` och har data och metoder (operationer)
- Vad ett objekt kan göra beskrivs av en *klass* (en mall för objekt).
- Exempel: "`new Scanner(System.in)`" skapar ett objekt som vi kan använda för att läsa från tangentbordet. På objektet anropar vi metoder (`nextDouble()`)
- Vi kommer att återkomma mer i detalj om objekt och klasser vecka 3-4.