

# EDA016 Programmeringsteknik för D

## Läsvecka 1: Introduktion

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

## 1 Introduktion

- Om denna kurs
- Vad är programmering?
- Vårt första Java-program
- Grundläggande programkonstruktioner i Java
  - Sekvens, alternativ, variabler, uttryck och värden
  - Algoritmer, SWAP, repetition (loopar), MIN/MAX
- Sammanfattning
- Meddelande från [Code@LTH](#)

# Om denna kurs

# Vad och hur?

## ■ *Vad* ska du lära dig?

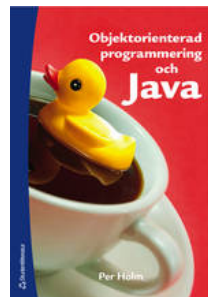
- Grundläggande principer för programmering  
⇒ Inga förkunskaper i programmering krävs
- Konstruktion av enkla algoritmer
- Tänka i abstraktioner
- Imperativ och objektorienterad programmering
- Programspråket Java
- Utvecklingsmiljön Eclipse: implementera, testa, felsöka

## ■ *Hur* ska du lära dig?

- Genom praktiskt eget arbete: **Lära genom att göra!**
- Genom studier av kursens teori: **Skapa förståelse!**
- Genom samarbete med dina kurskamrater: **Gå djupare!**

# Kurslitteratur

- "Objektorienterad programmering och Java" av Per Holm
- Kurskompendium med övningar och laborationer
- Bokpaket säljs på KFS  
John Ericssons väg 4  
<http://www.kfsab.se/>



# Personal

## Kursansvarig:

Björn Regnell, [bjorn.regnell@cs.lth.se](mailto:bjorn.regnell@cs.lth.se)

## Kurssekreterare:

Lena Ohlsson

Exp.tid 09.30 – 11.30 samt 12.45 – 13.30

## Handledare:

Maj Stenmark, Tekn. Lic., Doktorand

Gustav Cedersjö, Doktorand

Anton Klarén, D09

Maria Priisalu , D11

Anders Buhl, D13

Erik Bjäreholt, D13

Fatima Abou Alpha, D13

Cecilia Lindskog, D14

Emma Asklund, D14

# Kursmoment — varför?

- **Föreläsningar:** skapa översikt, ge struktur, förklara teori, svara på frågor, motivera varför
- **Övningar:** bearbeta teorin med avgränsade problem som mestadels löses med papper & penna, förberedelse inför laborationerna
- **Laborationer:** lösa programmeringsproblem praktiskt, obligatoriska uppgifter; lösningar redovisas för handledare
- **Resurstider:** få hjälp med övningar och laborationsförberedelser av handledare
- **Samarbetsgrupper:** grupplärande genom samarbete och dialog
- **Kontrollskrivning:** obligatorisk, diagnostisk, kamraträttad; kan ge samarbetsbonuspoäng till tentan
- **Inlämningsuppgift:** du visar att du kan skapa ett större program självständigt; redovisas för handledare
- **Tenta:** Skriftlig tentamen utan hjälpmedel, förutom **snabbreferens**.

# Nytt för i år

Årets kurs är i flera avseende väsentligt annorlunda and förra årets upplaga, så lita inte på allt som era äldre kursare säger :)

- Övningar blir resurstider i datorsal
- Inlämningsuppgift utan skriftlig rapport
- Samarbetskultur och grupplärande
- Nya övningar
- Nya laborationer
- Nya föreläsningar

Ändringarna är framtagna i samråd med studierådet. Mer om bakgrunden här:

<http://fileadmin.cs.lth.se/cs/Education/EDA016/2015/update.pdf>



# Detta är bara början...

Exempel på efterföljande kurser som bygger vidare på denna:

## ■ Årskurs 1

- Programmeringsteknik – fördjupningskurs
- Utvärdering av programvarusystem
- Diskreta strukturer

## ■ Årskurs 2

- Objektorienterad modellering och design
- Programvaruutveckling i grupp
- Algoritmer, datastrukturer och komplexitet
- Funktionsprogrammering

# Utveckling av mjukvara i praktiken

- **Inte bara kodning:** kravbeslut, releaseplanering, design, test, versionshantering, kontinuerlig integration, driftsättning, återkoppling från dagens användare, ekonomi & investering, gissa om morgondagens användare, ...
- **Teamwork:** Inte ensamma hjältar utan autonoma team i decentraliserade organisationer med innovationsuppdrag
- **Snabbhet:** Att koda innebär att hela tiden uppfinna nya "byggstenar" som ökar organisationens förmåga att snabbt skapa värde med hjälp av mjukvara. Öppen källkod. Skapa kraftfulla API.
- **Livslångt lärande:** Lär nytt och dela med dig hela tiden. Exempel på pedagogisk utmaning: hjälp andra förstå och använda ditt API  $\implies$  *Samarbetskultur*

# Att skapa koden som styr världen

## I stort sett alla delar av samhället styrs av mjukvara:

- kommunikation
- transport
- byggsektorn
- statsförvaltning
- finanssektorn
- media
- sjukvård
- övervakning
- integritet
- upphovsrätt
- miljö & energi
- sociala relationer
- utbildning

## Hur blir ditt framtida yrkesliv som systemutvecklare?

- Redan nu är det en skriande brist på utvecklare och bristen blir bara värre och värre...  
CS 2015-08-17
- Global kompetensmarknad  
CS 2015-06-14  
CS 2015-08-15
- Fokus på innovation, tid-till-marknad
- Autonoma utvecklingsteam i decentraliserade organisationer
- Öppen källkod

# Registrering

- Fyll i listan som jag skickar runt.
- Kryssa i kolumnen **Ska gå** om du ska gå kursen<sup>1</sup>
- Kryssa i kolumnen **Programmerat** om du har programmerat innan (även om bara lite grand)
- Kryssa i kolumnen **Kursombud** om du kan tänka dig att vara kursombud under kursens gång
  - Alla LTH-kurser ska utvärderas under kursens gång och efter kursens slut.
  - Till det behövs kursombud – ungefär 2 D-are och 2 W-are.
  - Ni kommer att bli kontaktade av studierådet.

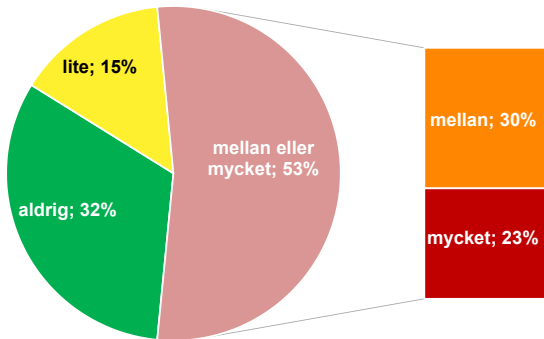
---

<sup>1</sup>Redan gått motsvarande högskolekurs? Ansök då om tillgodoräknande

# Förkunskaper

- Förkunskaper  $\neq$  Förmåga
- Varken kompetens eller personliga egenskaper är statiska
- "Programmeringskompetens" är inte *en* enda enkel förmåga utan en komplex sammansättning av flera olika förmågor som utvecklas genom hela livet
- Ett innovativt utvecklarteam behöver många olika kompetenser för att vara framgångsrikt

# Stor spridning i programmeringsförkunskaper bland D-are (enl. enkätsvar 2010-2014)



# Förkunskapsenkät

Fyll i denna enkät på rasten (eller senast ikväll kl 20:00)

<http://cs.lth.se/eda016/survey>

Ligger till grund för randomiserad gruppindelning,  
så att det blir en spridning av förkunskaper inom gruppen.

# Samarbetsgrupper

- Ni delas in i **samarbetsgrupper** om ca 5 personer baserat på förkunskapsenkäten, så att olika förkunskapsnivåer sammanförs
- 2 av de 11 laborationerna är mer omfattande **grupplabbar** och kommer att göras i samarbetsgrupperna (Lab 9 och 11 i Lp2)
- Kontrollskrivningen i halvtid kan ge **samarbetsbonus** (max 3p) som adderas till ordinarie tentans poäng (max 45p) med medelvärdet av gruppmedlemmarnas individuella kontrollskrivningspoäng

Bonus  $b$  för varje person i en grupp med  $n$  medlemmar med  $p_i$  poäng vardera på kontrollskrivningen:

$$b = \sum_{i=1}^n \frac{p_i}{n}$$



# Varför samarbetsgrupper?

Huvudsyfte: **Bra lärande!**

- Pedagogisk forskning stödjer tesen att lärandet blir mer djupinriktat om det sker i utbyte med andra
- Ett studiesammanhang med höga ambitioner och respektfull gemenskap gör att vi **når mycket längre**
- Varför ska du som redan kan mycket aktivt dela med dig av dina kunskaper?
  - Förstå bättre själv genom att förklara för andra
  - Träna din pedagogiska förmåga
  - Förbered dig för inför ditt kommande yrkesliv som mjukvaruutvecklare

# Samarbetskontrakt

Gör ett skriftligt samarbetskontrakt med dessa och ev. andra punkter som ni också tycker bör ingå:

- 1 Kom i tid till gruppmöten
- 2 Var väl förberedd genom självstudier inför gruppmöten
- 3 Hjälp varandra att förstå, men ta inte över och lös allt
- 4 Ha ett respektfullt bemötande även om ni har olika åsikter
- 5 Inkludera alla i gemenskapen

Diskutera hur ni ska uppfylla dessa innan alla skriver på.

Ta med samarbetskontraktet och visa för handledare på labb 1.

**Om arbetet i samarbetsgruppen inte fungerar ska ni mejla kursansvarig och boka mötestid!**

# Bestraffa inte frågor!

- Det finns bättre och sämre frågor vad gäller hur mycket man kan lära sig av svaret, men **all undran är en chans** att i dialog utbyta erfarenheter och lärande
- Den som frågar **vill veta** och berättar genom frågan något om nuvarande kunskapsläge
- Den som svarar får chansen att **reflektera** över vad som kan vara svårt och olika vägar till djupare förståelse
- I en hälsosam lärandemiljö är det **helt tryggt** att visa att man ännu inte förstår, att man gjort "fel", att man har mer att lära, etc.
- Det är viktigt att våga försöka även om det blir "fel":  
**det är ju då man lär sig!**

# Plagiatregler

Läs dessa regler noga och diskutera i samarbetsgrupperna:

- <http://cs.lth.se/utbildning/samarbete-eller-fusk/>
- <http://cs.lth.se/utbildning/foereskrifter-angaaende-obligatoriska-moment/>

Ni ska lära er genom **eget arbete** och genom **bra samarbete**. Samarbete gör att man lär sig bättre, men man lär sig inte av att bara kopiera andras lösningar. Plagiering är förbjuden och kan medföra disciplinärende och avstängning.

# En typisk kursvecka

- Gå på föreläsningar måndag–tisdag
- Jobba själv med boken, övningar, labbförberedelser
- Träffas i samlingsgruppen och hjälp varandra att förstå mer och fördjupa lärandet
- Gå till resurstider och få hjälp och tips av handledare, onsdag–torsdag
- Genomför obligatorisk laboration på fredagen

Se detaljerna och undantagen i [schemat i TimeEdit](#)

# Laborationer

- **Programmering lär man sig bäst genom att programmera...**
- Labbarna är **individuella** (utom 2) och **obligatoriska**
- Gör övningarna och labbförberedelserna noga *innan* själva labben – detta är ofta helt nödvändigt för att du ska hinna klart. Dina labbförberedelserna kontrolleras av handledare under labben.
- Är du sjuk? Anmäl det *före* labben till [bjorn.regnell@cs.lth.se](mailto:bjorn.regnell@cs.lth.se), få hjälp på resurstillfällena och redovisa på resurstid eller uppsamlingstid (helst inte på ordinarie tillfälle, men det är OK om handledaren har tid)
- Hinner du inte med hela? Se till att handledaren noterar din närvaro, och fortsatt på resurstid och uppsamlingstid.
- Läs noga anvisningarna i kompendiet
- Laborationstiderna är gruppindelade enligt **schemat**. Du ska gå till den tid och den sal som motsvarar din grupp.

# Resurstider

- På resurstiderna får du hjälp med övningar och laborationsförberedelser
- Kom till minst en resurstid per vecka (se [schema](#))
- Handledare gör ibland genomgångar för alla under resurstiderna
- Resurstiderna är inte gruppindelade i schemat. Du får i mån av plats gå på flera resurstider per vecka. Om det blir fullt i ett rum prioriteras dessa grupper för att minimera schemakrockar:

Tid Lp1	Sal	Grupper med prio
Ons 10-12 v1-7	Hacke	09 & 10
Ons 13-15 v1-7	Hacke	07 & 08
Ons 15-17 v1-7	Panter	05 & 06
Ons 15-17 v1-7	Val	03 & 04
Tor 13-15 v1-7	Mars	01 & 02
Tor 15-17 v1-7	Mars	11 & 12

# Att göra i Vecka 1: Rivstarta

- 1 Senast måndag 31/8 kl 20:00, fyll i <http://cs.lth.se/eda016/survey>
- 2 Läs följande kapitel i kursboken:  
1, 3.1-3.3, 5.1-5.3, 6.1-6.2, 7.1-7.3, 7.5-7.6, 7.8-7.9
- 3 Gör övning 1: Hello World, Hello Args, javac, editera—kompilera—exekvera—felsök, värden, uttryck variabler och tilldelning
- 4 Gör förberedelserna till laboration 1: skapa textfil, etc.
- 5 Träffas i samarbetsgrupper och hjälp varandra att förstå
- 6 Kom till **resurstiderna** och få hjälp och tips
- 7 Genomför laboration 1: Quiz — träna på att editera, läsa, ändra och felsöka i färdig programkod



# Vad är programmering?

# Programming unplugged: Två frivilliga?



# Editara och exekvera ett program



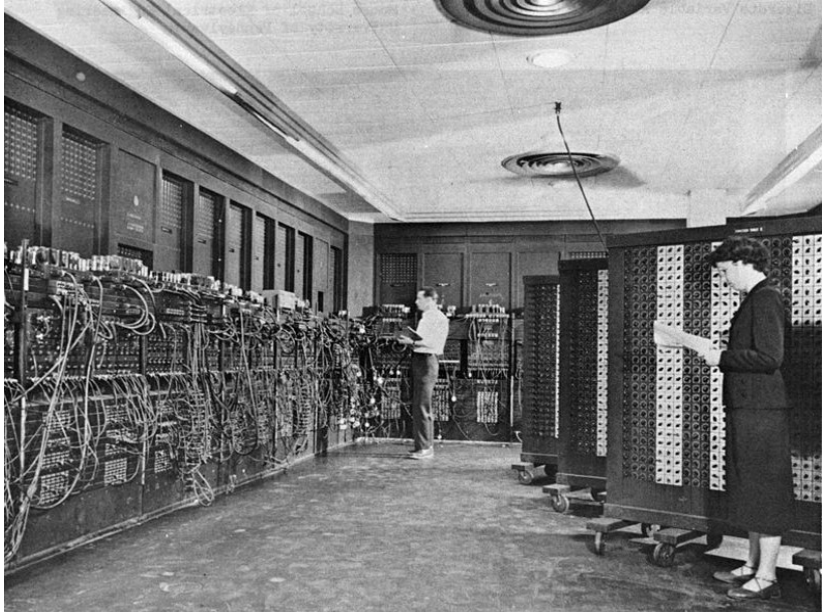
# Världens första programmerare



Ada Lovelace skrev världens första datorprogram på 1800-talet.

Programmet skulle köra på en kugghjulsdator som hennes kompis Charles Babbage försökte bygga.

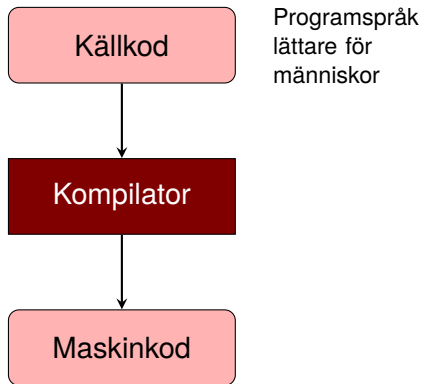
Vad är en dator?



# Vad är en kompilator?



Grace Hopper uppfann första kompilatorn 1952.



Vad finns det för programspråk?

# Exempel på vanliga programspråk

- 1 Java
- 2 C
- 3 C++
- 4 C#
- 5 Python
- 6 Objective-C
- 7 PHP
- 8 Visual Basic .NET
- 9 JavaScript
- 10 Perl

De 10 "vanligaste"?

- TIOBE Programming Community Index  
Augusti 2015
- Språktrender på GitHub  
2008-2015

# Vad är Java?

Java är egentligen flera saker:

- Ett populärt programspråk med en snabb kompilator
- Java Virtual Machine (JVM) som ger plattformsoberoende
- Java Development Kit (JDK) – massor med färdig kod



# Var finns Java?



# Varför Java?

- Java är ett av planeten jordens mest framgångsrika programspråk under de senaste decenniet
- Många inbyggda system använder Java (bilar, tvättmaskiner, kassa-apparater, nätverksutrustning, ...)
- Android som bygger på Java har ca 80% av smartphonemarknaden
- Java används som nybörjarspråk på en stor del av planetens tekniska högskolor
- Java utvecklas ständigt: Java 8 kombinerar nu objektorientering med funktionsprogrammering

Olika språk är bra på olika saker och en systemutvecklare **behöver kunna många olika språk**

# Utvecklingsverktyg

Systemutvecklare använder många olika verktyg:

- kompilator, t.ex. javac
- editor, t.ex. gedit, Sublime Text 3, ...
- terminal och kommandoskal, t.ex. bash, powershell
- integrerad utvecklingsmiljö (eng. Integrated Dev. Environment, IDE), t.ex. **Eclipse**, IntelliJ, ...
- versionshanteringssystem, t.ex. git, Mercurial, ...
- kodlagringsplats, t.ex. GitHub, Bitbucket, ...
- hypervisor för virtuella maskiner, t.ex. VirtualBox, VMWare, ...
- bughanteringssystem, t.ex. Bugzilla, Jira, ...
- byggverktyg, t.ex. Jenkins, Hudson, ...
- ...

# Vad är objektorientering?

- Det finns många olika **programmeringsparadigm** (sätt att programmera på), till exempel:
  - **imperativ programmering**: programmet är uppbyggt av sekvenser av olika satser som påverkar systemets tillstånd
  - **objektorienterad programmering**: en sorts imperativ programmering där programmet består av objekt som sammanför data och operationer på dessa data
  - **funktionsprogrammering**: programmet är uppbyggt av samverkande (matematiska) funktioner som undviker föränderlig data och tillståndsändringar
  - **deklarativ programmering, logikprogrammering**: programmet är uppbyggt av logiska uttryck som beskriver olika fakta eller villkor och exekveringen utgörs av en bevisprocedur som söker efter värden som uppfyller fakta och villkor

# Grundläggande principer i imperativ programmering

- **Sekvens:** Ett program innehåller sekvenser av *satser*. Ordningen mellan dessa har helt avgörande betydelse.
- **Alternativ:** Systemet reagerar på vad som händer och kan välja olika vägar genom programmet beroende på *variablers* värde  
Java: **if**-sats, **switch**-sats
- **Repetition:** Göra saker om och om igen  
Java: **while**-loop, **for**-loop
- **Abstraktion:** Kapsla in (komplexa) programdelar och sätta namn på dessa så att de enkelt går att återanvända utan att vi behöver "rota i inandömet".  
Java: klasser och metoder

# Hello World!

Vårt första Java-program i filen HelloWorld.java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hej och välkomna!");  
    }  
}
```

Kompilera och kör:

```
> javac HelloWorld.java  
> java HelloWorld  
Hej och välkomna!  
>
```

Ovan ingår i övning 1.

# Hello World! – Vad betyder egentligen allt detta?

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hej och välkomna!");  
    }  
}
```

- **public** Denna programdel är synlig "utåt" och kan användas av andra delar.
- **class** Ett slags "kodbyggblock" som samlar olika programdelar. All java-kod måste finnas i en klass. Det finns tusentals färdiga klasser att använda direkt i Java och man kan lätt skapa egna klasser. Klammerpar { } anger början och slut.
- **static** Denna programdel skapas direkt vid programmets start och det finns exakt *en* sådan här per klass.
- **void** Berättar för kompilatorn att inget värde returneras från denna programdel.
- **main** Berättar var exekveringen av programmet börjar.
- **( )** Parentespar berättar för kompilatorn att vi här kan ha parametrar.
- **String[] args** Möjliggör indata till programmet i form av flera textsträngar. Parametern args måste finnas i main, men vi använder den inte i detta program.
- **System.out.println** Den färdiga klassen System kan bl.a. skriva ut text. Textsträngar avgränsas av citationstecken. Semikolon avgränsar satser.

# Några grundläggande delar i ett Javaprogram

- värde (value): data som programmet kan använda  
42      "hej"      42.0      **true**
- uttryck (expression): data kombineras med operatorer och ger nya värden  
41+1    "h"+"ej"    43.5-1.5    **!false**
- deklaration av variabel (variable declaration): skapa plats i minnet för data  
**int** x = 42;
- tilldelningssats (assignment): ändra värdet på variabler  
x = 43;
- alternativ (choice): välj väg beroende på variablers värde  
**if, switch**
- repetition (loop): upprepa om och om igen  
**while, for**



# Värden och uttryck

```
public class Expressions {  
    public static void main(String[] args){  
  
        System.out.println( 42      ) // Heltalsvärde  
        System.out.println( 42.0    ) // Decimaltalsvärde  
        System.out.println( "Hej"   ) // Strängvärde  
        System.out.println( true    ) // Booleskt värde  
  
        System.out.println( 41 + 1 ) // Heltalsuttryck  
        System.out.println( 41.0 + 1.0 ) // Decimaltalsuttryck  
        System.out.println( "Hej" + "san!" ) // Stränguttryck  
        System.out.println( true && false ) // Booleskt uttryck  
  
    }  
}
```

# Alternativ

Välj väg genom programmet med **if**-sats.

```
public class Alternativ {  
    public static void main(String[] args){  
  
        if (true) {  
            System.out.println("Sant!");  
        } else {  
            System.out.println("Falskt!");  
        }  
  
    }  
}
```

En if-sats gör så att exekveringen av programmet kan delas upp i olika grenar; vilken gren som görs beror värdet av ett villkorsuttrycket: **true** eller **false**

# Alternativ med variabel

Det blir roligare om vi har en variabel:

```
public class AlternativeWithVariable {  
    public static void main(String[] args){  
  
        int x = 42;  
        if (x >= 42) {  
            System.out.println("Sant!");  
        } else {  
            System.out.println("Falskt!");  
        }  
    }  
}
```

# Alternativ med variabel som kan ändra sig

Det blir ännu roligare om vi har en variabel som kan anta olika värden beroende på vad som händer under exekveringens gång:

```
import java.util.Scanner;

public class AlternativeWithVariableThatCanChange {

    public static void main(String[] args){

        Scanner scan = new Scanner(System.in);
        System.out.print("Skriv heltal: ");
        int x = scan.nextInt();

        if (x == 42) { // OBS! dubbla likhetstecken
            System.out.println("Sant!");
        } else {
            System.out.println("Falskt!");
        }

    }

}
```

# Vad är egentligen en variabel?

- En variabel har ett **namn** och kan lagra ett **värde** av en viss **typ**
- Variabler måste **deklareras** och då får kompilatorn reda på vilket namnet är och vilken typ av värden som variabeln kan lagra:

```
int x;
```

- När variabler deklareras är det oftast bäst att direkt ge dem ett initialvärde:

```
int x = 42;
```

- En variabeldeklaration medför att plats i datorns minne reserveras.

Vi ritar detta såhär:

x	42
---	----

Dessa deklarationer...

```
int x = 42;  
int y = x + 1;
```

... ger detta innehåll någonstans i minnet:

x	42
y	43

# Syntaxregler för namn i Java

När kompilatorn "läser" <sup>2</sup> koden och försöker hitta variabelnamn, antar den att du följer de entydiga syntaktiska reglerna för språket.

För namn i Java gäller följande regler:

- Namn får inte vara **reserverade ord**
- Stora och små bokstäver spelar roll (eng. *case sensitive*)  
**int** highScore; och **int** highscore; ger alltså två *olika* variabler
- Namnet måste börja med en bokstav, ett understreck `_` eller ett dollartecken `$`
- Namn får *inte* innehålla blanktecken
- Namn får innehålla bokstäver, siffror, understreck `_` och dollartecken `$`, men *inte* andra specialtecken (alltså inte `%&@!{ } / + - * etc.`)

---

<sup>2</sup>man säger ofta "parsa" i stället för "läsa" när kompilatorn tolkar koden

# Vad händer egentligen vid en tilldelning?

- Med en **tilldelningssats** kan vi ge en tidigare deklarerad variabel ett nytt värde:

```
x = 1;
```

- Det gamla värdet försvinner för alltid och det nya värdet lagras istället:

x 1

- Likhetstecknet används alltså för att *ändra* variablers värden och det är ju *inte* samma sak som matematisk likhet<sup>3</sup>. Vi kan till exempel skriva denna tilldelningssats:

```
x = x + 1;    //Vad händer här?
```

---

<sup>3</sup>Arv från C, Fortran mfl. I **andra språk** används t.ex. `x := 42` eller `x <- 42`

# Övning: Tilldelningar i sekvens

Rita hur minnet ser ut  
efter varje rad nedan:

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

En variabel som inte (ännu) inte initierats har  
ett odefinierat värde.

	rad 1	rad 2	rad 3	rad 4	rad 5
u	<input data-bbox="666 498 749 553" type="text" value="?"/>	<input data-bbox="790 498 872 553" type="text"/>	<input data-bbox="913 498 996 553" type="text"/>	<input data-bbox="1037 498 1119 553" type="text"/>	<input data-bbox="1160 498 1243 553" type="text"/>
x	<input data-bbox="666 584 749 639" type="text" value="?"/>	<input data-bbox="790 584 872 639" type="text"/>	<input data-bbox="913 584 996 639" type="text"/>	<input data-bbox="1037 584 1119 639" type="text"/>	<input data-bbox="1160 584 1243 639" type="text"/>
y	<input data-bbox="666 670 749 725" type="text" value="?"/>	<input data-bbox="790 670 872 725" type="text"/>	<input data-bbox="913 670 996 725" type="text"/>	<input data-bbox="1037 670 1119 725" type="text"/>	<input data-bbox="1160 670 1243 725" type="text"/>
z	<input data-bbox="666 756 749 811" type="text" value="?"/>	<input data-bbox="790 756 872 811" type="text"/>	<input data-bbox="913 756 996 811" type="text"/>	<input data-bbox="1037 756 1119 811" type="text"/>	<input data-bbox="1160 756 1243 811" type="text"/>



# Vad är en algoritm?

En **algoritm** är en sekvens av instruktioner som beskriver hur man löser ett problem

Exempel:  
matrecept,  
uppdatera highscore i ett spel,  
...

# Algorithm-exempel: Highscore

**Problem:** Uppdatera high-score i ett spel

**Varför?** Så att de som spelar uppmuntras att spela mer :)

**Algorithm:**

- 1 *points*  $\leftarrow$  poängen efter senaste spelet
- 2 *highscore*  $\leftarrow$  bästa resultatet innan senaste spelet
- 3 **om** *points* är större än *highscore*  
    Skriv "Försök igen!"  
**annars**  
    Skriv "Grattis!"

# Algorithm-exempel: Highscore

```
1  import java.util.Scanner;
2
3  public class HighScore {
4      public static void main(String[] args){
5          Scanner scan = new Scanner(System.in);
6          System.out.println("Hur många poäng fick du?");
7          int points = scan.nextInt();
8          System.out.println("Vad var higscore före senaste spelet?");
9          int highscore = scan.nextInt();
10         if (points > highscore) {
11             System.out.println("GRATTIS!");
12         } else {
13             System.out.println("Försök igen!");
14         }
15     }
16 }
```

Det finns en bugg i denna implementation. Vilken?  
Fanns buggen redan i algoritmdesignen?

# Abstraktion – varför?

- 1 Dela upp problem i delproblem
- 2 Skapa ”byggblock” av kod som kan återanvändas
- 3 Dölja komplexiteten i lösningar
- 4 Abstraktion är själva **essensen i all programmering**

```
public static void main(String[] args){  
    askUser();  
    updateHighscore();  
}
```

# Vår första algoritmkluring: SWAP

```
1  import java.util.Scanner;
2
3  public class HighScoreAbstraction {
4
5      public static int points = 0;
6      public static int highscore = 0;
7      public static Scanner scan = new Scanner(System.in);
8
9      public static void askUser(){
10         System.out.println("Hur många poäng fick du?");
11         points = scan.nextInt();
12         System.out.println("Vad var higscore före senaste spelet?");
13         highscore = scan.nextInt();
14     }
15
16     public static void updateHighscore(){
17         if (points >= highscore) {
18             System.out.println("GRATTIS!");
19         } else {
20             System.out.println("Försök igen!");
21         }
22     }
23 }
```

# Vår första algoritmkluring: SWAP

**Problem:** läs in och byt plats på två tal i minnet

## Algoritm:

- 1 skapa en Scanner
- 2 läs in x
- 3 läs in y
- 4 Skriv ut x och y
- 5 byt plats på värdena mellan x och y
- 6 Skriv ut x och y

Varför kan det vara bra att kunna byta plats på olika värden?

Steg 5 är egentligen inte så lätt som det verkar och behöver göras i flera steg

# Vår första algoritmkluring: SWAP

```
1  import java.util.Scanner;
2
3  public class SwapQuest {
4      public static void main(String[] args){
5          //Steg 1: skapa en Scanner
6          Scanner scan = new Scanner(System.in);
7
8          int x = scan.nextInt(); //Steg 2: läs in x
9          int y = scan.nextInt(); //Steg 3: läs in y
10
11         //Steg 4: Skriv ut x och y
12         System.out.println("x: " + x + " y: " + y);
13
14         //Steg 5: byt plats på värdena mellan x och y  HUR???
15         // ... skriv SWAP-satser här ...
16         //Steg 6: Skriv ut x och y
17         System.out.println("x: " + x + " y: " + y);
18     }
19 }
```

# Vår första algoritmkluring: SWAP

```
1  import java.util.Scanner;
2
3  public class SwapSolution {
4      public static void main(String[] args){
5          Scanner scan = new Scanner(System.in);
6
7          int x = scan.nextInt();
8          int y = scan.nextInt();
9
10         System.out.println("x: " + x + " y: " + y);
11
12         int temp = x;
13         x = y;
14         y = temp;
15
16         System.out.println("x: " + x + " y: " + y);
17     }
18 }
```

Övning: Rita hur minnet ser ut efter respektive raderna 7, 8, 12, 13, 14



# Repetition med **while**-sats

- En av de saker en dator är *extra* bra på är att göra samma sak om och om igen utan att tröttna! <sup>4</sup>
- Med klockfrekvens i storleksordningen  $10^9$  Hz är det ganska många instruktioner som kan göras per sekund...

---

<sup>4</sup>Vilket ju människor är *extra* dåliga på :)

# Meddelande från **Code@LTH**