ex1compiler

Joachim von Hacht

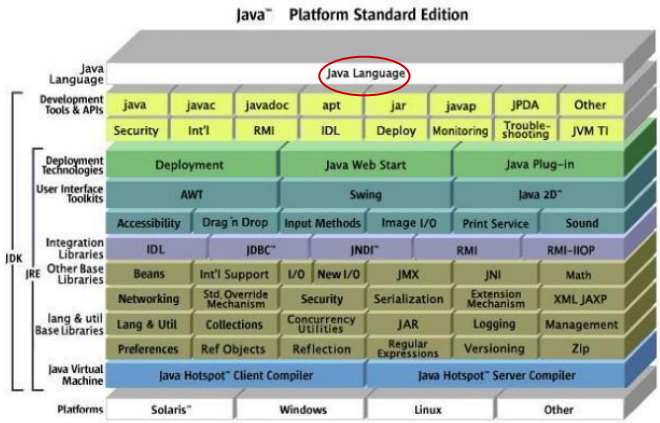
Språket Java

Språket Java beskrivs i en specifikation
- Vi använder versionerna 11 eller 13 av språket.
- Specifikationen beskriver hur vi skall skriva språket Java (syntax) och …
- … betydelsen av det vi skriver (semantik).

OBS! Att det vi går igenom i kursen gäller för många (de flesta) imperativa och/eller objektorienterade språken.
- Kursen är alltså inte en Java kurs.

För att kunna utveckla program i Java på din dator måste du installera Java Development Kit förkortat JDK (Java Runtime Environment, JRE som brukar finnas på de flesta datorer räcker inte)
- JDK:n innehåller en massa "verktygsprogram" som behövs/kan användas vid programutveckling framförallt finns en kompilator (mer strax).
- Popularitet för olika språk

Java-Plattformen

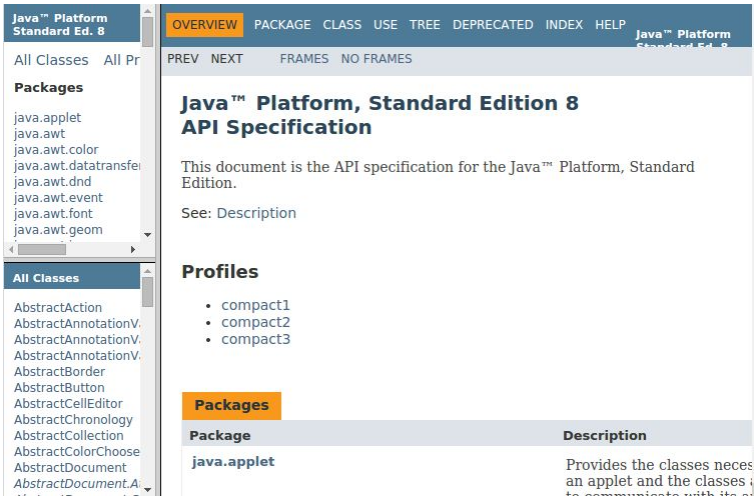


3

Java är en **plattform**

- Dels finns själva språket Java som är en ganska liten del
- Utöver detta finns bl.a en mängd **standardbiblioteket (libraries)**
  - Biblioteken kallas ofta för API:er (**Application Programming Interface**, programmeringsgränssnitt)
  - Ett API innehåller körbar kod som vi kan använda i vårt program (så slipper programmerare skriva samma sak om och om igen)
    - I vår kod betyder det att vi använder färdiga objekt, mer kommer ...
  - Standardbiblioteket är organiserat i olika grupper utefter ändamål
  - För att använda standard API:er skriver man t.ex. i programmet : import java.utils.List …. mer kommer ...
- Java-program körs i en speciell exekveringsmiljö, en "virtuell" dator, mer strax ...
- Java-plattformen = språket + standardbiblioteken + exekveringsmiljön

Java-Dokumentation



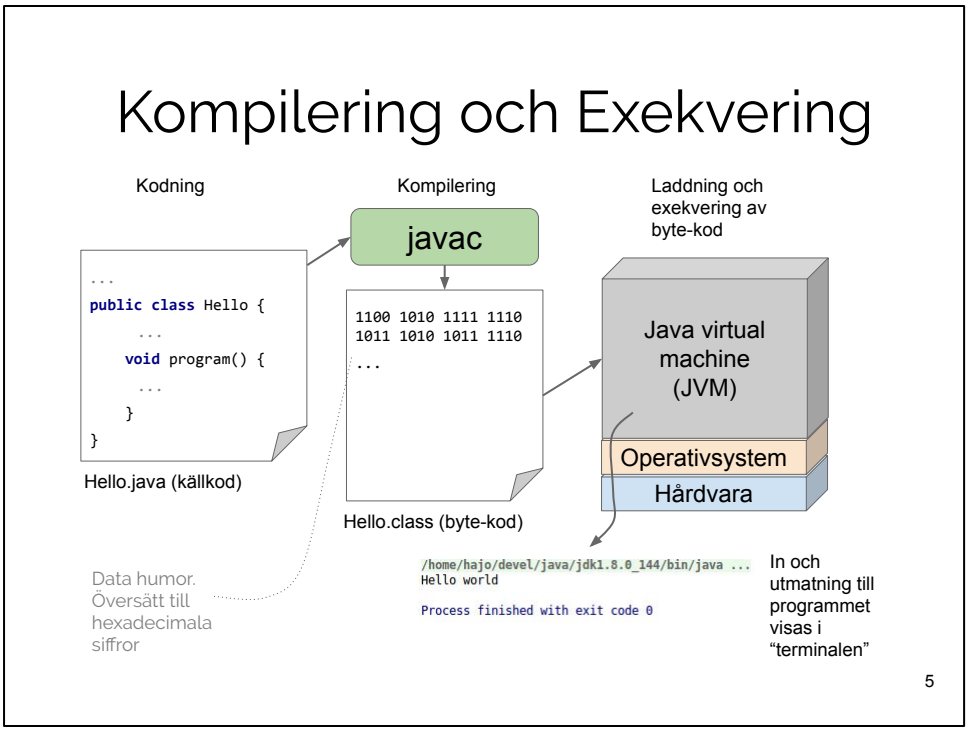
4

En svårighet med moderna språk är att de innehåller enorma standardbibliotek

- Förutom själva språket måste man lära sig hitta bland allt färdigt.
- I denna kurs används bara en mycket liten del … (inga större problem, absolut inget man behöver lära sig utantill)

Modern programutveckling innebär bl.a.

- Att man söker igenom Web:en efter bibliotek (standard eller andra s.k. tredjepartsbibliotek) med lämplig kod för uppgiften.
- Att man därefter sammanfogar den funna koden med “så lite som möjligt” av egen kod.
    - Mer om detta i senare kurser
    - I denna kurs skriver vi mycket själva för att lära oss.



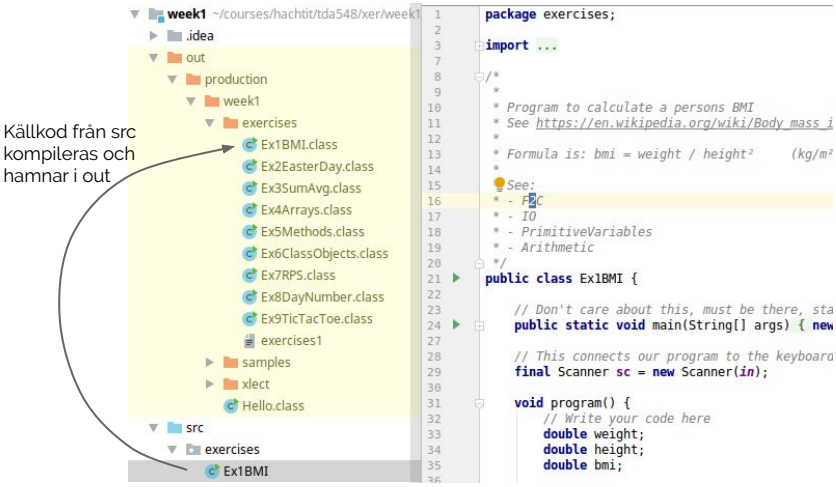
Innan ett Java-program kan exekveras (köras) måste det **kompileras (översättas)**

- Ett program kallat javac (= java compiler, en kompilator) som finns med i JDK:n översätter källkoden till en binärfil med byte-kod (byte)
- Byte-koden sparas i en ny fil, en class-fil som heter samma som källkoden men med suffixet .class istf .java. Vi har efter kompileringen två filer
- Det är class-filen som kommer att exekveras
    - Har vi källkoden kan vi alltid skapa en ny class-fil, det är källkoden som är viktig!

Ett kompilerat program (byte-koden) körs på en virtuell dator (Java Virtual Machine, JVM).

- Detta gör att Java-program utan någon modifikation kan köras på flera olika operativsystem (eftersom JVM:en eliminerar skillnader mellan olika operativsystem
    - Man säger att Java är **plattformsoberoende** (många andra språk måste kompileras om för varje operativsystem)
    - Dock måste din dator ha en JRE för att det skall fungera.

Kompilering och Exekvering i IntelliJ



6

Kompileringen sköts automatiskt av IntelliJ (i bakgrunden) så fort källkoden ändras.

- class-filerna hamnar i out-mappen (vi måste ange för IntelliJ vilken mapp den skall spara class-filerna i (File > Project Structure)
- Om problem: Kan vara bra att ta bort hela mappen production/ i out.

Dekompilator

- Om man klickar på en .class-fil så visas konstigt nog något som är väldigt likt källkoden (men den går inte att ändra).
- Beror på att IntelliJ har en dekompilator som helt enkelt översätter byte-koden tillbaks till läsbar källkod (en del försvinner).

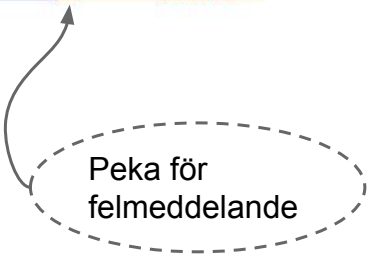
IntelliJ inspections: Innebär att förutom kompilering undersöker IntelliJ koden för att
hitta ev. tveksamheter som inte är kompileringsfel (onödig kod, risk för körningsfel , m.m.).

- IntelliJ mängder med inspektioner.

Kompileringsfel: Syntax

```
// Syntax error

void program() {
out.println(Hello world");
}
```



7

Vid översättningen kontroller kompilatorn programmet på en mängd olika sätt, t.ex. syntaxen (= rättstavning, korrekt ordföljd).

- Bryter vi mot syntaxreglerna för vi ett **kompileringsfel** (syntax error)
- Visas med röd understrykning i IntelliJ (peka på markering så kanske något tips visas)
- Ett program med kompileringsfel går inte att köra eftersom ingen bytekod har skapats.

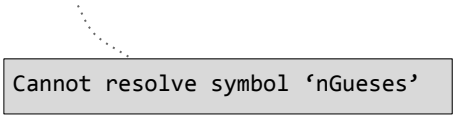
Ett fel kan generera följdfel!

- Rätta fel från upp/vänster till ner/höger

Namn

```
// Declaration
int nGuesses = 0;

// Can't find any declaration for name?
// So compile error (bad spelling).
out.println(nGueses);
```



8

För att något skall räknas som ett namn i Java måste det vara deklarerat.

- Deklarerat av oss eller någon annan.
  - I bilden har vi deklarerat namnet nGuesses (namn på en variabel)
  - out och println deklarerat av någon annan.
- Anger vi något namn som inte är deklarerat får vi ett kompileringsfel (t.ex. om vi stavar fel).
- Namn kallas Identifierare i programmeringssammanhang.

NOTERA: En variabeldeklaration ger en variabel!

- FInns inget sätt att deklarera flera variabler på samma gång.

Synlighetsområde

```
{
 int i = 0;

}

// Same name, but other
// scope, OK!
{
 int i = 2;

}
```

```
{

 int i = 0;

    {
        int i = 2; // Name clash!
        int j = 3;
    }


    j = 4; // Not visible
}
```

9

Synlighetsområdet (scope) anger var i programmet det är möjligt att använda ett namn t.ex. ett variabelnamn

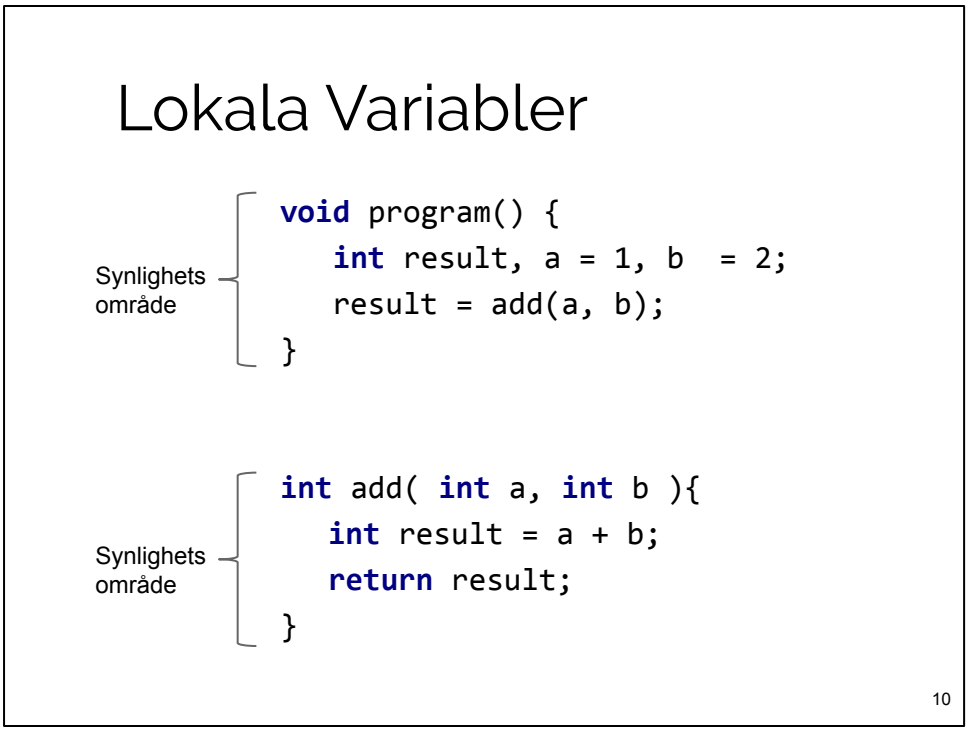
- I Java sammanfaller synlighetsområde och block (förenklat)

Synlighetsområde för variabler:

- Variabler är bara synliga (kan bara användas) i det synlighetsområde de är deklarerade (fr.o.m. deklarationen och vidare), förutom …
  - … nästlade synlighetsområden
    - Ett inre block kommer åt variabler i ett yttre som är deklarerade innan det inre blocket
    - Yttre block kommer inte åt variabler i ett inre

I Java gäller att variabler inte får ha samma namn inom samma synlighetsområde

- Inom olika synlighetsområden kan samma namn användas
  - Praktiskt: Slipper hitta på nya namn hela tiden!



Variabler deklarerade i metoder kallas **lokala variabler**

- Vi räknar även parametrar som lokala variabler
- Synlighetsområdet är metodkroppen (ett block)
- Lokala variabler måste ges ett värde (initieras eller tilldelas) innan de används, om ej kompileringsfel

Bilden:

- "result" i koden ovan syftar på två olika variabler med samma namn, den ena i program() den andra i add()
- På samma sätt med a och b.
- Eftersom de finns i olika synlighetsområden kan vi använda samma namn!
- I exemplet finns totalt 6 variabler (2 st a, 2 st b och 2 st result)

Anm: I program() ovan deklarerar vi flera variabler på samma rad

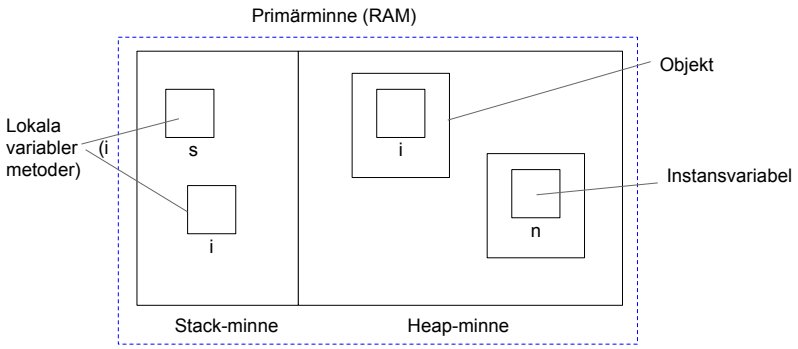
- Möjligt att göra men inte så bra, bättre med en/rad (görs av utrymmesskäl här)

Summa:

- Samma variabelnamn inom olika synlighetsområden syftar på på olika variabler
- Olika variabelnamn (ev inom olika synlighetsområden) kan syfta på

- samma sak, om referenser är inblandade, mer senare.

Variabler i Minnet



11

Alla variabler finns i minnet.
Lokala variabler

- Finns på stacken.
- Stacken växer vid metodanrop och nya variabler skapas.
- När metoden är klar återlämnas minnet och variablerna försvinner.
- Dvs. lokala variabler har en livslängd.
- Stacken har en maximal storlek (StackOveflowError)

Instansvariabler tillhör objekt.

- Instansvariablerna finns så länge objektet existerar
- Objekt skapas på en plats i minnet kallat heap:en
- Heap minnet är betydligt större än stackminnet (många GB)

Objektet existerar så länge det finns en referens till det. Mer om detta strax.

- Finns inga referenser alls till objektet kommer det att skräpsamlas (d.v.s. raderas ut minnet)

Kompilatorn och Deklarationer

```
public class M2DayNumber {
   ...
   final Scanner sc = new Scanner(in);

   void program() {
       out.print("Input the year > ");
       int year = sc.nextInt();
       out.print("Input the month number > ");
       int month = sc.nextInt();
       out.print("Input the day number > ");
       int day = sc.nextInt();
       int dayNbr = getDayNbr(year, month, day);
       printResult(year, month, day, dayNbr);
   }

   int getDayNbr(int year, int month, int day) {
       int n = sumToMonth(month - 1) + day;
       if (month > 2 && isLeapYear(year)) {
           n++;
       }
       return n;
   }

   int sumToMonth(int month)
   boolean isLeapYear(int year)
   ...
}
```

Olika synlighets områden

Scope:Namn	Representerar	Typ
0:sc	variabel	Scanner
0:program	metod	void (void)
0:getDayNbr	metod	int (int, int, int)
0:sumToMont	metod	int (int)
0:isLeapYear	metod	boolean (int)
1:year	variable	int
1:month	variable	int
1:day	variable	int
1:n	variable	int
2:month	variable	int
3:year	variable	int

Kompilatorn läser texten och **kommer ihåg**!

Kompilatorn använder deklarationerna under tiden den översätter källkoden.

- Kompilatorn har ett minne, den kommer ihåg vad och var vi har deklarerat saker (namn, synlighetsområde, vad namnet syftar på och typ, m.m.).

Kompilatorn läser källkoden på samma sätt som människor men …

- … läser yttersta blocket först. d.v.s. deklarationer av instansvariabler (variabler utanför alla metoder), metod (bara metodhuvudet) och klasser på samma sätt.
- Därefter läser den metod för metod (metodkroppar).
- I bilden: När kompilatorn läser metoden getDayNbr så vet den redan vad sumToMonth och isLeapYear är eftersom den redan läst yttersta blocket.

Kompilatorn hittar Fel

```
void program() {
   final int i = 123;  // Compiler remember
   boolean b = false;  // Compiler remember

   i++;          // Aha, i declared as final, compile error
   i[0] = 2;     // Aha, i not an array, compile error


   doIt(i);      // Name of a function, correct usage

   out.println( b++); // Aha! Can't increment boolean

   // etc.
}


int doIt(int i) { return 2 * i;}
```

Utifrån deklarationerna kan kompilatorn upptäcka om vi försöker använda namnen på ett felaktigt sätt (eftersom den bokför allt).

- Om så ett kompileringsfel.
- MYCKET bra service. Vi får felen redan vid kompileringen istället för senare vi körningen (kanske körning i många dagar...).
- Generell princip: Alltid bättre att hitta fel tidigt.

Kompilator och Värden

```
final Random rand = new Random();
final Scanner sc = new Scanner(in);
int i;

// What value will be in i? Don't know at compile time!
i = rand.nextInt(100);          // Value??
//i = rand.nextBoolean();      // Type error

i = sc.nextInt();             // Value??
```


Kompilatorn ser inga värden (höger sida vid tilldelning).

- Värden skapas/beräknas under körning … efter kompileringen!

Java Assembler

```
0:  iconst_2
1:  istore_1
2:  iload_1
3:  sipush  1000
6:  if_icmpge       44
9:  iconst_2
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge       31
16: iload_1
17: iload_2
18: irem
19: ifne    25
22: goto    38
25: ...
```


Endast för den nyfikne.