

ex2ifwhile

Joachim von Hacht

Booleska uttryck

```
!false           // Negation
!!true
true && true      // Logical and
true || false    // Logical or

1 == 1           // Equal. NOTE two "=" !!!
1 != 2           // Not equal
2 > 1            // Bigger than etc.
1 < 2
1 <= 1
1 >= 1

int i = 4;
1 <= i && i <= 8 // i in closed interval [1,8]
i < 1 || 8 < i   // outside intervall
```

2

Ett boolesk uttryck representerar värdet true eller false. Byggs upp m.h.a. logiska och/eller relationsoperatorer, literaler, variabler, m.m.

Lat Evaluering

```
int i = 4;
```

```
1 < 2 || i != 4;
```

Behöver inte evalueras

```
1 > 2 && i == 4;
```

Vid beräkning av uttryck med operatorerna || och && används lat evakuering

- För || :Om vänster operand (uttryck) är sant, evalueras inte högersidan (eftersom hela uttrycket är sant om någon av operanderna är sanna)
- För &&: Om vänster operand (uttryck) är falskt evalueras inte högersidan (eftersom hela uttrycket är falskt om någon av operanderna är falska)

Namnge Booleska Uttryck

```
// Hmm, what does it mean? Bad
if( score1 >= 10 || score2 >= 10 ){
    if( score1 != score2) {
        ...
    }
}

// Also bad
if( score1 >= 10 || score2 >= 10 && score1 != score2) {
    ...
}

// Better
boolean gameOver = score1 >= 10 ||
                    score2 >= 10 && score1 != score2;

if( gameOver ) { // Much clearer!
    ...
}
```

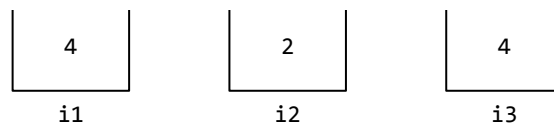
4

Om de booleska uttrycken blir för komplexa, ... skapa en boolesk variabel med ett beskrivande namn

- Mycket lättare att förstå

Likhet för Variabler

```
int i1 = 4;  
int i2 = 2;  
int i3 = 4;  
out.println(i1 == i2);    // False  
out.println(i1 == i3);    // True
```



5

Likhet för variabler innebär att innehållet i respektive variabel jämförs

- Använder == -operatorn.
- Så är det alltid, alltid innehållet i variabeln!
- Om samma innehåll så ger uttrycket värdet true annars false

Likhet för Flyttal

```
double d1 = 1.0;  
double d2 = d1 - 0.6 - 0.4;  
double d3 = d1 - 0.9 - 0.1;  
  
out.println(d2 == 0);    // True!  
out.println(d2 == d3);   // False!
```

6

Flyttal är närmevärden

- Skall aldrig jämföras med == (innehållet kan skilja för 13:e decimalen!)
- Man får använda lite matematik (absolutbelopp av skillnad) eller färdiga metoder i Java (t.ex. Double.compareTo() kommer senare)

Block med Satser

```
{    // Block start
    out.print("Hello ");
    out.print("world");
    out.println("!");
}    // Block end, no ;
```

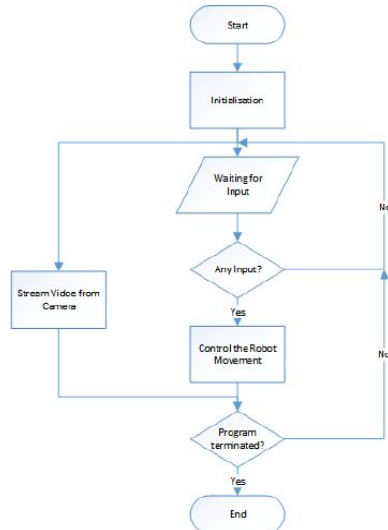
7

En sats kan alltid ersättas med ett block (av eventuellt flera satser)

- Ett block räknas som en sats bestående av 0-n ihopbakade satser
 - Tomma block kan förekomma (undvik)
- Inget “;” efter block *,
 - Behövs inte, det syns var blocket slutar även utan semikolon ...
 - .. nämligen vid }
 - Skriver man ; efter så betyder det bara en tom sats efter blocket.

*) Utom vid något speciellt tillfälle (inget att bekymra sig för i denna kurs).

Programflöde



8

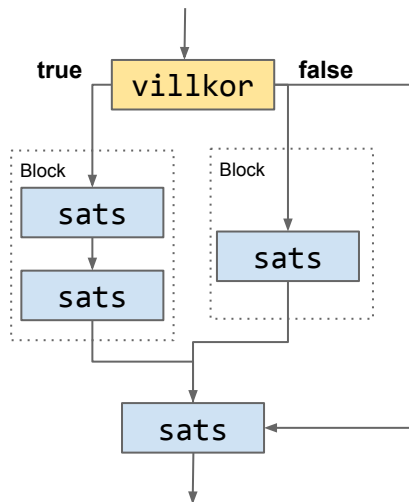
Ett program byggs upp av satser. Att bara skriva satser en efter en (en **sekvens**) räcker inte för att lösa problem.

Vi behöver två konstruktioner till för att styra i vilken ordning satserna exekveras

- **Selektion**, val. Programmet väljer mellan olika satser.
- **Iteration**, upprepning. Programmet upprepar ett antal satser
- Selektion och iteration kallas styrande satser, de styr programflödet.. ([control flow](#))
- Dessutom dyker det upp en sats som kan hoppa i flödet (hoppssatser är normalt inte bra, [en klassisk artikel](#))

Det finns bara tre olika konstruktioner: sekvens, selektion och iteration!

Selektion



En selektion (ett val) styrs av ett villkorsuttryck (ett boolesk uttryck)

- Selektion skrivs i Java som: **if**, **if-else**, **if-else if** eller **switch** satser (switch kommer senare)

if-satsen

```
int i = 4;

// If expression true ...
if (i % 2 == 0 ) {
    out.println("i is ..."); // .. do this
}
// ... else continue here
```

10

if-satsen

- Villkorsuttrycket skrivs i parentes efter if
- Uttrycket måste ha typen boolean
- Om sant körs blocket direkt efter villkoret
- Annars fortsätter programmet efter blocket (blocket hoppas över)

Stilen

- Inledande { på samma rad som if
- Indentera satser i blocken (sköts av IntelliJ)
- Som sagt: Inget ; efter ett block

if-else-satsen

```
int i = 4;

// If expression true ...
if ( 0 <= i && i < 4 ) {
    out.println("i is ..."); // .. do this
} else {
    out.println("i is ..."); // else this
}
// Continue here
```

11

Som if-satsen men om villkoret är falskt så körs blocket vid else

- Även här, se upp men krullparenteser.

if-else if-satsen

```
if (j == 3) {                                // if expression true ...
    out.println("j is 3");                  //... do this...
} else if (k <= 20) {                        // ... else if this true ...
    out.println("k <= 20");                // ... do this ...
} else if ...                               // ... etc.
    ...
} else {                                    // ... else ...
    out.println("j != 3 and k > 20");      // ...do this
}
// Continue here
```

12

Villkoren evalueras ett i taget uppifrån och ner.

- Om något villkor sant så körs blocket direkt efter.
 - Därefter fortsätter programmet efter satsen (efter sista blocket)
 - D.v.s.: om ett villkor sant så exekveras inga andra
 - Alltså viktigt i vilken ordning else if skrivs
- Om inget är sant så körs blocket vid else.

Fallgropar if-satsen

```
int n = ...;
if (n > 0)
    if (n < 5)
        out.println("a");
else
    out.println("b"); // n <= 0 or n >= 5?

if (nCoins <= 0);{ // OhOhhhh!
    out.println("..."); // Will always run
}
```

Använd
alltid
block!

13

Fallgropar (pitfalls), saker man får se upp med

“Dangling else”

- Indenteringen ger ett felaktigt intryck av vilka if och else som hör ihop!
- Java tar inte hänsyn till indentering
- else tillhör närmsta if (normalt sköter IntelliJ detta)

Tomma satsen

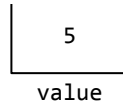
- Inget “;” efter villkorsparentesen.
- Om så körs den tomma satsen!

Använd alltid block för att visa vad som skall köras

- Gäller även om bara en enda sats skall köras!

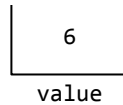
In- och Dekrementering

```
int value = 5;
```



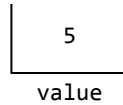
```
// Incrementation
```

```
value++:
```



```
// Decrementation
```

```
value--;
```

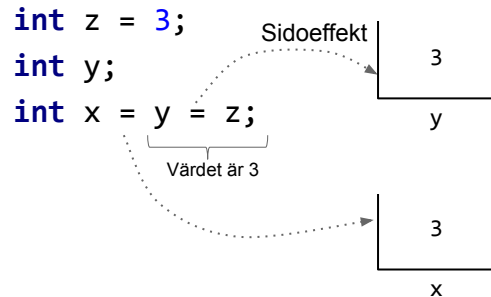


14

Inkrementering eller dekrementering ökar eller minskar en numerisk variabls värde med 1.

- Inkrementeringsoperatoren ++ ökar variabelns värde med 1
- Dekrementeringsoperatoren -- minskar variabelns värde med 1
- Kan inte användas för literaler: 5++ går inte (literaler är fixa värden)!
- ++/-- kan skrivas både före och efter variabelnamnet, vi skriver bara efter, mer senare.
- Vi använder detta bara för heltalsvariabler

Uttryck med Sidoeffekter



```
int a = 2;
out.println(a++); // Print 2
out.println(a);   // Print 3
```

15

Ett uttryck står för ett värde, ett värde beräknas men ...

- ... ibland sker något mer, kallas en **sidoeffekt**
 - Innebär vanligen att minnet ändras
- Tilldelning och in/de-krementering är uttryck med sidoeffekter.

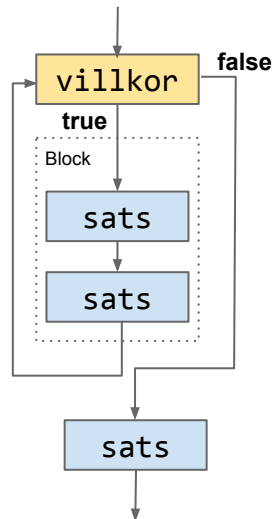
Tilldelning är ett uttryck med sidoeffekt

- Värdet av uttrycket är "det tilldelade"
- Sidoeffekten är att variabelns värde ändras.

In/de-krementering är uttryck med sidoeffekter

- Värdet av uttrycket då ++/-- står efter variabeln är det aktuella värdet (innan ändring)
- Sidoeffekten är att variabelns värde ökar/minskar med 1 (efter att värdet avlästs)

Iteration



16

Iterationen (upprepning) styrs av ett villkor på samma sätt som selektion (typ boolean)

- Om villkoret sant så körs efterföljande block och programmet "hoppas" upp till villkoret igen
- Om falskt så hoppas blocket över
- Iterationer kallas ofta **loopar**
- Iteration skriver vi som: while-satser, for-satser, kort for-sats eller forEach.

while-satsen

```
int value = 0;           // Loop variable
while (value < 5) {
    out.println(value);
    value++;             // Change last in loop
}
```

17

while-satsen

- Styr av villkorsuttrycket i parentesen (typen boolean)
- Loopen använder en "räknare" (loop-variabel) för att styra antalet varv i loopen (med något undantag, se loop and a half).
- Räknaren ändras i loopen för att så småningom göra villkorsuttrycket falskt.
 - Ändringen gör normalt sist i loopen.

Upp eller nedräkning i Java: Börjar normalt på 0 (alltså inte 1), man räknar från 0 och uppåt eller till 0 (inklusive), nedåt.

God praxis för loopar

- Använd loop-variabeln enbart till att räkna upp eller ned (behövs något mer, skapa en till (eller flera) andra variabler.

Fallgropar while-satsen

```
while( i < 5 ){  
    ...           // Must change i !  
}  
  
while( ... );{    // No ; after while!  
    i++;  
}  
  
while( ... != 0.01){ // double not exact!  
}  
  
while( i >= 0 ){    // Never false!  
    i++;  
}
```

18

Händer ibland att man missar och får en loop som inte **terminerar** (körs för evigt, programmet "hänger" sig)

- Man upplever att "inget händer" trots att programmet inte har avslutats

Saker att se upp med

- Villkoret måste påverkas i loopen, det måste bli falskt förr eller senare (undantag: Loop and a half se nedan)
- Inget semikolon efter parentesen, innebär att den tomma satsen körs i för evigt
Likhet för flyttal skall inte användas i villkoret (inte exakta)
- Felaktigt villkor i uttrycket
 - Ofta bättre att använda \leq , \geq i stället för $=$ eller \neq om man skulle missa det exakta värdet.
 - Se upp med \parallel i uttryck, ointuitivt, ... föredra $\&\&$!

OBOE

```
while( ... ){  
    }  
    n = ?
```



Ett mycket vanligt fel är att man kör loopen ett varv för mycket eller för litet

- Känt som "[off by one error](#)" (OBOE)
- Ofta orsakat av t.ex. $>$, $<$ istf \geq , \leq (eller tvärtom)

Nästlade Styrande Satser

```
while( ... ){  
    ...  
    if( ... ){  
        ...  
    }else {  
        ...  
    }  
    ...  
}  
...
```

```
if( ... ){  
    ...  
    while( ... ){  
        ...  
    }  
    ...  
}
```

20

Nästlade styrande satser innebär

- if- och/eller while-satser inuti if- och/eller while-satser (eller andra styrande satser)

Den STORA MAGIN ...!!!

- Genom att kombinera sekvenser, styrande och nästlade styrande satser skapar vi ett logiskt flöde som utför det programmet är tänkt att utföra
- Kan bli komplext att förstå
 - Mer än tre nästlade nivåer skall undvikas!
 - Noggrann indentering för att underlätta läsning är ett måste (IntelliJ sköter det)!

break-satsen

```
// Must be inside while-statement or  
// inside for or switch (more to come)  
while (...) {  
    ...  
    break; // Jump from here ...  
    ...  
}  
// ... to here
```

21

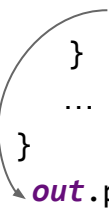
Break-satsen innebär att programmet hoppar från break-instruktionen ut loopen.

Kan break användas i switch-satsen och i while (eller for-looper mer senare)

- Vi använder hopp med försiktighet (så att inte koden blir svårtydd). Vi vill inte hoppa runt "hur som helst"

Loop and a Half

```
while (true) {  
    out.print("Input positive int > ");  
    int i = scan.nextInt();  
    if (i < 0) {  
        break;  
    }  
    ...  
}  
out.print("Loop ended");
```

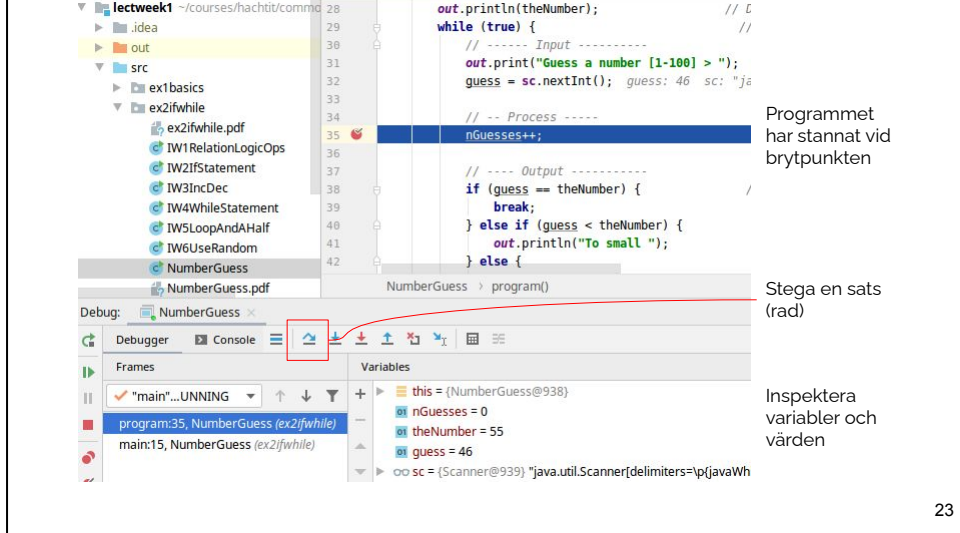


22

Ibland vill man köra en del av en loop innan man vet om man skall upprepa

- Om man t.ex. skall läsa ett värde i loopen som påverkar villkoret
- För att lösa detta används "loop and a half"-mönstret...
 - En variant använder while(true) i kombination med break-satsen
 - I princip är loopen "evig", det finns ingen räknare, typ i++

Avlusare (Debugger)



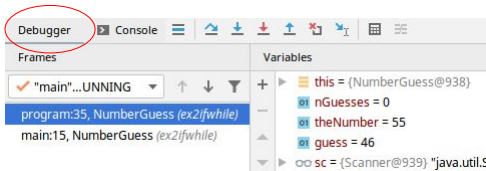
En avlusare (debugger) är ett program som kan köra ett annat (ditt) program sats för sats

- Finns inbyggd i IntelliJ
- Mycket effektivt för att undersöka programflöde och inspektera värden.

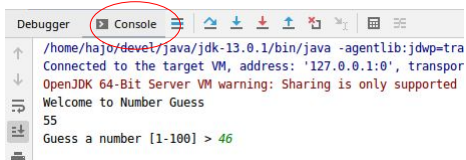
Avlusning (procedur)

- Klicka i vänstermarginalen för att få en brytpunkt (röd prick ovan).
 - En brytpunkt innebär att programmet kommer att stanna på markerad rad.
 - Klicka igen om du vill ta bort..
- Högerklicka i kodefönstret och välj Debug ...
- Avlusaren startar och kör programmet fram till brytpunkten. Där stannar det.
- Därefter kan du köra sats för sats genom att klicka Step Over (röd fyrkant i bilden)
- Den blå raden skall hoppa en sats då du klickar
- Du kan hela tiden inspektera variabelvärden i det nedre fönstret
- Avsluta genom att klicka röd fyrkant ner till vänster (visas ej)
- Hakar något upp sig ... börja om

Avlusare och IO



Programmet har stannat vid en inläsning



Byt till Console och mata in. Tryck enter.

24

Om programmet man avlusar innehåller inläsningssatser stannar det vi inläsningen.

- Man måste då byta till Console fönstret, skriva in data och trycka enter.
- Därefter kan man stega vidare (och inspektera i Debugger-fönstret)

Slumptal

```
import java.util.Random;

// Create random generator
final Random rand = new Random();

void program() {
    int value;
    // Ask it for a random int
    value = rand.nextInt(3) + 1; // 1-3
}
```

25

I spelprogram, simuleringar m. m. behövs ofta slumptal

- I Java-program kan man använda en [slumptalsgenerator](#)
- Man måste först skapa en generator (ett objekt)
- Därefter kan man be den om olika typer av slumptal t.ex. slumpmässiga heltal

Analys av kod

- final-raden betyder (ungefär) att vi skapat en slumptalsgenerator som heter rand.
 - Detta sker utanför (före) program(): Ett specialfall!
 - Ni skall undvika detta!
 - Deklarera allt i program() tills vidare ...
- I program() deklarerar vi en heltalsvariabel value
- Uttrycket rand.nextInt(3) säger åt slumptalsgeneratoren att skapa ett slumptal i intervallet [0, 2] (d.v.s. givet talet n för vi max ut n-1)
 - Genom att addera 1 flyttas intervallet därefter till [1,3]
 - Resultatet tilldelas value.