## 9.1 Sammendrag og oversigter.

Dette afsnit sætter det, du har lært i system og kan bruges som opslagsværk. Enkelte steder står der noget, som ikke er gennemgået endnu, men som er med for helhedens skyld.

### 9.2 Navngivningsregler

Variabler og metoder bør have lille startbogstav.

Eksempler: n, alder, tal, talUndersoeg, main(), println(), sqrt().

Klasser bør have stort startbogstav.

Eksempler: HejVerden, Cylinderberegning, Syvtabel2

Består navnet af flere ord, stryger man normalt mellemrummene og lader hvert af de efterfølgende ord starte med stort (nogen bruger også understreg \_ som mellemrum).

* Navnet kan bestå af A-Z, a-z, 0-9, $ og \_
* Det må ikke starte med et tal. Det kan have en vilkårlig længde.
* Lovlige navne: peter, Peter, $antal, var2, J2EE, dette\_er\_en\_test
* Ulovlige navne: 7eleven, dette-er-en-test, peter#

Da visse styresystemer endnu ikke understøtter æ, ø og å i filnavne, bør man undgå disse i klassenavne.

### 9.3 De simple typer

Her er en oversigt over alle de simple variabeltyper i Java.

| **Type** | **Art** | **Antal bit** | **Mulige værdier** | **Standardværdi** |
| --- | --- | --- | --- | --- |
| byte | heltal | 8 | -128 til 127 | 0 |
| short | heltal | 16 | -32768 til 32767 | 0 |
| int | heltal | 32 | -2147483648 til 2147483647 | 0 |
| long | heltal | 64 | -9223372036854775808 til 9223372036854775807 | 0 |
| float | kommatal | 32 | ±1.40239846E-45 til ±3.40282347E+38 | 0.0 |
| double | kommatal | 64 | ±4.94065645841246544E-324 til 1.79769313486231570E+308 | 0.0 |
| char | unicode | 16 | \u0000 til \uffff (0 til 65535) | \u0000 |
| boolean | logisk | 1 | true og false | false |

De vigtigste er int, double og boolean. I enkelte tilfælde bliver long og char også brugt, mens byte, short og float meget sjældent bruges.

### 9.4 Værditypekonvertering (eng: ”Casting”)

Konvertering til en anden type sker automatisk i de tilfælde, hvor der ikke mistes information (forstået på den måde, at intervallet af de mulige værdier udvides), dvs. følgende går godt:

* fra byte til short, int, long, float eller double
* fra short til int, long, float eller double
* fra int til long, float eller double
* fra long til float eller double
* fra float til double.

Den anden vej, dvs. hvor der muligvis mistes information, fordi intervallet af mulige værdier indsnævres, skal man skrive en eksplicit typekonvertering.

Det gøres ved at skrive en parentes med typenavnet foran det, der skal konverteres:

int x;

double y;

y = 3.8;

x = (int) y //eller med normal oprundingsregler x = Math.round(y);

Her skæres komma-delen af 3.8 væk og x får værdien 3.

Eksplicit typekonvertering sikrer at programmøren er bevidst om informationstabet (glemmes dette kommer oversætteren med fejlen: ”possible loss of precision: double, required: int”). Casting bruges

* fra double til float, long, int, short, char eller byte
* fra float til long, int, short, char eller byte
* fra long til int, short, char eller byte
* fra int til short, char eller byte
* fra short til char eller byte
* fra byte til char
* fra char til short eller byte.

Der kan ikke typekonverteres til eller fra boolean.

### 9.5 Aritmetiske operatorer

| **Operator** | **Brug** | **Forklaring** |
| --- | --- | --- |
| + | a + b | a lagt sammen med b |
| - | a – b | b trukket fra a |
| \* | a \* b | a gange b |
| / | a / b | a divideret med b |
| % | a % b | rest fra heltalsdivision af a med b |
| - | -a | den negative værdi af a |
| ++ | a++ | a = a+1; værdi før optælling |
| ++ | ++a | a = a+1; værdi efter optælling |
| -- | a-- | a = a-1; værdi før nedtælling |
| -- | --a | a = a-1; værdi efter nedtælling |

Operatorerne giver altid samme type som operanderne, der indgår. Det skal man være specielt opmærksom på for / (divisions) vedkommende, hvor resten mistes ved heltalsdivision.

Operatoren ++ tæller en variabel op med én : a++ svarer til a=a+1. Tilsvarende er a-- det samme som a=a-1.

### 9.6 Regning med logiske udtryk

u1 og u2 er to logiske udtryk eller logiske variabler

| **Operator** | **Brug** | **Forklaring** |
| --- | --- | --- |
| && | u1 && u2 | både u1 og u2 er sandt |
| || | u1 || u2 | u1 eller u2 er sandt |
| ! | ! u1 | negation af u1 |

**Operator &&** udtrykker, at både 1. **og** 2. udtryk skal være sandt:

| **1. udtryk** | **2. udtryk** | **1. udtryk && 2. udtryk** |
| --- | --- | --- |
| FALSK | FALSK | FALSK |
| FALSK | SAND | FALSK |
| SAND | FALSK | FALSK |
| SAND | SAND | SAND |

F.eks. er udsagnet (a > 5 && a < 10) sandt, hvis a er større end 5, og a er mindre end 10.

**Operator ||** udtrykker, at 1. **eller** 2. udtryk skal være sandt.

| **1. udtryk** | **2. udtryk** | **1. udtryk || 2. udtryk** |
| --- | --- | --- |
| FALSK | FALSK | FALSK |
| FALSK | SAND | SAND |
| SAND | FALSK | SAND |
| SAND | SAND | SAND |

F.eks. er udsagnet (a > 5 || a == 0) sandt, hvis a er større end 5, eller a er 0.

**Operator !** Udtrykker, at udtrykket skal **negeres**, dvs. at (!u1) er sandt, hvis u1 er falsk, og falsk hvis u1 er sandt, f.eks. er udsagnet (!(a > 5)) sandt, hvis der ikke gælder at a er større end 5 (det er det samme som (a <= 5)).

I visse andre programmeringssprog skrives AND for &&, OR for || og NOT for !

### 9.7 Sammenligningsoperatorer

| **Operator** | **Brug** | **Forklaring** |
| --- | --- | --- |
| > | a > b | a større end b |
| >= | a >= b | a større end el. lig med b |
| < | a < b | a mindre end b |
| <= | a <= b | a mindre end el. lig med b |
| == | a == b | a er lig med (identisk med) b |
| != | a != b | a forskellig fra b |

### 9.9 Gode råd om programmering

* Gennemtænk problemstillingen, inden du sætter dig til computeren.
* Formulér problemet eller formålet med programmet.
* Overvej mulige løsningsstrategier. De fleste problemer kan løses på mere end én måde.
* Lav en skitse til programmet i pseudokode (på papiret med danske ord). Du kan også tegne flowdiagrammer (der beskriver rækkefølgen tingene sker i).
* Det er ikke altid, man kan tænke hele programmet igennem på forhånd. Ved mere komplicerede programmer må man skifte mellem kodning og refleksion over koden.
* Når du sidder ved computeren, så skriv ganske få linier ad gangen, og afprøv. På den måde er det ofte lettere at se problemet, hvis programmet ikke virker.
* Gør flittigt brug af System.out.println(...) til at kontrollere, om programmet gør som forventet.
* Lær en standardiseret indrykning fra starten og følg den stringent

## 9.10 Opgaver

### Befordringsfradrag

Lav et program, som udregner befordringsfradraget (det der kan trækkes fra i skat ud fra, hvor langt der er mellem arbejde og hjem

Satserne for indkomståret 2015 er fastsat således:  
  
0-24 km inkl.: intet fradrag  
25 - 120 km inkl.: 2,05 kr. pr. km  
over 120 km: 1,03 kr. pr. km(udkantskommuner 2,05 kr)

(I denne opgave ser vi bort fra at i bestemte udkantskommuner gives den høje sats også over 120 km.)

Opvarmning:

Udregn, og udskriv fradraget pr. dag fra 25 til 75 km på hver sin linie.  
Udregn, og udskriv fradraget pr. dag fra 25 til 150 km på hver sin linie.

**Opgave 9.1:**

Udregn, og udskriv fradraget pr. dag fra 10 til 150 km på hver sin linie.   
Kun hver 10. km udskrives (10km, 20km, 30km...).

**Ekstra opgave til lektion 9:**

Forbedring af et gammelt program: (Fra 5.1) Gæt et tal mellem 0 og 100.

(Nogle af jer har allerede løst nogle af under-opgaverne)

1. Pak hele programmet ind i en While-løkke så man kan vælge om man vil spille igen
2. Udforsk og brug metoden Math.random( ) til at lade programmet selv generere tallet, der skal gættes.
3. Optæl hvor mange gæt brugeren har brugt. Indsæt evt. en grænse for hvor mange gæt man har.
4. Giv brugeren point. F.eks. 1.gæt 100p, 2-4 gæt: 5p, 5-10 gæt 1 p, >10 gæt 0p. Hvis der spilles flere runder skal pointene tælles sammen og antal point og antal runder oplyses ved afslutning.

God arbejdslyst /Just