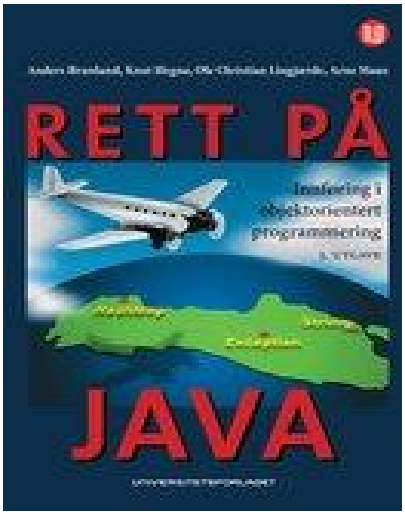
PENSUM

INF1000

H11



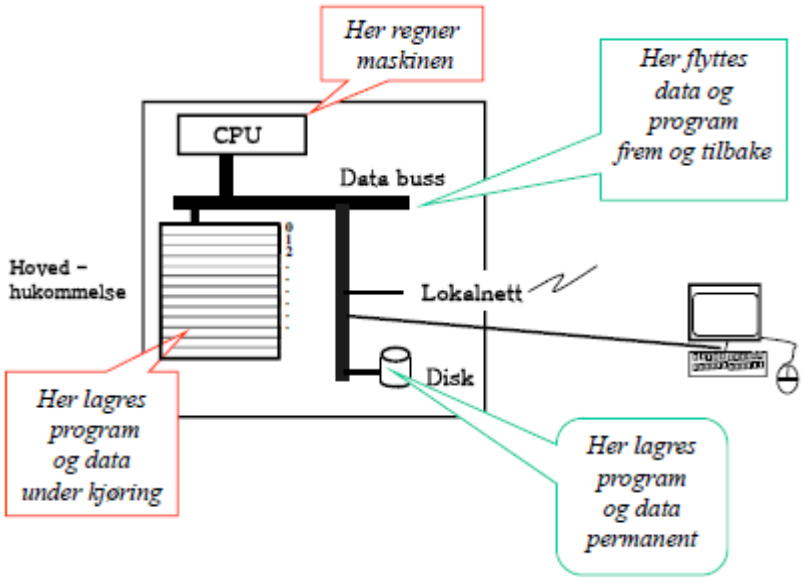
Rett På Java, , 3. Utgave

Joakim Myrvoll Johansen

STIKKORDREGISTER:

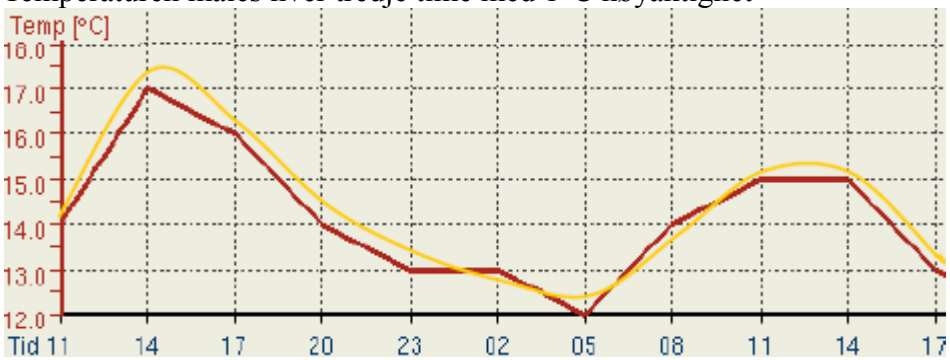
INF1000

- ✗ Hva er INF1000?
  - Felles innføringskurs i Objektorientert programmering for ca 7
  - bachelor-programmer (MatNat, Jus)
  - 10 studiepoeng
  - ca. 500 studenter
  - Et frittstående introduksjonskurs for deg som vil lære å programmere Java og kanskje ta flere kurs senere.
  - Programmering videreføres i INF1010
- ✗ Innhold:
  - Litt datateknologi
  - Noe tekstbehandling
  - Mye programmering
- ✗ Hva er en datamaskin?



- ✗ Programmering:
  - Vi skriver våre programmer på en måte som er lettest for oss mennesker (til editoren)
  - Denne skrivemåten kalles et programmeringsspråk
  - En programtekst skrevet i et slikt programmeringsspråk kan:
    - lett oversettes (av oversetteren) til enkle operasjoner,
    - som lagres i hovedhukommelsen og
    - så kjøres (av kjøre-programmet)
  - Det er mange programmeringsspråk - det vi bruker I INF1000 heter Java

- ✗ Hva jobber datamaskiner egentlig med?
  - I våre dager: Digitale datamaskiner
    - Digital (fra Bokmålsordboka): ..som gjengir fysiske størrelser med diskrete tegn (oftest siffer)
  - Baserer all lagring og prosessering av data på det binære tallsystemet med kun to siffere: 0 og 1
  - 0 og 1 representeres på laveste nivå gjerne som spenning/ ikke spenning. Eller enda mer nøyaktig:
    - 0-0.8 volt = ”lav” = 0
    - 2-5 volt = ”høy” = 1
  - Også lys, magnetisme, hull/ ikke hull, ….
- ✗ Datamaskinverdenen er binær digital
  - 2 diskrete verdier, 0 og 1
  - 0 og 1 kalles binære sifre – binary digits – bits
  - Alt i datamaskinen er representert ved sekvenser av bits – bitmønstre
  - Moderne datamaskiner arbeider gjerne med grupper på 8 bits
    - En slik gruppe på 8 bits kalles en byte.
- ✗ Hva betyr bitmønsteret?
  - Bitmønsteret 10100100 kan bl.a. være:
    - 164 (tolket som binærtall)
    - -36 (negativt binærtall med fortegnsbit)
    - -90 (negativt tall på toerkomplementsform)
    - € (lSO 8859-15)
    - ¤ (Windows Codepage 1252)
    - (som gråtone)
    - …
- ✗ Analog virkelighet – digital representasjon
  - Analog: ”..basert på fysiske, kontinuerlig varierbare størrelser”
  - Digital: ”..som gjengir fysiske størrelser med diskrete tegn”
  - Forutsetter diskretisering og kvantisering
  - Eksempel:
    - Temperaturen måles hver tredje time med 1°C nøyaktighet



Binærtall - tallsystemer

- ✗ Potensregning – kort repetisjon
  - grunntalleksponent = grunntall * grunntall * ... * grunntall

    eksponent antall ganger
  - Spesialregel: grunntall0 = 1
  - Eks:
    - $10^3 = 10*10*10 = 1000$
    - $2^4 = 2 * 2 * 2 * 2 = 16$
    - $5^0 = 1$
- ✗ Hva er et tallsystem?
  - Trenger noen symboler (siffere), og en mekanisme for å kombinere disse slik at vi kan håndtere "uendelig" store tall
  - I vårt (desimale, dekadiske) titallsystem har vi 10 siffere (0-9), og bruker sammensetninger av disse for å lage tall større enn 9 vha mekanismen posisjonssystemet.

    $19 = 1*10^1 + 9*10^0$
  - Det binære tallsystem (totallsystemet) har bare 2 siffere – men bruker samme mekanisme (posisjonssystemet) for å håndtere tall større enn 1

    $1\ 0\ 1\ 0\ 1 = 1*2^4 + 1*2^2 + 1*2^0$
  - Eksempel på en annen mekanisme: Romertall. Har symboler som I, V, X og L – her omfatter reglene for å kombinere til nye tall bla. Subtraksjon.

    $X\ I\ X = 10 + (10-1)$
- ✗ Titallsystemet – et posisjonssystem
  - I titallsystemet (desimalsystemet) har vi
    - Grunntall 10
    - 10 sifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - Større tall konstrueres ved hjelp av posisjonssystemet:

10^6 (1 000 000)	10^5 (100 000)	10^4 (10 000)	10^3 (1 000)	10^2 (100)	10^1 (10)	10^0 (1)

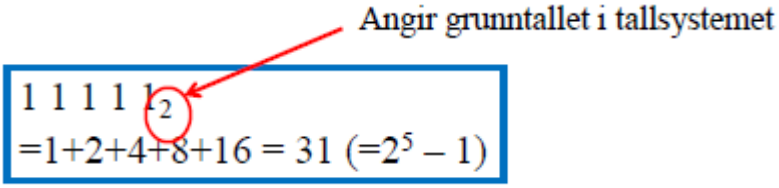
- Det binære tallsystemet
  - Det binære tallsystemet (totallsystemet) har
    - grunntall 2
    - 2 sifre: 0 og 1

2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)

- Titallsystemet --> totallsystemet (binærtall)
  - Gitt et tall x i titallsystemet.
  - Start på posisjon 0.
  - Hvis x er oddetall, sett 1 i denne posisjonen (ellers 0).
  - Sett x lik x heltallsdividert med 2.
  - Fortsett med neste posisjon til venstre, så lenge $x \neq 0$.

```
int[] tilBintall(int x) {
    int[] bintall = new int[8];

    int pos = 0;


    do {
        if (x % 2 == 1) {
            bintall[pos] = 1;
        } else {
            bintall[pos] = 0;
        }
        x = x/2;
        pos++;

    } while (x != 0);

    return bintall;
}
```

- Bitposisjoner og bitmønstre
  - For et tall x i titallsystemet, hvor mange sifre (bits) trenger det tilsvarende binærtallet?
  - Det største tallet som kan representeres med b bits er $2^b - 1$.
  - Vi må altså finne den minste b'en slik at $x \leq 2^b - 1$.



- ✗ Det heksadesimale tallsystemet
  - I det heksadesimale tallsystemet har vi
    - grunntall 16
    - 16 sifre: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

16^5 (1 048 576)	16^4 (65 536)	16^3 (4 096)	16^2 (256)	16^1 (16)	16^0 (1)

- ✗ Konvertering binært <--> heksadesimalt

Binært	Heksadesimalt
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7

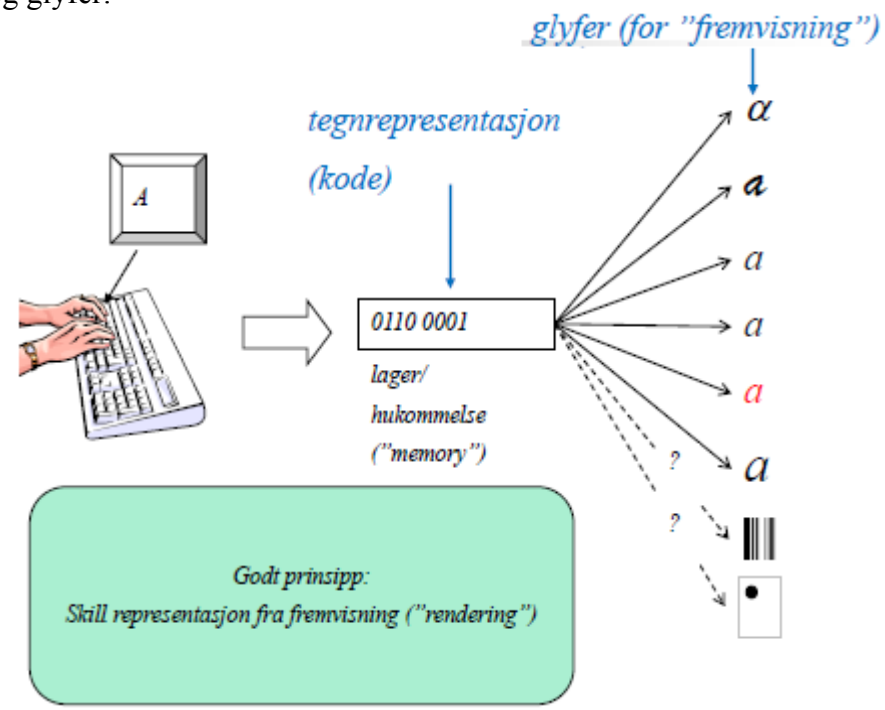
Binært	Heksadesimalt
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

  - For å angi at noe er skrevet på heksadesimal form kan vi enten
    - føye til 16 som subskript, f.eks. $A1_{16}$, eller
    - føye til 0x i forkant, f.eks. 0XA1

Tegn og tekster

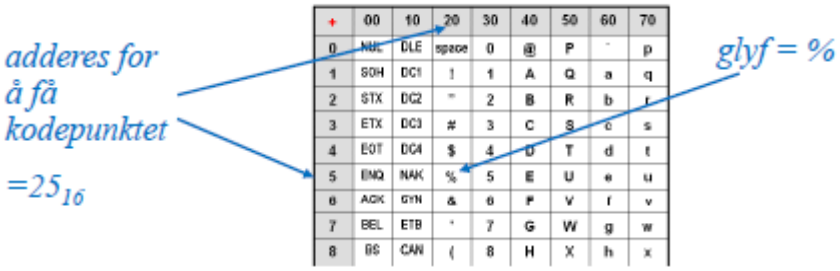
- ✗ Problemstilling
  - Hvert tegn i teksten representeres av et unikt bitmønster.
  - Eksempel:
    - E = 01000101
    - H= 01001000
    - I = 01001001
    - HEI = 01001000 01000101 01001001
  - Sender (skriver) og mottaker (leser) må være enige om kodingen.
    - Vi trenger standarder!
- ✗ Aktuelle spørsmål
  - Hvilke tegn skal representeres?
  - Hvor mange bits per tegn?
  - Fast eller variabelt antall bits per tegn?
  - Bør det være noen form for systematikk i bitmønstrene?
  - Ulike svar i ulike standarder – gis i kodetabeller

✗ Om tegn og glyfer:



✗ Kodetabeller

- Hvert tegn som inngår i tegnsettet tilordnes et kodepunkt (ofte angitt på heksadesimal form)
  - Tegnets "numeriske" verdi
  - Det som står i kodetabellen
- Representasjon:
  - Hvordan tegnet representeres som et bitmønster



+	00	10	20	30	40	50	60	70
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x

<utsnitt av ascii-tabell>

✗ ASCII kodetabell
- American Standard Code for Information Interchange
- 1963 →

	00	10	20	30	40	50	60	70
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

✗ Linjeskift: LF og CR
- LF (Line Feed):
  - 0x0A
  - "Indicates movement of the printing mechanism or display cursor to the next line."
- CR (Carriage Return):
  - 0x0D
  - "Indicates movement of the printing mechanism or display cursor to the starting position of the same line."
- Linjeskift representeres i dag på ulike måter:
  - PC: CR + LF
  - Mac: CR
  - UNIX: LF

✗ ISO 646-60 kodetabell
- Bygger på ASCII.
- [\ ] { | } er ofret til fordel for Æ Ø Å æ ø å
- Lignende tilpasninger er gjort i tilsvarende standarder for andre språkmiljøer.

	00	10	20	30	40	50	60	70
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	Æ	k	æ
C	FF	FS	,	<	L	Ø	l	ø
D	CR	GS	-	=	M	Å	m	å
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

- ISO 8859-1 (Latin-1) kodetabell

	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0
0	NUL	DLE	space	0	@	P	`	p			no break space	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q			¡	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r			¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s			£	³	Ã	Ó	ã	ó
4	EOT	DC4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	*un-defined*		¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w			§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x			¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y			©	¹	É	Ù	é	ù
A	LF	SUB	*	:	J	Z	j	z			ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{			«	»	Ë	Û	ë	û
C	FF	FS	,	<	L	\	l	|		¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}			-	½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~			®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL			¯	¿	Ï	ß	ï	ÿ

- ETSI GSM 03.38 kodetabell for SMS

	00	10	20	30	40	50	60	70
0	@	Δ	space	0	¡	P	¿	p
1	£	_	!	1	A	Q	a	q
2	$	Φ	"	2	B	R	b	r
3	¥	Γ	#	3	C	S	c	s
4	è	Λ	¤	4	D	T	d	t
5	é	Ω	%	5	E	U	e	u
6	ù	Π	&	6	F	V	f	v
7	ì	Ψ	'	7	G	W	g	w
8	ò	Σ	(	8	H	X	h	x
9	Ç	Θ	)	9	I	Y	i	y
A	LF	Ξ	*	:	J	Z	j	z
B	Ø	ESC	+	;	K	Ä	k	ä
C	ø	Æ	,	<	L	Ö	l	ö
D	CR	æ	-	=	M	Ñ	m	ñ
E	Å	unde f	.	>	N	Ü	n	ü
F	å	É	/	?	O	§	o	à

... pluss disse 10 "escape"-sekvensene

€	ESC e
FF	ESC LF
[	ESC <
\	ESC /
]	ESC >
^	ESC Λ
{	ESC (
|	ESC @
}	ESC)
~	ESC =

- Den endelige løsning? Unicode og ISO 10646
  (se http://www.unicode.org/charts/)
  - 17 ”plan” med 256 *256 kodepunkter i kodetabellen
    - Ca 130 000 private
    - Ca 870 000 ennå ikke brukt
  - Kodepunktet representeres med 21 bit, evt innledende 0’er
  - Første 256 tegn identisk med ISO 8859-1
  - For hvert tegn finnes:
    - en representativ glyf
    - kodepunktet
    - et navn
    - klassifisering
    - skriveretning
  - Vedtatte tegn med kodepunkter skal aldri endres

- ✗ String-metoden compareTo i Java
  - Basert på Unicode-verdiene til hvert enkelt tegn i de to tekstene:

```
void sammenlign(String s1, String s2) {
    if (s1.compareTo(s2) < 0) {
        System.out.println(s1 + " " + s2);
    } else if (s1.compareTo(s2) = 0) {
        System.out.println("Like!");
    } else { // s1.compareTo(s2) > 0
        System.out.println(s2 + " " + s1);
    }
}
```

    NB: Hva skjer hvis tekstene inneholder de norske tegnene æøå?

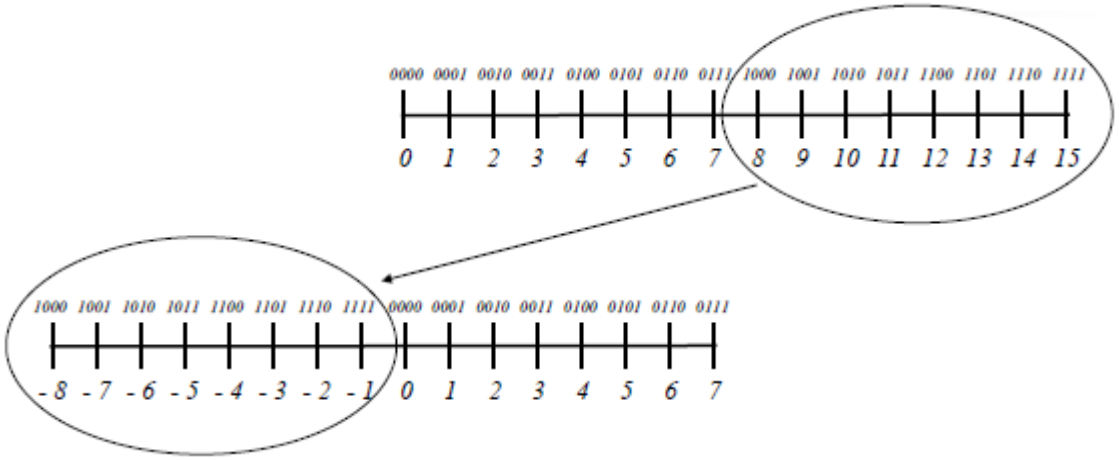
- ✗ Representasjon av kodepunkter i Unicode
  - I Unicode representeres ikke de 21 biters kodepunktene direkte, men styres av kodinger: Unicode Transformation Formats (UTF)
  - UTF-8, UTF-16 og UTF-32: Tallet angir minimum antall bit som benyttes til representasjonen
  - UTF-8: ”litt komplisert, men genial”
    - 8, 16, 24 eller 32 bit representasjon av hvert tegn
    - alle tegn fra ASCII representeres direkte: bruker 8 bit og innledes med 0
    - “bakoverkompatibel” med ASCII og ISO 8859-1
- ✗ Big endian vs. Little endian
  - I representasjoner som krever mer enn én byte, finnes det to mulige rekkefølger av bytene:
    - Starte med den mest signifikante (”Big endian”)
    - Starte med den minst signifikante (”Little/small endian”)
  - Eksempel:
    UTF-16 Big endian for A er 0x 00 41
    UTF-16 Little endian for A er 0x 41 00
  - Begge muligheter blir brukt i praksis, og dette kan gi problemer når data overføres fra et maskinmiljø til et annet!

- ✗ Byte order mark (BOM)
  - Et ”Byte order mark” (BOM) er tegnet ”Zero width no-break space” med kodepunkt U+FEFF i begynnelsen av en Unicode-fil.
  - Siden det ikke finnes noe tegn med kodepunkt FFFE, kan BOM brukes til å finne filformatet (UTF-32, UTF-16, UTF-8 og Big eller Small endian):

Koding	BOM-bitmønster
UTF-32, big-endian	0x 00 00 FE FF
UTF-32, little-endian	0x FF FE 00 00
UTF-16, big-endian	0x FE FF
UTF-16, little-endian	0x FF FE
UTF-8	0x EF BB BF

Negative og reelle tall

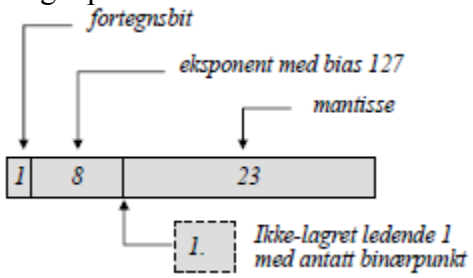
- ✗ Ulike klasser tall
  - De naturlige tallene: $\mathbb{N} = \{ 1, 2, 3, \ldots \}$
  - De hele tallene: $\mathbb{Z} = \{ \ldots, -2, -1, 0, 1, 2, \ldots \}$
  - De rasjonale tallene: $\mathbb{Q}$ = alle tall som kan skrives som en brøk
  - De reelle tallene: $\mathbb{R}$ = alle tallene på tallinjen
- ✗ Negative tall: toer-komplement



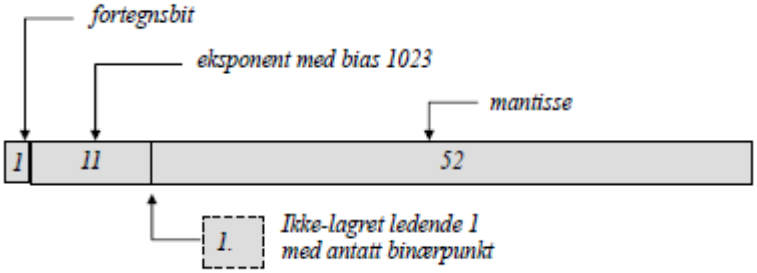
  - For å negere et tall:
    1. Snu alle bit’ene i tallet (0 ★ 1).
    2. Legg til 1.

- ✗ Negative tall: Bias
  - Et alternativ er å legge til en konstant bias til alle tallene.
  - Med 8 bitposisjoner og bias 128 kan tallene fra -128 til 127 representeres ved hjelp av tallene fra 0 til 255.
  - Eksempler:
    53: -21:

- ✗ Flyttall
  - I titallsystemet kan et tall skrives på formen mantisse * 10eksponent
  - Eksempler:
    $0.5 * 10^2 = 0.5 * 100 = 50$
    $0.5 * 10^0 = 0.5 * 1 = 0.5$
    $0.5 * 10^{-1} = 0.5 * 0.1 = 0.05$
    $-5 * 10^{-1} = -0.5 * 0.1 = -0.05$
  - Tilsvarende kan vi skrive binære flyttall på formen
    mantisse * $2^{\text{eksponent}}$
  - For flyttall må vi altså representere både eksponent og mantisse. Begge må kunne være positive, negative og null.
- ✗ Binære flyttall: IEEE 754 single precision

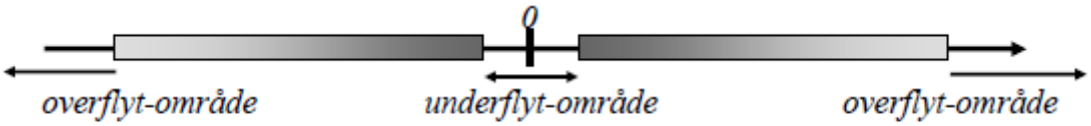


- ✗ Binære flyttall: IEEE 754 double precision



- ✗ IEEE 754: Spesielle verdier
  - Null: Både eksponent og mantisse er 0
  - Uendelig: Eksponent med bare 1ere, mantisse med bare 0ere
  - Not A Number: Eksponent med bare 1ere, mantisse ≠ 0

    - Nødvendig for main
    - Ellers det samme som <ingenting> i vårt tilfelle

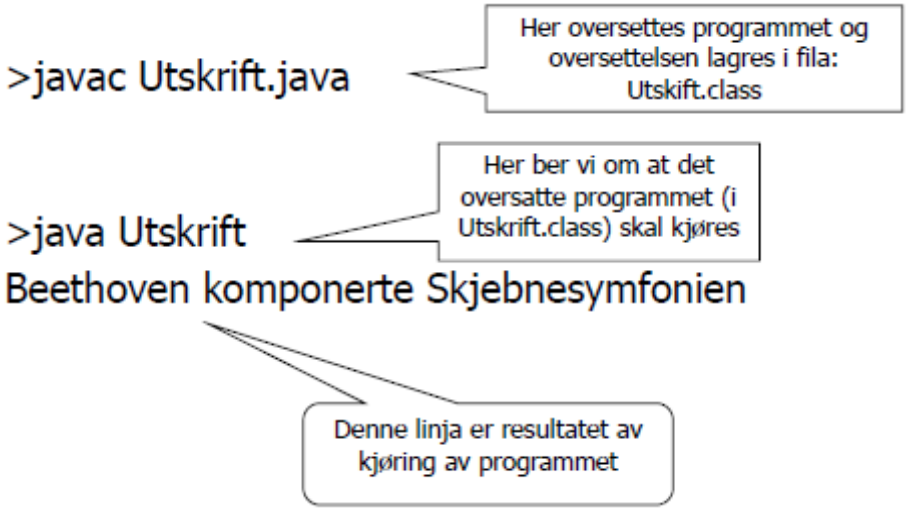


datatype	antall biter	minste positive tall	største positive tall
float	32	$2^{-126} \approx 10^{-44,85}$	$(2 - 2^{-23}) * 2^{127} \approx 10^{38,53}$
double	64	$2^{-1022} \approx 10^{-323,3}$	$(2 - 2^{-52}) * 2^{1023} \approx 10^{308.3}$

Slik delvis sperring av adgang sikrer oss at vi selv bestemmer hvordan en variabel i en klasse skal endres.

JAVA

- Kompilering (=oversetting) og kjøring (av det oversatte)



- Programmering generelt
  - Vi skriver programmet som en tekst i en editor (TexPad, emacs,..)
  - Vi lagrer filen med programmet lik navnet på klassen og med java etter punktum
    Eks: Filnavn.java
  - Vi lar kompilatoren javac oversette .java filen og legge oversettelsen i en ny fil - her: Filnavn.class
  - Vi starter opp kjøresystemet java med Filnavn som parameter på samme linje (den forstår at dette er Filnavn.class)
  - Kjøresystemet leser så denne og utfører de instruksjonene som ligger på .class fila - her: Filnavn.class
  - Kommandoene som ligger i main blir da utført,
    - en etter en
    - ovenfra og nedover (til vi har utført siste ordre i main)

✗ Kommentarer I programmene
- Kommentarer gjør programmene lettere å forstå for en menneskelig leser
- De oversettes ikke: kompilatoren hopper over dem
- To typer kommentarer:

```
// Her er en kommentar som varer ut linja

/* Her er en kommentar
som varer
helt til hit */
```

- Gode programmer har kommentarer, men ikke på hver linje

✗ Blokker i programmer
- En programblokk er en samling programsetninger omsluttet av krøllparenteser

```
class Utskrift {
    public static void main(String[] args) {
        System.out.println("Beethoven komponerte  Skjebnesymfonien");
    }
}
```

- Blokker kan ligge inne i blokker (se over)
- En variabels skoper fra deklarasjonsstedet til blokken avsluttes
- Variable er ikke definerte (synlige) utenfor sine skop

✗ Hvordan løse oppgaver?

1. Se oppgaven utenfra:
- Hva skal være inndata (input) til programmet?
- Hvordan skal programmet få tak i inndataene?
- Hva skal være utdata (output) fra programmet?
- Hvordan skal utdataene presenteres for brukeren?

2. Hvordan transformere inndata til utdata?
- Hvordan skal dataene representeres (lagres)?
- Spesifiser en sekvens av trinn der:
  - hvert trinn gjør en enkel ting med dataene
  - hvert trinn er enkelt å programmere

3. Skriv programkode (og test løsningen)

✗ Programskisse –”Pseudokode”

```
class Addering {
public static void main (String[] args) {
<deklarasjoner>

<sett tall1 lik 123456>
<sett tall2 lik 99999>

<regn ut summen>

<skriv ut>
}
}
```

Tegn gjerne variablene og følg med på verdiene de får når du utfører handlingene en etter en – simuler maskinen!

✗ Objektorientert programmering

- Programmeringsspråk er designet etter ulike ”paradigmer” eller hovedprinsipper.
- Java er et objektorientertspråk (andre typer er imperative eller funksjonelle)
- Det betyr at språket spesielt støtter programmereren i å tenke på det som skal beskrives og bearbeides i form av objekter – som har både egenskaper (variable) og handlinger (programsetninger samlet i metoder) knyttet til seg.
- Hvilke variable og metoder som hører til et objekt bestemmes av hvilken klasseobjektet tilhører – en klasse beskriver et mønster for en type objekter.
- Viktige egenskaper ved objektorienterte programmer er muligheten til å skjule detaljer som ikke er vesentlig for omgivelsene (vi velger selv hva som skal synes og brukes fra utsiden av en klasse) og å grupperevariable og metoder som hører sammen.

✗ Objektorientering: Hvorfor?

- Et program (særlig i eldre og mer lavnivå språk) kan bestå av en lang rekke setninger der kontrollen hopper frem og tilbake, kanskje bare med direkte ”hopp”-instruksjoner (”goto”)
- Data kan deklareres som en ”uendelig” lang rekke felles variable, som brukes når de trengs og er tilgjengelige over alt
- Dette er uproblematisk for datamaskinen som skal utføre programmet, MEN:
  - det blir fort vanskelig å holde oversikten for programmereren
  - svært tidkrevende å finne feil
  - nesten umulig å sette seg inn i for andre (eller huske hvordan det fungerte noen måneder etter at man skrev det)
  - vanskelig (farlig!) å utvide/ endre
  - nesten umulig å dele opp for å samarbeide om utvikling

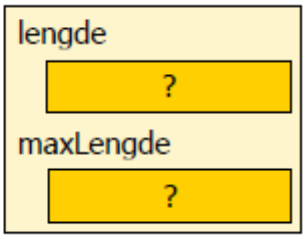
- ✗ Objektorientering: Hvordan?
  "Think globally, act locally!"
  - Data:
    - Tenk helhet, overordnet gruppering gjennom identifisering av viktige ”ting” (substantiver) -> klasser
    - Arbeid lokalt med bestanddeler, representasjon -> variable
- ✗ Handlinger (metoder):
  - Først hvilke oppgaver som skal løses, og hvor de best løses (hvor er dataene de skal arbeide med)
  - ..deretter: Hvordan løse oppgaven

```
int formue = 20;

if (formue > 100000000) {
    System.out.println ("Faktisk en stor formue! ");
} else if (formue > 1000000) {
    System.out.println ("Faktisk ganske romslig!");
} else {
 System.out.println ("Men ikke svært mye!");
}
```

VARIABLER

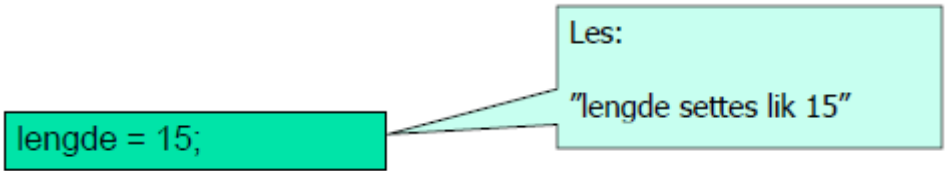
- ✗ Variabel:
  - En plass i hukommelsen
    - For å kunne huske og forandre på data, trenger programmet variable å lagre dem i.
    - Vi reserverer plass i hukommelsen ved å deklarereen variabel – da oppgir vi type og navn

```
int lengde;
int maxLengde;
```

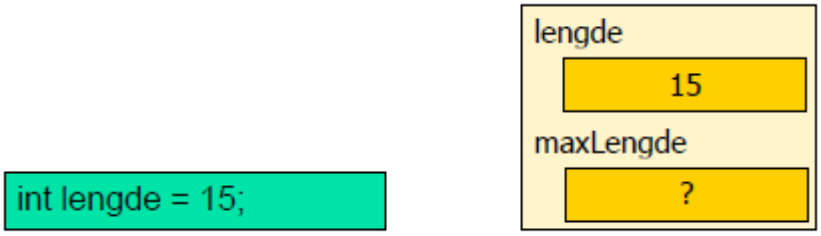


    - Typen avgjør hva slags verdi som kan lagres i variabelen: int er en type for å lagre heltall
    - Du bestemmer navnet: start med (helst liten) bokstav, deretter tall og bokstaver

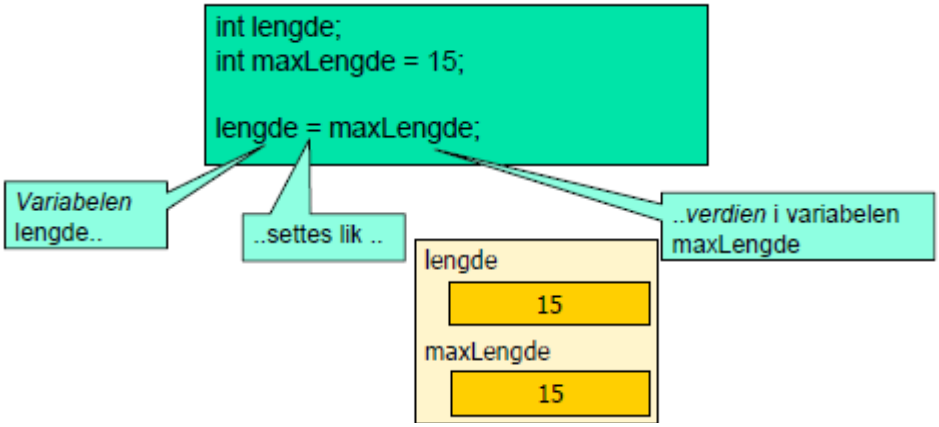
- Tilordninger
  - Når variabelen er deklarert, kan vi plassere en verdi av riktig type der
  - Dette kalles en tilordning



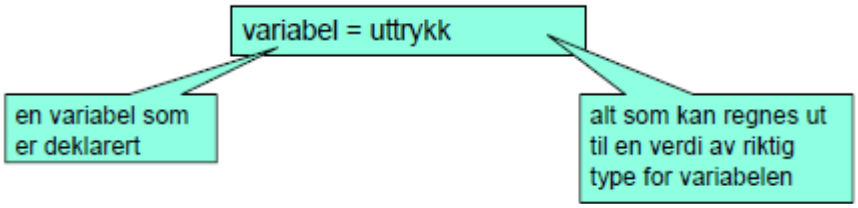
  - Kan også gjøres sammen med deklarasjonen



- Avlesing
  - Verdien i en variabel kan leses av og for eksempel benyttes for å gi verdi til en annen variabel ved en tilordning



- Uttrykk
  - Den generelle formen for en tilordning er



  - Uttrykket på høyre side leses av og beregnes først – deretter plasseres resultatet som verdi i variabelen på venstre side
  - Eksempler:

- Datatyper i Java

Datatype	Beskrivelse	Eksempel
int	heltall	int k = 3;
double	desimaltall	double x = 3.14;
boolean	boolsk (logisk) verdi	boolean b = true;
char	tegn	char c = '@';
String	tekst	String s = "Hei på deg";

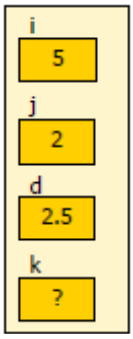
  Det finnes flere (short, long, byte, ..) som du gjerne kan lese om i boken, men som ikke er sentrale å kjenne til i dette kurset.

- Numeriske datatyper
  - int, double, float, short og long er numeriske datatyper i Java
  - Vi vil benytte int og double
  - Mulige verdier i en int-variabel er heltall
    - -2
    - 0
    - 72636
  - Mulige verdier i en double-variabel er desimaltall (flyttall):
    - 3.14
    - 0.00045
    - -3.72
    - 5.00000 (kan godt skrives som 5 i programmet ditt)

- Numeriske uttrykk
  Kan stå på høyresiden av tilordning til variabel av samme numeriske type (f. eks. int eller double)
  - Numeriske literaler – brukes direkte i programmet
    - 3.14
    - 0
  - Numeriske konstanter – deklareres med fast verdi
    - final int MAX_LENGDE= 10
    - final double KORT_PI= 3.14
  - Numeriske variable
  - Operasjoner med numerisk resultat

- Numeriske operasjoner
  - Numeriske operasjoner med egne operatorer (symboler):
    - De 4 regneartene: + - * /
    - Inkrementering og dekrementering: ++ --
  - Utført på kun heltall blir resultatet alltid heltall
  - Utført på desimaltall kan resultatet bli desimaltall

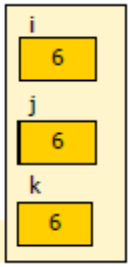
```
int i = 10/2;          // gir 5
int j = 10/4;          // gir 2 (heltallsdivisjon)

double d = 10/4.0;     // gir 2.5
int k = 10/4.0;        // kan ikke lagres i int =>kompilatorfeil
```



- Presedens
  - Operasjonene i et uttrykk utføres i en nøye bestemt rekkefølge
  - Rekkefølgen bestemmes først av presendensregler, eks
    - først innholdet i parenteser, deretter
    - ++ og -- (eneste operatorer som endret variablene i et uttrykk!)
    - * og /
    - + og -
  - For operasjoner med samme presedens: Fra venstre mot høyre

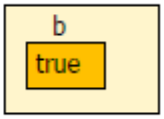
```
int i = 1 + 2*2;
int j = i++;        // i leses til j, deretter økes i med 1
int k = ++j;        // j økes med 1, detter leses verdien til k
```

Resultat etter de 3 programsetningene:



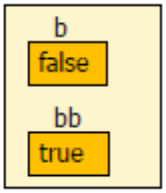
✗ Datatypen boolean
- I programmer har vi ofte bruk for å holde rede på om noe er sant eller usant, f.eks om (x > 0)
- Java har derfor en egen boolsk (logisk) datatype som bare har to lovlige verdier:
  - true
  - false

```
boolean b = true;          // Her får b verdien true
```



- Boolske (logiske) uttrykk
  - Logiske uttrykk gir boolsk verdi som resultat
  - Logiske operasjoner:
    - (0 == 5) testerom 0 har samme verdi som 5!!
    - Brukes for sammenligning av to uttrykk av samme type
    - Ikke å forveksle med = som brukes for tilordning!
    - Andre logiske operatorer:
      - < og >
      - <= og >=
      - !=
      - && og ||

```
boolean b = (0==5);          // Her får b verdien false
boolean bb = (0 <= 5);       // Her får bb verdien true
```



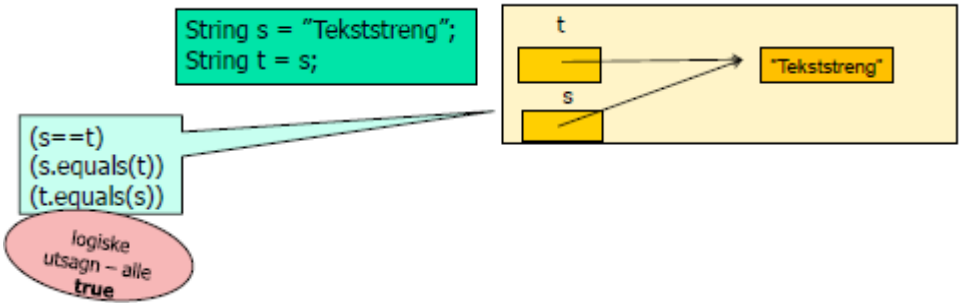
- Variabeltypen String
  - String-variable peker på en tekststreng

```
String s = "Hei på deg";
```

  - To tekster kan konkateneres (spleises) ved hjelp av tegnet +

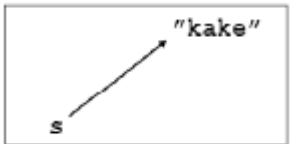
```
String t = s + ", Siri!";        // t peker nå på teksten "Hei på deg, Siri!"

System.out.println (t);          // skriver dette ut på skjermen
```

  - Merk at operatoren + dermed har to ulike betydninger – avhengig av om (minst) ett av argumentene er en tekst
- Kopiering og testing av String-verdier
  - String-variable er referanser (pekere) til tekstobjekter



  - Tekstobjekter kan imidlertid ikke endres – alle endringer fører til opprettelse av et nytt tekstobjekt, slik at andre tekstvariable som peker på samme objekt ikke endres (forskjellig fra arrayobjekter!)
  - Uttrykket (s==t) er bare true dersom de peker på den samme tekststrengen. Skal vi teste på om to tekststrenger er like(men ikke nødvendigvis den samme) må vi bruke (s.equals(t))

✗ Tekster og klassen String
- En tekststreng er en sekvens av tegn (null, en eller flere), f.eks.
  - ””
  - ”&”
  - ”Arne er student”
- Hver tekststreng vi lager er et objekt av typen String
- String-objektet kan i seg selv ikke endre (Immutable).
- En String-variabel (f.eks. String s) er en referanse til et slikt objekt
  - Resultatet av å utføre String s = ”kake” ;



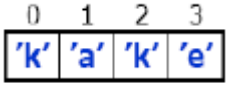
- For å finne lengden (dvs antall tegn i) en tekst:

```
int lengde = s.length();
```

✗ Bruk av spesialtegn
- Både i char-uttrykk og String-uttrykk kan vi ha mange ulike typer tegn
- Alle Unicode-tegn er tillatt
- Unicode er en standard som tillater tusenvis av tegn (ulike varianter fins; den som støttes av Java tillater 65536 ulike tegn)
- Alle tegnene kan angis som ’\uxxxx’ hvor hver x er en av
  0, 1, 2, ..., 9, A, B, C, D, E, F
  Eksempel: ’\u0041’ er tegnet ’A’
- Noen spesialtegn har egen forkortelse:
  - \t tabulator
  - \r vognretur (skriving starter først på linja)
  - \n linjeskift
  - \” dobbelt anførselstegn
  - \’ enkelt anførselstegn
  - \\ bakslask

✗ Unicode (http://www.unicode.org)

✗ De enkelte tegnene i en tekststreng
- Tegnene i en tekststreng har posisjoner indeksert fra 0 og oppover



- Vi kan få tak i tegnet i en bestemt posisjon:

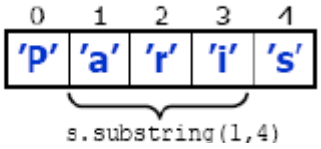
```
String s = "kake";
char c = s.charAt(1);
// Nå er c == 'a'
```

- Vi kan erstatte alle forekomster av et tegn med et annet tegn:

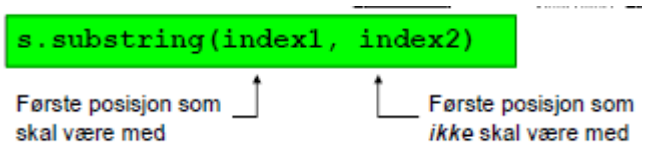
```
String s1 = "kake";
String s2 = s1.replace('k', 'r');
// Nå er s2 en referanse til tekststrengen "rare"
```

✗ Deler av en tekststreng
- Vi kan trekke ut en del av en tekststreng:

```
String s = "Paris";
String s1 = s.substring(1,4);
// Nå er s1 tekststrengen "ari"
```



- Generelt:



- Siste del av en tekststreng:

```
String s = "Paris er hovedstaden i Frankrike";
String s1 = s.substring(6);
// Nå er s1 tekststrengen "er hovedstaden i Frankrike"
```

✗ Alfabetisk ordning
- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Er s foran t i alfabetet?

```
int k = s.compareTo(t);
if (k < 0) {
    System.out.println("s er alfabetisk foran t");
} else if (k == 0) {
    System.out.println("s og t er like");
} else {
    System.out.println("s er alfabetisk bak t");
}
```

Husk at det er Unicode-verdien som brukes her og at det kan gi uventet resultat!

✗ Inneholder en tekst en annen?
- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Inneholder s teksten t?

```
int k = s.indexOf(t);
if (k < 0) {
    System.out.println("s inneholder ikke t");
} else {
    System.out.println("s inneholder t");
    System.out.println("Posisjon i s: " + k);
}
```

✗ Starter en tekst med en annen?
- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Starter s med teksten t?

```
boolean b = s.startsWith(t);
if (b) {
    System.out.println("s starter med t");
} else {
    System.out.println("s starter ikke med t");
}
```

✗ Slutter en tekst med en annen?
- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Slutter s med teksten t?

```
boolean b = s.endsWith(t);
if (b) {
    System.out.println("s ender med t");
} else {
    System.out.println("s ender ikke med t");
}
```

✗ Fra tall til tekst og omvendt
- For å konvertere fra tall til tekst:

```
String s1 = String.valueOf(3.14);
String s2 = String.valueOf('a');
String s3 = String.valueOf(false);
String s4 = "" + 3.14;
String s5 = "" + 'a';
String s6 = "" + false;
```

- For å konvertere fra tekst til tall:

```
int k = Integer.parseInt(s);
double x = Double.parseDouble(s);
```

og tilsvarende for de andre numeriske datatypene…

✗ Utskrift fra variable
- Vi har brukt variable i tilordning av verdi til andre variable
- Verdien i en variabel kan også skrives ut på skjerm
- System.out.println () kan skrive ut både numeriske verdier, tegn og tekst-verdier

```
class Utskrift {
   public static void main (String [] args) {
            String s = "Hei! ";
            int i = 16;
            char c = '#';
            System.out.println (s);
            System.out.println (i);
            System.out.println (c);
   }
}
```

Kjøre-eksempel

```
>
>javac Utskrift.java
>java Utskrift
Hei!
16
#
>
```

- To måter å skrive ut flere verdier på samme linje:
  - Bruk System.out.print () for hver verdi untatt den siste
  - Oppgi flere verdier med + mellom i System.out.println ()

```
class Utskrift {
   public static void main (String [] args) {
            System.out.print ("En forelesningstime varer ");
            System.out.println (" 45 minutter.");
   }
}
```

Kjøre-eksempel (likt for begge)

```
>javac Utskrift.java
>java Utskrift
En forelesningstime varer 45 minutter.
>
```

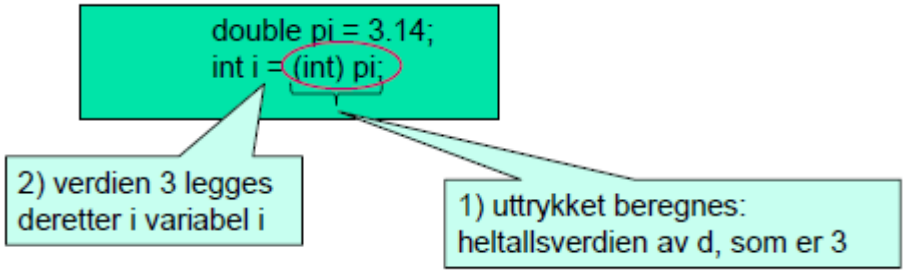
```
class Utskrift {
   public static void main (String [] args) {
            int timeLen = 45;
            System.out.println ("En forelesningstime varer " + timeLen + " minutter.");
   }
}
```

- Ting å passe på:
  - Variable kan ikke få navn med andre tegn enn tall og bokstaver, og må begynne med bokstav (vanligvis liten)
  - De kan ikke hete det samme som reserverte Java-ord
  - Vi kan ikke deklarere flere variable med samme navn
  - Variable kan ikke leses før de er tilordnet en verdi
  - Tips for å skjønne hva et program gjør: Tegn variablene, og oppdater verdier deres etter hvert som programsetningene utføres

✗ Konvertering
- Vi vet at numeriske beregninger ga heltall-svar hvis alle verdiene som inngikk var heltall, ellers double-svar
- Tilsvarende ved sammenligning på tvers av numeriske typer
  - Hvis det er både int og double variable i et uttrykk, gjøres de automatisk (implisitt) om til double før uttrykket beregnes.
  - NB! dette endrer ikke variable i uttrykket!

```
double pi = 3.14;
int i = 3;

if (i == pi) {}          // vurderes som (3.0 == 3.14) altså false
```

- Konvertering ved tilordning
  - Tilordning på tvers av numeriske typer
  - Plassere int-verdi i en double-variabel: Helt greit (konverteres implisitt – dvs uten at du ber om det)
  - Plassere double-verdi i en int-variabel: Må konverteres først (= si fra at du er villig til å miste noe informasjon)
  - Kalles eksplisitt konvertering – fordi du ber om det



  - NB! variabelen pi er ikke endret etter dette
- Objekter og pekere
  - Vi lager pekere og objekter når vi bruker arrayer og klasser
    - int [] a = new int [1024];
    - Student s = new Student();
    - String navn = "Jens";
  - Selve variabelen (a,s, navn) er en peker som:
    - Inneholder adressen til hvor objektet (array-objektet eller objektet fra klassen) er i hukommelsen .
    - Tegnes som en pil
- Ingenting i pekeren, "null"
  - Hva om vi ikke har laget et objekt ?
    - int [] b ;
    - Student t;
    - String etternavn;
  - Hva peker de på? Svar: ingenting!
  - Denne 'ingenting'-verdien heter null og kan testes på og brukes i tilordninger:
    - if (etternavn == null) print("etternavn mangler");
    - if (b != null) print("arrayen b finnes");
    - if (a != null && a[0] > 10) print("a[0] er stor nok");
      else print ("enten finnes ikke arrayen a eller a[0] er for liten");
    - s = null;
    - etternavn = null;

✗ Klasse-variabel (=statisk variabel)

- Setter vi static foran en variabel, er det er bare én felles variabel med det navnet for alle objektene.
- Setter vi static foran en metode, har den bare utsikt til :
  - sine egne lokale variable og parametere
  - andre statiske variable og metoder
  - klassenavnene
- Statiske metoder og variable kan man få adgang til både
  - via klassenavnet og punktum
  - via peker til et objekt av klassen og punktum
- Eksempel:

```
class B {
       static int i = 0;
       double x = 0.0;
}
class A
{      int k;

    public static void  main ( String[] args)  {
           B b1 = new B(), b2 = new B();
           // endre klassevariable (det er bare en felles)
           System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
           b1.i = 4;
           System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
           // endre objektvariabel (en kopi i hvert objekt)
           System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
           b1.x = 2;
           System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
      }
}
```

```
>java A
b1.i :0, b2.i:0
b1.i :4, b2.i:4
b1.x :0.0, b2.x:0.0
b1.x :2.0, b2.x:0.0
```

```
class A2
{       int k;  // objektvariabel 'k'
    public static void  main ( String[] args)  {
            k = 1;
 }}
```

```
>javac a2.java
a2.java:6: non-static variable k cannot be referenced from a static context
        k = 1;
        ^
1 error
```

```
class A2
{     int k;

    public static void  main ( String[] args)  {
            A2 aa = new A2();
            aa.k = 1;
      }
}
```

```
>javac A2.java

>
```

Innkapsling

- ✗ Innkapsling

Ikke alt i et objekt bør være synlig fra resten av programsystemet!

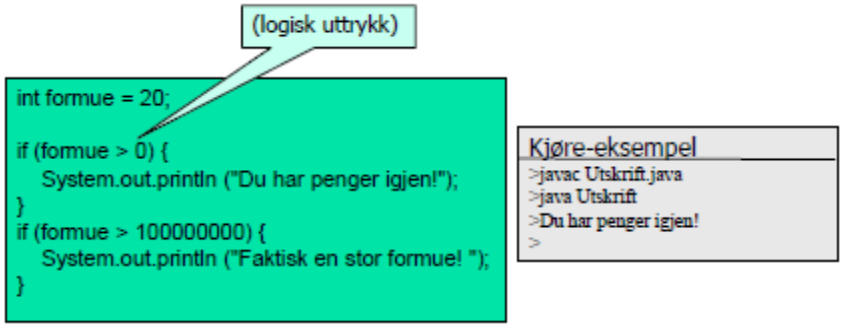
  - Vi ønsker ofte at resten av systemet bare skal se deler av et objekt
    - eks: int saldo i Konto1-objektet bør være skjult, resten av programmet skal bare bruke metodene settInn() og taUt().
  - Vi kan regulere tilgangen til variable og metoder ved å sette en av følgende modifikatorer foran en variabel- eller metodedeklarasjon:
    - private
    - public
    - protected

- ✗ Hvis alle .java filene ligger på samme filområde
  - ingenting foran en deklarasjon/metode
    - fullt tilgjengelig for all annen kode på samme filområde
    - usynlig /sperret for kode på andre filområder.
  - private
    - kun synlig fra metoder deklarert i samme klasse, usynlig/sperret for all annen kode
  - public:
    - Synlig for ”all” annen kode (forutsetter katalogen referert i CLASSPATH)
    - Nødvendig for main
    - Ellers det samme som <ingenting> i vårt tilfelle

    Slik delvis sperring av adgang sikrer oss at vi selv bestemmer hvordan en variabel i en klasse skal endres.

IF-SETNINGER

- ✗ Forgreninger: if
  - Ofte vil vi at maskinen skal utføre ulike instruksjoner avhengig av for eksempel en verdi i en variabel
  - Da kan vi bruke if-konstruksjonen i Java for å bestemme under kjøring om en eller flere setninger skal utføres



  - Andre eksempler på logiske uttrykk (betingelser) å teste på:

```
if (!false) {
        System.out.println ("Utføres alltid")
}
```

```
if (true) {
        System.out.println ("Utføres alltid")
}
```

```
if (0 != 0) {
        System.out.println ("Utføres aldri");
}
```

```
int i = 5;
if ((i < 10) && (i > 10)) {
        System.out.println ("Utføres aldri");
}
```

- ✗ Forgreninger: if – else
  - Når man ønsker alternative setninger utført hvis betingelsen er usann brukes konstruksjonen if-else

```
int formue = 20;

if (formue > 100000000) {
   System.out.println ("Faktisk en stor formue! ");
} else if (formue > 1000000) {
   System.out.println ("Faktisk ganske romslig!");
} else {
 System.out.println ("Men ikke svært mye!");
}
```

  - Slike if-else-setninger kan kobles i kjede med flere mulige utfall

LØKKER

- ✗ Løkker
  - Ofte ønsker vi å utføre en eller flere programsetninger mer enn én gang
  - Dette gjøres ved hjelp av løkker
  - Hovedsakelig to typer behov:
    - Utfør setningen(e) inntil en betingelse oppfylles -> while-løkke
    - Utfør setningen(e) et bestemt antall ganger –> for-løkke

- ✗ while-løkker
  - Vi kan utføre en blokk med setninger flere ganger ved hjelp av en while-løkke

```
while (<logisk uttrykk>) {
    <setning 1;>
    <setning 2;>
    .....
    <setning n;>
}
```

  - Hvis det logiske uttrykket er true, utføres blokken i while-løkka
  - Når blokken som hører til løkka er utført, vil programmet returnere til toppen og evaluere det logiske uttrykket på nytt
  - Hvis det logiske uttrykket er false, fortsetter utførelsen etter blokken.
  - Eksempel:

```
class SkrivLinjer {
    public static void main (String [] args) {
        int k = 1;
        while (k <= 5) {
            System.out.println(
                "Nå har k verdien " + k);
            k = k + 1;
        }
        System.out.println("Nå er k lik " + k);
    }
}
```

Ved Kjøring:

```
> java SkrivLinjer
Nå har k verdien 1
Nå har k verdien 2
Nå har k verdien 3
Nå har k verdien 4
Nå har k verdien 5
Nå er k lik 6
>
```

✗ Evig løkke

- Dersom testen i while-løkka aldri blir usann(false), vil utførelsen av while-løkka aldri stoppe. Dette kalles en evig løkke.
- To eksempler:

```
class EvigLokkeOpplagt {
    public static void main (String [] args) {
        while (true) {
            System.out.println("INF 1000");
        }
    }
}
```

```
class EvigLokkeIkkeSaOpplagt {
    public static void main (String [] args) {
        int i = 1, j = 2;
        while (i < j) {
            System.out.println("Nå er i < j (i=" +
                i + ", j=" + j + ")");
            i++; j++;
        }
    }
}
```

- Ved kjøring:

```
> java EvigLokkeOpplagt
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000
INF 1000

. . .
. . .
```

```
> java EvigLokkeIkkeSaOpplagt

. . .
. . .

Nå er i < j (i=82155, j=82156)
Nå er i < j (i=82156, j=82157)
Nå er i < j (i=82157, j=82158)
Nå er i < j (i=82158, j=82159)
Nå er i < j (i=82159, j=82160)
Nå er i < j (i=82160, j=82161)
Nå er i < j (i=82161, j=82162)
Nå er i < j (i=82162, j=82163)
Nå er i < j (i=82163, j=82164)

. . .
```

(Kan stoppes med Ctrl+C)

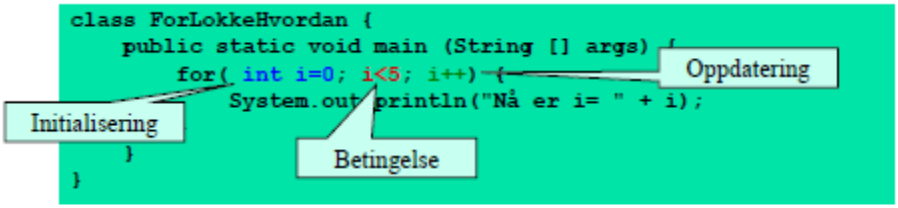
- Variant av while – do-while
  - Formen på en do-while løkke:

```
do {
      <setning 1;>
      <setning 2;>
      .....
      <setning n;>
} while (<logisk uttrykk>);
```

  - Noen foretrekker denne fremfor while-løkker når løkke-innmaten alltid skal utføres minst en gang.
- For-løkker
  - En annen måte å få utført en instruksjon (eller blokk) mange ganger er ved hjelp av en for-løkke:

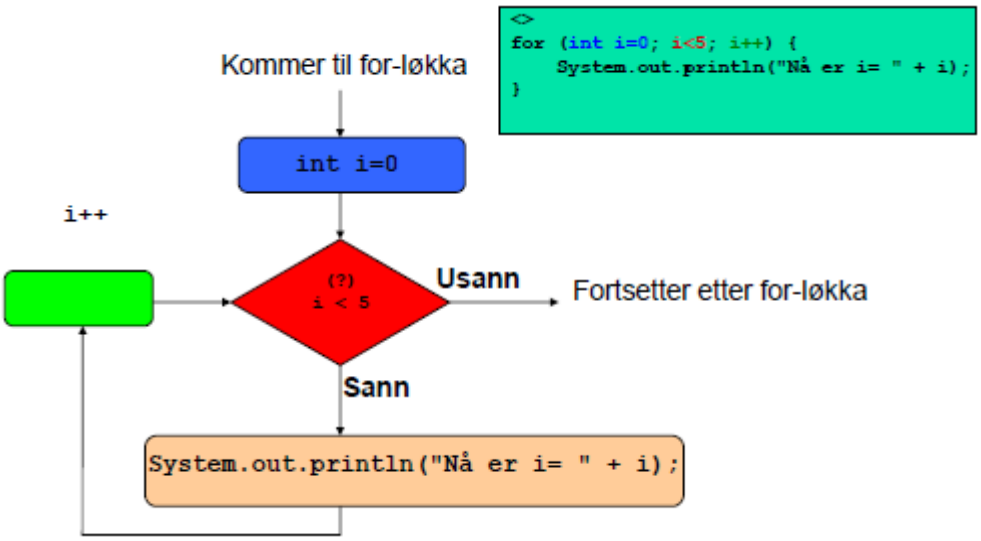
```
for (<initialisering>; <betingelse>; <oppdatering>){
      <setning 1;>
      <setning 2;>
      ....
      <setning n;>
}
```

  - Eksempel:




```
> java ForLokkeHvordan
Nå er i= 0
Nå er i= 1
Nå er i= 2
Nå er i= 3
Nå er i= 4
>
```

✗ Flytdiagram: Hvordan for-løkka virker



✗ Nesting av løkker
- Det er ofte behov for å neste løkke-setninger inne i hverandre
- Man må passe på å bruke forskjellig "tellevariabel" i hver løkke
- I dette eksemplet er i og j brukt.

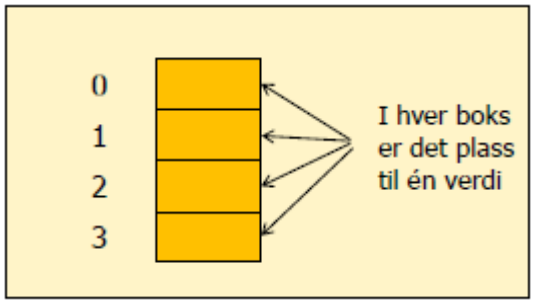
```
class NestetForLokke {
    public static void main (String [] args) {
        for( int i=1; i<=10; i++) {
            for( int j=1; j<=10; j++) {
                int produkt = i * j;
                System.out.println(i + "*" + j +"=" + produkt);
            }
        }
    }
}
```

✗ Nesting av løkker – variant
- Nedenfor har vi endret den innerste for-løkken slik at den ikke skifter linje for hver produkt, men kun når i inkrementeres
- Senere vil vi se hvordan tabellen kan lages med rette kolonner

```
class NestetForLokke {
    public static void main (String [] args) {
        for( int i=1; i<=10; i++) {
            for( int j=1; j<=10; j++) {
                int produkt = i * j;
                System.out.print(i + "*" + j +"=" + produkt);
            }
            System.out.println ();
        }
    }
}
```

ARRAY

- Arrayer
  - En variabel holder kun én verdi
  - En array kan holde mange verdier av samme type
  - Verdiene i en array med lengde N får hver sin indeks (=posisjon) fra 0 til N-1
  - En array med lengde 4 kan tegnes slik:



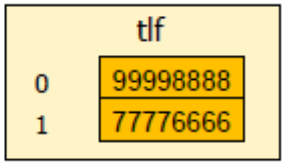
- Deklarere og opprette en array
  - Deklarere en array (angi type og gi den navn)
    dataType [] arrayNavn;
  - Opprette en array (sette av plass i hukommelsen)
    arrayNavn = new dataType [N]
  - Deklarere og opprette samtidig:
    datatype[] arrayNavn = new dataType [N];
  - Eks:

```
int [] soltimer = new int [31];
int lengde = 31;
double [] nedbør = new double [lengde];
String [] navn = new String [5];
```

lengden kan være en variabel

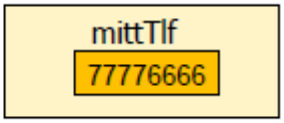
- Bruk av arrayen
  - Plasser i en array brukes som variable: Oppgi arraynavn og indeks
  - Tilordning

```
int [] tlf = new int [2];
tlf [0] = 99998888;
tlf [1] = 77776666;
```



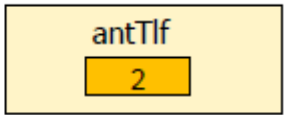
  - Lesing

```
int mittTlf = tlf [1];
```



  - Få tak i lengden på arrayen

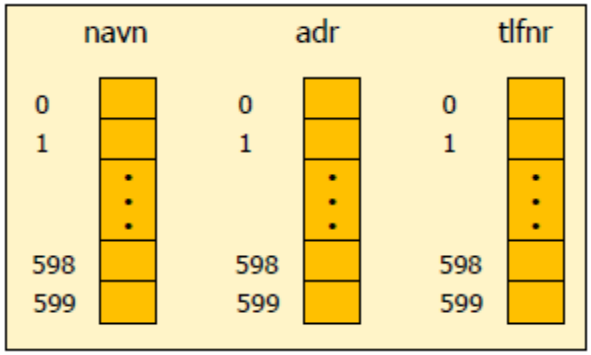
```
int antTlf = tlf.length;
```



✗ Eksempel – arrayer

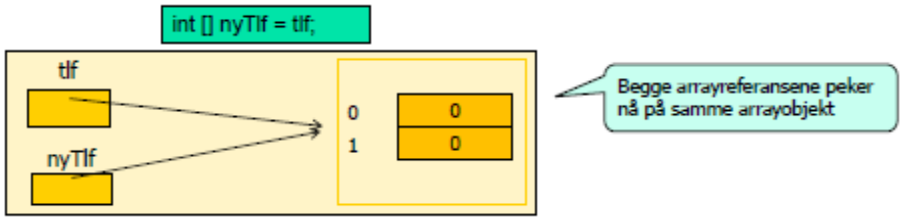
- Anta at vi ønsker å lagre navn, adresse og telefonnummer for de som følger et bestemt kurs med maksimalt 600 studenter

```
String [] navn = new String [600];
String [] adr = new String [600];
int [] tlfnr = new int [600];
```

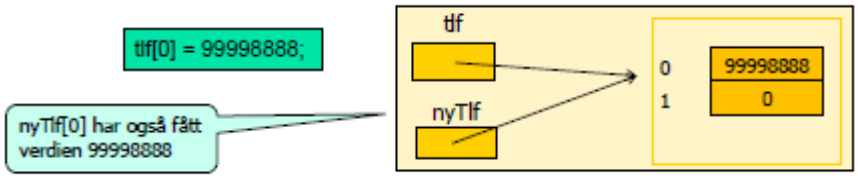


✗ Kopiering av arrayreferanse

- Vi setter en ny arrayreferanse lik den vi har

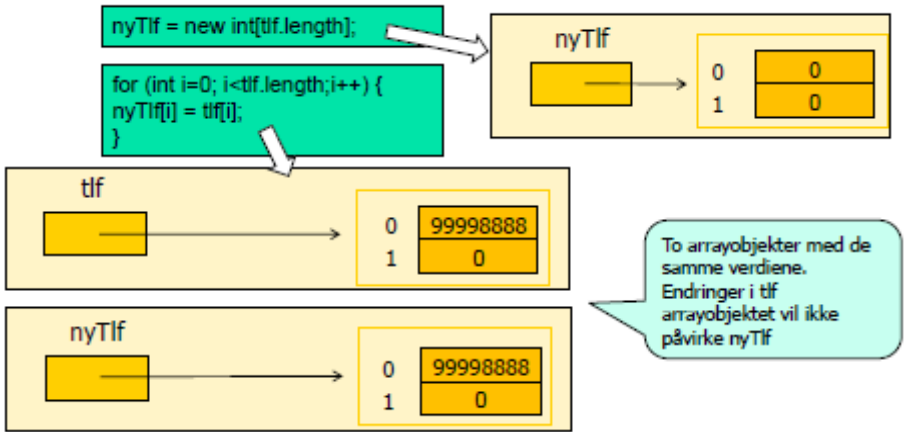


- Hvis vi nå endrer verdiene i arrayobjektet – vil endringene synes uansett hvilken arrayreferanse vi leser av



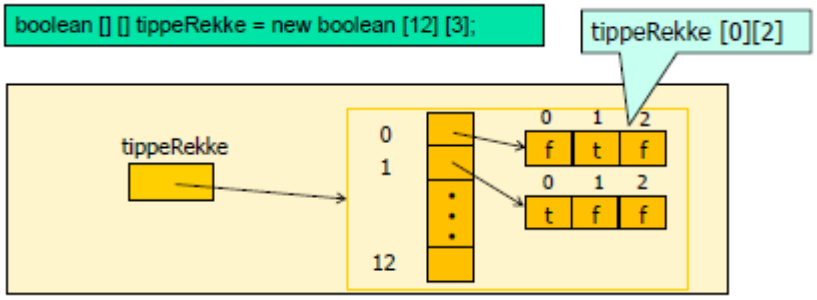
✗ Kopiering av array

- Skal vi ha en egen kopi av alle plassene i arrayen må vi sette av ny plass med new, og kopiere en og en verdi, f eks. i en for-løkke



- Flerdimensjonal array
  - Dersom vi ønsker å representere en todimensjonal tabell av verdier av samme type kan vi deklarere en todimensjonal array

```
boolean [] [] tippeRekke = new boolean [12] [3];
```



  - Dette kan generaliseres for N dimensjoner – her 3

```
boolean [] [] [] tippeKupong = new boolean [10][12] [3];
```

```
class NestetForLokke {
    public static void main (String [] args) {
        for( int i=1; i<=10; i++) {
            for( int j=1; j<=10; j++) {
                int produkt = i * j;
                System.out.print(i + "*" + j +"=" + produkt);
            }
            System.out.println ();
        }
    }
}
```

- Initialisering og utskrift av array
  - Vi ønsker å lagre navnet på ukedagene – og bruker en array med 7 plasser av typen String
  - Deretter skriver vi disse ut på en linje med blanke imellom

```
class NestetForLokke {
    public static void main (String [] args){
        String [] ukedag =
{"mandag","tirsdag","onsdag","torsdag","fredag","lørdag","søndag"};

        for (int i = 0; i<7; i++) {
            System.out.print (ukedag[i] + "    ");
        }
        System.out.println ();
    }
}
```

✗ Arrayer av pekere til objekter

- Vi kan lage arrayer av pekere til objekter (men ikke av objektene direkte) omlag på samme måte som vi lager arrayer av int, double og String
- Har vi deklarert klassen Kurs {...} kan vi lage array som følger:

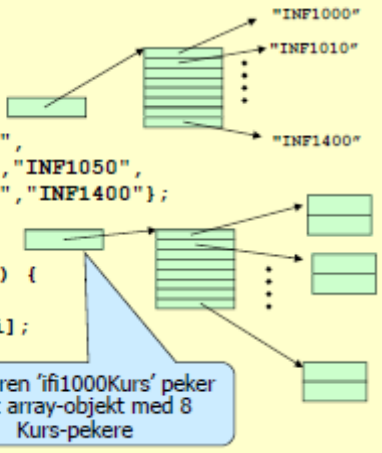
```
Kurs [] ifiKurs;
```
Her deklareres 'bare array-pekeren

```
ifiKurs = new Kurs[120];
```
Her lages array-objektet med 120 'tomme' pekere

```
ifiKurs[0] = new Kurs();
```
Her settes det første pekeren i array-objektet til å peke på et nytt Kurs-objekt

- Eksempel:

```
class Kurs {
   String kurskode;
   int studiepoeng=10;

   void skrivUt() {
     System.out.println("Kurs med kode:"
         + kurskode + ", og stp:"
          + studiepoeng);
   }
}

class KursRegister2 {
  public static void main(String args []) {

    String [] kursKoder= {"INF1000","INF1010",
                          "INF1020","INF1040","INF1050",
                           "INF1060","INF1070","INF1400"};
    Kurs [] ifi1000Kurs = new Kurs[8];

    for(int i = 0; i < kursKoder.length; i++) {
        ifi1000Kurs[i] = new Kurs();
        ifi1000Kurs[i].kurskode = kursKoder[i];
        ifi1000Kurs[i].skrivUt();
    }
  }
}
```



arraypekeren 'ifi1000Kurs' peker til et array-objekt med 8 Kurs-pekere

- Eksekvering av KursRegister2

```
>java KursRegister2
Kurs med kode:INF1000, og stp:10
Kurs med kode:INF1010, og stp:10
Kurs med kode:INF1020, og stp:10
Kurs med kode:INF1040, og stp:10
Kurs med kode:INF1050, og stp:10
Kurs med kode:INF1060, og stp:10
Kurs med kode:INF1070, og stp:10
Kurs med kode:INF1400, og stp:10
```

KLASSER OG METODER

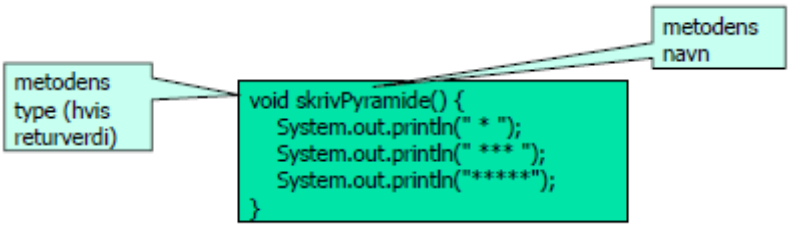
- ✗ Metoder – hvorfor?
  - Ofte har vi bruk for å utføre (omtrent) samme instruksjoner flere steder i et program, eller i mange programmer. Gjenbruk
    - reduserer størrelsen på programmet
    - bedrer oversikten
    - forenkler vedlikeholdet
    - gir færre feil
  - Da er det nyttig å kunne samle en eller flere instruksjoner med et selvvalgt navn, som så kan benyttes en eller flere ganger i programmet vårt
- ✗ Metoder – eksempler
  - Vi har brukt en del metoder allerede, for eksempel

```
System.out.println(" * ");
double d = tastatur.inDouble();
String s = tastatur.inWord();
int i = (int) Math.round(d);
```

  - Også main() er en metode:

```
public static void main(String[] args) {
.....
}
```

  - Vi skal nå se nærmere på hva metoder egentlig er, hvordan de brukes og hvordan vi kan lage våre egne metoder.
- ✗ Metoder
  - En metode er en navngitt blokk med instruksjoner (programsetninger) som vi får utført ved å angi metodens navn
  - Den deklareres inne i en klasse, og kan senere kalles på hver gang vi ønsker den utført
  - Eksempel på deklarasjon:

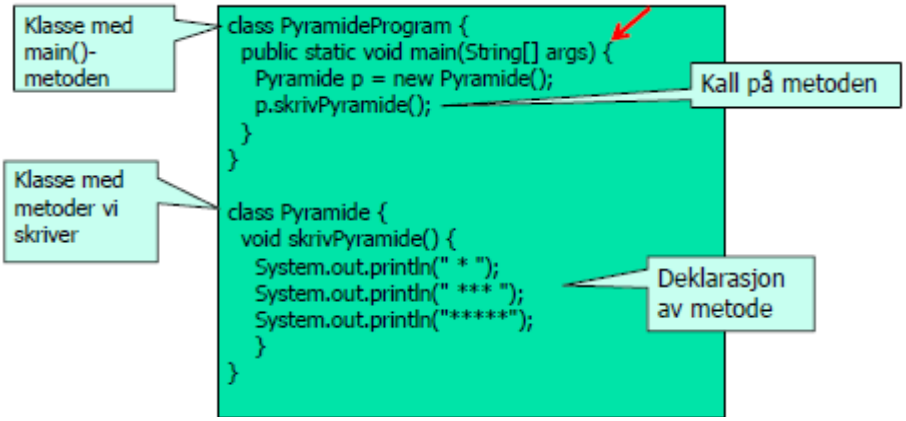


  - Bruk av (kall på) metoden:

```
skrivPyramide();     // Her utføres setningen i metoden
```

- Programstruktur med metoder
  - Alle Java-programmer består av en eller flere klasser, med en eller flere metoder. Metoder deklareres inne i klasser.
  - Klassen med metoden main() i, har samme navn som filen, og main() – som er der utføring av programmet starter og slutter – er den eneste metoden vi har skrevet, med all funksjonalitet i seg.
  - En klasse med main()-metoden, der program-utførelsen starter og avsluttes
  - En klasse med datastruktur, og metoder som bearbeider denne

- Enkel programstruktur med metoder
  - Enkel mal for programmer med metoder (her et program fra filen PyramideProgram.java)

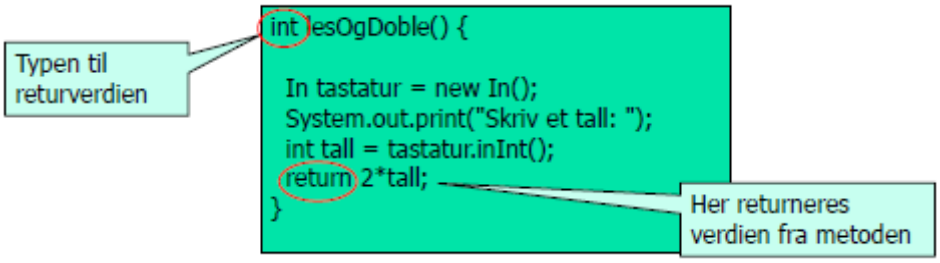


- Metode med returverdi
  - Ofte ønsker vi at resultatet fra en metode skal kunne brukes videre i resten av programmet, som for eksempel:

```
In tastatur = new In();

int k = tastatur.inInt();
```

  - Her inInt() en metode i klassen In, som vi får tilgang til via pekervariabelen tastatur.
  - Metoden inInt() returnerer et heltall (en int).
  - Dette heltallet tar vi vare på i variabelen k

- Deklarasjon av metode med returverdi
  - Her er en metode som leser et heltall og returnerer det dobbelte:



  - Metoder som ikke returnerer noen verdi deklareres med typen void.

- Metode med returverdi: Eksempel
  - Metoden inInt() leser inn et vilkårlig heltall. Noen ganger ønsker vi å sikre oss at vi får et positivt heltall. Vi kan da lage en egen metode for dette:

```
int lesPositivtHeltall() {

  In tastatur = new In();
  int tall;
  do {
    System.out.print ("Gi et positivt tall: ");
    tall = tastatur.inInt();
  } while (tall <= 0);

  return tall;
}
```

Deklarasjon av metoden

- Program med bruk av metode med returverdi: Eksempel

```
import easyIO.*;
class TestPosTall {
    public static void main(String[] args) {
            PosTall pt = new PosTall();

            int i = pt.lesPositivtHeltall();
            System.out.println(i + " er et positivt heltall");
    }
}
class PosTall {
    int lesPositivtHeltall() {
            In tastatur = new In();
            int tall;
            do {
                System.out.print("Gi et positivt tall: ");
                tall = tastatur.inInt();
            } while (tall <= 0);
            return tall;
    }
}
```

- Metoder med parametere
  - Ofte ønsker vi at samme metode skal kunne brukes for litt ulike input-verdier, f. eks:

```
System.out.println(" * ");

System.out.println("Hei verden");
```

  - Her er println() en metode som tar en tekst som input (parameter)
  - Metoden gjør det samme uansett hvilken String-verdi vi kaller med: Skriver den ut på skjermen

✗ Metode med parameter: Eksempel

- Parameteren avgjør antall linjer i ”pyramiden”:

```
class PyramideProgram {
   public static void main(String[] args) {
            Pyramide p = new Pyramide();
            p.skrivPyramide(4);
   }
}
class Pyramide {
   void skrivPyramide(int antall) {

            for (int i = 1; i<=antall; i++) {
               for (int j = 1; j<=i; j++) {
                        System.out.print("*");
               }
               System.out.println ();
            }
   }
}
```

parameter angis med type og navn som skal benyttes inni metoden

✗ Gangetabell-metode: Eksempel

- Metode som skriver ut n-gangen fra 1 til 10

```
void gangeTabell(int n) {
   for (int i = 1; i <= 10; i++) {
      System.out.println(i * n);
   }
}
```

n

2

- Eksempler på kall på metoden

```
gangeTabell(2);
gangeTabell(15);
```

n

15

✗ Bruk av gangetabell-metode: Eksempel

```
import easyIO.*;
class GangeProgram {
   public static void main(String[] args) {
            In tastatur = new In();
            Utregning utr = new Utregning();
            System.out.print("Hvilken gangetabell vil du skrive? ");
            int tall = tastatur.inInt();
            utr.gangeTabell(tall);
   }
}
class Utregning {
   void gangeTabell(int n) {
            for (int i = 1; i <= 10; i++) {
               System.out.println(i * n);
            }
   }
}
```

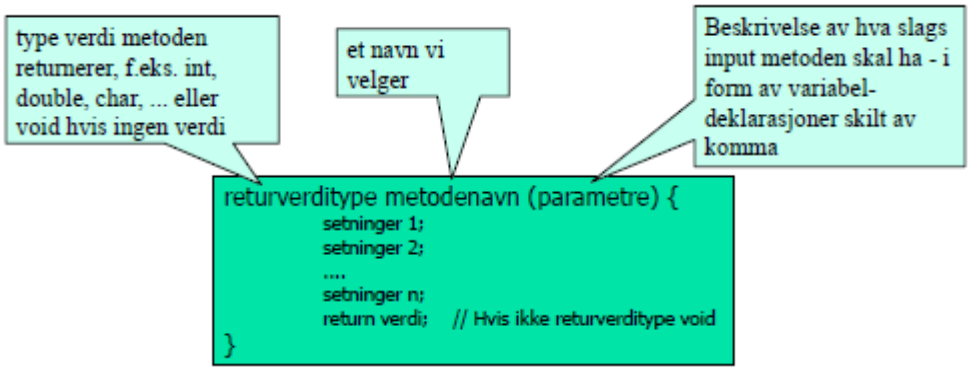
- Metoder med flere parametere: Eksempel
  - Vi kan utvide gangeTabell-metoden med en parameter som angir hvor mye av tabellen som skal skrives:

```
void gangeTabell(int n, int slutt) {
    for (int i = 1; i <= slutt; i++) {
        System.out.println(i * n);
    }
}
```

  - Eksempel på kall på metoden:

```
gangeTabell(2, 20);
gangeTabell(15, 10);
```

- Oppsummering, metoder:
  - Deklarasjon:
    - Java-programmene så langt i kurset består av to klasser, startklassen med main og en annen klasse hvor det kan det befinne seg en eller flere metoder.
    - De metodene vi ser på så langt i kurset har følgende form:



  - Kall:
    - Når vi benytter en metode sier vi at vi kaller på metoden
    - Kall på metode uten parametere – eks:

```
minMetode();
```

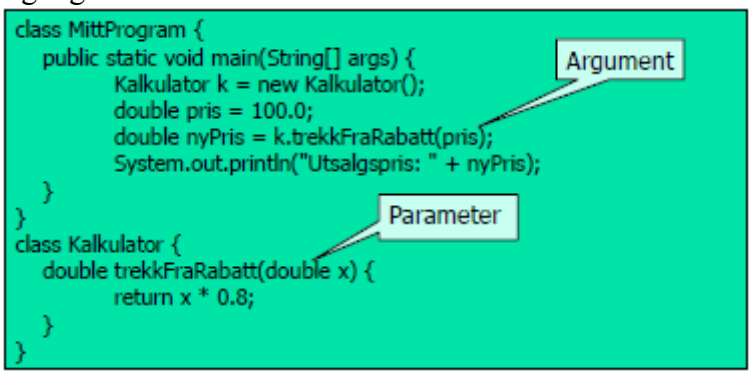
    - Kall på metode med parametere – eks:

```
minMetode2 (2, "Hei!");
```

    - Kall på metode med parameter som returnerer en verdi – eks:

```
int i = minMetode3 (10);
```

- Parametere og argumenter



  Merk: argumenter til metodekallet kalles også for aktuelle parametre mens parametre i deklarasjonen da kalles formelle parametre.

  - Parametre
    - Deklareres mellom parentesene i toppen (= den første linjen) av metode-deklarasjonen. De er "vanlige variable" som bare eksisterer inne i metoden og så lenge denne eksekverer.
  - Argumenter
    - Verdier som oppgis mellom parentesene når vi kaller på en metode.
    - Antall argumenter må samsvare med antall parametre i metoden
    - Argumentenes datatyper må samsvare med datatypen til tilsvarende parameter.

✗ En klasse er noe - en metode gjør noe

- Metoder: Vi deler opp handlingene i programmet i metoder. En metode er da noen vanlige programsetninger som vi setter krøll-parenteser rundt. Metoden gjør det navnet på metoden sier. Vi velger selv navnet på de metodene vi lager.
  - EKS Banksystem: En metode for hver av handlingene: innskudd, uttak, beregnRenter, skrivRapport,...
- Klasser: Vi deler dataene og metodene i programmet opp i deler slik at hver av disse (klassene) tilsvarer en naturlig del av problemet:
  - EKS Banksystem: En klasse for hver av Banken, Kunde, Konto,...
- En klasse er noe, en metode gjør noe.
  Metodene er inne i klasser.
  (og alle setninger i et Javaprogram som 'gjør noe' som: tilordning, if, while,.. er inne i moder.)

- Annen type oppsett for programmer:
  - Minst to klasser, en med "main" som starter den andre

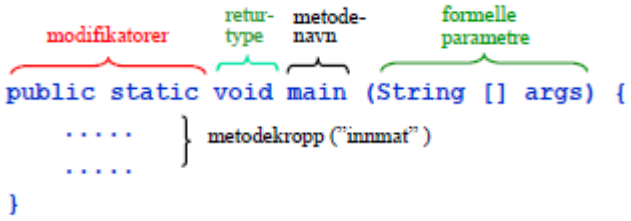
```
import easyIO.*;

class MittProgram {
    public static void main (String[] args) {
            // her lager vi et objekt av den andre klassen
               Student s = new Student();
               // her kan vi kall på metodene i Student - eks:
               s.skriv();
               System.out.println("Programmet ferdig - ha det");
        }
}

class Student {
    String navn;  // evt. data i klassen'Student'

    Student () {
               // startmetode, f.eks initialisering
             navn = "Ola";
    }

    void skriv() {
          // her er en egen metode
          System.out.println("Navnet mitt er:" + navn);
    }
}
```

  - Under kjøring

```
class MittProgram {
    public static void main (String[] args) {
               Student s = new Student();
               s.skriv();
               System.out.println("Programmet ferdig - ha det");
        }
}

class Student {
      String navn;
        Student () {
            navn = "Ola";
        }
        void skriv() {
           System.out.println("Navnet mitt er:" + navn);
        }
}
```

```
Navnet mitt er:Ola
Programmet ferdig - ha det
```

- ✗ Metode-deklarasjon (lage metoden)
  - En metode er essensielt en navngitt 'blokk' med setninger som vi kan få utført hvor som helst i et program ved å angi metodens navn.
  - Beskrivelsen av hva metoden skal hete og hvilke setninger som skal ligge i metoden kalles en metode-deklarasjon.
  - main-metoden er et eksempel på en metode-deklarasjon:



  - En klasse kan inneholde vilkårlig mange metode-deklarasjoner.
  - static brukes (nesten) ikke unntatt for main
- ✗ Å benytte en metode
  - Når vi benytter en metode sier vi at vi kaller på metoden.
  - For å kalle på en metode uten parametre, skriver vi ganske enkelt som en setning:
    metodenavn();
  - For å kalle på en metode med parametre, må vi i tillegg oppgi like mange verdier som metoden har parametre, og i'te verdi må ha samme datatype som i'te parameter i metodedeklarasjonen. Eksempel:
    metodenavn2(34.2, 53, 6);
  - Hvis metoden returnerer en verdi, kan vi velge om verdien skal tas vare på eller ikke når metoden kalles. Eksempel på å ta vare på verdien:
    int alder = metodenavn3(25.3, 52, 7);

- ✗ Metode uten input/output: Eksempel
  - Følgende metode skriver ut fire linjer med stjerner på skjermen:

```
void skrivStjerner () {
   String s = "****************************";
   System.out.println(s);
   System.out.println(s);
   System.out.println(s);
   System.out.println(s);
}
```

  - Forklaring:
    - void er en returverditype som forteller at metoden ikke gir noe output
    - skrivStjerner er det navnet vi har valgt å gi metoden

- Levetiden til parametre og variable
  - Vi kan ha adgang til tre typer variable i en metode:
    - Objektvariable: dette er variable som er deklarert på klassenivå, inne i klassen, men utenfor metodene.
    - Lokale variable: dette er variable som deklareres inni metoden. Disse er definert fra og med der deklarasjonen gjøres og til slutten av blokken de er deklarert i.
    - Parametre: dette er variable som deklareres i hodet på metoden. Disse er definert i hele metodekroppen.
  - Viktig: ved gjentatte kall på en metode er det et nytt sett med lokale variable og parametre som lages hver gang (men det er de 8 samme objektvariablene hvis metoden er i samme objekt).
  - Eksempel:

```
class  Start {
    public static void main (String[] args) {
       Variabeltyper vt = new Variabeltyper();
       int intervall = 3;      // 'interval' er lokal variabel
       vt.økTid(intervall);
        vt.økTid(intervall);
      }
}

class Variabeltyper {
   int tid = 0;              // 'tid' er objektvariabel

   void økTid (int t) {   // 't' er parameter
       tid += t;
     System.out.println(tid);
   }
}
```

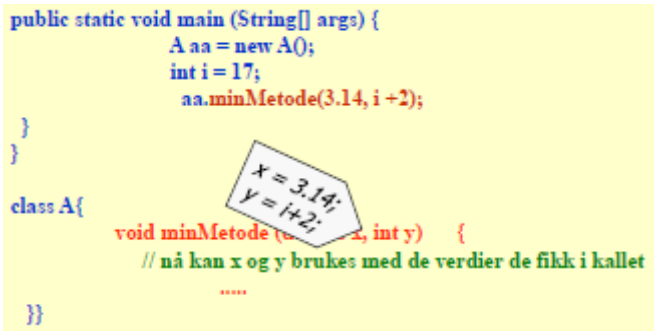
- Parametre (formelle parametere) og argumenter (aktuelle parametere)

```
class Eksempel {
   public static void main (String[] args) {
      A aa = new A();
      aa.minMetode(3.14, 365);
   }
}

Class A {
   void minMetode (double x, int y) {
      .....
   }
}
```

Argumenter

Parametre

- ✗ Verdien til parmeterene kopieres over til metoden
  - Når vi kaller metoden (bruker navnet i en annen metode), så overføres verdienene til de parameterene som ble brukt i kallet slik:



    De verdiene som ble brukt ved kallet, blir kopiert over i parameterene før setningene i metoden blir utført
- ✗ Oppsummering om metoder
  - Deklarasjon (lage metoden) :
    - Man pakker sammen de handlinger som hører sammen (gjør noe sammen) med krøllparenteser, og gir metoden et navn med vanlige parenteser bak navnet.
    - Man må også si om metoden returnere noe:
      - Returnerer ingenting: sett da void foren navnet
      - Returneren en verdi, sett typen til verdien foran navnet (Eks: int, double, int[],..) Metoden må da si return XXX; et sted i koden og hvor XXX er et uttrykk av den typen metoden skal returnere (eks return i +14;).
      - (Hvis metoden bare tilhører klassen, skrives static foran returtypen)
    - Hvis metoden trenger noen data som den skal jobbe med for å gjøre 'jobben', settes de med type inn i parentesen bak navnet
      - Eks: double kvadratrot(double x) {...; return ... }
  - Bruk / kall på metoden:
    - Man nevner navnet (i koden til en metode) med evt. Parametere og med navnet på objektet b til klassen 'foran punktum':
      y = 2.0 + b.kvadratrot(x*3.14);
- ✗ Konstruktører – startmetoder i klasser
  - Når vi lager et objekt av en klasse med new, kaller vi egentlig en metode som heter det samme som klassen (derfor parentesen bak klassenavnet).
  - Vi får automatisk med en slik konstruktør-metode fra oversetteren dersom vi ikke skriver en slik konstruktør selv. Den vi får automatisk er uten parametere og gjør ingen ting.
  - Konstruktører nyttes i all hovedsak til å gi fornuftige startverdier for variable i objektet som dannes.
  - De konstruktørene vi skriver kan ha parametere.
  - Konstruktørene skal ikke ha noen type foran seg, heller ikke void.
  - Vi kan ha flere konstruktører i en klasse, men da må parameterne være ulike i antall eller typen av parametrene

- Eksempel, klasse med en konstruktør

```
class Student {
   String navn;
   Kurs [] mineKurs = new Kurs[3];

   Student(String navn, Kurs [] k){
     this.navn = navn;
     for (int i = 0; i<k.length; i++ ){
         mineKurs[i] = k[i];
         mineKurs[i].antStudenter++;
     }
   }
```

- Eksempel, klasse med to konstruktører

```
class Student {
   String navn;
   Kurs [] mineKurs = new Kurs[3];

   Student() {
     mineKurs = new Kurs[0];
   }

   Student(String navn, Kurs [] k){
     this.navn = navn;
     for (int i = 0; i<k.length; i++ ){
         mineKurs[i] = k[i];
         mineKurs[i].antStudenter++;
     }
   }
```

- this.
  - Av og til trenger vi en peker til det objektet metoden vi utfører er inne i. Java-ordet this gir oss alltid det.
  - Brukes i to situasjoner:
    - Vi har en konstruktor, og parametrene til denne heter det samme som objekt-variable i objektet. Eks:

```
class A {
   int antall;
   A (int antall ){
        this.antall = antall;
   }
} // end A
.. A apek = new A(12);
```

    - Vi skal kalle en metode i et annet objekt (gjerne av en annen klasse). Da kan vi bruke this for å overføre en parameter til metoden om hvilket objekt kallet kom fra.

EasyIO

Les fra skjerm

- ✗ Innlesning fra terminal
  - Innlesning fra terminal kan gjøres på flere måter i Java. I INF1000 bruker vi pakken easyIO. Du må da skrive i toppen av programmet:

```
import easyIO.*;
```

  - Inne i klassen skriver vi følgende før vi kan starte innlesning:

```
In tastatur = new In();
```

  - Så kan vi lese inn fra terminal (=tastatur), f.eks. et heltall:

```
int radius;
System.out.print("Oppgi radiusen: ");
radius = tastatur.inInt();
```

- ✗ Eksempel – innlesning med sjekk
  - Lag et program som leser et heltall mellom 1 og 100 fra terminal.
  - Hvis det innleste tallet ikke ligger i det lovlige intervallet, skal programmet be om nytt tall.
  - Dette gjentas inntil brukeren skriver et lovlig tall.
  - Skriv til slutt ut en tekst som inneholder tallet.

```
import easyIO.*;
class LesVerdiSjekk {
    public static void main (String[] args) {
        In tast = new In();
        System.out.print("Oppgi verdi (1,2,...,100): ");

        int verdi = tast.inInt();

        while ((verdi < 1) || (verdi > 100)) {
            System.out.println("Ulovlig verdi!");
            System.out.print("Prøv igjen: ");
            verdi = tast.inInt();
        }

        System.out.println("Du oppga verdien " +
            verdi);
    }
}
```

  - Kompilering og kjøring:

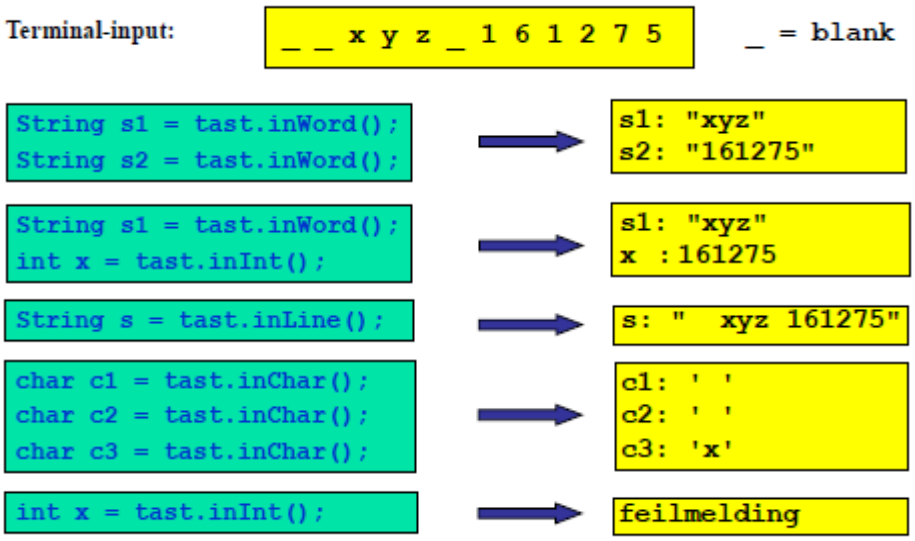
```
> javac LesVerdiSjekk.java
> java LesVerdiSjekk
Oppgi verdi (1,2,...,100): 101
Ulovlig verdi!
Prøv igjen: 0
Ulovlig verdi!
Prøv igjen: 3
Du oppga verdien 3
>
```

- Lesemetoder

```
// Opprette forbindelse med tastatur:
In tastatur = new In();

// Lese et heltall:
int k = tastatur.inInt();

// Lese et desimaltall:
double x = tastatur.inDouble();

// Lese et enkelt tegn:
char c = tastatur.inChar();

// Lese et enkelt ord:
String s = tastatur.inWord();

// Lese resten av linjen - evt neste linje hvis tomt:
String s = tastatur.inLine();
```

- Hvilken lesemetode skal jeg velge?
  - Først:
    - importere easyIO og åpne forbindelse til tastaturet.
  - Lese item for item:
    - For å lese et heltall: inInt() [egentlig: tastatur.inInt()]
    - For å lese et desimaltall: inDouble()
    - For å lese ett ord: inWord()
    - For å lese resten av linjen (gjør et linjeskift først om tomt): inLine()
  - Lese linje for linje:
    - Kun inneværende linje uansett om den er tom: Bruk readLine()
  - Lese tegn for tegn:
    - For å lese neste tegn (også blanke): inChar()
- Hvordan lesemetodene virker
  - Metodene inInt(), inDouble() og inWord() virker slik:
    - De hopper over eventuelle innledende blanke tegn.
    - De leser så alt fram til neste blanke tegn eller linjeskift. Dersom det som leses ikke er et heltall når inInt() brukes eller et desimaltall når inDouble() brukes, gis det en feilmelding og man får en ny sjanse (3 sjanser).
  - Metoden inChar() virker slik:
    - Den leser neste tegn, enten det er et blankt tegn eller ikke.
  - Metoden inLine() virker slik:
    - Den leser alt fram til slutten av linjen (inkludert blanke tegn), men ignorerer første linje hvor det kun står (igjen) et linjeskift.

✗ Hvordan lesemetodene virker



- Eksempel:
  - Problem: Lag et program som leser fra terminal disse dataene om en person:
    - Navn
    - Yrke
    - Alder
  - ... og som skriver ut dataene på skjermen etterpå.
  - Framgangsmåte:
    - Vi bruker inLine() til å lese navn og yrke, og inInt() til å lese alder.

```
import easyIO.*;
class LesDataOmPerson {
   public static void main (String [] args) {
         String navn, yrke;
         int alder;

         In tast = new In();
         System.out.print("Navn: ");
         navn = tast.inLine();

         System.out.print("Yrke: ");
         yrke = tast.inLine();

         System.out.print("Alder: ");
         alder = tast.inInt();

         System.out.print("Hei " + navn + ", du er " +
                  alder);
         System.out.println(" år gammel og jobber som " +
                  yrke);
   }
}
```

- Eksempel:

```
import easyIO.*;

class LesInnOrd {
   public static void main (String [] args) {
        In tastatur = new In();
        System.out.print("Skriv tre ord: ");
        String s1 = tastatur.inWord();
        String s2 = tastatur.inWord();
        String s3 = tastatur.inWord();

        System.out.println(
                "Du skrev disse ordene:");
        System.out.println("  " + s1);
        System.out.println("  " + s2);
        System.out.println("  " + s3);
   }
}
```

Skriv til skjerm

- ✗ Formatert utskrift til skjerm
  - Formatert utskrift vil si at vi angir nøyaktig hvordan utskriften skal se ut og plasseres på skjermen. I INF1000 bruker vi pakken easyIO –samme som for innlesing.
  - I toppen av programmet (før class) skriv:

```
import easyIO.*;
```

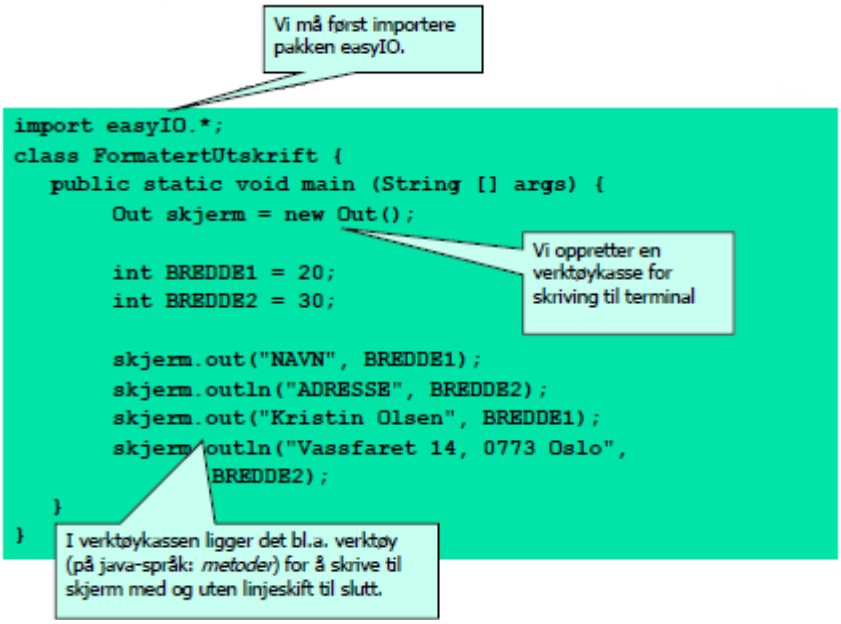
  - Inne i klassen skriver vi så:

```
Out skjerm = new Out();
```

  - Så kan vi skrive ut det vi ønsker, f.eks.:

```
double pi = 3.1415926;
skjerm.out(pi, 2, 6);  // Skriv ut pi med 2 desimaler
                       // høyrejustert på 6 plasser.
```

- Eksempel:



Resultat:

```
> javac FormatertUtskrift.java
> java FormatertUtskrift
NAVN                ADRESSE
Kristin Olsen       Vassfaret 14, 0773 Oslo
>
```

✗ Tre måter å skrive ut på

- Uten formatering:

```
skjerm.out("Per Hansen");
skjerm.out(12345);
skjerm.out(3.1415, 2);    // 2 desimaler
```

- Angi utskriftsbredde:

```
skjerm.out("Per Hansen", 15);      // Bredde 15 tegn
skjerm.out(12345, 15);             // Bredde 15 tegn
skjerm.out(3.1415, 2, 15);         // Bredde 15 tegn,
                                   // 2 desimaler
```

- Angi utskriftsbredde og justering:

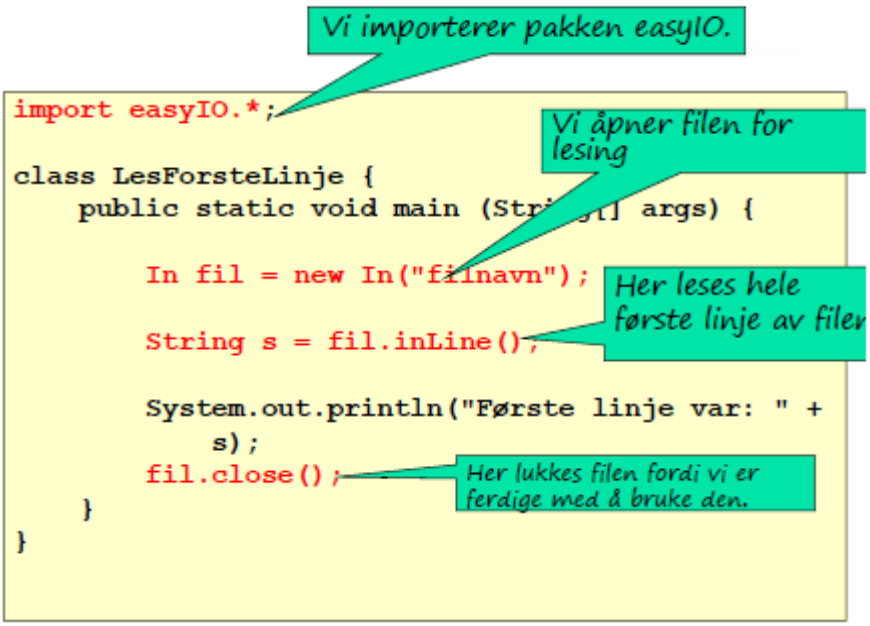
```
skjerm.out("Per Hansen", 15, Out.RIGHT);   // Høyrejuster
skjerm.out(12345, 15, Out.CENTER);         // Senterjuster
skjerm.out(3.1415, 2, 15, Out.LEFT);       // Venstrejuster
```

Lese og skrive fra/til fil

- ✗ Lese og skrive fra/til fil
  - Klassene In og Out i easyIO
  - Les dokumentasjonen
    - In og Out + Format
      - brukes i INF1000
      - Format brukes til mer 'finjustert' formattering
    - InExp og OutExp
      - gir feilmeldinger hvis du gjør noen feil
      - vanskeligere å bruke enn In og Out
      - blir vanskeligere kode, brukes noe i INF1010
    - Det er langt flere metoder enn de som gjennomgås her
  - easyIO ble laget fordi javas innebygde IO-metoder var for kompliserte
    - bedre nå, men fortsatt noe vanskeligere enn easyIO, særlig for det vi skal lære dag: Lesing & skriving på filer

Lese fra fil

- ✗ Eksempel:



- ✗ Tre måter å lese på
  - Ord for ord (tall for tall, ...)
    - Leser med inDouble(), inInt(), inWord(), osv
    - Ser etter slutten med lastItem()
    - Hopper over skilletegn (= hoppeover-tegn mellom ordene man leser)
      - linjeskift-tegnene (+ noen sære tegn) er alltid skilletegn
      - Hvis man ikke gjør noe er også blanke, tab,... skilletegn
      - Brukeren kan også spesifisere skilletegn: i = inInt(”F(,)”);
        (da er bare linjeskift + de som spesifiseres F(,) slike tegn det hoppes over før det leses neste ’ord’)
  - Tegn for tegn (Ikke så mye brukt)
    - Leser med inChar() [inChar(false)], men ikke det samme som inChar(true)
    - Ser etter slutten med endOfFile()
    - Kan kopier en fil tegn for tegn uansett hva den inneholder
  - Linje for linje
    - Leser med readLine()
    - Ser etter slutten med endOfFile()
  - Lesemåtene kan blandes (hvis man vet hva man gjør)
- ✗ Lese ord for ord (item)
  - Metoder:
    - inInt() for å lese et heltall
    - inDouble() for å lese et flyttall
    - inWord() for å lese et ord
    - lastItem() for å sjekke om slutten av filen er nådd
  - Eksempel: lese en fil tall for tall

```
In fil = new In("item.txt");
while (!fil.lastItem()) {
    int k = fil.inInt();
    System.out.println("Tallet var " + k);
}
```

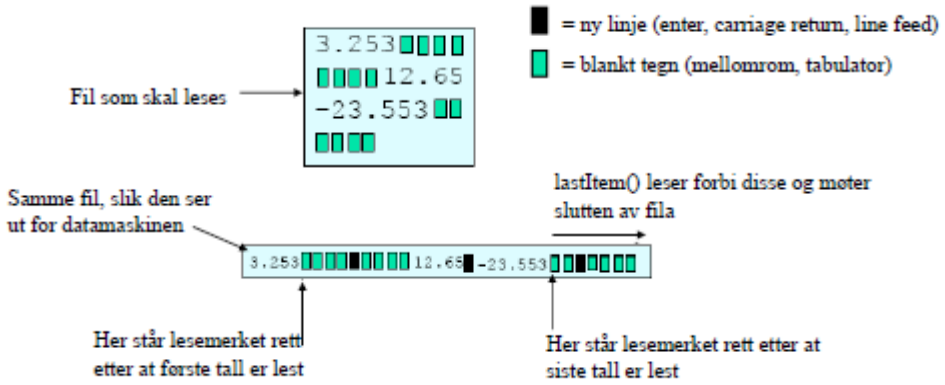
- ✗ Lese linje for linje
  - Metoder:
    - readLine() for å lese en linje
    - inLine() for å lese resten av en linje (leser neste linje hvis det ikke er mer igjen enn linjeskift på nåværende linje)
    - endOfFile() for å sjekke om slutten av filen er nådd
  - Eksempel: lese en fil linjevis

```
In fil = new In("fil.txt");
while (!fil.endOfFile()) {
    String s = fil.readLine();
    System.out.println("Linjen var " + s);
}
```

- ✗ Program som leser en tekstfil linje for linje: Eksempel

```
import easyIO.*;
class LinjeForLinje {
    public static void main (String[] args) {
        In innfil = new In("filnavn");
        String[] s = new String[100];
        int ant = 0;
        while (!innfil.endOfFile()) {
            s[ant] = innfil.readLine();
            ant = ant + 1;
        }
        for (int i=0; i<ant; i++) {
            System.out.println(s[i]);
        }
    }
}
```

- ✗ astItem og endOfFile, lesemerket
  - endOfFile() sjekker 'bare' om siste tegn på fila er lest
  - lastItem() søker seg fram til første ikke-blanke tegn og returnerer true hvis slutten av fila nås og false ellers.
  - Eksempel:



- ✗ Når filens lengde er kjent
  - Når et program skal lese en fil, må det ha en mulighet til å avgjøre når slutten av filen nådd – ellers kan det oppstå en feilsituasjon.
  - Metodene lastItem() og endOfFile() kan benyttes til dette.
  - Noen ganger er filens lengde kjent på forhånd:
    - lengden er kjent før programmet kjøres
    - lengden ligger lagret i begynnelsen av filen
  - Da kan vi i stedet for løkke benytte en for-løkke.
  - Eksempel:
    - Program som leser en fil med 10 desimaltall, hvor tallene er atskilt med blanke tegn og/eller linjeskift:

```
import easyIO.*;
class Les10Tall {
    public static void main (String[] args) {
        double[] x = new double[10];
        In innfil = new In("tall.txt");
        for (int i=0; i<10; i++) {
            x[i] = innfil.inDouble();
        }
        // Nå kan vi evt. gjøre noe med verdiene
        // i arrayen x
    }
}
```

- Nok at tallene er atskilt
  - Programmet over, ville gitt akkurat samme resultat for alle disse filene:

```
15.2
6.23
3.522
3.6
8.893
-3.533
65.23
22.01
45.02
7.2
```

```
15.2  6.23
3.522 3.6
8.893 -3.533
65.23 22.01
45.02 7.2
```

```
15.2   6.23   3.522  3.6
8.893  -3.533
65.23  22.01  45.02


       7.2
```

- Eksempel: fil med lengde-info
  - Program som leser en fil med desimaltall, hvor tallene er atskilt med blanke tegn og/eller linjeskift. Antall tall som skal leses ligger øverst i filen

```
import easyIO.*;
class LesTallMedLengde {
    public static void main (String[] args) {
        double[] x; // bestemmer ikke lengden ennå
        In innfil = new In("tall-med-lengde.txt");
        int lengde = innfil.inInt();// nå vet vi lengden
        x = new double[lengde];
        for (int i=0; i<lengde; i++) {
            x[i] = innfil.inDouble();
        }
        for (int i=0; i<lengde; i++) {
            System.out.println( i + " = " + x[i]);
        }
    }
}
```

- Eksempel: fil med sluttmerke
  - Program som leser en fil med desimaltall, hvor tallene er atskilt med blanke tegn og/eller linjeskift Slutten av filen er markert med tallet -999

```
import easyIO.*;
class LesTallMedMerke {
  public static void main (String [] args) {
    double [] x = new double[100]; // antar max 100 tall
    In innfil = new In("tall-med-merke.txt"); // på fil
    double siste = 0;
    int ant = 0;
    while (siste != -999) {
      siste = innfil.inDouble();
      if (siste != -999) {
        x[ant] = siste;
        ant = ant + 1;
      }
    }    // Nå ligger det verdier i
  }      // x[0], x[1], ...., x[ant-1]
}
```

✗ Lese en fil med mer komplisert format

- Anta at vi skal lese en fil med følgende format:
  - Først er det en linje med 3 overskrifter (separert av blanke tegn)
  - Deretter kommer det en eller flere linjer, som hver består av et heltall, et desimaltall og en tekststreng (separert av blanke tegn)
- Eksempel:

```
Antall   Pris    Varenavn
35       23.50   Oppvaskkost
53       33.00   Kaffe
97       27.50   Pizza
...      ...     ...
...      ...     ...
```

- Fremgangsmåte:
  - Den første linja er spesiell, og vi tenker oss her at den ikke er så interessant - vi ønsker bare å få lest forbi den. Det kan vi gjøre med inLine().
  - De andre linjene har samme format, så vi kan lage en løkke hvor hvert gjennomløp av løkken leser de tre itemene på en linje. Vi bruker da henholdsvis inInt(), inDouble() og inWord().
  - For å vite når filen er slutt, kan vi enten bruke endOfFile() eller
  - lastItem(). Siden vi leser filen itemvis, er det mest naturlig å bruke lastItem(). Da får vi heller ikke problemer dersom det skulle ligge noen blanke helt på slutten av filen – og det gjør det som oftest (ofte et siste linjeskift på siste linje).
  - Vi hopper over detaljene.

Program som leser en fil med tre kolonner: en kolonne med int, en kolonne med desimaltall, og en kolonne med tekst.

```
import easyIO.*;
class LesVarer {
    public static void main (String[] args) {
        In innfil = new In("varer.txt");
        int [] t   = new int[100];
        double[] x = new double[100];
        String[] s = new String[100];
        int ant = 0;

        innfil.readline();
        while (!innfil.lastItem()) {
            t[ant] = innfil.inInt();
            x[ant] = innfil.inDouble();
            s[ant] = innfil.inWord("\n");
            ant = ant + 1;
        }
        for (int i=0; i<ant; i++) {
            System.out.println(t[i] + ":" + x[i] + "-" +
                s[i]);
        }
    }
}
```

- Eksempel på å gi skilletegn ved innlesing
  - Vi kan ved innlesing i easyIO spesifisere hvilke tegn vi vil hoppe over ved innlesing – i inInt, inWord, inDouble,.. kan skilletegn gis
  - Anta at kollonne k og rad r skal gis som: S(r,k) – eks: S(0,4)

```
import easyIO.*;
class Skilletegn {
    public static void main (String[] args) {
        int r,k;
        String skille =" S(,)";
        In tast = new In(); Out skjerm = new Out();

        skjerm.out("Gi rad r og kollonne k som S(r,k):");
        r = tast.inInt(skille);
        k = tast.inInt(skille);

        skjerm.outln("Du ga r=" +r+", og k=" +k);
    }
}
```

✗ Noen nyttige hjelpemidler (ikke pensum)

- Sjekke om det finnes en fil med et bestemt navn:

```
if (new File("filnavn").exists()) {
    System.out.println("Filen finnes");
}
```

- Slette en fil:

```
if (new File("filnavn").delete()) {
    System.out.println("Filen ble slettet");
}
```

- Avgjøre hvilket filområde programmet ble startet fra:

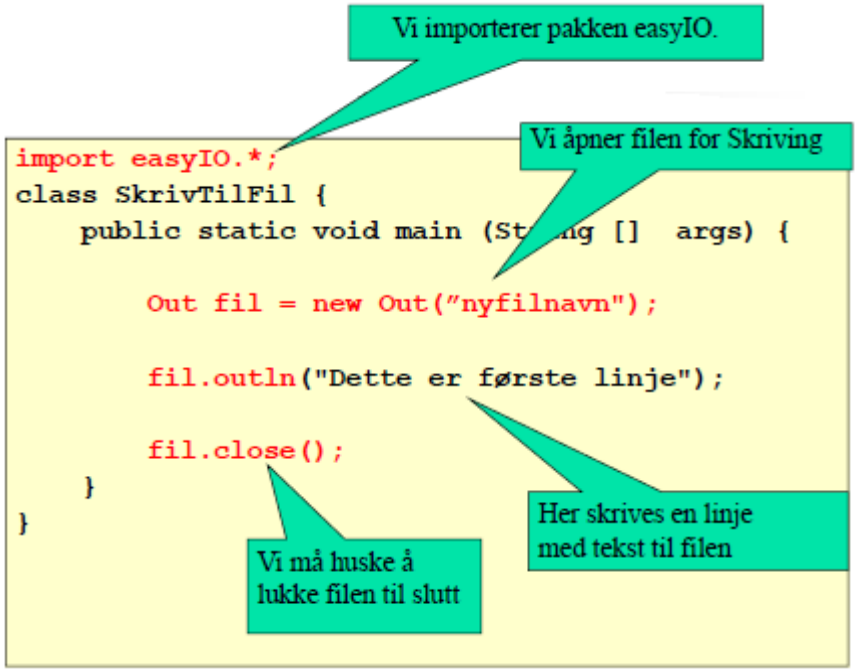
```
String curDir = System.getProperty("user.dir");
```

- Lage liste over alle filer og kataloger på et filområde:

```
String [] allefiler = new File("filområdenavn").list();
```

Skriv til fil

- Skrive til fil:



- EasyIO: Hvilke skrivemetoder finnes?

Datatype	Eksempel	Beskrivelse
int	fil.out(x); fil.out(x, 6);	Skriv x Skriv x høyrejustert på 6 plasser
double	fil.out(x, 2); fil.out(x, 2, 6);	Skriv x med 2 desimaler Skriv x med 2 desimaler på 6 plasser
char	fil.out(c);	Skriv c
String	fil.out(s); fil.out(s, 6);	Skriv s Skriv s på 6 plasser (venstrejustert)
	fil.outln();	Skriv en linjeskift
	fil.close();	Lukk filen

Merk: dersom antall plasser spesifiseres og det ikke er plass til det som skal skrives ut, vil det som skrives ut avsluttes med tre punktumer: ...

Klasser og objekter

- ✗ Objekter
  - Hvorfor deler vi verden inn i ulike ting/gjenstander når vi snakker om den ?
    - En blomst, fjorten trær, ti mennesker, en bil, en vei, mange murstein, en bankkonti,....
  - Svar : For bedre å kunne tenke om, snakke om og forstå verden.
  - Verden består av mange objekter, noen ganske like, noen ulike
    - Ser vi rundt oss i et auditorium, ser vi
      - Studenter (mange), stoler, murstein, lysarmatur, blyanter,...
    - Vi ser at når vi betrakter verden, deler vi den opp i et passe antall enheter/deler/'klumper' som vi har egne navn for.
    - Slike enheter/'klumper' vi deler verden inn i, kaller vi objekter
    - Mange av objektene i verden er egentlig like, av samme type, men kan skille seg fra hverandre med feks. ulike navn
      - Studentene Kia og Espen
      - to bøker med ulik tittel/forfatter
    - Objekter som er egentlig er like sier vi tilhører samme klasse
      - De beskrives av samme sett med variable, men har ulike verdier på noen disse (to biler av samme bilmerke men med ulike reg.nummer)
- ✗ Klasser og objekter i verden
  - To objekter kan også være helt vesensforskjellige (f.eks et tre og en lastebil)
    - De er da to objekter av hver sin klasse (klassen Tre og klassen Lastbil)
  - Hva vi velger å betrakte som et objekt, og hvilke klasser vi bruker for å beskrive en problemstilling, er ikke bestemt på forhånd. Innenfor vide rammer bestemmer vi det selv.

    Omdebattert av filosofene, men som OO-programmerer, hevder jeg :
    - Klasser er generelle begreper som egentlig ikke eksisterer i verden – det som eksisterer er objektene.
    - Generelle begreper (klasser) er menneskenes måte å beskrive og strukturere verden, ikke gitt en gang for alle og vi kan godt lage oss nye begreper.
- ✗ Hvor mange klasser (og objekter) er det ?
  - Hvilke klasser vi bruker til å beskrive et problem, varierer ofte etter hvor detaljert vi betrakter en problemstilling og hvilke spørsmål vi ønsker å kunne gi svar på:
    - Beskriver vi problemet med veitrafikk og køer på veiene, er vi neppe interessert i mer enn å telle antall biler og kanskje skille mellom lastebiler, busser og personbiler.
    - Beskriver vi problemet til en bilfabrikk, trenger vi en meget detaljert og komplisert beskrivelse av hver bil (bestående av motor, hjul, karosseri, lys,..., hver beskrevet med sin klasse) og mange ulike typer av biler. Vi se da en rekke klasser som hver beskriver sin sine objekter.

- ✗ Objektorientert Programmering – I
  - Når vi betrakter et problem vi skal lage et datasystem for, gjør vi to avgrensninger:
    1. Vi ser bare på en del av verden (vårt problemområde)
    2. Innenfor problemområdet betrakter og beskriver vi bare det som er der med en viss detaljeringsgrad - bare så mange detaljer vi trenger for å svare på de spørsmål datasystemet skal kunne gi svar på.

  - Eks: Hvordan beskrive en student ? Skal vi lage:
    a) Et Studentregister, registrerer vi bare navn, personnummer, adresse, tidligere utdannelse og kurs (avlagte og kurs vedkommende tar nå)
    b) Et legesystem for studenter, ville vi ta med svært mange opplysninger om hver student (medisiner, sykdommer, resultat fra blodprøver, vekt...) som vi ikke ville drømme om å ha i et vanlig studentregister
- ✗ Objektorientert Programmering – II
  - Når vi skal lage et programsystem, så skal det i størst mulig grad være en modell av vårt problemområde – en en-til-en kopi:
    - Ett objekt i problemområdet skal medføre at det skal være ett objekt i programmet som representerer dette 'verdens-objektet'.

      (i tillegg kommer mange klasser og objekter i programmet for å lese fra tastatur og fil, skrive til skjerm,..)
  - Eks.: Hver virkelig student skal ha sitt Student-objekt i et studentregister-system.
- ✗ Forholdet mellom klasser og objekter i Java
  - Klasser er objektfabrikker. Vi sier new på en klasse og får et objekt tilbake.
  - Det objektet er en kopi av klassen (deklarasjonene inne i klassen + metodene i klassen).
  - Har klassen en start-metode (som heter det samme som klassen), har vi også greid å gi spesielle startverdier til noen av variablene i objektet.
  - En Klasse er altså som en slags støpeform/ fabrikk som lar oss lage en eller flere objekter av den klassen.
  - Det er kall på metoder i objektene som gjør at programmet vårt gjør noe.
    - Et lite unntak her for statiske metoder

✗ Hvordan lage klasser og objekter i et program.

- Klasser deklarerer vi med `class`
- Vi lager pekere til objekter ved å deklarere dem med klassenavnet
- Vi lager et objekt med å si `new` foran et klassenavn
- Forholdet mellom et objekt og en peker er som en array-peker og et array-objekt

```
class Student {
   String navn, adresse;
}

class StudentRegister {
  public static void
     main(String args []) {

    Student s1, s2;

    s1 = new Student();
    s2 = new Student();

   }
}
```

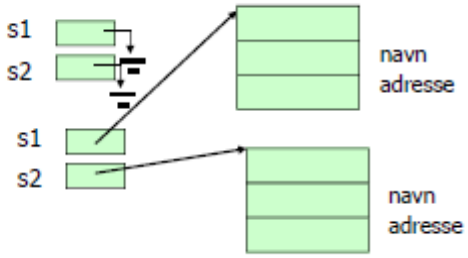
✗ Hva er et objekt i programmet?

- Et objekt er et område i lageret som inneholder en kopi av alle de metodene og variable i en klasse det ikke står static foran
- Klassene er en slags mal/form/oppskrift/fabrikk som vi kan lage objekter med.
- Lager vi to, tre,.. objekter av klassen, får vi to, tre,.. slike kopier
- De variable og metode det ikke står static foran, kalles objekt-variable og objekt-metoder
- De variable og metoder det står static foran, kalles klassemetoder og klasse-variable, og blir ikke med i objektene (men ligger lagret i bare ett eksemplar et annet sted)

✗ Peker og objekter

```
class Student {
   String navn, adresse;
}

class StudentRegister {
  public static void
     main(String args []) {

    Student s1,
            s2;

    s1 = new Student();
    s2 = new Student();

   }
}
```

- En peker inneholder adressen til hvor et objekt ligger i lageret
- eller den inneholder 'null' (= ikke-objekt)
- Vi tegner adressen som en 'pil'



- ✗ Programmer i Java består av en eller flere klasser
  - Vi deler opp programmet vårt i flere klasser
    - Fordi hver programdel (klasse) skal være mulig å holde oversikt over – ikke for stor
    - Fordi en klasse skal være god modell av en del av problemet vårt vi lager program for.
      - Anta at vi hadde et datasystem som omhandlet kurs og studenter. Da ville vi ha en klasse Student og en klasse Kurs i programmet.
  - En klasse inneholder
    - Deklarasjon av null eller flere variable
      - som beskriver ett eksemplar av det klassen er modell av
    - Null eller flere metoder
  - En klasse representer et generelt begrep som:
    Student, Kurs, Person, Dokument, Eiendom, DVDRegister, DVD, Billett,
    Bil, Fly, Tre, Ku, Motorsykkel...
- ✗ Objekter og pekere, og hvordan få adgang til innmaten av et objekt (.)
  - Når vi har laget et objekt med new, har vi altså fått en kopi av objekt-variablene og – metodene, men hvordan får tak i dem ?
  - Vi bruker operatoren . (punktum).
    - Foran punktumet har vi navnet på en peker til et objekt.
    - Etter punktum har vi navnet på en variabel eller metode inne i objektet –
    - Punktumet leses som 'sin' eller 'sitt'
      - eks: La s1 peke på et Student-objekt.

```
s1.navn = "Ola N";
```

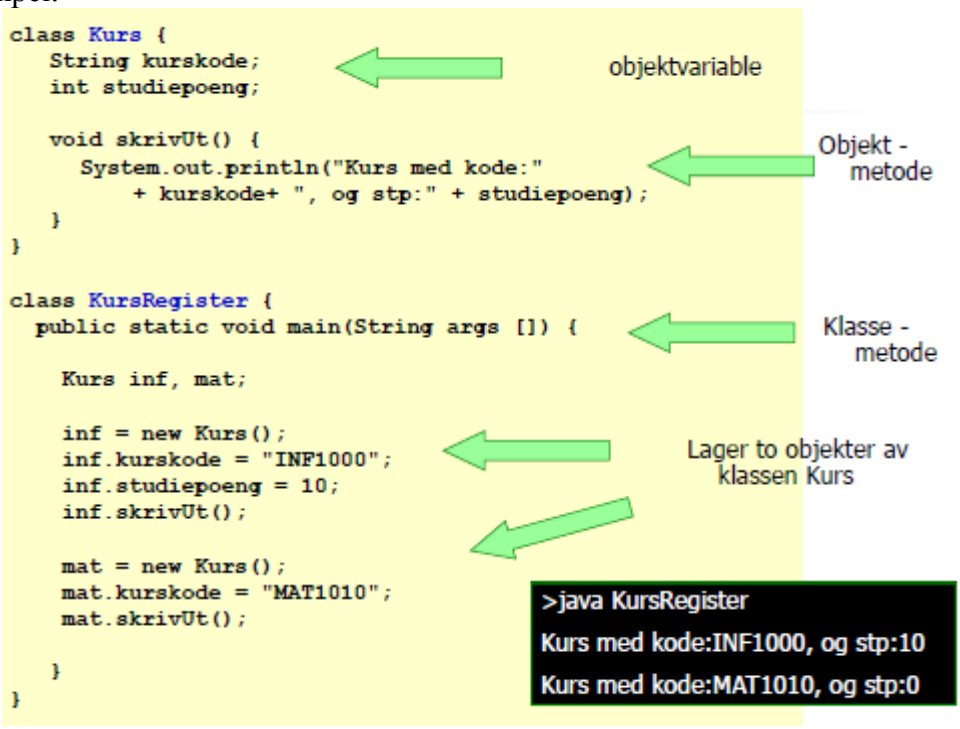
- ✗ Oppsummering om klasser, objekter, pekere og .
  - Verden består av objekter av ulike typer (klasser).
    Ofte er det mange objekter av en bestemt type.
  - Objekter som er av samme klasse, beskrives med de samme variablene, men vil ha forskjellige verdier på noen av disse.
    - Eks: To bankkonti med ulik eier og kontonummer, men kan f.eks ha samme beløp på saldo (tilfeldigvis)
  - Vi lager OO-programmer ved å lage en modell av problemområdet i Javaprogrammet
    - ett objekt i verden gir ett tilsvarende Java-objekt i programmet
    - Objekter kan være av ulik type, og for hver slik type deklarerer vi en klasse i programmet
  - Et Javaprogram består av en eller flere klasser
  - En klasse er en deklarasjon av data og metoder for ett objekt av klassen.
  - Vi deklarerer pekere til objekter av en bestemt klasse – f.eks.
    class Kurs {..} slik:
    ```
    Kurs kurs14, k2, k;
    ```
  - Vi lager objekter fra klassen med new
    ```
    k2 = new Kurs();
    ```

- Et objekt inneholder en kopi av alle ikke-statiske variable og ikke-statiske metoder i klasse
  - Disse kalles objekt-variable og objekt-metoder
- Vi får adgang (lese, skrive og kalle metoder) til det som er inni et objekt ved . operatoren :
  - Vi må ha en peker til et objekt etterfulgt av punktum .

    s2.adresse ="bokhandelen i Kabul";

    s1.skrivUt();

✗ Eksempel:



✗ Stringer er ordentlige objekter
- String er en klasse i Java-biblioteket, men har en egen spesiell syntaks (skrivemåte) så det ser ut som den er en av de basale typene (som int, double,..).
- Når vi har en string, har vi alså både en peker (den vi deklarerer navnet på) og et stringobjekt.
- String-objekter kan ikke endres (trenger du endrbare tekster, bruk klassen StringBuffer)
- Egen skrivemåte for stringkonstanter:
  - String s = ”En fin dag i mai”;

  Er det samme som:
  - String s = new String(”En fin dag i mai”);
- Klassen String inneholder mer enn 50 metoder for konvertering mellom ulike datatyper og tekst, samt tekstsøking.

- ✗ null, && og søppeltømmeren
  - Av og til har vi behov for å teste om en peker virkelig peker på et objekt eller ikke:

```
Student s = hyblene[i][j].leietager ;
if (s != null && s.navn.equals(”Ola”)) {
    // her kommer vi bare hvis s peker på et studentobjekt
    // og navnet i det studentobjektet er lik “Ola”
    .....
}
```

  - Hvis vi vil fjerne et objekt fra en peker og fra hele systemet:

```
hyblene[i][j].leietager = null;
```

  - Et objekt som ingen pekere peker på, blir tatt av søppeltømmeren – et program som automatisk startes hvis det er lite plass i hukommelsen:
- ✗ Hva er en klasse?
  - En klasse er et mønster for/ en beskrivelse av hvordan ett objekt av en bestemt type i vårt problem er.
    - Inneholder variable som beskriver egenskaper for ett slikt objekt – eks:
      - Navn, adresse, studiepoeng, kurs... for klassen Student
      - Registreringsnummer, eier, type, årsmodel for klassen Bil
    - Inneholder metoder som er fornuftig handlinger for ett slikt objekt – eks:
      - skrivUtVitnemål(), meldPåEmne(),.. i klassen Student
      - beregnÅrsavgift(), skiftEier(),.. i klassen Bil
    - En egenskap i en klasse som viser til en annen klasse representeres som en peker-variabel til et objekt av den andre klassen. (Til ”mange” objekter: Peker-array eller HashMap)
- ✗ Skille mellom deklarasjon og bruk av en klasse
  - Klassedeklarasjoner i programmet medfører ’lite’ aktivitet:
    - De variable og metodene det står static foran er tilgjengelige (klassevariable og klassemetoder)
    - ..dvs static metoder kan kalles (eks. main ved start av programmet)
    - Mønsteret for nye objekter av klassen er tilgjengelig
    - Hvis det aldri blir sagt new på klassen, finnes det ingen objekter, og dermed ingen objektvariable eller objektmetoder
  - Når utførelsen kommer til en setning med new på en klasse, får vi laget et objekt av klassen
    - Objektet inneholder alle objekt-variable og – metoder (variable og metoder som ikke har static foran deklarasjonen i klassen)
    - Det kalles automatisk en konstruktør-metode i klassen, og først når den er ferdig, returnerer new med det nye objektet. Konstruktøren sørger for at objektvariablene i det nye objektet har de initialverdiene vi ønsker. Disse verdiene kan for eksempel hentes fra parametre til konstruktøren eller leses fra fil eller terminal.

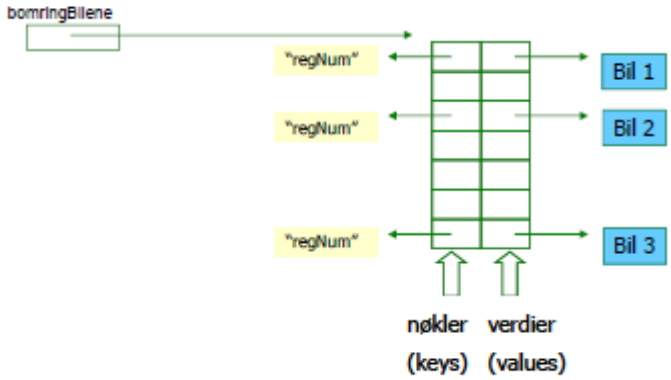
- ✗ Klassemetoder(-variable) og objektmetoder(-variable)
  - Klassemetoder (static-metoder)
    - Definert selv om det ikke er laget noen objekter av klassen
    - Kan ”ses” av alle objekter av klassen
    - Kan brukes av andre gjennom dot-notasjon: <klassenavn>.metode(...)
    - Har ikke tilgang til objektvariable eller objektmetoder
  - Objektmetoder
    - Bare definert i objekter av klassen (opprettet med ”new”)
    - Kan ”ses” av objektet som metoden befinner seg i
    - Kan brukes av andre gjennom dot-notasjon: <peker>.metode(...)
    - Har tilgang til alle variable (både klassevariable og objektvariable) og alle metoder (både klassemetoder og objektmetoder)
- ✗ Forskjeller mellom klasser og metoder
  - Begge lager objekter når de kalles, men:
  - Et metode-objekt:
    - fjernes når metoden returnerer
    - inneholder ’bare’ variable og parametere som alle er skjult for resten av programmet
  - Et objekt laget med new fra en klasse:
    - er i hukommelsen etter at det er laget (så lenge det minst er en peker som peker på det)
    - kan inneholde både metoder og variable, som kan nyttes av resten av programmet (med en peker og .)

HashMap

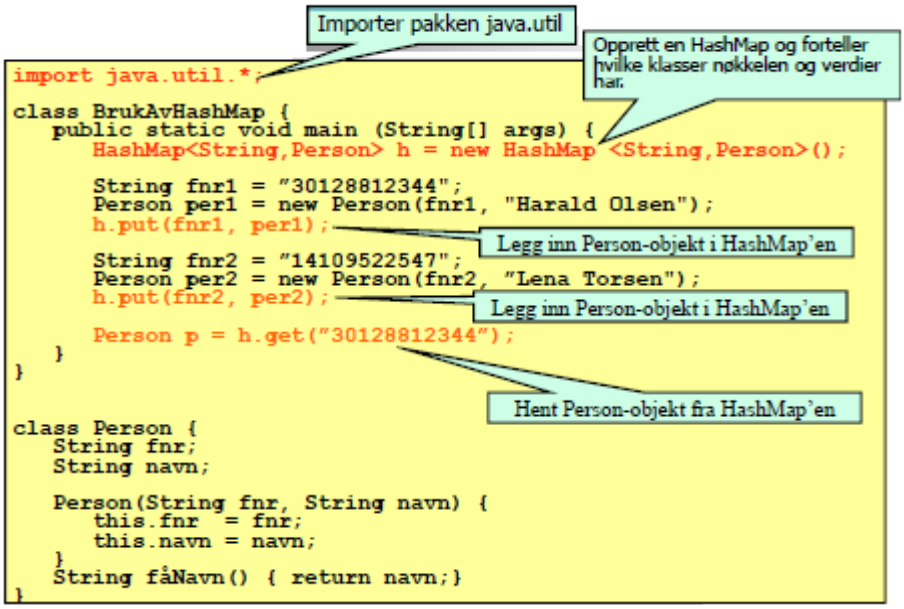
- ✗ Holde orden på objekter – HashMap
  - Ofte har vi flere, mange objekter av en bestemt klasse - eks. :
    - elever på en skole
    - biler som har passert bomringen i Oslo
    - telefonsamtaler fra en bestemt person,.
  - Vi har hittil lært arrayer (Elev [] elevene = new Elev[400], Bil [] bomringBiler = new Bil[10000];.....) og må da passe på at vi har
    - nok plass
    - for å finne et bestemt objekt må vi ofte lete gjennom 'hele' arrayen
  - Vi skal nå lære en bedre måte å lagre objekter hvor det er viktig å raskt og enkelt finne igjen ett av objektene (som da har ett bestemt kjennetegn som: Navnet til eleven, registreringsnummeret til en bil,...)
  - Et slikt kjennetegn som skiller ett objekt fra alle andre objekter, kaller vi en nøkkel (key)
  - HashMap er svaret
- ✗ HashMap = lagre objekter med en søkenøkkel
  - Brukes til å holde orden på en samling objekter
  - Alternativ til arrayer
  - Med arrayer kan man:
    - I en array legger vi inn objekter i en bestemt posisjon, og vi må gå tilbake til denne posisjonen/indeksen når vi senere skal se på objektet. Indeksen er et heltall mellom 0 og length-1.
  - To viktige forskjeller mellom arrayer og HashMap:
    - I en HashMap oppgir vi en bestemt nøkkel (vanligvis en tekststreng) når vi legger vi inn et nytt objekt (kalt verdien), og vi oppgir denne nøkkelen når vi senere skal se på objektet. Dvs. indeksen er en tekststreng.
    - En HashMap har ingen gitt lengde når vi lager den. Den 'vokser' når legger inn nye elementer (inntil hele lagerplassen er oppbrukt)
  - En Hashmap er mer fleksibel (men langsommere) måte å lagre flere/mange elementer i et program

- ✗ Hvordan vi skal tenke oss en HashMap
  - en HashMap er som en slags dobbelt-array (f.eks bomringBilene)

```
HashMap bomringBilene = new HashMap();
```



- ✗ Ulike versjoner i Java 1.4 (gammel) og Java 1.5/1.6/1.7 av HashMap
  - Vi anbefaler klart at 1.5-måten nyttes, da den hjelper deg mot visse feil (som ellers er lett å gjøre).
  - 1.4 måten gjennomgås bare en gang (bare) fordi mange gamle programmer inneholder slik kode.
- ✗ Eksempel på bruk av HashMap (1.5-1.7)




```
import java.util.*;

class BrukAvHashMap {
   public static void main (String[] args) {
      HashMap<String,Person> h = new HashMap <String,Person>();

      String fnr1 = "30128812344";
      Person per1 = new Person(fnr1, "Harald Olsen");
      h.put(fnr1, per1);

      String fnr2 = "14109522547";
      Person per2 = new Person(fnr2, "Lena Torsen");
      h.put(fnr2, per2);

      Person p = h.get("30128812344");
   }
}

class Person {
   String fnr;
   String navn;

   Person(String fnr, String navn) {
      this.fnr  = fnr;
      this.navn = navn;
   }
   String fåNavn() { return navn;}
}
```

✗ Opprette en HashMap, Java1.4 (gammel) og Java 1.5-1.6

- I starten av programmet:
  import java.util.*;
  Dette importerer pakken java.util hvor bl.a. klassen HashMap ligger.
- I klassen eller metoden som skal bruke HashMap'en Java 1.4:
  HashMap h = new HashMap();
- I klassen eller metoden som skal bruke HashMap'en Java 1.5 og nyere (best):
  HashMap <String,Person> h = new HashMap <String,Person>();
- I Java 1.5 forteller vi hvilke klasser nøkkel- og verdi-objektene kommer fra. Vi sier at vi da låser objektene til både nøkkelen og verdi-objektene til å være av disse typene.
- NB: Hvis tabellen skal brukes av flere metoder i en klasse, deklareres variabelen ovenfor i starten av klassen (som en objektvariabel).
  Hvis tabellen kun skal brukes av en enkelt metode, er det naturlig å deklarere HashMap variabelen ovenfor inni den aktuelle metoden.

✗ Legge inn objekt i HashMap (samme i 1 4 og 1 5-1 7)

- Et hvilket som helst objekt i Java kan legges inn i en HashMap men i 1.5 må det være av den klassen vi har 'lovet' systemet
- Når vi legger et objekt inn i HashMap'en, må vi samtidig oppgi en nøkkel, dvs en tekststreng som entydig identifiserer objektet.
- Vi trenger denne nøkkelen dersom vi senere skal finne eller fjerne objektet i HashMap'en.
- Eksempel:

```
String fnr = "30126512345";
Person p = new Person(fnr, "Kari Olsen");
h.put(fnr, p);
```

  Her lager vi først et Person-objekt (med passende argumenter) og legger det deretter inn i tabellen med fødselsnummeret som nøkkel.
- Dersom vi legger inn flere objekter med samme nøkkel, er det bare det sist innlagte objektet som blir liggende i tabellen (de andre overskrives):

```
Person p1 = new Person(...);
Person p2 = new Person(...);
Person p3 = new Person(...);
String navn = "Jens";
h.put(navn, p1); // p1 legges inn
h.put(navn, p2); // p2 legges inn og p1 overskrives
h.put(navn, p3); // p3 legges inn og p2 overskrives
```

- Noen ganger må vi konstruere en nøkkel ut fra flere variable for å få entydighet:

```
String lengdegrad = "67.3";
String breddegrad = "53.3";
String posisjon = lengdegrad + ";" + breddegrad;
Fjelltopp fjell = new Fjelltopp(posisjon, "Bjørnefjell");
h.put(posisjon, fjell);
```

✗ Hente objekt fra HashMap – Java 1.4 og 1.5

Java 1.4: For å hente et objekt med utgangspunkt i nøkkelen:

```
// 1.4: Vi vil finne en person ut fra fødselsnummeret:
Person p = (Person) h.get(fnr);
```

- Legg merke til at vi i 1.4 i starten må skrive i parentes navnet på klassen som objektet tilhører - i dette tilfellet klassen Person.
- Årsaken er at i 1.4 HashMap'en ikke holder rede på hvilken klasse objektene som legges inn har - bare at det er objekter. Når objektene hentes ut må vi derfor "minne Java på" hvilken klasse objektet var av (dette er egentlig et møte med en avansert og svært nyttig mekanisme i objektorienterte språk som kalles arv og som blir tatt opp i INF1010).

Java 1.5: For å hente et objekt med utgangspunkt i nøkkelen , nå trenger vi ikke si hvilken klasse objektet har (det har vi jo sagt i deklarasjonen av HashMapen h):

```
// 1.5: Vi vil finne en person ut fra fødselsnummeret:
Person p = h.get(fnr)
```

- Merk: hente et objekt fra en HashMap slik som over medfører ikke at objektet fjernes fra HashMap'en (vi får bare en kopi av en peker til objektet).

✗ Fjerne objekt fra HashMap

- For å fjerne et objekt med gitt fødselsnummer som nøkkel:

```
h.remove(fnr);
```

- Dersom det ligger et objekt i HashMap'en med den gitte nøkkelen, blir objektet fjernet og setningen ovenfor returnerer med en peker til objektet som fjernes.
- Dersom det ikke ligger et objekt i HashMap'en med den gitte nøkkelen, returnerer setningen ovenfor verdien null.

✗ Løp gjennom alle objekter i HashMap Java 1.4

- For å løpe gjennom alle objektene i en HashMap, lager vi en oppramsing:

```
Iterator it = h.values().iterator();
```

- Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
    Person p = (Person) it.next();
    System.out.println("Navn: " + p.fåNavn());
}
```

- Løp gjennom alle objekter i HashMap Java 1.5
  - For å løpe gjennom alle objektene i en HashMap, lager vi en oppramsing og låser samtidig det vi skal hente til en bestemt klasse:

```
Iterator <Person> it = h.values().iterator();
```

  - Deretter kan vi se på hvert enkelt objekt i HashMap'en ved å gå i løkke:

```
while (it.hasNext()) {
     Person p = it.next();
     System.out.println("Navn: " + p.fåNavn());
}
```

  - Bedre: Vi kan også i 1.5 nytte den nye for-løkka som automatisk lager en iterator

```
for (Person p: h.values()) {
     System.out.println("Navn: " + p.fåNavn());
}
```

- To måter å løpe gjennom en HashMap – 1.5
  - Løpe gjennom objektene (som på forrige foil):

```
Iterator <Person>it = h.values().iterator();

while (it.hasNext()) {
  Person p = it.next();
  <gjør noe med objektet p>
}
```

  - Løpe gjennom nøklene:

```
Iterator <String> it = h.keySet().iterator();

while (it.hasNext()) {
  String nøkkel =  it.next();
  <gjør noe med nøkkelen>
}
```

- Metoder i HashMap

Metode	Eksempel	Beskrivelse
put	`h.put(nøkkel, objekt);`	Legg inn objekt med gitt nøkkel
get -1.4 get -1.5	`Person p = (Person) h.get(nøkkel);` `Person p = h.get(nøkkel);`	Finn objekt
remove	`h.remove(nøkkel);`	Fjern objekt

containsKey	`if (h.containsKey(nøkkel)) {`
	`  // gjør et eller annet`
	`}`
		Sjekk om nøkkel finnes i tabell

values	`Iterator it = h.values().iterator();`	Lag oppramsing av objektene
keySet	`Iterator it = h.keySet().iterator();`	Lag oppramsing av nøklene

✗ Iterator (oppramsing)

	Eksempel
	`Iterator it1 = h.values().iterator();`
	`Iterator it2 = h.keySet().iterator();`
		Beskrivelse
		deklararasjon

hasNext()	`while (it.hasNext()) {`
	`  < les neste og gjør noe>;`
	`}`
		returnerer true hvis flere objekter igjen i oppramsingen

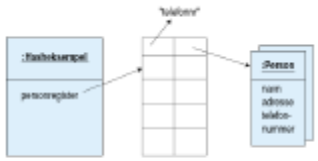
next() 1.5 next() 1.4	`Person p = it.next();` `Person p = (Person) it.next();`	Finn neste objekt

remove()	`Person p = it.next();`
	`if (p.navn.equals("Arne"))`
	`  it.remove();`
		Fjern siste objekt som ble returnert med next()

	`while (it1.hasNext()) {`
	`Person p1 = it1.next();`
	`}`
	`for (Person p2 : h.values()){`
	`  ....`
	`}`
		To måter å gå gjennom alle verdiene (objektene) i h

✗ Eksempel:

```
import java.util.*;
import easyIO.*;

class Hasheksempel {
   public static void main(String[] argv) {
      In tastatur = new In();
      HashMap <String,Person> personregister =  new HashMap <String,Person>();
      System.out.print("Antall personer som registreres : ");
      int ant = tastatur.inInt();

      for (int i = 0; i < ant; i++) {
         System.out.println("Gi neste person");
         Person p = new Person(tastatur);
         personregister.put(p.telefonnr, p);
      }
      // Skriv ut alle personobjektene
      System.out.println("Viser alle personer" +
      "(ukjent rekkefølge):");

      for (Person p: personregister.values()){
         p.skrivData();
      }
   }
}
```



```
class Person {
   String navn, adresse, telefonnr;

   Person (In tastatur) {
      System.out.print("Oppgi navn : ");
      navn = tastatur.inLine();
      System.out.print("Oppgi adresse : ");
      adresse = tastatur.inLine();
      System.out.print("Oppgi telefonnummer : ");
      telefonnr = tastatur.inLine();
   }
   void skrivData() {
      System.out.println("Navn : " + navn);
      System.out.println("Adresse : " + adresse);
      System.out.println("Telefonnummer : "
        +telefonnr);
   }
   String faNavn() { return navn;}
}
```

Eksempel fra boka s.189

FEIL

- ✗ Program-eksempler med feil (1)

```
class Eksempel1 {
  public static void main (String [] args) {
    boolean b;
    b = 2;            // Her prøver vi å sette b lik et heltall
  }
}
```

Feil under kompilering:

```
> javac Eksempel1.java
Eksempel1.java:4: incompatible types
found : int
required: boolean
          b = 2;     // Her prøver vi å sette b lik et heltall
              ^
1 error
>
```

- ✗ Program-eksempler med feil (2)

```
class Eksempel2 {
   public static void main (String [] args) {
          int x = 3;
          int y = 0;
          int z = x / y;       // Heltallsdivisjon med null
   }
}
```

Feil under kjøring:

```
>java Eksempel2
Exception in thread "main" java.lang.ArithmeticException: / by zero
          at Eksempel2.main(Eksempel2.java:5)
```

- ✗ Program-eksempler med feil (3)

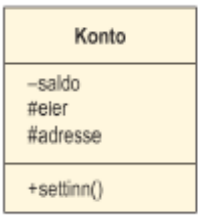
```
class Eksempel3 {
   public static void main (String [] args) {
          int x = 2;
          int y = 4;

          x = y; //prøver å bytte verdier mellom x og y
          y = x; //når vi leser fra y er gammel verdi overskrevet

          System.out.println ("x=" + x + " og y=" + y);
   }
}
```

Logisk feil:

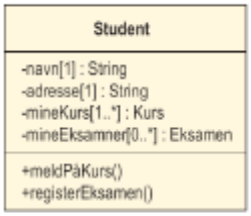
```
>java Eksempel3
x=4 og y=4
>
```

UML-diagrammer

- ✗ UML-diagrammer av programmene våre
  - Hvorfor tegne diagrammer over programmene?
    - Oversikt
    - Samarbeid med andre programmererer / systemutviklere
    - Arkitekter, ingeniører tegner først, så bygger de !
    - Enklere å endre en tegning enn programmet
  - Objektdiagrammer
  - Klassediagrammer
  - UML – diagrammene er litt annerledes enn det vi har tegnet hittil (men mye av det samme)
    (i UML er det mer enn 10 andre diagramtyper vi ikke skal lære)
- ✗ Klassediagrammer
  - En mer kompakt måte enn objektdiagrammer å tegne sammenhengen i programmet
  - Skiller seg fra objektdiagrammer ved at vi ikke direkte tegner datastrukturen (pekere og pekerarrayer), men bare forhold (assosiasjoner, forbindelser) mellom klassene.
  - I klassediagrammer dokumenterer vi også sentrale metoder.
  - Forholdene er linjer med et logisk navn og antall objekter i hver ende
  - Anta at vi har laget en class Konto med tre objektvariable: saldo, eier og adresse og en metode: settinn().



- ✗ Tre (fire) mulig felter i tegning av en klasse



- Navnefeltet (alltid)
  - klassenavnet

Kan utelates:

- Variabelfeltet (attributtene)
  - variabelnavn evt. med type
- Metode-feltet
  - Evt med parametere og returverdi
- (Unntaks-feltet)

Symboler for synlighet
(fra resten av programmet)

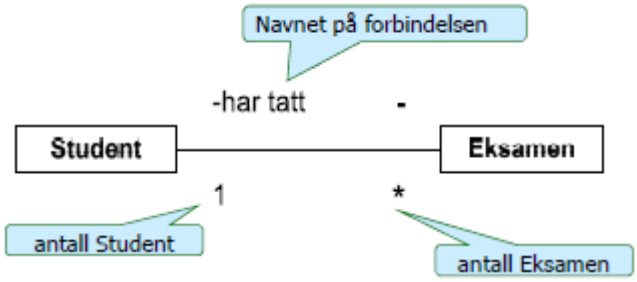
\+ public

\- private

\# protected

~ package

- ✗ UML Klassediagrammet kan nyttes til
  - Modell av problemområdet (domenemodell)
  - Modell av klassene i programmet
    (+ modell av databasen,....)
  - Men siden vi skal modellere virkeligheten en-til-en i programmet vårt, så blir de like i utgangspunktet
- ✗ Forhold mellom klasser
  - En student har null eller flere eksamener
  - Vi tegner et forhold mellom to klasser som har med hverandre å gjøre logisk sett, og:
  - hvor vi i programmet vil kunne følge pekere for å få adgang til variable eller metoder
  - Vi skriver hvor mange objekter det maksimalt på ett tidspunkt kan være på hver side av et slikt forhold
  - Siden vi med: Eksamen mener en avlagt enkelt-eksamen, vil en Eksamen bare være tilknyttet en bestemt student



- Forbindelsen leses fra venstre:
  En student har tatt null, en eller flere Eksamener"
- Antallet objekter angis slik:

Skrivemåte	Betydning
1	en
*	null, en eller flere
1..*	minst en
3..*	minst tre
3,4,5	tre, fire eller fem

✗ Eksempel:
- Studentregister3 – med tillegg: klassen Kurs vet hvilke Studenter som tar kurset
- Et studentregister holder orden på studentene og kursene, og en student tar 3 kurs hvert semester

✗ Regler for å plassere riktige antall på et forhold

1. Anta at du står i ett objekt av en klasse og ser over til (langs en forbindelse) til en annen klasse:
2. Hvor mange objekter ser du da maksimalt på et gitt tidspunkt av den andre klassen
3. Det antallet noteres (jfr. tabellen) på den andre siden
4. Du går så over forbindelsen til den andre klassen og antar at du nå står i ett objekt av denne klassen og gjenntar pkt. 1-3

✗ Hvilke forhold skal vi ha med i klassediagrammet ?
- Slike forhold hvor ett objekt av den ene klassen:
  - inneholder
  - består av
  - eier,..

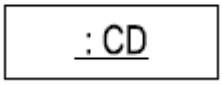
  en eller flere objekter av den andre klassen
- Det vi i programmet vil følge en peker for å få tak i verdien på visse variable i den andre klassen eller kalle en metode.

  Det er da ikke 'naturgitt' hvilke forhold vi har i et klassediagram, det avhenger av hvilke spørsmål vi vil være interessert i å svare på.

✗ Objekt-diagrammer
- Vi tegner en typisk situasjon av objekter i systemet vårt, når vi har fått datastrukturen på plass.
- Vi tegner og navngir bare de mest sentrale dataene som:
  - pekere
  - peker-arrayer
  - noen sentrale variable i objektene

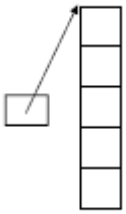
- ✗ Tegning av et objekt (med mer eller mindre detaljer)
  - To eller ett felt(er) i en boks
  - Navnfeltet
    - objektnavn:klassenavn

    eller bare
    - :klassenavn
  - Attributt-feltet (kan være tomt)
    - Navnet på sentrale objektvariable evt. også med verdier

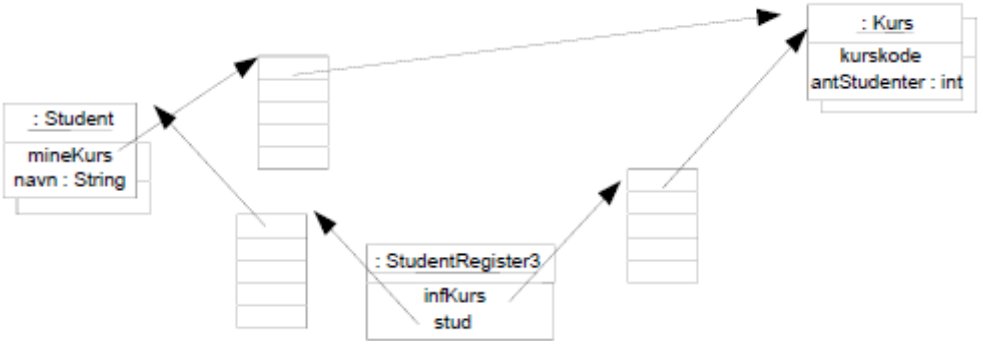


- ✗ Andre elementer i et objektdiagram
  - Pekere

  - Peker-arrayer



- ✗ Eksempel: Et helt studentregister med kurs, studenter og registeret
  - Vi har Studenter på Ifi som første semester tar tre kurs, samtidig som vi har behov for å registrere kurs og hvor mange studenter som tar hvert kurs.
  - Vi tegner først en tenkt datastruktur – et UML objektdiagram
  - så skriver vi programmet
  - Objektdiagrammet
    - en forenkling av programmet. Det tar bare med den essensielle datastrukturen (mest pekere og peker-arrayer) som holder datastrukturen sammen



- Program:

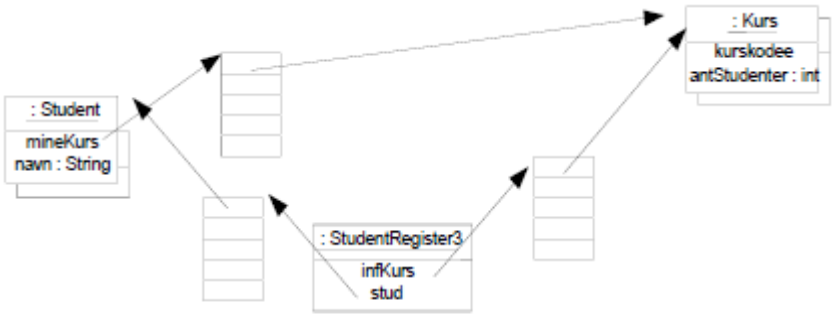
```
class StudentRegister3{
    public static void main(String args []) {
          StudentRegister3 sr = new StudentRegister3();
    }

    StudentRegister3() {
         String [] kurskode = {"INF1000","INF1040","MAT1030"};

         // lag kurs
         Kurs [] infKurs = new Kurs[3];
         for (int i = 0 ; i< infKurs.length; i++)
               infKurs[i] = new Kurs(kurskode[i]);

         //lag studenter på informatikk bachelor
          Student [] stud = new Student[3];
          stud[0] = new Student("Ola N", infKurs);
          stud[1] = new Student("Åsne S",infKurs);
          stud[3] = new Student();

         for (int i = 0 ; i< stud.length; i++)
               stud[i].skrivUt();

     }
}
```

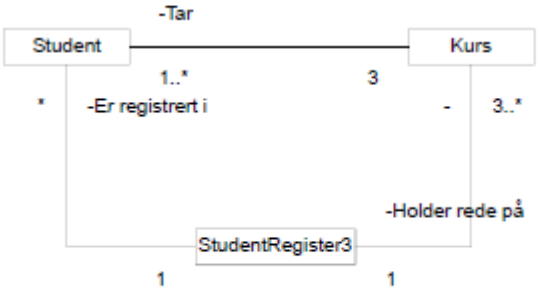
✗ Program forts.

```
class Student {
   String navn;
   Kurs [] mineKurs = new Kurs[3];

  Student(){mineKurs = new Kurs[0];}

   Student(String navn, Kurs [] k){
     this.navn = navn;
     for (int i = 0; i<k.length; i++ )
          mineKurs[i] = k[i];
          mineKurs[i].antStudenter++;
   }   }

   void skrivUt() {
     System.out.println("Student med navn:"+ navn+ ",og kurs:");
         for (int i = 0;i < mineKurs.length; i ++)
                System.out.println(mineKurs[i].kurskode);
}}

class Kurs {
  String kurskode ;
  int antStudenter = 0;

  Kurs(String k) {
       kurskode = k;
 }}
```

```
>java StudentRegister3
Student med navn:Ola N, og kurs:
INF1000
INF1040
MAT1030
Student med navn:Åsne S, og kurs:
INF1000
INF1040
MAT1030
Student med navn:null, og kurs:
```

✗ Sammenligning: Objektdiagram og Klassediagram





Oppramstyper

- Enummerering – å lage egne oppramstyper
  - Brukes til å lage 'typer' som har et lite antall verdier, ofte tekst

```
enum Farge {rød,grønn,blå,gul;};

class EnumEks {
  public static void main(String[] args) {

    Farge f = Farge.rød;
    System.out.println("Farge f er:" + f);

    for (Farge ff: Farge.values())
     System.out.println("ff:"+ ff
        +", nummer:" + ff.ordinal());
   }
} // end class EnumEks
```

```
Farge f er:rød
ff:rød, nummer:0
ff:grønn, nummer:1
ff:blå, nummer:2
ff:gul, nummer:3
```

- Slik 'enum' kan ha metoder , og hver verdi har et tall assosiert ved seg.
  En enum virker omtrent som en klasse-deklarasjon.

```
public enum Karakter {
    A(6),
    B(5),
    C(4),
    D(3),
    E(2),
    F(0);

    final int verdi;

    boolean erBedre(Karakter k){
        return this.verdi > k.verdi;
    }

    Karakter (int v){
        verdi = v;
    }
  }// end enum Karakter
```

```
class EnumEks2 {
  public static void main(String[] args) {

    Karakter min = Karakter.A, din = Karakter.E;
    System.out.println("Karakteren min  er:" + min);

    if (min.erBedre(din))
      System.out.println("Min karakter:" + min
      + " er bedre enn din " + din);
     }
} // end class EnumEks2
```

```
Karakteren min  er:A
Min karakter:A er bedre enn din E
```

Sortering

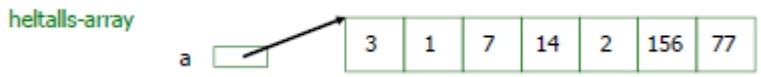
- ✗ Sortering
  - Lære å løse et vanskelig problem
    - Sortering – mange metoder, her Innstikksortering
    - Sortere hva:
      - Heltall
      - Tekster
  - Lære abstraksjon
    - Når vi har løst ett problem, kan lignende problemer løses tilsvarende
  - Lære å lage ’proff’ programvare ved å lage en generell klasse (en vektøyboks) for sortering
    - Hvordan deklarere en slik klasse
    - Javadoc – lage dokumentasjon
    - Testing
    - Hvordan utvikle programmet
  - Mange datatyper kan sorteres
    - Tall
    - Tekster (leksikografisk =i samme rekkefølge de ville stått i et leksikon)
    - Tabeller av tekster eller tall
  - Vi må ha en algoritme (fremgangsmåte) for sortering
    - Det finns mange titalls (hundretalls) metoder for sortering
    - Her skal vi lære den som er raskest når vi skal sortere få elementer, si < 50 elementer
- ✗ Hvorfor sorterer vi
  - For å få noen tall i sortert rekkefølge
    - eks: lotto-tallene
  - Sortere tekster (navnelister)
  - Sortere noen opplysninger som hører sammen. Sorterer da på en av opplysningene.
    - Eks. Telefonkatalogen: navn, adresse, telefonnummer sortert på navn
    - Eks. Oslo Maraton eller Birken sortert etter :navn, plassering
- ✗ Vi skal først lære å sortere heltall
  - Dette skal vi så med minimale endringer bruke til å sortere:
    - String-arrayer (tekster)

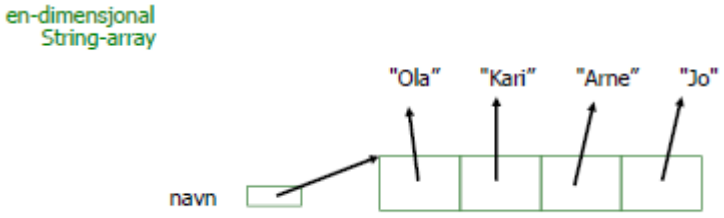
✗ Vi ønsker en klasse med to varianter av sortering:
Heltall og tekster

```
public class ISort {

  public static void sorter(int [] a) {

  }
  public static void sorter(String [] a) {

  }
} // end class ISort
```

Test-program for sortering

```
class TestInnstikkSortering
{

  public static void  main ( String[] args)  {

      int [] a = {3,1,7,14,2,156,77};
      String [] navn = {"Ola", "Kari", "Arne", "Jo"};

        // sorter heltall - skriv ut
        ISort.sorter(a);
        for (int i = 0; i < a.length; i++)
          System.out.println("a[" + i +"]= " + a[i]);

          System.out.println("\n Test tekst-sortering:");

        // sorter Stringer - skriv ut
        ISort.sorter(navn);
        for (int i = 0; i < navn.length; i++)
          System.out.println("navn[" + i +"]= " + navn[i]);


}}
```



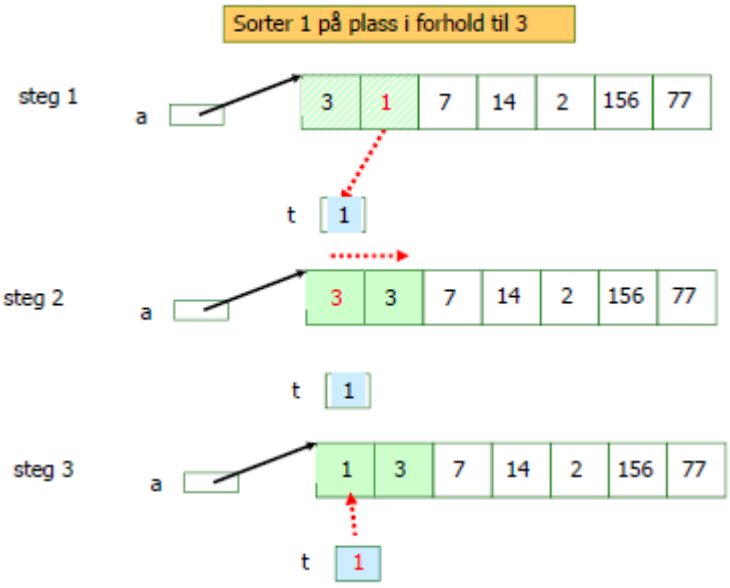


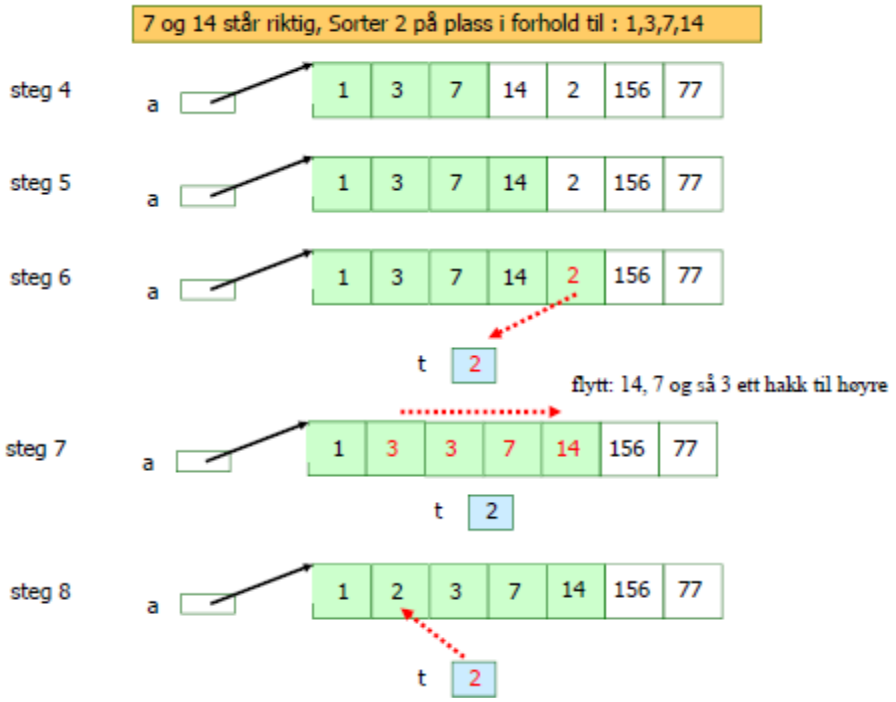
```
>java InnstikkSortering
a[0]= 3
a[1]= 1
a[2]= 7
a[3]= 14
a[4]= 2
a[5]= 156
a[6]= 77

 Test tekst-sortering:
navn[0]= Ola
navn[1]= Kari
navn[2]= Arne
navn[3]= Jo
```

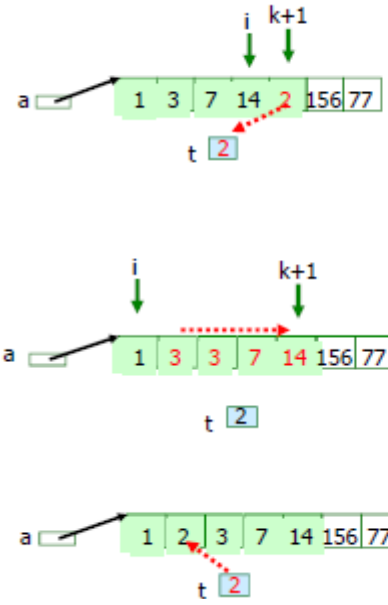
Test av test-programmet med **tomme** sortering-metoder

✗ En algoritme for å sortere heltall – innstikksmetoden

- Se på arrayen ett for ett element fra venstre mot høyre
- Sorterer det vi hittil har sett på, ved :
  - Hvis det nye elementet vi ser på ikke er sortert i forhold til de vi allerede har sett på:
    - Ta ut dette elementet (gjem verdien i en variabel t)
    - Skyv på de andre elementene vi her sett på en-etter-en, ett hakk høyreover til elemetet i t kan settes ned på sortert plass.
    - Da er den delen vi har sortert ett element lenger (fra venstre)
  - Når vi har sett på alle elementene, er hele arrayen sortert
  - Observasjon : Det første elementet, er det sortert i forhold til seg selv







Kode for å flytte **ett** element på plass :

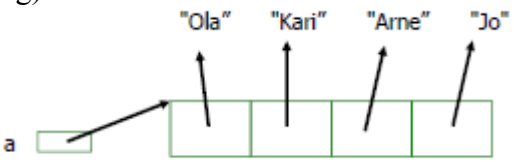
```
// a[k +1] står på
// feil plass, ta den ut
int t = a[k + 1], i = k;

// skyv a[i] mot høyre ett hakk til
// vi finner riktig plass til t
while (i >= 0 && a[i] > t) {
   a[i + 1] = a[i];
   i--;
}

// sett t inn på riktig plass
  a[i + 1] = t;
```

```
public class ISort {

  public static void sorter(int [] a) {
       for (int k = 0 ; k < a.length-1; k++) {
         if (a[k] > a[k+1]) {
              // a[k +1 ] står på feil plass, ta den ut
              int t = a[k + 1], i = k;
              // skyv a[i] mot høyre ett hakk til
              // vi finner riktig plass til t
              while (i >= 0 && a[i] > t) {
                     a[i + 1] = a[i];
                     i--;
              }
              // sett t inn på riktig plass
              a[i + 1] = t;
          }
       }
  } // end heltall-sortering
```

```
>java InnstikkSortering
a[0]= 1
a[1]= 2
a[2]= 3
a[3]= 7
a[4]= 14
a[5]= 77
a[6]= 156
 Test tekst-sortering:
navn[0]= Ola
navn[1]= Kari
navn[2]= Arne
navn[3]= Jo
```

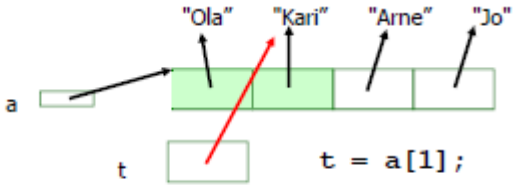
Resultat av sortering med heltalls-metoden kodet, den andre uten kode

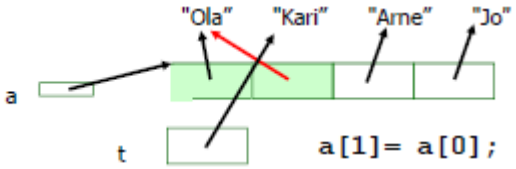
✗ Sortering av tekster (String)

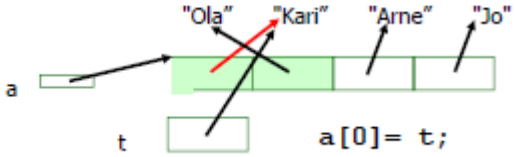


– Vi skal sortere denne ved å bytte om på pekerne (la a[0] peker på ”Arne”,..osv) med innstikkmetoden

Sortere de to første elementene ved å bytte om pekere







```
public static void sorter(int [] a) {
       // Sorterer heltallsarrayaen 'a'.
       for (int k = 0 ; k < a.length-1; k++) {
          if (a[k] > a[k+1]) {
               int t = a[k + 1];
               int i = k;
               while (i >= 0 && a[i] > t) {
                       a[i + 1] = a[i];
                       i--;
               }
               a[i + 1] = t;
       }  }
  } // end heltall-sortering

  public static void sorter(String [] a) {
        // Sorterer String-arrayen 'a'.
       for (int k = 0 ; k < a.length-1; k++) {
           if( a[k].compareTo(a[k+1]) > 0 ){
               String t = a[k + 1];
               int i = k;
               while (i >= 0 &&  ( a[i].compareTo(t) > 0) ){
                       a[i + 1] = a[i];
                       i--;
               }
                a[i + 1] = t;
       }  }
  } // end String-sortering
```

```
>java InnstikkSortering
a[0]= 1
a[1]= 2
a[2]= 3
a[3]= 7
a[4]= 14
a[5]= 77
a[6]= 156
 Test tekst-sortering:
navn[0]= Arne
navn[1]= Jo
navn[2]= Kari
navn[3]= Ola
```

Test med heltall og enkel String-sortering kodet

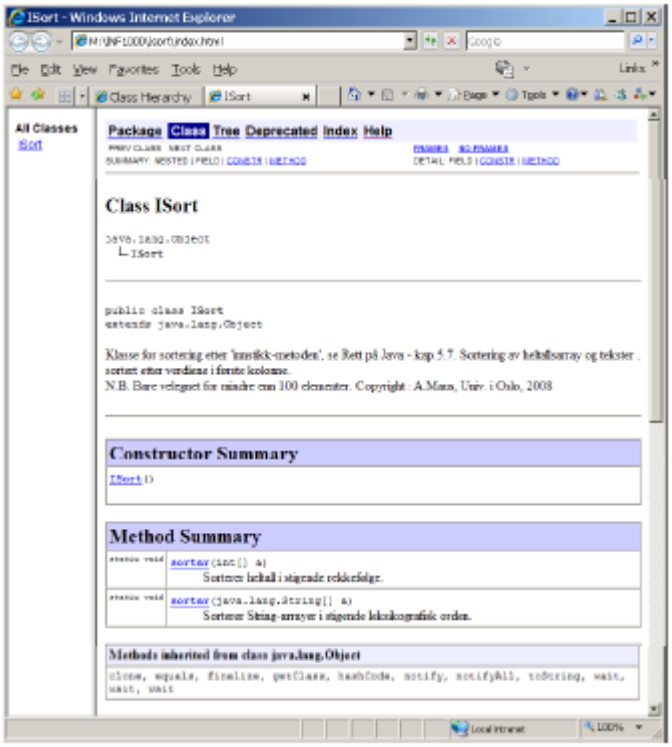
Javadoc

- Javadoc – proff dokumentasjon av klassene
  - Legg inn spesielle kommentarer i programmet ditt (over hver metode og klasse)
  - I disse kommentarene kan man legge HTMLkommandoer (som
 for å få linjeskift)
  - Kjør programmet 'javadoc', og automatisk har du en fin dokumentasjon
  - Største fordel: Kode og dukumentasjon vedlikeholdes på samme fil.

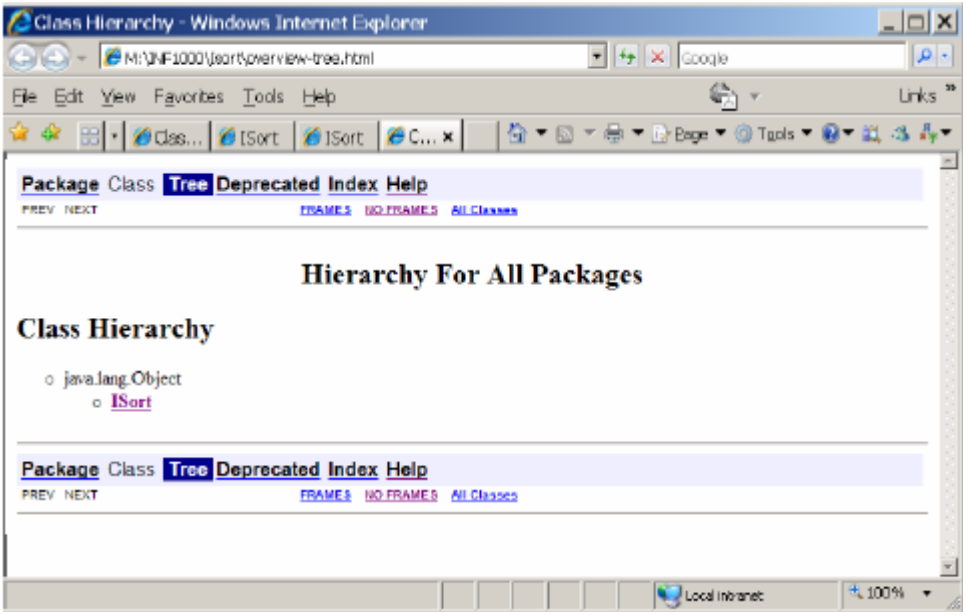
```
/**
* Klasse for sortering etter 'innstikk-metoden', se
* Rett på Java - kap.5.6.
* Sortering av heltallsarray, tekster og en to-dimensjonal
* tekst-array  sortert etter verdiene i første kolonne.<br>
*
* N.B. Bare velegnet for mindre enn 100 elementer.
*
* Copyright : A.Maus, Univ. i Oslo, 2008
*************************************************************/
public class ISort {

  /**
  * Sorterer heltall i stigende rekkefølge.
  * @param a heltallsarrayen som sorteres. <br>
  * Endrer parameter-arrayen.
  ******************************************/
  public static void sorter(int [] a) {
  }
  /**
  * Sorterer String-arrayer i stigende leksikografisk orden.
  * @param a arrayen som sorteres.<br>
  * Endrer parameter-arrayen
  ******************************************/
  public static void sorter(String [] a) {

  }

} // end class ISort
```

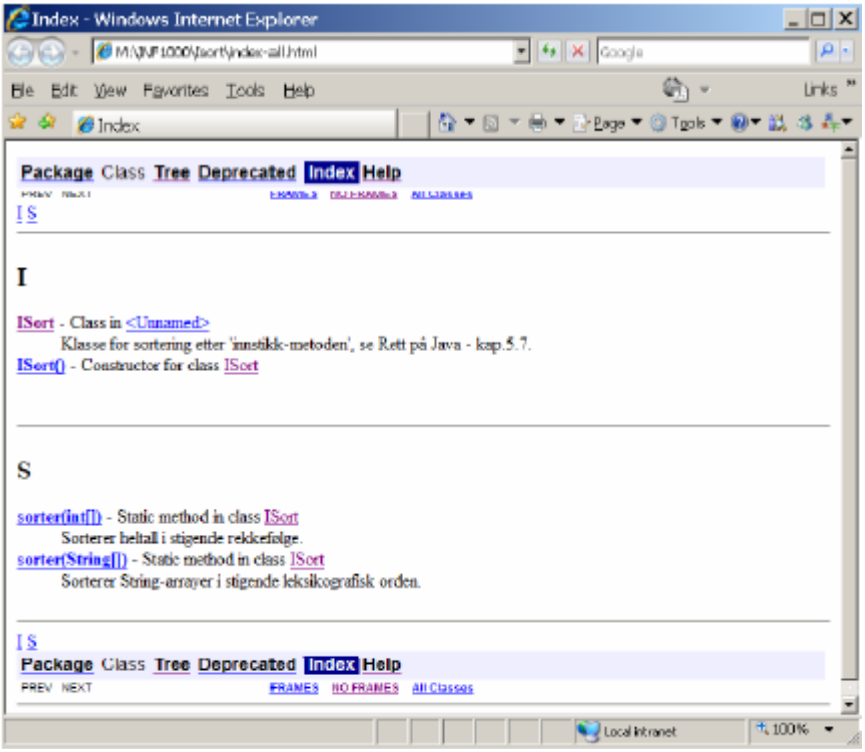
- Dokumentasjon av klassen og metodene – javadoc

```
M:\INF1000\Isort>javadoc -package ISort.java
Loading source file ISort.java...
Constructing Javadoc information...
Standard Doclet version 1.5.0_02
Building tree for all the packages and classes...
Generating ISort.html...
Generating package-frame.html...
Generating package-summary.html...
Generating package-tree.html...
Generating constant-values.html...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating allclasses-noframe.html...
Generating index.html...
Generating help-doc.html...
Generating stylesheet.css...

M:\INF1000\Isort>
```


ISort - Windows Internet Explorer
File Edit View Favorites Tools Help
All Classes
ISort
Package Class Tree Deprecated Index Help
Class ISort
java.lang.Object
└ ISort
public class ISort
extends java.lang.Object
Klasse for sortering etter 'innstikk-metoden'. se Rett på Java - kap 5.7. Sortering av heltallsarray og tekster . sortert etter verdiene i første kolonne.
N.B. Bare velegnet for mindre enn 100 elementer. Copyright : A.Maus, Univ. i Oslo, 2008
Constructor Summary
ISort()
Method Summary
static void sorter(int[] a)
Sorterer heltall i stigende rekkefølge.
static void sorter(java.lang.String[] a)
Sorterer String-arrayer i stigende leksikografisk orden.
Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait


ISort - Windows Internet Explorer
File Edit View Favorites Tools Help
Method Detail
sorter
public static void sorter(int[] a)
Sorterer heltall i stigende rekkefølge.
Parameters:
a - heltallsarrayen som sorteres Endrer parameter-arrayen.
sorter
public static void sorter(java.lang.String[] a)
Sorterer String-arrayer i stigende leksikografisk orden.
Parameters:
a - arrayen som sorteres Endrer parameter-arrayen
Package Class Tree Deprecated Index Help


Class Hierarchy - Windows Internet Explorer
M:\INF1000\isort\overview-tree.html
File Edit View Favorites Tools Help
Package Class Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES All Classes
Hierarchy For All Packages
Class Hierarchy
java.lang.Object
ISort
Package Class Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES All Classes
Local intranet
100%


Index - Windows Internet Explorer
M:\INF1000\isort\index-all.html
File Edit View Favorites Tools Help
Index
Package Class Tree Deprecated Index Help
I S
I
ISort - Class in <Unnamed>
Klasse for sortering etter 'innstikk-metoden', se Rett på Java - kap.5.7.
ISort() - Constructor for class ISort
S
sorter(int[]) - Static method in class ISort
Sorterer heltall i stigende rekkefølge.
sorter(String[]) - Static method in class ISort
Sorterer String-arrayer i stigende leksikografisk orden.
I S
Package Class Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES All Classes
Local intranet
100%