

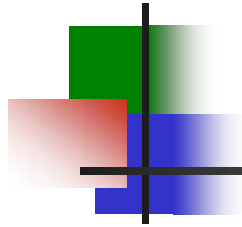


# Inf1000 uke 5 – 20. sept 2011

---

Litt om klasser og objekter, repetisjon  
av metoder, filer med easyIO, tekst

Arne Maus og Siri Moe Jensen,  
Inst. for informatikk, Univ i Oslo



# 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.)

*Metoder i to klasser skal vi lære idag*

*Klasser, objekter og metoder i flere klasser skal vi lære om to uker*

Vi skal fra nå ha et **nytt 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);
    }
}
```



# Hva skjer når vi kjører det

```
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:

```
      modifikatorer      retur-   metode-   formelle
                        type      navn      parametre
      ┌──────────┬──┴──┬──┴──┬──────────┐
public static void main (String [] args) {
    .....      } metodekropp ("innmat" )
    .....
}
```

- 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 metode-deklarasjonen. 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);
```



# Eksempel: metode uten input/output

- 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 deklarerert på klassenivå, inne i klassen, men utenfor metodene.
  - **Lokale variable**: dette er variable som deklarereres inni metoden. Disse er definert fra og med der deklarasjonen gjøres og til slutten av blokken de er deklarerert i.
  - **Parametre**: dette er variable som deklarereres 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 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 og argumenter

```
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

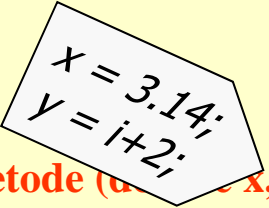
Merk: et annet navn for argumenter er *aktuelle parametre*, og et annet navn for parametre er *formelle parametre*.



## Verdien til parameterene 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:

```
public static void main (String[] args) {  
    A aa = new A();  
    int i = 17;  
    aa.minMetode(3.14, i +2);  
}  
}  
  
class A{  
    void minMetode (double x, int y)  {  
        // nå kan x og y brukes med de verdier de fikk i kallet  
        ....  
    }  
}
```



*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);**

## Enklere å løse Oblig2 med metoder (8 stk)

```
import easyIO.*;

class Oblig2 {
    public static void main(String[] args) {
        Olje ol = new Olje();
        ol.ordreløkke(); // Kjører metoden ordreløkke i klassen Olje
        System.out.println("-- Programmet avslutter --");
    }
} // end class Oblig2

class Olje {
    In tast = new In();
    Out skjerm = new Out();
    // Her kan du deklarere arrayene eier[][] og utvunnet[][]:

    void ordreløkke() {
        int ordre = 0;

        while (ordre != 6) {
            skrivMeny(); // metode som
            ordre = velgOperasjon(); // (kode som leser fra tastatur og returnerer ordrenum.)

            switch (ordre) {
                case 1: kjøpEtFelt(); break;
                case 2: listeOverFeltMedEier(); break;
                case 3: lagOversiktskart(); break;
                // ... fyll inn case-er for de to andre ordrene her.
                default: break;
            }
        }
    } // end while

    void kjøpEtFelt() {...}
    void listeOverFeltMedEier() {

        // ... osv. (3 metoder + skrivMeny() og velgOperasjon()-metodene)
    } // end class Olje
}
```



# 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;
```



## II) 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 i dag: Lesing & skriving på **filer**



## Eksempel

Vi importerer pakken easyIO.

```
import easyIO.*;
```

Vi åpner filen for lesing

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

```
        In fil = new In("filnavn");
```

Her leses hele første linje av filen

```
        String s = fil.inLine();
```

```
        System.out.println("Første linje var: " +  
                             s);
```

```
        fil.close();
```

Her lukkes filen fordi vi er ferdige med å bruke den.

```
    }
```

```
}
```



# 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);
}
```



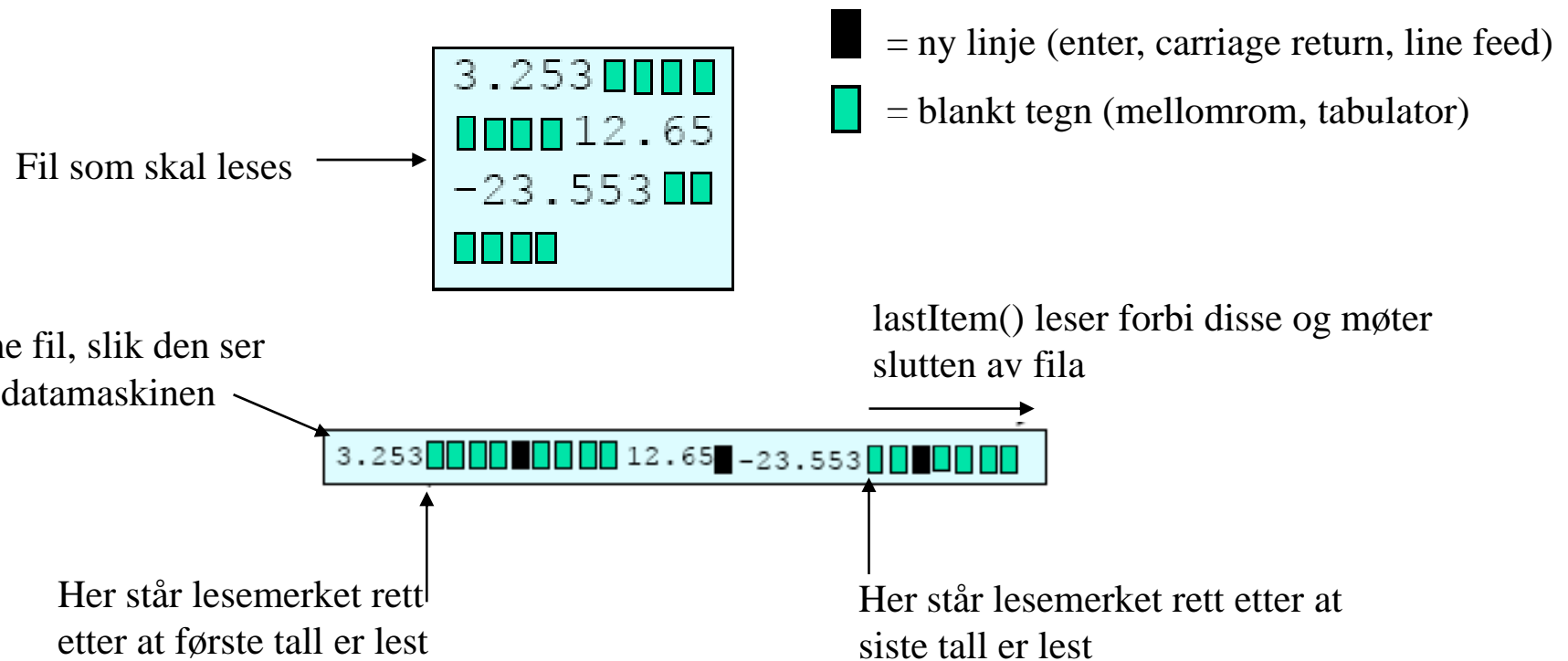
## Program som leser en tekstfil linje for linje:

```
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]);
        }
    }
}
```

# lastItem 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 benytte en for-løkke.



## Eksempel: fil med kjent lengde

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 på forrige foil 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.

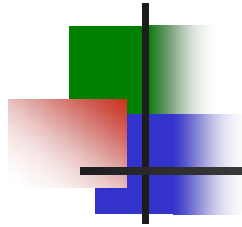
```
import easyIO.*;
class LesTallMedMerke {
    public static void main (String [] args) {
        double [] x = new double[100]; // antar max 100 tall
        In innfil = new In("tall-med-merke.txt"); // på fil
        double siste = 0;
        int ant = 0;
        while (siste != -999) {
            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:

<b>Antall</b>	<b>Pris</b>	<b>Varenavn</b>
35	23.50	Oppvaskkost
53	33.00	Kaffe
97	27.50	Pizza
...	...	...
...	...	...



## Fremgangsmåte

---

- Den første linje 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 enlinje. 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.



# Eksempel

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();
```



## Skrive til fil

Vi importerer pakken easyIO.

```
import easyIO.*;
class SkrivTilFil {
    public static void main (String [] args) {

        Out fil = new Out("nyfilnavn");

        fil.outln("Dette er første linje");

        fil.close();

    }
}
```

Vi åpner filen for Skrivning

Vi må huske å  
lukke filen til slutt

Her skrives en linje  
med tekst til filen



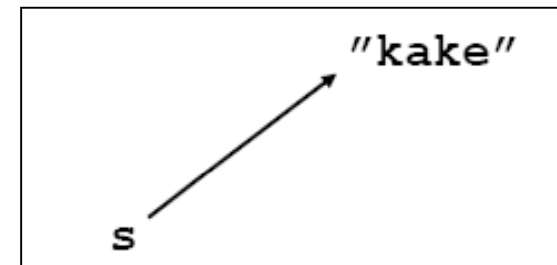
## EasyIO: Hvilke skrivemetoder finnes?

Datatype	Eksempel	Beskrivelse
int	<code>fil.out(x);</code> <code>fil.out(x, 6);</code>	Skriv x Skriv x høyrejustert på 6 plasser
double	<code>fil.out(x, 2);</code> <code>fil.out(x, 2, 6);</code>	Skriv x med 2 desimaler Skriv x med 2 desimaler på 6 plasser
char	<code>fil.out(c);</code>	Skriv c
String	<code>fil.out(s);</code> <code>fil.out(s, 6);</code>	Skriv s Skriv s på 6 plasser (venstrejustert)
	<code>fil.outln();</code>	Skriv en linjeskift
	<code>fil.close();</code>	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: ...**

# 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>)

	000	001	002	003	004	005	006	007
0	NUL	DLE	SP	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

	008	009	00A	00B	00C	00D	00E	00F
0	XXX	DCS	NB SP	°	À	Đ	à	đ
1	XXX	PU1	¡	±	Á	Ñ	á	ñ
2	BPH	PU2	¢	²	Â	Ò	â	ò
3	NBH	STS	£	³	Ã	Ó	ã	ó
4	IND	CCH	¤	´	Ä	Ô	ä	ô
5	NEL	MW	¥	µ	Å	Ö	å	ö
6	SSA	SPA	¦	¶	Æ	Ø	æ	ø
7	ESA	EPA	§	·	Ç	×	ç	÷
8	HTS	SOS	¨	¸	È	Ø	è	ø
9	HTJ	XXX	©	¹	É	Ù	é	ù
A	VTS	SCI	ª	º	Ê	Ú	ê	ú
B	PLD	CSI	«	»	Ë	Û	ë	û
C	PLU	ST	¬	¼	Ì	Ü	ì	ü
D	RI	OSC	½	½	Í	Ý	í	ý
E	SS2	PM	¾	¾	Î	Þ	î	þ
F	SS3	APC	—	¿	Ï	ß	ï	ÿ

	16A	16B	16C	16D	16E	16F
0	Ÿ	†	‡	¼	½	¾
1	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
2	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
3	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
4	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
5	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
6	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
7	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
8	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
9	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
A	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
B	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
C	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
D	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
E	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ
F	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ	Ÿ



## Teste om to tekster er like

- For å teste om to tekststrenger er like, brukes equals:  
// Anta at s og t er tekstvariable (og at s ikke har verdien **null**)

```
if (s.equals(t)) {  
    System.out.println("Tekstene er like");  
} else {  
    System.out.println("Teksten er forskjellige");  
}
```

- Bruk av == virker av og til, men ikke alltid:

```
String s = "abc";  
String t = "def";  
String tekst1 = s + t;  
String tekst2 = s + t;
```

Nå er `tekst1.equals(tekst2)` **true**, mens `tekst1 == tekst2` er **false**.



## De enkelte tegnene i en tekststreng

- Tegnene i en tekststreng har posisjoner indeksert fra 0 og oppover

0	1	2	3
'k'	'a'	'k'	'e'

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

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

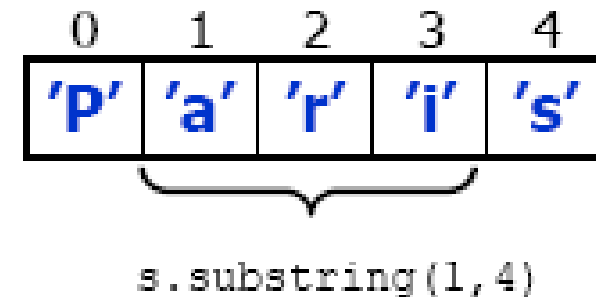
- Vi kan erstatte alle forekomster av et tegn med et annet tegn:

```
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:

```
s.substring(index1, index2)
```

Første posisjon som  
skal være med

Første posisjon som  
*ikke* skal være med

- 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!

# Oppgave 1

- Hva skriver programmet

```
import easyIO.*;
class Alfabetisk {
    public static void main (String [] args) {
        String sString = "abCDØÅ";
        String tString = "bCdDØÆ";

        for(int i=0; i < sString.length();i++){
            String s = sString.substring(i, i+1);
            String t = tString.substring(i, i+1);

            int k = s.compareTo(t);

            if (k < 0) {
                System.out.print(s + " er alfabetisk foran " + t);
            } else if (k == 0) {
                System.out.print(s + " er lik " + t);
            } else {
                System.out.print(s + " er alfabetisk bak " + t);
            }

            System.out.print("\n");
        }
    }
}
```

M:\INF1000\Programmer>java Alfabetisk

a er alfabetisk foran b	k er -1
b er alfabetisk bak C	k er 31
C er alfabetisk foran d	k er -33
D og D er like	k er 0
° er alfabetisk bak ≠	k er 32
† er alfabetisk foran ‡	k er -1

arnem@sviurr ~/INF1000/Programmer> java Alfabetisk

a er alfabetisk foran b	k er -1
b er alfabetisk bak C	k er 31
C er alfabetisk foran d	k er -33
D og D er like	k er 0
ø er alfabetisk bak Ø	k er 32
Å er alfabetisk foran Æ	k er -1



## 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...



## Oppgave 2

- Hva skriver programmet ut?

```
import easyIO.*;
class SlutterMedOppgave {
    public static void main (String [] args) {
        String s = "julenisse";
        String t = s.substring(4);
        String v = s.substring(0,4);
        if(s.startsWith("jule")){
            System.out.println("A");
        }
        if(t.startsWith("jule")){
            System.out.println("B");
        }
        if(v.startsWith("jule")){
            System.out.println("C");
        }
    }
}
```



## Oppgave 3

---

- Hva skriver programmet ut?

```
import easyIO.*;
class ReplaceOppgave {
    public static void main (String [] args) {
        String s = "javaprogram";
        String l = s;
        s.replace('a', 'i');
        l.replace('a', 'o');
        System.out.println(s);
        System.out.println(l);

        l="jp";
        System.out.println(s);
    }
}
```