

I-120 (H-2003)

Foreleser:

Michal Walicki

e-post: michal@ii.uib.no

kontor: HiB, 4145

Gruppeansvarlig:

Ørjan Bergman

epost: orjanb@ii.uib.no

websider: [studentportal.uib.no] --> I-120 --> Meldingar (--> kursside)

foreløpig: [http:// www.ii.uib.no / ~orjanb / i120 / i120.htm](http://www.ii.uib.no/~orjanb/i120/i120.htm)

Sjekk den jevnlig (hver uke) –

oppgavetekster, samt all informasjon

(også den som ikke er annonsert på forelesninger) legges ut der

Pensum:

- “Data Structures and Algorithms using JAVA”
M.T.Goodrich & R.Tamassia, second edition
(store deler)
- ev. notater delt ut på forelesning

Mandag

Tirsdag

Onsdag

Torsdag

Fredag

	Forelesninger (Realfagbygget) Tirsdag, onsdag 12:15-14			

Mandag	Tirsdag	Onsdag	Torsdag	Fredag
	<p data-bbox="405 516 1163 716"> Forelesninger (Realfagbygget) Tirsdag, onsdag 12:15-14 </p>			<p data-bbox="1432 321 1988 885"> Oppgavegjennomgang Fredag, 10:15-12 Aud. A (Terminal) Ørjan B. </p> <ul style="list-style-type: none"> <li data-bbox="1432 581 1816 625">– noen løsningsforslag <li data-bbox="1432 646 1927 787">– oppgaver til neste uke <ul style="list-style-type: none"> <li data-bbox="1465 698 1810 742">• legges ut på nettet <li data-bbox="1465 743 1927 787">• til 29.8 ligger allerede ute <li data-bbox="1432 808 1869 852">– starter på fredag 29.8

Mandag	Tirsdag	Onsdag	Torsdag	Fredag
	<p data-bbox="405 516 1163 716" style="color: red;">Forelesninger (Realfagbygget) Tirsdag, onsdag 12:15-14</p>			<div data-bbox="1432 321 1988 885" style="background-color: #e0e0e0; padding: 5px;"> <p data-bbox="1442 329 1978 524">Oppgavegjennomgang Fredag, 10:15-12 Aud. A (Terminal) Ørjan B.</p> <ul style="list-style-type: none"> <li data-bbox="1442 589 1816 630">– noen løsningsforslag <li data-bbox="1442 654 1927 792">– oppgaver til neste uke <ul style="list-style-type: none"> <li data-bbox="1474 703 1808 743">• legges ut på nettet <li data-bbox="1474 751 1927 792">• til 29.8 ligger allerede ute <li data-bbox="1442 816 1864 857">– starter på fredag 29.8 </div> <div data-bbox="1432 885 1988 1273" style="padding: 5px;"> <p data-bbox="1505 930 1944 1019" style="text-align: center;">Alle må jobbe med disse på egen hånd</p> </div>

Mandag	Tirsdag	Onsdag	Torsdag	Fredag
				<p>Oppgavegjennomgang Fredag, 10:15-12 Aud. A (Terminal) Ørjan B.</p> <ul style="list-style-type: none"> – noen løsningsforslag – oppgaver til neste uke <ul style="list-style-type: none"> • legges ut på nettet • til 29.8 ligger allerede ute – starter på fredag 29.8
	<p>Forelesninger (Realfagbygget) Tirsdag, onsdag 12:15-14</p>			<p>Alle må jobbe med disse på egen hånd</p>
<p>Diskusjonsgrupper – diskusjon av ukens oppgaver (for aktive studenter)</p>				

Mandag	Tirsdag	Onsdag	Torsdag	Fredag
10:15-12 IKT	10:15-12			Oppgavegjennomgang Fredag, 10:15-12 Aud. A (Terminal) Ørjan B. – noen løsningsforslag – oppgaver til neste uke <ul style="list-style-type: none"> • legges ut på nettet • til 29.8 ligger allerede ute – starter på fredag 29.8
	Forelesninger (Realfagbygget) Tirsdag, onsdag 12:15-14			
16:15-18				Alle må jobbe med disse på egen hånd
Diskusjonsgrupper <ul style="list-style-type: none"> – diskusjon av ukens oppgaver (for aktive studenter) – påmelding – e-mail: orjanb@ii.uib.no, med 1- og 2-prioritet – starter neste uke (grupperom 4, Realfagbygget) 				

Obligatoriske øvelser

2 obligatoriske øvelser som må godkjennes for å ta eksamen:

Oblig 1 ut *ca.* 19/9 inn *ca.* 3/10

Oblig 2 ut *ca.* 17/10 inn *ca.* 14/11

Eksamen /12

Pensum: fra boken (foreløpig)

	unntatt	kursorisk	tema
KAP. 1			JAVA – I-110 (Gjennomgås ikke unntatt 1.8)
KAP. 2			OO, ABSTRAKSJON ...
KAP. 3			ALGORITME-TIDSANALYSE
KAP. 4	4.2.4, 4.5	4.1.3, 4.2.3, 4.4	STABEL, KØ, LISTE, <i>ADAPTER</i>
KAP. 5			SEKVENSS; RANK, <i>POSITION</i>; <i>ENUMERATION-ITERATOR</i>
KAP. 6		6.4.4	TRÆR, B-TRÆR + BSF/DFS (PRE/POST/IN-ORDER)
KAP. 7		7.3.5, 7.4	PRIORITETSKØ, HEAP; TOTALORDNING/<i>COMPARATOR</i>
KAP. 8	8.6, 8.7	8.3.3 - 8.3.7	ORDBOK, BST, HASHTAB
KAP. 9	9.3 - 9.6		SØKETRÆR: BINÆRE, AVL
KAP. 10	10.7	10.5	SORTERING:QUICKSORT (MERGESORT); RANDOMISERING
KAP. 11	GÅR UT		
KAP. 12			GRAFER: DFS/BFS, SSSP, MST

Pensum: fra boken (foreløpig)

unntatt

kursorisk

tema

KAP. 1

JAVA – I-11

tt 1.8)

KAP. 2

KAP. 3

KAP. 4

4.2.4, 4.5

KAP. 5

KAP. 6

INF-102 (???)

1. bytt påmelding til I-120

ITERATOR

KAP. 7

(PRE/POST/IN-ORDER)

KAP. 8

8.

, HEAP; TOTALORDNING/**COMPARATOR**

KAP. 9

9.3 -

ADBOK, BST, HASHTAB

KAP. 10

10.7

SØKETRÆR: BINÆRE, AVL

SORTERING: QUICKSORT (MERGESORT); RANDOMISERING

~~KAP. 11~~

GÅR UT

KAP. 12

GRAFER: DFS/BFS, SSSP, MST

Pensum: fra boken (foreløpig)

unntatt

kursorisk

tema

KAP. 1

JAVA – I-11

tt 1.8)

KAP. 2

KAP. 3

KAP. 4

4.2.4, 4.5

KAP. 5

KAP. 6

INF-102 (???)

1. bytt påmelding til I-120

KAP. 7

2. er det noen som skal ta INF-102,

KAP. 8

8. – så vil vi underveis informere om deler av I-120 pensum som ikke inngår i INF-102

KAP. 9

9.3 -

– samme forelesninger, øvelsesoppleg og obligatoriske oppgaver

, HEAP; TOTALORDNING/**COMPARATOR**

KAP. 10

10.7

ADBOK, BST, HASHTAB

KAP. 11

GÅR UT

KAP. 12

GRAFER: DFS/BFS, SSSP, MST

Mål og Mening

Ikke kun programmering (slik som I110)

men **effektiv** og **korrekt** programmering
dvs **algoritmer** og **strukturering**

Mål og Mening

Ikke kun programmering (slik som I110)

men **effektiv** og **korrekt** programmering
dvs **algoritmer** og **strukturering**

type tekst/program	boksider	tekstlinjer	listelengde
Wolfe 'Forfengelighetens fyrverkeri'	661	26 440	132m
Tolkien 'Hobbiten+Ringtrilogien+Simarillion'	1 847	73 880	222m
enkel kompilator	300	12 000	36m
lønnssystem, industri	600	24 000	80m
utlånssystem, forsikring	7 500	300 000	900m
abonnentsystem, avis	10 650	425 000	1 275m
logistikksystem, oljeplattform	18 750	750 000	2 250m

Mål og Mening

Ikke kun programmering (slik som I110)

men **effektiv** og **korrekt** programmering
dvs **algoritmer** og **strukturering**

type tekst/program	boksider	tekstlinjer	listelengde
Wolfe 'Forfengelighetens fyrverkeri'	661	26 440	132m
Tolkien 'Hobbiten+Ringtrilogien+Simarillion'	1 847	73 880	222m
enkel kompilator	300	12 000	36m
lønnssystem, industri	600	24 000	80m
utlånssystem, forsikring	7 500	300 000	900m
abonnentsystem, avis	10 650	425 000	1 275m
logistikksystem, oljeplattform	18 750	750 000	2 250m

mengde data : **n** **n*10**

program

effektivt

1 min

10 min

lite effektivt

1 min

1 000 min

Programmereres aktivitet

år	nye prosjekter	vedlikehold, forbedring	vedlikehold / total
1950	90	10	10%

Programmereres aktivitet

år	nye prosjekter	vedlikehold, forbedring	vedlikehold / total
1950	90	10	10%
1960	8 500	1 500	15%
1970	65 000	35 000	35%
1980	1 200 000	800 000	40%
1990	3 000 000	4 000 000	57%
2000	4 000 000	6 000 000	60%

Programmereres aktivitet

år	nye prosjekter	vedlikehold, forbedring	vedlikehold / total
1950	90	10	10%
1960	8 500	1 500	15%
1970	65 000	35 000	35%
1980	1 200 000	800 000	40%
1990	3 000 000	4 000 000	57%
2000	4 000 000	6 000 000	60%
2010	5 000 000	9 000 000	64%
2020	7 000 000	14 000 000	66%

Manglende struktur og abstraksjon:

Programmereres aktivitet

år	nye prosjekter	vedlikehold, forbedring	vedlikehold / total
1950	90	10	10%
1960	8 500	1 500	15%
1970	65 000	35 000	35%
1980	1 200 000	800 000	40%
1990	3 000 000	4 000 000	57%
2000	4 000 000	6 000 000	60%
2010	5 000 000	9 000 000	64%
2020	7 000 000	14 000 000	66%

Manglende struktur og abstraksjon:

1. Typing

– grensesnitt vs. implementasjon

2. Objekt-orientering og Typing

3. Pakker

4. Abstrakte Data Typer

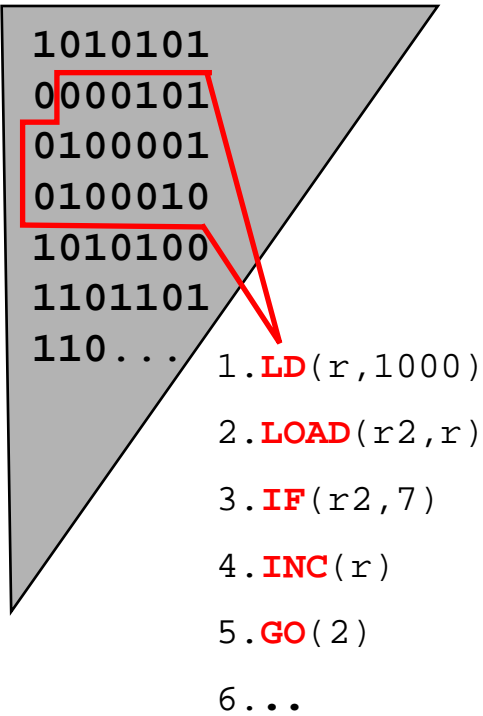
– grensesnitt (+ pseudokode)

– programmering med grensesnitt

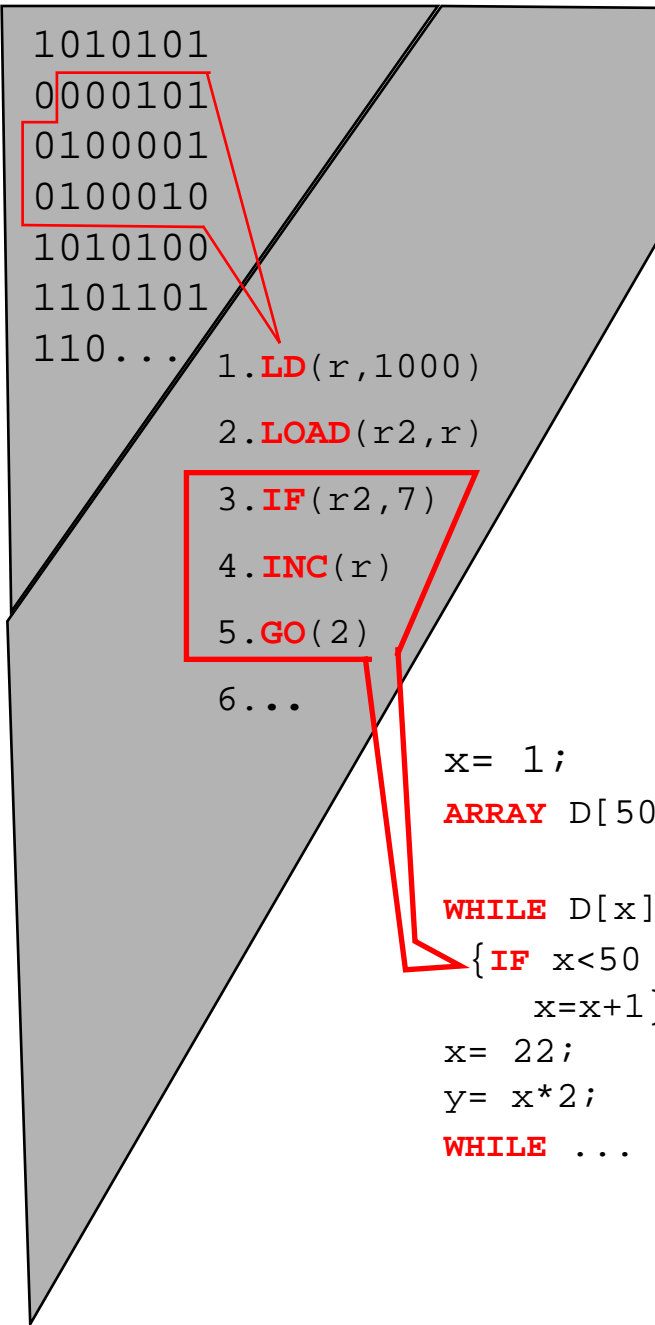
1. Typing og abstraksjon

1010101
0000101
0100001
0100010
1010100
1101101
110...

1. Typing og abstraksjon



1. Typing og abstraksjon



1. Typing og abstraksjon

1010101
0000101
0100001
0100010
1010100
1101101
110...

1. **LD**(r,1000)

2. **LOAD**(r2,r)

3. **IF**(r2,7)

4. **INC**(r)

5. **GO**(2)

6...

x= 1;
ARRAY D[50];

WHILE D[x]-k
{ **IF** x<50
 x=x+1}

x= 22;
y= x*2;

WHILE ...

INT max

STRING ARRAY

K[MAX],

D[MAX];

STRING PROCEDURE

get(**STRING** k) {

INT x;

BOOLEAN fant;

 x= 1;

 fant= false;

while !fant **do**

 { x=x+1;

 if K[x]==k

 fant=true;}

 return D[k];

}

PUT("NOK", "ORD");

GET("NOEKKEL");

1. Typing og abstraksjon

```
1010101  
0000101  
0100001  
0100010  
1010100  
1101101  
110...
```

```
1. LD(r,1000)  
2. LOAD(r2,r)  
3. IF(r2,7)  
4. INC(r)  
5. GO(2)  
6...
```

```
x= 1;  
ARRAY D[50];
```

```
WHILE D[x]-k  
{ IF x<50  
  x=x+1}  
x= 22;  
y= x*2;  
WHILE ...
```

```
INT max  
STRING ARRAY  
  K[MAX],  
  D[MAX];
```

```
STRING PROCEDURE  
get(STRING k) {  
  INT x;  
  BOOLEAN fant;  
  x= 1;  
  fant= false;  
  while !fant do  
  { x=x+1;  
    if K[x]==k  
      fant=true;}  
  return D[k];  
}
```

```
PUT("NOK", "ORD");  
GET("NOEKKEL");
```

```
RECORD ELEM {String k,d; int u;}
```

```
ELEM ARRAY N[50];
```

```
String PROCEDURE  
get(String k)  
{...}
```

1. Typing og abstraksjon

```
1010101
0000101
0100001
0100010
1010100
1101101
110...
```

```
1. LD(r,1000)
2. LOAD(r2,r)
3. IF(r2,7)
4. INC(r)
5. GO(2)
6...
```

```
x= 1;
ARRAY D[50];
```

```
WHILE D[x]-k
{ IF x<50
  x=x+1}
x= 22;
y= x*2;
WHILE ...
```

```
INT max
STRING ARRAY
    K[MAX],
    D[MAX];
```

```
STRING PROCEDURE
get(STRING k) {
    INT x;
    BOOLEAN fant;
    x= 1;
    fant= false;
    while !fant do
    { x=x+1;
      if K[x]==k
        fant=true;}
    return D[k];
}
```

```
PUT("NOK", "ORD");
GET("NOEKKEL");
```

```
RECORD ELEM {String k,d; int u;}
```

```
ELEM ARRAY N[50];
```

```
String PROCEDURE
get(String k)
{...}
```

```
CLASS ELEM {
    private String
        k,d;
    public String
        key(){return k;}
}
```

```
CLASS ORDBOK {
    private
        ELEM[] N[50];
    public String
        get(String e)
    public String
        put(Elem k)
}
```

1. Typing og abstraksjon

```

1010101
0000101
0100001
0100010
1010100
1101101
110...

```

```

1. LD(r,1000)
2. LOAD(r2,r)
3. IF(r2,7)
4. INC(r)
5. GO(2)
6...

```

```

x= 1;
ARRAY D[50];

```

```

WHILE D[x]-k
{ IF x<50
  x=x+1}
x= 22;
y= x*2;
WHILE ...

```

```

INT max
STRING ARRAY
    K[MAX],
    D[MAX];

```

```

STRING PROCEDURE
get(STRING k) {
    INT x;
    BOOLEAN fant;
    x= 1;
    fant= false;
    while !fant do
    { x=x+1;
      if K[x]==k
        fant=true;}
    return D[k];
}

```

```

PUT("NOK", "ORD");
GET("NOEKKEL");

```

```

RECORD ELEM {String k,d; int u;}

```

```

ELEM ARRAY N[50];

```

```

String PROCEDURE
get(String k)
{...}

```

```

CLASS ELEM {
    private String
        k,d;
    public String
        key(){return k;}
}

```

```

CLASS ORDBOK {
    private
        ELEM[] N[50];
    public String
        get(String e)
    public String
        put(Elem k)
}

```


1.a En type

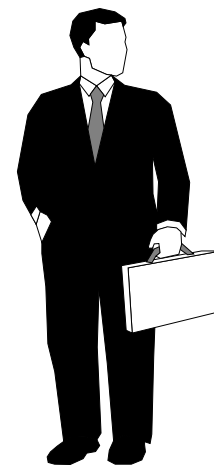
int, boolean, char, String, ...

gir bruker (programmerer) mulighet til å

- *konstruere nye instanser og*
- *behandle disse gjennom et gitt grensesnitt*

```
INT x=0, y=5;    x=x+2; x=(x % y); y=(y-x); ...
```

```
BOOLEAN a, b;    a=x<y; b=(b OR a); b=!b; ...
```



1.a En type

int, boolean, char, String, ...

gir bruker (programmerer) mulighet til å

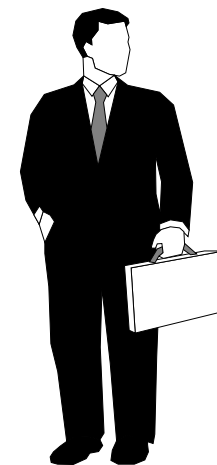
- *konstruere nye instanser og*
- *behandle disse gjennom et gitt grensesnitt*

```
INT x=0, y=5;    x=x+2; x=(x % y); y=(y-x); ...
```

```
1,2,3... : → INT
  +_ : INT × INT → INT
  -_ : INT × INT → INT
  %_ : INT × INT → INT
```

```
BOOLEAN a, b;    a=x<y; b=(b OR a); b=!b; ...
```

```
true,false : → BOOLEAN
  ==_ : INT × INT → BOOLEAN
  <_ : INT × INT → BOOLEAN
  !_ : BOOLEAN → BOOLEAN
```



1.a En type

int, boolean, char, String, ...

gir bruker (programmerer) mulighet til å

- *konstruere nye instanser og*
- *behandle disse gjennom et gitt grensesnitt*

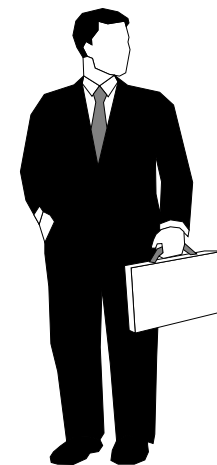
```
INT x=0, y=5;    x=x+2; x=(x % y); y=(y-x); ...
```

```
1,2,3... : → INT
  +_ : INT × INT → INT
  -_ : INT × INT → INT
  %_ : INT × INT → INT
```

```
BOOLEAN a, b;    a=x<y; b=(b OR a); b=!b; ...
```

```
true,false : → BOOLEAN
  ==_ : INT × INT → BOOLEAN
  <_ : INT × INT → BOOLEAN
  !_ : BOOLEAN → BOOLEAN
```

- *uten å vite noenting om implementasjon !!!*



1.b Brukerdefinerte typer

```
PUBLIC CLASS ORDBOK {  
    private Elem[] N;  
    PUBLIC ORDBOK(int k) { N= new Elem[k]; }  
    PUBLIC ELEM GET(STRING e) {...}  
    PUBLIC VOID PUT(ELEM k) {...}  
}
```

<pre>NEW_ : INT → ORDBOK _.PUT_ : ORDBOK × ELEM → ORDBOK _.GET_ : ORDBOK × STRING → ELEM</pre>
--

1.b Brukerdefinerte typer

```
PUBLIC CLASS ORDBOK {  
    private Elem[] N;  
    PUBLIC ORDBOK(int k) { N= new Elem[k]; }  
    PUBLIC ELEM GET(STRING e) {...}  
    PUBLIC VOID PUT(ELEM k) {...}  
}
```

<pre>NEW_ : INT → ORDBOK _.PUT_ : ORDBOK × ELEM → ORDBOK _.GET_ : ORDBOK × STRING → ELEM</pre>
--

```
OrdBok ob= NEW ORDBOK(100);  
ob.PUT(new Elem("a", "aaaaaaa"));  
ob.PUT(new Elem("b", "bbbb"));  
ob.PUT(new Elem("c", "cccccc"));  
Elem e= ob.GET("b");
```

1.b Brukerdefinerte typer

```
PUBLIC CLASS ORDBOK {  
    private Elem[] N;  
    PUBLIC ORDBOK(int k) { N= new Elem[k]; }  
    PUBLIC ELEM GET(STRING e) {...}  
    PUBLIC VOID PUT(ELEM k) {...}  
}
```

<pre>NEW_ : INT → ORDBOK _.PUT_ : ORDBOK × ELEM → ORDBOK _.GET_ : ORDBOK × STRING → ELEM</pre>
--

```
OrdBok ob= NEW ORDBOK(100);  
ob.PUT(new Elem("a", "aaaaaaa"));  
ob.PUT(new Elem("b", "bbbb"));  
ob.PUT(new Elem("c", "cccccc"));  
Elem e= ob.GET("b");
```

utvider språket

2. Objekt-orientering og typing

Typing:

- *krever økt programmeringsdisiplin*
- *tillater å oppdage mange feil under kompilering*
- *øker betraktelig pålitelighet av programvare*
- *fører til muligheter for*
 - *gjenbruk og*
 - *modularisering av programvare*

2. Objekt-orientering og typing

Typing:

- *krever økt programmeringsdisiplin*
- *tillater å oppdage mange feil under kompilering*
- *øker betraktelig pålitelighet av programvare*
- *fører til muligheter for*
 - *gjenbruk og*
 - *modularisering av programvare*

En klassedeklarasjon definerer en type, nemlig:

- **et grensesnitt:**
 - et sett med operasjoner som kan utføres på instanser
- **en sort T:** en mengde av instanser med dette grensesnittet
- **(en implementasjon av T og operasjonene)**

2. Objekt-orientering og typing

CLASS ELEM

```
{ PRIVATE String k,d;
  PUBLIC ELEM(String a, String b) { k= a; d= b; }
  PUBLIC STRING KEY() {return k;}
  PUBLIC STRING DATA() {return d;}
}
```

CLASS ORDBOK

```
{ PRIVATE Elem N[]; PRIVATE int max;
  PUBLIC ORDBOK(int m)
    { max= m; N= new Elem[max]; }
  PUBLIC VOID PUT(Elem e)
    { max++; N[max]= e; }
  PUBLIC ELEM GET(String k)
    { for (int i=1; i<=max; i++)
      if (N[i].key()==k) return N[i];
      return null; }
}
```

CLASS ORDBOKLS EXTENDS ORDBOK

```
{ PUBLIC VOID LISTSORTERT()
  { for (int i=1; i<=max; i++)
    for (int j=1; j<=max-i; j++)
      if (N[j].key()>N[j+1].key()) swap(j,j+1)
    for (int i=1; i<=max; i++)
      skriv(N[i]);
  } }
```

2. Objekt-orientering og typing

CLASS ELEM

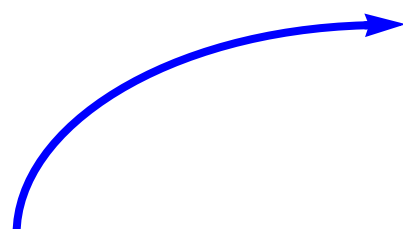
```
{ private String k,d;  
PUBLIC ELEM(String a, String b) { k= a; d= b; }  
PUBLIC STRING KEY() {return k;}  
PUBLIC STRING DATA() {return d;}  
}
```

CLASS ORDBOK

```
{ private Elem N[]; private int max;  
PUBLIC ORDBOK(int m)  
    { max= m; N= new Elem[max]; }  
PUBLIC VOID PUT(Elem e)  
    { max++; N[max]= e; }  
PUBLIC ELEM GET(String k)  
    { for (int i=1; i<=max; i++)  
      if (N[i].key()==k) return N[i];  
      return null;  
    }  
}}
```

CLASS ORDBOKLS EXTENDS ORDBOK

```
{ PUBLIC VOID LISTSORTERT()  
    { for (int i=1; i<=max; i++)  
      for (int j=1; j<=max-i; j++)  
        if (N[j].key()>N[j+1].key()) swap(j,j+1)  
      for (int i=1; i<=max; i++)  
        skriv(N[i]);  
    }  
}
```



3. Utvikling av typer = utvikling av språket

Packages

- *flere relaterte ADTer og klasser vil ofte utgjøre en helhet*
- *moderne språk distribueres sammen med et sett **pakker***
- *disse er **ikke en del** av språket men*
- *er verktøy utviklet **i** språket for **spesifikke formål***

3. Utvikling av typer = utvikling av språket

Packages

- *flere relaterte ADTer og klasser vil ofte utgjøre en helhet*
- *moderne språk distribueres sammen med et sett **pakker***
- *disse er **ikke en del** av språket men*
- *er verktøy utviklet i språket for spesifikke formål*

java.util.Date

public Date()

Allocates a Date object and initializes it so that it represents the time at which it was allocated measured to the nearest millisecond.

public boolean after(Date when) Tests if this *Date* is after the specified *Date*

@Param: *when* - a *Date*.

@Returns: *true* iff this *Date* is after the argument *Date*; *false* otherwise

public int setMinutes(int minutes)

@Param: *minutes* - the value of the minutes to be set for this *Date*

public void getMinutes()

@Returns: the number of minutes past the hour represented by this *Date* ; The value returned is between 0 and 59

3. Utvikling av typer = utvikling av språket

Packages

- flere relaterte ADTer og klasser vil ofte utgjøre en helhet
- moderne språk distribueres sammen med et sett **pakker**
- disse er **ikke en del** av språket men
- er verktøy utviklet i språket for spesifikke formål

java.util.Date

public Date()

Allocates a Date object and initializes it so that it represents the time at which it was allocated measured to the nearest millisecond.

public boolean after(Date when) Tests if this *Date* is after the specified *Date*

@Param: *when* - a *Date*.

@Returns: *true* iff this *Date* is after the argument *Date*; *false* otherwise

public int setMinutes(int minutes)

@Param: *minutes* - the value of the minutes to be set for this *Date*

public void getMinutes()

@Returns: the number of minutes past the hour represented by this *Date* ; The value returned is between 0 and 59

public Object get(Object key)

@Param: *key* - a key in this dictionary.

@Returns: the value to which the *key* is mapped in this dictionary; *null* if none

public Object put(Object key, Object value)

Maps the specified *key* to the specified *value* in this dictionary. Neither the *key* nor the *value* can be *null*. The *value* can be retrieved by calling the *get* method with a *key* that is equal to the original *key*.

@Param: *key* - the hashtable key;
value - the value;

@Returns: the previous *value* to which the *key* was mapped in this dictionary, or *null* if the *key* did not have a previous mapping.

@exception: *NullPointerException* – if the *key* or *value* is *null*.

...

3.a En pakke

- *er en type (eller samling av typer) som utvider språket til et domenespesifikt språk*
- *dokumentasjon av metoder er*
 - *alt som er tilgjengelig utenfra og, faktisk*
 - *alt som bruker trenger for å benytte seg av pakken*
- *bruker vet ikke (og vil ikke vite) hvordan tingene er implementert inne i pakken*

3.a En pakke

- *er en type (eller samling av typer) som utvider språket til et domenespesifikt språk*
- *dokumentasjon av metoder er*
 - *alt som er tilgjengelig utenfra og, faktisk*
 - *alt som bruker trenger for å benytte seg av pakken*
- *bruker vet ikke (og vil ikke vite) hvordan tingene er implementert inne i pakken*

dvs.

Alt som trenges

for å benytte seg av en Type (pakke, klasse...)

er kjennskap til

spesifikasjon av grensesnittmetoder (interface)

3.b Package ... ?

1. Opprett i din hjemmekatalog en underkatalog og kall den, f.eks.

‘Pakker’ – alle dine pakker skal ligge i denne katalogen

2. I Unix, kjør kommando (ikke hvis du bruker ‘bash’-skall!)

```
> setenv CLASSPATH . : /Home/user/kurs/i120 : $HOME/Pakker : /usr/java/  
lib
```

(dette skal helst legges inn i din **.cshrc** fil)

3. Når du lager en ny pakke, f.eks **niceIO**

– opprett katalog **‘niceIO’** i katalogen **‘Pakker’**

– alt som hører til **‘niceIO’**-pakken skal legges i filer i katalogen

‘ \$HOME/Pakker/niceIO/ ’

– enhver fil f.eks. **‘PenOutput.java’** i denne katalogen skal starte med
package niceIO ;

4. For å bruke en pakke, **niceIO**, i en klasse **MinKlasse**, skriv

```
import niceIO.* ;
```

på toppen – før **public class MinKlasse { ...** i filen **‘MinKlasse.java’**

For å bruke bare en spesifikk klasse, **PenOutput**, fra denne pakken:

```
import niceIO.PenOutput ;
```

5. F.eks. **java.awt** er en pakke for grafisk brukergrensesnitt;

– alle dens klasser ligger i katalogen **‘ /usr/java/src/java/awt ’**

– du bruker den/importerer ved å starte din **‘MinKlasse.java’** med

```
import java.awt.* ;
```


4. Abstrakte DataTyper

1. *klasse* = *grensesnitt* + *implementasjon*
Hva *Hvordan*

4. Abstrakte DataTyper

1. *klasse* = *grensesnitt* + *implementasjon*
Hva *Hvordan*

2. *Abstrakt DataType* = *klasse* – *implementasjon*

ADT = *grensesnitt: Hva*

4. Abstrakte DataTyper

1. *klasse* = *grensesnitt* + *implementasjon*
Hva *Hvordan*

2. *Abstrakt DataType* = *klasse* – *implementasjon*

ADT = *grensesnitt: Hva*

(abstract class vs. interface)

3. \emptyset = *grensesnitt* \cap *implementasjon*

CLASS ELEM

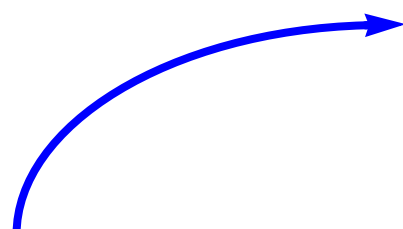
```
{ private String k,d;  
PUBLIC ELEM(String a, String b) { k= a; d= b; }  
PUBLIC STRING KEY() {return k;}  
PUBLIC STRING DATA() {return d;}  
}
```

CLASS ORDBOK

```
{ private Elem N[]; private int max;  
PUBLIC ORDBOK(int m)  
    { max= m; N= new Elem[max]; }  
PUBLIC VOID PUT(Elem e)  
    { max++; N[max]= e; }  
PUBLIC ELEM GET(String k)  
    { for (int i=1; i<=max; i++)  
        if (N[i].key()==k) return N[i];  
      return null;  
    }  
}}
```

CLASS ORDBOKLS EXTENDS ORDBOK

```
{ PUBLIC VOID LISTSORTERT()  
    { for (int i=1; i<=max; i++)  
        for (int j=1; j<=max-i; j++)  
            if (N[j].key()>N[j+1].key()) swap(j,j+1)  
        for (int i=1; i<=max; i++)  
            skriv(N[i]);  
    }  
}
```



4. ADT

INTERFACE ELEM

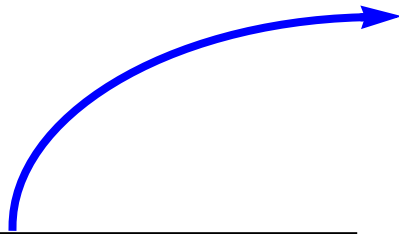
```
{  
  PUBLIC ELEM(String a, String b)  
  PUBLIC STRING KEY();  
  PUBLIC STRING DATA();  
}
```

INTERFACE ORDBOK

```
{  
  PUBLIC ORDBOK(int m)  
  PUBLIC VOID PUT(Elem e);  
  PUBLIC ELEM GET(String k);  
}
```

INTERFACE ORDBOKLS EXTENDS ORDBOK

```
{  
  PUBLIC VOID LISTSORTERT();  
}
```



4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

/ SS=Selection Sort - sorterer input array:**

*** @param - int tab[0...n-1]**

*** @return - sortert tab**

***/**

4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

```
/** SS=Selection Sort - sorterer input array:
 *   @param - int tab[0...n-1]
 *   @return - sortert tab
 *   for ( k = 0,1,2...n-1) {
 *       finn i = indeks til minste elementet m i
 *       tab[k...n-1]; m=tab[i]
 *       bytt elementene ved indeks k og i
 *   }
 */
```


4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

/ SS=Selection Sort - sorterer input array:**

```
*   @param - int tab[0...n-1]
*   @return - sortert tab
*   for ( k = 0,1,2...n-1 ) {
*       finn i = indeks til minste elementet m i
*       tab[k...n-1]; m=tab[i]
*       bytt elementene ved indeks k og i
*   }
*/
```

Dette er nok for å se at det virker:

k=0 2 4 1 3 5

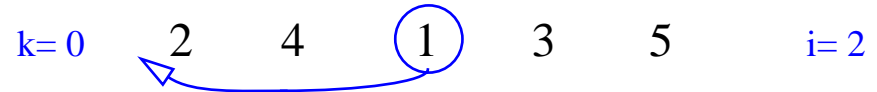
4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{
  int m, i;
  for ( int k = 0; k < tab.length;k++)
  {
    m = tab[k]; i = k;
    for ( int j = k+1; j < tab.length; j++)
      if (tab[j] < m) { m = tab[j]; i = j; }
    tab[i] = tab[k];
    tab[k] = m;
  }
  return tab;
}
```

/ SS=Selection Sort - sorterer input array:**

```
*   @param - int tab[0...n-1]
*   @return - sortert tab
*   for ( k = 0,1,2...n-1) {
*     finn i = indeks til minste elementet m i
*       tab[k...n-1]; m=tab[i]
*     bytt elementene ved indeks k og i
*   }
*/
```

Dette er nok for å se at det virker:



4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

/ SS=Selection Sort - sorterer input array:**

```
*   @param - int tab[0...n-1]
*   @return - sortert tab
*   for ( k = 0,1,2...n-1) {
*       finn i = indeks til minste elementet m i
*           tab[k...n-1]; m=tab[i]
*       bytt elementene ved indeks k og i
*   }
*/
```

Dette er nok for å se at det virker:

k=0	2	4	1	3	5	i=2
k=1	1	4	2	3	5	

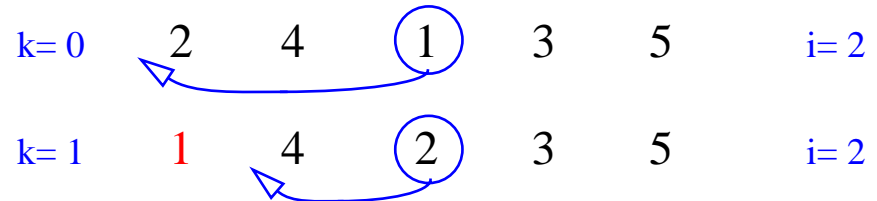
4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

/** SS=Selection Sort - sorterer input array:

```
*   @param - int tab[0...n-1]
*   @return - sortert tab
*   for ( k = 0,1,2...n-1) {
*       finn i = indeks til minste elementet m i
*           tab[k...n-1]; m=tab[i]
*       bytt elementene ved indeks k og i
*   }
*/
```

Dette er nok for å se at det virker:

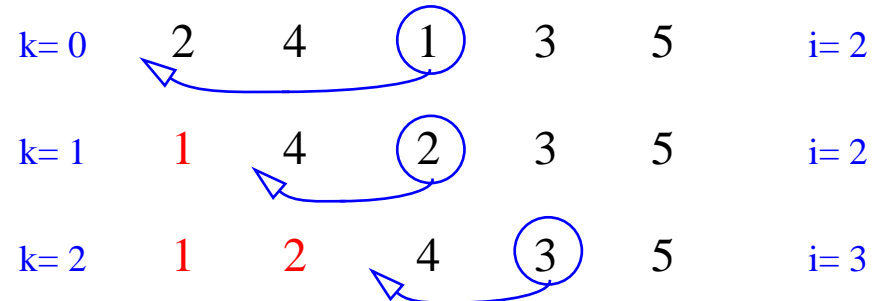


4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

```
/** SS=Selection Sort - sorterer input array:
 *   @param - int tab[0...n-1]
 *   @return - sortert tab
 *   for ( k = 0,1,2...n-1) {
 *       finn i = indeks til minste elementet m i
 *           tab[k...n-1]; m=tab[i]
 *       bytt elementene ved indeks k og i
 *   }
 */
```

Det er nok for å se at det virker:

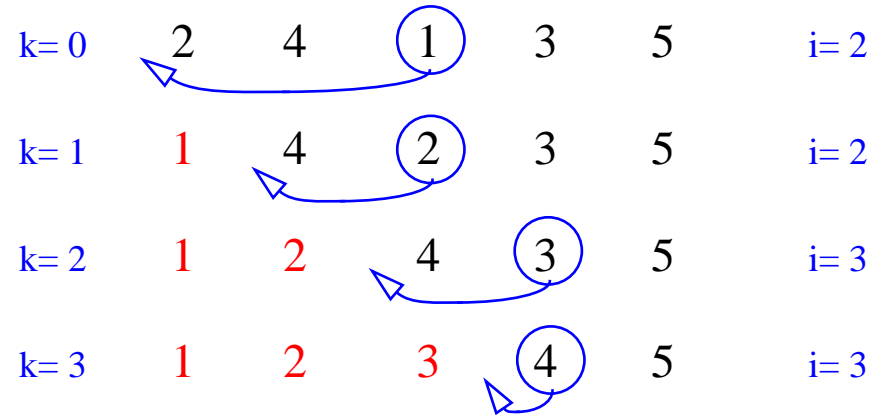


4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

```
/** SS=Selection Sort - sorterer input array:
 *   @param - int tab[0...n-1]
 *   @return - sortert tab
 *   for ( k = 0,1,2...n-1) {
 *       finn i = indeks til minste elementet m i
 *           tab[k...n-1]; m=tab[i]
 *       bytt elementene ved indeks k og i
 *   }
 */
```

Dette er nok for å se at det virker:

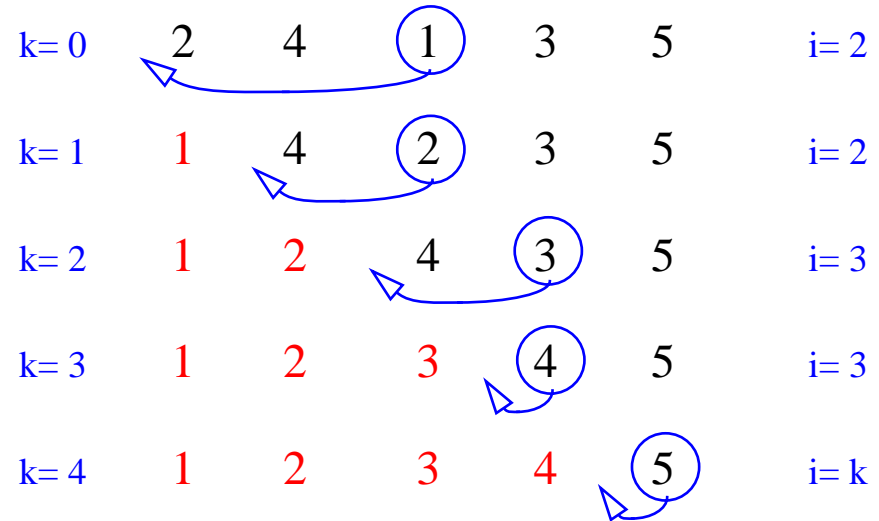


4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{   int m, i;
    for ( int k = 0; k < tab.length;k++)
    {   m = tab[k]; i = k;
        for ( int j = k+1; j < tab.length; j++)
            if (tab[j] < m) { m = tab[j]; i = j; }
        tab[i] = tab[k];
        tab[k] = m;
    }
    return tab;
}
```

```
/** SS=Selection Sort - sorterer input array:
 *   @param - int tab[0...n-1]
 *   @return - sortert tab
 *   for ( k = 0,1,2...n-1) {
 *       finn i = indeks til minste elementet m i
 *           tab[k...n-1]; m=tab[i]
 *       bytt elementene ved indeks k og i
 *   }
 */
```

Dette er nok for å se at det virker:

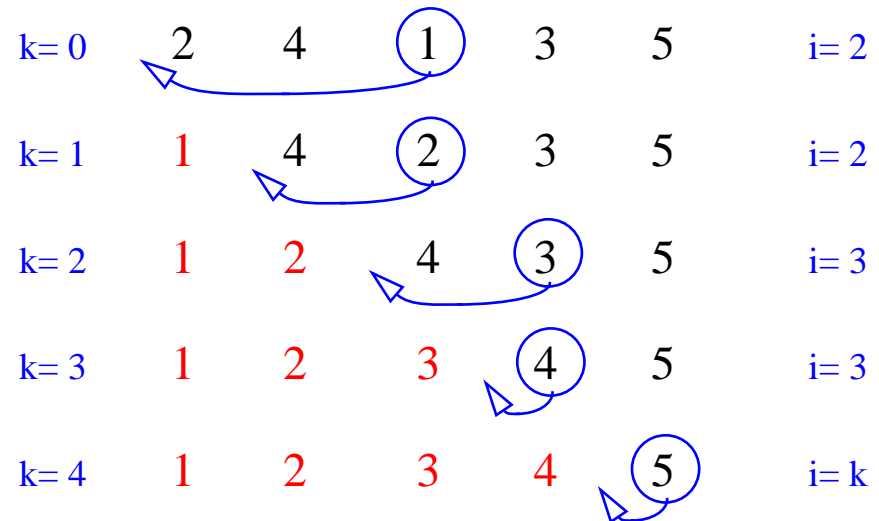


4.a Pseudokode ... dokumentasjon = forklaring

```
public int[] SS( int[] tab)
{
  int m, i;
  for ( int k = 0; k < tab.length;k++)
  {
    m = tab[k]; i = k;
    for ( int j = k+1; j < tab.length; j++)
      if (tab[j] < m) { m = tab[j]; i = j; }
    tab[i] = tab[k];
    tab[k] = m;
  }
  return tab;
}
```

```
/** SS=Selection Sort - sorterer input array:
 *   @param - int tab[0...n-1]
 *   @return - sortert tab
 *   for ( k = 0,1,2...n-1) {
 *     finn i = indeks til minste elementet m i
 *       tab[k...n-1]; m=tab[i]
 *     bytt elementene ved indeks k og i
 *   }
 */
```

Dette er nok for å se at det virker:



Løkkeinvariant:

Etter **k-te** iterasjon er elementene **0...k** riktig plassert

4.b ADT = programmering med egenskaper (grensesnitt)

```
/** SS - sorterer input array:  
 *   @param - int tab[0...n]  
 *   @return - sortert tab  
 *   for ( k = 0,1,2...n-1) {  
 *       i = indeksen til minste elementet m i  
 *       tab[k...n]; m=tab[i]  
 *       bytt elementene ved indeks k og i  
 *   }  
 */
```

```
public void SS(int[] N) { // int <  
    int m, i;  
    for (int k=0; k < N.length; k++) {  
        m = N[k]; i = k;  
        for (int j=k+1; j < N.length; j++)  
            if ( N[j] < m ) { m = N[j]; i= j; }  
        N[i]= N[k]; N[k]= m;  
    } }  
}
```

4.b ADT = programmering med egenskaper (grensesnitt)

```
/** SS - sorterer input array:
 *   @param - int tab[0...n]
 *   @return - sortert tab
 *   for ( k = 0,1,2...n-1) {
 *       i = indeksen til minste elementet m i
 *       tab[k...n]; m=tab[i]
 *       bytt elementene ved indeks k og i
 *   }
 */
```

```
public void SS(int[] N) { // int <
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[j] < m ) { m = N[j]; i= j; }
        N[i]= N[k]; N[k]= m;
    } }
```

```
public void SS(int[] N) { // int >
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[j] > m ) { m = N[j]; i= j; }
        N[i]= N[k]; N[k]= m;
    } }
```

4.b ADT = programmering med egenskaper (grensesnitt)

```
/** SS - sorterer input array:
 *   @param - int tab[0...n]
 *   @return - sortert tab
 *   for ( k = 0,1,2...n-1) {
 *       i = indeksen til minste elementet m i
 *       tab[k...n]; m=tab[i]
 *       bytt elementene ved indeks k og i
 *   }
 */
```

```
public void SS(int[] N) { // int <
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[j] < m ) { m = N[j]; i=j; }
        N[i]= N[k]; N[k]= m;
    } }
```

```
public void SS(int[] N) { // int >
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[j] > m ) { m = N[j]; i=j; }
        N[i]= N[k]; N[k]= m;
    } }
```

```
public void SS(Date[] N) { // Date <
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if (N[j] . before(m)) { m = N[j]; i=j; }
        N[i]= N[k]; N[k]= m;
    } }
```

4.c Comparator ADT

public interface IntComparator

```
{    /** @return true iff a < b; false otherwise */
boolean isLessThan (int a, int b)

    /** @return true iff a > b; false otherwise */
boolean isGreaterThan (int a, int b)

    /** @return true iff a = b; false otherwise */
boolean isEqualTo (int a, int b)

    /** @return true iff a <= b; false otherwise */
boolean isLessThanOrEqualTo (int a, int b)

    /** @return true iff a >= b; false otherwise */
boolean isGreaterThanOrEqualTo (int a, int b)
}
```

```
public void SS(int[] N) {    // int >
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[ j ] > m ) { m = N[j]; i= j; }
        N[i]= N[k]; N[k]= m;
    }
}
```

```
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[ j ] > m ) { m = N[j]; i= j; }
        N[i]= N[k]; N[k]= m;
    } }
```

4.c Comparator ADT

```
public interface IntComparator
```

```
{    /** @return true iff a < b; false otherwise */
```

```
boolean isLessThan (int a, int b)
```

```
    /** @return true iff a > b; false otherwise */
```

```
boolean isGreaterThan (int a, int b)
```

```
    /** @return true iff a = b; false otherwise */
```

```
boolean isEqualTo (int a, int b)
```

```
    /** @return true iff a <= b; false otherwise */
```

```
boolean isLessThanOrEqualTo (int a, int b)
```

```
    /** @return true iff a >= b; false otherwise */
```

```
boolean isGreaterThanOrEqualTo (int a, int b)
```

```
}
```

```
public void SS(int[] N) {    // int >
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[ j ] > m ) { m = N[j]; i= j; }
        N[i]= N[k]; N[k]= m;
    }
}
```

```
        m = N[k]; i = k;
        for (int j=k+1; j < N.length; j++)
            if ( N[ j ] > m ) { m = N[j]; i= j; }
        N[i]= N[k]; N[k]= m;
    } }
}
```

```
public void GSS( int[] N, IntComparator CP ) {
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; ind = k;
        for (int j=k+1; j < N.length; j++)
            if ( CP.isLessThan (N[ j ], m) ) { m = N[j]; i= j; }
        N[i]= N[k]; N[k]= m;
    } }
}
```

bruker ADT IntComparator – dvs. vilkårlig implementasjon !!!

```
int[] N;
```

```
class IntNormalComp implements IntComparator
```

```
public boolean isLessThan (int a, int b)  
    { return a < b; }
```

```
public boolean isGreaterThan (int a, int b)  
    { return a > b; }
```

```
public boolean isEqualTo (int a, int b)  
    { return a == b; }
```

```
public boolean isLessThanOrEqualTo (int a, int b)  
    { return a <= b; }
```

```
public boolean isGreaterThanOrEqualTo (int a, int b)  
    { return a >= b; }
```

...

```
GSS(N, new IntNormalComp()); -- sorterer N i stigende rekkefølge
```

```
int[] N;
```

```
class IntReversComp implements IntComparator  
public boolean isLessThan (int a, int b)  
    { return a > b; }  
public boolean isGreaterThan (int a, int b)
```

```
class IntNormalComp implements IntComparator
```

```
public boolean isLessThan (int a, int b)  
    { return a < b; }  
public boolean isGreaterThan (int a, int b)  
    { return a > b; }  
public boolean isEqualTo (int a, int b)  
    { return a == b; }  
public boolean isLessThanOrEqualTo (int a, int b)  
    { return a <= b; }  
public boolean isGreaterThanOrEqualTo (int a, int b)  
    { return a >= b; }
```

```
, int b)
```

```
qualTo (int a, int b)
```

```
OrEqualTo (int a, int b)
```

...

GSS(N, new **IntNormalComp**()); -- sorterer N i stigende rekkefølge

GSS(N, new **IntReversComp**()); -- sorterer N i nedstigende rekkefølge

...

4.c Comparator ADT

```
public interface IntComparator
{
    /** @return true iff a < b; false otherwise */
    boolean isLessThan (int a, int b)

    /** @return true iff a > b; false otherwise */
    boolean isGreaterThan (int a, int b)

    /** @return true iff a = b; false otherwise */
    boolean isEqualTo (int a, int b)

    /** @return true iff a <= b; false otherwise */
    boolean isLessThanOrEqualTo (int a, int b)

    /** @return true iff a >= b; false otherwise */
    boolean isGreaterThanOrEqualTo (int a, int b)
}
```

```
public void SS(int[] N) { // int >
```

```
int m, i;
for (int k=0; k < N.length; k++) {
    m = N[k]; ind = k;
    for (int j=k+1; j < N.length; j++)
        if (N[j] > m) { m = N[j]; i = j; }
    N[i] = N[k]; N[k] = m;
}
```

```
    m = N[k]; i = k;
    for (int j=k+1; j < N.length; j++)
        if ( N[ j ] > m ) { m = N[j]; i = j; }
    N[i] = N[k]; N[k] = m;
} }
```

```
public void GSS( int[] N, IntComparator CP ) {
    int m, i;
    for (int k=0; k < N.length; k++) {
        m = N[k]; ind = k;
        for (int j=k+1; j < N.length; j++)
            if ( CP.isLessThan (N[ j ], m) ) { m = N[j]; i = j; }
        N[i] = N[k]; N[k] = m;
    } }
```

bruker **ADT IntComparator** – dvs. vilkårlig implementasjon !!!

4.c Comparator ADT

```
public interface Comparator
```

```
{    /** @return true iff a < b; false otherwise */
```

```
boolean isLessThan (Object a, Object b)
```

```
    /** @return true iff a > b; false otherwise */
```

```
boolean isGreaterThan (Object a, Object b)
```

```
    /** @return true iff a = b; false otherwise */
```

```
boolean isEqualTo (Object a, Object b)
```

```
    /** @return true iff a <= b; false otherwise */
```

```
boolean isLessThanOrEqualTo (Object a, Object b)
```

```
    /** @return true iff a >= b; false otherwise */
```

```
boolean isGreaterThanOrEqualTo (Object a, Object b)
```

```
}
```

```
public void SS(int[] N) {    // int >
```

```
    int m, i;
```

```
    for (int k=0; k < N.length; k++) {
```

```
        m = N[k]; ind = k;
```

```
        for (int j=k+1; j < N.length; j++) {
```

```
            m = N[k]; ind = k;
```

```
            for (int j=k+1; j < N.length; j++)
```

```
                if (N[j].before(m)) { m = N[j]; i = j; }
```

```
            N[i] = N[k]; N[k] = m;
```

```
        }  
    }  
}
```

```
        m = N[k]; i = k;  
        for (int j=k+1; j < N.length; j++)  
            if (N[j].before(m)) { m = N[j]; i = j; }  
        N[i] = N[k]; N[k] = m;  
    }  
}
```

```
public void GSS(Object[] N, Comparator CP) {
```

```
    int i; Object m;
```

```
    for (int k=0; k < N.length; k++) {
```

```
        m = N[k]; ind = k;
```

```
        for (int j=k+1; j < N.length; j++)
```

```
            if (CP.isLessThan(N[j], m)) { m = N[j]; i = j; }
```

```
        N[i] = N[k]; N[k] = m;
```

```
    }  
}
```

bruker *ADT Comparator* – dvs. vilkårlig implementasjon !!!

```
Integer[] N;  
Date[] D;
```

```
class IntReversComp implements Comparator  
public boolean isLessThan (Object a, Object b)  
    { return ((Integer)a).intValue() > ((Integer)b).intValue(); }  
public boolean isGreaterThan (Object a, Object b)
```

```
class IntNormalComp implements Comparator  
public boolean isLessThan (Object a, Object b)  
    { return ((Integer)a).intValue() < ((Integer)b).intValue(); }  
public boolean isGreaterThan (Object a, Object b)  
    { return ((Integer)a).intValue() > ((Integer)b).intValue(); }  
public boolean isEqual (Object a, Object b)  
    { return ((Integer)a).intValue() == ((Integer)b).intValue(); }  
public boolean isLessThanOrEqual (Object a, Object b)  
    { return ((Integer)a).intValue() <= ((Integer)b).intValue(); }  
public boolean isGreaterThanOrEqual (Object a, Object b)  
    { return ((Integer)a).intValue() >= ((Integer)b).intValue(); }
```

```
    { return ((Integer)a).intValue() < ((Integer)b).intValue(); }  
    { return ((Integer)a).intValue() > ((Integer)b).intValue(); }
```

```
isEqual (Object a, Object b)
```

```
isLessThanOrEqual (Object a, Object b)
```

```
class DateNormalComp implements Comparator  
public boolean isLessThan (Object a, Object b)  
    { Date ad= (Date)a; Date bd= (Date)b;  
      if (ad.getYear() < bd.getYear()) return true;  
      else if (ad.getYear() > bd.getYear()) return false;  
      else if (ad.getDay() < bd.getDay()) return true;  
      else if (ad.getDay() > bd.getDay()) return false;  
      else if (ad.getHour() < bd.getHour()) return true;  
      else if ... }  
public boolean isGreaterThan (Object a, Object b) { ... }
```

```
...  
GSS(N, new IntNormalComp());  
GSS(N, new IntReversComp());  
GSS(D, new DateNormalComp());  
...
```

4.d Modularisering av implementasjon

1. GSS benytter kun abstrakte egenskaper – grensesnitt metoder – av *Comparator* (og *Object[]*)
2. den er uavhengig av hvordan disse er implementert
3. og kan akseptere enhver ny (endret/forbedret) implementasjon så lenge det er en implementasjon av respektivt grensesnitt

4.d Modularisering av implementasjon

1. GSS benytter kun abstrakte egenskaper – grensesnitt metoder – av **Comparator** (og **Object[]**)
2. den er uavhengig av hvordan disse er implementert
3. og kan akseptere enhver ny (endret/forbedret) implementasjon så lenge det er en implementasjon av respektivt grensesnitt

1. overalt i mine programmer der jeg har behov for sammenlikning skal jeg bruke **Comparator ADT**

2. alle mine (A)DTer vil benytte kun **Comparator** (i grensesnitt) og ikke noen implementasjon av denne

3. og alle mine **implementasjoner** skal foreta sammenlikninger kun gjennom **Comparator** grensesnittet

4.d Modularisering av implementasjon

1. GSS benytter kun abstrakte egenskaper – grensesnitt metoder – av **Comparator** (og **Object[]**)
2. den er uavhengig av hvordan disse er implementert
3. og kan akseptere enhver ny (endret/forbedret) implementasjon så lenge det er en implementasjon av respektivt grensesnitt

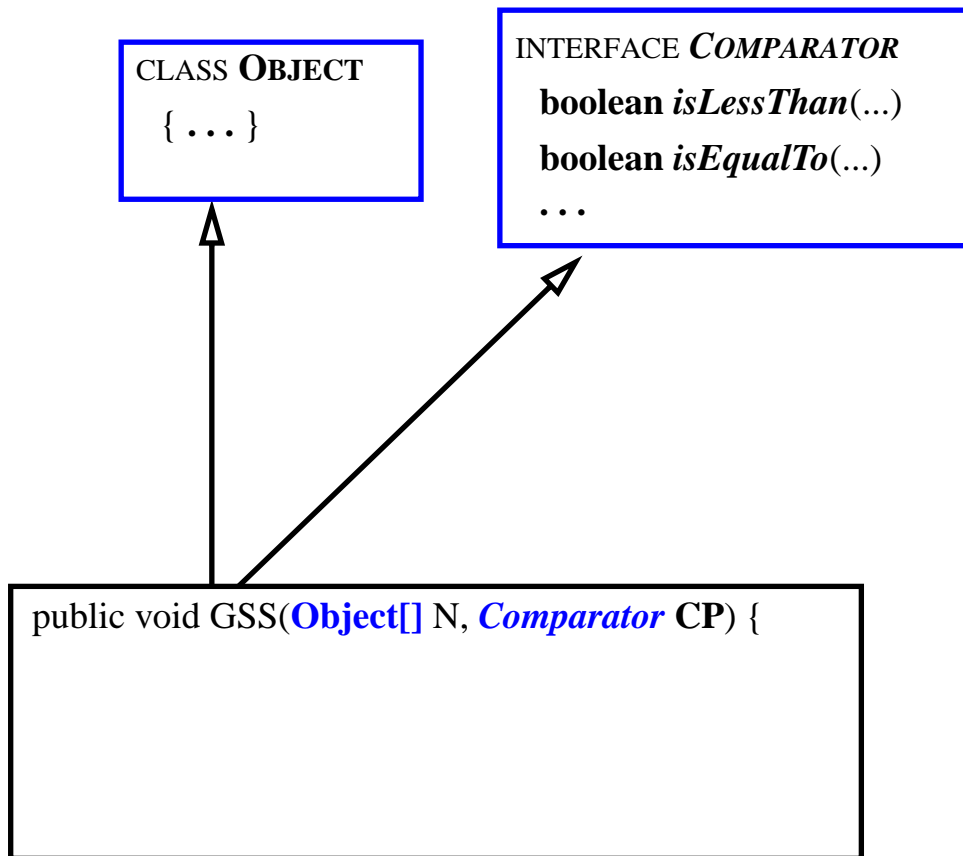
```
CLASS OBJECT  
{ ... }
```

```
INTERFACE COMPARATOR  
  boolean isLessThan(...)  
  boolean isEqualTo(...)  
  ...
```

1. overalt i mine programmer der jeg har behov for sammenlikning skal jeg bruke **Comparator ADT**
2. alle mine (A)DTer vil benytte kun **Comparator** (i grensesnitt) og ikke noen implementasjon av denne
3. og alle mine **implementasjoner** skal foreta sammenlikninger kun gjennom **Comparator** grensesnittet

4.d Modularisering av implementasjon

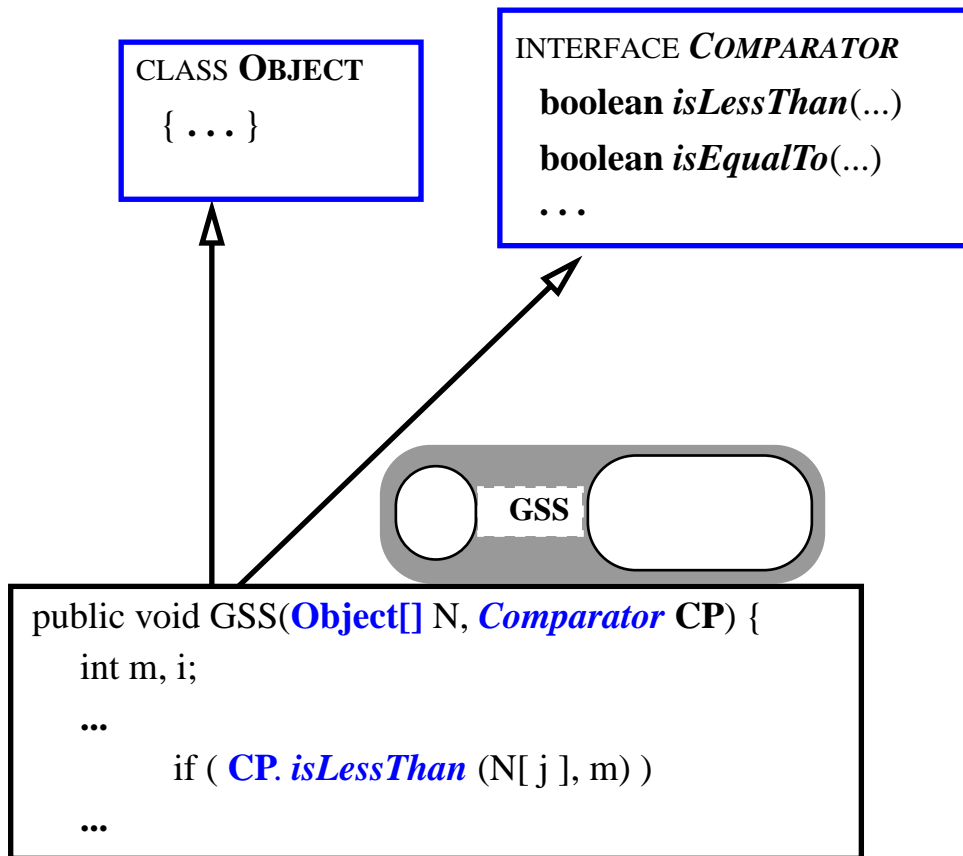
1. GSS benytter kun abstrakte egenskaper – grensesnitt metoder – av **Comparator** (og **Object[]**)
2. den er uavhengig av hvordan disse er implementert
3. og kan akseptere enhver ny (endret/forbedret) implementasjon så lenge det er en implementasjon av respektivt grensesnitt



1. overalt i mine programmer der jeg har behov for sammenlikning skal jeg bruke **Comparator ADT**
2. alle mine (A)DTer vil benytte kun **Comparator** (i grensesnitt) og ikke noen implementasjon av denne
3. og alle mine **implementasjoner** skal foreta sammenlikninger kun gjennom **Comparator** grensesnittet

4.d Modularisering av implementasjon

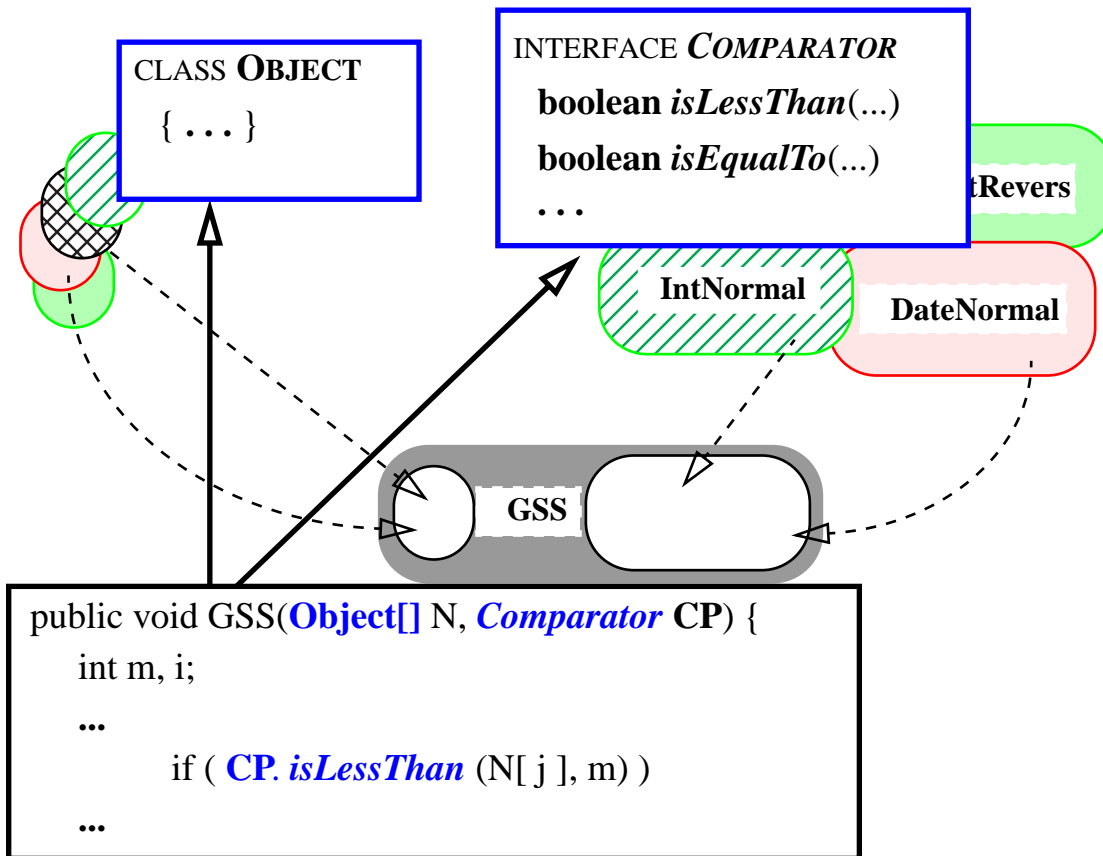
1. GSS benytter kun abstrakte egenskaper – grensesnitt metoder – av **Comparator** (og **Object[]**)
2. den er uavhengig av hvordan disse er implementert
3. og kan akseptere enhver ny (endret/forbedret) implementasjon så lenge det er en implementasjon av respektivt grensesnitt



1. overalt i mine programmer der jeg har behov for sammenlikning skal jeg bruke **Comparator ADT**
2. alle mine (A)DTer vil benytte kun **Comparator** (i grensesnitt) og ikke noen implementasjon av denne
3. og alle mine **implementasjoner** skal foreta sammenlikninger kun gjennom **Comparator** grensesnittet

4.d Modularisering av implementasjon

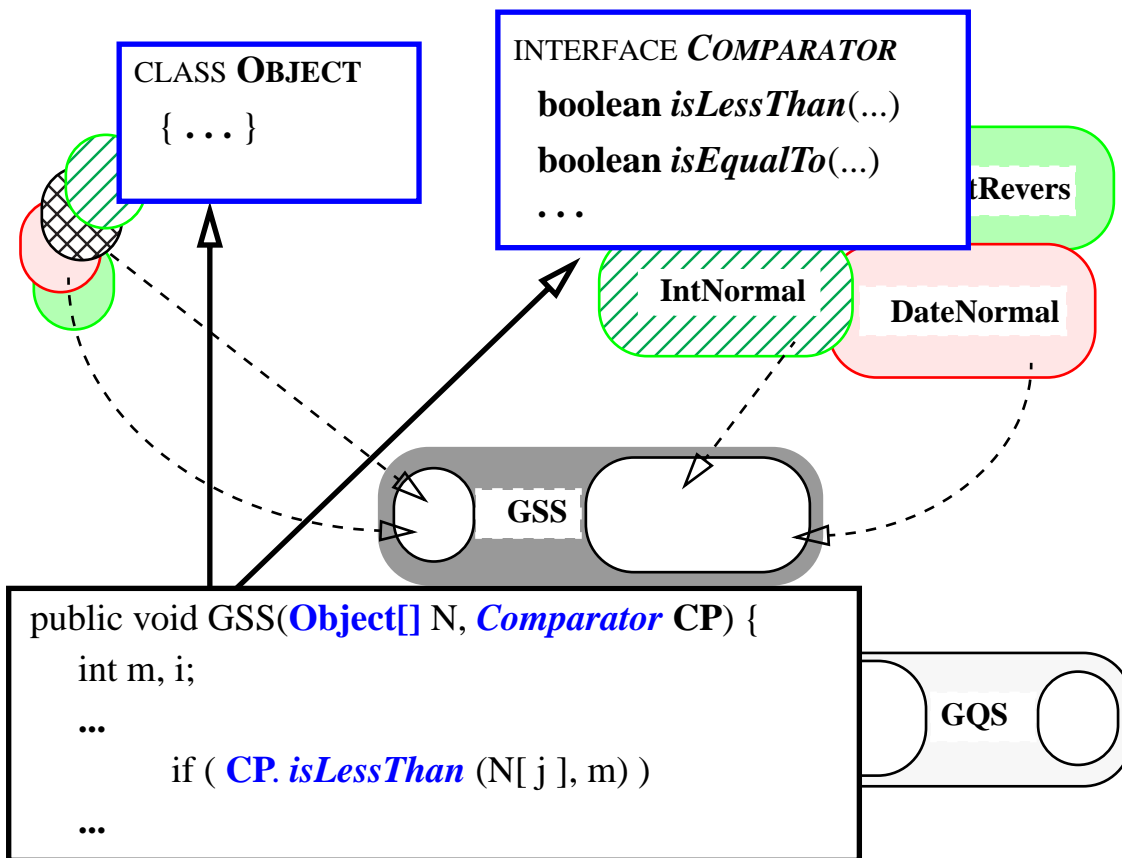
1. GSS benytter kun abstrakte egenskaper – grensesnitt metoder – av **Comparator** (og **Object[]**)
2. den er uavhengig av hvordan disse er implementert
3. og kan akseptere enhver ny (endret/forbedret) implementasjon så lenge det er en implementasjon av respektivt grensesnitt



1. overalt i mine programmer der jeg har behov for sammenlikning skal jeg bruke **Comparator ADT**
2. alle mine (A)DTer vil benytte kun **Comparator** (i grensesnitt) og ikke noen implementasjon av denne
3. og alle mine **implementasjoner** skal foreta sammenlikninger kun gjennom **Comparator** grensesnittet

4.d Modularisering av implementasjon

1. GSS benytter kun abstrakte egenskaper – grensesnitt metoder – av **Comparator** (og **Object[]**)
2. den er uavhengig av hvordan disse er implementert
3. og kan akseptere enhver ny (endret/forbedret) implementasjon så lenge det er en implementasjon av respektivt grensesnitt



1. overalt i mine programmer der jeg har behov for sammenlikning skal jeg bruke **Comparator ADT**
2. alle mine (A)DTer vil benytte kun **Comparator** (i grensesnitt) og ikke noen implementasjon av denne
3. og alle mine **implementasjoner** skal foreta sammenlikninger kun gjennom **Comparator** grensesnittet

The “Millennium Bug”

```
MinSS store= new MinSS();
store.ins(new prod(...)) ;
....
public void remOverdue() {
    boolean done= false;
    aa = getYear();
    mm = getMonth();
    dd = getDay();
    prod p= (prod)store.getMin();
    while (! done) {
        if ( p.a < aa || (p.a==aa && p.m < m)
            || (p.a==aa && p.m==m && p.d < dd) )
            { store.remMin();
              p= (prod)store.getMin();
            } else done = true ;
    }
}
```

The “Millennium Bug”

```
MinSS store= new MinSS();
store.ins(new prod(...)) ;
....
public void remOverdue() {
    boolean done= false;
    aa = getYear();
    mm = getMonth();
    dd = getDay();
    prod p= (prod)store.getMin();
    while (! done) {
        if ( p.a < aa || (p.a==aa && p.m < m)
            || (p.a==aa && p.m==m && p.d < dd) )
            { store.remMin();
              p= (prod)store.getMin();
            } else done = true ;
    }
}
```

The “Millennium Bug”

```
MinSS store= new MinSS();
store.ins(new prod(...)) ;
....
public void remOverdue() {
    boolean done= false;
    aa = getYear();
    mm = getMonth();
    dd = getDay();
    prod p= (prod)store.getMin();
    while (! done) {
        if ( p.a < aa || (p.a==aa && p.m < m)
            || (p.a==aa && p.m==m && p.d < dd) )
            { store.remMin();
              p= (prod)store.getMin();
            } else done = true ;
    }
}
```

```
class prod {
    public int d, m, a;
    public boolean lt(prod p) {
        if ( a < p.a || (a==p.a && m < p.m)
            || (a==p.a && m==p.m && d < p.d) )
            return true;
        else return false; }
}
```

The “Millennium Bug”

```
MinSS store= new MinSS();
store.ins(new prod(...)) ;
```

....

```
public void remOverdue() {
```

```
    boolean done= false;
```

```
    aa = getYear();
```

```
    mm = getMonth();
```

```
    dd = getDay();
```

```
    prod p= (prod)store.getMin();
```

```
    while (! done) {
```

```
        if ( p.a < aa || (p.a==aa && p.m < m)
```

```
            || (p.a==aa && p.m==m && p.d < dd) )
```

```
            { store.remMin();
```

```
              p= (prod)store.getMin();
```

```
            } else done = true ;
```

```
class prod {
    public int d, m, a;
    public boolean lt(prod p) {
        if ( a < p.a || (a==p.a && m < p.m)
            || (a==p.a && m==p.m && d < p.d) )
            return true;
        else return false; }
}
```

```
n && d < p.d) )
```

*60% av kostnader går ikke til programutvikling
men til vedlikehold !!!*

ADT programmering

```
import IDate, IMin, Comparator;  
MinSS store;
```

```
store.ins(new prod(...)) ;
```

```
....
```

```
public void remOverdue() {  
    boolean done = false;  
    Date today = getDate();  
    prod p= (prod)store.getMin();  
    while (! done) {  
        if (p.dt ?<? today) {  
            store.remMin();  
            p= (prod)store.getMin();  
        }  
        else done = true;  
    }  
}
```

```
...
```

```
class prod {  
    public Date dt;  
    ... }  
}
```

```
public class Date {  
    int d() ...  
    int m() ...  
    int a() ...  
    setA(int aa) ...  
    ... }  
}
```

ADT programmering

```
import IDate, IMin, Comparator;  
MinSS store;
```

```
store.ins(new prod(...)) ;
```

```
....
```

```
public void remOverdue() {  
    boolean done = false;  
    IDate today = getDate();  
    prod p= (prod)store.getMin();  
    while (! done) {  
        if (p.dt ?<? today) {  
            store.remMin();  
            p= (prod)store.getMin();  
        }  
        else done = true;  
    }  
}
```

```
...
```

```
class prod {  
    public IDate dt;  
    ... }  
}
```

```
public interface IDate {  
    int d();  
    int m();  
    int a();  
    setA(int aa);  
    ... }  
}
```

ADT programmering

```
import IDate, IMin, Comparator;
MinSS store;
Comparator dcp ;
store = new MinSS(dcp);
store.ins(new prod(...)) ;
....
public void remOverdue() {
    boolean done = false;
    IDate today = getDate();
    prod p= (prod)store.getMin();
    while (! done) {
        if (dcp.isLessThan(p.dt, today)) {
            store.remMin();
            p= (prod)store.getMin();
        }
        else done = true;
    }
    ...
}
```

```
class prod {
    public IDate dt;
    ... }
}
```

```
public interface IDate {
    int d();
    int m();
    int a();
    setA(int aa);
    ... }
}
```


ADT programmering

```
import IDate, IMin, Comparator;
MinSS store;
Comparator dcp = new DatoCp();
store = new MinSS(dcp);
store.ins(new prod(...));
```

....

```
public void remOverdue() {
    boolean done = false;
    IDate today = getDate();
    prod p= (prod)store.getMin();
    while (! done) {
        if (dcp.isLessThan(p.dt,today)) {
            store.remMin();
            p= (prod)store.getMin();
        }
        else done = true;
    }
}
```

...

```
class prod {
    public IDate dt;
    ... }
```

```
public interface IDate {
    int d();
    int m();
    int a();
    setA(int aa);
    ... }
```

```
public class DatoCp implements Comparator {
```

```
//sammlikner IDate-objekter
```

```
public boolean isLessThan(Object d1,d2) {
```

```
    IDate e= (IDate)d1; IDate f= (IDate)d2;
```

```
    if (e.a() < f.a() || (e.a() == f.a() && e.m() < f.m()))
```

```
        ||(e.a() == f.a() && e.m() == f.m() && e.d() < f.d()) )
```

```
        return true;
```

```
    else return false;
```

```
    }
```

```
public boolean isEqualTo(Object d1,d2) {
```

```
    ...
```

ADT programmering

```
import IDate, IMin, Comparator;
MinSS store;
Comparator dcp = new DatoCp();
store = new MinSS(dcp);
store.ins(new prod(...));
```

....

```
public void remOverdue() {
    boolean done = false;
    IDate today = getDate();
    prod p= (prod)store.getMin();
    while (! done) {
        if (dcp.isLessThan(p.dt,today)) {
            store.remMin();
            p= (prod)store.getMin();
        }
        else done = true;
    }
}
```

...

```
class prod {
    public IDate dt;
    ... }
```

```
public interface IDate {
    int d();
    int m();
    int a();
    setA(int aa);
    ... }
```

```
public class DatoCp implements Comparator {
    //samlikner IDate-objekter
    public boolean isLessThan(Object d1,d2) {
        IDate e= (IDate)d1; IDate f= (IDate)d2;
        if (e.a() < 100) e.setA(1900+e.a());
        if (f.a() < 100) f.setA(1900+f.a());
        if (e.a() < f.a() || (e.a() == f.a() && e.m() < f.m())
            ||(e.a() == f.a() && e.m() == f.m() && e.d() < f.d() )
            return true;
        else return false;
    }
    public boolean isEqualTo(Object d1,d2) {
        ...
    }
}
```

type tekst/program	boksider	tekstlinjer
i-120	10	400
enkel kompilator	300	12 000
lønnssystem, industri	600	24 000
utlånssystem, forsikring	7 500	300 000
abonnentsystem, avis	10 650	425 000
logistikksystem, oljeplattform	18 750	750 000

Oppsummering: Programutvikling

Bruk av ADTer gir modulære programmer med økt gjenbrukbarhet og forenklet vedlikehold

*Programmerer (bruker) vil ikke vite hvordan moduler/pakker han benytter er implementert
– man er interessert kun i hvilken funksjonalitet de tilbyr, dvs. kun i grensesnittet*

ADT = grensesnittmetoder med tilstrekkelig dokumentasjon (interface + Javadoc)

klasse = implementasjon av en ADT med et valg av datastruktur og passende algoritmer

Oppsummering: Programutvikling

Bruk av ADTer gir modulære programmer med økt gjenbrukbarhet og forenklet vedlikehold

*Programmerer (bruker) vil ikke vite hvordan moduler/pakker han benytter er implementert
– man er interessert kun i hvilken funksjonalitet de tilbyr, dvs. kun i grensesnittet*

ADT = grensesnittmetoder med tilstrekkelig dokumentasjon (interface + Javadoc)

klasse = implementasjon av en ADT med et valg av datastruktur og passende algoritmer

1. Finn ut hvilke moduler du trenger :

- hvordan de skal samarbeide og
- gjennom hvilke grensesnitt
 - noen vil være tilgjengelig fra eksisterende bibliotek/pakker
 - andre vil du måtte lage selv

For hver enkelt modul

2. velg en data struktur og
3. design nødvendige algoritmer

(Her skal du tenke effektivitet)

4. Implementer alle moduler og sett de sammen iflg. 1.

Oppsummering: Programutvikling

Bruk av ADTer gir modulære programmer med økt gjenbrukbarhet og forenklet vedlikehold

*Programmerer (bruker) vil ikke vite hvordan moduler/pakker han benytter er implementert
– man er interessert kun i hvilken funksjonalitet de tilbyr, dvs. kun i grensesnittet*

ADT = grensesnittmetoder med tilstrekkelig dokumentasjon (interface + Javadoc)

klasse = implementasjon av en ADT med et valg av datastruktur og passende algoritmer

1. Finn ut hvilke moduler du trenger :

- hvordan de skal samarbeide og
- gjennom hvilke grensesnitt
 - noen vil være tilgjengelig fra eksisterende bibliotek/pakker
 - andre vil du måtte lage selv

For hver enkelt modul

2. velg en data struktur og
3. design nødvendige algoritmer

(Her skal du tenke effektivitet)

4. Implementer alle moduler og sett de sammen iflg. 1.

Metodologi og teknikker

- 1 ADT programmering vs. i-110 interface, OO
- 2 mer ADT og OO
- 3 rekursjon
- 4 algoritmeanalyse, implementasjon

Deretter mer konkret

- algoritmer og datastrukturer