I-120 (H-2003)

Foreleser:
Michal Walicki
e-post: michal@ii.uib.no
kontor: HiB, 4145

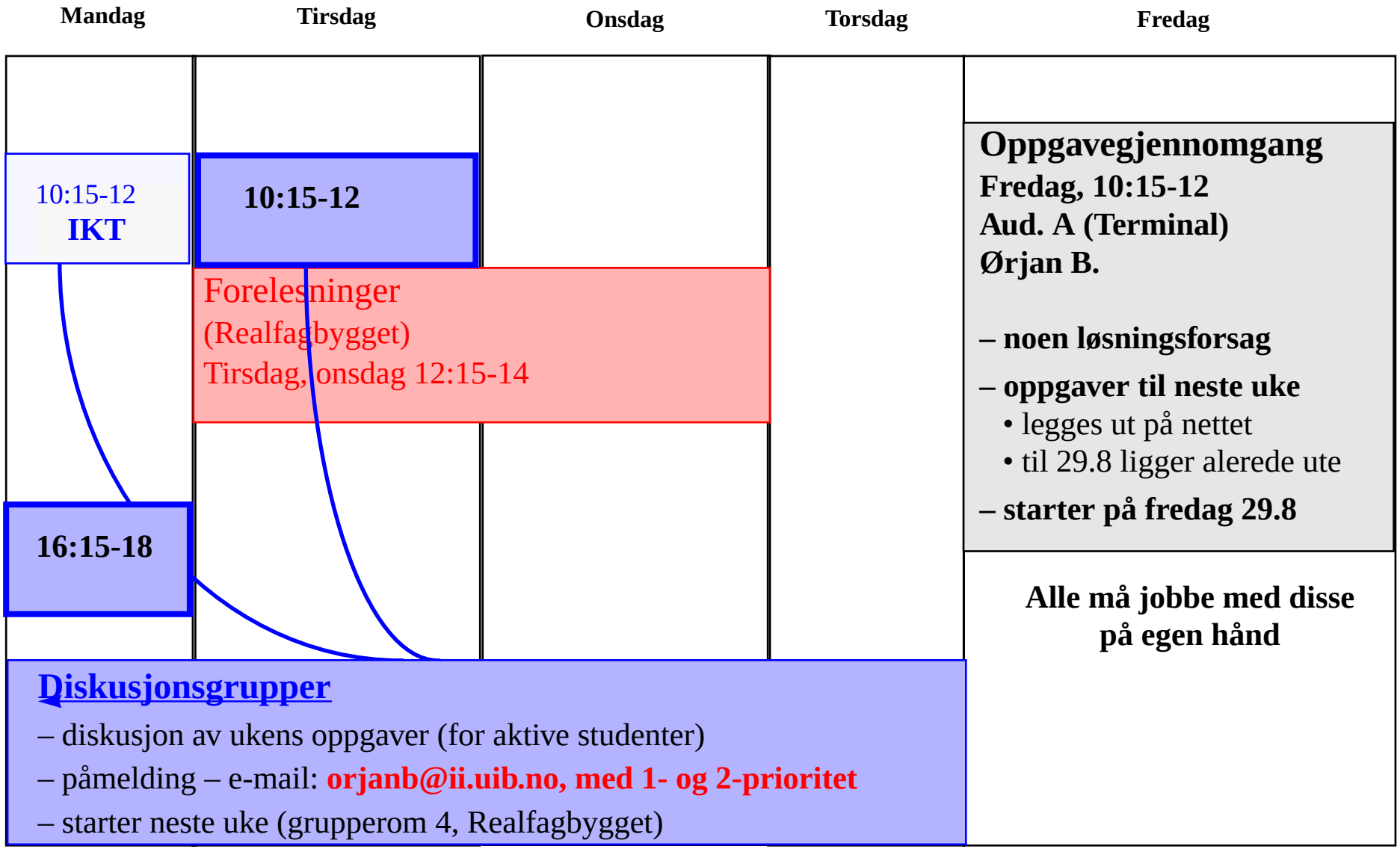
Gruppeansvarlig:
Ørjan Bergman
epost: orjanb@ii.uib.no

websider: [studentportal.uib.no] --> I-120 --> Meldingar (--> kursside)
foreløpig: http:// www.ii.uib.no / ~orjanb / i120 / i120.htm
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
10:15-12
IKT
10:15-12
Forelesninger
(Realfagbygget)
Tirsdag, onsdag 12:15-14
16:15-18
Diskusjonsgrupper
– 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)
Oppgavegjennomgang
Fredag, 10:15-12
Aud. A (Terminal)
Ørjan B.
– noen løsningsforsag
– oppgaver til neste uke
• legges ut på nettet
• til 29.8 ligger alerede ute
– starter på fredag 29.8
Alle må jobbe med disse
på egen hånd

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 (**Gjennomgåes 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, *ADAPTER***
KAP. 5			**SEKVENS; RANK, *POSITION*; *ENUMERATION-ITERATOR***
KAP. 6		**6.4.4**	**TRÆR, B-TRÆR + BSF/DFS (PRE/POST/IN-ORDER)**
KAP. 7		**7.3.5, 7.4**	**PRIORITETSKØ, HEAP; TOTALORDNING/*COMPARATOR***
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			***N-ITERATOR***
KAP. 6			**(PRE/POST/IN-ORDER)**
KAP. 7			**, HEAP; TOTALORDNING/*COMPARATOR***
KAP. 8	8.		**RDBOK, BST, HASHTAB**
KAP. 9	9.3 -		**SØKETRÆR: BINÆRE, AVL**
KAP. 10	10.7	–	**SORTERING:QUICKSORT (MERGESORT); RANDOMISERING**
~~**KAP. 11**~~	~~GÅR UT~~		
KAP. 12			**GRAFER: DFS/BFS, SSSP, MST**

INF-102 (???)

1. bytt påmelding til I-120
2. er det noen som skal ta INF-102,
   – så vil vi underveis informere om deler av I-120 pensum som ikke inngår i INF-102
   – samme forelesninger, øvelsesoppleg og obligatoriske oppgaver

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, oljeplatform	18 750	750 000	2 250m

mengde data :	n	n*10
program		
effektivt	1 min	10 min
lite effektivt	1 min	1 000 min

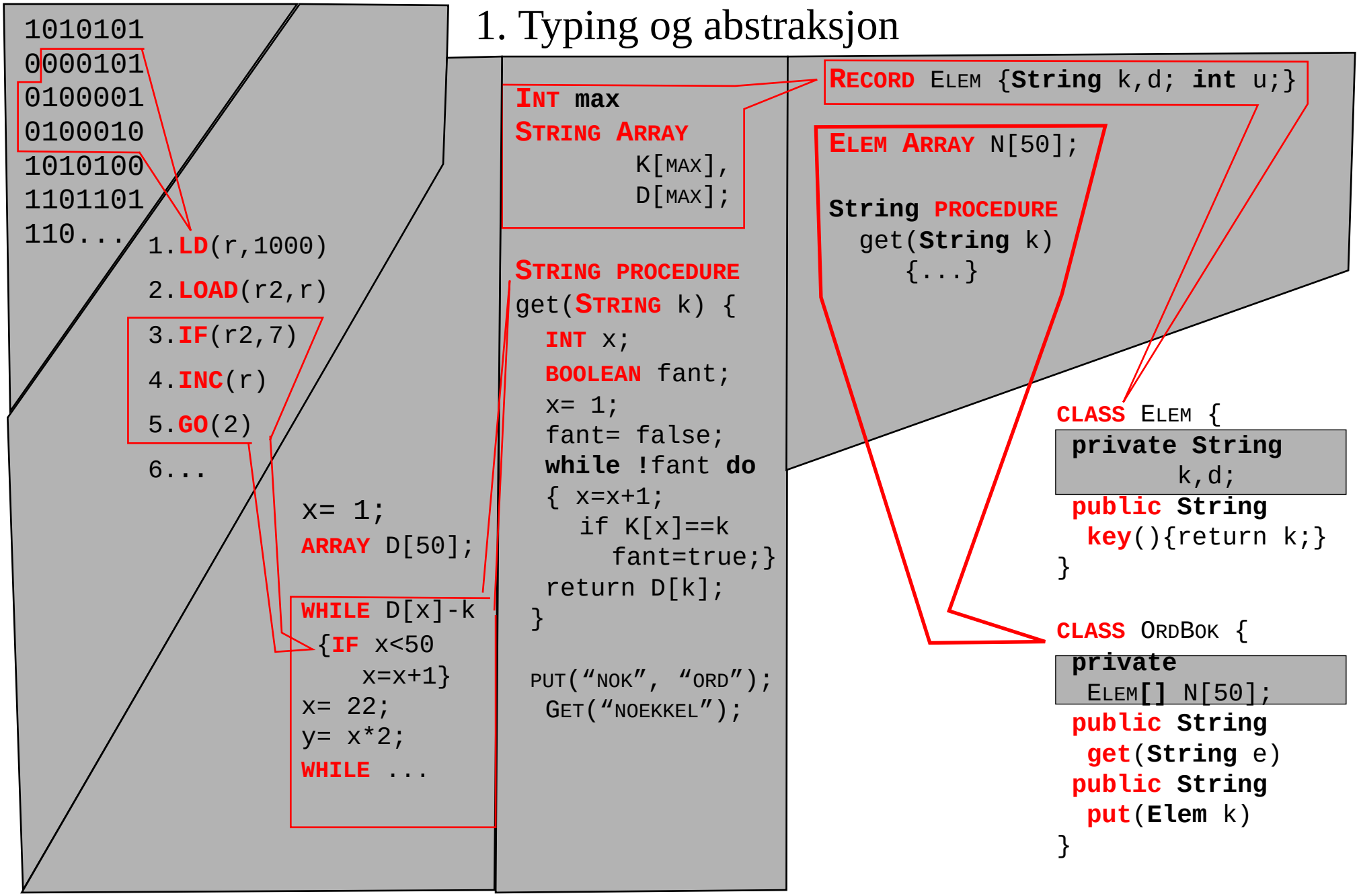
Programmereres aktivitet

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



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) {...}
}
```

```
   NEW_ :  INT → ORDBOK
_.PUT_ :  ORDBOK × ELEM → ORDBOK
_.GET_ :  ORDBOK × STRING → ELEM
```

```
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

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]);
} }
```

Utvikling av typer = utvikling av språket

Packages

- *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.Dictionary

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***.

. . .

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.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 ... ?

- *En Abstract Data Type ADT (interface) samler noen relaterte funksjoner*
- *Dens implementasjon i en klasse bruker en valgt Data Struktur*
- *Flere ADT'er og klasser vil ofte utgjøre en helhet – en pakke – for håndtering av et spesielt sett av problemer.*

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**

   > 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*

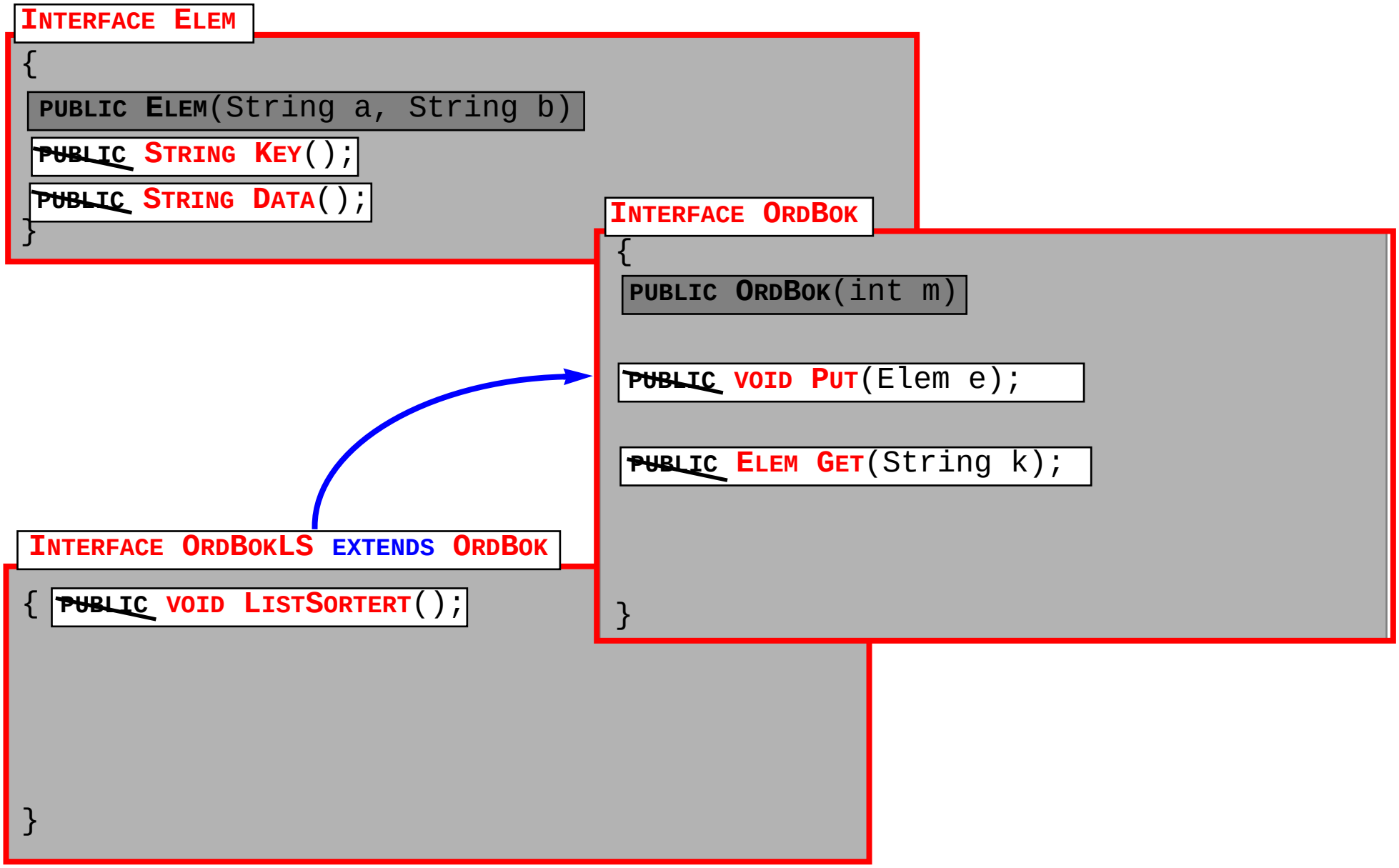
2. ***Abstrakt DataType*** = ***klasse*** – ***implementasjon***

ADT = ***grensesnitt:*** *Hva*

(abstract class vs. interface)

3. Ø = ***grensesnitt*** ∩ ***implementasjon***

4. ADT

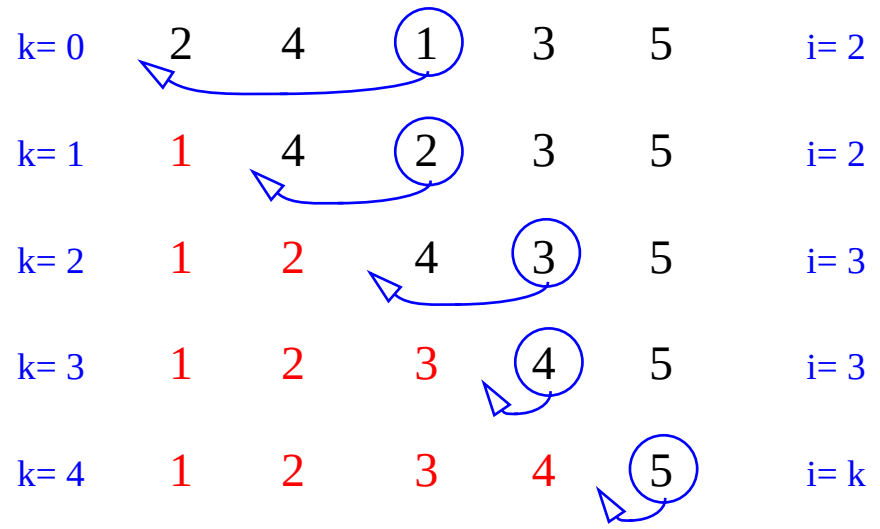

INTERFACE ELEM
{
PUBLIC ELEM(String a, String b)
PUBLIC STRING KEY();
PUBLIC STRING DATA();
}
INTERFACE ORDBOK
{
PUBLIC ORDBOK(int m)
PUBLIC VOID PUT(Elem e);
PUBLIC ELEM GET(String k);
}
INTERFACE ORDBOKLS EXTENDS ORDBOK
{ PUBLIC VOID LISTSORTERT();
}

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;
}
```

```
/** 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;
}   }
```

```
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)
}
```

```
blic 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.le
            if ( N[ j ] > m ) { m
        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]; i = 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)
}
```

```
blic void SS(int[] N) {       //  int >
int m, i;
for (int
    m = N
    for (i
        if (
    N[i]=
}
```

```
public void SS(Date[] N) {        //  Date <
  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 ] . 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.le
            if ( N[ j ] > m ) { m
        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]; i = 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)
e() < ((Integer)b).intValue(); }
ect a, Object b)

qualTo (Object a, Object b)

OrEqualTo (Object a, Object b)
```

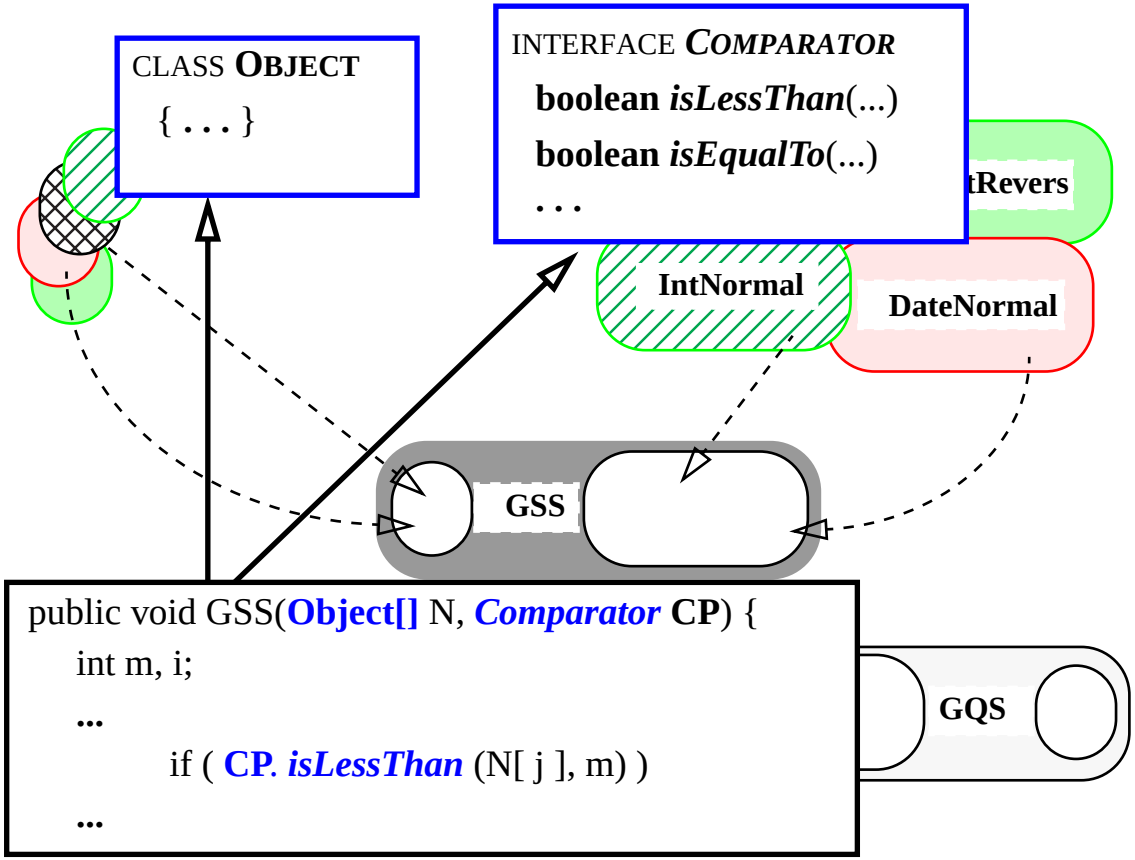
```
class IntNormalComp implements Comparator
  public boolean isLessThan (Object a, Object b)
    { return ((Integer)a).intValue() < ((Integer)b).intValue(); }
  public boolean isGreaterThan (Object a, Object b)
    { return ((Integer)a).intValue() > ((Integer)b).intValue(); }
  public boolean isEqualTo (Object a, Object b)
    { return ((Integer)a).intValue() == ((Integer)b).intValue(); }
  public boolean isLessThanOrEqualTo (Object a, Object b)
    { return ((Integer)a).intValue() <= ((Integer)b
  public boolean isGreaterThanOrEqualTo (Ob
    { return ((Integer)a).intValue() >= ((Integer)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 implementeasjon 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 ;
```

```
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; }
```

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

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() < 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) {
    ...
```

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