Datastrukturer och algoritmer

Ulf Nilsson
Institutionen för datavetenskap
Linköpings universitet

013–281935
ulfni@ida.liu.se
http://www.ida.liu.se/labs/logpro/ulfni/

Centrala begrepp

Abstraktion

Representation

Inkapsling

Ärvning

Återanvändning

Effektivitet

Modularisering

Datastrukturer

Datatyper

Abstrakta datatyper

Algoritmer

Abstraktion

Att bortse från onödiga detaljer.

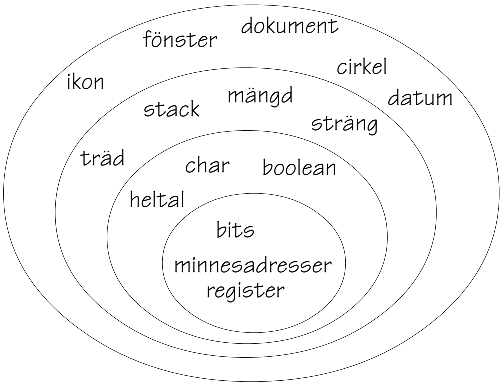
- Kontrollabstraktion:

```
/* Beräkna fakulteten av n */
int fact(int n) {
  int i = 1;
  while( n > 0 ) {
    i = i * n;
    n--;
  }
  return i;
}

int binomial(int n, int n) {

  return fact(n)/(fact(m)*fact(n - m));
}
```

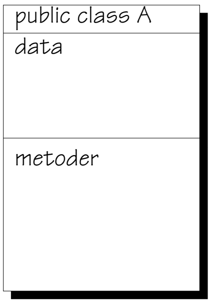
- Dataabstraktion:

```
public class Complex {
  private …

  Complex(double r, double i);
  public double Real();
  public double Imag();
  public  Complex add(Complex x);
  …

}
```

Datatyper



Klasser



public class A

data

metoder

```
public class IntCell {

  private int content = 0;

  /* Selector */
  public int read() {
    return content;
  }

  /* Muterare */
  public void write(int x) {
    content = x;
  }
}
```

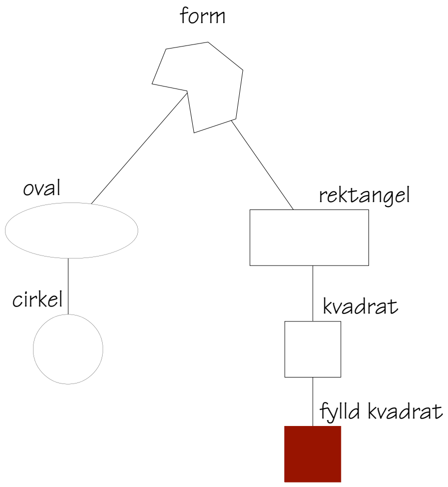
Datum

```
public class Date {

  private int month;
  private int day;
  private int year;

  /* Konstruerare */
  public Date(int m, int d, int y) {
    month = m;
    day = d;
    year = y;
  }

  /* Selektor */
  public int month() {
    return month;
  }

  /* etc */

  public boolean equals(Date d) {
    return d.month == month &&
           d.day == day &&
           d.year == year;
  }

  public String toString() {
    return (year + "-" + month + "-" + day);
  }

}
```

Account

```
public class Account {

  public Account(int initBalance) {
    balance = initBalance;
  }

  private int balance;

  public int balance() {
    return balance;
  }

  public int deposit(int amount) {
    balance = balance + amount;
    return balance;
  }

  public int withdraw(int amount) {
    if( balance < amount ) {
      return 0;
    } else {
      balance = balance - amount;
      return amount;
    }
  }

}
```

Ärvning



```
public class rektangel extends form {
  …
}

public class kvadrat extends rektangel {
  …
}
```

Abstrakta datatyper

Vad karaktäriserar ett rationellt tal?

```
class Rat {

  /* Konstruktor */
  Rat(int x, int y) ;

  /* Selector */
  int nom() ;

  /* Selektor */
  int denom() ;

}
```

Korrekthetskrav:

- Om x = new Rat(a, b) så $\frac{a}{b} = \frac{x.nom\ ()}{x.denom\ ()}$

Rationella tal

```
class Rational extends Rat {

  Rational(int x, int y) {
    super(x, y);
  }

  Rational mult(Rational a) {
    return new Rational( nom()*a.nom(),
                         denom()*a.denom());
  }

  boolean equals(Rational a) {
    return (nom()*a.denom() == a.nom()*denom());
  }

  public String toString() {
    return (nom() + "/" + denom());
  }

}


Rational a = new Rational(2, 3);
Rational b = new Rational(4, 4);
Rational c = a.mult(b);

if( a.equals(c) )
  System.out.println(a + " equals " + c);
else
  System.out.println(a + " does not equal " + c);
```

Representation 1

```
class Rat {

  private int x, y;

  Rat(int x, int y) {
    this.x = x;
    this.y = y;
  }

  int nom() {
    return x;
  }

  int denom() {
    return y;
  }

}
```

Representation 2

```
class Rat {

  private int x, y;

  static int gcd(int a, int b) {
    /* Beräkna största gemensamma */
    /* delaren av a och b */
  }

  Rat(int a, int b) {
    int c = gcd(a, b);
    x = a / c;
    y = b / c;
  }

  int nom() {
    return x;
  }

  int denom() {
    return y;
  }

}
```

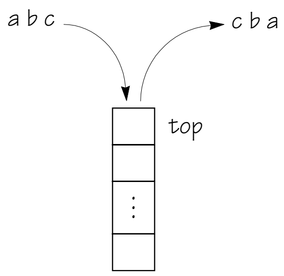
Representation 3

```
class Rat {

  private int x;

  Rat(int a, int b) {
    x = 2^a*3^b;
  }

  int nom() {
    int i = 0;
    while( x är delbar med 2) i++;
    return i;
  }

  int denom() {
    int i = 0;
    while( x är delbar med 3) i++;
    return i;
  }

}
```

Abstrakta datatyper

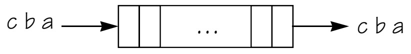
- Minnescell
- Stack
- Kö
- Prioritetskö
- Tabell (ordlista)
- Träd
- Mängder
- Grafer
- ...

Stack (LIFO-kö)



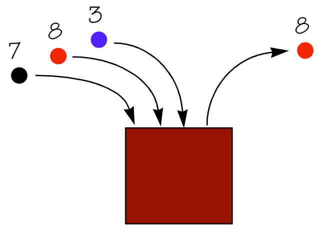
```
interface Stack {

  public Stack();
  public boolean empty()
  public void push(Object x)
  public Object pop()
  public Object top()

}


…
stack = new Stack();
…
stack.pop(stack.push(x)) == x
…
```

Kö (FIFO-kö)



```
public class Queue {

  public Queue();
  public boolean empty()
  public void enqueue(Object x)
  public Object dequeue()
  public Object front()

}
```

Prioritetskö



```
public class PriorityQueue {

  public PriorityQueue();
  public boolean empty();
  public void insert(Object x);
  public Object remove();
  …

}
```

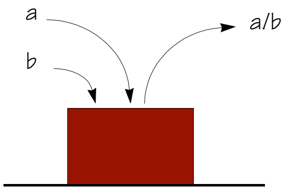
Tabell eller Ordlista

123	Anna
231	Calle
333	Stina
223	Bertil

```
public Class Table {

  public void insert(Key, Info);
  public Info lookup(Key);
  public void delete(Key);
  public void update(Key, Info);

}
```

- Jämför partiell funktion
- Jämför databas
- Kallas ibland ”Associativt minne”

Mängder



```
public Class Set {

  Set();
  public boolean empty();
  public void insert(Object x);
  public void remove(Object x);
  public boolean member(Object x);
  public Set union(Set x);
  public Set intersection(Set x);
  public boolean equals(Set x);

}
```

Sortering 1

```
static void sort(int[] a) {

  int j, tmp;

  for( int i = 0; i < a.length; i++ ) {
    /* Sök efter index till minsta elementet */
    j  = findMin(a, i , a.length);
    tmp = a[i];
    a[i] = a[j];
    a[j] = tmp;
  }
}

static int findMin(int[] a, int m, int n) {
  int min = m;

  while( m < n ) {
    if( a[m] < a[min] ) min = m;
    m++;
  }
  return min;
}
```

Sortering 2

```
static void bsort(int[] a) {

  boolean sorted = false;
  int tmp;

  while( ! sorted ) {
    sorted = true;
    for(int i = 1; i < a.length; i++ ) {
      if( a[i] < a[i-1] ) {
        tmp = a[i];
        a[i] = a[i-1];
        a[i-1] = tmp;
        sorted = false;
      }
    }
  }
}
```

Primtalstestning

```
public static boolean isPrime(int n) {

  if( n < 2 ) return false;
  else {
    int lim =
      Math.min((int)Math.sqrt((double)n)+1, n/2);
    for( int i = 2; i <= lim; i++ )
      if( n % i == 0 ) return false;
    return true;
  }
}

.
.
.

for( int i = 0; i <= 100; i++ )
  if( isPrime(i) ) System.out.print(i + " ");
System.out.println("done");
```

Eratosthenes ”sållning” (sieve)

```
static boolean[] mkSieve(int n) {

  boolean[] sieve = new boolean[n + 1];

  sieve[0] = sieve[1] = false;
  for( int i = 2; i <= n; i++ ) sieve[i] = true;

  int lim = (int)Math.sqrt(n) + 1;
  for( int i = 2; i <= lim; i++ ) {
    if( sieve[i] ) {
      for( int j = i*i; j <= n; j = j + i )
        sieve[j] = false;
    }
  }
  return sieve;
}

.
.
.

boolean[] isPrime = mkSieve(100);
for( int i = 0; i <= 100; i++ )
  if( isPrime[i] ) System.out.print(i + " ");
System.out.println("done");
```