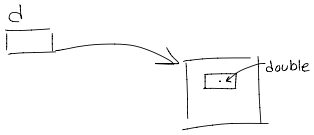
Rekursiva funktioner

```
import static javax.swing.JOptionPane.*;

public class NfakDemo {

  static int nfak(int n) {
    if (n <= 0)
      return 1;
    else
      return n * nfak(n-1);
  }

  public static void main(String[] arg) {
    String s = null;
    while ( (s = showInputDialog("n = ?")) != null )
          showMessageDialog(null, "n! = " + nfak(Integer.parseInt(s)));
  }
}
```

Klasser

Math - Klass med massa matematiska saker, massor av klassmetoder och klassvariabler.

Wrapper classes
Double (stort D, namnet på en klass) d = new Double();
Används för att agera referens till ett objekt.
Integer -> int
Double -> double
Float -> float



Wrapper classes innehåller även hjälpmetoder till respektive klass.

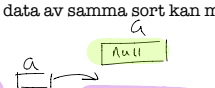
double x = d.getValue(); används för att hämta värdet från ovanstående d, emellertid finns den nyimplementerade funktionen boxing och unboxing.

Unboxing
x = d;

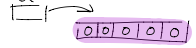
Boxing
d = x;

Array (Fält)

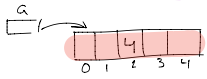
Om man har flera data av samma sort kan man bilda en array med dem.
Ex
int [] a;



a = new int [5];



a[2] = 4;



i = a[0];

a[i+j]=10;

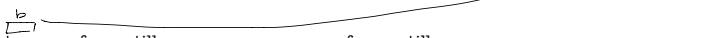
om indexeringssiffran blir högre än vad som faktiskt finns får du ett system error och du har gjort fel!

n = a.length; Ger arrayens längd!

Ex
```
for (int i = 0; i < a.length; i++)
      a[i] = 9;
```

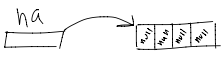
a = new int[10];

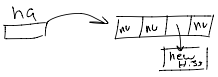
b = a; ger inte en ny array utan en referens till samma array som a refererar till.



När man definierar en array måste man definiera vad man ska ha i den. I exemplet ovan är det int, men du kan ha vad som helst!

Hiss [] ha = new Hiss[4];





ha[2].körTill(2); kommer ge ett exikveringsfel eftersom det är null arrayen.

ha[2] = new Hiss(); skapar ett objekt och i sin tur en referens till det objektet vilket placeras på plats 2 i arrayen, nu funkar ha.körTill ovan!

1. Skapa referensvariabeln.
2 . Skapa arrayen.
3. Skapa objekten som ska pekas ut i arryaen.

Det blir alltid en kopia!!

Listor

Standardklasser

```
import java.util.*;      // innehåller bl.a. listklasserna

List<String> ls = new LinkedList<String>();
List<Punkt>  lp = new LinkedList<Punkt>();
```

Enklare:

```
List<String> ls = new LinkedList<>();  // från Java version 7, ”diamant”
List<Punkt>  lp = new LinkedList<>();
```

`List` är ett *generiskt* gränssnitt (interface). Bestämmer egenskaperna sett utifrån.
`LinkedList` är en *generisk* klass. Innehåller implementeringen.

Annan klass möjlig:

```
List<String> ls = new ArrayList<>();
List<Punkt>  lp = new ArrayList<>();
```

Lägga in, ändra, avläsa och ta ut element:

```
lp.add(new Punkt(5, 7));
ls.add("EEE");
lp.add("ABC");      // FEL!!
ls.add("VVV");          // "EEE", "VVV"
ls.add("XXX");          // "EEE", "VVV", "XXX"
ls.add(0, "JJJ");       // "JJJ", "EEE", "VVV", "XXX"
ls.set(1, "AAA");       // "JJJ", "AAA", "VVV", "XXX"
ls.remove(3);           // "JJJ", "AAA", "VVV"
String t = ls.get(1);   // t får värdet "AAA"
ls.add(t);              // "JJJ", "AAA", "VVV", "AAA"
```

Skapa kopia av en lista:

```
List<String> ls2 = new ArrayList<String>(ls);
```

Löpa igenom en lista:

```
for (int i = 0; i<ls.size(); i++)   // sämre lösning
  System.out.println(ls.get(i));

for (String s : ls)                 // förenklat skrivsätt, bättre
  System.out.println(s);
```

Välja ut en del av en lista:

```
for (String s : ls.subList(1,3))
  System.out.println(s);

ls.subList(1, 3).clear();   // ls blir "JJJ", "AAA"
```

Listor med enkla element

```
List<int> intl = new LinkedList<>();  // FEL!!
```

Använd en *wrapper class*:

```
List<Integer> li = new LinkedList<>();
li.add(9);        // 9
li.add(0, 2);     // 2, 9
li.add(0, 5);     // 5, 2, 9
li.add(2, 7);     // 5, 2, 7, 9
li.set(3, -4);    // 5, 2, 7, -4
li.remove(2);     // 5, 2, -4
int j = li.get(2);   // j får värdet -4
```