Interface, Abstract Class ...

Det finnes ingenting

(ingen konkret program)

som kan gjøres med interface

men som ikke kan gjøres uten

Det finnes ingenting

(ingen konkret program)

som kan gjøres med bruk av unntak

men som ikke kan gjøres uten

Det finnes ingenting

(ingen konkret program)

som kan gjøres med typer/kasting

men som ikke kan gjøres uten

Det finnes ingenting

(ingen konkret program)

som kan gjøres med klasser/hierarki

men som ikke kan gjøres uten

Det finnes ingenting

(ingen konkret program)

som kan gjøres Objekt-Orientert

men som ikke kan gjøres uten objekter

Det finnes ingenting

(ingen konkret program)

som kan programmeres

men som ikke kan gjøres med bare

0011010101000101010100000111010110001 0100001010101010010101001001001010001 0100010010 1101010....

Datamaskin og Programmering =

strøm av / strøm på

```
10101010000101010000101000101010100.......
```

```
LOAD(r1,1) LOAD(r2,1) MULT(r1,r2) INC(r1)
IF(r1,r2,5) INC(r1) ADD(r1,r2) INC(r2) .....
```

```
x= 1; y=1; y= x*y;
if x then y=5;
while x do { y=y+1; z= y-x; x=x+1; }
while v do { y=y+1; z= y-v; v=v+1; }
```

```
int x; int y;
procedure p(int x)
while x>0 do { y= y+1; z= y-x; x=x+1; }
p(x)
p(v)
```

```
class Student {
   int aar; boolean mann;
   procudre registrer(Kurs k) {...}
   procedure fikKarakter(Kurs k, int e) {...}
}
```

```
interface Student {
   procedure registrer(Kurs k);
   procedure fikKarakter(Kurs k, int e);
}

abstract class .....
```

Det finnes ingenting

(ingen konkret program)

som kan gjøres med abstract class

men som ikke kan gjøres uten

Jeg vil ...

ha mange forskjellige sorteringsmetoder ... for å sammenlikne de

de skal sortere tabeller .... med noe

og sortering skal skje mht. .... en eller annen sammenlikningsoperasjon

```
public interface Sort {
    void sort();
    void show();
    void set(Object[] t);
}
```

???

```
public class Sort {
      Object[] tab;
      set(Object[] t) {
          tab= new Object[t.size];
          for (int k=0; k<t.size; k++) tab[k]= t[k]; }
      void swap(int i,j) {...}
      void show() { skriv ut tabellen }
      void sort() { }
}
```

g= new Sort() !
new SortL.sort() !

```
public abstract class Sort { Object[] tab;
      void set(Object[] t) {...}
      void show() {...}
      void swap(int i,j) {...}
      abstract void sort();
}
```

```
public class SortM implements Sort {
      Object[] tab;
      void sort() { mergesort(tab) }
}
```

```
public class SortS implements Sort {
  Object[] tab;
  void sort() { seleksjonsort(tab) }
}
```

```
public class SortM extends Sort {
      void sort() { mergesort(tab) }
}
```

```
public class SortS extends Sort {
      void sort() { seleksjonsort(tab) }
}
```

```
public class SortL extends Sort {
      void sort() { gjør det i morgen }
}
```

abstract class

```
Sort {
    protected Object[] tab;
    protected Comp cp;
    public void set(Object[] t) { set tab = t}
    public void show() { skriv tab }
    public void swap(int i,j) { bytt om i og j }

    public abstract void sort();
    public Sort(int m, Comp c) {
        tab= new Object[m];
        cp= c; }
}
```

Dette begrenser enhver (programmerer) som skal bruke klassen:

– `Sort s = new Sort()` – er ulovlig
  `s.sort()` – vil aldri forekomme

– `class SortS extends Sort { ...`
  `public void sort() {` **må implementeres** `} ... }`
  med mindre man vil si
  `abstract class SortS extends Sort {`
  ... med nye ting men ingen implementasjon av `sort() ... }`

– `SortS` kan deklarere nye konstruktorer – men kan alltid bruke super...

interface	**abstract class**
ingen implementasjon	delvis implementasjon
ingen datastruktur	mulighet for en datastruktur
ingen konstruktorer	mulighet for konstruktorer
multipel arv (av type)	enkel arv (som for klasser)

interface	**abstract class**
ingen implementasjon	**delvis implementasjon**
• ingen datastruktur	**• mulighet for en datastruktur**
• ingen konstruktorer	**• mulighet for konstruktorer**
multipel arv (av type)	**enkel arv (som for klasser)**

1.

```
interface Stack {
  void push(Object o);
  Object pop();
  Object peek();
  boolean empty();   }
```

```
abstract class Stack {
  abstract void push(Object o);
  abstract Object pop();
  abstract Object peek();
  abstract boolean empty();   }
```

2.

```
interface Sort {
  void sort();
  void set(Object[] t);
  void show();
  void swap(int i,j);
 }
```

```
abstract class Sort {
  abstract void sort();
  void set(Object[] t) { ... }
  void show() { ... }
  void swap(int i,j) { ... }
  Object[] tab;    }
```

interface brukes (1.) for å definere **"abstrakte typer"**
= **ingen føringer** på implementasjon = kun **typen** for grensesnitt operasjoner

abstract class brukes (2.) når
– data typen er **ikke "helt abstrakt"** : **en del** av implementasjon er bestemt
– **noen metoder**, vesentlige for data typen, har **ingen "generisk/typisk"** implementasjon
– "*abstrakt*" er en beskjed til programmerer – og kompilator! – om dette

vanlig *class* brukes når
– data typen er **"konkret"** : **alle metoder** har en **"generisk"** implementasjon og subklasser kan spesialisere/overskrive denne
– for å implementere interface og abstract class