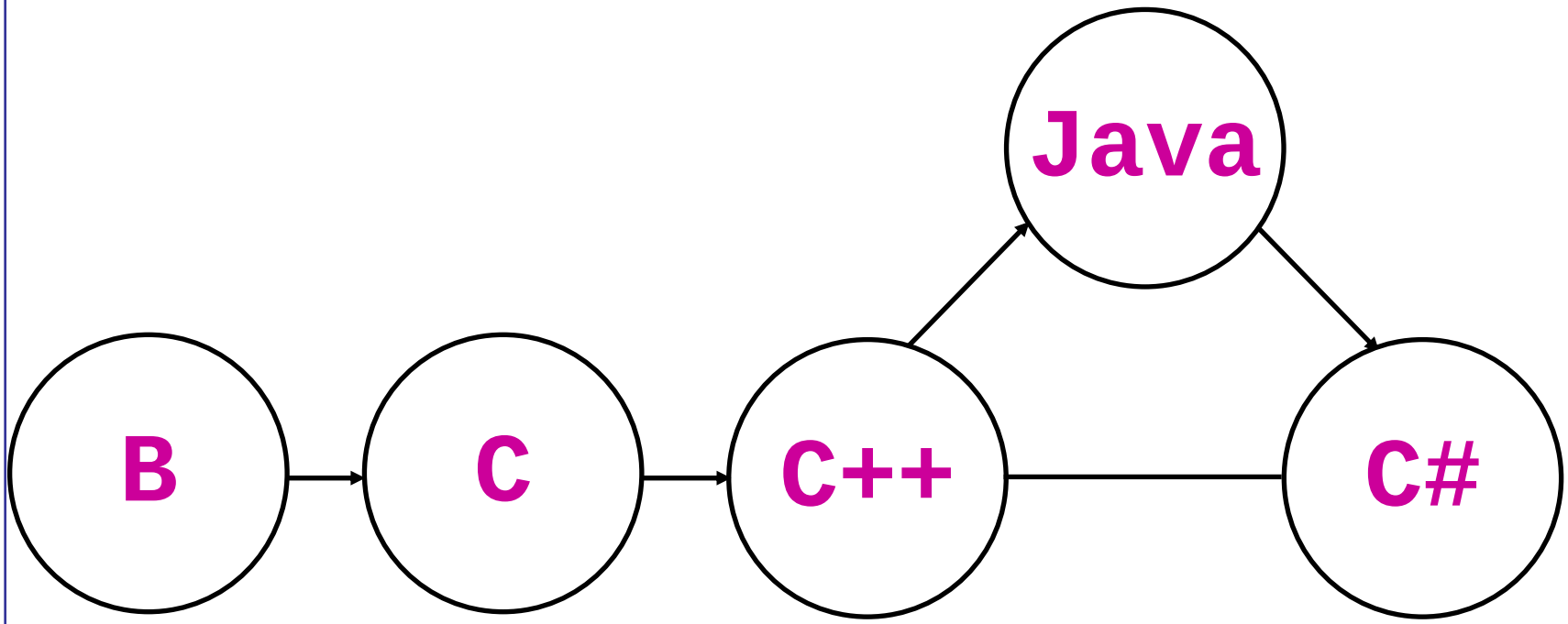
C grupas programmēšanas valodas



Programmā ir:

1. Algoritmi.
2. Datu struktūras.

N. Virts: "Algoritmi + datu struktūras = programmas"

Jautājums: **kas** ir galvenais?

C, Pascal: algoritmi. Programma ir apakšprogrammu *koks*.

C++, Java, C#: datu struktūras. Programma ir *grafs*.

Programmēšanas paradigmas

Procedūrāli orientētā paradigma

Uzmanības centrā – **darbības vārdi.** Orientācija uz *algoritmiem*.

Objektorientētā paradigma

Uzmanības centrā – **lietvārdi.** Orientācija uz *klasēm* un *objektiem*.

Loģiskā paradigma

Uzmanības centrā – **mērķi.**
Orientācija uz *pirmās kārtas predikātiem*.

Klases un objekti

Klase ir "rasējums" objektu izveidošanai.

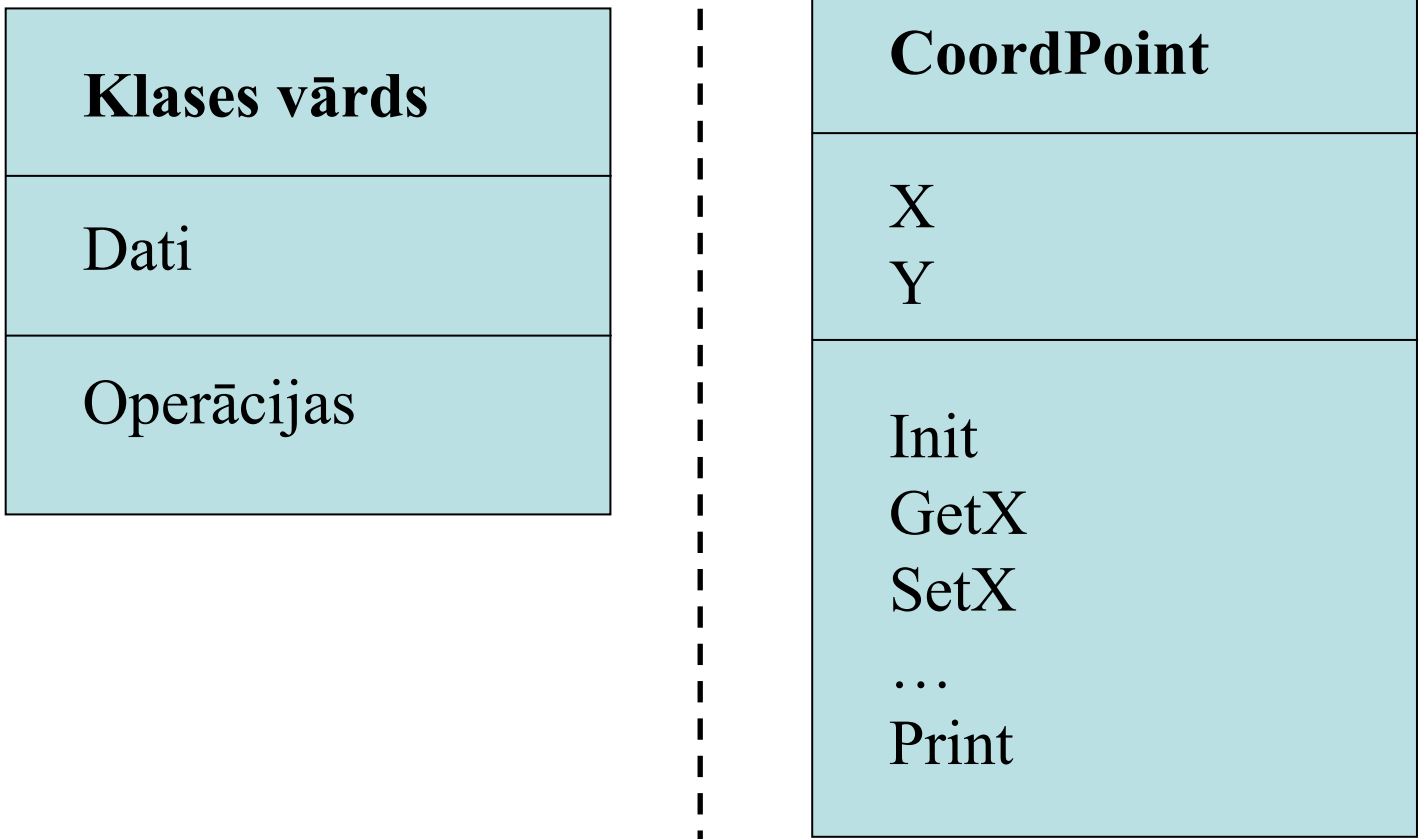
Objekts ir klases piemērs.

Cilvēks ir klase. Cilvēkam ir kāds vārds, uzvārds, dzimums…

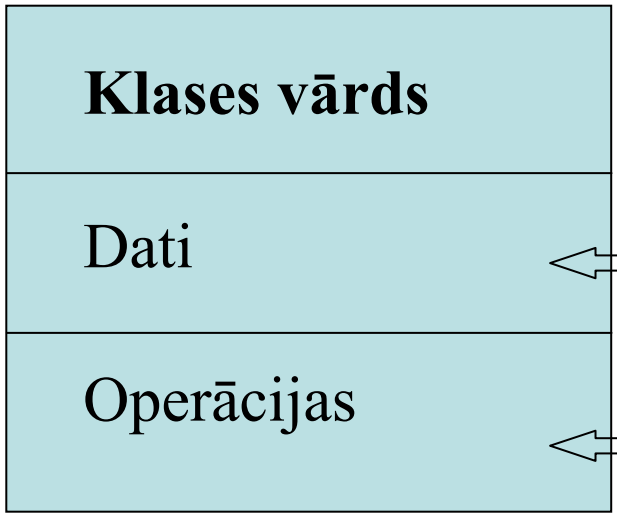
Jānis, Sergejs, Zaiga, Tatjana ir objekti. Viņiem ir konkrētais vārds, uzvārds, dzimums…

Objektam ir *stāvoklis*, *uzvedība* un *identiskums*.

Klases (izveidošanas principi)



Klases (terminoloģija)



Attributes
(atribūti)

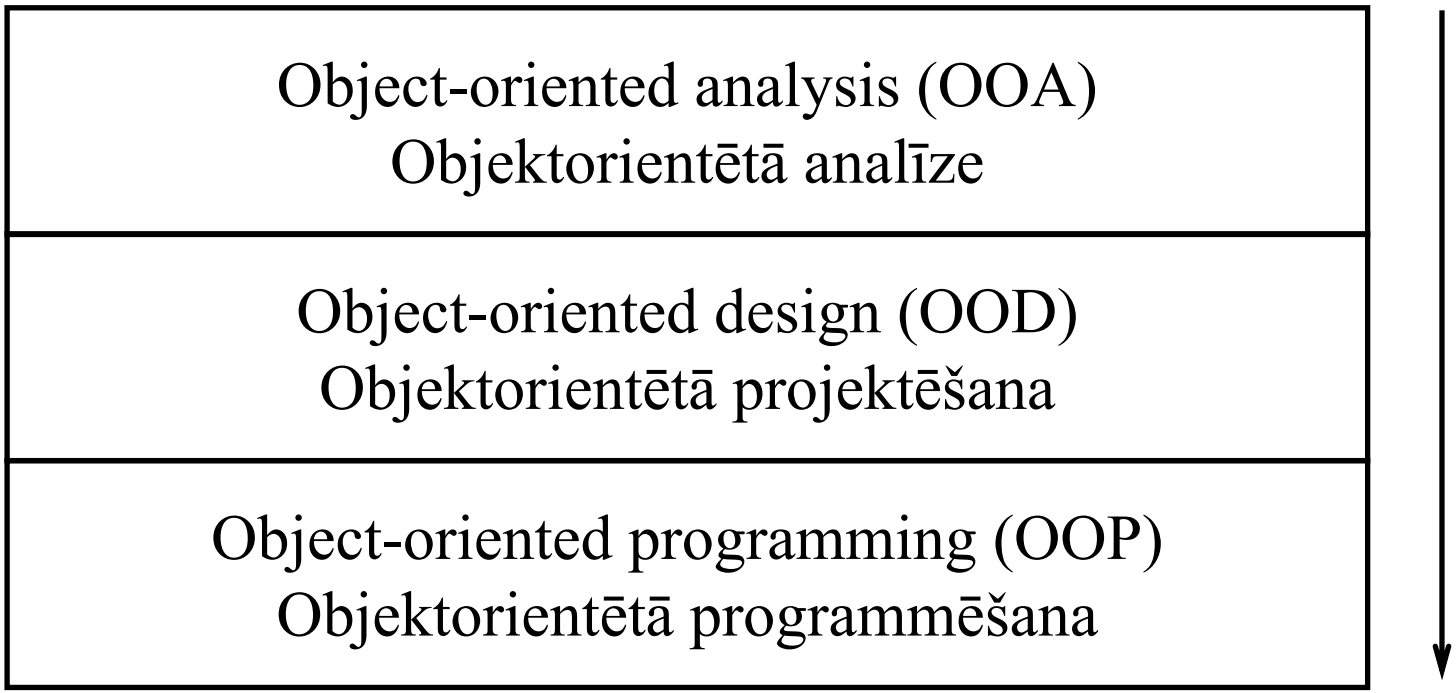
Methods
(metodes)

Members
(locekļi)

vai

Features
(īpašības)

Objektorientētās pieejas sastāvdaļas



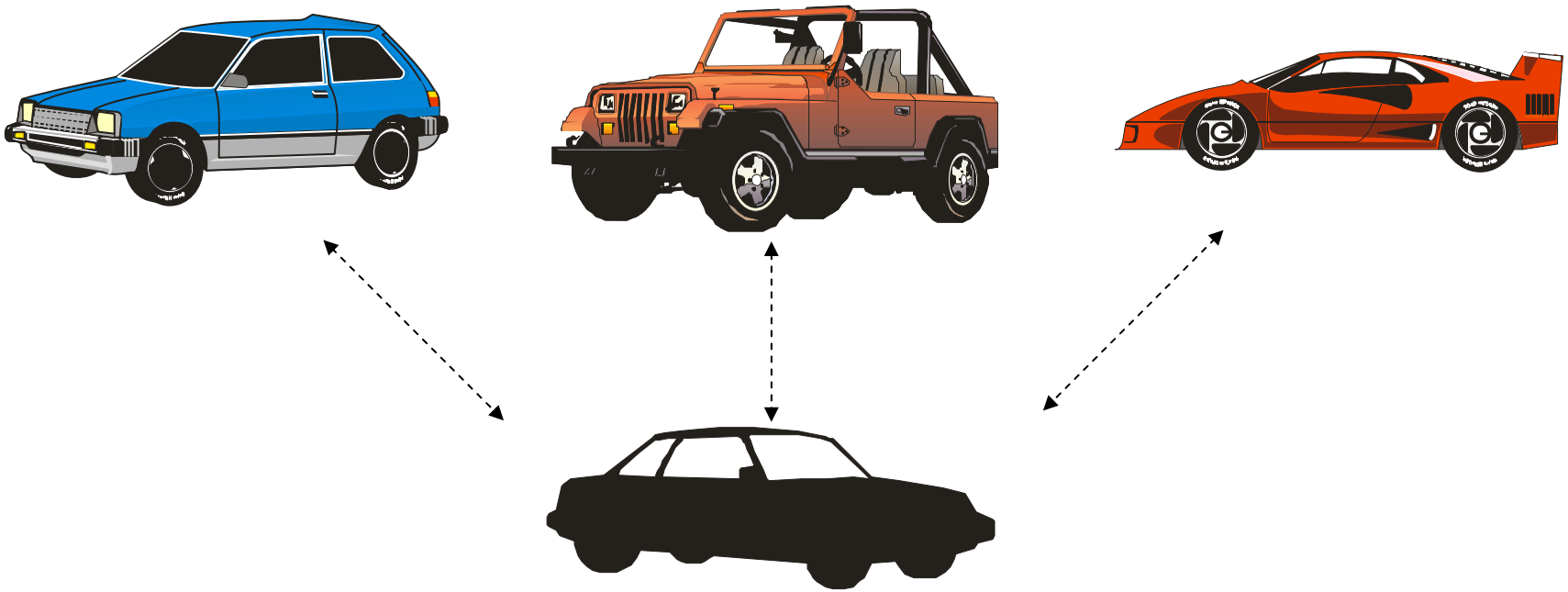
OOA – klašu un objektu meklēšana problēmas apgabalā.

OOD – objektorientētā dekompozīcija. Sistēmas strukturēšana (diagrammas).

OOP – objektorientētā realizācija (izmantojot OO programmēšanas valodas).

1. Abstrakcija (abstraction)

- ✓Svarīgākās īpašības (dati plus operācijas)
- ✓Kā izskatās objekts no ārpuses?



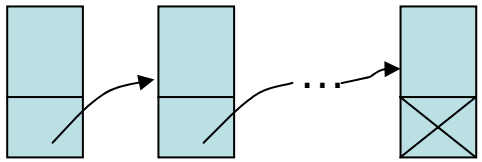
2. Iekapsulēšana (encapsulation)

Iekapsulēšana ir informācijas slēpšana.

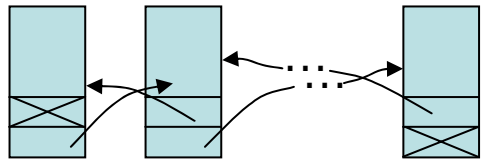
Steka abstrakcija

Steka realizācijas

Masīvs



Vienkāršsaistīts saraksts

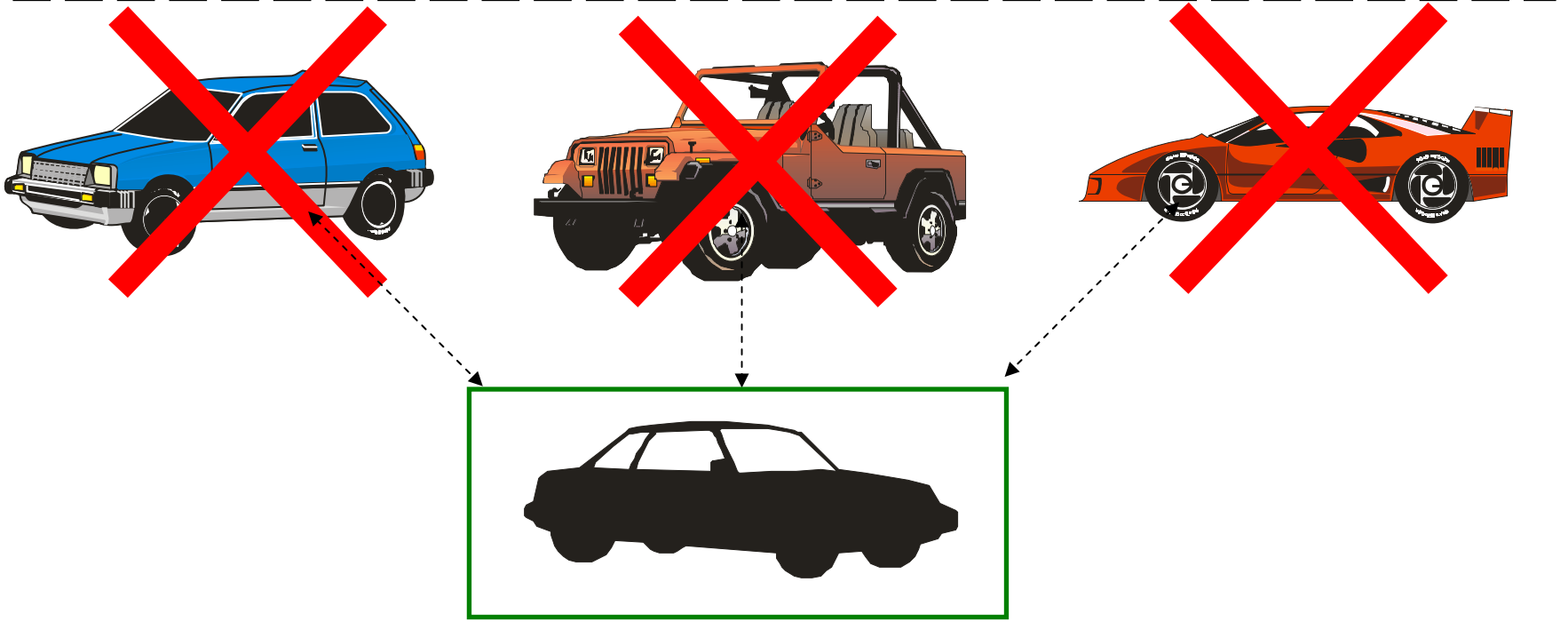


Divkāršsaistīts saraksts

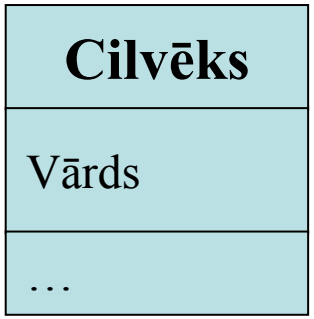
Abstrakcija + iekapsulēšana = …

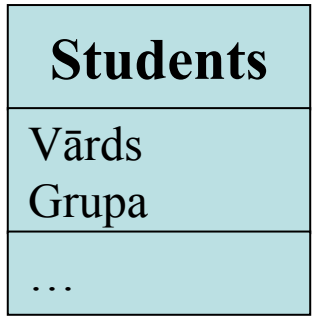
Rezultāts: *vienīgi iespējama* uztvere.

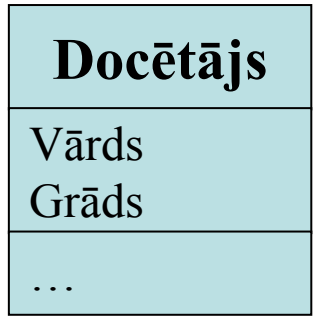
Abstrakcija *atver*, bet iekapsulēšana – *aizsargā*.



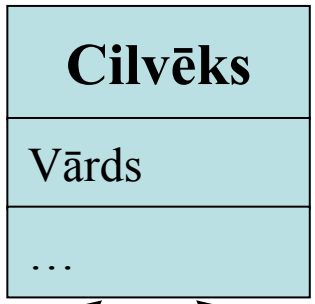
3. Hierarhija (hierarchy)



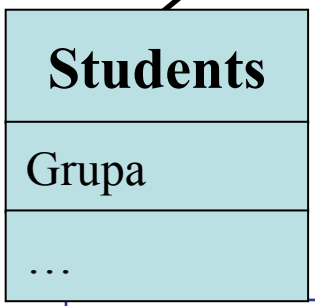


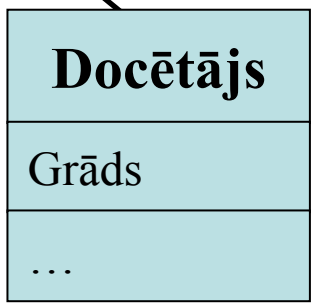


Hierarhiju realizē ar *mantošanas* palīdzību (attieksme *"is-a"*)



super class (superklase) vai **base class** (bāzes klase)

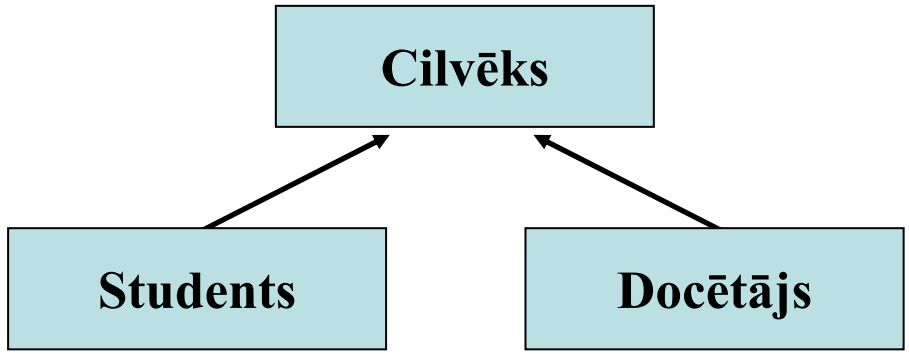




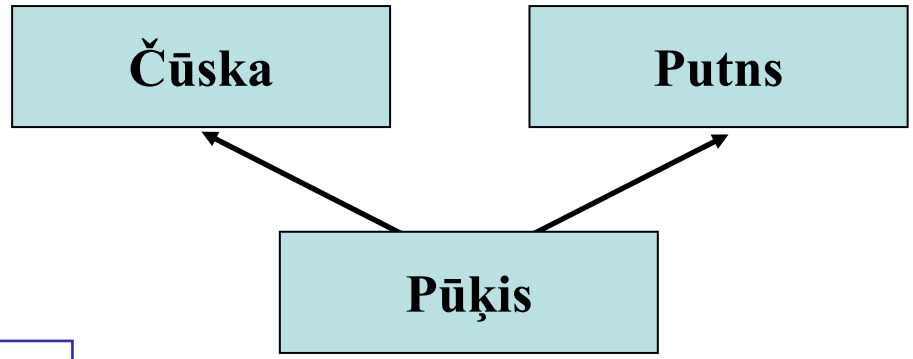
subclasses (apakšklases) vai **derived classes** (atvasinātas klases)

Mantošanas veidi

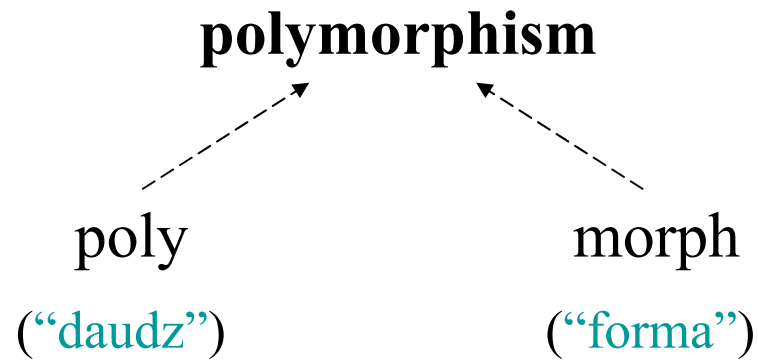
Single inheritance (*vienkāršā* mantošana)



Multiple inheritance (*daudzkāršā* mantošana)



4. Polimorfisms (polymorphism)



Dalīšana *C* valodā

5 / 2 = **2**

5.0 / 2 = **2.5**

Pārvietošanas operācija *move()*

move(PENCIL)

move(COMPUTER)

move(AUTO)

Struktūras *C* valodā

```
struct CoordPoint {
   int X;
   int Y;
};
```

```
void InitCoordPoint(struct CoordPoint*
   ParmCoordPoint,int ParmX, int ParmY) {
      ParmCoordPoint->X = ParmX;
      ParmCoordPoint->Y = ParmY;
}
```

```
struct CoordPoint CP;
InitCoordPoint(&CP, 1, 3);
SetX(&CP, 2);
PrintCoordPoint(CP);
CP.X = 4;
```

Ieraksti *Pascal* valodā

```
Type
   CoordPoint = record
      X : Integer;
      Y : Integer;
   End;
-------------------------------------------------
Procedure InitCoordPoint(Var ParmCoordPoint :
   CoordPoint; ParmX, ParmY : Integer);
Begin
   ParmCoordPoint.X := ParmX;
   ParmCoordPoint.Y := ParmY;
End;
-------------------------------------------------
Var
   CP : CoordPoint;
Begin
   InitCoordPoint(CP, 1, 3);
   SetX(CP, 2);
   PrintCoordPoint(CP);
   CP.X := 4;
```

Struktūras *C++* valodā

```
struct CoordPoint {
   int X;
   int Y;
   void InitCoordPoint(int ParmX, int ParmY) {
      X = ParmX;
      Y = ParmY;
   }
   ...
};
```

```
CoordPoint CP;

CP.InitCoordPoint(1, 3);
CP.PrintCoordPoint();
CP.SetX(2);
CP.PrintCoordPoint();
CP.X = 4;
```

CoordPoint ir *klase*; CP ir *objekts*.

`CP.SetX(2);` // *metodes izsaukums* vai *ziņojuma nosūtīšana* objektam *CP*

Iekapsulēšana *C++* valodā

```
struct CoordPoint {
   private:
      int X;
      int Y;
   public:
      void InitCoordPoint(int ParmX, int ParmY);
      ...
};
```

Metožu interfeiss

```
void CoordPoint::InitCoordPoint(int ParmX, int ParmY){
   X = ParmX;
   Y = ParmY;
}
```

Metožu realizācija

```
CoordPoint CP;
CP.InitCoordPoint(1, 3);
CP.SetX(2);
CP.PrintCoordPoint();
// CP.X = 4; aizliegts !!!
```

Konstruktori un *destruktori* *C++* valodā

```
class CoordPoint {
    ...
    public:
        CoordPoint();
        CoordPoint(int ParmX, int ParmY);
        ~CoordPoint() {
            cout << "Object destroyed !" << endl;
        }
        ...
};
```

```
CoordPoint::CoordPoint() {
    X = 1;
    Y = 3;
}
CoordPoint::CoordPoint(int ParmX, int ParmY) {
    X = ParmX;
    Y = ParmY;
}
```

Klases *Pascal* valodā

```
Type
   CoordPoint = object
      private
         X, Y : Integer;
      public
         Constructor InitDefCoordPoint;
         Constructor InitCoordPoint(ParmX, ParmY :
Integer);
         Destructor Done;
         Function GetX:Integer;
         ...
   End;
-----------------------------------------------
Constructor CoordPoint.InitDefCoordPoint;
Begin
   X := 1;
   Y := 3;
End;
-----------------------------------------------
   CP : CoordPoint;
Begin
   CP.InitDefCoordPoint;
```