Rīgas Tehniskā universitāte
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts
Lietišķo datorzinātņu katedra

DPI230 Objektorientētā programmēšana

Profesors Uldis Sukovskis

Vērtēšana

- **Laboratorijas darbi**
  - **Laboratorijas darbi jāizpilda tikai Lietišķo datorzinātņu katedras datorklasē**
  - **Atzīme par laboratorijas darbiem ietilpst gala atzīmē**
- **Eksāmens**
  - **Kursa beigās jākārto rakstisks eksāmens**
  - **Gala atzīmi (A) aprēķina no eksāmena atzīmes (E) un atzīmes par laboratorijas darbiem (LD)**

$$A = 0.65 * E + 0.35 * LD$$

Mācību materiāli

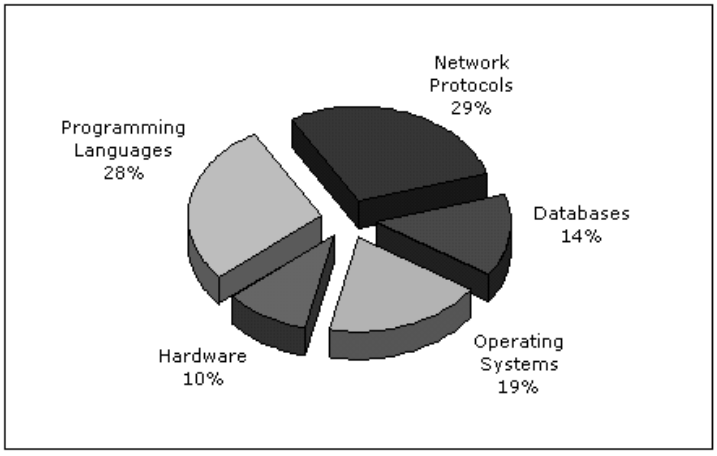
- Mācību materiāli tiek ievietoti portāla ORTUS e-studiju vidē
  - Prezentāciju slaidi
  - Programmu piemēri
  - Citi materiāli

- Noderīgas saites par objektorientēto programmēšanu C++
  - www.cplusplus.com
  - www.cpp-tutorial.com/tutorial-cpp.html
  - www.cprogramming.com
  - www.programmingtutorials.com
  - cplus.about.com

3

IT zināšanu pieprasījums darba tirgū

Which skill-set was in highest demand in IT projects?



Avots: IT-Careernet, www.it-careernet.com

4

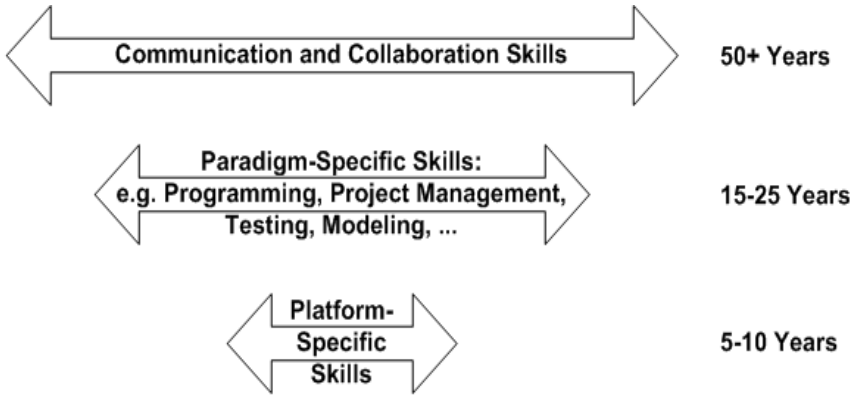
IT zināšanu pieprasījums darba tirgū

	Software Languages	Databanks	Operating Systems	Hardware Profiles
1	C	Access	Windows	PC
2	C++	SQL	Windows NT	INTEL Platforms
3	Java	Oracle	MS DOS	Sun
4	HTML	ODBC	UNIX	HP
5	JavaScript	MS SQL Server	LINUX	IBM RS/6000
6	Pascal	dBase	Novell	IBM Mainframes
7	Basic	DB2	OS/2	IBM
8	Assembler	mSQL / mySQL	DOS	Iomega
9	Visual Basic	Informix	Solaris	IBM AS/400
10	Cobol	JDBC	Novell Netware	HP Workstations
11	Perl	Lotus Notes	HP-UX	IBM AT
12	VBA (Excel, Access, u.a.)	Paradox	Sun-OS	Amiga

Avots: IT-Careernet, www.it-careernet.com

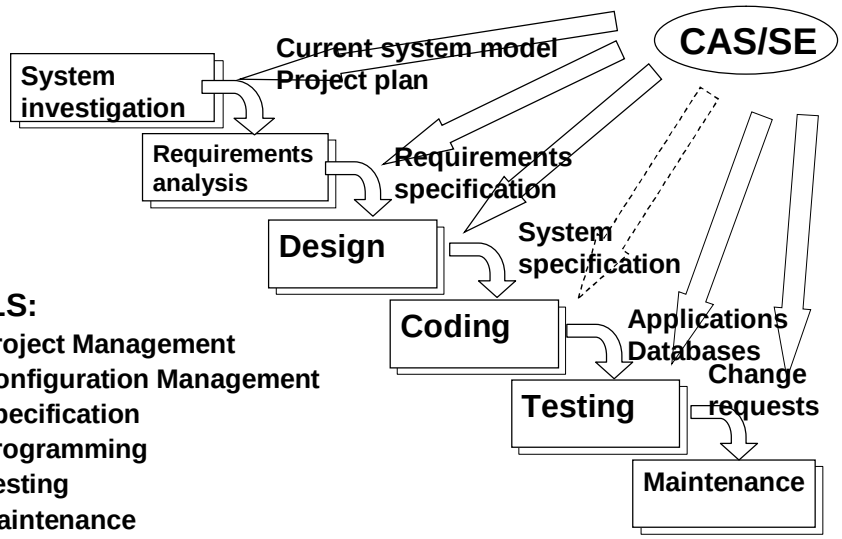
5

IT zināšanu ilglaicīgums



6

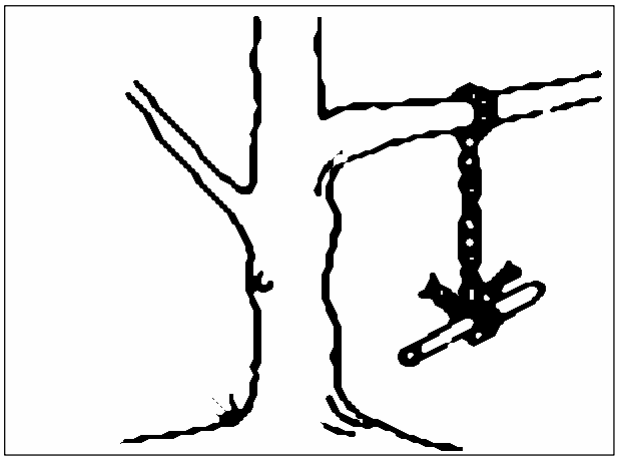
Programmatūras izstrādes dzīves cikls
Software Development Life Cycle (SDLC)



TOOLS:

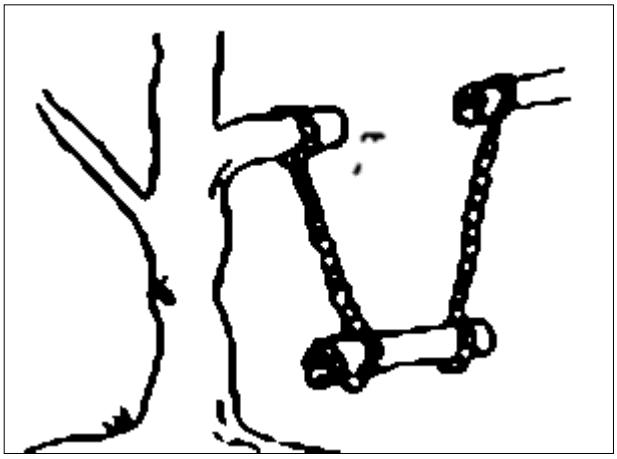
- Project Management
- Configuration Management
- Specification
- Programming
- Testing
- Maintenance

What the user asked for ?



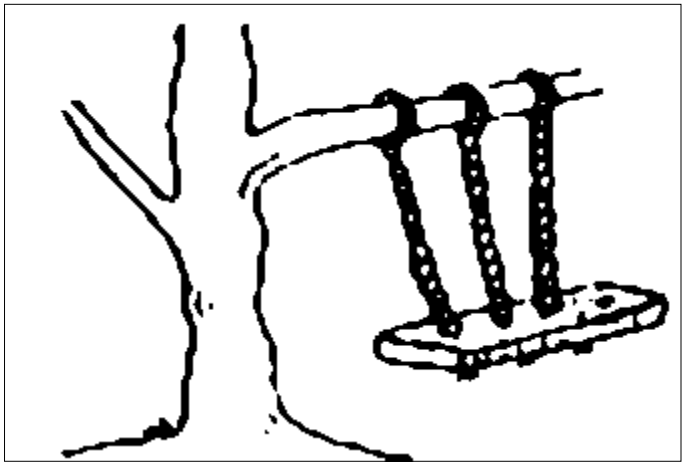
8

How the analyst saw it ?



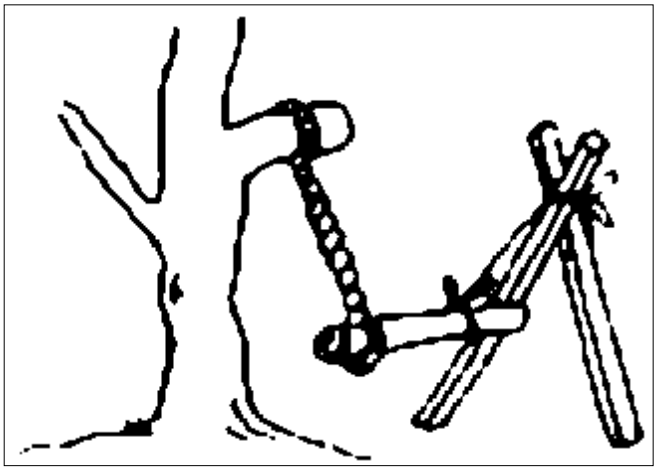
9

How the system was designed?



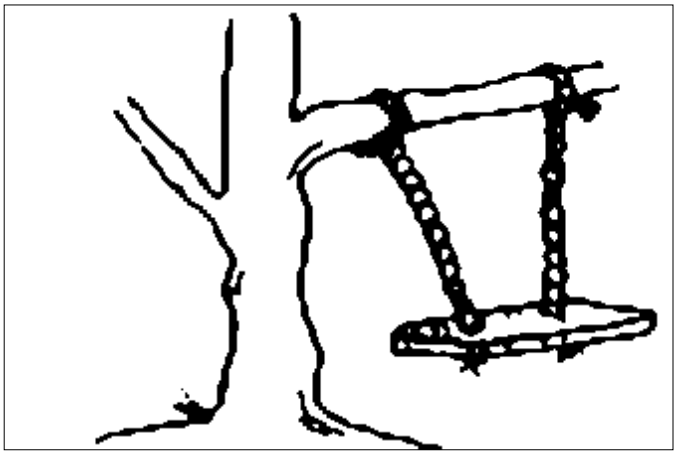
10

As the programmer wrote it



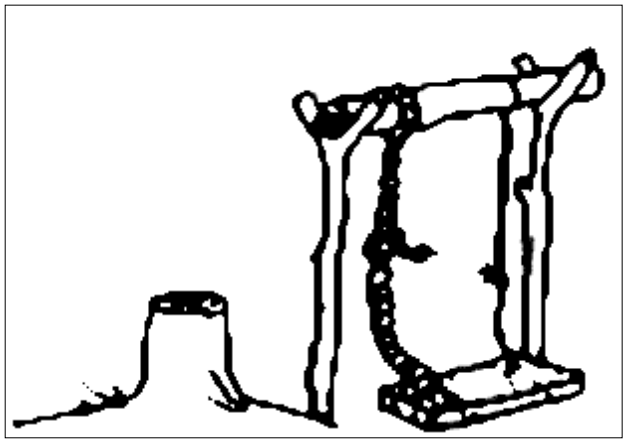
11

What the user really wanted?



12

How it actually works?



13

Vienkāršas programmas piemērs

Uzrakstīt programmu, kas ļauj aprēķināt trīsstūra laukumu, ja doti tā malu garumi.

Lietosim Hērona formulu :

$$S=\sqrt{p(p-a)(p-b)(p-c)}$$

a, b, c - trīsstūra malu garumi,

p - pusperimetrs

$$p=\frac{a+b+c}{2}$$

14

Trīsstūra klase

Triangle.h

```
class Triangle {
    public:
        int a, b, c;
        float area();
        int perimeter();
};
```

15

Trīsstūra klases funkcijas

Triangle.cpp

```
#include <math.h>
#include "Triangle.h"

float Triangle::area()
{    float p;
     p = (a + b + c)/2.0;  // half perimeter
     return (sqrt(p*(p - a)*(p-b)*(p - c)));
}

int Triangle::perimeter()
{
     return (a + b + c);
}
```

16

Kā tas strādā

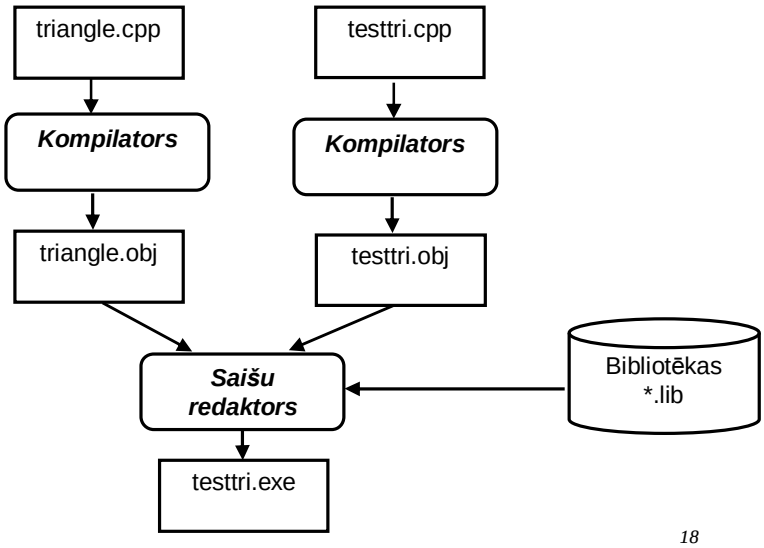
Testtri.cpp

```
#include <iostream.h>
#include "Triangle.h"

void main()
{
   Triangle t;    // Objekts
   t.a = 50;      // Malu garumi
   t.b = 60;
   t.c = 35;
   float ta;
   ta = t.area();
   cout<< "Area of triangle is " << ta << '\n';
   cout<< "Perimeter is "<<t.perimeter()<< '\n';
}
```

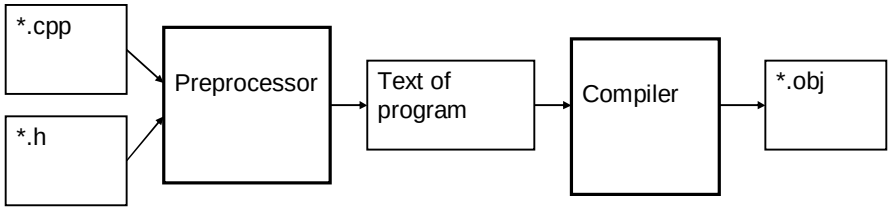
17

Programmas izveidošana



18

Preprocesors



Programmas tekstā rindas, kas sākas ar simbolu #, tiek apstrādātas kā <u>preprocesora direktīvas</u>.

```
#define
#undef
#ifdef
#ifndef
#include
```

19

Preprocesors (turpinājums)

`#define` *identifikators [virkne]*

```
#define PI 3.14159
. . .
c = 2 * PI * r;     // c = 2 * 3.14159 * r;
cout << "2*PI*r";   // cout << "2*PI*r";

#define SIZE  128
. . .
#undef SIZE
. . .
#define SIZE  1024
. . .
#undef SIZE

#ifndef SIZE
#define SIZE  1024
#endif
```

20

Preprocesors (turpinājums)

Fails MyClass.h

```
#ifndef _MYCLASS_H_
#define _MYCLASS_H_

class MyClass {
   int Duplicate(int x);
};
#endif _MYCLASS_H_
```

Fails MyClass.cpp

```
#include “MyClass.h”

int MyClass::Duplicate(int x) {
   return x + x;
}
```

Fails Main.cpp

```
#include “MyClass.h”

void main(void) {
   MyClass c1;
   ...
}
```

21

Preprocesors (turpinājums)

#define ***identifikators(parametri) virkne***

```
#define CUBE(X) X * X * X
. . .
int a, b;
. . .
a = 2;
b = CUBE(a);
                  // b = a * a * a;
b = CUBE(a+1);
                  // b = a + 1 * a + 1 * a + 1;

#define CUBE(X) ((X) * (X) * (X))
. . .
b = CUBE(a+1);
                  // b = ((a + 1) * (a + 1) * (a + 1));
```

22

Preprocesors (turpinājums)

```
#define CUBE(X) ((X) * (X) * (X))

int cube(int x)
{
  return x * x * x;
}

int a, b;

a = 3;
b = cube( a++ );    // a = 4, b = 27

a = 3;
b = CUBE( a++ );    // a = 6, b = 27 !!!
```

23

Preprocesors (turpinājums)

#include < *file* >
#include " *file* "
#include *name*

#include <string.h> no standarta kataloga

#include "mystring.h” no tekošā kataloga

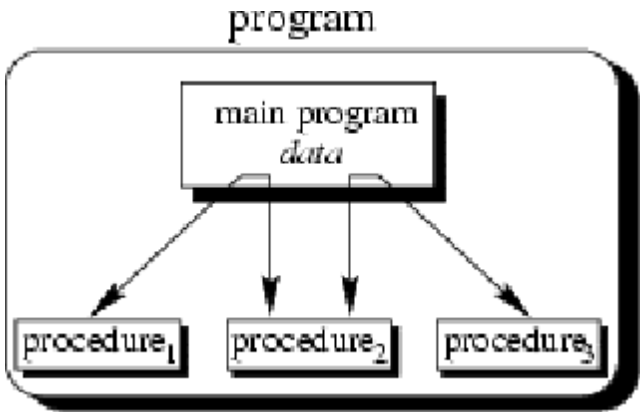
```
#include "c:\cprog\sample25\tryit.h"

#define MYHEADER "c:\cprog\sample25\second.h"

#include  MYHEADER
#include  "MYHEADER"
```

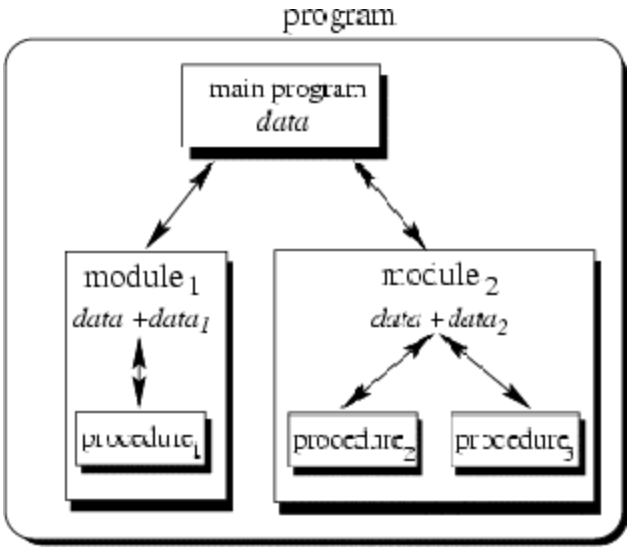
24

Procedurālā programmēšana



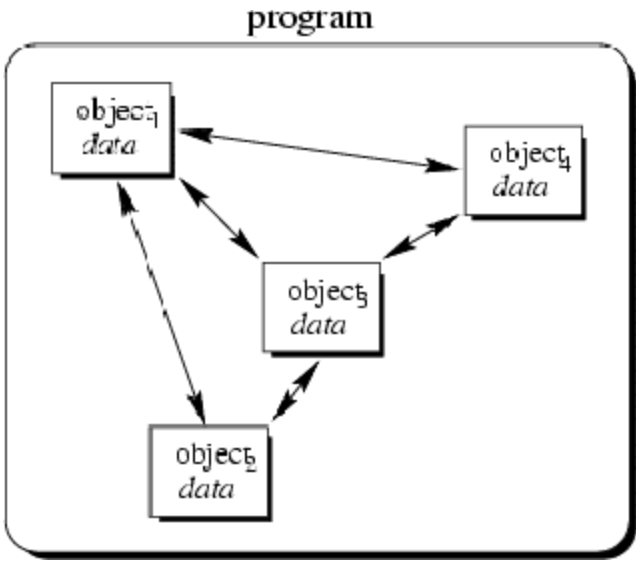
25

Modulārā programmēšana



26

Objektorientētā programmēšana



27

Objektorientētās programmēšanas pamatjēdzieni

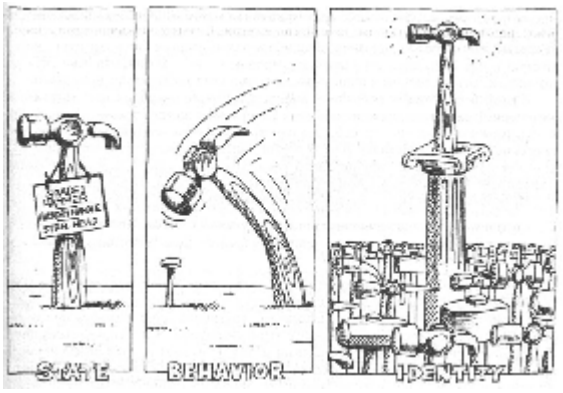
- Objekts
- Ziņojums
- Klase
- Instance, eksemplārs
- Metode

- Abstrakcija
- Iekapsulēšana
- Mantošana
- Polimorfisms

28

Objekts

- **Objekts ir klases eksemplārs**
- **Objektam ir identitāte, stāvoklis un uzvedība**




```
Triangle t;                          // Objekts t
t.a = 50; t.b = 60; t.c = 35;        // Malu garumi
x = t.area();                        // Laukums
```

29

Ziņojums

n Veids, kā objekti savā starpā sazinās – objekta metodes izsaukums

```
...
Word W;
W.OpenDocument("c:\Docs\Letter.doc");
W.PrintDocument();
...


Triangle t13;
...
s = t13.area();
```

30

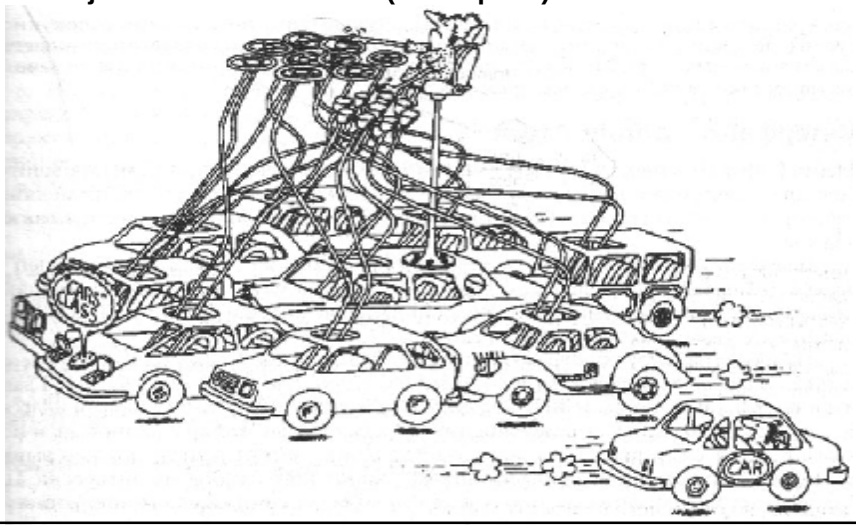
Metode

- Definē algoritmu, kuru izpilda objekts, reaģējot uz ziņojumu
- Metode ir kods, kas ir piesaistīts vai nu klasei (klases metode jeb statiskā metode), vai objektam (instances metode)

31

Klase

- **Klase reprezentē objektu kopu ar līdzīgu struktūru un uzvedību**
- **Objekts ir klases instance (eksemplārs)**



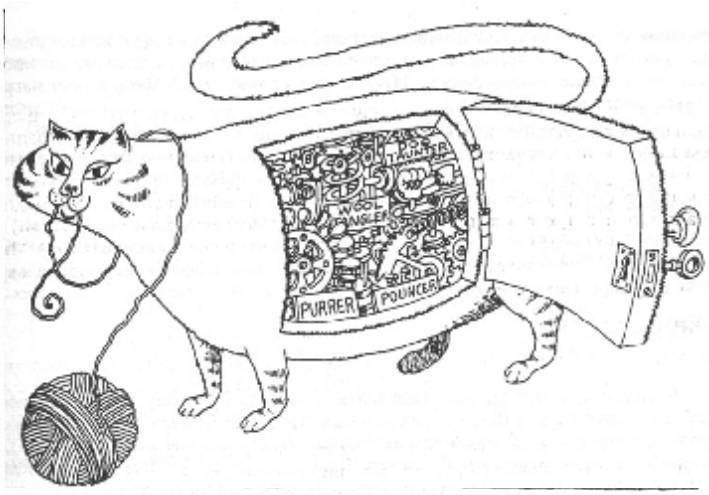
Abstrakcija

- Vienkāršots apraksts vai skats uz kaut ko, kas uzsver tā būtiskākās raksturiezīmes vai mērķus, tajā pašā laikā atmetot detaļas, kas nav būtiskas vai ir traucējošas uztverei.
- Izdala būtiskākos kāda objekta raksturojumus, kas to atšķir no visiem pārējie objektiem, tādā veidā, no vērotāja viedokļa, precizē tā konceptuālās robežas.

33

Iekapsulēšana

- Iekapsulēšana apraksta objekta spēju paslēpt savus datus un metodes no ārienes



34

Iekapsulēšana valodā C++

Klases

```
class Triangle {
public: int a, b, c;
        float area();
        int perimeter();
};

class Dog {
      private: char Name[12];
      public:  char* getName();
};
```

Struktūras

```
struct Record {
      int   number;
      char* name;
};
```

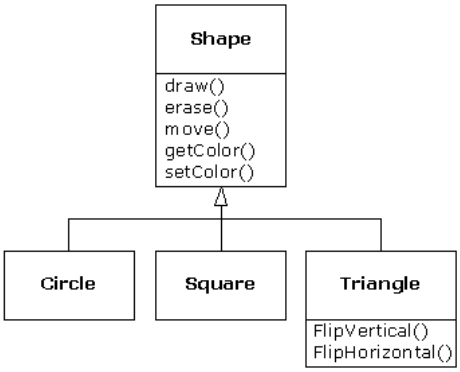
35

Mantošana

- Apakšklase manto virsklases struktūru un uzvedību
- Veids, kā formēt jaunas klases, izmanto iepriekš definētās klases, kur jaunās klases pārņem iepriekšējo klašu īpašības
- Izmanto, lai veicinātu esošā koda izmantošanu ar nelielām vai vispār bez modifikācijām

36

Mantošana valodā C++



37

Polimorfisms

- **Vārdam 'polimorfisms' ir grieķu izcelsme un tā nozīme ir 'tāds, kuram ir vairākas formas'.**
- **Objektorientētajā programmēšanā polimorfisma princips ir 'viens interfeiss, vairākas metodes'.**
- **Valodā C++ polimorfismu visbiežāk lieto attiecībā uz funkcijām un operācijām, kad vienādi nosauktas funkcijas vai operācijas realizē dažādas darbības.**

Operācija <<

```
cout << "Ievadiet paroli: ";
...
int n, flag;
flag = n << 2;
```

38

Polimorfisms

```
class Square {
public:
   int side;
   int area();
   int perimeter();
};


int Square::area()
{
   return side * side;
}

int Square::perimeter()
{
   return 4*side;
}
```

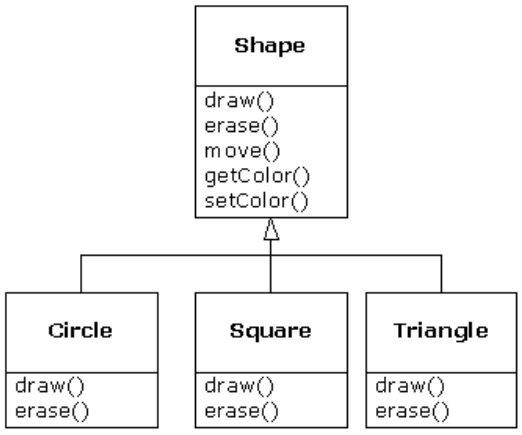
```
class Circle {
public:
   int radius;
   float area();
   float perimeter();
};


float Circle::area()
{
   return 3.14159*radius*radius;
}

float Circle::perimeter()
{
   return (2 * 3.14159 * radius);
}
```

39

Polimorfisms



40

Uzdevuma risināšanas soļi

1. **Analizēt uzdevumu un identificēt galvenos objektus.**
2. **Aprakstīt šo objektu iekšējo datu struktūru.**
3. **Grupēt objektus, veidojot augstākas klases. Uzzīmēt klašu hierarhiju.**
4. **Katrai klasei noteikt metodes un to pieejamības noteikumus.**
5. **Izstrādāt kopējo uzdevuma risināšanas shēmu.**
6. **Uzrakstīt katras metodes algoritmu.**
7. **Pārskatīt un uzlabot klases un uzdevuma risināšanas shēmu.**