# **Formulated by Alan Kay**

# **The 5 most important principles of OOP**

1. **Everything is an object**. . Each object is a variable of its own - an object (like any variable) holds data that defines the state of the object. However, some functions (methods) are inextricably linked to each object, through which the state of the object (behavior of the object!) can be changed. Virtually any concept can be declared as an object in the program.
2. **A program is a collection of objects** that send messages to each other. If one object wants to send a message to another object (e.g. a request about its status), this object calls the corresponding method (function) of the other object.

3.**Each object has its own personal memory**, which, as a rule, is closed to other objects. Therefore, the construction of the object remains hidden (hidden implementation), because as a rule, you can communicate with the object only through its methods (object interface, interface).

**4.Each object is of a specific type**. In other words, each specific object is an instance of an object type (class). The user of the class wants to know the answers to the questions:

– what services do these types of objects offer?

– what messages can be sent to this type of object?

**5.All objects of the same type have the same construction**. Therefore, all objects of a given type can receive the same messages.

It is possible to derive subtypes from more general object types. For example, if the Shape class is created, it is possible to derive the Circle object type from this object type, which inherits all properties (data and methods) of the Shape class. Therefore, each object of the Ring type can also receive all such messages that can be handled by objects of the Shape type.

Therefore, by writing the program code (methods) of the Shape class once, this program section is automatically transferred to the program code of all classes derived from the Shape class!

**The derivation-inheritance scheme is the most important mechanism that ensures the power and efficiency of OOP.**

OOP LANGUAGES

Three features:

1) **encapsulation (encapsulation)** – the language has tools that allow data and operations performed with them (methods) to be combined into one programmatic unit (object type)

NB! as a rule, only methods can access object data

 through!

2) **inheritance** – new object types can be derived from each object type, with the descendants inheriting the properties of the ancestor (both data and methods)

3) **polymorphism** – methods with the same name can be implemented in different ways on different objects

###  Objektitüüp Stack

OP 1

OP 2

## OP 4

OP 3

pop

## view

push

init

#### Object types in C++

**Descriptors: struct union class**

 <kirjeldaja> <objektitüübi\_identifikaator>

 {

 <muutujad\_ja\_meetodid\_=\_LIIKMED\_(*members*)>

 }

// IntStack.cpp - täisarvude LiFo

#include <iostream.h>

struct **Elem**

{

 int **arv**;

 Elem \***next**;

};

class **Stack**

{

private:

 Elem \***top**;

public:

 void **init**( void ) { top = NULL; }

void **push**( int a )

 {

 Elem \*uus = new Elem;

 uus -> arv = a;

 uus -> next = top;

 top = uus;

 }

int pop( int &a)

 {

 if( !top ) return 0; // FALSE kui tühi

 Elem \*vana = top;

 a = top -> arv;

 top = top -> next;

 delete vana; // vabastada mälu

 return 1; // TRUE kui võetakse element

 }

 void view( void )

 {

 Elem \*jooksev = top;

 int n = 0;

 while( jooksev )

 {

 cout << jooksev -> arv << "\n";

 n++;

 };

 cout << "\n Kokku " << n <<

" elementi\n";

 }

};

int **main**( void )

{

 Stack mag; // uus ilming

 mag.init();

 for( int i=0; i<10; i++)

 mag.push( i );

 mag.view();

 Stack pinu; // uus ilming

 pinu.init();

int num;

 while( mag.pop( num ))

 pinu.push( num );

 pinu.view();

 pinu.top ->arv = 7; // Viga! privaatne!

 mag.top = NULL; // Viga! privaatne!

 return 0;

}

**Access specifiers: public: private: protected**

***1) methods of a class (object type) have access to all members of that class***

***2) only public members are accessible (using compound name) through the object type manifestation***

By default, members are:

class -> private

struct -> public

union -> public