Object-Oriented Programming

Lecture 1

Agata Półrola
<agata.polrola@wmii.uni.lodz.pl>

Literature

Bruce Eckel: *Thinking in C++. Second edition. Vol. 1. (http://mindview.net/Books/TICPP/ThinkingInCPP2e.html)*

Robert Lafore: *Object-Oriented Programming in C++.*

Nicolai Josuttis: *Object-Oriented Programming in C++.*

Stanley Lippman, Josée Lajoie, Barbara Moo: *C++ Primer. Forth edition.*

Bjarne Stroustrup: *Language C++.*

Stephen Prata: *C++ Primer Plus.*

Michael Ben-Ari: *Understanding programming languages.*

http://cppreference.com

http://www.cplusplus.com

Programming paradigms

- declarative programming (tell the computer what do you want)
- imperative programming (tell the computer how to do it)
- procedural programming (bottom-up design):
  - global data structures with free access (risky)
  - many loose functions building a program
- object programming (top-down design):
  - protected & separated data structures (black-box solution)
  - functions operating on small parts of data (encapsulation)
- generic programming (static polymorphism)
- object-oriented programming (object programming + dynamic polymorphism)
- modular programming, event programming, functional programming, ...

Object-oriented paradigm

Four fundamentals of the object-oriented programming (OOP) :

- **Abstraction** (abstract data types, ADTs):
  - classes as the abstract and simplified model of real beings (structural and behavioral), objects as instances of classes
- **Encapsulation** (information enclosing and hiding):
  - functions tied together with data (methods)
  - data protection (public methods, sending messages, object interface)
- **Inheritance** (code reuse, interface extensions):
  - building specialised objects based on general ones
- **Polymorphism** (object-orientedness):
  - single abstract interface – multiple different forms of inherited objects

Advantages of object-orientedness

- structural and behavioral design similar to the human's thinking of the reality
- increased code safety and reuse (data protection and inheritance)
- a code is easier to write and (the most important) it is far easier to read
- clear modular code organisation (encapsulation)
- the encapsulation prevents function name clashes
- the encapsulation prevents from «giving a monkey a razor»
- better team support with the code separation
- proper initialisation and cleanup (constructors, destructors)
- abstract, perspective and dynamic programming (late binding, polymorphism)
- better code stability by the exception handling (exception inheritance)

History of object-orientedness

- 1960s – Simula 67 : classes, static instances, ship simulations
- 1971 – Smalltalk : dynamic objects created «on-the-fly», object-orientedness
- 1983 – C++ : the extension of the procedural language C
- 1980s : extensions of other existing procedural languages (Ada, Fortran, Pascal , Basic, Eiffel, Lisp, Perl, OCaml)
- 1991 – Java : C++ based language for virtual machine with garbage collector (cross-platform)
- 1990s : primarily object-oriented, interpreted, dynamic-typed languages (Python, Ruby), object-oriented scripting languages (JavaScript, PHP)
- 2002 - .NET framework : cross-language inheritance (C#, VB.NET, J#)
- 2000s : newer object-oriented languages (Swift, Kotlin, TypeScript, ...)
- 2020 – C++20 : the newest standard of the C++ language

Classes and objects

Structural approach (without the encapsulation):

```
struct Obj //static data structure
{
  int a, b;
};

void set(Obj &o, int _a, int _b) //global function(& reference)
{
  o.a = _a;
  o.b = _b;
}

Obj x; //variable
set(x, 1, 2); //function call
```

Classes and objects

Object approach (the encapsulation):

```
struct Obj //abstract data type (the class)
{
  int a, b; //fields

  void set(int _a, int _b) //method
  {
    a = _a;
    b = _b;
  }
};

Obj x; //the object (an instance of the class)
x.set(1, 2); //message to the object
```

Interface and implementation

File obj.h (abstract type interface):

```
struct Obj //abstract data type (the class)
{
  int a, b; //fields
  void set(int _a, int _b); //method prototype
};
```

File obj.cpp (abstract type implementation):

```
#include ”obj.h”
void Obj::set(int _a, int _b) //method body (:: scope operator)
{
  a = _a;  b = _b;
}
```

File main.cpp:

```
#include ”obj.h”
int main()
{
  Obj x; //the object (an instance of the class)
  x.set(1, 2); //message to the object
}
```