# Advanced Programming in C++

فرشاد حکیم پور



## Hello World

```
#include <iostream>
int main()
{
  std::cout << "Hello......." << std::endl;
  return 0;
}
```



## Functions

```
#include <iostream>
#include <cmath>

double triangleArea(double a, double b, double c)
{
  double k = (a + b + c)/2;
  return sqrt(k * (k - a) * (k - b) * (k - c));
}

int main()
{
  std::cout << triangleArea(3, 4, 5) << std::endl;
  return 0;
}
```



## Functions

- Problems are decomposed to (smaller) sub-problems
- A function is used to write code to solve a sub-problem
- Function *call* = function *invocation*
- Parameter passing: *call-by-value*



## main()

- "`main()`" is the starting point for execution
- It returns an integer value
- It returns zero for normal completion



## Object Oriented Programming

- Everything is an object
- A program is a bunch of objects telling each other what to do by sending messages
- Each object has its own memory made up of other objects (to represent the state of the object)
- Every object has a type (*class*)
- All objects of a particular type can receive the same messages (method calls)

## Objects

```
#include <iostream>
#include <cmath>

class triangle{
private:
   double a, b, c;
public:
   triangle(double _a, double _b, double _c)
   {   a = _a; b = _b; c = _c;  }
   double area(){
       double k = (a + b + c)/2;
       return sqrt(k * (k - a) * (k - b) * (k - c));
   }
};
int main(){
   triangle my_shape = triangle(3,4,5);
   std::cout << my_shape.area() << std::endl;
   return 0;
}
```

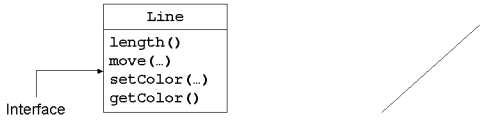
7

## Creating an object

- For *static storage allocation* simply call the constructor (memory allocated right at the start of the execution).
- Later on we will have *dynamic storage allocation* by "**`new`**" and "**`delete`**"

8

## Class




9

## Encapsulation

- Hiding implementation from client developer.
- Only the interface of the classes in software libraries are exposed.
- Client programmer sees what is important for him right away.
- Class creator can change the implementation without the client knowledge.

10

## Access Specifiers

- Access Specifiers apply to both identifiers and function definitions:
  - Public: accessible by all objects
  - Private: accessible only within the object class
  - Protected: accessible by the object class and inheriting object classes

11

## Objects

```
#include <iostream>
#include <cmath>

class triangle{
private:
   double a, b, c;
public:
   triangle(double _a, double _b, double _c)
   {   a = _a; b = _b; c = _c;  }
   double area(){
       double k = (a + b + c)/2;
       return sqrt(k * (k - a) * (k - b) * (k - c));
   }
};
int main(){
   triangle my_shape = triangle(3,4,5);
   std::cout << my_shape.area() << std::endl;  return
   0;
}
```

12

## Reuse

- Reuse is an important advantage of O.O. programming.

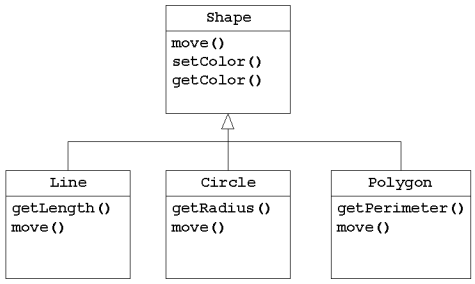
13

## Inheritance (Reuse)

- Subclass/Super-class
- Specialization/Generalization
- “is-a” relation(ship)
- subsumption
- Different type of inheritance

14

## Interface Reuse




15

## Aggregation (Reuse)

- Composing a class from several classes.
- Part-Whole, Composition, Consists of, Containing
- Treated as a type of association

16

## Polymorphism

- A method that can be applied to values of different types is known as a *polymorphic* function.
- O-O uses *ad-hoc* polymorphism
- In *parametric* polymorphism methods handle values identically without depending on their type.

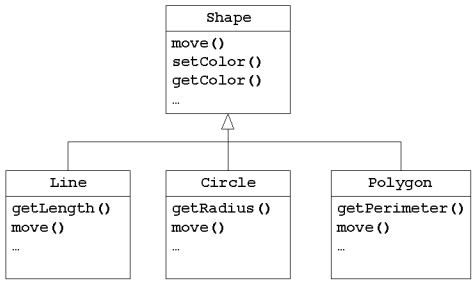
17

## Method Overloading

- If a method call is coded to a super-class and later on the class have a new subclass, no changes are needed.

18

## Interface Reuse




19

## OO Summary

- Encapsulation
- Inheritance
- Polymorphism
- Method overloading (supports ad-hoc polymorphism)

20

## C++ Fundamental (or Basic or Native or Atomic) Types

- Numbers
  - fixed-point
    - "`int`" minimum two bytes in Standard C but four bytes in C++ compilers (MVC and gcc)
  - floating-point
    - "`float`" 4 bytes, single precision
    - "`double`" 8 bytes, double precision

21

## C++ Fundamental (or Basic or Native or Atomic) Types

- Boolean
  - "`bool`" (one byte)
- Character
  - "`char`" (one byte)

22

## Short and Long Specifiers

- Fixed-point
  - "`short int`" (`short`) two bytes
  - "`long int`" (`long`) four bytes (in MVC & gcc)
- Floating-point
  - `short` specifier is not applicable, at all.
  - `long float` is not allowed (error in gcc) but some systems (such as MVC) interpret it as `double`
  - Not really useful apart from `long double` (in gcc 12 bytes) and even that is not for MVC

23

## Signed and Unsigned Specifiers

- Only applicable to fixed-point numbers and charachters
  - "signed short int" -32767 to +32768
  - "unsigned short int" 0 to 65535
  - "signed int" -2,147,483,647 to 2,147,483,648
  - "unsigned int" 0 to 4,294,967,295

24