Java™ Education & Technology Services

Object Oriented programming Using C++

Introduction

Real World

Computer World

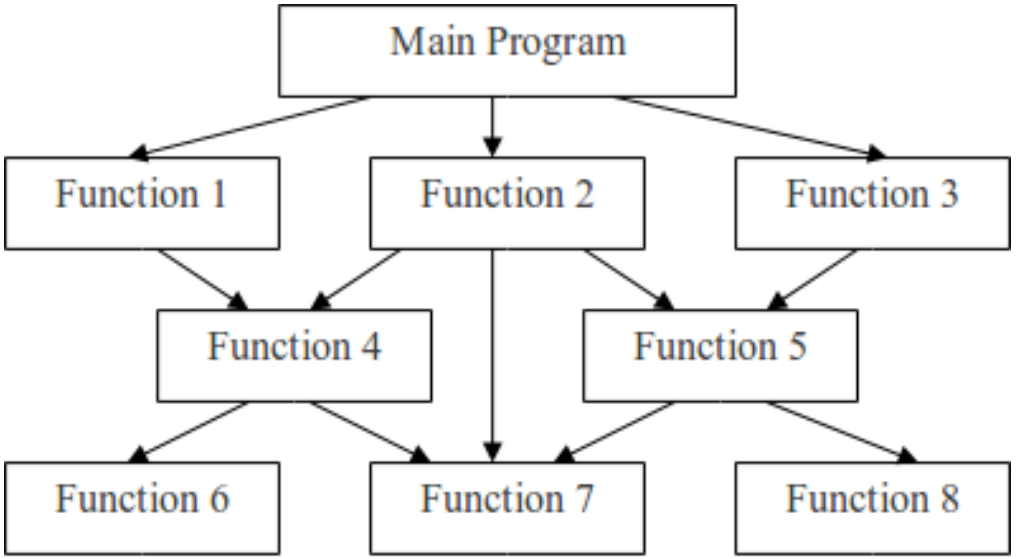
Problem

Program

- Solve &Transfer
- Transfer To solve

Structure programming

- Transfer the problem to set of functions.
- The main function will handle the communication among them.



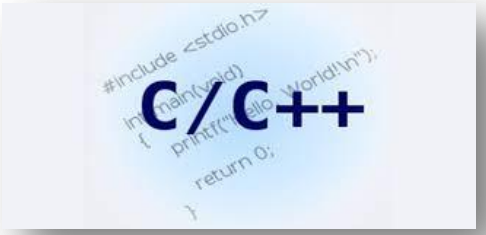
- Problems Here :
  - Transformation
  - Reusability
  - Maintainability

How to use?

How to make?

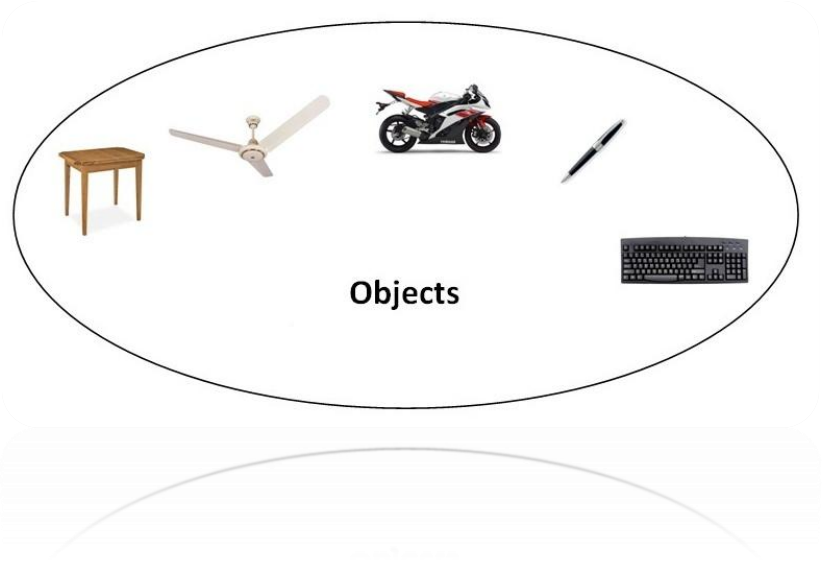
OOP programming

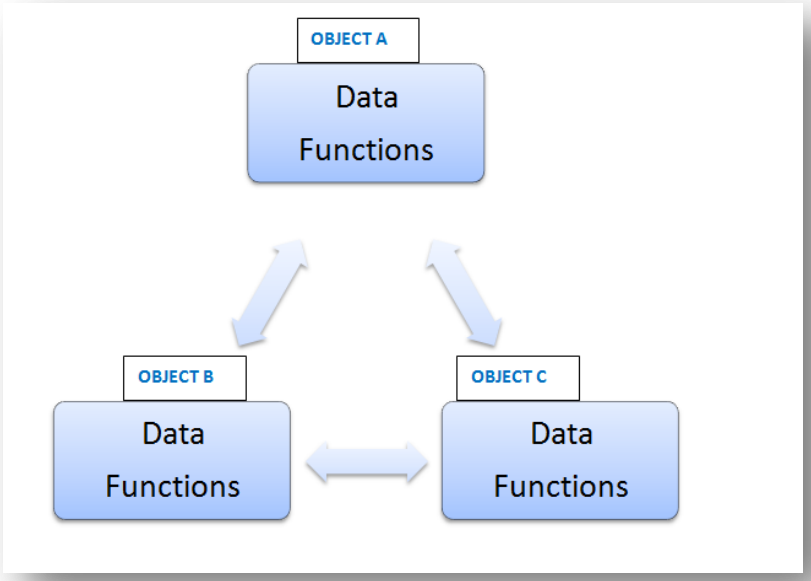
- To solve structure programming problems we will use OOP.
- C language is the development of B language.
- Object C is not fully OOP.
- Small talk is very restricted OOP (not popular).
- C++ is one step after C [1980- used in 1987].



Why OOP programming?

- How to solve the problem of transformation?
  - Convert the problem to set of Units – Objects instead of function.
  - Object is a s et of attributes (Data) and set of functions (behavior).
  - These objects communicate by their functions.





Why OOP programming?

– **Example**

PERSON

Attributes

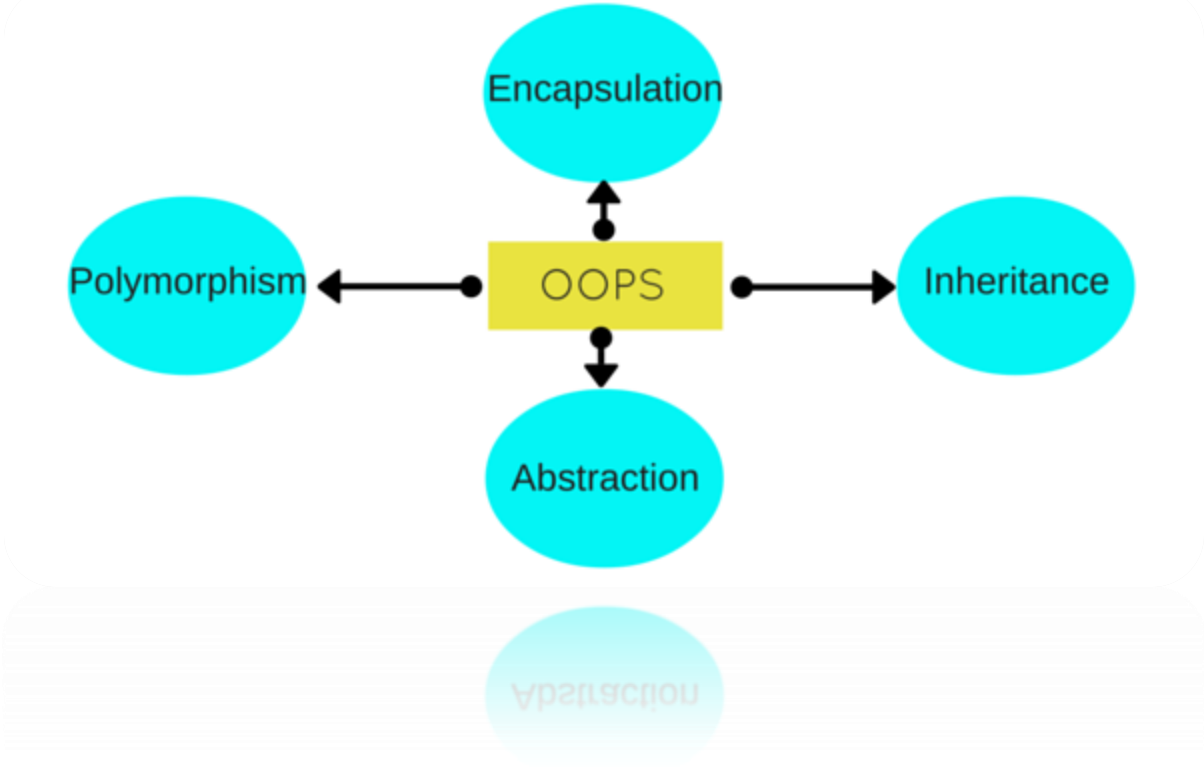
- First name
- Last name
- Mail address
- Birth date

Methods

- Send a mail
- Ask if the person is over 18

OOP programming

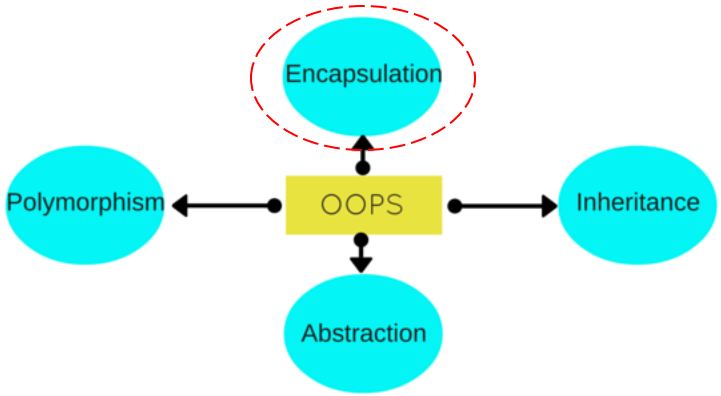
– How to solve the problem of Reusability?



OOP programming

1. Encapsulation :

- Functions cover attributes.
- There is **No** common variables.
- Nothing depends on another thing.



PERSON

Attributes

- First name
- Last name
- Mail address
- Birth date

Methods

- Send a mail
- Ask if the person is over 18

- First name = “Ali”
- Obj.SetFirstName(“Ali”) √

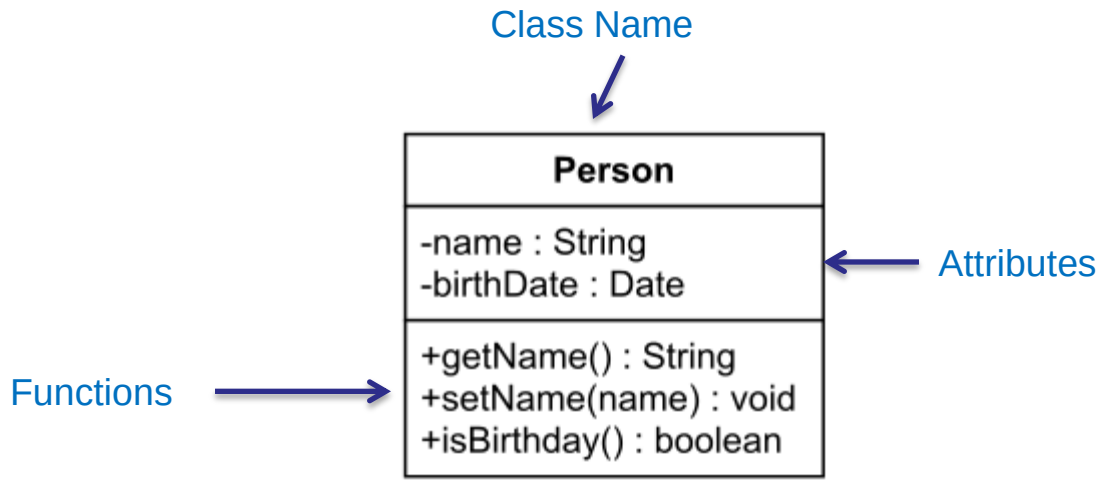
OOP programming

1. Encapsulation :

- **Class:**
  - Design of objects.
  - Template to create object.
  - Object factory.
- **Object:**
  - is real image of the class.
  - Can make many objects from the same class.
  - Difference among objects from the same class is different attributes values.

1. Encapsulation :

- **Class Vs Object**



Obj 1

---

name = “Ali”
birthDate=1990

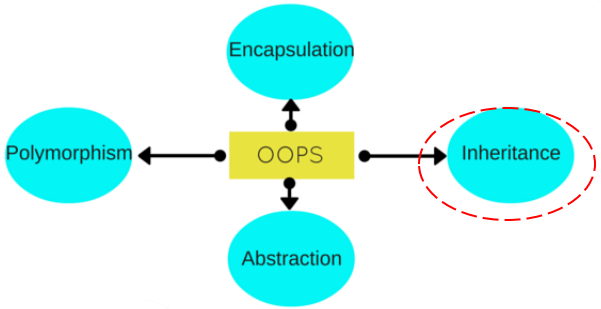
Obj 2

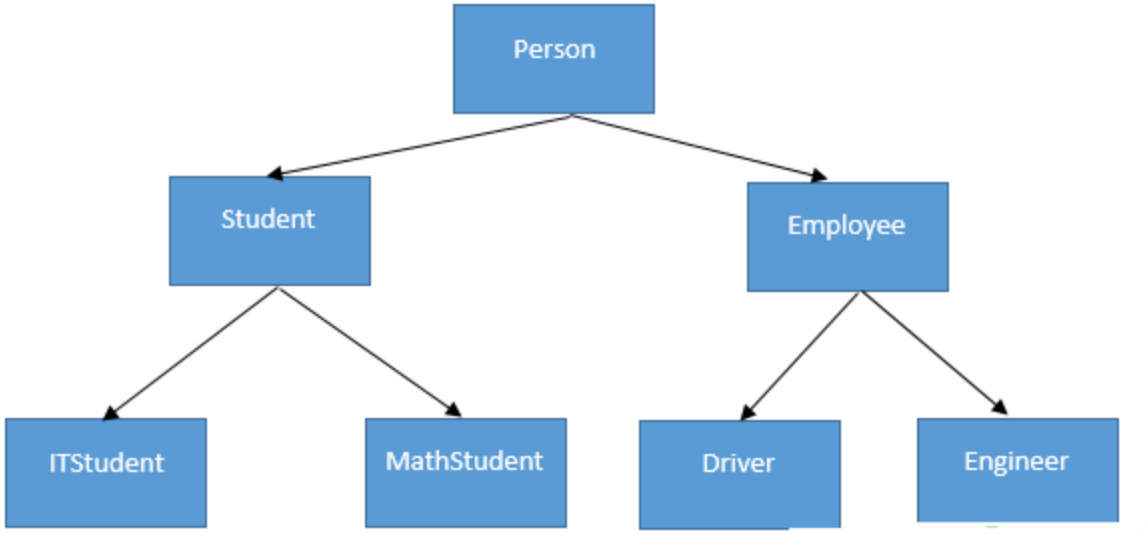
---

name = “Joe”
birthDate=1980

2. Inheritance:

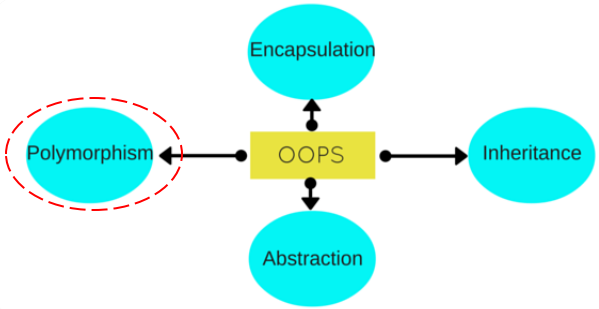
- Gives more usability and maintainability.





3. Polymorphism:

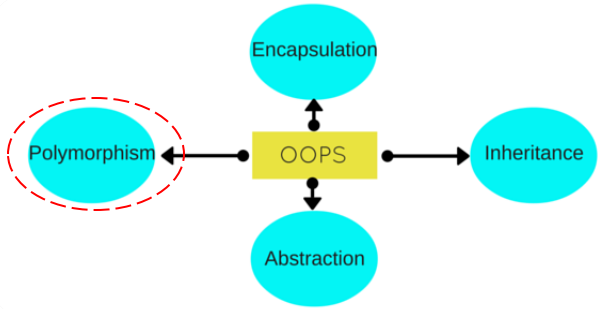
- Functions with the same name
- but with different parameters & body.



C	C++
Abs(int)	Abs(int)
LAbs(Long)	Abs(long)
FAbs(float)	Abs(float)

3. Polymorphism:

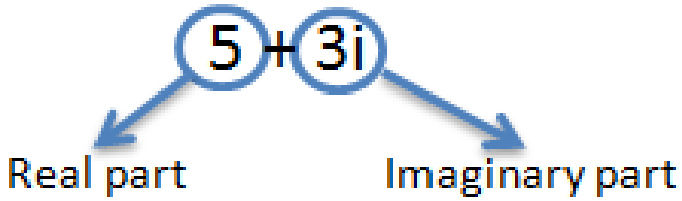
- **Function Signature:**
  1. Name
  2. No. of parameters.
  3. Type of parameters.
  4. Order of Them.



float setSalary (float x) { …….. }

– Function return type is not a part of its signature ☺

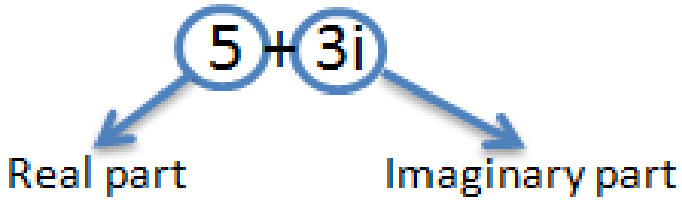
- Complex Number:
  - √ -ve
  - UML



Complex
- real : float - img : float
+ getReal () : float + getImg () : float + setReal (float) : void + setImg (float) : void + printComplext() : void

– Complex Number:

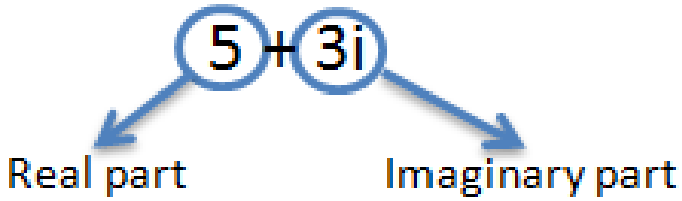
- √ -ve
- In **C +** using Class and member functions:




```
class Complex
{
    float real;
    float imag;
public:
    void setReal(float) ;
    void setImag(float) ;
    float getReal() ;
    float getImag() ;
    void print();
};
```

Functions Prototype

- Complex Number:
  - √ -ve
  - In **C +** using Class and member functions:




```
void Complex::setImag(float i)
{
    imag = i ;
}

float Complex::getReal()
{
    return real ;
}

float Complex::getImag()
{
    return imag ;
}

void Complex::print()
{
    if(imag<0)
    {
        cout<<real<<" - "<<fabs(imag)<<"i"<<endl;
    }
```

Lab Exercise

Lab Exercise

- **1st Assignment :**

1. Implement Complex class with :
    1. Setters , getters and print functions as members functions.
    2. And "add" and "subtract" as stand alone functions.

2. Implement Swap function:
    1. Once call by value.
    2. Once call by Address
    3. And call by Reference.

```
int main() {
    Complex myComp1, myComp2, resultComp ;
    // Read Real & Img parts For myComp1 & myComp2 from the user
    myComp1.print() ;
    myComp2.print() ;

    resultComp = add(myComp1, myComp2) ;
    resultComp.print() ;

    resultComp = subtract(myComp1, myComp2) ;
    resultComp.print() ;

    return 0;
}
```