Object Oriented Programming:

It is basically a methodology like other programming methodologies e.g structure programming, modular programming, procedural programming or aspect oriented programming etc but it is one of the most famous and well known paradigm of programing language. All the latest languages supports OOP. It is more related to internal designing of a software. It works with the programmer of a software.

In object oriented programming the application development become more systematic. The flow of oop is from engineering to development of a software. Let us understand it through

some analogy like before constructing a house a civil engineer will design it's architect (Blue print) and after that start construction same is the case with software development Object Oriented programming is deals with the infrastructure or design of the software. It is totally like how you see or understand a system.

C++, Java, Python,Perl, Ruby ,Golang etc and many other languages support oop.

Principles of Object Oriented Programming :

- **Abstraction:**
- **Encapsulation:**
- **Inheritance:**
- **Polymorphism:**

Simple OOP program:

```
#include <iostream>

using namespace std;

class Rectangle{

  public:

  int length;

  int breadth;

  int area(){

     return length*breadth;
```

```
  }
  int perimeter(){
     return 2*(length+breadth);
  }
};

int main()
{
   Rectangle r1;
   r1.breadth=10;
   r1.length=20;
   cout<<"Area is "<<r1.area()<<endl;
   cout<<"Perimeter is "<<r1.perimeter();



}
```

←----------→

Pointer to objects:

We can use pointer in-place of objects while accessing data-members and member functions. The basic syntax is (->) using arrow operator instead of dot operator.

Q: Why do we need Constructors?

When an Object of a class is created the data-members of a class has garbage before initialization or before calling set functions for that data members. In order to deal with that situation a constructor is created to solve this issue. A Constructor will automatically initialize the members when an object is created.

Types of functions mostly used in a Class:

- **Constructor:**
  Example:
  Rectangle();
  Rectangle(int l, int b);
  Rectangle(Rectangle &r);

- **Mutator:**
  Void setlength();
  Void setBreadth();
- **Accessor:**
  Int getlength();
  Int getbreadth();
- **Facilitator:**
  Int area();

Int perimeter();
Int isSquare();

- **Enquiry:**

  isSquare();

  isEmpty();

- **Destructor:**

  ~Rectangle();

Scope resolution Operator:

Example:

```
#include <iostream>

using namespace std;
class Rectangle{
   private:
   int length;
   int breadth;
   public:
   int area();
   int perimeter();
   Rectangle(int l, int b){
      length=l;
```

```
        breadth=b;
    }



};
int Rectangle::area(){
    return length*breadth;
}
int Rectangle::perimeter(){
    return 2*length*breadth;
}

int main()
{
 Rectangle r(10,20);
 cout<<"Area is "<<r.area();
}
```

Operator Overloading:

Operator overloading is like a giving function an english name instead of using words like for add function use named the function add **e.g**

```
Int add(int t, int u){
Return t+u;
}
Int operator+(int t, int u){
Return t+u;
}
```

Here is a short program for the brief explanation of operator overloading :

```
#include <iostream>

using namespace std;
class Complex{
   public:
   int real;
   int imaginary;

   Complex add(Complex c){
      Complex temp;
      temp.real=real + c.real;
      temp.imaginary= imaginary + c.imaginary;
```

```
        return temp;


    }
};
int main()
{
  Complex c1, c2, c3;
  c1.real=5; c1.imaginary=3;
  c2.real=10; c2.imaginary=5;
  c3=c1.add(c2);
  cout<<c3.real<<"+"<<"i"<<c3.imaginary;
}
```

Inheritance:

Acquiring the features of existing class into a new class(deriving a new class from existing class).