Object Oriented Programming using CPP

Lecture 1

Presented By

Dr. Sunirmal Khatua

Visvesvaraya Young Faculty Fellow, Govt. of India
Assistant Professor, University of Calcutta

CPP Programming

- C++ is a statically typed, compiled, general-purpose, case-sensitive programming language that supports procedural and object-oriented.
- Like C Language, C++ is also a Middle-Level Language.
- C++ is a superset of C, and any legal C program is also a legal C++ program
- C++ fully supports object-oriented programming, including the four pillars of object-oriented development:
  - Encapsulation
  - Data Hiding
  - Inheritance
  - Polymorphism

Hello World in C++ Programming

```c
#include<stdio.h>
int main()
{
    printf("Hello World!");
}
```

```c
#include <iostream>
using namespace std;
int main(){
     cout <<"Hello World";
}
```

```c
using namespace first_space;
int main ()
{
  func();
  return 0;
}
```

```c
#include <iostream>
// first name space
namespace first_space
{
  void func()
  {
     std::cout << "Inside first_space" << endl;
  }
}
```

```c
// second name space
namespace second_space
{
  void func()
  {
     std::cout << "Inside second_space" << endl;
  }
}
```

Classes and Objects

```
#include <iostream>
using namespace std;

class Box{
    public:
        double length;
        double breadth;
        double height;
        double getVolume(){
            return length*breadth*height;
        }
};

int main()
{
    Box b1;
    b1.length = 10;
    b1.breadth = 10;
    b1.height = 10;
    cout << "Volume="<<b1.length*b1.breadth* b1.height);
}
```

Classes and Scope Resolution

We can define a method outside the Class using Scope Resolution Operator(::)

```
class Box{
     public:
          double length;
          double breadth;
          double height;

          double getVolume(){
               return length*breadth*height;
          }
};
```

```
class Box{
     public:
          double length;
          double breadth;
          double height;

          double getVolume();
};

double Box::getVolume(){
     return length*breadth*height;
}
```

Constructor and Destructor

- A special method having the following properties:
  - Same name as the class
  - Doesn't have any return type
  - Used to initialize the object
  - Every class must have a constructor. If we missed one, the runtime environment will provide a default constructor automatically.

- Types of Constructor
  - Default Constructor
  - Parameterized Constructor
  - Copy Constructor

```
int main()
{
    Box b1(20,20,20);
    cout << "Volume="<<b1.getVolume();
}
```

```
class Box{
    public:
        double length;
        double breadth;
        double height;

        Box(double l, double b, double h){
            length = l;
            breadth = b;
            height = h;
        }

        double getVolume(){
            return length*breadth*height;
        }
};
```

Note : If you write any kind of constructor, then the runtime environment will not provide the default constructor automatically

Data Hiding

- Data Hiding is an Object Oriented Feature that controls the visibility of the class's member variables and member functions.
- Access Specifier is used to implement Data Hiding .
- There are 3 access specifiers as listed below.

```
class ClassName{
      public:
            //Public Members
      protected:
            //Protected Members
      private:
            //Private Members

}
```

Accessible everywhere

Accessible within the class as well as in derived / subclasses

Accessible only within the Class

Private is the default access specifier

Visibility through Friend Functions

- A **friend function** is a function that is not a member of a class but has access to the class's private and protected members.

```
class ClassName {
  private:
    int privateVar;
  public:
    ClassName(int val) {
      privateVar = val;
    }

    // Declare a friend function
    friend void display(const ClassName& obj);
};
```

```
int main() {
    ClassName obj(42);
    display(obj); // Calling the friend
function
    return 0;
}
```

```
void display(const ClassName& obj) {
    // Accessing private member
    cout << "Private Variable: " << obj.privateVar << endl;
}
```

Visibility through Friend Functions

- A **friend function** can be a friend of multiple classes

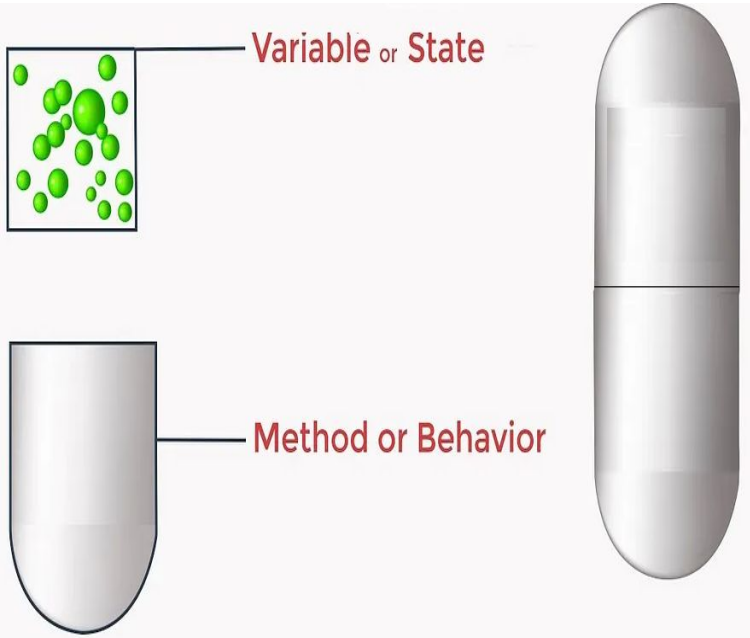
```
class ClassA {
private:
    int a;

public:
    ClassA(int val) : a(val) {}

    // Declare friend function
    friend int sum(const ClassA& objA, const ClassB& objB);
};
```

```
class ClassB {
private:
    int b;

public:
    ClassB(int val) : b(val) {}

    // Declare friend function
    friend int sum(const ClassA& objA, const ClassB& objB);
};
```

```
int sum(const ClassA& objA, const ClassB& objB) {
    return objA.a + objB.b; // Access private members of both classes
}
```

Encapsulation

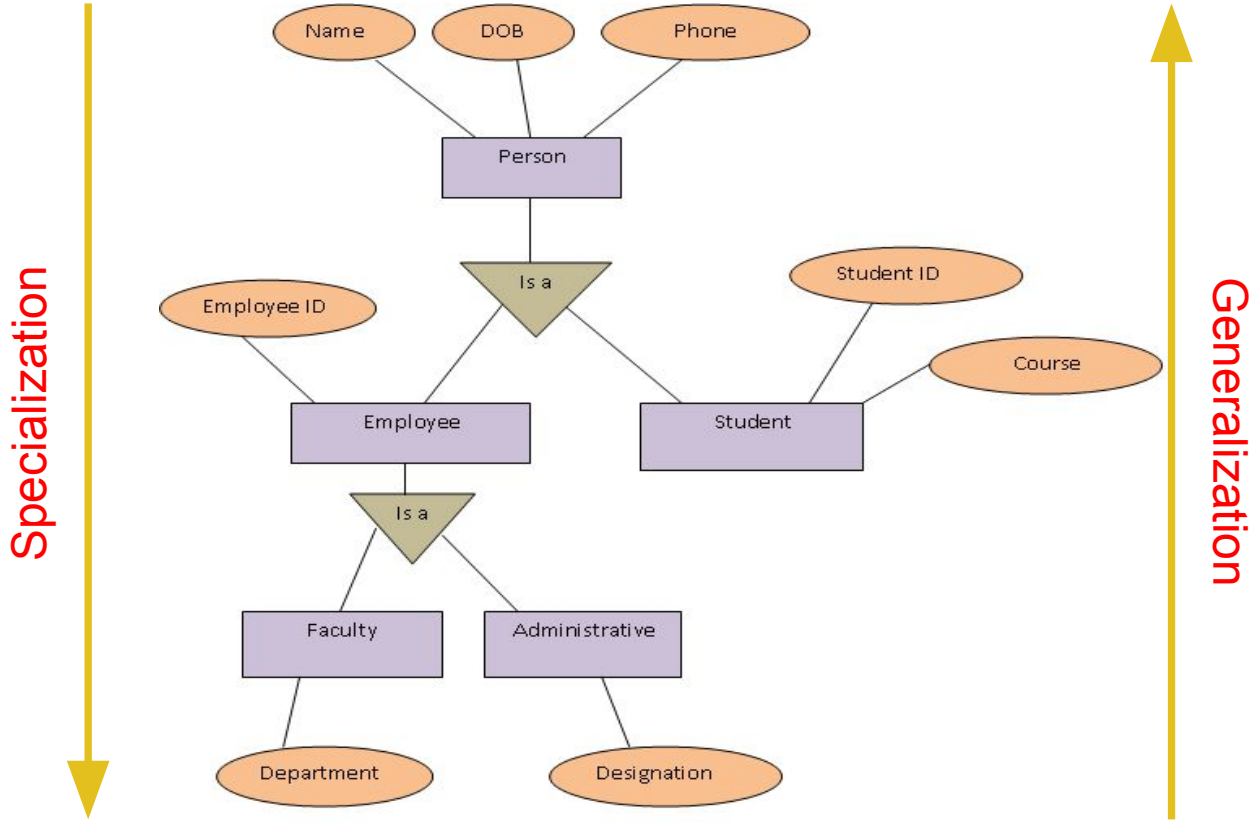
- Combining data and the methods that manipulate it into a single entity is referred to as Encapsulation.
- It also implement Data Hiding




```
class ClassName{
      private:
            int marks;
      public:
            double setMarks(int m){
                  marks = m;
            }

       double getMarks(){
                  return marks;
            }
}
```

Inheritance

Inheritance allows a class to reuse code from another class.




```
class Person{
     protected:
          string name,dob,phone;
};
class Employee:Protected Person{
     protected:
          int employeeId;
};

class Faculty:Employee{Emplyoee{
     private:
          string department;
};
```

Base Class Access Specifier	Public Inheritance	Protected Inheritance	Private Inheritance
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not Accessible	Not Accessible	Not Accessible

Polymorphism : Looks similar behaves differently

- Function Overloading allows same name for two different methods within the same class.

- The overloaded methods must differ in signature.

- Signature is identified by:
  - *No. of arguments*
  - *Types of arguments*
  - *Order of arguments*

- It implements compile time polymorphism

Note : Return Type is not a part of Signature

```
class Calculator{
      public:
            int add(int a, int b){
                  return a+b;
            }
            int add(int a, int b, int c){
                  return a+b+c;
            }
};

int main()
{
      Calculator c;
      cout<<"10+20="<<c.add(10,20)<<endl;
      cout<<"10+20+30="<<c.add(10,20,30)<<endl;
}
```

Polymorphism : Looks similar behaves differently

- Operator Overloading allows an user to redefine the behavior of an operator.

```
class Box{
      public:
           int length, breadth, height;
           Box(){
            length=0;breadth=0;height=0;
           }
           Box(int l, int b, int h){
            length = l;breadth = b; height = h;
           }

           Box operator+(Box b){
              Box temp;
              temp.length = length+b.length;
              temp.height = height+b.height;
              temp.breadth = breadth+b.height;
              return temp;
           }
};
```

```
int main()
{
      Box b1(10,10,10);
      Box b2(20,20,20);
      Box b3 = b1+b2;
      cout<<b3.length;
}`
```

Polymorphism : Looks similar behaves differently

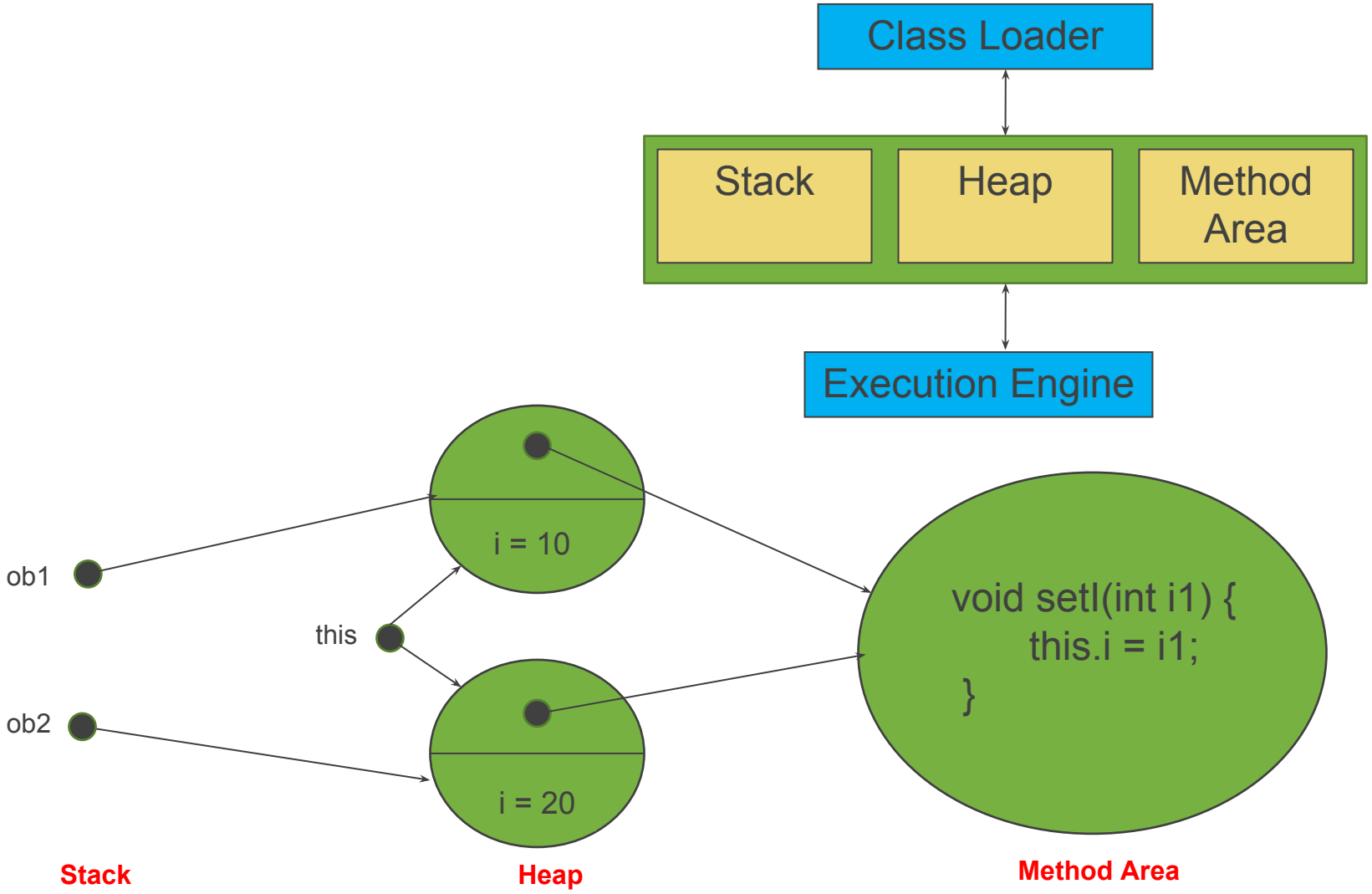
- Function Overriding allows a sub class to write a method with the same signature of a method in the super class.

```
class Animal{
      public:
            void types();
};

class Goat:Animal{
      public:
         void type(){
           cout<<"Herbivorous Animal."<<endl;
         }
};
class Tiger:Animal{
      public:
         void type(){
           cout<<"Carnivorous Animal."<<endl;
         }
};
```

```
int main()
{
      Goat g = Goat();
      g.types();
      Tiger t = Tiger();
      t.types();
}
```

In-Memory Structure for Objects

```
class C {
    int i;
    public:
    void setI(int i1){
          i = i1;
     }
};

int main(){
    C ob1;
    ob1.setI(10);
    C ob2;
    ob2.setI(20);
}
```



Thank You