📘 OBJECT ORIENTED PROGRAMMING

WEEK 2: Class, Object, Encapsulation, Simple UML

---

1 What is Object Oriented Programming (OOP)?

Object Oriented Programming is a programming style where we organize code using **classes and objects**.

Instead of writing everything in functions like C language, in C++ we model real-world things as objects.

🧠 Real-Life Idea

Think about:

- Car
- Student
- Bank Account
- Mobile Phone

Each has:

- **Properties (Data)**
- **Behaviors (Functions)**

OOP helps us represent these things in code.

---

2 CLASS

🔷 Definition

A **class** is a blueprint or template used to create objects.

👉 It does NOT occupy memory.
👉 It only defines structure.

🔷 Syntax of Class

```
class ClassName
{
private:
    // data members

public:
    // member functions
};
```

🔷 Example 1: Student Class

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

class Student
{
public:
    string name;
    int age;

    void display()
    {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};
```

Here:

- `name` and `age` → Data members
- `display()` → Member function

---

3 OBJECT

🔷 Definition

An **object** is a real instance of a class.

👉 Object occupies memory.
👉 It uses class structure.

🔷 Creating Object

```
Student s1;
```

🔷 Complete Example

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

class Student
{
public:
    string name;
    int age;

    void display()
    {
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

int main()
{
    Student s1;      // object created

    s1.name = "Ali";
    s1.age = 20;

    s1.display();

    return 0;
}
```

🔎 Output:

```
Name: Ali
Age: 20
```

---

4 ACCESS SPECIFIERS

C++ provides 3 access specifiers:

Access Specifier	Meaning
public	Accessible everywhere
private	Accessible only inside class
protected	Used in inheritance

---

5 ENCAPSULATION

🔷 Definition

Encapsulation means:

👉 Hide data
👉 Allow access through functions

In simple words:
We protect our data.

🔷 Why Encapsulation is Important?

Without encapsulation:

- Anyone can change data
- Data can become invalid

With encapsulation:

- We control how data changes

🔷 Example Without Encapsulation (Bad Practice)

```
class BankAccount
{
public:
    double balance;
};
```

Anyone can do:

```
account.balance = -5000;   // Wrong!
```

🔷 Example With Encapsulation (Correct Way)

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

class BankAccount
{
private:
    double balance;

public:
    void setBalance(double b)
    {
        if(b >= 0)
            balance = b;
        else
            cout << "Invalid Balance!" << endl;
    }

    double getBalance()
    {
        return balance;
    }
};

int main()
{
    BankAccount acc;

    acc.setBalance(5000);
    cout << "Balance: " << acc.getBalance() << endl;

    return 0;
}
```

🔎 Output:

```
Balance: 5000
```

✔ Data is safe
✔ Invalid values prevented

That is **Encapsulation**.

---

6 REAL LIFE EXAMPLE (Car)

```
class Car
{
private:
    int speed;

public:
    void setSpeed(int s)
    {
        if(s >= 0 && s <= 200)
            speed = s;
        else
            cout << "Invalid Speed!" << endl;
    }

    void showSpeed()
    {
        cout << "Car Speed: " << speed << endl;
    }
};
```

We protect speed from invalid values.

---

7 UML CLASS DIAGRAM

UML = Unified Modeling Language

It is a visual way to represent class structure.

🔷 Basic UML Structure

```
-----------------
|    Student    |
-----------------
| - name: string|
| - age: int    |
-----------------
| + display()   |
-----------------
```

Symbols Meaning:

Symbol	Meaning
-	private
+	public

🔷 UML of BankAccount

```
-------------------------
|     BankAccount       |
-------------------------
| - balance: double     |
-------------------------
| + setBalance(double)  |
| + getBalance():double |
-------------------------
```

8 Difference Between Class and Object

Class	Object
Blueprint	Real instance
No memory allocated	Memory allocated
Logical entity	Physical entity

9 Memory Concept

When you create object:

```
Student s1;
```

Memory is allocated for:

- s1.name
- s1.age

But functions are shared.

🔟 Multiple Objects Example

```
Student s1, s2;

s1.name = "Ali";
s1.age = 20;

s2.name = "Sara";
s2.age = 22;
```

Each object has separate data.

11 Mini Practice Example

Example: Rectangle

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

class Rectangle
{
private:
    double length;
    double width;

public:
    void setValues(double l, double w)
    {
        length = l;
        width = w;
    }

    double area()
    {
        return length * width;
    }
};

int main()
{
    Rectangle r1;

    r1.setValues(5, 3);

    cout << "Area: " << r1.area() << endl;

    return 0;
}
```

🏁 Summary

- Class → Blueprint
- Object → Instance
- Encapsulation → Data hiding
- UML → Visual representation
- Access Specifiers → Control access