Advance Programming Techniques (APT)

Lecture # 6

Ehtisham Rasheed

Department of Computer Science
University of Gurjat, Gujrat

Object Oriented Programming

- Object Oriented Programming (OOP) is a strategy that provides some principles for developing software
- It is a methodology and other methodologies also exist e.g. Procedural Programming or Modular Programming
- OOP is the latest methodology
- Nowadays, almost all languages support OOP
- OOP vs Modular Programming

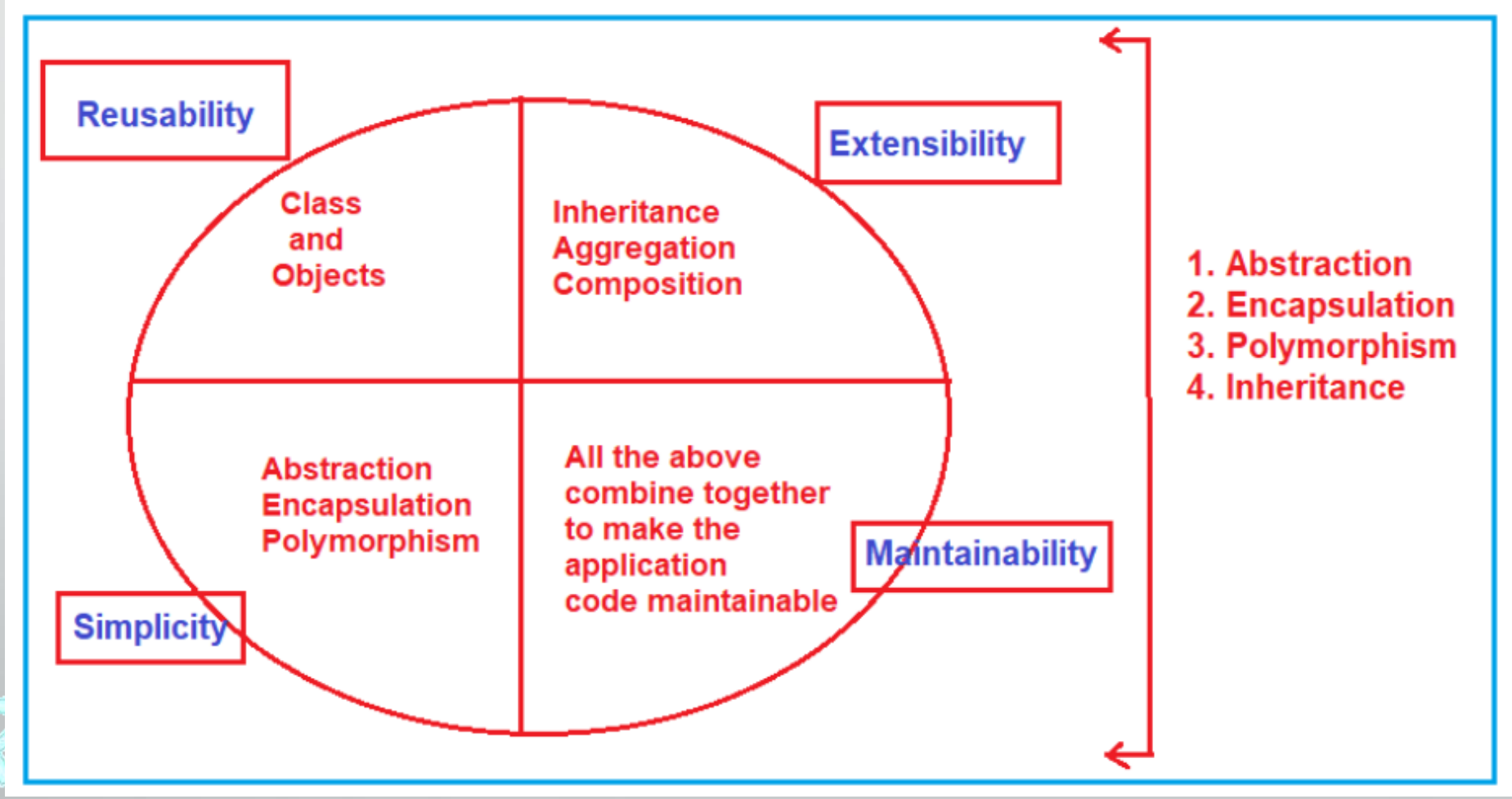
Problems with Modular Programming

- Reusability
  - We must write same code at multiple places, increasing code duplication
- Extensibility
  - It is not possible in modular programming to extend the features of a function. We've to write a new function if we need changes
- Simplicity
  - As extensibility and reusability are impossible in Modular Programming, we usually end up with many functions and scattered code
- Maintainability
  - As we don't have Reusability, Extensibility, and Simplicity in modular Programming, it is very difficult to manage and maintain the application code

Object Oriented Programming

- Object Oriented Programming in C# is a design approach where we think in terms of real-world objects rather than functions or methods
- Unlike procedural programming, in OOP, programs are organized around objects and data rather than action and logic
- Please see the diagram on next page to understand this better

Object Oriented Programming



Object Oriented Programming

- Reusability
  - Rather than copy and pate function code, we create objects from classes and can reuse it whenever we need
- Extensibility
  - Extensibility in OOP is addressed using Inheritance, Aggregation and Composition
- Simplicity
  - In modular programming, we end up with lots of functions and scattered code, and from anywhere we can access the functions, security is less. In OOPs, this problem is addressed using Abstraction, Encapsulation, and Polymorphism
- Maintainability
  - As OOPs address Reusability, Extensibility, and Simplicity, we have good, maintainable, and clean code

Object Oriented Programming Principles

- OOP provides four principles

1. **Abstraction**
2. **Encapsulation**
3. **Inheritance**
4. **Polymorphism**

- **Note:** Don't consider Class and Objects as OOPs principles. We use classes and objects to implement OOP principles

Abstraction and Encapsulation

- The process of representing the essential features without including the background details is called **Abstraction**
  - it is a process of defining a class by providing necessary details to the external world
- The process of binding the data and functions together into a single unit (i.e., class) is called **Encapsulation**
  - it is a process of defining a class by hiding its internal data members from outside the class and accessing those internal data members only through publicly exposed methods or properties
  - Data encapsulation is also called data hiding
  - Abstraction and Encapsulation are related to each other. We can say that Abstraction is logical thinking, whereas Encapsulation is its physical implementation

Inheritance

- The process by which the members of one class are transferred to another class is called **inheritance**
- The class from which the members are transferred is called the Parent/Base/Superclass
- The class that inherits the members is called the Derived/Child/Subclass
- We can achieve code **extensibility** through inheritance

Polymorphism

- Polymorphism is derived from the Greek word, where Poly means many and morph means shapes/faces/ behaviors
- So, the word polymorphism means the ability to take more than one form
- Technically, we can say that the same function/operator will show different behaviors by taking different types of values or with a different number of values, called Polymorphism
- There are two types of polymorphism
  - Static polymorphism/compile-time polymorphism/early binding
    - Function overloading and operator overloading
  - Dynamic polymorphism/run-time polymorphism/late binding
    - Function overriding

Class and Objects

- A class is a user-defined data type that represents both state and behavior
  - State represents properties and behavior is the action that object can perform
  - In other words, class is a template that describes the details of an object
- An object is an instance of a class
  - A class is brought live by creating objects
  - The set of activities that the object performs defines the object's behavior
  - All the members of a class can be accessed through the object
  - To access the class members, we need to use the dot (.) operator

Create Class in C#

```
public class Student
{
    private string name;
    private int age;

    0 references
    public void Introduce()
    {
        Console.WriteLine($"Hi, I'm {name} and I'm {age} years old.");
    }
}
```

Classes in C#

- Create class and object in C#
- Access Modifiers
  - Public, private, protected, internal
- Accessing class members
- Constructors

Constructor

- It is a method that automatically executes when we create object of a class
- There are five types of constructors
  - Default or Parameter Less Constructor
    - System-defined default constructor
    - User-defined default constructor
  - Parameterized Constructor
  - Copy Constructor
  - Static Constructor
  - Private Constructor

System-Defined Default Constructor

```
class Employee
{
    private int age;
    private string name;
    private bool isPermanent;

    0 references
    public void display()
    {
        Console.WriteLine("Employee age is: " + age);
        Console.WriteLine("Employee name is: " + name);
        Console.WriteLine("Is employee permanent: " + isPermanent);
    }
}
```

User-Defined Default Constructor

```
class Employee
{
    private int age;
    private string name;
    private bool isPermanent;

    1 reference
    public Employee()
    {
        age = 30;
        name = "Babar Azam";
        isPermanent = true;
    }

    1 reference
    public void display()
    {
        Console.WriteLine("Employee age is: " + age);
        Console.WriteLine("Employee name is: " + name);
        Console.WriteLine("Is employee permanent: " + isPermanent);
    }
}
```

Parameterized Constructor

```
public Employee(int age, string name, bool isPermanent)
{
    this.age = age;
    this.name = name;
    this.isPermanent = isPermanent;
}
```

```
static void Main(string[] args)
{
    Employee e1 = new Employee(40, "Shahid Afridi", true);
    e1.display();
}
```

Copy Constructor

```
public Employee(Employee e)
{
    age = e.age;
    name = e.name;
    isPermanent = e.isPermanent;
}
```

```
static void Main(string[] args)
{
    Employee e1 = new Employee();
    Employee e2 = new Employee(e1);
    e2.display();
}
```

Static Constructor

- Special constructors used to initialize static data members
- They run automatically before the first instance of the class is created
- Key points about static constructors:
  - They do not have access modifiers or parameters
  - A class can have only one static constructor
  - It can’t be called explicitly, it is always called implicitly
  - It can only access the static members of the class
  - Static constructor will be invoked only once i.e. at the time of class loading

Static Constructor

- Static constructors are commonly used to:
  - Initialize static variables that need more than simple assignment
  - Load configuration data or resources needed for the class
  - Enforce runtime checks or prepare static caches

Static Constructor

```
public class Logger
{
    public static string LogFilePath;

    // Static constructor to initialize static members
    static Logger()
    {
        // Initialize the static LogFilePath once before the class is used
        LogFilePath = "/var/log/app_log.txt";
        Console.WriteLine("Static constructor called: LogFilePath initialized.");
    }

    public static void LogMessage(string message)
    {
        // Imagine this writes the message to the log file
        Console.WriteLine($"Logging to {LogFilePath}: {message}");
    }
}
```

Private Constructor

- The constructor declared as private is known as a private constructor
- When a class contains a private constructor then we cannot create an object for the class outside of the class
- So, private constructors are used to create an object for the class within the same class

Why use Private Constructor?

- To restrict object creation from outside the class
- To implement the singleton design pattern
  - It ensures that a class has only one instance throughout the application and provides a global point of access to that instance
  - To manage shared resources like database connections or configuration settings
- Classes with only static members
  - Sometimes a class is designed solely to hold static methods (like utility/helper classes). Making the constructor private prevents creating instances that serve no purpose