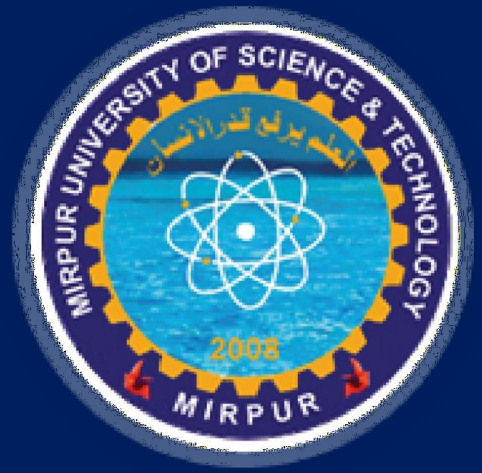


MUST

Wisdom & Virtue

MIRPUR UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF SOFTWARE ENGINEERING

Object Oriented Programming

Lecture 5: Abstraction in OOP

Lecturer

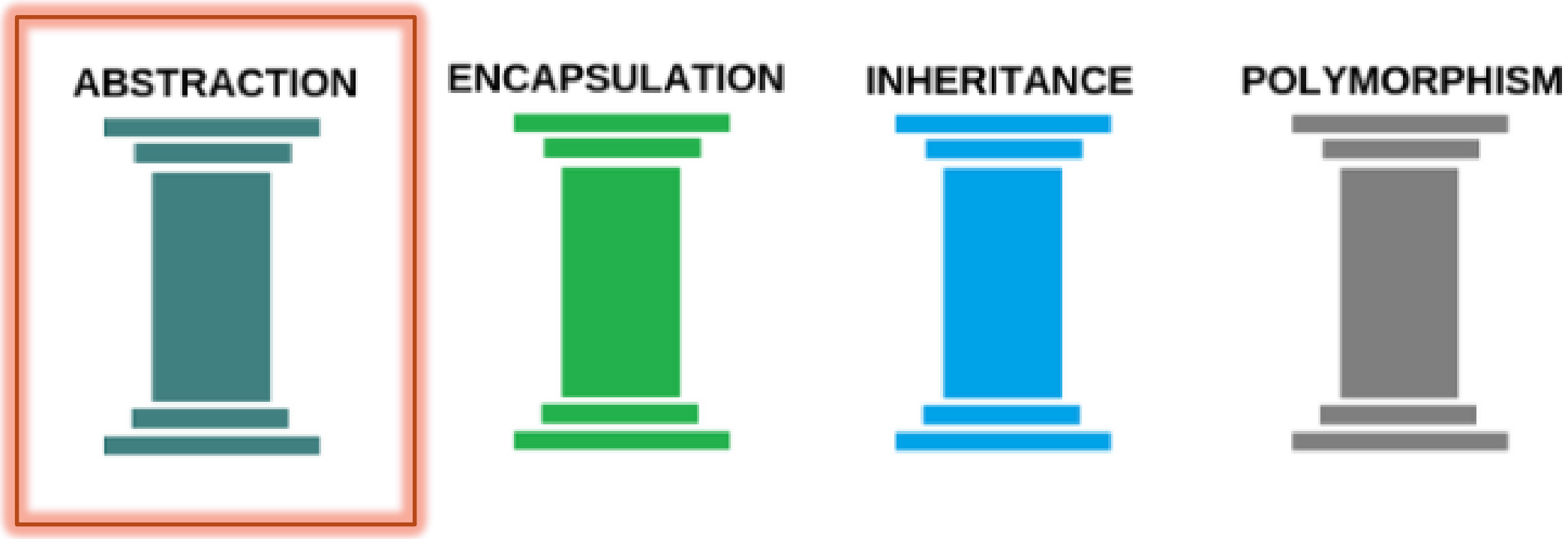
Last Lecture

- What is a Class?
- How to create and use a class in C#?
- How to declare multiple instance of a class

This Lecture

- Abstraction in OOP
- Abstraction at Design Level
- Abstraction at Programming Level (Information Hiding)

Four Pillars of OOP



Abstraction in OOP

Abstraction in OOP

- Abstraction in Object Oriented Paradigm
  1. Design Level (Object)
  2. Programming Level (Class)

Abstraction at Design Level

Abstraction at Design Level

"Only include details in the system that are required for making a functional system"

- **Example** of a student Object in **Course Management System**
- Student
  - Name
  - Address
  - Sibling
  - Father Business

Relevant to our Problem

Not Relevant to our Problem

To simplify our System and to reduce complexity, we will consider on relevant details of an Object

Abstraction in Real Life

Real life objects have a lot of attributes and many kind of behaviors

But most of the time we are interested in only that part of the objects that is related to the problem we are currently going to solve

Abstraction in Real Life

for example, in implementing a school system

We don’t need to take care of the personnel life of a student or a teacher

As it will not affect our system

Abstraction in Real Life

So, we will see these objects in the perspective of school system and will ignore their other characteristics, this concept is called "***Abstraction***".

Abstraction is a way to cope with complexity and it is used to simplify things.

Principle of abstraction at Design Level:

"Capture only those details about an object that are relevant to current perspective"

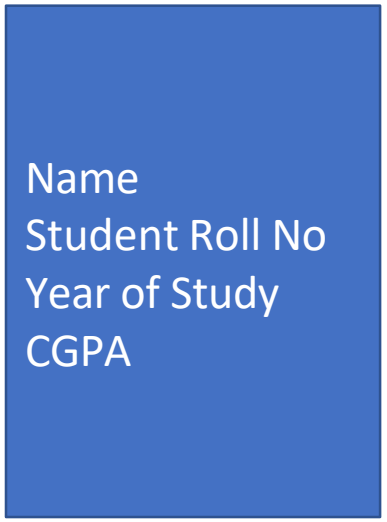
Abstraction Example:

Suppose we have the following statement in our problem statement

"Ali is a PhD student and teaches BS students"

*Here object Ali has two **perspectives** one is his **student perspective** and second is his **teacher perspective.***

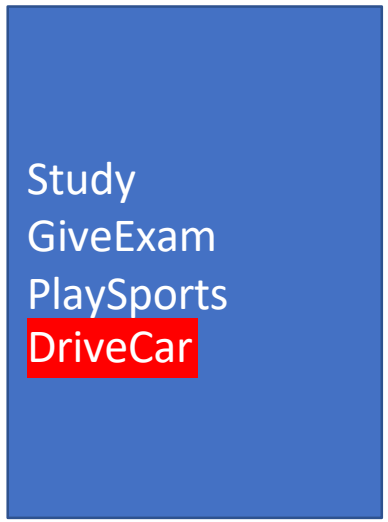
We can sum up Ali's attributes as follows



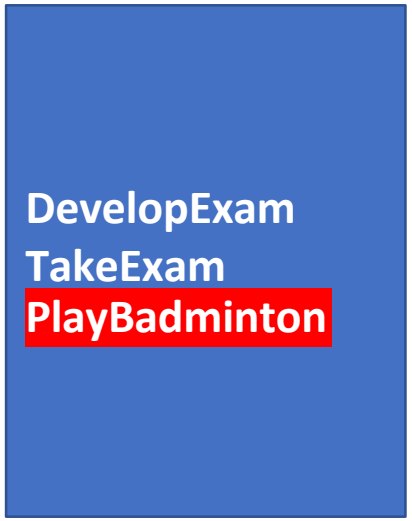
Student Perspective

Teacher Perspective

Behaviour

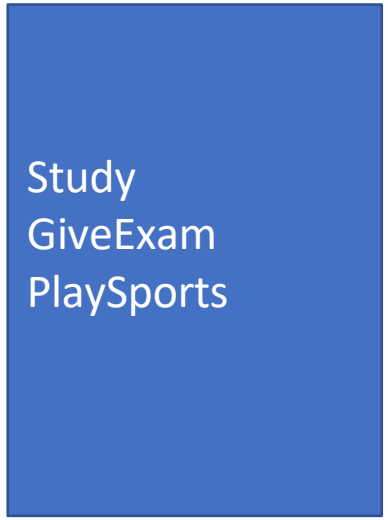


Student Perspective

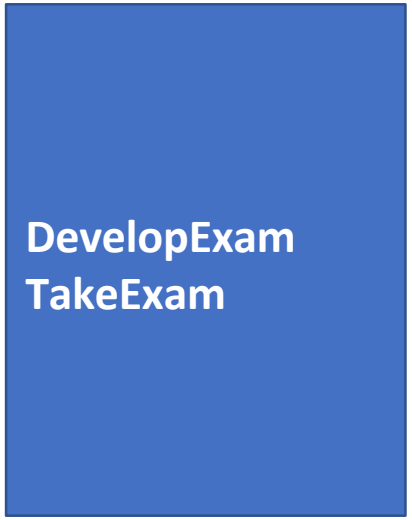


Teacher Perspective

Behaviour



Student Perspective



Teacher Perspective

Abstraction Is

- Your task is to search only relevant details in the current problem statement.
  - **i.e** the problem statement is demonstrating that Ali is only student and teacher in current scenario.
- **This is called Abstraction**

Abstraction

Abstraction is a way to cop with complexities

Principal of abstraction: Capture only those details about the object that are relevant to the current perspective

Abstraction at Programming Level

Information Hiding

Information Hiding

- Information hiding is one of the most important principles of OOP inspired from real life which says that all information should not be accessible to all persons.
- Private information should only be accessible to its owner.
- By Information Hiding we mean "**Showing only those details to the outside world which are necessary for the outside world and hiding all other details from the outside world**."

Information Hiding in OOP

- In OOP Information Hiding is achieved by
  1) Defining Scope of Class members (Access Specifier)
  2) Hiding Implementation details of Methods (Separation of Interface and Implementation)

Access Specifiers

Access specifiers

- These are used to enforce access restrictions to members of a class, there are three access specifiers

1. **'public'** enables a member to be accessed outside the class with its object
2. **'private'** restricts a member to be accessed with in the class
3. **'protected'** to be discussed when we cover inheritance

Public & Private Members in a Class

```
class Student
{
    public string Name;
    public int RollNumber;
    public string Class;
    public int Age;
    private int YearOfBirth;

    private int ClaculateYearOfBirth()
    {
        return YearOfBirth = System.DateTime.Now.Year - Age;
    }

    public string Introduce()
    {

        return "Name: " + Name + "\nRoll Number: " + RollNumber + "\nClass: " + Class + "\nAge:" + Age+ "\nYearOfBirt: " + ClaculateYearOfBirth();
    }
}
```

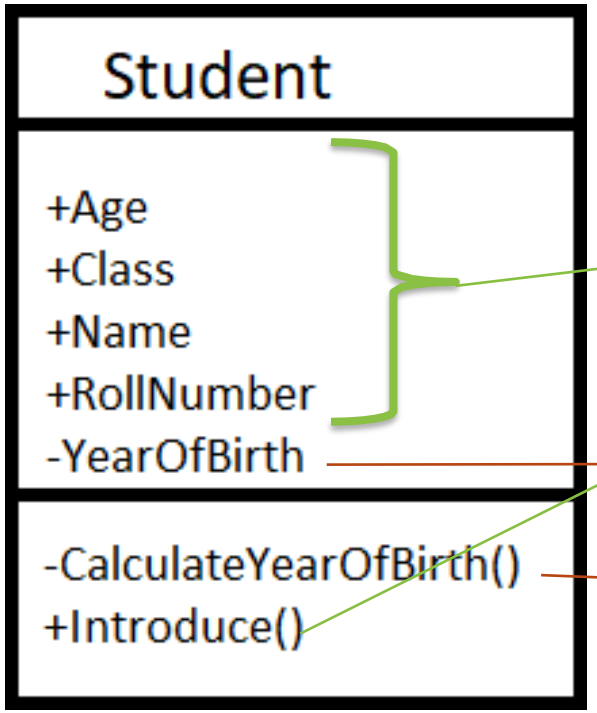
Public Members

Can be accessed outside the class

Private Members

Cannot be accessed outside the class

Public & Private Members in a UML Notation



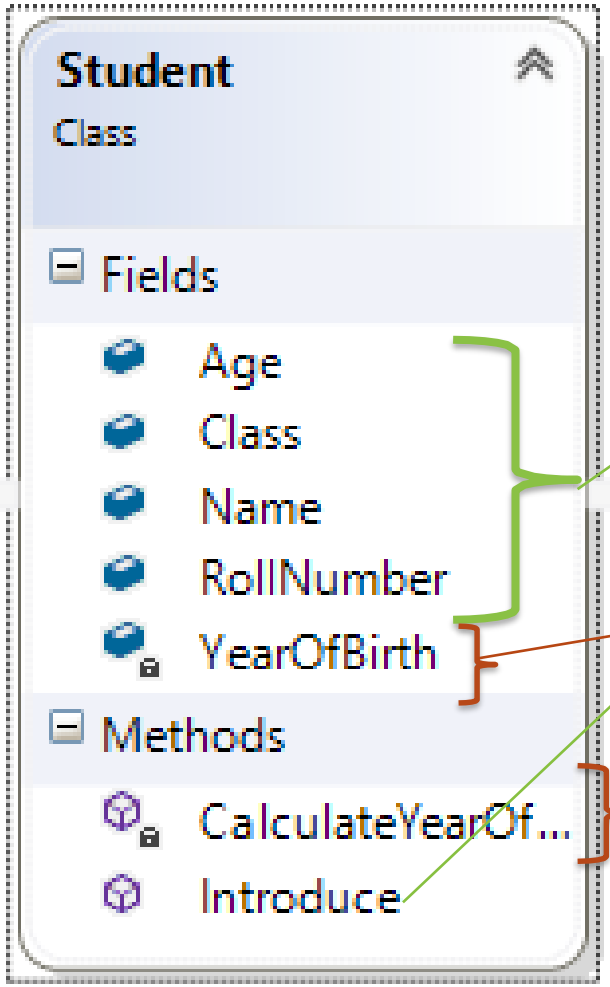
Public Members

Shown by + Symbol (Accessible to every one)

Private Members

Shown by - Symbol (Not Accessible to every one)

Public & Private Members in a Class Diagram (C#)



Public Members

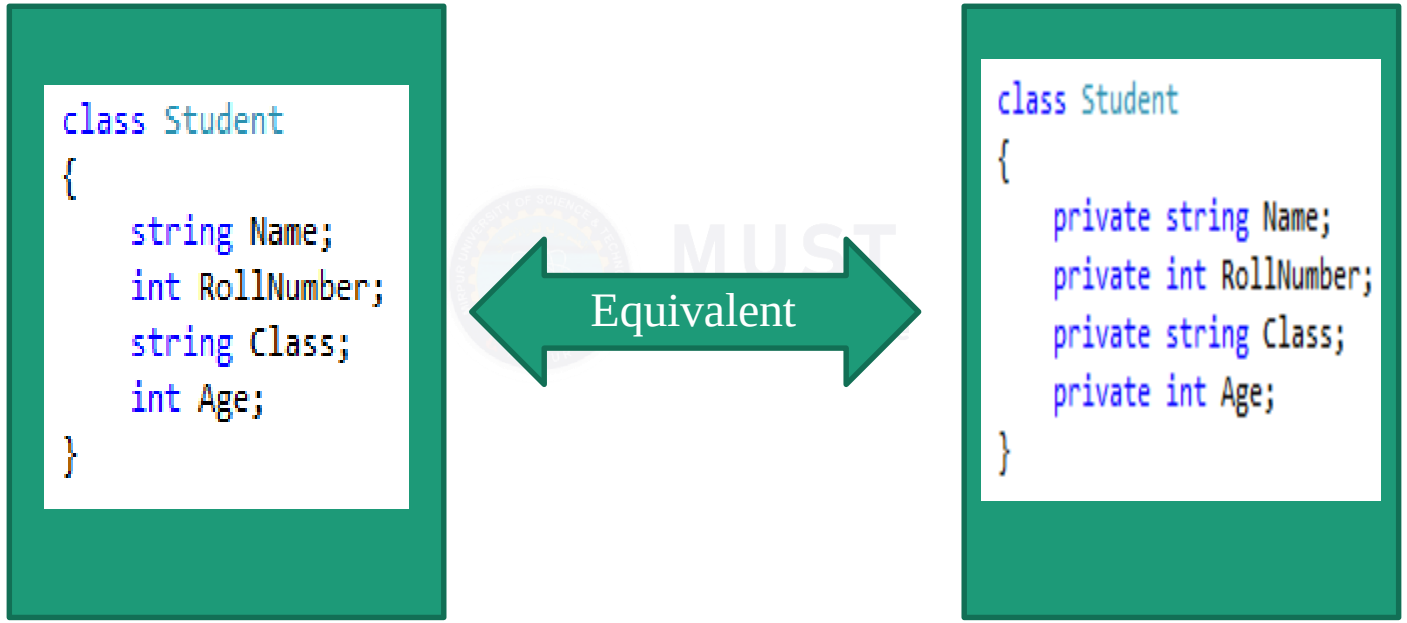
Shown to be Unlocked (Accessible to every one)

Private Members

Shown to be Locked (Not Accessible to every one)

Public & Private Members in a Class Diagram (C# Demonstration)

Default Access Specifier



Separation of Interface & Implementation

Real Life example of separation of interface and implementations

- Driver has a standard interface to drive a car and using that interface the drive can drive any car regardless of its model or type whatever engine type it has or whatever type of fuel it is using.

What is a Constructor?

- A **constructor** is a **special method** in a class.
- It is **automatically called** when an object is created.
- Used to **initialize object properties**.
- Constructor name is **same as class name**.
- It has **no return type**, not even void.

Types of Constructors

- **Default Constructor**
  → Takes no parameters.
- **Parameterized Constructor**
  → Takes values to initialize object fields.
- **Static Constructor**
  → Used to initialize static data, runs only once.

Default Constructor – Example

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

    public Student()  // Default Constructor
    {
        name = "No Name";
        age = 0;
    }

    public void Show()
    {
        Console.WriteLine("Name: " + name + ", Age: " + age);
    }
}
```

Usage

```
Student s1 = new Student();
s1.Show();
```

Parameterized Constructor – Example

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

    public Student(string n, int a)  // Parameterized Constructor
    {
        name = n;
        age = a;
    }

    public void Show()
    {
        Console.WriteLine("Name: " + name + ", Age: " + age);
    }
}
```

Parameterized Constructor – Usage

```
Student s2 = new Student("Ali", 21);
s2.Show();
```

Parameterized Constructor – Usage

- Name is always same as class name.
- No return type (not even void).
- Can be overloaded (multiple constructors in same class).
- Called only once when object is created.

•Used to **initialize object properties**.
•Constructor name is **same as class name**.
•It has **no return type**, not even void.

Interface

- Interface is a set of functions of an object that it wants to expose to other objects (**Public Functions**).
- Interfaces are necessary for object communication.
- Each object provides interface/s (operations) to other objects through these interfaces other objects communicate with this object.

Abstraction

Design Level

Abstraction at **Object** Level

- Abstraction in Object Oriented paradigm reduce complexity at the design level.
- Capture only those details about an object that are relevant to current perspective

Programming Level

Abstraction at **Class** Level

- A Class can decide which data member will be visible to outside world and which is not (Access Specifiers).
- One class should not know the inner details of another in order to use it, just knowing the interfaces should be good enough.

References

- Object Oriented Programing , Virtual University , Lecture 2, Online Available at: https://ocw.vu.edu.pk/CourseDetails.aspx?cat=Computer+Science%2FInformation+Technology+&course=CS304
- Object Oriented Programing , Virtual University , Lecture 7, Online Available at: https://ocw.vu.edu.pk/CourseDetails.aspx?cat=Computer+Science%2FInformation+Technology+&course=CS304

THANKS