Object Oriented Programming

Lab Manual 2

Introduction

After a week of rigorous coding, Welcome back!

You have learned all about the C# in the previous lab manuals.
Let's move on to the next, new, and interesting concepts.

Students, the Object-Oriented Programming is different from Procedural programming as it is about creating objects that combine data in a single unit.

Learning Objectives:

1. Package the related data as single unit (Class)
2. Create variables (Object) to use the data.
3. Explain Role of the Constructors (Copy Constructors) and Memory Representation of the objects.
4. Behavior of the class (Identify and write the functions in side the class)

Let's do some coding.

Scanior 1

Class	Driver Code

```
public class Student
{
    public string Name;
    public float EcatMarks;
    public float FscMarks;
    public float MatricMarks;
    public float Aggrigate;
}
```

```
0 references
static void Main(string[] args)
{
    Student student = new Student();
    student.Name = "Ali";
    student.EcatMarks = 300;
    student.MatricMarks = 800;
    student.FscMarks = 980;


    Student student2 = student;
    student2.Name = "Sarmad";

    Console.WriteLine(student2.Name);
    Console.WriteLine(student.Name);

    Console.ReadKey();
}
```

What will be the output of this code given the reason?

Your code is creating a simple class Student with some fields like Name, EcatMarks, FscMarks, MatricMarks, and Aggregate. In your Main method, you create an instance of the Student class called student, set some values, and then create another instance called student2 and assign the values of student to it. After that, you change the Name of student2 and print both student.Name and student2.Name to observe the impact on the original object.

Now, let's clarify and simplify the explanation:

Here's the key point to understand: When you assign one object to another in C# (e.g., student2 = student), you are not creating a new object. Instead, both variables (student and student2) now refer to the same object in memory. Therefore, any changes made through one variable will affect the other.

In your example, when you change the Name of student2, it also changes the Name of the original student because they both point to the same object in memory. This is due to the reference assignment (student2 = student).

So, if you print both student.Name and student2.Name after changing student2.Name, you will see that both names are now "Sarmad" because they both refer to the same Student object.

It's important to be aware of this behavior when working with object assignments in C#. If you want to create a truly independent copy of an object, you would need to implement a copy constructor or another method to explicitly duplicate the object's values.

This is how we define the copy constructor in the class

```
public class Student
{
    public string Name;
    public float EcatMarks;
    public float FscMarks;
    public float MatricMarks;
    public float Aggrigate;

    1 reference
    public Student()
    {

    }
    0 references
    public Student(Student student)
    {
        Name=student.Name;
        EcatMarks=student.EcatMarks;
        FscMarks=student.FscMarks;
        MatricMarks=student.MatricMarks;
        Aggrigate=student.Aggrigate;
    }
}
```

Explanation:

The default constructor (public Student()) allows the creation of Student objects without specifying initial values. The attributes will be initialized to their default values (e.g., null for string, 0 for float).

The copy constructor (public Student(Student student)) takes another Student object as a parameter and copies its values to create a new Student object with the same attributes. This is useful when you want to create a new instance of a Student with the same attributes as an existing one.

Self Assessment Task 1:

You are developing a Point of Sale (POS) system to manage transactions in a retail store. The system uses a Transaction class to represent individual transactions. Your manager has requested

the implementation of a copy constructor for the Transaction class to enable easy duplication of transaction details. Here is list of transaction attributes TransactionId, ProductName, Amount, Date and Time (string)

Behavior of the Class

In simple terms, the behavior of a class in programming refers to the actions or operations that the class can perform. It involves defining functions (methods) inside the class that operate on the data stored within the class. These functions represent the capabilities or behaviors associated with the class.

Class with its behavior

```
2 references
public class Dog
{
    4 references
    public string Name { get; set; }
    2 references
    public int Age { get; set; }
    1 reference
    public void Bark()
    {
        Console.WriteLine($"{Name} is barking!");
    }

    1 reference
    public void Eat(string food)
    {
        Console.WriteLine($"{Name} is eating {food}.");
    }
    1 reference
    public int ConvertToHumanYears()
    {
        return Age * 7; // Assuming 1 dog year is equivalent to 7 human years
    }
}
```

Driver Code

```
0 references
static void Main(string[] args)
{
    Dog myDog = new Dog();
    myDog.Name = "Buddy";
    myDog.Age = 3;

    myDog.Bark();
    myDog.Eat("dog food");
    int humanYears = myDog.ConvertToHumanYears();
    Console.WriteLine($"{myDog.Name}'s age in human years: {humanYears}");


    Console.ReadKey();
}
```

Self Assessment Task 2: Imagine you are tasked with building a basic calculator class in C#. The class should allow users to perform arithmetic operations such as addition, subtraction, multiplication, and division. Make sure you create these functions inside the class and you have to initialize the class with the constructor.

Self Assessment Task 3: Imagine you are developing a simulation of an Automated Teller Machine (ATM) in C#. Your task is to create an `ATM` class that models the functionality of a typical ATM. The class should support the following operations:

1. **Initialization:**
    - Create an `ATM` class.
    - Include initializes the ATM with a specified balance.
2. **Deposit:**
    - Implement a method named `deposit` that takes an amount as an argument and adds it to the ATM's balance.
3. **Withdrawal:**

- Implement a method named `withdraw` that takes an amount as an argument and deducts it from the ATM's balance. Ensure that the withdrawal amount does not exceed the available balance.

4. **Check Balance:**
   - Implement a method named `check_balance` that returns the current balance of the ATM.
5. **Transaction History:**
   - Implement a method named `transaction_history` that maintains a record of all transactions made on the ATM. The transaction history should include details such as the type of transaction (deposit or withdrawal) and the amount involved.
6. **Card Validation:**
   - Implement a method named `validate_card` that simulates card validation. For simplicity, you can assume that a valid card is present and skip any actual card validation checks.
7. **PIN Validation:**
   - Implement a method named `validate_pin` that simulates PIN validation. For simplicity, you can assume a correct PIN is entered and skip any actual PIN validation checks.

Congratulations !!!! You have just learned how to create different types of constructors for creating class objects.

You may take a two minute break, as there is much more code to come.

Challenge # 1:
Student Management System with Class and Behavior

Identify the behavior of your last developed student management system and make possible changes into it.

Task: Write a Menu driven program that shows the following four menu options

1. Add Student.
2. Show Students.
3. Calculate Aggregate
4. Top Students.

- Add Student allows users to add a Student's information that includes Student Name, matric marks, fsc marks, ecat marks.
    - **Hint:** Take input from the user in simple variables and then Use the parameterized constructor to create the object.
- Show Students displays all the added students on the screen.
- Calculate Aggregate will calculate the aggregate of all the available students.
- Top Students lists the information of the top 3 students.

Note: Remember to make single responsibility functions for each functionality. Remember that you have seen how to pass the object as parameter to a function and how to return an object from a function. You have also seen how to create an array of objects. (Use it well)

Challenge # 2:
Products Management System with Class

Task: Identify the behavior of your last developed student management system and make possible changes into it. .

Write a program that shows three menu options

1. Add Products.
2. Show Products.
3. Total Store Worth.

- Add Product allows the user to add product information that includes ID, Name, price, Category, BrandName, Country.
- Show Product displays all the added products on the screen.
- Total Store Worth calculates the sum of the price of all the products.

Challenge # 3:

Task. Identify the behavior of your last developed student management system and make possible changes into it.

Convert the signUp/signIn application that you developed in the previous lab by using the class concepts.

Make a class named MUser with 3 attributes namely

- Username
- Password
- Role

The data should be loaded from the file and loaded into the attributes of the class.

Good Luck and Best Wishes !!

Happy Coding ahead :)