Lesson 21 - Classes & Objects II

Value and Reference Types

Variables for built-in, primitive data types store the value of the variable, whereas class object variables store the reference to the value. What does this really mean?

Stack vs Heap

There are two types of memory allocation in .NET. Stack is used for static memory allocation, and Heap is used for dynamic memory allocation. Dynamic memory allocation means that the size is not fixed, and space must be dynamically allocated.

For reference types, the value is stored on the **heap**, and the reference to that value, which is its memory address, is stored on the **stack**.

Encapsulation

Encapsulation means surrounding something to protect it. In programming, this means to combine different things together, and to restrict access to these things and their inner workings from different contexts. The implementation of this is done using **access modifiers**, which defines the scope and visibility of a class member.

> Encapsulation is also referred to as **information hiding**.

Access Modifiers

1. `public` : Makes the member accessible outside the class.
2. `private` : Makes the member accessible only from within the class and hides it from outside.
3. `protected`
4. `internal`
5. `protected internal`

Constructors

A class constructor is a special method, having the **same name** as the class, which is executed whenever you create an object of that class. This method needs to be `public`, and

does not have any `return type`.

Recall the student class:

> ```
> class Student
> {
>     public int Age;
>     public string Name;
>     public int[] Grades = new int[3];
>
>     public void DisplayGrades()
>     {
>         Console.WriteLine("Student: " + Name);
>         foreach (var grade in Grades)
>             Console.WriteLine(grade);
>     }
> }
> ```

Let's modify it a bit and also add a constructor to it:

```
Student talha = new("Talha", 21, 8, 7, 10);
Console.WriteLine(talha.AverageGrade);

public class Student
{
    public int Age;
    private string Name;
    public int[] Grades = new int[3];
    public double AverageGrade;

        public Student(string name, int age,
                int grade1, int grade2, int grade3)
        {
                Name = name;
                Age = age;
                Grades = new int[] { grade1, grade2, grade3 };
                AverageGrade = Grades.Average();
        }

    public void DisplayGrades()
    {
        Console.WriteLine("Student: " + Name);
        foreach (var grade in Grades)
            Console.WriteLine(grade);
    }
}
```

```
OUTPUT: 8.333333333333334
```

Multiple Constructors

The default constructor takes no parameters. But as shown above, we can add parameters to creation using custom constructors.

Just like methods, constructors can also be overloaded.

```
public class Student
{
    public int Age;
    private string Name;
    public int[] Grades = new int[3];
    public double AverageGrade;

        public Student(string name, int age,
                int grade1, int grade2, int grade3)
        {
                Name = name;
                Age = age;
                Grades = new int[] { grade1, grade2, grade3 };
                AverageGrade = Grades.Average();
        }

        public Student(string firstName, string lastName)
        {
                Name = firstName + lastName;
        }

    public void DisplayGrades()
    {
        Console.WriteLine("Student: " + Name);
        foreach (var grade in Grades)
            Console.WriteLine(grade);
    }
}
```

```
public class BankAccount
{
        public string AccountTitle;
        public string AccountNumber;
        private int Balance;
        private string PhoneNumber;
        private string Address;

        public BankAccount(string firstName, string lastName,
```

```
                int openingBalance, string phoneNumber, string address,
                string accountNumber)
        {
                AccountTitle = firstName + " " + lastName;
                Balance = openingBalance;
                PhoneNumber = phoneNumber;
                Address = address;
                AccountNumber = accountNumber;
        }

        public void CheckBalance()
        {
                Console.WriteLine("\nAccount Title: " + AccountTitle);
                Console.WriteLine("Account Number: " + AccountNumber);
                Console.WriteLine("Current Balance: $" + Balance);
        }

        public void Deposit(int amountToDeposit)
        {
                CheckBalance();
                Balance += amountToDeposit;
                Console.WriteLine("Deposited $" + amountToDeposit);
                Console.WriteLine("New Balance: $" + Balance + "\n");
        }

        public void WithDraw(int amountToWithDraw)
        {
                CheckBalance();
                if (amountToWithDraw > Balance)
                {
                        Console.WriteLine("Insufficient Balance");
                        return;
                }
                Balance -= amountToWithDraw;
                Console.WriteLine("Withdrew $" + amountToWithDraw);
                Console.WriteLine("New Balance: $" + Balance + "\n");
        }
}
```