Lesson 23 - Classes & Objects IV

Static Variables

Class members (variables, properties, methods) can be declared as static. This makes those members belong to the class itself, instead of belonging to individual objects. No matter how many objects of the class are created, there is only one copy of the static member.

```
Console.WriteLine($"Total Accounts: {BankAccount.AccountsInBank}");

BankAccount blankAccount = new(100);
blankAccount.ShowDetails();
Console.WriteLine($"Total Accounts: {BankAccount.AccountsInBank}");

Console.WriteLine();

BankAccount titleAccount = new("Talha", 150);
titleAccount.ShowDetails();
Console.WriteLine($"Total Accounts: {BankAccount.AccountsInBank}");

Console.WriteLine();

BankAccount deadAccount = new();
deadAccount.ShowDetails();
Console.WriteLine($"Total Accounts: {BankAccount.AccountsInBank}");

public class BankAccount
{
    public static int AccountsInBank { get; set; }

    public int Balance { get; set; } = 0;
    private readonly string accountTitle = "default";

    public BankAccount() => AccountsInBank++;

    public BankAccount(int balance)
    {
        Balance = balance;
        AccountsInBank++;
    }
    public BankAccount(string accountTitle, int balance)
    {
        Balance = balance;
        this.accountTitle = accountTitle;
        AccountsInBank++;
    }

    public void ShowDetails()
        => Console.WriteLine($"Title: {accountTitle}\nBalance: {Balance}");
}
```

```
OUTPUT:
Total Accounts: 0
Title: default
Balance: 100
Total Accounts: 1

Title: Talha
Balance: 150
Total Accounts: 2

Title: default
Balance: 0
Total Accounts: 3
```

> No matter how many `BankAccount` objects are instantiated, there is always only one `accountsInBank` variable that belongs to the `BankAccount` class because it was declared `static`.

Because of their global nature, static members can be accessed directly using the class name without an object. The `accountsInBank` variable is shared between all `BankAccount` objects. For this class, each time an object is created, the static value is *incremented*.

> You must access static members using the class name. If you try to access them via an object of that class, you will generate an error.

Static Methods

The same concept applies to static methods.

```
public static void CheckAccountsInBank()
        => Console.WriteLine("Accounts open in Bank: " + AccountsInBank);
```

Constant Variables

By definition, constants are static.

```
public const int BRANCH_CODE = 0419;
```

Static Classes

An entire class can be declared as `static`. A **static class** can contain only static members. You cannot instantiate an object of a static class, as only one instance of the static class can exist in a program. Static classes are useful for combining logical properties and methods. A good example of this is the `Math` class. It contains various useful properties and methods for mathematical operations.

```
Math.Pow(2, 3);
```

> All members of this class can be accessed without creating an instance, using the class name.

Math Class

`Math.PI` the constant PI.
`Math.E` represents the natural logarithmic base e.
`Math.Max()` returns the larger of its two arguments.
`Math.Min()` returns the smaller of its two arguments.
`Math.Abs()` returns the absolute value of its argument.
`Math.Sin()` returns the sine of the specified angle.
`Math.Cos()` returns the cosine of the specified angle.
`Math.Pow()` returns a specified number raised to the specified power.
`Math.Round()` rounds the decimal number to its nearest integral value.
`Math.Sqrt()` returns the square root of a specified number.

Array Class

`Array.Reverse()` reverses the given array.
`Array.Sort()` sorts the given array.

String Class

`String.Concat()` will combine two strings
`String.Equals()` checks if two strings are equal, returns a boolean result.

DateTime Class

`DateTime.Now()` represents current day and night.
`DateTime.Today()` represents current day.
`DateTime.DaysInMonth()` return number of days in specified month.

Console Class

`Console.WriteLine()` output to screen.
`Console.ReadLine()` take input from user.

Convert Class

`Convert.ToInt32()`
`Convert.ToDouble()`
`Convert.ToBoolean()`
`Convert.ToStrint()`

Assignment 6

Create a `Rectangle` class, which has the following members:

1. `Type` this must store whether the rectangle is a square or not
2. `Length`
3. `Width`
4. `Area()`
5. `Perimeter()`

You need to overload the *constructor* so the user can create both a rectangle, and a square (i.e., you only enter one value, which is the same for both length and width).

You must also overload the `Area()` and `Perimeter()` methods, since they will have different definitions for square and rectangle.

> Area of Rectangle: $L \times W$
> Perimeter of Rectangle: $2 \times (L + W)$

> Area of Square: L^2 (you can use `Math.Pow()` here)
> Perimeter of Rectangle: $4 \times L$

You must also create static methods that let you calculate the **Perimeter** and **Area** of any *Length* and *Width* values, without having to create an object first.