### Exercise 1: Static Class Definition

**Lab Exercise:**

1. Define a static class named `MathUtilities`.

2. Add a static method `Square(int number)` that returns the square of the given number.

3. Call the method from the `Main` method and print the result.

**Solution:**

```csharp
public static class MathUtilities
{
    public static int Square(int number)
    {
        return number * number;
    }
}

class Program
{
    static void Main(string[] args)
    {
        int result = MathUtilities.Square(5);
        Console.WriteLine($"Square of 5 is: {result}");
    }
}
```

### Exercise 2: Static Constructor

**Lab Exercise:**

1. Create a class `DatabaseConnection` with a static constructor.

2. Use the static constructor to initialize a static field `ConnectionString`.

3. Add a static method `GetConnectionString()` to return the connection string.

**Solution:**

```csharp
public class DatabaseConnection
{
    private static string ConnectionString;

    static DatabaseConnection()
    {
        ConnectionString = "Server=myServer;Database=myDB;User Id=myUsername;Password=myPassword;";
    }

    public static string GetConnectionString()
    {
        return ConnectionString;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(DatabaseConnection.GetConnectionString());
    }
}
```

### Exercise 3: Static Variables

**Lab Exercise:**

1. Create a class `Counter` with a static variable `count`.

2. Increment `count` in the constructor of the class.

3. Display the value of `count` each time an object of the class is created.

**Solution:**

```csharp
public class Counter
{
    private static int count = 0;

    public Counter()
    {
        count++;
        Console.WriteLine($"Count: {count}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Counter c1 = new Counter();
        Counter c2 = new Counter();
        Counter c3 = new Counter();
    }
}
```

### Exercise 4: Static Members vs Non-static Members

**Lab Exercise:**

1. Create a class `Student` with a static field `TotalStudents` and a non-static field `Name`.

2. Increment `TotalStudents` in the constructor.

3. Add a method `DisplayStudentInfo()` to show the student's name and total students.

**Solution:**

```csharp
public class Student
{
    public string Name { get; set; }
    public static int TotalStudents { get; private set; }

    public Student(string name)
    {
        Name = name;
        TotalStudents++;
    }

    public void DisplayStudentInfo()
    {
        Console.WriteLine($"Name: {Name}, Total Students: {TotalStudents}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Student s1 = new Student("Alice");
        Student s2 = new Student("Bob");
        s1.DisplayStudentInfo();
        s2.DisplayStudentInfo();
    }
}
```

### Exercise 5: Static Methods

**Lab Exercise:**

1. Create a static class `TemperatureConverter`.
2. Add static methods `CelsiusToFahrenheit` and `FahrenheitToCelsius`.
3. Use these methods in the `Main` method to convert temperatures.

**Solution:**

```csharp
public static class TemperatureConverter
{
    public static double CelsiusToFahrenheit(double celsius)
    {
        return (celsius * 9 / 5) + 32;
    }

    public static double FahrenheitToCelsius(double fahrenheit)
    {
        return (fahrenheit - 32) * 5 / 9;
    }
}

class Program
{
    static void Main(string[] args)
    {
        double celsius = 25;
        double fahrenheit = TemperatureConverter.CelsiusToFahrenheit(celsius);
        Console.WriteLine($"{celsius} °C = {fahrenheit} °F");

        fahrenheit = 77;
        celsius = TemperatureConverter.FahrenheitToCelsius(fahrenheit);
        Console.WriteLine($"{fahrenheit} °F = {celsius} °C");
    }
}
```

### Exercise 6: Static Properties

**Lab Exercise:**

1. Create a class `Configuration` with a static property `AppName`.

2. Set a default value in the static constructor.

3. Allow the application name to be retrieved and updated.

**Solution:**

```csharp
public class Configuration
{
    public static string AppName { get; set; }

    static Configuration()
    {
        AppName = "My Application";
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine($"App Name: {Configuration.AppName}");
        Configuration.AppName = "New Application Name";
        Console.WriteLine($"Updated App Name: {Configuration.AppName}");
    }
}
```

### Exercise 7: Static vs Non-static Methods

**Lab Exercise:**

1. Create a class `Account` with static and non-static methods.
2. Add a static method `CalculateInterest` and a non-static method `Deposit`.
3. Demonstrate calling both methods.

**Solution:**

```csharp
public class Account
{
    public double Balance { get; set; }

    public void Deposit(double amount)
    {
        Balance += amount;
        Console.WriteLine($"Deposited: {amount}, New Balance: {Balance}");
    }
```

    public static double CalculateInterest(double balance, double rate)
    {
        return balance * rate / 100;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Account account = new Account();
        account.Deposit(1000);

        double interest = Account.CalculateInterest(account.Balance, 5);
        Console.WriteLine($"Interest: {interest}");
    }
}
~~~

### Exercise 8: Static Fields in Different Instances

**Lab Exercise:**

1. Create a class `ShoppingCart` with a static field `TotalCarts` and a non-static field `CartId`.
2. Increment `TotalCarts` in the constructor and assign `CartId`.
3. Display the `TotalCarts` and `CartId` for each object.

**Solution:**

```csharp
public class ShoppingCart
{
    public static int TotalCarts { get; private set; }
~~~

```
    public int CartId { get; private set; }

    public ShoppingCart()
    {
        TotalCarts++;
        CartId = TotalCarts;
    }

    public void DisplayCartInfo()
    {
        Console.WriteLine($"Cart ID: {CartId}, Total Carts: {TotalCarts}");
    }
}

class Program
{
    static void Main(string[] args)
    {
        ShoppingCart cart1 = new ShoppingCart();
        ShoppingCart cart2 = new ShoppingCart();

        cart1.DisplayCartInfo();
        cart2.DisplayCartInfo();
    }
}
```

### Exercise 9: Static Class with Static Methods

**Lab Exercise:**

1. Define a static class `Utility` with static methods `IsEven` and `IsOdd`.

2. Use these methods to check if a number is even or odd in the `Main` method.

**Solution:**

```csharp
public static class Utility
{
    public static bool IsEven(int number)
    {
        return number % 2 == 0;
    }

    public static bool IsOdd(int number)
    {
        return number % 2 != 0;
    }
}

class Program
{
    static void Main(string[] args)
    {
        int number = 10;
        Console.WriteLine($"{number} is even: {Utility.IsEven(number)}");
        Console.WriteLine($"{number} is odd: {Utility.IsOdd(number)}");
    }
}
```

### Exercise 10: Static Members and Inheritance

**Lab Exercise:**

1. Create a base class `Animal` with a static method `GetTotalAnimals`.

2. Create a derived class `Dog` that inherits from `Animal`.

3. Increment the animal count in the `Dog` constructor and demonstrate calling `GetTotalAnimals`.

**Solution:**

```csharp
public class Animal
{
    protected static int TotalAnimals = 0;

    public static int GetTotalAnimals()
    {
        return TotalAnimals;
    }
}

public class Dog : Animal
{
    public Dog()
    {
        TotalAnimals++;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Dog dog1 = new Dog();
        Dog dog2 = new Dog();
        Console.WriteLine($"Total Animals: {Animal.GetTotalAnimals()}");
    }
}
```