Exercise 1: Partial Classes

Lab Exercise:

1. Create a class `Person` using partial classes.

2. Split the class into two files: one for properties (`FirstName` and `LastName`) and one for a method `GetFullName()`.

3. Demonstrate using the `Person` class in the `Main` method.

Solution:

PersonPart1.cs

```csharp
public partial class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

PersonPart2.cs

```csharp
public partial class Person
{
    public string GetFullName()
    {
        return $"{FirstName} {LastName}";
    }
}
```

Program.cs

```csharp
class Program
{
    static void Main(string[] args)
    {
        Person person = new Person { FirstName = "John", LastName = "Doe" };
        Console.WriteLine(person.GetFullName());
    }
}
```

Exercise 2: Extension Methods

Lab Exercise:

1. Create an extension method `WordCount` for the `string` class that counts the number of words in a string.

2. Demonstrate using this extension method in the `Main` method.

Solution:

```csharp
public static class StringExtensions
{
    public static int WordCount(this string str)
    {
        return str.Split(new char[] { ' ', '.', '?' }, StringSplitOptions.RemoveEmptyEntries).Length;
    }
}

class Program
{
    static void Main(string[] args)
```

```
    {
        string sentence = "Hello world! This is an example sentence.";
        int count = sentence.WordCount();
        Console.WriteLine($"Word Count: {count}");
    }
}
```

Exercise 3: Collection Initializers

Lab Exercise:

1. Create a `List` of integers using collection initializers.
2. Add integers to the list during initialization and display the contents.

Solution:

```csharp
class Program
{
    static void Main(string[] args)
    {
        List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

        Console.WriteLine("Numbers in the list:");
        foreach (int number in numbers)
        {
            Console.WriteLine(number);
        }
    }
}
```

Exercise 4: Object Initializers

Lab Exercise:

1. Create a class `Car` with properties `Make`, `Model`, and `Year`.

2. Create an instance of `Car` using object initializers and display its properties.

Solution:

```csharp
public class Car
{
    public string Make { get; set; }
    public string Model { get; set; }
    public int Year { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
        Car car = new Car { Make = "Toyota", Model = "Corolla", Year = 2020 };

        Console.WriteLine($"Car: {car.Make} {car.Model} {car.Year}");
    }
}
```

Exercise 5: Nullable Types

Lab Exercise:

1. Create a nullable integer and demonstrate checking its value.

2. Use the `GetValueOrDefault()` method to provide a default value if the nullable integer is `null`.

Solution:

```csharp
class Program
{
    static void Main(string[] args)
    {
        int? nullableInt = null;

        if (nullableInt.HasValue)
        {
            Console.WriteLine($"Value: {nullableInt.Value}");
        }
        else
        {
            Console.WriteLine("No value assigned.");
        }

        int defaultValue = nullableInt.GetValueOrDefault(10);
        Console.WriteLine($"Default Value: {defaultValue}");
    }
}
```

Exercise 6: Enums

Lab Exercise:

1. Create an enum `Days` that represents the days of the week.
2. Write a method `GetDayMessage(Days day)` that returns a message depending on the day.
3. Demonstrate using the enum and the method.

Solution:

```csharp
public enum Days
{
   Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday
}

class Program
{
   static void Main(string[] args)
   {
      Days today = Days.Wednesday;
      Console.WriteLine(GetDayMessage(today));
   }

   static string GetDayMessage(Days day)
   {
      switch (day)
      {
         case Days.Monday:
            return "Start of the work week!";
         case Days.Friday:
            return "Almost the weekend!";
         case Days.Saturday:
         case Days.Sunday:
            return "Enjoy the weekend!";
         default:
            return "Just another day.";
      }
   }
}
```

```

### Exercise 7: Tuples

**Lab Exercise:**

1. Create a method `GetPersonInfo()` that returns a tuple containing a person's name and age.
2. Demonstrate using the method to retrieve and display the tuple values.

**Solution:**

```csharp
class Program
{
    static void Main(string[] args)
    {
        var personInfo = GetPersonInfo();
        Console.WriteLine($"Name: {personInfo.name}, Age: {personInfo.age}");
    }

    static (string name, int age) GetPersonInfo()
    {
        return ("John Doe", 30);
    }
}
```

Exercise 8: `const` Keyword

Lab Exercise:

1. Create a class `Constants` with a `const` field `Pi` representing the value of Pi.
2. Demonstrate using this constant in a method to calculate the area of a circle.
```

Solution:

```csharp
public class Constants
{
    public const double Pi = 3.14159;
}

class Program
{
    static void Main(string[] args)
    {
        double radius = 5.0;
        double area = CalculateArea(radius);
        Console.WriteLine($"Area of circle with radius {radius}: {area}");
    }

    static double CalculateArea(double radius)
    {
        return Constants.Pi * radius * radius;
    }
}
```

Exercise 9: `readonly` Keyword

Lab Exercise:

1. Create a class `Circle` with a `readonly` field `radius`.
2. Initialize the `radius` field through a constructor.
3. Demonstrate creating an instance of `Circle` and attempting to modify the `radius`.

Solution:

```csharp
public class Circle
{
    public readonly double Radius;

    public Circle(double radius)
    {
        Radius = radius;
    }

    public double CalculateCircumference()
    {
        return 2 * Constants.Pi * Radius;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Circle circle = new Circle(10);
        Console.WriteLine($"Circumference: {circle.CalculateCircumference()}");

        // circle.Radius = 15; // This will cause a compile-time error because Radius is readonly
    }
}
```

Exercise 10: Anonymous Types

Lab Exercise:

1. Create an anonymous type to represent a `Book` with properties `Title`, `Author`, and `Price`.

2. Display the properties of the anonymous type in the `Main` method.

Solution:

```csharp
class Program
{
    static void Main(string[] args)
    {
        var book = new { Title = "The Catcher in the Rye", Author = "J.D. Salinger", Price = 9.99 };

        Console.WriteLine($"Book: {book.Title}, Author: {book.Author}, Price: {book.Price}");
    }
}
```