.NET FRAMEWORK PROGRAMMING

Implementing Interfaces and Inheriting from Classes

Programming Practices #6_4

==========

Defining interfaces with default implementations

1. In the PacktLibrary project, add a new file named IPlayable.cs.
2. Modify the statements to define a public IPlayable interface with two methods to Play and Pause, as shown in the following code:

```
namespace Packt.Shared;

public interface IPlayable
{
   void Play();
   void Pause();
}
```

3. In the PacktLibrary project, add a new class file named DvdPlayer.cs.
4. Modify the statements in the file to implement the IPlayable interface, as shown in the following code:

```
using static System.Console;

namespace Packt.Shared;

public class DvdPlayer : IPlayable
{
   public void Pause()
   {
      WriteLine("DVD player is pausing.");
   }

   public void Play()
   {
      WriteLine("DVD player is playing.");
   }
}
```

This is useful, but what if we decide to add a third method named Stop? Before C# 8.0, this would be impossible once at least one type implements the original

interface. C# 8.0 allows an interface to add new members after release as long as they have a **default implementation**.

5. Modify the IPlayable interface to add a Stop method with a default implementation, as shown highlighted in the following code:

```
using static System.Console;

namespace Packt.Shared;

public interface IPlayable
{
   void Play();
   void Pause();
   void Stop() // default interface implementation
   {
      WriteLine("Default implementation of Stop.");
   }
}
```

6. Build the PeopleApp project and show your result.

Programming Practices #6_5

==========

Inheriting from classes

The Person type we created earlier derived (inherited) from object, the alias for System.Object. Now, we will create a subclass that inherits from Person:

1. In the PacktLibrary project, add a new class file named Employee.cs.
2. Modify its contents to define a class named Employee that derives from Person, as shown in the following code:

```
using System;

namespace Packt.Shared;

public class Employee : Person
{
}
```

3. In Program.cs of PeopleApp, add statements to create an instance of the Employee class, as shown in the following code:

```
Employee first = new()
{
   Name = "Your Name", //input your name here
   DateOfBirth = new(year: 1990, month: 7, day: 28) //input
your dateOfBirth here
};
first.WriteToConsole();
```

4. Then in Employee.cs, add statements to define two properties for an employee code and the date they were hired, as shown in the following code:

```
public string? EmployeeCode { get; set; }
public DateTime HireDate { get; set; }
```

5. In Program.cs of PeopleApp, add statements to set John's employee code and hire date, as shown in the following code:

```
first.EmployeeCode = "FR001";
first.HireDate = new(year: 2021, month: 8, day: 20);
WriteLine($"{first.Name} was hired on
{first.HireDate:dd/MM/yy}"); //this is one line statement
```

6. Run the code and show your result.

Programming Practices #6_6

==========

Overriding members

1. In Program.cs of PeopleApp, add a statement to write the value of the john variable to the console using its string representation, as shown in the following code:

```
WriteLine(first.ToString());
```

2. Run the code and note that the ToString method is inherited from System.Object, so the implementation returns the namespace and type name, as shown in the following output:

```
Packt.Shared.Employee
```

3. In Person.cs, override this behavior by adding a ToString method to output the name of the person as well as the type name, as shown in the following code:

```
// overridden methods
public override string ToString()
{
return $"{Name} is a {base.ToString()}";
}
```

The base keyword allows a subclass to access members of its superclass; that is, the base class that it inherits or derives from.

4. Run the code and show your result.

Working with Common .NET Types

Programming Practices #7_1

==========

Working with numbers

One of the most common types of data is numbers. The most common types in .NET for working with numbers are shown in the following table:

Namespace	Example type(s)	Description
System	SByte, Int16, Int32, Int64	Integers; that is, zero and positive and negative whole numbers
System	Byte, UInt16, UInt32, UInt64	Cardinals; that is, zero and positive whole numbers
System	Half, Single, Double	Reals; that is, floating-point numbers
System	Decimal	Accurate reals; that is, for use in science, engineering, or financial scenarios
System.Numerics	BigInteger, Complex, Quaternion	Arbitrarily large integers, complex numbers, and quaternion numbers

Let's explore numerics:

1. Open the Praktikum workspace that you've created before and then add a new folder to the folder named WorkingWithNumbers.
2. From inside WorkingWithNumbers folder, create a new console app.
3. In Program.cs, delete the existing statements and add a statement to import System.Numerics, as shown in the following code:

```
using static System.Console;
using System.Numerics;
```

4. Add statements to output the maximum value of the ulong type, and a number with 30 digits using BigInteger, as shown in the following code:

```
WriteLine("Working with large integers:");
WriteLine("----------------------------------");

ulong big = ulong.MaxValue;
WriteLine($"{big,40:N0}");

//this is one line statement
BigInteger bigger =
BigInteger.Parse("123456789012345678901234567890");

WriteLine($"{bigger,40:N0}");
```

5. Run the code and show your result.

Programming Practices #7_2

==========

Working with text

The most common types in .NET for working with text are shown in the following table:

Namespace	Type	Description
System	Char	Storage for a single text character
System	String	Storage for multiple text characters
System.Text	StringBuilder	Efficiently manipulates strings
System.Text.RegularExpressions	Regex	Efficiently pattern-matches strings

Let's explore some common tasks when working with text;

1. For example, sometimes you need to find out the length of a piece of text stored in a string variable:
   1.1. From Praktikum workspace create another folder named WorkingWithText then create a new console app.
   1.2. In Program.cs, delete the existing statements and add a statement to statically import System.Console, as shown in the following code:

```
using static System.Console;
```

   1.3. Still from Program.cs, add statements to define a variable to store the name of the city London, and then write its name and length to the console, as shown in the following code:

```
string city = "London";
WriteLine($"{city} is {city.Length} characters long.");
```

   1.4. The string class uses an array of char internally to store the text. It also has an indexer, which means that we can use the array syntax to read its characters. Array indexes start at zero. Add a statement to write the characters at the first and third positions in the string variable, as shown in the following code:

```
WriteLine($"First char is {city[0]} and third is
{city[2]}.");
```

   1.5. Sometimes, you need to split some text wherever there is a character, such as a comma. Add statements to define a single string variable containing comma-separated city names, then use the Split method and specify that you want to treat commas as the separator, and then enumerate the returned array of string values, as shown in the following code:

```
string cities = "Paris,Tehran,Chennai,Sydney,New
York,Medellín";

string[] citiesArray = cities.Split(',');

WriteLine($"There are {citiesArray.Length} items in the
array.");

foreach (string item in citiesArray)
{
   WriteLine(item);
}
```

1.6. Run the code and show your result.

Programming Practices #7_3

==========

Working with dates and times

The most common types in .NET for working with text are shown in the following table: After numbers and text, the next most popular types of data to work with are dates and times. The two main types are as follows:

- DateTime: represents a combined date and time value for a fixed point in time.
- TimeSpan: represents a duration of time.

These two types are often used together. For example, if you subtract one DateTime value from another, the result is a TimeSpan. If you add a TimeSpan to a DateTime then the result is a DateTime value.

A common way to create a date and time value is to specify individual values for the date and time components like day and hour, as described in the following table:

Date/time parameter	Value range
year	1 to 9999
month	1 to 12
day	1 to the number of days in that month
hour	0 to 23
minute	0 to 59
second	0 to 59

Let's see what you might want to do with dates and times:

1. From Praktikum workspace create another folder named WorkingWithTime then create a new console app.
2. In Program.cs, delete the existing statements and then add statements to initialize some special date/time values, as shown in the following code:

```
using static System.Console;

WriteLine("Earliest date/time value is: {0}",
   arg0: DateTime.MinValue);

WriteLine("UNIX epoch date/time value is: {0}",
   arg0: DateTime.UnixEpoch);

WriteLine("Date/time value Now is: {0}",
   arg0: DateTime.Now);

WriteLine("Date/time value Today is: {0}",
   arg0: DateTime.Today);
```

3. Add statements to define Christmas Day in 2022 and display it in various ways, as shown in the following code:

```
DateTime christmas = new(year: 2022, month: 12, day: 25);

WriteLine("Christmas: {0}",
   arg0: christmas); // default format

WriteLine("Christmas: {0:dddd, dd MMMM yyyy}",
   arg0: christmas); // custom format

WriteLine("Christmas is in month {0} of the year.",
   arg0: christmas.Month);

WriteLine("Christmas is day {0} of the year.",
   arg0: christmas.DayOfYear);

WriteLine("Christmas {0} is on a {1}.",
   arg0: christmas.Year,
   arg1: christmas.DayOfWeek);
```

4. Add statements to perform addition and subtraction with Christmas, as shown in the following code:

```
DateTime beforeChristmas =
christmas.Subtract(TimeSpan.FromDays(12));

DateTime afterChristmas = christmas.AddDays(12);

WriteLine("12 days before Christmas is: {0}",
   arg0: beforeXmas);

WriteLine("12 days after Christmas is: {0}",
   arg0: afterXmas);

TimeSpan untilChristmas = christmas - DateTime.Now;

WriteLine("There are {0} days and {1} hours until Christmas.",
   arg0: untilChristmas.Days,
   arg1: untilChristmas.Hours);

WriteLine("There are {0:N0} hours until Christmas.",
   arg0: untilChristmas.TotalHours);
```

5. Run the code and show your results.