1: Point Class (with Tutor)

Create a new C# project for defining and testing the class *Point* described during the lecture. The project should include the usual file *Program.cs*, created as part of the project template, with the definition of the *Program* class and the *Main* program entry point. The *Point* class should be defined in a separate file *Point.cs*, which needs to be added to the project. The *Main* will contain the instructions to create two points (p1 = (5, 1) and p2 = (7, 2)) and to print their coordinates on the screen.

Solution

Point.cs

```
using System;

namespace Shapes
{
    class Point
    {
        // attributes that store the information about the point
        // they represent the x and y coordinates

        private int x;
        private int y;

        // Constructor method: it sets the attributes x and y to the
        // values xarg and yarg that are passed when a new object is
        // created

        public Point(int xarg, int yarg)
        {
            x = xarg;
            y = yarg;
        }

        // Method that simply prints on the Console the coordinates of
        // the Point, i.e., the values of the attributes x and y

        public void Display()
        {
            Console.WriteLine($"[{x}, {y}]");
        }
    }
}
```

Program.cs

```
using System;

namespace Shapes
{
    class Program
    {
        static void Main(string[] args)
        {
            // Creating two object instances of the class Point
            // using the defined constructor
            // Constructor to invoke: public Point(int xarg, int yarg)
```

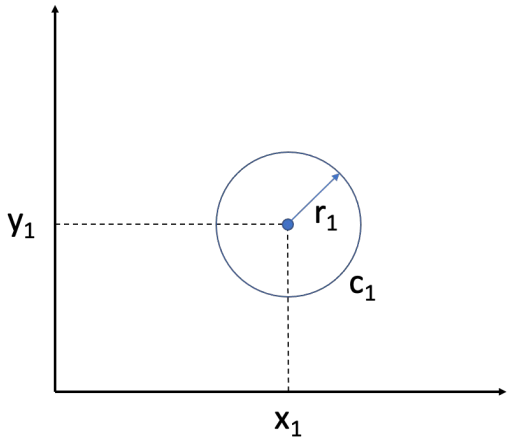
```
            Point p1 = new Point(5, 1);
            Point p2 = new Point(7, 2);

            // Invoking the Display method defined in the class Point
            // on each of the created object instances (p1 and p2)

            p1.Display();
            p2.Display();
        }
    }
}
```

2: Circle class (with Tutor)

Think about the possible design of a class to represent circles, as it was briefly discussed in the lecture.



For instance:

```
Point p1 = new Point(6, 4);
double r1 = 1.0;

Circle c1 = new Circle(p1, r1)
```

A *Circle* object should include a `Display()` method that prints the coordinates of its centre and the value of the radius. Remember that there exists a relationship between *Circle* and *Point* objects. As such, the above object *c1* can "send a message" to the object *p1* to request to display the coordinates of that point—this can be done without *c1* having to access *p1*'s attributes directly (hint: the `Display()` method of the *Circle* can invoke the `Display()` method of the *Point* object representing its centre).

Moreover, the *Circle* class should include two additional methods for calculating and printing a circle's *circumference* and *area*.

The *Circle* class should be added to the project of exercise 1, and at least two *Circle* objects should be instantiated in the *Main entry point* created (in the *Program* class). Read the coordinates of the centre and the radius of each circle as input from the keyboard. Show the features of the created *Circle* objects, i.e., their *circumference* and *area,* via invoking the above-mentioned methods.

Solution

As for the previous exercise, a new class must be added to the project. The class name should be Circle. This will create a new file Circle.cs, inside the project.

Circle.cs

```
using System;
```

```
namespace Shapes
{
    class Circle
    {
        // attributes of the Circle class
        // A circle is identified by a centre and a radius.
        // Please note that centre is an object of the class Point.
        // The radius is a primitive double value.

        private Point centre;
        private double radius;

        // Constructor method that initialises the attributes with
        // the provided arguments c and r

        public Circle(Point c, double r)
        {
            centre = c;
            radius = r;
        }

        // Prints the information of the circle
        // The coordinates of the centre are printed via invoking
        // the Display method of the 'centre' attribute as this is
        // an object of the class Point.

        public void Display()
        {
            Console.WriteLine("Center: ");
            centre.Display();
            Console.WriteLine($"Radius: {radius}");
        }

        // Calculates the Area of the circle by using the
        // 'radius' attribute

        public void Area()
        {
            Console.WriteLine(Math.PI * radius * radius);
        }

        // Calculates the circumference of the circle by using the
        // 'radius' attribute

        public void Circumference()
        {
            Console.WriteLine(2 * Math.PI * radius);
        }
    }

}
```

Program.cs

```
using System;

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

```
            int x;
            int y;
            double radius;

            Console.WriteLine("Insert the coordinates of the centre of Circle
1: ");
            x = Convert.ToInt32(Console.ReadLine());
            y = Convert.ToInt32(Console.ReadLine());
            Point p1 = new Point(x, y);

            Console.WriteLine("Insert the value of the radius of Circle 1: ");
            radius = Convert.ToDouble(Console.ReadLine());
            Circle c1 = new Circle(p1, radius);

            Console.WriteLine("Insert the coordinates of the centre of Circle
2: ");
            x = Convert.ToInt32(Console.ReadLine());
            y = Convert.ToInt32(Console.ReadLine());
            Point p2 = new Point(x, y);

            Console.WriteLine("Insert the value of the radius of Circle 2: ");
            radius = Convert.ToDouble(Console.ReadLine());
            Circle c2 = new Circle(p2, radius);

            c1.Display();
            c1.Circumference();
            c1.Area();

            c2.Display();
            c2.Circumference();
            c2.Area();
        }
    }
}
```

3: Distance between two Points (independent work)

Modify the definition of the *Point* class (of exercise 1) to include the following method:

```
public void DistanceFrom(Point p2)
```

The method should print on the screen the distance between the point on which it is invoked and the *Point* object passed as the argument.
The formula to calculate the distance between two points (x_1, y_1) and (x_2, y_2) in a two-dimensional plane is:

$$\sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$$

To implement the formula, use the methods `Math.Sqrt` and `Math.Pow`. To access an attribute of a class, use the same *dot* notation we have been using for the methods. Use the following code as a starting point and replace the green text with the proper attributes of the objects.

```
public void DistanceFrom(Point p2)
{
    double distance = Math.Sqrt(Math.Pow(replace: x2 – x1, 2) +
                      Math.Pow(replace: y2 – y1, 2));

    Console.WriteLine("Distance is: " + distance);
}
```

Are you able to access the x and y coordinates of the p2 object using *p2.x* and *p2.y* from the body of the `DistanceFrom` method? Why?
Test the `DistanceFrom(Point p2)` method from the Main entry point of the *Program* class as in the previous exercises.

Solution.

The class *Point.cs* of exercise 1 needs to be modified to include the following additional method definition:

```
public void DistanceFrom(Point p2)
{
    double distance = Math.Sqrt(Math.Pow(p2.x - x, 2) + Math.Pow(p2.y - y,
2));
    Console.WriteLine("Distance is: " + distance);
}
```

The `DistanceFrom` method in the *Point* class calculates and prints the Euclidean distance between the current *Point* object (i.e., the object on which it is invoked) and another *Point* object *p2* taken as a parameter. It calculates the Euclidean distance between the two points—with coordinates *(x1, y1)* and *(x2, y2)*—using the formula:

$$\sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$$

In this case, *x1* and *y1* are the attributes *x* and *y* of the current object, and *x2* and *y2* are the attributes *x* and *y* of the *p2* object. The attributes *x* and *y* of *p2* are private. However, they are accessible by the `DistanceFrom` method because both the method and the attributes are encapsulated within the (same) *Point* class.

In object-oriented programming, private members (attributes or methods) are encapsulated within objects and are not accessible directly from objects of other classes. However, they are accessible from objects of the same class. This is a fundamental concept of encapsulation, which helps control access to the internal state of an object and ensures that data integrity is maintained.

In the `DistanceFrom` method, `Math.Pow` calculates the squares of the differences of the x and y coordinates of the two points. Specifically, it computes (p2.x - x) squared and (p2.y - y) squared. These squared differences are added together and then the `Math.Sqrt method calculates` the square root, which gives the actual Euclidean distance.

To use this method (from the Main):

```
Point p4 = new Point(5, 1);
Point p5 = new Point(7, 2);
p4.DistanceFrom(p5);      // prints Distance is: 2.23606797749979
```

1. Two *Point* objects, *p4* and *p5*, are created and initialised with the coordinates (5, 1) and (7, 2) respectively.
2. The `DistanceFrom` method is called on the *p4* object with *p5* as its argument.
3. Inside the `DistanceFrom` method, the distance between the current *Point* object (*p4*, with coordinates (5, 1)) and the *p2* object (*p5*, with coordinates (7, 2)) is calculated using the Euclidean distance formula.
4. The differences in x and y coordinates are calculated as follows:
    - (p2.x - x) is (7 - 5), which equals 2.
    - (p2.y - y) is (2 - 1), which equals 1.
5. The squared differences are calculated:
    - $(p2.x - x)^2$ is 2^2, which is 4.
    - $(p2.y - y)^2$ is 1^2, which is 1.
6. These squared differences are added together: 4 + 1, which is 5.
7. The square root of the sum is calculated: $\sqrt{5}$, which is approximately 2.236.
8. The calculated distance, approximately 2.236, is printed to the console.

4: Segment Class

Try to design a class that models segment objects, e.g.:

```
Point p1 = new Point(2, 3);
Point p2 = new Point(3, 4);

Segment s1 = new Segment(p1, p2)
```

The class should include a method `Length()` that prints the length of a *Segment* object. As in the previous exercise, use the dot notation to access an attribute of a *Point* object. Can you still access the x and y coordinates of the p2 object using *p2.x* and *p2.y*? To solve the problem, consider the `DistanceFrom` method defined in the *Point* class in the previous exercise. Can it be used to calculate the length of a segment?

Solution

Segment.cs

If you tried to implement the *Segment* class as follows, it would not work.

```
using System;

namespace Shapes
{
    class Segment
    {
        private Point start;
        private Point end;

        public Segment(Point s, Point e)
        {
            start = s;
            end = e;
        }

        public void Length()
        {
            // does not work because x and y are private attributes of the
Point class
            double length = Math.Sqrt(Math.Pow(end.x - start.x, 2) +
                            Math.Pow(end.y - start.y, 2));
            Console.WriteLine("Length is: " + length);
        }

    }
}
```

The reason is that you are attempting to access the attributes of the *Point* class from (an object of) another class. This would not work because those attributes are *private*. A possible way to access the values of the attributes is to create some public methods that read and *return* them. We will see this approach next week.

A reasonable implementation of the method `Length` may be based on the method `DistanceFrom` we have already implemented within the *Point* class. This would allow reusing the same code already defined there:

```
public void Length()
{
    Console.WriteLine("Length is equal to distance!");
    start.DistanceFrom(end);
}
```