برنامه نویسی پیشرفته
C#

۹ مهر ۹۸

ملکی مجد

مباحث

- کوییز

- تعریف کلاس شامل متد و داده
- کنترل دسترس پذیری اعضای کلاس با کلمه های کلیدی public و private
- ساختن یک شی با استفاده از کلمه کلیدی new برای فراخوانی constructor
- نوشتن و صدا زدن constructor
- متد و داده های مختص به کلاس (static)

کوییز

برنامه ای بنویسید که از کاربر عدد n (کمتر از ۱۰۰) را بگیرد
سپس در ادامه n عدد از کاربر دریافت کند
اعداد را از آخر به اول در یک خط خروجی چاپ کند

همچنین متدی بنویسید که با دریافت یک آرایه و تعداد اعضای آن کمترین مقدار آرایه را برگرداند
با فراخوانی متدی که نوشته اید کمترین مقدار دریافت شده از ورودی را چاپ کنید.

class

- Class

  systematically arrange information and behavior into a meaningful entity

- Encapsulation
  - The idea is that a program that uses a class should not have to account for how that class actually works internally
  - the program simply creates an instance of a class and calls the methods of that class.
- two purposes of Encapsulation
  - combine methods and data within a class
  - control the accessibility of the methods and data

Define class

```
class Circle
{
    int radius;

    double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

Create object

```
class Circle
{
    int radius;

    double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

```
Circle c;           // Create a Circle variable
c = new Circle();   // Initialize it

int i;
i = 42;
```

Object assignment

```
class Circle
{
    int radius;

    double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

```
Circle c;
c = new Circle();
Circle d;
d = c;
```

Controlling accessibility

- private (default)
- public

```
class Circle
{
    private int radius;

    public double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

Constructor

- When you use the *new* keyword to create an object, the runtime needs to construct that object by using the definition of the class
  - runtime grab a piece of memory
  - fill it with the fields defined by the class
  - invoke a constructor to perform any initialization required
- A *constructor* is a special method that runs automatically when you create an instance of a class
  - same name as the class
  - cannot return a value
- Every class must have a constructor. If you don't write one, the compiler automatically generates a default constructor for you

Constructor - default

```
class Circle
{
    private int radius;

    public Circle()  // default constructor
    {
        radius = 0;
    }

    public double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

Constructor - private

- If keyword public is omitted, the constructor will be private (just like any other method and field).
- If the constructor is private, it cannot be used outside the class

- beyond the scope of the current discussion

Use constructor and public method

```
Circle c;
c = new Circle();
double areaOfCircle = c.Area();
```

Overloading constructors

```
class Circle
{
    private int radius;

    public Circle()  // default constructor
    {
        radius = 0;
    }

    public Circle(int initialRadius) // overloaded constructor
    {
        radius = initialRadius;
    }

    public double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

Overloading constructors

```
class Circle
{
    private int radius;

    public Circle()  // default constructor
    {
        radius = 0;
    }

    public Circle(int initialRadius) // overloaded constructor
    {
        radius = initialRadius;
    }

    public double Area()
    {
        return Math.PI * radius * radius;
    }
}
```

```
Circle c;
c = new Circle(45);
```

Writing default constructor

- if you write your own constructor for a class, the compiler does not generate a default constructor. Therefore, if you've written your own constructor that accepts one or more parameters and you also want a default constructor, you'll have to write the default constructor yourself.

static methods and data

- Consider class Math
  - Math m = new Math();
  - double d = m.Sqrt(42.24);
  - double d = Math.Sqrt(42.24);
  - Math.PI
- In C#, all methods must be declared within a class. However, if you declare a method or a field as *static*, you can call the method or access the field by using the name of the class.

Creating a shared field

- Defining a field as static makes it possible for you to create a single instance of a field that is shared among all objects created from a single class
  - Nonstatic fields are local to each instance of an object

Example of shared field

```
class Circle
{
    private int radius;
    public static int NumCircles = 0;

    public Circle()  // default constructor
    {
        radius = 0;
        NumCircles++;
    }

    public Circle(int initialRadius) // overloaded constructor
    {
        radius = initialRadius;
        NumCircles++;
    }
}
```

Example of shared field

```
class Circle
{
    private int radius;
    public static int NumCircles = 0;

    public Circle()  // default constructor
    {
Console.WriteLine($"Number of Circle objects: {Circle.NumCircles}");
    }

    public Circle(int initialRadius) // overloaded constructor
    {
        radius = initialRadius;
        NumCircles++;
    }
}
```

Creating a static field by using the const keyword

- By prefixing the field with the *const* keyword, you can declare that a field is static but that its value can never change.
  - you can declare a field as *const* only when the field is a numeric type (such as *int* or *double*), a string, or an enumeration

```
class Math
{
    ...
    public const double PI = 3.14159265358979;
}
```

static classes

- A static class can contain only static members
- The purpose of a static class is purely to act as a holder of utility methods and fields

static using statements

- Whenever you call a static method or reference a static field, you must specify the class to which the method or field belongs, such as *Math.Sqrt*
- Static *using* statements enable you to bring a class into scope and omit the class name when accessing static members

```
using static System.Math;
using static System.Console;
...
var root = Sqrt(99.9);
WriteLine($"The square root of 99.9 is {root}");
```