FUNDAMENTALS OF PROGRAMMING WITH C#

RANDOM OBJECTS

Liu Fan
isslf@nus.edu.sg

Total:15

Objectives

- Write a program that uses Random object to generate random numbers

Agenda

- Using Random Class
- Introduction of Class and Object

Generating Random Integer

- The code below generate random integer.
- We use System.Random class to do that
- There are two versions of Next method to generate random integer between
  - 0 to N (0 <= number < N)
  - Lower bound to upper bound (lower <= number < upper bound)

```
Random rnd = new Random();
Console.WriteLine(rnd.Next(5)); //generate 0<=random number <5
Console.WriteLine(rnd.Next(5)); //another random number
Console.WriteLine(rnd.Next(10, 20)); // generate 10<=random number< 20
Console.WriteLine(rnd.Next(10, 20)); //another random number
```

https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.7.2

Generating Random Double

- We can use NextDouble() method to generate random double that is >= 0.0 and < 1.0
- How do we generate random double between lower bound and upper bound?

```
Random rnd = new Random();
Console.WriteLine(rnd.NextDouble());       //between 0 and 1
Console.WriteLine(rnd.NextDouble() * 10); //between 0 and 10
Console.WriteLine(100 + rnd.NextDouble()
                      * (150-100)); // between 100 and 150
```

Classes and objects

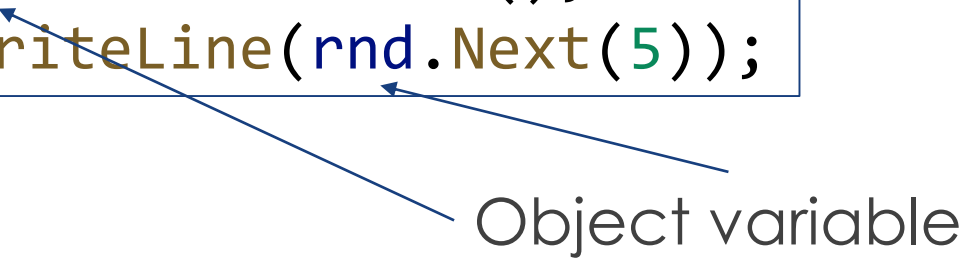
```
Random rnd = new Random();
Console.WriteLine(rnd.Next(5));
```

Class name

```
Console.WriteLine(Math.Sqrt(5));
```

Classes and objects

```
Random rnd = new Random();
Console.WriteLine(rnd.Next(5));
```




```
Console.WriteLine(Math.Sqrt(5));
```

- We see the difference in the way we call the `Sqrt()` method and `Next()` method
- `Math.Sqrt()` is a static method. We call the method by referring to the class name
- `Random.Next()` is not a static method. We can call the method only after we instantiate (create) a Random object and refer to the variable containing the object.

Classes and Objects

- Objects has to be created using `new` keyword

```
Random rnd = new Random();
```

- The above code means that we are creating a new `Random` object
  - `Random` can be considered the type of this object or we call it the class.
  - Other way to say this is that `rnd` variable contain an object that is an instance of Random class.

Classes and Objects

- Classes are the blueprint of objects
  - Real-life analogy: design of a car vs. actual cars that you own
  - Student class: represent what can be done to a student record in the system
    Student object: represent each of the student records, each contains different information
- Classes are the concepts/classifications of objects
  - Male and female as concepts, individual people are instances of these concepts

Classes and Objects

- Classes sometimes is also used to represent things that is treated as singular
  - Real-life example: the world, earth, justice
  - In system: Console, Math

- Classes and objects have properties to store the information pertaining the class/object.
  - Two object of the same class can have different information e.g. two student records contains different names and student IDs
  - It's more common for objects to have properties than classes.

Classes and Objects

- Classes and objects can have methods
  - Methods can be attached at the class level (static methods
  - Methods can work on the object itself (non static methods

Summary

- We have learned how to use Random class
- We have to instantiate a Random object, store it in a variable and then use the method afterwards
- We have covered basic understanding of classes and objects
  - Hopefully it make things clearer when you read C# codes online

APPENDIX(NOT REQUIRED)

- public by itself means this is an instance-based member that is accessible to external callers (those with access to the type itself).

- static by itself means the member is not instance-based: you can call it without needing any particular instance (or even any instance at all); without an accessibility qualifier, non-public is assumed - so the member will not be accessible to external callers.

- public static is a static method that is accessible to external callers.

- Memory usage is identical in both cases: any variables declared in the method are scoped to the method-call itself (as an implementation detail: via the stack; also: I'm assuming no "captured variables", and no async or yield usage),

- Accessibility:
  - none specified: defaults to "private" (or "internal" for outer-classes)
  - "private": only available to code inside that type
  - "protected": available to code inside that type or sub-types
  - "internal": available to code in the same assembly
  - "protected internal": either "protected" **or** (union) "internal"
  - "public": available to all callers with access to the type
- Static / etc:
  - none specified: instance-based; an instance is required, and code has automatic access to instance-members (via this.) and static members
  - "static": no instance is required; code has automatic access to static members only