The `static` Keyword

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Static methods
  - Declaring and using
  - Restrictions on static methods
- Static variables
  - Constants

Calling Methods on Classes

- Methods are usually called on an object (instance of a class)

```
Rectangle rect = new Rectangle();
rect.SetLength(12);
```

A Rectangle object

Method in Rectangle class

- Sometimes we call a method using the name of the class:

Class, not an object

```
Console.WriteLine("Hello!");
Array.Resize(ref myArray, 6);
```

How does that work?

Declaring Methods with `static`

- Somewhere in the standard library:

```
class Console
{
   public static void WriteLine(string value)
   {
      ...
   }
}
```

- `static` keyword: This method "belongs to" the class, not an instance of the class – you don't need an instance to call it

Static and Instances

- Non-static methods apply to a particular instance, can modify it

```
rect.SetLength(12);
```

```
class Rectangle
{
  private int length;
  private int width;
  public void SetLength(int lengthParameter)
  {
    length = lengthParameter;
  }
```

length stored in rect is modified

- Static methods do not affect any instance

```
Console.WriteLine("Hello!");
```

No Console object to modify

Static and Instances

- Static methods cannot access instance variables

```
class Rectangle
{
  private int length;
  private int width;
  public void SetLength(int lengthParameter)
  {
    length = lengthParameter;
  }
  public static int ComputeArea()
  {
    return length * width; ❌
  }
```

```
rect.SetLength(12);
```

```
Rectangle.ComputeArea();
```

Which Rectangle's length should this use?

AUGUSTA UNIVERSITY

Why Use Static Methods?

- Functions that don't need any object state
  - Input/output with parameters and return value
- Group together related or similar functions
- In class `Math`:

```
public static double Pow(double x, double y)
```

Computes and returns x^y

```
public static double Sqrt(double x)
```

Computes and returns $\sqrt{x}$

```
public static int Max(int x, int y)
```

Returns the parameter that is larger

```
public static int Min(int x, int y)
```

Returns the parameter that is smaller

Answering An Old Question

- Why is `Main` declared like this?

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

- Static methods can be called without an object instance
- When program starts, no objects exist yet
- .NET runtime can call `Program.Main` without creating an object

Calling Static Methods

- Normal static method call: `ClassName.MethodName(args)`
- Can call a static method *within the same class* using just `MethodName(args)`

```
class Array
{
  public static void Copy(Array source, Array dest, int length)
  {
    ...
  }     Not Array.Copy
  public static void Resize<T>(ref T[] array, int newSize)
  {
    T[] newArray = new T[newSize]
    Copy(array, newArray, Math.Min(array.Length, newSize));
    array = newArray;
  }
```

Methods to "Help" Main

- Class that contains Main can have other methods too
- Static methods can be called from Main
- Define separate parts of program, or reuse code multiple times

```
class Program
{
  static void Main(string[] args)
  {
    int result = DoPart1(25, 35);
    DoPart2("Hello", result);
  }
  static int DoPart1(int a, int b)
  {
    ...
  }
  static void DoPart2(string x, int y)
  {
    ...
  }
```

Outline

- Static methods
  - Declaring and using
  - Restrictions on static methods
- **Static variables**
  - Constants

Static Variables

```
class Rectangle
{
  public static int NumRectangles = 0;
```

In Program.cs:

```
Rectangle rect = new Rectangle();
Rectangle.NumRectangles++;
```

- Stored with the class, not an object instance
- Only one copy in entire program
- Any object’s method will access the same variable
- Can be public, since there is no object to “encapsulate” within

Counting Instances

```
class Rectangle
{
  public static int NumRectangles = 0;
  private int length;
  private int width;
  public Rectangle(int lengthP, int widthP)
  {
    length = lengthP;
    width = widthP;
    NumRectangles++;
  }
}
```

Shared by all Rectangle objects; stored with the Rectangle class

Increments the single shared counter

In Program.cs:

```
Rectangle rect1 = new Rectangle(2, 4);
Rectangle rect2 = new Rectangle(7, 5);
Console.WriteLine(Rectangle.NumRectangles
  + " rectangle objects have been created");
```

Revisiting Constants

- Recall: const keyword defines a constant

```
class Calendar
{
  public const int MONTHS = 12;
  private int currentMonth;
}
```

```
static void Main(string[] args)
{
  int numMonths = Calendar.MONTHS;
  Calendar myCal = new Calendar();
}
```

- These are also static variables
  - Only one copy for entire program
  - Stored with the class, accessed using the class name

Static Methods and Variables

- Static methods *can* access static variables

```
class Rectangle
{
  private static int NumRectangles = 0;
  public static int GetNumRectangles()
  {
    return NumRectangles;
  }
}
```

Incremented in constructor, as before

Can't access length or width, but can access NumRectangles

```
Rectangle rect1 = new Rectangle(2, 4);
Rectangle rect2 = new Rectangle(7, 5);
Console.WriteLine(Rectangle.GetNumRectangles()
  + " rectangle objects have been created");
```

Static Access Summary

- Static variables and instance variables are both **fields** of a class

Can Access?

	Static Field	**Non-Static Field**
Static method	Yes	No
Non-static method	Yes	Yes

Static Class Declarations

- If a class is declared `static`, all its members must be static
- Useful for "utility library" classes, like `System.Math`

```
static class Math
{
  public static double Sqrt(double x)
  {
    ...
  }
  public static double Pow(double x, double y)
  {
    ...
  }
}
```