Outline

- Instance variables and default values

- Constructors

  - Definition and usage

  - Multiple constructors

- ToString Methods

Remember This Lab Activity?

```
class Program
{
    static void Main(string[] args)
    {
        Rectangle myRect = new Rectangle();
        Console.WriteLine($"Length is {myRect.GetLength()}");
        Console.WriteLine($"Width is {myRect.GetWidth()}");
    }
}
```
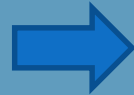
No SetLength or SetWidth

Output:
```
Length is 0
Width is 0
```

AUGUSTA UNIVERSITY

Variables and Default Values

- **Local** variables have **no** default value: you must assign them a value before using them

```
int myVar1;
int myVar2 = myVar1 + 5;
```
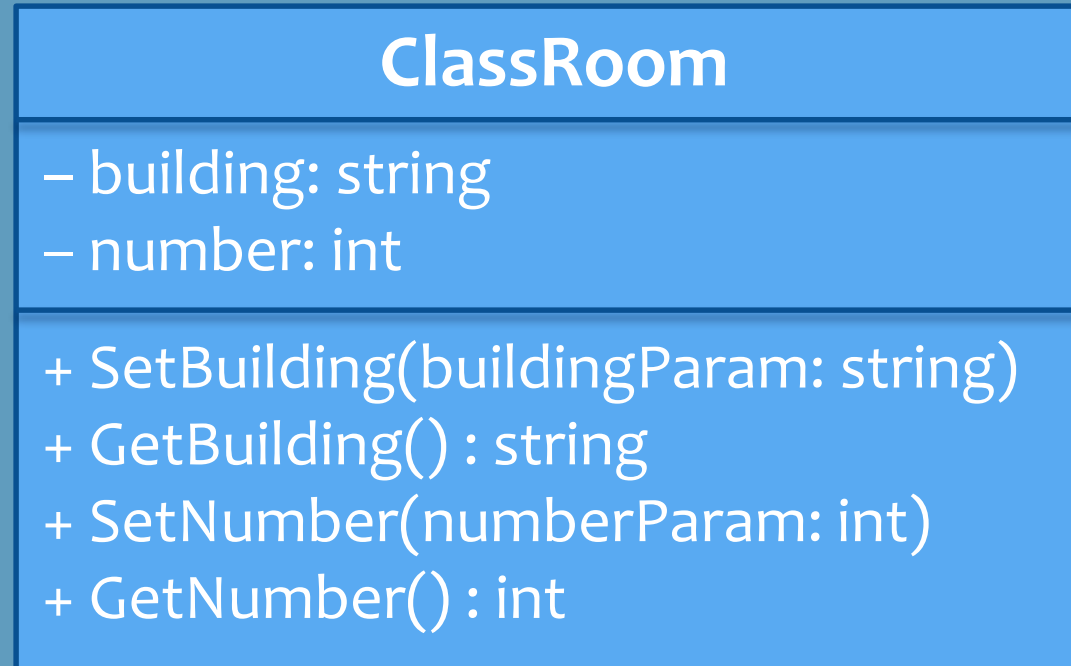
→ Error! Can't use unassigned variable myVar1

- **Instance** variables (in an object) have default values:

Type	Default Value
Numeric types	0
string	null
bool	false
char	'\0'

Example Class: ClassRoom

- UML diagram for the class:

ClassRoom
– building: string – number: int
+ SetBuilding(buildingParam: string) + GetBuilding() : string + SetNumber(numberParam: int) + GetNumber() : int

ClassRoom Implementation

```csharp
class ClassRoom
{
    private string building;
    private int number;

    public void SetBuilding(string buildingParam)    ← Set accessor for building
    {
        building = buildingParam;
    }
    public string GetBuilding()    ← Get accessor for building
    {
        return building;
    }
```

AUGUSTA
UNIVERSITY

ClassRoom Implementation

```
    public void SetNumber(int numberParam)    ← Set accessor for number
    {

        number = numberParam;

    }
    public int GetNumber()                     ← Get accessor for number
    {

        return number;

    }
}
```

AUGUSTA UNIVERSITY

Default Values for ClassRoom

```csharp
static void Main(string[] args)
{

    ClassRoom english = new ClassRoom();
    Console.WriteLine($"Building is {english.GetBuilding()}");
    Console.WriteLine($"Room number is {english.GetNumber()}");
}
```

What will this print?

A null string prints nothing

Output:

```
Building is
Room number is 0
```

AUGUSTA
UNIVERSITY

Outline

- Instance variables and default values
- **Constructors**
  - Definition and usage
  - Multiple constructors
- ToString Methods

AUGUSTA
UNIVERSITY

Object Instantiation

- Look carefully at instantiation syntax:

```
ClassRoom english = new ClassRoom();
```

Parentheses, just like a method call
e.g. GetBuilding()

- Instantiation does call a method: the **constructor**

- Constructor: A method that creates an instance of an object

- If you don't write one, C# generates a "default" constructor

AUGUSTA
UNIVERSITY

Constructor Syntax

- Method name must equal class name

- No return type, not even `void`
  - Output of method is always an instance of the class

Parameter type   Parameter name

```
class ClassRoom
{
   public ClassRoom(string buildingParam, int numberParam)
   {
      //body of constructor goes here…
   }
}
```

Access
modifier

comma   second parameter

AUGUSTA
UNIVERSITY

Constructor Implementation

- Constructor "sets up" object
- Body of constructor: assign values to all instance variables

```
class ClassRoom
{
    public ClassRoom(string buildingParam, int numberParam)
    {
        building = buildingParam;
        number = numberParam;
    }
}
```

Instance variables hidden to save space

no return statement – return value is "this object"

AUGUSTA
UNIVERSITY

Constructor Usage

- Instantiation calls a constructor
- Just like other method calls, arguments go in parentheses

Instantiation with new

First argument: building

Second argument: number

```csharp
static void Main(string[] args)
{
    ClassRoom csci = new ClassRoom("Allgood East", 356);
    Console.WriteLine($"Building is {csci.GetBuilding()}");
    Console.WriteLine($"Room number is {csci.GetNumber()}");
}
```

Output:
```
Building is Allgood East
Room number is 356
```

AUGUSTA UNIVERSITY

Multi-Parameter Methods

- Can use same syntax for ordinary methods, e.g. in Rectangle:

```
public void MultiplyBoth(int lengthFactor, int widthFactor)
{
    length *= lengthFactor;
    width *= widthFactor;
}
```

- Use it like this:

```
myRect.SetLength(5);
myRect.SetWidth(10);
myRect.MultiplyBoth(3, 5);
```

Now myRect has length 15 and width 50

AUGUSTA UNIVERSITY

Multi-Parameter Methods

- Order of arguments matters

```
myRect.MultiplyBoth(3, 5);
```
```
myRect.MultiplyBoth(5, 3);
```

lengthFactor   widthFactor

lengthFactor   widthFactor

- Types must match

```
ClassRoom csci = new ClassRoom(356, "Allgood East");
```
⟹ Error!

buildingParam, must be a string

numberParam, must be an int

AUGUSTA
UNIVERSITY

Outline

- Instance variables and default values

- Constructors

  o Definition and usage

  o **Multiple constructors**

- ToString Methods

AUGUSTA
UNIVERSITY

C# Constructor Rules

- If you don't write a constructor, C# generates a "default" one
  - o Sets instance variables to their default values
- If you do write a constructor, no "default" constructor is generated
  - o Now that we've written a ClassRoom constructor, this doesn't work:

```
ClassRoom csci = new ClassRoom();
```
➡️ Error! Constructor requires 2 arguments

- What if we still want the no-argument constructor?

AUGUSTA UNIVERSITY

Multiple Constructors

```csharp
class ClassRoom
{
  public ClassRoom(string buildingParam, int numberParam)
  {
    building = buildingParam;
    number = numberParam;
  }
  public ClassRoom()
  {
    building = null;
    number = 0;
  }
}
```

Instance variables hidden to save space

Constructor with no parameters

Same as C#'s default constructor

AUGUSTA UNIVERSITY

Which Constructor is Called?

- Instantiation calls the constructor that matches the arguments

```
static void Main(string[] args)
{
    ClassRoom csci = new ClassRoom("Allgood East", 356);
    //csci has building = "Allgood East" and number = 356
    ClassRoom english = new ClassRoom();
    //english has building = null and number = 0;
}
```

string, int: Matches first constructor

No arguments: matches second constructor

AUGUSTA UNIVERSITY

Outline

- Instance variables and default values
- Constructors
  - Definition and usage
  - Multiple constructors
- **ToString Methods**

AUGUSTA
UNIVERSITY

Converting Numbers to Strings

- Recall: String interpolation uses `ToString` "behind the scenes"
- `ToString()` method returns the object converted to a string

Result: "42"

```csharp
int num = 42;
string intText = num.ToString();
Console.WriteLine($"num is {num}");
Console.WriteLine($"num is {num.ToString()}");
```

This is the same as this

- C# datatypes already have `ToString()` defined, but your classes need their own `ToString()`

Writing ToString

- Header of a ToString method is always the same

Keyword override: ToString is
defined in parent class object

Access must
be public

Return type   No parameters

```
class ClassRoom
{
    public override string ToString()
    {
        return building + " " + number;
    }
}
```

Automatically calls
number.ToString()

String concatenation

- Body of ToString: return a string representation of the object

AUGUSTA
UNIVERSITY

Using ToString

- ToString() will be called automatically when your object needs to be converted to a string

- Can also call it "explicitly" like any other method

```csharp
static void Main(string[] args)
{
    ClassRoom csci = new ClassRoom("Allgood East", 356);
    Console.WriteLine(csci);
    Console.WriteLine($"The classroom is {csci}");
    Console.WriteLine("The classroom is " + csci.ToString());
}
```

Summary

- Instance variables and default values
- Constructors
  - Definition and usage
  - Multiple constructors
- ToString Methods

AUGUSTA
UNIVERSITY