Constructors and ToString Methods

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Instance variables and default values
- Constructors
    - Definition and usage
    - Multiple constructors
    - Constructors in UML
- 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
```

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

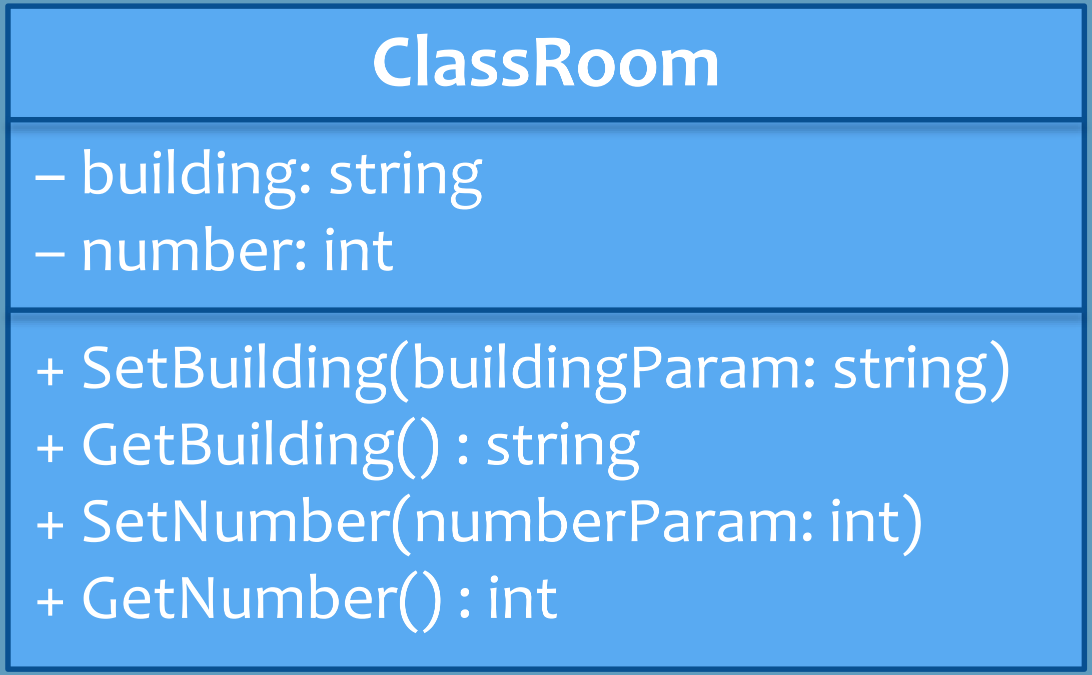
- **Instance** variables (in an object) have default values:

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

AUGUSTA UNIVERSITY

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

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

  public void SetBuilding(string buildingParam)
  {
    building = buildingParam;
  }
  public string GetBuilding()
  {
    return building;
  }
```

Set accessor for building

Get accessor for building

ClassRoom Implementation

```
  public void SetNumber(int numberParam)
  {
    number = numberParam;
  }
  public int GetNumber()
  {
    return number;
  }
}
```

Set accessor for number

Get accessor for number

Default Values for ClassRoom

```
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
```

Outline

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

Object Instantiation

- Look carefully at instantiation syntax:

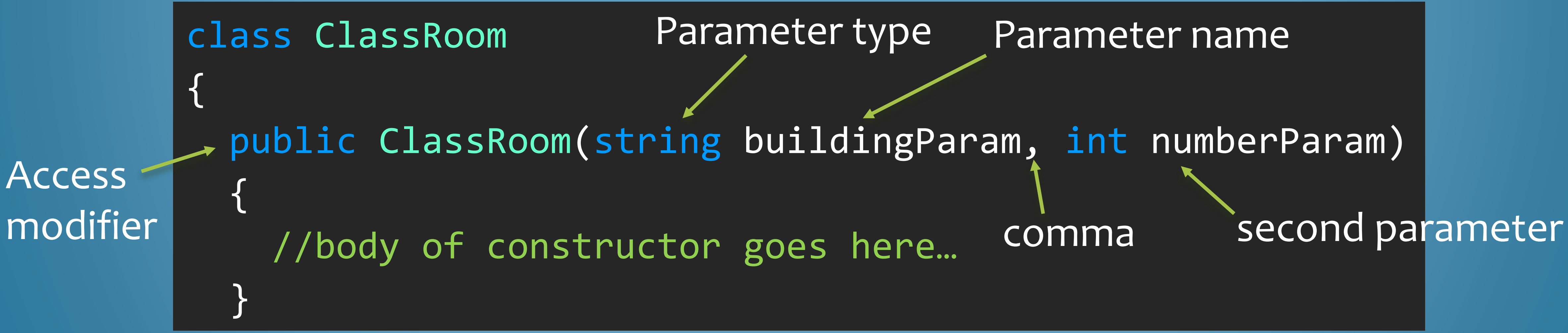
```
ClassRoom english = new ClassRoom();
```

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

Constructor Syntax

- Method name must equal class name
- No return type, not even void
    - Output of method is always an instance of the class



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

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

```
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);
```

lengthFactor widthFactor

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

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
    - Definition and usage
    - **Multiple constructors**
    - Constructors in UML
- ToString Methods

C# Constructor Rules

- If you don't write a constructor, C# generates a "default" one
  - Sets instance variables to their default values
- If you do write a constructor, no "default" constructor is generated
  - 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?

Multiple Constructors

```
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

Constructors and Default Values

- Any instance variable not initialized will get its default value

```
public ClassRoom()
{
  building = null;
  number = 0;
}
```

is the same as

$\longrightarrow$

```
public ClassRoom()
{
}
```

```
public ClassRoom()
{
  building = "Unknown";
}
```

Result:

$\longrightarrow$

building is “Unknown”
number is 0

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

Writing a Constructor

- Add a parameter for each attribute that needs an initial value
- Assign parameters to instance variables, or provide a “sensible” default
- How would we add a constructor to Rectangle?

```java
public Rectangle(int initLength, int initWidth)
{
  length = initLength;
  width = initWidth;
}
```

Order matters!

`new Rectangle(6, 4)`

`≠ new Rectangle(4, 6)`

"Partial" Constructors

- Constructors don't *need* 1 parameter per instance variable
- Consider the Account class:
- 2 attributes; "standard" constructor would have 2 parameters

```
class Account
{
  private decimal balance;
  private string ownerName;
```

- Can also write a constructor with 1 parameter
  - 1 attribute needs to be initialized, the other can get a default value

```
public Account(string name)
{
  ownerName = name;
}
```

balance gets the default value `0`

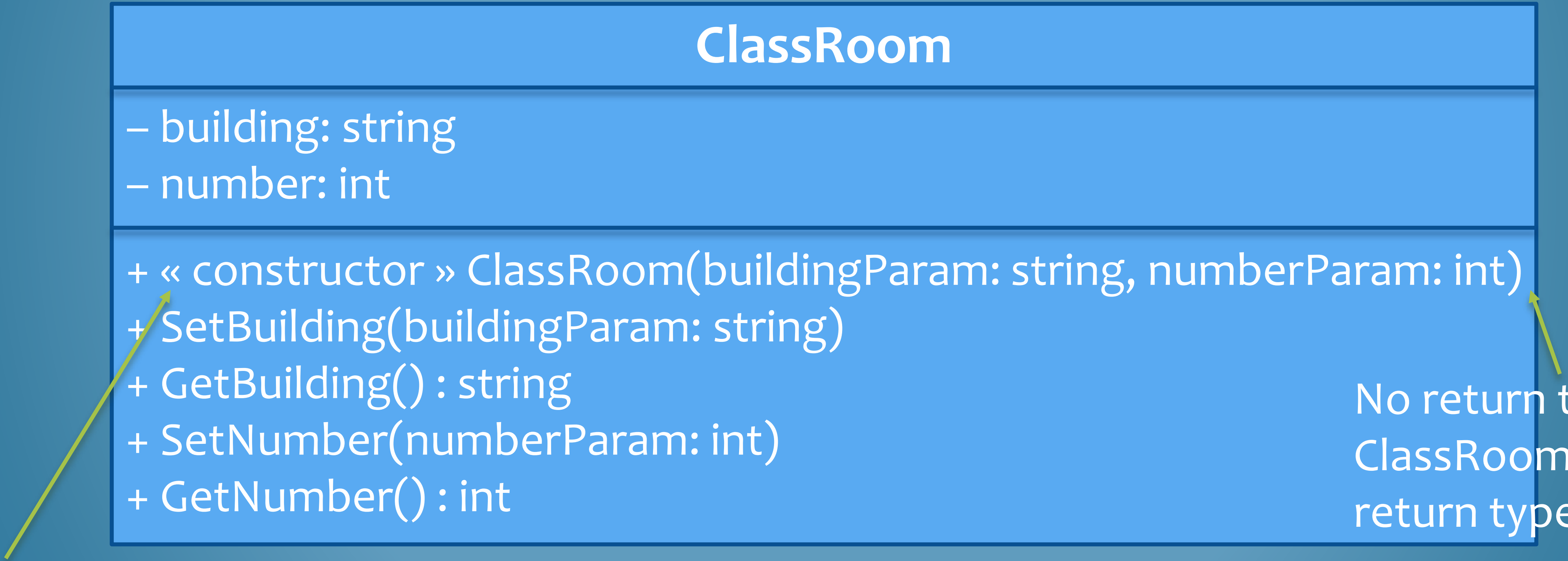
AUGUSTA UNIVERSITY

Outline

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

Constructors: Part of the Interface

- Non-default constructors should be planned in UML



ClassRoom

– building: string
– number: int

+ « constructor » ClassRoom(buildingParam: string, numberParam: int)
+ SetBuilding(buildingParam: string)
+ GetBuilding() : string
+ SetNumber(numberParam: int)
+ GetNumber() : int

No return type; ClassRoom is the return type

Constructor annotation; not really necessary

Outline

- Instance variables and default values
- Constructors
  - Definition and usage
  - Multiple constructors
  - Constructors in UML
- **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"

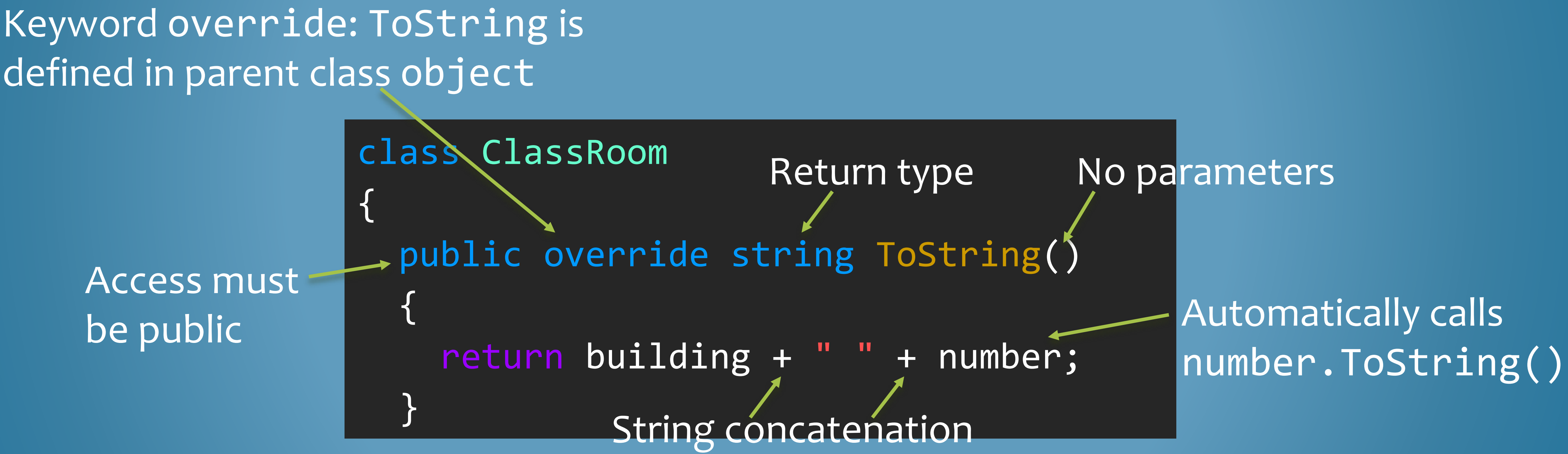
```
int num = 42;
string intText = num.ToString();
Console.WriteLine($"num is {num}");
Console.WriteLine($"num is {num.ToString()}");
```

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



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

Keyword override: ToString is defined in parent class object
Return type
No parameters
Access must be public
Automatically calls number.ToString()
String concatenation

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

Writing ToString

- Goal of ToString: Produce "human-readable" information
- Include all attributes of object

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

```
class Rectangle
{
  private int length;
  private int width;
  public override string ToString()
  {
    return $"{length} x {width}"
      + " rectangle";
  }
```

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

```
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());
}
```

No $ necessary

Concatenation

AUGUSTA UNIVERSITY

Using ToString

- If written well, makes displaying output much easier

```
ClassRoom csci = new ClassRoom("Allgood East", 356);
Console.WriteLine("Your classroom is "
  + $"{csci.GetBuilding()} {csci.GetNumber()}");
Console.WriteLine($"Your classroom is {csci}");
```

Get each attribute separately

Display all attributes at once

```
Rectangle myRect = new Rectangle(3, 6);
Console.WriteLine("My rectangle has length "
  + $"{myRect.GetLength()} and width {myRect.GetWidth()}");
Console.WriteLine($"My rectangle is a {myRect}");
myRect.MultiplyBoth(3, 5);
Console.WriteLine($"After MultiplyBoth: {myRect}");
```

"3 x 6 rectangle"

"9 x 30 rectangle"