Intro to Classes and Objects

Principles of Computer Programming I

Spring/Fall 20XX

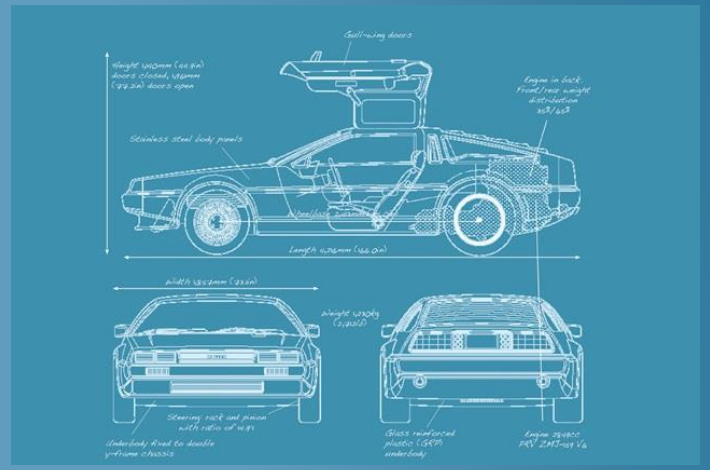
AUGUSTA UNIVERSITY

Outline

- Class and Object Basics
- Writing Our First Class
- Using a Class
- Principles of Accessors

Reminder: Classes and Objects

- Class = blueprint, template for object
  - Code that describes an object
- Object = single instance of class
  - Running code, with specific values/state
- Instantiate = create an object from a class
- Attribute = data stored in object
- Method = function that uses object's data
  - Defined in class, but executed on specific object (usually)

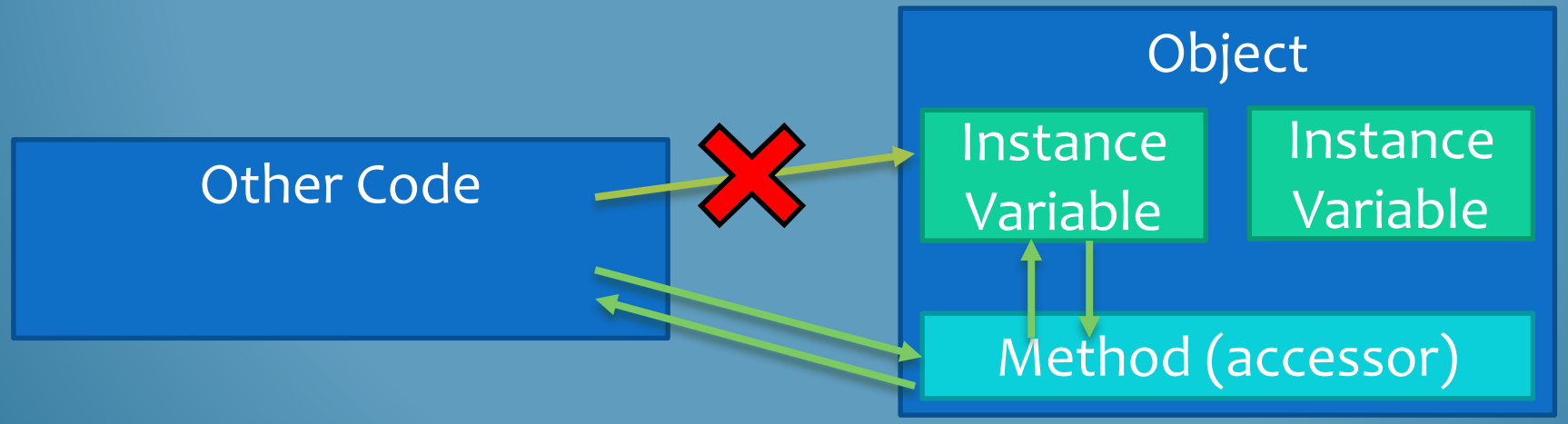


Class

Object

Encapsulation

- Attribute data is stored in **instance variables**
- Instance variables are "hidden" inside the object
- Other code cannot access instance variables directly; only the object's methods can access them
- **Accessor:** method that allows other code to access attributes

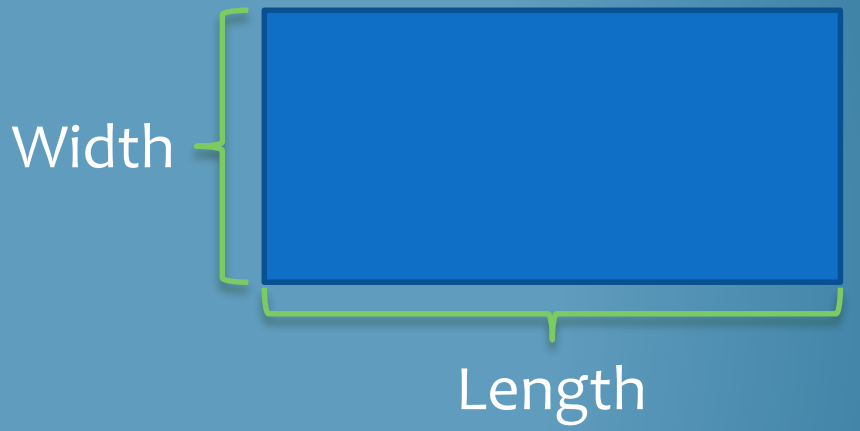


Outline

- Class and Object Basics
- **Writing Our First Class**
- Using a Class
- Principles of Accessors

The Rectangle Class

- Attributes:
  - Length
  - Width
- Methods:
  - Get and set length (accessors)
  - Get and set width (accessors)
  - Compute area



Code: Attributes and Setter

```
class Rectangle
{
  private int length;
  private int width;

  public void SetLength(int lengthParameter)
  {
    length = lengthParameter;
  }
  //continued…
```

Declare an instance variable

Access modifier: ensures that `length` cannot be accessed outside of `Rectangle`

Declare a method named `SetLength`

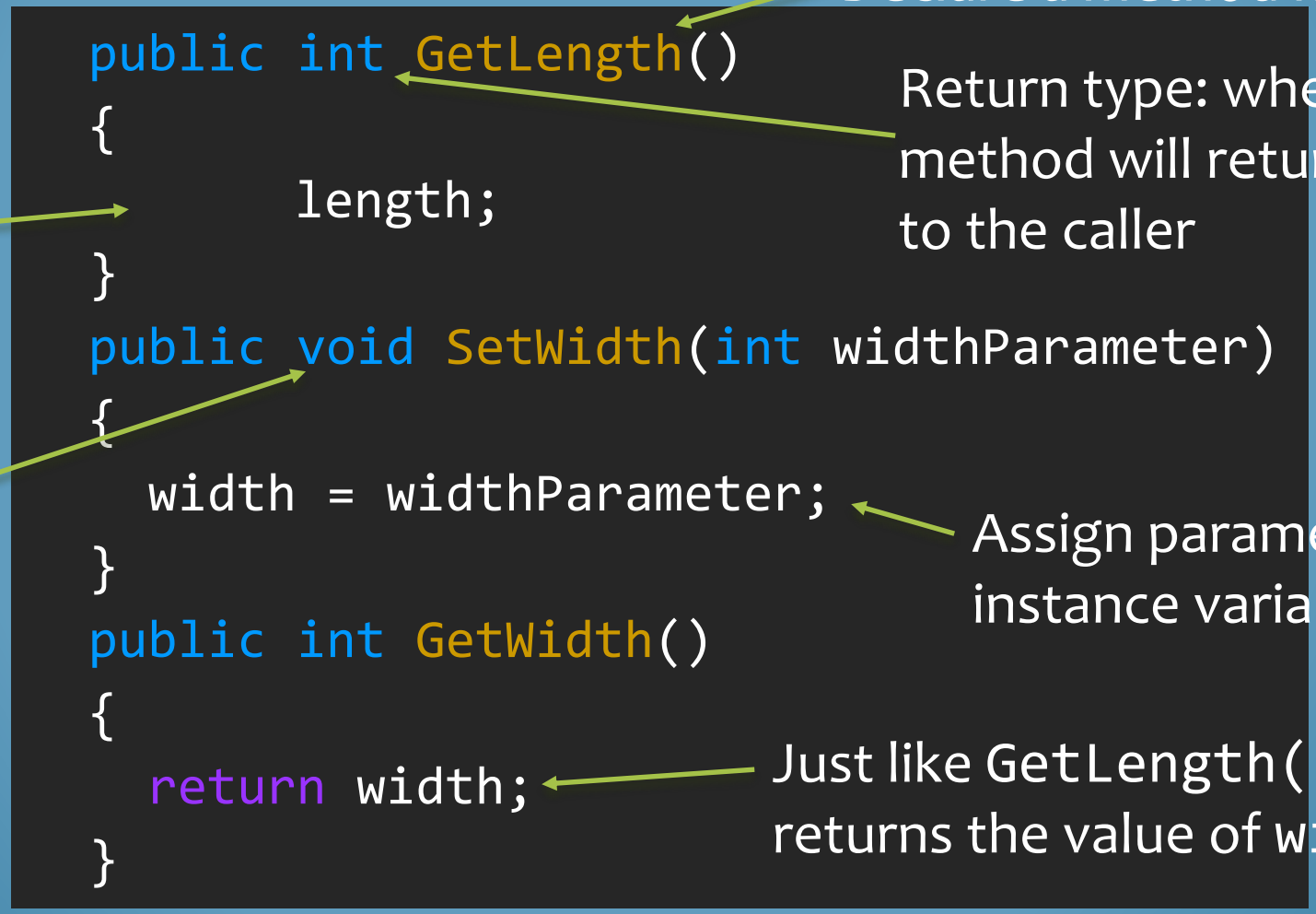
Parameter type: `int`

Access modifier: `SetLength` can be called by any code

Parameter name: `lengthParameter`

Method body: Assign parameter's value to instance variable

Code: More Accessors



Declare a method named `GetLength`

Return type: when called, this method will return an `int` value to the caller

`return` keyword: directs method to return a value

Return type `void`: This method does not return a value

Assign parameter's value to instance variable

Just like `GetLength()`, but returns the value of `width` instead

Code: ComputeArea

Note: Not an accessor. Does not get or set an attribute

Return type: `int`

```
public int ComputeArea()
{
    return length * width;
}
}
```

`int * int = int`

`return` uses result of expression as value to return

End of class declaration

Expression computed first, using current values of `length` and `width`

Outline

- Class and Object Basics
- Writing Our First Class
- **Using a Class**
- Principles of Accessors

A Program That Uses Rectangle

```
class Program
{
  static void Main(string[] args)
  {
    Rectangle myRectangle = new Rectangle();
    myRectangle.SetLength(12);
    myRectangle.SetWidth(3);
    int area = myRectangle.ComputeArea();
    Console.WriteLine("Your rectangle's length is" +
      $"{myRectangle.GetLength()}, and its width is" +
      $"{myRectangle.GetWidth()}, so its area is {area}.");
  }
}
```

Declare a variable with type Rectangle

Instantiate a Rectangle object

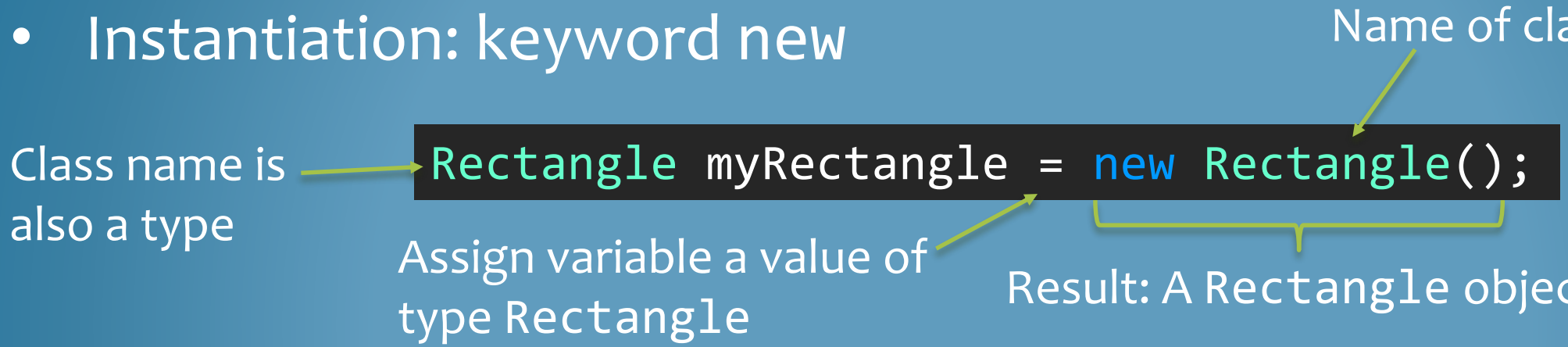
Call the SetLength method with argument 12

Call the ComputeArea method, store its return value in a variable

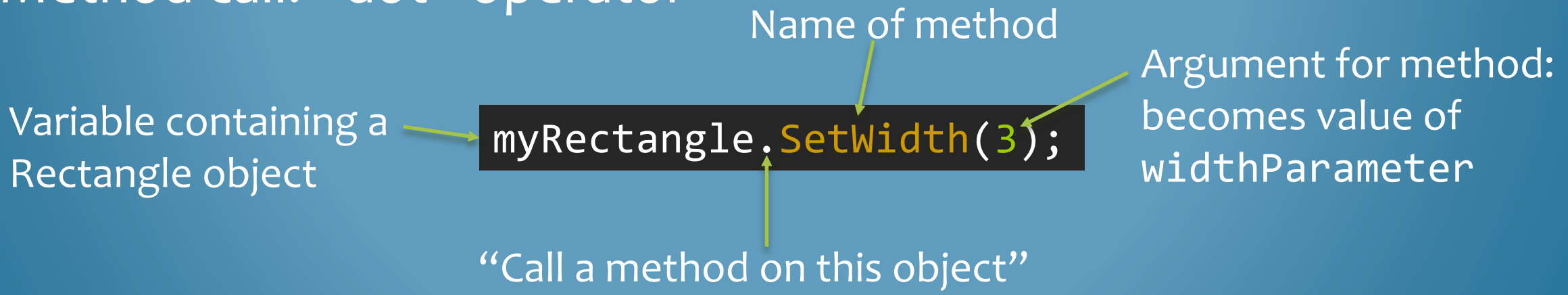
Call the GetLength and GetWidth methods, put their return values in a string

Syntax Details

- Instantiation: keyword new



- Method call: "dot" operator



Flow of Control

In Program.cs:

```
Rectangle myRectangle = new Rectangle();
myRectangle.SetLength(12);
…


int myLength = myRectangle.GetLength();
…
```

The length variable stored in myRectangle

Value: 12

In Rectangle.cs:

```
public void SetLength(int lengthParam)
{
   length = lengthParam;
}
```

```
public int GetLength()
{
   return length;
}
```

The length variable stored in myRectangle

Value: 12

Outline

- Class and Object Basics
- Writing Our First Class
- Using a Class
- **Principles of Accessors**

Accessor Methods Provide Access

- Each **attribute** of a class can have corresponding **accessor** methods
- Getter: lets other code **read** the attribute
- Setter: lets other code **write** the attribute

```
class Rectangle
{
  private int length;
  private int width;

  public void SetLength(int lengthParameter)
  {
    length = lengthParameter;
  }
  public int GetLength()
  {
    return length;
  }
```

Instance variable for length attribute

Setter for length attribute

Getter for length attribute

Structure of a Getter

Important parts of a "getter" method:

1. Return type equal to the instance variable's type
2. No parameters; takes no input
3. Body must `return` the instance variable

```
private int length;
```

```
public int GetLength()
{
  return length;
}
```

```
private int width;
```

```
public int GetWidth()
{
  return width;
}
```

Structure of a Setter

Important parts of a "setter" method:

1. Return type is `void` – no `return` statement
2. One parameter, with same type as instance variable
3. Body assigns the parameter to the instance variable

```
private int length;
```

```
public void SetLength(int lengthParameter)
{
  length = lengthParameter;
}
```

Instance variable

Parameter

Non-Accessor Methods

- Not all methods that use instance variables are accessors
- Some methods **compute** and **return** a value based on instance variables ("read-only" methods)

```
class Rectangle
{
  private int length;
  private int width;

  public int ComputeArea()
  {
    return length * width;
  }
```

Returns the result; object is unchanged

Reads current value of instance variable

Modifying (Mutating) Methods

- Methods other than setters can change instance variables
- **Mutate**, or modify, the object's state (attributes)
- Example: Increase the size of a rectangle's dimensions by 1

```
class Rectangle
{
  private int length;
  private int width;

  public void IncreaseSize()
  {
    length += 1;
    width += 1;
  }
```

Writes new values to instance variables

Compute `length + 1`, assign result to `length`