Scope, Constants, and Reference Types

Principles of Computer Programming I

Spring/Fall 20XX

AUGUSTA UNIVERSITY

Outline

- Variable Scope
- Constants
- Reference Types
  - Usage in variables
  - Usage in parameters
- Object parameters and `private`

Local vs. Instance Variables

- Instance Variables
  - Stored in object
  - Shared by all methods
  - Changes persist after method finishes executing
- Local variables
  - Visible only to one method
  - Disappear after method finishes executing

```
class Rectangle
{
  public void SwapDimensions()
  {
    int temp = length;
    length = width;
    width = temp;
  }
  public int GetLength()
  {
    return length;
  }
```

After this, `GetLength()` will return the new `length`

Cannot use `temp` here

Variable Scope

- Variables exist only in limited **time** and **space**
- Scope of a variable = region where it is accessible
- Time: *after* it is declared
- Space: within the same *code block* (defined by braces {}) where it is declared

```
class Rectangle
{
  private int length;
  private int width;
  public void SwapDimensions()
  {
    temp = 1;
    int temp;
    temp = length;
    length = width;
    width = temp;
  }
}
```

Scope = inside class Rectangle

Doesn't exist yet

Scope = inside this method

More Scope Examples

```
class Rectangle
{
  public void SwapDimensions()
  {
    int temp = length;
    length = width;
    width = temp;
  }
  public void SetWidth(int widthParam)
  {
    int temp = width;
    width = widthParam;
  }
}
```

Same name, different variables, different scopes

Parameter scope = within this method

Scope in Separate Classes

```
class Program
{
  static void Main()
  {
     int length = 5;
     Rectangle r1 = new Rectangle();
     r1.SetLength(length);
     r1.SetWidth(length * 2);
  }
}
```

Scope = inside method Main

Only Main's length is in scope here

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

Scope = inside Rectangle

Only Rectangle's length is in scope here

A Scope Pitfall

- What's the problem with this code?

```
class Rectangle
{
  private int length;
  private int width;
  public void UpdateWidth(int newWidth)
  {
    int width = 5;
    width = newWidth;
  }
}
```

A Scope Pitfall

- Two variables can have the same name if they have different scopes
- The variable with the "closer" scope *shadows* or *hides* the variable with the "farther" scope
- Probably not what you wanted

```
class Rectangle
{
  private int length;
  private int width;
  public void UpdateWidth(int newWidth)
  {
    int width = 5;
    width = newWidth;
  }
}
```

Scope = all of class Rectangle

Scope = inside this method

This means the local `width`, **not** the instance variable!

Shadowing and this

- Keyword `this` specifies the instance variable, not the local

```
class Rectangle
{
  private int length;
  private int width;
  public void UpdateWidth(int newWidth)
  {
    int width = 5;
    this.width = newWidth;
  }
}
```

Not an instance variable

Can only mean an object member

```
class Rectangle
{
  private int length;
  private int width;
  public void SetWidth(int width)
  {
    this.width = width;
  }
}
```

Instance variable

Parameter

Scope in Separate Classes

Same name, different scopes

```
class Prism
{
  private int length;
  private int width;
  private int depth;
  public void SetLength(int lengthP)
  {
    length = lengthP;
  }
  public int GetLength()
  {
    return length;
  }
}
```

Scope = inside Prism

Only Prism's length is in scope here

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

Scope = inside Rectangle

Only Rectangle's length is in scope here

Outline

- Variable Scope
- **Constants**
- Reference Types
  - Usage in variables
  - Usage in parameters
- Object parameters and `private`

Constants

- Named values that can't change – like variables, but don't vary
- Can only be built-in types (int, double, etc.) not your own classes
- Convention: Named using ALL CAPS

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

Safe because it can't change

```
static void Main(string[] args)
{
  decimal yearlyPrice = 2000.0m;
  decimal monthlyPrice = yearlyPrice
    / Calendar.MONTHS;
  Calendar myCal = new Calendar();
}
```

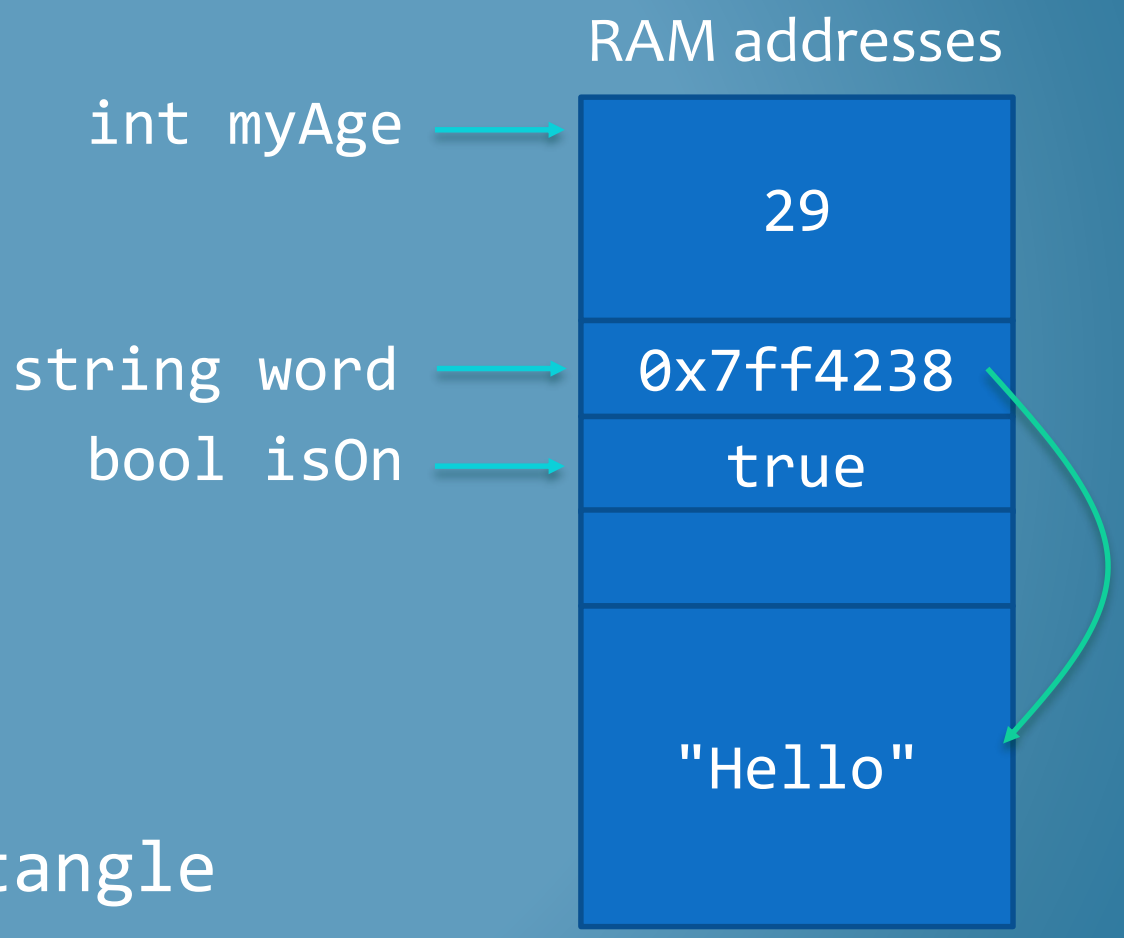
Access using class name, not object name

Outline

- Variable Scope
- Constants
- **Reference Types**
  - Usage in variables
  - Usage in parameters
- Object parameters and `private`

Recall: Value vs. Reference Types

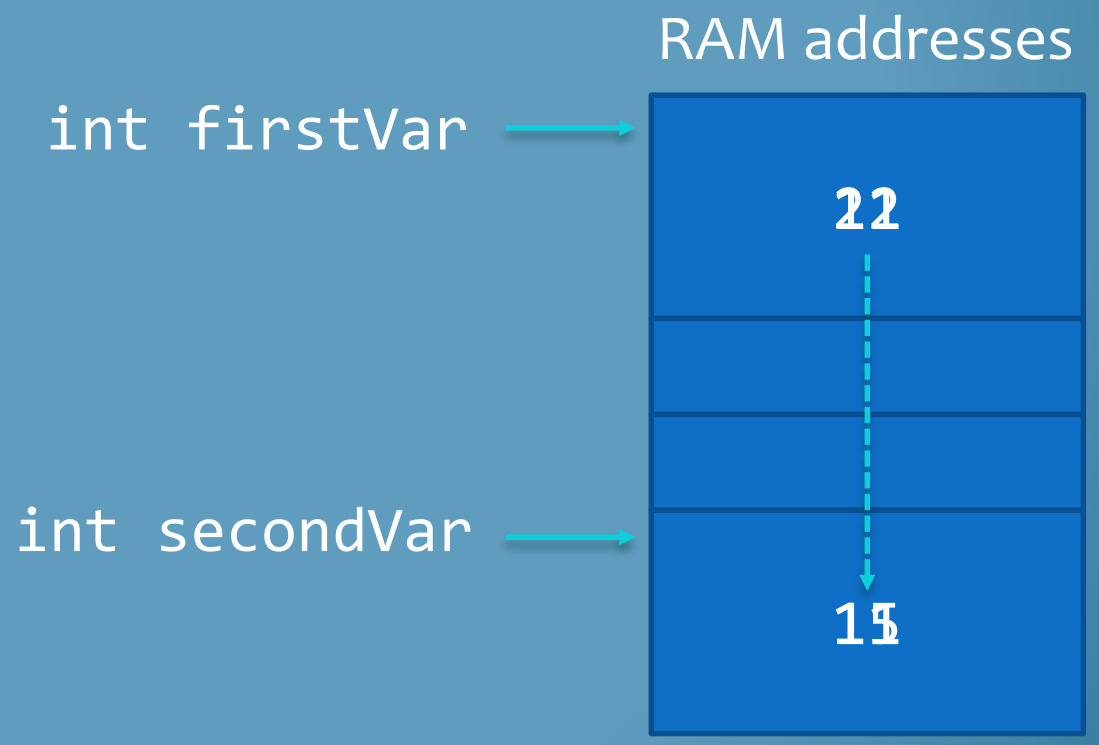
- **Value Type** variables: Memory location stores the value
  - `int`, `long`, `float`, `double`, `decimal`, `char`, `bool`
- **Reference Type** variables: Memory location stores a reference to the value
  - `string`, `object`
  - Any object you create, e.g. `Rectangle`



Assigning Value-Type Variables

- Assignment stores a copy of a variable's value
- For value types, this creates two separate variables

```
int firstVar = 11;
int secondVar = firstVar;
firstVar *= 2;
secondVar += 4;
```



Assigning Reference Variables

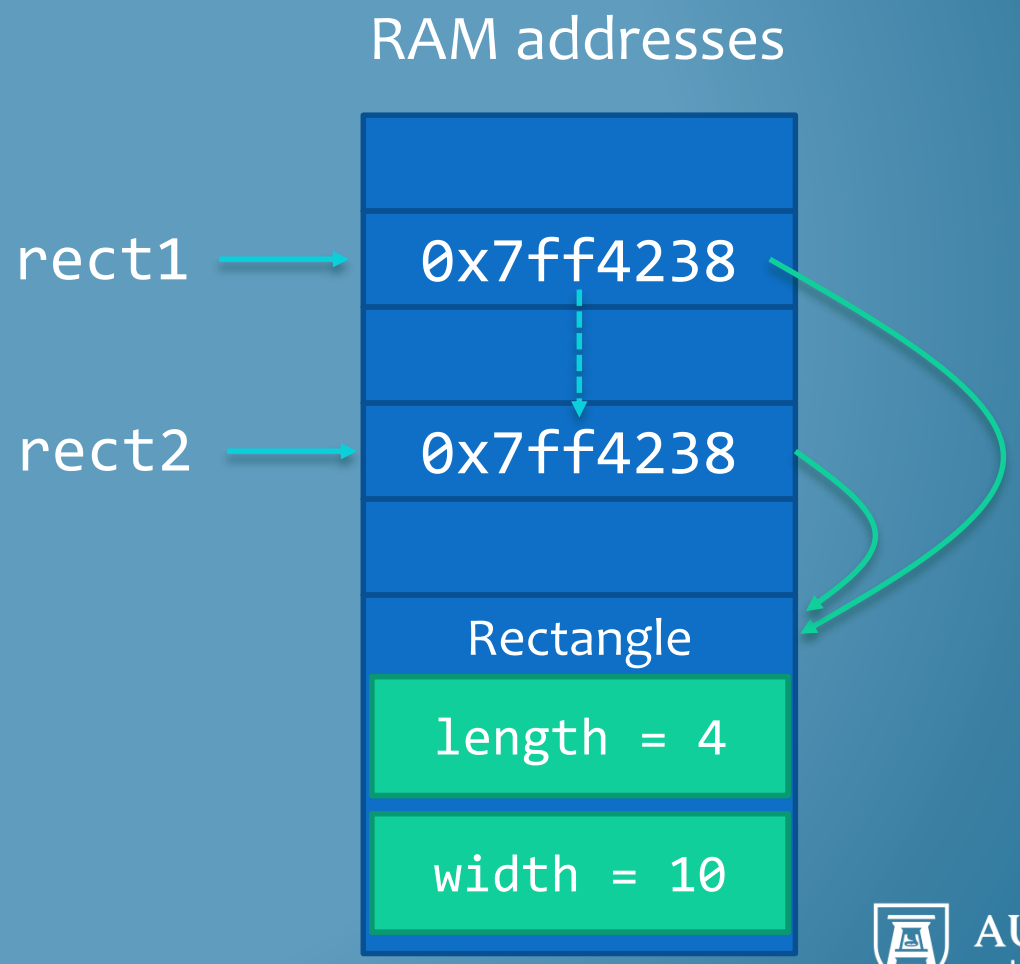
```
static void Main(string[] args)
{
  Rectangle rect1 = new Rectangle();
  rect1.SetLength(8);
  rect1.SetWidth(10);
  Rectangle rect2 = rect1;
  rect2.SetLength(4);
  Console.WriteLine($"Rectangle 1: {rect1.GetLength()} "
    + $"by {rect1.GetWidth()}");
  Console.WriteLine($"Rectangle 2: {rect2.GetLength()} "
    + $"by {rect2.GetWidth()}");
}
```

What does this print?

Assigning Reference Variables

- Assignment copies the variable (i.e. the *reference*), not the object it refers to

```
Rectangle rect1 = new Rectangle();
rect1.SetLength(8);
rect1.SetWidth(10);
Rectangle rect2 = rect1;
rect2.SetLength(4);
```



Copying an Object

- The only way to create an object is with new

```
Rectangle rect1 = new Rectangle();
rect1.SetLength(8);
rect1.SetWidth(10);
Rectangle rect2 = new Rectangle();
rect2.SetLength(rect1.GetLength());
rect2.SetWidth(rect1.GetWidth());
rect2.SetLength(4);
Console.WriteLine($"Rectangle 1: {rect1.GetLength()} "
  + $"by {rect1.GetWidth()}");
Console.WriteLine($"Rectangle 2: {rect2.GetLength()} "
  + $"by {rect2.GetWidth()}");
```

rect2 is a new, distinct Rectangle object

Copy each attribute individually

Now rect2 is a copy of rect1

Change the copy, not the original

Application to Method Parameters

- Parameters are initialized by assignment:

```
rect1.SetLength(8);
```

```
lengthP = 8;
```

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

- If parameter is a reference type (object), this will copy the reference, not the object

Objects Can Change Other Objects

In Rectangle.cs:

```
public void CopyToOther(Rectangle otherRect)
{
  otherRect.SetLength(length);
  otherRect.SetWidth(width);
}
```

Modifies the object referred to by otherRect

A reference to an object; the object exists outside the method

This object's instance variables

In Program.cs:

```
Rectangle rect1 = new Rectangle();
Rectangle rect2 = new Rectangle();
rect1.SetLength(8);
rect1.SetWidth(10);
rect1.CopyToOther(rect2);
```

rect2 starts with length 0 and width 0

Now rect2 has length 8 and width 10

CopyToOther In More Detail

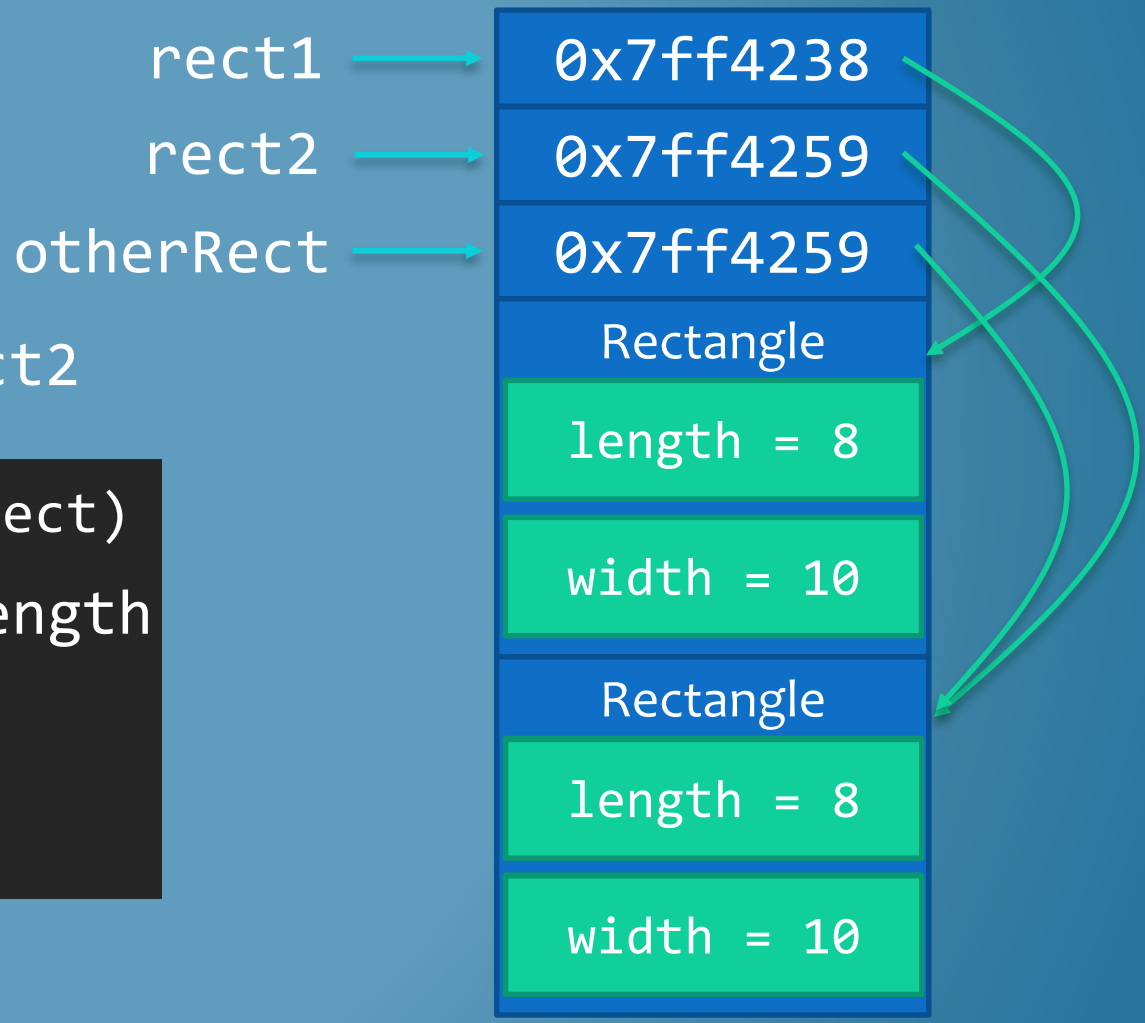
```
rect1.CopyToOther(rect2);
```

this = rect1

otherRect = rect2

```
public void CopyToOther(Rectangle otherRect)
{
  otherRect.SetLength(length);
  otherRect.SetWidth(width);
}
```

rect1.length



Outline

- Variable Scope
- Constants
- Reference Types
  - Usage in variables
  - Usage in parameters
- **Object parameters and `private`**

Encapsulation and Access Control

- `private` modifier = code outside this class cannot access
- Enforces encapsulation of instance variables

```
class Program
{
  static void Main()
  {
     Rectangle r1 = new Rectangle();
     r1.width = 22;
     r1.SetWidth(23);
  }
}
```

Error! `width` is private

```
class Rectangle
{
  private int length;
  private int width;
  public void SetWidth(int newWidth)
  {
    width = newWidth;
  }
}
```

What About Parameters?

- `CopyToOther` is inside the `Rectangle` class
- Its parameter is a `Rectangle` object

```
class Rectangle
{
  private int length;
  private int width;
  public void CopyToOther(Rectangle otherRect)
  {
    otherRect.length = length;
    otherRect.width = width;
  }
}
```

- Is this legal?
- Does it violate encapsulation?