

Scope of Objects and Variables

University of Mount Union

CSC 120

Lecture 32

What is the scope of an object?

- The term “scope” in programming languages refers to the part of the program in which an object can be accessed or referred to
 - object in this sense means a variable, a parameter, a property in a class definition, etc.
- It is determined by where the object is declared
- Objects may only be accessed in the same block in which they are declared

Types of Scope

- **Class-level scope:**
 - Object is declared in a class but OUTSIDE of any method body
 - Objects with class-level scope exist everywhere in the class (and may be accessed in any method)
- These object declarations may start with a scope keyword:
 - `private` : cannot be accessed directly by name from another class (only with getters and setters)
 - other scope keywords: `public`, `protected`

Types of Scope - 2

- **Local (or method-level) scope:**
 - Object is declared in a method of a class, but outside a block defined with { } inside the method
 - this can either be a parameter of the method or a variable/object declared in the body of the method
 - Objects with local scope only exist in the method in which they are defined
- These object declarations MAY NOT start with a scope keyword, such as `private`, `public`, etc.:
 - Format is simply: `DataType objectName;`
 - Example: `Double atomicweight, tensileStrength;`

Types of Scope - 3

- **Block-level scope:**

- Object is declared inside a block defined with { } inside a method
 - Typically the body of an if statement, an else statement, or a loop
 - Can also be in the heading of a for loop (first part, or initialization part)
- Objects with block scope only exist in the block in which they are defined
- Such object declarations MAY NOT start with a scope keyword, such as `private`, `public`, etc.

An example

- This code contains an error. Why?

```
public void someMethod( ) {  
    for ( Integer x = 1; x <= 10; x++ ) {  
        System.out.println("x = " + x);  
    }  
    System.out.println("Exited because x = " + x);  
} // end of someMethod
```

An example

- This code contains an error. Why?

```
public void someMethod( ) {  
    for ( Integer x = 1; x <= 10; x++ ) {  
        System.out.println("x = " + x);  
    }  
    System.out.println("Exited because x = " + x);  
} // end of someMethod
```

- Here, we are attempting to access a block-level variable outside the block in which it is defined (x only exists in the for loop)
- How can this error be fixed?

An example

- The solution: use local scope for x

```
public void someMethod( ) {  
    Integer x; // local scope; exists in entire method  
    for ( x = 1; x <= 10; x++ ) {  
        System.out.println("x = " + x);  
    }  
    System.out.println("Exited because x = " + x);  
} // end of someMethod
```

- Notice that first part of for loop is NOT

```
Integer x = 1;
```


Object Naming Rules

- Remember that every object must have a unique name
 - so no two objects can have the same name
 - Important: this rule only applies to objects that are IN THE SAME SCOPE

Objects in different scopes can have the same names

- for example, a local variable with the same name as a class-level property
- This fact can lead to some tricky situations....

Same name, different scope

- Consider this example.

```
public class MUPanel {  
    Double tax = 0.07;  
  
    public MUPanel() {  
        Double amtDue = calculateCost(100.00);  
        System.out.println( "You owe $"  
            + amtDue + " using a tax rate of "  
            + tax );  
    } // end of constructor  
  
    public Double calculateCost(Double beforeTaxAmt) {  
        Double tax = 0.10;  
        Double total;  
        total = beforeTaxAmt + beforeTaxAmt*tax;  
        return total;  
    } // end of calculateCost  
  
}
```

Same name, different scope

- Output produced by the previous code:

You owe \$110.00 using a tax rate of 0.07

Same name, different scope

- Output produced by the previous code:

You owe \$110.00 using a tax rate of 0.07

- The issue is two variables with different scopes that have the same name
 - A class-level property named tax with a value of 0.07
 - A local variable named tax with a value of 0.10
- Inside the calculateCost method, a reference to tax resolves to the local object, not the class-level one
- References ALWAYS refer to the object with the “smallest” or “closest” scope

Parameter Names in Constructors and Setters

- This is why we have used different names for the parameters in our Constructor and Setter methods so far this semester:

```
public class Dog {  
    String name, breed;  
    Integer age, weight;  
  
    public Dog(String n, String b, int a, int w) {  
        name = n;  
        breed = b;  
        age = a;  
        weight = w;  
    } // end of constructor  
  
    public void setWeight(int w) {  
        weight = w;  
    } // end of setWeight  
} // end of class Dog
```

Parameter Names in Constructors and Setters

- If we tried to use the same names for the properties and the parameters, it wouldn't work:

```
public class Dog {  
    String name, breed;  
    Integer age, weight;  
  
    public Dog(String name, String breed, int age, int weight) {  
        name = name;  
        breed = breed;  
        age = age;  
        weight = weight;  
    } // end of constructor  
  
    public void setWeight(int weight) {  
        weight = weight;  
    } // end of setWeight  
  
} // end of class Dog
```

Parameter Names in Constructors and Setters

- The problem with a statement such as

`breed = breed;`

is that the system has no way to know that we want the parameter to be on the r.h.s. of the assignment and the class-level property to be on the r.h.s. of the assignment

- The rule is that we use the “most local” object when resolving a name, so both sides of the assignment use the parameter, and nothing is stored in the properties of the class
- We need new notation to allow us to access the properties on the l.h.s. in this situation

Special Notation for accessing class-level objects anywhere

- Oftentimes, we will want to access a class-level property or object in a method that has a local variable with the same name
- To do that, we use the **this** keyword
 - Precede the name of the class-level object with **this**.
 - Such references always access the class-level object
 - A way to overcome the naming conflict

This is very useful in constructors and setters....

this. is the solution to this problem!

- same names for the properties and the parameters, not a problem when we use this.:

```
public class Dog {  
    String name, breed;  
    Integer age, weight;  
  
    public Dog(String name, String breed, int age, int weight) {  
        this.name = name;  
        this.breed = breed;  
        this.age = age;  
        this.weight = weight;  
    } // end of constructor  
  
    public void setWeight(int weight) {  
        this.weight = weight;  
    } // end of setWeight  
  
} // end of class Dog
```

Either style is acceptable

- Using the same names for parameters and properties (which requires the use of `this.`) was not shown earlier this semester because of complexity and possible confusion

Either style is acceptable

- Using the same names for parameters and properties (which requires the use of `this.`) was not shown earlier this semester because of complexity and confusion
- Why show it now?
 - NetBeans has a feature that we will use in Lab # 9 next time that is very convenient, but uses `this.` notation
 - So to use this feature, you need to understand this other notation
- Believe me, you'll thank me once you see it....