Instances of Objects

by Sophia

WHAT'S COVERED

In this lesson, you will learn about instantiating objects when using Java. A class defines a data type, and any object instantiated from a class has the type of that class. This allows programmers to use custom classes and the objects created from them—like another variable—to pass data from one method to another. Specifically, this lesson covers:

1. Object Scope

To illustrate the points made in this tutorial, let's use the relatively simple `PeopleCounter` class. It is defined like this, in a file named `PeopleCounter.java`:

```
public class PeopleCounter {
  private long count = 0;

  public void anotherOne() {
    count++;
    System.out.println("So far " + count);
  }
}
```

As with any variable, an object is local in scope (visibility). The **scope** of a variable or object refers to where it can be used in a program. If it is declared inside a method, the variable is in scope only in the body of that method. If it is declared at the class level (inside the class but not inside of a specific method), it is in scope anywhere in the class (in the application's driver class, such a variable needs to be declared as static, if it is to be accessed from within `main()`). Here is an example where a `PeopleCounter` object is constructed in the `main()` method:

```
class PeopleCounterProgram {

  public static void main(String[] args) {
    // Construct a PeopleCounter
    PeopleCounter pc = new PeopleCounter();
    // Count 3 people one by one
    pc.anotherOne();
    pc.anotherOne();
    pc.anotherOne();
  }
}
```

If there were a second method in the `PeopleCounterProgram`, the code in that method would not be able to access the instance named `pc` in the code above:

```
class PeopleCounterProgram {

  public static void main(String[] args) {
    // Construct a PeopleCounter
    PeopleCounter pc = new PeopleCounter();
    // Count 3 people one by one
    pc.anotherOne();
    pc.anotherOne();
    pc.anotherOne();
    anotherMethod();
  }

  public static void anotherMethod() {
    // This will not work - the instance pc is not in scope in this method
    pc.anotherOne();
  }
}
```

Trying to run this version of the program produces an error:

```
PeopleCounterProgram.java:15: error: cannot find symbol
    pc.anotherOne();
    ^
  symbol:   variable pc
  location: class PeopleCounterProgram
1 error
error: compilation failed
```

If the object needs to be used by another method, it needs to be passed as a parameter.

TERM TO KNOW

Scope
The scope of a variable or object refers to where it can be used in a program.

2. Passing Objects as Parameters

Java allows the programmer to pass objects as parameters to methods, assuming that the method being called has a signature that allows the specific type of object for a given parameter. Here is an example program that includes a method called:

```
class PeopleCounterProgram {

  public static void main(String[] args) {
    // Construct a PeopleCounter
    PeopleCounter pc = new PeopleCounter();
    // Count one
    pc.anotherOne();
    System.out.println("Calling countUsingParameter(): ");
    countUsingParameter(pc);
    System.out.println("Using original PeopleCounter pc: ");
    pc.anotherOne();
  }

  public static void countUsingParameter(PeopleCounter counter) {
    System.out.println("Now in countWithReturn()");
    counter.anotherOne();
    System.out.println("Leaving countWithReturn()");
  }
}
```

The output from this code allows you to see the flow through the code and where the PeopleCounter object is being used:

```
So far 1
Calling countUsingParameter():
Now in countWithReturn()
So far 2
Leaving countWithReturn()
```

```
Using original PeopleCounter pc:
So far 3
```

The fact that the count is in the correct sequence, even though the object is passed as a parameter, shows that the same object is being used, both in the body of the `main()` method and in the `countUsingParameter()` method.

3. Returning Objects from Methods

It is also possible to have an object returned from a method. Here is another program that uses the `PeopleCounter`. In this case, the `getPeopleCounter()` method creates a `PeopleCounter` object and returns it to the caller:

```
class PeopleCounterProgram {
  public static void main(String[] args) {
    System.out.println("Getting PeopleCounter...");
    PeopleCounter counter = getPeopleCounter();
    System.out.println("Using new PeopleCounter...");
    counter.anotherOne();
  }

  public static PeopleCounter getPeopleCounter() {
    System.out.println("Returning new PeopleCounter...");
    return new PeopleCounter();
  }
}
```

HINT

Note that the line that calls the `getPeopleCounter()` method also declares a `PeopleCounter` variable to hold the object returned by the method.

The caller is responsible for providing a variable to hold the resulting object:

EXAMPLE

```
PeopleCounter counter = getPeopleCounter();
```

The results from running this program should look like this:

```
Getting PeopleCounter...
Returning new PeopleCounter...
```

```
Using new PeopleCounter...
So far 1
```

SUMMARY

In this lesson, you learned how **objects** created from classes can be used like any other kind of variable to pass data between methods, noting that as with any variable, an object is local in **scope**. You also learned that objects that are instances of a class can be used as method parameters and return types; Java allows the programmer to **pass objects as parameters** to methods, as well as **return objects from methods**.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Scope
The scope of a variable or object refers to where it can be used in a program.