Table of Contents

Preface

This module is one in a series of modules designed to teach you about the essence of Object-Oriented Programming (OOP) using Java with particular emphasis on accessibility for blind students.

Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the images and listings while you are reading about them.

***Note regarding images:** A primary purpose of this collection is to provide material that is accessible to blind students. The term "image" will be used to describe a block of material that is different from a "listing." However, images in this collection will consist solely of text that can be read using an audible screen reader or a braille display. Images in this collection will not consist of inaccessible groups of colored pixels as is the case in many of the other collections that I have published.*

Images

- Image 1. Screen output.

Listings

- Listing 1. The class named Radio01.
- Listing 2. Constructing a Radio object.
- Listing 3. Programming the radio buttons.
- Listing 4. Pressing a button on the radio.
- Listing 5. The Radio class.
- Listing 6. An instance variable.

Preview

This module will concentrate primarily on a discussion of the Java class.

A simple Java program will be discussed to illustrate the definition and use of two different classes. Taken in combination, these two classes simulate the manufacture and the use of the car radio object discussed in an earlier module.

You will see how to write code to create a new **Radio** object by applying the **new** operator to the class named **Radio**. You will also see how to save that object's reference in a reference variable of type **Radio**.

You will see how to write code that is used to simulate the association of a radio button with a particular radio station.

You will see how to write code that is used to simulate the pressing of a radio button to play the radio station associated with that button.

You will see the definition of a class named **Radio01**. This class consists simply of the **main** method. The **main** method of a Java application is executed by the Java Virtual Machine when the application is run. Thus, it is the driver for the entire application.

You will see the definition of a class named **Radio**. This class includes one instance variable and two instance methods.

(The instance variable is a reference variable that refers to a special kind of object that I refer to as an array object. I will provide a very brief discussion on array objects in this module. I will have more to say about array objects in a subsequent module.)

I will provide a short discussion of class variables, which are not used in this program. I will explain that the use of class variables can often lead to undesirable side effects.

Finally, I will provide a very brief discussion of the syntax of a simple class definition in Java.

Discussion and sample code

What is a class?

I explained in an earlier module that a class is a plan from which many objects can be created. I likened the class definition to the plans from which millions of nearly identical car radios can be produced.

A simple Java program

In order to help you to get started on the right foot, and in support of future discussions, it will be advantageous to provide and discuss a simple Java program in this module.

The car radio example

Listing 9, near the end of this module, shows the code for a simple Java application that simulates the manufacture and the use of a car radio.

Explain in fragments

In order to help you to focus specifically on important sections of code, I will explain the behavior of this program in fragments.

Top-level classes

This program contains two top-level class definitions. *(Java also supports inner classes in addition to top-level classes. Inner classes will be explained in detail in subsequent modules in this series.)*

The class named Radio01

The definition of a class named **Radio01**, is shown in its entirety in Listing 1. The other class named **Radio** will be discussed later.

Listing 1. The class named Radio01.

```
public class Radio01{
  public static void main(
                        String[] args){
    Radio myObjRef = new Radio();
    myObjRef.setStationNumber(3,93.5);
    myObjRef.playStation(3);
  }//end main
}//end class Radio01
```

The class named **Radio01** consists simply of the **main** method. The **main** method of a Java application is executed by the Java Virtual Machine when the application is run. Thus, it is the driver for the entire application.

The driver class

The code in Listing 1 simulates the manufacturer of the radio and the use of the radio by the end user. Without getting into a lot of detail regarding Java syntax, I will further subdivide and discuss this code in the following listings.

Constructing a Radio object

As discussed in an earlier module, the code in Listing 2 applies the **new** operator to the constructor for the **Radio** class, causing a new object to be created according to the plans specified in the class named **Radio**.

Listing 2. Constructing a Radio object.

```
    Radio myObjRef = new Radio();
```

Saving a reference to the Radio object

Also as discussed in an earlier module, the code in Listing 2 declares a reference variable of type **Radio** and stores the new object's reference in that variable.

Programming the radio buttons

The code in Listing 3 is new to this discussion. This statement simulates the process of associating a particular radio station with a particular button - programming a button on the radio.

As I explained in an earlier module, this is accomplished for my car radio by manually tuning the radio to a desired station and then holding the radio button down until it beeps. You have probably done something similar to this to the radio in your family's car.

Listing 3. Programming the radio buttons.

```
    myObjRef.setStationNumber(3, 93.5);
```

The statement in Listing 3 accomplishes the association of a simulated button to a simulated radio station by calling the method named **setStationNumber** on the reference to the **Radio** object. *(Recall that this sends a message to the object asking it to change its state.)*

The parameters passed to the method cause radio button number 3 to be associated with the frequency 93.5 MHz. *(The value 93.5 is stored in the variable that represents button number 3.)*

Sending a message to the object

Using typical OOP jargon, the statement in Listing 3 sends a message to the **Radio** object, asking it to change its state according to the values passed as parameters.

Pressing a button on the radio

Finally, the code in Listing 4 calls the method named **playStation** on the **Radio** object, passing the integer value 3 *(the button number)* as a parameter.

Listing 4. Pressing a button on the radio.

```
    myObjRef.playStation(3);
```

Another message

This code sends a message to the object asking it to perform an action. In this case, the action requested by the message is:

- Tune yourself to the frequency previously associated with button number 3
- Play the radio station that you find at that frequency through the speakers

How does this simulated radio play?

This simple program doesn't actually play music. As you will see later, this causes the message shown in Image 1 to appear on the computer screen, simulating the selection and playing of a specific radio station.

Image 1. Screen output.

```
Playing the station at 93.5 Mhz
```

The Radio class

Listing 5 shows the class definition for the **Radio** class in its entirety.

Listing 5. The Radio class .

```
class Radio{
  //This class simulates the plans from
  // which the radio object is created.
  protected double[] stationNumber =
                       new double[5];

  public void setStationNumber(
                int index,double freq){
    stationNumber[index] = freq;
  }//end method setStationNumber

  public void playStation(int index){
    System.out.println(
            "Playing the station at "
               + stationNumber[index]
               + " Mhz");
  }//end method playStation
```

Listing 5. The Radio class .

```
}//end class Radio
```

Note that the code in Listing 5 does not contain an explicit constructor. If you don't define a constructor when you define a new class, a default version of the constructor is provided on your behalf. That is the case for this simple program.

(Constructors will be explained in detail in subsequent modules.)

The plans for an object

The code in Listing 5 provides the plans from which one or more objects that simulate physical radios can be constructed.

An object instantiated *(an object is an instance of a class)* from the code in Listing 5 simulates a physical radio. I will subdivide this code into fragments and discuss it in the following listings.

An instance variable

In an earlier module, I explained that we often say that an object is an instance of a class. *(A physical radio is one instance of the plans used to produce it.)* The code in Listing 6 shows the declaration and initialization of what is commonly referred to as an instance variable.

Listing 6. An instance variable.

```
  protected double[] stationNumber =
                            new double[5];
```

Why call it an instance variable?

The name *instance variable* comes from the fact that every instance of the class *(object)* has one. *(Every radio produced from the same set of plans has the ability to associate a frequency with each selector button on the front of the radio.)*

Class variables - an aside

Note that Java also supports something called a *class variable*, which is different from an *instance variable*.

Class variables are shared among all of the objects created from a given class. Stated differently, no matter how many objects are instantiated from a class definition, they all share a single copy of each class variable.

There is no analogy to a class variable in a physical radio object. Radios are installed in different cars separated from each other by thousands of miles. Therefore, there can be no sharing of anything among different physical radio objects.

(Well, that may not be entirely true. In today's technology, different radio objects could potentially share something at a common location via satellite communications, but my car radio doesn't do anything like that.)

Class variables can cause undesirable side effects

While class variables are relatively easy to use in Java, they are difficult to explain from an OOP viewpoint. Also, it is my opinion that from a good overall design viewpoint, class variables should be used very sparingly, if at all.

Therefore, for the first several modules, I will exclude the possibility of class variables in this series of modules. *(I will explain the use of class variables in Java in a subsequent module.)*

Reference to an array object

Now, let's get back to the instance variable named **stationNumber** shown in Listing 6. Without getting into a lot of detail, this variable is also a reference variable, referring to an array object.

The array object encapsulates a simple one-dimensional array with five elements of type **double**. *(Java array indices begin with zero, so the index values for this array extend from 0 to 4 inclusive. I will also discuss array objects in more detail in a subsequent module.)*

Persistence

The array object is where the data is stored that associates the frequency of a radio station with the simulated physical button on the front of the radio.

Each element in the array corresponds to one frequency-selector button on the front of the radio. Hence, the radio simulated by an object of the **Radio** class has five simulated frequency-selector buttons.

The array object exists when the code in Listing 6 has finished executing. Each element in the array has been initialized to a default value of 0.0 *(double-precision float value of zero)*.

The setStationNumber method

Listing 7 shows the **setStationNumber** method in its entirety

Listing 7. The setStationNumber method.

```
  public void setStationNumber(
                int index,double freq){
    stationNumber[index] = freq;
  }//end method setStationNumber
```

Associates radio station with button

This is the method that is used to simulate the behavior of having the user associate a particular button with a particular radio station. *(Recall that this is accomplished on my car radio by manually tuning the radio to a specific station and then holding the button down until it beeps. Your family's car radio probably operates in some similar way.)*

This method receives two incoming parameters:

- An integer that corresponds to a button number *(button numbers are assumed to begin with 0 and extend through 4 in order to match array indexes)*
- A frequency value to be associated with the indicated button.

Save the frequency value

The code in the method stores the frequency value in an element of the array object discussed earlier.

The element number is specified by the value of index shown in square brackets in the assignment expression. *(This syntax is similar to storing a value in an array element in most programming languages that I am familiar with.)*

Pressing a radio button to select a station

Listing 8 shows the **playStation** method. This is the method that simulates the result of having the user press a button on the front of the radio to select a particular radio station for play.

Listing 8. The playStation method.

```
  public void playStation(int index){
    System.out.println(
            "Playing the station at "
               + stationNumber[index]
               + " Mhz");
  }//end method playStation
```

Selecting and playing a radio station

The method receives an integer index value as an incoming parameter. This index corresponds to the number of the button pressed by the user. This method simulates the playing of the radio station by

- Extracting the appropriate frequency value from the array object, and
- Displaying that value on the computer screen along with some surrounding text.

When called by code in the **main** method of this program, this method produces the message shown in Image 1 on the computer screen

That summarizes the behavior of this simple program.

Class definition syntax

There are a number of items that can appear in a class definition, including the following:

- Instance variables
- Class variables
- Instance methods
- Class methods
- Constructors
- Static initializer blocks
- Inner classes

Let's keep it simple

In order to make these modules as easy to understand as possible, the first several modules will ignore the possibility of class variables, class methods, static initializer blocks, and inner classes.

As mentioned in the earlier discussion of class variables, these elements aren't particularly difficult to use, but they create a lot of complications when attempting to explain OOP from the viewpoint of Java programming.

Therefore, the first several modules in the series will assume that class definitions are limited to the following elements:

- Instance variables
- Instance methods
- Constructors

A constructor

A constructor is used only once in the lifetime of an object. It participates in the task of creating *(instantiating)* and initializing the object. Following instantiation, the state and behavior of an object depends entirely on instance variables, class variables, instance methods, and class methods.

Instance variables and methods

The class named **Radio** discussed earlier contains

- One instance variable named **stationNumber**, and
- Two instance methods named **setStationNumber** and **playStation**.

Summary

This module has concentrated primarily on a discussion of the Java class.

A simple Java program was discussed to illustrate the definition and use of two different classes. Taken in combination, these two classes simulate the manufacture and use of the car radio object introduced in an earlier module.

You saw how to write code to create a new **Radio** object by applying the **new** operator to the class named **Radio**.

You also saw how to save that object's reference in a reference variable of type **Radio**.

You saw how to write code *(in an instance method named **setStationNumber**)* used to simulate the association of a radio button with a particular radio station.

You saw how to write code *(in an instance method named **playStation**)* to simulate the pressing of a radio button to play the radio station associated with that button.

You saw the definition of the class named **Radio01**, which consists simply of the **main** method. The **main** method of a Java application is executed by the Java Virtual Machine when the application is run.

You saw the definition of the class named **Radio**. This class includes one instance variable and two instance methods. *(The instance variable is a reference variable that refers to a special kind of object that I refer to as an array object. I provided a very brief discussion on array objects. I will have more to say on this topic in a subsequent module.)*

I provided a short discussion of class variables, which are not used in this program. I explained that the use of class variables can often lead to undesirable side effects.

Finally, I provided a very brief discussion of the syntax of a simple class definition in Java.

What's next?

Recall that in order to understand OOP, you must understand the following three concepts:

- Encapsulation
- Inheritance
- Polymorphism

The next module will begin a discussion of inheritance. Overall, the discussion of inheritance will require more than one module. In the next module, I will discuss how the definition of a class defines a new data type. I will show you how to extend an existing class. I will explain what is inherited through inheritance. I will discuss code reuse and explicit constructors.

Finally, I will illustrate all of the above in a program that produces an audio output.

Miscellaneous

This section contains a variety of miscellaneous information.

Housekeeping material

- Module name: Jbs1020-Classes
- File: Jbs1020.htm
- Published: 08/12/14

Affiliation: Prof. Baldwin is a professor of Computer Information Technology at Austin Community College in Austin, TX.

Complete program listing

Listing 9 provides a complete listing of the program named **Radio01**.

Listing 9. The program named Radio01.

```
/*File Radio01.java
Copyright 2001, R.G.Baldwin
Simulates manufacture and use of a
car radio.

This program produces the following
output on the computer screen:

Playing the station at 93.5 Mhz
**************************************/

public class Radio01{
  //This class simulates the
  // manufacturer and the human user
  public static void main(
                        String[] args){
    Radio myObjRef = new Radio();
    myObjRef.setStationNumber(3,93.5);
    myObjRef.playStation(3);
  }//end main
}//end class Radio01
  //-----------------------------------//

class Radio{
  //This class simulates the plans from
  // which the radio object is created.
  protected double[] stationNumber =
                         new double[5];

  public void setStationNumber(
                int index,double freq){
```

Listing 9. The program named Radio01.

```
    stationNumber[index] = freq;
  }//end method setStationNumber

  public void playStation(int index){
    System.out.println(
            "Playing the station at "
               + stationNumber[index]
               + " Mhz");
  }//end method playStation

}//end class Radio
```

-end-