Java3020: Interfaces, Object Arrays, etc.[*]

R.G. (Dick) Baldwin

Abstract

Learn about interfaces, arrays of type Object, etc.

1 Table of Contents

- Preface (p. 1)
  - Viewing tip (p. 1)
    - Figures (p. 2)
    - Listings (p. 2)
- Preview (p. 2)
- General background information (p. 3)
- Discussion and sample code (p. 4)
- Run the program (p. 11)
- Summary (p. 11)
- What's next? (p. 11)
- Online video links (p. 11)
- Miscellaneous (p. 12)
- Complete program listing (p. 12)

2 Preface

This module is one of a series of modules designed to teach you about Object-Oriented Programming (OOP) using Java.

2.1 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 figures and listings while you are reading about them.

[*]Version 1.4: Dec 12, 2012 6:28 am -0600
[†]http://creativecommons.org/licenses/by/3.0/

2.1.1 Figures

- Figure 1 (p. 3) . Command line output for Prob05.

2.1.2 Listings

- Listing 1 (p. 4) . Beginning of driver class for Prob05.
- Listing 2 (p. 4) . The interface named Prob05X.
- Listing 3 (p. 5) . Beginning of the class named Prob05MyClassA.
- Listing 4 (p. 6) . The method named getModifiedData.
- Listing 5 (p. 6) . The method named getData.
- Listing 6 (p. 6) . Overridden toString method.
- Listing 7 (p. 7) . Beginning of the class named Prob05MyClassB.
- Listing 8 (p. 7) . The method named getModifiedData.
- Listing 9 (p. 8) . The getData and toString methods.
- Listing 10 (p. 8) . Print three items of information.
- Listing 11 (p. 9) . Three more print statements.
- Listing 12 (p. 10) . Print the references to the two objects.
- Listing 13 (p. 12) . Complete program listing

3 Preview

In this module, you will learn about :

- Interface definitions
- Implementing an interface in a class definition
- Defining interface methods in a class definition
- Storing references to new objects in elements of an array of type **Object**
- Casting elements to an interface type in order to call interface methods
- Parameterized constructors
- Overridden **toString** method

Program specifications

Write a program named **Prob05** that uses the class definition shown in Listing 1 (p. 4) to produce the output shown in Figure 1 (p. 3) on the command line screen.

Command line output for Prob05.

```
    Prob05
Put your first name here
Put your last name here
-18 -17 -16
-17 -17 -17
-12 -12 -12
```

Figure 1: Command line output for Prob05.

No graphic output images required

There are no graphic output images required by this program. Therefore, it can be compiled and executed without a requirement to have Ericson's media library on the classpath.

Required text output

The output, which appears on the command line screen, consists of the six lines of text shown in Figure 1 (p. 3) .

Because the program generates random data for testing, the actual values will differ from one run to the next. However, in all cases:

- The values in the first row of numbers will be a sequence of consecutive integers in increasing algebraic order from left to right.
- All three values in the second row of numbers will match the value of the center number in the first row of numbers.
- All three values in the third row of numbers will be algebraically five greater than the values in the second row of numbers.

New classes

You may define new classes as necessary to cause your program to behave as required, but you may not modify the class definition for the class named **Prob05** shown in Listing 1 (p. 4) .

4 General background information

Among other things, this program illustrates:

- Interface definitions
- Implementing an interface in a class definition
- Defining interface methods in a class definition
- Storing references to new objects in elements of an array of type **Object**
- Casting elements to an interface type in order to call interface methods
- Parameterized constructors
- Overridden **toString** method

5 Discussion and sample code

Will explain in fragments

I will explain this program in fragments. A complete listing of the program is provided in Listing 13 (p. 12) near the end of the module.

Beginning of driver class for Prob05

The driver class for **Prob05** begins in Listing 1 (p. 4) .

Listing 1: Beginning of driver class for Prob05.

```
    import java.util.*;

class Prob05{
  public static void main(String[] args){

    Random generator = new Random(new Date().getTime());
    int randomData = (byte)generator.nextInt();

    Object[] var1 = new Object[2];

    var1[0] = new Prob05MyClassA(randomData);
    var1[1] = new Prob05MyClassB(randomData);
```

Behavior of the code in Listing 1

Listing 1 (p. 4) does the following:

- Gets and saves a random value of type **int** .
- Instantiates a new two-element array object of type **Object** . *(A reference to any object of any class or interface type can be stored in an array element of type **Object** .)*
- Populates the array object with references to objects of the classes:
  - Prob05MyClassA
  - Prob05MyClassB

The same random value is passed to the constructor for both objects when they are instantiated.

Put the driver class on temporary hold

At this point, I am going to put the driver class named **Prob05** on temporary hold and explain the class named **Prob05MyClassA** .

The interface named Prob05X

Having glanced ahead, I know that the class named **Prob05MyClassA** implements the interface named **Prob05X** so I will explain that interface first.

The interface named **Prob05X** is shown in its entirety in Listing 2 (p. 4) .

Listing 2: The interface named Prob05X.

```
    interface Prob05X{
  public int getModifiedData();
  public int getData();
}//end interface
```

An interface definition

An interface definition can contain only two kinds of members:

- Constants
- Method declarations

By now, you should have studied interfaces in my online tutorials. Therefore, this explanation will be very brief.

Method declarations

Listing 2 (p. 4) contains two method declarations.

A method declaration does not have a body. Its purpose is to establish the programming interface for that method in any class that implements the interface *(return type, name, arguments, etc.)* .

A method declaration provides no information about the behavior of the method.

A method declaration in an interface is implicitly abstract.

A concrete definition is required

Any class that implements an interface:

- Must provide a **concrete** version of every method that is declared in the interface, or
- The class must be declared **abstract** . *(In this case, abstract essentially means incomplete.)*

The class named Prob05MyClassA

The class named **Prob05MyClassA** , which implements the interface named **Prob05X** , must provide concrete versions of the methods named:

- public int getModifiedData()
- public int getData()

Beginning of the class named Prob05MyClassA

The class named **Prob05MyClassA** begins in Listing 3 (p. 5) .

Listing 3: Beginning of the class named Prob05MyClassA.

```
  class Prob05MyClassA implements Prob05X{
private int data;//instance variable

Prob05MyClassA(int inData){//constructor
  System.out.println("Prob05");
  System.out.println("Put your first name here");
  data = inData;
}//end constructor
```

This class implements the interface named **Prob05X** .

A private instance variable

Listing 3 (p. 5) begins by declaring a private instance variable of type **int** named **data** . As a private instance variable, it is accessible by any method or constructor defined within the class but is not accessible to methods from outside the class.

The constructor

The constructor for the class is shown in its entirety in Listing 3 (p. 5) .

The constructor begins by displaying the problem number and the student's first name on the command line screen.

Then it assigns the value of the incoming parameter named **inData** to the variable named **data** . This makes that value available to the methods that are defined within the class.

The method named getModifiedData

We learned earlier [1] that the class named **Prob05MyClassA**

[1] http://cnx.org/content/m44214/1.4/Java3020old.htm#concrete

- must provide a concrete definition of the method named **getModifiedData** ,
- because that method is declared in the interface named **Prob05X** ,
- which is implemented by the class.

With the exception of some very subtle differences *(that are beyond the scope of this course)* , that concrete definition must match the signature of the declared method.

Code for the method named getModifiedData

The method named **getModifiedData** is shown in its entirety in Listing 4 (p. 6) .

When this method is called, it

- subtracts a value of 1 from the value stored in the instance variable named **data** , and
- returns that modified value.

Listing 4: The method named getModifiedData.

```
    public int getModifiedData(){
  return data - 1;
}//end getModifiedData()
```

The method named getData

We also learned earlier that the class named **Prob05MyClassA**

- must provide a concrete definition of the method named **getData,**
- which is also declared in the interface named **Prob05X** .

Code for the method named getData

The method named **getData** is shown in its entirety in Listing 5 (p. 6) .

This method returns a copy of the value stored in the variable named **data** .

Listing 5: The method named getData.

```
    public int getData(){
  return data;
}//end getData()
```

A round trip

When the code in Listing 1 (p. 4) instantiates an object of the **Prob05MyClassA** class, it passes a random value as a parameter to the constructor.

The constructor shown in Listing 3 (p. 5) stores that random value in the instance variable named **data** .

When the method named **getModifiedData** is called, it returns a value that is the original random value less 1.

When the method named **getData** is called, it returns a copy of the original random value.

The toString method

The class named **Prob05MyClassA** extends the class named **Object** by default. It inherits a method named **toString** from the class named **Object** . The inherited method has very specific behavior.

Overridden toString method

The code in Listing 6 (p. 6) overrides the inherited method to provide a different behavior when the method is executed in conjunction with an object of the **Prob05MyClassA** class.

The new behavior is to construct and return a string version of the value obtained by adding 5 to the value stored in **data** , which is the original random value.

Listing 6: Overridden toString method.

```
      public String toString(){
    return "" + (data + 5);
  }//end toString()

}//end class Prob05MyClassA
```

The end of the class named Prob05MyClassA

Listing 6 (p. 6) also signals the end of the class definition for the class named **Prob05MyClassA** .

The class named Prob05MyClassB

Referring back to the code in the driver class in Listing 1 (p. 4) , we see that the driver also instantiates an object of the class named **Prob05MyClassB** , passing the same random value to the constructor for the class.

The reference to the object is stored in the second element of the array object of type **Object** referred to by the reference variable named **var1** .

Beginning of the class named Prob05MyClassB

The beginning of the class named **Prob05MyClassB** is shown in Listing 7 (p. 7) .

Listing 7: Beginning of the class named Prob05MyClassB.

```
    class Prob05MyClassB implements Prob05X{
  private int data;

  Prob05MyClassB(int inData){
    System.out.println("Put your last name here");
    data = inData;
  }//end constructor
```

Implements Prob05X

The first thing we notice is that this class also implements the interface named **Prob05X** . This requires that the class provide concrete definitions of the two methods declared in that interface.

Save the incoming parameter value

The constructor for the **Prob05MyClassB** class, which is shown in Listing 7 (p. 7) , saves the incoming parameter value in a private instance variable named **data** .

Unrelated to the variable named data from before

It is important to note that this variable named **data** is completely unrelated to the private instance variable named **data** that is declared in Listing 3 (p. 5) , even though they are the same type and they have the same name.

They belong to two different objects. Objects do not share instance variables.

The two objects are related

However, even though the two objects instantiated in Listing 1 (p. 4) are instantiated from different classes, they are related in the sense that they have two ancestors in common. They both extend the class named **Object** by default and they both explicitly implement the interface named **Prob05X** . That means that they can both be treated as either type **Object** or type **Prob05X** .

Related through the interface by design

Because all classes are direct or indirect subclasses of the class named **Object** , all objects instantiated for any class are related at the **Object** level. However, the objects in this program are related through the **Prob05X** interface only because I designed the program that way.

The method named getModifiedData

The method named **getModifiedData** is shown in Listing 8 (p. 7) .

Listing 8: The method named getModifiedData.

```
      public int getModifiedData(){
    return data + 1;
  }//end getModifiedData()
```

Same behavior is not required

A comparison of Listing 8 (p. 7) with Listing 4 (p. 6) exposes a very important aspect of interface implementation.

If two different classes implement the same interface, they each must provide concrete definitions of all the method declared in the interface. When providing such concrete definitions, both classes must match the method signatures of the declared methods.

However, the behavior of a method as defined in one class is not required to be the same as the behavior of the method having the same signature in the other class.

The behavior is different

For example, the code in Listing 4 (p. 6) *subtracts 1* from the value of **data** and returns that modified value.

The code in Listing 8 (p. 7) *adds 1* to the value of **data** and returns that modified value.

Therefore, the behavior of the method named **getModifiedData** in an object instantiated from the class named **Prob05MyClassB** is completely different from the behavior of the method having the same signature in an object of the class named **Prob05MyClassA** .

The getData and toString methods

Listing 9 (p. 8) shows the **getData** and **toString** methods as defined in the class named **Prob05MyClassB** .

Listing 9: The getData and toString methods.

```
      public int getData(){
    return data;
  }//end getData()

  public String toString(){
    return "" + (data + 5);
  }//end toString()

}//end class Prob05MyClassB
```

The behavior is the same

If you compare Listing 9 (p. 8) with Listing 5 (p. 6) and Listing 6 (p. 6) , you will see that these two methods are defined the same in both classes. Therefore, these two methods have the same behavior regardless of which of the two objects instantiated in Listing 1 (p. 4) they are called on.

Back to the driver class named Prob05

Returning now to the driver class named **Prob05** where we left off in Listing 1 (p. 4) , Listing 10 (p. 8) contains three statements that print information on the command line screen.

Listing 10: Print three items of information.

```
        System.out.print(
              ((Prob05X)var1[0]).getModifiedData() + " ");

    System.out.print(randomData + " ");

    System.out.println(
                    ((Prob05X)var1[1]).getModifiedData());
```

Three print statements

The first two statements in Listing 10 (p. 8) call the **print** method and the last statement calls the **println** method.

When the **println** method is called, the onscreen cursor advances to the left side of the next line after the material has been printed.

However, when the **print** method is called, the cursor remains at the right end of the printed material.

Therefore, calling **print print println** in succession will cause three items of information to be printed on the same line.

A cast is required

Recall that the reference to each object instantiated in Listing 1 (p. 4) is stored in an array element as type **Object** .

A reference to any object can be stored in a reference of type **Object** because the **Object** class is the superclass of all classes. *(References to array objects can also be stored as type* ***Object*** *but that fact is not germane to this program.)*

Only eleven methods can be called on type Object

However, once an object's reference is stored as type **Object** , the only methods that can be called on that object *(without casting)* are the eleven methods that are defined in the **Object** class. That group of eleven methods includes the method named **toString** but it does not include the methods named **getData** and **getModifiedData** .

Must change the type of the reference

Therefore, the first statement in Listing 10 (p. 8) requires that a **cast** to be used to change the type of the reference back to a type on which the method can be called. There are a couple of choices in this regard.

Could cast to the class type

First, it is always possible to cast the reference back to the class from which the object was instantiated. Therefore, it would work to cast the reference from array element 0 in Listing 10 (p. 8) to type **Prob05MyClassA** and to cast the reference from array element 1 to type **Prob05MyClassB** .

Cast to the interface type

In this program, there is another choice. Because both classes implement the interface named **Prob05X** , and the method named **getModifiedData** is declared in that interface, it also works to cast both references to the common interface type **Prob05X** .

That is what was done in Listing 10 (p. 8) . Both references were cast to the interface type **Prob05X** .

The printed values

The first statement in Listing 10 (p. 8) calls the method named **getModifiedData** as defined in Listing 4 (p. 6) . This causes the original random value *less 1* to be printed.

The second statement in Listing 10 (p. 8) simply prints the original random value that was saved in the variable named **randomData** in Listing 1 (p. 4) .

The third statement in Listing 10 (p. 8) calls the method named **getModifiedData** as defined in Listing 8 (p. 7) . This causes the original random value *plus 1* to be printed.

Because this is a call to the **println** method, the onscreen cursor advances to the left side of the next line after the value is printed.

The three statements in Listing 10 (p. 8) cause the first three values shown in Figure 1 (p. 3) to be printed on the command line screen.

Three more print statements

Continuing with the driver class named **Prob05** , Listing 11 (p. 9) shows three more print statements.

Listing 11: Three more print statements.

```
    System.out.print(((Prob05X)var1[0]).getData() + " ");
System.out.print(randomData + " ");
```

```
    System.out.println(((Prob05X)var1[1]).getData());
```

A cast is required

In this case, the **getData** method belonging to each of the objects is called in the first and third statements. *(Once again a cast is required.)*

Behavior of the getData methods is the same

Recall that the behavior of the **getData** method is the same in both objects. It simply returns a copy of the original random value that was passed to the constructor when each of the objects was instantiated.

The three statements in Listing 11 (p. 9) produce the second set of three matching values shown in Figure 1 (p. 3) .

These three values match because all three print statements are printing essentially the same value. The original random value is printed in the middle statement in Listing 11 (p. 9) . A copy of the original random value is printed in the first and third statements.

Print the references to the two objects

Things get a little bit more complicated in Listing 12 (p. 10) .

Listing 12: Print the references to the two objects.

```
        System.out.print(((Prob05X)var1[0]) + " ");
    System.out.print(randomData + 5 + " ");
    System.out.println(((Prob05X)var1[1]));

  }//end main
}//end class Prob05
```

An automatic call to the toString method

Whenever an object's reference is passed to either the **print** method or the **println** method, the first thing that happens is that the **toString** method is called on the reference. The **toString** method always returns a reference to an object of the **String** class, and it is that string that is printed.

Inherited default behavior of the toString method

As I mentioned earlier, the **toString** method is defined with default behavior in the **Object** class. Since every class is a subclass of the **Object** class, every class inherits that method.

If the **toString** method is not overridden in a class or in any of the superclasses of a given class and the **toString** method is called on an object of the given class, the default behavior of the **toString** method will occur.

Can override to change the behavior

However, any class can override the **toString** method to produce different behavior and can pass that behavior down the inheritance hierarchy to subclasses of the class that overrides the method.

The toString method is overridden

In this program, the **toString** method is overridden in exactly the same way in both the **Prob05MyClassB** class and the **Prob05MyClassB** class. *(See Listing 6 (p. 6) and Listing 9 (p. 8) .)* Therefore, when the **toString** method is called on an object of either class, it will return a string representation of the value stored in the variable named **data** plus 5.

Pass object references to the print and println methods

The first statement in Listing 12 (p. 10) passes the reference to the object stored in the first element of the array to the **print** method and the third statement passes the reference to the object stored in the second element of the array to the **println** method.

Execute overridden toString methods and print the returned values

The **print** and **println** methods cause the code in Listing 6 (p. 6) and Listing 9 (p. 8) to be executed. In both cases, this code returns a string that represents the original random value plus 5. This is the value that is displayed.

Print the random value plus 5

The second statement in Listing 12 (p. 10) adds five to the original random number and prints the result. These three statements produce the third line of text in Figure 1 (p. 3) where all three values are the algebraic sum of the original random number plus 5.

Important - The cast is not required

Even though the references extracted from the array in the first and third statements in Listing 12 (p. 10) are cast to the interface type **Prob05X** , that cast is unnecessary.

Because the original definition of the **toString** method appears in the class named **Object** , the **toString** method can be called on those objects even while they are being treated as though they are of type **Object** .

Runtime polymorphism

Furthermore, a very powerful capability of OOP known as runtime polymorphism would cause the overridden versions of the methods defined in Listing 6 (p. 6) and Listing 9 (p. 8) to be executed instead of the default version of the method defined in the **Object** class.

The end of the main method

Listing 12 (p. 10) signals the end of the **main** method and the end of the class named **Prob05** . When the **main** method has nothing further to do, it terminates causing the program to terminate and return control to the operating system.

6 Run the program

I encourage you to copy the code from Listing 13 (p. 12) . Compile the code and execute it. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

7 Summary

In this module, you learned about :

- Interface definitions
- Implementing an interface in a class definition
- Defining interface methods in a class definition
- Storing references to new objects in elements of an array of type **Object**
- Casting elements to an interface type in order to call interface methods
- Parameterized constructors
- Overridden **toString** method

8 What's next?

You will learn how to scale images and how to rotate and translate images using the AffineTransform class in the next module.

9 Online video links

Select the following links to view online video lectures on the material in this module.

- ITSE 2321 Lecture 10 [2]
    - Part01 [3]

[2]http://www.youtube.com/playlist?list=PL3DB0B7840C943C4C
[3]http://www.youtube.com/watch?v=10R_Xgo9QEo

- Part02 [4]
- Part03 [5]
- Part04 [6]

10 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Interfaces, Object Arrays, etc.
- File: Java3020.htm
- Published: August 2, 2012
- Revised: November 14, 2012

NOTE: **Disclaimers:** **Financial** : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

11 Complete program listing

A complete listing of the program discussed in this module is shown in Listing 13 (p. 12) below.

Listing 13: Complete program listing.

```
    /*File Prob05 Copyright 2008 R.G.Baldwin
*********************************************************/

import java.util.*;

class Prob05{
  public static void main(String[] args){

    Random generator = new Random(new Date().getTime());
    int randomData = (byte)generator.nextInt();
```

[4]http://www.youtube.com/watch?v=vNPd6Sd7Wk8
[5]http://www.youtube.com/watch?v=_JFcPromgGk
[6]http://www.youtube.com/watch?v=A3bgpy5dCtQ

```
    Object[] var1 = new Object[2];

    var1[0] = new Prob05MyClassA(randomData);
    var1[1] = new Prob05MyClassB(randomData);

    System.out.print(
              ((Prob05X)var1[0]).getModifiedData() + " ");
    System.out.print(randomData + " ");
    System.out.println(
                    ((Prob05X)var1[1]).getModifiedData());

    System.out.print(((Prob05X)var1[0]).getData() + " ");
    System.out.print(randomData + " ");
    System.out.println(((Prob05X)var1[1]).getData());

    System.out.print(((Prob05X)var1[0]) + " ");
    System.out.print(randomData + 5 + " ");
    System.out.println(((Prob05X)var1[1]));

  }//end main
}//end class Prob05
//=======================================================//

interface Prob05X{
  public int getModifiedData();
  public int getData();
}//end interface
//=======================================================//

class Prob05MyClassA implements Prob05X{
  private int data;

  Prob05MyClassA(int inData){
    System.out.println("Prob05");
    System.out.println("Put your first name here");
    data = inData;
  }//end constructor
  //---------------------------------------------------//

  public int getModifiedData(){
    return data - 1;
  }//end getModifiedData()
  //---------------------------------------------------//

  public int getData(){
    return data;
  }//end getData()
  //---------------------------------------------------//

  public String toString(){
    return "" + (data + 5);
  }//end toString()
}//end class Prob05MyClassA
//======================================================//

class Prob05MyClassB implements Prob05X{
  private int data;

  Prob05MyClassB(int inData){
    System.out.println("Put your last name here");
    data = inData;
  }//end constructor

  public int getModifiedData(){
    return data + 1;
  }//end getModifiedData()

  public int getData(){
    return data;
  }//end getData()

  public String toString(){
    return "" + (data + 5);
  }//end toString()
}//end class Prob05MyClassB
```

-end-