

AP0060: SELF-ASSESSMENT, MORE ON ARRAYS*

R.G. (Dick) Baldwin

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 3.0[†]

Abstract

Part of a self-assessment test designed to help you determine how much you know about arrays in Java.

1 Table of Contents

- Preface (p. 1)
- Questions (p. 1)
 - 1 (p. 1) , 2 (p. 2) , 3 (p. 3) , 4 (p. 3) , 5 (p. 4) , 6 (p. 5) , 7 (p. 6) , 8 (p. 7) , 9 (p. 8) , 10 (p. 9) , 11 (p. 9) , 12 (p. 10) , 13 (p. 11) , 14 (p. 12) , 15 (p. 13)
- Listings (p. 13)
- Miscellaneous (p. 14)
- Answers (p. 14)

2 Preface

This module is part of a self-assessment test designed to help you determine how much you know about object-oriented programming using Java.

The test consists of a series of questions with answers and explanations of the answers.

The questions and the answers are connected by hyperlinks to make it easy for you to navigate from the question to the answer and back.

I recommend that you open another copy of this document in a separate browser window and use the links to under Listings (p. 13) to easily find and view the listings while you are reading about them.

3 Questions

3.1 Question 1 .

What output is produced by the program shown in Listing 1 (p. 2) ?

- A. Compiler Error

*Version 1.3: Dec 3, 2012 9:43 pm -0600

[†]<http://creativecommons.org/licenses/by/3.0/>

- B. Runtime Error
- C. I'm OK
- D. None of the above

Listing 1: Listing for Question 1.

```
public class Ap064{
public static void main(
                String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    double[] A = new double[2];
    A[0] = 1.0;
    A[1] = 2.0;
    Object B = A;

    System.out.println("I'm OK");
} //end doArrays()
} // end class
```

Answer and Explanation (p. 22)

3.2 Question 2

What output is produced by the program shown in Listing 2 (p. 2) ?

- A. Compiler Error
- B. Runtime Error
- C. 1.0 2.0
- D. None of the above.

Listing 2: Listing for Question 2.

```
public class Ap065{
public static void main(
                String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    double[] A = new double[2];
    A[0] = 1.0;
    A[1] = 2.0;
    Object B = A;
```

```
        System.out.println(
            B[0] + " " + B[1]);
    }//end doArrays()
}// end class
```

Answer and Explanation (p. 22)

3.3 Question 3

What output is produced by the program shown in Listing 3 (p. 3) ?

- A. Compiler Error
- B. Runtime Error
- C. 1.0 2.0
- D. None of the above.

Listing 3: Listing for Question 3.

```
public class Ap066{
public static void main(
    String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    double[] A = new double[2];
    A[0] = 1.0;
    A[1] = 2.0;
    Object B = A;

    double C = (double)B;
    System.out.println(
        C[0] + " " + C[1]);
} //end doArrays()
} // end class
```

Answer and Explanation (p. 21)

3.4 Question 4

What output is produced by the program shown in Listing 4 (p. 3) ?

- A. Compiler Error
- B. Runtime Error
- C. 1.0 2.0
- D. None of the above.

Listing 4: Listing for Question 4.

```

    public class Ap067{
    public static void main(
                        String args[]){
        new Worker().doArrays();
    }//end main()
}//end class definition

class Worker{
    public void doArrays(){
        double[] A = new double[2];
        A[0] = 1.0;
        A[1] = 2.0;
        Object B = A;

        double[] C = (double[])B;
        System.out.println(
                        C[0] + " " + C[1]);
    }//end doArrays()
}// end class

```

Answer and Explanation (p. 21)

3.5 Question 5

What output is produced by the program shown in Listing 5 (p. 4) ?

- A. Compiler Error
- B. Runtime Error
- C. 1.0 2.0
- D. None of the above.

Listing 5: Listing for Question 5.

```

    public class Ap068{
    public static void main(
                        String args[]){
        new Worker().doArrays();
    }//end main()
}//end class definition

class Worker{
    public void doArrays(){
        double[] A = new double[2];
        A[0] = 1.0;
        A[1] = 2.0;
        Object B = A;

        String[] C = (String[])B;
        System.out.println(
                        C[0] + " " + C[1]);
    }//end doArrays()
}// end class

```

Answer and Explanation (p. 20)

3.6 Question 6

What output is produced by the program shown in Listing 6 (p. 5) ?

- A. Compiler Error
- B. Runtime Error
- C. 1 2
- D. None of the above

Listing 6: Listing for Question 6.

```
public class Ap069{
public static void main(
                        String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    Subclass[] A = new Subclass[2];
    A[0] = new Subclass(1);
    A[1] = new Subclass(2);

    System.out.println(
        A[0] + " " + A[1]);
} //end doArrays()
} // end class

class Superclass{
private int data;
public Superclass(int data){
    this.data = data;
} //end constructor

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

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

class Subclass extends Superclass{
public Subclass(int data){
    super(data);
} //end constructor
} //end class Subclass
```

Answer and Explanation (p. 20)

3.7 Question 7

What output is produced by the program shown in Listing 7 (p. 6) ?

- A. Compiler Error
- B. Runtime Error
- C. 1 2
- D. None of the above

Listing 7: Listing for Question 7.

```
public class Ap070{
public static void main(
    String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    Subclass[] A = new Subclass[2];
    A[0] = new Subclass(1);
    A[1] = new Subclass(2);

    Superclass[] B = A;
    System.out.println(
        B[0] + " " + B[1]);
} //end doArrays()
} // end class

class Superclass{
private int data;
public Superclass(int data){
    this.data = data;
} //end constructor

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

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

class Subclass extends Superclass{
public Subclass(int data){
    super(data);
} //end constructor
} //end class Subclass
```

Answer and Explanation (p. 20)

3.8 Question 8

What output is produced by the program shown in Listing 8 (p. 7) ?

- A. Compiler Error
- B. Runtime Error
- C. 1 2
- D. None of the above

Listing 8: Listing for Question 8.

```
public class Ap071{
public static void main(
    String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    Superclass[] A = new Superclass[2];
    A[0] = new Superclass(1);
    A[1] = new Superclass(2);

    Subclass[] B = (Subclass[])A;
    System.out.println(
        B[0] + " " + B[1]);
} //end doArrays()
} // end class

class Superclass{
private int data;
public Superclass(int data){
    this.data = data;
} //end constructor

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

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

class Subclass extends Superclass{
public Subclass(int data){
    super(data);
} //end constructor
} //end class Subclass
```

Answer and Explanation (p. 19)

3.9 Question 9

What output is produced by the program shown in Listing 9 (p. 8) ?

- A. Compiler Error
- B. Runtime Error
- C. 1 2
- D. None of the above

Listing 9: Listing for Question 9.

```
public class Ap072{
public static void main(
                String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    Subclass[] A = new Subclass[2];
    A[0] = new Subclass(1);
    A[1] = new Subclass(2);

    Superclass[] B = A;
    Subclass[] C = (Subclass[])B;
    System.out.println(
                C[0] + " " + C[1]);
} //end doArrays()
} // end class

class Superclass{
private int data;
public Superclass(int data){
    this.data = data;
} //end constructor

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

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

class Subclass extends Superclass{
public Subclass(int data){
    super(data);
}
```

```

} //end constructor
} //end class Subclass

```

Answer and Explanation (p. 19)

3.10 Question 10

What output is produced by the program shown in Listing 10 (p. 9) ?

- A. Compiler Error
- B. Runtime Error
- C. 1.0 2.0
- D. D. None of the above

Listing 10: Listing for Question 10.

```

public class Ap073{
public static void main(
                        String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    double[] A = new double[2];
    A[0] = 1.0;
    A[1] = 2.0;
    Object B = A;

    System.out.println(
        ((double[])B)[0] + " " +
        ((double[])B)[1]);
} //end doArrays()
} // end class

```

Answer and Explanation (p. 18)

3.11 Question 11

What output is produced by the program shown in Listing 11 (p. 9) ?

- A. Compiler Error
- B. Runtime Error
- C. 1 2
- D. D. None of the above

Listing 11: Listing for Question 11.

```
public class Ap074{
public static void main(
                String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    int[] A = new int[2];
    A[0] = 1;
    A[1] = 2;

    double[] B = (double[])A;

    System.out.println(
                B[0] + " " + B[1]);
} //end doArrays()
} // end class
```

Answer and Explanation (p. 17)

3.12 Question 12

What output is produced by the program shown in Listing 12 (p. 10) ?

- A. Compiler Error
- B. Runtime Error
- C. 1 2
- D. None of the above

Listing 12: Listing for Question 12.

```
public class Ap075{
public static void main(
                String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    int[] B = returnArray();
    for(int i = 0; i < B.length;i++){
        System.out.print(B[i] + " ");
    } //end for loop
    System.out.println();
} //end doArrays()

public int[] returnArray(){
    int[] A = new int[2];
```

```
A[0] = 1;
A[1] = 2;
return A;
} //end returnArray()
} // end class
```

Answer and Explanation (p. 17)

3.13 Question 13

What output is produced by the program shown in Listing 13 (p. 11) ?

NOTE:

- A. Compiler Error
- B. Runtime Error
- C. 0 0 0
0 1 2
- D. None of the above

Listing 13: Listing for Question 13.

```
public class Ap076{
public static void main(
                String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    int[] A[];
    A = new int[2][3];

    for(int i=0; i<A.length;i++){
        for(int j=0;j<A[0].length;j++){
            A[i][j] = i*j;
        } //end inner loop
    } //end outer loop

    for(int i=0; i<A.length;i++){
        for(int j=0;j<A[0].length;j++){
            System.out.print(
                A[i][j] + " ");
        } //end inner loop
        System.out.println();
    }
}
```

```

    }//end outer loop

} //end doArrays()
} // end class

```

Answer and Explanation (p. 16)

3.14 Question 14

What output is produced by the program shown in Listing 14 (p. 12) ?

- A. Compiler Error
- B. Runtime Error
- C. 1 2
- D. None of the above

Listing 14: Listing for Question 14.

```

public class Ap077{
public static void main(
                        String args[]){
    new Worker().doArrays();
} //end main()
} //end class definition

class Worker{
public void doArrays(){
    Subclass[] A = new Subclass[2];
    A[0] = new Subclass(1);
    A[1] = new Subclass(2);

    Object X = A;
    Superclass B = A;
    Subclass[] C = (Subclass[])B;
    Subclass[] Y = (Subclass[])X;
    System.out.println(
        C[0] + " " + Y[1]);
} //end doArrays()
} // end class

class Superclass{
private int data;
public Superclass(int data){
    this.data = data;
} //end constructor

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

public String toString(){

```

```

        return "" + data;
    }//end toString()
}//end class SuperClass

class Subclass extends Superclass{
    public Subclass(int data){
        super(data);
    }//end constructor
}//end class Subclass

```

Answer and Explanation (p. 15)

3.15 Question 15

What output is produced by the program shown in Listing 15 (p. 13) ?

- A. Compiler Error
- B. Runtime Error
- C. 0 0.0 false 0
- D. None of the above

Listing 15: Listing for Question 15.

```

    public class Ap078{
    public static void main(
                String args[]){
        new Worker().doArrays();
    }//end main()
}//end class definition

class Worker{
    public void doArrays(){
        int[] A = new int[1];
        double[] B = new double[1];
        boolean[] C = new boolean[1];
        int[] D = new int[0];

        System.out.println(A[0] + " " +
                B[0] + " " +
                C[0] + " " +
                D.length);

    }//end doArrays()
}// end class

```

Answer and Explanation (p. 14)

4 Listings

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 listings while you are reading about them.

- Listing 1 (p. 2) . Listing for Question 1.
- Listing 2 (p. 2) . Listing for Question 2.
- Listing 3 (p. 3) . Listing for Question 3.
- Listing 4 (p. 3) . Listing for Question 4.
- Listing 5 (p. 4) . Listing for Question 5.
- Listing 6 (p. 5) . Listing for Question 6.
- Listing 7 (p. 6) . Listing for Question 7.
- Listing 8 (p. 7) . Listing for Question 8.
- Listing 9 (p. 8) . Listing for Question 9.
- Listing 10 (p. 9) . Listing for Question 10.
- Listing 11 (p. 9) . Listing for Question 11.
- Listing 12 (p. 10) . Listing for Question 12.
- Listing 13 (p. 11) . Listing for Question 13.
- Listing 14 (p. 12) . Listing for Question 14.
- Listing 15 (p. 13) . Listing for Question 15.

5 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: Housekeeping material

- Module name: Ap0060: Self-assessment, More on Arrays
- File: Ap0060.htm
- Originally published: 2002
- Published at cnx.org: December 3, 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.

6 Answers

6.1 Answer 15

C. 0 0.0 false 0

6.1.1 Explanation 15

You can initialize array elements

You can create a new array object and initialize its elements using statements similar to the following:

NOTE:

```
int[] A = {22, 43, 69};
X[] B = {new X(32), new X(21)};
```

What if you don't initialize array elements?

If you create a new array object without initializing its elements, the value of each element in the array is automatically initialized to a default value.

Illustrating array element default initialization

This program illustrates default initialization of **int** , **double** , and **boolean** arrays.

The default values are as follows:

- zero for all numeric values
- false for all **boolean** values
- all zero bits for char values
- null for object references

An array with no elements ...

This program also illustrates that it is possible to have an array object in Java that has no elements. In this case, the value of the **length** property for the array object is 0.

Give me an example

For example, when the user doesn't enter any arguments on the command line for a Java application, the incoming **String** array parameter to the **main** method has a length value of 0.

Another example

It is also possible that methods that return a reference to an array object may sometimes return a reference to an array whose length is 0. The method must satisfy the return type requirement by returning a reference to an array object. Sometimes, there is no data to be used to populate the array, so the method will simply return a reference to an array object with a **length** property value of 0.

Back to Question 15 (p. 13)

6.2 Answer 14

A. Compiler Error

6.2.1 Explanation 14

Assigning array reference to type **Object**

As you learned in an earlier module, you can assign an array object's reference to an ordinary reference variable of the type **Object** . It is not necessary to indicate that the reference variable is a reference to an array by appending square brackets to the type name or the variable name.

Only works with type **Object**

However, you cannot assign an array object's reference to an ordinary reference variable of any other type. For any type other than **Object** , the reference variable must be declared to hold a reference to an array object by appending empty square brackets onto the type name or the variable name.

The first statement in the following fragment compiles successfully.

NOTE:

```

    Object X = A;
    Superclass B = A;

```

However, the second statement in the above fragment produces a compiler error under JDK 1.3, which is partially reproduced below.

NOTE:

```

Ap077.java:22: incompatible types
found   : Subclass[]
required: Superclass
    Superclass B = A;

```

Both **Superclass** and **Object** are superclasses of the array type referred to by the reference variable named **A** . However, because of the above rule, in order to cause this program to compile successfully, you would need to modify it as shown below by adding the requisite empty square brackets to the **Superclass** type name.

NOTE:

```

    Object X = A;
    Superclass[] B = A;

```

Back to Question 14 (p. 12)

6.3 Answer 13

NOTE:

```

C.  0 0 0
    0 1 2

```

6.3.1 Explanation 13

Syntactical ugliness

As I indicated in an earlier module, when declaring a reference variable that will refer to an array object, you can place the empty square brackets next to the name of the type or next to the name of the reference variable. In other words, either of the following formats will work.

NOTE:

```

    int[][] A;
    int B[][];

```

What I may not have told you at that time is that you can place some of the empty square brackets in one location and the remainder in the other location.

Really ugly syntax

This is indicated by the following fragment, which declares a reference variable for a two-dimensional array of type **int** . Then it creates the two-dimensional array object and assigns the array object's reference to the reference variable.

NOTE:

```
int[] A[];  
A = new int[2][3];
```

While it doesn't matter which location you use for the square brackets in the declaration, it does matter how many pairs of square brackets you place in the two locations combined. The number of dimensions on the array (*if you want to think of a Java array as having dimensions*) will equal the total number of pairs of empty square brackets in the declaration of the reference variable. Thus, in this case, the array is a two-dimensional array because there is one pair of square brackets next to the type and another pair next to the variable name.

This program goes on to use nested for loops to populate the array and then to display the contents of the elements.

I personally don't use this syntax, and I hope that you don't either. However, even if you don't use it, you need to be able to recognize it when used by others.

Back to Question 13 (p. 11)

6.4 Answer 12

C. 1 2

6.4.1 Explanation 12

The length property

This program illustrates the use of the array property named **length**, whose value always matches the number of elements in the array.

As a Java programmer, you will frequently call methods that will return a reference to an array object of a specified type, but of an unknown length. (*See, for example, the method named **getEventSetDescriptors** that is declared in the interface named **BeanInfo**.*) This program simulates that situation.

Returning a reference to an array

The method named **returnArray** returns a reference to an array of type **int** having two elements. Although I fixed the size of the array in this example, I could just as easily have used a random number to set a different size for the array each time the method is called. Therefore, the **doArrays** method making the call to the method named **returnArray** has no way of knowing the size of the array referred to by the reference that it receives as a return value.

All array objects have a length property

This could be a problem, but Java provides the solution to the problem in the **length** property belonging to all array objects.

The **for** loop in the method named **doArrays** uses the **length** property of the array to determine how many elements it needs to display. This is a very common scenario in Java.

Back to Question 12 (p. 10)

6.5 Answer 11

A. Compiler Error

6.5.1 Explanation 11

You cannot cast primitive array references

You cannot cast an array reference from one primitive type to another primitive type, even if the individual elements in the array are of a type that can normally be converted to the new type.

This program attempts to cast a reference to an array of type **int[]** and assign it to a reference variable of type **double []**. Normally, a value of type **int** will be automatically converted to type **double** whenever there is a need for such a conversion. However, this attempted cast produces the following compiler error under JDK 1.3.

NOTE:

```
Ap074.java:19: inconvertible types
found   : int[]
required: double[]
double[] B = (double[])A;
```

Why is this cast not allowed?

I can't give you a firm reason why such a cast is not allowed, but I believe that I have a good idea why. I speculate that this is due to the fact that the actual primitive values are physically stored in the array object, and primitive values of different types require different amounts of storage. For example, the type **int** requires 32 bits of storage while the type **double** requires 64 bits of storage.

Would require reconstructing the array object

Therefore, to convert an array object containing **int** values to an array object containing **double** values would require reconstructing the array object and allocating twice as much storage space for each element in the array.

Restriction doesn't apply to arrays of references

As you have seen from previous questions, such a casting restriction does not apply to arrays containing references to objects. This may be because the amount of storage required to store a reference to an object is the same, regardless of the type of the object. Therefore, the allowable casts that you have seen in the previous questions did not require any change to the size of the array. All that changed was some supplemental information regarding the type of objects to which the elements in the array refer.

Back to Question 11 (p. 9)

6.6 Answer 10

C. 1.0 2.0

6.6.1 Explanation 10

Assigning array reference to variable of type Object

A reference to an array can be assigned to a non-array reference of the class named **Object**, as in the following statement extracted from the program, where A is a reference to an array object of type **double**.

NOTE:

```
Object B = A;
```

Note that there are no square brackets anywhere in the above statement. Thus, the reference to the array object is not being assigned to an array reference of the type **Object[]**. Rather, it is being assigned to an ordinary reference variable of the type **Object**.

Downcasting to an array type

Once the array reference has been assigned to the ordinary reference variable of the type **Object**, that reference variable can be downcast and used to access the individual elements in the array as illustrated in the following fragment. Note the empty square brackets in the syntax of the cast operator **(double[])**.

NOTE:

```
System.out.println(
    ((double[])B)[0] + " " +
    ((double[])B)[1]);
```

Placement of parentheses is critical

Note also that due to precedence issues, the placement of both sets of parentheses is critical in the above code fragment. You must downcast the reference variable before applying the index to that variable.

Back to Question 10 (p. 9)

6.7 Answer 9

C. 1 2

6.7.1 Explanation 9**General array casting rule**

The general rule for casting array references (*for arrays whose declared type is the name of a class or an interface*) is:

A reference to an array object can be cast to another array type if the elements of the referenced array are of a type that can be cast to the type of the elements of the specified array type.

Old rules apply here also

Thus, the general rules covering conversion and casting up and down the inheritance hierarchy and among classes that implement the same interfaces also apply to the casting of references to array objects.

A reference to an object can be cast down the inheritance hierarchy to the actual class of the object. Therefore, an array reference can also be cast down the inheritance hierarchy to the declared class for the array object.

This program declares a reference to, creates, and populates an array of the class type **Subclass** . This reference is assigned to an array reference of a type that is a superclass of the actual class type of the array. Then the superclass reference is downcast to the actual class type of the array and assigned to a different reference variable. This third reference variable is used to successfully access and display the contents of the elements in the array.

Back to Question 9 (p. 8)

6.8 Answer 8

B. Runtime Error

6.8.1 Explanation 8**Another ClassCastException**

While it is allowable to assign an array reference to an array reference variable declared for a class that is further up the inheritance hierarchy (*as illustrated earlier*) , it is not allowable to cast an array reference down the inheritance hierarchy to a subclass of the original declared class for the array.

This program declares a reference for, creates, and populates a two-element array for a class named **Superclass** . Then it downcasts that reference to a subclass of the class named **Superclass** . The compiler is unable to determine that this is a problem. However, the runtime system throws the following exception, which terminates the program at runtime.

NOTE:

```
java.lang.ClassCastException: [LSuperclass;
at Worker.doArrays(Ap071.java:19)
at Ap071.main(Ap071.java:9)
```

Back to Question 8 (p. 7)

6.9 Answer 7

C. 1 2

6.9.1 Explanation 7

Assignment to superclass array reference variable

This program illustrates that, if you have a reference to an array object containing references to other objects, you can assign the array object's reference to an array reference variable whose type is a superclass of the declared class of the array object. (*As we will see later, this doesn't work for array objects containing primitive values.*)

What can you do then?

Having made the assignment to the superclass reference variable, whether or not you can do anything useful with the elements in the array (*without downcasting*) depends on many factors.

No downcast required in this case

In this case, the ability to display the contents of the objects referred to in the array was inherited from the class named **Superclass**. Therefore, it is possible to access and display a **String** representation of the objects without downcasting the array object reference from **Superclass** to the actual type of the objects.

Probably need to downcast in most cases

However, that will often not be the case. In most cases, when using a reference of a superclass type, you will probably need to downcast in order to make effective use of the elements in the array object.

Back to Question 7 (p. 6)

6.10 Answer 6

C. 1 2

6.10.1 Explanation 6

Straightforward array application

This is a straightforward application of Java array technology for the storage and retrieval of references to objects.

The program declares a reference to, creates, and populates a two-element array of a class named **Subclass**. The class named **Subclass** extends the class named **Superclass**, which in turn, extends the class named **Object** by default.

The super keyword

The class named **Subclass** doesn't do anything particularly useful other than to illustrate extending a class.

However, it also provides a preview of the use of the **super** keyword for the purpose of causing a constructor in a subclass to call a parameterized constructor in its superclass.

Setting the stage for follow-on questions

The main purpose for showing you this program is to set the stage for several programs that will be using this class structure in follow-on questions.

Back to Question 6 (p. 5)

6.11 Answer 5

B. Runtime Error

6.11.1 Explanation 5

ClassCastException

There are some situations involving casting where the compiler cannot identify an erroneous condition that is later identified by the runtime system. This is one of those cases.

This program begins with an array of type `double []`. The reference to that array is converted to type `Object`. Then it is cast to type `String []`. All of these operations are allowed by the compiler.

However, at runtime, the runtime system expects to find references to objects of type `String` in the elements of the array. What it finds instead is values of type `double` stored in the elements of the array.

As a result, a `ClassCastException` is thrown. Since it isn't caught and handled by the program, the program terminates with the following error message showing on the screen.

NOTE:

```
java.lang.ClassCastException: [D
at Worker.doArrays(Ap068.java:17)
at Ap068.main(Ap068.java:6)
```

Back to Question 5 (p. 4)

6.12 Answer 4

C. 1.0 2.0

6.12.1 Explanation 4

Finally, we got it right

Finally, we managed to get it all together. The program compiles and executes correctly. This program illustrates the assignment of an array object's reference to a reference variable of type `Object`, and the casting of that reference of type `Object` back to the correct array type in order to gain access to the elements in the array.

But don't go away, there is a lot more that you need to know about arrays in Java. We will look at some of those things in the questions that follow.

Back to Question 4 (p. 3)

6.13 Answer 3

A. Compiler Error

6.13.1 Explanation 3

Must use the correct cast syntax

While it is possible to store an array object's reference in a reference variable of type `Object`, and later cast it back to an array type to gain access to the elements in the array, you must use the correct syntax in performing the cast. This is not the correct syntax for performing that cast. It is missing the empty square brackets required to indicate a reference to an array object.

A portion of the compiler error produced by JDK 1.3 is shown below:

NOTE:

```
Ap066.java:17: inconvertible types
found   : java.lang.Object
required: double
    double C = (double)B;
```

Back to Question 3 (p. 3)

6.14 Answer 2

A. Compiler Error

6.14.1 Explanation 2

Must cast back to an array type

This program illustrates another very important point. Although you can assign an array object's reference to a reference variable of type **Object**, you cannot gain access to the elements in the array while treating it as type **Object**. Instead, you must cast it back to an array type before you can gain access to the elements in the array object.

A portion of the compiler error produced by JDK 1.3 is shown below:

NOTE:

```
Ap065.java:18: array required, but java.lang.Object found
B[0] + " " + B[1]);
```

Back to Question 2 (p. 2)

6.15 Answer 1

C. I'm OK

6.15.1 Explanation 1

Assigning array reference to type Object

This program illustrates a very important point. You can assign an array object's reference to an ordinary reference variable of type **Object**. Note that I didn't say **Object[]**. The empty square brackets are not required when the type is **Object**.

Standard containers or collections

Later on, when we study the various containers in the Java class libraries (*see the Java Collections Framework*), we will see that they store references to all objects, including array objects, as type **Object**. Thus, if it were not possible to store a reference to an array object in a reference variable of type **Object**, it would not be possible to use the standard containers to store references to array objects.

Because it is possible to assign an array object's reference to a variable of type **Object**, it is also possible to store array object references in containers of type **Object**.

Back to Question 1 (p. 1)

-end-