17.6.1 The Serializable Interface Not every object can be written to an output stream. Objects that can be so written are said to be serializable. A serializable object is an instance of the java.io.Serializable interface, so the object’s class must implement Serializable.

The Serializable interface is a marker interface. Since it has no methods, you don’t need to add additional code in your class that implements Serializable. Implementing this interface enables the Java serialization mechanism to automate the process of storing objects and arrays. To appreciate this automation feature, consider what you otherwise need to do in order to store an object. Suppose that you wish to store an ArrayList object. To do this, you need to store all the elements in the list. Each element is an object that may contain other objects. As you can see, this would be a very tedious process. Fortunately, you don’t have to go through it manually. Java provides a built-in mechanism to automate the process of writing objects. This process is referred as objectserialization, which is implemented in ObjectOutputStream. In contrast, the process of reading objects is referred as object deserialization, which is implemented in ObjectInputStream. Many classes in the Java API implement Serializable. All the wrapper classes for primitive-type values: java.math.BigInteger, java.math.BigDecimal, java.lang. String, java.lang.StringBuilder, java.lang.StringBuffer, java.util.Date, and java.util.ArrayList implement java.io.Serializable. Attempting to store an object that does not support the Serializable interface would cause a NotSerializableException. When a serializable object is stored, the class of the object is encoded; this includes the class name and the signature of the class, the values of the object’s instance variables, and the closure of any other objects referenced by the object. The values of the object’s static variables are not stored.

Note Nonserializable fields If an object is an instance of Serializable but contains nonserializable instance data fields, can it be serialized? The answer is no. To enable the object to be serialized, mark these data fields with the transient keyword to tell the JVM to ignore them when writing the object to an object stream. Consider the following class: public class C implements java.io.Serializable { private int v1; private static double v2; private transient A v3 = new A(); } class A { } // A is not serializable When an object of the C class is serialized, only variable v1 is serialized. Variable v2 is not serialized because it is a static variable, and variable v3 is not serialized because it is marked transient. If v3 were not marked transient, a java.io.NotSerializableException would occur.

Note Duplicate objects If an object is written to an object stream more than once, will it be stored in multiple copies? No, it will not. When an object is written for the first time, a serial number is created for it. The JVM writes the complete contents of the object along with the serial number into the object stream. After the first time, only the serial number is stored if the same object is written again. When the objects are read back, their references are the same since only one object is actually created in the memory.

**17.6.2 Serializing Arrays**

An array is serializable if all its elements are serializable. An entire array can be saved into a file using writeObject and later can be restored using readObject. Listing 17.7 stores an array of five int values and an array of three strings, and reads them back to display on the console.

1. import java.io.\*;
2. public class TestObjectStreamForArray {
3. public static void main(String[] args)
4. throws ClassNotFoundException, IOException {
5. int[] numbers = {1, 2, 3, 4, 5};
6. String[] strings = {"John", "Susan", "Kim"};
7. try ( // Create an output stream for file array.dat
8. ObjectOutputStream output = new ObjectOutputStream(new
9. FileOutputStream("array.dat", true));
10. ) {
11. // Write arrays to the object output stream
12. output.writeObject(numbers);
13. output.writeObject(strings);
14. }
15. try ( // Create an input stream for file array.dat
16. ObjectInputStream input =
17. new ObjectInputStream(new FileInputStream("array.dat"));
18. ) {
19. int[] newNumbers = (int[])(input.readObject());
20. String[] newStrings = (String[])(input.readObject());
22. // Display arrays
23. for (int i = 0; i < newNumbers.length; i++)
24. System.out.print(newNumbers[i] + " ");
25. System.out.println();
27. for (int i = 0; i < newStrings.length; i++)
28. System.out.print(newStrings[i] + " ");
29. }
30. }
31. }

Lines 14–15 write two arrays into file array.dat. Lines 22–23 read two arrays back in the same order they were written. Since readObject() returns Object, casting is used to cast the objects into int[] and String[]

**Listing 17.8 TestRandomAccessFile.java**

1. import java.io.\*;
2. public class TestRandomAccessFile {
3. public static void main(String[] args) throws IOException {
4. try ( // Create a random access file
5. RandomAccessFile inout = new RandomAccessFile("inout.dat", "rw");
6. ) {
7. // Clear the file to destroy the old contents if exists
8. inout.setLength(0);
10. // Write new integers to the file
11. for (int i = 0; i < 200; i++)
12. inout.writeInt(i);
14. // Display the current length of the file
15. System.out.println("Current file length is " + inout.length());
17. // Retrieve the first number
18. inout.seek(0); // Move the file pointer to the beginning
19. System.out.println("The first number is " + inout.readInt());
21. // Retrieve the second number
22. inout.seek(1 \* 4); // Move the file pointer to the second number
23. System.out.println("The second number is " + inout.readInt());
25. // Retrieve the tenth number
26. inout.seek(9 \* 4); // Move the file pointer to the tenth number
27. System.out.println("The tenth number is " + inout.readInt());
29. // Modify the eleventh number
30. inout.writeInt(555);
32. // Append a new number
33. inout.seek(inout.length()); // Move the file pointer to the end
34. inout.writeInt(999);
36. // Display the new length
37. System.out.println("The new length is " + inout.length());
39. // Retrieve the new eleventh number
40. inout.seek(10 \* 4); // Move the file pointer to the eleventh number
41. System.out.println("The eleventh number is " + inout.readInt());
42. }
43. }
44. }

A RandomAccessFile is created for the file named inout.dat with mode rw to allow both read and write operations in line 6. 714 Chapter 17 Binary I/O inout.setLength(0) sets the length to 0 in line 9. This, in effect, deletes the old contents of the file. The for loop writes 200 int values from 0 to 199 into the file in lines 12–13. Since each int value takes 4 bytes, the total length of the file returned from inout.length() is now 800 (line 16), as shown in the sample output. Invoking inout.seek(0) in line 19 sets the file pointer to the beginning of the file. inout.readInt() reads the first value in line 20 and moves the file pointer to the next number. The second number is read in line 24. inout.seek(9 \* 4) (line 27) moves the file pointer to the tenth number. inout.readInt() reads the tenth number and moves the file pointer to the eleventh number in line 28. inout .write(555) writes a new eleventh number at the current position (line 31). The previous eleventh number is deleted. inout.seek(inout.length()) moves the file pointer to the end of the file (line 34). inout.writeInt(999) writes a 999 to the file (line 35). Now the length of the file is increased by 4, so inout.length() returns 804 (line 38). inout.seek(10 \* 4) moves the file pointer to the eleventh number in line 41. The new eleventh number, 555, is displayed in line 42.

CODE !

package COSC603\_2022\_03\_18;

import java.io.\*;

import java.util.Date;

public class C1\_ObjectStream {

public static void main(String[] *args*) throws IOException, ClassNotFoundException {

try (ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream("src/COSC603\_2022\_03\_18/object.dat")))

{

output.writeUTF("Dave");

output.writeDouble(100.5);

output.writeObject(new Date());

output.writeObject(someclass); /// it cause runtime error error because some class are not Seriazabale

}

File file= new File("src/COSC603\_2022\_03\_18/object.dat");

try (ObjectInputStream input = new ObjectInputStream(new FileInputStream(file)))

{

String name= input.readUTF();

double score= input.readDouble();

Date today= (Date)input.readObject(); // We can use try and catch or throws new exception: ClassNotFoundException

System.out.println(name+ " " + score + " " + today);

}

}

}

code 2

if use other classes and seriaizaed

package COSC603\_2022\_03\_18;

import java.io.\*;

import java.util.Date;

public class C1\_ObjectStream {

public static void main(String[] *args*) throws IOException, ClassNotFoundException {

MyClass someclass= new MyClass();

AnotherClass anotherClass= new AnotherClass();

try (ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream("src/COSC603\_2022\_03\_18/object.dat")))

{

output.writeUTF("Dave");

output.writeDouble(100.5);

output.writeObject(new Date());

output.writeObject(someclass); /// it cause runtime error error because some class are not Seriazabale

output.writeObject(anotherClass); /// it is ok because impelements Seriazabale

}

File file= new File("src/COSC603\_2022\_03\_18/object.dat");

try (ObjectInputStream input = new ObjectInputStream(new FileInputStream(file)))

{

String name= input.readUTF();

double score= input.readDouble();

Date today= (Date)input.readObject(); // We can use try and catch or throws new exception: ClassNotFoundException

System.out.println(name+ " " + score + " " + today);

}

}

}

/////////////////////////////////////////////////////////////////////////////

class MyClass{

}

class AnotherClass implements Serializable{

}

code3

package COSC603\_2022\_03\_18;

import java.io.\*;

import java.util.Date;

public class C1\_ObjectStream {

public static void main(String[] *args*) throws IOException, ClassNotFoundException {

MyClass someclass= new MyClass();

AnotherClass anotherClass= new AnotherClass();

try (ObjectOutputStream output = new ObjectOutputStream(new FileOutputStream("src/COSC603\_2022\_03\_18/object.dat")))

{

output.writeUTF("Dave");

output.writeDouble(100.5);

output.writeObject(new Date());

//output.writeObject(someclass); /// it cause runtime error error because some class are not Seriazabale

output.writeObject(anotherClass); /// it is ok because impelements Seriazabale

}

File file= new File("src/COSC603\_2022\_03\_18/object.dat");

try (ObjectInputStream input = new ObjectInputStream(new FileInputStream(file)))

{

String name= input.readUTF();

double score= input.readDouble();

Date today= (Date)input.readObject(); // We can use try and catch or throws new exception: ClassNotFoundException

AnotherClass aClass= (AnotherClass)input.readObject();

System.out.println(name+ " " + score + " " + today);

System.out.println(aClass.x);

}

}

}

/////////////////////////////////////////////////////////////////////////////

class MyClass{

}

class AnotherClass implements Serializable{

public int x=100;

private static int **y**=1000;

private ***transient*** MyClass someclass;

}