

java.io and Serialization

For more details on SUN Certifications, visit [JavaScjpDumps](http://www.javascribdumps.com)

Q: 01 Click the Task button.

Chain these constructors to create objects to read from a file named "in" and to write to a file named "out."

```
reader = [Place here] [Place here] "in" );  
writer = [Place here] [Place here] [Place here] "out" );
```

Constructors

<code>new FileReader()</code>	<code>new PrintReader()</code>	<code>new BufferedReader()</code>
<code>new BufferedWriter()</code>	<code>new FileWriter()</code>	<code>new PrintWriter()</code>

Solution:

```
reader = new BufferedReader(new FileReader("in");  
writer = new PrintWriter (new BufferedWriter (new FileWriter("out")));
```

Q: 02 Given:

```
12. import java.io.*;  
13. public class Forest implements Serializable {  
14. private Tree tree = new Tree();  
15. public static void main(String [] args) {  
16. Forest f = new Forest();  
17. try {  
18. FileOutputStream fs = new FileOutputStream("Forest.ser");  
19. ObjectOutputStream os = new ObjectOutputStream(fs);  
20. os.writeObject(f); os.close();  
21. } catch (Exception ex) { ex.printStackTrace(); }  
22. } }
```

23.

24. class Tree { }

What is the result?

- A. Compilation fails.
- B. An exception is thrown at runtime.
- C. An instance of Forest is serialized.
- D. An instance of Forest and an instance of Tree are both serialized.

Answer: B

Q: 03 Click the Task button.

Place the code fragments into position to use a BufferedReader to read in an entire text file.

```
class PrintFile {  
    public static void main(String[] args){  
        BufferedReader buffReader = null;  
        //more code here to initialize buffReader  
        try {  
            String temp;  
  
            while( Place here Place here ) {  
                System.out.println(temp);  
            }  
        } catch Place here   
            e.printStackTrace();  
        }  
    }  
}
```

Code Fragments

(temp = buffReader.readLine())

& & buffReader.hasNext()

(temp = buffReader.nextLine())

(IOException e) {

!= null

(FileNotFoundException e) {

Done

Solution:

1. (temp = buffReader.readLine())
2. != null
3. (IOException e){

Q: 04 Assuming that the serializeBanana() and the deserializeBanana() methods will correctly use Java serialization and given:

13. import java.io.*;

```

14. class Food implements Serializable {int good = 3;}
15. class Fruit extends Food {int juice = 5;}
16. public class Banana extends Fruit {
17. int yellow = 4;
18. public static void main(String [] args) {
19. Banana b = new Banana(); Banana b2 = new Banana();
20. b.serializeBanana(b); // assume correct serialization
21. b2 = b.deserializeBanana(); // assume correct
22. System.out.println("restore "+b2.yellow+ b2.juice+b2.good);
24. }
25. // more Banana methods go here 50. }

```

What is the result?

- A. restore 400 B. restore 403
- C. restore 453 D. Compilation fails.
- E. An exception is thrown at runtime.

Answer: C

Q: 05 Which three statements concerning the use of the java.io.Serializable interface are true? (Choose three.)

- A. Objects from classes that use aggregation cannot be serialized.
- B. An object serialized on one JVM can be successfully deserialized on a different JVM.
- C. The values in fields with the volatile modifier will NOT survive serialization and deserialization.
- D. The values in fields with the transient modifier will NOT survive serialization and deserialization.
- E. It is legal to serialize an object of a type that has a supertype that does NOT implement java.io.Serializable.

Answer: B, D, E

Q: 06 Assuming that the serializeBanana2() and the deserializeBanana2() methods will correctly use Java serialization and given:

```

13. import java.io.*;
14. class Food {Food() { System.out.print("1"); } }
15. class Fruit extends Food implements Serializable {
16. Fruit() { System.out.print("2"); } }
17. public class Banana2 extends Fruit { int size = 42;
18. public static void main(String [] args) {
19. Banana2 b = new Banana2();
20. b.serializeBanana2(b); // assume correct serialization
21. b = b.deserializeBanana2(b); // assume correct
22. System.out.println(" restored " + b.size + " "); }
23. // more Banana2 methods
24. }

```

What is the result?

- A. Compilation fails.
- B. 1 restored 42
- C. 12 restored 42
- D. 121 restored 42
- E. 1212 restored 42
- F. An exception is thrown at runtime.

Answer: D

Q: 7 When comparing java.io.BufferedWriter to java.io.FileWriter, which capability exists as a method in only one of the two?

- A. closing the stream
- B. flushing the stream
- C. writing to the stream
- D. marking a location in the stream
- E. writing a line separator to the stream

Answer: E

Question: 8

Given:

```
10. class MakeFile {  
11. public static void main(String[] args) {  
12. try {  
13. File directory = new File("d");  
14. File file = new File(directory,"f");  
15. if(!file.exists()) {  
16. file.createNewFile();  
17. }  
18. } catch (IOException e) {  
19. e.printStackTrace  
20. }  
21. }  
22. }
```

The current directory does NOT contain a directory named "d."

Which three are true? (Choose three.)

- A. Line 16 is never executed.
- B. An exception is thrown at runtime.
- C. Line 13 creates a File object named "d."
- D. Line 14 creates a File object named "f."
- E. Line 13 creates a directory named "d" in the file system.
- F. Line 16 creates a directory named "d" and a file 'f' within it in the file system.
- G. Line 14 creates a file named 'f' inside of the directory named "d" in the file system.

Answer: BCD

Q: 09 Click the Task button.

The `doesFileExist` method takes an array of directory names representing a path from the root filesystem and a file name. The method returns true if the file exists, false if it does not.

Place the code fragments in position to complete this method.

```
public static boolean doesFileExist(String[] directories, String filename) {
```

Place here

```
    for ( String dir : directories ) {
```

Place here

```
    }
```

Place here

Place here

```
}
```

Code Fragments

`path = path.getSubdirectory(dir);`

`return ! file.isNew();`

`return (file != null);`

`String path = "";`

`path = path.getFile(filename);`

`File path = new File("");`

`return file.exists();`

`return path.isFile();`

`File file = new File(path, filename);`

`path = new File(path, dir);`

`File path = new File(File.separator);`

`path = path + File.separator + dir;`

Solution:

1. `String path=""`;
2. `path=path+File.separator+dir;`
3. `File file=new File(path,filename);`
4. `return file.exists();`

Q:10 Click the Exhibit button.

Which code, inserted at line 14, will allow this class to correctly serialize and deserialize?

```

1. import java.io.*;
2. public class Foo implements Serializable
{
3.     public int x, y;
4.     public Foo( int x, int y ) { this.x =
x; this.y = y; }
5.
6.     private void writeObject(
ObjectOutputStream s )
7.         throws IOException {
8.         s.writeInt(x); s.writeInt(y) ;
9.     }
10.
11.    private void readObject(
ObjectInputStream s )
12.        throws IOException,
ClassNotFoundException {
13.
14.        // insert code here
15.
16.    }
17. }

```

- A. s.defaultReadObject();
- B. this = s.defaultReadObject();
- C. y = s.readInt(); x = s.readInt();
- D. x = s.readInt(); y = s.readInt();

Answer: D

Question: 11

Given:

```

10. public class Foo implements java.io.Serializable {
11. private int x;
12. public int getX() { return x; }
12.publicFoo(int x){this.x=x; }
13. private void writeObject( ObjectOutputStream s)
14. throws IOException {
15. // insert code here
16. }
17. }

```

Which code fragment, inserted at line 15, will allow Foo objects to be correctly serialized and deserialized?

- A. s.writeInt(x);
- B. s.serialize(x);
- C. s.writeObject(x);
- D. s.defaultWriteObject();

Answer: D

12 Click the Task button.

Place the Fragments into the program, so that the program will get lines from a text file, display them, and then close all the resources.

Program

```
import java.io.*

public class ReadFile {
    public static void main(String [] args) {
        try {
            File ? = new File("MyText.txt");
            ? = new ? (x1);
            ? x4 = new ? (x2);
            String x3 = null;
            while (( x3 = ? . ? ()) != null) {
                System.out.println(x3);
            } ? . ? ();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Code Fragments

BufferedReader
StreamReader
FileReader
readLine
readLn
read
closeFile
close
x1 x2
x3 x4

Done

Solution:

```
import java.io.*;

public class ReadFile{
    public static void main(String s[] ){
        try {
            File x1=new File("MyText.txt");
            FileReader x2=new FileReader(x1);
            BufferedReader x4=new BufferedReader(x2);
            String s3=null;
            while((x3 = x4.readLine( )) != null ) {
                System.out.println(x3);
            }x4.close( );
        }
    }
}
```

```

        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

```

13. Given:

```

import java.io.*;
class Player {
    Player() { System.out.print("p"); }
}
class CardPlayer extends Player implements Serializable {
    CardPlayer() { System.out.print("c"); }
    public static void main(String[] args) {
        CardPlayer c1 = new CardPlayer();
        try {
            FileOutputStream fos = new FileOutputStream("play.txt");
            ObjectOutputStream os = new ObjectOutputStream(fos);
            os.writeObject(c1);
            os.close();
            FileInputStream fis = new FileInputStream("play.txt");
            ObjectInputStream is = new ObjectInputStream(fis);
            CardPlayer c2 = (CardPlayer) is.readObject();
            is.close();
        } catch (Exception x) { }
    }
}

```

What is the result?

- A. pc
- B. pcc
- C. pcpc
- D. pcpc
- E. Compilation fails.
- F. An exception is thrown at runtime.

Answer:

-> **C** is correct. It's okay for a class to implement Serializable even if its superclass doesn't. However, when you deserialize such an object, the non-serializable superclass must run its constructor. Remember, constructors don't run on deserialized classes that implement Serializable.

-> **A, B, D, E,** and **F** are incorrect based on the above.

14. Given:

bw is a reference to a valid **BufferedWriter** And the snippet:

```

15. BufferedWriter b1 = new BufferedWriter(new File("f"));
16. BufferedWriter b2 = new BufferedWriter(new FileWriter("f1"));
17. BufferedWriter b3 = new BufferedWriter(new PrintWriter("f2"));
18. BufferedWriter b4 = new BufferedWriter(new BufferedWriter(bw));

```


What is the result?

- A. Compilation succeeds.
- B. Compilation fails due only to an error on line 15.
- C. Compilation fails due only to an error on line 16.
- D. Compilation fails due only to an error on line 17.
- E. Compilation fails due only to an error on line 18.
- F. Compilation fails due to errors on multiple lines.

Answer:

-> **B** is correct. BufferedWriters can be constructed only by wrapping a Writer. Lines 16, 17, and 18 are correct because BufferedWriter, FileWriter, and PrintWriter all extend Writer. (Note: BufferedWriter is a decorator class. Decorator classes are used extensively in the java.io package to allow you to extend the functionality of other classes.)

-> **A, C, D, E,** and **F** are incorrect based on the above. (Objective 3.2)

15. Given:

```
import java.io.*;
class Keyboard { }
public class Computer implements Serializable {
    private Keyboard k = new Keyboard();
    public static void main(String[] args) {
        Computer c = new Computer();
        c.storeIt(c);
    }
    void storeIt(Computer c) {
        try {
            ObjectOutputStream os = new ObjectOutputStream(
                new FileOutputStream("myFile"));
            os.writeObject(c);
            os.close();
            System.out.println("done");
        } catch (Exception x) {System.out.println("exc"); }
    }
}
```

What is the result? (Choose all that apply.)

- A. exc
- B. done
- C. Compilation fails.
- D. Exactly one object is serialized.
- E. Exactly two objects are serialized.

Answer:

-> **A** is correct. An instance of type Computer Has-a Keyboard. Because Keyboard doesn't implement Serializable, any attempt to serialize an instance of Computer will cause an exception to be thrown.

-> **B, C, D,** and **E** are incorrect based on the above. If Keyboard did implement Serializable then

two objects would have been serialized.

16. Given:

```
import java.io.*;
class Directories {
    static String [] dirs = {"dir1", "dir2"};
    public static void main(String [] args) {
        for (String d : dirs) {
            // insert code 1 here
            File file = new File(path, args[0]);
            // insert code 2 here
        }
    }
}
```

and that the invocation

`java Directories file2.txt`

is issued from a directory that has two subdirectories, "dir1" and "dir1", and that "dir1" has a

file "file1.txt" and "dir2" has a file "file2.txt", and the output is "false true", which set(s) of code fragments must be inserted? (Choose all that apply.)

- A. `String path = d;`
`System.out.print(file.exists() + " ");`
- B. `String path = d;`
`System.out.print(file.isFile() + " ");`
- C. `String path = File.separator + d;`
`System.out.print(file.exists() + " ");`
- D. `String path = File.separator + d;`
`System.out.print(file.isFile() + " ");`

Answer:

-> **A** and **B** are correct. Because you are invoking the program from the directory whose direct subdirectories are to be searched, you don't start your path with a `File.separator` character. The `exists()` method tests for either files or directories; the `isFile()` method tests only for files. Since we're looking for a file, both methods work.

-> **C** and **D** are incorrect based on the above

17. Given:

```
import java.io.*;
public class TestSer {
    public static void main(String[] args) {
        SpecialSerial s = new SpecialSerial();
        try {
            ObjectOutputStream os = new ObjectOutputStream(
                new FileOutputStream("myFile"));
            os.writeObject(s); os.close();
        }
    }
}
```

```

System.out.print(++s.z + " ");
ObjectInputStream is = new ObjectInputStream(
new FileInputStream("myFile"));
SpecialSerial s2 = (SpecialSerial)is.readObject();
is.close();
System.out.println(s2.y + " " + s2.z);
} catch (Exception x) {System.out.println("exc"); }
}
}
class SpecialSerial implements Serializable {
transient int y = 7;
static int z = 9;
}

```

Which are true? (Choose all that apply.)

- A. Compilation fails.
- B. The output is 10 0 9
- C. The output is 10 0 10
- D. The output is 10 7 9
- E. The output is 10 7 10
- F. In order to alter the standard deserialization process you would override the readObject() method in SpecialSerial.
- G. In order to alter the standard deserialization process you would override the defaultReadObject() method in SpecialSerial.

Answer:

-> **C** and **F** are correct. **C** is correct because static and transient variables are not serialized when an object is serialized. **F** is a valid statement.

-> **A**, **B**, **D**, and **E** are incorrect based on the above. **G** is incorrect because you don't override the defaultReadObject() method, you call it from within the overridden readObject() method, along with any custom read operations your class needs.