Exercises and Solutions – Advanced Java Programming

- Exceptions and Assertions
- Input and Output in Java
- Strings, Formatters and Wrappers
- Collections and Generics
- Threads in Java

<u>**Exceptions and Assertions Review Questions :**</u>

1. Consider the following code fragment:

```
1. public class ExceptionHandleNot2{
2. public static void main(String[] args) {
3. String[] week_days = {"Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"};
4. for (int i=1; i<=7; i++) {
5. System.out.println("The day of the week: " + week_days[i]);
6. }
7. }
8. }
```

What is the output of this code?

A. Line 5 prints out the names of all the days of the week.
B. A compiler error occurs.
C. A runtime error occurs and nothing is printed out.
D. A runtime error occurs after printing out some days of the week.

2. Which one of the following code fragments presents the most appropriate way of throwing exceptions?

Assume that the variable i is already properly defined elsewhere, is in scope,
and has an appropriate value.

A.
```
if ( i > 10) {
throws new IndexOutOfBoundsException("Index is out of bound!");
}
```
B.
```
if ( i > 10) {
throw new IndexOutOfBoundsException("The value of index i=" +
i + " is out of bound!" );
}
```

C. IndexOutOfBoundsException iob = new IndexOutOfBoundsException
("Index out of bound!");
if (i > 10) {
throw iob;
}

D. if (i > 10) {
throw "Index is out of bound!";
}

3. Consider the following code fragment:

```
public class ExceptionHandleTest{
public static void main(String[] args) {
int x = 15;
int y = 1;
try{
System.out.println ("x/y: " + x/y);
System.out.println("x*y: " + x*y);
} catch (ArithmeticException ae) {
System.out.println("An exception occurred: " + ae);
}
catch (ArrayIndexOutOfBoundsException oe) {
System.out.println("An exception occurred: " + oe);
}
finally {
System.out.println("finally block must be executed!");
}
System.out.println("x-y: " + (x-y));
}
}
```

Which of the following lines would be part of the output? (Choose all that apply.)

A. x/y: 15
B. x*y: 15
C. finally block must be executed!
D. x-y: 14

4. Consider the following code fragment:

```
public class Question4{
public static void main(String[] args) {
int[] x = {0, 1, 2, 3, 4};
try{
System.out.println ("x[6]: " + x[6]);
System.out.println("x[3]: " + x[3]);
} catch (IndexOutOfBoundsException ie) {
System.out.println("Some kind of index out of bound! ");
}
catch (ArrayIndexOutOfBoundsException oe) {
System.out.println("Array index out of bound! " );
}
finally {
System.out.println("finally block must be executed!");
}
System.out.println("x[0]: " + x[0]);
}
}
```

What is the output of this code?

A. Array index out of bound!
finally block must be executed!

B. Some kind of index out of bound!
finally block must be executed!

C. Some kind of index out of bound!
Array index out of bound!
finally block must be executed!

D. A compiler error occurs.

5. Consider the following code fragment:

```
public class AssertionExample2 {
public static void main(String[] args) {
System.out.println(args.length);
assert args.length != 0;
}
}
```

Which of the following must be done in order for the code to throw an AssertionError? (Choose all that apply.)

A. The code must be compiled with the -source 1.4 option if you are using JDK 5.0.
B. The program must be executed with the -ea option.
C. At least one argument must be given in the execution command.
D. No argument should be given in the execution command.

6. Which of the following is true about assertions in Java? (Choose all that apply.)

A. You use assertions to report errors to the users of an application.

B. Assertions are mostly used during testing to uncover internal program errors.

C. Assertions are used to report recoverable problems from one part of an application to another part.

D. An assertion error is thrown if the condition specified by <condition> in assert: <condition> is true.

E. An assertion error is thrown if the condition specified by <condition> in assert: <condition> is false.

7. Consider the following code fragment:

```
1. public class TryFinallyTest{
2. public static void main(String[] args) {
3. try{
4. System.out.println ("I was in try");
5. }
6. finally {
7. System.out.println("I was in finally");
8. }
9. }
10. }
```

What is the result of executing this code?

A. I was in try
I was in finally
B. I was in try
C. I was in finally
D. A compiler error occurs at line 6.
E. The program compiles but throws an exception during execution.

8. Consider the following code fragment:

```
1. public class TryFinallyTest{
2. public static void main(String[] args) {
3. try{
4. System.out.println ("I was in try");
5. }
9. }
10. }
```

What is the result of executing this code?
A. I was in try
B. A compiler error occurs.
C. The program runs but produces no output.
D. The program compiles but throws an exception during execution.

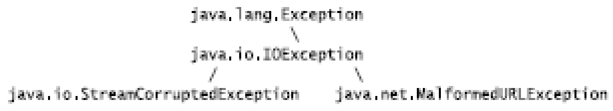
9. Consider the following code:

```
1. class CodeWalkSix {
2. public static void main(String [] args) {
3. String stri = "inner";
4. String stro = "outer";
5. try {
6. throw new Exception();
7. } catch (Exception eo) {
8. try {
9. throw new Exception();
10. } catch (Exception ei) {
11. System.out.print(stri);
12. } finally {
13. System.out.print(" finally ");
14. }
15. System.exit(1);
16. }finally {
17. System.out.print(stro);
18. }
19. System.out.print(stro);
20. }
21. }
```

What is the result?

A. inner finally
B. inner finally outer outer
C. inner finally outer
D. Compilation fails.
E. No output.

10. Consider the following class hierarchy and code fragment:



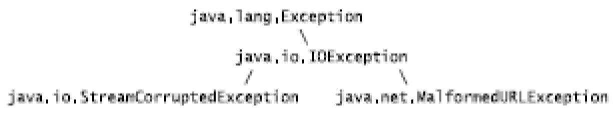
```
                  java.lang.Exception
                           \
                  java.io.IOException
                    /              \
java.io.StreamCorruptedException    java.net.MalformedURLException
```

```
1. try {
2. // assume s is previously defined
3. URL u = new URL(s);
4. // in is an ObjectInputStream
5. Object o = in.readObject();
6. System.out.println("Success");
7. }
8. catch (MalformedURLException e) {
9. System.out.println("Bad URL");
10. }
11. catch (StreamCorruptedException e) {
12. System.out.println("Bad file contents");
13. }
14. catch (Exception e) {
15. System.out.println("General exception");
16. }
17. finally {
18. System.out.println("Doing finally part");
19. }
20. System.out.println("Carrying on");
```

What lines are output if the constructor at line 3 throws a MalformedURLException? (Choose all that apply.)

A. Success
B. Bad URL
C. Bad file contents
D. General exception
E. Doing finally part
F. Carrying on

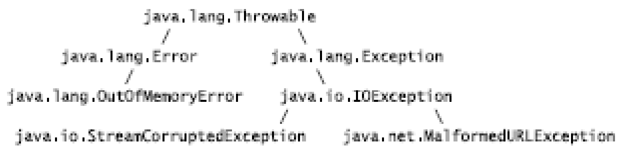
11. Consider the following class hierarchy and code fragment:


java.lang.Exception
java.io.IOException
java.io.StreamCorruptedException
java.net.MalformedURLException

```
1. try {
2. // assume s is previously defined
3. URL u = new URL(s);
4. // in is an ObjectInputStream
5. Object o = in.readObject();
6. System.out.println("Success");
7. }
8. catch (MalformedURLException e) {
9. System.out.println("Bad URL");
10. }
11. catch (StreamCorruptedException e) {
12. System.out.println("Bad file contents");
13. }
14. catch (Exception e) {
15. System.out.println("General exception");
16. }
17. finally {
18. System.out.println("Doing finally part");
19. }
20. System.out.println("Carrying on");
```

What lines are output if the methods at lines 3 and 5 complete successfully without throwing any exceptions? (Choose all that apply.)
A. Success
B. Bad URL
C. Bad file contents
D. General exception
E. Doing finally part
F. Carrying on

12. Consider the following class hierarchy and code fragment:


java.lang.Throwable
java.lang.Error
java.lang.Exception
java.lang.OutOfMemoryError
java.io.IOException
java.io.StreamCorruptedException
java.net.MalformedURLException

```
1. try {
2. // assume s is previously defined
3. URL u = new URL(s);
4. // in is an ObjectInputStream
5. Object o = in.readObject();
6. System.out.println("Success");
7. }
8. catch (MalformedURLException e) {
9. System.out.println("Bad URL");
10. }
11. catch (StreamCorruptedException e) {
12. System.out.println("Bad file contents");
13. }
14. catch (Exception e) {
15. System.out.println("General exception");
16. }
17. finally {
18. System.out.println("Doing finally part");
19. }
20. System.out.println("Carrying on");
```

What lines are output if the method at line 5 throws an OutOfMemoryError? (Choose all that apply.)

A. Success
B. Bad URL
C. Bad file contents
D. General exception
E. Doing finally part
F. Carrying on

13. Which of the following are appropriate situations for assertions?

A. Preconditions of a public method
B. Postconditions of a public method
C. Preconditions of a private method
D. Postconditions of a private method

14. Consider the following code:

```
public class Assertification {
public static void main(String[] args) {
assert args.length == 0;
}
}
```

Which of the following conditions must be true in order for the code to throw an AssertionError? Assume you are using release 5.0. (Choose all that apply.)

A. The source code must be compiled with the -source 1.5 flag.
B. The application must be run with the -enableassertions flag or another assertionenabling flag.
C. The args array must have exactly zero elements.
D. The args array must have one or more elements.

15. Which of the following is the most appropriate way to handle invalid arguments in a public method?

A. Throw java.lang.InvalidArgumentException.
B. Throw java.lang.IllegalArgumentException.
C. Check for argument validity in an assert statement, which throws AssertionError when the arguments are invalid.
D. Use non-assert code to check for argument validity. If invalid arguments are detected, explicitly throw AssertionError.

16. Suppose a method called finallyTest() consists of a try block, followed by a catch block, followed by a finally block. Assuming the JVM doesn't crash and the code does not execute a System.exit() call, under what circumstances will the finally block *not* begin to execute?

A. The try block throws an exception, and the catch block also throws an exception.
B. The try block throws an exception that is not handled by the catch block.
C. The try block throws an exception, and the catch block calls finallyTest() in a way that causes another exception to be thrown.
D. If the JVM doesn't crash and the code does not execute a System.exit() call, the finally block will always execute.

17. When is it appropriate to pass a cause to an exception's constructor?

A. Always
B. When the exception is being thrown in response to catching of a different exception type
C. When the exception is being thrown from a public method
D. When the exception is being thrown from a private method
18. Which of the following should always be caught?

A. Runtime exceptions
B. Checked exceptions
C. Assertion errors
D. Errors other than assertion errors

19. When does an exception's stack trace get recorded in the exception object?

A. When the exception is constructed
B. When the exception is thrown
C. When the exception is caught
D. When the exception's printStackTrace() method is called

20. When is it appropriate to write code that constructs and throws an error?

A. When a public method's preconditions are violated
B. When a public method's postconditions are violated
C. When a nonpublic method's preconditions are violated
D. When a nonpublic method's postconditions are violated
E. Never

Exceptions and Assertions Review Questions Solutions:

1. Answer: D
Line 5 will generate an ArrayIndexOutofBoundsException, which is an unchecked exception. So, it will be thrown only at runtime when i=7.

2. Answer: B
B has the correct syntax, and you can use the keywords throw and new in the same statement. Note that the keyword to be used here is throw and not throws.

3. Answer: A, B, C, and D
The finally block is always executed except under special circumstances.

4. Answer: D
The following rule is violated: the more specific catch block should precede the less specific catch block.

5. Answer: B and D
The support for assertions is enabled by default when compiling code in J2SE 5.0. The assert statement asserts that the number of command arguments is not equal to 0. Therefore, an AssertionError will be thrown if the number of arguments is 0.

6. Answer: B and E
A, C, and D are false statements about assertions.

7. Answer: A
The finally block can exist with the catch block, and the finally block is always executed instead under extreme circumstances such as a return or an exit statement.

8. Answer: B
The try block must be followed by either a catch block or a finally block.

9. Answer: A
The code is just fine. The execution will stop at the System.exit(1) statement.

10. B, E, F. The exception causes a jump out of the try block, so the message Success from line 6 is not printed. The first applicable catch is at line 8, which is an exact match for the thrown exception. This results in the message at line 9 being printed, so B is one of the required answers. Only one catch block is ever executed, so control passes to the finally block, which results in the message at line 18 being output; so E is part of the correct answer. Execution continues after the finally block. This results in the output of the message at line 20, so F is also part of the correct answer.

11. A, E, F. With no exceptions, the try block executes to completion, so the message Success from line 6 is printed and A is part of the correct answer. No catch is executed, so B, C, and D are incorrect. Control then passes to the finally block, which results in the message at line 18 being output, so E is part of the correct answer. Because no exception was thrown, execution continues after the finally block, resulting in the output of the message at line 20; so F is also part of the correct answer.

12. E. The thrown error prevents completion of the try block, so the message Success from line 6 is not printed. No catch is appropriate, so B, C, and D are incorrect. Control then passes to the finally block, which results in the message at line 18 being output; so option E is part of the correct answer. Because the error was not caught, execution exits the method and the error is rethrown in the caller of this method; so F is not part of the correct answer.

13. B, C, D. Assertions should not be used to check preconditions of a public method.

14. B, D. The 1.4 compiler only treated assert as a keyword (and not an ordinary identifier) if the -source 1.4 flag appeared in the command line. However, 5.0 does not require a -source flag. So A is not a requirement. If the application is not run with assertions explicitly enabled, all assert statements will be ignored. If the args array does not have exactly zero arguments, no AssertionError will be thrown.

15. B. Assertions should not be used to check preconditions in a public method. Some kind of runtime exception should be thrown, and IllegalArgumentException is the most appropriate class name for this situation. There is no such thing as java.lang.InvalidArgumentException.

16. D. Unless execution terminates abnormally, the finally block will always eventually execute.

17. B. Exception chaining is appropriate when an exception is being thrown in response to catching a different exception type.

18. B. Runtime exceptions don’t have to be caught. Errors should never be caught.

19. A. The stack trace is recorded when the exception is constructed.

20. E. It is never appropriate for application programmers to construct and throw errors.

Exceptions and Assertions Coding Questions:

Checked Exceptions

- Consider the code below
- Write code that will allow this code to compile (hint: surround the code with a try catch block to handle the checked exception produced). You will also need a main method
- Compile and run the code

```
void callingMethod() {
calledMethod();
}
void calledMethod() throws IOexception {
throw new IOException();
}
```

Runtime Exceptions

- Consider the code below
- Write code to handle an arithmetic exception in the case of a divide by zero operation
- Compile and run the code

```
public class ExceptionHandle1{
public static void main(String[] args) {
int x = 15;
int y = 0;

System.out.println ("x/y: " + x/y);
System.out.println("x*y: " + x*y);


System.out.println("x-y: " + (x-y));
}
}
```

Catch-Try-Finally

- Consider the code below
  - Compile and run the program
  - Note the exection flow of control
- Add a finally block
  - Within the finally block print out the following string ”finally block must be exectued”
  - Compile and run the program
  - Note the execution flow of control
- Modify the line “int y = 0;” to “int y = 1;”
  - Compile and run the program

- Note the exection flow of control

```
public class ExceptionHandle2{
public static void main(String[] args) {
int x = 15;
int y = 0;
try{
System.out.println ("x/y: " + x/y);
System.out.println("x*y: " + x*y);
} catch (ArrayIndexOutOfBoundsException oe) {
System.out.println("An exception occurred: " + oe);
}

System.out.println("x-y: " + (x-y));
}
}
```

Solution

```
public class ExceptionHandle2{
public static void main(String[] args) {
int x = 15;
int y = 0;
try{
System.out.println ("x/y: " + x/y);
System.out.println("x*y: " + x*y);
} catch (ArrayIndexOutOfBoundsException oe) {
System.out.println("An exception occurred: " + oe);
}
finally {
System.out.println("finally block must be executed!");
}
System.out.println("x-y: " + (x-y));
}
}
```

Multiple Catch Blocks

- Compile, run and understand the following code
- What line needs to be modified to prevent a runtime exception

```
1. public class MultipleExceptions{
2. public static void main(String[] args) {
3. int[] x = {0, 1, 2, 3, 4};
4. try{
5. System.out.println ("x[6]: " + x[6]);
6. System.out.println("x[3]: " + x[3]);
7. }
8. catch (ArithmeticException ae) {
9. System.out.println("An arithmetic exception occurred: " + ae);
10. }
11. catch (ArrayIndexOutOfBoundsException oe) {
12. System.out.println("Array index out of bound! ");
13. }
14. catch (IndexOutOfBoundsException ie) {
15. System.out.println("Some kind of index out of bound! ");
16. }
17. finally {
18. System.out.println("finally block must be executed!");
19. }
20. System.out.println("x[0]: " + x[0]);
21. }
22. }
```

Throwing Exceptions

- Compile, run, and understand the following code
- Modify the code to throw a more general run-time exception with the following message: “This is a more general exception”
- Compile and run the code.

```
1. public class ThrowExample{
2. public static void main(String[] args) {
3. double x = 15.0;
4. double y = 0.0;
5. try{
6. ThrowExample te = new ThrowExample();
7. double z = te.doubleDivide(x, y);
8. System.out.println ("x/y: " + x/y);
9. }
10. catch (ArithmeticException ae) {
11. System.out.println("An exception occurred: " + ae);
12. }
System.out.println("x-y: " + (x-y));
}
double doubleDivide(double x, double y) {
if(y==0.0) {
throw new ArithmeticException("Integer or not, please do not divide by zero!");
} else {
return x/y;
}
}
}
```

Declaring exceptions

- Modify the code below such that it compiles
    - Hint you will need to declare an exception in a method definition
- Run the program

```
import java.io.*;

class Declare{

public static void main(String [] args){

Declare d = new Declare();

try{

d.callingMethod();

}

catch (Exception e)

{System.out.println(e);}

}

 void callingMethod() {

 calledMethod();

 }

 void calledMethod() throws IOException {

 throw new IOException();

 }

}
```

Assertions

- Compile and run the code below with assertions enabled
- Run the code with assertions disabled
- Modify the code below such that no assertion error is thrown when the application runs with assertions enabled
- Compile and run the applications with assertions enabled

```
public class AssertionExample{
public static void main(String[] args) {
int x = 15;
DataAccess da = new DataAccess();
assert da.dataIsAccesible():"Data is not acceptable";
System.out.println("Value of x: " + x);
}
}
class DataAccess {
public boolean dataIsAccesible(){
return false;
}
}
```

Input and Output Review Questions:

NOTE questions 21 -25 were not covered in lectures – these will not be on our Exam but may be asked in SCJP – further reading is required to answer these.

1. While navigating the file system by using the methods of the File class, which of the following operations can you perform?

A. Change the current working directory.
B. Delete a file.
C. Create a file.
D. Change the security on the file.

2. When you instantiate the File class, a file is created on the file system.

A. True
B. False

3. Which of the following streams will enable you to read the primitive data types from a file into your Java program?

A. DataInputStream and FileInputStream
B. DataInputStream
C. FileInputStream

D. FileReader

4. Which of the following statements are true? (Choose all that apply.)

A. Readers and writers are used for I/O on 16-bit Unicode characters.
B. FileInputStream and FileOutputStream can be used to handle I/O on 16-bit Unicode characters.
C. FileInputStream and FileOutputStream can be used to read image files.
D. FileInputStream and FileOutputStream can be used to read text files.

5. Which of the following streams enables you to write an object to a file?

A. ObjectOutputStream and FileOutputStream
B. FileOutputStream
C. ObjectOutputStream
D. ObjectInputStream

6. Consider the following code:

```
1. FileOutputStream out=new FileOutputStream("objectStore.ser");
2. ObjectOutputStream os = new ObjectOutputStream(out);
3. os.writeObject("Object on the fly!");
```

What is the result of this code?

A. The string "Object on the fly!" is written into the file objectStore.ser.
B. Line 3 causes a compiler error.
C. The code compiles fine but an exception is thrown at line 3 when the code is executed.
D. An exception occurs at line 2.
E. A compiler error occurs at line 2.

7. Consider the following code:

```
FileOutputStream fo = FileOutputStream("myFile.txt");
DataOutputStream do = DataOutputStream(fo);
do.writeByte(9);
do.writeFloat(4.20f);
do.close();
```

How many 8-bit bytes does the code write into the file?

A. 2
B. 5
C. 3
D. 10

8. Consider the following code:

```
1. import java.io.*;
2. public class QuestionEight {
3. public static void main(String[] args) throws IOException {
4. File inputFile = new File("scjp.txt");
5. File outputFile = new File("scjpcopy.txt");
6. BufferedReader in = new BufferedReader(inputFile);
7. BufferedWriter out = new BufferedWriter(new FileWriter(outputFile));
8. String line;
9. while ((line = in.readLine()) != null){
10. out.write(line);
11. out.newLine();
12. }
13. in.close();
14. out.close();
15. }
16.}
```

What is the output of this code?

A. A compiler error occurs at line 3.
B. A compiler error occurs at line 6.
C. The code compiles fine but throws an exception during execution at line 6.
D. A compiler error occurs at line 7.
E. The code compiles and executes without any error or exception.

9. Which of the following statements are true? (Choose all that apply.)

A. When an object is serialized, the whole class definition is saved.
B. When an object is serialized, the whole object state (all the data variables with their values) are saved.
C. FileInputStream cannot read text files; you can only use it to read image files.
D. You cannot read a file directly from FileReader.
E. None of the above.

10. Consider the code below. What is the result?

```
import java.io.*;
class CodeWalkSeven{
public static void main(String [] args) {
Car c = new Car("Nissan", 1500, "blue");
System.out.println("before: " + c.make + " " + c.weight);
try {
     FileOutputStream fs = new FileOutputStream("Car.ser");
     ObjectOutputStream os = new ObjectOutputStream(fs);
     os.writeObject(c);
     os.close();
}catch (Exception e) { e.printStackTrace(); }
try {
     FileInputStream fis = new FileInputStream("Car.ser");
     ObjectInputStream ois = new ObjectInputStream(fis);
     c = (Car) ois.readObject();
     ois.close();
}catch (Exception e) { e.printStackTrace(); }
     System.out.println("after: " + c.make + " "+ c.weight);
}
}
class NonLiving {}
class Vehicle extends NonLiving {
    String make = "Lexus";
    String color = "Brown";
}
class Car extends Vehicle implements Serializable {
  protected int weight = 1000;
Car(String n, int w, String c) {
    color = c;
    make = n;
    weight = w;
  }
}
```

A. before: Nissan 1500 after: Lexus 1000
B. before: Nissan 1500 after: Lexus 1500
C. before: Nissan: 1000 after: Lexus 1000
D. before: Nissan: 1000 after: Lexus 1500
E. Compilation fails.

11. Which of the statements below are true? (Choose all that apply.)

A. UTF characters are all 8 bits.
B. UTF characters are all 16 bits.
C. UTF characters are all 24 bits.
D. Unicode characters are all 16 bits.
E. Bytecode characters are all 16 bits.
F. None of the above.

12. Which of the statements below are true? (Choose all that apply.)

A. When you construct an instance of File, if you do not use the file-naming semantics of the local machine, the constructor will throw an IOException.
B. When you construct an instance of File, if the corresponding file does not exist on the local file system, one will be created.
C. When an instance of File is garbage collected, the corresponding file on the local file system is deleted.
D. None of the above.

13. Which of the statements below are true? (Choose all that apply.)

A. To change the current working directory, call the setWorkingDirectory() method of the File class.
B. To change the current working directory, call the cd() method of the File class.
C. To change the current working directory, call the changeWorkingDirectory() method of the File class.
D. None of the above.

14. How do you use the File class to list the contents of a directory?

A. String[] contents = myFile.list();
B. File[] contents = myFile.list();
C. StringBuilder[] contents = myFile.list();
D. The File class does not provide a way to list the contents of a directory.

15. How many bytes does the following code write to file dest?

```
try {
FileOutputStream fos = newFileOutputStream(“dest”);
DataOutputStream dos = new DataOutputStream(fos);
dos.writeInt(3);
dos.writeDouble(0.0001);
dos.close();
fos.close();
}
catch (IOException e) { }
```

A. 2
B. 8
C. 12
D. 16
E. The number of bytes depends on the underlying system.

16. What does the following code fragment print out at line 9?

```
1. FileOutputStream fos = new FileOutputStream(“xx”);
2. for (byte b=10; b<50; b++)
3. fos.write(b);
4. fos.close();
5. RandomAccessFile raf = new RandomAccessFile(“xx”, “r”);
6. raf.seek(10);
7. int i = raf.read();
8. raf.close();
9. System.out.println(“i = “ + i);
```

A. The output is i = 30.
B. The output is i = 20.
C. The output is i = 10.
D. There is no output because the code throws an exception at line 1.
E. There is no output because the code throws an exception at line 5.

17. A file is created with the following code:

```
1. FileOutputStream fos = new FileOutputStream(“datafile”);
2. DataOutputStream dos = new DataOutputStream(fos);
3. for (int i=0; i<500; i++)
4. dos.writeInt(i);
```

You would like to write code to read back the data from this file. Which solutions will work? (Choose all that apply.)

A. Construct a FileInputStream, passing the name of the file. Onto the FileInputStream, chain a DataInputStream, and call its readInt() method.
B. Construct a FileReader, passing the name of the file. Call the file reader’s readInt() method.
C. Construct a PipedInputStream, passing the name of the file. Call the piped input stream’s readInt() method.
D. Construct a RandomAccessFile, passing the name of the file. Call the random access file’s readInt() method.
E. Construct a FileReader, passing the name of the file. Onto the FileReader, chain a DataInputStream, and call its readInt() method.

18. Which of the following is true?

A. Readers have methods that can read and return floats and doubles.
B. Readers have methods that can read and return floats.
C. Readers have methods that can read and return doubles.
D. Readers have methods that can read and return ints.
E. None of the above.

19. You execute the following code in an empty directory. What is the result?

```
File f1 = new File(“dirname”);
File f2 = new File(f1, “filename”);
```

A. A new directory called dirname is created in the current working directory.
B. A new directory called dirname is created in the current working directory. A new file called filename is created in directory dirname.
C. A new directory called dirname and a new file called filename are created, both in the current working directory.
D. A new file called filename is created in the current working directory.
E. No directory is created, and no file is created.

20. What is the result of attempting to compile and execute the following code fragment? Assume that the code fragment is part of an application that has write permission in the current working directory. Also assume that before execution, the current working directory does not contain a file called datafile.

```
1. try {
2. RandomAccessFile raf = new
3. RandomAccessFile(“datafile” ,”rw”);
4. BufferedOutputStream bos = new
5. BufferedOutputStream(raf);
6. DataOutputStream dos = new
7. DataOutputStream(bos);
8. dos.writeDouble(Math.PI);
9. dos.close();
10. bos.close();
11. raf.close();
12. }
13. catch (IOException e) { }
```

A. The code fails to compile.
B. The code compiles but throws an exception at line 4.
C. The code compiles and executes but has no effect on the local file system.
D. The code compiles and executes; afterward, the current working directory contains a file called datafile.

21. Suppose you are writing a class that will provide custom serialization. The class implements java.io.Serializable (not java.io.Externalizable). What access mode should the writeObject() method have?

A. public
B. protected
C. default
D. private

22. Suppose you are writing a class that will provide custom deserialization. The class implements java.io.Serializable (not java.io.Externalizable). What access mode should the readObject() method have?

A. public
B. protected
C. default
D. private

23. Suppose class A extends Object; class B extends A; and class C extends B. Of these, only class C implements java.io.Serializable. Which of the following must be true in order to avoid an exception during deserialization of an instance of C?

A. A must have a no-args constructor.
B. B must have a no-args constructor.
C. C must have a no-args constructor.
D. There are no restrictions regarding no-args constructors.

24. Suppose class A extends Object; Class B extends A; and class C extends B. Of these, only class C implements java.io.Externalizable. Which of the following must be true in order to avoid an exception during deserialization of an instance of C?

A. A must have a no-args constructor.
B. B must have a no-args constructor.
C. C must have a no-args constructor.
D. There are no restrictions regarding no-args constructors.

25. Given the following class:

```
public class Xyz implements java.io.Serializable {
public int iAmPublic;
private int iAmPrivate;
static int iAmStatic;
transient int iAmTransient;
volatile int iAmVolatile;
. . .
}
```

Assuming the class does not perform custom serialization, which fields are written when an instance of Xyz is serialized? (Choose all that apply.)

A. iAmPublic
B. iAmPrivate
C. iAmStatic
D. iAmTransient
E. iAmVolatile

26. What method of the java.io.File class can create a file on the hard drive?
A. newFile()
B. makeFile()
C. makeNewFile()
D. createFile()
E. createNewFile()

27. Which of the following are true? (Choose all that apply.)

A. System.out has a println() method.
B. System.out has a format() method.
C. System.err has a println() method.
D. System.err has a format () method.
28. What happens when you try to compile and run the following application?

```
1. import java.io.*;
2.
3. public class Xxx {
4. public static void main(String[] args) {
5. try {
6. File f = new File("xxx.ser");
7. FileOutputStream fos = new FileOutputStream(f);
8. ObjectOutputStream oos = new ObjectOutputStream(fos);
9. oos.writeObject(new Object());
10. oos.close();
11. fos.close();
12. }
13. catch (Exception x) { }
14. }
15. }
```

A. Compiler error at line 9.
B. An exception is thrown at line 9.
C. An exception is thrown at line 10.
D. No compiler error and no exception.

29. Which of the following are valid mode strings for the RandomAccessFile constructor? (Choose all that apply.)

A. “r”
B. “ro”
C. “rw”
D. “rws”
E. “rwd”

30. Which of the following are valid arguments to the DataInputStream constructor?

A. File
B. FileReader
C. FileInputStream
D. RandomAccessFile

Input and Output Code Quesions:

File Handling

- Compile, Run and Understand the following program
    - Is a file or new directory structure actually created on the file system?

```
import java.io.*;
class TestFileConstructors {
public static void main (String[] args){
try{
File f1 = new File("java/scjp");
File f2 = new File("java/scjp", "temp/myProg.java");
File f3 = new File(f1, "temp/myProg.java");
System.out.println("path for f1: " + f1.getCanonicalPath());
System.out.println("path for f2: " + f2.getCanonicalPath());
System.out.println("path for f3: " + f3.getCanonicalPath());
}catch (IOException ioe){
ioe.printStackTrace();
}
}
}
```

Streams

- Compile, run and Understand the following code
    - Note: the file scjp.txt must exist in the directory where you run your program
    - Note: make sure the scjp file you use has some data inside
    - Open the scjpcopy.txt
    - Compare it to scjp.txt
    - What happens if scjp.txt is not in the current directory

```
import java.io.*;
public class FileByteCopier {
public static void main(String[] args) throws IOException {
File inputFile = new File("scjp.txt");
File outputFile = new File("scjpcopy.txt");
FileInputStream in = new FileInputStream(inputFile);
FileOutputStream out = new FileOutputStream(outputFile);
int c;
while ((c = in.read()) != -1)out.write(c);
in.close();
out.close();
}
}
```

- Compile, run and Understand the following code
- Note: the output is written out to orders.txt
- Modify the data that is written to the file and screen, compile and run again

- Can you write code to read the information back in from the file???? And then display it to the screen???

```
import java.io.*;

public class ReadWriteDataTest {

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

   String dataFile = "orders.txt";

  DataOutputStream out = new DataOutputStream(new FileOutputStream(dataFile));

        double[] priceList = { 19.99, 29.99, 39.99};

        int[] copies = { 100000, 50000,70000};

        String[] titleList = { "SCJP Study Guide",

                               "SCBCD Study Guide",

                               "SCSA Study Guide"};

       // Write out into the file.

        for (int i = 0; i < priceList.length; i++) {

            out.writeDouble(priceList[i]);

            out.writeChar('\t');

            out.writeInt(copies[i]);

            out.writeChar('\t');

            out.writeChars(titleList[i]);

            out.writeChar('\n');

        }
        out.close();
    }
}
```

Readers and Writers

- Compile, run and understand the following code
- Note how the BufferedReader and BufferedWriter are constructred
- What is the advantage of using BufferedReader and BufferedWriter over FileReader and Filewriter

```
import java.io.*;

public class FileBufferCopier {

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

            File inputFile = new File("scjp.txt");

            File outputFile = new File("scjpcopy.txt");

      BufferedReader in = new BufferedReader(new FileReader(inputFile));

      BufferedWriter out = new BufferedWriter(new FileWriter(outputFile));

      String line;

      while ((line = in.readLine()) != null){

       out.write(line);

       out.newLine();

      }
      in.close();
      out.close();
    }
}
```

Serialization

- Write an application that
    - Contains a class called ClassForSerialization
    - implements the Serializable interface
    - contains two instance variables of type int
        - One of these variables should not be serialised with the object (hint-remember the transient keyword)
    - contains a main method that creates an instance of the class
    - serializes the object – i.e. writes it to a file
    - reads the objects back in and access the seriailized instance variable and prints it to a screen

Solution

```
import java.io.*;

class ClassForSerialization implements Serializable{

int x=9;
transient int y=10;

public static void main (String[] args){
try{
FileOutputStream out = new FileOutputStream("objectStore.ser");
ObjectOutputStream os = new ObjectOutputStream(out);

os.writeObject(new ClassForSerialization());

os.flush();

FileInputStream in = new FileInputStream("objectStore.ser");
ObjectInputStream is = new ObjectInputStream(in);

ClassForSerialization cfs = (ClassForSerialization)is.readObject();
System.out.println(cfs.x);
}
catch(Exception e){
System.out.println(e);
}
}
}
```

- **Compile, run and Understand the following code:**

```
import java.io.*;
class CodeWalkSeven{
public static void main(String [] args) {
Car c = new Car("Nissan", 1500, "blue");
System.out.println("before: " + c.make + " "+ c.weight);
try {
FileOutputStream fs = new FileOutputStream("Car.ser");
ObjectOutputStream os = new ObjectOutputStream(fs);
os.writeObject(c);
os.close();
}catch (Exception e) { e.printStackTrace(); }
try {
FileInputStream fis = new FileInputStream("Car.ser");
ObjectInputStream ois = new ObjectInputStream(fis);
c = (Car) ois.readObject();
ois.close();
}catch (Exception e) { e.printStackTrace(); }
System.out.println("after: " + c.make + " ". + c.weight);
}
20. }
21. class NonLiving {}
22. class Vehicle extends NonLiving {
23. String make = "Lexus";
24. String color = "Brown";
25. }
26. class Car extends Vehicle implements Serializable {
27. protected int weight = 1000;
28. Car(String n, int w, String c) {
29. color = c;
30. make = n;
31. weight = w;
32. }
33. }
```

<u>Input and Output Review Questions Solutions:</u>

1. Answer: B, C, and D
There is no method available to change the current working directory. You can create and delete a file by using the createNewFile() and delete() methods, respectively. You can also change the security on a file to read-only by invoking the method setReadOnly().

2. Answer: B
Creating an instance of the File class does not create a file on the system. However, you can invoke an appropriate method of the File instance to create the file if you want to.

3. Answer: A
You need DataInputStream chained to FileInputStream in order to read primitive data types from a file; just one of these streams will not do the job. FileInputStream reads bytes and passes them to DataInputStream, which converts them into data types. The FileReader class is used to read 16-bit characters, and not primitive data types.

4. Answer: A, C, and D
FileInputStream and FileOutputStream are limited to reading only 8-bit bytes; they cannot read 16-bit characters.

5. Answer: A
You need ObjectOutputStream chained to FileOutputStream in order to write an object from your program into a file; just one of these streams will not do the job.

6. Answer: A
The ObjectOutputStream can be used to write strings to a file.

7. Answer: B
The byte is 1 byte long and the float is 4 bytes long.

8. Answer: B
The high-level classes such as BufferedReader cannot be directly connected to I/O devices such as files.

9. Answer: E

When an object is serialized, the class name, non-static data, and non-transient data are saved. FileInputStream and FileReader are low-level streams and can be directly connected to a file. FileInputStream reads the data in binary format, but it can read the text files.

10. Answer: B
Only the Serializable objects are serialized. Although the class Car extends Vehicle, Vehicle is not Serializable. So, when the object of Car is retrieved, the constructor of Vehicle is executed and the make variable is reset to Lexus although it was stored as Nissan.

11. Answer: D.
UTF characters are as big as they need to be. Unicode characters are all 16 bits. There is no such thing as a bytecode character; bytecode is the format generated by the Java compiler.

12. Answer: D.
A, B, and C are all false. The File constructor doesn't check the file-naming semantics. Construction and garbage collection of a File have no effect on the local file system.

13. Answer: D.
The File class does not provide a way to change the current working directory.

14. Answer: A.
The list() method returns an array of strings.

15. Answer: C.
The writeInt() call writes out an int, which is 4 bytes long; the writeDouble() call writes out a double, which is 8 bytes long. The total is 12 bytes.
16. Answer: B.
All the code is perfectly legal, so no exceptions are thrown. The first byte in the file is 10, the next byte is 11, the next is 12, and so on. The byte at file position 10 is 20, so the output is i = 20.

17. Answer: A, D.
Option A chains a data input stream onto a file input stream. D simply uses the RandomAccessFile class. B fails because the FileReader class has no readInt() method; readers and writers handle only text. C fails because the PipedInputStream class has nothing to do with file I/O. (Piped input and output streams are used in inter-thread communication.) E fails because you cannot chain a data input stream onto a file reader. Readers read chars, and input streams handle bytes

18. Answer: E.
Readers and writers deal only with character I/O.

19. Answer: E.
Constructing an instance of the File class has no effect on the local file system.

20. Answer: A.
Compilation fails at lines 4 and 5, because there is no constructor for BufferedOutputStream that takes a RandomAccessFile object as a parameter. You can be sure of this even if you are not familiar with buffered output streams, because random-access files are completely incompatible with the stream/reader/writer model.

21. Answer: D.
Default serialization is bypassed only if the writeObject() method has private access.

22. Answer:D.
Default deserialization is bypassed only if the readObject() method has private access.

23. Answer: B.
The lowest-level non-serializable superclass of the object being deserialized must have a no-args constructor.

24. Answer: C.
An externalizable object must have a no-args constructor.

25. Answer: A, B, E.
Default serialization writes all non-static non-transient fields.

26. Answer: E.
The createNewFile() method creates a new empty file.

27. Answer: A, B, C, D.
Both System.out and System.err are instances of PrintStream, which has a println() method and (as of version 5.0) a format() method.

28. Answer: B.
The writeObject() method is declared to take an Object argument. At runtime, there is a precondition check to make sure the argument implements Serializable, which Object doesn't do.

29. Answer: A, C, D, E.
Only "ro" is not valid. "r" opens for reading only. "rw" opens for reading and writing. "rws" opens for reading and writing, with immediate updating of data and metadata changes. "rwd" opens for reading and writing, with immediate updating of data (but not metadata) changes.

30. Answer: C.
A DataInputStream reads bytes from its data source, which must be an InputStream. The only valid option is C, FileInputStream.

Strings, Formatters and Wrappers Review Questions:

1. Given the following:

```
public class StringIndexMute{
public static void main(String[] args){
StringBuilder str = new StringBuilder("0123 456 ");
if (str.length() == 9)
str.insert(9, "abcde");
str.delete(2,5);
System.out.println(str.indexOf("d"));
}
}
```

What is the result?
A. 9
B. 8
C. 7
D. -1
E. Compilation fails.
F. An exception is thrown at runtime.

2. Consider the following code fragment:

```
1. public class MyStringClass extends String {
2. public static void main(String[] args) {
3. String str= "I" + "We";
4. System.out.println(str);
5. }
6.}
```

Which of the following is true about this code fragment?

A. A compiler error occurs at line 3.
B. A compiler error occurs at line 1.
C. An exception is thrown at execution time.
D. The code compiles and executes without any error, and generates the output IWe.

3. Consider the following line of code:

```
String str = new String("Hello");
```

Which of the following lines of code modify the string to which str refers? (Choose all that apply.)

A. str.concat("dear");
B. str.substring(2);
C. str.replace('H', 'M');
D. str.trim();
E. None of the above

4. Consider the following code fragment:

```
String s1 = "Hello";
String s2 = new String(s1);
if ( s1 == s2 ) {
System.out.println("s1 and s2 point to the same thing!");
}else {
System.out.println("s1 and s2 do not point to the same thing!");
}
}
```

What would be the output of this code fragment?

A. s1 and s2 point to the same thing!
B. s1 and s2 do not point to the same thing!

C. Compiler error at line 3
D. Error at execution time

5. Consider the following code fragment:

```
1. String s1 = "Hello";
2. String s2 = new String(s1)
3. if ( s1.equals(s2) ) {
4. System.out.println("s1 and s2 point to identical strings!");
5. }else {
6. System.out.println("s1 and s2 do not point to identical thing!");
7. }
```

What would be the output of this code fragment?

A. s1 and s2 point to identical strings!
B. s1 and s2 do not point to identical strings!
C. Compiler error at line 3
D. Exception thrown at execution time

6. Consider the following code fragment:

```
1. String str = new String("Hello");
2. str.concat(" dear");
3. System.out.println(str);
```

What would be the output of this code fragment?

A. Hello dear
B. Hello
C. Compiler error at line 1
D. Exception thrown at execution time
E. dear

7. Consider the following code fragment:

```
1. StringBuffer sbuf = new StringBuffer("Hello");
2. sbuf.append(" dear");
3. System.out.println(sbuf);
```

What would be the output of this code fragment?

A. Hello dear
B. Hello
C. Compiler error at line 1
D. Exception thrown at execution time
E. dear

8. Consider the following lines of code:

```
String s1 = "Whatever";
String s2 = new String("Whatever");
String s3 = new String ("Who");
```

Which of the following statements is true? (Choose all that apply.)

A. The compiler will create two strings Whatever and Who and put them in the pool, and there will be a string Whatever and Who created at runtime.
B. The compiler will create two strings Whatever and Who and put them in the pool, and there will be no string created at runtime.
C. The compiler will create two copies of Whatever and one copy of Who and put them in the pool, and there will be strings Whatever and Who created at runtime.
D. The compiler will create one copy of Whatever and put it in the pool, and there will be strings Whatever and Who created at compile time.
E. The compiler will create a string Whatever and put it in the pool, and there will be a string Who created at runtime.

9. Given the following:

```
class PrintWriterFormat{
public static void main(String[] args){
int x = 123456789;
int y = 987654321;
float z = 7;
System.out.format("-%5d ", x);
System.out.printf("-%5d- ", y);
System.out.printf("-%4.1d- ", z);
}
}
```

What is the result?
A. -123456789 -987654321-
B. -12345 -98765-
C. - 123456789 - 987654321 -
D. Compilation fails.
E. An exception is thrown at runtime.

10. Consider the code:

```
1. import java.io.PrintWriter;
2. class CodeWalkEight{
3. public static void main(String [] args) {
4. TheBooleanGame bg = new TheBooleanGame();
5. bg.printBoolean();
6. }
7. }
8. class TheBooleanGame {
9. public void printBoolean(){
10. String s1 = "1";
11. String s2 = "0";
12. String s3 = "null";
13. String s4 = "True";
14. String s5 = null;
15. Boolean b = new Boolean("True");
16. boolean b2 = false;
17. System.out.printf("%b %b %b %b %b %b %b", s1, s2, s3,s4, s5, b, b2);
18. }
19. }
```

What is the result?

A. true false true true false true false
B. true false false true false true false
C. true true true true false true false
D. Compilation fails.
E. An exception is thrown at runtime.

11. Given a string constructed by calling s = new String(“xyzzy”), which of the calls modifies the string?

A. s.append(“aaa”);
B. s.trim();
C. s.substring(3);
D. s.replace(‘z’, ‘a’);
E. s.concat(s);
F. None of the above

12. Which one statement is true about the following code?

```
1. String s1 = “abc” + “def”;
2. String s2 = new String(s1);
3. if (s1 == s2)
4. System.out.println(“== succeeded”);
5. if (s1.equals(s2))
6. System.out.println(“.equals() succeeded”);
```

A. Lines 4 and 6 both execute.
B. Line 4 executes and line 6 does not.
C. Line 6 executes and line 4 does not.
D. Neither line 4 nor line 6 executes.

13. Which one statement is true about the following code fragment?

```
1. String s = “abcde”;
2. StringBuffer s1 = new StringBuffer(“abcde”);
3. if (s.equals(s1))
4. s1 = null;
5. if (s1.equals(s))
6. s = null;
```

A. Compilation fails at line 1 because the String constructor must be called explicitly.
B. Compilation fails at line 3 because s and s1 have different types.
C. Compilation succeeds. During execution, an exception is thrown at line 3.
D. Compilation succeeds. During execution, an exception is thrown at line 5.
E. Compilation succeeds. No exception is thrown during execution.

14. In the following code fragment, after execution of line 1, sbuf references an instance of the

StringBuffer class. After execution of line 2, sbuf still references the same instance.

1. StringBuffer sbuf = new StringBuffer(“abcde”);
2. sbuf.insert(3, “xyz”);

A. True
B. False

15. In the following code fragment, after execution of line 1, sbuf references an instance of the StringBuffer class. After execution of line 2, sbuf still references the same instance.

1. StringBuffer sbuf = new StringBuffer(“abcde”);
2. sbuf.append(“xyz”);

A. True
B. False

16. In the following code fragment, line 4 is executed.

1. String s1 = “xyz”;
2. String s2 = “xyz”;
3. if (s1 == s2)
4. System.out.println(“Line 4”);

A. True
B. False

17. In the following code fragment, line 4 is executed.

1. String s1 = “xyz”;
2. String s2 = new String(s1);
3. if (s1 == s2)
4. System.out.println(“Line 4”);

A. True
B. False

18. Suppose prim is an int and wrapped is an Integer. Which of the following are legal Java statements? (Choose all that apply.)
A. prim = wrapped;
B. wrapped = prim;
C. prim = new Integer(9);
D. wrapped = 9;

19. Which line of code tells a scanner called sc to use a single digit as a delimiter?
A. sc.useDelimiter(“d”);
B. sc.useDelimiter(“\d”);
C. sc.useDelimiter(“\\d”);
D. sc.useDelimiter(“d+”);
E. sc.useDelimiter(“\d+”);
F. sc.useDelimiter(“\\d+”);
20. Which of the following statements are true?

A. StringBuilder is generally faster than StringBuffer.
B. StringBuffer is generally faster than StringBuilder.
C. StringBuilder is threadsafe; StringBuffer is not.
D. StringBuffer is threadsafe; StringBuilder is not.

Strings, Formatters, and Wrappers Review Question Solutions:

1. Answer: A
Recall that the String indexing starts from 0 and spaces are counted.

2. Answer: B
You cannot extend the final class String.

3. Answer: E
Strings are immutable.

4. Answer: B
The statement with the new operator is executed at runtime, and a new string is created even if an identical string exists in the pool.

5. Answer: A
The equals(…) method in the String class performs a deep comparison; that is, even if two string references point to two different but identical strings, the comparison returns true.

6. Answer: B
The str still refers to the old string because it is not changed. The String objects are immutable.

7. Answer: A
The StringBuffer objects are changed; they are not immutable.

8. Answer: A, D , E
The compiler creates a string when it sees a string literal, but it does not create duplicates. The statement with the new operator will be executed at runtime, and the string will be created even if an identical string was created by the compiler. So A is correct and since D and E are subsets of the correct answer A they must also be correct.

9. Answer: E
An exception is thrown at runtime because d in the format string is used for integers and not for floats.

10. Answer: C
Any non-null string evaluates to true, and a null string evaluates to false. Also, note that line 15 involves wrappers.

11. Answer:F.
Strings are immutable.

12. Answer: C.
Because s1 and s2 are references to two different objects, the == test fails. However, the strings contained within the two String objects are identical, so the equals() test passes.

13. Answer: E.
A is wrong because line 1 is a perfectly acceptable way to create a String and is actually more efficient than explicitly calling the constructor. B is wrong because the argument to the equals() method is of type Object; thus any object reference or array variable may be passed. The calls on lines 3 and 5 return false without throwing exceptions because s and s1 are objects of different types.

14. Answer: A.
The StringBuffer class is mutable. After execution of line 2, sbuf refers to the same object, although the object has been modified.

15. Answer: A.
The StringBuffer class is mutable. After execution of line 2, sbuf refers to the same object, although the object has been modified.

16. Answer: A.
Line 1 constructs a new instance of String and stores it in the string pool. In line 2, “xyz” is already represented in the pool, so no new instance is constructed.

17. Answer: B.
Line 1 constructs a new instance of String and stores it in the string pool. Line 2 explicitly constructs another instance.

18. Answer: A, B, C, D. All four statements are legal, thanks to boxing and unboxing.

19. Answer: C.
The + sign matches one or more occurrences, so it won’t match just a single digit. The correct string is “\d”; the extra escape is consumed by the Java compiler when it builds the literal string.

20. Answer: A, D.
StringBuffer is threadsafe, StringBuilder is fast.

Strings, Formatters and Wrappers Code Questions:

Strings

- Compile, run and understand the following code

```
public class StringEqual {
public static void main(String[] args) {
String str1 = "Hello Dear!";
String str2 = "Hello Dear!";
String str3 = new String ("Hello Dear!");
if (str1.equals(str2)) {
System.out.println("str1 and str2 refer to identical strings.");
} else {
System.out.println("str1 and str2 refer to non-identical strings.");
}
if (str1 == str2) {
System.out.println("str1 and str2 refer to the same string.");
} else {
System.out.println("str1 and str2 refer to different strings.");
}
if (str1.equals(str3)) {
```

```
System.out.println("str1 and str3 refer to identical strings.");
} else {
System.out.println("str1 and str3 refer to non-identical strings.");
}
if (str1 == str3) {
System.out.println("str1 and str3 refer to the same string.");
} else {
System.out.println("str1 and str3 refer to different strings.");
}
}
}
```

- Compile, run and understand the following code

```
public class StringAndBuffer{
public static void main(String[] args) {
String sl = "String literal!";
String sn = new String("String new");
StringBuffer sb = new StringBuffer ("String buffer");
sl.concat(" Ya!");
sn.concat(" Ya!");
sb.append(" Ya!");
System.out.println("sl after concat(): " + sl);
System.out.println("sn after concat(): " + sn);
System.out.println("sb after append(): " + sb);
}
}
```

Locales

- Compile, run and understand the following code

```
import java.util.*;
class LocalityTest {
public static void main(String[] args){
Locale india= new Locale("pa", "IN");
Locale unitedStates = new Locale("en", "US");
Locale america = Locale.US;
Locale english = new Locale("en");
if(unitedStates.equals(america)){
System.out.println("For some folks, America means United States!");
}
if(unitedStates.equals(india)){
System.out.println("There is some bug in the code!");
}else {
System.out.println("The fact that Columbus stumbled into America when he was looking for India does not make America and India the same!");
}
System.out.println("The default Locale of this JVM is: "+ Locale.getDefault());
System.out.println("The default language for this instance of India is: "+ india.getLanguage());
System.out.println("The display language for this instance of India is: "+ india.getDisplayLanguage());
System.out.println("The display name for this instance of US: "+ america.getDisplayName());
}
}
```

- Compile, run and understand the following code

```
import java.util.*;
import java.text.*;
class DateFormatter{
public static void main(String[] args){
Date today = new Date();
Locale india= new Locale("en", "IN");
Locale america = new Locale("en", "US");
Locale germany = new Locale("de", "DE");
DateFormat nfIndia = DateFormat.getDateInstance(DateFormat.DEFAULT, india);
DateFormat nfAmerica = DateFormat.getDateInstance(DateFormat.DEFAULT, america);
DateFormat nfGermany = DateFormat.getDateInstance(DateFormat.DEFAULT, germany);
System.out.println(nfIndia.format(today));
System.out.println(nfAmerica.format(today));
System.out.println(nfGermany.format(today));
}
}
```

Pattern matching

- Compile, run and understand the following code
- Run the code as follows:
  - java EmailVailidator
  - java EmailValidator www.myemailaddress@crovan.com
  - java EmailValidator myemailaddress@crovan.com
  - java EmailValidator @myemailaddress@crovan.com

```
import java.util.regex.*;
public class EmailValidator {
public static void main(String[] args) {
String email="";
if(args.length < 1){
System.out.println("Command syntax: java EmailValidator <emailAddress>");
System.exit(0);
}else {
email = args[0];
}
//Look for for email addresses starting with
//invalid symbols: dots or @ signs.
Pattern p = Pattern.compile("^\\.+|^\\@+");
Matcher m = p.matcher(email);
if (m.find()) {
System.err.println("Invalid email address: starts with a dot or an @ sign.");
System.exit(0);
}
//Look for email addresses that start with www.
p = Pattern.compile("^www\\.");
m = p.matcher(email);
if (m.find()) {
System.out.println("Invalid email address: starts with www.");
System.exit(0);
}
p = Pattern.compile("[^A-Za-z0-9\\@\\.\\_]");
m = p.matcher(email);
if(m.find()) {
System.out.println("Invalid email address: contains invalid characters");
} else{
System.out.println(args[0] + " is a valid email address.");
}
}
}
```

- Compile, run and understand the following code
- Modify the application to such that the strings is split around the pattern “c”
- Compile and run the application

```
import java.util.regex.*;
public class SplitTest {
public static void main(String[] args) {
String input= "www.cs.cornell.edu";
Pattern p = Pattern.compile("\\.");
String pieces[] = p.split(input);
for (int i=0; i<pieces.length; i++){
System.out.println(pieces[i]);
}
}
}
```

Wrappers

- Compile, run and understand the following code
- replace the byte on line 3 with an int data type
- modify the corresponding wrapper class also

```
1. public class ConversionMachine{
2. public static void main(String[] args){
3. byte b = 4;
4. Byte wbyte = new Byte(b);
5. double d = 354.56d;
6. Double wdouble = new Double(d);
7. System.out.println("wrapped Inside Byte: " + b);
8. System.out.println("double value extracted from Byte: " + wbyte.doubleValue());
9. System.out.println("Wrapped Inside Double: " + d);
10. System.out.println("byte value extracted from Double: " + wdouble.byteValue());
11. }
12.}
```

Collections and Generics Review Questions:

1. Which of the following statements is true? (Choose all that apply.)

A. A Set is a collection that does not allow duplicates.
B. A Map can store duplicate values.
C. A List is a collection that is ordered by index.
D. A List is a collection that cannot have duplicates.
E. The JDK provides a direct implementation of the Collection interface.

2. Which of the following classes implements the java.util.List interface? (Choose all that apply.)

A. java.util.Vector
B. java.util.LinkedLIst
C. java.util.HashTable
D. java.util.OrderedList

3. Which of the following statements is not true about the hashcode? (Choose all that apply.)
A. The hashcode values of two equal objects must be equal.
B. The hashcode values of two unequal objects must be unequal.
C. If the hashcode values of two unequal objects are always unequal, it improves the performance.
D. A function that always return a constant is not a very efficient function.

4. Consider the following code fragment:

```
1. int i = 5;
2. printIt(i);
3. void printIt(Integer wi) {
4. int j = wi;
5. System.out.println("The value is: " + j);
6. }
```

What is the output of this code?

A. The value is: 5
B. Compiler error at line 3
C. Compiler error at line 2
D. Compiler error at line 4
E. Runtime error

5. Which of the following are illegal lines of code?

A. HashMap<Integer, String> hmap = new HashMap<Integer, String>();
B. ArrayList<int> list = new ArrayList<int>();
C. List<String> list2 = new ArrayList<String>();
D. HashSet<String> set = new HashSet<String>;

6. Consider the following code fragment:

```
Integer w1 = new Integer(2);
Integer w2 = new Integer(2);
if(w1 == w2){
System.out.println("w1 is equal to w2!");
}else {
System.out.println("w1 is not equal to w2!");
}
```

What is the output of this code?
A. Compiler error
B. Runtime error
C. w1 is equal to w2!
D. w1 is not equal to w2!

7. Consider the following code fragment:

```
1. ArrayList<ObjectOne> list = new ArrayList<ObjectOne>();
2. list.add(new ObjectOne());
3. list.add(new ObjectOne());
4. list.add(new ObjectOne());
5. Collections.sort(list);
6. class ObjectOne {
7. private int x = 0;
8. private int y = 0;
9. }
```

What is the output of this code fragment?

A. Compiler error at line 5
B. Runtime error at line 5
C. Compiler error at line 3
D. Runtime error at line 3
E. No errors

8. Consider the following code fragment:

```
1. ArrayList<Integer> list = new ArrayList<Integer>();
2. list.add(new Integer(1));
3. list.add(new Integer(2));
4. list.add(new Integer(3));
5. Iterator<Integer> itr = list.iterator();
6. for(Integer wij:list){
7. System.out.println("number: " + wij);
8. }
```

What is the output from this code fragment?

A. number: 1
number: 2
number: 3
B. Compiler error at line 5
C. Compiler error at line 6
D. Compiler error at line 7
E. Runtime error

9. Which of the following collections can you use to store key-value pairs and is thread safe?

A. HashTable
B. HashMap
C. TreeMap
D. Vector

10. Consider the following code fragment:

```
1. import java.io.*;
2. class MyClass{
3. public static void main(String[] args) throws IOException {
4. NumberStore ns = new NumberStore();
5. System.out.println("The number: " + ns.getInteger());
6. ns.setInteger(10);
7. System.out.println("The number: " + ns.getInteger());
8. }
9. }
10. class NumberStore {
11. int i =5;
12. public void setInteger(Integer x){
13. System.out.println("The number passed in is: " + x);
14. i = x;
15. }
16. public int getInteger( ){
17. return i;
18. }
19.}
```

What is the output?

A. Compiler error at line 6
B. Execution error at line 13
C. Compiler error at line 3
D. The number: 5
The number passed in is: 10
The number: 10

11. Consider the following code:

```
class CodeWalkNine{
public static void main(String [] args) {
NissanMaxima s1 = new NissanMaxima(); s1.color = "blue";
NissanMaxima s2 = new NissanMaxima(); s2.color = "blue";
System.out.print("Nissan Maxima: ");
if (s1.equals(s2)) System.out.print("equals ");
if (s1 == s2) System.out.print("== ");
s1=s2;
if (s1.equals(s2)) System.out.print("equals_now ");
if (s1 == s2) System.out.println("==_now");
Lexus x1 = new Lexus(); x1.color = "red";
Lexus x2 = new Lexus(); x2.color = "red";
System.out.print("Lexus: ");
if (x1.equals(x2)) System.out.print("equals ");
if (x1 == x2) System.out.print("== ");
x1=x2;
if (x1.equals(x2)) System.out.print("equals_now ");
if (x1 == x2) System.out.print("==_now");
}
}

class NissanMaxima {
String color;
public String getColor() {
return color;
}
}
class Lexus {
String color;
public boolean equals(Object o) {
return color.equals(((Lexus)o).color);
}
}
```

What is the output?

A. Nissan Maxima: equals ==_now
Lexus: equals ==_now

B. Nissan Maxima: equals equals_now
Lexus: equals equals_now ==_now

C. Nissan Maxima: equals_now ==_now
Lexus: equals_now ==_now

D. Nissan Maxima: equals_now ==_now
Lexus: equals equals_now ==_now

12. Which of the following are legal? (Choose all that apply.)

A. List<String> theList = new Vector<String>;
B. List<String> theList = new Vector<String>();
C. Vector <String> theVec = new Vector<String>;
D. Vector <String> theVec = new Vector<String>();

13. Given the following,

Map<String> names = new HashMap<String>();

which of the following are legal? (Choose all that apply.)

A. Iterator<String> iter = names.iterator();
B. for (String s:names)
C. while (String s:names)

14. Which of the following classes implement java.util.List?

A. java.util.ArrayList
B. java.util.HashList
C. java.util.StackList
D. java.util.Stack

15. Which of the following are methods of the java.util.SortedSet interface?

A. first
B. last
C. headSet
D. tailSet
E. subSet

16. Which of the following are methods of the java.util.SortedMap interface?

A. first
B. last
C. headMap
D. tailMap
E. subMap

17. What happens when you try to compile and run this application?

```
1. import java.util.*;
2.
3. public class Apple {
4. public static void main(String[] a) {
5. Set<Apple> set = new TreeSet<Apple>();
6. set.add(new Apple());
7. set.add(new Apple());
8. set.add(new Apple());
9. }
10. }
```

A. Compiler error.
B. An exception is thrown at line 6.
C. An exception is thrown at line 7.
D. An exception is thrown at line 8.
E. No exception is thrown.

18. Given arrays a1 and a2, which call returns true if a1 and a2 have the same length, and a1[i].equals(a2[i]) for every legal index i?

A. java.util.Arrays.equals(a1, a2);
B. java.util.Arrays.compare(a1, a2);
C. java.util.List.compare(a1, a2);
D. java.util.List.compare(a1, a2);

Collections and Generics Review Questions Solutions:

1. Answer: A, B, and C
A Set does not allow duplicate elements. In a Map, values can be duplicates but keys cannot be. A List is ordered by index and is allowed to have duplicates. JDK does not provide a direct implementation of the Collection interface, but it does provide implementations of its subinterfaces.

2. Answer: A and B
Vector and LinkedList both implement List, while HashTable implements Map. JDK does not offer the class java.util.OrderedList.

3. Answer: B
The hashcode values of two unequal objects don’t have to be unequal, but a hash function that does provide unequal hashcode values for unequal objects does help improve performance.

4. Answer: A
The conversion between primitives and wrappers has been automated and is called autoboxing.

5. Answer: B and D
B is incorrect because collections contain only objects and not primitives. D is incorrect because the parentheses are required to specify the constructor HashSet<String>(). C is correct because this assignment conversion is possible as ArrayList implements List.

6. Answer: D
The object references w1 and w2 do not refer to the same object.

7. Answer: A
The ObjectOne class does not implement the Comparable interface, so you cannot ask for natural ordering.

8. Answer: A
Line 5 defines a valid generic iterator, and line 6 is a for-each loop.

9. Answer: A
HashTable, HashMap, and TreeMap are all implementations of the Map interface and hence let you store key-value pairs, but only HashTable is synchronized (that is, thread safe). Vector is thread safe, but it is an implementation of List and therefore does not allow you to store key-value pairs.

10. Answer: D
The main(…) method can throw an exception. Lines 6 and 13 will not generate errors because autoboxing is in effect.

11. Answer: D
To understand the answer, you need to recall the following facts:
-Lexus overrides the shallow equals(…) implementation in the Object class, whereas NissanMaxima
does not.
-When two object references refer to the same object, they will pass the shallow equality test.

12. Answer: B, D.
A and C are illegal, because a constructor call consists of a class name followed by parentheses. B and D add the parentheses. B assigns a Vector of Strings to a List of Strings, which is a legal assignment conversion.

13. Answer: A, B.
A is the way to get a generic Iterator over a generic Set. B uses the enhanced for loop to iterate the Set's elements. C is illegal because there is no such thing as an enhanced while loop.

14. Answer: A, D.
The HashList and StackList classes do not exist.

15. Answer: A, B, C, D , E.
These methods all exist.

16. Answer: C, D, E.
The SortedMap interface has firstKey() and lastKey() methods, but not first() or last().

17. Answer: C.
The Apple class doesn't implement Comparable, so a tree set doesn't know how to handle it. The problem appears when the second Apple instance is added to the set, requiring the set to perform a comparison between its two members. The add() method throws ClassCastException.

18. Answer: A.
The Arrays class has a static equals() method that compares arrays member by member.

<u>Collections and Generics Code Questions:</u>

The equals() method

- Compile, run and understand the following code
- Why does the test on line 6 fail?
- Why does the test on line 11 pass?

```
1. public class ObjectTest {
2. public static void main(String[] args) {
3. ObjectOne obj1a = new ObjectOne(1,2);
4. ObjectOne obj1b = new ObjectOne(1,2);
5. ObjectTwo obj2 = new ObjectTwo();
6. if (obj1a.equals(obj1b)) {
7. System.out.println("obj1a is equal to obj1b");
8. }else {
9. System.out.println("obj1a is not equal to obj1b");
10. }
11. if(obj2.equals(obj1a)){
12. System.out.println("obj1a is equal to obj2");
13. }else {
14. System.out.println("obj1a is not equal to obj2");
15. }
16. }
17. }
18. class ObjectOne {
19. private int x = 0;
20. private int y = 0;
21. ObjectOne(int x, int y) {
22. this.x = x;
23. this.y = y;
24. }
25. public int getX() {
26. return x;
27. }
28. public int getY() {
29. return y;
30. }
31. }
32. class ObjectTwo {
33. private int x = 1;
34. private int y = 2;
35. public boolean equals(Object object) {
36. ObjectOne obj = (ObjectOne) object;
37. if ((obj.getX() == x) && (obj.getY() == y)) {
38. return true;
39. } else {
40. return false;
41. }
}
}
```

Performing a Search

- Compile, Run and understand the following code:

```
import java.util.*;
class SearchCollectionTest {
public static void main(String[] args) {
String [] str = {"Mark", "Ready", "Set", "Go"};

System.out.println("Unsorted:");
for (String s : str) System.out.print(s + " ");
System.out.println("\nGo = " + Arrays.binarySearch(str, "Go"));
Arrays.sort(str);
System.out.println("Sorted in natural order:");
for (String s : str) System.out.print(s + " ");
System.out.println("\nGo = " + Arrays.binarySearch(str, "Go"));
System.out.println("Sorted in reverse order using a Comparator:");
MyReverseSorter ms = new MyReverseSorter();
Arrays.sort(str, ms);
for (String s : str) System.out.print(s + " ");
System.out.println("\nGo = " + Arrays.binarySearch(str, "Go"));
System.out.println("Go = " + Arrays.binarySearch(str, "Go", ms));
}
}
class MyReverseSorter implements Comparator<String> {
public int compare(String s1, String s2) {
return s2.compareTo(s1);
}
}
```

Generics

- Compile, Run and understand the following code:

```
import java.util.*;
public class GenericTest{
public static void main(String[] args) {
ArrayList<String> myList = new ArrayList<String>();
String st1 = "ready";
String st2 = "set";
String st3 = "go";
myList.add(st1);
myList.add(st2);
myList.add(st3);
String st;
Iterator<String> itr = myList.iterator();
while(itr.hasNext()){
st = itr.next();
System.out.println(st);
}
}
}
```

Autoboxing

- Compile, run and Understand the following code

```
import java.util.*;
public class AutoboxingTest{
public static void main(String[] args) {
Integer wi1 = 1;
wi1++;
Integer wi2 = 2;
if(wi1==wi2){
System.out.println("Area: " + areaOfASquare(4.0d));
}
}
public static Double areaOfASquare(Double side){
return side*side;
}
}
```

- Compile, run and Understand the following code

```
import java.util.*;
public class AutoboxingCollection{
public static void main(String[] args) {
HashMap<String, Integer> hm = new HashMap<String, Integer>();
for (String word : args) {
Integer freq = hm.get(word);
hm.put(word, (freq == null ? 1 : freq + 1));
}
System.out.println(hm);
}
}
```

Threads in Java Review Questions:

1. Which method is used to perform interthread communications? (Choose all that apply.)

A. yield()
B. sleep(…)
C. notify()
D. wait()

2. Which of the following are the methods of the Object class? (Choose all that apply.)
A. yield()
B. sleep(…)
C. run()

D. wait()
E. notify()
3. Consider the following code fragment:

```
1. public class ThreadTest {
2. public static void main(String[] args) {
3. Counter ct = new Counter();
4. ct.start();
5. System.out.println("The thread has been started");
6. }
7. }
8. class Counter extends Thread {
9. protected void run() {
10. System.out.println("Hello");
11. }
12. }
```

What would be the output of this code fragment? (Choose all that apply.)

A. The thread has been started.
Hello
B. Hello
The thread has been started.
C. Either A or B
D. Compiler error on line 9

4. What is true about the wait() method? (Choose all that apply.)

A. A thread calls the wait() method to temporarily stop another thread from running.
B. When a thread executes a call to the wait() method, it itself stops executing temporarily.
C. A call to wait() stops the application from executing.
D. The wait() method belongs to the Object class.
E. The wait() method belongs to the Thread class.

5. Consider the following code:

```
1. public class ThreadOrder {
2. static int count=0;
3. public static void main(String[] args) {
4. Counter ct = new Counter();
5. Tracker trk1 = new Tracker(ct, "thread one");
6. Tracker trk2 = new Tracker(ct, "thread two");
7. trk1.start();
8. trk2.start();
9. }
10.}
11. class Tracker extends Thread {
12. Counter ct;
13. String message;
14. Tracker(Counter ct, String msg) {
15. this.ct = ct;
16. message = msg;
17. }
18. public void run() {
19. System.out.println(message);
20. }
21. }
22. class Counter {
23. private int count=0;
24. public int nextCounter() {
25. synchronized(this) {
26. count++;
27. return count;
28. }
29. }
30. }
```

What would be the output of this code fragment? (Choose all that apply.)

A. thread one
thread two
B. thread two
thread one
C. Sometimes A, sometimes B
D. Runtime exception on line 8

6. What would happen when a thread executes the following statement in its run() method? (Choose all that apply.)

sleep(500);

A. It is going to stop execution, and start executing exactly 500 milliseconds later.
B. It is going to stop execution, and start executing again not earlier than 500 milliseconds later.
C. It is going to result in a compiler error because you cannot call the sleep(…) method inside the run() method.
D. It is going to result in a compiler error because the sleep(…) method does not take any argument.
7. A thread thr is waiting along with some other threads in the waiting pool. How could the method notify() be used to put this thread out of the wait state?

A. Execute thr.notify() from a synchronized piece of code.
B. Execute notify(thr) from a synchronized piece of code.
C. With notify(), you cannot specify which thread would be put out of the wait state.

8. Which of the following methods guarantee to put a thread out of the running state?
A. wait()
B. yield()
C. sleep(500)
D. kill()
E. notify()

9. Which of the following are the valid Thread constructors?
A. Thread()
B. Thread(int millisec)
C. Thread(Runnable r)

D. Thread(Runnable r, String name)
E. Thread(int priority)

10. Which of the following are the methods defined in the Thread class? (Choose all that apply.)

A. yield()
B. sleep(…)
C. run()
D. wait()
E. notify()

11. Given the following code, what is the result?

```
class CodeWalkTen {
public static void main(String [] args) {
Thread myThread = new Thread(new MyThreadClass(Thread.currentThread()));
myThread.start();
System.out.print("after_start ");
myThread.run();
System.out.print("Dead_thread ");
}
}
class MyThreadClass implements Runnable {
Thread mine;
MyThreadClass(Thread mine) { this.mine = mine; }
public void run() {
System.out.print("In_run ");
}
}
```

A. after_start In_run Dead_thread
B. after_start In_run Dead_thread In_run
C. after_start In_run In_run
D. Exception thrown at runtime

12. Which one statement is true concerning the following code?

```
class Greebo extends java.util.Vector
implements Runnable {
public void run(String message) {
System.out.println("in run() method: " +
message);
}
}

class GreeboTest {
public static void main(String args[]) {
Greebo g = new Greebo();
Thread t = new Thread(g);
t.start();
}
}
```

A. There will be a compiler error, because class Greebo does not correctly implement the Runnable interface.
B. There will be a compiler error at line 13, because you cannot pass a parameter to the constructor of a Thread.
C. The code will compile correctly but will crash with an exception at line 13.
D. The code will compile correctly but will crash with an exception at line 14.
E. The code will compile correctly and will execute without throwing any exceptions.

13. Which one statement is always true about the following application?

```
1. class HiPri extends Thread {
2. HiPri() {
3. setPriority(10);
4. }
5.
6. public void run() {
7. System.out.println(
8. “Another thread starting up.”);
9. while (true) { }
10. }
11.
12. public static void main(String args[]) {
13. HiPri hp1 = new HiPri();
14. HiPri hp2 = new HiPri();
15. HiPri hp3 = new HiPri();
16. hp1.start();
17. hp2.start();
18. hp3.start();
19. }
20. }
```

A. When the application is run, thread hp1 will execute; threads hp2 and hp3 will never get the CPU.
B. When the application is run, thread hp1 will execute to completion, thread hp2 will execute to completion, then thread hp3 will execute to completion.
C. When the application is run, all three threads (hp1, hp2, and hp3) will execute concurrently, taking time-sliced turns in the CPU.
D. None of the above scenarios can be guaranteed to happen in all cases.

14. A thread wants to make a second thread ineligible for execution. To do this, the first thread can call the yield() method on the second thread.

A. True
B. False

15. A thread’s run() method includes the following lines:

```
1. try {
2. sleep(100);
3. } catch (InterruptedException e) { }
```

Assuming the thread is not interrupted, which one of the following statements is correct?

A. The code will not compile, because exceptions cannot be caught in a thread’s run() method.
B. At line 2, the thread will stop running. Execution will resume in, at most, 100 milliseconds.
C. At line 2, the thread will stop running. It will resume running in exactly 100 milliseconds.
D. At line 2, the thread will stop running. It will resume running some time after 100 milliseconds have elapsed.

16. A monitor called mon has 10 threads in its waiting pool; all these waiting threads have the same priority. One of the threads is thr1. How can you notify thr1 so that it alone moves from the Waiting state to the Ready state?

A. Execute notify(thr1); from within synchronized code of mon.
B. Execute mon.notify(thr1); from synchronized code of any object.
C. Execute thr1.notify(); from synchronized code of any object.
D. Execute thr1.notify(); from any code (synchronized or not) of any object.
E. You cannot specify which thread will get notified.

17. If you attempt to compile and execute the following application, will it ever print out the message In xxx?

```
class TestThread3 extends Thread {
public void run() {
System.out.println(“Running”);
System.out.println(“Done”);
}

private void xxx() {
System.out.println(“In xxx”);
}

public static void main(String args[]) {
TestThread3 ttt = new TestThread3();
ttt.xxx();
ttt.start();
}
}
```

A. Yes
B. No

18. A Java monitor must either extend Thread or implement Runnable.

A. True
B. False

19. Which of the following methods in the Thread class are deprecated?

A. suspend() and resume()
B. wait() and notify()

C. start() and stop()
D. sleep() and yield()

20. Which of the following statements about threads is true?

A. Every thread starts executing with a priority of 5.
B. Threads inherit their priority from their parent thread.
C. Threads are guaranteed to run with the priority that you set using the setPriority() method.
D. Thread priority is an integer ranging from 1 to 100.

21. Which of the following statements about the wait() and notify() methods is true?

A. The wait() and notify() methods can be called outside synchronized code.
B. The programmer can specify which thread should be notified in a notify() method call.
C. The thread that calls wait() goes into the monitor's pool of waiting threads.
D. The thread that calls notify() gives up the lock.

22. Which of the following may not be synchronized?

A. Blocks within methods
B. Static methods
C. Blocks within static methods
D. Classes

23. Which of the following calls may be made from a non-static synchronized method?

A. A call to the same method of the current object.
B. A call to the same method of a different instance of the current class.
C. A call to a different synchronized method of the current object.
D. A call to a static synchronized method of the current class.

24. How many locks does an object have?

A. One
B. One for each method
C. One for each synchronized method
D. One for each non-static synchronized method

25. Is it possible to write code that can execute only if the current thread owns multiple locks?

A. Yes.
B. No.

26. Which of the following are true? (Choose all that apply.)

A. When an application begins running, there is one daemon thread, whose job is to execute main().
B. When an application begins running, there is one non-daemon thread, whose job is to execute main().
C. A thread created by a daemon thread is initially also a daemon thread.
D. A thread created by a non-daemon thread is initially also a non-daemon thread.

27. Which of the following are true?

A. The JVM runs until there is only one daemon thread.
B. The JVM runs until there are no daemon threads.
C. The JVM runs until there is only one non-daemon thread.
D. The JVM runs until there are no non-daemon threads.

28. How do you prevent shared data from being corrupted in a multithreaded environment?

A. Mark all variables as synchronized.
B. Mark all variables as volatile.
C. Use only static variables.
D. Access the variables only via synchronized methods.

29. How can you ensure that multithreaded code does not deadlock?

A. Synchronize access to all shared variables.
B. Make sure all threads yield from time to time.
C. Vary the priorities of your threads.
D. A, B, and C do not ensure that multithreaded code does not deadlock.

30. Which of the following are true? (Choose all that apply.)

A. When you declare a method to be synchronized, the method always synchronizes on the lock of the current object.
B. When you declare a method to be synchronized, you can specify the object on whose lock the method should synchronize.
C. When you declare a block of code inside a method to be synchronized, the block always synchronizes on the lock of the current object.
D. When you declare a block of code inside a method to be synchronized, you can specify the object on whose lock the block should synchronize.

31. Suppose you want to create a custom thread class by extending java.lang.Thread in order to provide some special functionality. Which of the following must you do?

A. Declare that your class implements java.lang.Runnable.
B. Override run().
C. Override start().
D. Make sure that all access to all data is via synchronized methods.

Threads in Java Review Questions Solutions:

1. Answer: A
A yield() call is designed to let the other threads with the same priority take their turn, although there is no guarantee that this will happen.

2. Answer: D and E
The methods start(), run(), sleep(), and yield() belong to the Thread class, while the methods wait(), notify(), and notifyAll() belong to the Object class.

3. Answer: D
The run() method in the class Thread is public. You cannot assign a weaker access privilege while overriding the method.

4. Answer: B and D
A, C, and E are false statements about the wait() method.

5. Answer: C
It is unpredictable in which order the threads will be executed.

6. Answer: B
You cannot predict exactly when after 500 milliseconds the thread will be scheduled to run.

7. Answer: C
You have no direct control over exactly which thread will be put out of the wait state as a result of the notify() method call.

8. Answer: A and C
A call to yield() will only work if a thread of the same or higher priority is there. There is no method called kill().

9. Answer: A, C, and D
The constructors specified in B and E do not exist in the Thread class.

10. Answer: A, B, and C
The wait() and notify() methods belong to the Object class.

11. Answer: B
B is a possible output because you can execute the run() method of a dead thread; it will execute in the caller's thread.

12. Answer: A.
The Runnable interface defines a run() method with void return type and no parameters. The method given in the problem has a String parameter, so the compiler will complain that class Greebo does not define void run() from interface Runnable. B is wrong, because you can definitely pass a parameter to a thread's constructor; the parameter becomes the thread's target. C, D, and E are nonsense.

13. Answer: C.
There is no way to predict how thread priority manipulation will affect the specific performance of individual threads. Priority manipulation only affects overall statistical behavior.

14. Answer: B.
The yield() method is static and always causes the current thread to yield. In this case, ironically, the first thread will yield.

15. Answer: D.
The thread will sleep for 100 milliseconds (more or less, given the resolution of the JVM being used). Then the thread will enter the Ready state; it will not actually run until the scheduler permits it to run.

16. Answer: E.
When you call notify() on a monitor, you have no control over which waiting thread gets notified.

17. Answer: A.
The call to xxx() occurs before the thread is registered with the thread scheduler, so the object executes the method in the main thread.

18. Answer: B.
A monitor is an instance of any class that has synchronized code.

19. Answer: A.
The suspend() and resume() methods were deprecated in the Java 2 release. They still appear on the API page for Thread but should not be used.

20. Answer: B.
Threads inherit their priority from their parent thread. A is incorrect because, although the default priority for a thread is 5, it may be changed by the parent thread. C is incorrect because Java does not make any promises about priority at runtime. Finally, D is incorrect because thread priorities range from 1 to 10.

21. Answer: C.
The thread that calls wait() goes into the monitor's pool of waiting threads. Option A is incorrect because wait() and notify() must be called from within synchronized code. Option B is incorrect because the notify() call arbitrarily selects a thread to notify from the pool of waiting threads. Option D is incorrect because the thread that calls wait() is the thread that gives up the lock.

22. Answer: D.
You can synchronize a block inside a method by preceding the block with synchronized. You can declare a static method to be synchronized; since there is no object, the thread that executes the method will synchronize on a lock belonging to the class. You can synchronize a block inside a static method just as you would a block within a non-static method; again, synchronization will occur on a lock belonging to the class.

23. Answer: A, B, C, D.
All four situations are legal. Java has no restrictions regarding which methods may call which methods, with respect to synchronization.

24. Answer: A.
An object has only one lock, which controls access to all synchronized code

25. Answer: A.
One way to do this is to have a synchronized method call a synchronized method of a different object.
26. Answer: B, C, D.
A is wrong because main() is executed by a non-daemon thread. B is correct because daemon threads are for the JVM's infrastructure. Non-daemon threads are for programmers. The JVM initially creates a non-daemon thread to run main(). C and D are both correct because a thread's daemon state is the same as that of its creating thread.

27. Answer: D.

The JVM runs until the only threads are its own infrastructure-supporting daemon threads, that is, until there are no more non-daemon threads.

28. Answer: D.
Variables may not be synchronized. Making the variables volatile or static doesn't address the problem. Shared data in a multithreaded environment should be protected from corruption by ensuring that all access to the data is via synchronized methods. Moreover, all methods should synchronize on the same lock.

29. Answer: D.
The only way to ensure that multithreaded code won't deadlock is to be careful. There is no single technique that can guarantee non-deadlocking code.

30. Answer: A, D.
A method may synchronize only on the lock of the current object. A block may synchronize on any object's lock.

31. Answer: B.
Your class should provide a run() method to implement the desired functionality. There is no need to declare that it implements Runnable. Overriding start() is almost never a good idea. You need to provide synchronized access to data only if that data might be corrupted by other threads.

Threads in Java Code Questions:

Threads

- Compile, run and understand the following code
- Which line makes the thread a candidate for running?
- Create a second thread that counts from 5-10 and output each of the values by extending Thread and overriding the run method
- Start this thread from the main method

```
1. public class ThreadTest {
2. public static void main(String[] args) {
3. Counter ct = new Counter();
4. ct.start();
5. System.out.println("The thread has been started");
6. }
7. }
8. class Counter extends Thread {
9. public void run() {
10. for ( int i=1; i<=5; i++) {
11. System.out.println("Count: " + i);
12. }
13. }
14. }
```

Solution

```
public class ThreadTest {
public static void main(String[] args) {
Counter ct = new Counter();
ct.start();
System.out.println("The thread has been started");
Counter2 ct2 = new Counter2();
ct2.start();
System.out.println("The thread has been started");

}

}
class Counter extends Thread {
public void run() {
for ( int i=1; i<=5; i++) {
System.out.println("Count: " + i);
}

}
}

class Counter2 extends Thread {
public void run() {
for ( int i=5; i<=10; i++) {
System.out.println("Count: " + i);
}

}
}
```

- Compile, run and understand the following code
- Create another class called RunCounter2 that extends Nothing and implements Runnable
- In the run method count from 5-10 and output each of the values
- Start the thread from main

```
public class RunnableTest {
public static void main(String[] args) {
RunCounter rct = new RunCounter();
Thread th = new Thread(rct);
th.start();
System.out.println("The thread has been started");
}
}
class RunCounter extends Nothing implements Runnable {
public void run() {
for ( int i=1; i<=5; i++) {
System.out.println("Count: " + i);
}
}
}
class Nothing {
}
```

Solution

```
public class RunnableTest {
  public static void main(String[] args) {
       RunCounter rct = new RunCounter();
       Thread th = new Thread(rct);
       th.start();
       RunCounter2 rct2 = new RunCounter2();
       Thread th2 = new Thread(rct2);
       th2.start();
       System.out.println("The thread has been started");
    }
}
class RunCounter extends Nothing implements Runnable {
  public void run() {
    for ( int i=1; i<=5; i++) {
     System.out.println("Count: " + i);
    }
 }
}
class RunCounter2 extends Nothing implements Runnable {
  public void run() {
    for ( int i=5; i<=10; i++) {
     System.out.println("Count: " + i);
    }
 }
}
  class Nothing {
  }
```

- Compile, run and understand the following code

```
public class MultipleThreads {
public static void main(String[] args) {
System.out.println("The main thread of execution started");
RunCounter rct1 = new RunCounter("First Thread");
RunCounter rct2 = new RunCounter("Second Thread");
RunCounter rct3 = new RunCounter("Third Thread");
}
}
class RunCounter implements Runnable {
Thread myThread;
RunCounter(String name) {
myThread = new Thread(this, name);
myThread.start();
}
public void run() {
for ( int i=1; i<=5; i++) {
System.out.println("Thread: " + myThread.getName() + " Count: " + i);
}
}
}
```

DeadLocks

- Compile, run and understand the following code
    - Note when you execute this code, the output is not deterministic – why???

```java
public class DeadLockTest {
public static void main(String[] args) {
Object obj1 = "objectA";
Object obj2 = "objectB";
DeadLock t1 = new DeadLock(obj1, obj2);
DeadLock t2 = new DeadLock(obj2, obj1);
t1.start();
t2.start();
System.out.println("The threads have been started");
}
}
class DeadLock extends Thread {
private Object resourceA;
private Object resourceB;
public DeadLock(Object a, Object b){
resourceA = a;
resourceB = b;
}
public void run() {
while (true) {
System.out.println("The thread " + Thread.currentThread().getName() + " waiting for a lock on " + resourceA);
synchronized (resourceA){
System.out.println("The thread " + Thread.currentThread().getName()
+ " received a lock on " + resourceA);
System.out.println("The thread " + Thread.currentThread().getName()
+ " waiting for a lock on " + resourceB);
synchronized (resourceB){
System.out.println("The thread " + Thread.currentThread().getName() + " received a lock on " + resourceB);
try{
Thread.sleep(500);
}catch (Exception e){}
}
}
}
}
}
```