

#### SECTION: 1: 1: Declarations, Initialization and Scoping

**OBJECTIVE: 1.1:** Develop code that declares classes (including abstract and all forms of nested classes), interfaces, and enums, and includes the appropriate use of package and import statements (including static imports)

##### 1) Given:

```
5.  enum Towns1{NY, LA, SF}
6.
7.  public class DeclareEnum {
8.
9.      enum Towns2{NY, LA, SF};
10.
11.     public static void main(String [] args) {
12.         enum Towns3{NY, LA, SF};
13.     }
14. }
```

##### What is the result?

- a) The code compiles.
- b) Compilation fails due to an error on line 5.
- c) Compilation fails due to an error on line 9.
- d) Compilation fails due to an error on line 12. (\*)
- e) Compilation fails due to errors on lines 5 and 12.
- f) Compilation fails due to errors on lines 9 and 12.

##### REFERENCE:

JLS 3.0, 8.9

Option D is correct. An enum may NOT be declared in a method.

**OBJECTIVE: 1.5:** Given a code example, determine if a method is correctly overriding or overloading another method, and identify legal return values (including covariant returns), for the method.

##### 2) Given:

```
1. class SuperFoo {
2.     SuperFoo doStuff(int x) {
3.         return new SuperFoo();
4.     }
5. }
6.
7. class Foo extends SuperFoo {
8.     // insert code here
9. }
```

##### And four declarations:

```
Foo doStuff(int x) { return new Foo(); }
Foo doStuff(int x) { return new SuperFoo(); }
SuperFoo doStuff(int x) { return new Foo(); }
```

```
SuperFoo doStuff(int y) { return new SuperFoo(); }
```

**How many, inserted independently at line 8, will compile?**

- a) 0
- b) 1
- c) 2
- d) 3 (\*)
- e) 4

**REFERENCE:**

JLS 3.0,

Option D is correct. `Foo doStuff()` cannot return a `SuperFoo` and co-variant returns are legal.

**SECTION: 2: 2: Flow Control**

**OBJECTIVE: 2.3:** Develop code that makes use of assertions, and distinguish appropriate from inappropriate uses of assertions.

**3) Given:**

```
1. class TestAssert {
2. public static void main(String [] args) {
3. assert(false): "more info ";
4. System.out.println("after assert ");
5. }
6. }
```

**Which is true?**

- a) The command-line invocation `java TestAssert` will produce the output `more info`.
- b) The command-line invocation `java TestAssert` will produce the output `after assert`. (\*)
- c) The command-line invocation `java TestAssert` will produce the output `more info after assert`.
- d) The command-line invocation `java TestAssert` will produce the output `after assert more info`.

**REFERENCE:**

JLS 3.0, and Javadocs for assertions

Option B is correct because assertions were not enabled when the class was invoked.

**SECTION: 3: 3: API Contents**

**OBJECTIVE: 3.3:** Develop code that serializes and/or de-serializes objects using the following APIs from `java.io`: `DataInputStream`, `DataOutputStream`, `FileInputStream`, `FileOutputStream`, `ObjectInputStream`, `ObjectOutputStream`, and `Serializable`.

**4) Given:**

```
11. class Ford extends Car implements Serializable {
12. Ford() { System.out.print("new Ford "); }
13. }
14.
15. class Car {
16. Car() { System.out.print("new Car "); }
17. }
```

**If you attempt to deserialize a properly serialized instance of `Ford`, what is the result?**

- a) `new Car` (\*)

- b) new Ford
- c) new Car new Ford
- d) new Ford new Car
- e) Compilation fails.
- f) An exception is thrown at runtime.

**REFERENCE:**

API

Option A is correct. A superclass does not have to be serializable, but its constructor will run when a serializable subclass instance is deserialized.

**OBJECTIVE: 3.5:** Write code that uses standard J2SE APIs in the java.util and java.util.regex packages to format or parse strings or streams. For strings, write code that uses the Pattern and Matcher classes and the String.split method. Recognize and use regular expression patterns for matching (limited to: .(dot), \*(star), +(plus), ?, \d, \s, \w, [], ()) The use of \*, +, and ? will be limited to greedy quantifiers, and the parenthesis operator will only be used as a grouping mechanism, not for capturing content during matching. For streams, write code using the Formatter and Scanner classes and the PrintWriter.format/printf methods. Recognize and use formatting parameters (limited to: %b, %c, %d, %f, %s) in format strings.

**5) Given:**

```

1. import java.io.PrintWriter;
2.
3. class DoFormat {
4.     public static void main(String [] args) {
5.         int x = 42;
6.         int y = 12345;
7.         float z = 7;
8.         System.out.format("-%4d- ", x);
9.         System.out.format("-%4d- ", y);
10.        System.out.format("-%4.1d- ", z);
11.    }
12. }

```

**What is the result?**

- a) Compilation fails.
- b) -42- -1234- -7.0-
- c) - 42- -1234- - 7.0-
- d) - 42- -12345- - 7.0-
- e) An exception is thrown at runtime. (\*)

**REFERENCE:**

API

Option E is correct. A d in the format string is for integers, NOT floats.

**SECTION: 4: 4: Concurrency**

**OBJECTIVE: 4.4:** Given a scenario, write code that makes appropriate use of wait, notify, or notifyAll.

**6) Which two are true? (Choose two.)**

- a) The code outputs 1 3 4.
- b) The code outputs 3 4 1.
- c) The code outputs 1 2 3 4.
- d) The code outputs 1 3 4 2. (\*)
- e) The code never completes.

f) The code runs to completion. (\*)

### REFERENCE:

JLS 3.0,

Options D and F are correct. The `if` block puts the first thread in a wait state. The `else` block sleeps for two seconds then notifies the first thread, which then completes.

### EXHIBIT:

```
1. class Waiting implements Runnable {
2.     boolean flag = true;
3.     public synchronized void run() {
4.         if (flag) {
5.             flag = false;
6.             System.out.print("1 ");
7.             try { this.wait(); } catch (Exception e) { }
8.             System.out.print("2 ");
9.         }
10.        else {
11.            flag = true;
12.            System.out.print("3 ");
13.            try { Thread.sleep(2000); } catch (Exception e) { }
14.            System.out.print("4 ");
15.            notify();
16.        }
17.    }
18.    public static void main(String [] args) {
19.        Waiting w = new Waiting();
20.        new Thread(w).start();
21.        new Thread(w).start();
22.    }
23. }
```

### SECTION: 5: 5: OO Concepts

**OBJECTIVE: 5.2:** Given a scenario, develop code that demonstrates the use of polymorphism. Further, determine when casting will be necessary and recognize compiler vs. runtime errors related to reference casting.

#### 7) Given:

```
1. class Alpha { void m1() {} }
2. class Beta extends Alpha { void m2() { } }
3. class Gamma extends Beta { }
4.
5. class GreekTest {
6.     public static void main(String [] args) {
7.         Alpha [] a = {new Alpha(), new Beta(), new Gamma() };
8.         for(Alpha a2 : a) {
9.             a2.m1();
10.            if (a2 instanceof Beta || a2 instanceof Gamma)
11.                // insert code here
12.            }
13.        }
14.    }
```

**Which code, inserted at line 11, will compile but cause an exception to be thrown at runtime?**

- a) `a2.m2();`
- b) `((Beta)a2).m2();`
- c) `((Alpha)a2).m2();`
- d) `((Gamma)a2).m2();` (\*)

**REFERENCE:**

JLS 3.0,

Option D is correct. Options A and C will NOT compile, option B will compile and run.

Option D throws an exception because type `Alpha` has no `m2` method.

**SECTION: 6: 6: Collections / Generics**

**OBJECTIVE: 6.2:** Distinguish between correct and incorrect overrides of corresponding `hashCode` and `equals` methods, and explain the difference between `==` and the `equals` method.

**8) Given:**

```
1. class Sock {
2.     String size;
3.     String color;
4.     public boolean equals(Object o) {
5.         Sock s = (Sock) o;
6.         return size.equals(s.size) && color.equals(s.color);
7.     }
8. }
```

**Which two are true? (Choose two.)**

- a) Two instances of `Sock` with the same size and color will have the same hashcode.
- b) Two instances of `Sock` with the same size and color might have different hashcodes. (\*)
- c) A `Hashtable` that uses `Sock` instances as keys will always be able to successfully retrieve objects stored in it.
- d) A `Hashtable` that uses `Sock` instances as keys will NOT always be able to successfully retrieve objects stored in it. (\*)

**REFERENCE:**

API,

Options B and D are correct. For `Maps` to work with `Sock`, `hashCode` must be properly overridden.

**OBJECTIVE: 6.5:** Use capabilities in the `java.util` package to write code to manipulate a list by sorting, performing a binary search, or converting the list to an array. Use capabilities in the `java.util` package to write code to manipulate an array by sorting, performing a binary search, or converting the array to a list. Use the `java.util.Comparator` and `java.lang.Comparable` interfaces to affect the sorting of lists and arrays. Furthermore, recognize the effect of the "natural ordering" of primitive wrapper classes and `java.lang.String` on sorting.

**9) Given:**

```
1. import java.util.*;
2. class MyList {
3.     public static void main(String [] args) {
4.         LinkedList<String> list = new LinkedList<String>();
5.         list.add("one "); list.add("two "); list.add("three ");
6.         String [] sa = new String[3];
7.         // insert code here
8.         for(String s : sa)
9.             System.out.print(s);
10.    }
11. }
```

**Which, inserted at line 7, allows the code to compile and run without exception?**

- a) `sa = list.toArray();`

- b) `sa = list.toArray(sa); (*)`
- c) `sa = (String) list.toArray();`
- d) `sa = (String []) list.toArray();`

**REFERENCE:**

JLS 3.0,

Option B is correct. This is the correct syntax to convert a `List` to a `String` array.

**SECTION: 7: 7: Fundamentals**

**OBJECTIVE: 7.5:** Given the fully-qualified name of a class that is deployed inside and/or outside a JAR file, construct the appropriate directory structure for that class. Given a code example and a classpath, determine whether the classpath will allow the code to compile successfully.

**10) Given a JAR file named `MyJar.jar` containing:  
`com/Gamma.class`**

**And that this class was compiled from the following file:**

1. `package com;`
2. `public class Gamma { }`

**The directory you are in contains a subdirectory `jarDir` that contains `MyJar.jar`. Which command line will correctly invoke the compiler for a Java file named `Test.java` that uses the `Gamma` class?**

- a) `javac -path MyJar.jar Test.java`
- b) `javac -classpath MyJar.jar Test.java`
- c) `javac -path jarDir/MyJar.jar Test.java`
- d) `javac -path jarDir/com/MyJar.jar Test.java`
- e) `javac -classpath jarDir/MyJar.jar Test.java (*)`
- f) `javac -classpath jarDir/com/MyJar.jar Test.java`

**REFERENCE:**

API

Option E is the correct command-line syntax to include the JAR file in the classpath.