

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.

1) 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:

```
I.   Foo doStuff(int x) { return new Foo(); }
II.  Foo doStuff(int x) { return new SuperFoo(); }
III. SuperFoo doStuff(int x) { return new Foo(); }
IV.  SuperFoo doStuff(int y) { return new SuperFoo(); }
```

Which, inserted independently at line 8, will compile?

- a) Only I.
- b) Only IV.
- c) Only I and III.
- d) Only I, II, and III.
- e) Only I, III, and IV. (*)
- f) All four declarations will compile.

REFERENCE:

JLS 3.0

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

OBJECTIVE: 2.2: Develop code that implements all forms of loops and iterators, including the use of `for`, the enhanced `for` (`for-each`), `do`, `while`, `labels`, `break`, and `continue`; and explain the values taken by loop counter variables during and after loop execution.

2) Given:

```
5. public class Buddy {
6.     public static void main(String[] args) {
7.         def:
8.         for(short s = 1; s < 7; s++) {
9.             if(s == 5) break def;
10.            if(s == 2) continue;
11.            System.out.print(s + ".");
12.        }
13.    }
```

14. }

What is the result?

- a) 1.
- b) 1.2.
- c) 1.3.4. (*)
- d) 1.2.3.4.
- e) 1.3.4.5.6.
- f) 1.2.3.4.5.6.
- g) Compilation fails.

REFERENCE:

JLS 3.0

Option C is correct. The continue skips the current iteration, the break ends the entire loop.

OBJECTIVE: 2.5: Recognize the effect of an exception arising at a specified point in a code fragment. Note that the exception may be a runtime exception, a checked exception, or an error.

3) Given:

```
1. class Birds {
2.     public static void main(String [] args) {
3.         try {
4.             throw new Exception();
5.         } catch (Exception e) {
6.             try {
7.                 throw new Exception();
8.             } catch (Exception e2) { System.out.print("inner "); }
9.             System.out.print("middle ");
10.        }
11.        System.out.print("outer ");
12.    }
13. }
```

What is the result?

- a) inner
- b) inner outer
- c) middle outer
- d) inner middle outer (*)
- e) middle inner outer
- f) Compilation fails.
- g) An exception is thrown at runtime.

REFERENCE:

JLS 3.0

Option D is correct. It is legal to nest try/catches and normal flow rules apply.

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:

```
2. import java.io.*;
3. public class Network {
4.     public static void main(String[] args) {
5.         Traveler t = new Traveler();
```

```

6.     t.x1 = 7; t.x2 = 7; t.x3 = 7;
7.     // serialize t then deserialize t
8.     System.out.println(t.x1 + " " + t.x2 + " " + t.x3);
9.     }
10.    }
11.    class Traveler implements Serializable {
12.        static int x1 = 0;
13.        volatile int x2 = 0;
14.        transient int x3 = 0;
15.    }

```

If, on line 7, t is successfully serialized and then deserialized, what is the result?

- a) 0 0 0
- b) 0 7 0 (*)
- c) 0 7 7
- d) 7 0 0
- e) 7 7 0
- f) 7 7 7

REFERENCE:

API

Option B is correct. Because static variables belong to the class and not to instances of a class, they are not serialized.

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) Which regex pattern finds both 0x4A and 0x5 from within a source file?

- a) 0[xX][a-fA-F0-9]
- b) 0[xX](a-fA-F0-9)
- c) 0[xX]([a-fA-F0-9])
- d) 0[xX]([a-fA-F0-9])+ (*)
- e) 0[xX]([a-fA-F0-9])?

REFERENCE:

API,

Option D is correct. The + quantifier finds 1 or more occurrences of hex characters after an 0x is found.

OBJECTIVE: 4.3: Given a scenario, write code that makes appropriate use of object locking to protect static or instance variables from concurrent access problems.

6) Given:

```

5.    public class Lockdown implements Runnable {
6.        public static void main(String[] args) {
7.            new Thread(new Lockdown()).start();
8.            new Thread(new Lockdown()).start();
9.        }
10.       public void run() { locked(Thread.currentThread().getId()); }
11.       synchronized void locked(long id) {

```

```

12.     System.out.print(id + "a ");
13.     System.out.print(id + "b ");
14.     }
15. }

```

What is true about possible sets of output from this code?

- a) Set 6a 7a 7b 8a and set 7a 7b 8a 8b are both possible.
- b) Set 7a 7b 8a 8b and set 6a 7a 6b 7b are both possible. (*)
- c) It could be set 7a 7b 8a 8b but set 6a 7a 6b 7b is NOT possible.
- d) It could be set 7a 8a 7b 8b but set 6a 6b 7a 7b is NOT possible.

REFERENCE:

JLS 3.0

Option B is correct. Two different Lockdown objects are using the locked() method.

OBJECTIVE: 5.5: Develop code that implements "is-a" and/or "has-a" relationships.

7) A programmer wants to develop an application in which Fizzlers are a kind of Whoosh, and Fizzlers also fulfill the contract of Oompahs. In addition, Whooshes are composed with several Wingits.

Which code represents this design?

```

a) class Wingit { }
class Fizzler extends Oompah implements Whoosh { }
interface Whoosh {
Wingits [] w;
}
class Oompah { }
b) class Wingit { }
class Fizzler extends Whoosh implements Oompah { }
class Whoosh {
Wingits [] w;
}
interface Oompah { } (*)
c) class Fizzler { }
class Wingit extends Fizzler implements Oompah { }
class Whoosh {
Wingits [] w;
}
interface Oompah { }
d) interface Wingit { }
class Fizzler extends Whoosh implements Wingit { }
class Wingit {
Whoosh [] w;
}
class Whoosh { }

```

REFERENCE:

Sun's Java course OO226

Option B is correct. 'Kind of' translates to extends, 'contract' translates to implements, and 'composed' translates to a has-a implementation.

OBJECTIVE: 6.3: Write code that uses the generic versions of the Collections API, in particular, the Set, List, and Map interfaces and implementation classes. Recognize the limitations of the non-generic Collections API and how to refactor code to use the generic versions. Write code that uses the NavigableSet and NavigableMap interfaces.

8) Given:

```
4. import java.util.*;
5. public class Quest {
6.     public static void main(String[] args) {
7.         TreeMap<String, Integer> myMap = new TreeMap<String,
Integer>();
8.         myMap.put("ak", 50); myMap.put("co", 60);
9.         myMap.put("ca", 70); myMap.put("ar", 80);
10.        NavigableMap<String, Integer> myMap2 = myMap.headMap("d",
true);
11.        myMap.put("fl", 90);
12.        myMap2.put("hi", 100);
13.        System.out.println(myMap.size() + " " + myMap2.size());
14.    }
15. }
```

What is the result?

- a) 4 4
- b) 5 4
- c) 5 5
- d) 6 5
- e) 6 6
- f) Compilation fails.
- g) An exception is thrown at runtime. (*)

REFERENCE:**API**

Answer: G is correct. Line 12 causes a "key out of range" exception.

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:

```
3. import java.util.*;
4. public class ToDo {
5.     public static void main(String[] args) {
6.         String[] dogs = {"fido", "clover", "gus", "aiko"};
7.         List dogList = Arrays.asList(dogs);
8.         dogList.add("spot");
9.         dogs[0] = "fluffy";
10.        System.out.println(dogList);
11.        for(String s: dogs) System.out.print(s + " ");
12.    }
13. }
```

What is the result?

- a) [fluffy, clover, gus, aiko]
fluffy, clover, gus, aiko,
- b) [fluffy, clover, gus, aiko]
fluffy, clover, gus, aiko, spot,
- c) [fluffy, clover, gus, aiko, spot]
fluffy, clover, gus, aiko,

- d) [fluffy, clover, gus, aiko, spot]
fluffy, clover, gus, aiko, spot,
- e) Compilation fails.
- f) An exception is thrown at runtime. (*)

REFERENCE:

API

Option F is correct. The `asList()` method creates a fixed-size list that is backed by the array, so no additions are possible.

OBJECTIVE: 7.2: Given an example of a class and a command-line, determine the expected runtime behavior.

10) Given:

```
1. class x {
2.     public static void main(String [] args) {
3.         String p = System.getProperty("x");
4.         if(p.equals(args[1]))
5.             System.out.println("found");
6.     }
7. }
```

Which command-line invocation will produce the output found?

- a) java -Dx=y x y z
- b) java -Px=y x y z
- c) java -Dx=y x x y z (*)
- d) java -Px=y x x y z
- e) java x x y z -Dx=y
- f) java x x y z -Px=y

REFERENCE:

API for java command

Option C is correct. `-D` sets a property and `args[1]` is the second argument (whose value is `y`)