**:|**

*Fodder For Thought*

# Java 7 Certification Exam Sample Questions

Posted on March 11, 2015

I will be sitting a Java certification exam soon, and Grad Daniel and I tried Oracle's exam prep sample questions today. I found it a little frustrating that the sample questions didn't explain why the correct answers were the correct answers, and if you didn't know what was wrong, it made it pretty tough to Google the correct solution. So I thought I'd provide my own explanations for the correct answers (and yes, the code is this badly formatted in the sample questions, artificially adding to the difficulty).

**1. Given:**

```
public class Sequence {

Sequence() { System.out.print("c "); }

{ System.out.print("y "); }

public static void main(String[] args) {

new Sequence().go();

}

void go() { System.out.print("g "); }

static { System.out.print("x "); }

}
```

What is the result?

A) c x y g

B) c g x y

C) x c y g

D) x y c g

E) y x c g

F) y c g x

The correct answer is D. We were lucky to stumble upon this SO answer that explains it. Essentially, the bit that prints out "x " is inside what's called a *static initialization block*, and that code gets run first, before anything else. The bit that prints "y " is called an *initilization block*, and that gets injected into the beginning of the constructor so that is printed next(however, if the constructor called super(), that would execute first). Then the rest of the constructor is called, printing "c ". Finally, the method go() is called, printing "g ".

**2. Given:**

```
public class MyStuff {

MyStuff(String n) { name = n; }

String name;

public static void main(String[] args) {

MyStuff m1 = new MyStuff(“guitar”);

MyStuff m2 = new MyStuff(“tv”);

System.out.println(m2.equals(m1));

}

public boolean equals(Object o) {

MyStuff m = (MyStuff) o;

if(m.name != null)

return true;

return false;

}

}
```

What is the result?

A) The output is “true” and MyStuff fulfills the Object.equals() contract.

B) The output is “false” and MyStuff fulfills the Object.equals() contract.

C) The output is “true” and MyStuff does NOT fulfill the Object.equals() contract.

D) The output is “false” and MyStuff does NOT fulfill the Object.equals() contract

E) Compilation fails

The correct answer is C. The output is “true” because *m1.name* is not null. However, the equals() method does *not* fulfill the Object.equals() contract, which you can read about here. The relevant bit is:

It is reflexive: for any non-null reference value x, x.equals(x) should return true.

It is symmetric: for any non-null reference values x and y, x.equals(y) should return true if and only if y.equals(x) returns true.

It is transitive: for any non-null reference values x, y, and z, if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true.

It is consistent: for any non-null reference values x and y, multiple invocations of x.equals(y) consistently return true or consistently return false, provided no information used in equals comparisons on the objects is modified.

For any non-null reference value x, x.equals(null) should return false.

The last part fails, if you try m2.equals(null), you will get a NullPointerException. It also fails if you do:

```
MyStuff m1 = new MyStuff(null);

MyStuff m2 = new MyStuff("asdf");
```

m1.equals(m2) returns true.

m2.equals(m1) returns false.

It's not symmetric, and will not be transitive.

**3. Given:**

```
import java.util.*;

public class Primes {

public static void main(String[] args) {

List p = new ArrayList();

p.add(7);

p.add(2);

p.add(5);

p.add(2);

p.sort();

System.out.println(p);

}

}
```

What is the result?

A) [2, 5, 7]

B) [2, 2, 5, 7]

C) [7, 2, 5, 2]

D) [7, 5, 2, 2]

E) Compilation fails

The correct answer is E. List does not have a sort() method. You can use Collections.sort() to sort a list.

**4. Given:**

```
public class MyLoop {

public static void main(String[] args) {

String[] sa = {“tom “, “jerry “};

for(int x = 0; x < 3; x++) {

for(String s: sa) {

System.out.print(x + ” ” + s);

if( x == 1) break;

}

}

}

}
```

What is the result?

A) 0 tom 0 jerry 1 tom

B) 0 tom 0 jerry 1 tom 1 jerry

C) 0 tom 0 jerry 2 tom 2 jerry

D) 0 tom 0 jerry 1 tom 2 tom 2 jerry

E) 0 tom 0 jerry 1 tom 1 jerry 2 tom 2 jerry

F) Compilation fails.

The correct answer is D.

begin loop 1

x = 0;

begin loop 2

s = “tom ”

**print "0 tom "**

s = "jerry "

**print "0 jerry "**

end loop 2

x++

x = 1

begin loop 2

s = "tom "

**print "1 tom "**

x ==1, so break loop 2

x++

x = 2

begin loop 2

s = "tom "

**print "2 tom "**

s = "jerry "

**print 2 "jerry "**

end loop 2

x++

x = 3

x < 3 is false

end loop 1

**5. Given:**

interface Rideable {

String getGait();

}

public class Camel implements Rideable {

int weight = 2;

public static void main(String[] args) {

new Camel().go(8);

}

void go(int speed) {

++speed;

weight++;

int walkrate = speed * weight;

System.out.print(walkrate + getGait());

}

String getGait() {

return " mph, lope";

}

}

What is the result?

A) 16 mph, lope

B) 18 mph, lope

C) 24 mph, lope

D) 27 mph, lope

E) Compilation fails.

F) An exception is thrown at run time.

The correct answer is E. The method *getGait()* in Camel has "default" or "package-private" level access. The method *getGait()* in interface Rideable is public. *Camel.getGait()* needs to match the same access level as the interface.

Methods in interfaces are by default public. Methods in concrete classes are by default "default".

**6. Given:**

class Alpha {

String getType() { return "alpha"; }

}

class Beta extends Alpha {

String getType() { return “beta”; }

}

class Gamma extends Beta {

String getType() { return “gamma”; }

public static void main(String[] args) {

Gamma g1 = new Alpha();

Gamma g2 = new Beta();

System.out.println(g1.getType() + ” ”

+ g2.getType());

}

}

What is the result?

A) alpha beta

B) beta beta

C) gamma gamma

D) alpha alpha

E) Compilation fails.

The correct answer is E. Alpha and Beta cannot be assigned to an object of Gamma as Gamma is the subclass of Beta, which is a subclass of Alpha. The opposite can be done, i.e. a Gamma assigned to an Alpha, or a Beta, in which case it would print “gamma gamma”.

**7. Given:**

class Feline {

public String type = “f “;

public Feline() {

System.out.print(“feline “);

}

```
}

public class Cougar extends Feline {

public Cougar() {

System.out.print("cougar ");

}

public static void main(String[] args) {

new Cougar().go();

}

void go() {

type = "c ";

System.out.print(this.type + super.type);

}

}
```

What is the result?

A) cougar c c

B) cougar c f

C) feline cougar c c

D) feline cougar c f

E) Compilation fails

F) An exception is thrown at run time.

The correct answer is C. When Cougar's constructor is called, it calls Feline's constructor first, which prints "feline ". Then Cougar's constructor is called, printing "cougar ". go() assigns "c " to the type of the object, which Cougar inherits from Feline. So this.type and super.type are both "c ".

——————————-

Today I learned about initialiser blocks, and I just have to say, in all the code our team has written, I have never seen this once. I'm kinda tempted to slip it in to mess with people, but I think it breaks the readability of code.

Like

Be the first to like this.

**Related**

Oracle Java SE 7 Programmer II - Sample Questions
In "programming"

End of the DotA Guide
In "DotA 2"

Good News, Everybody
In "life"

This entry was posted in programming. Bookmark the permalink.

## 2 Responses to *Java 7 Certification Exam Sample Questions*

Pingback: *2016 in review, part 3 | :|*

Pingback: *The Yearly With Fodder | :|*

**:|**