1Z0-804 Q&As

Java SE 7 Programmer II

Pass Oracle 1Z0-804 Exam with 100% Guarantee

Free Download Real Questions & Answers **PDF** and **VCE** file from:

https://www.geekcert.com/1Z0-804.html

100% Passing Guarantee
100% Money Back Assurance

Following Questions and Answers are all new published by Oracle Official Exam Center

- **Instant Download** After Purchase
- **100% Money Back** Guarantee
- **365 Days** Free Update
- **800,000+** Satisfied Customers

QUESTION 1

Which two properly implement a Singleton pattern?

A. class Singleton { private static Singleton instance; private Singleton () {} public static synchronized Singleton getInstance() { if (instance = = null) { instance = new Singleton (); } return instance; } }

B. class Singleton { private static Singleton instance = new Singleton(); protected Singleton () {} public static Singleton getInstance () { return Instance; } }

C. class Singleton { Singleton () {} private static class SingletonHolder { private static final Singleton INSTANCE = new Singleton (); } public static Singleton getInstance () { return SingletonHolder.INSTANCE; } }

D. enum Singleton { INSTANCE; }

Correct Answer: AB

A: Here the method for getting the reference to the SingleTon object is correct.

B: The constructor should be private such as:

private static Singleton instance = new Singleton();

Note: Java has several design patterns Singleton Pattern being the most commonly used.

Java Singleton pattern belongs to the family of design patterns, that govern the instantiation process. This design pattern proposes that at any time there can only be one instance of a singleton (object) created by the JVM.

The class\\'s default constructor is made private, which prevents the direct instantiation of the object by others (Other Classes). A static modifier is applied to the instance method that returns the object as it then makes this method a class level method that can be accessed without creating an object.

QUESTION 2

Given the code fragment:

public class DisplaValues {

public void printNums (int [] nums){

for (int number: nums) { System.err.println(number);

}

}

}

Assume the method printNums is passed a valid array containing data. Why is this method not producing output on the console?

A. There is a compilation error.

B. There is a runtime exception.

C. The variable number is not initialized.

D. Standard error is mapped to another destination.

Correct Answer: D

The code compiles fine.

The code runs fine.

The err stream can be redirected.

Note:

System.out.println -> Sends the output to a standard output stream. Generally monitor.

System.err.println -> Sends the output to a standard error stream. Generally monitor.

err is the "standard" error output stream. This stream is already open and ready to accept output data.

Typically this stream corresponds to display output or another output destination specified by the host environment or user. By convention, this output stream is used to display error

messages or other information that should come to the immediate attention of a user even if the principal output stream, the value of the variable out, has been redirected to a file or other destination that is typically not continuously monitored.

Reference: java.lang.System

QUESTION 3

Given:

```
class Erupt implements Runnable {

public void run() {

System.out.print(Thread.currentThread().getName());

}

}

public class Yellowstone {

static Erupt e = Erupt();

Yellowstone() { new Thread(e, "const").start(); } // line A

public static void main(String[] args) {

new Yellowstone();
```

```
new Faithful().go();

}

static class Faithful {

void go() { new Thread(e, "inner").start(); } // line B

}

}
```

What is the result?

A. Both const and inner will be in the output.

B. Only const will be in the output.

C. Compilation fails due to an error on line A.

D. Compilation fails due to an error on line B.

E. An exception is thrown at runtime.

Correct Answer: A

The code compiles fine.

Note: The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

This interface is designed to provide a common protocol for objects that wish to execute code while they are active. For example, Runnable is implemented by class Thread. Being active simply means that a thread has been started and has

not yet been stopped.

In addition, Runnable provides the means for a class to be active while not subclassing Thread. A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In

most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods. This is important because classes should not be subclassed unless the programmer intends on modifying

or enhancing the fundamental behavior of the class.

Note 2: start()

Causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread. Reference: java.lang Interface Runnable

QUESTION 4

Sam has designed an application. It segregates tasks that are critical and executed frequently from tasks that are non critical and executed less frequently. He has prioritized these tasks based on their criticality and frequency of execution. After close scrutiny, he finds that the tasks designed to be non critical are rarely getting executed.

From what kind of problem is the application suffering?

A. race condition

B. starvation

C. deadlock

D. livelock

Correct Answer: C

Starvation describes a situation where a thread is unable to gain regular access to shared resources and is unable to make progress. This happens when shared resources are made unavailable for long periods by "greedy" threads. For example, suppose an object provides a synchronized method that often takes a long time to return. If one thread invokes this method frequently, other threads that also need frequent synchronized access to the same object will often be blocked.

Reference: The Java Tutorial, Starvation and Livelock

QUESTION 5

A valid reason to declare a class as abstract is to:

A. define methods within a parent class, which may not be overridden in a child class

B. define common method signatures in a class, while forcing child classes to contain unique method implementations

C. prevent instance variables from being accessed

D. prevent a class from being extended

E. define a class that prevents variable state from being stored when object Instances are serialized

F. define a class with methods that cannot be concurrently called by multiple threads

Correct Answer: B

Note: An abstract method in Java is something like a pure virtual function in C++ (i.e., a virtual function that is declared = 0). In C++, a class that contains a pure virtual function is called an abstract class and cannot be instantiated. The same

is true of Java classes that contain abstract methods.

Any class with an abstract method is automatically abstract itself and must be declared as such.

An abstract class cannot be instantiated.

A subclass of an abstract class can be instantiated only if it overrides each of the abstract methods of its superclass and provides an implementation (i.e., a method body) for all of them. Such a class is often called a concrete subclass, to

emphasize the fact that it is not abstract.

If a subclass of an abstract class does not implement all the abstract methods it inherits, that subclass is itself abstract.

static, private, and final methods cannot be abstract, since these types of methods cannot be overridden by a subclass. Similarly, a final class cannot contain any abstract methods.

A class can be declared abstract even if it does not actually have any abstract methods.

Declaring such a class abstract indicates that the implementation is somehow incomplete and is meant to serve as a superclass for one or more subclasses that will complete the implementation. Such a class cannot be instantiated.