# 1Z0-805 Q&As

Upgrade to Java SE 7 Programmer

## Pass Oracle 1Z0-805 Exam with 100% Guarantee

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

**https://www.geekcert.com/1Z0-805.html**

100% Passing Guarantee
100% Money Back Assurance

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

**QUESTION 1**

Which five items are provided by the Java concurrency utilities?

A. High-performance, flexible thread pools

B. Dynamic adjustment of thread priorities

C. Collection classes designed for concurrent access

D. Atomic variables

E. synchronized wrappers for collection classes in the java.util package,

F. Asynchronous execution of tasks

G. Counting semaphores

H. Concurrent collection sorting implementations

Correct Answer: ACDEG

The Java 2 platform includes a new package of concurrency utilities. These are classes that are designed to be used as building blocks in building concurrent classes or applications. Just as thecollections framework simplified the organization and manipulation of in- memory data by providing implementations of commonly used data structures, the concurrency utilities simplify the development of concurrent classes by providing implementations of building blocks commonly used in concurrent designs. The concurrency utilities include a high- performance, flexible thread pool; a framework for asynchronous execution of tasks; a host of collection classes optimized for concurrent access; synchronization utilities such as counting semaphores (G); atomic variables; locks; and condition variables.

The concurrency utilities includes:

*

Task scheduling framework. The Executor interface standardizes invocation, scheduling, execution, and control of asynchronous tasks according to a set of execution policies. Implementations are provided that enable tasks to be executed within the submitting thread, in a single background thread (as with events in Swing), in a newly created thread, or in a thread pool, and developers can create customized implementations of Executor that support arbitrary execution policies. The built-in implementations offer configurable policies such as queue length limits and saturation policy that can improve the stability of applications by preventing runaway resource use.

*

Fork/join framework. Based on the ForkJoinPool class, this framework is an implementation of Executor. It is designed to efficiently run a large number of tasks using a pool of worker threads (A) . A work-stealing technique is used to keep all the worker threads busy, to take full advantage of multiple processors.

*

(C) Concurrent collections. Several new collections classes were added, including the new Queue, BlockingQueue and BlockingDeque interfaces, and high-performance, concurrent implementations of Map, List, and Queue. See the Collections Framework Guide for more information.

*

(D) Atomic variables. Utility classes are provided that atomically manipulate single variables (primitive types or references), providing high-performance atomic arithmetic and compare-and-set methods. The atomic variable implementations in the java.util.concurrent.atomic package offer higher performance than would be available by using synchronization (on most platforms), making them useful for implementing high-performance concurrent algorithms and conveniently implementing counters and sequence number generators.

*

(E) Synchronizers. General purpose synchronization classes, including semaphores, barriers, latches, phasers, and exchangers, facilitate coordination between threads.

*

Locks. While locking is built into the Java language through the synchronized keyword, there are a number of limitations to built-in monitor locks. The java.util.concurrent.locks package provides a high-performance lock implementation with the same memory semantics as synchronization, and it also supports specifying a timeout when attempting to acquire a lock, multiple condition variables per lock, nonnested ("hand-over-hand") holding of multiple locks, and support for interrupting threads that are waiting to acquire a lock.

*

Nanosecond-granularity timing. The System.nanoTime method enables access to a nanosecond-granularity time source for making relative time measurements and methods that accept timeouts (such as the BlockingQueue.offer, BlockingQueue.poll, Lock.tryLock, Condition.await, and Thread.sleep) can take timeout values in nanoseconds. The actual precision of the System.nanoTime method is platform-dependent.

Reference: Java SE Documentation, Concurrency Utilities

---

**QUESTION 2**

Which code fragments print 1?

A. String arr [] = {"1", "2", "3"}; List arrList = new LinkedList (Arrays.asList (arr)); System.out.println (arrList.get (0));

D. String arr [] = {"1","2","3"}; List arrList = new LinkedList (Arrays.asList (arr)); System.out.println (arrList.get (0));

E. String arr [] = {"1","2","3"}; List extendsString > arrList =new LinkedList (Arrays.asList (arr)); System.out.println (arrList.get (0));

Correct Answer: AC

Note:You can replace the type arguments required to invoke the constructor of a generic class with an empty set of type parameters () as long as the compiler can infer the type arguments from the context. This pair of angle brackets is informally called the diamond.

---

**QUESTION 3**

What statement is true about thread starvation?

A. Thread "A" is said to be starved when it is frequently unable to gain access to a resource that it shares with another thread.

B. Thread "A" is said to be starved when it is blocked waiting for Thread "13," which in turn waiting for Thread "A."

C. Starvation can occur when threads become so busy responding to other threads that they move forward with their other work.

D. When some of the processors in a multi-processor environment go offline and the live thread(s) blocked waiting for CPU cycles, that blocking is referred to as "thread starvation."

Correct Answer: A

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 methodfrequently, other threads that also need frequent synchronized access to the same object will often be blocked.

Reference: The Java Tutorials,Starvation and Livelock

**QUESTION 4**

Which two statements are true about the walkFileTree method of the files class?

A. The file tree traversal is breadth-first with the given FileVisitor invoked for each file encountered.

B. If the file is a directory, and if that directory could not be opened, the postVisitFileFailed method is invoked with the I/O exception.

C. The maxDepth parameter\\'s value is the maximum number of directories to visit.

D. By default, symbolic links are not automatically followed by the method.

Correct Answer: CD

C: The method walkFileTree(Path start, Set options, int maxDepth, FileVisitor