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.lead4pass.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

Given the following code fragment:

public static void getInfo() {

//insert code here

List fontCatalog = new ArrayList();

fontCatalog.add("Algerian");

fontCatalog.add("Cambria");

fontCatalog.add("Lucida Bright");

category.put("firstCategory",fontCatalog);

List entrySet = new ArrayList(category.entrySet());

Iterator it = entrySet.iterator();

while(it.hasNext())

{ System.out.println(it.next)

);

}

}

Which two code fragments, when inserted independently at line **, enable the code to compile?

A. Map category = new HashMap ();

B. Map category = new HashMap();

C. Map category = new HashMap ();

D. Map category = new HashMap ();

E. Map category = new HashMap ();

F. Map category = new HashMap ();

Correct Answer: CE

E: Redundant type arguments in new expressions. Use diamond operator instead.

QUESTION 2

Which two code blocks correctly initialize a Locale variable?

A. Locale loc1 = "UK";

B. Locale loc2 = Locale.get Instance ("ru");

C. Locale loc3 = Locale.getLocaleFactory("RU");

D. Locale loc4 = Locale.UK;

E. Locale loc5 = new Locale("ru", "RU");

Correct Answer: DE

Reference: The Java Tutorials,Creating a Locale

QUESTION 3

Given:

```
public class SampleClass {
public static void main(String[] args)
{ SampleClass sc = new
SampleClass(); sc.processCD();
}
private void processCD() {
try (CDStream cd = new CDStream()) {
cd.open();
cd.read();
cd.write("lullaby");
cd.close();
} catch (Exception e)
{ System.out.println("Exception
thrown");
}
}
class CDStream {
String cdContents = null;
public void open()
```

{ cdContents = "CD

Contents";

System.out.println("Opened CD stream");

}

public String read() throws Exception {

throw new Exception("read error");

}

public void write(String str)

{ System.out.println("CD str is: " +

str);

}

public void close() {

cdContents = null;

}

What is the result?

A. Compilation CD stream

B. Opened CD thrown

C. Exception thrown

D. Opened CD stream CD str is: lullaby

Correct Answer: A

In this example the compilation of line " try (CDStream cd = new CDStream()) {" will fail, as try-with-resources not applicable to variable type CDStream.

Note: The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after the program is finished with it. The try-with- resources statement ensures that each resource is closed at the end of the statement. Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

Reference: The Java Tutorials,The try-with-resources Statement

---

QUESTION 4

Given the code fragment:

try {

String query = "SELECT * FROM Employee WHERE ID=110";

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query); // Line 13

System.out.println("Employee ID: " + rs.getInt("ID")); // Line 14

} catch (Exception se)

{ System.out.println("Error"

);

}

Assume that the SQL query matches one record. What is the result of compiling and executing this code?

A. The code prints error.

B. The code prints the employee ID.

C. Compilation fails due to an error at line 13.

D. Compilation fails due to an error at line 14.

Correct Answer: B

Assuming that the connection conn has been set up fine, the code will compile and run fine.

Note#1: The GetInt method retrieves the value of the designated column in the current row of this ResultSet object as an int in the Java programming language.

Note 2: A table of data representing a database result set, which is usually generated by executing a statement that queries the database.

A ResultSet object maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The next method moves the cursor to the next row, and because it returns false when there are no more rows in the

ResultSet object, it can be used in a while loop to iterate through the result set.

A default ResultSet object is not updatable and has a cursor that moves forward only. Thus, you can iterate through it only once and only from the first row to the last row. It is possible to produce ResultSet objects that are scrollable and/or updatable. Reference: The Java Tutorials,Interface ResultSet

---

QUESTION 5

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

To Read the Whole Q&As, please purchase the Complete Version from Our website.

Try our product !

100% Guaranteed Success
100% Money Back Guarantee
365 Days Free Update
Instant Download After Purchase
24x7 Customer Support
Average 99.9% Success Rate
More than 800,000 Satisfied Customers Worldwide
Multi-Platform capabilities - Windows, Mac, Android, iPhone, iPod, iPad, Kindle

We provide exam PDF and VCE of Cisco, Microsoft, IBM, CompTIA, Oracle and other IT Certifications.
You can view Vendor list of All Certification Exams offered:

https://www.lead4pass.com/allproducts

Need Help

Please provide as much detail as possible so we can best assist you.
To update a previously submitted ticket: