Oracle

Exam 1z0-805

Upgrade to Java SE 7 Programmer

Verson: Demo

[Total Questions: 10]

Question No : 1

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");

Answer: D,E
Reference: The Java Tutorials, Creating a Locale

Question No : 2

Given the code fragment:

SimpleDateFormat sdf;

Which code fragment displays the three-character month abbreviation?

A. sdf = new SimpleDateFormat ("mm", Locale.UK);
System.out.println ("Result: " + sdf.format(new Date()));
B. sdf = new SimpleDateFormat (“MM”, Locale.UK);
System.out.println ("Result: " + sdf.format(new Date()));
C. sdf = new SimpleDateFormat (“MMM”, Locale.UK);
System.out.println ("Result: " + sdf.format(new Date()));
D. sdf = new SimpleDateFormat (“MMMM”, Locale.UK);
System.out.println ("Result: " + sdf.format(new Date()));

Answer: C
Explanation: C: Output example: Apr

Note: SimpleDateFormat is a concrete class for formatting and parsing dates in a locale-sensitive manner. It allows for formatting (date -> text), parsing (text -> date), and normalization.
SimpleDateFormat allows you to start by choosing any user-defined patterns for date-time formatting. However, you are encouraged to create a date-time formatter with either getTimeInstance, getDateInstance, orgetDateTimeInstance in DateFormat. Each of these

class methods can return a date/time formatter initialized with a default format pattern. You may modify the format pattern using the applyPattern methods as desired.

Question No : 3

Given the code fragment:

SimpleDateFormat sdf;

Which code fragment displays the two-digit month number?

A. sdf = new SimpleDateFormat ("mm", Locale.UK);
System.out.println (“Result: ” + sdf.format(new Date()))
B. sdf = new SimpleDateFormat ("MM", Locale.UK);
System.out.println (“Result: ” + sdf.format(new Date()))
C. sdf = new SimpleDateFormat ("MMM", Locale.UK);
System.out.println ("Result: " + sdf.format(new Date()))
D. sdf = new SimpleDateFormat ("MMMM", Locale.UK);
System.out.println ("Result: " + sdf.format(new Date()))

Answer: B

Explanation: B: Output example (displays current month numerically): 04

Note: SimpleDateFormat is a concrete class for formatting and parsing dates in a locale-sensitive manner. It allows for formatting (date -> text), parsing (text -> date), and normalization.
SimpleDateFormat allows you to start by choosing any user-defined patterns for date-time formatting. However, you are encouraged to create a date-time formatter with either getTimeInstance, getDateInstance, orgetDateTimeInstance in DateFormat. Each of these class methods can return a date/time formatter initialized with a default format pattern. You may modify the format pattern using the applyPattern methods as desired.

Question No : 4

Given:

```
class Fibonacci extends RecursiveTask<Integer> {

final int n;

Fibonacci (int n) { this.n = n }

Integer compute () {

if (n <= 1)

return n;

Fibonacci f1 = new Fibonacci (n – 1);

f1.fork;

Fibonacci f2 = new Fibonacci (n – 2);

return f2.compute() + f1.join;  // Line **

}

}
```

Assume that line ** is replaced with:

```
return f1.join() + f2.compute();  // Line **
```

What is the likely result?

A. The program produces the correct result, with similar performance to the original.
B. The program produces the correct result, with performance degraded to the equivalent of being single-threaded.
C. The program produces an incorrect result.
D. The program goes into an infinite loop.
E. An exception is thrown at runtime.
F. The program produces the correct result, with better performance than the original.

Answer: B

Explanation: Changing the code is not useful. In the original code (return f2.compute() + f1.join;) f1 and f2 are run in parallel. The result is joined.
With the changed code (return f1.join() + f2.compute();) f1 is first executed and finished, then is f2 executed.

Note 1: The join method allows one thread to wait for the completion of another.

If t is a Thread object whose thread is currently executing,
t.join();
causes the current thread to pause execution until t's thread terminates.

Note 2: New in the Java SE 7 release, the fork/join framework is an implementation of the ExecutorService interface that helps you take advantage of multiple processors. It is designed for work that can be broken into smaller pieces recursively. The goal is to use all the available processing power to enhance the performance of your application.

As with any ExecutorService, the fork/join framework distributes tasks to worker threads in a thread pool. The fork/join framework is distinct because it uses a work-stealing algorithm. Worker threads that run out of things to do can steal tasks from other threads that are still busy.

Reference: The Java Tutorials, Joins, Fork/Join

Question No : 5

Which code fragment is required to load a JDBC 3.0 driver?

A. DriverManager.loadDriver(“org.xyzdata.jdbc.network driver”);
B. Class.forname (“org.xyzdata.jdbc.NetWorkDriver”);
C. Connection con = Connection.getDriver
(“jdbc:xyzdata: //localhost:3306/EmployeeDB”);
D. Connection con = Drivermanager.getConnection
(“jdbc:xyzdata: //localhost:3306/EmployeeDB”);

Answer: B

Explanation: Note that your application must manually load any JDBC drivers prior to version 4.0.
The simplest way to load a driver class is to call the Class.forName() method.
If the fully-qualified name of a class is available, it is possible to get the corresponding Class using the static method Class.forName().

Question No : 6

Given:

```
class Fibonacci extends RecursiveTask<Integer> {
final int n;
Fibonacci (int n) { this.n = n }
Integer compute () {
if (n <= 1)
return n;
Fibonacci f1 = new Fibonacci (n – 1);
f1.fork;  // Line X
Fibonacci f2 = new Fibonacci (n – 2); // Line Y
return f2.compute() + f1.join;
}
}
```

Suppose that lines X and Y are transposed:

```
Fibonacci f2 = new Fibonacci (n – 2); // Line Y
f1.fork;  // Line X
```

What is the likely result?

A. The program produces the correct result, with similar performance to the original
B. The program produces the correct result, with performance degraded to the equivalent of being single-threaded.
C. The program produces an incorrect result
D. The program goes into an infinite loop
E. An exception is thrown at runtime
F. The program produces the correct result, the better performance than the original.

Answer: A

Explanation: The degree of parallelism is not changed. Functionality is the same.

Question No : 7

Given this code fragment:

```
ResultSet rs = null;

try (Connection conn = DriverManager. getConnection (url) ) {

Statement stmt = conn.createStatement();

rs  stmt.executeQuery(query);

//-.. other methods }

} catch  (SQLException se)  {

System.out.println  ("Error");

}
```

Which object is valid after the try block runs?

A. The Connection object only
B. The Statement object only
C. The Result set object only
D. The Statement and Result Set object only
E. The connection, statement, and ResultSet objects
F. Neither the Connection, Statement, nor ResultSet objects

Answer: C

Explanation: Generally, JavaScript has just 2 levels of scope: global and function. But, try/catch is an exception (no punn intended). When an exception is thrown and the exception object gets a variable assigned to it, that object variable is only available within the "catch" section and is destroyed as soon as the catch completes.

Question No : 8

Given the fragment:

```
public class CustomerApplication {

public static void main (String [] args) {

CustomerDAO custDao = new CustomerDAOMemoryImp1 ();

// . . . other methods

}

}
```

Which two valid alternatives to line 3 would decouple this application from a specific implementation of customerDAO?

A. CustomerDAO custDao = new customerDAO();
B. CustomerDAO custDao = (CustomerDAO) new object();
C. CustomerDAO custDao = CustomerDAO.getInstance();
D. CustomerDAO custDao = (CustomerDAO) new CustomerDAOmemoryImp1();
E. CustomerDAO custDao = CustomerDAOFactory.getInstance();

Answer: C,E

Explanation:

Note: Data Access Layer has proven good in separate business logic layer and persistent layer. The DAO design pattern completely hides the data access implementation from its clients. The interfaces given to client does not changes when the underlying data source mechanism changes. this is the capability which allows the DAO to adopt different access scheme without affecting to business logic or its clients. generally it acts as a adapter between its components and database. The DAO design pattern consists of some factory classes, DAO interfaces and some DAO classes to implement those interfaces.

Question No : 9

Which two descriptions are benefits of using PreparedStatement objects over static SQL in JDBC?

A. Conversion to native SQL
B. Supports BLOB types on every type of database

C. Prevention of SQL injection attacks
D. Improved performance from frequently run SQL queries
E. Built in support for multi database transaction semantics

Answer: A,D

Explanation: Sometimes it is more convenient to use a PreparedStatement object for sending SQL statements to the database. This special type of statement is derived from the more general class, Statement, that you already know.

If you want to execute a Statement object many times, it usually reduces execution time to use a PreparedStatement object instead.

The main feature of a PreparedStatement object is that, unlike a Statement object, it is given a SQL statement when it is created. The advantage to this is that in most cases, this SQL statement is sent to the DBMS right away, where it is compiled. As a result, the PreparedStatement object contains not just a SQL statement, but a SQL statement that has been precompiled. This means that when the PreparedStatement is executed, the DBMS can just run the PreparedStatement SQL statement without having to compile it first.

Although PreparedStatement objects can be used for SQL statements with no parameters, you probably use them most often for SQL statements that take parameters. The advantage of using SQL statements that take parameters is that you can use the same statement and supply it with different values each time you execute it.

Reference: The Java Tutorials, Using Prepared Statements

Question No : 10

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

Answer: A,C,D,E,G

Explanation: 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 the collections 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

Thank You For Trying 1Z0-805 PDF Demo

To try our 1Z0-805 Premium Files visit link below:

https://examsland.com/latest-1Z0-805-exam-questions/

Start Your 1Z0-805 Preparation

Use Coupon EL25 for extra 25% discount on the purchase of Practice Test Software.