Oracle

1Z0-805 Exam

Upgrade to Java SE 7 Programmer

Question: 1

Which statement is true about the take method defined in the WatchService interface?

A. Retrieves and removes the next watch key, or returns null of none are present.
B. Retrieves and removes the next watch key. If a queued key is not immediately available, the program waits for the specified wait time.
C. Retrieves and removes the next watch key: waits if no key is yet present.
D. Retrieves and removes all pending events for the watch key, returning a list of the events that were retrieved.

Answer: C

Explanation:
The WatchKey take() method retrieves and removes next watch key, waiting if none are yet present.
Note: A watch service that watches registered objects for changes and events. For example a file manager may use a watch service to monitor a directory for changes so that it can update its display of the list of files when files are created or deleted.
A Watchable object is registered with a watch service by invoking its register method, returning a WatchKey to represent the registration. When an event for an object is detected the key is signalled, and if not currently signalled, it is queued to the watch service so that it can be retrieved by consumers that invoke the poll or take methods to retrieve keys and process events. Once the events have been processed the consumer invokes the key's reset method to reset the key which allows the key to be signalled and re-queued with further events.
Reference: Interface WatchService

Question: 2

Given the code fragment:

```
private static void copyContents (File source, File target) {
try {inputStream fis = new FileInputStream(source);
outputStream fos = new FileOutputStream (target);
byte [] buf = new byte [8192];
int i;
while ((i = fis.read(buf)) != -1) {
fos.write (buf, 0, i);
}
 //insert code fragment here. Line **
System.out.println ("Successfully copied");
}
```

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

A.
```
} catch (IOException | NoSuchFileException e) {
System.out.println(e);
}
```

B. } catch (IOException | IndexOutOfBoundException e) {
System.out.println(e);
}
C. } catch (Exception | IOException | FileNotFoundException e) {
System.out.println(e);
}
D. } catch (NoSuchFileException e) {
System.out.println(e);
}
E. } catch (InvalidPathException | IOException e) {
System.out.println(e);
}

Answer: B, D, E

Explanation:
B: Two mutually exclusive exceptions. Will work fine.
D: A single exception. Will work fine.
E: Two mutually exclusive exceptions. Will work fine.
Note: In Java SE 7 and later, a single catch block can handle more than one type of exception. This feature can reduce code duplication and lessen the temptation to catch an overly broad exception.
In the catch clause, specify the types of exceptions that block can handle, and separate each exception type with a vertical bar (|).
Note 2: NoSuchFileException: Checked exception thrown when an attempt is made to access a file that does not exist.
InvalidPathException: Unchecked exception thrown when path string cannot be converted into a Path because the path string contains invalid characters, or the path string is invalid for other file system specific reasons.
FileNotFoundException: Signals that an attempt to open the file denoted by a specified pathname has failed.
This exception will be thrown by the FileInputStream, FileOutputStream, and RandomAccessFile constructors when a file with the specified pathname does not exist. It will also be thrown by these constructors if the file does exist but for some reason is inaccessible, for example when an attempt is made to open a read-only file for writing.
Incorrect answers:
A: This first exception is of type IOException; therefore, it catches any IOexception, including NoSuchFileException. This code will not compile.
C: This first exception is of type Exception; therefore, it catches any exception, including IOException and FileNotFoundException. This code will not compile.

Question: 3

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.

Answer: C, D

Explanation:
C: The method walkFileTree(Path start, Set<FileVisitOption> options, int maxDepth, FileVisitor<? super Path> visitor)
walks a file tree.
The maxDepth parameter is the maximum number of levels of directories to visit. A value of 0 means that only the starting file is visited, unless denied by the security manager. A value of MAX_VALUE may be used to indicate that all levels should be visited. The visitFile method is invoked for all files, including directories, encountered at maxDepth, unless the basic file attributes cannot be read, in which case the visitFileFailed method is invoked.
D: You need to decide whether you want symbolic links to be followed. If you are deleting files, for example, following symbolic links might not be advisable. If you are copying a file tree, you might want to allow it. By default, walkFileTree does not follow symbolic links.
Incorrect answers:
A: A file tree is walked depth first, but you cannot make any assumptions about the iteration order that subdirectories are visited.
B: The method visitFileFailed(T file, IOException exc) is invoked for a file that could not be visited. This method is invoked if the file's attributes could not be read, the file is a directory that could not be opened, and other reasons. However, there is no method named postVisitFileFailed.
Reference: The Java Tutorials, Walking the File Tree
Reference: walkFileTree

Question: 4

Which code fragments print 1?

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

Answer: A, C

Explanation:
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.
Incorrect answers:
B: The Array is of type char. The List is of type Integer. Incompatible types.
E: Type mismatch (Integer and char).

Question: 5

Given the code fragment:

```
public static void main(String[] args) {
String source = "d:\\company\\info.txt";
String dest = "d:\\company\\emp\\info.txt";
//insert code fragment here Line **
} catch (IOException e) {
System.err.println ("Caught IOException: " + e.getmessage();
}
}
```

Which two try statements, when inserted at line **, enable the code to successfully move the file info.txt to the destination directory, even if a file by the same name already exists in the destination directory?

A. try {FileChannel in = new FileInputStream(source).getChannel();
FileChannel out = new FileOutputStream(dest).getChannel ();
in.transferTo (0, in.size(), out);
B. try {Files.copy(Paths.get(source), Paths.get(dest));
Files.delete(Paths.get(source));
C. try {Files.copy(Paths.get(source), Paths.get(dest));
Files.delete(Paths.get(source));
D. try {Files.move (Paths.get(source), Paths.get(dest));
E. try {BufferedReader br = Files.newBufferedReader(Paths.get(source), Charset.forName ("UTF-8"));
BufferedWriter bw = Files.newBufferedWriter (Paths.get(dest), Charset.forName ("UTF-8"));
String record = "";
while ((record = br.readLine()) != null){
bw.write (record);
bw.newLine();
}
Files.delete(Paths.get(source));

Answer: B, D

Explanation:
Incorrect answers:
A: Copies (not moving) the file, but the original file is left.
C: Moves a file, but only if the destination does not exist.
E. The file is moved fine, but the content of the file is lost.

Question: 6

What design pattern does the Drivermanager.getconnection () method characterize?

A. DAO
B. Factory
C. Singleton
D. composition

Answer: B

Explanation:
DriverManager has a factory method getConnection() that returns a Connection object.
Note 1: A factory method is a method that creates and returns new objects.
The factory pattern (also known as the factory method pattern) is a creational design pattern. A factory is a Java class that is used to encapsulate object creation code. A factory class instantiates and returns a particular type of object based on data passed to the factory. The different types of objects that are returned from a factory typically are subclasses of a common parent class.
Note 2:
The method DriverManager.getConnection establishes a database connection. This method requires a database URL, which varies depending on your DBMS. The following are some examples of database URLs: MySQL, Java DB.

Question: 7

Given the code fragment:
dataFormat df;
Which statement defines a new DataFormat object that displays the default date format for the UK Locale?

A. df = DateFormat.getDateInstance (DateFormat.DEFAULT, Locale(UK));
B. df = DateFormat.getDateInstance (DateFormat.DEFAULT, UK);
C. df = DateFormat.getDateInstance (DateFormat.DEFAULT, Locale.UK);
D. df = new DateFormat.getDataInstance (DataFormat.DEFAULT, Locale.UK);
E. df = new DateFormat.getDateInstance (DateFormat.DEFAULT, Locale(UK));

Answer: C

Explanation:
DateFormat is an abstract class that provides the ability to format and parse dates and times. The getDateInstance() method returns an instance of DateFormat that can format date information. It is available in these forms:
static final DateFormat getDateInstance( )
static final DateFormat getDateInstance(int style)
static final DateFormat getDateInstance(int style, Locale locale)
The argument style is one of the following values: DEFAULT, SHORT, MEDIUM, LONG, or FULL. These are int constants defined by DateFormat.

Incorrect answers:
A, B, E: Incorrect syntax for the Locale. The correct syntax is: Locale.UK
D: Incorrect syntax.

Question: 8

Given three resource bundles with these values set for menu1: (The default resource bundle is written in US English.)
English US resource Bundle
Menu1 = small
French resource Bundle
Menu1 = petit
Chinese Resource Bundle
Menu = 1
And given the code fragment:
Locale.setDefault (new Locale("es", "ES")); // Set default to Spanish and Spain
loc1 = Locale.getDefault();
ResourceBundle messages = ResourceBundle.getBundle ("messageBundle", loc1);
System.out.println (messages.getString("menu1"));
What is the result?

A. No message is printed
B. petit
C. :
D. Small
E. A runtime error is produced

Answer: E

Explanation:
Compiles fine, but runtime error when trying to access the Spanish Resource bundle (which does not exist):
Exception in thread "main" java.util.MissingResourceException: Can't find bundle for base name messageBundle, locale es_ES

Question: 9

Given:
import java.util.*;
public class StringApp {
public static void main (String [] args) {
Set <String> set = new TreeSet <> ();
set.add("X");
set.add("Y");
set.add("X");
set.add("Y");
set.add("X");

```
Iterator <String> it = set.iterator ();
int count = 0;
while (it.hasNext()) {
switch (it.next()){
case "X":
System.out.print("X  ");
break;
case "Y":
System.out.print("Y  ");
break;
}
count++;
}
System.out.println ("\ncount = " + count);
}
}
```

What is the result?

A. X X Y X Y
count = 5
B. X Y X Y
count = 4
C. X Y
count = s
D. X Y
count = 2

Answer: D

Explanation:
A set is a collection that contains no duplicate elements. So set will include only two elements at the start of while loop. The while loop will execute once for each element. Each element will be printed.
Note:
* public interface Iterator
An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:
Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
Method names have been improved.
* hasNext
public boolean hasNext()
Returns true if the iteration has more elements. (In other words, returns true if next would return an element rather than throwing an exception.)
* next
public Object next()
Returns the next element in the iteration.

Question: 10

Given the code fragment:
List<Person> pList = new CopyOnWriteArrayList<Person>();
Which statement is true?

A. Read access to the List should be synchronized.
B. Write access to the List should be synchronized.
C. Person objects retrieved from the List are thread-safe.
D. A Person object retrieved from the List is copied when written to.
E. Multiple threads can safely delete Person objects from the List.

Answer: C

Explanation:
CopyOnWriteArrayList produces a thread-safe variant of ArrayList in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array.
Note: his is ordinarily too costly, but may be more efficient than alternatives when traversal operations vastly outnumber mutations, and is useful when you cannot or don't want to synchronize traversals, yet need to preclude interference among concurrent threads. The "snapshot" style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, so interference is impossible and the iterator is guaranteed not to throw ConcurrentModificationException. The iterator will not reflect additions, removals, or changes to the list since the iterator was created. Element-changing operations on iterators themselves (remove, set, and add) are not supported. These methods throw UnsupportedOperationException.
All elements are permitted, including null.
Memory consistency effects: As with other concurrent collections, actions in a thread prior to placing an object into a CopyOnWriteArrayList happen-before actions subsequent to the access or removal of that element from the CopyOnWriteArrayList in another thread.
Reference: java.util.concurrent.CopyOnWriteArrayList<E>