Domain : Secure Coding in Java SE Application

Q1 : To which of the following attacks is this code vulnerable?

```
public boolean getEmployee(Integer id) throws SQLException {

var sql = “SELECT * FROM emp WHERE id = ?”;

try (var stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, id);

try (var rs = stmt.executeQuery(sql)) {

return rs.next();

}

}

}
```

A. Leaking Resources

B. Leaking Confidential data

C. DoS

D. SQL injection

E. None of these

Correct Answer: E

Explanation

Choice E is the correct answer. this code is not vulnerable to any of these attacks. This is a trick question that might appear in the exam.

Choice A is incorrect. The usage of tryWithResources statement ensures that PreparedStatement and ResultSet objects are closed automatically and hence there is no leakage of resources.

Confidential data leak can happen if sensitive data is logged in log files, exception messages etc. **Choice B is incorrect** because no confidential data leaking is possible on this code.

Denial-of-Service (DoS) attack is an explicit attempt to prevent legitimate users from using a service by hackers. Such an attack typically launched by sending continuous requests to the server for a particular web resource. **Choice C is also incorrect** because there is no Denial of Service attack possible here.

SQL injection is a common attack that consists of insertion of SQL code via input data from the client application. **Choice D is incorrect** because SQL injection is not applicable here because of the proper use of PreparedStatement with bind variables here.

Reference:
https://www.oracle.com/java/technologies/javase/seccodeguide.html

Domain : Annotations

Q2 : Which of the following are true about annotations?

A. Annotation names are not case sensitive

B. Annotations always contain elements

C. Annotations can be applied to classes, methods, expressions, and annotations

D. When using a marker annotation, parentheses are optional

E. None of these

F. All of these

Correct Answers: C and D

Explanation

Annotations, a form of metadata, provide data about a program that is not part of the program itself. Annotations have no direct effect on the operation of the code they annotate.

Choice C is correct. Annotations can be applied to declarations: declarations of classes, fields, methods, and other program elements.

Choice D is also correct. If the annotation has no elements, then the parentheses can be omitted. Such annotations are called marker annotations.

Choice A is incorrect. Annotation names are case sensitive.

Choice B is incorrect. An annotation can have elements, these look like methods. However, these are optional. The only purpose is to mark a declaration and hence are called marker annotations.

References:
https://docs.oracle.com/javase/tutorial/java/annotations/basics.html,
https://docs.oracle.com/javase/tutorial/java/annotations/declaring.html

Domain : Java I/O API

Q3 : Which code fragment when inserted at line 3 will produce the output as "../../a.txt"

```
1 var path1 = Path.of("a.txt");

2 var path2 = Path.of("b/c.txt");

3 // insert code here
```

A. System.out.println(path1.relativize(path2));

B. System.out.println(path2.relativize(path1));

C. System.out.println(path2.normalize(path1));

D. System.out.println(path1.normalize(path2));

E. None of these

Correct Answer: B

Explanation

The relativize(Path) method constructs a relative path between the current path and a given path. If both path values are relative, then the relativize() method computes the paths

as if they are in the same current working directory. Alternatively, if both path values are

absolute, then the method computes the relative path from one absolute location to another,

regardless of the current working directory.

Choice B is correct. To get to a.txt from the current path of b/c.txt , you need to go up two levels (the file itself counts as one level) and then select a.txt. The output of choice B is "../../a.txt". **Thus choice B is correct and E is incorrect.**

Calling relativize() is on path2 will get to the path b/c.txt from the current path of a.txt, resulting in the output "../b/c.txt". This is not the expected result and hence **choice A is incorrect.**

The normalize() method is invoked on a Path object to eliminate unnecessary redundancies in a path. An empty path is returned if this path does not have a root component and all name elements are redundant. This method takes no arguments. Hence choices and D will cause compiler errors. **Thus choices C and D are incorrect.**

References:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/nio/file/Path.html, https://docs.oracle.com/javase/tutorial/essential/io/pathOps.html

Domain : Localization

Q4 : What is the result of compiling and running this code?

double d = 1234567.890;

NumberFormat f2 = new DecimalFormat(“$000,000,000.00000”);

System.out.println(f2.format(d));

A. Does not compile

B. Throws exception

C. Prints $001,234,567.89000

D. Prints $1,234,567.89

E. Prints a different result

Correct Answer: C

Explanation

The format method of DecimalFormat accepts a double value as an argument and returns the formatted number in a String. The pattern parameter passed to the DecimalFormat constructor is the number pattern that numbers should be formatted according to.

There are 11 Special Pattern Characters, but the most important are:

0 – prints a digit if provided, 0 otherwise

hash – prints a digit if provided, nothing otherwise

. – indicate where to put the decimal separator

, – indicate where to put the grouping separator

In the code fragment given, “$000,000,000.00000” is the format string given. So the output should have a length of 9 digits before the decimal point and 5 digits after that. To the number 1234567.890, the format method adds leading and trailing zeros to make the output the desired length, as 0 is used as the pattern character. Also, a dollar sign is prefixed, as the pattern starts with it. Hence **choice C is correct.**

If hash was used instead of 0 in the pattern, 1234567.89 would have no trailing or leading zeros appended to it and D would have been correct. Hence, in this case, **option D is incorrect.**

Options A and B are incorrect because there are no such errors. As **C is correct, choice E is automatically incorrect.**

Reference:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/text/DecimalFormat.html

Domain : Exception Handling

Q5 : Which feature in Java is an attempt to reduce the number of NullPointerExceptions?

A. Optional

B. Enum

C. Predicate

D. Annotation

E. Module

F. None of these

Correct Answer: A

Explanation

Java 8 Optional class allows representing optional values instead of null references. This reduces the possibility of NullPointerExceptions. Optional can be considered as a single-value container that either contains a value or doesn't. The advantage of using Optional as compared to null references is that the Optional class forces you to think about the case when the value is not present. As a consequence, you can prevent unintended null pointer exceptions. Thus **choice A is correct and F is incorrect.**

An enum type is a special data type that enables for a variable to be a set of predefined constants. The variable must be equal to one of the values that have been predefined for it. It has nothing to do with NullPointerException and hence **choice B is incorrect.**

Predicate is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference. Hence **C is also incorrect.**

Annotations, a form of metadata, provide data about a program that is not part of the program oo.itself. Annotations have no direct effect on the operation of the code. So **D is incorrect** too.

.A Java module is a packaging mechanism that enables you to package a Java application or Java API as a separate Java module. So **E is incorrect** too.

Reference:
https://www.oracle.com/technical-resources/articles/java/java8-optional.html

Domain : Working with Arrays and Collections

Q6 : What will be the output of this code?

```
Set<String> colors = new HashSet<>(); //line 1

colors.add("yellow"); //line 2

colors.add("Blue"); //line 3

colors.sort((a1, a2) -> a1.compareTo(a2)); //line 4

System.out.println(colors);
```

A. Prints [yellow, Blue]

B. Exception at runtime

C. Prints [Blue, yellow]

D. Compiler error at line 1

E. Compiler error at line 4

F. None of these

Correct Answer: E

Explanation

A Set is an unordered collection with no duplicate elements. As a HashSet does not maintain the order of its elements, sorting of HashSet is not possible. However, a List is a sortable collection, which takes a Comparator, as the argument. A Comparator is an object that defines a compare() method that can be used to compare two objects, this defines the sort order. However, there is no such sort() method in Set.

Choice E is correct. As the sort() method is not defined for Set implementations, line 4 in the given example does not compile. The compiler complains that the sort() method is undefined.

Choice A is incorrect. As there is a compiler error in line 4, nothing is printed.

Choice B is incorrect. As the program has a compiler error, a runtime exception cannot be thrown.

Choice C is incorrect because the program cannot be executed due to the compiler error. If the sample code had List and ArrayList instead of Set and HashSet, the output would have been [Blue, yellow].

Choice D is incorrect. The diamond operator was introduced in Java 7 to simplify instantiation of generic classes. When the diamond <> operator is used on the right side as in line 1. the compiler can infer that the class instantiated is to have the same type as the variable it is assigned to. Hence there is no compiler error in line 1.

Reference:
https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html

Domain : Working with Streams and Lambda expressions

Q7 : When will the Student object created on line 3 become eligible for garbage collection?

```
public class Student { // line 1

  public static void main(String[] args) { // line 2

    Student one = new Student(); // line 3

    Student two = one; // line 4

    Student three = two; // line 5

    one = null; // line 6

    Student four = one; // line 7

    two = null; // line 8

    three = new Student(); // line 9

    System.gc(); // line 10

  } // Line 11

}
```

A. After line 6

B. After line 7

C. After line 8

D. After line 9

E. After line 10

F. After line 11

G. None of these

Correct Answer: D

Explanation

The Student object from line 3 has three references to it: one, two and three. The references one and two are set to null on lines 6 and 8, respectively. The reference three is made to point to a new Student object in line 9. Hence, there will be no more references after line 9 and is thus eligible for GC (garbage collection). Hence, **choice D is correct.**

Choice A is incorrect because only the reference one is set to null after line 6. The references two and three still point to the object and hence it cannot be garbage collected.

Choice B is incorrect because the reference four is set to null in line 7 and this has no effect on GC.

Choice C is incorrect because only the references one and two are set to null by line 8, the reference three still points to the object and hence it cannot be garbage collected.

As no more references exist for the object after line 9, the object becomes eligible for GC that time itself. This makes **choices E, F and G incorrect.** Also, note that calling System.gc() has no effect on eligibility for garbage collection.

References:
https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html, https://javarevisited.blogspot.com/2011/04/garbage-collection-in-java.html#axzz6p6PJImg8

Domain : Exception Handling

Q8 : What will be the result?

```
public static void main(String[] args) {

    try {

        FileReader fileReader = new FileReader("c:\\data\\input-text.txt");

        int data = fileReader.read();
```

```
    } catch (IOException | IllegalStateException ex) {
        ex = null;
    }
}
```

A. Does not compile because FileReader is not closed after use

B. Does not compile because multiple exceptions cannot be caught in a single catch block

C. Does not compile because IllegalStateException does not extend IOException

D. Does not compile because ex cannot be reassigned to anything

E. Does not compile because null value cannot be assigned to any exception variable

F. Runs without any errors

G. Throws an exception at runtime

Correct Answer: D

Explanation

Java provides a feature named multi-catch blocks in which you can combine multiple catch handlers. The catch clauses of multiple exceptions can be combined using a single pipe symbol (|). : If a catch block handles more than one exception type, then the catch parameter is implicitly final, and thus it cannot be reassigned to anything. Hence ex=null assignment does not compile. Thus, **choice D is correct.**

Ideally, resources such as FileReader must be closed after use to prevent any resource leak. However, this does not cause any error while compiling. Hence, **choice A is incorrect.**

As multiple exceptions can be handled in a multi-catch block, **choice B is incorrect.** In a multi-catch block, you cannot combine catch handlers for two

exceptions that share a base- and derived-class relationship. Hence, **choice C is incorrect.**

Choice E is incorrect because there is no such rule that a null value cannot be assigned to an exception variable.

As **choice D is correct, choices F and G are also incorrect.**

Reference:
https://docs.oracle.com/javase/7/docs/technotes/guides/language/catch-multiple.html

Domain : S2- Controlling Program Flow

Q9 : What is the result of the following code snippet?

```
public class Switch1 {

private static final String APPLE = “APPLE”;

private static String mango;

public static void main(String[] args) {

String fruit = “Berry”;

mango = “Mango”;

int i = 0;

switch (fruit) {

case “Mango”:

break;

case APPLE:

i++;
```

default:

i++;

case “VIOLIN”:

i++;

case “BERRY”:

++i;

break;

}

System.out.print(i);

}

}

A. 0

B. 1

C. 2

D. Throws an exception

E. 3

F. The code does not compile

Correct Answer: E

Explanation

The switch statement is used where there are a number of possible execution paths. It works with the byte, short, char, and int primitive data types. It also works with enumerated types,the String class, and some wrapper classes.

- In the case of String expressions, the comparison is case sensitive like the String.equals method. Hence, the value “Berry” does not match the case “BERRY” or any other case. As a result, the default case is executed, which first increments i to 1. As there is no break statement, the next two cases are also executed, which result in i being incremented twice more. Thus the value of i is printed as 3. Thus **option E is correct** and the other options are incorrect.
- As String is a valid expression type in the switch statement. Also, case expressions must be constants. APPLE is a constant as it is declared as final. Hence, there are no compiler errors or exceptions. So **options D and F are incorrect.**

References:
https://docs.oracle.com/javase/8/docs/technotes/guides/language/strings-switch.html, https://docs.oracle.com/javase/tutorial/java/nutsandbolts/switch.html

Domain : Secure Coding in Java SE Application

Q10 : What is the security issue in the given code?

```
public class LibClass {

transient boolean flag = false;

private static final String FILESEPARATOR = “file.separator”;

public static String getPropValue() {

return AccessController.doPrivileged(new PrivilegedAction<String>() {

public String run() {

return System.getProperty(FILESEPARATOR);

}

});

    }

    public static void main(String args[]) throws Exception {
```

```
try (var ois = new ObjectOutputStream(new BufferedOutputStream(new
FileOutputStream("hello.txt")))) {

ois.writeObject("Hello");

}

System.out.println(getPropValue());

   }

}
```

A. Code is not secure because reading a system property will always result is loss of sensitive data

B. Code is not secure because the system property is retrieved using a hard-coded value

C. Code is not secure because the variable flag is declared transient

D. Code is not secure because the resources are not closed after use

E. There is no such security issue in the code

Correct Answer: E

Explanation

Option E is correct. There is no security issue in the code. No sensitive data is leaked, resources left open or there is any vulnerability.

Reading a system property might be necessary at times and in such cases privileged code can be given access only to that specific property, Thus sensitive data can be protected. Hence **option A is incorrect.**

For safety reasons, the inputs passed to doPrivileged must be restricted to a limited set of acceptable (usually hard-coded) values. Instead of allowing the code to access any system property, only the given (hard-coded) system property can be accessed in this code. As this is secure, **option B is also incorrect.**

Specifying transient variables does not cause any insecurity. In fact, it is advisable to keep sensitive data transient to prevent it from being serialized. Hence **option C is also incorrect.**

When closing chained/wrapped streams, we need to close only the outermost stream. In the given code, ObjectOutputStream is the outermost stream. This and the connected streams are automatically closed when the try-with-resources block ends. Hence, **option D is also incorrect.**

Reference:
https://www.oracle.com/java/technologies/javase/seccodeguide.html

Domain : Annotations

Q11 : Which annotation can be used to indicate that a method may be removed in a future release?

A. @Retention

B. @SuppressWarnings

C. @Deprecated

D. None of these

Correct Answer: C

Explanation

@Deprecated annotation can be used to indicate that the marked element (module, class, method, or member) should no longer be used. From Java 9, two optional attributes got added to the @Deprecated annotation: since and forRemoval. The since attribute defines the Java version in which the element was marked deprecated first. The default value is an empty string. The forRemoval attribute (default value is false) can be specified as true if the element will be removed in the next release. Hence **option C is correct.**

@Retention annotation is used to indicate how long annotations with the annotated type are to be retained. The possible values are below.

SOURCE Used only in the source file, discarded by the compiler

CLASS Stored in the .class file but not available at runtime (default compiler behavior)

RUNTIME Stored in the .class file and available at runtime

As this does not indicate anything about the use of an element in the future versions, **option A is incorrect.**

The @SuppressWarnings annotation Indicates that the named compiler warnings should be suppressed in the annotated element (and in all program elements contained in the annotated element). Hence **option B is also incorrect.**

Reference:
https://docs.oracle.com/en/java/javase/11/core/enhanced-deprecation1.html

Domain : Concurrency

Q12 : What is the expected result of executing the following code?

```
ExecutorService service = null;

Runnable task2 = () -> {

for (int i = 0; i < 3; i++) {

    try {

       Thread.sleep(3000);

    } catch (InterruptedException e) {

     e.printStackTrace();

    }

}

};
```

```
try {
   service = Executors.newSingleThreadExecutor();
   service.execute(task2);
   service.execute(task2);
   service.execute(task2);
 } finally {
   if (service != null)
      service.shutdown();
  }
service.awaitTermination(2, TimeUnit.SECONDS);
System.out.println("Done");
}
```

A. It will immediately print "Done"

B. It will wait for 2 seconds and then print "Done"

C. It will wait for 3 seconds and then print "Done"

D. It will wait for 5 seconds and then print "Done"

E. It will wait for 9 seconds and then print "Done"

F. Code causes compiler error

G. It will throw an exception at runtime

Correct Answer: B

Explanation

The awaitTermination() method waits the specified time until all tasks have completed execution, returning earlier if all tasks finish or an InterruptedException is detected.

In the main thread, three tasks are submitted to an ExecutorService. Then the ExecutorService is shut down. Each task takes at least 3 seconds to complete (Thread.sleep() is called for 3 seconds). As the awaitTermination() method is invoked passing 2 seconds, it will return with a value of false after two seconds. Hence, **option B is correct and options A, C, D and E are incorrect.**

There is no compiler error or exception thrown. Hence, **options F and G are incorrect.**

Reference:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/ExecutorService.html

Domain : Secure Coding in Java SE Application

Q13 : Which of the following are needed to be used together to prevent SQL Injection attacks while executing SQL queries?

1. Use of CallableStatement
2. Use of PreparedStatement
3. Passing user-supplied data as bind variables
4. Passing user-supplied data concatenated to SQL query

Correct Answers: B and C

Explanation

SQL injection is a hacking technique used to exploit an application's vulnerability by passing user supplied data as part of an SQL query. To prevent this, PreparedStatement needs to be used by replacing the values in the bind variables ("?") within the query with user supplied data. Hence, **options B and C are correct.**

Unsanitized user data should never be concatenated with the query, hence **option D is incorrect.**

Callable statement is used to execute stored procedures and not for SQL queries as specified in the question. Hence, **option A is also incorrect.**

Reference:
https://www.oracle.com/java/technologies/javase/seccodeguide.html

Domain : Java I/O API

Q14 : Given

```
class Test {
    public static void main(String[] args) throws IOException {
        Path root = Path.of("root");
        Path b = Path.of("root/a/b");
        Path c = Path.of("root/c");
        Files.createDirectories(b);
        Files.createDirectory(c);
        Files.walk(root).forEach(System.out::println);
    }
}
```

Which of the following can be the given program's output?

A. b c

B. a a/b c

C. a c a/b

D. root/a root/c root/a/b

E. root root/a root/a/b root/c

F. root root/a root/c root/a/b

Correct Answer: E

Explanation

The Files class includes two methods for walking the directory tree using a depth-first search.

public static Stream<Path> walk(Path start, FileVisitOption… options) throws IOException

public static Stream<Path> walk(Path start, int maxDepth, FileVisitOption… options) throws IOException

The static method walk() uses lazy evaluation and evaluates a Path only as it gets to it.

Here's a description of the walk method from the Java SE API Specification:

Return a Stream that is lazily populated with Path by walking the file tree rooted at a given starting file. The file tree is traversed depth-first, the elements in the stream are Path objects that are obtained as if by resolving the relative path against start.

The stream walks the file tree as elements are consumed. The Stream returned is guaranteed to have at least one element, the starting file itself.

In the given program, the *root* directory must be present in the output. Hence, **option E is correct.**

As the root directory is not present, **options A, B, C and D are incorrect.**

Option F is also incorrect as the path *root/a/b* must be right behind the path *root/a* due to depth-first traversal.

Reference:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/nio/file/Files.html#walk(java.nio.file.Path,java.nio.file.FileVisitOption…)

Domain : S2- Controlling Program Flow

Q15 : Given:

```
int i, j;

for (i = j = 0; ; ++i, j–) {

   if (i – j > 10) {

        break;

    }

}

System.out.println(i + ” ” + j);
```

What is the output of the given code?

A. 6 -6

B. 5 -5

C. 4 -4

D. It runs into an infinite loop

E. Compilation fails

Correct Answer: A

Explanation

The for statement is used to iterate over a range of values. It repeatedly loops until a particular condition is satisfied. The general form of the for statement can be expressed as follows:

for (initialization; termination;

```
    increment) {

   statement(s)

}
```

There is nothing wrong with the given code, hence it compiles without any issues. The variables i and j are initialized to 0. The body of the for construct keeps running until i – j > 10. This happens when variables i and j reach 6 and -6, respectively. At this point, the break statement is executed and the for construct exits. Hence, **option A is correct** and the others are incorrect.

Reference:
https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

Domain : Working with Streams and Lambda expressions

Q16 : Which of the following descriptions about streams is false?

A. A stream is not a data structure that stores elements

B. An operation on a stream produces a result, but does not modify its source

C. Many stream operations can be implemented lazily

D. While collections have a finite size, streams need not

E. An element in a stream can be visited more than once to optimize operations

F. None of the above

Correct Answer: E

Explanation

Here's an excerpt from the Java 11 API Specification:

Streams differ from collections in several ways:

No storage. A stream is not a data structure that stores elements; instead, it conveys elements from a source such as a data structure, an array, a generator function, or an I/O channel, through a pipeline of computational operations. Hence, the description in option A is true, which makes **option A incorrect.**

Functional in nature. An operation on a stream produces a result, but does not modify its source. For example, filtering a Stream obtained from a collection produces a new Stream without the filtered elements, rather than removing elements from the source collection. Hence, the description in option B is true, which makes **option B incorrect.**

Laziness-seeking. Many stream operations, such as filtering, mapping, or duplicate removal, can be implemented lazily, exposing opportunities for optimization. For example, “find the first String with three consecutive vowels” need not examine all the input strings. Stream operations are divided into intermediate (Stream-producing) operations and terminal (value- or side-effect-producing) operations. Intermediate operations are always lazy. Hence, the description in option C is true, which makes **option C incorrect.**

Possibly unbounded. While collections have a finite size, streams need not. Short-circuiting operations such as limit(n) or findFirst() can allow computations on infinite streams to complete in finite time. Hence, the description in option D is true, which makes **option D incorrect.**

Consumable. The elements of a stream are only visited once during the life of a stream. Like an Iterator, a new stream must be generated to revisit the same elements of the source. Hence, the description in option E is false, which makes option **E the correct option.**

Reference:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/stream/package-summary.html

Domain : Java Object-Oriented Approach

Q17 : Given:

```
class SuperTest {

   public Object myMethod(Object… args) {
```

```
        // A valid body
    }
}
class Test extends SuperTest {
    // Method 1
    public Object myMethod(String… args) {
        // A valid body
    }
    // Method 2
    public Object myMethod(Integer[] args) {
        // A valid body
    }
    // Method 3
    public Object myMethod(Object arg) {
        // A valid body
    }
    // Method 4
    public String myMethod(Object[] args) {
        // A valid body
    }
}
```

Which method in the Test class doesn't overload the only method in the SuperTest class?

A. Method 1

B. Method 2

C. Method 3

D. Method 4

E. None of the above

Correct Answer: D

Explanation

As per the Oracle Java Specification, an instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass overrides the superclass's method.

Only the method in option D has the same name, number and type of parameters, and return type as the superclass method, hence this is correct. It is important to note that varargs is just syntactic sugar for arrays, hence method 4 overrides the super-class's method.

The first three methods of the Test class don't have the same parameters as the method in the SuperTest class, hence they don't override. Hence, the other options are incorrect.

Reference: https://docs.oracle.com/javase/tutorial/java/IandI/override.html

Domain : Concurrency

Q18 : Which of the following is NOT true about Runnable and Callable?

A. Both Runnable and Callable can be used to construct a Thread object

B. A Callable task can throw a checked exception, while a Runnable task cannot

C. An ExecutorService can execute a collection of Callable tasks at once

D. When submitting a Runnable, an ExecutorService returns a Future instance

E. None of the above

Correct Answer: A

Explanation

The Thread class doesn't have a constructor that accepts a Callable argument; hence, option A is false and hence the correct answer.

The Callable.call method specifies the Exception class in its declaration; thus, it can throw any checked exception. In contrast, the Runnable.run method doesn't specify any exception class and cannot throw a checked exception. Therefore, option B is true and hence incorrect.

The invokeAll() method of ExecutorService executes the given Callable tasks, returning the result of one that has completed successfully. The submit() method of ExecutorService Submits a Runnable task for execution and returns a Future representing that task. Thus, we can see that Options C and D are true and hence incorrect as per the invokeAll and submit() methods defined in the ExecutorService interface.

References:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Thread.html,
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Runnable.html#run(),
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/Callable.html#call(),
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/concurrent/ExecutorService.html

Domain : Java I/O API

Q19 : Given:

Path p = Path.of("test.txt");

Files.writeString(p, “Hello”); // Line 1

Files.writeString(p, “Goodbye”); // Line 2

System.out.println(Files.readString(p));

Suppose the file “test.txt” didn’t exist. What is the given code fragment’s output?

A. Hello

B. Goodbye

C. Hello Goodbye

D. An exception is thrown on line 1

E. An exception is thrown on line 2

Correct Answer: B

Explanation

The writeString(Path, String) is used to write to a file. It takes the Path of the file and a String which is to be written into the file. It has optional parameters such as charset and open option.

When the Files.writeString method is called the first time, a file with the specified name is created and contains the string “Hello”. Subsequently, the second invocation of that method

replaces the existing content with the new string. Therefore, there’s no exception, the final string inside the given file is “Goodbye”. Thus **option B is correct** and the others are incorrect.

Reference:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/nio/file/Files.html#writeString(java.nio.file.Path,java.lang.CharSequence,java.nio.file.OpenOption…)

Domain : Java Platform Module System

Q20 : Which of the following isn't a valid option of the jdeps command?

A. –generate-module-info

B. –generate-open-module

C. –check-deps

D. –list-deps

E. –list-reduced-deps

F. –print-module-deps

Correct Answer: C

Explanation

The jdeps command is used to launch the Java class dependency analyzer.

This is how he command is used.

jdeps [options] path …

Here are descriptions of some options of the jdeps command:

–generate-module-info dir

Generates module-info.java under the specified directory. The specified JAR files will be analyzed. This option cannot be used with –dot-output or –class-path options. Use the –generate-open-module option for open modules.

–generate-open-module dir

Generates module-info.java for the specified JAR files under the specified directory as open modules. This option cannot be used with the –dot-output or –class-path options.

–check module-name [, module-name…]

Analyzes the dependencies of the specified modules. It prints the module descriptor, the resulting module dependencies after analysis, and the graph after transition reduction. It also identifies any unused qualified exports.

–list-deps

Lists the module dependencies and also the package names of JDK internal APIs (if referenced).

–list—reduced-deps

Same as –list-deps without listing the implied reads edges from the module graph. If module M1 reads M2, and M2 requires transitive on M3, then M1 reading M3 is implied and is not shown in the graph.

–print-module-deps

Same as –list-reduced-deps with printing a comma-separated list of module dependencies. The output can be used by jlink –add-modules to create a custom image that contains those modules and their transitive dependencies.

Reference: https://docs.oracle.com/en/java/javase/11/tools/jdeps.html

Domain : Localization

Q21 : Given

DateFormat formatter = new SimpleDateFormat(“Timezone zz”);

formatter.setTimeZone(TimeZone.getTimeZone(“PST”));

Date date = new Date();

String output = formatter.format(date);

System.out.println(output);

What is the output of the given code?

A. Timezone -0700

B. Timezone -07:00

C. Timezone PDT

D. Timezone Pacific Daylight Time

E. An IllegalArgumentException is thrown

Correct Answer: E

Explanation

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.

When you create a SimpleDateFormat object, a pattern String is specified, whose contents determine the format of the date and time. In a date-time format pattern, all letters (from “A” to “Z” and from “a” to “z”) are reserved. If we want to include letters in a formatted string, we must put them in single quotes, such as ‘Timezone’.

In the given code, the word Timezone isn’t escaped, hence the program attempts to parse its letters. Since “T”, the first letter in the word, isn’t a predefined pattern letter, the program fails with an IllegalArgumentException.

If the pattern string had been escaped correctly, option C would have been the correct answer.

Reference:
https://docs.oracle.com/javase/tutorial/i18n/format/simpleDateFormat.html#datepattern

Domain : Java Platform Module System

Q22 : Given a module named service, which contains a service interface called com.acme.spi.MyService. Its service provider, com.acme.MyServiceImpl, is enclosed in another module called impl.

Which of the following is a correct declaration of the service module?

A. module service { exports com.acme.spi; }

B. module service { requires impl; }

C. module service { exports com.acme.spi; requires impl; }

D. module service { exports com.acme.spi; uses com.acme.MyServiceImpl; }

E. module service { exports com.acme.spi.MyService; provides com.acme.MyServiceImpl; }

F. module service { exports com.acme.spi.MyService; requires impl; uses com.acme.MyServiceImpl; }

Correct Answer: A

Explanation

A service is composed of an interface, the classes the interface references, and a way of looking up implementations of the interface. The service provider interface specifies what behavior the service will have. A service locator is able to find the classes that implement a service provider interface.

When using a service loader, the service interface and its clients know nothing about service providers. Therefore, the service module shouldn't have any information about the impl module as well as the enclosed service provider. Option A is correct as it does not require any module.

Options B and C are incorrect as they require the impl module. In addition, the declaration in option B doesn't even export the service interface for public use.

Options D, E and F are incorrect since the directives are incorrectly used.

Reference:
http://http://openjdk.java.net/projects/jigsaw/spec/sotms/#services

Domain : Exception Handling

Q23 : Given:

```
public class MyResource implements AutoCloseable {
   public void open() throws IOException {
       throw new IOException("open");
   }
   public void close() {
       throw new ArithmeticException("close");
   }
}
```

And:

```
public class Test {
   public static void main(String[] args) throws IOException {
       try (MyResource myResource = new MyResource()) {
           myResource.open();
           throw new NullPointerException("try");
       }
   }
}
```

Which exceptions are suppressed when the given program runs and throws an exception?

A. IOException only

B. ArithmeticException only

C. NullPointerException only

D. IOException and ArithmeticException

E. IOException and NullPointerException

F. ArithmeticException and NullPointerException

Correct Answer: B

Explanation

The try-with-resources statement is a try statement that declares one or more resources and ensures that each resource is closed at the end of the statement, whether an exception occurs or not.

According to the Oracle's Java Tutorials, if an exception is thrown from the try block and one or more exceptions are thrown from the try-with-resources statement, then those exceptions thrown from the try-with-resources statement are suppressed.

In the given code, the close() method is called when the resource is closed. Therefore, the exception thrown by this method is suppressed. In this case, this exception is ArithmeticException and hence **option B is correct.**

The exception propagating up the call stack is the one that is thrown from the try block, which is an IOException in this case. Hence, **D and E are incorrect.**

The NullPointerException is not thrown within the close() method and hence, it is also not suppressed. Hence, **option C is also incorrect.**

Reference:
https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html#suppressed-exceptions

Domain : Working with Streams and Lambda expressions

Q24 : Given:

Stream.of(Optional.ofNullable(null)).findFirst().ifPresent(System.out::println);

What is the output of the given code fragment?

A. Null

B. Optional.empty

C. Nothing

D. A NoSuchElementException is thrown

E. A NullPointerException is thrown

F. Compilation fails

Correct Answer: B

Explanation

The ifPresent method enables us to run a lambda expression on the wrapped value if it's found to be non-null. This method takes a Consumer as the argument and returns void. The ofNullable() method returns an Optional describing the given value, if non-null, otherwise returns an empty Optional.

The Optional.ofNullable method produces an empty Optional object since the argument is null. This is the only element in the stream, hence the Stream.findFirst operation returns an Optional object wrapping that empty Optional. The empty Optional is then printed out by the consumer passed to the Optional.ifPresent method. Hence, **option B is correct.**

As the empty Optional is not null or empty, **options A and C are incorrect.**

As there are no exceptions or compiler errors, the **options D, E and F are incorrect.**

Reference:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Optional.html

Domain : Java Platform Module System

Q25 : Given service interface MyService and service implementation class MyServiceImpl. Suppose all the module declarations are valid. Which two of the following are correct ways to get an instance of the service in the client module?

A. MyService service = new MyServiceImpl();

B. MyService service = ServiceLoader.load(MyService.class);

C. MyService service = new ServiceLoader().load(MyService.class);

D. MyService service = ServiceLoader.load(MyServiceImpl.class);

E. MyService service = ServiceLoader.load(MyService.class).findFirst().get();

F. MyService service = ServiceLoader.load(MyService.class).iterator().next();

Correct Answers: E and F

Explanation

A ServiceLoader is an object that locates and loads service providers deployed in the run time environment. Invoking the load() method on the Service Loader creates a new service loader for the given service. The findFirst() method is invoked to load the first available service provider of this loader's service. Hence, **options E and F are correct.**

When using a service loader, the implementation class isn't involved in the client code. Instead, it's loaded dynamically at runtime. This means **options A and D are incorrect.**

Option C is incorrect because the ServiceLoader class doesn't have a public constructor.

Option B is incorrect as the load method returns a ServiceLoader instance rather than an instance of the implementation class.

Reference:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ServiceLoader.html