Quick Quiz August 14, 2022

Test ID: 219913969

Question #1 of 50

Question ID: 1327757

Which three declaration statements correctly initialize their variables? (Choose three.)

A) `int i2 = -6,000;`

B) **`boolean b1 = (6 < 4);`**

C) `var v;`

D) `boolean b2 = 1;`

E) **`float f2 = 10.01E2f;`**

F) **`var v = true;`**

G) `int i1 = 40.4;`

H) `float f1 = 10.01;`

Explanation

The following three declaration statements correctly initialize their variables:

```
boolean b1 = (6 < 4);
```

```
float f2 = 10.01E2f;
```

```
var v = true;
```

The `boolean` variable `b1` is assigned a conditional expression that evaluates to `false` . A `boolean` variable can be only two possible values: `true` and `false` . The `float` variable `f2` is assigned a floating-point number with no loss of precision. By default, floating-point literals are treated as double values. The floating-point literal assigned to `f2` uses the `f` postfix, so that the literal is treated as a float value. The variable `v` is declared using the `var` keyword, meaning that the data type is inferred by the value assigned to it. Thus, `v` will be allocated as a `boolean` value.

Java provides eight primitive data types, each with a default value:

- `byte`: 8-bit data type ranging from `-128` to `127` with a default value of `0`.
- `short`: 26-bit data type ranging from `-32,768` to `32,767` with a default value of `0`.
- `int`: 32-bit data type ranging from `-2,147,483,648` to `2,147,483,647` with a default value of `0`.
- `long` 64-bit data type ranging from `-9,223,372,036,854,775,808` to `9,223,372,036,854,775,807` with a default value of `0L`.
- `float`: 32-bit floating point data type ranging from `3.40282347e38` to `1.40239846e-45` with default value of `0.0f`.
- `double`: 64-bit floating point data type ranging from `1.7976931348623157e308` to `4.9406564584124654e-324` with a default value of `0.0d`.

- `boolean`: Has only two possible values of `true` and `false`, with a default value of `false`.
- `char`: 16-bit Unicode character with default value of `"u0000"`.

In Java SE 8 and above, you can use the `int` data type to represent an unsigned 32-bit integer with a range of `0` to `2e32-1`. Similarly, you use an unsigned version of `long` to represent an unsigned 64-bit integer with a range of `0` to `2e64-1`.

When declaring and initializing variables in Java, you should delineate each declaration with a semi colon (`;`). In Java SE 10 and above, you also use the `var` keyword to infer the data type of the variable based on its initialization.

The declaration statement `boolean b2 = 1;` does not correctly initialize its variable. A `boolean` variable can be only two possible values: `true` and `false`.

The declaration statements `int i1 = 40.4;` and `int i2 = -6,000;` do not correctly initialize their variables. An `int` variable can be assigned only integer values between `-2,147,483,648` and `2,147,483,647` (inclusive). A valid literal cannot include a decimal point or comma separator. In Java 7 and above, you can use the underscore in numeric literals like a comma separator. For example, `int i2 = -6_000`; is a valid declaration statement.

The declaration statement `float f1 = 10.01`; does not correctly initialize its variable. Because floating-point literals are treated as `double` values, a loss of precision warning will prevent the code from compiling. You can either cast the value as `double` or use the `f` postfix in the literal to ensure compilation.

The declaration statement `var v;` will not compile, because it does not specify an initialization value from which to infer a data type. When using the `var` keyword, you must ensure the data type can be inferred by providing an initial value.

Objective:
Working with Java Data Types

Sub-Objective:
Use primitives and wrapper classes, including, operators, parentheses, type promotion and casting

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Primitive Data Types

OpenJDK > Java Local Variable Type Inference: Frequently Asked Questions

---

Question #2 of 50

Question ID: 1327791

Given the following:

```
public class Java11 {
   static int modify (int[] i) {
```

```
    i[0] += 10;
    return i[0] + 10;
  }
  public static void main(String[] args) {
    int[] i = {10};
    //insert code here
  }
}
```

Which statement(s) should be inserted in the code to output `35`?

A) `System.out.println(modify(i));`

B) `modify(i); System.out.println(i[0]);`

C) `System.out.println(modify(i)+ 15);`

D) **`modify(i); System.out.println(i[0] + 15);`**

Explanation

The statements `modify(i); System.out.println(i[0] + 15);` should be inserted in the code to output `35`. Initially, the value for the first element of the `i` array is set to 10. Because an array is an object, the object reference is passed as an argument to the `modify` method. Any modifications to the array will persist after the method returns. In the `modify` method, the first element is incremented by 10 so that it is now 20. In the `println` method, 15 is added to the first element so that the output is 20 + 15, or `35`.

The statement `System.out.println(modify(i));` should not be inserted in the code to output `35`. The output is `30` because the first element is incremented by 10 and then the return value is the first element value (20) plus 10.

The statement `System.out.println(modify(i)+ 15);` should not be inserted in the code to output `35`. The output is `45` because the first element is incremented by 10, then the return value is the first element value (20) plus 10 and finally 15 is added to the return value.

The statements `modify(i); System.out.println(i[0]);` should not be inserted in the code to output `35`. The output is `20` because the first element of the `i` array is set to 10 in the `main` method and then incremented by 10, so that its value is now 20.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Declare and instantiate Java objects including nested class objects, and explain objects' lifecycles (including creation, dereferencing by reassignment, and garbage collection)

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Passing Information to a Method or a Constructor

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Creating Objects

---

Question #3 of 50

Question ID: 1327911

Consider the following code:

```
interface TestInterface {
    static void methodA() {
        printThis();
        System.out.println("This is method1");
    }
    static void methodB() {
        printThis();
        System.out.println("This is method2");
    }
    private abstract void printThis() {
        System.out.println("Method begins");
        System.out.println("Method works!");
    }
    default void testmethods() {
        methodA();
        methodB();
    }
}
public class TestClass implements TestInterface{
    public static void main(String args[]) {
        TestClass tc = new TestClass();
        tc.testmethods();
    }
}
```

What would be the output?

A) `This is method1`

B) `This is method2`

C) `Method works!`

D) **None. The code does not compile.**

Explanation

The code does not compile and produces no output. You need to use the `private` and `static` access modifiers with the `printThis()` method. This allows you to share common code inside of `static` methods using a non-static `private` method.

A `private` method inside a Java interface allows you to avoid redundant code by creating a *single* implementation of a method inside the interface itself. This was not possible before Java 9. You can create a `private` method inside an interface using the `private` access modifier.

You cannot add the keyword `abstract` because the `abstract` and `private` keywords cannot be used together in a Java interface. This is because `private` and `abstract` both have separate uses in Java. When a method is deemed `private`, it indicates that any subclass cannot inherit it or override it. However, when a method is deemed `abstract`, it needs to be inherited and overridden by subclasses.

The other options are incorrect because no output is displayed due to compilation failure.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Create and use interfaces, identify functional interfaces, and utilize private, static, and default methods

References:

Oracle > Java Documentation > The Java Tutorials > Interfaces and Inheritance > Defining an Interface

Oracle > Java Documentation > The Java Tutorials > Interfaces and Inheritance > Default Methods

Chapter 3. Java Interfaces > 3.1. Create and use methods in interfaces

---

Question #4 of 50

Question ID: 1327779

Given:

```
public class Java11 {
   public static void main (String[] args) {
     StringBuilder sb = new StringBuilder("Test");
     System.out.println(sb.length());
   }
}
```

What is the output when this program is executed?

A) `12`

B) `4`

C) `16`

D) `20`

Explanation

The output is `4`. The `length` method returns the number of characters stored by the `StringBuilder` object. The literal string `Test` is 4 characters in length. Because this underlying `String` object is not modified, the number of characters remains at the initial count.

The output is not `12` or `16`. The initial capacity defaults to 16, but the `length` matches the number of characters in the underlying `String` object.

The output is not `20`. The initial capacity of the `StringBuilder` object is 20, but not its `length`. If a `String` object is specified in the constructor, then the initial capacity is 16 plus the length of the `String` object. In this case, the literal string `Test` is 4 characters in length, so the capacity is 16 + 4 or 20.

Objective:
Working with Java Data Types

Sub-Objective:
Handle text using String and StringBuilder classes

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Numbers and Strings > The StringBuilder Class

Java API Documentation > Java SE 11 & JDK 11 > Class StringBuilder

---

Question #5 of 50

Question ID: 1328144

Given:

```
int [] res = IntStream.of(1,2,3,4,5,6,7,8,9,10)
  .parallel()
  .collect( ()->new int[2],  // line n1
  (a,b)->{if (b%2==0) a[1]+=b; else a[0]+=b;}, // line n2
  (a,b)->{a[0]+=b[0];a[1]+=b[1];});
System.out.println("Odd sum = " + res[0] + " even sum = " + res[1]);
```

What is the result?

A) `Odd sum = 25 even sum = 30`

B) Compilation fails at line `n1`

C) **Compilation fails at line `n2`**

D) `Odd sum = 55 even sum = 55`

E) Non-deterministic output

Explanation

The result is the following output:

```
Odd sum = 25 even sum = 30
```

This collect method takes three arguments. The first is a `Supplier<R>` where *R* is the result type of the collection operation. In this case, the supplier creates an array of two `int` values, and that is consistent with the result type for the operation.

The second argument for this collector is a `BiConsumer<R,T>` where *T* is the stream type. Since the stream contains integers, these might be `int` primitives or `Integer` objects. The behavior of the block lambda is to test if the second argument (the item from the stream) is odd or even, and to add that stream value to one or the other element of the array depending on whether the stream value is odd or even. The arguments are `int[2]` and `int`/`Integer`, and are used correctly for those types.

The return type of the `BiConsumer` is `void`, and the block of the lambda does not return anything. The resulting behavior is a collection of the sums for odd and even numbers seen in the two elements of the array of int, and to output the odd sum, *25*, and the even sum, *30*.

The output will not have the same value of 55 for both odd and even numbers because the lambda expression in the second argument checks for whether items are odd or even and third argument increments these values independently.

The output is deterministic. The collect operation differs from a reduce operation in that each thread that is created in a parallel stream situation is given its own mutable storage for collecting intermediate results. Because each thread is given its own mutable storage, the resulting output is predictable and expected.

The code at line `n1` is correct. It defines a supplier of `int[2]`, which is appropriate as the first argument to the collector and compatible with the rest of the collector arguments.

The code at line `n2` is also correct. It defines a lambda that is compatible with `BiConsumer<int[2], int>`.

Objective:
Working with Streams and Lambda expressions

Sub-Objective:
Perform decomposition and reduction, including grouping and partitioning on sequential and parallel streams

References:

The Java Tutorials > Collections > Aggregate Operations > Parallelism

Java Platform Standard Edition 11 > API > java.util.stream > Stream

Question #6 of 50

Question ID: 1327829

Which statement is true about constructor overloading?

A) A default constructor can be overloaded in the same class.

B) The constructor must use the `this` keyword.

C) A default constructor can be overloaded in a subclass.

D) The constructor must use a different name.

Explanation

A default constructor can be overloaded in a subclass. If no constructor is defined for a class, then the compiler will automatically provide the default constructor. Because a subclass can define its own constructors without affecting the superclass, a constructor with parameters can be defined that invokes the superclass constructor, implicitly or explicitly.

A default constructor cannot be overloaded in the same class. This is because once a constructor is defined in a class, the compiler will not create the default constructor. Thus, an attempt to overload the default constructor will effectively remove it from the class.

The constructor must not use a different name. In the same class, an overloaded constructor uses the same name. Because subclasses differ in name from their superclass, an overloaded constructor will have a different name.

The constructor does not need to use the `this` keyword. The `this` keyword allows a constructor to reference other constructor methods and/or instance context. Using the `this` keyword is not required in an overloaded constructor.

You can use the `this` keyword in a constructor to call another constructor in the same class. This technique is called explicit constructor invocation.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Initialize objects and their members using instance and static initializer statements and constructors

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Providing Constructors for Your Classes

Question #7 of 50

Question ID: 1328165

In a multi-threaded application, a single `Queue` object must be read by multiple threads and modified by another thread. Multiple threads should be able to read concurrently or allow a write operation by a single thread. Which strategy should you use?

A) Implement a custom lock with the `ReadWriteLock` interface.

B) Replace the `Queue` object with a `BlockingQueue` object.

C) Use a synchronized block for read and write operations.

D) Use the `volatile` keyword when declaring the Queue variable.

Explanation

To meet the scenario requirements, you should implement a custom lock with the `ReadWriteLock` interface. Using the `ReadWriteLock` interface, there are two associated locks, one for read operations and another for write operations. The read lock is not exclusive and can be held by multiple threads that perform read operations, if no write operations are performed. The write lock is exclusive, so that only a single thread can perform write operations.

You should not use a synchronized block for read and write operations because this strategy will limit read and write operations to a single thread at a time.

You should not use the `volatile` keyword when declaring the `Queue` variable because this strategy will only affect the `Queue` object itself, not read/write operations on its contents. The `volatile` keyword ensures that a shared variable value is visible to multiple threads. You should use the `volatile` keyword on a shared variable when write operations do not depend on its value and locking is not required during access.

You should not replace the `Queue` object with a `BlockingQueue` object because this strategy will limit all read and write operations. The `BlockingQueue` prevents memory consistency errors, so that write operations are visible to all threads when they occur. This is equivalent to using a single mutex lock, not a lock pair.

Objective:
Concurrency

Sub-Objective:
Develop thread-safe code, using different locking mechanisms and java.util.concurrent API

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.util.concurrent.locks > Interface ReadWriteLock

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Lock Objects

---

Question #8 of 50

Question ID: 1327812

Given the following code line:

```
printToConsole(1.2);
```

Which overloaded method is invoked?

A)
```
void printToConsole(Integer intObj) {
   System.out.println("My integer object is " +
intObj.toString());
}
```

B)
```
void printToConsole(Object obj) {
   System.out.println("My object is " + obj.toString());
}
```

C)
```
void printToConsole(Double dblObj) {
   System.out.println("My double object is " +
dblObj.toString());
}
```

D)
```
void printToConsole(float fl) {
   System.out.println("My float is " + fl);
}
```

Explanation

The following overloaded method is invoked:

```
void printToConsole(Double dblObj) {
   System.out.println("My double object is " + dblObj.toString());
}
```

This method is invoked because the default primitive type for a literal fractional value is `double`, and this type is automatically boxed as a `Double` object. Thus, the overloaded method with a `Double` parameter is invoked. When a method is invoked on an overloaded method, only the list of parameter data types determines which method is executed.

The overloaded method with a `float` parameter is not invoked. The default primitive type for a literal fraction value is `double`, not `float`. This overloaded method would be executed if the invocation contains a valid `float` value such as 1 or 1.2f.

The overloaded method with an `Integer` parameter is never invoked. The default primitive type for a literal fraction value is `double`, not `int`. Neither the `int` nor its box class `Integer` support fractional amounts. They support only whole numbers. This overloaded method is not invoked because the method with the `float` parameter will execute when an `int` value or variable is specified as an argument.

The overloaded method with an `Object` parameter is not invoked. All objects inherit from the `Object` class, so this overloaded method is an effective catch-all for an argument whose data type is not explicitly declared in another

overloaded method.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Define and use fields and methods, including instance, static and overloaded methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Defining Methods

---

Question #9 of 50

Question ID: 1328154

What is the difference between the `Executor` and `ExecutorService` interfaces?

A) The `Executor` interface only supports the `submit` method.

B) The `ExecutorService` interface provides methods for lifecycle management of both `Runnable` and `Callable` objects.

C) The `Executor` interface supports both `Runnable` and `Callable` objects.

D) **The `ExecutorService` interface does not support the `execute` method.**

Explanation

The main difference between the two interfaces is that the `ExecutorService` interface provides methods for lifecycle management of both `Runnable` and `Callable` objects. The additional `submit` method returns a `Future` object to retrieve values from `Callable` objects and manage the status of `Runnable` and `Callable` objects.

The `Executor` interface does not support the `submit` method. The `Executor` interface supports only the `execute` method.

The `ExecutorService` interface supports the `execute` method, in addition to the `call` method.

The `Executor` interface only supports `Runnable` objects, not `Callable` objects. The `ExecutorService` interfaces supports both `Runnable` and `Callable` objects.

Objective:
Concurrency

Sub-Objective:
Create worker threads using Runnable and Callable, and manage concurrency using an ExecutorService and java.util.concurrent API

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Executor Interfaces

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.util.concurrent > Interface Executor

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.util.concurrent > Interface ExecutorService

---

Question #10 of 50

Question ID: 1327883

Given the code fragment:

```
CharSequence obj = new StringBuilder ( new String("String") );
```

Which is the reference type?

A) `CharSequence`

B) `StringBuilder`

C) `String`

D) `Object`

Explanation

The reference type is `CharSequence`. The reference type corresponds to the type in the variable declaration.

The reference type is not `Object`. Although all classes implicitly extend the `Object` class, the reference type is the explicit data type declared for the variable.

The reference type is not `String`. Although `StringBuilder` uses the `String` class, the reference type is the explicit data type declared for the variable.

The reference type is not `StringBuilder`. `StringBuilder` is the object type because the object type corresponds to the instantiated class.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Utilize polymorphism and casting to call methods, differentiate object type versus reference type

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Classes and Objects

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Interfaces and Inheritance > Polymorphism

---

Question #11 of 50

Question ID: 1327997

Which statement is true about the `Statement` objects?

A) **`CallableStatement` objects support both input and output parameters.**

B) `Statement` objects only support input parameters.

C) `Statement` objects support multiple open result sets.

D) `PreparedStatement` objects only support output parameters.

Explanation

`CallableStatement` objects support both input and output parameters. Because `CallableStatement` objects contain stored procedures, they support input/output and return parameters.

`PreparedStatement` objects do not support output parameters. `PreparedStatement` objects support only input parameters.

`Statement` objects do not support input parameters. `Statement` objects only support simple SQL statements with no parameters.

`Statement` objects do not support multiple open result sets. Only a single result set can be open from the same `Statement` object. If the same `Statement` object is executed more than once, the previous result set is closed. Attempting to access the closed result set will throw a `SQLException`. The following code fragment demonstrates this runtime error:

```
String query = "SELECT Name FROM Customer WHERE City='Atlanta'";
try (Statement stmt = cn.createStatement()){
  ResultSet rs1 = stmt.executeQuery(query);
  stmt.executeQuery("SELECT Name FROM Customer WHERE City='Miami'");
  while (rs1.next()) {
    //This throws an error, because rs1 is closed automatically
  }
} catch (SQLException ex) {
  System.err.println(ex);
}
```

Objective:
Database Applications with JDBC

Sub-Objective:
Connect to and perform database SQL operations, process query results using JDBC API

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > JDBC(TM) Database Access > JDBC Basics > Processing SQL Statements with JDBC

Question #12 of 50

Question ID: 1327841

Which statement is true about applying encapsulation principles to a class?

A) Mutator methods should return direct references to field objects.

B) Accessor methods should return direct references to field objects.

C) The default modifier should be `protected` for all fields.

D) The default modifier should be `private` for all fields.

Explanation

The default modifier should be `private` for all fields. Access through methods, rather than directly to fields, is a core principle of encapsulation. Thus, using the most restrictive access modifier `private` is recommended unless there is some reason to make an exception.

The default modifier should not be `protected` for all fields because subclasses will have direct access to fields. The access modifier `private` is the recommended for all fields, with only `protected` applied to fields that must be directly accessible to subclasses.

Accessor methods should not return direct references to field objects. Direct access will allow other classes to modify fields directly without going through mutator methods.

Mutator methods should not return direct references to field objects. Mutator methods rarely return values, and instead perform field modifications. Those return values should not include direct references to field objects.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Understand variable scopes, apply encapsulation and make objects immutable

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Controlling Access to Members of a Class

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Object-Oriented Programming Concepts > What Is an Object?

Question #13 of 50

Question ID: 1327957

Once you have a module declared and the dependencies identified, you need to compile your Java module and send the generated class files to a specific folder.

Which command would you use to do this?

A) `java`

B) **`javac -d`**

C) `javac -g`

D) `javac -s`

Explanation

The command you would use to compute your Java module and send the class file to a specific folder is `javac -d`. The `-d` option stands for the directory path. Here you articulate where your class files are to be generated and sent. The `-s` option is used to specify where the source files are to be placed. The `-g` option is used to enable detailed debugging during the compile process.

The command `javac -s` is used to determine the source file location and is not used to specify the class directory location.

The command `javac -g` is used to add detailed debugging to your output and is not used for sending the class file to a specific directory.

The command `java` is not used to compile your Java source code but is used to execute the Java program.

Objective:
Java Platform Module System

Sub-Objective:
Deploy and execute modular applications, including automatic modules

References:

docs.oracle.com > Module jdk.compiler > javac

fedoraproject.org > JDK on Fedora > JDK components

Question #14 of 50

Question ID: 1328106

Given the code fragment:

```
Stream.of("Fred", "Jim", "Sheila")
  /* insert here */
  .forEach(System.out::println);
```

And the desired output:

```
4
3
6
```

Which code should be inserted at `/* insert here */` to generate the desired output?

A) **`.map(String::length)`**

B) `.filter(s->s.length())`

C) `.flatMap(s->s.length())`

D) `.count()`

Explanation

You should insert `.map(String::length)` to generate the desired output.

The `Stream.map` method is used to take one item from a stream and create new data from it. In this case, the conversion required is to take a `String` from the stream and send a number representing the length of that `String` downstream.

The code `.filter(s->s.length())` is incorrect because the `Stream.filter` method does not convert the data items. Instead, it optionally removes some items from the stream.

The code `.flatMap(s->s.length())` is incorrect because the `flatMap` method converts a single item in a data stream into any number of items of potentially different types. While it could in theory achieve the desired effect, using it to do so would be very unusual and involve unnecessary computation. In this specific code fragment, it would not achieve the desired effect anyway, since the `Function` to be provided must return a `Stream<T>`, not a single item. So, in this case, the lambda expression would not match the argument type of `flatMap`, and compilation would fail.

The code `.count()` is incorrect. The count method would cause the overall code fragment to fail to compile in this case, since count is a terminal operation. It returns a long, and therefore invoking `forEach` on the returned value of count would fail. Further, the `count` method counts all the items that reach it, not some individual aspect of each item.

Objective:
Working with Streams and Lambda expressions

Sub-Objective:
Use Java Streams to filter, transform and process data

References:

Java Platform Standard Edition 11 > API > java.util.stream > Stream

---

Question #15 of 50

Question ID: 1328082

Which statement compiles and executes, but does not produce any output?

A) `Optional.empty().ifPresent(System.out::println);`

B) `Optional.empty().get();`

C) `Optional.of(null);`

D) `Optional.empty().orElseGet(()->{System.out.println("No");return false;});`

Explanation

The following statement compiles and executes, but does not produce any output:

`Optional.empty().ifPresent(System.out::println);`

The `ifPresent` method causes the `Consumer` argument to be invoked with the contents of the `Optional` as an argument to the accept method in that `Consumer`, but only if the `Optional` is not empty. Because the `Optional` in this item is empty, no output is produced.

The code that calls the get method on an empty `Optional` throws a `NoSuchElementException`..

The code that calls the `orElseGet` method on an empty `Optional` invokes the behavior in the `Supplier` argument, and produces the output `No`.

The code that uses the factory method of with a null argument throws a `NullPointerException`. To create an `Optional` with a possibly null argument, you should use the `Optional.ofNullable` method.

Objective:
Working with Streams and Lambda expressions

Sub-Objective:
Implement functional interfaces using lambda expressions, including interfaces from the java.util.function package

References:

Java Platform Standard Edition 11 > API > java.util > Optional

Question #16 of 50

Question ID: 1328031

Given the following output:

```
8
5
2
-1
```

Which code fragment generates this output?

A)
```
int x = 0, y = 10;
do {
   x += 2;
   System.out.println(x - y);
   y --;
} while ( x - y > 0 );
```

B)
```
int x = 0, y = 10;
do {
   x += 2;
   System.out.println(y - x);
   y --;
} while ( y - x > 0 );
```

C)
```
int x = 10, y = 0;
do {
   x += 2;
   System.out.println(y - x);
   y --;
} while ( y - x > 0 );
```

D)
```
int x = 10, y = 0;
do {
   x += 2;
   System.out.println(x - y);
   y --;
} while ( x - y > 0 );
```

Explanation

The following code fragment will generate the required output:

```
int x = 0, y = 10;
do {
   x += 2;
   System.out.println(y - x);
   y --;
} while ( y - x > 0 );
```

This code fragment initializes x to 0 and y to 10. In the `do-while` block, x is incremented by 2 and y is decremented by 1. The difference between y and x is printed and repeated until the difference is 0 or less. The variable x is incremented by 2 before the difference is printed, while y is decremented afterwards. The following table tracks variable values for each iteration:

Iteration	x value	Difference	y value
1	2	8	9
2	4	5	8
3	6	2	7
4	8	-1	6

The code fragments that initialize x to 10 and y to 0 will not generate the required output. If the difference in the expression for the print statement and the `while` statement is `y - x`, then the output will be `-12`. Otherwise, the result will be an endless loop because the expression `y - x` will always be positive.

The code fragment that initializes x to 0 and y to 0 and uses the expression `y - x` for the print statement and `while` statement will not generate the required output. The output will be `-8`.

Objective:
Controlling Program Flow

Sub-Objective:
Create and use loops, if/else, and switch statements

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > The while and do-while Statements

---

Question #17 of 50

Question ID: 1327956

Which of the following is true when declaring a Java module?

A) The Java module must contain at least one class member.

B) **The Java module must be declared directly under the root directory.**

C) The Java module name does not need to be unique in a directory.

D) The Java module can use implied packages without explicit declaration.

Explanation

When you declare a Java module, it is required that the module be declared under the root directory. If you had packages defined under a directory path such as `javaprogam/finance/usingmymodules`, then your root directory must be the same name as your module. All packages and coding files should be placed under the root directory. The root directly in this example can also be added to a JAR file or to the `classpath` parameter. When declaring your module name, you also should not use _ (underscores) in your naming conventions, but instead and reserve it for packages, classes, variables, and methods.

It is required that your Java module be a unique name and not duplicated within the directory.

Your Java module must explicitly declare the packages that are required for your Java code. This is a key benefit of using modular JDK.

If you are planning on using a specific module, but do not predefine a package that needs to be exported in the module, then you can create an empty module declaration as follows:

```
module javaprogam/finance/usingmymodules {

}
```

Objective:
Java Platform Module System

Sub-Objective:
Deploy and execute modular applications, including automatic modules

References:

tutorials.jenkov.com > Java Modules > Java Module Basics

openjdk.java.net > JEP 200: The Modular JDK > Design principles

---

Question #18 of 50

Question ID: 1327847

Which statement is true about a composition relationship?

A) Implementation modifications in the container class will affect the implementation of the constituent class.

B) Methods in the constituent class can provide unique implementation to methods in the container class.

C) Methods in the container class can delegate implementation to methods in the constituent class.

D) Interface modifications in the constituent class will affect the interface of the container class.

Explanation

In a composition relationship, methods in the container class can delegate implementation to methods in the constituent class. Method delegation, also known as request-forwarding, is a key aspect of a composition relationship. In a composition relationship, an instance forwards method invocation to another class instance.

Implementation modifications in the container class will not affect the implementation of the constituent class. Implementation modifications made in one class will not affect the implementation of another.

Interface modifications in the constituent class will not affect the interface of the container class. Interface modifications in the constituent class will affect implementation in the container class but will not affect its interface.

Methods in the constituent class cannot provide unique implementation to methods in the container class. Implementation is not shared between container and constituent classes. In class inheritance, subclasses can provide unique implementation to methods in the superclass. This concept is known as *polymorphism*.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Create and use subclasses and superclasses, including abstract classes

References:

JavaWorld > Java QA > Delegates

JavaWorld > Core Java > Java Platform APIs > Inheritance versus composition: Which one should you choose? > Page 2 > Comparing composition and inheritance

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Object-Oriented Programming Concepts > What is Inheritance

---

Question #19 of 50

Question ID: 1328166

Which statement is true about the `ReadWriteLock` interface?

A) There is one shared lock used for both read-only and write operations.

B) A read lock can be used by multiple reader threads, while a writer thread can use the write lock.

C) The shared lock is not exclusive.

D) There are two locks: one for read-only operations and one for write operations.

Explanation

The `ReadWriteLock` interface has two locks: one for read-only operations and one for write operations. The read lock is not exclusive and can be held by multiple threads that perform read operations, if no write operations are performed. The write lock is exclusive, so that only a single thread can perform write operations.

There is not one shared lock for read-only and write operations. The `ReadWriteLock` interface has two locks.

A read lock cannot be used by multiple threads if a writer thread is using the write lock. If a writer thread is using the write lock, then the read lock cannot be shared between multiple threads.

The shared lock is exclusive if performing a write operation. If no write operation is being performed, then the read lock can be shared across multiple threads.

Objective:
Concurrency

Sub-Objective:
Develop thread-safe code, using different locking mechanisms and java.util.concurrent API

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.util.concurrent.locks > Interface ReadWriteLock

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Lock Objects

---

Question #20 of 50

Question ID: 1328182

You have a Java application that provides directory listing using the following code:

```
class MisterLister {
  public static void main(String[] args) throws Exception {
    String directory = System.getProperty("dir");
    Runtime run = Runtime.getRuntime();
    Process prc = run.exec("cmd.exe /C dir " + directory);
    int res = prc.waitFor();
    if (res != 0) {
      System.out.println("Process failed by: " + res);
    }
    InputStream input = (res == 0) ? prc.getInputStream() : prc.getErrorStream();
    int i;
    while ((i = input.read()) != -1) {
     System.out.print((char) ch);
```

```
    }
  }
}
```

What would you use to secure this application? (Choose all that apply.)

A) **`Pattern.matches()`**

B) **`Collections.unmodifiable`**

C) `Identity.hashmap`

D) `File.list()`

E) `ServiceLoader`

Explanation

You can use either `Pattern.matches()` to sanitize user input or use `File.list()` instead of using `Runtime.exec()`.

To sanitize user input, you would use the following code before the arguments are passed to the `Runtime.exec()` method:

```
if (!Pattern.matches("[0-9A-Za-z@.]+", directory)) {
  // Error handling code
}
```

The following code illustrates a more secure directory listing application using the `File.list()` method:

```
import java.io.File;

class MisterLister2 {
  public static void main(String[] args) throws Exception {
    File directory = new File(System.getProperty("dir"));
    if (!directory.isDirectory()) {
      System.out.println("Directory Error!");
    } else {
      for (String filename : dir.list()) {
        System.out.println(filename);
      }
    }
  }
}
```

Using `Collections.unmodifiable` is an incorrect option. You do this to protect collections.

Using `ServiceLoader` is an incorrect option. For any service to be used successfully, each of its service providers is required to be found and then loaded. For this purpose, the `ServiceLoader` class exists and takes care of the

same.

Using `Identity.hashmap` is an incorrect option. You use this when needing to secure identity equality operations.

You need to validate the parameters to methods for securing applications. Three ways to do this are to validate inputs to methods, validate outputs from untrusted objects as inputs, and create wrappers for native methods.

- When validating inputs, take care to avoid integer overflows in input values and addition of (`../`) as arguments, which can be used to traverse directories.
- Return values of certain methods need to be checked; for example, attackers could use `ClassLoader` instances passed as return values.
- Native code needs to be placed in a wrapper so that only its functionality is exposed and not its internal working and parameters. This way input validation for the native code can be done by the Java wrapper. The following demonstrates how this can be implemented:

```
public final class NativeCodeWrapper {
  // native method kept private
  private native void nativeMethod(byte[] data, int offset, int len);
  // wrapper method performs checks
  public void methodChecker(byte[] data, int offset, int len) {
    // Here we check the parameters passed to the native method and throw
    // an argument if there is a problem with the arguments
    // we now call the native method and pass the validated arguments
    nativeMethod(data, offset, len);
  }
}
```

Mutability can cause security issues in a Java application. Some ways to ensure that doesn't happen is to follow these guidelines:

- Keep value types immutable. To do this, you declare fields `final` so that the object cannot be modified post construction.
- Copy mutable output values. You should create a copy of an internal mutable object if a method returns a reference to it.
- Make copies of input values input values for mutables and subclasses. Attackers can take advantage of a TOCTOU (Time-of-check Time-of-use) inconsistency where an object from an untrusted source may pass a `SecurityManager` check but different values may then be used by the object. To avoid this risk, you need to create a copy of the input object and then perform method logic on the copy and not the actual object.
- Provide copy functionality for mutable classes. You need to create a means to safely copy instances of a mutable class. You do this by making a static creation method, a copy constructor, or a `public` copy method for all final classes.
- Check identity equality. Identity equality methods like `Object.equals` could be overidden so that an object can be made to look *equal* to another object even when it isn't. A way to safeguard against this is to use `Identity.HashMap` or other collection implementations for ensuring identity equality.

- Check input and output to and from untrusted objects. When an object is passed to untrusted objects, the data in the object being passed should be cloned before passing by an appropriate `clone()` method. Similarly, when an object is received from an untrusted source it needs to be copied and validated before being passed on.
- Create wrapper methods for internal state code. If a class has a state that will be accessed publicly, then you need to create a `public` wrapper method around it and make it a `private` field. Similarly, if you have a state in a class that will be accessed by subclasses then you need to make it a `private` field and create a `protected` wrapper method for it.
- Finalize `public static` fields. `public static` fields need to be declared as `final` to ensure that they are not accessed and modified by attackers. When using arrays or lists, use methods like `unmodifiableList` to ensure that the list isn't modified and so secured.
- Hide mutable statics. `private static` members should be treated as if they were `public` to ensure safety of code.
- Hide collections that can be modified. Collections exposed to other classes should be made read-only or unmodifiable. You create unmodifiable collections using `of/ofEntries`, `copyOf`, or `Collections.unmodifiable` methods.

Objective:
Secure Coding in Java SE Application

Sub-Objective:
Develop code that mitigates security threats such as denial of service, code injection, input validation and ensure data integrity

References:

Oracle Technology Network > Java > Secure Coding Guidelines for Java SE

Sanitizing User Inputs

Question #21 of 50

Question ID: 1328186

Consider the following class:

```
public final class bitCoinHolder implements Serializable { //Line 1
  private static final long serialVersionUID = 1L; // Line2
  private double userID; // Line 3
  private String username; // Line 4
  private String bankname; // Line 5
  private String accountnumber; // Line 6
  public String readACValue() { // Line 7
    return accountnumber; // Line 8
  }
  private void readObject(java.io.ObjectInputStream in) { // Line 9
```

```
    java.io.ObjectInputStream.GetField field = in.readFields(); // Line 10
    this.accountnumber = ((String)field.get("accountnumber"); //Line 11
  }
}
```

What should you do to protect data during deserialization? (Choose all that apply.)

A) Remove `implements Serializable` from Line 1

B) Add `File.list` after Line 11

C) Add a `clone()` method after Line 11

D) Add `checkPermission` after Line 7

E) Add `getSecurityManager` at Line 9

Explanation

The correct options are:

- Remove `implements Serializable` from Line 1
- Add a `clone()` method after Line 11

Both of these approaches are necessary to protect sensitive data during serialization and deserialization.

The other options are incorrect because they do not aid in securing a serializable class. `File.list` and `checkPermission` could be used to secure the Java application in other ways, however.

The best course of action is to simply not serialize sensitive classes. Deserialization creates a new class instance without invoking the constructor of the class. This causes methods like `ObjectInputStream.defaultReadObject` to assign objects to fields that are non-transient and then not return. This can cause a security risk because hackers can bring in their own objects into the application. So, to protect against this you should use `ObjectInputStream.readFields` to first copy before making assignments to fields. The following code illustrates this technique:

```
public final class safeByte implements java.io.Serializable {
  private static final long serialVersionUID = 1L;
  private byte[] bData;

  public safeByte(byte[] data) {
    this.bData = data.clone(); // Making a copy pre-assignment.
  }
  private void readObject(java.io.ObjectInputStream in)throws
   java.io.IOException, ClassNotFoundException {
    java.io.ObjectInputStreadm.GetField allfields = in.readFields();
    this.bData = ((byte[])allfields.get("data")).clone();
  } // We assign cloned data from allfields to the class field
}
```

You should also do input validation on the `readObject` method in a constructor

Additionally, you should have several `SecurityManager` checks *outside* of the class constructor. This needs to be done in all instances of `readObject` and `readObjectNoData` method implementations for the class. Failure to do this allows a hacker to create an instance of the class during deserialization. Similarly, all instances of `writeObject` methods for the class must also have a `SecurityManager` check embedded so that an attacker cannot serialize the data of the class to read its internal fields. The `SecurityManager` check needs to exist inside calling methods for a call which are used to read its internal values. This is done by executing the `securityManagerCheck()` method from within the method you need to secure. The following code illustrates how you can embed this check inside a class method that returns the value of its internal fields:

```
public String readValue() {
  // The SecurityManager check needs to run before the value can be read
  securityManagerCheck();
  return value;
}
```

The security permissions for Java applications should be kept at minimum because keeping these at maximum can allow attackers to circumvent all security checks during serialization and deserialization. These permissions are checked in `java.security.GuardedObject`.

Finally, you can filter untrusted classes from accessing your code by using the `ObjectInputFilter` API. Classes that can cause issues in the Java runtime environment need to be blacklisted.

Objective:
Secure Coding in Java SE Application

Sub-Objective:
Secure resource access including filesystems, manage policies and execute privileged code

References:

Oracle Technology Network > Java > Secure Coding Guidelines for Java SE

Question #22 of 50

Question ID: 1328181

Which of these techniques will you use to prevent external attacks on a Java package? (Choose all that apply.)

A) **Implement the `Cloneable` class for all classes in the package**

B) Keep unrelated code together

C) Declare all `public` classes `final`

D) **Mark the package as sealed in the manifest**

E) Keep all classes and interfaces `public`

Explanation

You should mark the package as sealed in the manifest and declare all `public` classes as `final`.

Implementing the `Cloneable` class and keeping all classes and interfaces `public` can both cause a security breach. Implementing the `java.lang.Cloneable` class allows attackers to clone existing objects and so confuse instances of the class being attacked. Finally, default methods on interfaces can be used by attackers. For this reason, all interfaces implemented by a sensitive class need to be monitored.

Keeping unrelated code together is incorrect because untrusted code can access other parts of the same package, causing a security breach.

To successfully secure the code of a system, you need to reduce its vulnerable areas. Here are some of the ways to limit the accessibility and extensibility of code to prevent external attacks:

- Restrict access to classes, interfaces, methods, and their fields.
- Restrict access to packages.
- Isolate unrelated code.
- Restrict instances of `ClassLoader`.
- Restrict the extensibility of classes and methods.
- Analyze superclass-subclass relationships.

Provide restricted access to classes, interfaces, methods and their fields. Classes and interfaces should be made public only if they are to be a part of published API; otherwise, keep them `private` for security. Similarly, packages should be marked *sealed* in the manifest for the jar file for package.

Keep restricted access to packages. You need to set the `package.access` security property to restrict access to packages. The following code indicates how this is done:

```
private static final String PACKAGE_ACCESS_KEY = "package.access";
  static {
    String packAccess = java.security.Security.getProperty(PACKAGE_ACCESS_KEY);
    java.security.Security.setProperty( PACKAGE_ACCESS_KEY,
     (
     (packAccess == null ||
     packAccess.trim().isEmpty()) ? "" :(packAccess + ",")) +
     "xyz.lord.java.security.");
  }
```

Keep code that is unrelated isolated. Code in containers that is unrelated to the application should be isolated because untrusted code can easily reference its origin, causing a security breach. Also, Mutable statics and exceptions can breach isolation of code. You need to ensure that untrusted code does not have package-private access to secure the application. This can be implemented by using separate class loader instances for libraries and other code.

Restrict instances of `ClassLoader`. `ClassLoader` instances need to be restricted because they can access client code and get data from resource URLs. They can also be cast to certain subclasses that have malicious code. The

`java.lang.Class.getClassLoader` method can bypass checks made by `SecurityManager`. You safeguard against this by not invoking certain methods on `ClassLoader`, `Class` or `Thread` instances which are received from untrusted code.

Restrict the extensibility of classes and methods. If a class is not made `final`, it can be overridden by an attacker. Finalizing a class and then making its constructor `private` helps secure against this threat. Certain subclasses can override even the `Object.finalize` method. In order to protect classes from malicious subclassing, you perform a security check inside the class you want to protect using a code fragment similar to this:

```
public class Security {
  public Security () {
    // this constructor is accessible
    this(securityManagerCheck());
  }
  private Security (Void ignore) {
    // this constructor is private
  }
  private static Void securityCheck() {
    SecurityManager sec = System.getSecurityManager();
    if (sec != null) {
      sec.checkPermission();
    }
    return null;
  }

}
```

Analyze superclass-subclass relationships. There are situations where parent classes may be updated with new methods which are not overridden in subclasses, which can cause security vulnerabilities. For example, the `Provider` class is an extension of `Hashtable`. When `Hashtable` got a new method `entrySet` that allows entries to be removed from `Hashtable`, the `Provider` class was not updated to override this method. This caused a security breach by allowing hackers to bypass the `SecurityManager` check and delete mappings in `Provider` by running the `entrySet` method.

Objective:
Secure Coding in Java SE Application

Sub-Objective:
Develop code that mitigates security threats such as denial of service, code injection, input validation and ensure data integrity

References:

Oracle Technology Network > Java > Secure Coding Guidelines for Java SE

Question #23 of 50

Question ID: 1327925

Examine the following code fragment:

```
public static void printGrades() {
  Map<Double, List<String>> grades = new HashMap<Double, ArrayList<String>>();
  List<String> students;
  students = Arrays.asList("Sam", "Mary", "Ann");
  grades.put(98.5, students);
  students = Arrays.asList("George", "Aima");
  grades.put(88.2, students);
  students = Arrays.asList("Bob", "Christen", "Robin");
  grades.put(76.8, students);
  for (Map.Entry<Double, List<String>> gradeEntry : grades.entrySet()) {
    System.out.printf("Students with %s : %s\n",
      gradeEntry.getKey(), gradeEntry.getValue().toString());
  }
}
```

What is the result?

A)
```
Students with [George, Aima] : 88.2
Students with [Sam, Mary, Ann] : 98.5
Students with [Bob, Christen, Robin] : 76.8
```

B)
```
Students with 88.2 : [George, Aima]
Students with 98.5 : [Sam, Mary, Ann]
Students with 76.8 : [Bob, Christen, Robin]
```

C) Code compilation fails.

D) Code throws a runtime exception.

E)
```
Students with :
```

Explanation

Code compilation fails because of how the `grades` variable is assigned. The type parameter `List<String>` in the `grades` variable declaration does not match `ArrayList<String>` in the `HashMap` instantiation. Type parameters must match if not inferred.

Either of the following code statements should be used in its place to compile:

```
Map<Double, List<String>> grades = new HashMap<Double, List<String>>();
```

```
Map<Double, List<String>> grades = new HashMap<>();
```

The second code statement relies on type inference by using an empty set of type parameters, known as the *diamond*.

The code will not throw a runtime exception. The code will fail compilation, because of how the `grades` variable is assigned.

The code will not print output. The code will fail compilation, because of how the `grades` variable is assigned. If this statement is corrected, then the following output will be printed:

```
Students with 88.2 : [George, Aima]
Students with 98.5 : [Sam, Mary, Ann]
Students with 76.8 : [Bob, Christen, Robin]
```

Objective:
Working with Arrays and Collections

Sub-Objective:
Use generics, including wildcards

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Generics > Type Inference

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Collections > Lesson: Implementations

---

Question #24 of 50

Question ID: 1328064

Given the custom exception:

```
class OutsideStringException extends StringIndexOutOfBoundsException {}
class CannotFindPatternInStringException extends OutsideStringException {}
```

Given the method:

```
boolean searchString(String pattern, String str) throws OutsideStringException {
  //implementation omitted
}
```

Which handling action is required when invoking the `searchString` method for the code to compile?

A) **No handling action is required.**

B) Catch `CannotFindPatternInStringException`.

C) Catch `StringIndexOutOfBoundsException`.

D) Specify `StringIndexOutOfBoundsException`.

E) Specify `CannotFindPatternInStringException`.

Explanation

No handling action is required for the code to compile because `OutsideStringException` is a subclass of `StringIndexOutOfBoundsException`, which is a subclass of `RuntimeException`. Checked exceptions are only `Throwable` and its subclasses, excluding `RuntimeException` and its subclasses.

Caching or specifying `StringIndexOutOfBoundsException` or `CannotFindPatternInStringException` is not required for the code to compile. Runtime exceptions are abnormal conditions internal to the application and often are unrecoverable. The compiler does not check catching and specifying `RuntimeException` and its subclasses.

Objective:
Exception Handling

Sub-Objective:
Create and use custom exceptions

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Exceptions > The Catch or Specify Requirement

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Exceptions > Specifying the Exceptions Thrown by a Method

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Exceptions > Creating Exception Classes

---

Question #25 of 50

Question ID: 1327786

Given the following code fragment:

```
Customer cust = new Customer();
```

Which part of the statement instantiates the `Customer` object?

A) `Customer cust`

B) **`Customer()`**

C) `cust`

D) `new`

Explanation

The part of the statement that instantiates the `Customer` object is `new`. The `new` keyword is required to create the object.

`cust` is not the part of the statement that instantiates the `Customer` object. The variable name is `cust`.

`Customer cust` is not the part of the statement that instantiates the `Customer` object. The declaration part of the statement is `Customer cust`.

`Customer()` is not the part of the statement that instantiates the `Customer` object. The initialization part of the statement is `Customer()`.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Declare and instantiate Java objects including nested class objects, and explain objects' lifecycles (including creation, dereferencing by reassignment, and garbage collection)

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Classes and Objects > Creating Objects

---

Question #26 of 50

Question ID: 1327954

Which Java 11 module implements a new method called `repeat` that gives you the ability to duplicate values in a `String`?

A) **`java.compiler`**

B) `java.logging`

C) `java.base`

D) `java.desktop`

Explanation

The Java 11 module `java.base` consists of a package called `java.lang`. This package has several methods, and of the many new methods, one of the them, `repeat`, gives you the ability to duplicate values within a `String`.

The return type of this method consists of a `String` and requires a `count` parameter to determine the number of times the string value is duplicated. The method signature is:

```
String repeat(int count)
```

This is an example using the `String.repeat` method:

```
var Mynewstr = "CyberVista";

var MyRepeatObject = Mynewstr.repeat(2);

System.out.println("My String Value is Repeated = " + MyRepeatObject);
```

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

//My String Value is Repeated = CyberVistaCyberVista
```

The `java.compiler` is used for compiling java source code and consists of java compiler APIs. This module consists of several packages like `javax.tools` and `javax.lang.model`.

The `java.desktop` consists of Java APIs that are related to audio, imaging, and printing.

The `java.logging` is a utility module used for Java 2 platforms that allows developers to debug their applications to provide detailed logging options.

Objective:
Java Platform Module System

Sub-Objective:
Deploy and execute modular applications, including automatic modules

References:

Oracle Documentation > Java SE 11 API > Class String

Question #27 of 50

Question ID: 1328183

You have written a class to store employee records including bank account numbers.

```
public abstract class Employee {
  protected Employee() {
    //Insert code here
  }
  private Employee(Void ignored) {
    // Implementation omitted
  }
  private static Void securityCheck() {
    SecurityManager secured = System.getSecurityManager();
    if (secured != null) {
      secured.checkCreateClassLoader();
    }
    return null;
  }
}
```

What line of code will you insert to disallow malicious subclassing of this class?

A) `PreparedStatement ps = con.prepareStatement(sql);`

B) `this(securityManagerCheck());`

C) `Runtime run = Runtime.getRuntime();`

D) `System.out.println("Check this!");`

Explanation

The following line of code needs to be inserted:

```
this(securityManagerCheck());
```

The other options are incorrect because they do not invoke the `SecurityManager` check, which is required for securing constructors.

Secure construction of confidential classes by ensuring a check by `SecurityManager` each time a class can be instantiated. This check should be done at the beginning of each `public` or `protected` constructor in the class. Checks must also be enforced before any `public static` factory methods and `readObject` or `readObjectNoData` methods of serializable classes.

To secure sensitive objects during construction, you need to hide constructors by using `static` factory methods instead of using `public` constructors. Also, inheritance should be implemented using delegation instead of using inheritance. You also need to avoid cloning and using implicit constructors.

Objective:
Secure Coding in Java SE Application

Sub-Objective:
Secure resource access including filesystems, manage policies and execute privileged code

References:

Oracle Technology Network > Java > Secure Coding Guidelines for Java SE

Question #28 of 50

Question ID: 1327928

Given the code fragment:

```
SortedMap<String,Double> grades = new TreeMap<>();
grades.put("B", 2.67);
grades.put("A", 4.0);
grades.put("F", 0.0);
grades.put("C", 2.0);
grades.put("C-", 2.0);
grades.put("B", 3.33);
```

Which three statements will output a value less than 3.0?

A) `System.out.print(grades.get(grades.firstKey()));`

B) **`System.out.print(grades.get("C-"));`**

C) **`System.out.print(grades.get(grades.lastKey()));`**

D) `System.out.print(grades.get("B"));`

E) **`System.out.print(grades.get("C"));`**

Explanation

The following three statements will output a value less than 3.0:

```
System.out.print(grades.get("C"));
```

```
System.out.print(grades.get("C-"));
```

```
System.out.print(grades.get(grades.lastKey()));
```

The first and second statements will output 2.0, while the last statement will output 0.0. The `lastKey` method retrieves the last sorted key in the Map. `SortedMap` automatically sorts its keys in a natural sequence. In this scenario, keys are sorted by ascending alphabetic letters.

The statement `System.out.print(grades.get("B"));` does not output a value less than 3.0. Because the last `put` method replaces the value for the `B` key, the value 2.67 is replaced with 3.33.

The statement `System.out.print(grades.get(grades.firstKey()));` does not output a value less than 3.0. The value is 4.0 because `SortedMap` automatically sorts `A` as the first key.

Objective:
Working with Arrays and Collections

Sub-Objective:
Use a Java array and List, Set, Map and Deque collections, including convenience methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Collections > Interfaces > The Map Interface

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Collections > Implementations > Map Implementations

---

Question #29 of 50

Question ID: 1327948

Given the following:

```
int[][][] matrix = new int[5][5][5];
```

Which line of code sets the first element of the matrix array?

A) `matrix[[1][1][1]] = 42;`

B) `matrix[0,0,0] = 42;`

C) **`matrix[0][0][0] = 42;`**

D) `matrix[[0][0][0]] = 42;`

E) `matrix[1][1][1] = 42;`

F) `matrix[1,1,1] = 42;`

Explanation

The following line of code sets the first element of the matrix array:

```
matrix[0][0][0] = 42;
```

When accessing an element in a multidimensional array, each dimension index should be specified in separate square bracket pairs. Indexes in arrays are zero-based, so that the first element is always located at position 0. The first element of a multidimensional array is accessed by specifying 0 for each of its dimension indices.

The lines of code that specify position 1 for each dimension index will not set the first element. Indexes in arrays are zero-based, so that the first element is always located at position 0. Specifying 1 for each dimension index will access the second element within the second inner array of the second outer array.

The lines of code that use commas and additional square brackets will not set the first element. These lines of code contain syntax errors and will fail to compile.

Objective:
Working with Arrays and Collections

Sub-Objective:
Use a Java array and List, Set, Map and Deque collections, including convenience methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Arrays

---

Question #30 of 50

Question ID: 1328141

Given:

```
Stream.of(new Student("Fred"), new Student("Jim"))
  .reduce(/*expression here*/)
```

Which statement(s) are true of the expression that must be provided at the point `/*expression here*/`? (Choose all that apply.)

A) The behavior is expected to mutate data.

B) The behavior can safely modify the `Student` objects.

C) The expression must implement `BinaryOperator<Student>`.

D) The behavior must be associative.

Explanation

The expression must implement `BinaryOperator<Student>` and its behavior must be associated. The `reduce` operation requires that the provided argument implements `BinaryOperator<T>`, where *T* is the data type of the stream. The behavior provided in the implementation of the `BinaryOperator<T>` must adhere to certain constraints. One of these is that the behavior must be associative. This means that for an operation called *op(a,b)* given the values *A*,*B*,*C*, then *op(A, op(B,C))* must be equal to *op(B, op(A,C))*. In other words, the order in which the operation is applied must not affect the correctness of the result.

The behavior is not expected to mutate data. Accessing mutable state should be avoided because it forces undesirable behavior, either non-deterministic memory behavior, or non-scalable thread synchronization.

The behavior should not modify the Student objects. Side effects, such as modifying data, whether in the stream or elsewhere, should be avoided.

Objective:
Working with Streams and Lambda expressions

Sub-Objective:
Perform decomposition and reduction, including grouping and partitioning on sequential and parallel streams

References:

The Java Tutorials > Collections > Aggregate Operations > Parallelism

Java Platform Standard Edition 11 > API > java.util.stream > Stream

---

Question #31 of 50

Question ID: 1327818

Given:

```
public class Circle {
   public double getCircumference(double radius ) {
     return java.lang.Math.PI * 2 * radius;
   }
   public static double getArea(double radius) {
```

```
    return java.lang.Math.PI * radius * radius;
  }
}
```

Which two code fragments will fail compilation?

A) `Circle.getCircumference(10.5);`

B) `double c = Circle.getCircumference(10.5);`

C) `new Circle().getCircumference(10.5);`

D) `Circle.getArea(5.5);`

E) `new Circle().getArea(5.5);`

F) `double a = new Circle().getArea(5.5);`

Explanation

The following code fragments will fail compilation:

```
Circle.getCircumference();
double c = Circle.getCircumference(10.5);
```

These code fragments fail compilation because the `getCircumference` method is an instance method and cannot be invoked as a `static` class method. The code must instantiate the `Circle` class before accessing the `getCircumference` method.

The other code fragments will compile because the `getCircumference` method is accessible from an instance context and the `getArea` method is accessible from either a `static` or instance context.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Define and use fields and methods, including instance, static and overloaded methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Understanding Instance and Class Members

---

Question #32 of 50

Question ID: 1327934

Which statement is true about using `ArrayList` objects?

A) `ArrayList` objects are fixed in size and not resizable like arrays.

B) `ArrayList` objects require less memory than arrays.

C) `ArrayList` objects provide better performance than arrays.

D) **`ArrayList` objects require less low-level implementation than arrays.**

Explanation

`ArrayList` objects require less low-level implementation than arrays. Classes in the Collections Framework, like `ArrayList`, provide implementation for useful data structures and algorithms. For example, rather than implementing the logic for creating a new array when more elements are required, an `ArrayList` object automatically resizes to accommodate additional elements.

`ArrayList` objects do not require less memory than arrays. `ArrayList` objects provide a variety of additional functionality over arrays, which increases its memory footprint.

`ArrayList` objects do not provide better performance than arrays. Although the Collections Framework provides implementation to reduce development time, collections do not provide better performance than simple arrays. However, complex algorithms for sorting and handling elements provide better performance in collections than if they were implemented manually using an array.

`ArrayList` objects are not fixed in size and arrays are not resizable. Arrays are fixed in size, while `ArrayList` objects are resizable.

Objective:
Working with Arrays and Collections

Sub-Objective:
Use a Java array and List, Set, Map and Deque collections, including convenience methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Collections > Lesson: Introduction to Collections

Oracle Documentation > Java SE 11 API > Class ArrayList<E>

Question #33 of 50

Question ID: 1327958

Consider the following service interface implementation:

```
package package2;
import package1.SpeakerInterface;

public class SpeakerImplementer implements SpeakerInterface {
    @Override
    public void speak() {
```

```
        System.out.println("Speaking from SpeakerImplementer!");
    }
}
```

Which fragment of code will correctly create a service provider module?

A)
```
module modPro {
  requires modServ;
  provides package1.SpeakerInterface with
package2.SpeakerImplementer;
}
```

B)
```
module modPro {
  requires modServ;
  package package2;
}
```

C)
```
module modPro {
  requires modServ;
  import package2.SpeakerImplementer;
}
```

D)
```
module modPro {
  requires modServ;
  import package1;
}
```

Explanation

You should use the fragment of code that contains the `provides` and `with` keywords:

```
provides package1.SpeakerInterface with package2.SpeakerImplementer;
```

The other options do not correctly implement the service provider functionality, which requires the use of the `provides` and `with` keywords to the exact service interface to be used.

The `java.util.ServiceLoader` class allows for the use of the Service Provider Interface (SPI), or Service for short, and for implementations of this class, called Service Providers. Java 9 and onwards allows the development of Services and Service Providers as modules. Service modules declare several interfaces whose implementations are provided by provider modules at runtime. Each provider module declares the specific implementations of Service modules that it is providing. Every module that finds and loads Service providers needs a `uses` directive within its declaration.

For any service to be used successfully, each of its service providers is required to be found and then loaded. The `ServiceLoader` class exists for this purpose and performs this function. Any module that is created to find and load service providers requires the `uses` keyword as a directive within its declaration specifying the service interface it uses.

Objective:
Java Platform Module System

Sub-Objective:
Declare, use, and expose modules, including the use of services

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 > API Specification > java.base > java.util > java.lang > Class ServiceLoader<S>

Oracle Technology Network > Java SE 9 and JDK 9 > ServiceLoader

---

Question #34 of 50

Question ID: 1328005

Consider the following code:

```
public class AnnotationsTest {
 // Insert annotation here
 public void play() {
  System.out.println("Time to play");
 }
 public static void main(String args[])  {
  AnnotationsTest at = new AnnotationsTest();
 at.play();
 }
}
```

The `play()` method is now obsolete and replaced by a newer implementation. Which annotation will you insert for it?

A) `@Inherited`

B) **`@Deprecated`**

C) `@Documented`

D) `@SupressWarnings`

Explanation

You should use the `@Deprecated` annotation because the `play` method is no longer in use. `@Deprecated` is a marker annotation indicating to the compiler that the associated declaration is has now been replaced with a newer one.

`@Documented` is an incorrect option. It does not mark obsolete code but indicates Javadoc to include annotations.

`@Inherited` is an incorrect option. It does not mark obsolete code but inheritance by a subclass. It can be used on class declarations.

`@SupressWarnings` is an incorrect option. It does not mark obsolete code by specifies exact warnings that the compiler needs to ignore.

Java has seven predefined annotation types:

- `@Retention` – This indicates how long an annotation is retained. It has three values: SOURCE, CLASS, and RUNTIME.
- `@Documented` – This indicates to tools like Javadoc to include annotations in the generated documentation, including the type information for the annotation.
- `@Target` – This is meant to be an annotation to another annotation type. It takes a single argument that specifies the type of declaration the annotation is for. This argument is from the enumeration `ElementType`:
  - ANNOTATION_TYPE – This is used for another annotation
  - CONSTRUCTOR – For constructors
  - FIELD – For fields
  - METHOD – For methods
  - LOCAL VARIABLE – For local variables
  - PARAMETER – For parameters
  - PACKAGE – For packages
  - TYPE – This can include classes, interfaces or enumerations
- `@Inherited` – This can only be used on annotation declarations. It makes an annotation for a superclass become inherited by a subclass. It is used only for annotations on class declarations.
- `@Deprecated` – This is a marker annotation indicating that the associated declaration is has now been replaced with a newer one.
- `@Override` – This is a marker annotation only to be used with methods that override methods from the parent class. It helps ensure methods are overridden and not just overloaded.
- `@SupressWarnings` – This specifies warnings in string form that the compiler must ignore.

Objective:
Annotations

Sub-Objective:
Create, apply, and process annotations

References:

Oracle Technology Network > Java SE Documentation > Annotations

Oracle Technology Network > Java SE Documentation > Annotations > Predefined Annotation Types

---

Question #35 of 50

Question ID: 1327835

Given:

```
class CardDeck {
   public CardDeck get(int suits, boolean includeJokers) { /*code omitted*/ }
}
```

Which change will prevent other classes from directly instantiating the `CardDeck` class?

A) Adding the `private` modifier to the class

B) **Adding a parameterless constructor with no modifier to the class**

C) Adding a parameterless constructor with the `private` modifier to the class

D) Changing the `public` modifier to `protected` on the `get` method

E) Changing the `public` modifier to `private` on the `get` method

F) Adding the `protected` modifier to the class

Explanation

Adding a parameterless constructor with the `private` modifier to the class will prevent other classes from directly instantiating the `CardDeck` class. To prevent the compiler from automatically providing a default constructor, you must first declare a constructor in the `CardDeck` class. To prevent other classes from accessing the constructor, you must apply the `private` modifier.

Adding the `private` or `protected` modifier to the class would not prevent instantiation. A modifier at the class level will affect access to all members within the class, not only constructors. Also, the modifiers `private` or `protected` are not valid for a class unless they are nested within another class.

Adding a parameterless constructor with no modifier to the class will not prevent all other classes from directly instantiating the class. The default access level will allow instantiation by all classes within the same package.

Changing the `public` modifier to `protected` or `private` on the `get` method will not prevent instantiation. These modifiers would only affect access to the `get` method, not access to the default constructor that is provided automatically by the compiler.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Understand variable scopes, apply encapsulation and make objects immutable

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Controlling Access to Members of a Class

---

Question #36 of 50

Question ID: 1328041

Given:

```
public void copy(Path srcFile, Path destFile) {
   try {
     byte[] readBytes = Files.readAllBytes(srcFile);
     Files.write(destFile, readBytes);
   } catch (_______________ e ) {
     System.err.println(e.toString());
   }
}
```

Which insertion will allow the code to compile?

A) **`IOException`**

B) `Error`

C) `FileSystemNotFoundException`

D) `IOError`

E) `FileNotFoundException`

Explanation

To allow the code to compile, the insertion should specify the class `IOException`. Although it was not given as an alternative, the insertion could also specify its superclass `Exception`. The `readAllBytes` and write methods specify that they throw `IOException`, so any invocations of those methods must either catch `IOException` or specify that the parent method throws that exception. This is known as a checked exception. Checked exceptions include any exceptions, except for `RuntimeException` and its subclasses.

The code will not compile if the insertion specifies `Error` or `IOError`. The `readAllBytes` and `write` methods require handling `IOException` or its superclass `Exception`. Errors are abnormal conditions external to the application and often are unrecoverable. The compiler does not check catching and specifying `Error` and its subclasses.

The code will not compile if the insertion specifies `FileNotFoundException`. The `readAllBytes` and `write` methods require handling `IOException` or its superclass `Exception`. Subclasses of the `IOException` can be specified in the `catch` clause, but the more general `IOException` must be also caught, or the code will not compile.

The code will not compile if the insertion specifies `FileSystemNotFoundException`. The `readAllBytes` and `write` methods require handling `IOException` or its superclass `Exception`. Runtime exceptions are abnormal conditions internal to the application and often are unrecoverable. The compiler does not check catching and specifying `RuntimeException` and its subclasses.

Objective:

Exception Handling

Sub-Objective:

Handle exceptions using try/catch/finally clauses, try-with-resource, and multi-catch statements

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Exceptions > The Catch or Specify Requirement

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Exceptions > The catch Blocks

---

Question #37 of 50

Question ID: 1327979

Consider the following code:

```
public class Serializer {
  public static void main(String[] args) {
    Employee e1 = new Employee("Josh", 30, "Male");
    Employee e2 = new Employee("Ann", 30, "Female");

    try {
      FileOutputStream fo = new FileOutputStream(new File("empObjects.txt"));
      ObjectOutputStream oo = new ObjectOutputStream(fo);
      oo.writeObject(e1);
      oo.writeObject(e2);
      oo.close();
      fo.close();

      FileInputStream fin = new FileInputStream(new File("empObjects.txt"));
      ObjectInputStream oin = new ObjectInputStream(fin);
      String emp1 = oin.readObject();
      String emp2 = oin.readObject();
      System.out.println(emp1);
      System.out.println(emp2);
      oin.close();
      fin.close();

    } catch (FileNotFoundException e) {
      System.out.println("File was not found");
    } catch (IOException e) {
      System.out.println("Stream" could not be initialized);
    } catch (ClassNotFoundException cnfe) {
```

```
      cnfe.printStackTrace();
    }
  }
}
```

What would be the output of this code?

A) `ClassNotFoundException`

B)
```
Josh
30
Male
Ann
30
Female
```

C) `FileNotFoundException`

D) **`IOException`**

Explanation

The code will throw a `ClassNotFoundException` because the `Employee` object is not cast correctly as a `String` object. Java objects are converted into a stream of bytes by using `java.io.ObjectOutputStream`. This allows you to write objects to a file. For this to happen, the class performing the operation needs to implement the `Serializable` interface. When an object is read, the `ObjectInputStream` class attempts to match each attribute of the object being read into the attributes of the class it is being cast into. If the matching does not happen correctly, a `ClassNotFoundException` exception is thrown.

The following output is not displayed because an exception is thrown instead:

```
Josh
30
Male
Ann
30
Female
```

The code will not throw `IOException` or `FileNotFoundException` exceptions because it throws the `ClassNotFoundException`.

Serialization is a process of converting objects to a stream of bytes that can be written. Objects converted to a byte stream can then be written to a file. The Java library provides Serialization API for this process. A Java object is serializable if its class or superclass implements the `java.io.Serializable` or `java.io.Externalizable` interfaces.

Objective:

Java File I/O

Sub-Objective:

Implement serialization and deserialization techniques on Java objects

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 > API Specification > java.base > java.io > Interface Serializable

Oracle Technology Network > Software Downloads > Documentation > Java Object Serialization

Oracle Java Documentation > Learning the Java Language > Interfaces and Inheritance > Interfaces

---

Question #38 of 50

Question ID: 1327918

Given:

```
public enum GPA {
  LOW, MID, HIGH, TOP;
  public static void main(String[] args) {
    System.out.println(GPA.MID);
  }
}
```

What is the result?

A) Compilation fails.

B) An exception is thrown at runtime.

C) `MID`

D) `1`

E) `2`

Explanation

The result of the output is `MID`. The implicit invocation of the `toString` method will output the name of the enumeration constant because `toString` is overridden in the `Enum` class. The return of `toString` is the same as the `name` method.

The result is not the output `1`. This would be the output if the ordinal method were invoked. The ordinal method returns the zero-based index of the enumeration constant.

The result is not the output `2`. This would be the output if the ordinal method were invoked with the constant `GPA.HIGH`, not `GPA.MID`.

The result is not an exception at runtime or failure in compilation. There are no unexpected arguments or invalid syntax in the given code.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Create and use enumerations

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Classes and Objects > Enum Types

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition > 8 API Specification > java.lang > Class Enum

---

Question #39 of 50

Question ID: 1327857

Which two statements are true about abstract and concrete types?

A) An abstract type can be an abstract class or interface.

B) Subinterfaces of an interface must use the `implements` keyword.

C) Subclasses of an interface class must use the `extends` keyword.

D) A concrete type cannot contain abstract members.

E) Subclasses of an abstract class must use the `implements` keyword.

F) An abstract type cannot contain implementation.

Explanation

The following two statements are true about abstract and concrete types:

- An abstract type can be an abstract class or interface.
- A concrete type cannot contain abstract members.

Both interfaces and abstract classes are abstract types. Interfaces cannot contain implementation, constructors, or non-final instance fields, whereas abstract classes can contain these things. A concrete type cannot contain abstract members, whereas abstract classes and interfaces can contain these things.

An abstract class can contain implementation, while an interface cannot.

Subclasses of an abstract class and subinterfaces of an interface must not use the `implements` keyword. The `extends` keyword should be used instead. Subclasses of abstract classes must use inheritance to provide implementation, whereas subtyping between interfaces also uses the `extends` keyword.

Subclasses of an interface class must use the `implements` keyword, not the `extends` keyword. Whether the subclass is abstract or concrete, a class that uses an interface must use the `implements` keyword. The `extends` keyword is only used between classes or between interfaces.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Create and use subclasses and superclasses, including abstract classes

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Interfaces and Inheritance > Abstract Methods and Classes

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Interfaces and Inheritance > Implementing an Interface

---

Question #40 of 50

Question ID: 1328067

Which two components of a class declaration are used in a lambda expression? (Choose two.)

A) **Name of the method being implemented**

B) **Expression representing the value to be returned by the method being implemented**

C) Name of the interface being implemented

D) **Argument list of the method being implemented**

E) Declaration of the type being returned by the method being implemented

Explanation

The two components of a class declaration used in a lambda expression are an argument list of the method being implemented and an expression representing the value to be returned by the method being implemented. The basic form of a lambda is simply the argument list to the method being implemented, an arrow symbol (`->`), and the expression that is to be calculated and returned by the method.

The names of the interface, the method being implemented, and type returned by the method are inferred from the context and are not specified, so they are not used in the lambda expression.

Objective:
Working with Streams and Lambda expressions

Sub-Objective:
Implement functional interfaces using lambda expressions, including interfaces from the java.util.function package

References:

The Java Tutorials > Learning the Java Language > Classes and Objects > Syntax of Lambda Expressions

---

Question #41 of 50

Question ID: 1328157

Which three collections are available to avoid memory consistency errors in a multi-threaded application?

A) `ArrayList`

B) `TreeMap`

C) `HashMap`

D) `CyclicBarrier`

E) **`BlockingQueue`**

F) **`ConcurrentHashMap`**

G) **`CopyOnWriteArrayList`**

Explanation

The three collections are `BlockingQueue`, `ConcurrentHashMap`, and `CopyOnWriteArrayList`. To prevent memory consistency errors, a resource must ensure that write operations are visible to all threads when they occur, so that subsequent operations are consistent known as a *happens-before* relationship. The `java.util.concurrent` package includes collections that define this relationship when adding an element for subsequent reading and deleting operations. Thus, these collections and their operations are safe to be used by multi-threaded applications.

`ArrayList` is not implemented to avoid memory consistency errors. The thread-safe version of `ArrayList` that avoids memory consistency errors is `CopyOnWriteArrayList`.

`HashMap` is not implemented to avoid memory consistency errors. The thread-safe version of `HashMap` that avoids memory consistency errors is `ConcurrentHashMap`.

`TreeMap` is not implemented to avoid memory consistency errors. The thread-safe version of `TreeMap` that avoids memory consistency errors is `ConcurrentSkipListMap`.

`CyclicBarrier` is not a collection, but a synchronization aid for handling threads. A `CyclicBarrier` determines a size of specific number of waiting threads, which once exceeded, triggers an optional predefined action. The `await` method will pause the thread until all other threads in the barrier have been paused, while the `reset` method reinitializes the barrier.

Objective:
Concurrency

Sub-Objective:
Create worker threads using Runnable and Callable, and manage concurrency using an ExecutorService and java.util.concurrent API

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.util.concurrent > Class CopyOnWriteArrayList<E>

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Concurrent Collections

---

Question #42 of 50

Question ID: 1328158

Which statement is true about collections in the `java.util.concurrent` package, such as `BlockingQueue` and `CopyOnWriteArrayList`?

A) Read access to these collections are not thread-safe.

B) **Read and write access to these collections is thread-safe.**

C) Deletion of elements in these collections is not thread-safe.

D) Copying these collections is thread-safe.

Explanation

Read and write access to collections in the `java.util.concurrent` package is thread-safe. To prevent memory consistency errors, a resource must ensure that write operations are visible to all threads when they occur, so that subsequent operations are consistent known as a *happens-before* relationship. The `java.util.concurrent` package includes collections that define this relationship when adding an element for subsequent reading and deleting operations. These collections and their operations are safe to be used by multi-threaded applications.

Read access, in addition to write access, to these collections is thread-safe.

Deletion of elements is included in write access. Deletion of elements in these collections is thread-safe.

Copying these collections is not thread-safe. Only read/write access is thread-safe, not copying the collection itself.

Objective:
Concurrency

Sub-Objective:
Create worker threads using Runnable and Callable, and manage concurrency using an ExecutorService and java.util.concurrent API

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Concurrent Collections

Question #43 of 50

Question ID: 1327792

Given the following:

```
public class MyBasicClass {
   //Insert code here
}
```

Which three lines of code can be included in the class?

A) `import java.text.*;`

B) `module basicModule {}`

C) **`enum ClassType {basic, advanced}`**

D) `package basicPackage;`

E) **`void BasicMethod() {}`**

F) **`static final int VAL=1000;`**

Explanation

The three lines of code can be included in the class as follows:

```
public class MyBasicClass {
  enum ClassType {basic, advanced}
  void BasicMethod() {}
  static final int VAL=1000;
}
```

A class body can include static and non-static fields, methods, constructors, and nested enumerations and classes.

The code line `package basicPackage;` cannot be included in the class because a `package` statement must be the first executable line in the source file, not inside a class body.

The code line `import java.text.*;` cannot be included in the class because `import` statements are not allowed in class bodies.

The code line `module basicModule {}` cannot be included in the class source file, but must be included in a separate `module-info.java` file.

Objective:
Java Object-Oriented Approach

Sub-Objective:

Declare and instantiate Java objects including nested class objects, and explain objects' lifecycles (including creation, dereferencing by reassignment, and garbage collection)

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Classes and Objects > Summary of Creating and Using Classes and Objects

Oracle.com > Java 9 | Excerpt > Understanding Java 9 Modules

Question #44 of 50

Question ID: 1327804

Given the following:

```
public class SmartPhone {
   float screenResolution, width, height;
   public static void main (String[] args) {
     SmartPhone phone;
     phone.height = 112.2f;
     phone.width = 56.8f;
     System.out.format("%.0f dpi - %.1f X %.1f",
     phone.screenResolution, phone.height, phone.width);
   }
}
```

What is the result?

A) `0 dpi - 112.2 X 56.8`

B) `null dpi - 112.2 X 56.8`

C) A runtime error is produced.

D) **A compile error is produced.**

Explanation

A compile error is produced because the `phone` variable does not reference an instantiated `SmartPhone` object. Local variables in methods are not provided default values automatically. To access instance members in a class, you must first instantiate the class and access the field using dot notation by referencing the instance.

The result is not `null dpi - 112.2 X 56.8` because the code does not compile. If the `phone` variable did reference an instantiated `SmartPhone` object, then the `screenResolution` field would not be `null`. Instance and `static` fields are provided default values automatically. The default value for the `float` type is 0.0.

The result is not `0 dpi - 112.2 X 56.8` because the code does not compile. If the `phone` variable did reference an instantiated `SmartPhone` object, then this would be the output.

A runtime error is not produced because the code does not compile. If the `phone` variable were set to `null`, then a `NullPointerException` would be thrown.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Define and use fields and methods, including instance, static and overloaded methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Classes and Objects > Using Objects

---

Question #45 of 50

Question ID: 1327837

Given:

```
public class Card {
   enum Suit {CLUB, DIAMOND, HEART, SPADE}
   enum Color {BLACK, RED}
   //add code here
}
```

Which two field declarations, when inserted in the code, demonstrate good encapsulation principles?

A) `private Color color;`

B) `public Color color;`

C) `private Suit suit;`

D) `public Suit suit;`

E) `Suit suit;`

F) `Color color;`

Explanation

The following two field declarations, when inserted in the code, demonstrate good encapsulation principles:

```
private Suit suit;
private Color color;
```

Both field declarations use the `private` modifier, which should be the default for all fields. Access through methods, rather than directly to fields, is a core principle of encapsulation. Thus, using the most restrictive access modifier `private` is recommended unless there is some reason to make an exception.

The field declarations without an access modifier, when inserted in the code, do not demonstrate good encapsulation principles. Without an access modifier, fields are directly accessible to classes within the same package.

The field declarations with the `protected` modifier, when inserted in the code, do not demonstrate good encapsulation principles. The access modifier `private` is the recommended access modifier for all fields, with only `protected` applied to fields that must be directly accessible to subclasses.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Understand variable scopes, apply encapsulation and make objects immutable

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Controlling Access to Members of a Class

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Object-Oriented Programming Concepts > What Is an Object?

Question #46 of 50

Question ID: 1327961

Consider the following module definition

```
module modProv {            // Line 1
    requires modServ;         // Line 2
    provides package1.SpeakerInterface
  with package2.SpeakerImplementation;  //Line 3
    exports package2;          // Line 4
}:
```

Which line of code is not an appropriate way of defining the module?

A) Line 1

B) Line 2

C) Line 3

D) **Line 4**

Explanation

Line 4 is inappropriate because a service provider that is developed as a module must *not* export the implementation of the service. There exists no support for modules that specify other service providers in separate modules via a `provides` directive.

Line 1 is appropriate because it is a valid usage of the `module` keyword for definition of a module.

Line 2 is appropriate because it is a valid usage of the `requires` keyword needed for specifying a necessary module.

Line 3 is appropriate because it contains necessary uses of the `provides` and `with` keywords, both of which are necessary for finding and loading service providers.

When service providers are deployed as modules, they need to specified using the `provides` keyword within the declaration of the module. Using the `provides` directive helps specify the service as well as the service provider. This directive helps to find the service provider if another module that has a `uses` directive for the same service gets a service loader for that service.

Service providers that are created inside modules cannot control when they are instantiated. However, if the service provider declares a `provider` method, then the service loader will invoke an instance of the service provider via that method. If a `provider` method is not declared in the service provider, there is a direct instantiation of the service provider by the service provider's constructor. In this case the service provider needs to be assignable to the class or interface of the service. A service provider that exists as an automatic module inside an application's module path needs a provider constructor.

Objective:
Java Platform Module System

Sub-Objective:
Declare, use, and expose modules, including the use of services

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 > API Specification > java.base > java.util > java.lang > Class ServiceLoader<S>

Oracle Technology Network > Java SE 9 and JDK 9 > ServiceLoader

---

Question #47 of 50

Question ID: 1328159

Given the code fragment:

```
List<String> empList = new ArrayList<String>();
```

Which two changes will make read/write operations on `empList` thread-safe?

A) The `ArrayList` class should be replaced with the `CopyOnWriteArrayList` class.

B) The `ArrayList` class should be replaced with the `ConcurrentHashMap` class.

C) The code fragment should invoke the `CyclicBarrier.await` method on the current thread.

D) The code fragment should be declared in a synchronized block.

E) The code fragment should invoke the `Thread.interrupt` method on the current thread.

F) Read/write operations should be declared in a synchronized block.

G) The code fragment should invoke the `Thread.sleep` method.

Explanation

To ensure that read/write operations on `empList` are thread-safe, you should make one of the following two changes:

- Read/write operations should be declared in a synchronized block.
- The `ArrayList` class should be replaced with the `CopyOnWriteArrayList` class.

To prevent memory consistency errors, a resource must ensure that write operations are visible to all threads when they occur, so that subsequent operations are consistent known as a *happens-before* relationship. You can either use the thread-safe collection for the `ArrayList` class that provides automatic synchronization, namely `CopyOnWriteArrayList`, or perform manual locking using a synchronized block.

The code fragment should not be declared in a synchronized block. Instantiation within a synchronized block will not ensure that read/write operations are thread-safe.

The `ArrayList` class should not be replaced with the `ConcurrentMap` class. The thread-safe version of `ArrayList` is the `CopyOnWriteArrayList` collection, while the interface `ConcurrentMap` is the thread-safe subinterface of `Map`.

The code fragment should not invoke the `Thread.sleep` method. This method will not make read/write operations thread-safe. The `Thread.sleep` method will only pause execution of the current thread.

The code fragment should not invoke the `Thread.interrupt` method on the current thread. This method will not make read/write operations thread-safe. The `Thread.interrupt` method will only terminate or redirect execution a thread.

The code fragment should not invoke the `CyclicBarrier.await` method on the current thread. This is an instance method requiring an initialized `CyclicBarrier` object. A `CyclicBarrier` determines a size of specific number of waiting threads, which once exceeded, triggers an optional predefined action. The `await` method will pause the thread until all other threads in the barrier have been paused.

Objective:
Concurrency

Sub-Objective:

Create worker threads using Runnable and Callable, and manage concurrency using an ExecutorService and java.util.concurrent API

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.util.concurrent > Class CopyOnWriteArrayList<E>

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Intrinsic Locks and Synchronization

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Concurrent Collections

---

Question #48 of 50

Question ID: 1327807

Given the following code fragment:

```
public class StandardMethods {
   public static void printPerimeter(double... sides) {
     double result = 0;
     for (double side: sides) {
       result += side;
     }
     System.out.println("Perimeter is " + result);
   }
}
```

Assuming the given code is in the same package, which code lines will compile? (Choose all that apply.)

A) `double perimeter = StandardMethods.printPerimeter();`

B) **`StandardMethods.printPerimeter();`**

C) `double perimeter = StandardMethods.printPerimeter(7.5, 9.8, 11);`

D) **`StandardMethods.printPerimeter(7.5, 9.8, 11);`**

Explanation

The following code lines will compile:

```
StandardMethods.printPerimeter();
StandardMethods.printPerimeter(7.5, 9.8, 11);
```

Both code lines invoke the `printPerimeter` method without expecting a return value because the method declares `void`. Because varargs (`...`) is used for the `printPerimeter` method, an arbitrary number of arguments can be specified during invocation. The first code line invokes `printPerimeter` with no arguments, while the second code line invokes `printPerimeter` with three arguments.

The code lines that attempt to retrieve a return value will not compile. The `getSurfaceArea` method declares `void`, so no return value is expected. The compiler will indicate that the variable `perimeter` of the `double` type is not compatible with the `void` type.

Objective:
Java Object-Oriented Approach

Sub-Objective:
Define and use fields and methods, including instance, static and overloaded methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Passing Information to a Method or a Constructor

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Classes and Objects > Defining Methods

---

Question #49 of 50

Question ID: 1328060

Given:

```
import java.io.*;
public class EmployeeStreamReader {
  private FileInputStream fileStream;
  public EmployeeStreamReader(String filename) throws FileNotFoundException{
    fileStream = new FileInputStream(filename);
  }
  public Employee read() throws IOException {
    StringBuilder strData = new StringBuilder("");
    int ch;
    while( (ch = fileStream.read()) != -1) {
      strData.append((char)ch);
    }
    return new Employee(strData.toString());
  }
  public void close() throws IOException {
    fileStream.close();
```

```
    fileStream = null;
  }
}
```

Which modification should you apply to the class so that it can be used in a try-with-resources statement?

A) Remove the `throws` clause from the `close` method.

B) Remove the `throws` clause from the class constructor.

C) Add an `open` method to the class.

D) **Make the class implement the `AutoCloseable` interface.**

Explanation

To make the class usable in a try-with-resources statement, you should make the class implement the `AutoCloseable` interface. The following try-with-resources block will now compile:

```
Employee emp = null;
try (
  EmployeeStreamReader reader = new EmployeeStreamReader("emp.dat")) {
  emp = reader.read();
} catch (Exception ex) {
  System.err.print(ex.getMessage());
}
```

You should not add an `open` method to the class. An `open` method is not required of classes declared in a try-with-resources statement.

You do not need to remove the `throws` clause from the `close` method. The `close` method may either handle or throw an Exception, and the `FileInputStream.close` method throws an `IOException`.

You do not need to remove the `throws` clause from the class constructor. The `FileInputStream` constructor throws the `FileNotFoundException`, and must be declared in a `throws` clause if not handled.

Objective:
Exception Handling

Sub-Objective:
Handle exceptions using try/catch/finally clauses, try-with-resource, and multi-catch statements

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 8 API Specification > java.lang > Interface AutoCloseable

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Exceptions > The try-with-resources Statement

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Exceptions > Specifying the Exceptions Thrown by a Method

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 8 API Specification > java.io > Class FileInputStream

---

Question #50 of 50

Question ID: 1327943

Which statement is true about array declarations?

A) Curly braces can contain a comma-delimited list of dimensions.

B) Square brackets can be placed after the data type or variable name.

C) Square brackets can contain a comma-delimited list of values.

D) **An array's dimension can be set without using the `new` keyword.**

Explanation

Square brackets can be place after the data type or variable name. The following code demonstrates placing square brackets after the data type:

```
String[] strArray;  //preferred
```

The following code demonstrates placing square brackets after the variable name:

```
String strArray[];  //not encouraged, but legal syntax
```

The compiler will accept either style to declare an array without specifying a dimension.

An array's dimension must be set using the `new` keyword. The dimension should be declared in the second pair of square brackets that proceed the `new` keyword.

Curly braces cannot contain a comma-delimited list of dimensions. Curly braces contain a comma-delimited list of values for both one-dimensional and multidimensional arrays.

Square brackets cannot contain a comma-delimited list of values. The second pair of square brackets after the new keyword contains the dimension.

Objective:
Working with Arrays and Collections

Sub-Objective:
Use a Java array and List, Set, Map and Deque collections, including convenience methods

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Arrays