Quick Quiz July 14, 2022

Test ID: 216184549

Question #1 of 50

Question ID: 1328000

Which type of object should you use to execute stored procedures?

A) `ResultSet`

B) `FilteredRowSet`

C) `CallableStatement`

D) `Statement`

E) `CachedRowSet`

F) `PreparedStatement`

Explanation

You should use the `CallableStatement` interface to execute stored procedures. The correct option is to use the following code:

```
String storedProc = "{call addAgent(?, ?, ?)}";
CallableStatement callSt = con.prepareCall(sqlQuery);
callSt.setString(1, "James");
callSt.setString(2, "Bond");
callSt.setLong(3, 007);
callSt.addBatch();
callSt.setString(1, "Alec");
callSt.setString(2, "Trevelyan");
callSt.setLong(3, 006);
callSt.addBatch();
int[] res = callSt.executeBatch();
```

`CallableStatement` allows you to store three types of parameters for stored procedures:

1. *in* parameters are sent as input values to stored procedures.
2. *out* parameters are used to store the output and result parameters returned from the stored procedure.
3. *in-out* parameters support bi-directional input and output with the stored procedure.

Note that only in-out and out parameters must be registered before calling the stored procedure using the `CallableStatement` interface.

You should not use `Statement` interface or its subinterface `PreparedStatement` because they do not support stored procedures.

You should not use `ResultSet` or its subinterfaces `CachedRowSet` and `FilteredRowSet` because these interfaces are used for iterating through returned rows with read operations. `CachedRowSet` and `FilteredRowSet` are examples of disconnected interfaces, supporting read operations after the connection has been closed.

Objective:
Database Applications with JDBC

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

References:

Oracle Technology Network > MySQL Connector > Using JDBC CallableStatements to Execute Stored Procedures

---

Question #2 of 50

Question ID: 1327863

Given the following class:

```
class Fan {
   void sight(String celebrity);
   abstract String meet(String celebrity);
}
```

Which statement is true about `Fan`?

A) The `sight` method is implicitly public.

B) The `sight` method is implicitly abstract.

C) The class is implicitly public.

D) The class is implicitly abstract.

E) **The `meet` method is implicitly package-level.**

Explanation

The `meet` method of the `Fan` class is implicitly package-level. When no access modifier is specified, the default access is package-level.

The class is not implicitly abstract. A class that contains an abstract method must be declared as `abstract`. Because the `Fan` class contains the abstract method `meet` and is not declared as abstract, the `Fan` class will fail compilation.

The class and sight methods are not implicitly public. When no access modifier is specified, the default access is package-level.

The `sight` method is not implicitly abstract. In a class, only those members that are declared as `abstract` are abstract. Because the `sight` method does not contain a method body, the `Fan` class will fail compilation.

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 > Classes and Objects > Controlling Access to Members of a Class

---

Question #3 of 50

Question ID: 1328001

Consider the following code:

```
public class Vader {
  public void speak() {
    System.out.println("Luke, I am your father.");
  }
}

public class Luke extends Vader {
  @Override
  public void speak(int x) {
    System.out.println("It’s not true!");
  }
}

public class Jedi {
  public static void main(String args[]) {
    Luke skyW = new Luke();
    skyW.speak();
  }
}
```

What would be the result of compiling the code above?

A) `Luke, I am your father.`

B) **Runtime error occurs.**

C) `It's not true!`

D) Compilation fails.

Explanation

A compiler error would be generated because a method annotated as `@Override` does not override the method in the parent class correctly. An error of the following kind would be displayed:

`Method does not override or implement a method from a supertype.`

To ensure correct compilation, the parameter `int x` needs to be removed from the argument list in the overridden `speak()` method.

The other options are incorrect because a compiler error would occur, and so none of the messages in the other options would be displayed.

Annotations in Java provide metadata for the code and can be used to keep instructions for the compiler. They can also be used to set instructions for tools that process source code. Annotations start with the `@` symbol and attach metadata to parts of the program like variables, classes, methods, and constructors, among others.

Objective:
Annotations

Sub-Objective:
Create, apply, and process annotations

References:

Oracle Technology Network > Java SE Documentation > Annotations

---

Question #4 of 50

Question ID: 1327851

Given:

```
public class OutputSuperClass {
   OutputSuperClass() {
     System.out.println("Super");
   }
}

public class OutputSubClass extends OutputSuperClass {
   OutputSubClass () {
     System.out.println("Sub 1");
   }
```

```
   OutputSubClass (int x) {
     //insert code here
     System.out.println("Sub 2");
   }
}
```

Which statement, when inserted in the code, will generate the output `Sub 1`?

A) `super();`

B) `this();`

C) `this.OutputSubClass();`

D) `super.OutputSubClass();`

Explanation

The statement `this();`, when inserted in the code, will generate the output `Sub 1`. This statement references the parameterless constructor, which generates the output `Sub 1`.

The statement `super();` and `super.OutputSubClass();`, when inserted in the code, will not generate the output `Sub 1`. These statements use the `super` keyword reference the superclass. The first code references the superclass constructor, while the second statement fails compilation because constructors are not accessible using dot notation.

The statement `this.OutputSuperClass();`, when inserted in the code, will not generate the output `Sub 1`. The `this` keyword in dot notation referenced the current object, but cannot access constructors in this way. This statement will fail compilation.

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 > Classes and Objects > Using the this Keyword

---

Question #5 of 50

Question ID: 1327815

Given:

```
public class Factorial {
   public static int get() {
```

```
    return 1;
  }
  public static int get(int n) {
    int fact = get();
    for (int i = n; i > 1; i--) {
      fact *= i;
    }
    return fact;
  }
  public static int get(int n, int k) {
    return get(n) / get(n-k);
  }
}
```

Which code line will generate the output `120`?

A) `System.out.print(Factorial.get());`

B) `System.out.print(Factorial.get(10,5));`

C) `System.out.print(Factorial.get(5));`

D) `System.out.print(Factorial.get(10));`

Explanation

The following code line will generate the output `120`:

`System.out.print(Factorial.get(5));`

This code line invokes the second overloaded `get` method with a single `int` parameter. The method returns the product of 5 * 4 * 3 * 2 * 1, which is 120.

The code line that invokes the `get` method with no arguments will not output `120`. The get method with no parameters is invoked and will output `1`.

The code line that invokes the `get` method with argument 10 will not output `120`. This code line invokes the second overloaded `get` method with a single `int` parameter. The method returns the factorial of 10, which is 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 3,628,800.

The code line that invokes the `get` method with two int arguments will not output `120`. This code line invokes the third overloaded `get` method with two `int` parameters. The method returns the division of 10 factorial (3,628,800) divided by 5 factorial (120), which is 30,240.

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 #6 of 50

Question ID: 1327831

Given:

```
class Job {
   private String name;
   private String[] reqs;
   public Job(String name, String... reqs) {
     this.name = name;
     this.reqs = reqs;
   }
   public void post() {/*Implementation omitted*/}
   //insert code here
}

class Programmer extends Job {
   private String language;
   public Programmer(String name, String... reqs) {super(name, reqs);}
   public void post(String language) {
     post();
     post(language.toCharArray());
   }
}
```

Which method, when inserted in the code, will compile?

A) `public void post(String rawData) {/*Implementation omitted*/}`

B) `void post(String rawData) {/*Implementation omitted*/}`

C) `private void post(char[] rawData) {/*Implementation omitted*/}`

D) **`protected void post(char[] rawData) {/*Implementation omitted*/}`**

Explanation

The following method, when inserted in the code, will compile:

```
protected void post(char[] rawData) {/*Implementation omitted*/}
```

Because this overloaded method is invoked in the subclass `Programmer` , it must be declared with an access modifier other than `private` . This method is declared with the access modifier `protected`, so that it is available only to subclasses, such as `Programmer`.

The method declared with the `private` keyword will not compile when inserted in the code. This is because the method is unavailable to the `Programmer` class. The access modifier `private` indicates that a class member is only available to that class.

The following methods, when inserted in the code, will not compile:

```
public void post(String rawData) {/*Implementation omitted*/}
void post(String rawData) {/*Implementation omitted*/}
```

Although both methods are available to the `Programmer` class, they do not match the signature required by the `Programmer` class. The statement `post(language.toCharArray());` requires a method that specifies a `char` array parameter.

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 #7 of 50

Question ID: 1327972

Given:

```
import java.io.*;

public class FileManipulator {
  public static void main (String[] args) {
    try (FileWriter fw = new FileWriter("email.txt", true)) {
      fw.write("\nPRIORITY: HIGH");
    } catch (Exception ex) {
      System.err.println(ex);
    }
  }
}
```

And the contents of the `email.txt` file:

```
TO: admin@company.com
FROM: customer@home.net
SUBJECT: LOGIN ISSUE
BODY: Forgot my password. Please reset!
```

What are the contents of `email.txt` after the program is executed?

A) `PRIORITY: HIGH`

B)
```
PRIORITY: HIGH
TO: admin@company.com
FROM: customer@home.net
SUBJECT: LOGIN ISSUE
BODY: Forgot my password. Please reset!
```

C)
```
PRIORITY: HIGHany.com
FROM: customer@home.net
SUBJECT: LOGIN ISSUE
BODY: Forgot my password. Please reset!
```

D)
```
TO: admin@company.com
FROM: customer@home.net
SUBJECT: LOGIN ISSUE
BODY: Forgot my password. Please reset!
PRIORITY: HIGH
```

Explanation

The contents of `email.txt` will be the following after the program is executed (modification in bold):

```
TO: admin@company.com
FROM: customer@home.net
SUBJECT: LOGIN ISSUE
BODY: Forgot my password. Please reset!
PRIORITY: HIGH
```

The overloaded constructor for `FileWriter` accepts a `boolean` indicating whether to append content to an existing file. The default behavior when instantiating `FileWriter` is to overwrite existing file content. The `FileWriter` class provides only the `write` method for output.

The contents of `email.txt` will not be `PRIORITY: HIGH` after the program is executed. This would be the content if `false` was specified or if the `boolean` value was omitted in the `FileWriter` constructor.

The file `email.txt` will not include `PRIORITY: HIGH` prepended to or overwriting the beginning of its contents. The `FileWriter` class does not provide random access to file content. It only provides appending and overwriting behaviors.

Objective:
Java File I/O

Sub-Objective:
Read and write console and file data using I/O Streams

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 > API Specification > java.io > Class FileWriter

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > Basic I/O > Reading, Writing, and Creating Files

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Numbers and Strings > Formatting Numeric Print Output

---

Question #8 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) Keep all classes and interfaces `public`

B) Mark the package as sealed in the manifest

C) Implement the `Cloneable` class for all classes in the package

D) **Declare all `public` classes `final`**

E) Keep unrelated code together

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 #9 of 50

Question ID: 1328161

Given the following code:

```
import java.util.*;
import java.util.concurrent.*;

public class ListThread extends Thread {
  private List<Integer> list_copy;
  public ListThread (List<Integer> list) {
    list_copy = list;
  }
  @Override
  public void run() {
    for(Integer i : list_copy) {
      try {
```

```
        String thName = Thread.currentThread().getName();
        list_copy.remove(i);
        System.out.println( list_copy + "(" + thName + ") ");
      } catch (Exception ex) {
        System.err.println(ex.getClass());
      }
    }
  }
  public static void main(String[] args) {
    List<Integer> list = new ArrayList<>();
    list.add(1);list.add(2);list.add(3);
    list.add(4);list.add(5);list.add(6);
    Thread th1 = new ListThread(list);
    Thread th2 = new ListThread(list);
    th1.start();th2.start();
  }
}
```

Which output fragment is the most likely included in the result?

A)
```
[2, 3, 4, 5, 6](Thread-0)
[3, 4, 5, 6](Thread-0)
[4, 5, 6](Thread-0)
[2, 3, 4, 5, 6](Thread-1)
[5, 6](Thread-1)
[5, 6](Thread-1)
[5, 6](Thread-1)
[6](Thread-1)
](Thread-1)
[5, 6](Thread-0)
](Thread-0)
```

B)
```
[2, 3, 4, 5, 6](Thread-0)
[3, 4, 5, 6](Thread-1)
[4, 5, 6](Thread-0)
[5, 6](Thread-1)
[6](Thread-0)
](Thread-1)
```

C)
```
[2, 3, 4, 5, 6](Thread-0)
[3, 4, 5, 6](Thread-0)
[4, 5, 6](Thread-0)
[5, 6](Thread-0)
[6](Thread-0)
[](Thread-0)
[2, 3, 4, 5, 6](Thread-1)
[](Thread-1)
```

D) `class java.lang.IndexOutOfBoundsException`

E) `class java.lang.ArrayIndexOutOfBoundsException`

F)
```
Exception in thread "Thread-1"
java.util.ConcurrentModificationException
Exception in thread "Thread-0"
java.util.ConcurrentModificationException
```

Explanation

The following output is most likely to be included in the result:

```
Exception in thread "Thread-1" java.util.ConcurrentModificationException
Exception in thread "Thread-0" java.util.ConcurrentModificationException
```

This exception is thrown by both threads because modification operations are not thread-safe in an `ArrayList` when using an iterator. To use an iterator that supports thread safety, you can either create a derived class with synchronization code or use the thread-safe variant of `ArrayList`: `CopyOnWriteArrayList`. The `CopyOnWriteArrayList` class copies its instance, so that changes are not committed until all modification actions occur. If the `CopyOnWriteArrayList` class is used in this code, then the `ConcurrentModificationException` will not be thrown and included in the output.

The output must include an exception because the `ArrayList` class does not support thread-safe operations using an iterator.

The output will not include an `IndexOutOfBoundsException` and `ArrayIndexOutOfBoundsException` because this code uses iterators, not manual index-based access.

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 #10 of 50

Question ID: 1327865

Given the following code:

```
public interface Card {}
public abstract class PlayingCard implements Card {}
public class PokerCard extends PlayingCard {}
public class FlashCard implements Card {}
public class NoteCard implements Card {}

public class Game {
   public static void main(String[] args) {
     //insert code here
     System.out.println((c instanceof Card) ? "Card!" : "Not Card?" );
     System.out.println((c instanceof PlayingCard) ? "PlayingCard!" : "Not PlayingCard?" );
     System.out.println((c instanceof FlashCard) ? "FlashCard!" : "Not FlashCard?" );
     System.out.println((c instanceof NoteCard) ? "NoteCard!" : "Not NoteCard?" );
   }
}
```

And given the following output:

```
Card!
PlayingCard!
Not FlashCard?
Not NoteCard?
```

Which statement, when inserted in the code, will generate the required output?

A) `NoteCard c = new PlayingCard();`

B) `PokerCard c = new PlayingCard();`

C) **`FlashCard c = new PokerCard();`**

D) `Card c = new PokerCard();`

Explanation

The statement `Card c = new PokerCard();`, when inserted in the code, will generate the required output. This is because `PokerCard` is a subclass of `PlayingCard` and implicitly implements the `Card` interface. Also, the object

type is `PokerCard`.

The statement `NoteCard c = new PlayingCard();`, when inserted in the code, will not generate the required output. This is because `PlayingCard` is an abstract class and cannot be instantiated. Also, `PlayingCard` is not a subclass of `NoteCard`, so the assignment operation is invalid. This statement will fail compilation.

The statement `FlashCard c = new PokerCard();`, when inserted in the code, will not generate the required output. This is because `PlayingCard` is not a subclass of `FlashCard`, so the assignment operation is invalid. This statement will fail compilation.

The statement `PokerCard c = new PlayingCard();`, when inserted in the code, will not generate the required output. This is because `PlayingCard` is an abstract class and cannot be instantiated. Also, the `instanceof` operation with fail with `NoteCard` and `FlashCard` because `PlayingCard` is incompatible with these types.

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 > Language Basics > Equality, Relational, and Conditional Operators

---

Question #11 of 50

Question ID: 1328010

Consider the following code:

```
public @interface Wizards {
 int age() default 300;
 String wizardName();
}

//Insert code here
public class Merlin {
 void magic() {
  System.out.println("abracadabra!");
 }
```

What code will you use to apply the annotation `Wizards` to `Merlin`?

A) `@Inherited Wizards (age=777, wizardName="Merlin")`

B) `@Wizards (age=777, wizardName="Merlin ")`

C) **`@interface Wizards (age=777, wizardName="Merlin")`**

D) `@Override Wizards (age=777, wizardName="Merlin")`

Explanation

You should use the following code:

```
@Wizards(age=777, wizardName="Merlin")
```

This allows you to set specific element values for the custom annotation `Wizards`.

The `@interface` option is incorrect. You use `@interface` when defining a custom annotation.

`@Inherited` is an incorrect option. You use the `@Inherited` annotation when you want subclasses to inherit custom annotations of a parent class.

`@Override` is an incorrect option. It 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.

You can have custom annotations for annotating elements of your program like methods, variables and constructors. These annotations are applied before the program element is declared. The syntax for user-defined annotations is as given below:

```
[Access-Specifier] @interface<NameofAnnotation>{
 Data-Type <Name of Method>() [default-value];
}
```

The `NameofAnnotation` specifies what your annotation is called. The `default-value` field is optional. The specified `Data-Type` of the method needs to be either enum, primitive, String, class or an array.

The other options referencing `@Deprecated`, `@Override`, and `@Inherited` annotations are syntactically incorrect and do not provide custom annotations. These are among Java's predefined annotation types.

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 #12 of 50

Question ID: 1328025

Given the following:

```
int x = 0;
```

Which code fragment increments x to 10?

A) `while (x < 10) { x++; }`

B) `while (x < 11 ? 1 : 0) { x++; }`

C) `while (x < 10 ? 1 : 0) { x++; }`

D) `while (x < 11) { x++; }`

Explanation

The code fragment `while (x < 10) { x++; }` increments x to 10. The expression in the `while` statement will be evaluated 11 times. In the first iteration, the value of x is 0. It is then incremented to 1 using the statement x++. In the final iteration where the `while` expression evaluates to true, x is 9, and the statement x++ increments x to 10.

The code fragment `while (x < 11) { x++; }` will not increment x to 10. The final value of x will be 11 because the expression in the `while` statement will evaluate to `true` when x is 10.

The code fragments `while (x < 10 ? 1 : 0) { x++; }` and `while (x < 11 ? 1 : 0) { x++; }` will not increment x to 10 because they will not compile. These expressions use the conditional operator (`?:`) to return an `int` value, which is not a compatible type for a `while` statement. To be a valid expression in a `while` statement, it must evaluate to a `boolean` value. The conditional operator (`?:`) uses a `boolean` expression but can return a data type other than `boolean` when the expression is `true` or `false`.

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 #13 of 50

Question ID: 1327772

```
public class JavaSETest {
  public static void main(String[] args) {
    int number = Integer.parseInt("Number 1");
```

```
    System.out.println(number);
  }
}
```

What is the output of this code?

A) `NumberFormatException`

B) `1`

C) `Number 1`

D) `Number`

Explanation

The code throws a `NumberFormatException`:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "Number 1"
  at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
  at java.lang.Integer.parseInt(Integer.java:580)
  at java.lang.Integer.parseInt(Integer.java:615)
  at javatest.app.JavaSETest.main(JavaSETest.java:14)
```

This is because the value passed to the `parseInt()` method for the `Integer` wrapper class is not correct for its type.

The other options are incorrect because this code generates an error and does not output anything.

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 > Numbers and Strings

---

Question #14 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) Non-deterministic output

B) `Odd sum = 55 even sum = 55`

C) Compilation fails at line `n1`

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

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

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 #15 of 50

Question ID: 1327970

Given:

```
import java.io.*;

class CharacterName implements Serializable {
  String given, sur;
}
class GameCharacter {
  CharacterName name = new CharacterName();
  int level, experience;
}
class PlayerCharacter extends GameCharacter implements Serializable {
  Date created = new Date();
  transient String player;
  static int numPlayers = 1;
}

public class ObjectSerializer {
  public static void main(String[] args) {
    PlayerCharacter pc = new PlayerCharacter();
    PlayerCharacter.numPlayers = 2;
    pc.name.given="Tristan"; pc.name.sur="Bolt";
    pc.level = 1; pc.experience = 1000;
    pc.player="Joshua";
    try(ObjectOutputStream strObj = new ObjectOutputStream(
     new FileOutputStream("object.txt")) ) {
      strObj.writeObject(pc);
    }catch (Exception ex) {
      System.err.print(ex);
    }
```

```
  }
}
```

Which field(s) of pc are stored in `object.txt` after the program executes?

A) `name`

B) `experience`

C) `created`

D) `player`

E) `level`

F) `numPlayers`

Explanation

Only the created field of `pc` is stored in `object.txt` after the program executes. A class that can be written to and read from a stream uses the marker interface `Serializable`. By default, all instance, non-transient fields are serializable, regardless of the access modifier. If a field is declared with the `transient` keyword, then this field will be omitted during serialization. Any inherited fields are serializable only if the superclass is also declared with the interface `Serializable`. Because `GameCharacter` does not implement `Serializable`, only fields declared in `PlayerCharacter` are serializable. The `created` field is the only one that does not include the keyword `static` or `transient`.

The `name` field is not stored in `object.txt` after the program executes. Although the class `CharacterName` is serializable, the `GameCharacter` class in which it is declared is not serializable. If `name` were declared in `PlayerCharacter`, then its fields `given` and `sur` would be serialized as an object field in `PlayerCharacter`.

The `level` and `experience` fields are not stored in `object.txt` after the program executes. Although `PlayerCharacter` inherits these fields from `GameCharacter`, `GameCharacter` does not implement `Serializable`. Thus, the fields in `GameCharacter` are not serializable.

The `player` field is not stored in `object.txt` after the program executes because player is declared with the `transient` keyword. By default, transient fields are not serializable.

The `numPlayers` field is not stored in `object.txt` after the program executes because `numPlayers` is declared with the `static` keyword. By default, static fields are not serializable. Static fields are associated with the class, not its instances. As long as the class is loaded, its fields are stored in memory.

Objective:
Java File I/O

Sub-Objective:
Read and write console and file data using I/O Streams

References:

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

Oracle Technology Network > Java SE > Java Language Specification > System Architecture > Chapter 1 > Defining Serializable Fields for a Class

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > Basic I/O > Object Streams

---

Question #16 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 must be associative.

C) **The behavior can safely modify the `Student` objects.**

D) **The expression must implement `BinaryOperator<Student>`.**

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 #17 of 50

Question ID: 1327840

Given:

```
public class Pedometer {
   private double stride;
   private double[] measurements;
}
```

Which code fragment is a method that meets good encapsulation principles?

A)
```
public void getStride(double stride) {
   stride = this.stride;
}
```

B)
```
public double[] getMeasurements() {
   return measurements.clone();
}
```

C)
```
public double[] getMeasurements() {
   return this.measurements;
}
```

D)
```
public void getStride(double stride) {
   this.stride = stride;
}
```

Explanation

The following code fragment is a method that meets good encapsulation principles:

```
public double[] getMeasurements() {
   return measurements.clone();
}
```

In this code fragment, the accessor method uses the `clone` method to return a copy of the `measurements` field, rather than a reference to the field itself. Accessor methods should return copies of field objects. A copy of a field will prevent other classes from modifying fields directly and will require going through mutator methods.

The two versions of the accessor method `getStride` do not meet good encapsulation principles. Neither method returns the value of the `stride` field as expected, but instead modifies the `stride` field like a mutator method. Also,

the statement `stride = this.stride;` overwrites the `stride` parameter, rather than assigning the `stride` parameter to the `stride` field as expected.

The accessor method `getMeasurements` that returns a direct reference to the `measurements` field object does not meet good encapsulation principles. 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.

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 #18 of 50

Question ID: 1327951

Which one of the following method signatures is required to sort a collection of `String` objects without using natural order?

A) `public boolean compare(String s1, String s2)`

B) `public boolean compareTo(String s)`

C) `public int compare(String s1, String s2)`

D) `public int compareTo(String s)`

Explanation

The method signature `public int compare(String s1, String s2)` is required to sort a collection of `String` objects without using natural order. The `compare` method returns 0 if the arguments are equal, a negative integer if the first argument is less than the second argument, and a positive integer if the first argument is greater than the second argument.

This method is provided by the `Comparator` interface. A `Comparator` implementation is required when sorting elements in an order other than the default natural order.

The method signatures that return a `boolean` value are not valid signatures for either the `compareTo` or `compare` methods. These methods return an `int` value, indicating the relative positioning of objects to each other.

The `compareTo` method is not required to sort a collection without using natural order. The `compareTo` method is provided by the `Comparable` interface for elements to be sorted in their natural order. The `Comparable` interface is implemented by elements that can be sorted within a collection. The `compareTo` method returns negative, zero or a positive integer if the object being compared to the one that calls the method is less than equal or greater.

Objective:
Working with Arrays and Collections

Sub-Objective:
Sort collections and arrays using Comparator and Comparable interfaces

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Collections > Interfaces > Object Ordering

---

Question #19 of 50

Question ID: 1328101

Your company is doing an inventory check for which you need to find out the number of laptops used in the Atlanta office. To do so, you access a collection of `Laptop` objects called `lappys` using the following code.

```
long laptopsinAtlanta = lappys.stream()
  // INSERT CODE
  .count();
```

Which of the following pieces of code will you insert into the code sample above to make it find the slowest speed laptop from the stream?

A) `.min(Comparator.comparing(lappy -> lappy.getSpeed()))`

B) `.reduce((lappy, speed) -> lappy + "" + speed)`

C) `.max(Comparator.comparing(lappy -> lappy.getSpeed()))`

D) `.count( lappy -> lappy.getSpeed())`

Explanation

You should use the following code fragment:

```
.min(Comparator.comparing(lappy -> lappy.getSpeed()))
```

The following code fragment is incorrect because it would find the *fastest* laptop in the office, not the slowest:

```
.max(Comparator.comparing(lappy -> lappy.getSpeed()))
```

The following code fragment is incorrect because the `reduce` method creates one element from many elements. It will not find a minimum value as required in this scenario:

```
.reduce((lappy, speed) -> lappy + "" + speed)
```

The following code fragment is incorrect because the `count` method is terminal. It will not find a minimum value as required in this scenario:

```
.count( lappy -> lappy.getSpeed())
```

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

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > Basic I/O > Object Streams Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > Basic I/O > Object Streams

---

Question #20 of 50

Question ID: 1328157

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

A) `TreeMap`

B) **`CyclicBarrier`**

C) **`ConcurrentHashMap`**

D) `HashMap`

E) `CopyOnWriteArrayList`

F) `ArrayList`

G) **`BlockingQueue`**

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 #21 of 50

Question ID: 1327886

Given:

```
Writer writer = new Programmer();
```

Assuming that `Programmer` is a subclass of `Writer`, which statement executes a method found only in the `Programmer` class?

A) `writer.learnLanguage("Java");`

B) `writer.write();`

C) **`((Programmer) writer).learnLanguage("Java");`**

D) `((Writer) writer).write();`

Explanation

The following statement executes a method found only in the `Programmer` class:

```
((Programmer) writer).learnLanguage("Java");
```

When instantiating a subclass and assigning it to a supertype variable, only supertype members are available using the variable. To overcome this limitation, you can use subtype casting as demonstrated by this statement.

The statements `writer.write();` and `((Writer) writer).write();` will not execute a method found only in the `Programmer` class. The first statement executes a method presumably found in the `Writer` class, while the second statement performed an unneeded cast, because the reference type is already `Writer`.

The statement `writer.learnLanguage("Java");` will not execute a method found only in the `Programmer` class. Because the `learnLanguage` method is not in the `Writer` class, this statement will fail compilation.

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 > Interfaces and Inheritance

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

---

Question #22 of 50

Question ID: 1327843

Given:

```
public class VarScope {
   int var;
   public static void main (String[] args) {
     int var = 10;
     VarScope scope = new VarScope();
     scope.var = var + 2;
     scope.adjustVar(scope.var + 2);
     System.out.println("var = " + var);
   }
   private void adjustVar(int var) {
     var += 2;
   }
}
```

What is the result?

A) `var = 16`

B) `var = 10`

C) `var = 14`

D) `var = 12`

Explanation

The result will be the output `var = 10`. The output is based on the local variable named `var` in the `main` method. The variable `var` in this scope is set to `10` and not modified until it is printed.

The result will not be the output `var = 12` because the local variable in the `main` method will be printed. This result would be the output if the instance variable `var` were printed, because the class variable is set to the local variable incremented by two.

The result will not be the output `var = 14` because the local variable in the `main` method will be printed. This result would be the output if the local variable `var` in the `adjustVar` method were printed before it is incremented by two.

The result will not be the output `var = 16` because the local variable in the `main` method will be printed. This result would be the output if the local variable `var` in the `adjustVar` method were printed after it is incremented by two.

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 > Variables

---

Question #23 of 50

Question ID: 1328016

Given the following:

```
if ( x < 10) {
   if (x > 0)
     System.out.print("She");
   else
     System.out.print("Sally");
   if (x < 5)
     System.out.print(" sells seashells");
   if ( x > 10)
     System.out.print(" will sell all her seashore shells");
```

```
   else if (x < 15)
     System.out.print(" by the");
   else if (x < 20)
     System.out.print(" on the");
   if ( x < 10)
     System.out.print(" seashore");
   else
     System.out.print(" seashell shore");
} else {
   System.out.print("Of that I'm sure");
}
```

Which value for the variable x will output `Sally sells seashells by the seashore`?

A) 5

B) **0**

C) 10

D) 1

Explanation

The value 0 for the x variable will output `Sally sells seashells by the seashore`. To meet the criteria of the first `if` statement, x must be less than `10`. To output `Sally`, x must be less than or equal to `0` to reach the second `else` statement. To output `sells seashells`, x must be less than `5` to meet the criteria of the third `if` statement. To output `by the`, x must be less than 15 but not greater than 10 to reach and meet the criteria of the first `else if` statement. Finally, x must be less than `10` to meet the criteria of the fifth `if` statement and output `seashore`.

The values 1 and 5 for x will not output `Sally`, but will output `She`. If x is set to 1, then the output will be `She sells seashells by the seashore`. If x is set to 5, then the output will be `She by the seashore`.

The value 10 for x will output `Of that I'm sure`, not `Sally`. This is because the criteria of the first `if` statement is not met.

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 if-then Statement

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Language Basics > Equality, Relational and Conditional Operators

Question #24 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) Add a `clone()` method after Line 11

B) Remove `implements Serializable` from Line 1

C) Add `File.list` 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 #25 of 50

Question ID: 1328030

Given the following code fragment:

```
int i = 6;
do {
   System.out.print(--i + " ");
} while (i > 0);
System.out.print("...BLAST OFF!");
```

What is the output?

A) `4 3 2 1 0 ...BLAST OFF!`

B) `6 5 4 3 2 1 ...BLAST OFF!`

C) **`5 4 3 2 1 0 ...BLAST OFF!`**

D) `5 4 3 2 1 ...BLAST OFF!`

E) `...BLAST OFF!`

Explanation

The output is `5 4 3 2 1 0 ...BLAST OFF!` In the `do-while` block, the variable `i` is decremented by 1 in each iteration, until `i` is equal to or less than 0. The value of `i` is initialized to 6, but the decrement operation occurs before the value is printed. Thus, the first iteration prints `5` until the value `0` is reached, exits the `do-while` block, and prints `...BLAST OFF!`

The output is not `...BLAST OFF!` because a `do-while` loop executes at least once. The expression is evaluated at the bottom and executes repeatedly if the expression evaluates to `true`.

The output is not the one that starts with the value `4` because the starting value for `i` is 6, not 5.

The output is not the one that starts with the value `6` or ends with the value `1` because the decrement operation in the `do-while` block precedes the variable. Thus, the decrement operation occurs before the value is printed.

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 #26 of 50

Question ID: 1327765

Given the following:

```
int i = 10;
```

Which two expressions evaluate to 3?

A) `i + (5 - 6) * 10 / 5`

B) `((i + 5 - 6) * 10) / 5`

C) `(i + 5) - 6 * 10 / 5`

D) `(i + 5 - 6) * 10 / 5`

E) `i + (5 - 6 * 10) / 5`

F) `(i + 5) - 6 * (10 / 5)`

Explanation

The following two expressions evaluate to 3:

```
(i + 5) - 6 * 10 / 5
```

```
(i + 5) - 6 * (10 / 5)
```

In the first expression, `(i + 5) - 6 * 10 / 5` first evaluates `i + 5` as 15 because these operators are inside the parentheses. The next evaluation is `6 * 10` as 60 because multiplicative operators precede additive operators. The next evaluation is `60 / 5` as 12 because multiplicative operators precede additive operators, and this operator is next in the sequence from left to right. The final evaluation is `15 - 12` as 3 because it involves the additive operator.

In the second expression, `(i + 5) - 6 * (10 / 5)` first evaluates `i + 5 as 15` and then evaluates `10 / 5` as 2 because these operators are inside two separate sets of parentheses. The next evaluation is `6 * 2` as 12 because multiplicative operators precede additive operators. The final evaluation is `15 - 12` as 3 because it involves the additive operator.

Knowing operator precedence can help you identify which parts of an expression are evaluated first and which parts will follow. Here is an operator precedence list from highest precedence to lowest precedence:

1. Postfix unary: `num++`, `num--` (value change only occurs *after* overall expression is evaluated)

2. Prefix unary: ++num, --num, +num, -num, ~ !
3. Multiply, Divide, Modulus: * / %
4. Add, Subtract: + -
5. Shift: << >> >>>
6. Relational: < > <= >= instanceof
7. Equality: == !=
8. Bitwise AND: &
9. Bitwise exclusive OR: ^
10. Bitwise inclusive OR: |
11. Logical AND: &&
12. Logical OR: ||
13. Ternary: ? :
14. Assignment: = += -= *= /= %= &= ^= |= <<= >>= >>>=

Unary operators (++, --) operate on a variable in the order in which they are placed.

The expression `i + (5 - 6) * 10 / 5` does not evaluate to 3. This expression evaluates to 8. First, `5 - 6` is evaluated as -1 because the operator is inside parentheses. The next evaluation is `-1 * 10` as -10 and then `-10 / 5` as -2 because operators with the same precedence level are evaluated from left to right. The final evaluation is `i + -2`, or `10 - 2` as 8.

The expressions `(i + 5 - 6) * 10 / 5` and `((i + 5 - 6) * 10) / 5` do not evaluate to 3. Both expressions evaluate to 18. In both expressions, `i + 5 - 6` is evaluated as 9. With or without the extra parentheses, the next evaluation is `9 * 10` as 90 because operators with the same precedence level are evaluated left to right. The final evaluation is `90 / 5` as 18.

The expression `i + (5 - 6 * 10) / 5` does not evaluate to 3. This expression evaluates to -1. In the parentheses, `6 * 10` is evaluated as 60 first and then `5 - 60` as -55 because multiplicative operators are evaluated before additive operators. For the same reason, `-55 / 5` is evaluated as -11 and then `i + -11` is evaluated as -1.

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 > Assignment, Arithmetic, and Unary Operators

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

Question #27 of 50

Question ID: 1328123

Consider the following code for a user database:

```
public class User {
  Integer userID;
  String firstName;
  String lastName;
  LocalDate dateofHire;
}

public class CreateUsersList {
  List<User> users;
  public void setup() {
    users = new ArrayList<>();
    users.add(new User(001, "Sean", "Benjamin", LocalDate.of(1990, Month.MAY, 20)));
    users.add(new User(002, "Sally", "Donner", LocalDate.of(2010, Month.JANUARY, 10)));
    users.add(new User(003, "Richard", "Anderson", LocalDate.of(2004, Month.JULY, 10)));
    users.add(new User(004, "Jessica", "Winters", LocalDate.of(2006, Month.JULY, 20)));
    users.add(new User(005, "Jonathan", "Steele", LocalDate.of(1990, Month.MAY, 20)));
    users.add(new User(006, "Cindy", "Summer", LocalDate.of(2008, Month.MAY, 15)));
    //Insert code here
  }
}
```

Which code statements that will enable you to sort this list by employee number using Java Stream API? (Choose all that apply.)

A)
```
Comparator<User> userIDSort = (u1, u2) -> Integer.compare(
  u1.returnUserID(), u2.returnUserID());
```

B)
```
users.stream().sorted(userIDSort)
  .forEach(s -> System.out.println(s));
```

C)
```
users.stream().map(userIDSort)
  .forEach(s -> System.out.println(s));
```

D)
```
List<User> userIDSort = (u1, u2) -> Integer.compare(
  u1.returnUserID(), u2.returnUserID());
```

Explanation

The correct options are:

```
Comparator<User> userIDSort = (u1, u2) -> Integer.compare(
  u1.returnUserID(), u2.returnUserID());
```

and

```
users.stream().sorted(userIDSort)
  .forEach(s -> System.out.println(s));
```

The `sorted` method returns a stream made of the elements of the stream on which this method was run, but sorted in a natural order.

The following option is incorrect because it implements `List`:

```
List<User> userIDSort = (u1, u2) ->  Integer.compare(  u1.returnUserID(), u2.returnUserID());
```

You need to create a `Comparator` interface that implements the comparison needed to perform the sorting.

The following option is incorrect because it uses the `map` method:

```
users.stream().map(userIDSort)
  .forEach(s -> System.out.println(s));
```

The map method does not organize the order of elements, but translates or modifies them.

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

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > Basic I/O > Object Streams

---

Question #28 of 50

Question ID: 1327896

Which type defines a valid interface?

A)
```
public interface HourlyWorker {
    public static final double minimum_wage = 7.25;
    public abstract HourlyWorker();
    public abstract void performWork(double hours);
}
```

B)
```
public interface HourlyWorker {
    public static double minimum_wage;
    public void performWork(double hours);
}
```

C)
```
public interface HourlyWorker {
    public static final double minimum_wage = 7.25;
    public void performWork(double hours) {
      //implementation
    }
}
```

D)
```
public interface HourlyWorker {
    double minimum_wage = 7.25;
    void performWork(double hours);
    default double getNormalWeeklyHours() {
      return 40;
    }
}
```

Explanation

The following type defines a valid interface:

```
public interface HourlyWorker {
   double minimum_wage = 7.25;
   void performWork(double hours);
   default double getNormalWeeklyHours() {
     return 40;
   }
}
```

An interface may contain only abstract, default, or static methods and class constants. When declared in an interface, all methods are implicitly public and abstract and all fields are `public`, `static`, and `final`. You can also use the `default` keyword to declare default methods to provide default implementation or the `static` keyword to declare a class-wide method. This code declares the constant `minimum_wage`, the abstract method `performWork`, and the default method `getNormalWeeklyHours`.

The type that does not set the `minimum_wage` constant is not a valid interface. Values for constants are not modifiable, so they must be assigned a value in their declaration.

The type that declares an abstract constructor is not a valid interface. Constructors can neither be abstract nor declared in an interface.

The type that contains implementation for the `performWork` method is not a valid interface. An interface can contain only abstract methods without implementation or default methods with implementation.

Objective:

Java Object-Oriented Approach

Sub-Objective:

Create and use interfaces, identify functional interfaces, and utilize private, static, and default methods

References:

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

---

Question #29 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) **Make the class implement the `AutoCloseable` interface.**

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

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

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

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 #30 of 50

Question ID: 1328068

Given this method:

```
public static <T,U> void show(Function<T,U> f, T in) {
   System.out.println("Result is " + f.apply(in));
}
```

Which code statement would generate the output `Result is 5`?

A) `show(f.apply(), "hello");`

B) `String s = "hello"; show((s)->s.length());`

C) `show(s->s.length(), "hello");`

D) `String s = "hello"; show(()->s.length(), s);`

Explanation

The code statement `show(s->s.length(), "hello");` will generate the output `Result is 5`. The lambda expression implementing the `Function` interface effectively creates an object that implements `Function`, and contains the behavior specified in the `apply` method declared by the `Function` interface. To invoke that behavior, the `apply` method, which is a member of the lambda expression, must be invoked. To achieve that in this scenario, the `show` method must be called with the lambda expression as the first argument, and a `String` as the second argument.

`The show(s->s.length(), "hello");` lambda is created correctly. When its `apply` method is invoked inside the `show` method, it will calculate the value 5, which is the length of the `String` provided as the second argument to the `show` method.

`String s = "hello"; show((s)->s.length());` is incorrect because of two issues. First, the local variable `s` will prevent compilation of the lambda `(s)->s.length()`, because the formal parameter of a lambda expression may not use a variable name that is already in scope. Second, the invocation of `show` will fail, because only one argument is provided, and two are required.

`String s = "hello"; show(()->s.length(), s);` has no parameters, so it does not implement the `Function` interface.

`show(f.apply(), "hello");` seems to invoke the behavior of a function. However, no object `f`, implementing `Function`, has been defined at this point. Further, the first argument to `show` must be the object implementing `Function`, not the result of calling the function.

Objective:
Working with Streams and Lambda expressions

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

References:

Oracle Documentation > Java SE 11 API > Interface Function<T,R>

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

Question #31 of 50

Question ID: 1327782

Consider the following code:

```
public class talker {
  public static void loopText(String mesg, int num, int thrd) {
    Runnable r = () -> {
      while (num > 0) {
        num--;
        System.out.println(mesg);
      }
    };
    for (int i = 0; i < thrd; i++) new Thread(r).start();

  }
  public static void main(String[] args){
    talker t;
    talker.loopText(“ich bin froh”, 4, 1);
  }
}
```

What will be the output of the given code fragment?

A) `ich bin froh`

B)
```
ich bin froh
ich bin froh
ich bin froh
ich bin froh
```

C) The code fails to compile.

D) `mesg`

Explanation

The code fails to compile because it is illegal to attempt to mutate a captured variable from a lambda expression. This is occurring in the line that says `num--`. `num` is a captured variable and as per lambda expression rules, it needs to be effectively final, which it is not as it keeps changing through various iterations of the code resulting in a compilation error.

The other options are incorrect because no output is displayed as the code does not compile successfully. If the line `num--`; were omitted from the code, then an output of `ich bin froh` would be displayed.

A lambda expression's body contains a scope which is the same as a regular nested block of code. Additionally, lambda expressions can access local variables from a scope which are functionally final. This means that these variables must either declared as `final` or are not modified.

Objective:
Working with Java Data Types

Sub-Objective:
Use local variable type inference, including as lambda parameters

References:

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

Lambda Expressions and Variable Scope

Question #32 of 50

Question ID: 1328048

Which exception class indicates that a character index is either negative or not less than the length of a string?

A) `BadIndexBoundsException`

B) `CharIndexOutOfBoundsException`

C) `StringIndexOutOfBoundsException`

D) `BadStringOperationException`

Explanation

The exception class that indicates a character index is either negative or not less than the length of a string is `StringIndexOutOfBoundsException`. The `charAt` method can throw this unchecked exception.

The exception class `BadStringOperationException` does not indicate that a character index is either negative or not less than the length of a string. This exception class indicates that an unexpected operation is provided when constructing a query.

The exception classes `BadIndexBoundsException` and `CharIndexOutOfBoundsException` are not valid exception classes provided by the Java SE 11 API.

Objective:
Exception Handling

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

References:

Oracle Documentation > Java SE 11 API > Class StringIndexOutOfBoundsException

---

Question #33 of 50

Question ID: 1327978

Consider the following class:

```
public class Employee {
  private static final long SerialVersionUID = 1234L;
  private String emp_name;
  private int emp_age;
  private String emp_gender;
  Employee() {};
  Employee(String emp-name, int emp-age, String emp-gender) {
    this.emp_name = emp-name;
    this.emp_age = emp-age;
    this.emp_gender = emp-gender;
  }
  @Override
  public String toString() {
    return "Employee Name:" + emp_name + "\nAge: " + emp_age + "\nGender: " + emp_gender;
  }
}
```

Which interface needs to be implemented by this class to successfully convert its objects to a byte stream?

A) `Comparable`

B) `Serializable`

C) `Iterable`

D) `Closeable`

Explanation

You should implement the `Serializable` interface. The class requires the following code in its definition:

```
public class Employee implements Serializable
```

Serialization is a process of converting objects to a stream of bytes that can be persisted. Objects converted to a byte stream can then be written to a file. The Java library provides the 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. The `Serializable` interface is a marker interface whose purpose is simply to *mark* the objects of classes that implement it so that these objects have certain capabilities.

Each serializable class requires a `SerialVersionUID`, which is a version number needed by the Serialization runtime environment for verifying the compatibility of the sender and receiver of a serialized object during the process of deserialization.

The `Iterable` interface allows objects implementing it to be targets for `foreach` statements. This interface does not help implement serialization.

The `Comparable` The interface, when implemented, allows objects to be compared with other objects. This interface does not help implement serialization.

The `Closeable` interface allows users to close the source or destination of data. This interface does not help implement serialization.

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 > Java SE > Java SE Documentation > Java Platform Standard Edition 11 > API Specification > java.base > Package: java.io

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

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

---

Question #34 of 50

Question ID: 1327919

Given:

```
public enum Architecture {
  ARM, x86, x86_64, RISC, MIPS, SPARC, UNIVAC;
  public static void main(String[] args) {
    for(Architecture a: Architecture.toArray()) {
      System.out.print(a.ordinal() + " ");
    }
  }
}
```

What is the result?

A) `ARM x86 x86_64 RISC MIPS SPARC UNIVAC`

B) `0 1 2 3 4 5 6`

C) An exception is thrown at runtime.

D) Compilation fails.

E) `1 2 3 4 5 6 7`

Explanation

Compilation fails. The `Enum` class does not provide the method `toArray`, which is available in the `Collection` class. The `Enum` class instead provides the values method that can be used to iterate through enumeration constants.

The result is not the output `0 1 2 3 4 5 6` because compilation fails. This would be the output if the `toArray` method were replaced with the `values` method. The ordinal method returns the zero-based index of an enumeration constant.

The result is not the output `1 2 3 4 5 6 7` because compilation fails. Also, the ordinal method returns a zero-based index.

The result is not the output `ARM x86 x86_64 RISC MIPS SPARC UNIVAC` because compilation fails. Also, this is not the result because the ordinal method is invoked. This would be the output if the `name` or `toString` methods were invoked.

The result is not an exception at runtime because the code fails to compile.

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

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

---

Question #35 of 50

Question ID: 1327777

Which statement is true about string equality?

A) The `equals` method compares object references.

B) The `compareTo` method returns `0` for equality.

C) The `compareTo` method returns `1` for equality.

D) The == operator compares character sequences.

Explanation

The `compareTo` method returns `0` for equality. The `compareTo` method returns a positive integer if the specified `String` object comes before the `String` instance, and returns a negative integer if the specified `String` object comes after the `String` instance. The `compareTo` method evaluates each character by its underlying Unicode value in sequence.

The `equals` method does not compare object references. The `equals` method compares character sequences.

The == operator does not compare character sequences. The == operator compares object references.

The `compareTo` method does not return `1` for equality. The `compareTo` method returns `0` for equality and returns a positive integer if the specified `String` object comes before the `String` instance.

Objective:
Working with Java Data Types

Sub-Objective:
Handle text using String and StringBuilder classes

References:

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

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Numbers and Strings > Comparing Strings and Portions of Strings

---

Question #36 of 50

Question ID: 1327975

Which method of the `BufferedReader` class ignores and discards a specified number of characters?

A) `skip`

B) `readLine`

C) `read`

D) `mark`

E) `reset`

Explanation

The `skip` method of the `BufferedReader` class ignores and discards a specified number of characters. Assume a text file named `simple.txt` with the following content:

```
Java Standard Edition
```

The following code would ignore the first five characters and generate the output `Standard Edition`:

```
import java.io.*;
public class BReader1 {
  public static void main(String[] args) {
    try (BufferedReader br = new BufferedReader(
      new FileReader("simple.txt"))) {
      br.skip(5);
      System.out.println(br.readLine());
    } catch(IOException ex) {
      System.err.print(ex);
    }
  }
}
```

The `read` and `readLine` methods of the `BufferedReader` class do not ignore and discard a specified number of characters. The `read` method can retrieve a single character or range of characters into an array, while the `readLine` method retrieves all characters from the current position to the end of the line as a `String`.

The `mark` and `reset` methods of the `BufferedReader` class do not ignore and discard a specified number of characters. The `mark` method saves the current position within the stream for a certain amount of characters, while the `reset` moves back to that saved position. Using the same text file `simple.txt`, the following code would generate the output `SE Java`:

```
import java.io.*;
public class BReader2 {
  public static void main(String[] args) {
    try (BufferedReader br = new BufferedReader(
      new FileReader("simple.txt"))) {
      br.mark(23);
      br.skip(5);
      System.out.print((char)br.read());
      br.skip(8);
      System.out.print((char)br.read() + " ");
      br.reset();
      char[] rdchars = new char[4];
      br.read(rdchars,0,4);
      System.out.print(rdchars);
    } catch(IOException ex) {
```

```
      System.err.print(ex);
    }
  }
}
```

Objective:
Java File I/O

Sub-Objective:
Read and write console and file data using I/O Streams

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.io > Class BufferedReader

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > Basic I/O > Reading, Writing, and Creating Files

---

Question #37 of 50

Question ID: 1328177

Consider the following code:

```
01  class exceptional {
02    public static void main(String[] args) throws FileNotFoundException {
03      FileInputStream input = new FileInputStream(System.getenv("APPDATA") + args[0]);
04    }
05  }
```

Which line of the code contains a possible security vulnerability?

A) Line 01

B) Line 03

C) Line 02

D) There are no obvious vulnerabilities in this code.

Explanation

Line 02 has a security risk. Throwing a `FileNotFoundException` can disclose sensitive file structure information to an attacker.

One way to make code more security-compliant is to issue error messages that do not expose any underlying file or directory information. An example would be an I/O error message like “File not valid”.

Also, you can use the `File.getCanonicalFile()` method that first canonicalizes the file name so that filepath name comparisons can be made in a more simplified way. This is illustrated in the following code:

```
File myfile = null;
try {
  myfile = new File(System.getenv("APPDATA") +
  args[0]).getCanonicalFile();
  if (!file.getPath().startsWith("e:\\mypath")) {
    System.out.println("File not valid");
    return;
  }
} catch (IOException excep) {
  System.out.println("File not valid");
}
```

The other lines of code do not contain a security risk.

To protect confidential information from illegal access, you need to follow these guidelines:

- Remove sensitive data from exceptions.
- Never log sensitive data.
- Remove sensitive data from memory after use.

Exceptions like the `FileNotFoundException` can contain data like filenames or pathnames that can be used by attackers to infiltrate a system. This can happen if the `java.io.FileInputStream` constructor is called and it tries to read a system file that may not exist. The thrown exception can expose the underlying files or directory structure of the system, which can then be a target for further attacks.

Exceptions thrown may also change their outputs in future when underlying libraries that they access may change with more information. This means that a safe exception could in future expose system details that could cause a vulnerability.

Certain sensitive data, like Social Security Numbers (SSNs), must never be kept in memory longer than necessary or logged. If an application uses a character array to store SSNs, those arrays need to be purged immediately after use. Similarly, parsing libraries might be logging the data they parse which could include SSNs. For this reason, programmers need to be careful about what data is being parsed by the chosen libraries and ensure it isn't confidential information.

After any processing of sensitive data, memory containing it must be zeroed right away so that the confidential data is not the target of debugging, confidentiality or debugging attacks. However, this may not always be possible given that certain libraries may keep copies of this data in other parts of memory. Also, adding these security measures to code can reduce the quality of the code.

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

Examples of Non-Secure Code

Question #38 of 50

Question ID: 1328021

Which statement is true about using a `String` object in a `switch` statement?

A) `String` comparisons in case labels are case-insensitive.

B) `String` comparisons in case labels are case-sensitive.

C) Execution falls through if `break` statements are specified in case labels.

D) Execution terminates if `break` statements are not specified in case labels.

Explanation

When using a `String` object in a `switch` statement, `String` comparisons in case labels are case-sensitive. The comparison in each case label represents an invocation of the `String.equals` method, which compares each case-sensitive character in the string literals. To work around case sensitivity, you could invoke the `toLowerCase()` or `toUpperCase()` methods on the `String` object and specify a lower-case or upper-case expression for each case label.

`String` comparisons are not case-insensitive. The comparison in each case label represents an invocation of the `String.equals` method.

Execution will not fall through if `break` statements are specified in case labels. `break` statements terminate execution when specified in a case label.

Execution will not terminate if `break` statements are not specified in case labels. Without `break` statements, execution will fall through case labels.

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 switch statement

---

Question #39 of 50

Question ID: 1328037

Which is true about branching statements in an iterative block?

A) A labeled `break` statement terminates only the innermost block.

B) An unlabeled `break` statement terminates the outermost block.

C) A `continue` statement terminates only `for` and `switch` blocks.

D) A `return` statement that does not return a value terminates all blocks in the current method.

Explanation

A `return` statement that does not return a value will exit all blocks in the current method. A `return` statement, whether it returns a value or not, exits the current method. By exiting the current method, all iterative blocks are effectively terminated.

An unlabeled `break` statement does not terminate the outermost block. Because no label is specified, the `break` statement will terminate the innermost iterative block.

A labeled `break` statement does not terminate only the innermost block. Because any labeled block can be specified in a `break` statement, an outer block can be terminated.

A `continue` statement does not terminate only `for` and `switch` blocks. A `continue` statement skips the code in the current iteration and continues to the next iteration of the block. A labeled continue statement skips iteration in an outer loop.

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 > Branching Statements

---

Question #40 of 50

Question ID: 1327987

Given the code fragment:

```
Path sPath = Paths.get("C:\\Documents\\JavaProjects\\Java11\\Upgrade.txt");
Path dPath = Paths.get("C:\\Documents\\JavaProjects\\Java11\\NIO\\src");
try {
  Files.move(sPath, dPath, StandardCopyOption.ATOMIC_MOVE);
} catch (IOException ex) {
  System.err.println("Error Happened!");
}
```

Assuming the file `Upgrade.txt` exists in both the source and destination path, what is the result of compiling and executing this code?

A) The `Upgrade.txt` file is removed from the `Java11` directory and placed in the `NIO\src` sub-directory in a single operation. The existing `Upgrade.txt` file is appended.

B) The `Upgrade.txt` file is removed from the `Java11` directory, placed in the `NIO` sub-directory, and renamed `src` in three distinct operations.

C) The `Upgrade.txt` file is removed from the `Java11` directory and placed in the `NIO\src` sub-directory in two distinct operations. The existing `Upgrade.txt` file is overwritten.

D) The `Upgrade.txt` file is removed from the `Java11` directory, placed in the `NIO` sub-directory, and renamed `src` in a single operation.

E) The `Upgrade.txt` file is copied from the `Java11` directory, placed in the `NIO` sub-directory, and renamed `src` in a single operation.

F) The `Upgrade.txt` file is copied from the `Java11` directory, placed in the `NIO` sub-directory, and renamed `src` in three distinct operations.

Explanation

The result of compiling and executing this code is the `Upgrade.txt` file is removed from the `Java11` directory, placed in the `NIO` sub-directory, and renamed `src` in a single operation. The `Files.move` operation removes the original file, while the `StandardCopyOption` value `ATOMIC_MOVE` ensures that the performed steps are performed as a single operation. In the `Files.move` and `Files.copy` methods, both `Path` arguments include the filename. In this scenario, the last entry in the destination `Path` object is *src*, so this is the new name of the moved file.

The `Upgrade.txt` file is not removed from the `Java11` directory and placed in the `NIO\src` sub-directory, because `src` is the destination filename. Both `Path` arguments include the filename, so that the last entry is the filename for both source and destination. If the same filename existed in the destination, the existing file would be overwritten only if the `StandardCopyOption` value `REPLACE_EXISTING` was specified. Otherwise, an exception would be thrown.

The result is not performed in two or three distinct operations, because `StandardCopyOption.ATOMIC_MOVE` is specified as an argument for the `Files.move` method. The `StandardCopyOption` value `ATOMIC_MOVE` ensures that

the performed steps are performed as a single operation.

The `Upgrade.txt` file is not copied from the `Java11` directory because the `Files.move` method removes the source file. The `Files.copy` method does not affect the source file, while the `Files.move` does.

Objective:
Java File I/O

Sub-Objective:
Handle file system objects using java.nio.file API

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.nio.file > Enum StandardCopyOption

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.nio.file > Class Files

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Basic I/O > Reading, Writing, and Creating Files

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Basic I/O > Managing Metadata

---

Question #41 of 50

Question ID: 1328134

Which two collectors are best for handling unordered streams in parallel?

A) `Collectors.groupingBy`

B) `Collectors.groupingByParallel`

C) `reduce(0, (long t, long u)->t+u)`

D) `Collectors.toMap`

E) `Collectors.groupingByConcurrent`

Explanation

Either collector `reduce(0, (long t, long u)->t+u)` or `Collectors.groupingByConcurrent` will best handle unordered streams in parallel. Ideally, a collector for a parallel stream will minimize the loss of concurrency. The simple reduce operation creates a new primitive value for every combination operation, so it does not require concurrency overhead. The `Collectors.groupingByConcurrent` is specifically designed to handle unordered and concurrent collection.

The `Collectors.groupingBy` and `Collectors.toMap` operations do not create concurrent collections. They are inefficient when working with parallel streams.

There is no such collector as `Collectors.groupingByParallel` provided by the Java API.

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 > Reduction

The Java Tutorials > Collections > Aggregate Operations > Parallelism

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

---

Question #42 of 50

Question ID: 1327805

Given the following class:

```
public class Java11 {
   static Customer cust;
   public static void main (String[] args) {
     cust.id = 1;
     cust.name = "Jessica Martinez";
     cust.display();
   }
}

class Customer {
   public int id;
   public String name;
   public boolean preferred;
   public void display() {
     String pOutput = (preferred)? "preferred" : "not preferred";
     System.out.format("%s (%d) is %.", name, id, pOutput);
   }
   public boolean isPreferred() {
     return preferred;
   }
}
```

What is the result?

A) `Jessica Martinez (1) is not preferred.`

B) A compile error is produced.

C) `Jessica Martinez (1) is preferred.`

D) A runtime error is produced.

Explanation

The result is a runtime error because the `cust` variable is `null` and throws a `NullPointerException` when attempting to access `Customer` instance members. By default, `static` and instance members are provided default values automatically. The default value for a reference type is `null`.

The result is not `Jessica Martinez (1) is preferred.` because a runtime error occurs before the `display` method is invoked. If the `cust` variable referenced an instantiated `Customer` object, then the preferred variable would default to the value `false`. This output would be the result if the `preferred` field were set explicitly to `true`.

The result is not `Jessica Martinez (1) is not preferred.` because a runtime error occurs before the `display` method is invoked. If the `cust` variable referenced an instantiated `Customer` object, then this output would be the result.

The result is not a compile error because the code has no syntax issues.

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 #43 of 50

Question ID: 1328028

Given the following output:

```
i: 0
i: 3
i: 6
```

Which three code fragments generate this output?

A)
```
for (var i = 0; i < 8; i += 2) {
   System.out.println("i: " + i);
   i++;
}
```

B)
```
for (var i = 0; i < 8;) {
   System.out.println("i: " + i);
   i += 3;
}
```

C)
```
for (var i = 0; i < 8;) {
   i += 3;
   System.out.println("i: " + i);
}
```

D)
```
for (var i = 0; i < 8; i++) {
   i += 2;
   System.out.println("i: " + i);
}
```

E)
```
for (var i = 0; i < 8; i += 2) {
   i++;
   System.out.println("i: " + i);
}
```

F)
```
for (var i = 0; i < 8; i += 3) {
   System.out.println("i: " + i);
}
```

Explanation

The following three code fragments generate the required output:

```
for (var i = 0; i < 8;) {
   System.out.println("i: " + i);
   i += 3;
}
```

```
for (var i = 0; i < 8; i += 3) {
   System.out.println("i: " + i);
}
```

```
for (var i = 0; i < 8; i += 2) {
   System.out.println("i: " + i);
   i++;
}
```

All code fragments print `i` before modifying its value, so that value for `i` is outputted as 0. The first code fragment increments `i` by 3 at the end of the `for` block manually without specifying an expression for loop modification in the `for` statement. The second code fragment increments `i` by 3 at the end of the `for` block using the loop modification expression in the `for` statement. The last code fragment effectively increments `i` by 3, by specifying an increment of 2 for the loop modification expression in the `for` statement and manually incrementing `i` by 1 at the end of the `for` block.

The remaining code fragments do not generate the required output. Although these code fragments do effectively increment `i` by 3, `i` is modified before its value is first printed.

The code fragment

```
for (var i = 0; i < 8; i++) {
   i += 2;
   System.out.println("i: " + i);
}
```

produces the following output:

```
i: 2
i: 5
i: 8
```

The code fragment

```
for (var i = 0; i < 8;) {
   i += 3;
   System.out.println("i: " + i);
}
```

produces the following output:

```
i: 3
i: 6
i: 9
```

The code fragment

```
for (var i = 0; i < 8; i += 2) {
   i++;
   System.out.println("i: " + i);
}
```

produces the following output:

```
i: 1
i: 4
i: 7
```

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 for statement

---

Question #44 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) `modify(i); System.out.println(i[0] + 15);`

D) `System.out.println(modify(i)+ 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 #45 of 50

Question ID: 1327995

```
01  try {
02    String dbURL = " jdbc:derby://localhost:1527/sample;user=test;password=p@$$w0rd";
03    Connection cn = DriverManager.getConnection(dbURL);
04    String query = "SELECT Customer_ID, Name, City, State, Zip FROM Customer";
05    Statement stmt = cn.createStatement(
06    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
07    ResultSet rs = stmt.executeQuery(query);
08    rs.next(); rs.previous(); rs.next();
09    System.out.println(rs.getInt(1) + "-" + rs.getString(2));
10  } catch (SQLException ex) {
11    System.err.println("ERROR!");
12  }
```

Assume the data table is named `Customer` and accessible using the `dbURL` variable. What is the result of compiling and executing this code?

A) `1-JumboCom`

B) Compilation fails.

C) `ERROR!`

D) `2-Livermore Enterprises`

Explanation

The result of compiling and executing this code is as follows:

`1-JumboCom`

On lines `05-06`, the result set is declared as scrollable and read-only. A scrollable result set supports arbitrary backward and forward movements. By default, the cursor of a result set points before the first row. On line `08`, the first `next` method moves the cursor to the first row, the `previous` method moves the cursor before the first row, and then the second `next` method moves the cursor back to the first row again. Thus, the retrieved column values are from the first row.

Compilation will not fail because this code fragment is syntactically correct.

The result will not be `2-Livermore Enterprises`. This would be the result if the `previous` method was not invoked on line `08`.

The result will not be `ERROR!` because no exception is thrown. An exception would be thrown if the result set was not scrollable as declared on lines `05-06` or the `next` method was invoked only once on line `08`.

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 > Retrieving and Modifying Values from Result Sets

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.sql > Interface ResultSet

---

Question #46 of 50

Question ID: 1328012

Which operator or method should determine whether two `String` variables have the same value?

A) `equals`

B) `contentEquals`

C) `===`

D) `==`

Explanation

The `equals` method determines whether two `String` variables have the same value. The `equals` method is overridden to determine equality between `String` objects based on their character sequence. The `equals` method checks the values *inside* the `String` variables rather than their object references.

The == operator does not determine whether two `String` variables have the same value. The == operator determines whether two `String` variables reference the same object.

The === operator does not determine whether two `String` variables have the same value. This operator is available in JavaScript, but not provided by Java.

The `contentEquals` method does not determine whether two `String` variables have the same value. This method determines whether a `String` variable and `StringBuilder` variable contain the same value.

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 > Numbers and Strings > Strings

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Numbers and Strings > Comparing Strings and Portions of Strings

---

Question #47 of 50

Question ID: 1328007

Consider the code below:

```
public class ClassA {
 @Deprecated
 public void show() {
  System.out.println("Class A show()");
 }
}

public class ClassB {
 // Insert Annotation Here
 public static void main(String args[]){
  ClassA A1 = new ClassA();
  A1.show();
```

```
 }
}
```

Which annotation should you add to the code to ensure the compiler does not display any warning messages?

A) `@SafeVarargs`

B) `@Override`

C) `@SupressWarnings`

D) `@Documented`

Explanation

The correct option is the `@SupressWarnings` annotation. You will need to use the annotation with the warning types you want to suppress indicated as strings:

```
@SuppressWarnings("deprecation")
```

```
@SuppressWarnings("deprecation")
public static void main(String args[]){
 ClassA A1 = new ClassA();
 A1.show();
}
```

`@Override` is incorrect because it does not suppress warnings. It 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.

`@Documented` is incorrect because it does not suppress warnings. This annotation is used to indicate that all elements that use the annotation are documented by JavaDoc.

`@SafeVarargs` is an incorrect option in this scenario. The `@SafeVarargs` annotation is used for methods or constructors that use `varargs` parameters. This annotation is used to ensure that unsafe operations are not performed by the method on the `varargs` parameters.

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 #48 of 50

Question ID: 1327953

Which of the following are principles of how JDK implements a modular structure? (Choose two.)

A) Standard modules can only contain standard API packages.

B) Additional modules not governed by JCP start with the prefix `jdk.`.

C) Standard modules will only depend on standard modules with no dependencies.

D) Standard modules managed by JCP start with the name `java.` .

E) Standard modules must grant implied readability only standard modules.

Explanation

The modular structure of how JDK designs these principles are based off standard modules who are managed by Java Community Program (JCP) and can be identified with the prefix `.java`. The other design principle is that the additional modules outside of the JCP start with `jdk.`.

The main goal of a modular JDK design was to make java implementations easier to maintain, improve security, and improve application performance and to give developers better tools for a more user-friendly experience.

Standard modules can only contain standard API packages. Actually, these standard modules can have API packages that are both standard and non-standard API packages.

Standard modules will only depend on standard modules with no dependencies. Standard modules can have dependencies that exists on more than one standard module and could have dependencies on non-standard dependencies as well.

You would not have standard modules that only grant implied readability only standard modules. If you have a non-standard module, you cannot grant implied readability from a standard module.

Objective:
Java Platform Module System

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

References:

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

---

Question #49 of 50

Question ID: 1327803

Given the following class:

```
1.   public class Machine {
2.     static String manufacturer;
3.     public static void main (String[] args) {
4.       //Insert code here
5.     }
6.   }
```

Which three code fragments correctly assign a value to the manufacturer field at line 4?

A) `Machine.manufacturer = "Oracle";`

B) `manufacturer = "Oracle";`

C)
```
Machine myMachine = new Machine();
myMachine.manufacturer = "Oracle";
```

D) `this.manufacturer = "Oracle";`

E) `super.manufacturer = "Oracle";`

Explanation

The following code fragments correctly assign a value to the manufacturer field at line 4:

```
manufacturer = "Oracle";
```

```
Machine.manufacturer = "Oracle";

Machine myMachine = new Machine();
myMachine.manufacturer = "Oracle";
```

All three code fragments access the `manufacturer` field as a `static` member. `static` members do not require class instantiation for access.

The first code fragment accesses the `manufacturer` field directly within the same class because the main method is also a `static` member. The second code fragment is the preferred approach to accessing `static` members outside the class. This code fragment specifies the class name `Machine` and `manufacturer` field using dot notation.

The third code fragment is confusing because it accesses the `static` member as if it were an instance member. Although this is syntactically correct, this approach is not recommended. This code fragment will compile because both `static` and instance members are available to objects.

The code fragment that specifies the keyword `this` does not correctly assign a value to the `manufacturer` field. The `this` keyword references the current instance, which is unavailable in a static context.

The code fragment that specifies the keyword `super` does not correctly assign a value to the `manufacturer` field. The `super` keyword references the parent class associated with the current instance, which is unavailable in a static 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 > Classes and Objects > Understanding Instance and Class Members

---

Question #50 of 50

Question ID: 1327967

Given:

```
import java.io.*;

public class FileLoader {
  public static void main(String[] args) {
    try (FileInputStream fs = new FileInputStream("input.txt")) {
      System.setOut(new PrintStream(new FileOutputStream("output.txt")));
      System.out.print(fs.read());
```

```
    } catch (Exception ex) {
      System.err.print("Error reading file");
    }
  }
}
```

And the contents of the `input.txt` file:

```
java basic io
```

What is the result?

A) The output `java basic io`

B) The program runs but prints no output.

C) The output `106`

D) The output `j`

E) Compilation fails.

Explanation

The program runs, but prints no output, because the following statement redirects the standard output stream to the file `output.txt`:

```
System.setOut(new PrintStream(new FileOutputStream("output.txt")));
```

By default, the standard streams use the console. This code redirects the standard output so that the console no longer displays output. Because `FileInputStream` is a byte stream, the contents of `output.txt` will contain the byte representation of the letter *j*, specifically the value 106.

There will be no console output because the standard output is redirected to the file `output.txt`. Also, `FileInputStream` is a byte stream, not a character stream, so the value `106`, not `j` or `java basic io`, will be written to `output.txt`.

Compilation will not fail because there are no syntax errors in the code.

Objective:
Java File I/O

Sub-Objective:
Read and write console and file data using I/O Streams

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > Basic I/O > I/O from the Command Line