Quick Quiz July 14, 2022

Test ID: 216187296

Question #1 of 50

Question ID: 1328205

You need to create an application that monitors the weather events on Earth.

Which class should you use to track meteorological events?

A) `Duration`

B) `Instant`

C) `TemporalUnit`

D) `Period`

Explanation

You should use the `Instant` class, which represents a single point on the current timeline and is often used for an event timestamp. The `Instant.now()` method creates an object representing single point on the current timeline, based on the system UTC clock.

The `Duration` class is an incorrect option because it represents an amount of time in terms of seconds and nanoseconds. You use the `Duration` class to represent elapsed time between two points of time. The method `Duration.between(inst1,inst2)` creates an object representing the elapsed time between two `Instant` objects `inst1` and `inst2`, in terms of seconds and nanoseconds.

The `TemporalUnit` interface is an incorrect option because it is a Java interface that represents the duration of time between two points of time. A `TemporalUnit` interface allows you to measure time in units. It is implemented by the `ChronoUnit` enumeration.

The `Period` class is an incorrect option because it represents an amount of time in terms of years, months, and days.

Objective:
Localization

Sub-Objective:
Implement Localization using Locale, resource bundles, and Java APIs to parse and format messages, dates, and numbers

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Java Date and Time Classes

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.time > Package java.time

---

Question #2 of 50

Question ID: 1327956

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

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

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

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

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

Explanation

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

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

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

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

```
module javaprogam/finance/usingmymodules {

}
```

Objective:
Java Platform Module System

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

References:

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

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

---

Question #3 of 50

Question ID: 1327768

Given the following:

```
public class Java11 {
   public static void main (String[] args) {
     String s1 = "salty";
     String s2 = new String("salty");
     String s3 = s2;
     if (s1.equals(s2) && s2.equals(s3))
       System.out.println("We are equal!");
     if (s1 == s2 && s2 == s3)
       System.out.println("We are really equal!");
   }
}
```

What is the result when this program is executed?

A) `We are really equal!`

B) Nothing prints.

C) `We are equal!`

D)
```
We are equal!
We are really equal!
```

Explanation

The result is the output `We are equal!`. This is because all three variables have the same character sequence, but do not reference the same `String` object. The `equals` method is overridden to determine equality between `String` objects based on their character sequence. When comparing objects, the `==` operator determines whether the same object is being referenced. In this case, the variables `s2` and `s3` reference the same `String` object, but the variables `s1` and `s2` do not.

The result will not include the output `We are really equal!`. The `==` operator determines whether two `String` variables reference the same object. Although the variables `s2` and `s3` do reference the same object, the variables `s1` and `s2` do not.

The result is not nothing prints. All three variables have the same character sequence, so the conditional expression in the first if statement will evaluate to `true` and print `We are equal!`.

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 > 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 #4 of 50

Question ID: 1327821

Given:

```
public class SuperEmptyClass {
   public SuperEmptyClass (Object obj) { /*Implementation omitted*/ }
}

public class EmptyClass extends SuperEmptyClass { }
```

Which constructor is provided to `EmptyClass` by the compiler?

A) `public EmptyClass(Object obj) { super(obj);}`

B) `public EmptyClass(Object obj) {}`

C) `public EmptyClass() {}`

D) `public EmptyClass() { super();}`

Explanation

The following constructor is provided to the `EmptyClass` by the compiler:

```
public EmptyClass() {super();}
```

This constructor is the default constructor. If no constructor is defined for a class, then the compiler will automatically provide the default constructor. The default constructor specifies no parameters and invokes the parameterless constructor of the superclass. In this scenario, the code will fail compilation because `SuperEmptyClass` does not provide a parameterless constructor.

The following constructor is not provided to EmptyClass by the compiler:

```
public EmptyClass() { }
```

The default constructor is not empty. The default constructor invokes the parameterless constructor of the superclass.

The constructors that specify an `Object` parameter are not provided to `EmptyClass`. The default constructor specifies no parameters.

Objective:

Java Object-Oriented Approach

Sub-Objective:

Initialize objects and their members using instance and static initializer statements and constructors

References:

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

---

Question #5 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) 0

B) 1

C) 5

D) 10

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 #6 of 50

Question ID: 1328172

Which statement is true about livelocked threads?

A) Thread A and Thread B are said to be livelocked if they are blocked forever while waiting for access to the same resource.

B) Thread A is said to livelocked if it is blocked from a shared resource while Thread B has access.

C) Thread A is said to be livelocked if it is frequently unable to access a resource shared with Thread B.

D) Thread A and Thread B are said to be livelocked if they are too busy responding to each other to complete their work.

Explanation

Thread A and Thread B are said to be livelocked if they are too busy responding to each other to complete their work. In this scenario, Thread A and Thread B are not blocked, but the access mechanism itself is consuming their normal execution.

Thread A and Thread B are not said to be livelocked if they are blocked forever while waiting for access to the same resource. This situation describes deadlocking.

Thread A is not said to be livelocked if it is frequently unable to access a resource shared with Thread B. This situation describes thread starvation.

Thread A is not said to livelocked if it is blocked from a shared resource while Thread B has access. This situation describes thread synchronization.

Objective:
Concurrency

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

References:

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

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

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Starvation and Livelock

---

Question #7 of 50

Question ID: 1327858

Given:

```
abstract class Voter {
   private String partyAffiliation;
   public Voter(String partyAffiliation) {
     this.partyAffiliation = partyAffiliation;
   }
   public String getPartyAffiliation() { return partyAffiliation; }
   public abstract void vote();
}
```

Which two classes correctly use `Voter`?

A)
```
class AmericanVoter implements Voter {
   public void vote() {System.out.println("Ballot cast!");}
}
```

B)
```
abstract class AmericanVoter extends Voter {}
```

C)
```
abstract class AmericanVoter implements Voter {}
```

D)
```
class AmericanVoter extends Voter {
   public AmericanVoter() {super("Independent");}
   public void vote() {System.out.println("Ballot cast!");}
}
```

E)
```
class AmericanVoter extends Voter {
   public String getPartyAffiliation() {return "Independent"; }
}
```

F)
```
abstract class AmericanVoter extends Voter {
   public AmericanVoter() {super("Independent");}
}
```

Explanation

The following classes correctly use `Voter`:

```
abstract class AmericanVoter extends Voter {
   public AmericanVoter() {super("Independent");}
}

class AmericanVoter extends Voter {
   public AmericanVoter() {super("Independent");}
   public void vote() {System.out.println("Ballot cast!");}
}
```

A subclass of an abstract class must either provide implementation of abstract members by overriding them or declare itself abstract. The first class does not override the abstract method `vote`, so it declares itself as abstract. The second class does override `vote`, so this class does not need to be declared as abstract. Both classes provide a constructor that explicitly invokes the superclass constructor because the superclass constructor is not parameterless.

The classes that do not explicitly invoke the superclass constructor do not correctly use `Voter`. Because `Voter` declares a constructor with a parameter, subclasses must contain a constructor that explicitly invokes this superclass constructor.

The classes that use the `implements` keyword do not correctly use `Voter`. Subclasses of abstract classes must use inheritance to provide implementation, so the `extends` keyword should be used instead.

The class that overrides `getPartyAffiliation`, not `vote`, does not correctly use `Voter`. The `vote` method is declared as abstract, so that method must be overridden. Overriding any other method is optional.

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

---

Question #8 of 50

Question ID: 1327907

You define the following interface to represent shippable goods:

```
interface Addressable {
  String getStreet();
  String getCity();
}
```

You want to add the following method to the `Addressable` interface:

```
String getAddressLabel() {
  return getStreet() + "\n" + getCity();
}
```

Unfortunately, the interface is implemented by many classes in the project. It is not feasible to modify every class definition and have existing classes fail to compile. What is the best solution to this design problem?

A) Add a static method to the interface

B) **Add a default method to the interface**

C) Convert the interface to an abstract class and add the method to the class

D) Create a helper, or utility, class and add the method to that class

Explanation

The default method mechanism is intended specifically to support interface evolution as described in this scenario. It allows you to add new instance methods to an interface along with a default implementation. That implementation can refer to an instance of the interface through the implied variable this, just as an instance method defined in a class can. If any particular implementation of the interface has reason to do so, the implementation can be replaced by overriding. Because of this, this option is the correct answer.

Converting the interface to an abstract class might work, but this approach has two distinct problems. First, every class that implements the interface must be changed to say extends Addressable, and a specific goal was to avoid

extensive code rewriting. Second, if any of the classes that implement Addressable already inherit from anything other than Object, then the solution will fail, since Java permits only one direct parent class.

Creating a utility class will work, but this approach has its own set of issues. It avoids making changes to other parts of code that do not yet need to use the new feature. Prior to Java 8, this would probably have been a good solution. Two disadvantages, though, are that the behavior must be in a separate class, and that such a method must be static. Being in a different class, it lacks cohesion with the other behaviors related to *addressableness*. It will not be so easy to know where to look. Being static will mean that the code does not look like object-oriented code. Instead of calling `myPlace.getAddressLabel()`, the code would probably resemble `AddressableUtils.getAddressLabel(myPlace)`. More importantly, static methods do not support overriding, so the behavior becomes very hard to modify for any given Addressable. In short, this approach makes the behavior brittle.

Creating a static method in the interface is technically feasible, but it has the same issue described for static behavior: it is brittle and poorly designed.

Objective:
Java Object-Oriented Approach

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

References:

The Java Tutorials > Learning the Java Language > Interfaces and Inheritance > Default Methods

---

Question #9 of 50

Question ID: 1327832

Which access modifier will permit access to class members only from subclasses and other classes in the same package?

A) `private`

B) `protected`

C) `public`

D) **no modifier**

Explanation

The access modifier `protected` will permit access to class members only from subclasses and other classes in the same package. This access level is recommended for methods that may be overridden or overloaded in subclasses.

The access modifier `private` will not permit access to class members only from other classes. This modifier will restrict access to members only within the same class.

The access modifier `public` will not restrict access to class members from other classes. This modifier will allow access to members from any class.

Default access, by specifying no modifier, will not permit access to class members from subclasses in other packages. This modifier will allow access to members from any class within the same package.

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 #10 of 50

Question ID: 1328024

Given the code fragment:

```
int i = 0;
while (i < 10 ? 1 : 0) {
   i++;
}
System.out.println(i);
```

What is the result?

A) Code compilation fails.

B) `10`

C) `9`

D) Code throws a runtime exception.

E) `0`

Explanation

Code compilation fails because the expression `i < 10 ? 1 : 0` is not valid for a `while` statement. The conditional operator (`?:`) uses a `boolean` expression but can return a data type other than `boolean` when the expression is `true` or `false`. In this case, the return value will be an `int`, either `1` or `0`, which is an incompatible type for a `while` statement.

The code does not throw a runtime exception because the code does not compile.

The result will not be the output `0`, `9`, or `10`. The syntax error prevents the code from being compiled and run. If the while expression were replaced with `i < 10`, then the result would be the output `10`.

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 #11 of 50

Question ID: 1327896

Which type defines a valid interface?

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

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

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

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

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 #12 of 50

Question ID: 1328135

Given the code fragment:

```
System.out.println(IntStream.iterate(0, (n)-> n+1)
  // line n1
  .findAny());
```

The result is the consistent output of `OptionalInt[0]`. If the code `.parallel()` is added at line n1, which statement(s) are true? (Choose all that apply.)

A) Compilation will fail.

B) Lines of output increase.

C) Output will become unpredictable.

D) Execution time decreases.

Explanation

The output will be unpredictable. It could be `OptionalInt[0]` or `OptionalInt[1024]`.The `findAny` method will terminate stream processing as soon as it receives an item from the stream. For a sequential, ordered stream, that item will be the first item that was produced by the stream source (in this case, the value zero). However, with a parallel stream, the input stream is broken into chunks, and chunks are processed potentially concurrently on multiple threads. In this situation, it is possible that a data item that was not originally first might be first to reach the `findAny` method. If that occurs, then the output will report `Optional[`*n*`]` where *n* is a number that might not be zero.

Execution time may not decrease. The purpose of parallel streams is to allow the use of concurrent hardware (multiple CPU cores). With a simple task, the overhead of concurrency management will exceed any benefit, and the execution time will increase.

Lines of output will not increase. In the parallel situation, many more items of data will probably be passed down the stream, but the `findAny` method always finds a single item.

Compilation will not fail. The method `parallel()` is a valid stream method, and the code will both compile and execute.

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 #13 of 50

Question ID: 1327933

Given the code fragment:

```
Set<String> set = new TreeSet<>();
set.addAll(Arrays.asList(new String[] {"first", "second"}));
```

Which statement will throw an exception at runtime?

A) `set.add(null);`

B) `set.add("first");`

C) `set.add("");`

D) `set.add("1");`

E) `set.add(1);`

Explanation

The statement `set.add(null);` will throw an exception at runtime because an empty object reference cannot be tested for equality. A `Set` cannot contain duplicate elements, so an implicit invocation of equals occurs when any element is added. A null reference is not a valid object upon which to invoke methods.

The statement `set.add(1);` will not throw an exception at runtime because this statement fails compilation. The generic `TreeSet` limits data types to `String` objects, so `int` values are prohibited by the compiler.

The following statements will compile and not throw an exception at runtime:

```
set.add("1");
set.add("first");
set.add("");
```

The first and third statements will add new elements, while the second statement will not add the element because it already exists in the `Set`.

Objective:
Working with Arrays and Collections

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

References:

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

---

Question #14 of 50

Question ID: 1328105

You are asked to compute the total weight of all red toys in a factory. Given the following:

```
int weight = toys.stream()
  .filter(e -> e.getShade() == RED)
  // INSERT CODE
  .sum();
```

Which code snippet(s) could you insert so the code will execute? (Choose all that apply.)

A) `.mapToDouble(e -> e.getSize())`

B) `.map(e -> e.getSize())`

C) `.peek(e -> e.getSize())`

D) `.mapToInt(e -> e.getSize())`

Explanation

You could insert either of these lines of code for it to compute correctly:

`.mapToInt(e -> e.getSize())`

`.mapToDouble(e -> e.getSize())`

The other options are incorrect because the `peek` method will only view the results it receives through the method after it executes. When the `map` method is used without the `Int` or `Double` versions, it will not return the result in numeric format as required to compute the total weight of the toys.

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 #15 of 50

Question ID: 1327954

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

A) `java.compiler`

B) `java.base`

C) `java.logging`

D) `java.desktop`

Explanation

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

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

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

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

```
var Mynewstr = "CyberVista";

var MyRepeatObject = Mynewstr.repeat(2);

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

System.out.println(" ");

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

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

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

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

Objective:
Java Platform Module System

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

References:

Oracle Documentation > Java SE 11 API > Class String

Question #16 of 50

Question ID: 1327881

Given:

```
public abstract class Writer {
   public void write() {System.out.println("Writing...");}
}

public class Author extends Writer {
   public void write() {System.out.println("Writing book");}
}

public class Programmer extends Writer {
   public void write() {System.out.println("Writing code");}
   public static void main(String[] args) {
     Writer w = new Programmer();
     w.write();
   }
}
```

What is the result?

A) `Writing...`

B) Compilation fails.

C) `Writing code`

D) An exception is thrown at run time.

E) `Writing book`

Explanation

The result is the output `Writing code`. This is because the object type determines which implementation of the `write` method is executed. Because the object type is `Programmer`, the `write` method found in `Programmer` will output `Writing code`.

The result is not the output `Writing ...` because the reference type does not determine which implementation of the `write` method is executed. The reference type is `Writer`, not the object type.

The result is not the output `Writing book`. This is because `Author` is not the object type.

Compilation does not fail, nor is an exception thrown at run time. The code is syntactically and semantically correct. A reference type can be a supertype of the object type.

Objective:
Java Object-Oriented Approach

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

References:

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

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

---

Question #17 of 50

Question ID: 1328197

You create code to store date and time information using Java classes:

```
LocalDate.of(2016, 8, 25); // 1
LocalDate.of(2016, Month.APRIL, 50); // 2
LocalDate date = new LocalDate(); // 3
LocalDate.now(); // 4
```

Which of these statements will generate a runtime or compiler error? (Choose all that apply.)

A) All of these

B) None of the above

C) 3

D) 2

E) 4

F) 1

Explanation

The option `LocalDate.of(2016, Month.APRIL, 50);` will generate a runtime error because it attempts to pass an invalid number of days to the `LocalDate` object, in this case 50.

The option `LocalDate date = new LocalDate();` will generate a compiler error because you cannot use an empty constructor on a Java date time class.

None of the other options will generate a compiler error because they are valid Java date time class applications.

Objective:
Localization

Sub-Objective:
Implement Localization using Locale, resource bundles, and Java APIs to parse and format messages, dates, and numbers

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Java Date and Time Classes

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Java Date Time Formatter

Question #18 of 50

Question ID: 1327786

Given the following code fragment:

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

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

A) `Customer()`

B) `cust`

C) `Customer cust`

D) `new`

Explanation

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

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

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

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

Objective:
Java Object-Oriented Approach

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

References:

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

Question #19 of 50

Question ID: 1328018

Which statement is true about `if` statements nested in `if` and `else` statements?

A) The outer `else` statement is evaluated only if the inner `if` statement(s) are evaluated.

B) The inner `if` statement(s) are evaluated only if the outer `if` statement is `true`.

C) The outer `else` statement is evaluated only if the inner `if` statement(s) are `true`.

D) The inner `if` statement(s) are evaluated only if the outer `if` statement is evaluated.

Explanation

The inner `if` statement(s) are evaluated only if the outer `if` statement is `true`. If the `if` statement is `false`, then the inner `if` statement(s) are not evaluated.

The inner `if` statement(s) are not evaluated only if the outer `if` statement is evaluated. The inner `if` statement(s) are not evaluated if the outer `if` statement evaluates to `false`.

The outer `else` statement is not evaluated only if the inner `if` statement(s) are evaluated or evaluated as `true`. Whether inner `if` statement(s) are evaluated or evaluated as `true` does not affect whether the outer `else` statement is evaluated. The `else` statement is reached when its associated `if` statement(s) are `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 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 #20 of 50

Question ID: 1327947

Given the following:

```
char[][] charArray2D = {{'c','u','p'},{'o'},{'f'},{'j','a','v','a'}};
```

Which two expressions will evaluate to `false`?

A) `charArray2D.length > 4`

B) `charArray2D.getClass().isArray()`

C) `charArray2D[0].length < 2`

D) `charArray2D[1].length < 2`

E) `charArray2D[1].getClass().isArray()`

F) `charArray2D[0].getClass().isArray()`

Explanation

The expressions `charArray2D.length > 4` and `charArray2D[0].length < 2` will evaluate to `false`. The first expression compares the number of elements in the first dimension, which is 4. The second expression compares the number of elements in the first array, which is 3.

The expressions that use the `getClass().isArray()` invocation will evaluate to `true`. `charArray2D` is an array, as is each of its child elements.

The expression `charArray2D[1].length < 2` will evaluate to `true`. This is because the second array has only one element.

Objective:
Working with Arrays and Collections

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

References:

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

---

Question #21 of 50

Question ID: 1328193

Given the contents of the following properties files:

```
#MessageTextBundle
strContinue = Are you sure you want to continue?

#MessageTextBundle_fr
strContinue = Êtes-vous sûr de que vouloir continuer?

#MessageTextBundle_fr_CA
strContinue = Voulez-vous continuer?
```

The application should display the following output:

```
Êtes-vous sûr de que vouloir continuer?
```

Which code fragment(s) will result in this output? (Choose all that apply.)

A)
```
ResourceBundle msgs =
  ResourceBundle.getBundle("MessageTextBundle", new
Locale("fr", "BR"));
System.out.println(msgs.getString("strContinue"));
```

B)
```
ResourceBundle msgs =
  ResourceBundle.getBundle("MessageTextBundle", new
Locale("es", "MX"));
System.out.println(msgs.getString("strContinue"));
```

C)
```
ResourceBundle msgs =
  ResourceBundle.getBundle("MessageTextBundle", new
Locale("fr", "US"));
System.out.println(msgs.getString("strContinue"));
```

D)
```
ResourceBundle msgs =
  ResourceBundle.getBundle("MessageTextBundle", new
Locale("fr", "CA"));
System.out.println(msgs.getString("strContinue"));
```

Explanation

The following code fragments will display the desired output:

```
ResourceBundle msgs =
ResourceBundle.getBundle("MessageTextBundle", new Locale("fr", "US"));
System.out.println(msgs.getString("strContinue"));
```

```
ResourceBundle msgs =
ResourceBundle.getBundle("MessageTextBundle", new Locale("fr", "BR"));
System.out.println(msgs.getString("strContinue"));
```

The `MessageTextBundle_fr.properties` file contains the value *Êtes-vous sûr de que vouloir continuer?* for the `strContinue` property. This properties file will match any locale that specifies the French language but does not match the Canada region. These two code fragments specify the French US and French Brazil locales. The `getString` or `getObject` methods accept a `String` key and returns a string or serialized object, respectively.

The code fragment that specifies the French language and Canada region will output `Voulez-vous continuer?` because this locale matches the `MessageTextBundle_fr_CA.properties` file.

The code fragment that specifies the Spanish language and Mexico region will output `Are you sure you want to continue?` because this locale does not match any specific properties file. Therefore, it will use the default `MessageTextBundle.properties` file.

Objective:
Localization

Sub-Objective:

Implement Localization using Locale, resource bundles, and Java APIs to parse and format messages, dates, and numbers

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Internationalization > Setting the Locale > Isolating Locale-Specific Data > Backing a ResourceBundle with Properties Files

---

Question #22 of 50

Question ID: 1327809

Which code fragment is a valid method declaration for an arbitrary number of values?

A)
```
public Result performOperation (Object varargs, OperationType
type) {
   //implementation omitted
}
```

B)
```
public Result performOperation (OperationType type, Object...
args) {
   //implementation omitted
}
```

C)
```
public Result performOperation (Object... args, OperationType
type) {
   /implementation omitted
}
```

D)
```
public Result performOperation (OperationType type, Object
varargs) {
   //implementation omitted
}
```

Explanation

The following code fragment is a valid method declaration for an arbitrary number of values:

```
public Result performOperation (OperationType type, Object... args) {
   //implementation omitted
}
```

Varargs allows an arbitrary number of arguments of the same data type. Varargs are declared using the ellipse (`...`) and must be declared as the last parameter in a method.

The code fragments that specify the name varargs do not indicate an arbitrary number of values. Varargs are declared using the ellipse (`...`), not using a specific name for the parameter.

The code fragments that place varargs as the first parameter of the method are not valid. Varargs must be declared as the last parameter in a method.

Objective:
Java Object-Oriented Approach

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

References:

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

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

---

Question #23 of 50

Question ID: 1327777

Which statement is true about string equality?

A) The == operator compares character sequences.

B) The `equals` method compares object references.

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

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

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 #24 of 50

Question ID: 1327760

Given the following:

```
public class Java11 {
   public static void main (String[] args) {
     int x = 1;
     modifyVar(x + 5);
     System.out.println("x: " + x);
   }
   public static void modifyVar(int var) {
     var = 10;
   }
}
```

What is the result when this program is executed?

A) `x: 10`

B) `x: 16`

C) `x: 6`

D) `x: 1`

Explanation

The following output is the result when the program is executed:

```
x: 1
```

Variables of primitive types hold only values, not references. When `var` is assigned to `x` because of invoking the `modifyVar` method, the value of `x` is copied into `var` . The actual argument is `x + 5`, so that `var` is initially set to `6`. Within the `modifyVar` method, the `var` variable is set to `10`, overwriting its previous value. Because `var` only received a copy of `x` in the expression `x + 5`, any modifications to `var` are discarded after returning from the `modifyVar` method. Thus, the value of `x` remains `1`.

The key differences between primitive variables and reference variables are:

- Reference variables are used to store addresses of other variables. Primitive variables store actual values. Reference variables can only store a reference to a variable of the same class or a sub-class. These are also referred to in programming as *pointers*.
- Reference types can be assigned `null` but primitive types cannot.
- Reference types support method invocation and fields because they reference an object, which may contain methods and fields.
- The naming convention for primitive types is camel-cased, while Java classes are Pascal-cased.

The results will not be the output with `x` set to `6` or `10`, because only a copy of the `x` value is stored in `var` for the `modifyVar` method. Initially, `var` is set to `6` and then set to `10`.

The result will not be the output with `x` set to `16`, because only a copy of the `x` value is stored in `var` and `var` is never set to `16`. The final value of `var` is `10`, not `16`.

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 Language Specification > Chapter 4. Types, Values, and Variables > 4.12. Variables

Primitive vs Reference Data Types

Question #25 of 50

Question ID: 1327785

Consider the following code fragment:

```
int str1 = 0;   // line 1
Comparator<String> cmp =
  (str1, str2) -> str1.length() - str2.length();   // line 2
```

What would you need to do to make this compile successfully?

A) remove line 1

B) change `int str1` to `int str2`

C) remove line 2

D) None of these

Explanation

You should remove line 1 because it is not legal to declare a variable that has the same name as a variable inside a lambda expression within the same scope. 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.

The option stating to remove line 2 is incorrect. To make the code compile correctly, line 1 needs to be removed so that the declaration of a variable with the same name as one inside the lambda expression can be removed.

The option stating to change `str1` to `str2` is incorrect because even `str2` is a variable which exists inside the lambda expression and so will generate a similar error.

Objective:
Working with Java Data Types

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

References:

Java Language Changes for Java SE 11

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 #26 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 `meet` method is implicitly package-level.

B) The class is implicitly public.

C) The `sight` method is implicitly public.

D) The class is implicitly abstract.

E) The `sight` method is implicitly abstract.

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 #27 of 50

Question ID: 1328152

Which statement is true about `Runnable` objects?

A) `Runnable` objects must implement the `run` and `call` methods.

B) `Runnable` objects can be run using the execute method of the `ExecutorService` interface.

C) `Runnable` objects return a value when executing.

D) `Runnable` objects can throw checked exceptions.

Explanation

`Runnable` objects can be run using the `execute` method of the `ExecutorService` interface. When executing a task using the `ExecutorService` class, you can invoke either the `execute` or `submit` methods. The `execute` method supports `Runnable` objects that do not return a value, while the `submit` method supports both `Runnable` and `Callable` objects. The `submit` method returns a `Future` object representing the pending results of the task.

`Runnable` objects cannot return a value when executing, unlike `Callable` objects. The method signature for the `run` method has no return type.

`Runnable` objects cannot throw checked exceptions, unlike `Callable` objects. The method signature for the `run` method does not include a declared exception.

`Runnable` objects do not need to implement the `call` method. `Runnable` objects must implement the `run` method, while `Callable` objects must implement the `call` method.

Objective:
Concurrency

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

References:

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

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

Question #28 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 `106`

B) The output `java basic io`

C) The program runs but prints no output.

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

---

Question #29 of 50

Question ID: 1328185

Consider the following code:

```
class Client implements Serializable {
  double account = 90876543;
  public String toString() {
    return String.valueOf(account);
  }
}
```

How would you ensure that sensitive data like an account number is protected during serialization of this class?

A) Using a `serialVersionUID`

B) Use `clone`

C) Use `readObjectNoData`

D) Use `readObject`

Explanation

You should use a `serialVersionUID`. The following code illustrates the use of a serial version UID for securing the serializable class:

```
class Client implements Serializable, Externalizable {
  private static final long serialVersionUID = 21L;
  transient double account = 90876543;
  public String toString() {
    return String.valueOf(numOfWeapons);
  }
}
```

`clone`, `readObject`, and `readObjectNoData` are incorrect because they do not aid in securing a serializable class. These methods should not be called from within a class constructor for security reasons because each of them is an overridable method.

To protect sensitive fields in serializable classes, the following need to be done:

- Fields need to be made `transient`
- The `Externalizable` interface must be implemented
- The `serialPeristentFields` array fields need to be defined correctly
- The `writeReplace` method must be implemented as it replaces the instance with a proxy
- The `writeObject` method must be implemented

The process of serialization circumvents the field access control aspect of Java that provides security. For this reason, serialization must be done very carefully. Deserialization must be avoided altogether when dealing with untrusted data.

When a class is made serializable, a `public` interface is invariably created for all fields of that class. A `public` constructor is added to a class when it is serialized. Additionally, lambdas and functional interfaces come with security issues and so should be used with care.

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 #30 of 50

Question ID: 1327922

Given the following contents of a source file:

```
class ThreadRunner <T extends Runnable> {
  private T runnable;
  public ThreadRunner(Class<T> c) {
    try {
      this.runnable = __________;
    } catch(Exception ex) {}
  }
  public void start() {
    Thread th = new Thread(this.runnable);
    th.start();
  }
}
class RunnableObject implements Runnable {
  public void run() {System.out.println("Running!");}
  public static void main(String[] args) {
    new ThreadRunner<>(RunnableObject.class).start();
  }
}
```

Which code segment should be inserted to compile the source file?

A) `new T()`

B) `c.newInstance()`

C) `new Runnable<T>()`

D) `new c()`

E) `new Class<T>()`

Explanation

The code segment `c.newInstance()` should be inserted to compile the source file. Generic parameters cannot be directly instantiated because the compiler must replace all parameters with existing types that match either the required bounds or `Object`, if the type parameter is unbounded. This is known as *type erasure*. To overcome this limitation, a `Class` argument must be accepted and the `newInstance` method invoked to delegate instantiation.

This approach does require exception handling at runtime, because the generic type may not have an accessible parameterless constructor, or it may represent a type that does not support instantiation.

The code segment `new T()` should not be inserted to compile the source file. This code segment will cause a compilation error because generic parameters cannot be directly instantiated.

The code segment `new c()` should not be inserted to compile the source file. This code segment will cause a compilation error because a type named `c` is neither declared in the source file nor available through the Java API.

The code segment `new Class<T>()` should not be inserted to compile the source file. This code segment will cause a compilation error because the `Class` constructor is not accessible.

The code segment `new Runnable<T>()` should not be inserted to compile the source file. This code segment will cause a compilation error because the `Runnable` interface does not accept a generic parameter, and interfaces cannot be instantiated.

Objective:
Working with Arrays and Collections

Sub-Objective:
Use generics, including wildcards

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Generics (Updated) > Type Erasure

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Generics (Updated) > Generic Types

Oracle Technology Network > Java SE > Java Language Specification > Classes > Class Declarations > 8.1.2. Generic Classes and Type Parameters

---

Question #31 of 50

Question ID: 1328045

Given the following code:

```
public String messWithString(String str)
  throws StringIndexOutOfBoundsException {
   //implementation omitted
}
```

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

A) Specify `StringIndexOutOfBoundsException`.

B) Catch `StringIndexOutOfBoundsException`.

C) No handling action is required.

D) Invoke the method within a `try` block without an associated `catch` or `finally` block.

Explanation

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

Neither caching nor specifying `StringIndexOutOfBoundsException` is required for the code to compile. Runtime exceptions are abnormal conditions internal to the application and often are unrecoverable. The compiler does not check catching and specifying `RuntimeException` and its subclasses.

Invoking the method within a `try` block without an associated `catch` or `finally` block is not required for the code to compile. A `try` block is required to have an associated `catch` or `finally` block. Declaring a `try` block without an associated `catch` or `finally` block is a syntax error and will cause the code to fail compilation.

Objective:
Exception Handling

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

References:

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

---

Question #32 of 50

Question ID: 1327999

Given the code segment:

```
String sqlQuery = "UPDATE Agents SET fname=? , lname=? where id=?";
PreparedStatement prepSt = con.prepareStatement(sqlQuery);
int res = 0;
prepSt.setString(1, "James");
prepSt.setString(2, "Bond");
prepSt.setLong(3, 007);
//Insert code here
prepSt.setString(1, "Alec");
prepSt.setString(2, "Trevelyan");
```

```
prepSt.setLong(3, 006);
//Insert code here
```

Which code should you insert to execute the code without errors?

A) `res = prepSt.executeUpdate();`

B) `prepSt = con.prepareStatement(sqlQuery);`

C) `res = prepSt.executeBatch();`

D) `prepSt = con.prepareCall(sqlQuery);`

Explanation

You should insert the following code:

```
String sqlQuery = "UPDATE Agents SET fname=? , lname=? where id=?";
PreparedStatement prepSt = con.prepareStatement(sqlQuery);
int res = 0;
prepSt.setString(1, "James");
prepSt.setString(2, "Bond");
prepSt.setLong(3, 007);
res = prepSt.executeUpdate();
prepSt.setString(1, "Alec");
prepSt.setString(2, "Trevelyan");
prepSt.setLong(3, 006);
res = prepSt.executeUpdate();
```

`PreparedStatement` objects can be executed multiple times with different values for their query parameters. To update twice, simply invoke the `execute`*`XXX`* after resetting the values for the query parameters.

You should not insert the following code:

```
prepSt = con.prepareStatement(sqlQuery);
```

This code will create a new statement rather than reusing the existing `PreparedStatement` object.

You should not use the following code:

```
prepSt = con.prepareCall(sqlQuery);
```

This code is used to prepare a `CallableStatement` object that references a stored procedure, not an ad hoc query specified in `sqlQuery`.

You should not use the following code:

```
res = prepSt.executeBatch();
```

This code is incorrect because the statement returns an array of `int` values, not a single `int` primitive value. This code could, however, be used at the end of the existing statement if the `addBatch` method were invoked as

specified below:

```
String sqlQuery = "UPDATE Agents SET fname=? , lname=? where id=?";
PreparedStatement prepSt = con.prepareStatement(sqlQuery);
prepSt.setString(1, "James");
prepSt.setString(2, "Bond");
prepSt.setLong(3, 007);
prepSt.addBatch();
prepSt.setString(1, "Alec");
prepSt.setString(2, "Trevelyan");
prepSt.setLong(3, 006);
prepSt.addBatch();
int[] res = prepSt.executeBatch();
```

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 > The Java Tutorials > JDBC™ Database Access > JDBC Basics > Using Prepared Statements

---

Question #33 of 50

Question ID: 1327804

Given the following:

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

What is the result?

A) A compile error is produced.

B) `null dpi - 112.2 X 56.8`

C) `0 dpi - 112.2 X 56.8`

D) A runtime error is produced.

Explanation

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

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

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

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

Objective:
Java Object-Oriented Approach

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

References:

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

---

Question #34 of 50

Question ID: 1328134

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

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

B) `Collectors.groupingByConcurrent`

C) `Collectors.toMap`

D) `Collectors.groupingBy`

E) `Collectors.groupingByParallel`

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 #35 of 50

Question ID: 1328153

Given the following code fragment:

```
01  ExecutorService ex;
02  ex = _____________________________;
03  Thread th = new CalcThread();
04  ex.submit(th);
```

Which code fragment, when inserted independently at line `02`, will ensure the `CalcThread` instance will run in a single thread?

A) `Executors.newSingleThreadExecutor()`

B) `Executors.createSingleThreadExecutor()`

C) `ExecutorService.getSingleThreadExecutor()`

D) `ExecutorService.buildSingleThreadExecutor()`

Explanation

To ensure the `CalcThread` instance will run in a single thread, you should insert the code fragment `Executors.newSingleThreadExecutor()` at line `02` as follows:

```
01  ExecutorService ex;
02  ex = Executors.newSingleThreadExecutor();
03  Thread th = new CalcThread();
04  ex.submit(th);
```

The `Executors` class contains factory and utility methods for `Executor`, `ExecutorService`, `ScheduledExecutorService`, `ThreadFactory`, and `Callable` objects. The `newSingleThreadExecutor` method returns an `ExecutorService` object for a single-threaded pool that supports a limitless queue of tasks for only one simultaneous thread at time.

The other code fragments are incorrect because the methods `getSingleThreadExecutor`, `buildSingleThreadExecutor`, and `createSingleThreadExecutor` are not provided in the `Executors` or `ExecutorService` interfaces.

Objective:
Concurrency

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

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Thread Pools

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

---

Question #36 of 50

Question ID: 1327958

Consider the following service interface implementation:

```
package package2;
import package1.SpeakerInterface;

public class SpeakerImplementer implements SpeakerInterface {
    @Override
    public void speak() {
        System.out.println("Speaking from SpeakerImplementer!");
    }
}
```

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

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

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

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

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

Explanation

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

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

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

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

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

Objective:
Java Platform Module System

Sub-Objective:

Declare, use, and expose modules, including the use of services

References:

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

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

---

Question #37 of 50

Question ID: 1327918

Given:

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

What is the result?

A) An exception is thrown at runtime.

B) `MID`

C) `2`

D) `1`

E) Compilation fails.

Explanation

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

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

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

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

Objective:

Java Object-Oriented Approach

Sub-Objective:

Create and use enumerations

References:

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

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

---

Question #38 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) `created`

B) `level`

C) `experience`

D) `numPlayers`

E) `name`

F) `player`

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 #39 of 50

Question ID: 1327939

Consider the following code:

```
import java.util.Set;
public class set_up {
    public static void main(String[] args){
    Set<String> test_set = Set.of("one", "two", "three", "three");
        for (String Str:test_set){
            System.out.println(Str);
        }
    }
}
```

What exception would be thrown on running this block of code?

A) `UnsupportedOperationException`

B) `IllegalArgumentsException`

C) `ArithmeticException`

D) `NullPointerException`

Explanation

An IllegalArgumentsException is thrown because the set being created includes duplicate elements. A key difference between `List` and `Set` is that a `List` allows duplicate entries but a `Set` does not. A `Map` has key value pairs, but key values are never duplicates. Additionally, a `List` is an ordered collection while a `Set` is not.

The `UnsupportedOperationException` is an incorrect option. This exception is thrown if an immutable `Set` was attempted to be modified *after* it was created. Factory methods in Java 9 and above for creating `Set`, `Map`, and `List` collections create immutable collections. An example of such a factory method is `Set.of()`.

`NullPointerException` is an incorrect option. This exception is thrown when a `List` or `Set` is created with null elements in it.

`ArithmeticException` is an incorrect option. This exception would be thrown when an illegal arithmetic operation is performed, for instance division by zero.

Java 9 and above includes a collection library that provides static factory methods for the interfaces `List`, `Set`, and `Map`. You can use these methods for creating immutable instances of collections. An immutable collection is a collection that cannot be modified after it is created. This includes the references made by the collections, their order, as well as the number of elements. Attempting to modify an immutable collection results in a `java.lang,UnsupportedOperationException` being thrown.

You can use the `Set.of()` static factory method for creating an immutable set with no null elements. If the elements in the set are serializable, then the set becomes serializable as well. It also has no duplicate elements.

Objective:
Working with Arrays and Collections

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

References:

Oracle Technology Network > Java > Oracle JDK9 Documentation > Java Platform, Standard Edition Core Libraries > Creating Immutable Lists, Sets, and Maps

---

Question #40 of 50

Question ID: 1328156

Given the following code fragment:

```
01  ScheduledExecutorService ex;
02  ex = _____________________________;
03  Callable c = new CalcThread();
04  ex._____________________________;
```

Which code fragments, when inserted independently at lines `02` and `04`, will ensure the `CalcThread` instance will run after at least a ten-second delay?

A) `scheduleAtFixedRate(c, 10,10, TimeUnit.SECONDS)`

B) `Executors.createScheduledThreadPool(1)`

C) `ExecutorService.getScheduledThreadPool(1)`

D) `ExecutorService.buildScheduledThreadPool(1)`

E) `Executors.newScheduledThreadPool(1)`

F) `schedule(c, 10, TimeUnit.SECONDS)`

G) `scheduleWithFixedDelay(c, 10,10, TimeUnit.SECONDS)`

Explanation

To ensure the `CalcThread` instance will run after at least a ten-second delay, you should insert the code fragment `Executors.newScheduledThreadPool(1)` at line `02` and `schedule(c, 10, TimeUnit.SECONDS)` at line `04` as follows:

```
01  ScheduledExecutorService ex;
02  ex = Executors.newScheduledThreadPool(1);
03  Callable c = new CalcThread();
04  ex.schedule(c, 10, TimeUnit.SECONDS);
```

The `Executors` class contains factory and utility methods for `Executor`, `ExecutorService`, `ScheduledExecutorService`, `ThreadFactory`, and `Callable` objects. The `newScheduledThreadPool` method returns a `ScheduledExecutorService` object for the execution of tasks after a fixed delay or for periodic execution. In this code fragment, the thread pool is set at an initial capacity of one thread. The `schedule` method accepts either a `Callable` or `Runnable` object that executes once after a specified delay.

The other code fragments are incorrect because the methods `getScheduledThreadPool`, `buildScheduledThreadPool`, and `createScheduledThreadPool` are not provided in the `Executors` or `ExecutorService` interfaces.

The code fragments that specify `scheduleAtFixedRate` and `scheduleWithFixedDelay` are incorrect because these methods do not accept `Callable` objects. Both methods execute `Runnable` objects repeatedly at a fixed period or delay, respectively.

Objective:
Concurrency

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

References:

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Essential Classes > Concurrency > Thread Pools

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

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

---

Question #41 of 50

Question ID: 1327781

Given:

```
public class Java11 {
   public static void main (String[] args) {
     StringBuilder sb = new StringBuilder("DataDataDataDataData");
     sb.delete(0,sb.length());
     System.out.println(sb.capacity());
   }
}
```

What is the output when this program is executed?

A) `0`

B) `20`

C) `16`

D) `36`

Explanation

The output is `36`. Because additional characters beyond the initial capacity have not forced the `StringBuilder` object to resize, its capacity has not changed.

The capacity of a `StringBuilder` object is 16 if not specified explicitly in the constructor as an `int` argument. If a `String` object is specified in the constructor, then the initial capacity is 16 plus the length of the String object. In this case, the literal string `DataDataDataDataData` is 20 characters in length, so the capacity is 16 + 20 or 36.

The output is not `0`. Removing characters does not modify the capacity of a `StringBuilder` object. Because the `delete` method removes all characters in the underlying `String` object, the `length` method would return 0, not the `capacity` method.

The output is not `16` or `20`. The initial capacity is 36 and would not be less unless specified explicitly in the constructor.

Objective:
Working with Java Data Types

Sub-Objective:
Handle text using String and StringBuilder classes

References:

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

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

---

Question #42 of 50

Question ID: 1327834

Given:

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

class MainClass {
   public static void main(String[] args) {
     new StandardMethods().printPerimeter(5,5);
   }
}
```

What is the result?

A) Compilation fails.

B)
```
Constructor invoked!
Perimeter is 10.0
```

C)
```
Perimeter is 10.0
```

D)
```
Constructor invoked!
```

E) An exception is thrown at runtime.

Explanation

Compilation fails because the statement `new StandardMethods().printPerimeter(5,5);` attempts to access the `private` constructor of `StandardMethods`. If a class has a constructor declared as private, then that class cannot be directly instantiated. Private members are only available within the same class.

The result is not output, because compilation fails. The following output would be the result if the `private` keyword were removed:

```
Constructor invoked!
Perimeter is 10.0
```

The following output would be the result if the constructor were removed:

```
Perimeter is 10.0
```

An exception is not thrown at runtime because compilation fails before reaching the Java runtime.

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 #43 of 50

Question ID: 1328202

Consider the following code that works with daylight savings time:

```
int thisYear = ZonedDateTime.now().getYear();
int thisMonth = ZonedDateTime.now().getMonth().getValue();
```

Which of the following code snippets correctly adds a week to the date information for the month?

A) `ZonedDateTime tzTime = ZonedDateTime.now().plusWeek(1);`

B)
```
ZonedDateTime tzTime =
ZonedDateTime.now().plus(Period.ofDays(7));
```

C) `ZonedDateTime tzTime = ZonedDateTime.now().plusMonth(0.25);`

D) `ZonedDateTime tzTime = ZonedDateTime.now().plusDays(7);`

Explanation

You should use the code as shown below:

```
ZonedDateTime tzTime = ZonedDateTime.now().plus(Period.ofDays(7));
```

This code uses the `ZonedDateTime` class add seven days to the current date/time in the current time zone. The `ZonedDateTime` class includes a date and time in a specific time zone and can support daylight savings changes. It works in correlation with the `java.util.Calendar` class.

The other options will not work in this scenario because only the `ZonedDateTime` class with the `Period` class can correctly modify time data when dealing with daylight savings time.

Objective:
Localization

Sub-Objective:

Implement Localization using Locale, resource bundles, and Java APIs to parse and format messages, dates, and numbers

References:

Oracle Technology Network > Java SE > Java SE Documentation > Java Platform Standard Edition 11 API Specification > java.util.timezone > Class TimeZone

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

---

Question #44 of 50

Question ID: 1327904

Given the following interfaces:

```
interface A1 {
  default void doStuff() {
    System.out.println("A1.doStuff()");
  }
}

interface A2 extends A1 {}

interface B1 {
  default void doStuff() {
    System.out.println("B1.doStuff()");
  }
}

class AlphabetSoup implements A2, B1 {
  // line n1
}
```

Which code fragment should be added at line `n1` to allow compilation to succeed?

A)
```
public abstract void doStuff();
```

B)
```
public void doStuff() {
    this.A2.doStuff();
  }
```

C)
```
public void doStuff() {
    A2.super.doStuff();
  }
```

D)
```
void doStuff() {
  A2.super.doStuff();
}
```

E)
```
public void doStuff() {
  A2.doStuff();
}
```

Explanation

The following code fragment should be added:

```
public void doStuff() {
  A2.super.doStuff();
}
```

When multiple conflicting implementations are inherited as shown in this question, a method must be explicitly coded to resolve the conflict. That implementation may either use existing implementations or code the behavior from scratch, as the programmer chooses. To refer to one of the default methods defined in the implemented interfaces, the syntax is `<Interface-Name>.super.<method-name>`.

`public abstract void doStuff();` fails to compile because you are not permitted to define an abstract method in a concrete class.

The code fragment with the method body `A2.doStuff()` fails to compile because `doStuff` is not a static method in the interface `A2`.

This code fragment fails because it attempts to define a default access method:

```
void doStuff() {
  A2.super.doStuff();
}
```

Because the interface method that it attempts to override is public (because all interface methods are public), it does not compile.

The code fragment with the method body this.`A2.doStuff()` fails to compile because `A2` is not a member of this.

Objective:
Java Object-Oriented Approach

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

References:

The Java Tutorials > Learning the Java Language > Interfaces and Inheritance

Question #45 of 50

Question ID: 1328061

Given the following code fragment:

```
public static void decryptFile(SecretKey key) throws Exception {
  String fileName = "encrypted.dat";
  Cipher calg = Cipher.getInstance("DES");
  calg.init(Cipher.DECRYPT_MODE, key);
  try (
    FileReader inputFS = new FileReader(fileName);
    CipherOutputStream decStream = new CipherOutputStream(System.out, calg);
  ){
    int charByte;
    while ((charByte = inputFS.read()) != -1)
    decStream.write(charByte);
    decStream.flush();
  }
}
```

Assume that the file encrypted.dat does not exist. What is the result of compiling and executing this code?

A) A `FileNotFoundException` is thrown at runtime.

B) The `FileReader` object remains open.

C) An `IOException` is thrown at runtime.

D) The `CipherOutputStream` object remains open.

Explanation

Assuming that the file encrypted.dat does not exist, this code will throw a `FileNotFoundException` at runtime. The `FileReader` constructor throws a `FileNotFoundException` if the specified file does not exist. Because execution does not move past the instantiation in the try-with-resources statement, this exception is not suppressed. If an exception is thrown from the try block, then exception(s) thrown by a try-with-resources statement are suppressed.

An `IOException` is not thrown at runtime. Because neither resource is instantiated, an `IOException` would not be thrown by implicitly closing resources.

The `CipherOutputStream` object does not remain open. The exception is thrown before this resource is instantiated.

The `FileReader` object does not remain open. The exception is thrown before this resource is instantiated.

Objective:
Exception Handling

Sub-Objective:

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

References:

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

---

Question #46 of 50

Question ID: 1327906

You define the following interface to handle photographic items:

```
public interface Photographer {
  Photograph makePhotograph(Scene s);
  /* insert here */ {
    List<Photograph> result = new ArrayList<>();
    for (Scene s : scenes) {
      result.add(makePhotograph(s));
    }
    return result;
  }
}
```

You need to define the `makePhotographs` method to support making multiple photographs with any `Photographer` object. This behavior should be modifiable by specific implementations.

Which code should be inserted at the point marked `/* insert here */` to declare the `makePhotographs` method?

A) `public List<Photograph> makePhotographs(Scene ... scenes)`

B) `static List<Photograph> makePhotographs(Scene ... scenes)`

C) `default List<Photograph> makePhotographs(Scene ... scenes)`

D) `List<Photograph> makePhotographs(Scene ... scenes);`

E) `List<Photograph> makePhotographs(Scene ... scenes)`

Explanation

You should declare the `makePhotographs` method as follows:

```
default List<Photograph> makePhotographs(Scene ... scenes)
```

A default method creates an instance method with an implementation. The implementation can be overridden in specializing types, which meets the scenario requirements. Interfaces can define method implementations only in two conditions: either the method is static, or the method is default. Otherwise, abstract methods may be declared, but implementation must be deferred to a specialized type.

You should not use the declaration `List<Photograph> makePhotographs(Scene ... scenes)`. Any method declared in an interface that is neither static nor default will be treated as an abstract method and must not have a body. This method declaration will cause the code to fail compilation because it attempts to declare an abstract-by-default method, and yet tries to provide a method body.

`List<Photograph> makePhotographs(Scene ... scenes);` will cause a compilation failure because the method body block immediately follows the semicolon (`;`) that terminates what is essentially a valid abstract method declaration.

`public List<Photograph> makePhotographs(Scene ... scenes)` is incorrect because it will be treated as an abstract method, and cannot have a body. Interface methods are public whether or not they are declared as such. This method declaration is effectively identical to `List<Photograph> makePhotographs(Scene ... scenes)`.

A static method declaration would not compile because in a static, there is no current context reference this. Without that context, the call to `makePhotographs` inside the statement `result.add(makePhotograph(s));` cannot be invoked. This method declaration would cause the code to fail compilation.

Objective:
Java Object-Oriented Approach

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

References:

The Java Tutorials > Learning the Java Language > Interfaces and Inheritance > Default Methods

---

Question #47 of 50

Question ID: 1328191

Which of the following lines are valid in a `.properties` file in Java? (Choose all that apply.)

A) `ResourceBundle.getBundle();`

B) `label1 = Label 1 Thank you for using this program`

C) `label1 = Label 1 Gracias por usar este programa!`

D) `label1`

Explanation

The following name value pairs comprise valid `.properties` file values in a Java application:

```
label1 = Label 1 Gracias por usar este programa!
```

```
label1 = Label 1 Thank you for using this program
```

The options `label1` and `ResourceBundle.getBundle();` are both incorrect as they do not confirm to the name-value pair format needed in a Java `.properties` file.

Objective:
Localization

Sub-Objective:
Implement Localization using Locale, resource bundles, and Java APIs to parse and format messages, dates, and numbers

References:

Oracle Technology Network > Java SE > Java Tutorials > Internationalization

Oracle Technology Network > Java SE > Java SE Documentation > The Java Tutorials > Learning the Java Language > Essential Classes > The Platform Environment > Properties

---

Question #48 of 50

Question ID: 1327813

Given the following code line:

```
printToConsole("Cup of Java with a shot of espresso");
```

Which overloaded method is invoked?

A)
```
void printToConsole(StringBuffer buffer) {
    System.out.println(buffer.toString());
}
```

B)
```
void printToConsole() {
    System.out.println("Hello, world!");
}
```

C)
```
void printToConsole(Integer intObj) {
    System.out.println("My number is " + intObj.toString());
}
```

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

Explanation

The following overloaded method is invoked:

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

```
}
```

This method is invoked because there is no overloaded method that accepts a `String` argument, and any argument other than a `StringBuffer` or `Integer` will invoke this method. All objects inherit from the `Object` class, so this overloaded method is an effective catch-all for an argument whose data type is not explicitly declared in another overloaded method. When a method is invoked on an overloaded method, only the list of parameter data types determines which method is executed.

The overloaded method with no parameters is not invoked. The invocation specifies a `String` argument, so this overloaded method will not be executed. This overloaded method would be executed if the invocation contained no argument.

The overloaded method with a `StringBuffer` parameter is not invoked. Although the invocation specifies a `String` argument, `String` arguments are not automatically converted into `StringBuffer` objects because these classes are not related with inheritance. This overloaded method would be executed if the invocation contained a `StringBuffer` argument.

The overloaded method with an `Integer` parameter is not invoked. `String` arguments are not converted into `Integer` objects because these classes are not related with inheritance. This overloaded method would be executed if the invocation contains an `Integer` argument.

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 #49 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) `CharIndexOutOfBoundsException`

B) `StringIndexOutOfBoundsException`

C) `BadStringOperationException`

D) `BadIndexBoundsException`

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 #50 of 50

Question ID: 1327902

Which statement is true about interfaces?

A) Interfaces do not support overloaded and overridden methods.

B) Interfaces only support public members.

C) Interfaces may contain only abstract, default, or static methods and class constants.

D) Interfaces can implement methods defined by other interfaces.

E) Interfaces support class instantiation, just as concrete classes do.

Explanation

Interfaces 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 class-wide methods.

You use interfaces when the classes that use the interface are completely unrelated in their implementation and possibly developed by different parties.

An interface cannot implement methods defined by other interfaces. Although an interface can extend another interface, interfaces cannot contain implementation for methods.

Interfaces do not support class instantiation like concrete classes do. Only concrete classes that implement the methods of an interface support class instantiation.

Interfaces do support overloaded and overridden methods. An interface can overload its own methods or extend another interface and overload or override its methods.

Interfaces do not only support public members. Since Java 9, interfaces also support private methods for code reuse between default methods without providing visibility to implementing classes.

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