Review Test Submission: Quiz 3: Java syntax, standard library, Android concepts & techniques

User	Khasiano Webb
Course	WTC-DeepDiveJava_Android-57588-nbennett-Sept2019
Test	Quiz 3: Java syntax, standard library, Android concepts & techniques
Started	10/24/19 11:58 PM
Submitted	10/25/19 12:01 AM
Status	Completed
Attempt Score	150 out of 150 points
Time Elapsed	2 minutes
Results Displayed	Submitted Answers, Feedback, Incorrectly Answered Questions

Question 1

10 out of 10 points

What will be the result if no constraints are specified for the children of a `ConstraintLayout`?

Selected Answer: All of the children will be aligned to the top-left of the ConstraintLayout, stacked on top of each other.

Response Feedback: The children of a `ConstraintLayout` must include constraint attributes that reference the parent or other children; otherwise, they will all be stacked on top of each other in the upper-left of the ConstraintLayout region.

Question 2

10 out of 10 points

An instance of **`java.io.InputStreamReader`** may be used to …

Selected Answer: … read **`char`** values (and **`char`** arrays) from a byte-oriented stream.

Response Feedback: An **`InputStreamReader`** is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified (or the default) charset.

Question 3

10 out of 10 points

← OK

What output is produced by the following code, when compiled and run as a Java application?

```
public class Test {

  private static int var1 = getValue();

  static {
    System.out.println("static initializer");
  }

  private static int var2 = getValue();

  private static int getValue() {
    System.out.println("getValue");
    return 1;
  }

  public static void main(String args[]) {
    System.out.println("main");
  }
}
```

Selected Answer:
```
getValue
static initializer
getValue
main
```

Response Feedback: When a class is loaded as a Java application, static initialization is performed before the **`main`** method is invoked. Static initialization consists of executing all static field initialization and **`static`** block statements, in order, from top to bottom. This initialization can include method calls - including calls to static methods in the same class.

Question 4

10 out of 10 points

What is the defining feature of MutableLiveData, vs other LiveData instances?

Selected Answer: We can change the contained value from outside the class

Question 5

10 out of 10 points

Which of the following functional interfaces is most closely associated with writing concurrent (i. e. multi-threaded) Java code?

Selected Answer: **`java.lang.Runnable`**

Response Feedback: We can think of a **`java.lang.Runnable`** instance as a task that can be assigned to run on an existing thread or a new thread. We usually implement that interface, and create instances of the implementations, when writing code that will run on multiple threads – including Android, Swing, and JavaFX applications, most of which use one thread for UI interaction, and additional threads for long-running, non-UI tasks.

Question 6

10 out of 10 points

The options that appear in a navigator drawer, bottom navigation view, or options menu are all typically (but not necessarily) inflated from what kind of resource?

Selected Answer: A **`menu`** resource, composed and stored as an XML document.

Response Feedback: The items shown in several different types of Android menus—including navigator drawers, option menus, and bottom navigation views—are most often inflated from **`menu`** resources.

Question 7

10 out of 10 points

What is the role of an object-relational mapping (ORM) library?

Selected Answer: Provide *persistence* (i.e. long-term storage and access in a data store, such as an RDBMS) for instances of one or more classes.

Response Feedback: An ORM library is used to provide object persistence, mapping state and identity information of object instances to a lower-level data-access layer. Typically, the ORM does not itself provide those lower-level data access services, nor act as the underlying RDBMS.

Question 8

5 out of 5 points

If we attempt to commit changes to a local repository, and are told by Git that there are no changes to commit, we may reasonably conclude that the copy of the repository hosted on GitHub is also up-to-date.

Selected Answer: False

Response Feedback: The act of committing finalizes and logs a staged set of changes to a repository. However, if that repository has remotes (on GitHub or elsewhere), committing does not push those changes to the remotes. Similarly, commits directly to a remote repository do not automatically cause the changes to be pulled to a local repository. Many Git client tools (such as those in IDEs such as Eclipse and IntelliJ IDEA) let us specify that a commit should be immediately followed by a push, but these are still separate operations.

In other words, a local repository with no staged changes to commit may have committed changes that have not yet been pushed to one or more remotes, and a remote repository with no staged changes to commit may have committed changes that have not yet been pulled to one or more local repositories.

Question 9

10 out of 10 points

Match each of the letter casing style listed below with the _most common_ use(s) of that style in Java programming.

Question	Selected Match
`UPPER_SNAKE_CASE`	D. **`static final`** field names
`UpperCamelCase`	B. Class and interface names
`lowerCamelCase`	A. names of methods, non-**`static`** fields, local variables, method parameters, lambda parameters
`lowercase`	C. package names, Java keywords

Question 10

5 out of 5 points

The Room ORM library connects to an underlying database using Java Database Connectivity (JDBC), and can thus be used to provide object-relational mapping to any database accessible to Java.

Selected Answer: False

Response Feedback: JDBC is a Java database access library supporting platform-independent connection to a wide variety of RDBMSs, regardless of the programming languages used to build those RDBMSs. Room, however, connects only to SQLite, only on the Android platform, and does so without the use of JDBC.

Question 11

10 out of 10 points

Which of the following is a valid variable declaration and assignment?

Selected Answer: **`char c = 97;`**

Response Feedback: A `char` is actually an integer-type primitive variable, holding a 2-byte unsigned value (the only unsigned integer intrinsic type in Java, in fact). As with other primitive types, many standard library methods (e.g. **`java.io.PrintStream.println`**) have **`char`**-specific overloads which convert **`char`** values to strings differently than other integer types are converted, but **`char`** is nonetheless an integer type.

A **`char`** literal can be expressed as a numeric value (in base-2, base-8, base-10, or base-16 representation, depending on the prefix) in the range 0 to 65535, or as a single character delimited by single quotes. In the latter case, the value may be expressed as a simple character (e.g. **`'a'`**) or as an escaped Unicode point (**`'\u0061'`**).

Question 12

10 out of 10 points

Which of the following classes should be extended to provide **`View`** instances to a **`RecyclerView`**, for display in a scrolling list?

Selected Answer: **`androidx.recyclerview.widget.RecyclerView.Adapter`**

Response Feedback: A **`RecyclerView.Adapter`** acts as a bridge between a **`RecyclerView`** and the underlying data for that view. The **`Adapter`** is responsible for constructing an individual **`View`** instance for each element in the data array or collection, and providing those instances to the **`RecyclerView`**, for the latter to display in a list or tabular format.

Question 13

15 out of 15 points

Which of the following are functional interfaces?

Selected Answers: **`java.util.function.Predicate`**
`java.util.Map`
`java.util.Comparator`
`java.util.List`
`java.io.File`

Response Feedback: An interface is a functional interface if it has exactly one unimplemented method. Any method with the modifier **`static`** or **`default`** is implemented in the interface itself, and is thus not unimplemented. Also, if an interface has any methods that are not implemented in the interface itself, but *are* implemented in **`java.lang.Object`**, such methods are not considered unimplemented, since all classes extend (directly or indirectly) **`Object`**, and thus will automatically have implementations of all its methods.

Question 14

10 out of 10 points

A given method is declared with the signature

```
void doSomething(Class c)
```

(**`Class`** refers to **`java.lang.Class`**, which – like all classes in **`java.lang`** – is imported automatically.) Which of the following would be accepted by the compiler as a valid invocation of the **doSomething** method?

Selected Answer: **`doSomething(Integer.class)`**

Response Feedback: A Java class is a type, not a value that can be declared as a parameter to a method, or passed as an argument in a method invocation. However, every Java class has a static field called **`class`**, which is an instance of the **`java.lang.Class`** object; thus we can use **`Class`** itself as the *type* of a parameter, and pass the **`class`** field of a class (or the result returned by a **`getClass()`** method call on an object instance) as an argument value for such a parameter.

15 out of 15 points

Question 15

Which of the following statements about the **`java.lang.Comparable<T>`** and **`java.util.Comparator<T>`** interfaces are true?

Selected Answers:

An instance of a class that implements **`Comparator<T>`** is able to compare two instances of the generic **`T`** class to each other.

`Comparable<T>` is only used for shuffling items in a **`Collection<T>`** or **`T[]`** array.

There is no practical difference between **`Comparable<T>`** and **`Comparator<T>`**.

`Comparator` is only used for comparing **`String`** values.

An instance of a class **`T`** that implements **`Comparable<T>`** is able to compare itself to any other instance of the class **`T`** (or a subclass of **`T`**).

Response Feedback:

`Comparable<T>` and **`Comparator<T>`** are both useful for any comparison-based sorting/ordering. However, **`Comparable<T>`** i s the interface implemented to allow an instance of class **`T`** to compare itself to another instance of the same class (or a subclass), while **`Comparator<T>`** is implemented to allow a comparator object to compare two instances of type **`T`** to each other.

Friday, October 25, 2019 12:01:08 AM MDT