Jake Taggart
Weekend 5 Research

Research

1. What is the difference in lateinit and lazy initialization?

   First of all, lazy initialization applies to val while lateinit applies to vars. If the developer is not exactly sure what the value of the variable will be when it's eventually initialized and could be subject to change, they should just use lateinit var. If we, however, know that the value will be set exactly once and remain final - such as in a Singleton pattern - lazy initialization will work perfectly. So if the value will be mutable, we want to use lateinit. If the value is going to be immutable, lazy works well.

2. What are the advantages of Kotlin over Java?

   First of all, it's far more concise in terms of code length. Some features that make this possible are the uses of data classes, smart casting, object properties, and type interfacing. Because of its concision, Kotlin is therefore safer from crashes due to the minimizing of possible developer mistakes over shorter volumes of code. Because of its inferred nature, it's also easier to maintain. Kotlin also notices error at compile time when possible. Finally, the minimization of null pointer exceptions is also huge.

3. What are checked and unchecked exceptions? Which is not supported by Kotlin?

   Checked exceptions are those which thrown at compile time. Unchecked exceptions are thrown at runtime. There are four types of unchecked exceptions: NullPointerException, ArrayIndexOutOfBonds, IllegalArgumentException, and IllegalStateException. Six checked exceptions include IOException, SQLException, DataAccessException, ClassNotFoundException, InvocationTargetException, and MalformedURLException. So far, Kotlin does not support checked exceptions.

4. What is MVP?

   MVP is an architectural design pattern used in Android development. Prior to MVP, the MVC (Model-View-Controller) pattern was used. MVP greatly improved upon that for Android intentions. It's used to separate concerns and improve maintainability. MVP stands for Model-View-Presenter. The Model layer is used for any data resources such as databasing, POJO's, defining REST calls, etc. The View Layer is for any UI element. The

Presenter layer acts upon the Model and the View by getting necessary data from the Model and presenting it in the view and getting any Input action from the user that may affect the Model and then acts upon the model accordingly. The Presenter has a one to one relationship with the View and needs to communicate with a contract which uses interfaces. For every View-Presenter relationship, there should be a contract interface that is used to communicate between the two.

5. What is MVVM?

   MVVM is an architectural design pattern used in Android development. Prior to MVVM, there was MVP and MVC. Both have been improved upon with MVVM. MVVM stands for Model View View-Model. Instead of using a presenter or controller, View-Model uses a way of binding the View and Model-View which automates the communication between the two. MVP's presenter has a reference to the view, but MVVM's View-Model does not. In MVVM, the view directly binds to the model-view. This is accomplished with LiveData. View and View-Model have a one to many relationship.