Jamarcus Coulter

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

Lateinit is used with var to assign a value to the var within the code and not upfront. Lazy is used with val and creates a lambda function to assign a value to the val. The lambda function is only called once and any sequential call to use the val will return value assigned to the val.

2. What are the advantages of Kotlin over Java?

Kotlin tries to eliminate the nullpointerexception that is quite common when coding in Java. Kotlin reduces the amount of boilerplate code that is necessary in Java. Kotlin does not require variable data type to be defined when declared and allows the compiler to gracefully assign the data type at compile time. Kotlin also allows for easy late initialization of variables. Kotlin also allows functions to be assigned to a variable and allows inline functions. Due to both using the JVM, Java code can be easily using within Kotlin code but not vice versa.

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

Checked exceptions are exceptions that are checked at compile time and a method with these exceptions must be coded to handle the exception or use the throws keyword for the exception.

Unchecked exceptions are exceptions not checked at compile time. In Java, these are errors or runtime exceptions. It is up to the developer to fix or catch these exceptions.

Kotlin does not support checked exceptions.

4. What is Model View Presenter?

Model View Presenter (MVP) is an architecture pattern to separate Views from business logic. The Presenter is a middle man in MVP. The View(UI) is decoupled from the Model(logic) by the way of the Presenter. The View passes user interaction data, things like inputs and requests, to the Presenter. The Presenter then accesses the Model with the passed data from the UI and fetches the proper data from the Model’s logic and updates the View with the fetched data, which might be displaying text or even opening another view. The Presenter implements methods from both the Model and the View to facilitate the communication with both. Unlike in MVC where the View interacts directly with the logic, MVP makes testing and reuse of code easier because the View or Model logic can tested independently due to the separation by the Presenter.

5. What is Model View ViewModel?

Model View ViewModel (MVVM) is an architecture pattern which separates the View (UI) from data and logic. The Model holds the data of the application. It does not interact with the View and generally the ViewModel will subscribe/observers to the Model to receive data and data changes. The View holds the UI elements of the application and can subscribe/observe to one or many different ViewModels to update the UI elements. The ViewModel is where the logic of the application is held. It sends a data stream for the View to listen to for updates and/or use callback methods. The ViewModel sends data to the Model and observes the Model or retrieves data from the Model to make updates to the View.