MVVM ARCHITECTURE

Understanding MVVM

Model:
Represents data and business logic.

Example:

```
data class Book(val title: String, val author: String)
```

ViewModel:
Acts as a bridge between Model and View.

Example:

```
class BookViewModel {
fun getBooks(): List<Book> {
return listOf(Book("Title1", "Author1"), Book
("Title2", "Author2"))
}
}
```

View:
User interface that displays data.

Example:

```
class BookActivity : AppCompatActivity() {
private lateinit var viewModel: BookViewModel
override fun onCreate(savedInstanceState: Bundle?)
{
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_book)
viewModel = BookViewModel()
val books = viewModel.getBooks()
// Display books in UI
}
}
```

Introduction to Inheritance

Basic Concepts:

Base Class:

```
open class Vehicle {
fun start() {
println("The vehicle is starting.")
}
}
```

Derived Class:

```
class Car : Vehicle() {
fun drive() {
println("The car is driving.")
}
}
```

Usage:

```
val myCar = Car()
myCar.start() // Output: The vehicle is starting.
myCar.drive() // Output: The car is driving.
```

ViewModel Class

Initialization and Usage:

```
class MyViewModel : ViewModel() {
var number: Int = 0
fun incrementNumber() {
number++
}
}
class MainActivity : AppCompatActivity() {
private val myViewModel: MyViewModel by viewModels()
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
myViewModel.incrementNumber()
}
}
```

Open Function in Kotlin

Example:

```
open class Vehicle {
open fun start() {
println("The vehicle is starting.")
}
}
class Car : Vehicle() {
override fun start() {
println("The car is starting with a
roar!")
}
}
val myCar = Car()
myCar.start() // Output: The car is
starting with a roar!
```

Override Function in Kotlin

Example Without super:

```
open class Vehicle {
open fun start() {
println("The vehicle is starting.")
}
}
class Car : Vehicle() {
override fun start() {
println("The car is starting with a roar!")
}
}
```

Example With super:

```
class SportCar : Vehicle() {
override fun start() {
super.start()
println("The sports car is ready to zoom!")
}
}
val sportCar = SportCar()
sportCar.start()
// Output:
// The vehicle is starting.
// The sports car is ready to zoom!
```

Understanding Interfaces

Defining and Implementing:

```
interface Drivable {
fun drive()
}
class Car : Drivable {
override fun drive() {
println("The car is driving.")
}
}
val myCar = Car()
myCar.drive() // Output: The car is
driving.
```

Default Implementation:

```
interface Drivable {
fun drive() {
println("Driving the vehicle.")
}
}
class Bicycle : Drivable
val myBicycle = Bicycle()
myBicycle.drive() // Output: Driving
the vehicle.
```

Repositories in Android Development

Example:

```
class UserRepository(private val
userDao: UserDao, private
val userService: UserService) {
fun getUser(userId: String):
LiveData<User> {
// Fetch data logic
}
}
class UserViewModel(private val
repository: UserRepositor
y) : ViewModel() {
fun getUser(userId: String):
LiveData<User> {
return repository.getUser(userId)
}
}
```

Repositories in Android Development

Definition: APIs are sets of rules that allow software applications to interact with each other.

Analogy: Think of an API as a restaurant menu. The menu (API) offers a list of dishes (services) you can order (request). The kitchen (system) prepares and serves the dish (response).