How to achieve abstraction in android Kotlin application?

Abstraction in an Android Kotlin application is typically achieved through interfaces, abstract classes, and clean architecture layers. The goal is to hide implementation details and expose only the necessary behaviors, making your code more modular, testable, and maintainable.

Here are several key ways to achieve abstraction in an Android Kotlin app:

□

1. Use Interfaces

Interfaces define contracts without implementation. You can program against interfaces instead of concrete implementations.

Example:

```
interface UserRepository {
  suspend fun getUser(userId: String): User
}
```

Then provide an implementation:

```
class UserRepositoryImpl(private val api: UserApi) :
UserRepository {
  override suspend fun getUser(userId: String): User {
    return api.fetchUser(userId)
  }
}
```

2. Use Abstract Classes

Use abstract classes when you want to provide some shared logic but leave specific parts for subclasses to implement.

```
abstract class BaseViewModel : ViewModel() {
  abstract fun fetchData()
}
```

3. Dependency Inversion (Clean Architecture)

High-level modules (e.g., ViewModel) should depend on abstractions, not concrete classes.

ViewModel depending on abstraction:

```
class UserViewModel(private val userRepository: UserRepository)
: ViewModel() {
  fun loadUser(id: String) {
    // logic using userRepository
  }
}
```

Use Hilt or Koin for DI to provide the implementation:

```
@Module
@InstallIn(SingletonComponent::class)
object RepositoryModule {
  @Provides
  fun provideUserRepository(api: UserApi): UserRepository =
```

```
UserRepositoryImpl(api)
}
```

4. Abstracting UI logic

Use sealed classes or state classes to abstract UI state:

```
sealed class UserUiState {
  object Loading : UserUiState()
  data class Success(val user: User) : UserUiState()
  data class Error(val message: String) : UserUiState()
}
```