DEVGRID

Clean Architecture to Android Development

Applying SOLID concepts

DG Conference Week

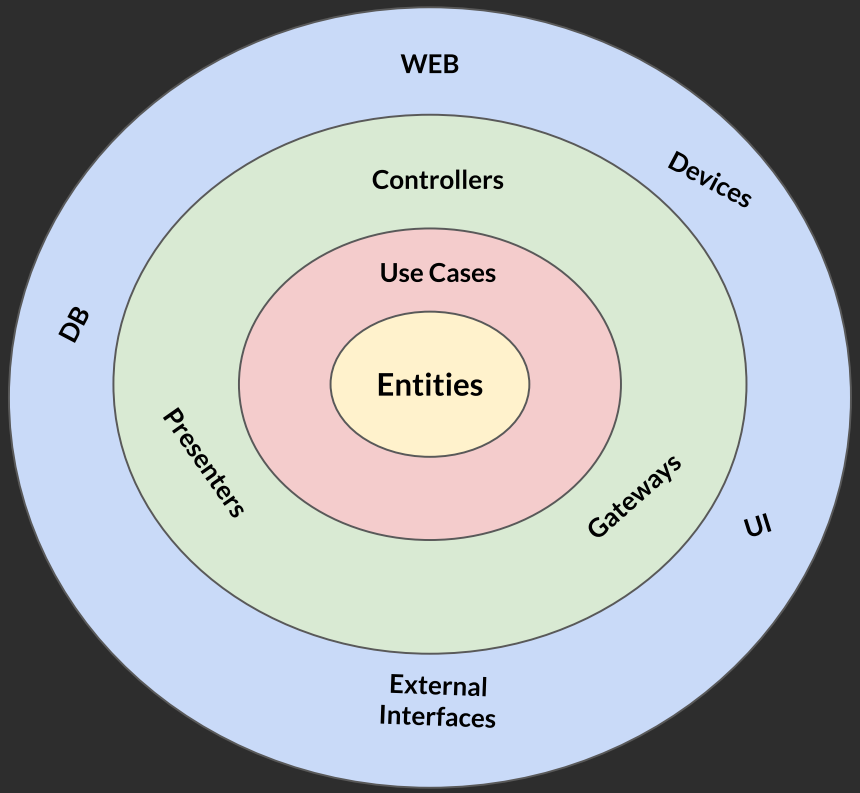
Created by *Gabriel B. Moro* - *moro@devgrid.co.uk*

Summary

- Overview about the *Onion Approach*
- Dependency Rule
- Abstraction Concept
- Cake Recipe to Android
- SOLID Principles
- Final Consideration

DEVGRID

Onion Approach




DEVGRID

DEVGRID

Dependency between **UI** and **Presentation** class

```
class LoginActivity : Activity() {

  lateinit var viewModel : LoginViewModel
  lateinit var tvUserName : EditText
  lateinit var tvPassword : PasswordText
  lateinit var btnLogin : Button

  override fun onCreate() {
        // binding

    // listeners
    btnLogin.setOnClickListener {
        viewModel.onLoginClickEvent(
        userName = tvUserName.text,
        password= tvPassword.text
      )
```

DEVGRID

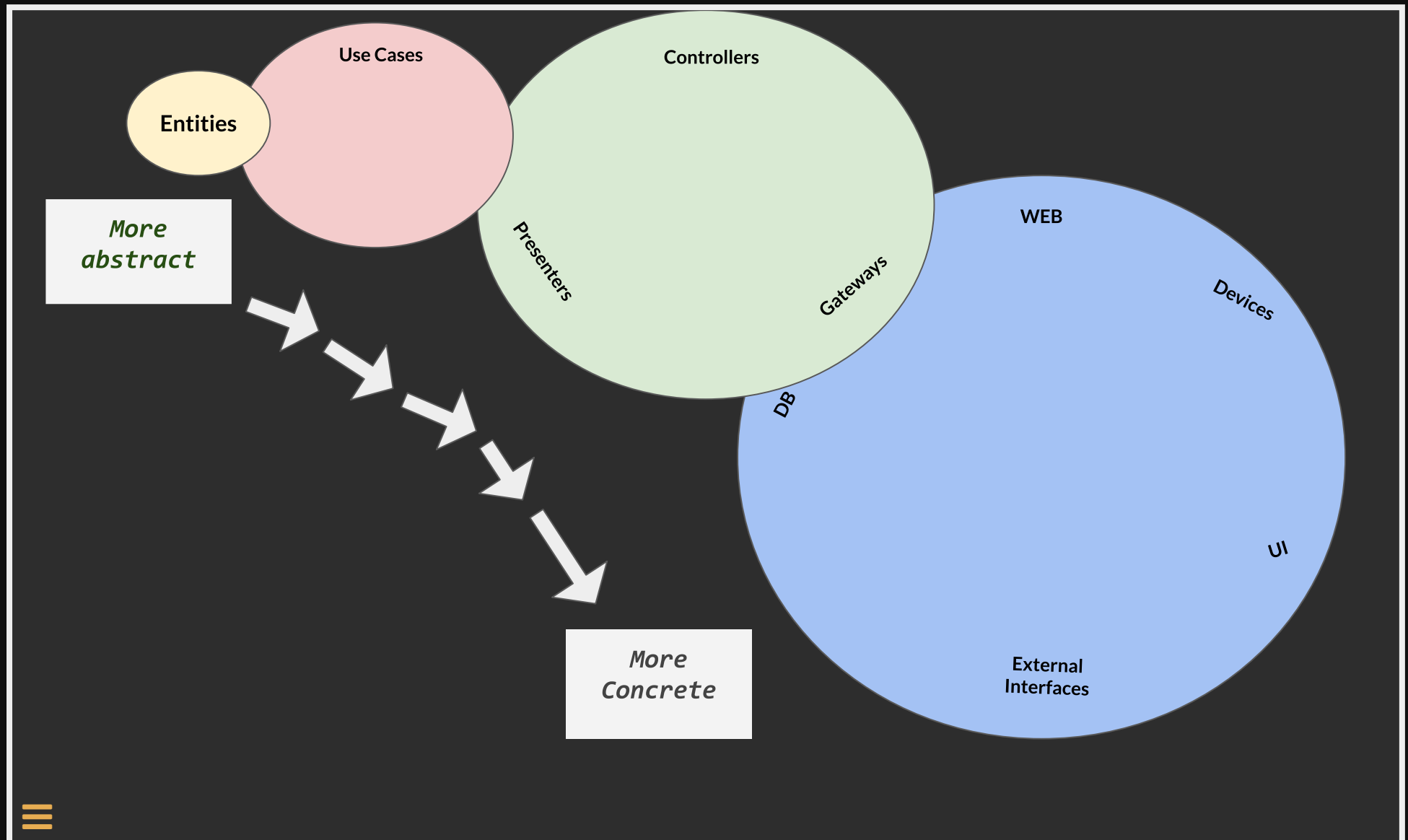
Dependency between **Presentation** and **Use Case** class

```
class LoginViewModel : ViewModel() {

  val userUseCase by inject<UserUseCase>()

  val onLoginSuccess = LiveData<Boolean>()

  fun onLoginClickEvent(userName : String,password : Password) {

    val encryptedPassword = password.toEncryptedPassword()

    userUseCase.login(
      userName = userName, encryptedPassword = encryptedPassword,
      success = { onLoginSuccess.postValue(true) },
      error = { onLoginSuccess.postValue(false) }
    )
  }
```

Dependency between **Use case** and **Entity** class

```
class UserUseCase {

    fun login(
        userName : String,
        encryptedPassword : String,
        success : (()->Unit),
        error : (()->Unit)
    ){
        // call for server or some repository
    }
}
```

Abstraction Concept




DEVGRID

Is there a *“cake recipe”* to implement the clean architecture to develop Android Apps?

Is there a *“cake recipe”* to implement the clean architecture to develop Android Apps?

In my opinion **no**, but we can define a good template ;)

DEVGRID

SOLID Principles

Single responsability

Open-closed

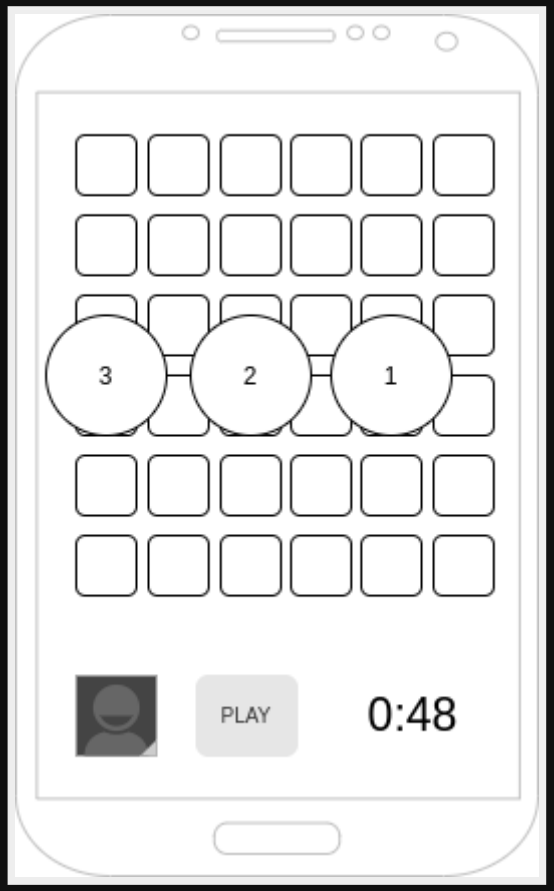
Liscov Substitution

Interface Segregation

Dependency Inversion

DEVGRID

Single Responsability



Open-closed

```
class LoginActivity : Activity() {

  override fun onCreate() { myCode }
}
```

```
class LoginActivity : Activity() {

  override fun onCreate() {
    super()
    myCode
  }
}
```

DEVGRID

Liscov Substitution

Look these two classes:

```
class Dog(val name : String) {

  fun walk() = println(“Walking…”)

  fun sleep() = println(“Sleeping …”)

}
```

```
class Pug(name : String) : Dog(name) {

  override fun walk() {
    super()
    sleep()
  }

}
```

DEVGRID

Liscov Substitution

```
class DogWalker {
  fun walk(dog : Dog, durationInMinutes:Int){
    object: CountDownTimer(durationInMinutes * 1000L, 200L) {
      override fun onTick(millisUntilFinished: Long) {
        dog.walk()
      }

    override fun onFinish() {
        dog.sleep()
      }
    }.start()
  }
}
```

```
fun main() {
  val dogWalker = DogWalker()
  dogWalker.walk(Dog(“Fred”), 500L)
}
```

Interface Segregation

```
interface RecyclerViewListeners {
  fun onClick(v : View, position : Int)
  fun onLongPress(v : View, position : Int, timePressed : Int)
  fun onSingleTap(v : View)
  fun onSingleDown(v : View)
}
```

```
class RecyclerViewOnlyBasicEvents : RecyclerView(), RecyclerViewL
  override fun onClick(v : View, position : Int){
    // Has implementation
  }
  override fun onLongPress(v : View, position : Int, timePressed
    // Has implementation
  }
  override fun onSingleTap(v : View) {
    // Not necessary
  }
  override fun onSingleDown(v : View) {
    // Not necessary
  }
}
```

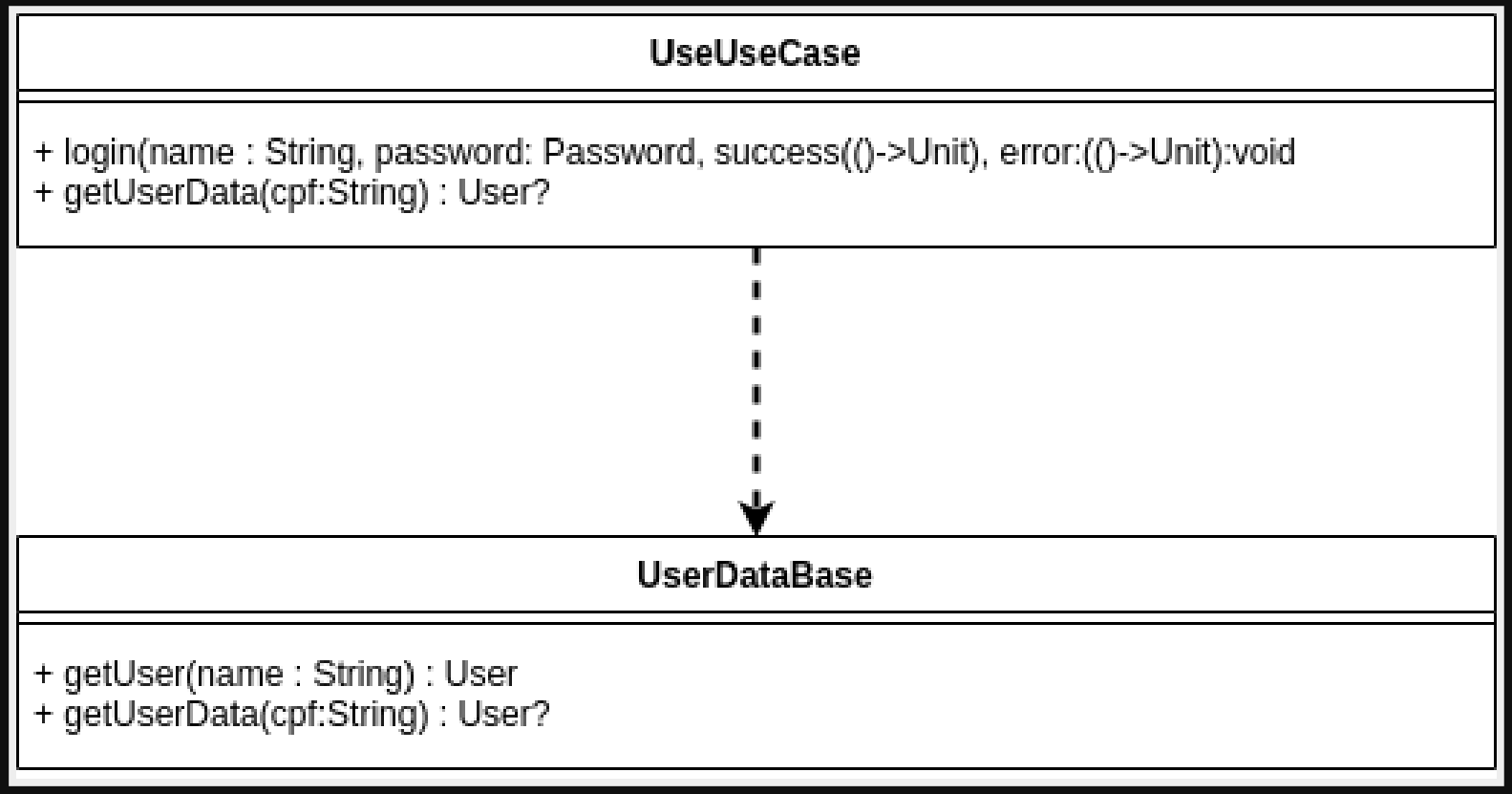
Interface Segregation

```
interface RecyclerViewListeners {
  interface BasicEvents {
    fun onClick(v : View, position : Int)
    fun onLongPress(v : View, position : Int, timePressed : Int)
  }
  interface SingleTaps {
    fun onSingleTap(v : View)
    fun onSingleDown(v : View)
  }
}
```

```
class RecyclerViewOnlyBasicEvents : RecyclerView(), RecyclerViewL
  override fun onClick(v : View, position : Int){
    // Has implementation
  }
  override fun onLongPress(v : View, position : Int, timePressed
    // Has implementation
  }
}
```

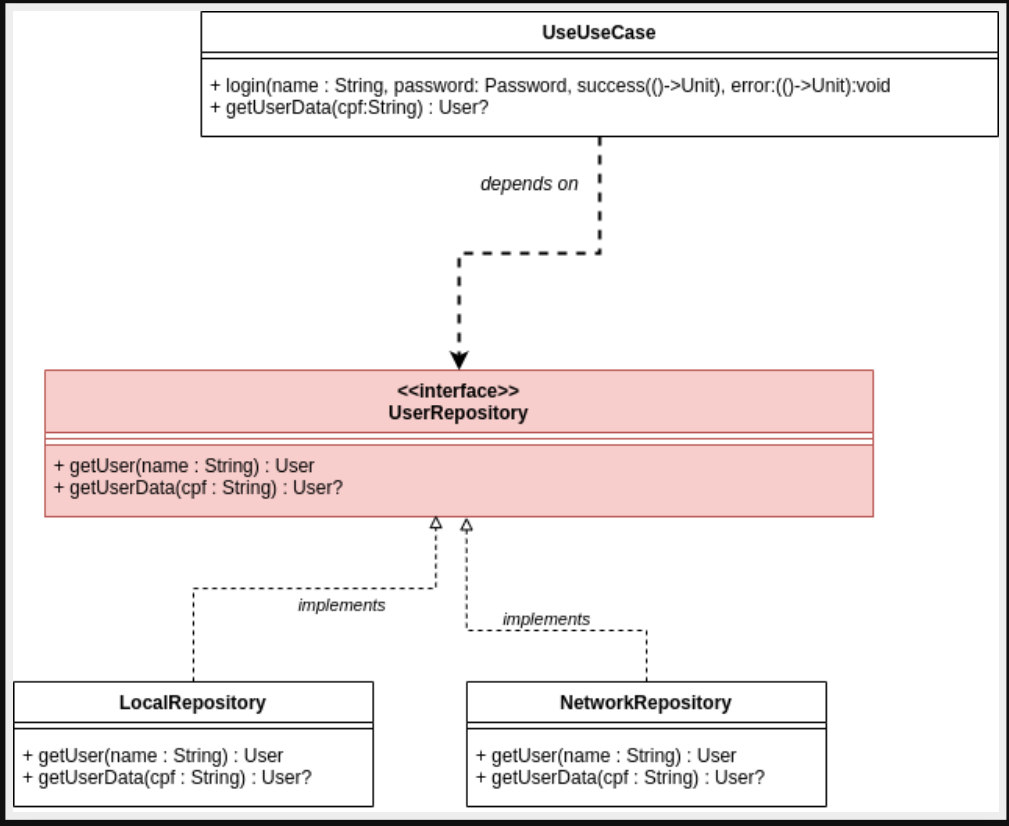
DEVGRID

Dependency Inversion



DEVGRID

Dependency Inversion



DEVGRID

Conclusions

It is hard to solve all of the problems that we have when we are developing software with good quality. To combine factors such as simplicity, performance, and efficiency are always a big challenge. For this reason, we build software in teams.

Useful Links

- Clean Architecture Tutorial for Android : Getting Started
- Essentials Components for Any Successful Android Project
- Getting started with Android Jetpack