Jetpack Compose with Kotlin

and Mastermind Game

Kotlin

- 100% Java interoperable
- Syntax similar to Java/C#/JavaScript
- Increased readability (minimum boilerplate)

What does Kotlin feature?

Null safety (drastically limiting the number of NPEs);

If, try-catch, when expressions;

Extension functions;

Var and val keywords;

Inline functions;

Named function parameters;

Multi-value return functions;

Semi-colons are optional

Functional programming constructs (like higher order functions)

Smart casting

Async/await

Data classes

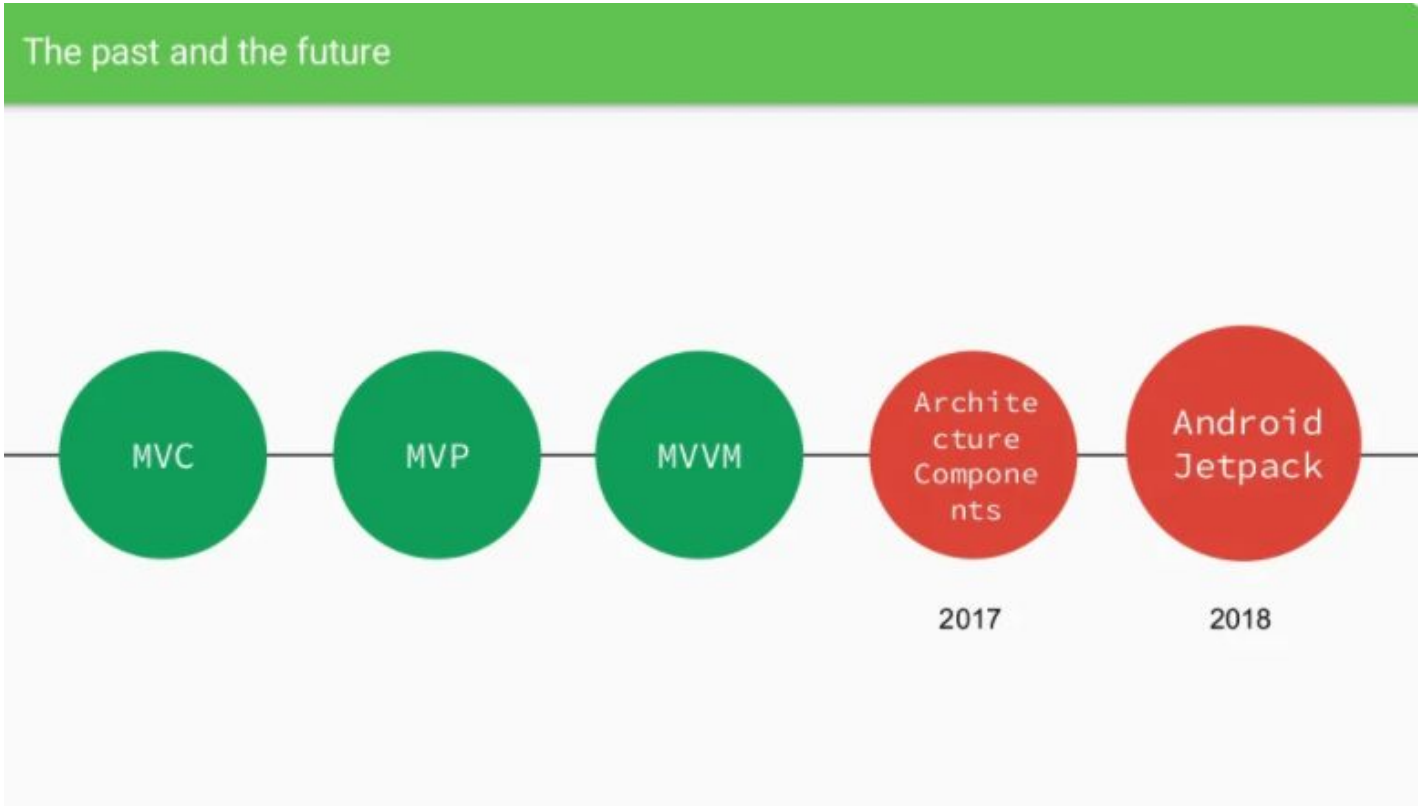
Properties (C#- like)

No checked exceptions

Operator overloading

and more …

Where’s Jetpack Compose come from



What is Jetpack Compose

1. A declarative toolkit for building UI
   a. Intuitive Kotlin APIs
2. Less Code
   a. less test and debug
   b. less to read, review and maintain
   c. “For the same Button class it [the code] was 10x of magnitude smaller.” (Twitter)
3. Powerful Tools
   a. easy to build your own widgets with existing components
   b. create beautiful apps with direct access to the Android platform APIs and built-in support for Material Design, Dark theme, animations, and more that combines a reactive programming model with the conciseness and ease of use of the Kotlin programming language.

Declarative vs Imperative Way

- Declarative programming is “the act of programming in languages that conform to the mental model of the developer rather than the operational model of the machine.”
- In computer science, declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow.
- “I draw the line between declarative and non-declarative at whether you can trace the code as it runs. Regex is 100% declarative, as it’s untraceable while the pattern is being executed.”

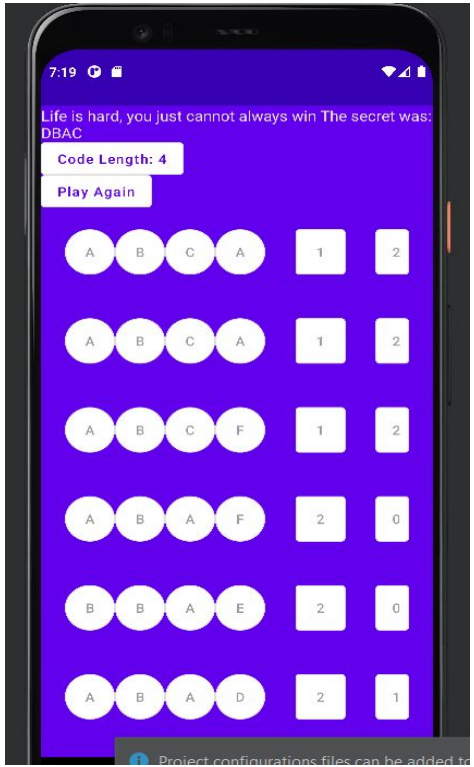
Reference: *Imperative vs Declarative Programming* Tyler McGinnis

Conclusions

- Fully compatible with your existing app/code
- Making things simpler is a big focus with Jetpack Compose
- Kotlin is really important to get an efficient declarative way of UI building.

Program “Mastermind” Game with Jetpack Compose

- Demo



Composable function

```
@Composable
private fun historyEntry(guess: String, right: String, wrong: String) {

    Row(modifier = Modifier.padding(24.dp)) {  this: RowScope
        for (letter in guess) OutlinedButton(
            modifier = Modifier.size(50.dp),  //avoid the oval shape
            shape = CircleShape,
            enabled = false,
            onClick = {}
        ) { Text(letter.toString())}
        Spacer(modifier = Modifier.size(30.dp))

        OutlinedButton(
            modifier = Modifier.size(50.dp),
            enabled = false,
            onClick = { }
        ) { Text(right) }

        Spacer(modifier = Modifier.size(30.dp))

        OutlinedButton(
            modifier = Modifier.size(50.dp),
            enabled = false,
            onClick = { }
        ) { Text(wrong) }
    }
}
```

Evaluating a guess

```
fun evaluateGuess(secret: String, guess: String): Evaluation {
    var rightPosition = 0
    var wrongPosition = 0
    var secretArr = secret.toCharArray()
    var guessArr = guess.toCharArray()

    for ( i in 0 .. secret.length -1 ) {
        if (secretArr[i] == guessArr[i]) {
            rightPosition++
            secretArr[i] = 0.toChar()
            guessArr[i] = 0.toChar()
        }
    }
    for (i in 0 .. secret.length - 1) {
        if (guessArr[i] != 0.toChar()) {
            val index = secretArr.indexOf(guessArr[i])
            if (index != -1) {
                secretArr[index] = 0.toChar()
                wrongPosition++
            }
        }
    }
    return Evaluation(rightPosition,wrongPosition)
}
```

Back up

MVC (Model View Controller)

Step 1: Incoming request directed to **Controller.**

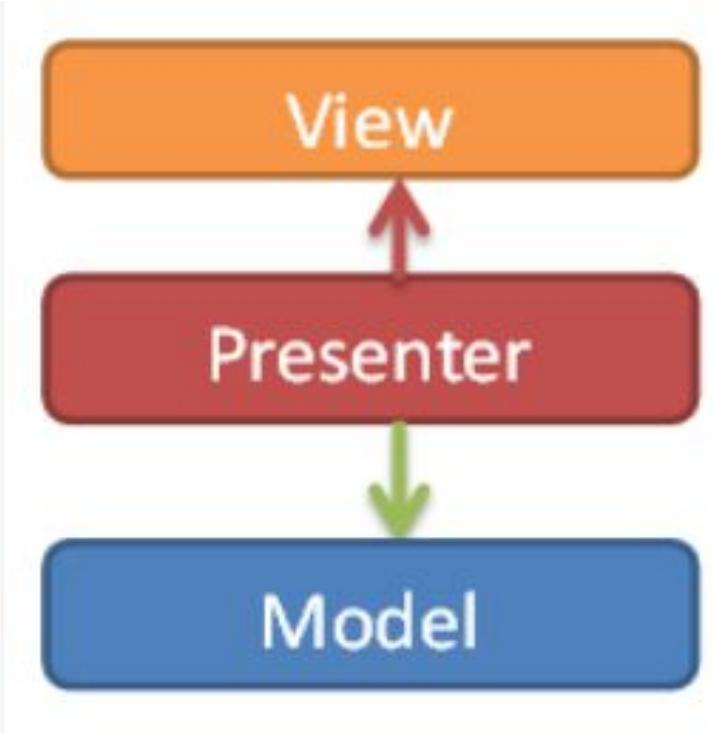
Step 2: **Controller** processes request and forms a data **Model.**

Step 3: **Model** is passed to **View.**

Step 4: **View** transforms **Model** into appropriate output format.

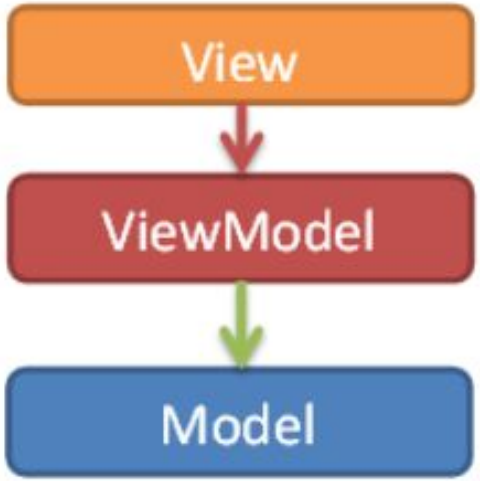
Step 5: Response is rendered.

MVP (Model View Presenter)



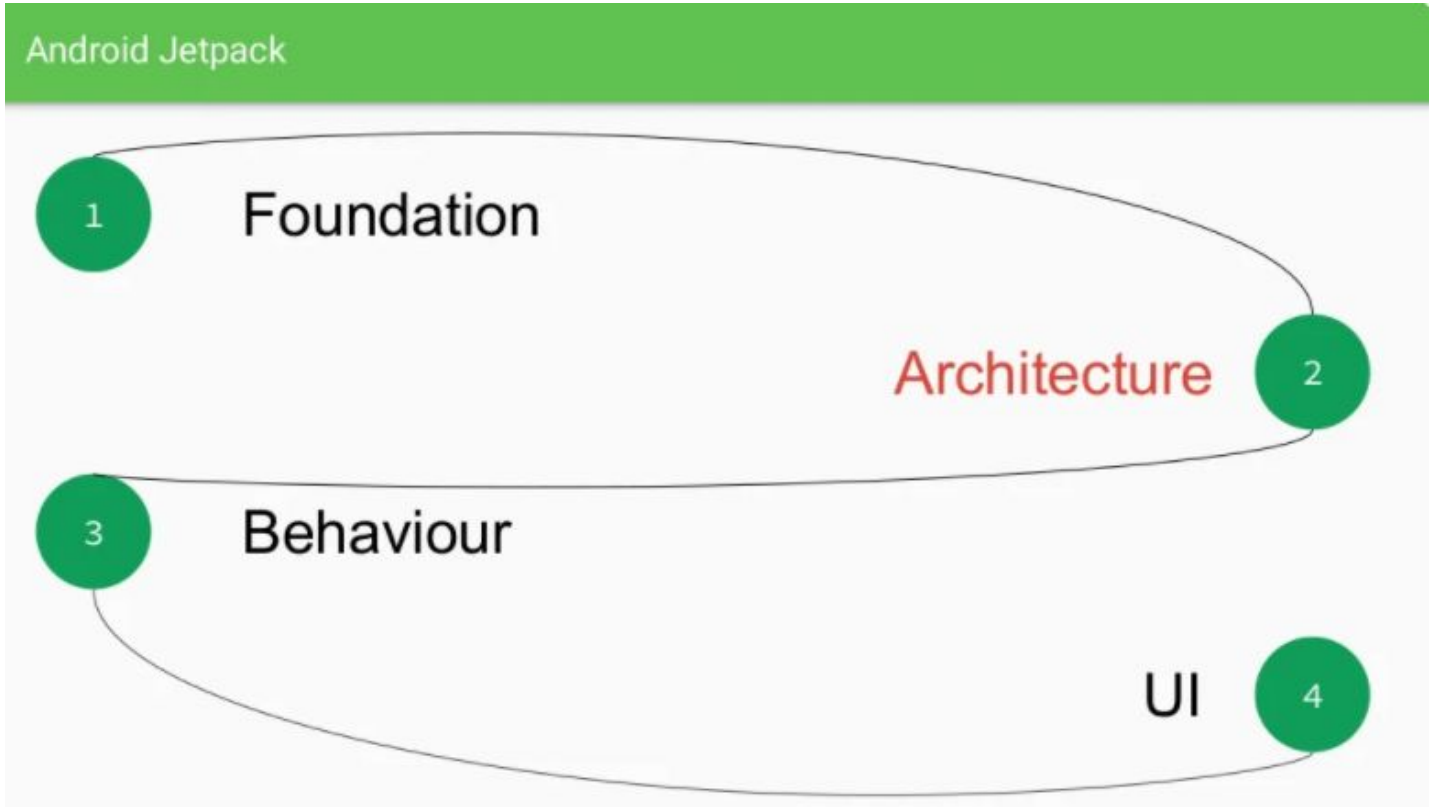
MVVM (Model View ViewModel)

1. `Model` (Business rule, data access, model classes)
2. `View` (User interface (XAML))
3. `ViewModel` (Agent or middle man between view and model)



Android Architecture Components

- Room
- Lifecycle
- ViewModel
- LiveData
- Data binding
- Paging
- Navigation
- WorkManager


Android Jetpack
1
Foundation
Architecture
2
3
Behaviour
UI
4