Kotlin 3.5 - Architektur und ViewModel

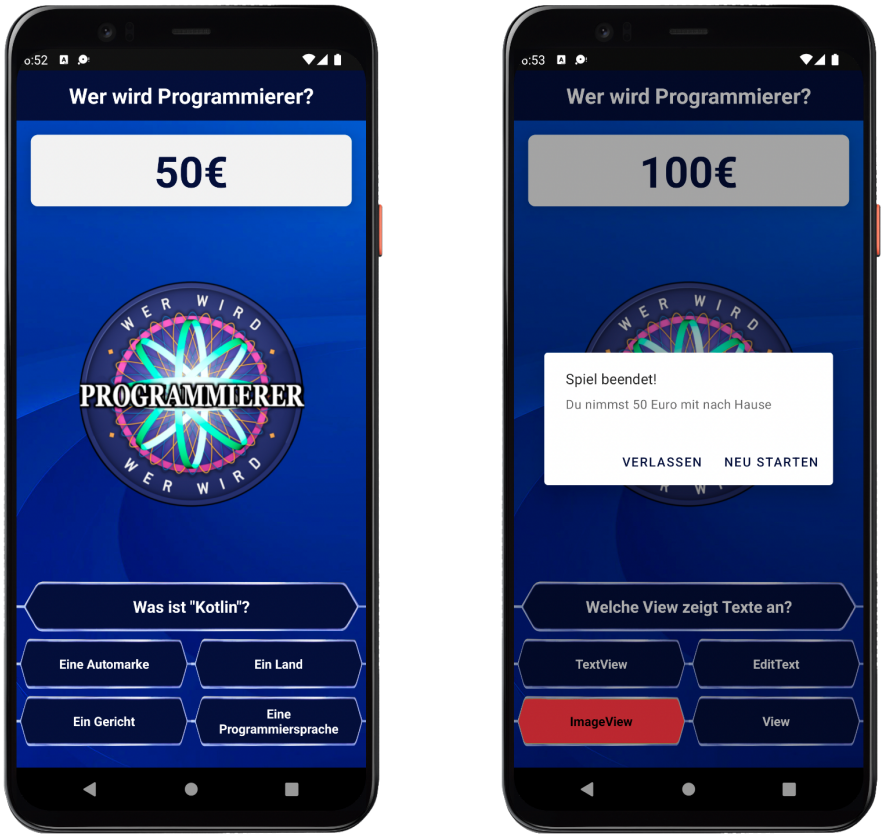
# Aufgaben

Hinweis: Zu bearbeiten ist Aufgabe 1.

Hinweis: Überprüfe,  bzw. Prüfe sind Schlüsselwörter und deuten häufig if - Anweisungen an.

## 1. WER WIRD PROGRAMMIERER? - ViewModel

In dieser Aufgabe geht es um die Millionen. Du programmierst die Logik der Quiz App “Wer wird Programmierer?”.




- Öffne das Projekt “Wer wird Programmierer”
- Schau dir die App an und führe sie einmal aus. Schau dir insbesondere die Klasse `Question` an und welche Informationen sie enthält
- Die Klasse `QuizFragment` kümmert sich um die UI Elemente, sorgt dafür, dass alles richtig angezeigt wird und gibt den Input des Users weiter. Die Klasse `QuizViewModel` beinhaltet die Logik des Spiels und sonst nichts anderes.
- Wir beginnen in Klasse `QuizViewModel` und programmieren hier die Logik des Spiels

- Erstelle eine Variable `repository`, in der ein Objekt der Klasse `QuizRepository` gespeichert ist.
- Greife nun über `repository` auf die Funktion `loadQuestions()` zu und speichere die zurückgegebene Liste in einer Variablen `questionsList`
- Erstelle zudem eine Variable `questionIndex` und weise ihr zu Beginn den Wert `0` zu
- Erstelle nun vier "verschachtelte" Variablen, die wir später im `QuizFragment` sehen können und die uns helfen, die UI Elemente richtig darzustellen:
  - Eine Variable `currentQuestion`, in der das aktuelle `Question` Objekt gespeichert wird (zu Beginn: das Erste in der Liste)
  - Eine Variable `moneyWon`, in der das erspielte Preisgeld gespeichert wird (zu Beginn: `0`)
  - Eine Variable `lastAnswer`, in der gespeichert wird, ob die letzte Antwort richtig oder falsch war (zu Beginn: `true`)
  - Eine Variable `wonTheMillion`, in der gespeichert wird, ob die Millionenfrage richtig oder falsch beantwortet wurde (zu Beginn: `false`)

  Hinweis: Richte dich bei der Verschachtelung nach diesem Beispiel:

```
private var _number = 0
val number: Int
   get() = _number
```

  Hinweis:
  Innerhalb des ViewModel wird nur die Variable mit dem Unterstrich verändert

- Schreibe nun den Inhalt der Funktion `restartGame()`. Diese Funktion wird aufgerufen, wenn das Spiel neu gestartet werden soll. Sie macht nichts anderes, als den Wert aller oben definierten Variablen auf den Wert zurückzusetzen, den die Variablen zu Beginn hatten.
- Wir kommen zur eigentlichen Spiellogik, diese soll in der Funktion `checkAnswer` programmiert werden. Die Funktion bekommt den Index der Antwort übergeben, die der User ausgewählt hat und Folgendes erledigen:
  - Prüfe, ob der übergebene Index gleich ist wie der, der als `rightAnswer` Attribut in der `_currentQuestion` gespeichert ist (also ob die Antwort stimmt)
  - Falls nicht, soll die Variable `_lastAnswer` auf `false` gesetzt werden und da das bedeutet, dass das Spiel vorbei ist, soll sonst nichts geschehen
  - Falls die Antwort richtig war, soll in dem `_moneyWon` Attribut jetzt das aktuelle Preisgeld gespeichert werden. Falls es sich außerdem um die letzte Frage gehandelt hat, soll `_wonTheMillion` auf `true` gesetzt werden, da das Spiel gewonnen ist. Falls es nicht die letzte Frage war, soll der Frage Index um

eins erhöht werden und die neue Frage aus der Liste in der `_currentQuestion` gespeichert werden.

- Jetzt, wo die Logik steht, sorgen wir in der Klasse `QuizFragment` noch dafür, dass alle UI Elemente richtig angezeigt und die User Eingaben richtig verarbeitet werden. In der Klasse befinden sich bereits ein paar Variablen und Funktionen, in denen hauptsächlich die Farben und Hintergründe der Elemente verwaltet werden. Schau dir die Klasse erst einmal genau an und programmiere dann die unvollständigen Funktionen:
    - Die Funktion `setPriceQuestionAnswer()` wird jedes Mal aufgerufen, wenn die Inhalte der Preisstufe, der Frage und der Antworten geändert werden sollen.
        - Setzte das `text` Attribut von `tvQuestion` und von `tvAnswerA` bis `tvAnswerD` auf die entsprechenden Attribute, die in `currentQuestion` im `viewModel` gespeichert sind
        - Setze das `text` Attribut der `TextView` `tvPrice` mithilfe von `getString()` auf die aktuelle Preisstufe. Übergebe `getString()` die String Ressource namens `current_price` als ersten Parameter, und als zweiten Parameter die Variable `price`, die in der `currentQuestion` des `viewModel` gespeichert ist.
            - Hinweis: getString() wird hier benutzt, um die String Resource mit dem aktuellen Wert des Preises zu formatieren.
    - Die Funktion `checkAnswerUpdateUI()` wird jedes Mal aufgerufen, wenn der User auf eine Antwort klickt (siehe `onClickListener` in `onViewCreated`). Sie bekommt die `TextView` übergeben, auf die geklickt wurde und den Index der Antwort, die die `TextView` enthält. Folgendes soll die Funktion machen:
        - Führe die vorhin programmierte Funktion `checkAnswer` des `viewModel` aus und übergebe ihr den Index der beantworteten Frage. Hierdurch werden alle Variablen im `viewModel` aktualisiert.
        - Überprüfe nun mithilfe der Variable `lastAnswer` aus dem ViewModel, ob die Frage richtig beantwortet wurde. Falls ja, setze den Hintergrund der übergebenen `TextView` auf `backgroundNormal`, die Schriftfarbe auf `white` und rufe die Funktion `setPriceQuestionAnswer()` auf, um die nächste Frage anzuzeigen. Falls nicht, setze den Hintergrund auf `backgroundWrong`, die Schriftfarbe auf `black` und führe die Funktion `showEndDialog` aus, mit dem Text aus der String Ressource `game_over` als Parameter.
        - Falls die Variable `wonTheMillion` im ViewModel jetzt `true` ist, soll der Hintergrund auf `backgroundCorrect` gesetzt werden und die Funktion `showEndDialog` aufgerufen werden, mit dem Text aus der String Ressource `game_won` als Parameter.

    - Zum Schluss wollen wir den End-Dialog einrichten, der bei Ende des Spiels angezeigt werden soll. Erstelle dafür in der Funktion `showEndDialog` einen neuen `MaterialAlertDialogBuilder` mit folgenden Parametern:
        - `requireContext()` für den Context
        - den übergebenen Parameter `title` für
        - Für die `Message` => Formatiere String Resource you_won_amount mit der moneyWon Variale aus dem ViewModel
        - Er soll nicht `cancelable` sein
        - Der `NegativeButton` soll die String Ressource `exit` anzeigen und die Funktion `exitGame()` ausführen
        - Der `PositiveButton` soll die String Ressource `play_again` anzeigen und die Funktion `restartGame()` ausführen
            - *Hinweis die beiden Button set Funktionen brauchen noch einen Funktionskörper in dem der Listener implementiert ist, welche die exitGame() bzw. restartGame() aufruft in folgendem Format:* ***{ x, y -> }***
        - zeige den Dialog mithilfe von `show()`
- Das ViewModel speichert die Informationen und den aktuellen Spielstand, selbst wenn z.B. das Smartphone in den Landscapemodus gedreht wird und das Layout neu gebaut wird, bleiben die angezeigten Informationen erhalten. Probiere es einmal aus:
- Führe die App aus und teste das Spiel. Schaffst du es bis zur Million?

Hinweis: Alle Parameter (mit Ausnahme des Contextes) können direkt über **set** Funktionen mit selbsterklärenden Namen gesetzt werden.
Beispiel Title = > `MaterialAlertDialogBuilder` (requireContext())
.setTitle(title)
.set<ParamterName>(<value>)

Viel Erfolg!