Name: Romisaa Mohamed Fadel

**Part A: Clean Architecture Theory**

**Question 1 :**

* Answer: **C**

**Question 2 :**

* Answer: **D**

**Question 3 :**

* Answer: **B**

**Question 4 :**

* Answer: **C**

**Question 5 :**

* Answer: **C**

**Part B: Flutter Widget Exploration**

* Challenge 1: **=>** [Github Repo link](https://github.com/romisaa5/FlutterMentors_Mentorship_R3/tree/main/week%202/challenge_1)
* Challenge 2: **=>** [Github Repo link](https://github.com/romisaa5/FlutterMentors_Mentorship_R3/tree/main/week%202/challenge_2)
* Challenge 3: **=>** [Github Repo link](https://github.com/romisaa5/FlutterMentors_Mentorship_R3/tree/main/week%202/challenge_3)

**Widget Knowledge Questions :**

**Question 6 :**

**What's the difference between Expanded and Flexible widgets? Provide a use case for each.**

* **Expanded:** Forces the child widget to fill all the remaining available space.
* **Use case:** A text field inside a Row that should take up all leftover space next to an icon.
* **Flexible:** Lets the child widget occupy only the space it needs but can shrink if necessary.
* ***Use case*:** An image in a Row that should keep its aspect ratio but still adjust when the screen size changes.

**Question 7 :**

**When would you use RepaintBoundary and why is it important for performance?**

**RepaintBoundary** is used to isolate a part of the widget tree from unnecessary repaints. When you wrap a widget with RepaintBoundary, Flutter creates a separate layer for it. This means if only that widget changes, Flutter repaints just that boundary instead of the entire parent tree.

It’s important for **performance optimization**, especially in complex UIs or animations, because it reduces the rendering workload and avoids expensive repaints across large parts of the widget tree.

**Question 8 :**

**Explain the difference between AnimatedContainer and AnimatedBuilder. When would you choose one over the other?**

**AnimatedContainer**:  
A ready-to-use widget that automatically animates changes to its properties, such as size, color, border, or padding. You just update the property, and it smoothly transitions from the old value to the new one. **Best use case**: When you need simple, direct animations without much control.

**AnimatedBuilder**:  
A more flexible widget that gives you full control over animations. It works with an AnimationController and Animation objects, letting you rebuild any part of the UI based on the animation’s value.  
**Best use case**: When you need complex or custom animations (e.g., moving along a path, synchronizing multiple animated properties).

**Question 9 :**

**What is the purpose of GlobalKey and when should you avoid using it?**

GlobalKey in Flutter is used to uniquely identify a widget across the widget tree. It allows you to access the widget’s state, its BuildContext, or its RenderObject from outside the widget itself. It’s especially useful when you need to:

1. Access a State of a StatefulWidget from a parent or another widget.
2. Preserve the state of a widget when it moves in the widget tree.
3. Access layout information like size or position via the widget’s context.

However, you should avoid using GlobalKey unnecessarily because:

* It’s expensive in terms of performance.
* Overusing it can make your code harder to maintain.
* If there’s a simpler solution using normal State management (setState, Provider, Riverpod, Cubit, etc.), that’s preferred.

**Question 10 :**

**Describe the widget lifecycle methods and when each is called.**

In Flutter, the lifecycle of a StatefulWidget is managed by its associated State object, which holds mutable data. The key lifecycle methods are:

* **createState()** → Called once when the widget is inserted into the widget tree, responsible for creating the State object.
* **initState()** → Called once when the State object is first created; used for initializations like setting up controllers or starting listeners.
* **didChangeDependencies()** → Called after initState() and whenever dependencies change (e.g., when an InheritedWidget the widget depends on updates).
* **build()** → Called frequently to describe how the widget should look based on the current state.
* **didUpdateWidget()** → Called when the parent widget rebuilds and passes new configuration to the current widget.
* **deactivate()** → Called when the widget is removed from the tree but might be reinserted later.
* **dispose()** → Called when the widget is permanently removed; used to clean up resources like controllers and subscriptions.

In short, **initState()** is for setup, **build()** is for rendering, **didUpdateWidget()** handles updates, and **dispose**() is for cleanup.

* **Mini App: Widgets =>** [Github Repo link](https://github.com/romisaa5/FlutterMentors_Mentorship_R3/tree/main/week%202/mini_app_widgets)
* **Mini (UI only) App: Clean Architecture =>** [Github Repo link](https://github.com/romisaa5/FlutterMentors_Mentorship_R3/tree/main/week%202/mini_ui_app)
* [**Problem Solving**](https://github.com/romisaa5/problem_solving_dart) **=>**[Github Repo link](https://github.com/romisaa5/problem_solving_dart)
* **Post LinkedIn => Link**