Mobile Applications
CSCI 448
Lecture 09

Saving State:

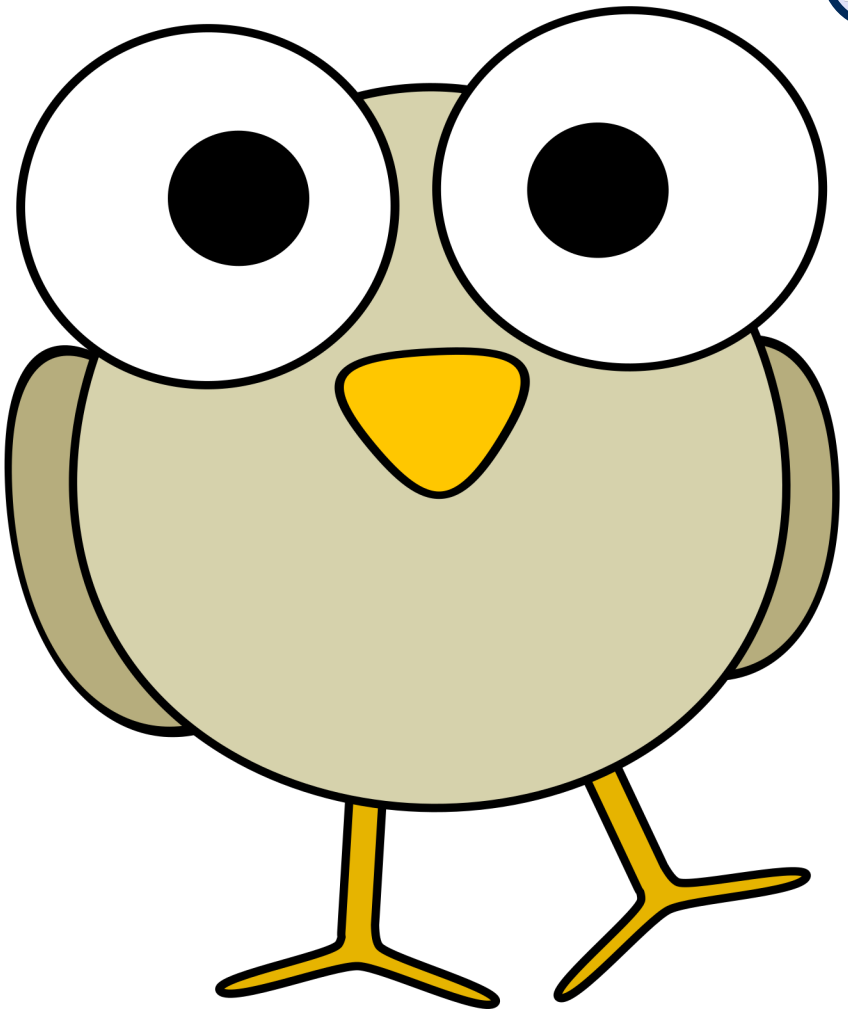
Jetpack ViewModel

Previously in CSCI 448

- Activity Lifecycle
  - States:

    Non-Existent → Stopped → Paused → Running
  - Lifecycle Methods
    - onCreate(Bundle?)
    - onStart()
    - onResume()
    - onPause()
    - onStop()
    - onDestroy()

Questions?



Learning Outcomes For Today

- Create an app that can save state across destruction and recreation via a ViewModel and a Saved Instance State (Bundle)

- Define the Decorator, Factory Method, & Singleton design patterns and map their applications to ViewModels

Persisting UI State

- State is lost from two scenarios when activity is destroyed & recreated

1. Configuration changes

2. Process death

Persisting UI State

- State is lost from two scenarios when activity is destroyed & recreated

1. Configuration changes (Today)

2. Process death (Monday)

On Tap For Today

- ViewModel
- Practice

On Tap For Today

- ViewModel

- Practice

Where is the state?

- View Model currently storing the state

Design Pattern #3: Decorator

- Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality
- Participants:
  - `Component`: defines the interface for objects that can have responsibilities added to them
  - `Decorator`: maintains a reference to a `Component` object and defines an interface that conforms to the `Component`'s interface
  - `ConcreteComponent`: defines an object to which additional responsibilities can be attached
  - `ConcreteDecorator`: adds responsibilities to the component

`ViewModel` Decoration

- `Component` → `Any`
- `Decorator` → `ViewModel`
- `ConcreteComponent` → `Question`
- `ConcreteDecorator` → `QuestionViewModel`

Why is state lost?

- View Model currently storing the state

Saving Data Across Config Changes

- Use a Jetpack `ViewModel`
  - "Lifecycle aware"
  - Observes the lifecycle of another component

`ViewModel` in Action

Figure 4.2 **`QuizViewModel`** scoped to **`MainActivity`**

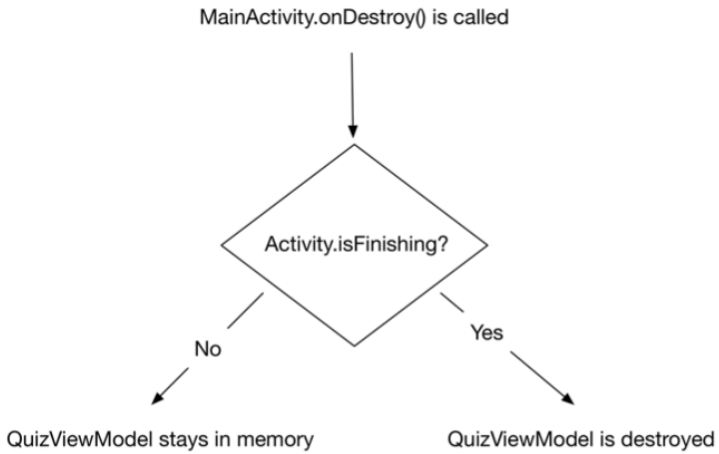
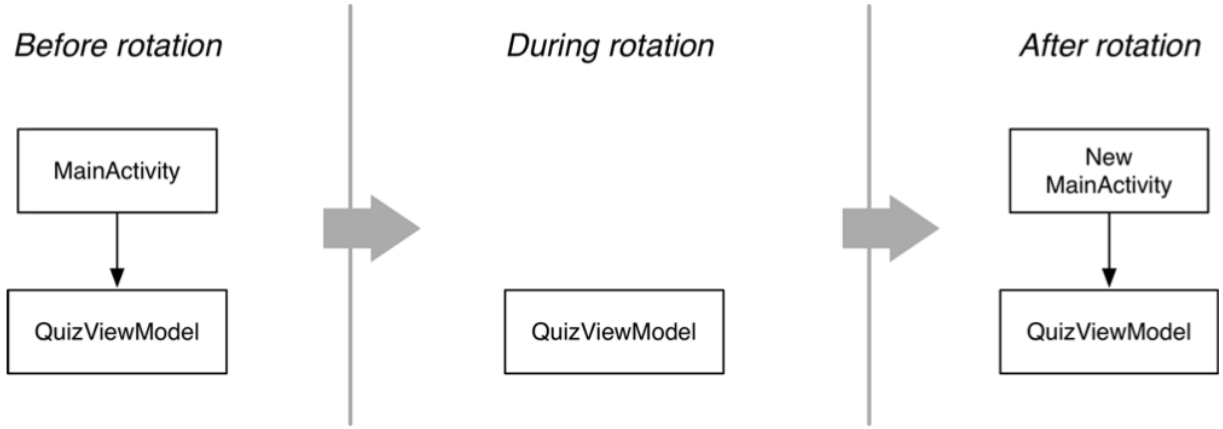


Figure 4.3 **`MainActivity`** and **`QuizViewModel`** across rotation



`ViewModel` Players

- `ViewModelStoreOwner` – owns a store
- `ViewModelStore` – map of `ViewModel` type to `ViewModel` instance (at most one instance per type)
- `ViewModelProvider` – returns an existing `ViewModel` from the `Owner`'s `Store` if one exists, otherwise creates a new `ViewModel`

Design Pattern #4: Singleton

- Ensure a class only has one instance, and provide a global point of access to it.

- Participants:
  - `Singleton`:
    - Defines an Instance operation that lets clients access its unique instance. Instance is a class operation.
    - May be responsible for creating its own unique instance

Design Pattern #5: Factory Method

- Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- Participants:
  - `Product`: defines the interface of objects the factory method creates
  - `Creator`: declares the factory method, which returns an object of type `Product`
  - `ConcreteProduct`: implements the Product interface
  - `ConcreteCreator`: overrides the factory method to return an instance of a `ConcreteProduct`

ViewModel Factory Method

- `Product` →
- `Creator` →
- `ConcreteProduct` →
- `ConcreteCreator` →
- `ViewModelStoreOwner` →

Android Design Patterns

- Behavioral Patterns
  1. Command – UI Event Handling
  2. Observer – State
- Creational Patterns
  3. Factory – ViewModelFactory
  4. Singleton – ViewModelProvider, Repository
- Structural Patterns
  5. Decorator – View Model

Persisting UI State

- State is lost from two scenarios when activity is destroyed & recreated

1. Configuration changes
   - Use `ViewModel`

2. Process death
   - Stay tuned!

On Tap For Today

- ViewModel
- Practice

To Do For Next Time

- Lab02 due tonight 2/3
- FP Proposal due tonight 2/3
- A1 due Tuesday 2/7
- FP Storyboards due next week 2/10