Testing

Application Development

NIFR 2023

Digital Kultur

DAGENS INDHOLD

Alle grupper laver en kort pitch (5-10 min) af deres projekter.

Her skal i forklare:
Hvad var jeres emne og problemstillinger?
Hvorfor valgte i emnet og problemstillingerne?
Hvordan har i samlet data / empiri?
Hvilket teori / litteratur har i inddraget?
Hvad er hovedpunkterne fra jeres analyse?
Hvad har i lært?

Agenda

Testing

- Brief recap of last week
  - Software Architecture
  - Design Patterns
- Testing / QA
  - Unit test recap
  - UI testing using espresso

Software Architecture

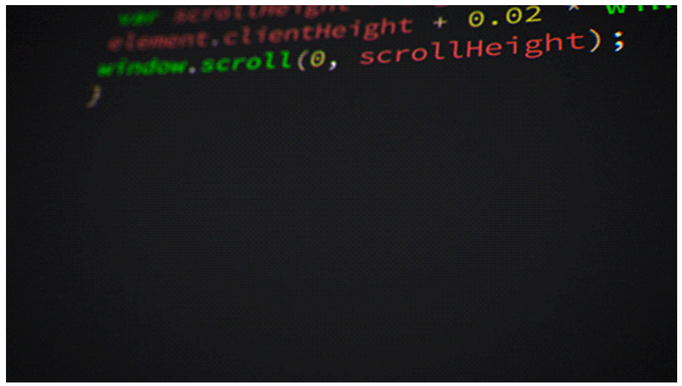
Why

https://earthlymission.com/how-many-lines-of-code-does-it-have/

What is (source) code?

Developer Perspective

On the surface



Application:
Collection of source code

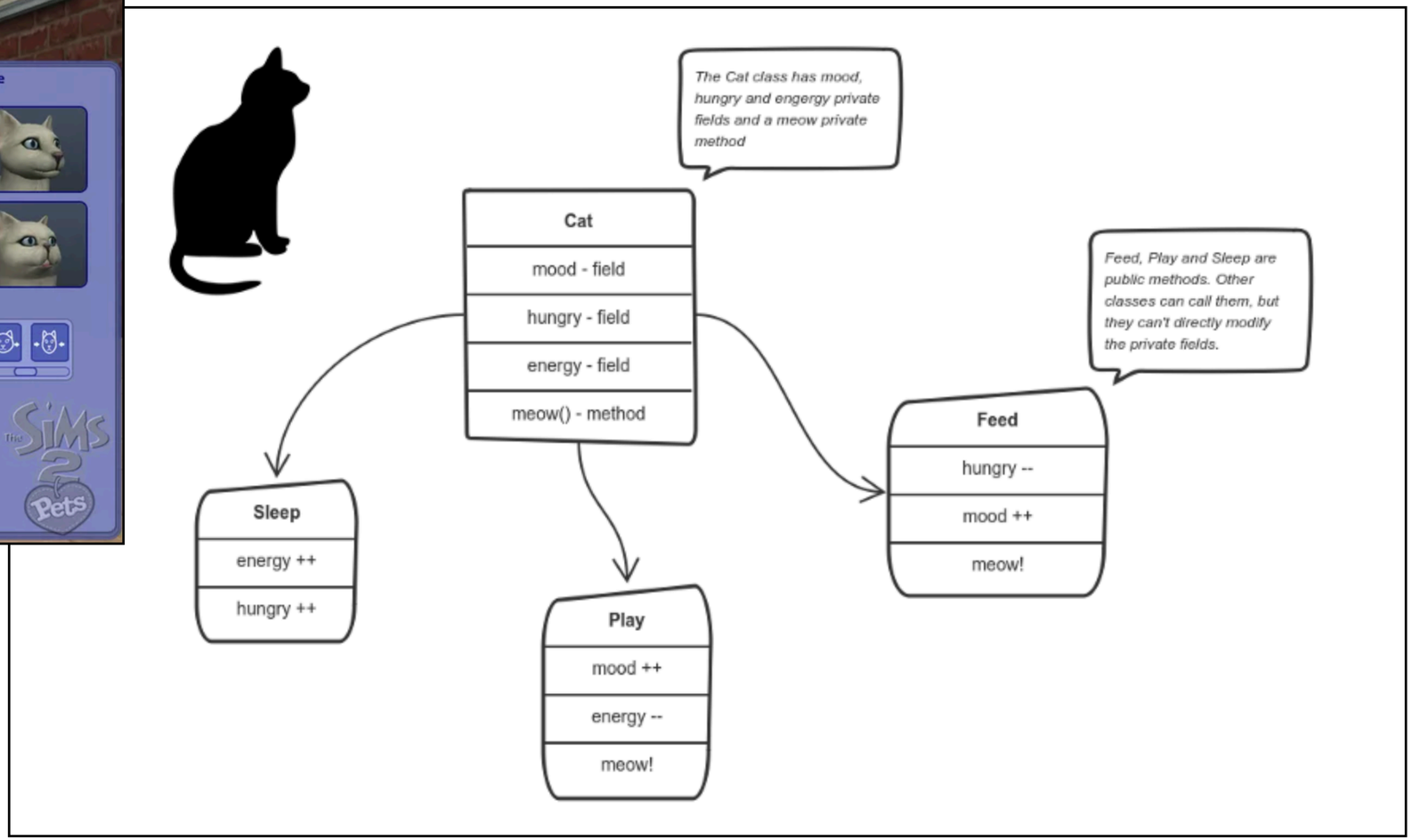
Developer Perspective

Behind the curtain

Application:
Collection of interacting components

Developer Perspective

Behind the curtain



Developer Perspective

Behind the curtain

Poul Madsen overhørte advarsler: Brutal leder på Ekstra Bladet gav ansatte ondt i maven i årevis

MEDIER | P | 🔖

5 vigtige detaljer du skal kende, før du anmoder om dine feriepenge

PRIVATØKONOMI | P | 🔖

Frankrigs jungle: »Jeg vil vise dig et sted. Du vil ikke tro det, når du ser det

INTERNATIO

5 hjerter: Det handler ikke om alkohol eller kvindeligt selskab. Det

```
public class Article{
    private String headline;
    private String author;
    private String category;
    private boolean isOnFrontPage;
}
```

Layered Architecture

Recommended app architecture

- Seperation of concerns
- UI layer display application data
- Domain Layer handles *business logic*
- Data layer handles data (fetching, writing, CRUD)

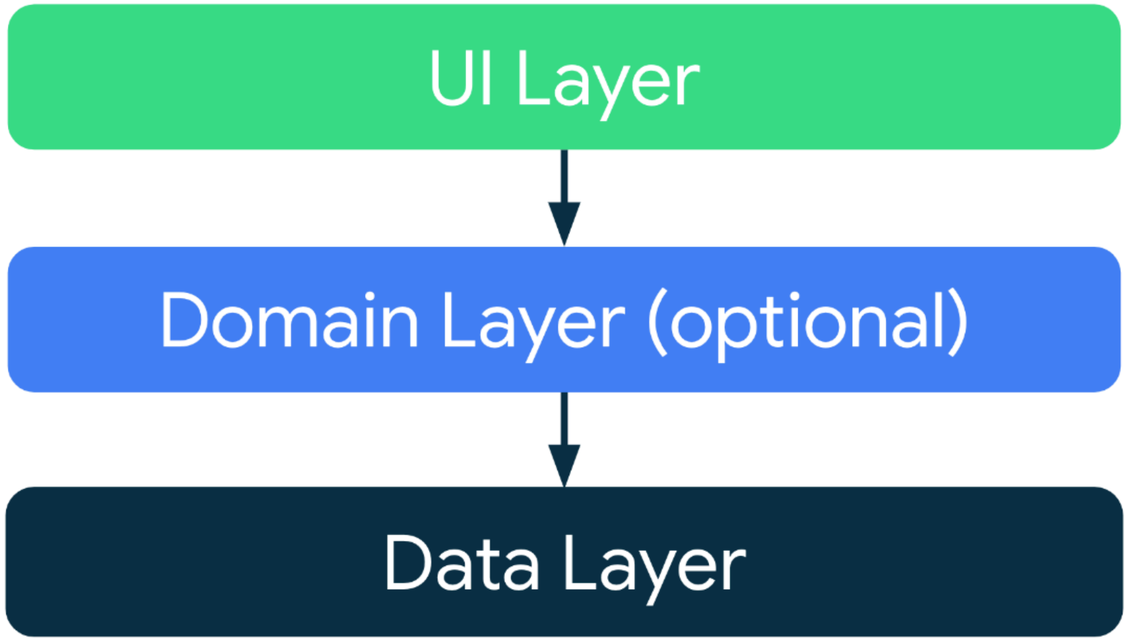


Figure 1. Diagram of a typical app architecture.

Domain/Business Layer

Definition

Sold out items should be marked with red font

Maximum tweet size is 280 characters

A user should always enter an address, otherwise we cannot deliver

Organisation

Application Architecture

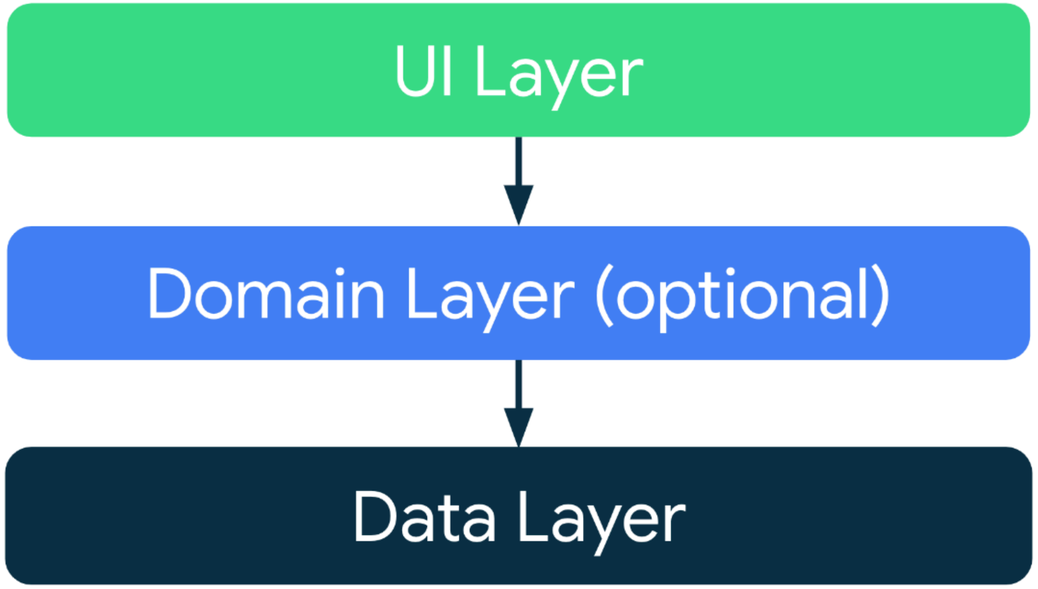
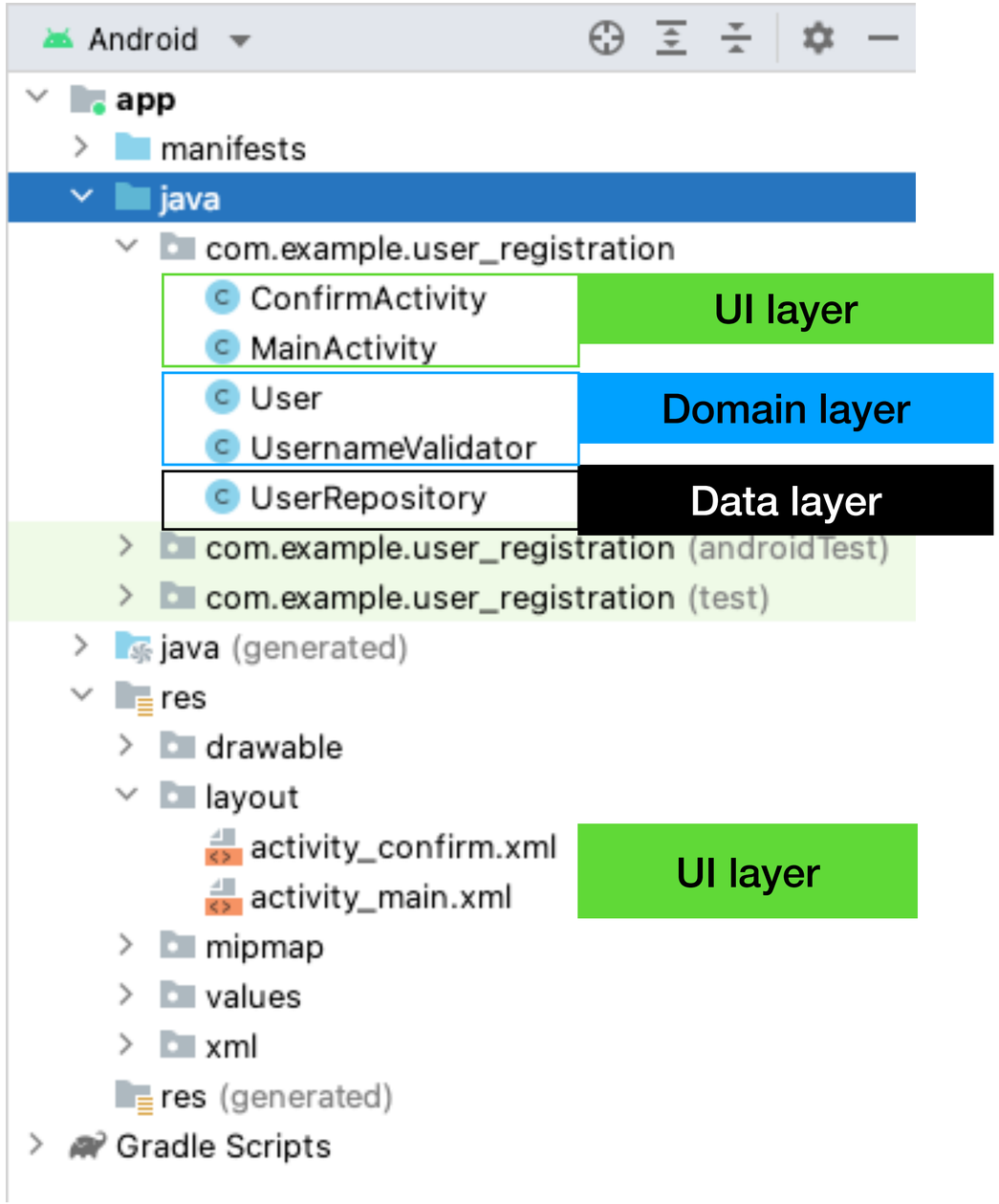
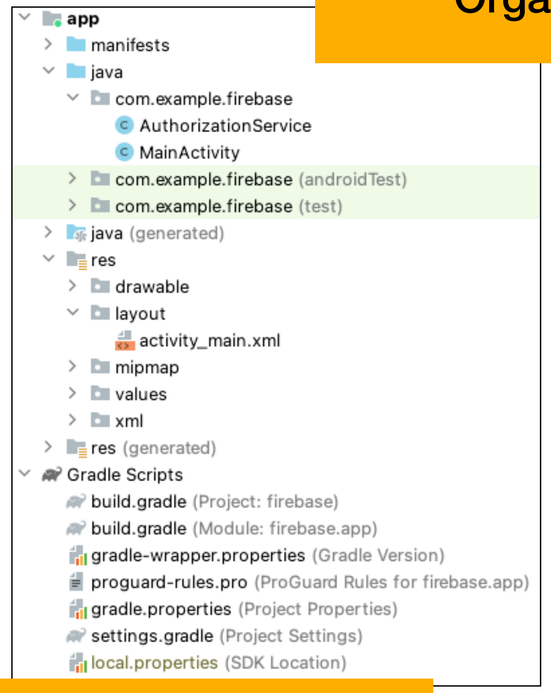


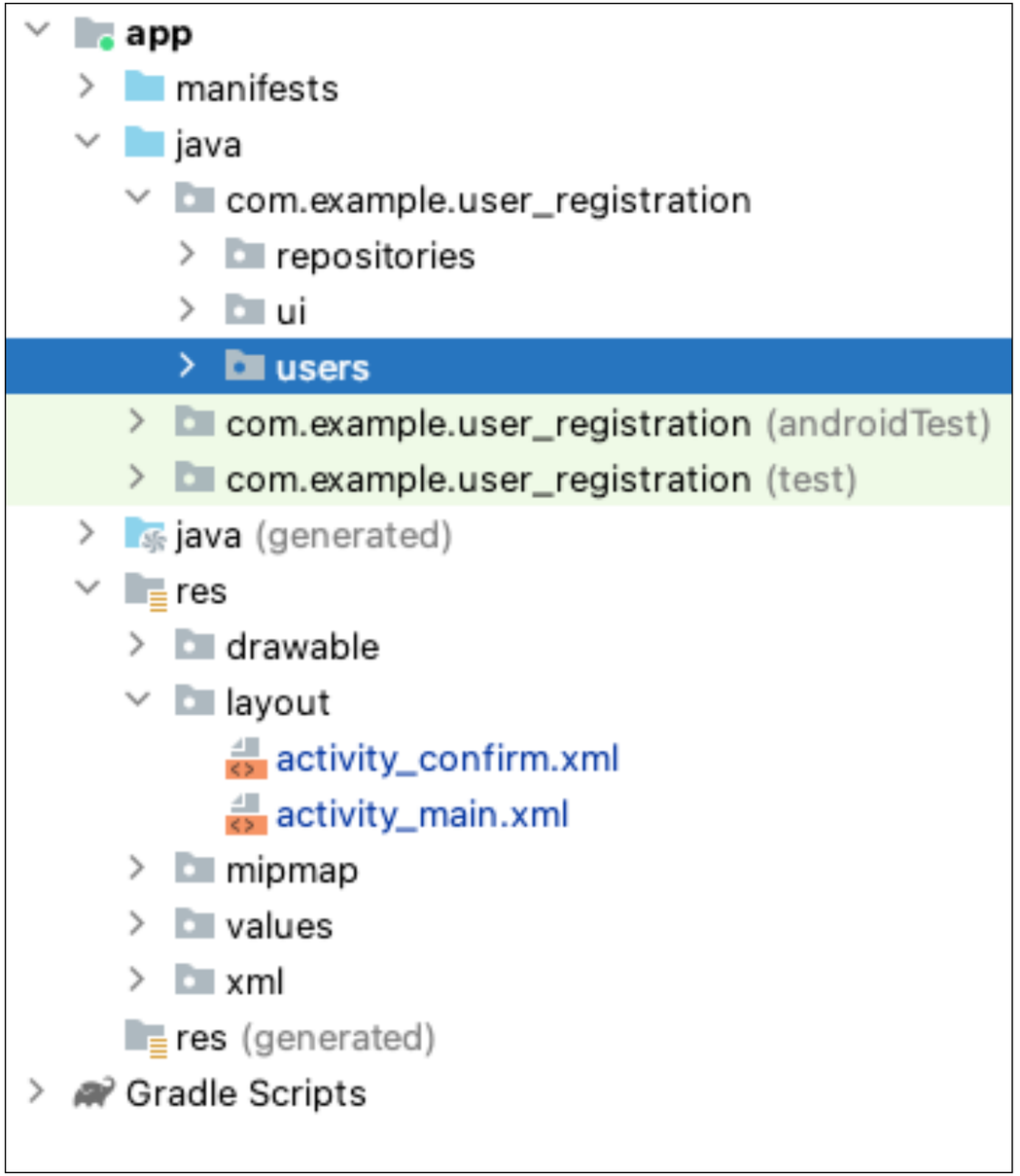
Figure 1. Diagram of a typical app architecture.



Organisation



Where do we put this file?



Separation of concerns

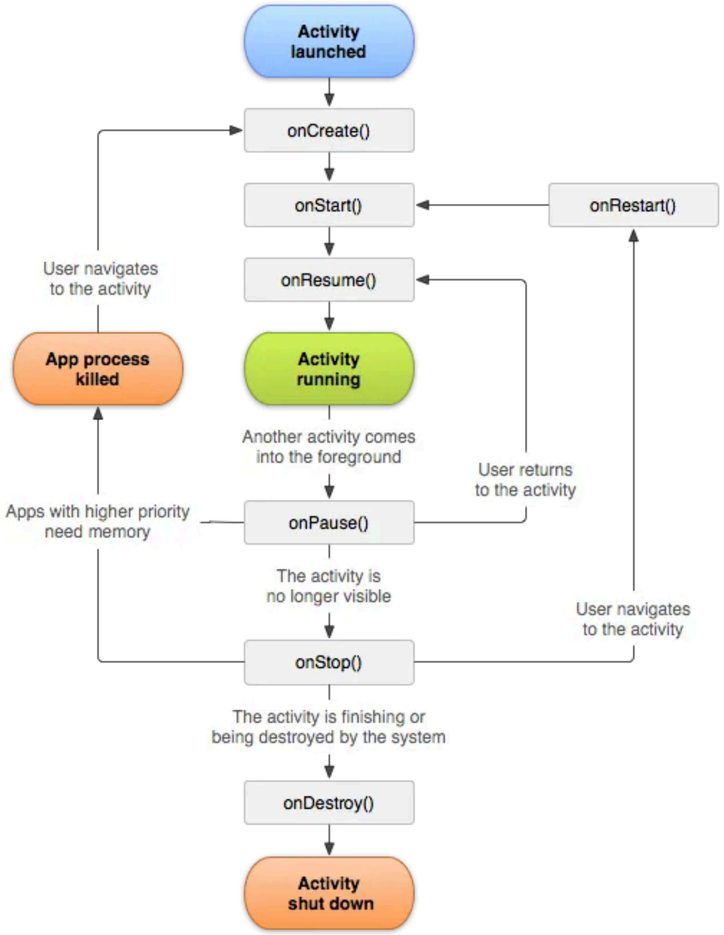
Architectural Principles

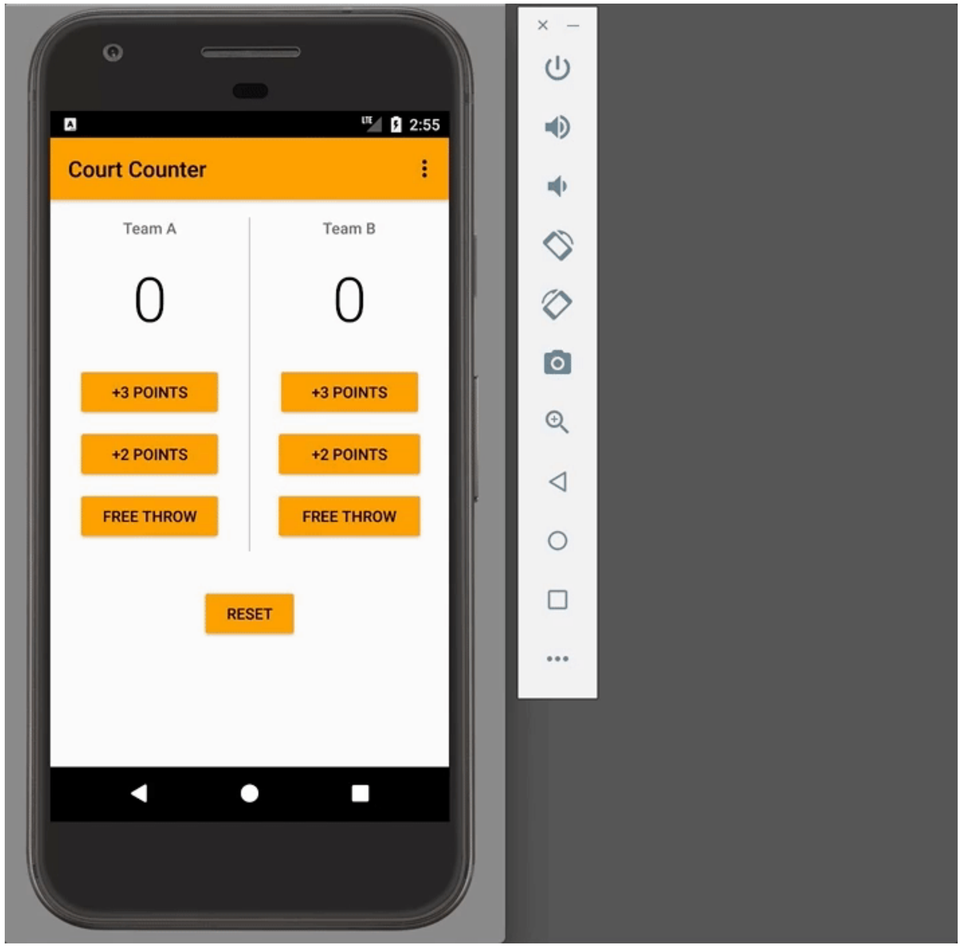
It's a common mistake to write all your code in an Activity or a Fragment. These UI-based classes should only contain logic that handles UI and operating system interactions.

https://developer.android.com/topic/architecture

Drive UI from data model

Architectural Principles

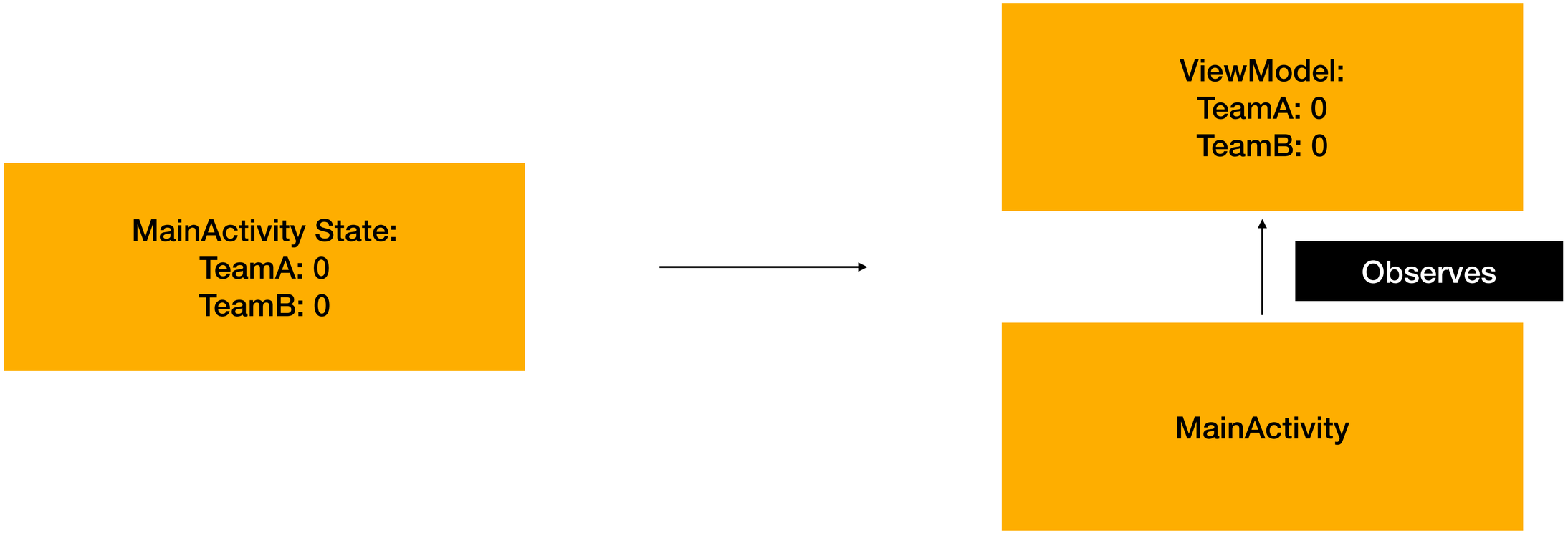




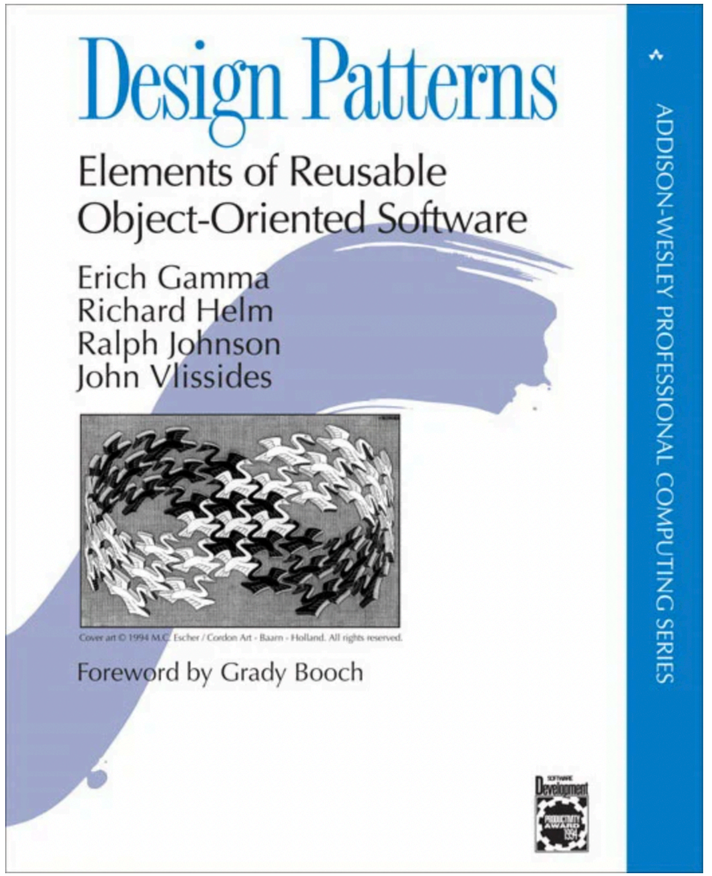
https://github.com/udacity/Court-Counter

https://medium.com/androiddevelopers/viewmodels-a-simple-example-ed5ac416317e

Drive UI from data model



Gang of Four (GoF)

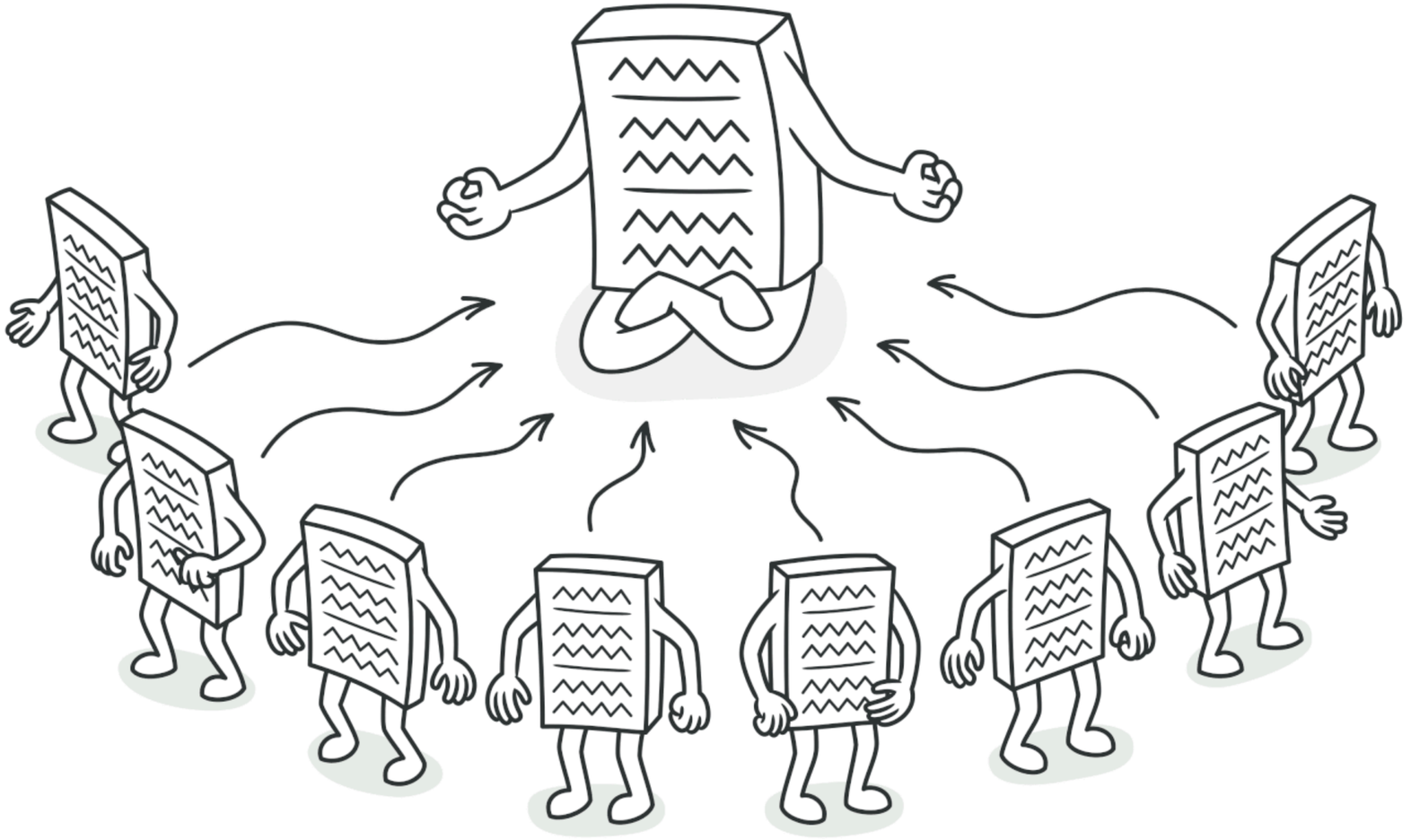


Design patterns are typical solutions to commonly occurring problems in software design. They are like pre-made blueprints that you can customize to solve a recurring design problem in your code.

You can’t just find a pattern and copy it into your program, the way you can with off-the-shelf functions or libraries. The pattern is not a specific piece of code, but a general concept for solving a particular problem. You can follow the pattern details and implement a solution that suits the realities of your own program.

- How could a design pattern be implemented and contribute in your application?
  - Make a suggestion to redesign a part of your application or a new feature, to implement a GoF pattern such as:
  - (Abstract) Factory, Singleton, Adapter, Template Method

Singleton



1. **Ensure that a class has just a single instance**. Why would anyone want to control how many instances a class has? The most common reason for this is to control access to some shared resource—for example, a database or a file.

   Here's how it works: imagine that you created an object, but after a while decided to create a new one. Instead of receiving a fresh object, you'll get the one you already created.

Basic Implementation

All implementations of the Singleton have these two steps in common:

- Make the default constructor private, to prevent other objects from using the `new` operator with the Singleton class.
- Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

```
public class Singleton{
    private static Singleton theSingleObject;

    //Private constructor
    private Singleton() {}

    //Public method to interact with instance
    public static Singleton getInstance(){
        //Lazy instantiation
        if (theSingleObject == null)
            theSingleObject = new Singleton();
        return obj;
    }
}
```

Singleton

Web version 9 (modular) | Web version 8 (namespaced) | Swift | Objective-C | Kotlin+KTX Android | Java Android | Dart Flutter | Go | More

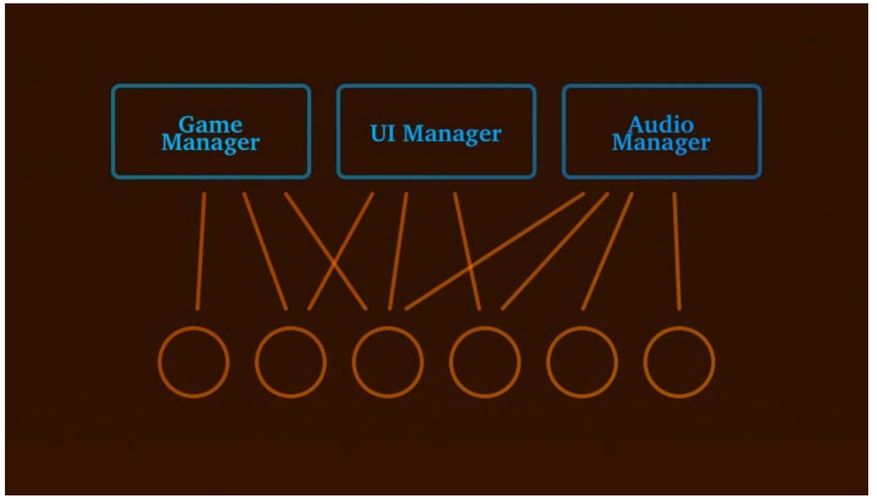
```java
// Access a Cloud Firestore instance from your Activity


FirebaseFirestore db = FirebaseFirestore.getInstance();
```

DocSnippets.java

Why use a singleton?

Singletons can be very convenient because they allow you to connect parts of your game more easily.

For example, you could use a singleton to make the player's position publicly accessible to enemies, or use one to expose the player's health to the UI.

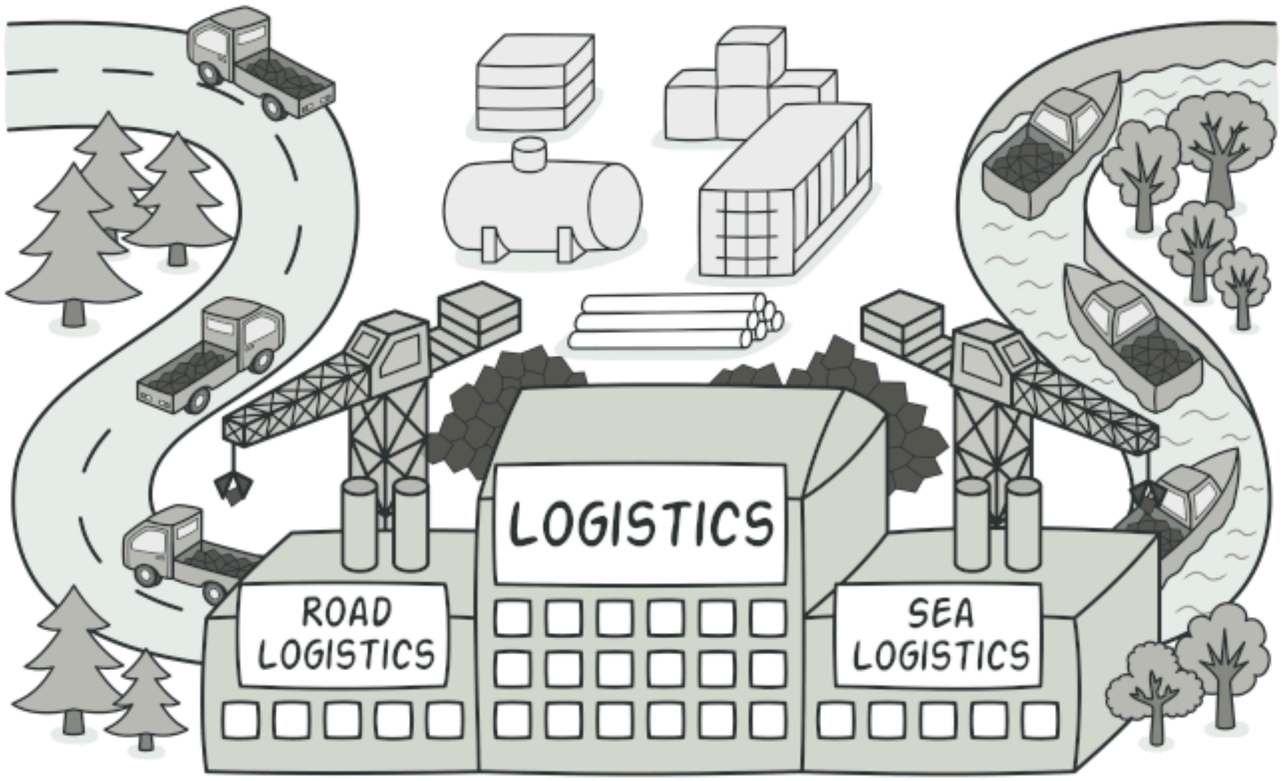


Patterns should adapt

You can’t just find a pattern and copy it into your program, the way you can with off-the-shelf functions or libraries. The pattern is not a specific piece of code, but a general concept for solving a particular problem. You can follow the pattern details and implement a solution that suits the realities of your own program.

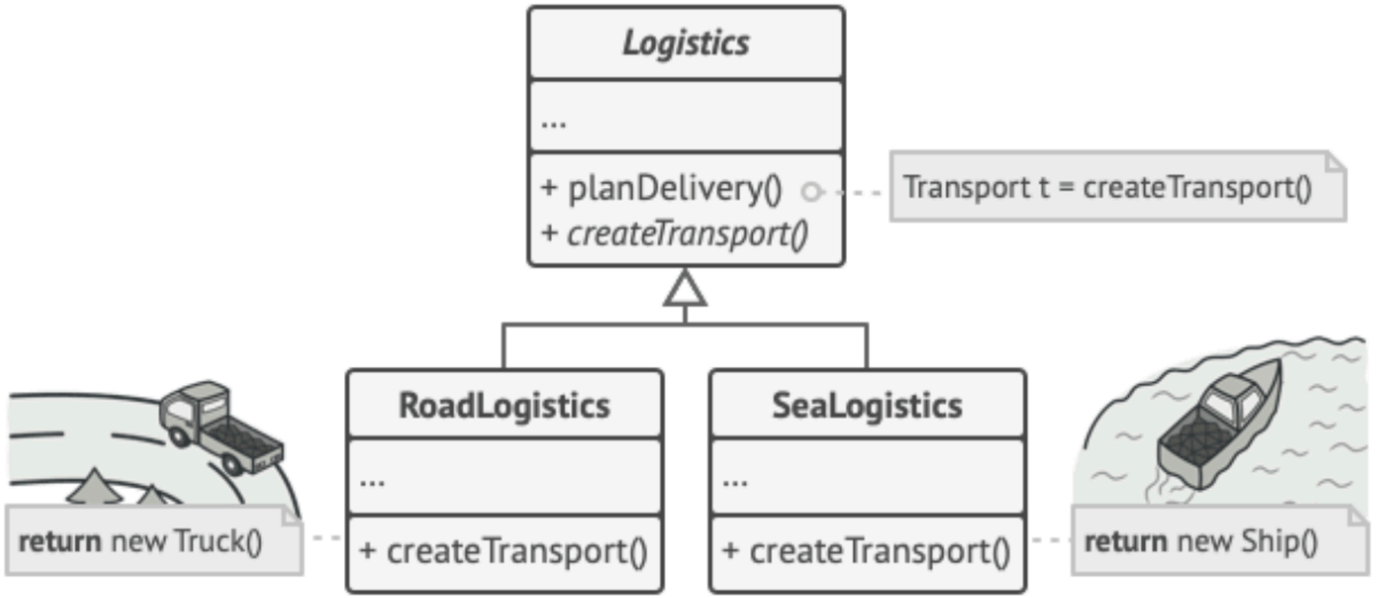
Factory

Factory Method is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.



😊 Solution

The Factory Method pattern suggests that you replace direct object construction calls (using the `new` operator) with calls to a special *factory* method. Don't worry: the objects are still created via the `new` operator, but it's being called from within the factory method. Objects returned by a factory method are often referred to as *products.*



Subclasses can alter the class of objects being returned by the factory method.

Singleton

Web version 9 (modular) | Web version 8 (namespaced) | Swift | Objective-C | Kotlin+KTX Android | Java Android | Dart Flutter | Go | More

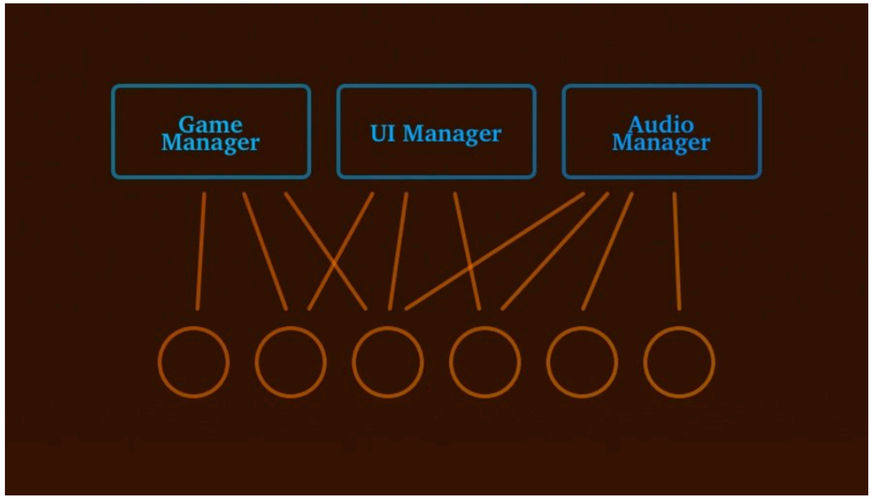
```java
// Access a Cloud Firestore instance from your Activity


FirebaseFirestore db = FirebaseFirestore.getInstance();
```

DocSnippets.java

Why use a singleton?

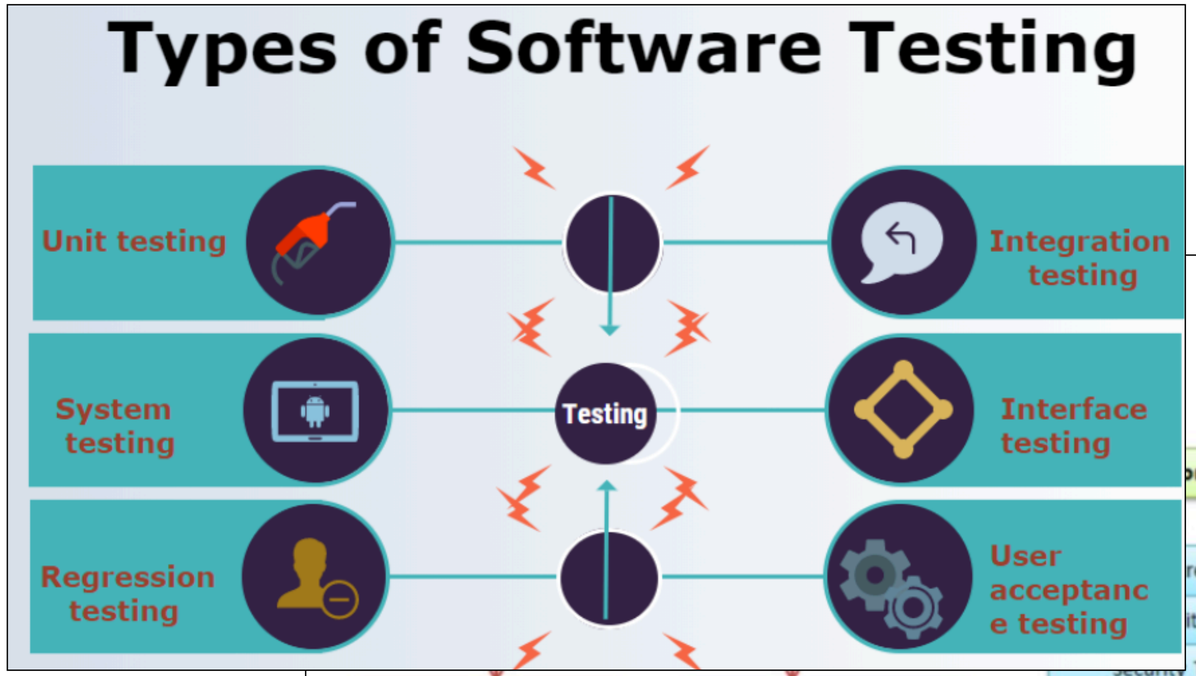
Singletons can be very convenient because they allow you to connect parts of your game more easily.

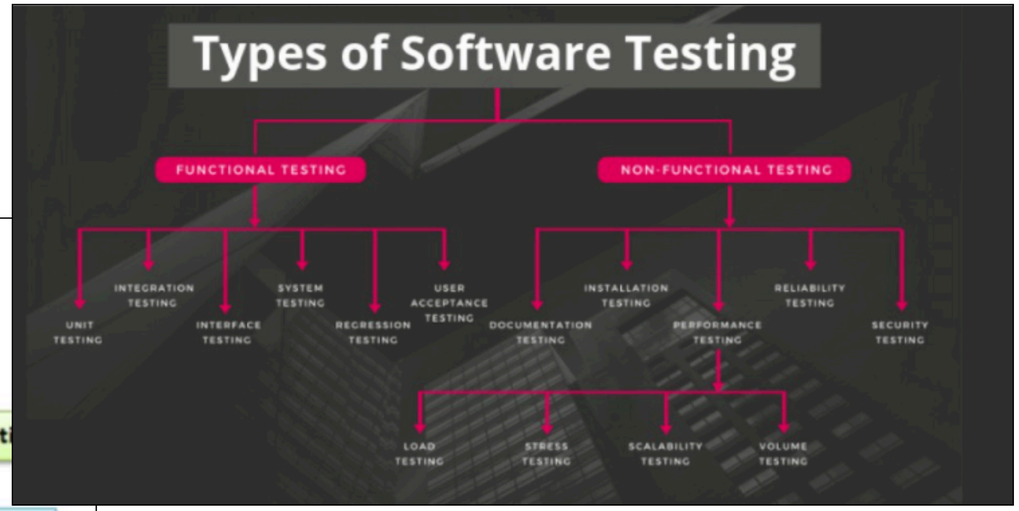
For example, you could use a singleton to make the player's position publicly accessible to enemies, or use one to expose the player's health to the UI.

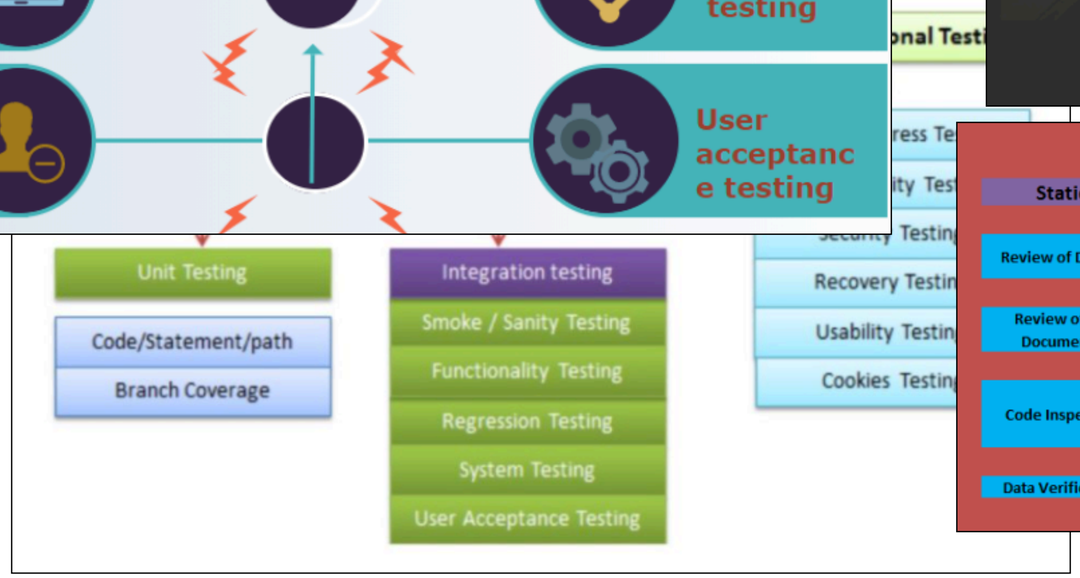


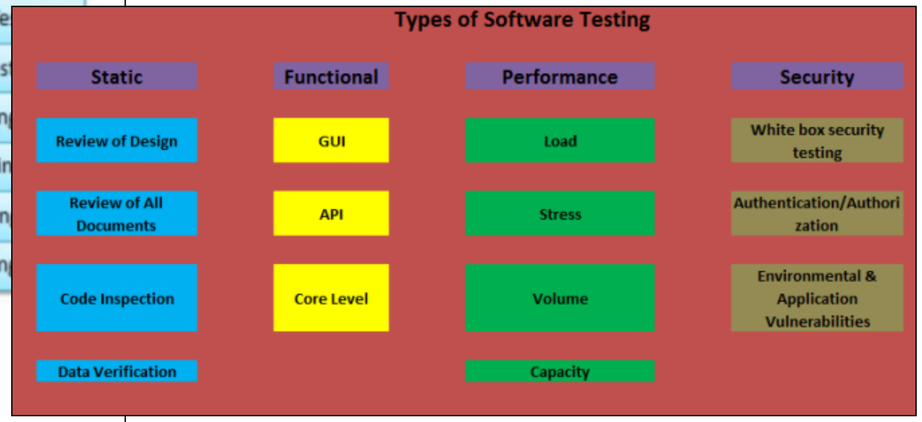
Testing / Quality Assurance (QA)

Software testing



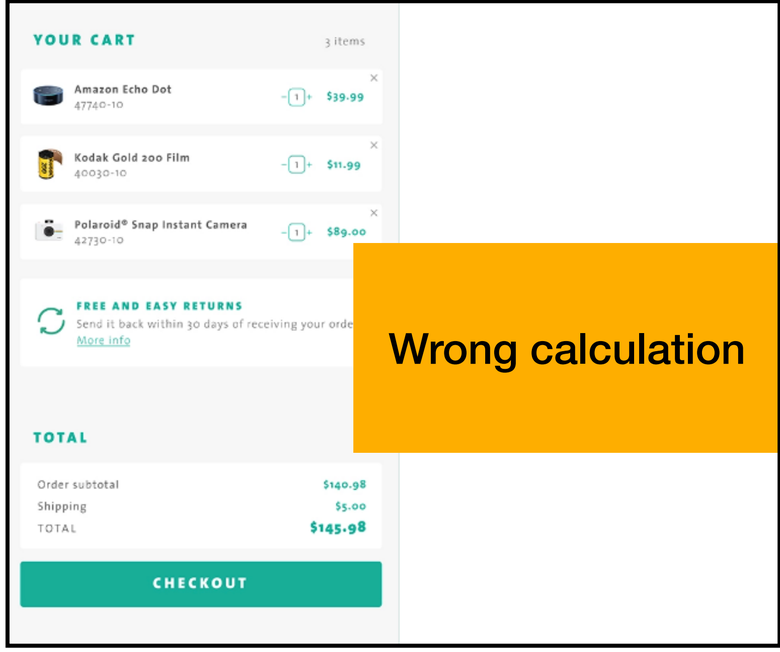






Summation

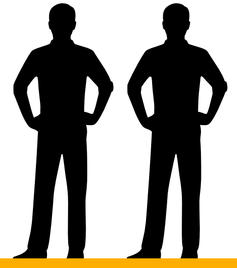
OnClick Event



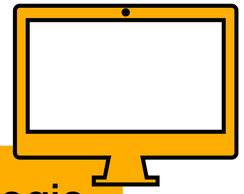
Wrong calculation

Persistance

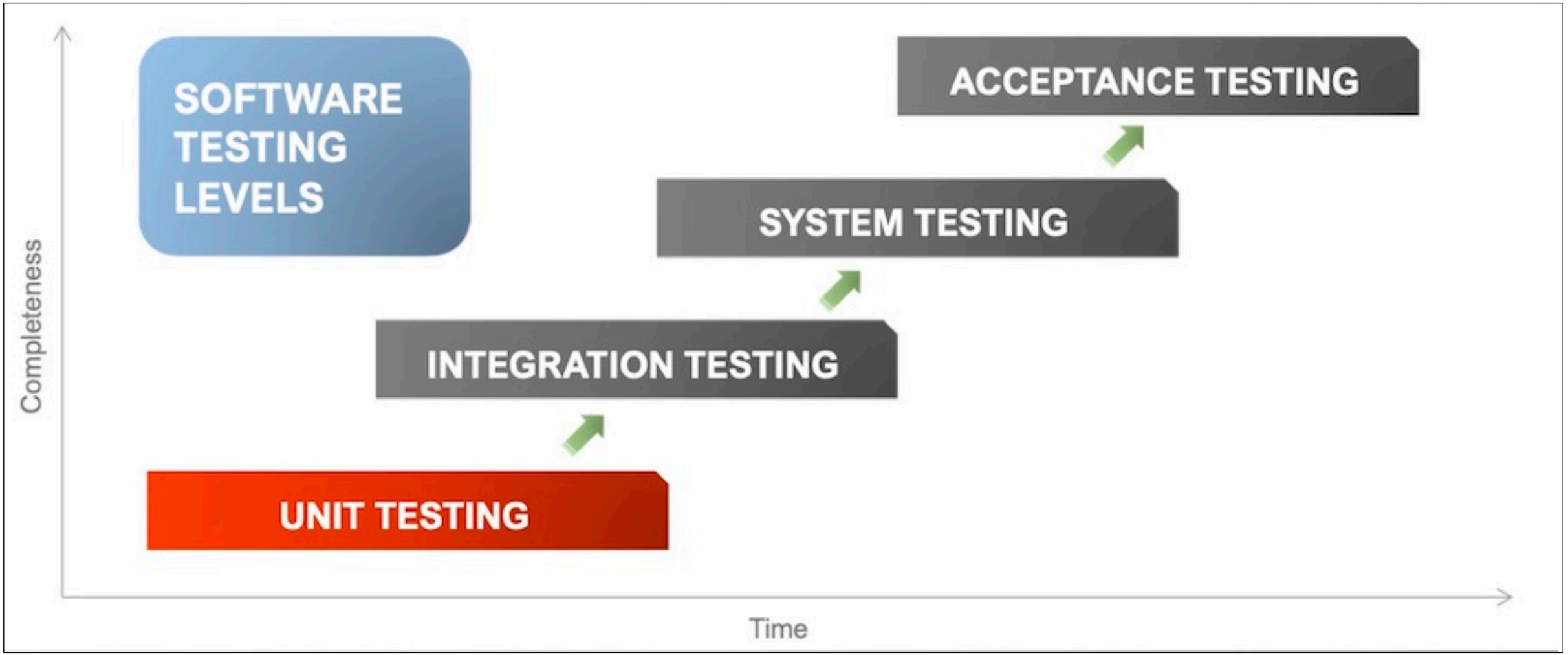
List updated

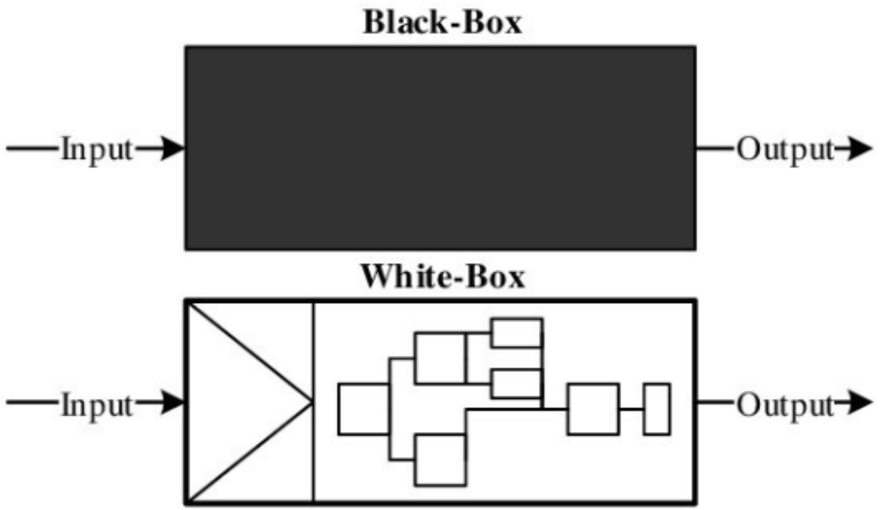


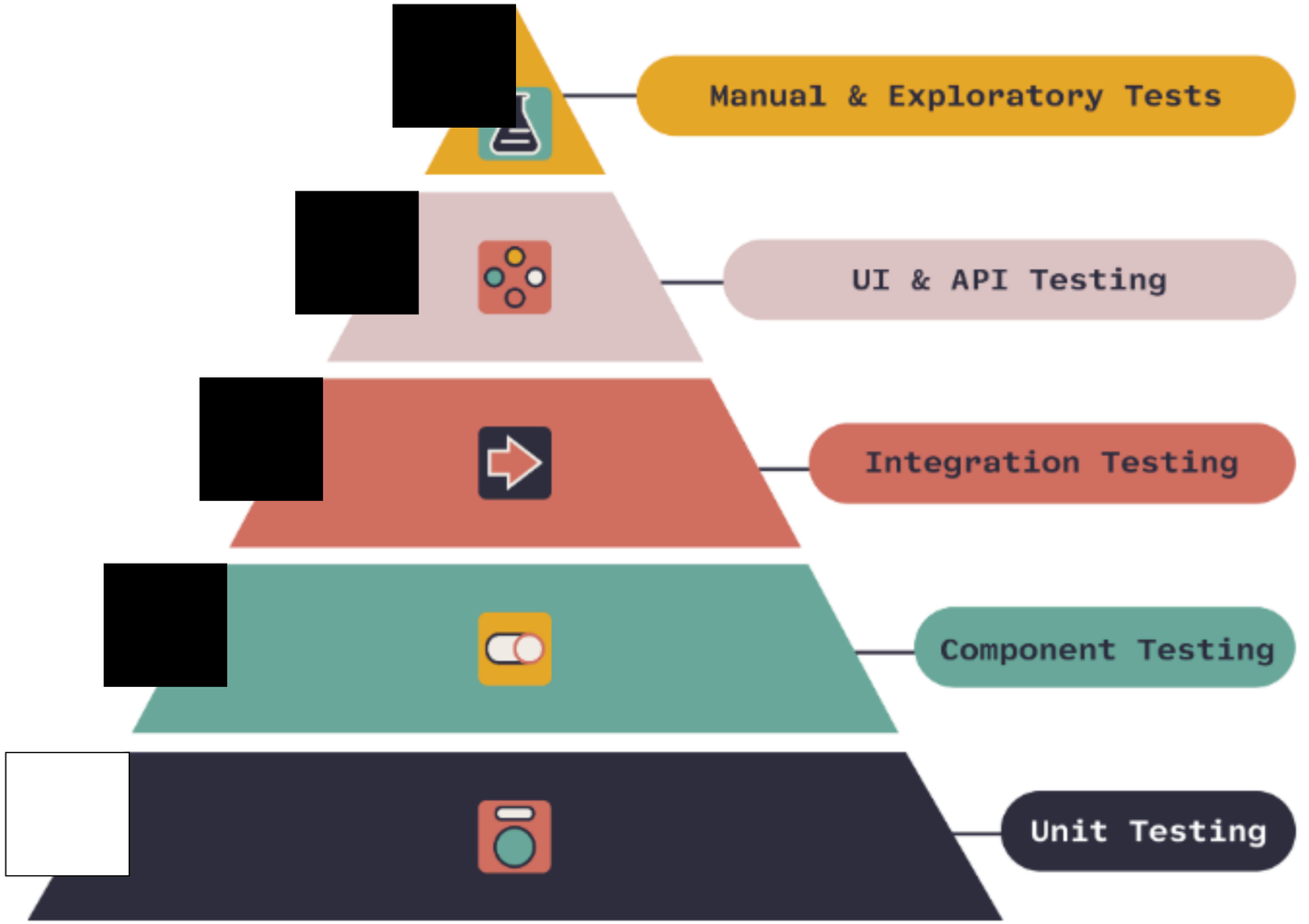
Session/user logic



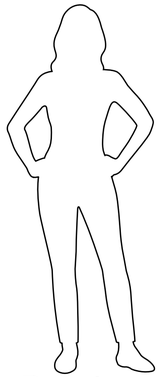
Render logic


SOFTWARE TESTING LEVELS
ACCEPTANCE TESTING
SYSTEM TESTING
INTEGRATION TESTING
UNIT TESTING
Completeness
Time


Black-Box
Input
Output
White-Box
Input
Output


Manual & Exploratory Tests
UI & API Testing
Integration Testing
Component Testing
Unit Testing

Unit test

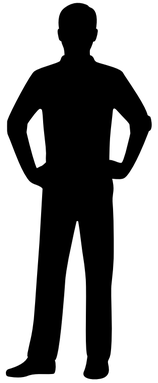


Developer

Writes code

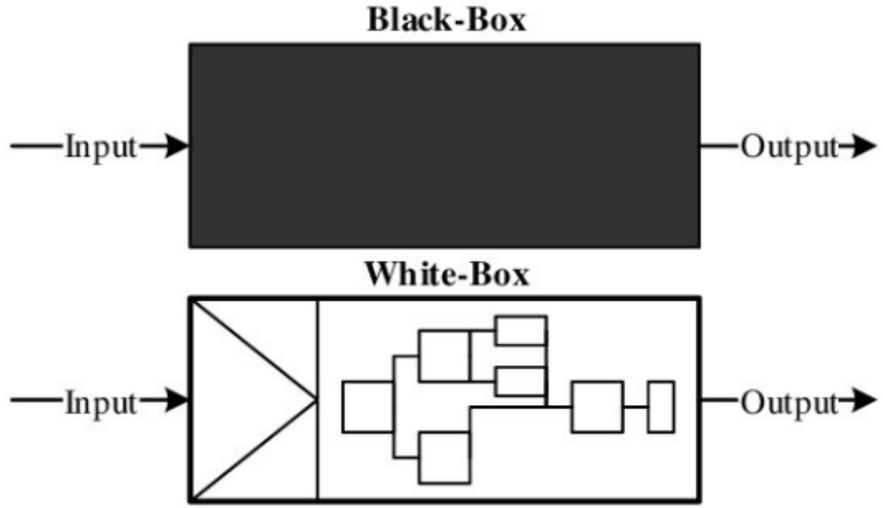
Writes unit tests

UI test

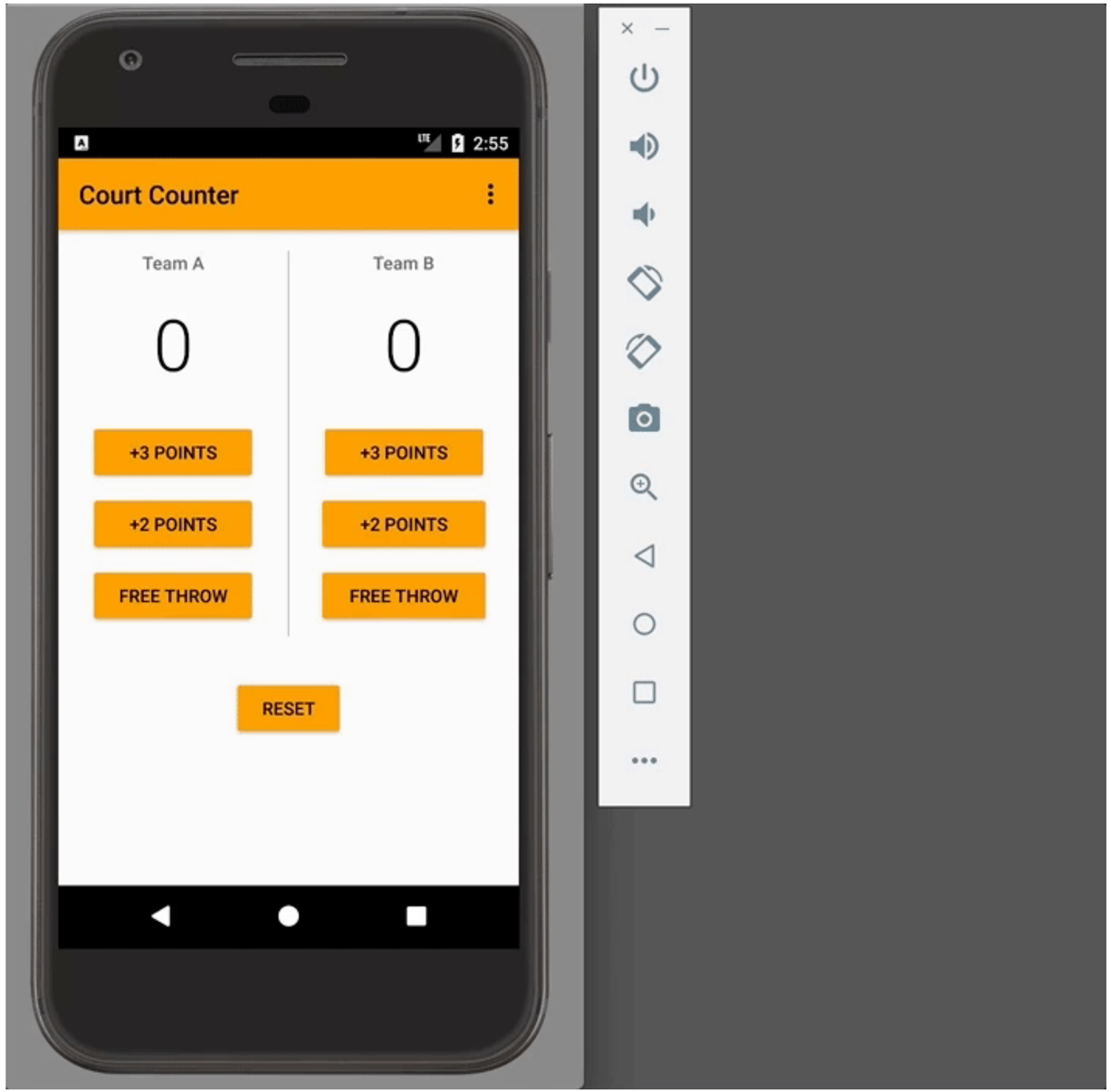


Tester

Tests UI from Requirements



Why?


2:55
Court Counter
Team A
Team B
0
0
+3 POINTS
+2 POINTS
FREE THROW
+3 POINTS
+2 POINTS
FREE THROW
RESET

Objective

1. Use espresso to verify that the UI works in portrait mode

2. Use espresso to verify that the UI has inconsistencies when changing orientation

3. (Advanced) Address & solve the issue - what is the problem?

https://github.com/nicklasdean/court-counter

Developer mode & Example

Rotating a device using code