

# Design Patterns

MSc in Computer Science

---

Produced  
by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics  
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE



# Patterns Labs

---

### Lab-01



Prepare an suitable version of Eclipse for the forthcoming labs. Download and become familiar with the pacemaker-console project. Explore the Strategy pattern in this context.

### Lab-02



Review the Java variants of the template method and strategy patterns. Reode these in Xtend. Reimplement Strategy using Lambdas in Xtend

### Lab-03



```
Welcome to pacemaker-console - ?help for
pm> cu a a a a
+-----+-----+-----+-----+
| ID | FIRSTNAME | LASTNAME | EMAIL | PAS
+-----+-----+-----+-----+
| 1 | a | a | a |
+-----+-----+-----+-----+

pm> cu b b b b
+-----+-----+-----+-----+
| ID | FIRSTNAME | LASTNAME | EMAIL | PAS
+-----+-----+-----+-----+
| 2 | b | b | b |
+-----+-----+-----+-----+
```

Rework the cliché library, using Strategy to delegate command processing. Implement the Command pattern into a simplified version of Pacemaker. Extend the implementation to include undo/redo capability

### lab-03b



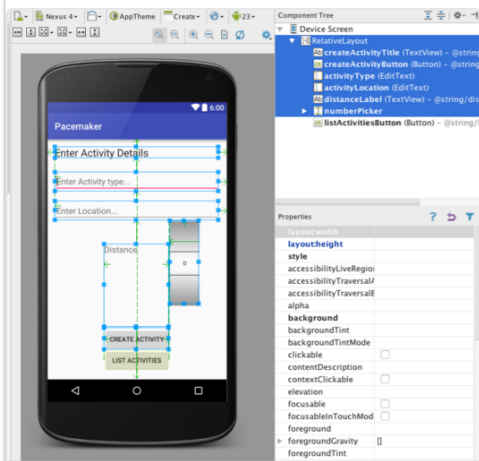
```
public Command copy()
{
    CreateUserCommand command = new CreateUserComm
    command.ser = user;
    return command;
}

and also in the DeleteUserCommand:

public Command copy()
{
    DeleteUserCommand command = new DeleteUserComm
    command.ser = user;
    return command;
}
```

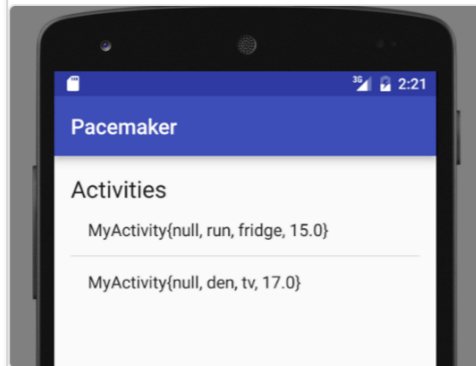
Explore some deficiencies in the pacemaker command pattern implementation. Introduce prototype into the pacemaker application to fix these issues.

### Lab-05a



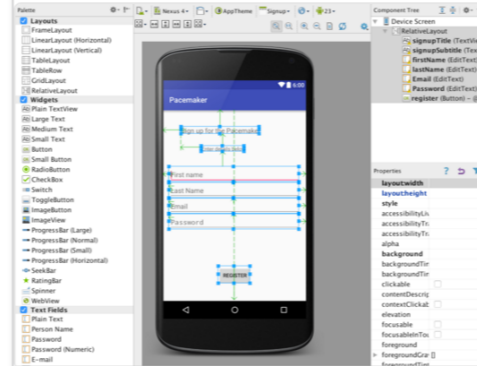
Layout a new Android project with a single activity. Design this activity to permit simple sports activities to be specified. Implement the Activity class to support these controls

### Lab-05b



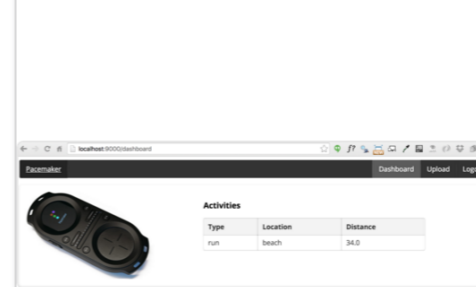
Extend the pacemaker-android app to enable activities to be listed. Explore three patterns in this context: Memento, Singleton, Adapter

### Lab-06a



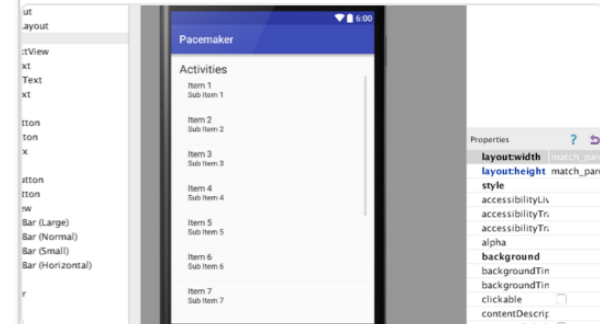
Build a set of views to support signup/login and navigation. Incorporate suitable models. Introduce the Facade pattern to encapsulate Model access.

### Lab-06b

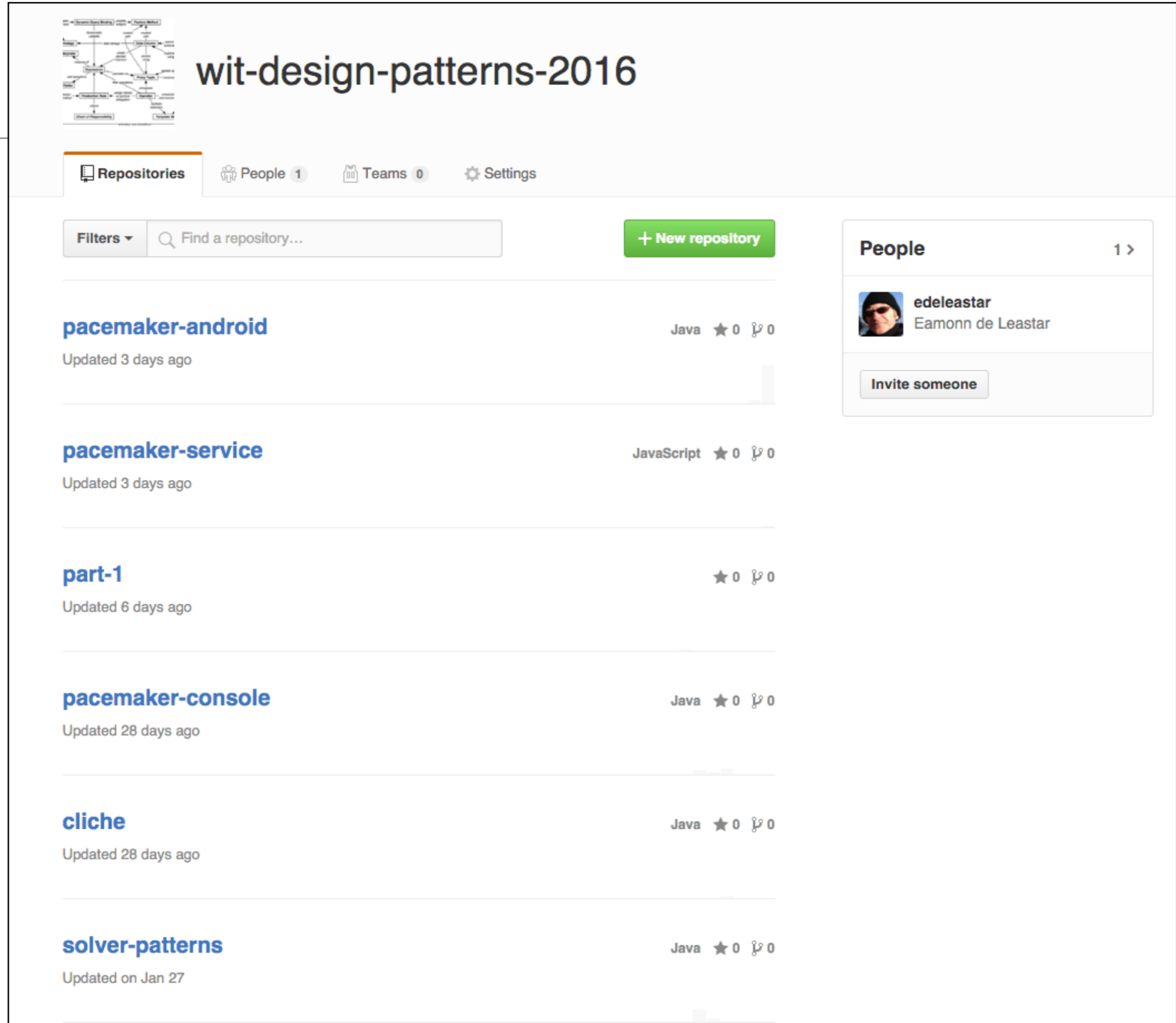


Extend the pacemaker-service application to incorporate service layer. Utilize the layer in the pacemaker-android to retrieve a list of current users

### Lab-06c



Complete the android application to include activity sync support using the Half Sync/Half Async Pattern. Overlay this with a Mediator implementation.



The screenshot shows the GitHub interface for the repository 'wit-design-patterns-2016'. At the top left is a repository logo depicting a complex network diagram. The repository name is prominently displayed in the center. Below the name, navigation tabs include 'Repositories', 'People 1', 'Teams 0', and 'Settings'. A search bar with the placeholder 'Find a repository...' and a '+ New repository' button are also visible. On the right side, a 'People' section features a profile for 'edelestar' (Eamonn de Leastar) with an 'Invite someone' button. The main content area lists six repositories, each with its name, language, star count, and fork count, along with the last update time.

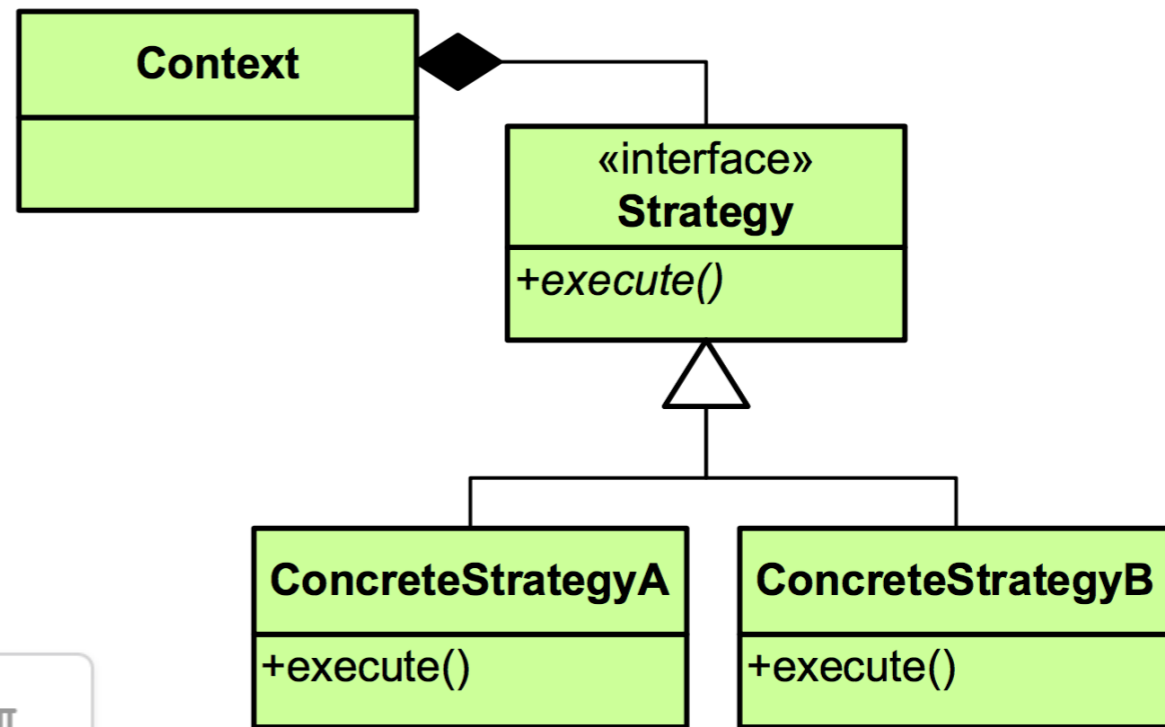
Repository Name	Language	Stars	Forks	Last Updated
pacemaker-android	Java	0	0	Updated 3 days ago
pacemaker-service	JavaScript	0	0	Updated 3 days ago
part-1		0	0	Updated 6 days ago
pacemaker-console	Java	0	0	Updated 28 days ago
cliche	Java	0	0	Updated 28 days ago
solver-patterns	Java	0	0	Updated on Jan 27

# Strategy

**Type:** Behavioral

**What it is:**

Define a family of algorithms, encapsulate each one, and make them interchangeable. Lets the algorithm vary independently from clients that use it.



## Lab-01



Prepare an suitable version of Eclipse for the forthcoming labs. Download and become familiar with the pacemaker-console project. Explore the Strategy pattern in this context.

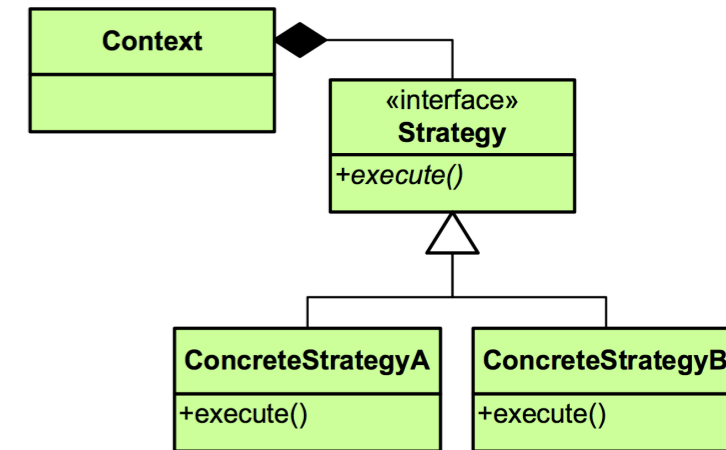
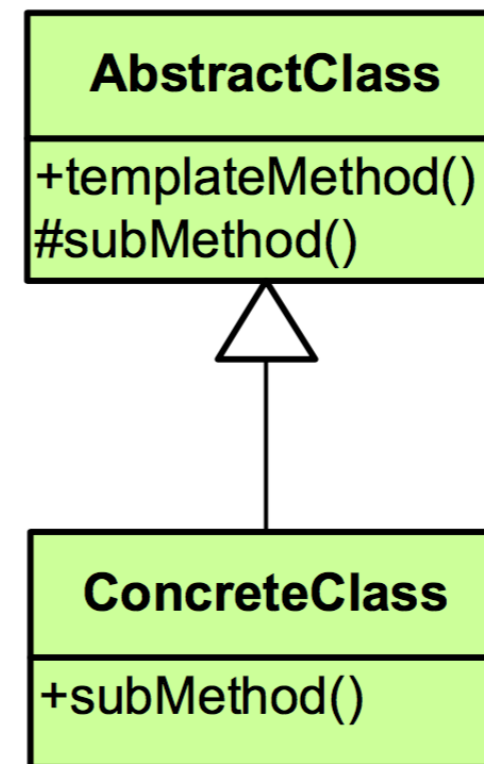
# Template Method

**Type:** Behavioral

## What it is:

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.

Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



Lab-02



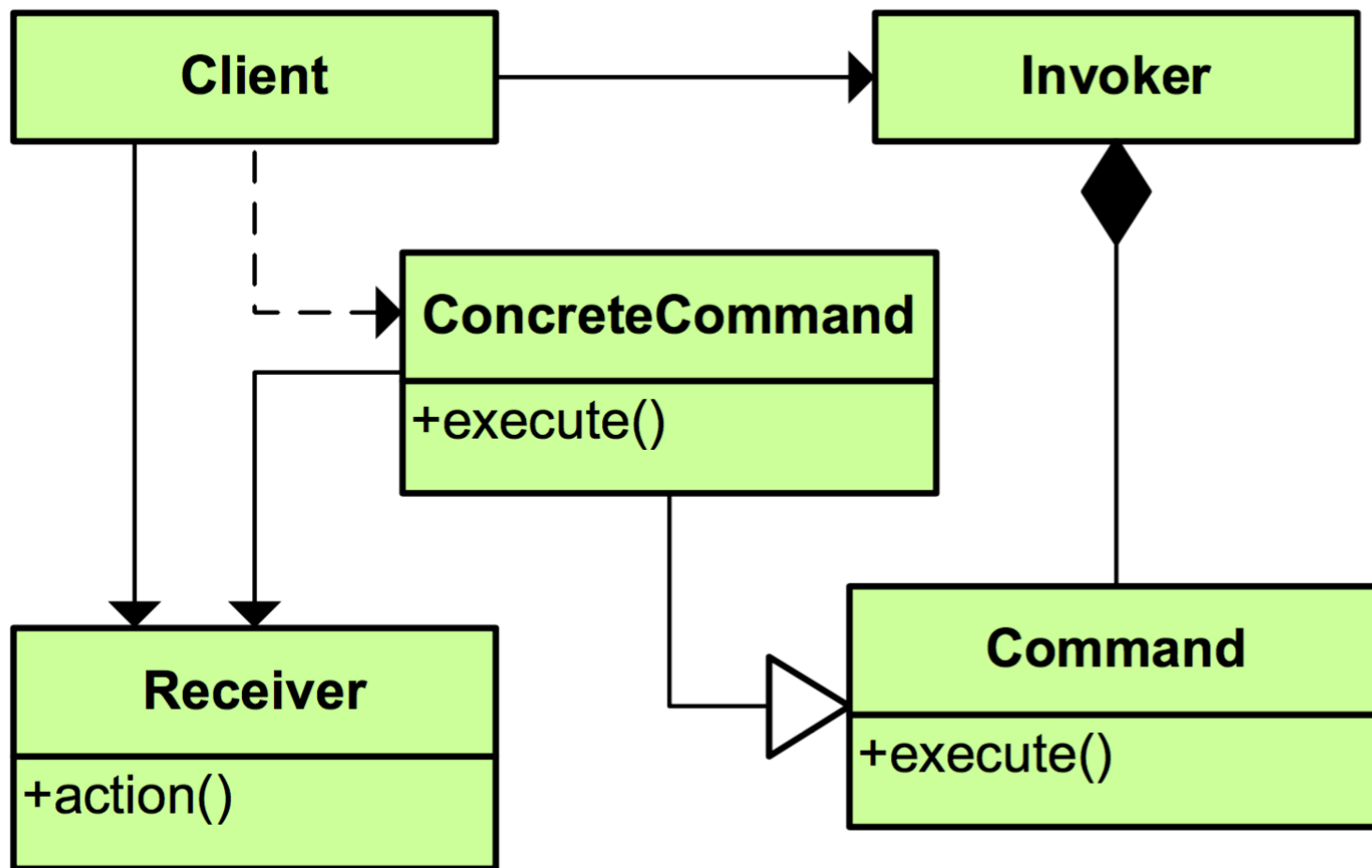
Xtend News Download Document

## JAVA 10, TODAY!

Review the Java variants of the template method and strategy patterns. Reoode these in Xtend. Reimplement Strategy using Lambdas in Xtend

<https://github.com/wit-design-patterns-2016/solver-patterns>

```
14 lambda examples
13 the lambda algoirthms + test (java 8)
12 java 8 findminima interface + solver
11 and extra test exploring the SAM feature of xtend
10 test for xtend lambda strategies
9 version of strategy using xtend lambdas
8 xtend strategy tests
7 strategy in xtend classes
6 test for xtend template method
5 extend version of template method + project configuration changes
4 strategy pattern tests
3 strategy pattern classes
2 template method test
1 template method example classes
```



# Command

Type: Behavioral

## What it is:

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations

<https://github.com/wit-design-patterns-2016/pacemaker-console>

```

11 Lab03a, origin/master) undo command support
10 simplified PacemakerShell class to use command pattern
9  command dispatcher + specification helper
8  initial command interface + commands
7  renamed dependent project
6  refactored to PacemakerShell to new main package. Adjust visibility (
5  refactor to use https://github.com/budhash/cliche.git (as an eclipse
4  (tag: Lab01) serialiser used in pacemaker service + command shell
3  JsonSerializer implementation
2  Serialiser strategy interface factored out from XMLSerializer
1  first version inherited from agile labs
  
```

## Lab-03



```

Welcome to pacemaker-console - ?help for
pm> cu a a a a
+-----+-----+-----+-----+
| ID | FIRSTNAME | LASTNAME | EMAIL | PAS
+-----+-----+-----+-----+
| 1 |          a |          a |          a |
+-----+-----+-----+-----+

pm> cu b b b b
+-----+-----+-----+-----+
| ID | FIRSTNAME | LASTNAME | EMAIL | PAS
+-----+-----+-----+-----+
| 2 |          b |          b |          b |
+-----+-----+-----+-----+
  
```

Rework the cliché library, using Strategy to delegate command processing. Implement the Command pattern into a simplified version of Pacemaker. Extend the implementation to include undo/redo capability

# Prototype

Type: Creational

## What it is:

Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

lab-03b

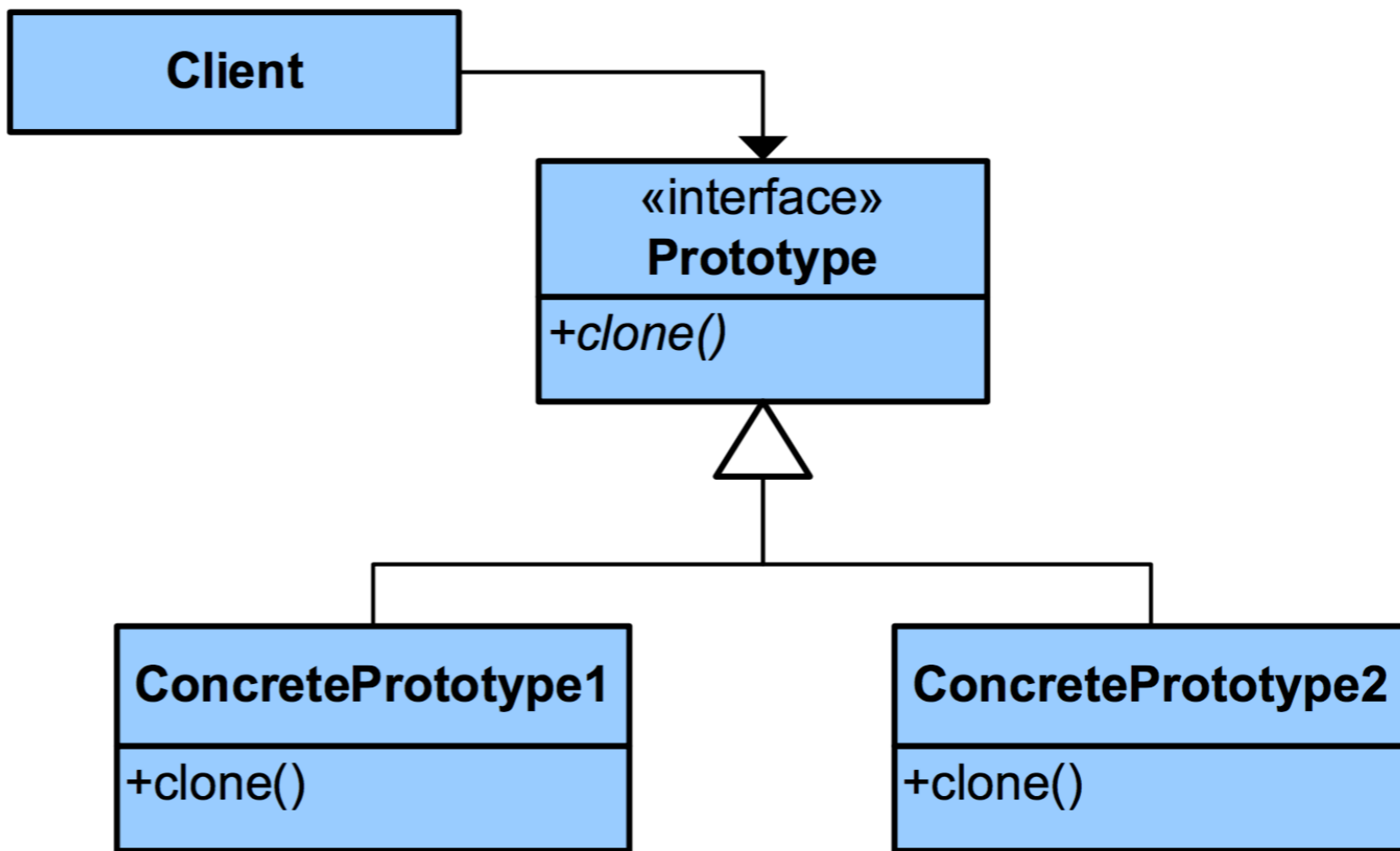


```
public Command copy()  
{  
    CreateUserCommand command = new CreateUserComm  
    command.ser = user;  
    return command;  
}
```

and also in the DeleteUserCommand:

```
public Command copy()  
{  
    DeleteUserCommand command = new DeleteUserComm  
    command.ser = user;  
    return command;  
}
```

Explore some deficiencies in the pacemaker command pattern implementation. Introduce prototype into the pacemaker application to fix these issues.





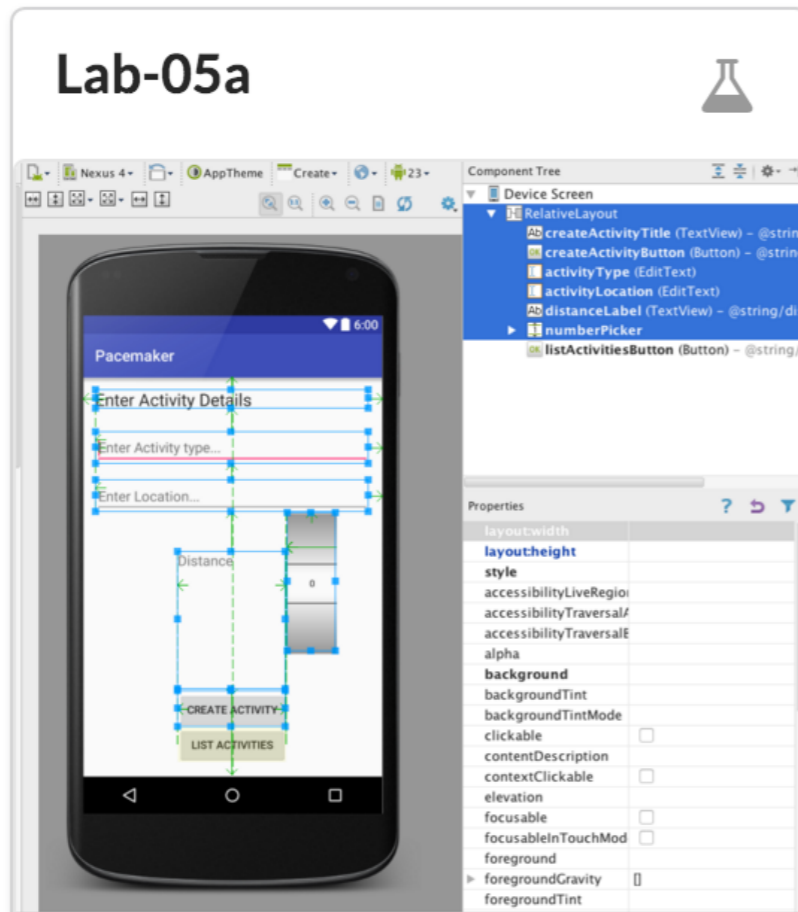
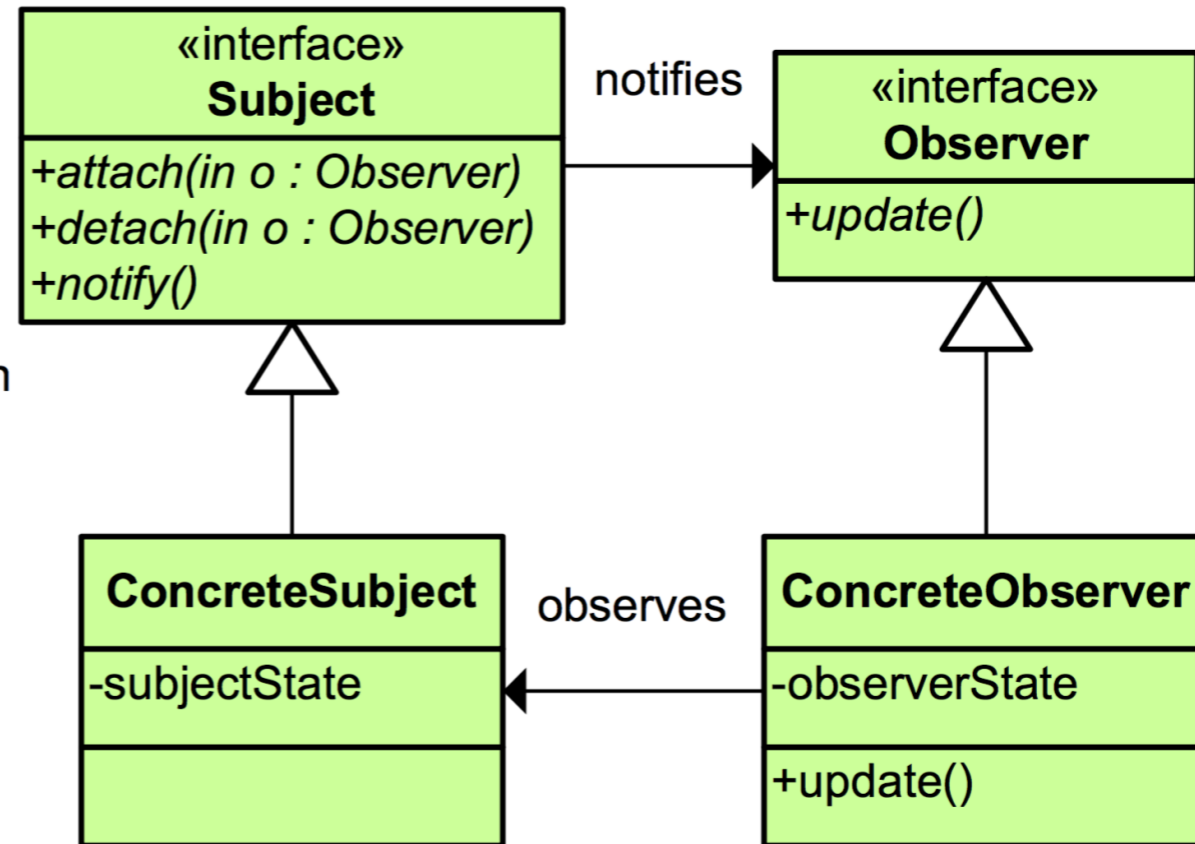
```
37 (tag: V7) adjust app and controllers to use mediator
36 mediator classes
35 (tag: V6) incorporate activities managements into controllers
34 facade now supports activities management
33 extend API to retrieve/upload activities
32 (tag: V5) remove reference to type in user
31 in Welcome, download list of users from pacemaker-play service
30 introduce PacemakerAPI wrapper classes to access service
29 rename user.type to user.kind to avoid json parsers problems
28 enable network access in AndroidManifest
27 JSON parsers for user model objects
26 http wrapper classes for REST access
25 import json + apache libraries into project
24 separate activities by user
23 (tag: V4) Facade pattern to encapsulate model management
22 track user signup + login. Maintain user list in PacemakerApp.
21 Welcome, Signup and Login activities + manifest adjustment
20 User model introduced
19 resources for welcome, signup and login activities
18 (tag: V3) refactored package structure into main, controllers, models
17 custom row for activity list adapter
16 (tag: V2) replace stock adapter with a customised ArrayAdapter
15 replace parcelable implementation with simple singleton access
14 application singleton introduced
13 render the activities with a simple ArrayAdapter
12 log the our to the console in ActivitiesList
11 parcel up the activities in CreateActivity
10 make MyActivity Parcelable
9 (tag: V1) Create MyActivities models objects in CreateActivities View
8 new button on CreateActivities view show activities
7 MyActivity model object introduced
6 added ActivitiesList activity
5 wired up controls in CreateActivity class
4 introduce button widget + event handler
3 placed button on layout
2 first update to layout
1 as generated by android studio 2.0
```

# Observer

Type: Behavioral

## What it is:

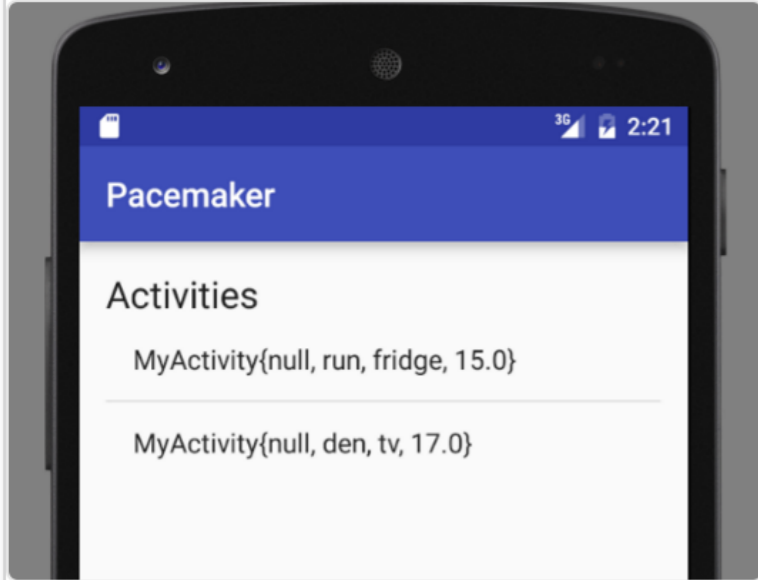
Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



Layout a new Android project with a single activity. Design this activity to permit simple sports activities to be specified. Implement the Activity class to support these controls

- 5 wired up controls in CreateActivity class
- 4 introduce button widget + event handler
- 3 placed button on layout
- 2 first update to layout
- 1 as generated by android studio 2.0

## Lab-05b



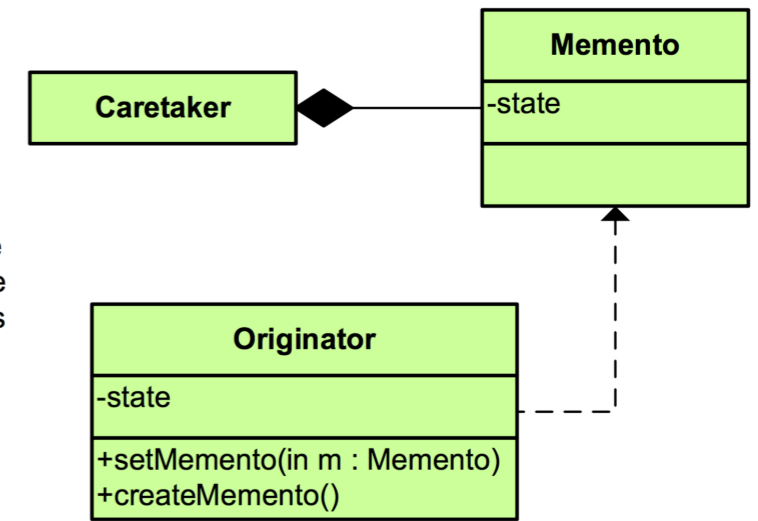
Extend the pacemaker-android app to enable activities to be listed. Explore three patterns in this context: Memento, Singleton, Adapter

## Memento

Type: Behavioral

### What it is:

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

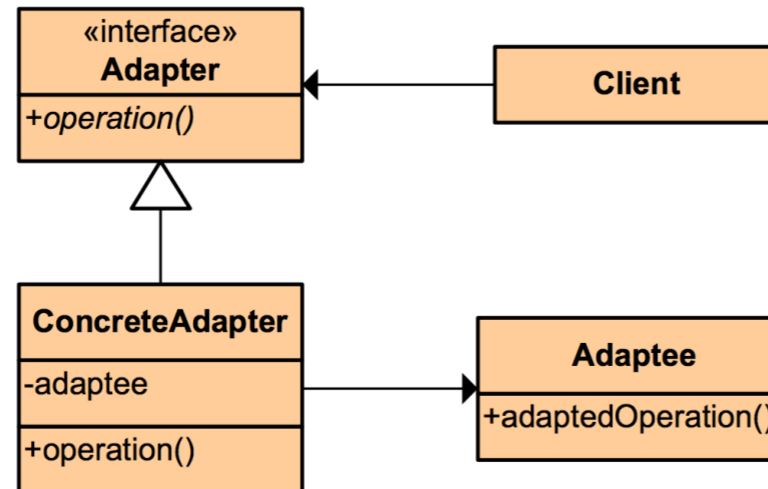


## Adapter

Type: Structural

### What it is:

Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

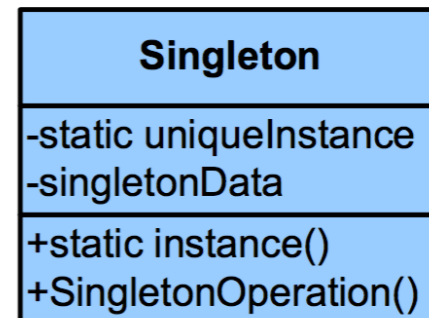


## Singleton

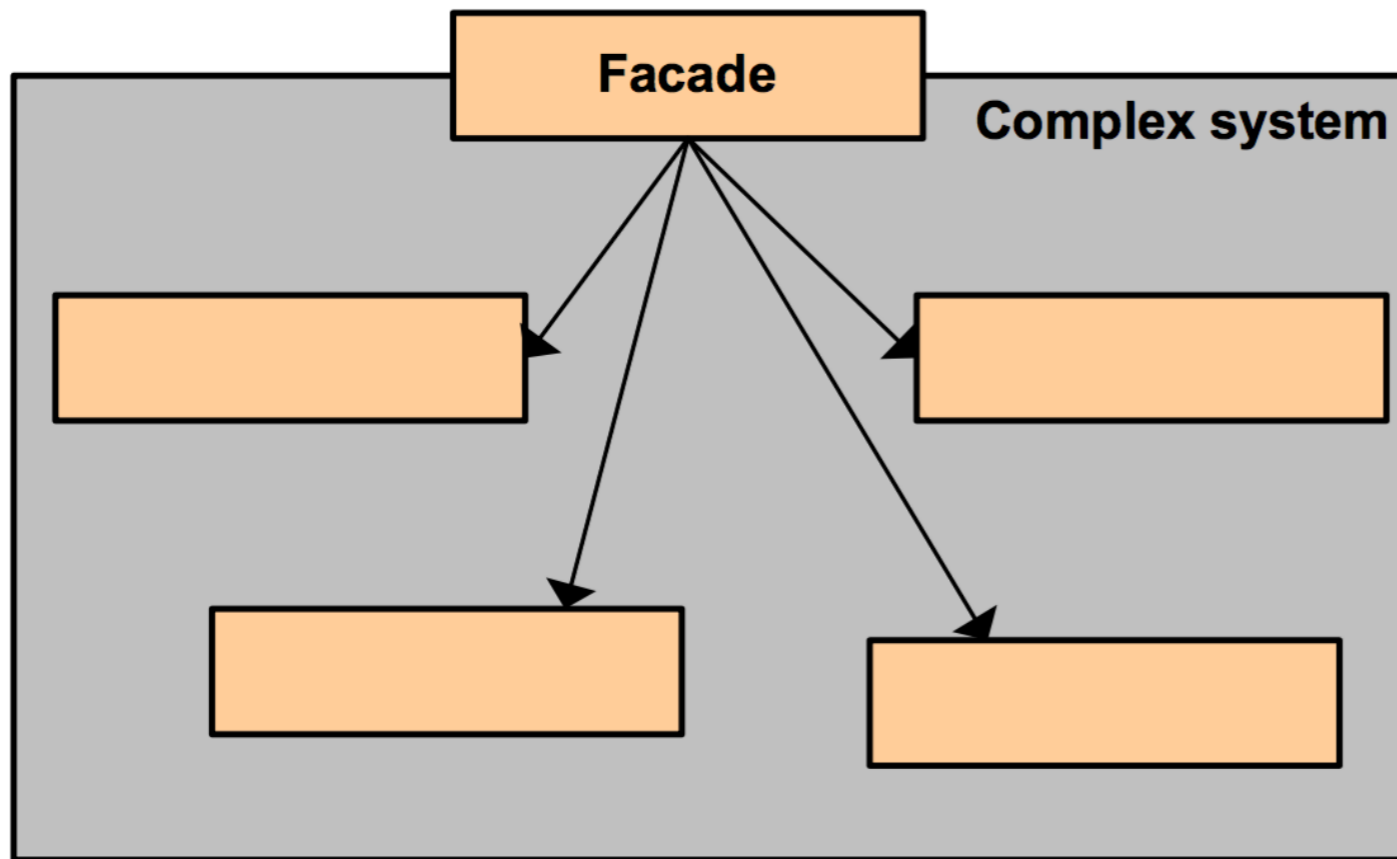
Type: Creational

### What it is:

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



```
16 (tag: V2) replace stock adapter with a customised ArrayAdapter
15 replace parcelable implementation with simple singleton access
14 application singleton introduced
13 render the activities with a simple ArrayAdapter
12 log the our to the console in ActivitiesList
11 parcel up the activities in CreateActivity
10 make MyActivity Parcelable
9 (tag: V1) Create MyActivities models objects in CreateActivities View
8 new button on CreateActivities view show activities
7 MyActivity model object introduced
6 added ActivitiesList activity
```



# Facade

**Type:** Structural

## What it is:

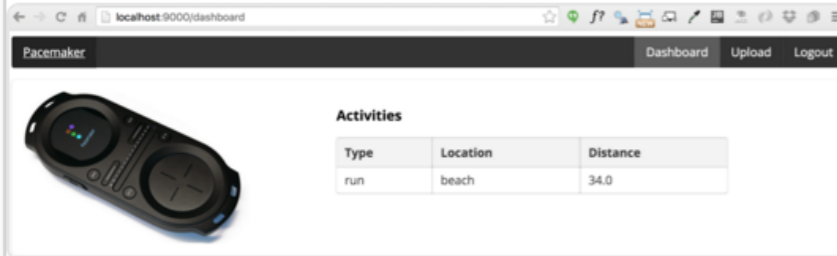
Provide a unified interface to a set of interfaces in a subsystem. Defines a high-level interface that makes the subsystem easier to use.

## Lab-06a

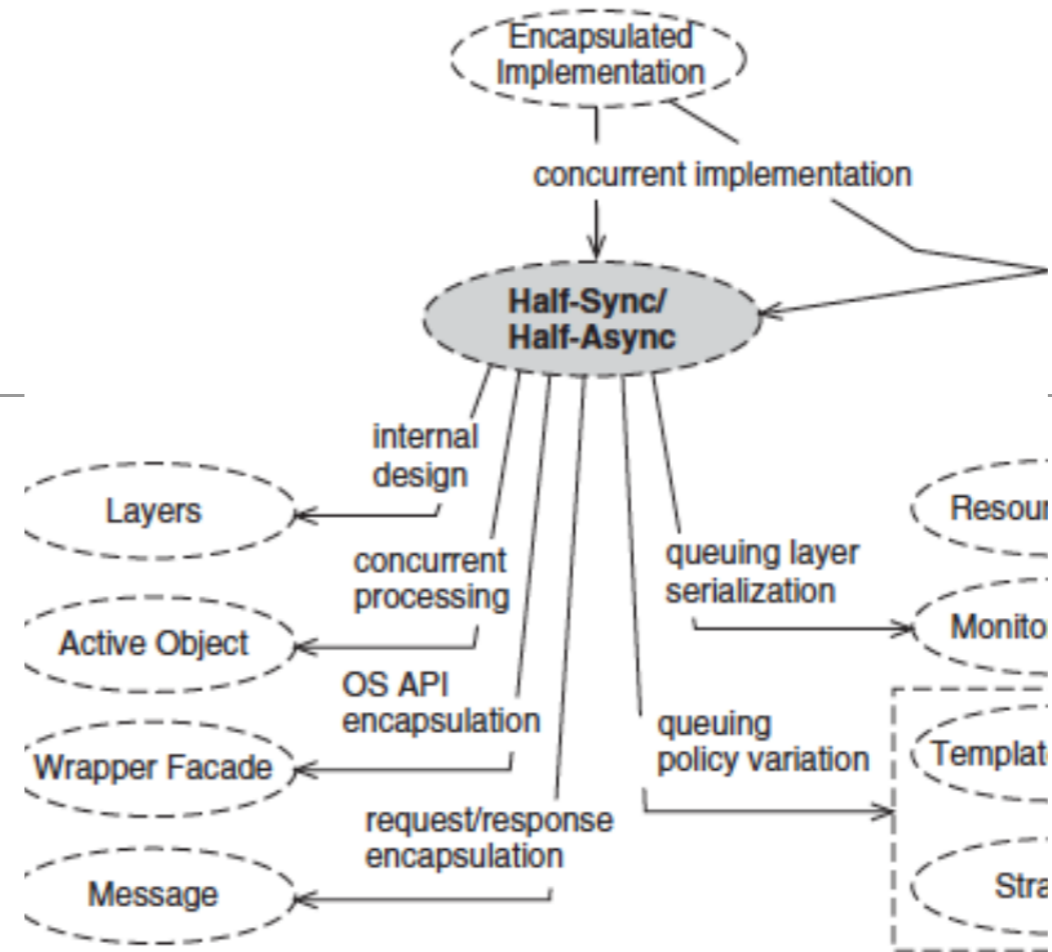
Build a set of views to support signup/login and navigation. Incorporate suitable models. Introduce the Facade pattern to encapsulate Model access.

- 23 **(tag: V4)** Facade pattern to encapsulate model management
- 22 track user signup + login. Maintain user list in PacemakerApp.
- 21 Welcome, Signup and Login activities + manifest adjustment
- 20 User model introduced
- 19 resources for welcome, signup and login activities
- 18 **(tag: V3)** refactored package structure into main, controllers, m
- 17 custom row for activity list adapter

## Lab-06b

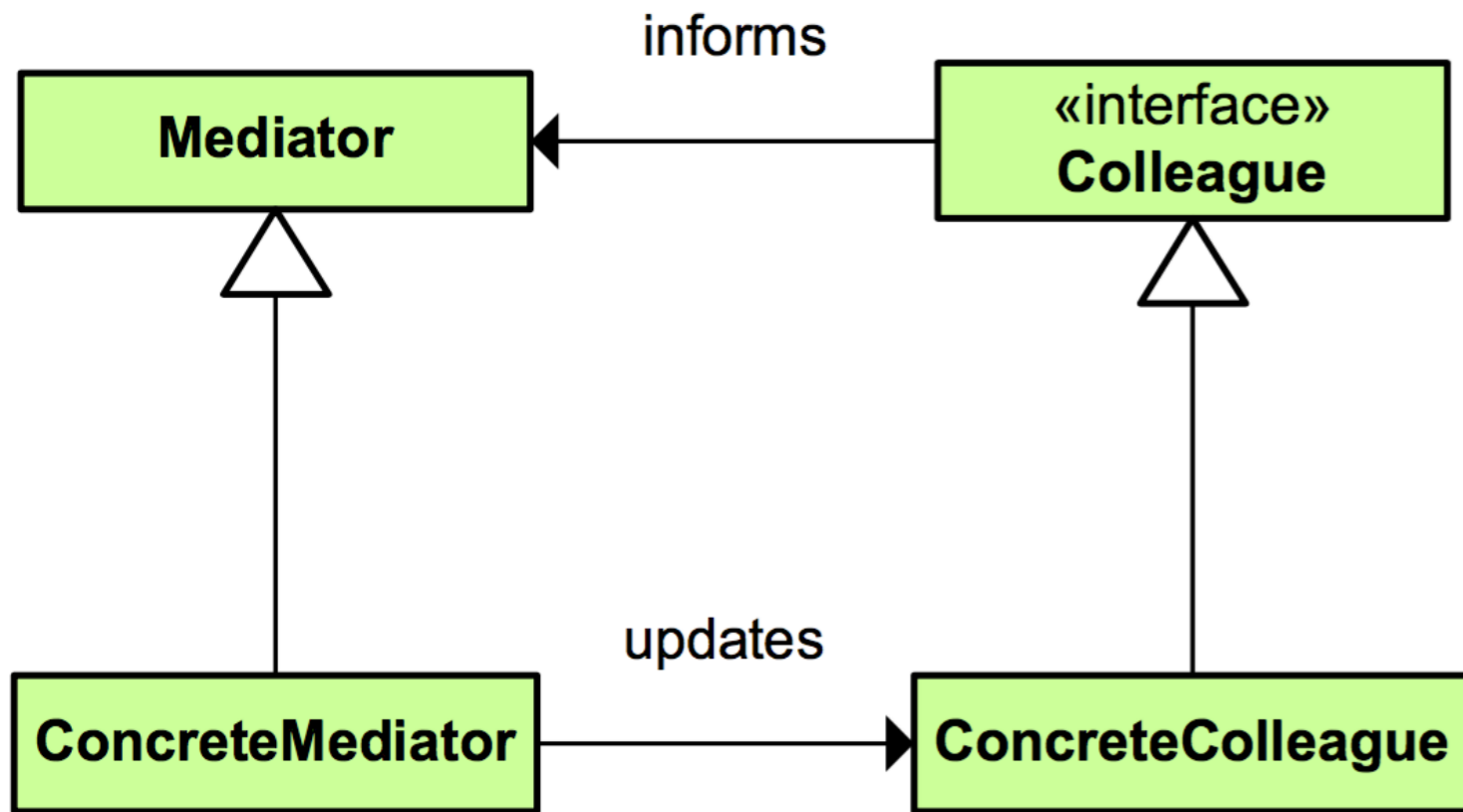


Retrieve a list of users from a remote REST service. Implement this using the Half Sync/Half Async Pattern.



```
32 (tag: V5) remove reference to type in user
31 in Welcome, download list of users from pacemaker-play service
30 introduce PacemakerAPI wrapper classes to access service
29 rename user.type to user.kind to avoid json parsers problems
28 enable network access in AndroidManifest
27 JSON parsers for user model objects
26 http wrapper classes for REST access
25 import json + apache libraries into project
24 separate activities by user
```

+ [wit-design-patterns-2016/pacemaker-service](https://github.com/wit-design-patterns-2016/pacemaker-service)



# Mediator

Type: Behavioral

## What it is:

Define an object that encapsulates how a set of objects interact. Promotes loose coupling by keeping objects from referring to each other explicitly and it lets you vary their interactions independently.

```

37 (tag: V7) adjust app and controllers to use mediator
36 mediator classes
35 (tag: V6) incorporate activities managements into controll
34 facade now supports activities management
33 extend API to retrieve/upload activities
  
```

## Lab-06c

Properties

- layout:width match\_parent
- layout:height match\_parent
- style
- accessibilityLis
- accessibilityTr
- accessibilityTr
- alpha
- background
- backgroundTin
- backgroundTin
- clickable
- contentDescrip

Complete the android application to include activity sync support using the Half Sync/Half Async Pattern. Overlay this with a Mediator implementation.

# pacemaker-android Releases

The screenshot shows the GitHub releases page for the repository `wit-design-patterns-2016 / pacemaker-android`. The page is in the `Releases` tab, showing a vertical timeline of seven releases from V1 to V7. Each release includes a commit hash, a download icon, and links for `zip` and `tar.gz` files. On the left side of the image, several lab names are listed with arrows pointing to specific releases on the timeline:

- Lab06c points to V7 (commit: 54784e6)
- Lab06b points to V6 (commit: 5333a26)
- Lab06a points to V4 (commit: e5bbae7)
- Lab05b points to V3 (commit: 820d46f)
- Lab05a points to V1 (commit: d82138d)

Lab Name	Release	Commit Hash	Download Links
Lab06c	V7	54784e6	zip, tar.gz
Lab06b	V6	5333a26	zip, tar.gz
Lab06a	V4	e5bbae7	zip, tar.gz
Lab05b	V3	820d46f	zip, tar.gz
Lab05a	V1	d82138d	zip, tar.gz

# Resources for Further Development (Project)

---

- **GoF Patterns**

- Common Design Patterns for Android
- Overview & Mapping of GoF design patterns with Android API's

- **Factory Patterns / Dependency Injection**

- Butterknife
- Dagger

- **Frameworks**

- Retrofit
- Realm

- **Architectural Patterns**

- ReactiveX
- Clean Architecture



# Common Design Patterns for Android

<http://www.raywenderlich.com/109843/common-design-patterns-for-android>

- Useful outline of selected GoF patterns in Android context

## Common Design Patterns for Android



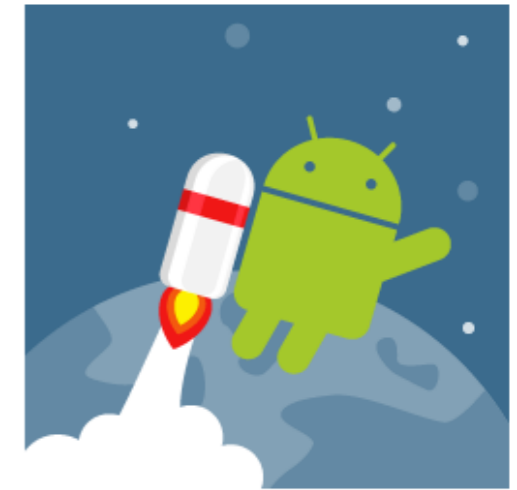
Matt Luedke on December 15, 2015



Beyond satisfying your clients and your employer, there's one more important individual to keep happy in your career as a developer: Future You! (The artist's conception of Future You to the right implies no guarantee of personal jetpack availability for developers in the near future.) :]

Future You will inherit the code you write at some point down the road, and will likely have a lot of questions about how and why you coded things the way you did. But instead of leaving tons of confusing comments in your code, a much better approach is to adopt common **design patterns**.

This article will introduce a few common design patterns for Android that you can use while developing apps. Design patterns are reusable solutions to common software problems. The design patterns covered here aren't an exhaustive list, nor an academically-citable paper. Rather, they serve as a workable references and starting points for further investigation.



"Future You"

### Getting Started

*"Is there anywhere in this project where I'll have to change the same thing in multiple places?" – Future You*

Future You should minimize time spent doing "detective work" looking for intricate project dependencies, so they would prefer a project that's as reusable, readable, and recognizable as possible. These goals span a single object all the way up to the entire project and lead to patterns that fall into the following categories:

- **Creational patterns:** how you *create* objects.
- **Structural patterns:** how you *compose* objects.
- **Behavioral patterns:** how you *coordinate* object interactions.

You may already be using one or several of these patterns already without having A Capitalized Fancy Name for it, but Future You will appreciate you not leaving design decisions up to intuition alone.

In the sections that follow, you'll cover the following patterns from each category and see how they apply to Android:

#### Creational

# Overview & Mapping of GoF design patterns with Android API's

<http://vardhan-justlikethat.blogspot.ie/2013/10/mapping-gof-design-patterns-with.html>

- Identifies and names Patterns in the Android SDK

Patterns	Definition	Android
<b>Creational</b>		
Singleton	<p>Ensure a class has only one instance and provide a global point of access to it.</p>	Application Class (AndroidManifest.xml's tag)
Abstract Factory	Provides an interface for creating families of related or dependent objects without specifying their concrete classes.	<p>Interface ComponentCallbacks (The set of callback APIs that are common to all application components Activity, Service, ContentProvider, and Application)</p> <p>Activity, Service, ContentProvider, AbstractAccountAuthenticator, ActionBar.Tab, ....</p>
Factory Method	<p>Define an interface for creating an object, but let the subclasses decide which class to instantiate. The Factory method lets a class defer instantiation to subclasses</p> <p>The Factory method works just the same way: it defines an interface for creating an object, but leaves the choice of its type to the subclasses, creation being deferred at run-time.</p>	<p>We can relate this design pattern with multi pane layout.</p> <p>It's like choosing which view to be loaded whether handset or tablet views.</p>
Builder	Construct a complex object from simple objects step by step	StringBuilder
<b>Structural</b>		
Adapter (Wrapper pattern / Decorator pattern)	<p>Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.</p> <p>(Adapter design pattern is used when you want two different classes with incompatible interfaces to work together)</p>	ListView, GridView, Spinner and Gallery for commonly used subclasses of AdapterView.

# Butterknife

<http://jakewharton.github.io/butterknife/>

- Simplify Resource Binding



## Butter Knife

Field and method binding for Android views

Javadoc · StackOverflow

Fork me on GitHub

### Introduction

Annotate fields with `@Bind` and a view ID for Butter Knife to find and automatically cast the corresponding view in your layout.

```
class ExampleActivity extends Activity {
    @Bind(R.id.title) TextView title;
    @Bind(R.id.subtitle) TextView subtitle;
    @Bind(R.id.footer) TextView footer;
```

```
@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.simple_activity);
    ButterKnife.bind(this);
    // TODO Use fields...
}
```

Instead of slow reflection, code is generated to perform the view look-ups. Calling `bind` delegates to this generated code that you can see and debug.

The generated code for the above example is roughly equivalent to the following:

```
public void bind(ExampleActivity activity) {
    activity.subtitle = (android.widget.TextView) activity.findViewById(2130968578);
    activity.footer = (android.widget.TextView) activity.findViewById(2130968579);
    activity.title = (android.widget.TextView) activity.findViewById(2130968577);
}
```

### RESOURCE BINDING

Bind pre-defined resources with `@BindBool`, `@BindColor`, `@BindDimen`, `@BindDrawable`, `@BindInt`, `@BindString`, which binds an `R.bool` ID (or your specified type) to its corresponding field.

```
class ExampleActivity extends Activity {
    @BindString(R.string.title) String title;
    @BindDrawable(R.drawable.graphic) Drawable graphic;
    @BindColor(R.color.red) int red; // int or ColorStateList field
    @BindDimen(R.dimen.spacer) Float spacer; // int (for pixel size) or float (for
    // ...
```

# Dagger

<http://square.github.io/dagger/>

# Dagger

Download v1.2.2



## A fast dependency injector for Android and Java

### Introduction

The best classes in any application are the ones that do stuff: the `BarcodeDecoder`, the `KoopaPhysicsEngine`, and the `AudioStreamer`. These classes have dependencies; perhaps a `BarcodeCameraFinder`, `DefaultPhysicsEngine`, and an `HttpStreamer`.

To contrast, the worst classes in any application are the ones that take up space without doing much at all: the `BarcodeDecoderFactory`, the `CameraServiceLoader`, and the `MutableContextWrapper`. These classes are the clumsy duct tape that wires the interesting stuff together.

Dagger is a replacement for these `FactoryFactory` classes. It allows you to focus on the interesting classes. Declare dependencies, specify how to satisfy them, and ship your app.

By building on standard `javax.inject` annotations (JSR-330), each class is easy to test. You don't need a bunch of boilerplate just to swap the `RpcCreditCardService` out for a `FakeCreditCardService`.

Dependency injection isn't just for testing. It also makes it easy to create reusable, interchangeable modules. You can share the same `AuthenticationModule` across all of your apps. And you can run `DevLoggingModule` during development and `ProdLoggingModule` in production to get the right behavior in each situation.

For more information, [watch an introductory talk](#) by Jesse Wilson at QCon 2012.

[Introduction](#)

[Using Dagger](#)

[Download](#)

[Upgrading from Guice](#)

[Contributing](#)

[License](#)

[Javadoc](#)

[dagger-discuss@](mailto:dagger-discuss@square.com)

[StackOverflow](#)

- Library incorporating IoC / Dependency Injections Pattern for Android

# Retrofit

<http://square.github.io/retrofit/>

## Retrofit

Download v2.0.0-beta4



A type-safe HTTP client for Android and Java

### Introduction

Retrofit turns your HTTP API into a Java interface.

```
public interface GitHubService {
    @GET("users/{user}/repos")
    Call<List<Repo>> listRepos(@Path("user") String user);
}
```

The `Retrofit` class generates an implementation of the `GitHubService` interface.

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("https://api.github.com")
    .build();

GitHubService service = retrofit.create(GitHubService.class);
```

Each `Call` from the created `GitHubService` can make a synchronous or asynchronous HTTP request to the remote webserver.

```
Call<List<Repo>> repos = service.listRepos("octocat");
```

Use annotations to describe the HTTP request.

- URL parameter replacement and query parameter support
- Object conversion to request body (e.g., JSON, protocol buffers)
- Multipart request body and file upload

Note: This site is still in the process of being expanded for the new 2.0 APIs.

Introduction

API Declaration

Retrofit Configuration

Download

Contributing

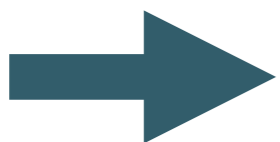
License

Javadoc

StackOverflow

• “Sane” Rest client development

• Watch this video



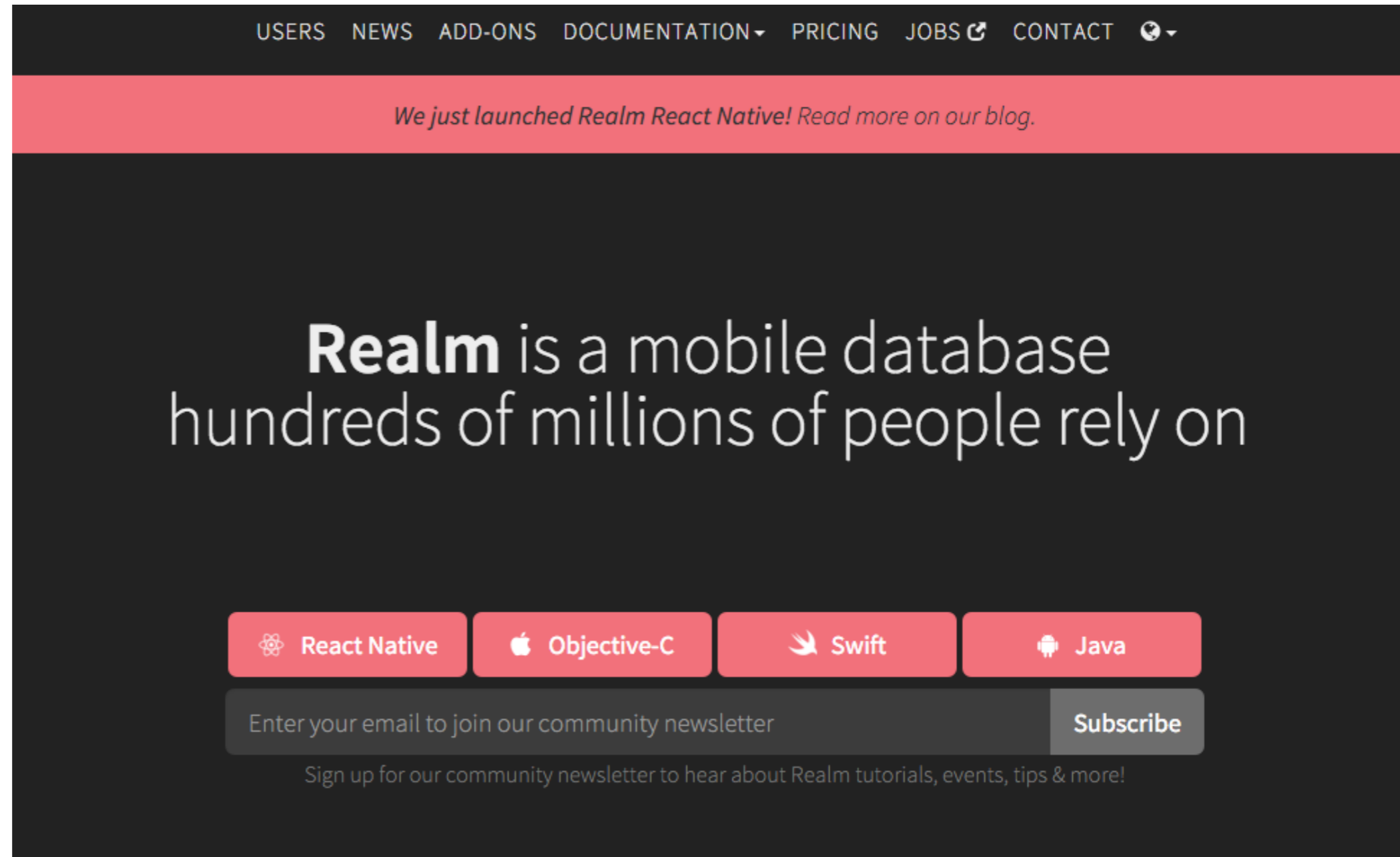
<https://realm.io/news/droidcon-jake-wharton-simple-http-retrofit-2/>

# Realm

<https://realm.io/>

---

- Local Persistence

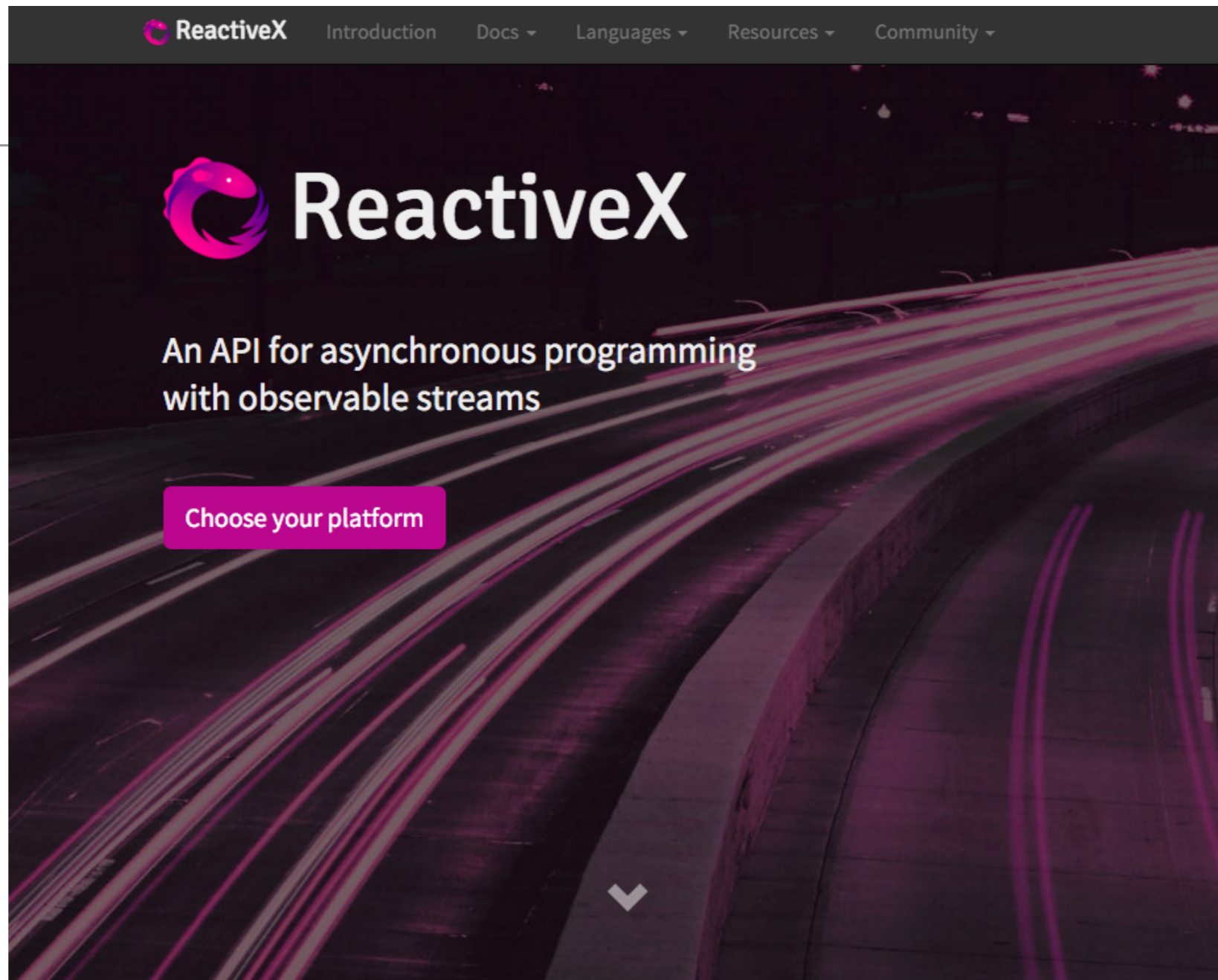


The screenshot shows the Realm website homepage. At the top, there is a navigation menu with links for USERS, NEWS, ADD-ONS, DOCUMENTATION (with a dropdown arrow), PRICING, JOBS (with an external link icon), and CONTACT (with a globe icon). Below the navigation is a red banner with the text: "We just launched Realm React Native! Read more on our blog." The main content area features the headline: "Realm is a mobile database hundreds of millions of people rely on". Below the headline are four red buttons with icons and text: "React Native" (with a React Native icon), "Objective-C" (with an Apple logo), "Swift" (with a Swift logo), and "Java" (with an Android logo). At the bottom, there is a dark grey input field with the placeholder text "Enter your email to join our community newsletter" and a "Subscribe" button. Below the input field is a small line of text: "Sign up for our community newsletter to hear about Realm tutorials, events, tips & more!"

# ReactiveX

<http://reactivex.io/>

- Reactive  
Approached



## The Observer pattern done right

ReactiveX is a combination of the best ideas from the **Observer** pattern, the **Iterator** pattern, and **functional programming**



# Resources for Further Android Development (9)

<http://fernandocejas.com/2014/09/03/architecting-android-the-clean-way/>

- Reactive  
Approached

## Architecting Android...The clean way?

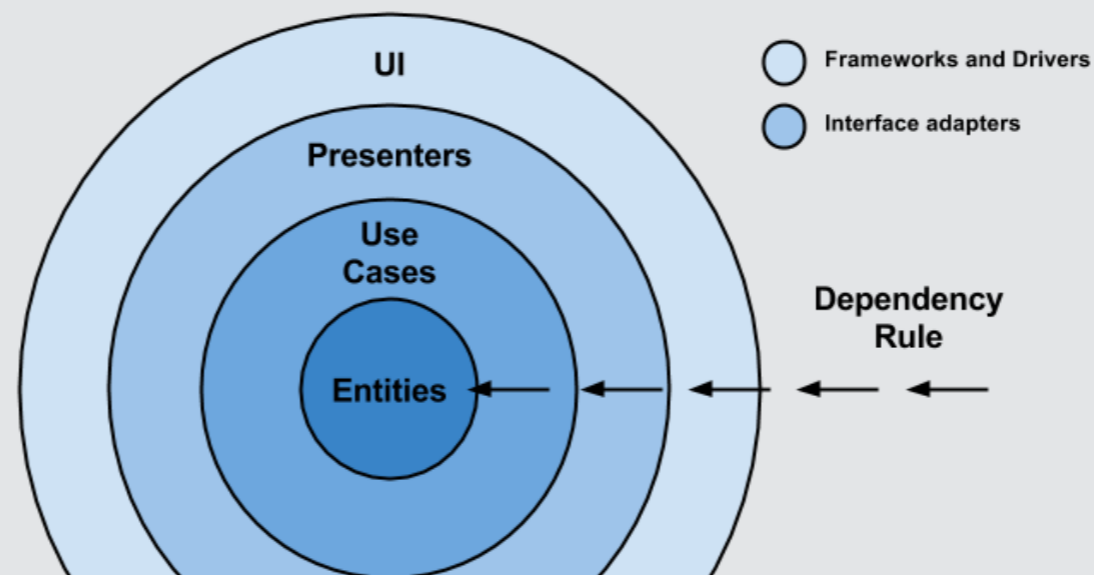
Over the last months and after having friendly discussions at [Tuenti](#) with colleagues like [@pedro\\_g\\_s](#) and [@flipper83](#) (by the way 2 badass of android development), I have decided that was a good time to write an article about **architecting android applications**. The purpose of it is to show you a little approach I had in mind in the last few months plus all the stuff I have learnt from investigating and implementing it.

### Getting Started

**We know that writing quality software is hard and complex:** It is not only about satisfying requirements, also should be robust, maintainable, testable, and flexible enough to adapt to growth and change. This is where **"the clean architecture"** comes up and could be a good approach for using when developing any software application.

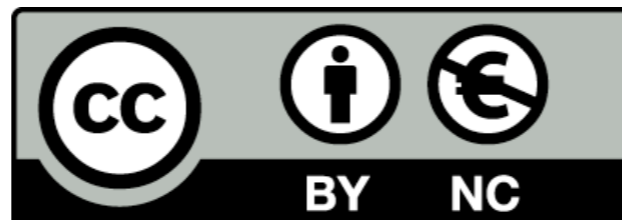
The idea is simple: **clean architecture** stands for a group of practices that produce systems that are:

- Independent of Frameworks.
- Testable.
- Independent of UI.
- Independent of Database.
- Independent of any external agency.









Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

