



Programowanie urządzeń mobilnych. Android

Karol Przystalski



Poznajmy się

2017 - doktorat uzyskany w PAN oraz UJ

od 2010 - CTO @ **Codete**

2007 - 2009 - Software Engineer @ **IBM**

Praca naukowa

Multispectral skin patterns analysis using fractal methods}, K. Przystalski and M. J. Ogorzalek. Expert Systems with Applications, 2017

<https://www.sciencedirect.com/science/article/pii/S0957417417304803>

kprzystalski@gmail.com

karol@codete.com



Zakres materiału

1. Podstawy języka Kotlin
2. Zaawansowany Kotlin
3. Podstawy systemu Android
4. Aktywności oraz podstawowe wzorce projektowe na platformie Android
5. Fragmenty
6. UI
7. REST oraz OAuth2
8. Serwisy
9. Notyfikacje
10. Wydajność

11. OpenGL, oraz VR
12. Wearables, NDK, IoT
13. Google Play
14. React Native



Zaliczenie

Średnia ocena ze wszystkich projektów.

Warunki:

- Termin na każdy projekt: max. 2 tygodnie po zajęciach
- Każdy projekt zaliczony na minimum 3.0

Egzamin: test wyboru, 30 pytań. Aby zaliczyć trzeba poprawnie odpowiedzieć na 20 pytań.

Terminy: tba

Kotlin



Trochę historii

Powstał jako (szybsza) alternatywą dla Scali w 2011 roku.

Twórcą języka jest Dmitry Jemerov, a firma która rozwija język to JetBrains.

Od 2017 jest wspieranym językiem na platformie Android.

Obecnie jedynie nieliczni decydują się na Javę pisząc aplikację na Androida.



Charakterystyka języka

Działa na JVM

Podobnie jak Scala, pisanie kodu w Kotlinie jest krótsze i szybsze

Statycznie typowany



Playground

**Zanim przejdziemy do pisania
aplikacji na platformę Android**

—



Garbage collector

Języki, które go stosują to m.in.: Java oraz inne na JVM, JavaScript, C#, Go, Ruby.

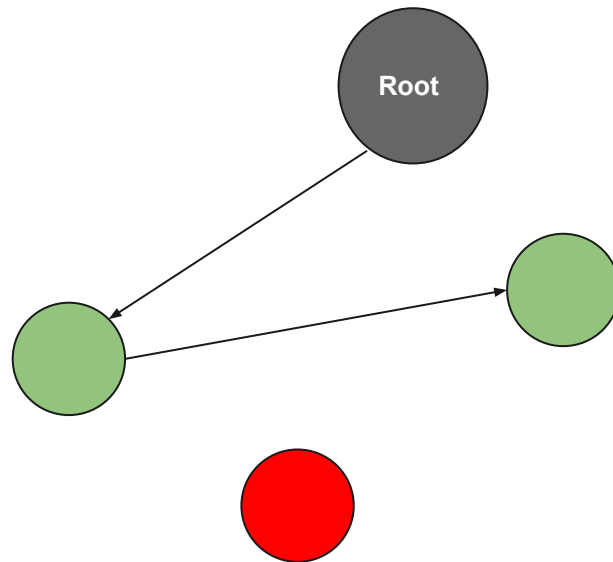
Istnieje wiele implementacji GC:

- Tracing GC - składa się z dwóch etapów: wykrywania „żyjących” obiektów oraz zwalnianie pamięci,
- RC GC - podobne do ARC; posiada osobny proces GC, który czyści pamięć

Garbage collector

Mark - zaznacz (zielone)

Sweep - usuń (czerwone)





Java Garbage collector

Istnieje wiele GC w Javie 11 (-XX:):

- Serial Collector - jeden wątek, dobry dla małych aplikacji
- Parallel Collector - wiele wątków; średnie aplikacje
- Garbage-First Collector - równoległe wątki; wykorzystywany na sprzęcie z wieloma procesorami i dużą ilością pamięci
- Z Garbage Collector - dostępny od Javy 11 zoptymalizowany na ograniczenie opóźnień po stronie aplikacji.

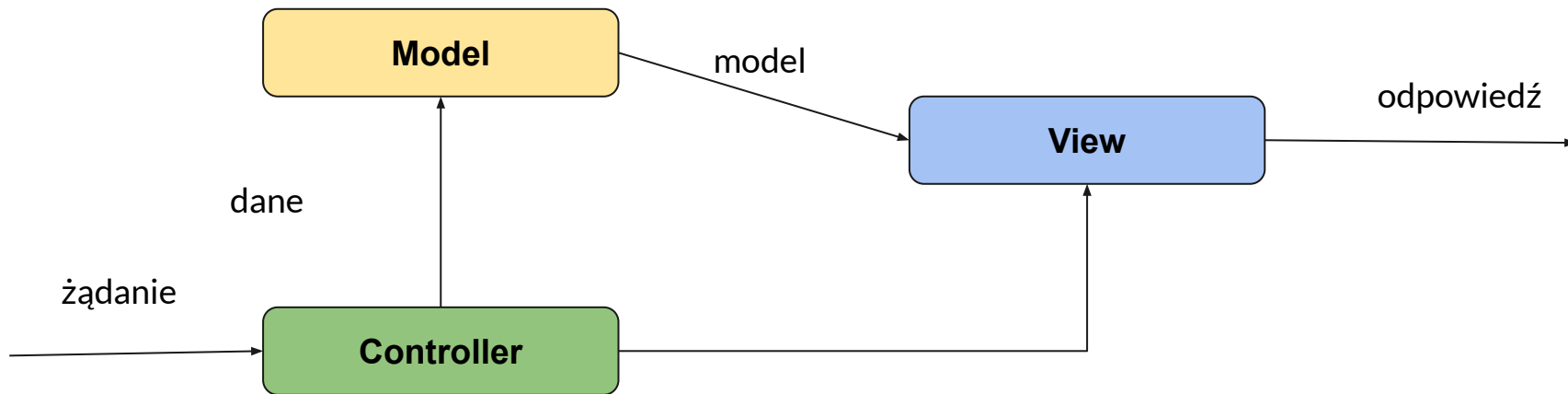



Zarządzanie pamięcią w różnych językach

- Python - wykorzystuje połączenie RC z GC.
- Java - GC
- JavaScript - GC
- Haskell - GC
- Go - GC via Go scheduler
- Ruby - GC
- Swift - ARC
- Lisp - GC
- COBOL - manualne
- Lua - GC



MVC





MVC - zalety

Logika biznesowa oddzielona jest od widoku,

Nie ma zależności modelu od widoku

Uporządkowany kod



MVC - wady

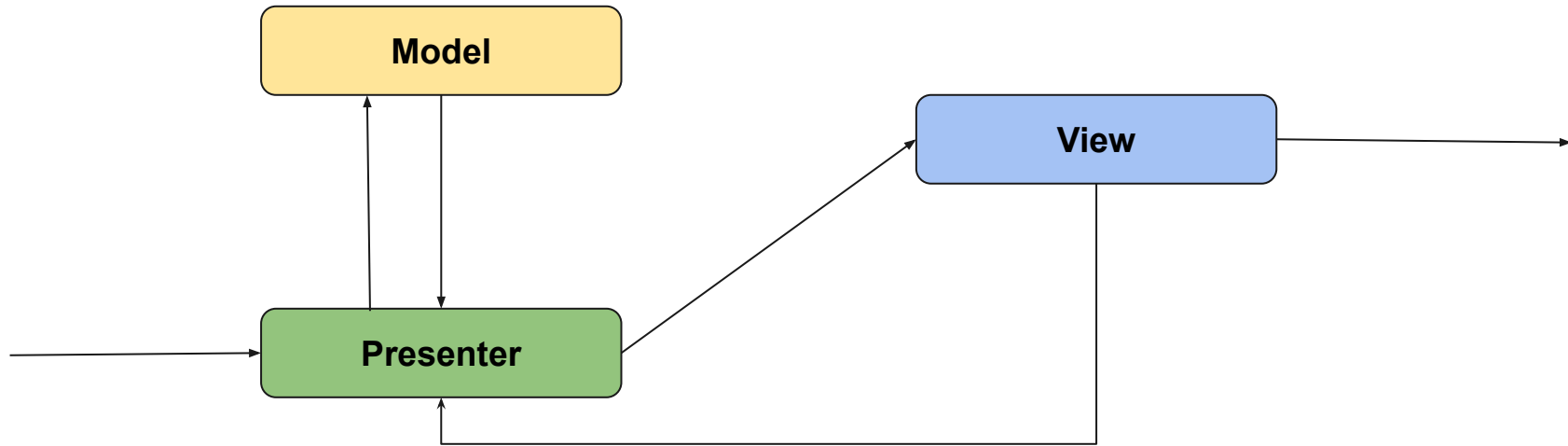
Złożoność aplikacji


Widok zależny od modelu

Widoki są trudne w testowaniu



MVP





MVP - zalety

Pochodna MVC

Oddzielona logika od widoku

Brak zależności modelu od widoku,

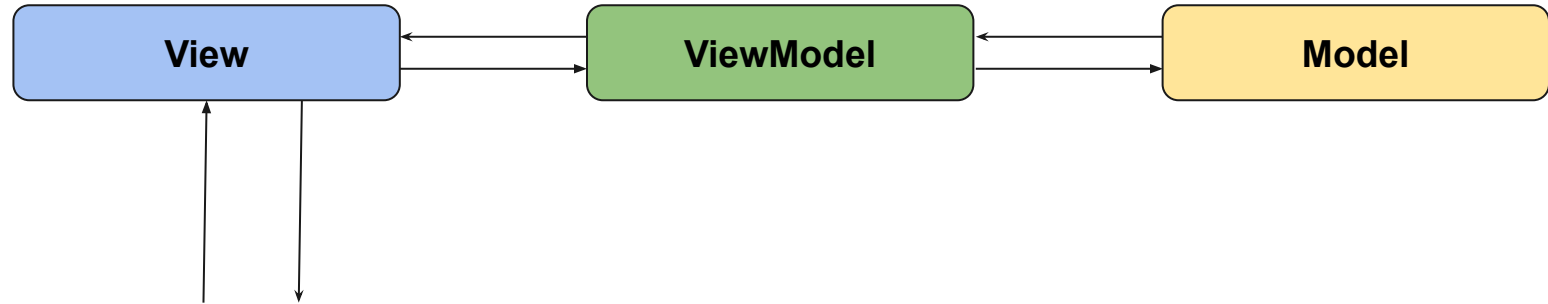



MVP - wady

Skomplikowana dla prostych aplikacji



MVVM






MVVM- zalety

Podział na widok i ViewModel

Testowanie jest prostsze niż w MVC

Asynchroniczność



MVVM - wady

ViewModel nie komunikuje się z warstwą widoku

Dużo klas, ponieważ każdy widok ma więcej niż jedną klasę po stronie widoku i ViewModel.

Wprowadzenie do platformy Android





SDK

SDK Tools - narzędzia do tworzenia aplikacji na Andorida

SDK Platform - przede wszystkim kod źródłowy (wersja API)

SDK Build tools - narzędzia do budowania aplikacji na Androida

SDK Platform tools - narzędzia, które wykorzystują SDK Platform, np. adb



Andorid NDK

Native Development Kit - pozwala na budowanie aplikacji na Androida w C/C++.

<https://developer.android.com/ndk>



Android inaczej

JavaScript

<https://reactnative.dev/>

Dart

<https://flutter.dev/>

Python

<https://beeware.org/>



Budowanie z linii poleceń

Budowanie:

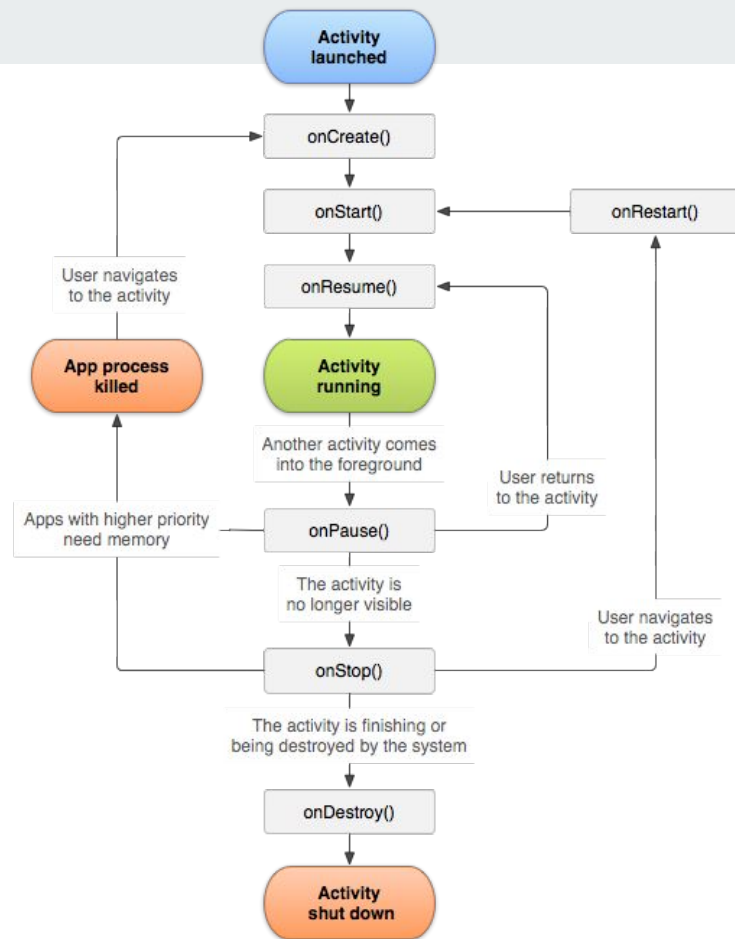
```
./gradlew assembleDebug
```

Instalowanie:

```
./gradlew installDebug
```

```
adb install .apk
```

Aktywności





Intencje (intent)

Łączy elementy w aplikacji Androidowej ze sobą.

Zawiera:

- akcję
- dane

Przykład: dwa widoki



Komponenty

Grupy komponentów:

- tekstowe
- przyciski
- widżety
- layouty
- kontenery
- helpery
- googlowskie

Struktura



ToDo

<https://github.com/android/architecture-samples/tree/todo-mvp-kotlin/todoapp>



Firestore

<https://github.com/firebase/quickstart-android/tree/master/database>



Realm

<https://github.com/realm/realm-kotlin-samples/tree/main/Bookshelf/androidApp>



MVVM

<https://github.com/ianishar/android-mvvm-architecture>

<https://github.com/PatilShreyas/Foodium>

<https://github.com/androiddevnotes/awesome-android-kotlin-apps>

Wzorce w *mobilkach*



Wzorce projektowe

Jest kilka grup wzorców projektowych:

- strukturalne
 - Adapter
 - Most (Bridge)
 - Kompozyt (Composite)
 - Dekorator (Decorator)
 - Fasada (Facade)
 - Pyłek (Flyweight)
 - Pełnomocnik (Proxy)



Wzorce projektowe

Jest kilka grup wzorców projektowych:

- strukturalne
- kreacyjne
- Fabryka abstrakcyjna (Abstract Factory)
- Budowniczy (Builder)
- Metoda wytwórcza (Factory Method)
- Object Pool (Pula obiektów)
- Prototyp (Prototype)
- Singleton



Wzorce projektowe

Jest kilka grup wzorców projektowych:

- strukturalne
 - kreacyjne
 - behawioralne
- Łańcuch zobowiązań (Chain of responsibility)
 - Polecenie (Command)
 - Interpreter
 - Iterator
 - Mediator
 - Memento
 - Obserwator (Observer)
 - Stan (State)
 - Strategia (Strategy)
 - Metoda szablonowa (Template Method)
 - Wizytator (Visitor)



Strukturalne wzorce projektowe



Adapter

Założenia

- Konwertuje interfejs klasy na interfejs, który jest oczekiwany przez klienta
- Pozwala na pracę pomiędzy klasami, które normalnie nie mogłyby pracować
- Pozwala na współpracę starego/innego rozwiązania z naszym

Przykład

Brak kompatybilności pomiędzy standardami obsługi odpowiedzi od serwera. Do obsługi możemy wykorzystać interfejs, który pozwoli na obsługę konkretnej odpowiedzi z serwera przez klasę, która nie została zaimplementowana do obsługi odpowiedzi danego typu. Innym przykładem jest wtyczka do prądu w UK, US oraz w Europie. Istnieją adaptery, które pozwalają na pracę wtyczek z kontaktami w różnych krajach.



Most (Bridge)

Założenia

- Celem jest rozdzielenie abstrakcji od implementacji
- Udostępnia hierarchię interfejsów oraz odpowiednią hierarchię implementacji

Przykład

Most rozbija część implementacji od jej abstrakcji, w taki sposób że są od siebie niezależne. Przykładem może być przełącznik do włączania/wyłączania światła. Abstrakcja jest jedna, a implementacji wiele.



Kompozyt (Composite)

Założenia

- Kompozyt tworzy reprezentację obiektów za pomocą drzewa. Jednocześnie kompozyt pozwala na dostęp do poszczególnych elementów ze względu na dziedziczenie po interfejsie nadrzędnym
- Upraszcza kod aplikacji
- Rekursywne

Przykład

Operacja arytmetyczna może być kompozytem, ponieważ $(2+3) - (6+2)$ jest tym samym co $2+3-6+2$.



Dekorator (Decorator)

Założenia

- Dodaje dodatkowe funkcjonalności do obecnie istniejących klas opakowując je, wywołując metody wewnątrz
- Jest alternatywą dla tworzenia podklas

Przykład

Dodanie kilku klas CSS do zwracanego przez klasę kodu HTML jest dobrym przykładem dekoratora.



Fasada (Facade)

Założenia

- Opakowuje złożoną część kodu przez proste interfejsy
- Uogólnia system za pomocą prostych interfejsów
- Wysokopoziomowe podejście

Przykład

Fasadą jest niemal każdy call center. Dzwonimy na numer biura obsługi klienta i wybieramy kolejno numery, aby skontaktować się z odpowiednią osobą, która jest odpowiednia dla rozwiązania naszego problemu.



Pyłek (Flyweight)

Założenia

- Pozwala na współdzielenie bardzo wielu małych obiektów
- Ma zastosowanie w usprawnieniach wydajności

Przykład

Przeglądarki wykorzystują ten wzorzec do ładowania dużej liczby rysunków.



Proxy

Założenia

- Tworzy obiekt pośredni w komunikacji pomiędzy dwoma innymi
- Pozwala na dostęp oraz kontrolę drugim obiektem

Przykład

Najprostszy przykładem jest proxy wykorzystywane w przeglądarkach.



Funkcyjne wzorce projektowe



Monoid

Założenia

- Struktura algebraiczna
- Występuje funkcja zwracająca wartość pustą oraz metoda łącząca zwracająca wartość podaną jako argument, np. combine

Przykład

MapReduce wykorzystuje monoidy. Option jest monoidem



Monada

Założenia

- Pozwala na zastąpienie wielu wywołań, które od siebie zależą funkcjami unit oraz bind
- Upraszcza operacje pomiędzy funkcjami zależnymi
- Jest monoidem

Przykład

Monada zamienia zagnieżdżone wyrażenia w proste (flatten)



Funktor

Założenia

- Funktor stosuje operację na elementach listy, funkcjach lub innych strukturach
- Zawiera funkcję mapującą (map, flatmap),
- Wyjście jest typu wejścia

Przykład

Funktory są często wykorzystywane przy listach, np. kiedy je mapujemy.



Kreacyjne wzorce projektowe



Fabryka abstrakcyjna (Abstract factory)

Założenia

- Dostarcza interfejs do tworzenia całej rodziny powiązanych obiektów bez specyfikowania klasy z implementacją
- Hierarchia, która enkapsuluje wiele klas (produktów)

Przykład

Maszyny produkujące elementy samochodów wykorzystują ten wzorzec. Mogą po podmianie niektórych części maszyny, mogą one produkować różne elementy samochodu do różnych ich modeli.



Budowniczy (Builder)

Założenia

- Oddziela logikę odpowiedzialną za tworzenie złożonych obiektów od ich reprezentacji
- Umożliwia na tworzenie różnych obiektów w zależności ich reprezentacji

Przykład

Tworzenie różnych rodzajów pizzy jest takim przykładem. Główny proces jest taki sam, różnią się jedynie



Fabryka (Factory method)

Założenia

- Definiuje interfejs dla tworzenia obiektu, jednocześnie pozostawia decyzję klasie, którą klasę zainicjalizować
- W odróżnieniu od budowniczego skupia się wokół konstruktora, jest prostsza.

Przykład

Jest wykorzystywana przez budowniczego.



Pula obiektów (Object pool)

Założenia

- Ma również zastosowanie przy projektach, gdzie istotna jest prędkość działania
- Pozwala na alokowanie pamięci dla grupy obiektów, ponieważ ta część jest najbardziej czasochłonna
- Udostępnia instancje obiektów

Przykład

W grach jest często wykorzystywana oraz wszędzie tam, gdzie wykorzystywane są identyczne obiekty wiele razy.



Prototyp (Prototype)

Założenia

- Jest prototypem obiektu
- W wielu językach klonuje albo kopiuje się prototyp

Przykład

W języku JavaScript jest podstawowym pojęciem.



Singleton

Założenia

- Jeden z prostszych wzorców
- Zapewnia istnienie jednej instancji danej klasy

Przykład

Wykorzystywany jest często w autoryzacji czy dostępie do bazy danych.