Aplikacje mobile w praktyce

Łukasz Beksa, Adam Nowicki

Katowice • 04 marca 2020

thinkforward

ING

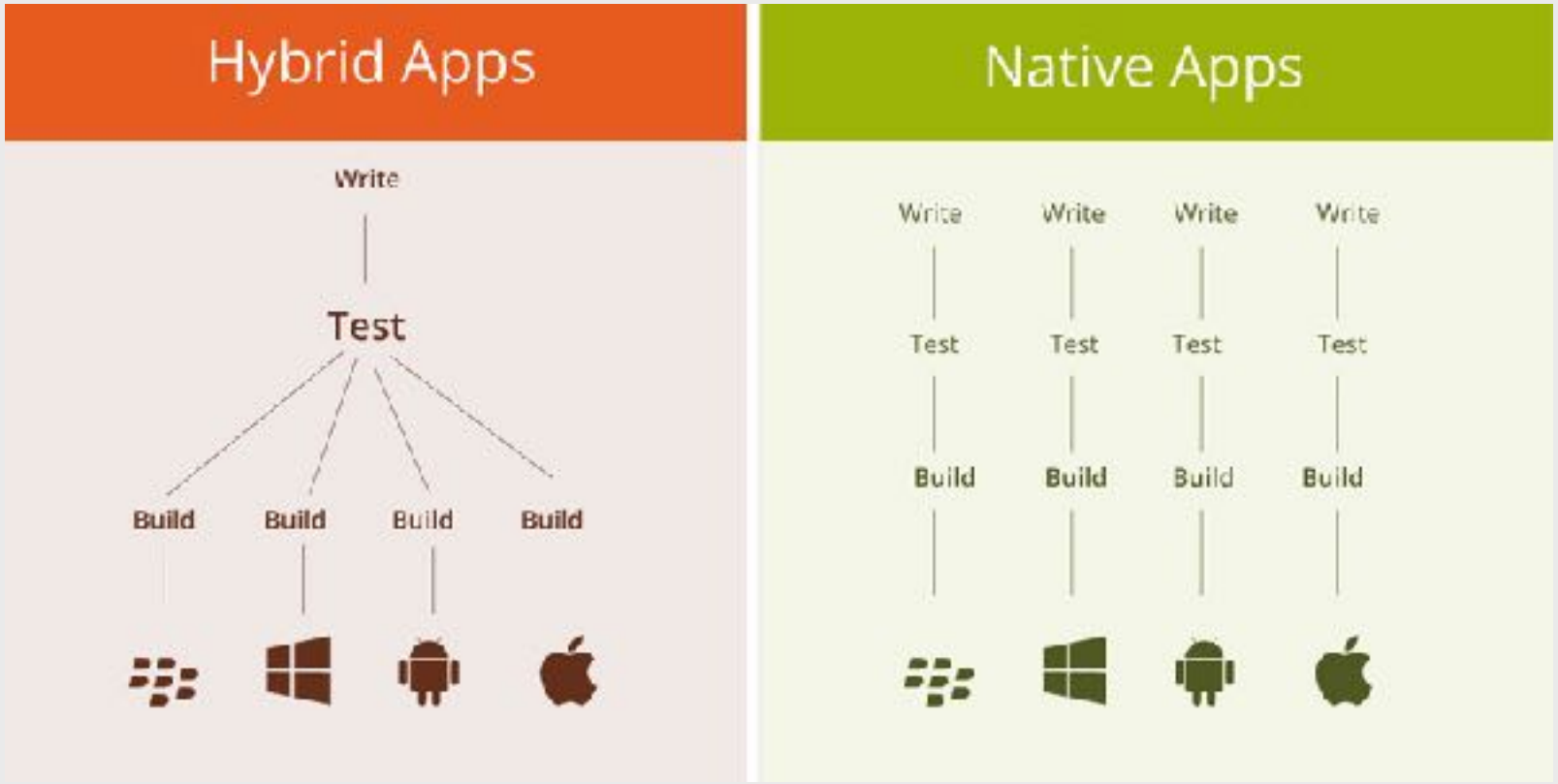

POZNAJMY
SIĘ
Memy.pl

Mamy różne doświadczenia

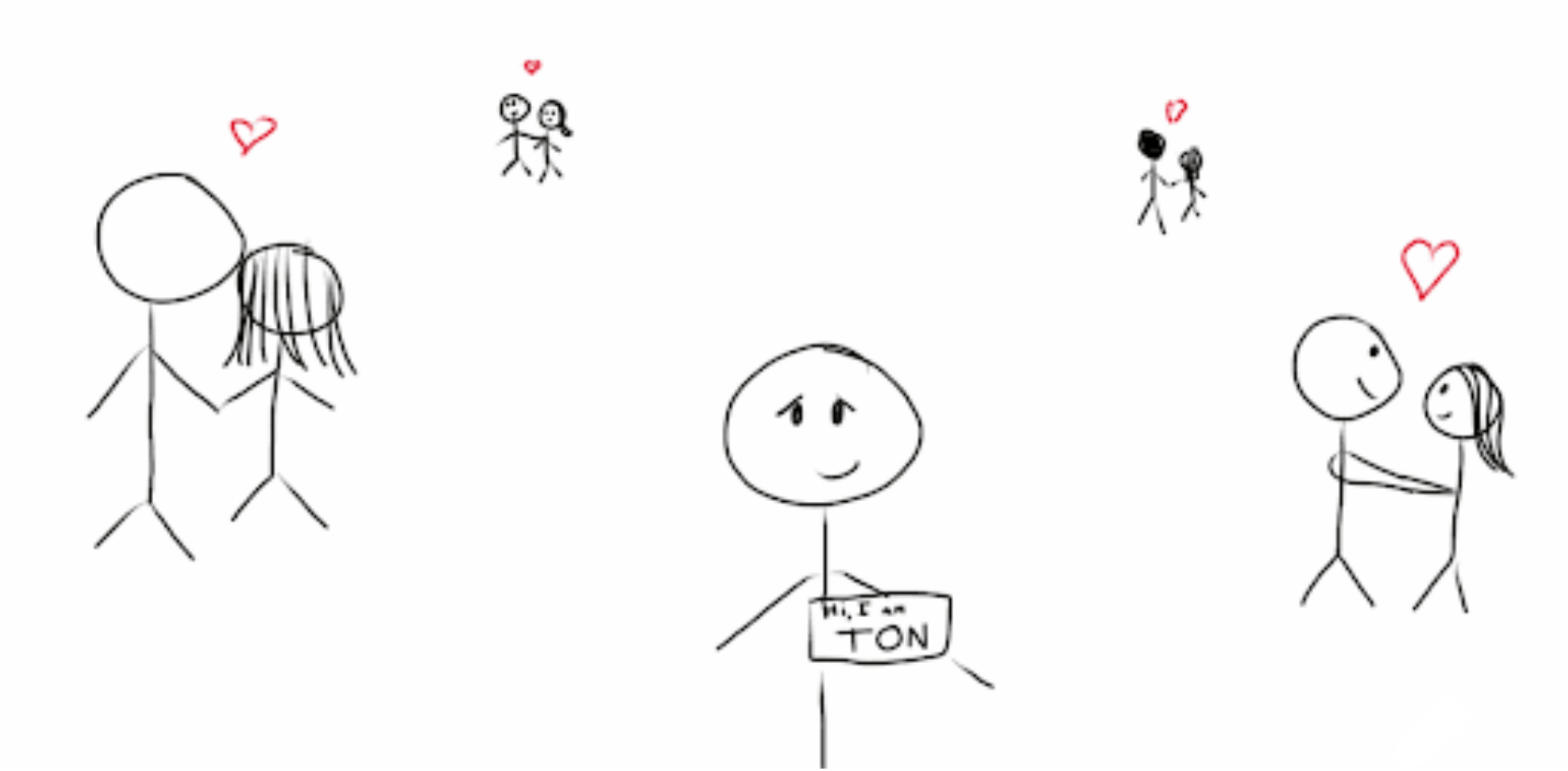
Aplikacja mobilna w liczbach (01-02-2020)

mobile only 1,1 mln	platformy 2	wersja 3.3.1
Native Dev Team 8	liczba pobrań 3 mln	interakcji dziennie 2,5 mln

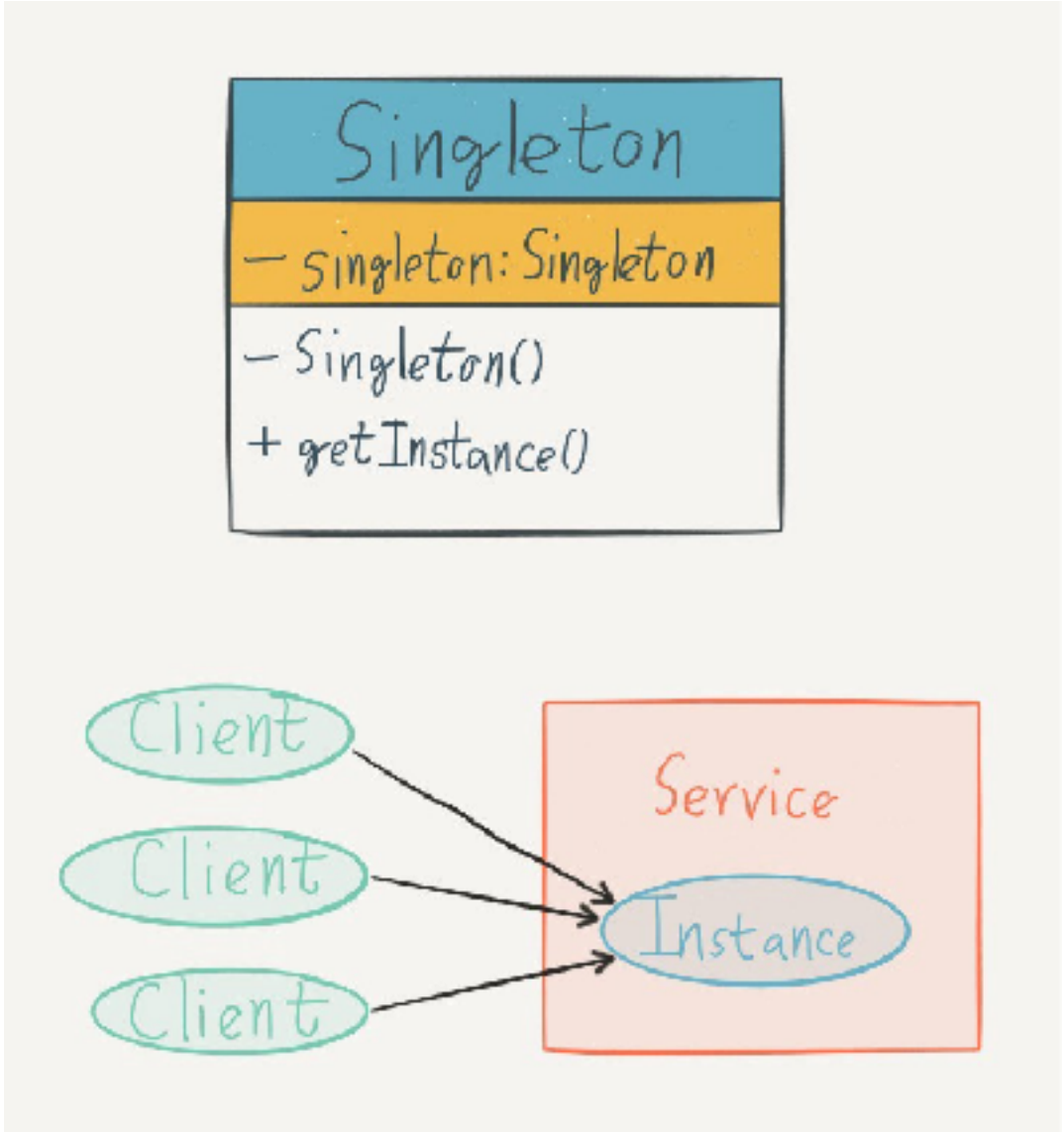
Cykl życia oprogramowania mobile



Wzorce projektowe, Singleton



Wzorce projektowe, Singleton



Singleton

```
public class SingletonSimpleEager {

    private static final SingletonSimpleEager instance = new
SingletonSimpleEager();

    private SingletonSimpleEager() {
    }

    public static SingletonSimpleEager getInstance() {
        return instance;
    }
}

// wywołanie
SingletonSimpleEager singleton = SingletonSimpleEager.getInstance();
```

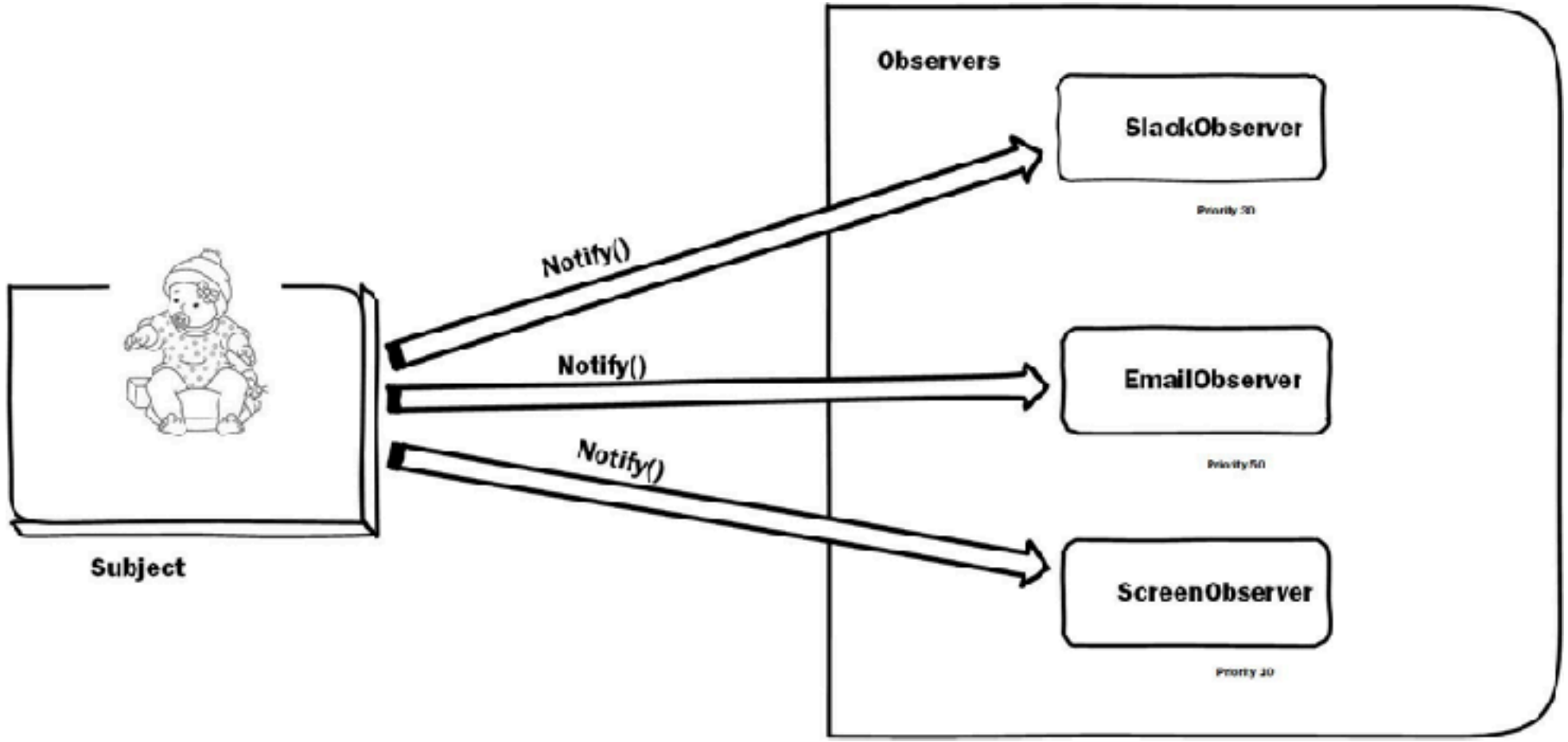
Singleton - jako antywrzozec

Poważnie utrudnia testowanie aplikacji
Testy są tylko utrudnione, jeżeli w singletonie przechowywany jest stan. Należy wtedy pamiętać, by był on odpowiednio zainicjowany lub wyczyszczony przed każdym wywołaniem testu.
Brak elastyczności
Taka jest właśnie specyfika singletonu, że już na poziomie kodu jest na sztywno określona liczba instancji.
Łamie zasadę jednej odpowiedzialności (*single responsibility principle*)
Klasa zaimplementowana jako singleton z założenia jest już odpowiedzialna za dwie rzeczy: za realizację swoich funkcji biznesowych oraz zarządzanie instancją.
Łamie zasadę otwarte-zamknięte (*Open/Closed principle*), ponieważ nie można go rozszerzać
W pierwotnej wersji wzorca rzeczywiście ciężko jest go rozszerzać. Można jednak połączyć singleton z fabryką i nie będzie stanowiło to już problemu.

Jest to obiektowy zamiennik zmiennej globalnej

https://stormit.pl/singleton/

Wzorce projektowe, Observer



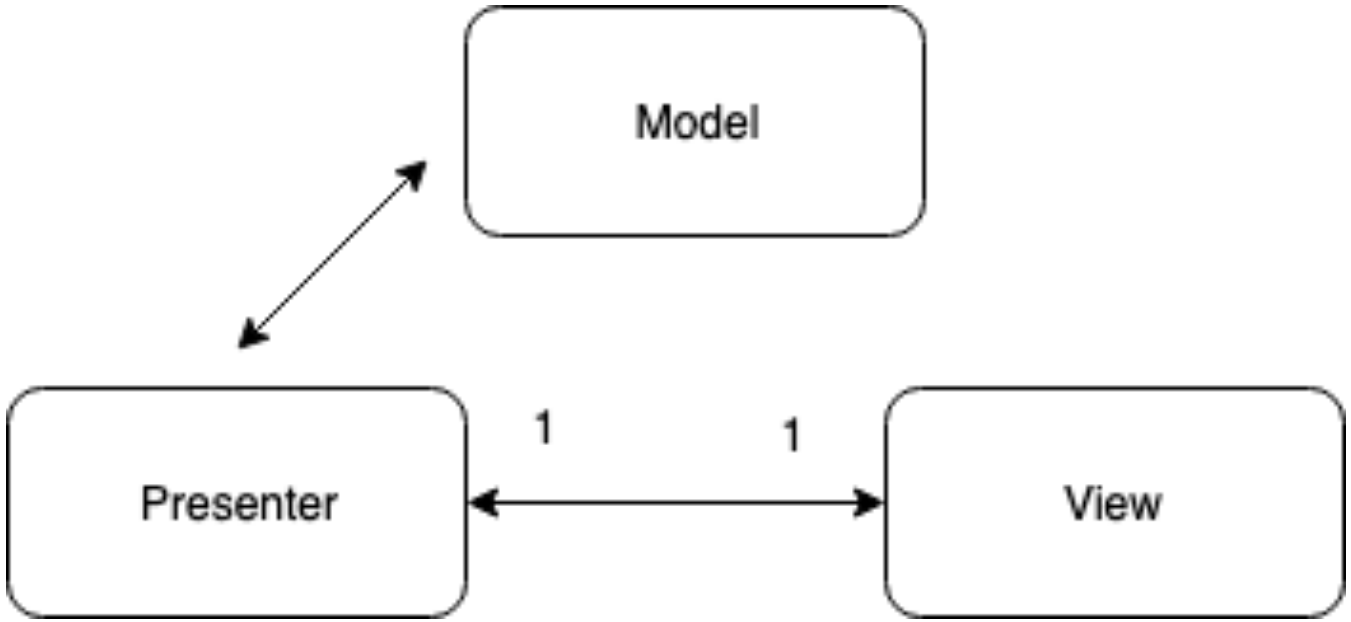
Wzorce, Dependency Injection - idea

```
class A {
 public int doSomeWork(int p1, int p2, String p3) {
  B objectB = new B(p1, p2, p3);
  int result = objectB.getResult();
  int finalResult = result * 2;
  return finalResult;
 }
}
```

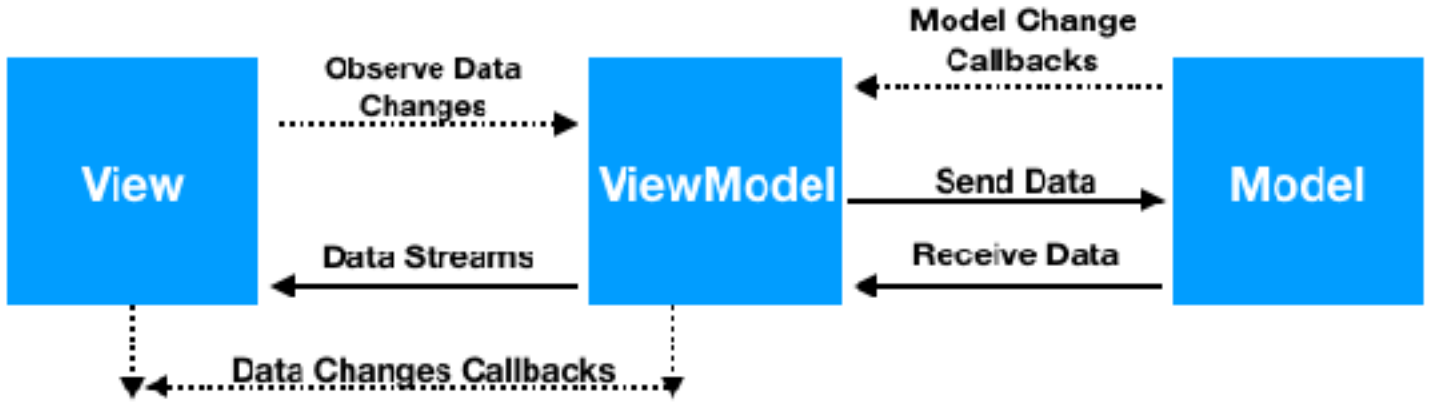
Wzorce, Dependency Injection - idea

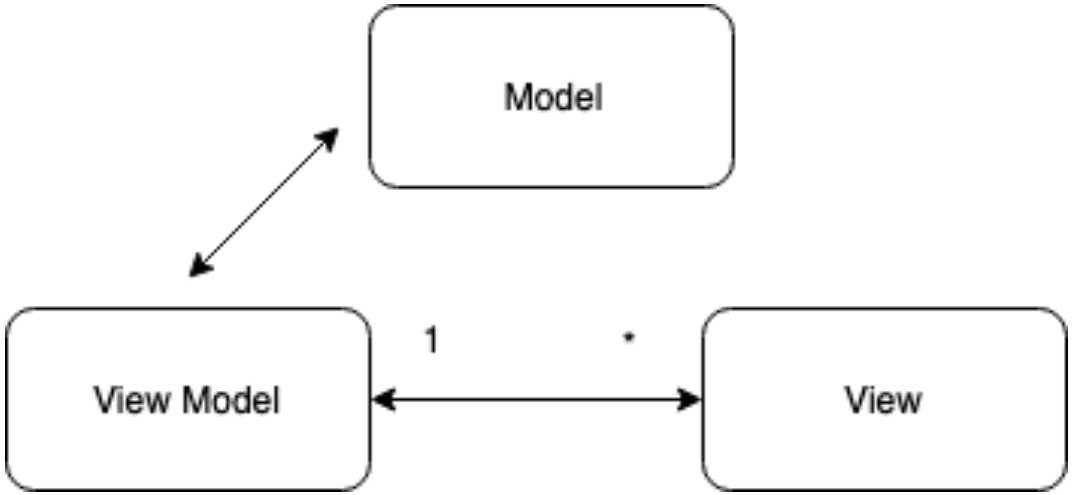
```
class A {
 private B objectB;
 public A(B objectB) {
  this.objectB = objectB;
 }

 public int doSomeWork() {
  int result = objectB.getResult();
  int finalResult = result * 2;
  return finalResult; }
 }
```

Wzorce projektowe Android - MVP



Wzorce projektowe android - MVVM





Viewmodel?

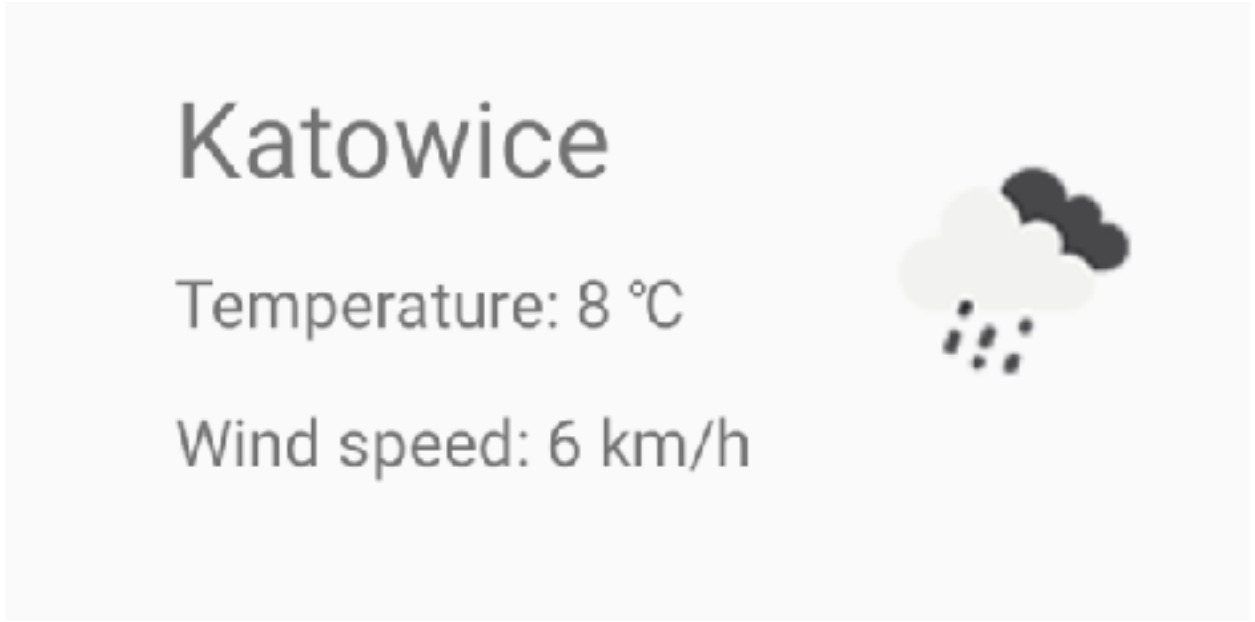
Centralnym elementem wzorca MVVM jest viewmodel. Można by powiedzieć, że to swego rodzaju odpowiednik controllera z MVC, ale byłoby to stwierdzenie bardzo na wyrost. Poniżej krótka charakterystyka tego elementu:

- Trzyma referencję do modelu.
- Wykształca mechanizm komend, które później są wykorzystywane w widoku do obsługi interakcji wywołanych przez użytkownika.
- Jest „bindowany” jako kontekst danych dla strony/widoku/okna.

***Data binding**
Pewną bardzo fajną cechą wzorca MVVM jest to, że dane przechowywane przez model są automatycznie odświeżane w widoku, w sytuacji gdy dojdzie do modyfikacji danych po stronie viewmodelu.

Przykładowa aplikacja

https://github.com/adamnovicki/StudentsTemplate



ING

Android Best Practice - AsyncTask

```
class DownloadFilesTask extends AsyncTask<URL, Integer, Void> {
     protected Long doInBackground(URL... urls) {
       //downloading . . .
     }

     protected void onProgressUpdate(Integer... progress) {
         setProgressPercent(progress[0]);
     }

     protected void onPostExecute(Void result) {
         showDialog("Downloaded");
     }
 }



new DownloadFilesTask().execute(url1, url2, url3);
```

Android Best Practice - Kotlin Coroutines

```
fun main() {
    GlobalScope.launch {  // launch a new coroutine in background and continue
        delay(1000L)      // non-blocking delay for 1 second
        println("World!") // print after delay
    }
    println("Hello,")     // main thread continues while coroutine is delayed
    Thread.sleep(2000L)   // block main thread for 2 seconds to keep JVM alive
}
```

```
Hello,
World!
```

Android Best Practice - Async callbacks / Coroutines

```
networkRequest() { result ->
    // Successful network request
    databaseSave(result) { rows ->
        // Result saved
    }
}
```

```
// The same code with coroutines
val result = networkRequest()
// Successful network request
databaseSave(result)
// Result saved
```

Android Best Practice - Retrofit

```
class WeatherApi(private val context: Context) {

    val SERVER_URL = "https://api.openweathermap.org"

    fun getWeatherApiService() : WeatherApiService {

        val cookieManager = CookieManager()
        cookieManager.setCookiePolicy(CookiePolicy.ACCEPT_ALL)

        val okBuilder = OkHttpClient.Builder()
        okBuilder.cookieJar(JavaNetCookieJar(cookieManager))

        val retrofit = Retrofit.Builder()
            .baseUrl(SERVER_URL)
            .addConverterFactory(MoshiConverterFactory.create())
            .addCallAdapterFactory(CoroutineCallAdapterFactory())
            .build()

        return retrofit.create(WeatherApiService::class.java)
    }
}


interface WeatherApiService {

    @GET("/data/2.5/forecast?APPID=049cc7883268b2ac341d14f127461559&")
    fun forecast(@Query("q") city: String): Deferred<WeatherRsp>
}
```

Android Best Practice - Retrofit

```
class WeatherRepository(private val weatherApiService: WeatherApiService) {

    suspend fun getWeather(city: String): Result<WeatherRsp> {

        var result: Result<WeatherRsp> = Result.success(WeatherRsp())

        withContext(Dispatchers.IO) {

            try {
                val request = weatherApiService.forecast(city)

                val response: WeatherRsp = request.await()
                Timber.d("onWeatherReceived ${response}")

                request.let {
                    if (it.isCompleted) {
                        result = Result.success(response)
                    }
                    else if (it.isCancelled) {
                        result = Result.error(CancelledFetchDataException())
                    }
                }
            } catch (ex: Throwable) {
                result = Result.error(NetworkException())
                Timber.d("onWeatherReceived NetworkException")
            }
        }
        return result
    }
}
```

Android Best Practice - Timber

```
Log.d("WeatherActivity", "getWeather")

Timber.d("getWeather")
```

```
D/WeatherActivity: getWeather
D/WeatherActivity: getWeather
```

WeatherRsp - response z API open weather


```
{
    "dt": 1569931200,
    "main": {
        "temp": 292.45,
        "temp_min": 291.4,
        "temp_max": 292.45,
        "pressure": 1006.75,
        "sea_level": 1006.75,
        "grnd_level": 971.85,
        "humidity": 49,
        "temp_kf": 1.05
    },
    "weather": [
        {
            "id": 800,
            "main": "Clear",
            "description": "clear sky",
            "icon": "01d"
        }
    ],
    "clouds": {
        "all": 0
    },
    "wind": {
        "speed": 8.28,
        "deg": 216.354
    },
    "sys": {
        "pod": "d"
    },
    "dt_txt": "2019-10-01 12:00:00"
},
```


```
data class WeatherUI (
    val temp: Long,
    val windSpeed: Long,
    val city: String,
    val iconType: IconType
)


enum class IconType {
    SUN,
    CLOUD,
    RAIN,
    SNOW
}
```


ING

WeatherViewModelMapper

```
class WeatherViewModelMapper {

    object WeatherToUI {
        fun map(rsp: Result<WeatherRsp>?): Result<WeatherUI> {
            lateinit var result: Result<WeatherUI>

            if (rsp?.resultType == ResultType.ERROR) {
                result = Result.error(error = rsp.error)

            } else {
                val details = rsp?.data?.list?.first()
                val weatherData = WeatherUI(
                    temp = getCelsiusTemp(details?.main?.temp ?: KELVIN_ZERO),
                    windSpeed = (details?.wind?.speed ?: 0.0).roundToLong(),
                    city = rsp?.data?.city?.name ?: "",
                    iconType = mapWeatherType(details?.weather?.first()?.id)
                )
                result = Result.success(data = weatherData)
            }
            return result
        }
    }
```

WeatherViewModelMapper - ikona pogody

```
companion object {

    private fun mapWeatherType(id: Int?): IconType {
        return when (id) {
            800 -> IconType.SUN
            in 801..804 -> IconType.CLOUD
            in 500..531 -> IconType.RAIN
            in 600..622 -> IconType.SNOW
            else -> IconType.CLOUD
        }
    }

    private const val KELVIN_ZERO = 273.15

    private fun getCelsiusTemp(kelvins: Double) : Long = (kelvins - KELVIN_ZERO).roundToLong()
}
```

Group 80x: Clouds

ID	Main	Description
801	Clouds	few clouds: 11-25%
802	Clouds	scattered clouds: 25-50%
803	Clouds	broken clouds: 51-84%
804	Clouds	overcast clouds: 85-100%

ING

Mamy dane

- Dane gotowe do zaprezentowania
- Wybór sposobu prezentacji
- Zwykłe Activity
- MVP
- MVVM

Android Best Practice - MVP - Presenter

```
interface View {
    fun refreshViews(weather: WeatherUI)
    fun showProgress()
    fun hideProgress()
}
```

```
init {
    this.weakView = WeakReference(view)
}
```

```
fun getWeather(city: String) {
   getWeatherUseCase(city)
}
```

```
weakView.get()?.refreshViews(weather)
```

Android Best Practice - MVP - View

```
@Inject
lateinit var presenter: WeatherPresenter


presenter.getWeather()


fun refreshViews(weather: WeatherUI) {
    cityTv.text = weather.city
    tempTv.text = getString(R.string.temp, weather.temp)
    windTv.text = getString(R.string.wind, weather.windSpeed)
}
```

Android Best Practice - MVVM - ViewModel

```
val weatherLiveData: MutableLiveData<WeatherUI> = MutableLiveData()
val isErrorLiveData: MutableLiveData<Boolean> = MutableLiveData()


updateWeatherLiveData(weatherRepository.getWeather(city))


private fun updateWeatherLiveData(result: Result<WeatherData>) {
    if (isResultSuccess(result.resultType)) {
        onResultSuccess(result.data)
    } else {
        onResultError()
    }
}


private fun onResultSuccess(weatherData: WeatherData?) {
    val weather = WeatherViewModelMapper.WeatherToUI.map(weatherData)
    weatherLiveData.postValue(weather)
}
```

Android Best Practice - MVVM - View

```
viewModel.weatherLiveData.observe(this, Observer(::onWeatherReceived))


viewModel.getWeather("Katowice")


private fun onWeatherReceived(weather: WeatherUI) {
    cityTv.text = weather.city
    tempTv.text = getString(R.string.temp, weather.temp)
    windTv.text = getString(R.string.wind, weather.windSpeed)
}
```

Android Best Practice - DataBinding - activity_main.xml

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable name="weather" type="pl.ing.cleanarchitecture.view.WeatherUI"/>
    </data>
    ...

    <TextView
        android:id=„@+id/tempTv"
        ...

        android:text="@{String.format(@string/temp, weather.temp)}"

    />

</layout>
```

Android Best Practice - DataBinding - MainActivity

```
private lateinit var binding: ActivityMainBinding


override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    binding = DataBindingUtil.setContentView(this, R.layout.activity_main)

    observeLiveData()
    getWeather()
}


private fun onWeatherReceived(weather: WeatherUI) {
    binding.weather = weather
}
```

Android Best Practice - Dependency Injection

- Dagger
- Koin

Android Best Practice - Dagger 2 - Component

```
@Singleton
@Component(
    modules = [
        AndroidSupportInjectionModule::class,
        AppModule::class,
        ViewModelModule::class,
        ActivityModule::class
    ]
)
interface AppComponent : AndroidInjector<MyApplication> {
    fun repo(): AppRepository

    @Component.Builder
    abstract class Builder : AndroidInjector.Builder<MyApplication>() {
        abstract override fun build(): AppComponent
    }
}
```

Android Best Practice - Dagger 2 - Module

```
@Module
class AppModule {

    @Singleton
    @Provides
    fun provideContext(app: MyApplication): Context {
        return app
    }

    @Singleton
    @Provides
    fun provideRepository(private val weatherApiService: WeatherApiService): WeatherRepository {
        return WeatherRepository(weatherApiService)
    }
}
```

Android Best Practice - Dagger 2 - App

```
val appComponent by lazy {
    DaggerAppComponent.builder().create(this) as AppComponent
}
```

```
override fun onCreate() {
    super.onCreate()

    appComponent.inject(this)
}
```

Android Best Practice - Dagger 2 - Activity

```
@Inject
lateinit var repo: WeatherRepository


override fun onCreate(savedInstanceState: Bundle?) {
    AndroidInjection.inject(this)
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
}
```

Android Best Practice - Koin

- Mniej generowanego kodu
- Łatwiejszy start
- Szybka konfiguracja
- Dobry dla mniejszych projektów
- Łatwiejsze testy

Koin

```
class App : Application() {

    var listofModules =
        listOf(
            MainModule.mainModule
        )

    override fun onCreate() {
        super.onCreate()

        startKoin {
            androidLogger()
            androidContext(this@App)
            modules(listofModules)

        }
    }
}
```

Koin

```
object MainModule {

    val mainModule = module {
        single { WeatherApi(androidContext()) }
        single { provideApiService(get()) }
        single { WeatherRepository(weatherApiService = get()) }
        viewModel { WeatherViewModel(weatherRepository = get()) }
    }

    private fun provideApiService(api: WeatherApi): WeatherApiService {
        return api.getWeatherApiService()
    }
}
```

https://github.com/adamnovicki/StudentsTemplate

Projekt

Projekt

API server: https://jsonplaceholder.typicode.com/

JSONPlaceholder comes with a set of 6 common resources:

/posts
100 posts
/comments
500 comments
/albums
100 albums
/photos
5000 photos
/todos
200 todos
/users
10 users

Zadanie, cz.1

Po uruchomieniu aplikacji wyświetl wszystkie posty dostępne pod API https://jsonplaceholder.typicode.com/posts **postronicowane po 10 postów na stronę***, w następującej strukturze:

Posty:
Nazwa użytkownika (klikalne)
Title
Body
Komentarze (Liczba) (klikalne)

https://jsonplaceholder.typicode.com/posts

Zadanie cz.2

Ekran komentarze - kieruje do niego link z pierwszej strony pod każdym postem.

API /comments

Niech wyświetli:

Name
Email
Body

https://jsonplaceholder.typicode.com/comments

Zadanie cz.3

Pole **Nazwa użytkownika (np.: Leanne Graham)** niech linkuje do kolejnego ekranu określającego użytkownika (API: /users):

Użytkownik
Username,
email,
website,
street,
city,
zipcode,
geo (link do widoku w postaci widoku mapy pokazujące współrzędne - lat, lng),
zdjęcia (link do nowego ekranu)

https://jsonplaceholder.typicode.com/users

/Users


```
{
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
```

Zadanie cz.4

Pole **zdjęcia** na ekranie użytkownika niech linkuje do ekranu wszystkich zdjęć danego użytkownika, które jest możliwość pobrać z API /albums oraz /photos

/photos

```
{
  "albumId": 1,
  "id": 10,
  "title": "beatae et provident et ut vel",
  "url": „https://via.placeholder.com/600/810b14", - duże
  "thumbnailUrl": „https://via.placeholder.com/150/810b14” - małe
}
```

https://jsonplaceholder.typicode.com/albums

https://jsonplaceholder.typicode.com/photos

Zadanie cz.5*

Zaskoczcie czymś nas!

Dziękujemy za uwagę