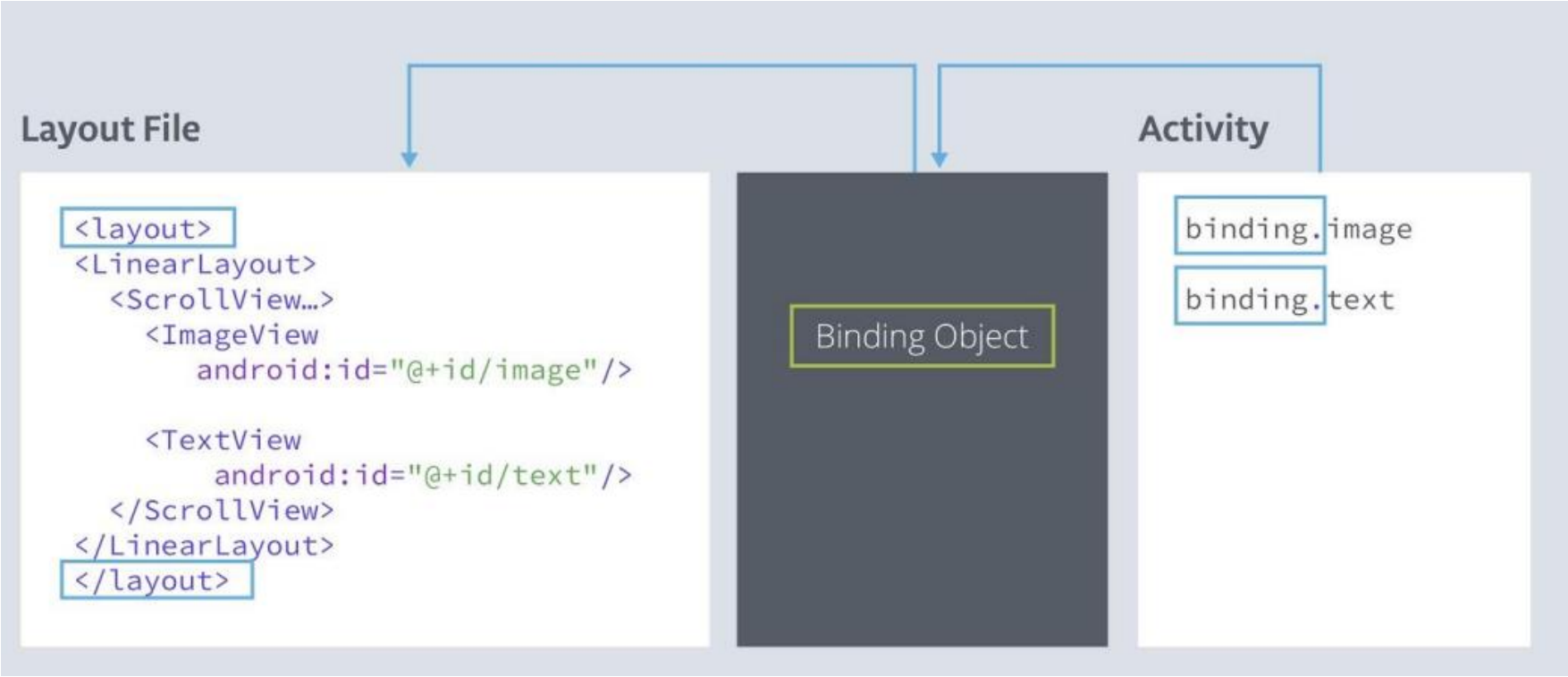
STU FEI
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Mobilné výpočty

Ing. Maroš Čavojský, PhD.

Data Binding



Data Binding

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto">
    <data>
        <variable
            name="viewmodel"
            type="com.myapp.data.ViewModel" />
    </data>
    <ConstraintLayout... /> <!-- UI layout's root element -->
</layout>
```

Data Binding

```
findViewById<TextView>(R.id.sample_text).apply {
    text = viewModel.userName
}
```

```
<TextView
    android:text="@{viewmodel.userName}" />
```

```
<CheckBox
    android:id="@+id/rememberMeCheckBox"
    android:checked="@={viewmodel.rememberMe}"
/>
```

STU
FEI

Data Binding

```
@BindingAdapter("app:goneUnless")
fun goneUnless(view: View, visible: Boolean) {
    view.visibility = if (visible) View.VISIBLE else View.GONE
}
```

STU
FEI

Data Binding

```
android:text="@{String.valueOf(index + 1)}"
android:visibility="@{age > 13 ? View.GONE : View.VISIBLE}"
android:transitionName='@{"image_" + id}'
```

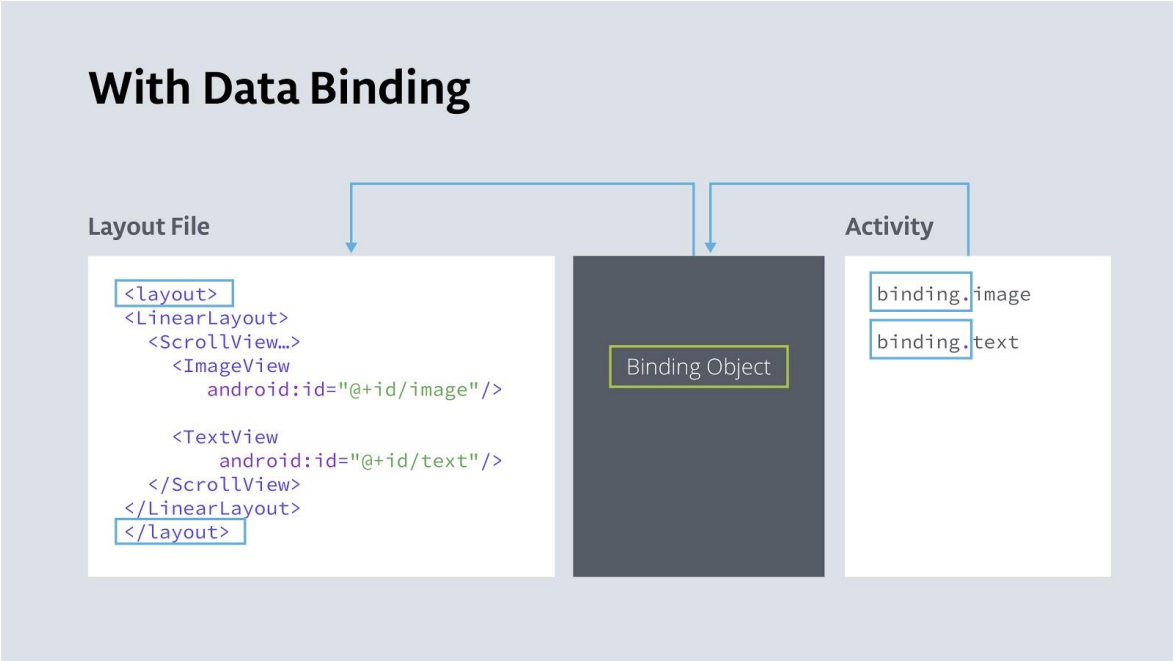
```
android:text="@{user.displayName ?? user.lastName}"
```

```
android:text="@{user.displayName != null ? user.displayName : user.lastName}"
```

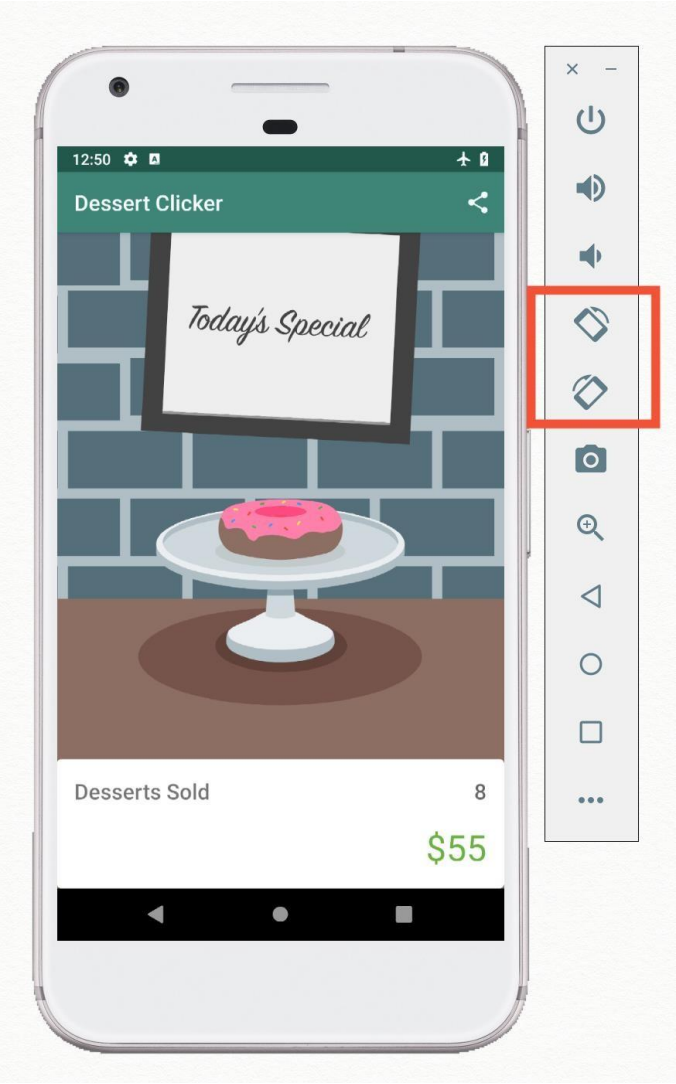
```
android:onCheckedChanged="@{() -> viewmodel.rememberMeChanged()}"
```

STU
FEI

- kód je jednoduchší a na jednom mieste
- dáta a UI je oddelené
- všetky elementy sú nájdené len raz a to pri prvom štarte, a bez následných opakovaní počas behu aplikácie
- type safety pre prístup k Views (kontrola kompilátorom pred spustením)



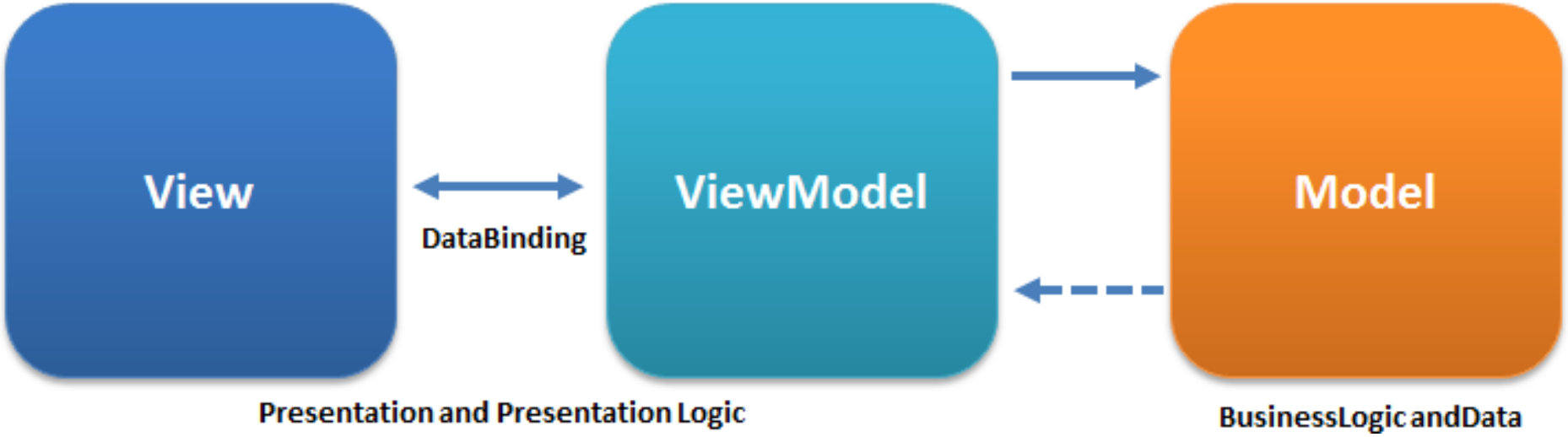
Rotácie zariadenia



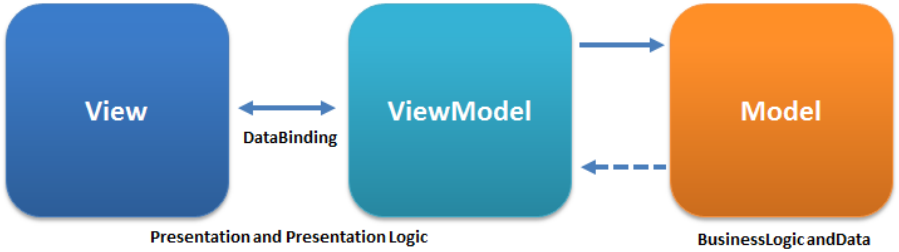
Riešenie

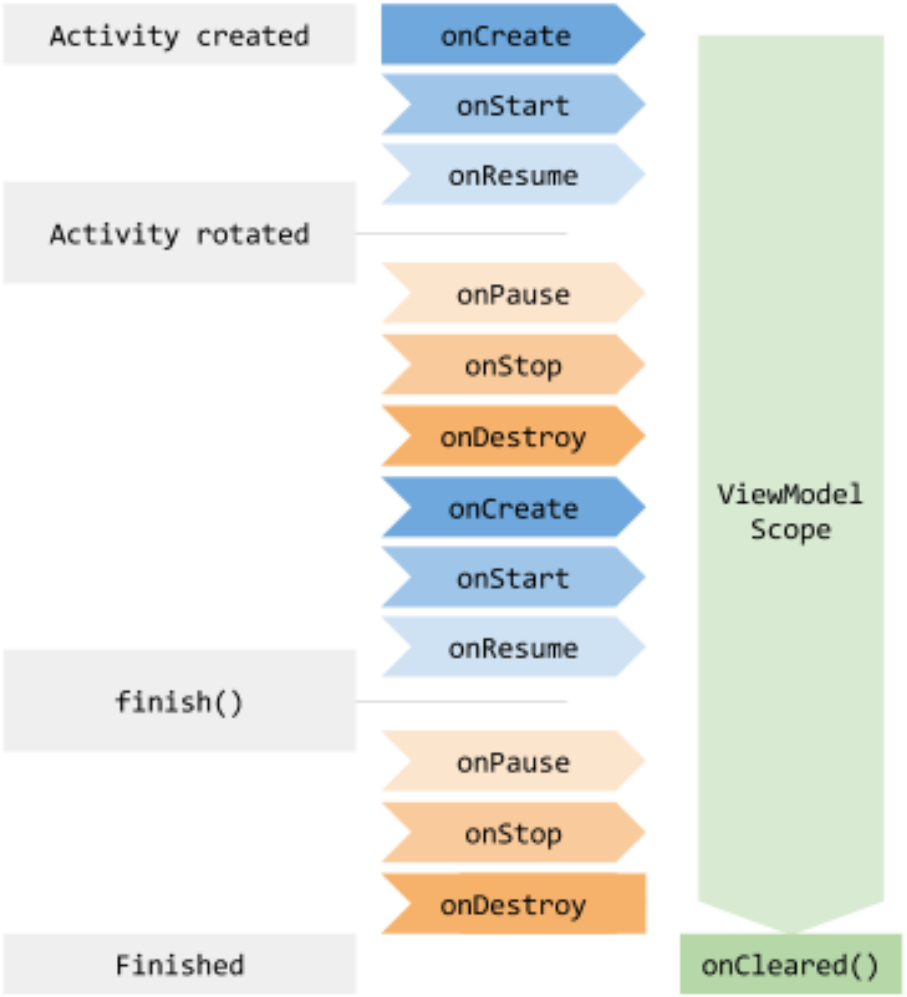
- Bundle
- Shared Preferences
- SQLite Databáza
- Zápis do súboru

Model-view-viewmodel

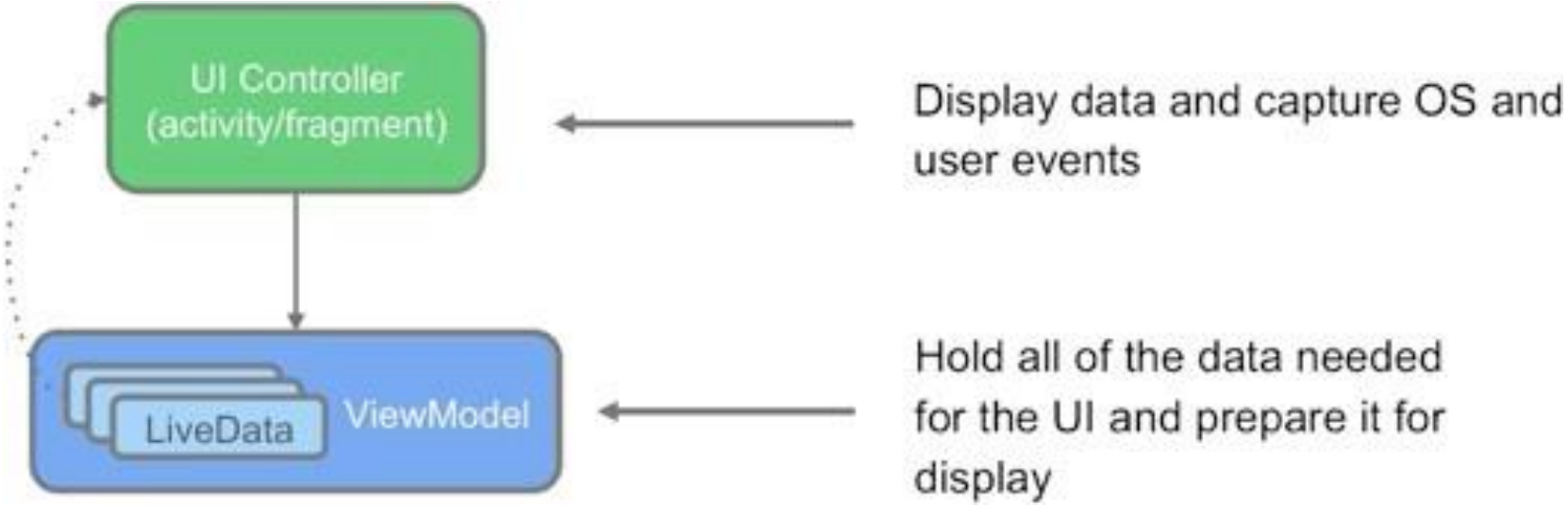


ViewModel


View
DataBinding
ViewModel
Model
Presentation and Presentation Logic
BusinessLogic andData


Activity created
onCreate
onStart
onResume
Activity rotated
onPause
onStop
onDestroy
onCreate
onStart
onResume
finish()
onPause
onStop
onDestroy
Finished
ViewModel Scope
onCleared()

Zodpovednosť



LiveData

- Zabezpečia, že UI bude zodpovedať aktuálnym dátam - Observer
- Žiadne úniky pamäte (Memory leaks) - Lifecycle
- Žiadne pády aplikácie, keď je aplikácia minimalizovaná
- Žiadne ošetrovanie životného cyklu – LiveData sú už Lifecycle aware
- Vždy aktuálne – po obnovení aktivity sa údaje obnovia
- Správne ošetrenie pri zmene konfigurácii
- Zdieľanie dát

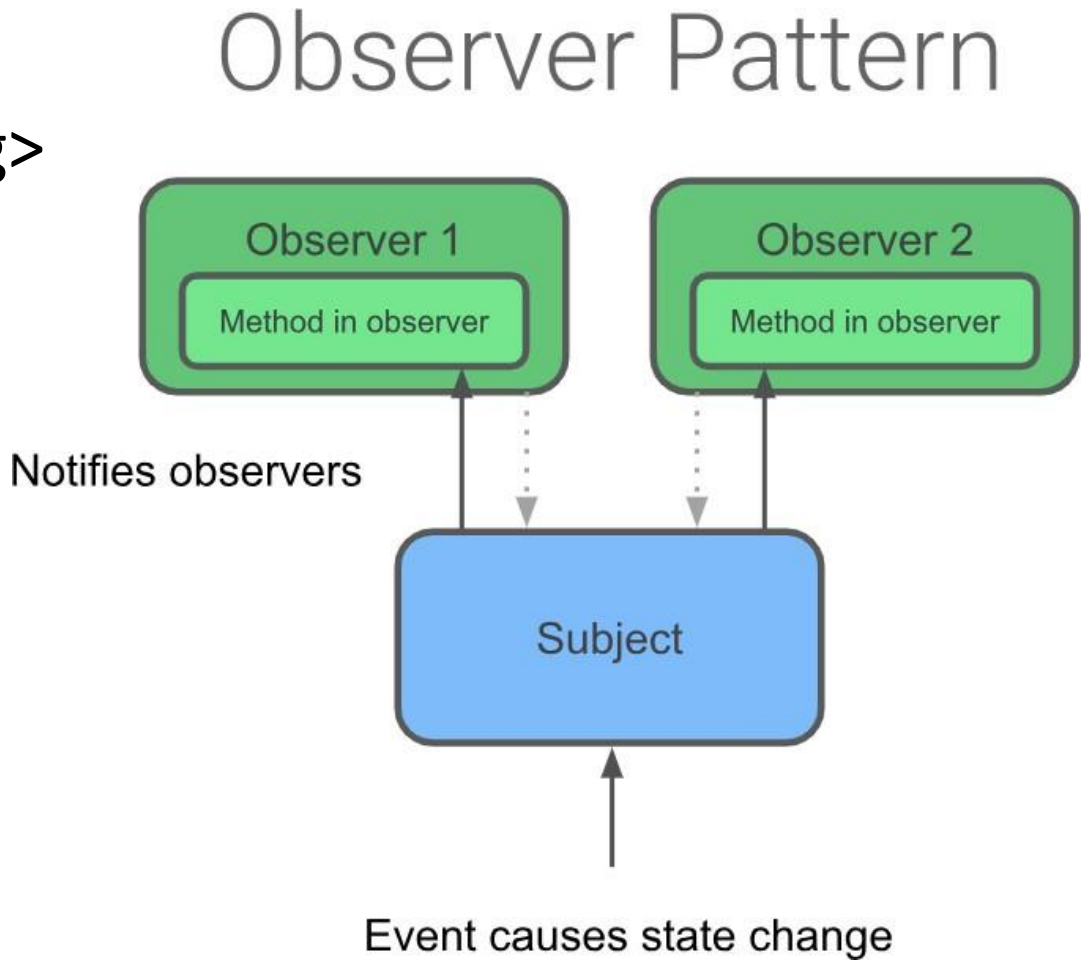
Obsah vo ViewModel

- Dáta a ich načítavanie a zapisovanie
- LiveData … String … LiveData<String>
- MutableLiveData<String>

Obsah vo ViewModel

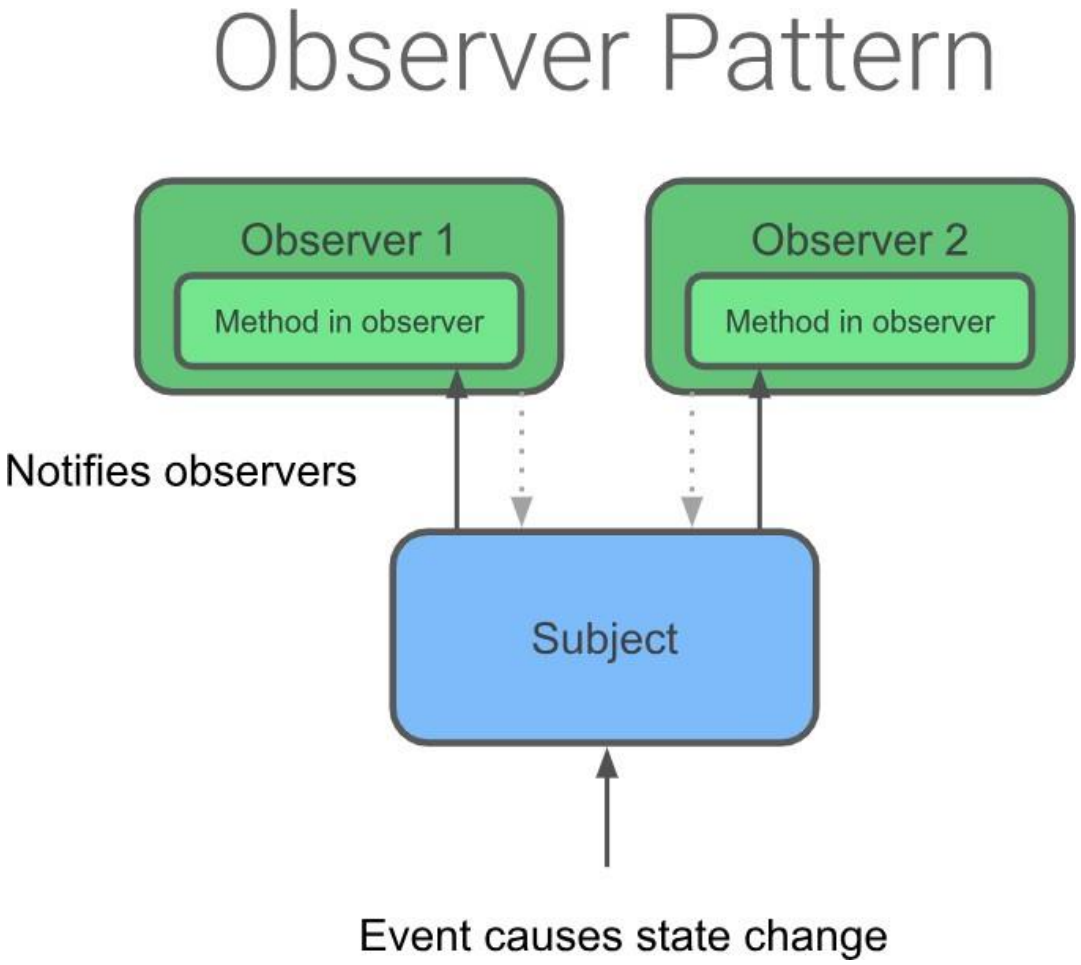
- Dáta a ich načítavanie a zapisovanie
- LiveData … String … LiveData<String>
- MutableLiveData<String>

- setValue() - v hlavnom vlákne
- postValue() – v pracovnom vlákne



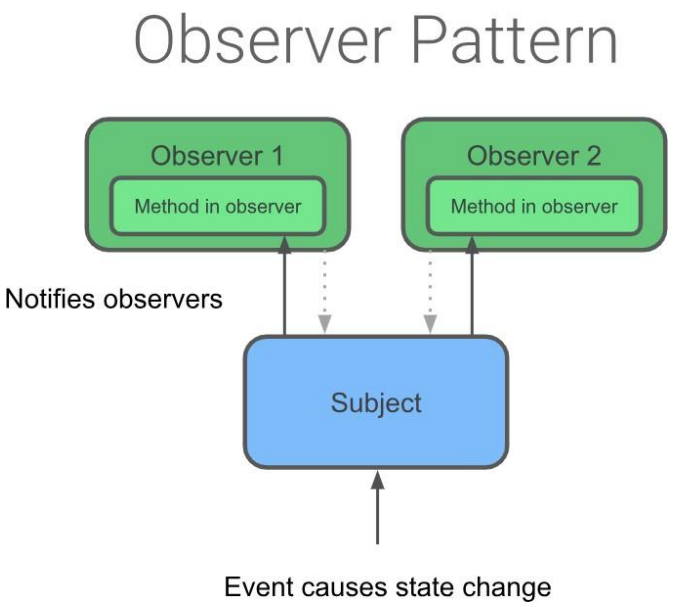
Obsah vo ViewModel

```
class NameViewModel : ViewModel() {

    // Create a LiveData with a String
    val currentName: MutableLiveData<String> by lazy {
        MutableLiveData<String>()
    }

    // Rest of the ViewModel...
}
```



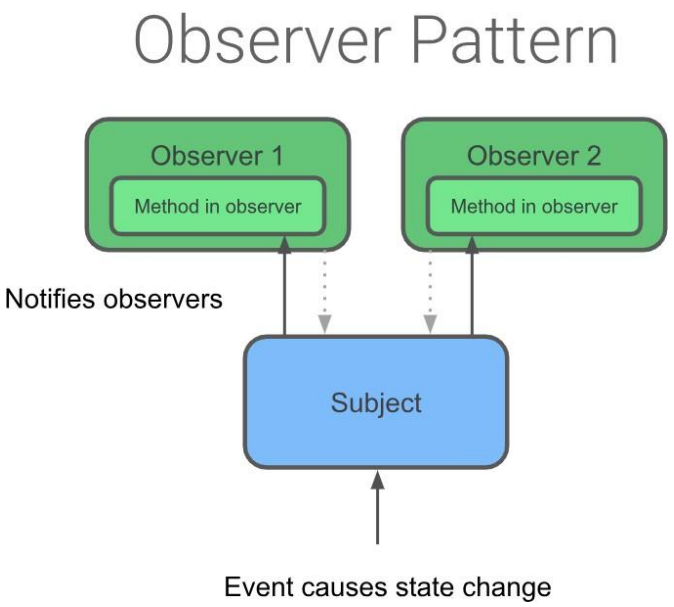
Obsah vo ViewModel

```
class NameActivity : AppCompatActivity() {

    private lateinit var model: NameViewModel

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

        // Other code to setup the activity...

        // Get the ViewModel.
        model = ViewModelProviders.of(this).get(NameViewModel::class.java)


        // Create the observer which updates the UI.
        val nameObserver = Observer<String> { newName ->
            // Update the UI, in this case, a TextView.
            nameTextView.text = newName
        }

        // Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.
        model.currentName.observe(this, nameObserver)
    }
}
```



Obsah vo ViewModel

```
class NameActivity : AppCompatActivity() {

    private lateinit var model: NameViewModel

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

        // Other code to setup the activity...

        // Get the ViewModel.
        model = ViewModelProviders.of(this).get(NameViewModel::class.java)


        // Create the observer which updates the UI.
        val nameObserver = Observer<String> { newName ->
            // Update the UI, in this case, a TextView.
            nameTextView.text = newName
        }

        // Observe the LiveData, passing in this activity as the LifecycleOwner and the observer.
        model.currentName.observe(this, nameObserver)
    }
}
```



Transformácie

```
val userLiveData: LiveData<User> = UserLiveData()
val userName: LiveData<String> = Transformations.map(userLiveData) {
    user -> "${user.name} ${user.lastName}"
}
```

```
private fun getUser(id: String): LiveData<User> {
  ...
}
val userId: LiveData<String> = ...
val user = Transformations.switchMap(userId) { id -> getUser(id) }
```

Transformácie

```
class MyViewModel(private val repository: PostalCodeRepository) : ViewModel() {

    private fun getPostalCode(address: String): LiveData<String> {
        // DON'T DO THIS
        return repository.getPostCode(address)
    }
}
```

```
class MyViewModel(private val repository: PostalCodeRepository) : ViewModel() {
    private val addressInput = MutableLiveData<String>()
    val postalCode: LiveData<String> = Transformations.switchMap(addressInput) {
            address -> repository.getPostCode(address) }


    private fun setInput(address: String) {
        addressInput.value = address
    }
}
```

Enkapsulácia (zabalenie) LiveData

```
private val _word = MutableLiveData<String>()
val word: LiveData<String>
    get() = _word
```

ViewModel data binding

ViewModel data binding

```
binding.correctButton.setOnClickListener { onCorrect() }
binding.skipButton.setOnClickListener { onSkip() }
binding.endGameButton.setOnClickListener { onEndGame() }

/** Methods for buttons presses **/
private fun onSkip() {
   viewModel.onSkip()
}
private fun onCorrect() {
   viewModel.onCorrect()
}
private fun onEndGame() {
   gameFinished()
}
```

ViewModel data binding

ViewModel data binding

```
<layout ...>

    <data>

        <variable
            name="gameViewModel"
            type="com.example.android.guesstheword.screens.game.GameViewModel" />
    </data>
```

```
// Set the viewmodel for databinding - this allows the bound layout access
// to all the data in the ViewModel
binding.gameViewModel = viewModel
```

```
<Button
    android:id="@+id/skip_button"
    ...
    android:onClick="@{() -> gameViewModel.onSkip()}"
    ... />
```

ViewModel data binding

```
<layout ...>

   <data>

       <variable
           name="gameViewModel"
           type="com.example.android.guesstheword.screens.game.GameViewModel" />
   </data>
```

```
<TextView
   android:id="@+id/word_text"
   ...
   android:text="@{gameViewModel.word}"
   ... />
```

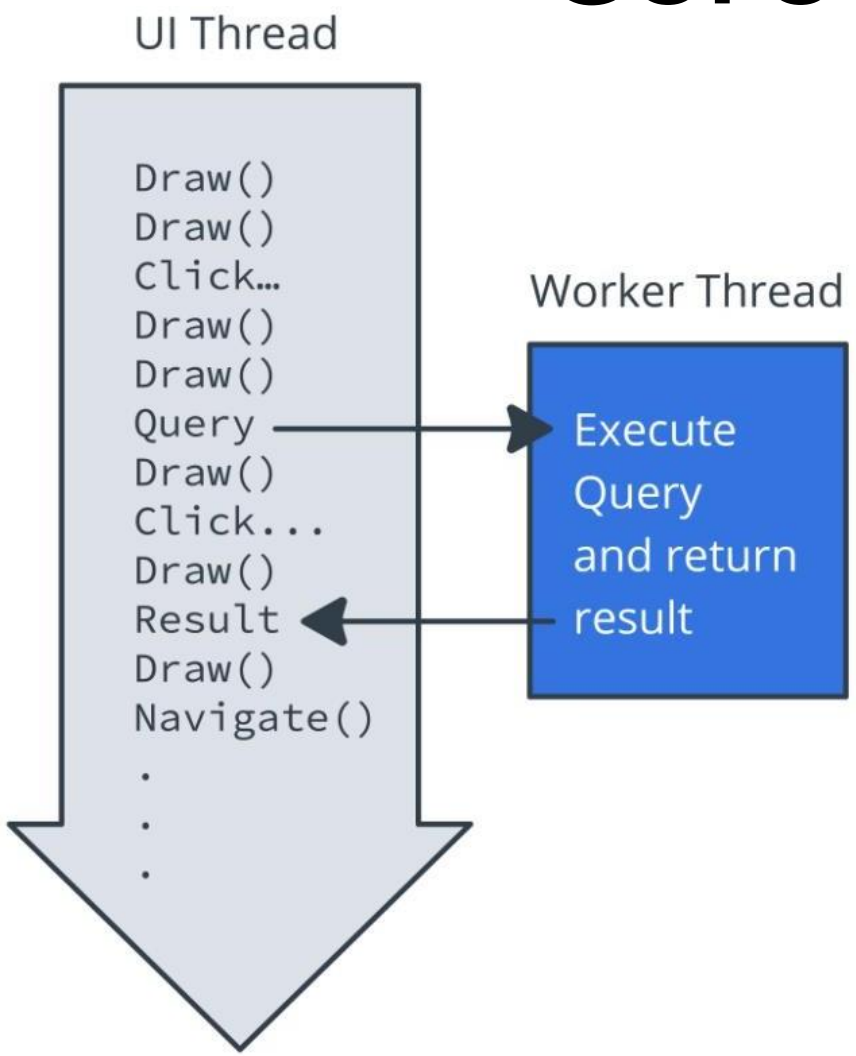
```
<TextView
   android:id="@+id/score_text"
   ...
   android:text="@{String.valueOf(scoreViewModel.score)}"
   ... />
```

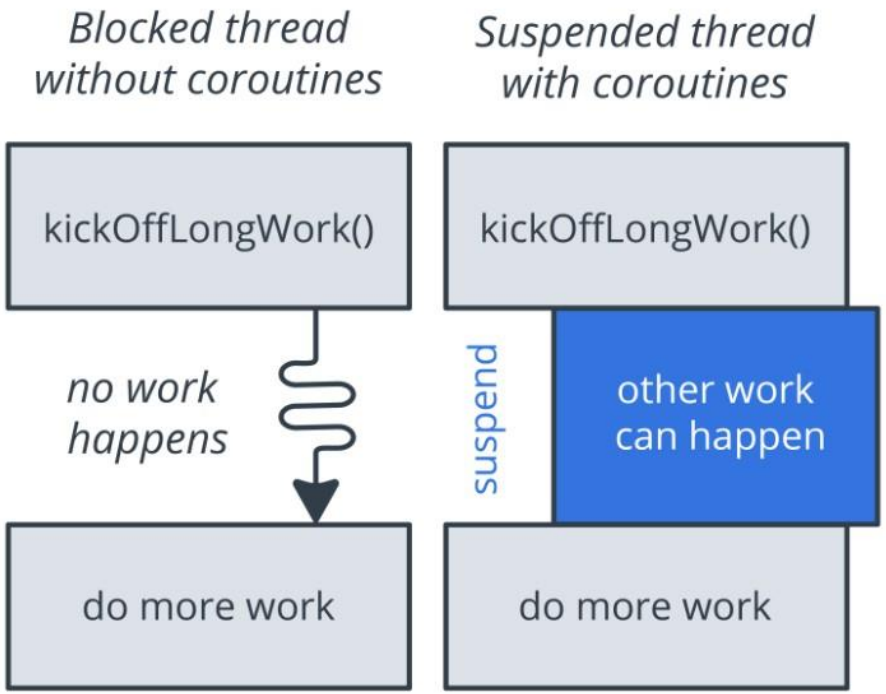
```
binding.gameViewModel = ...
// Specify the current activity as the lifecycle owner of the binding.
// This is used so that the binding can observe LiveData updates
binding.lifecycleOwner = this
```

ViewModel data binding

```
/** Setting up LiveData observation relationship **/
viewModel.word.observe(this, Observer { newWord ->
    binding.wordText.text = newWord
})
```

Coroutines + Room





Získanie údajov z Internetu

```
interface WebApi {
    @GET("realestate")
    suspend fun getProperties(): Response<List<MarsResponse>>

    companion object {
        private const val BASE_URL =
            "https://android-kotlin-fun-mars-server.appspot.com"

        fun create(context: Context): WebApi {

            val client = OkHttpClient.Builder()
                .build()

            val retrofit = Retrofit.Builder()
                .baseUrl(BASE_URL)
                .client(client)
                .addConverterFactory(GsonConverterFactory.create())
                .build()

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

}
```

Získanie údajov z Internetu

```
suspend fun loadMars(onError: (error: String) -> Unit) {

    try {
        val response = api.getProperties()
        if (response.isSuccessful) {
            response.body()?.let {
                return cache.insertImages(it.map { item ->
                            MarsItem(item.price, item.id, item.type, item.img_src)
                            })
            }
        }
        onError("Load images failed. Try again later please.")
    } catch (ex: ConnectException) {
        onError("Off-line. Check internet connection.")
        ex.printStackTrace()
        return
    } catch (ex: Exception) {
        onError("Oops...Change failed. Try again later please.")
        ex.printStackTrace()
        return
    }
}
```

Získanie údajov z Internetu

```
class MarsViewModel(private val repository: DataRepository) : ViewModel() {

    val error: MutableLiveData<String> = MutableLiveData()

    val images: LiveData<List<MarsItem>>
        get() = repository.getMars()

    init {
      loadMars()
    }

    fun loadMars() {
        viewModelScope.launch {
            repository.loadMars { error.postValue(it) }
        }
    }
}
```

SharedPreferences

mechanizmus v systéme Android na ukladanie jednoduchých párov kľúč-hodnota a je bežne používaný na ukladanie primitívnych typov dát. Táto možnosť ukladania je vhodná na ukladanie malých množstiev dát, ako sú nastavenia používateľa, konfigurácie aplikácií alebo používateľskej relácie.

K SharedPreferences môžete pristupovať prostredníctvom ľubovoľného kontextu v systéme Android (napríklad Activity, Service alebo BroadcastReceiver). Najčastejšie sa však používa metóda getSharedPreferences() alebo getPreferences().

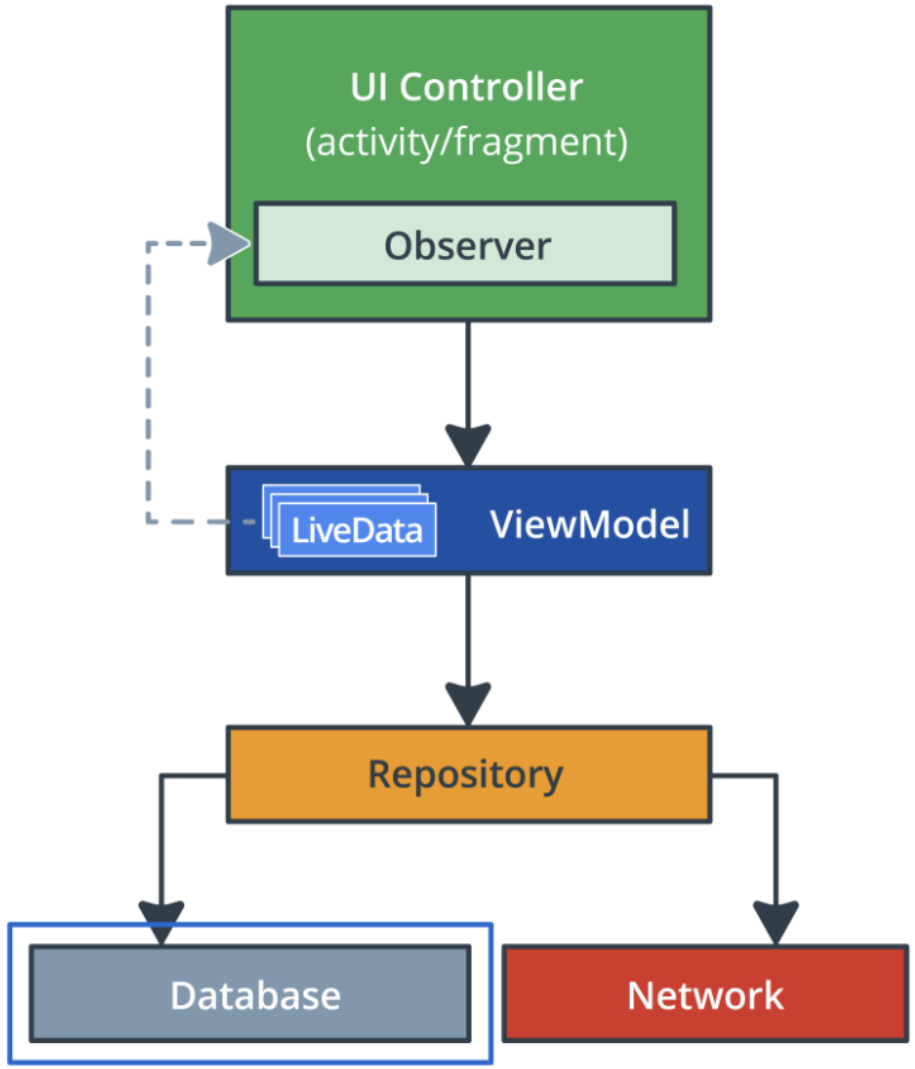
SharedPreferences

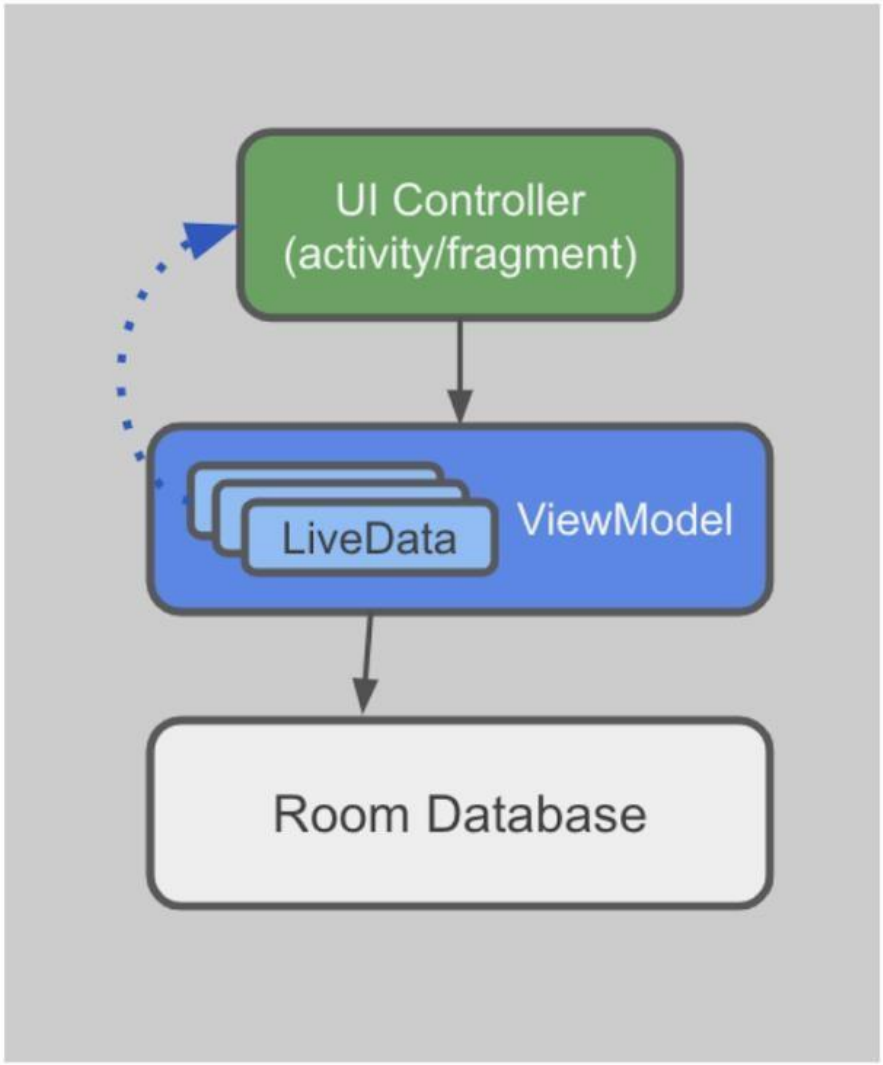
```
class PreferenceData private constructor() {

    private fun getSharedPreferences(context: Context?): SharedPreferences? {
        return context?.getSharedPreferences(
            shpKey, Context.MODE_PRIVATE
        )
    }

    companion object {
        @Volatile
        private var INSTANCE: PreferenceData? = null

        private val lock = Any()

        fun getInstance(): PreferenceData =
            INSTANCE ?: synchronized(lock) {
                INSTANCE
                    ?: PreferenceData().also { INSTANCE = it }
            }

        private const val shpKey = "eu.mcomputing.mobv.zadanie"
        private const val userKey = "userKey"

    }

}
```

SharedPreferences

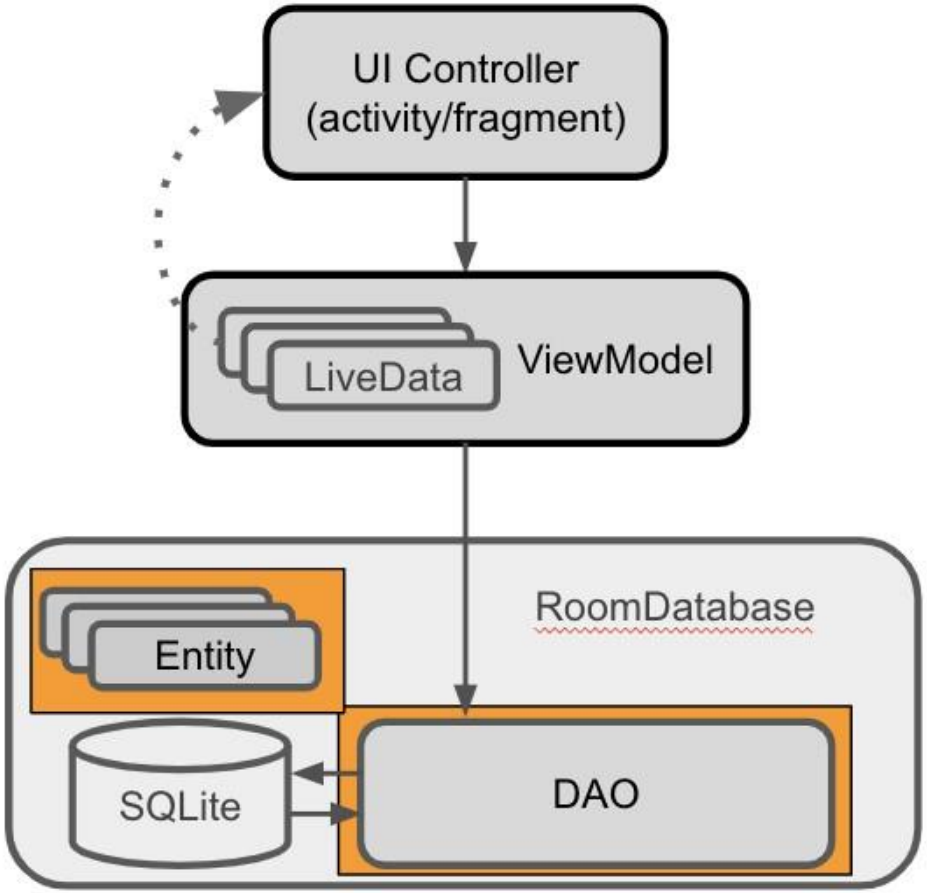
```
class PreferenceData private constructor() {
 ...

    fun clearData(context: Context?) {
        val sharedPref = getSharedPreferences(context) ?: return
        val editor = sharedPref.edit()
        editor.clear()
        editor.apply()
    }

    fun putUser(context: Context?, user: User?) {
        val sharedPref = getSharedPreferences(context) ?: return
        val editor = sharedPref.edit()
        user?.toJson()?.let {
            editor.putString(userKey, it)
        } ?: editor.remove(userKey)

        editor.apply()
    }

    fun getUser(context: Context?): User? {
        val sharedPref = getSharedPreferences(context) ?: return null
        val json = sharedPref.getString(userKey, null) ?: return null

        return User.fromJson(json)
    }
...
}
```

SQLite databáza - Room

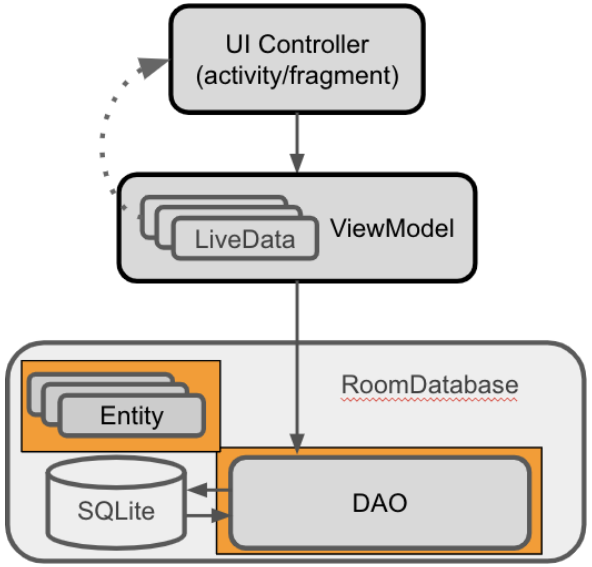


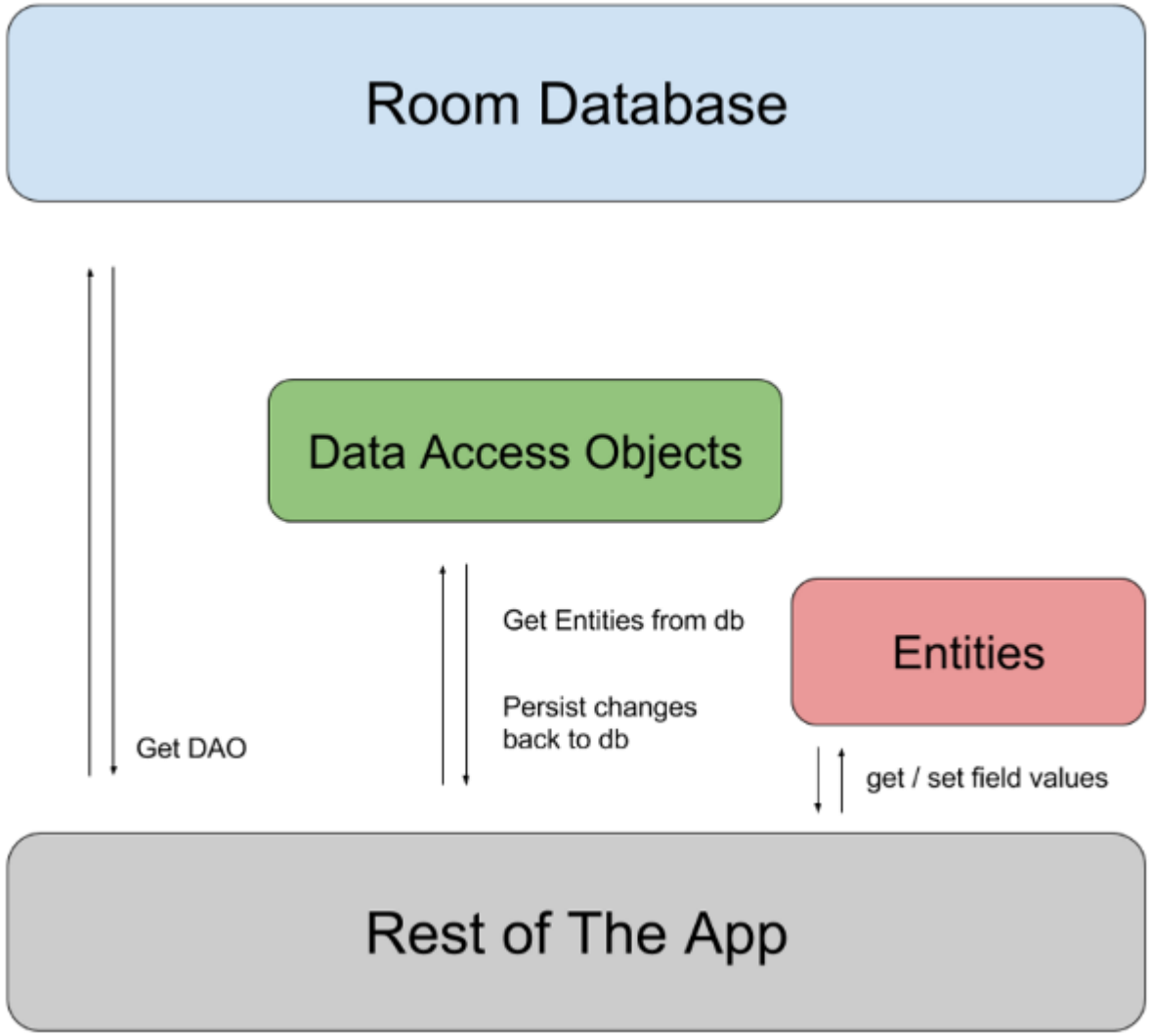


SQLite databáza - Room



SQLite databáza - Room

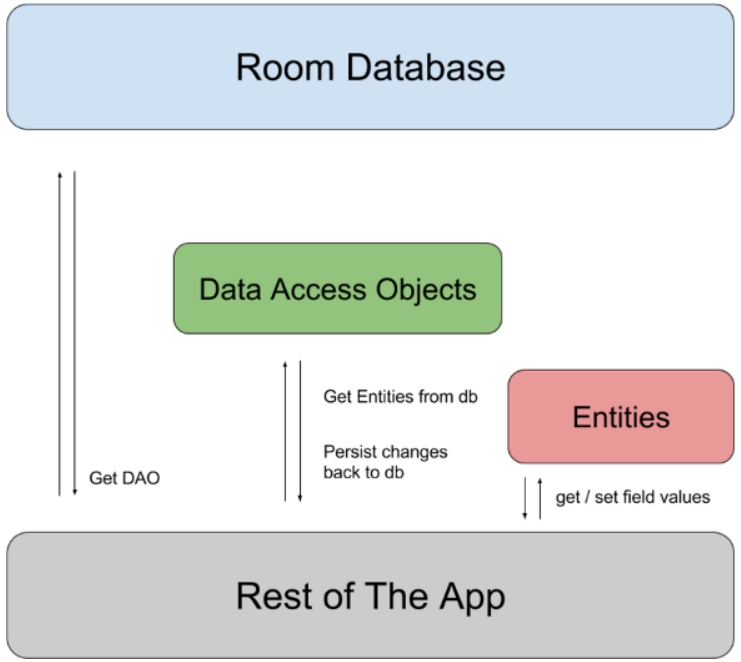




SQLite databáza - Room

```
@Database(entities = arrayOf(User::class), version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): UserDao
}
```

```
@Entity
data class User(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```




```
@Dao
interface UserDao {
    @Query("SELECT * FROM user")
    fun getAll(): List<User>

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    fun loadAllByIds(userIds: IntArray): List<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :first AND " +
           "last_name LIKE :last LIMIT 1")
    fun findByName(first: String, last: String): User

    @Insert
    fun insertAll(vararg users: User)

    @Delete
    fun delete(user: User)
}
```

SQLite databáza - Room

```
@Entity(tableName = "users")
data class User (
    @PrimaryKey val id: Int,
    @ColumnInfo(name = "first_name") val firstName: String?,
    @ColumnInfo(name = "last_name") val lastName: String?
)
```

```
@Entity
data class User(
    @PrimaryKey val id: Int,
    val firstName: String?,
    val lastName: String?,
    @Ignore val picture: Bitmap?
)
```

- https://developer.android.com/training/data-storage/room/defining-data

SQLite databáza - Room

```
@Entity(foreignKeys = arrayOf(ForeignKey(
            entity = User::class,
            parentColumns = arrayOf("id"),
            childColumns = arrayOf("user_id"))
        )
)
data class Book(
    @PrimaryKey val bookId: Int,
    val title: String?,
    @ColumnInfo(name = "user_id") val userId: Int
)
```

- https://developer.android.com/training/data-storage/room/relationships

Room - M:N vzťah

```
@Entity
data class Playlist(
    @PrimaryKey var id: Int,
    val name: String?,
    val description: String?
)

@Entity
data class Song(
    @PrimaryKey var id: Int,
    val songName: String?,
    val artistName: String?
)
```

```
@Entity(tableName = "playlist_song_join",
        primaryKeys = arrayOf("playlistId","songId"),
        foreignKeys = arrayOf(
                        ForeignKey(entity = Playlist::class,
                                   parentColumns = arrayOf("id"),
                                   childColumns = arrayOf("playlistId")),
                        ForeignKey(entity = Song::class,
                                   parentColumns = arrayOf("id"),
                                   childColumns = arrayOf("songId"))
                            )
        )
data class PlaylistSongJoin(
    val playlistId: Int,
    val songId: Int
)
```

- https://developer.android.com/training/data-storage/room/relationships

SQLite databáza - Room

```
@Dao
interface MyDao {
    @Query("SELECT * FROM user WHERE age BETWEEN :minAge AND :maxAge")
    fun loadAllUsersBetweenAges(minAge: Int, maxAge: Int): Array<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :search " +
           "OR last_name LIKE :search")
    fun findUserWithName(search: String): List<User>
```

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
fun insertUsers(vararg users: User)
```

```
@Update
fun updateUsers(vararg users: User)
```

```
@Delete
fun deleteUsers(vararg users: User)
```

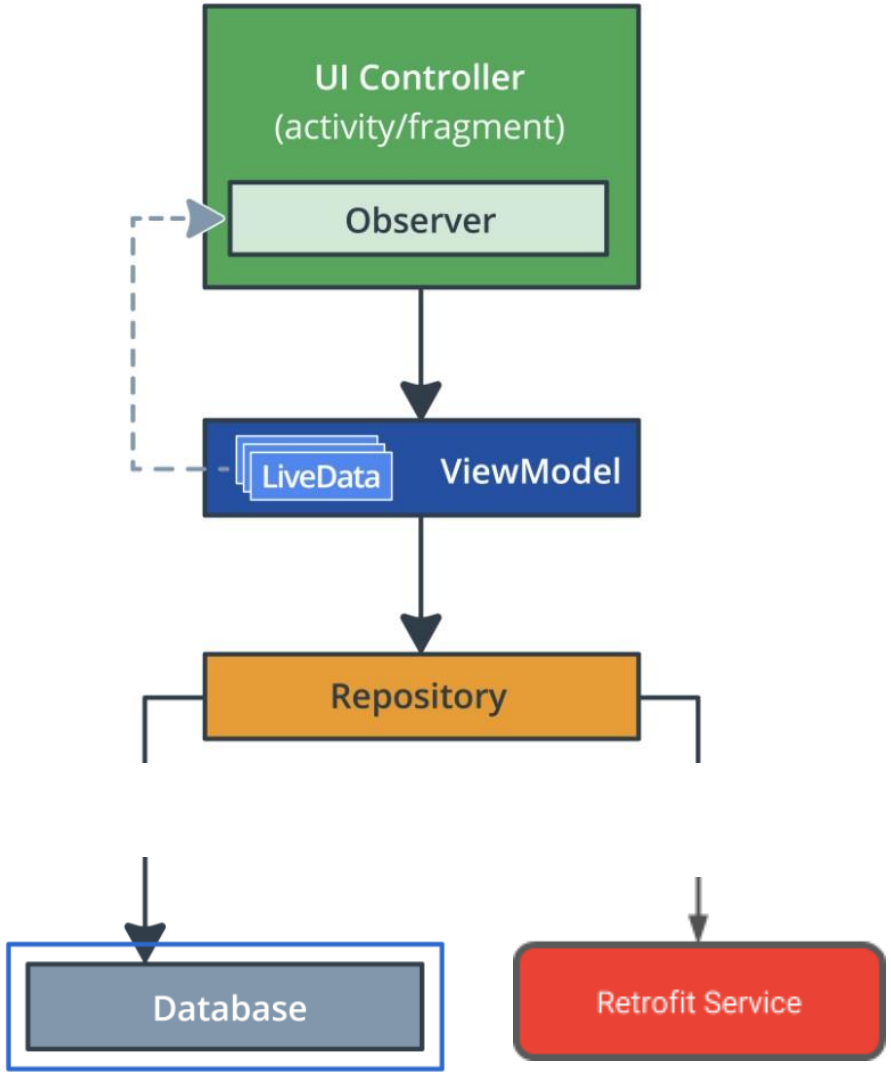
SQLite databáza - Room

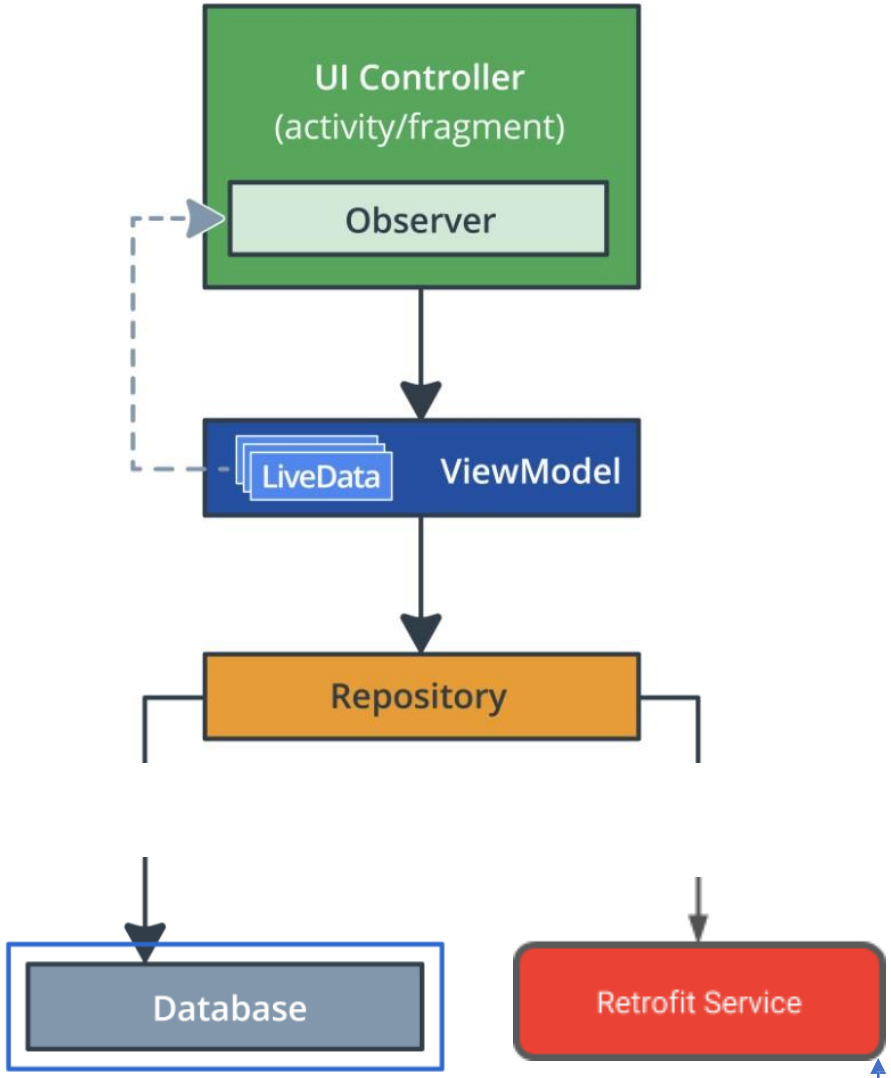
```
@Dao
interface MyDao {
    @Query("SELECT * FROM user WHERE age BETWEEN :minAge AND :maxAge")
    fun loadAllUsersBetweenAges(minAge: Int, maxAge: Int): Array<User>

    @Query("SELECT * FROM user WHERE first_name LIKE :search " +
           "OR last_name LIKE :search")
    fun findUserWithName(search: String): List<User>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertUsers(vararg users: User)

    @Update
    fun updateUsers(vararg users: User)

    @Delete
    fun deleteUsers(vararg users: User)
```

NA POZADÍ

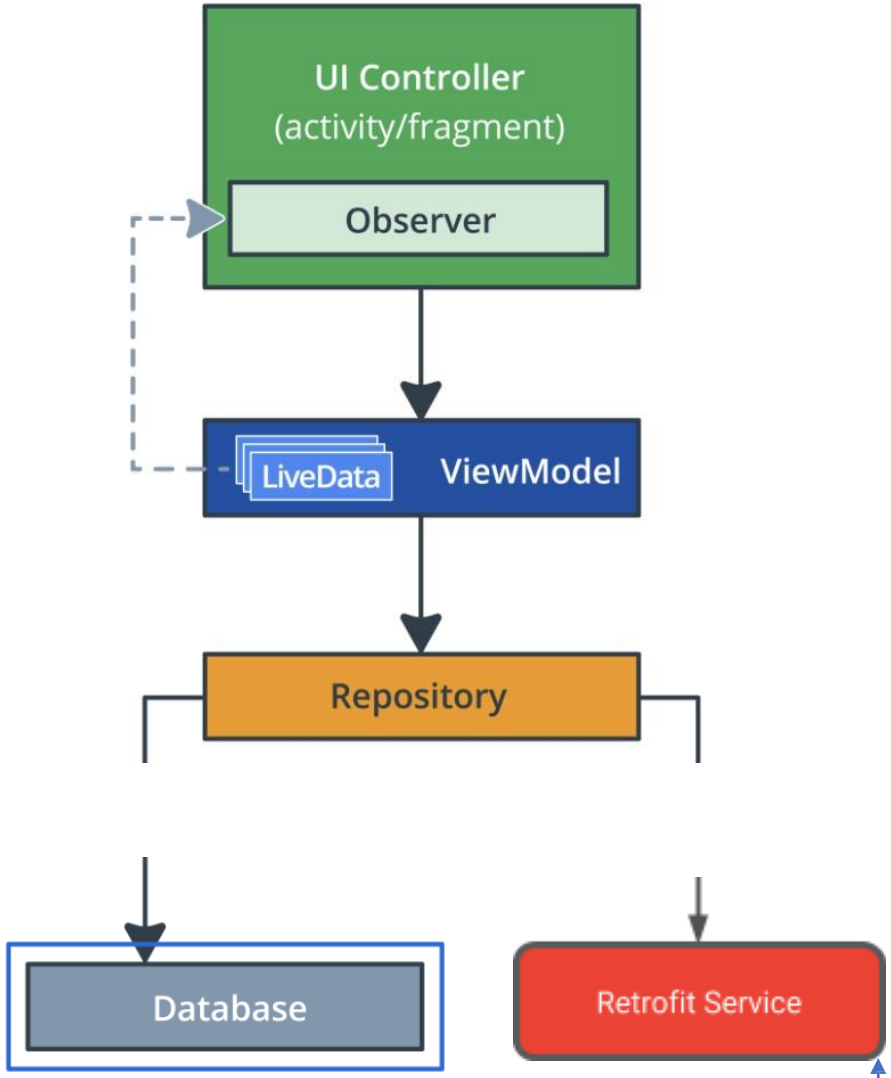
Coroutines + Room

```
@Dao
interface MyDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUsers(vararg users: User)

    @Update
    suspend fun updateUsers(vararg users: User)

    @Delete
    suspend fun deleteUsers(vararg users: User)

    @Query("SELECT * FROM user")
    suspend fun loadAllUsers(): Array<User>
}
```


UI Controller
(activity/fragment)
Observer
LiveData
ViewModel
Repository
Database
Retrofit Service

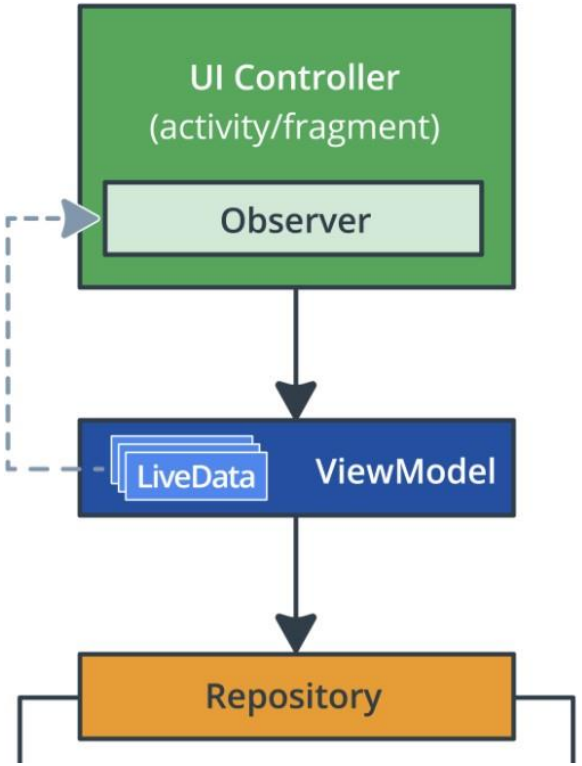



```
@GET("realestate")
    suspend fun getProperties():
Response<List<MarsResponse>>
```




```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertImages(items:List<MarsItem>)

@Query("SELECT * FROM images")
fun getImages(): LiveData<List<MarsItem>>
```

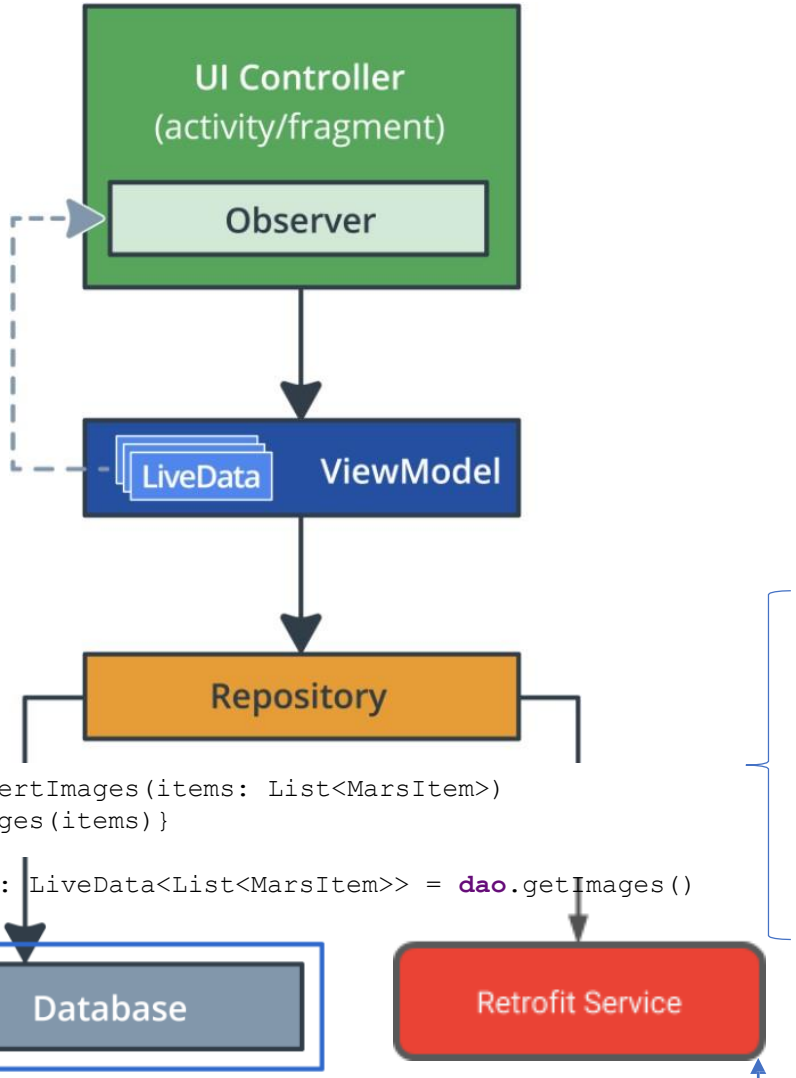
```
@GET("realestate")
    suspend fun getProperties():
Response<List<MarsResponse>>
```




```
suspend fun insertImages(items: List<MarsItem>)
{ dao.insertImages(items)}

fun getImages(): LiveData<List<MarsItem>> = dao.getImages()
```

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertImages(items:List<MarsItem>)

@Query("SELECT * FROM images")
fun getImages(): LiveData<List<MarsItem>>
```

```
@GET("realestate")
    suspend fun getProperties():
Response<List<MarsResponse>>
```

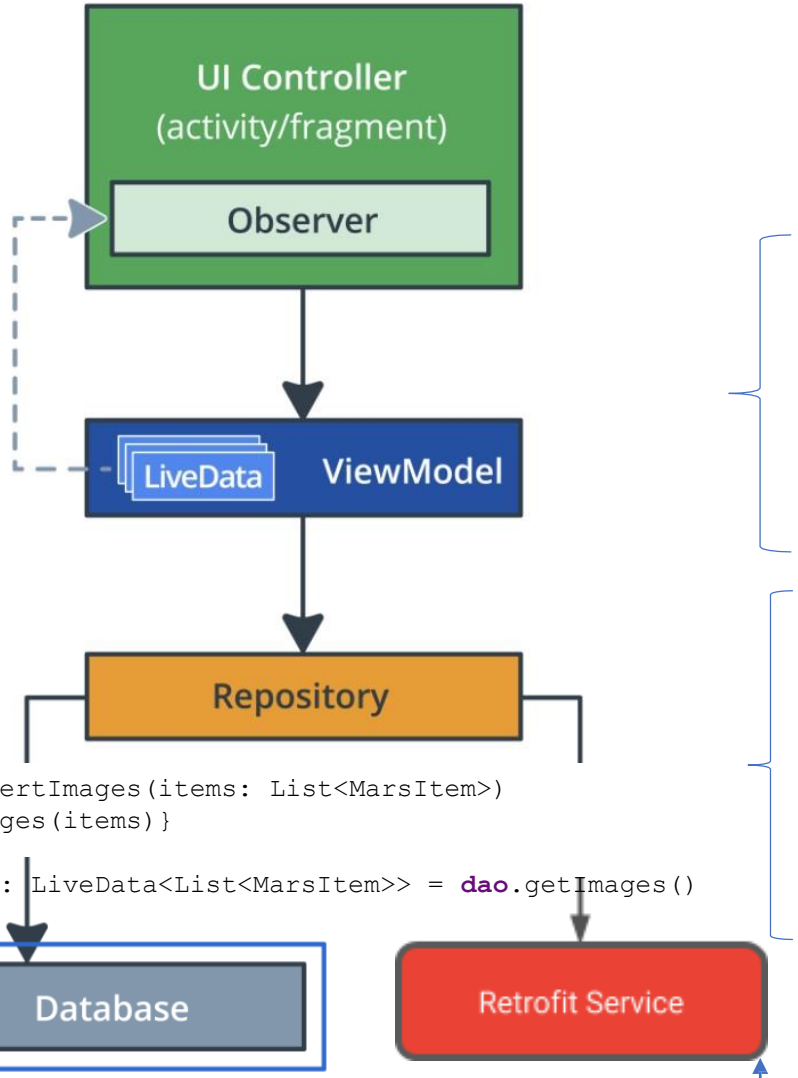



```
suspend fun insertImages(items: List<MarsItem>)
{ dao.insertImages(items)}

fun getImages(): LiveData<List<MarsItem>> = dao.getImages()
```

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertImages(items:List<MarsItem>)

@Query("SELECT * FROM images")
fun getImages(): LiveData<List<MarsItem>>
```

```
fun getMars(): LiveData<List<MarsItem>> = cache.getImages()
suspend fun loadMars(
        onError: (error: String) -> Unit)

{...
cache.insertImages(it.map { item ->
     MarsItem(item.price, item.id, item.type, item.img_src)
    })
 ...}
```

```
@GET("realestate")
    suspend fun getProperties():
Response<List<MarsResponse>>
```



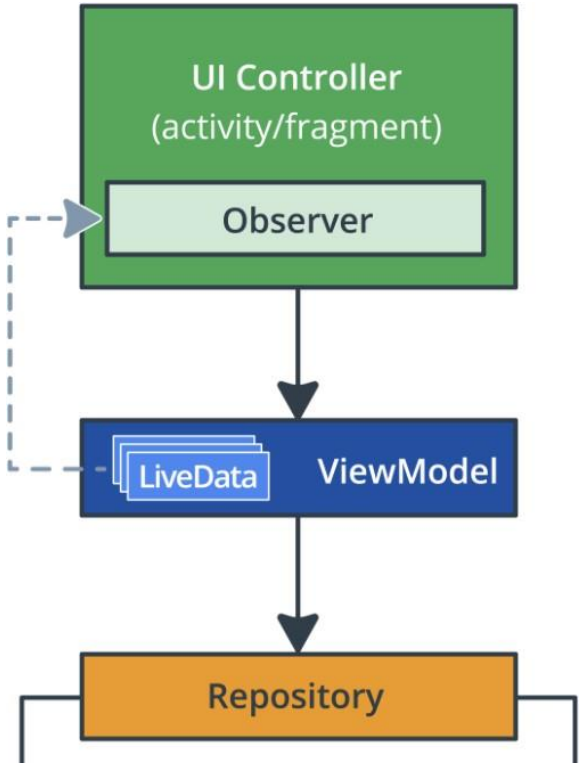

```
suspend fun insertImages(items: List<MarsItem>)
{ dao.insertImages(items)}

fun getImages(): LiveData<List<MarsItem>> = dao.getImages()
```

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertImages(items:List<MarsItem>)

@Query("SELECT * FROM images")
fun getImages(): LiveData<List<MarsItem>>
```

```
val images: LiveData<List<MarsItem>>
    get() = repository.getMars()

fun loadMars() {
    viewModelScope.launch {
        repository.loadMars { error.postValue(it) }
    }
}
```

```
fun getMars(): LiveData<List<MarsItem>> = cache.getImages()
suspend fun loadMars(
        onError: (error: String) -> Unit)

{...
cache.insertImages(it.map { item ->
     MarsItem(item.price, item.id, item.type, item.img_src)
    })
 ...}
```

```
@GET("realestate")
    suspend fun getProperties():
Response<List<MarsResponse>>
```




```
marsViewModel.images.observe(this) { adapter.data = it }

marsViewModel.error.observe(this){
        Toast.makeText(context,it,Toast.LENGTH_SHORT).show()
}
```

```
val images: LiveData<List<MarsItem>>
    get() = repository.getMars()

fun loadMars() {
    viewModelScope.launch {
        repository.loadMars { error.postValue(it) }
    }
}
```

```
fun getMars(): LiveData<List<MarsItem>> = cache.getImages()
suspend fun loadMars(
        onError: (error: String) -> Unit)

{...
cache.insertImages(it.map { item ->
     MarsItem(item.price, item.id, item.type, item.img_src)
    })
 ...}
```

```
suspend fun insertImages(items: List<MarsItem>)
{ dao.insertImages(items)}

fun getImages(): LiveData<List<MarsItem>> = dao.getImages()
```

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertImages(items:List<MarsItem>)

@Query("SELECT * FROM images")
fun getImages(): LiveData<List<MarsItem>>
```

```
@GET("realestate")
    suspend fun getProperties():
Response<List<MarsResponse>>
```

WorkManager

- Android Jetpack architecture component
- Recommended solution to execute background work (immediate or deferred)
- Opportunistic and guaranteed execution
- Execution can be based on certain conditions

This work is licensed under the Apache 2 license.

Declare WorkManager dependencies

```
implementation "androidx.work:work-runtime-ktx:$work_version"
```


This work is licensed under the Apache 2 license.

Important classes to know

- `Worker` - does the work on a background thread, override `doWork()` method
- `WorkRequest` - request to do some work
- `Constraint` - conditions on when the work can run
- `WorkManager` - schedules the `WorkRequest` to be run

This work is licensed under the Apache 2 license.

Define the work

```
class UploadWorker(appContext: Context, workerParams: WorkerParameters) :
        Worker(appContext, workerParams) {

    override fun doWork(): Result {

        // Do the work here. In this case, upload the images.
        uploadImages()

        // Indicate whether work finished successfully with the Result
        return Result.success()
    }
}
```


This work is licensed under the Apache 2 license.

Extend CoroutineWorker instead of Worker

```
class UploadWorker(appContext: Context, workerParams: WorkerParameters) :
        CoroutineWorker(appContext, workerParams) {

    override suspend fun doWork(): Result {

        // Do the work here (in this case, upload the images)
        uploadImages()

        // Indicate whether work finished successfully with the Result
        return Result.success()
    }
}
```


This work is licensed under the Apache 2 license.

WorkRequests

- Can be scheduled to run once or repeatedly
  - `OneTimeWorkRequest`
  - `PeriodicWorkRequest`
- Persisted across device reboots
- Can be chained to run sequentially or in parallel
- Can have constraints under which they will run

This work is licensed under the Apache 2 license.

Schedule a OneTimeWorkRequest

Create `WorkRequest`:

```
val uploadWorkRequest: WorkRequest =
   OneTimeWorkRequestBuilder<UploadWorker>()
       .build()
```

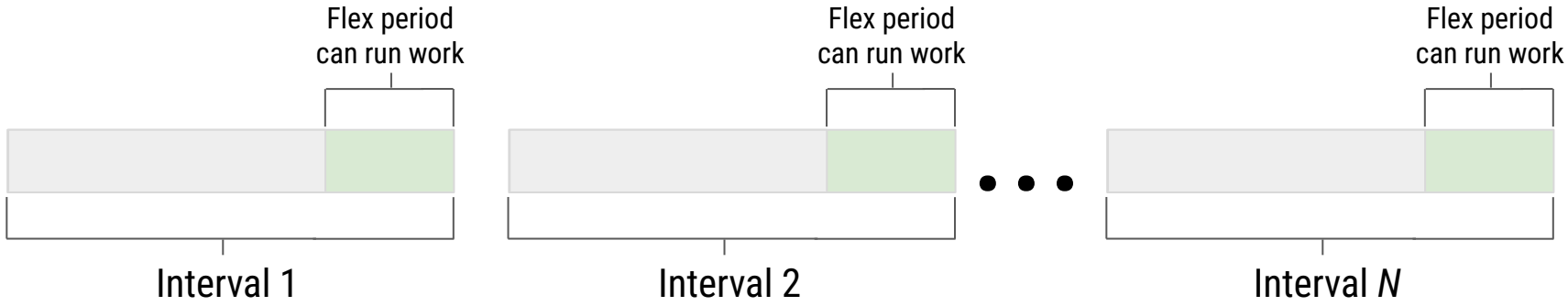
Add the work to the `WorkManager` queue:

```
WorkManager.getInstance(myContext)
    .enqueue(uploadWorkRequest)
```


This work is licensed under the Apache 2 license.

Schedule a PeriodicWorkRequest

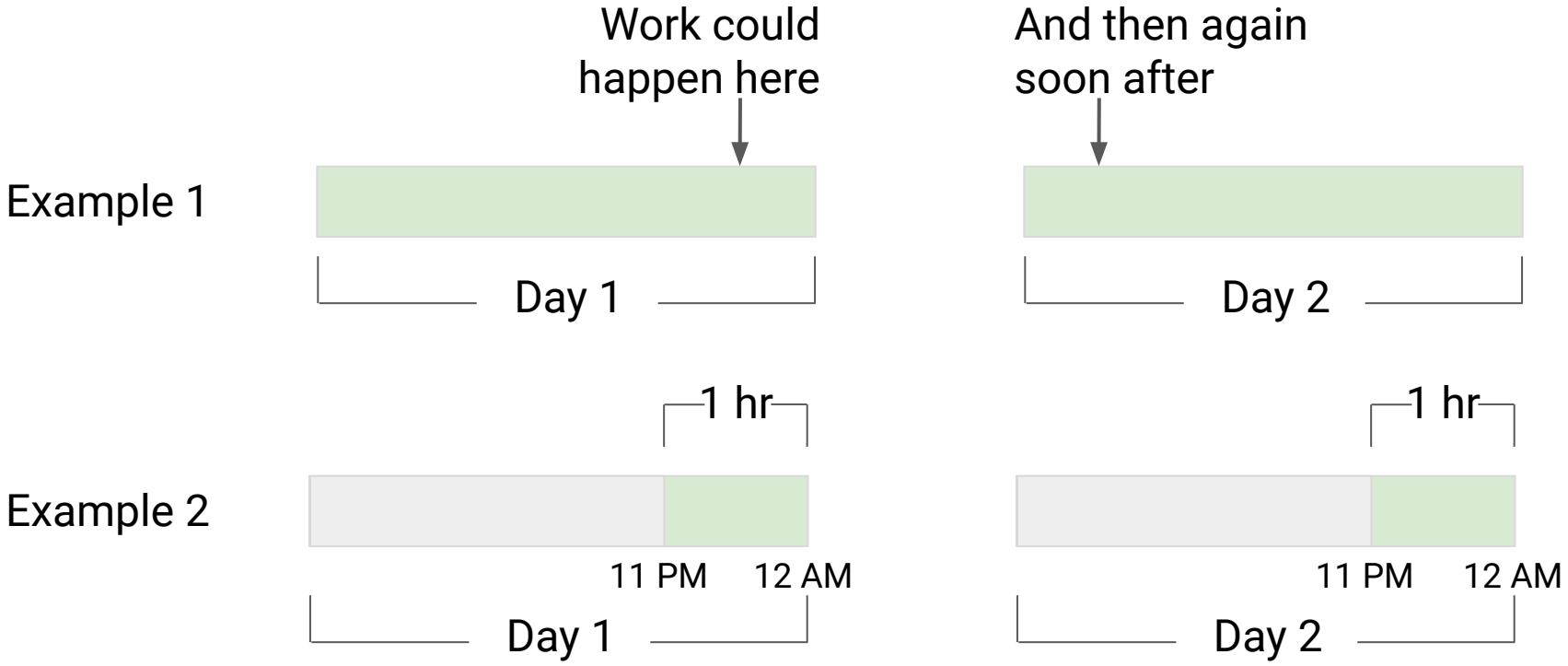
- Set a repeat interval
- Set a flex interval (optional)



Specify an interval using `TimeUnit` (e.g., `TimeUnit.HOURS`, `TimeUnit.DAYS`)

This work is licensed under the Apache 2 license.

Flex interval



This work is licensed under the Apache 2 license.

PeriodicWorkRequest example

```
val repeatingRequest = PeriodicWorkRequestBuilder<RefreshDataWorker>(
        1, TimeUnit.HOURS,     // repeatInterval
        15, TimeUnit.MINUTES   // flexInterval
    ).build()
```


This work is licensed under the Apache 2 license.

Enqueue periodic work

```
WorkManager.getInstance().enqueueUniquePeriodicWork(
    "Unique Name",
    ExistingPeriodicWorkPolicy.KEEP, // or REPLACE
    repeatingRequest
)
```


This work is licensed under the Apache 2 license.

Work input and output

This work is licensed under the Apache 2 license.

Define Worker with input and output

```
class MathWorker(context: Context, params: WorkerParameters):
      CoroutineWorker(context, params)  {

    override suspend fun doWork(): Result {
        val x = inputData.getInt(KEY_X_ARG, 0)
        val y = inputData.getInt(KEY_Y_ARG, 0)
        val result = computeMathFunction(x, y)
        val output: Data = workDataOf(KEY_RESULT to result)
        return Result.success(output)
    }
}
```


This work is licensed under the Apache 2 license.

Result output from doWork()

Result status	Result status with output
`Result.success()`	`Result.success(output)`
`Result.failure()`	`Result.failure(output)`
`Result.retry()`	

This work is licensed under the Apache 2 license.

Send input data to Worker

```
val complexMathWork = OneTimeWorkRequest<MathWorker>()
    .setInputData(
        workDataOf(
            "X_ARG" to 42,
            "Y_ARG" to 421,
        )
    ).build()

WorkManager.getInstance(myContext).enqueue(complexMathWork)
```


This work is licensed under the Apache 2 license.

WorkRequest constraints

This work is licensed under the Apache 2 license.

Constraints

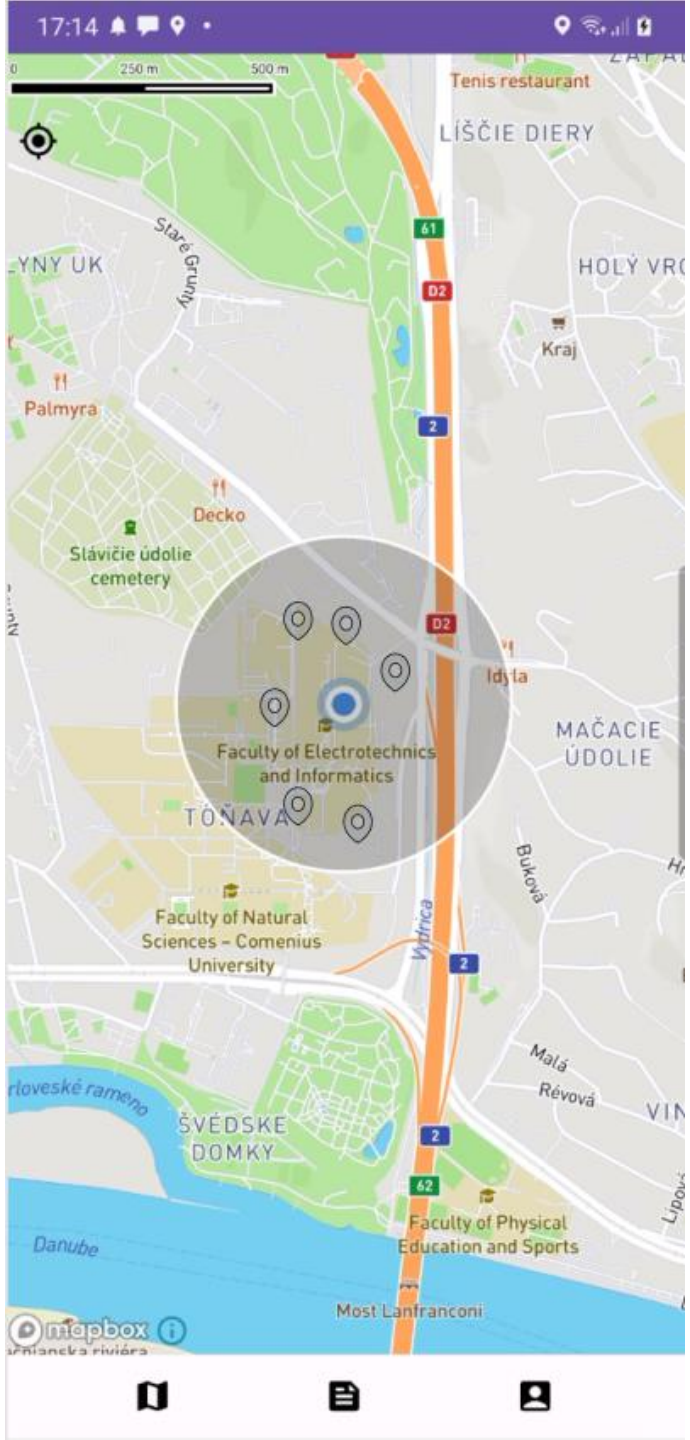
- `setRequiredNetworkType`
- `setRequiresBatteryNotLow`
- `setRequiresCharging`
- `setTriggerContentMaxDelay`
- `requiresDeviceIdle`

This work is licensed under the Apache 2 license.

Constraints example

```
val constraints = Constraints.Builder()
    .setRequiredNetworkType(NetworkType.UNMETERED)
    .setRequiresCharging(true)
    .setRequiresBatteryNotLow(true)
    .setRequiresDeviceIdle(true)
    .build()

val myWorkRequest: WorkRequest = OneTimeWorkRequestBuilder<MyWork>()
    .setConstraints(constraints)
    .build()
```


This work is licensed under the Apache 2 license.


17:14
250 m
500 m
Tenis restaurant
LÍŠČIE DIERY
Staré Grunty
HOLÝ VRC
Kraj
Palmyra
Decko
Slávičie údolie cemetery
Idyla
MAČACIE ÚDOLIE
Faculty of Electrotechnics and Informatics
TOŇAVA
Faculty of Natural Sciences – Comenius University
Vydrica
Bukova
Malá Révová
ŠVÉDSKE DOMKY
Faculty of Physical Education and Sports
Danube
Most Lanfranconi
mapbox

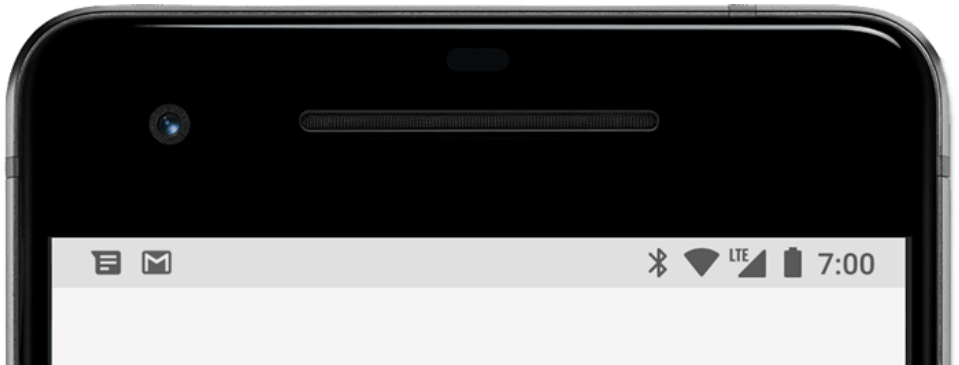
17:14
V okolí
Fero Mrkvicka
okruh 600 metrov
pred 3 hodinami
Joozko Mrkvicka
okruh 300 metrov
pred 2 hodinami
John Doe
okruh 900 metrov
pred 4 hodinami

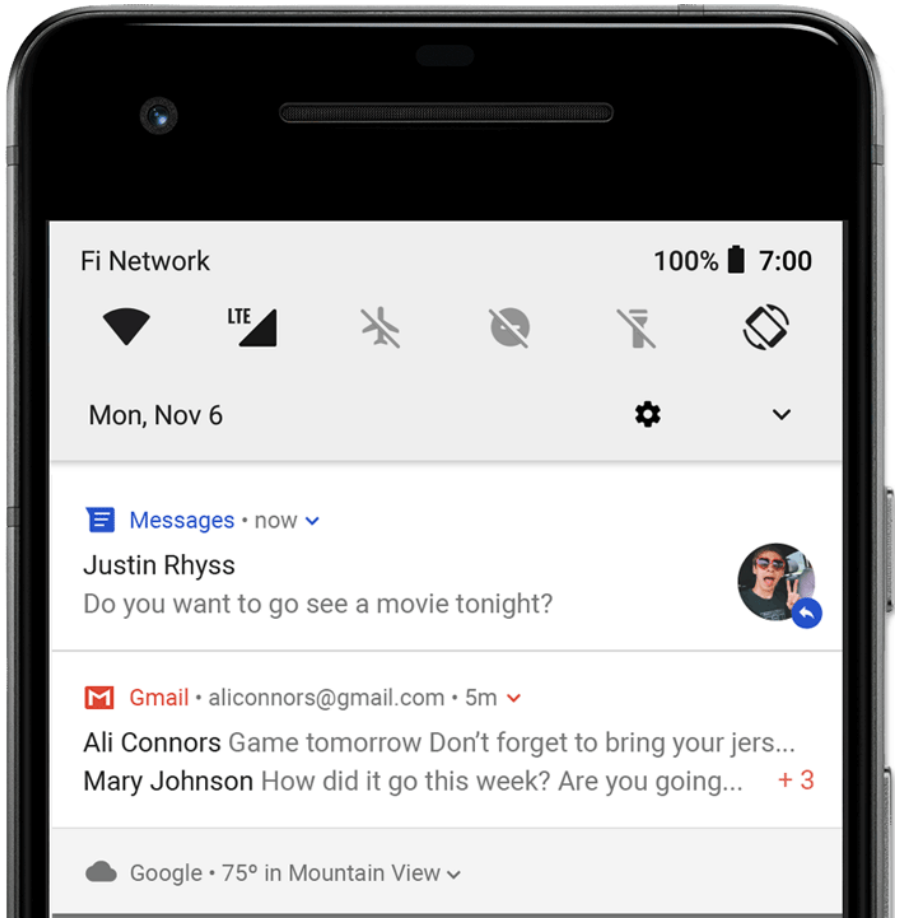
STU FEI

Notifikacie

Spolupráca s WorkManager a Repository na pozadí

Zdroj: https://developer.android.com/develop/ui/views/notifications





Notifikácie

- Pre Android 13 a vyššie treba povolenie od používateľa

```
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
```

Notifikácie

- Vytvorenie kanálu pre notifikácie

```
// Create the NotificationChannel, but only on API 26+ because
// the NotificationChannel class is new and not in the support library
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    val name = "MOBV Zadanie"
    val descriptionText = "Popis notifikacie"
    val importance = NotificationManager.IMPORTANCE_DEFAULT
        val channel_id = "kanal-1"
    val channel =
        NotificationChannel(channel_id, name, importance).apply {
            description = descriptionText
        }
    // Register the channel with the system
    val notificationManager: NotificationManager =
        appContext.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
    notificationManager.createNotificationChannel(channel)
}
```

Notifikacie

Zdroj: https://developer.android.com/develop/ui/views/notifications

Notification anatomy

The design of a notification is determined by system templates, and your app defines the contents for each portion of the template. Some details of the notification appear only in the expanded view.

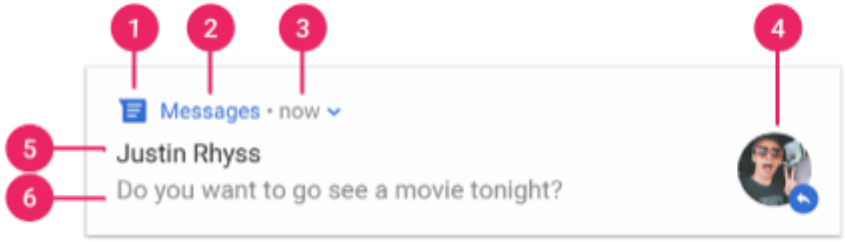


Figure 7. A notification with basic details.

The most common parts of a notification are indicated in figure 7, as follows:

1. Small icon: required; set using `setSmallIcon()`.
2. App name: provided by the system.
3. Time stamp: provided by the system, but you can override it using `setWhen()` or hide it using `setShowWhen(false)`.
4. Large icon: optional; usually used only for contact photos. Don't use it for your app icon. Set using `setLargeIcon()`.
5. Title: optional; set using `setContentTitle()`.
6. Text: optional; set using `setContentText()`.

Notifikácie

- Vytvorenie

```
val name = "MOBV Zadanie"
val descriptionText = "Popis notifikacie"
val importance = NotificationManager.IMPORTANCE_DEFAULT
val channel_id = "kanal-1"
val builder =
    NotificationCompat.Builder(appContext, channel_id).apply {
        setContentTitle("Titulok")
        setContentText("Text notifikacie")
        setSmallIcon(R.drawable.ikona_notifikacie)
        priority = NotificationCompat.PRIORITY_DEFAULT
    }

if (ActivityCompat.checkSelfPermission(
        appContext,
        Manifest.permission.POST_NOTIFICATIONS
    ) != PackageManager.PERMISSION_GRANTED
) {
    Log.d("Notifikacia","Chyba povolenie na notifikaciu");
    return
}

NotificationManagerCompat.from(context).notify(1, builder.build())
```