Intermediate Android
Geo Location, Advanced Testing, & Advanced Database

bangkit

Ground Rules

Observe the following rules to ensure a supportive, inclusive, and engaging classes

Give full attention in class

Mute your microphone when you're not talking

Keep your camera on

Turn on the CC Feature on Meet

Use raise hand or chat to ask questions

Make this room a safe place to learn and share

Material/Review

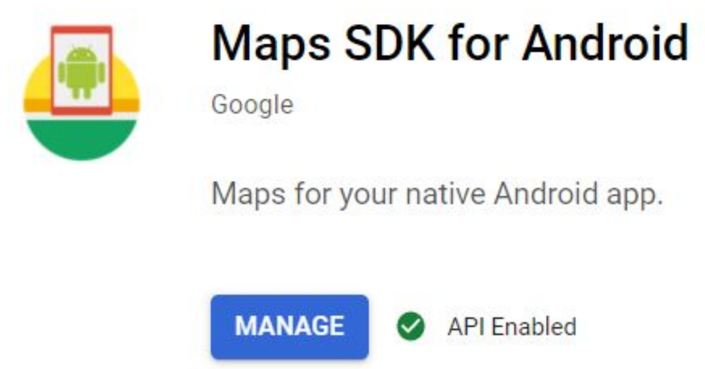
Material that has been Studied

- Learn how to integrate **Google Maps** in Android.
- Learn how to enable **Location Update**.
- Learn how to **add Geofencing** and handle **transitions**.
- Learn how to **plan a testing strategy.**
- Learn how to **create and use test doubles**.
- Learn how to **test repository, ViewModel,** and **Room**.
- Learn how to **integration test** in **Fragment**.
- Learn how to **end-to-end test** using **IdlingResource**.
- Learn how to **make relations in Room, add Pre-populated data,** and use **RawQuery.**
- Learn how to **use Paging** to your list app.

Geo Location

Adding a map to your app

- Obtain API keys to use Google Map from Google Cloud Console
- Include a Google Map in your app
  - Using SupportMapFragment
  - Using MapView
- Change the look and feel of the map
  - Map type (Normal, Satellite, Terrain, Hybrid)
  - Zoom level (1-20)
  - UI Control (Zoom control, compass, map toolbar)
  - Add marker
- Change map behavior
  - setOnMapClickListener
  - setOnPoiClickListener



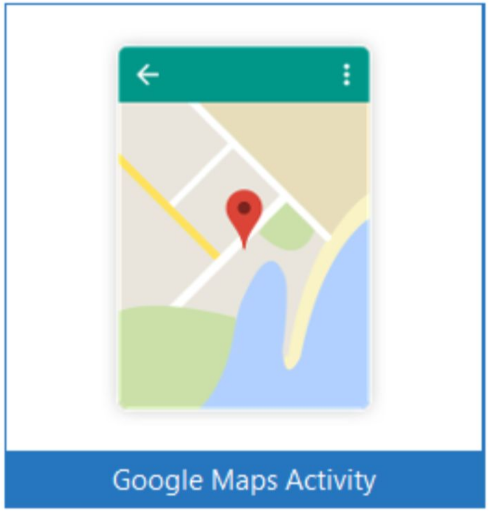
Maps SDK for Android

Google

Maps for your native Android app.

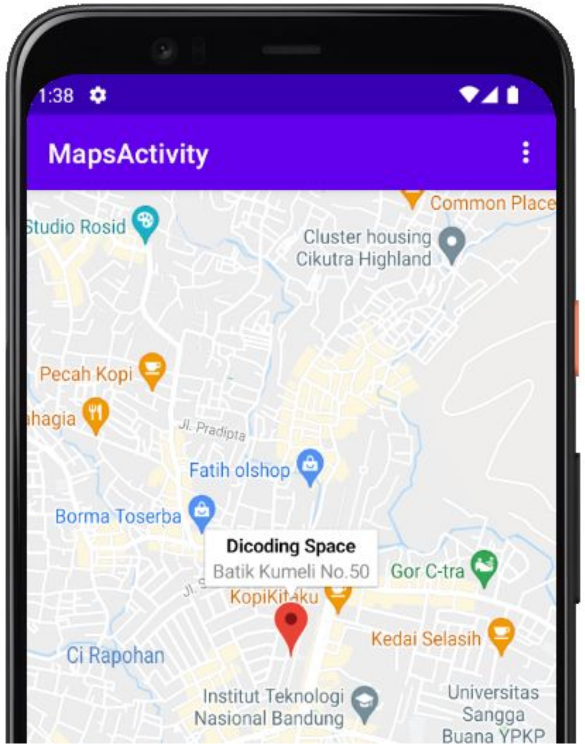
Create map example

```kotlin
class MapsActivity : AppCompatActivity(), OnMapReadyCallback {

   private lateinit var map: GoogleMap

   override fun onCreate(savedInstanceState: Bundle?) {
       ...
       val mapFragment = supportFragmentManager
           .findFragmentById(R.id.map) as SupportMapFragment
       mapFragment.getMapAsync(this)
   }
   override fun onMapReady(googleMap: GoogleMap) {
       map = googleMap
       ...
   }
}
```

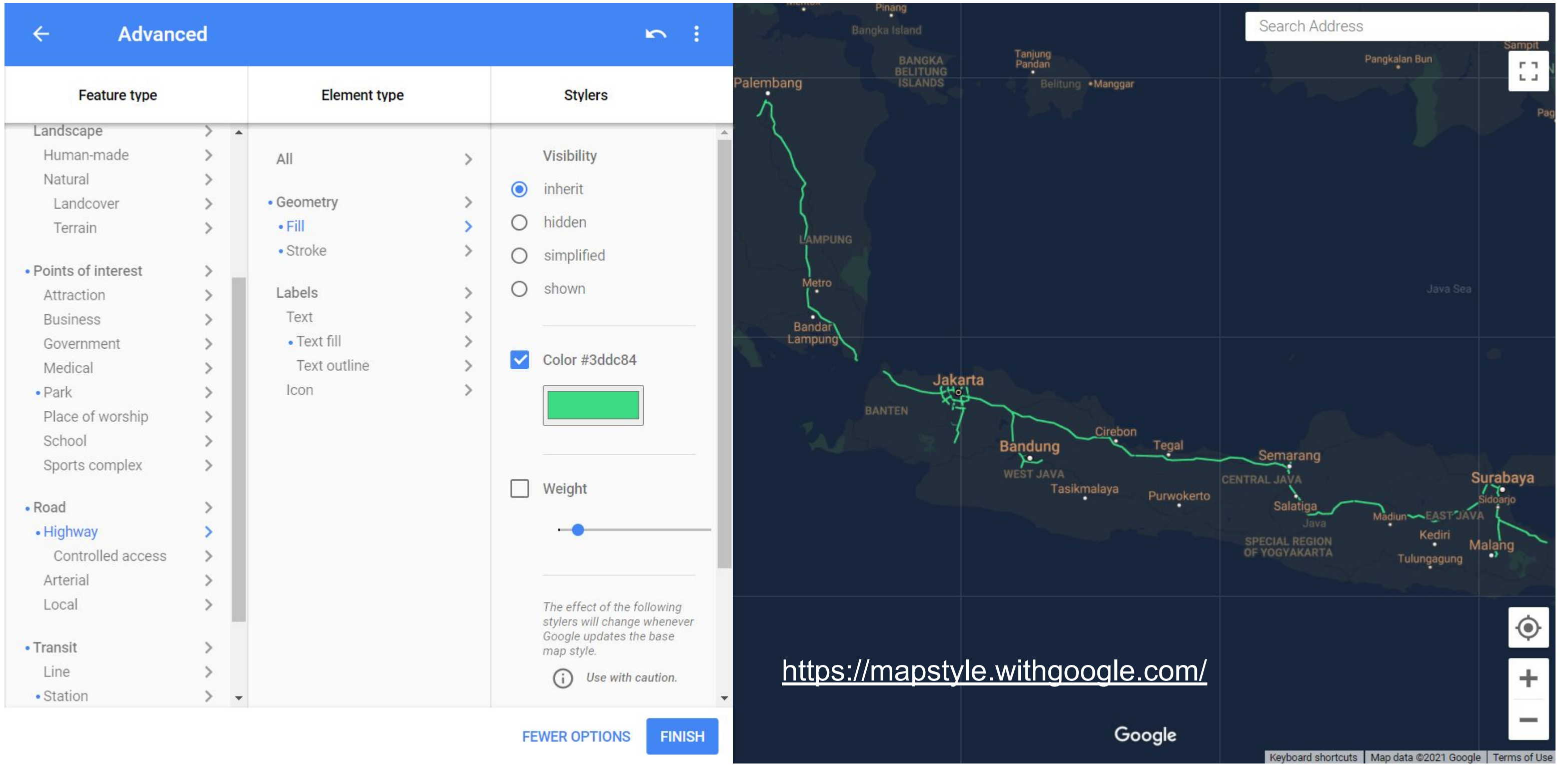


Configure UI Setting and add marker

```kotlin
override fun onMapReady(googleMap: GoogleMap) {
   map = googleMap
   map.uiSettings.isZoomControlsEnabled = true
   map.uiSettings.isIndoorLevelPickerEnabled = true
   map.uiSettings.isCompassEnabled = true
   map.uiSettings.isMapToolbarEnabled = true
   map.mapType = GoogleMap.MAP_TYPE_NORMAL

   val dicodingSpace = LatLng(-6.8957643, 107.6338462)
   val options = MarkerOptions().position(dicodingSpace)
.title("Dicoding Space").snippet("Batik Kumeli No.50")
   map.addMarker(options)

   map.animateCamera(CameraUpdateFactory.newLatLngZoom(
dicodingSpace, 15f))
```



Use style wizard to get styles



Visibility
[✓ inherit] hidden simplified shown

[✓] Color #3ddc84 🟩

[ ] Weight

The effect of the following stylers will change whenever Google updates the base map style.

Use with caution.

https://mapstyle.withgoogle.com/

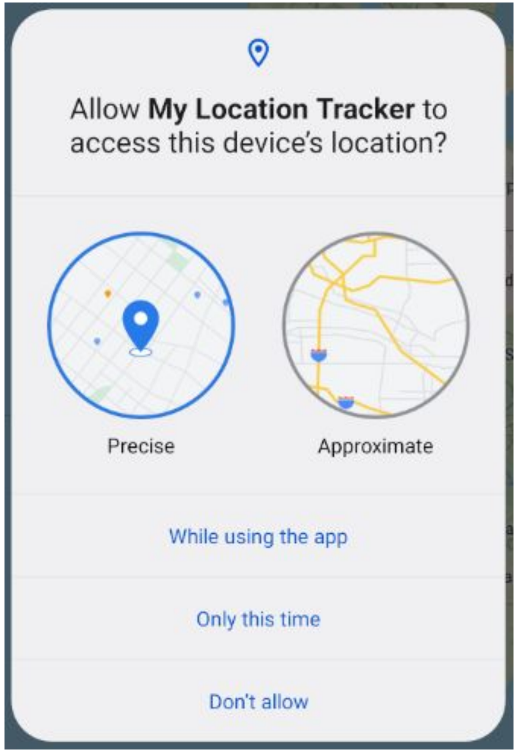
bangkit

Use style wizard to get styles

```
try {
    val success =
        map.setMapStyle(
          MapStyleOptions.loadRawResourceStyle(
             this,R.raw.map_style))
    if (!success) {
        Log.e(TAG, "Style parsing failed.")
    }
} catch (exception: Resources.NotFoundException) {
    Log.e(TAG, "Can't find style. Error: ", exception)
}
```

Location Permissions

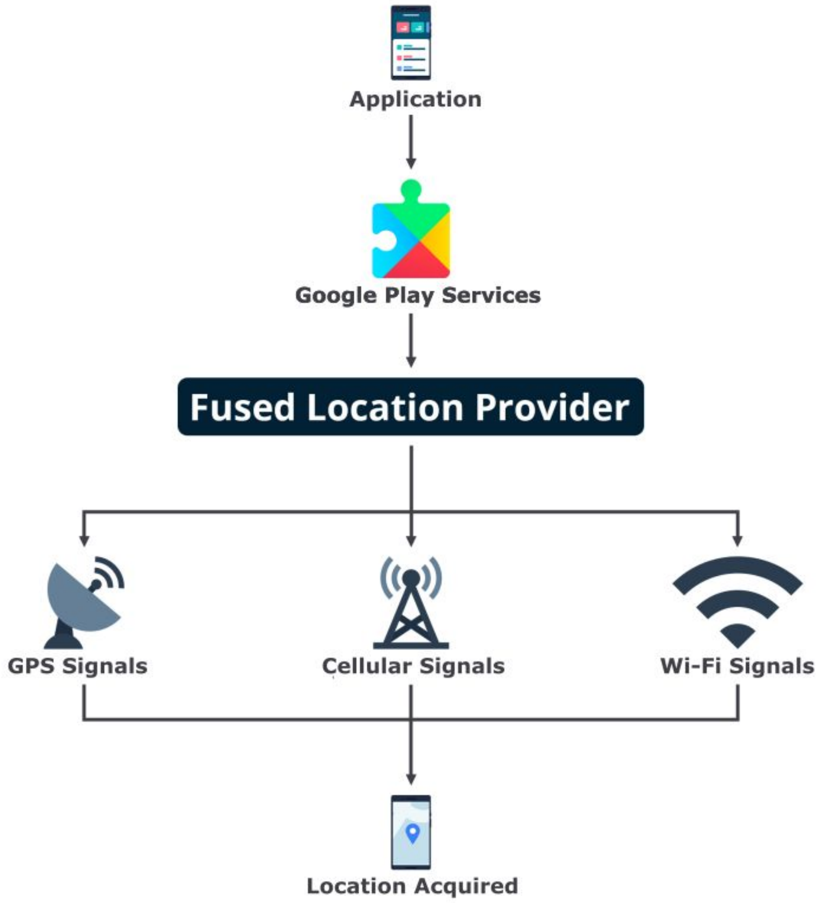
```
private val requestPermissionLauncher =
   registerForActivityResult(
       ActivityResultContracts.RequestMultiplePermissions()
   ) { permissions ->
       ...
   }
if (ContextCompat.checkSelfPermission(this, permissions) ==
PackageManager.PERMISSION_GRANTED) {
   // get location
} else {
   requestPermissionLauncher.launch(
       arrayOf(
           Manifest.permission.ACCESS_FINE_LOCATION,
           Manifest.permission.ACCESS_COARSE_LOCATION
       )
   )
}
```


Allow **My Location Tracker** to access this device's location?

Fused Location Provider

- Makes location requests combining GPS, Wi-Fi, and cell network
- Balances fast, accurate results with minimal battery drain
- Returns Location object with latitude and longitude

```
val fusedLocationClient =
LocationServices.getFusedLocation
ProviderClient(this)
```



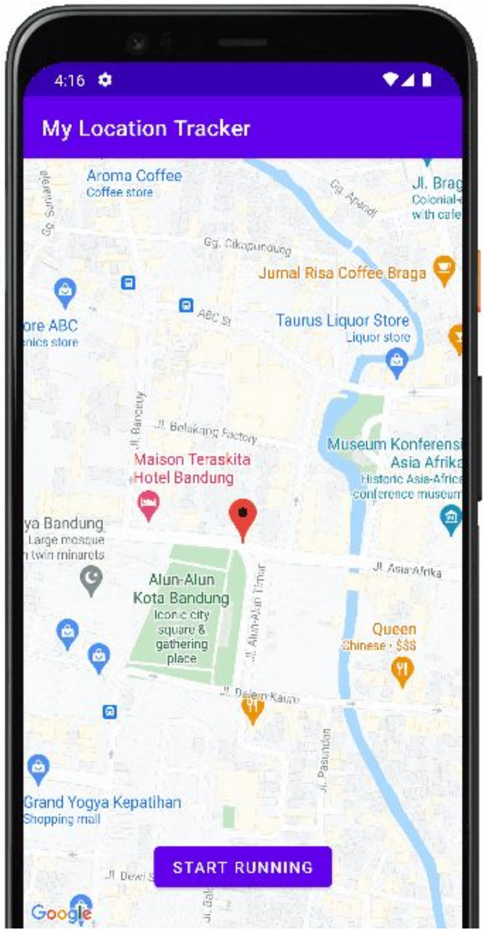
bangkit

Get Last Known Location

```
fusedLocationClient.lastLocation.addOnSuccessListener { location:
Location? ->
    if (location != null) {
        Log.d(TAG, location.latitude + ", " + location.longitude)
    } else {
        Toast.makeText(
            this@MapsActivity,
            "Location is not found. Try Again",
            Toast.LENGTH_SHORT
        ).show()
    }
}
```

Get Location Updates

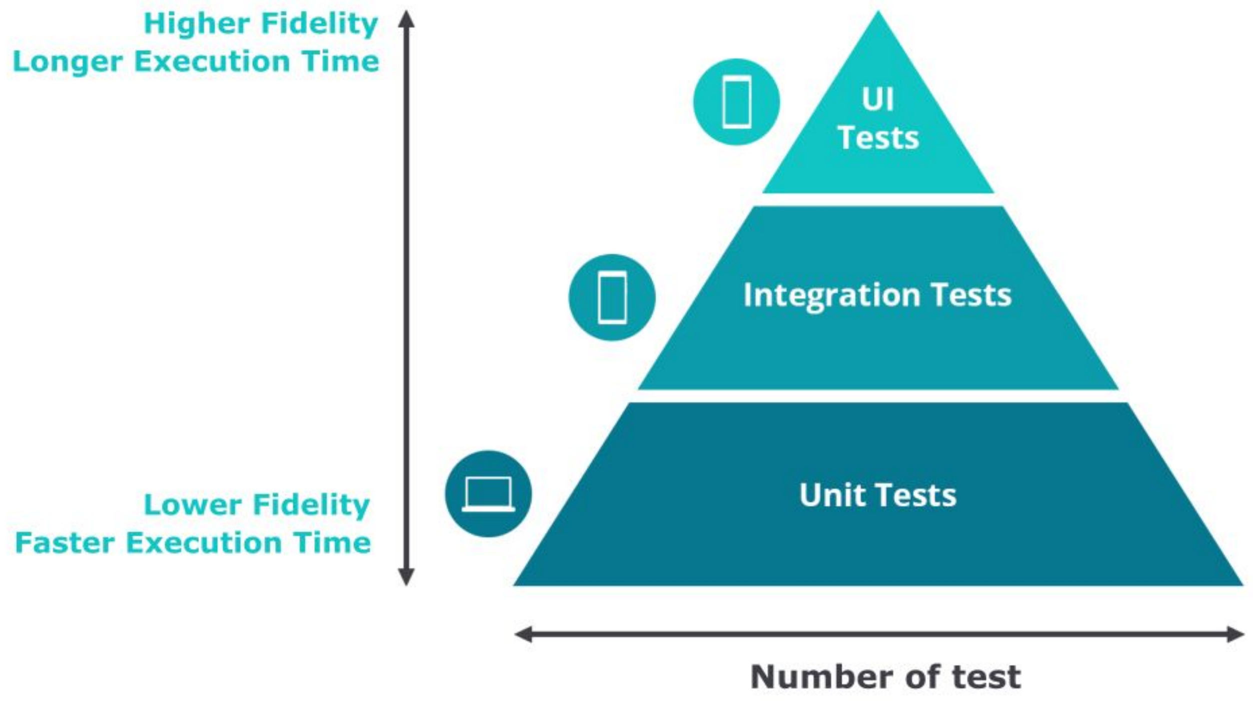
```
val locationRequest = LocationRequest.create().apply {
   interval = TimeUnit.SECONDS.toMillis(1)
   maxWaitTime = TimeUnit.SECONDS.toMillis(1)
   priority = LocationRequest.PRIORITY_HIGH_ACCURACY
}
val locationCallback = object : LocationCallback() {
   override fun onLocationResult(locationResult: LocationResult) {
       locationResult.lastLocation
       for (location in locationResult.locations) {
           // update UI such as draw polyline
       }
   }
}
fusedLocationClient.requestLocationUpdates(
   locationRequest,
   locationCallback,
   Looper.getMainLooper()
)
```



bangkit

Advanced Testing

Testing Pyramid



What to Test in Android

Unit Test

- Unit tests for **ViewModels**.
- Unit tests for the data layer, especially **repositories**. Most of the data layer should be platform-independent. Doing so enables test doubles to replace database modules and remote data sources in tests.
- Unit tests for **utility** classes such as string manipulation and math.

UI Test

- **Screen UI tests** check critical user interactions in a single screen.
- **User flow tests** or **Navigation tests**, covering most common paths. These tests simulate a user moving through a navigation flow.

Test Doubles

- **Fake** : A test double that has a "working" implementation of the class, but it's implemented in a way that makes it good for tests but unsuitable for production.
- **Mock** : A test double that behaves how you program it to behave and that has expectations about its interactions.
- **Stub** : A test double that behaves how you program it to behave but doesn't have expectations about its interactions.
- **Dummy** : A test double that is passed around but not used, such as if you just need to provide it as a parameter.
- **Spy** : A wrapper over a real object which also keeps track of some additional information.

Testing LiveData using Mockito & InstantTaskExecutorRule

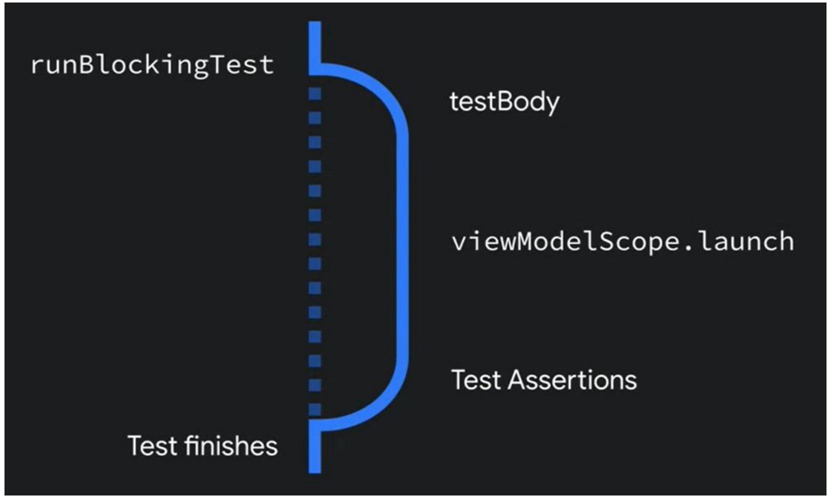
```
@get:Rule
var instantExecutorRule = InstantTaskExecutorRule()

@Mock
private lateinit var newsRepository: NewsRepository

@Test
fun `when Get HeadlineNews Should Not Null and Return Success`() {
   val expectedNews = MutableLiveData<Result<List<NewsEntity>>>()
   expectedNews.value = Result.Success(dummyNews)
   `when`(newsViewModel.getHeadlineNews()).thenReturn(expectedNews)

   val actualNews = newsViewModel.getHeadlineNews().getOrAwaitValue()

   Mockito.verify(newsRepository).getHeadlineNews()
   Assert.assertNotNull(actualNews)
   Assert.assertTrue(actualNews is Result.Success)
   Assert.assertEquals(dummyNews.size, (actualNews as Result.Success).data.size)
}
```

Testing LiveData using Mockito & InstantTaskExecutorRule

```
@get:Rule
var instantExecutorRule = InstantTaskExecutorRule()

@Mock
private lateinit var newsRepository: NewsRepository

@Test
fun `when Get HeadlineNews Should Not Null and Return Success`() {
    val observer = Observer<Result<List<NewsEntity>>> {}
    try {
        val expectedNews = MutableLiveData<Result<List<NewsEntity>>>()
        expectedNews.value = Result.Success(dummyNews)
        `when`(newsViewModel.getHeadlineNews()).thenReturn(expectedNews)

        val actualNews = newsViewModel.getHeadlineNews().observeForever(observer)

        Mockito.verify(newsRepository).getHeadlineNews()
        Assert.assertNotNull(actualNews)
    } finally {
        newsViewModel.getHeadlineNews().removeObserver(observer)
    }
}
```

Testing Coroutines using TestCoroutineDispatcher

```kotlin
@ExperimentalCoroutinesApi
@RunWith(MockitoJUnitRunner::class)
class NewsDetailViewModelTest{

   @Before
   fun setupDispatcher() {
       Dispatchers.setMain(testDispatcher)
   }

   @After
   fun tearDownDispatcher() {
       Dispatchers.resetMain()
       testDispatcher.cleanupTestCoroutines()
   }

   @Test
   fun `when bookmarkStatus false Should call
saveNews`() = runBlockingTest {
       // coroutines code
   }
}
```



Testing Coroutines using TestCoroutineDispatcher

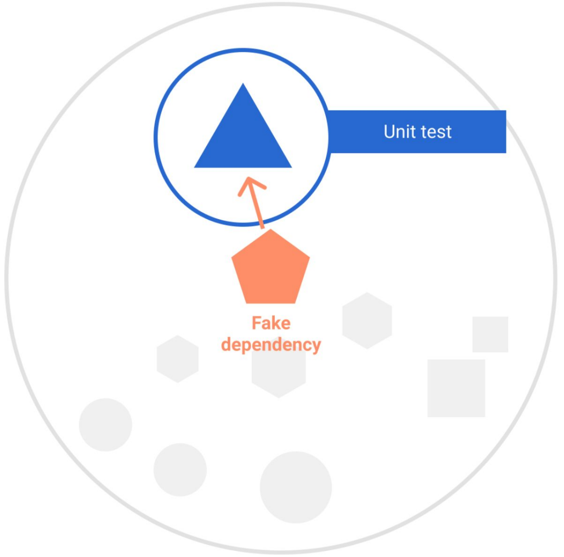
```kotlin
@ExperimentalCoroutinesApi
@RunWith(MockitoJUnitRunner::class)
class NewsDetailViewModelTest{

   @get:Rule
   var mainCoroutineRule = MainCoroutineRule()

   @Test
   fun `when bookmarkStatus false Should call
saveNews`() = mainCoroutineRule.runBlockingTest
{
          ...
   }
}
```

```kotlin
@ExperimentalCoroutinesApi
class MainCoroutineRule(val dispatcher:
TestCoroutineDispatcher =
TestCoroutineDispatcher()):
  TestWatcher(),
  TestCoroutineScope by
TestCoroutineScope(dispatcher) {
  override fun starting(description:
Description?) {
      super.starting(description)
      Dispatchers.setMain(dispatcher)
  }

  override fun finished(description:
Description?) {
      super.finished(description)
      cleanupTestCoroutines()
      Dispatchers.resetMain()
  }
}
```

Testing using a fake

```kotlin
class FakeNewsDao : NewsDao {
   private var newsData = mutableListOf<NewsEntity>()

   override fun getBookmarkedNews(): LiveData<List<NewsEntity>> {
       val observableNews = MutableLiveData<List<NewsEntity>>()
       observableNews.value = newsData
       return observableNews
   }
   override suspend fun saveNews(news: NewsEntity) {
       newsData.add(news)
   }
   override suspend fun deleteNews(newsTitle: String) {
       newsData.removeIf { it.title == newsTitle }
   }
}
```

```kotlin
val newsDao = FakeNewsDao()
val newsRepository = NewsRepository(newsDao)
```



Testing Room using a In Memory Database

```
@RunWith(AndroidJUnit4::class)
class NewsDaoTest{
   ...
   private lateinit var database: NewsDatabase
   private lateinit var dao: NewsDao

   @Before
   fun initDb() {
       database = Room.inMemoryDatabaseBuilder(
   ApplicationProvider.getApplicationContext(),
   NewsDatabase::class.java
       ).build()
       dao = database.newsDao()
   }

   @After
   fun closeDb() = database.close()
}
```

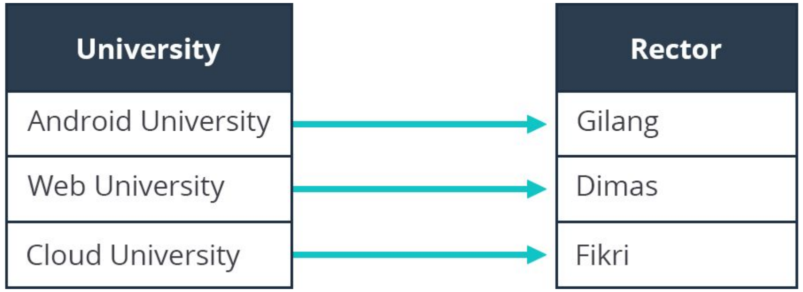
```
@Test
fun deleteNews() = runBlockingTest {
    dao.saveNews(sampleNews)
    dao.deleteNews(sampleNews.title)
    val actualNews =
    dao.getBookmarkedNews().getOrAwaitValue()

    Assert.assertTrue(actualNews.isEmpty())

    Assert.assertFalse(dao.isNewsBookmarked(sa
    mpleNews.title).getOrAwaitValue())
}
```

Advanced Database

bangkit

Database Relationship

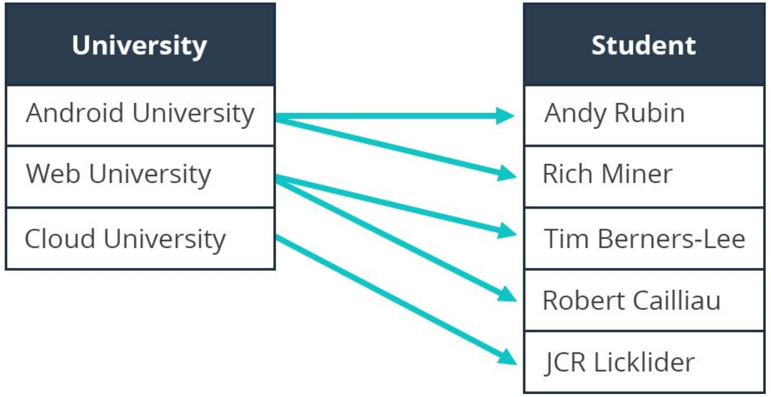
- One-to-one
- One-to-many
- Many-to-many



```
data class RectorAndUniversity(
    @Embedded
    val rector: Rector,
    @Relation(
        //column in Rector class
        parentColumn = "univId",
        //column in University class
        entityColumn = "universityId"
    )
    val university: University? = null
)
```

Database Relationship

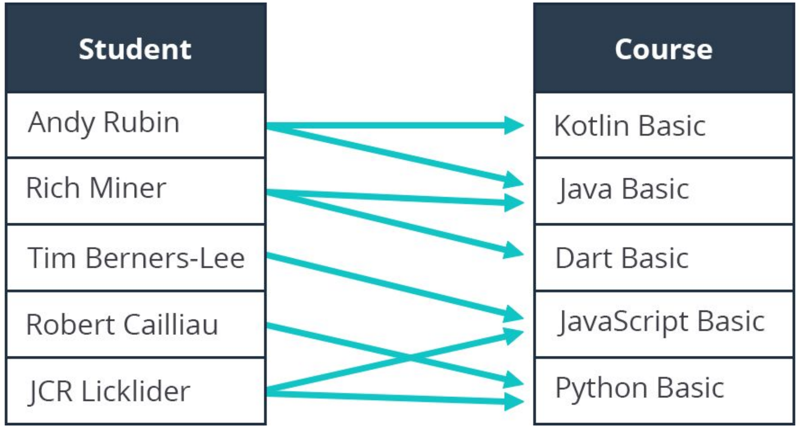
- One-to-one
- One-to-many
- Many-to-many



```kotlin
data class UniversityAndStudent(
   @Embedded
   val university: University,
   @Relation(
       //column in University class
       parentColumn = "universityId",
       //column in Student class
       entityColumn = "univId"
   )
   val student: List<Student>
)
```

Database Relationship

- One-to-one
- One-to-many
- Many-to-many



```kotlin
@Entity(primaryKeys = ["sId", "cId"])
data class CourseStudentCrossRef(
   val sId: Int,
   @ColumnInfo(index = true)
   val cId: Int,
)
data class StudentWithCourse(
   @Embedded
   val studentAndUniv: StudentAndUniversity,
   @Relation(
       parentColumn = "studentId",
       entity = Course::class,
       entityColumn = "courseId",
       associateBy = Junction(
           value = CourseStudentCrossRef::class,
           parentColumn = "sId",
           entityColumn = "cId"
       )
   )
   val course: List<Course>
)
```

Pre-Populate Database Room

- From Asset

```
.createFromAsset("initial_expense.db")
```

- From File System

```
.createFromFile(new File("database/initial_expense.db"))
```

- Using AddCallback Method

```
.addCallback(object :Callback(){
    override fun onCreate(db: SupportSQLiteDatabase) {
        super.onCreate(db)
        //insert new data
    }
})
```

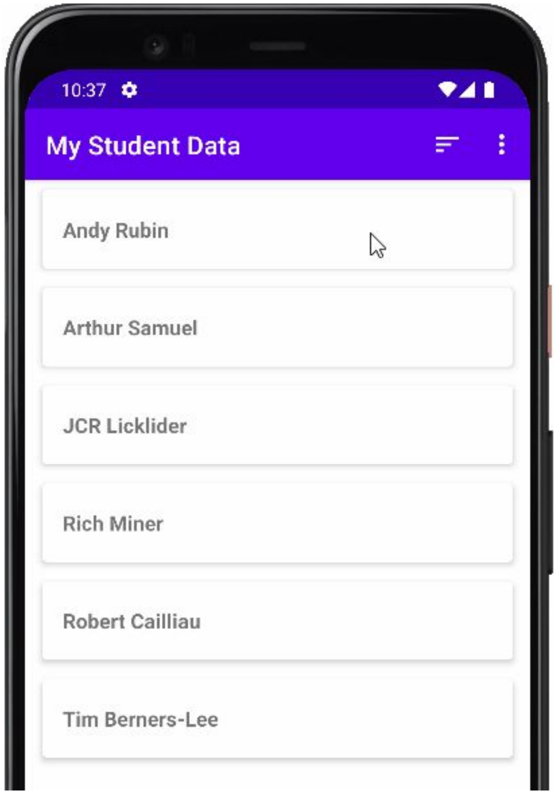
Migrating Room databases

```
@Database(
   entities = [Student::class],
   version = 2,
   autoMigrations = [
       AutoMigration(from = 1, to = 2, spec = StudentDatabase.MyAutoMigration::class),
   ],
   exportSchema = true
)

abstract class StudentDatabase : RoomDatabase() {

   @RenameColumn(tableName = "University", fromColumnName = "name", toColumnName =
"universityName")
   class MyAutoMigration : AutoMigrationSpec
   ...
}
```

RawQuery for Sorting List

```
@Dao
interface RawDao {
   @RawQuery(observedEntities = [Student::class])
   fun getStudent(query: SupportSQLiteQuery):
LiveData<List<Student>>
}
```

```
val query = StringBuilder().append("SELECT * FROM student ")
    when (sortType) {
        SortType.ASCENDING -> {
            simpleQuery.append("ORDER BY name ASC")
        }
        SortType.DESCENDING -> {
            simpleQuery.append("ORDER BY name DESC")
        }
              SortType.RANDOM -> {
           simpleQuery.append("ORDER BY RANDOM()")
              }
    }
val student = studentDao.getStudent(query)
```



10:37

My Student Data

Andy Rubin

Arthur Samuel

JCR Licklider

Rich Miner

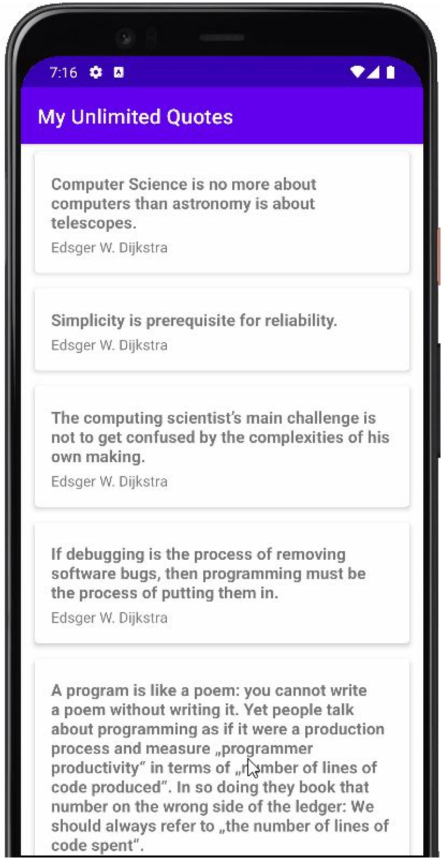
Robert Cailliau

Tim Berners-Lee

Paging 3

The Paging library helps you load and display pages of data from a larger dataset from local storage or over network.

- In-memory caching for your paged data.
- Built-in request deduplication, ensuring that your app uses network bandwidth and system resources efficiently.
- Configurable RecyclerView adapters which automatically request data as the user scrolls toward the end of the loaded data.
- First-class support for Kotlin coroutines and Flow, as well as LiveData and RxJava.
- Built-in support for error handling, including refresh and retry capabilities.


7:16

My Unlimited Quotes

Computer Science is no more about computers than astronomy is about telescopes.
Edsger W. Dijkstra

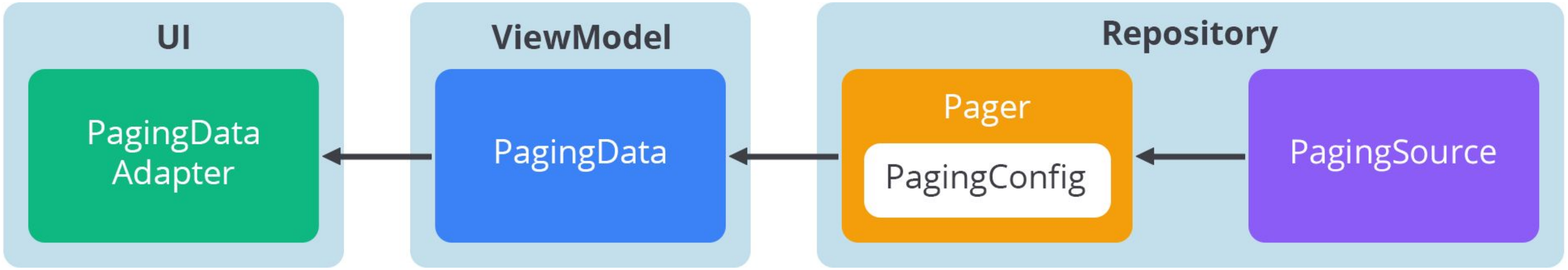
Simplicity is prerequisite for reliability.
Edsger W. Dijkstra

The computing scientist's main challenge is not to get confused by the complexities of his own making.
Edsger W. Dijkstra

If debugging is the process of removing software bugs, then programming must be the process of putting them in.
Edsger W. Dijkstra

A program is like a poem: you cannot write a poem without writing it. Yet people talk about programming as if it were a production process and measure „programmer productivity" in terms of „number of lines of code produced". In so doing they book that number on the wrong side of the ledger: We should always refer to „the number of lines of code spent".

Library Architecture



- **PagingSource**: Sets the next method of retrieving data from data sources, both from the internet and databases, and how to refresh the data.
- **Pager** : Converts PagingSource to PagingData. There are three types of output can be generated, such as Flow, LiveData, and Observable RxJava.
- **PagingConfig** : Set the configuration for data retrieval.
- **PagingData** : Wrapper used as a container to store data on each page.
- **PagingDataAdapter** : RecyclerView Adapter specifically for handling PagingData

Example Code

- PagingSource

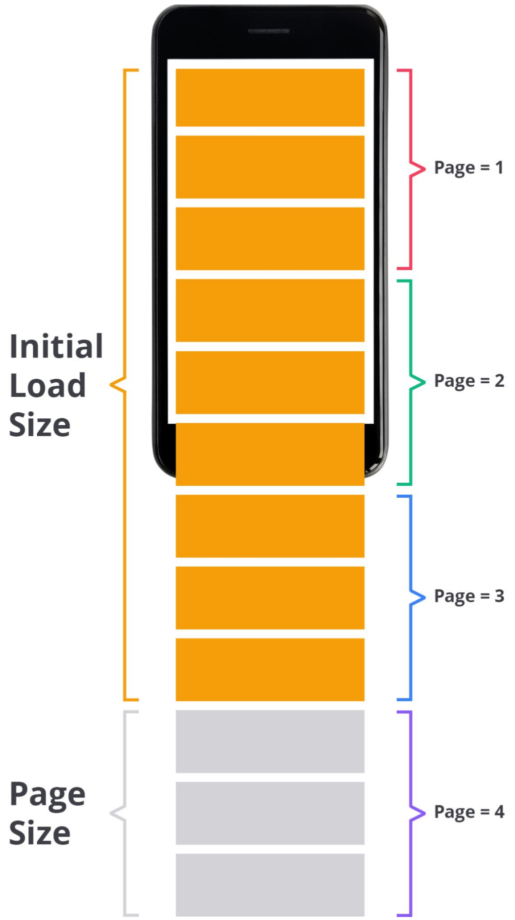
```
@Query("SELECT * FROM passenger")
fun getAllPassenger(): PagingSource<Int, Passenger>
```

- Pager

```
val data : LiveData<PagingData<DataItem>> = Pager(
   config = PagingConfig(
       initialLoadSize = 48
       pageSize = 12
   ),
   pagingSourceFactory = {
       passengerDao.getAllPassenger()
   }
).liveData
```

- PagingDataAdapter

```
class PassengerListAdapter : PagingDataAdapter<DataItem,
PassengerListAdapter.MyViewHolder>(DIFF_CALLBACK) {
```



Sharing

Demo Link

ILT 5

Quiz

Discussion

Thank You