TDD in Android

Example Application

In order to organize my daily activities I want to keep a task list

Empty Task List

- Given I have no tasks yet
- When I open task application
- Then I see Task List screen
- And I see an empty task list

Add Task Action

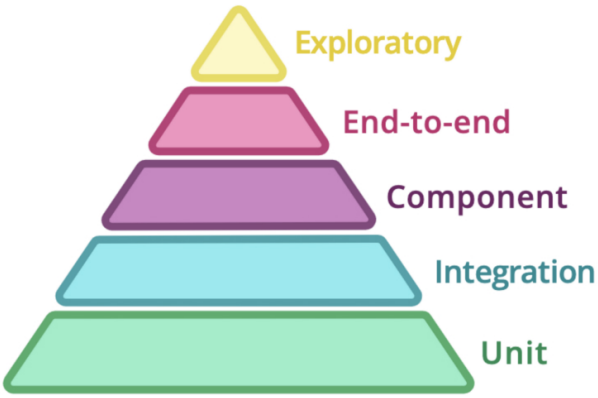
- Given I see the Task List screen
- When I click Add Task button
- Then I see Save Task screen

Save Task

- Given I see Save Task screen
- And I write call mum in the description
- When I click Save button
- Then I see Task List screen
- And I see call mum in the task list

Test Driven Development

These are the fundamentals of TDD. Although there are plenty of resources to learn the basics, there is still misunderstanding around how to properly implement an application using TDD. Let's consider the following test pyramid:



Top-Down vs. Bottom-Up

Let’s Code

Add Task Action and Empty Task List scenarios are much simpler. It is more natural to implement Empty Task List scenario first, since it is the starting state of the application and then Add Task Action scenario. Empty Task List is an edge case, it doesn't bring that much value to the user, therefore I will test-drive its implementation without the GUI. Also we should keep edge case scenarios in lower levels of the test pyramid, leaving the happy path scenarios in higher levels.

```
@Test
fun `Given I have no tasks yet
     When I open task application
     Then I see Task List screen And I see no tasks`() {

    val taskApplication = TaskApplication()

    taskApplication.open()

    taskApplication.withScreenCallback { screen ->
        assertEquals(emptyList<String>(), screen.tasks)
    }
}
```

Since this test keeps the UI out, some design decisions have to be made while writing it. A callback is registered in TaskApplication to receive TaskListScreen with the list of tasks.

```
class TaskApplication {
    private lateinit var screen: Screen

    fun open() {
        screen = TaskListScreen(emptyList())
    }

    fun withScreenCallback(callback: (TaskListScreen) -> Unit) {
        callback.invoke(screen)
    }
}

data class TaskListScreen(
    val tasks: List<String>
)
```

As we see the minimal possible changes were made to make the test pass. Let's tackle Add Task Action scenario next:

```
@Test
fun `Given I see Task List screen When I tap add task Then I see Save Task screen`() {

    val taskApplication = TaskApplication()
    taskApplication.open()

    taskApplication.addTask()

    taskApplication.withScreenCallback { screen ->
        assertEquals(true, screen is SaveTaskScreen)
    }
}
```

We had to add a Screen interface to create this test, in order to reuse withScreenCallback method for SaveTaskScreen.

```
class TaskApplication {

    private var screenCallback: ((Screen) -> Unit)? = null
    private lateinit var screen: Screen

    fun open() {
        screen = TaskListScreen(emptyList())
    }

    fun withScreenCallback(callback: (Screen) -> Unit) {
        this.screenCallback = callback
        this.screenCallback?.invoke(screen)
    }

    fun addTask() {
        screen = SaveTaskScreen()
        this.screenCallback?.invoke(screen)
    }
}

interface Screen

data class TaskListScreen(
    val tasks: List<String>
) : Screen

class SaveTaskScreen: Screen
```

Now we also can apply some refactors. The screenCallback variable can have a default no action value, then we can avoid the ? syntax and call the function directly. Also we can move Screen, TaskListScreen and SaveTaskScreen to their respective files.

```
class TaskApplication {

    private object NoActionScreenCallback : (Screen) -> Unit {
        override fun invoke(screen: Screen) {}
    }

    private var screenCallback : (Screen) -> Unit = NoActionScreenCallback
    private lateinit var screen: Screen

    fun open() {
        screen = TaskListScreen(emptyList())
    }

    fun withScreenCallback(callback: (Screen) -> Unit) {
        this.screenCallback = callback
        this.screenCallback(screen)
    }

    fun addTask() {
        screen = SaveTaskScreen()
        this.screenCallback(screen)
    }
}
```

Finally we can attack the Save Task scenario. Now we are going to add the UI to the test, making it more complete.

```
@RunWith(AndroidJUnit4::class)
class TaskManagerTest {

    @Test
    fun `Given I have no tasks
         And I see Save Task screen
         And I fill description
         When I tap Save button
         Then I see Task List screen with description`() {

        val scenario = ActivityScenario.launch(MainActivity::class.java)
        val addTask = withId(R.id.view_task_list_add_task_button)
        val description = withId(R.id.view_add_task_description_input_field)
        val saveTask = withId(R.id.view_add_task_save_task_button)

        onView(addTask).perform(click())
        onView(description).perform(replaceText("My description"))
        onView(saveTask).perform(click())

        onView(withText("My description")).check(matches(isDisplayed()))

        scenario.close()
    }
}
```

This test fails, because we don't have a MainActivity yet, and it is not declared in the AndroidManifest. Also the ids we are referencing don't exist. To make it pass some design decisions have to be made. Since there will two screens in the application we could use Fragments, Activities or Views to represent them. I will use Views and a single Activity, for personal preference.

```
class MainApplication : Application() {
    val taskApplication by lazy {
        TaskApplication()
    }

    override fun onCreate() {
        super.onCreate()
        taskApplication.open()
    }
}
```

```
class MainActivity : AppCompatActivity() {

    private val taskApplication by lazy {
        (application as MainApplication).taskApplication
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val frame = findViewById<FrameLayout>(R.id.activity_main_frame)

        taskApplication.withScreenCallback { screen ->
            frame.removeAllViews()
            when (screen) {
                is TaskListScreen -> {
                    val view = TaskListView(this)
                    view.application = taskApplication
                    frame.addView(view)
                    view.updateScreen(screen)
                }
                is SaveTaskScreen -> {
                    val view = SaveTaskView(this)
                    view.application = taskApplication
                    frame.addView(view)
                }
            }
        }
    }
}
```

```
class TaskListView : ConstraintLayout {

    lateinit var application: TaskApplication
    private lateinit var listView: RecyclerView

    constructor(context: Context) : super(context) {
        init()
    }

    constructor(context: Context, attrs: AttributeSet) : super(context, attrs) {
        init()
    }

    constructor(context: Context, attrs: AttributeSet?, defStyleAttr: Int) : super(context, attrs, defStyleAttr) {
        init()
    }

    fun updateScreen(screen: TaskListScreen) {
        (listView.adapter as TaskListAdapter).updateTasks(screen.tasks)
    }

    private fun init() {
        inflate(context, R.layout.view_task_list, this)
        findViewById<Button>(R.id.view_task_list_add_task_button).setOnClickListener {
            application.addTask()
        }
        listView = findViewById(R.id.view_task_list_view)
        listView.layoutManager = LinearLayoutManager(context);
        listView.adapter = TaskListAdapter(LayoutInflater.from(context))
    }
}
```

```
class SaveTaskView : ConstraintLayout {

    lateinit var application: TaskApplication
    lateinit var screen: SaveTaskScreen
    private lateinit var saveTaskButton: Button
    private lateinit var descriptionInputField: EditText

    constructor(context: Context) : super(context) {
        init()
    }

    constructor(context: Context, attrs: AttributeSet) : super(context, attrs) {
        init()
    }

    constructor(context: Context, attrs: AttributeSet?, defStyleAttr: Int) : super(context, attrs, defStyleAttr) {
        init()
    }

    private fun init() {
        inflate(context, R.layout.view_add_task, this)
        saveTaskButton = findViewById(R.id.view_add_task_save_task_button)
        descriptionInputField = findViewById(R.id.view_add_task_description_input_field)
        saveTaskButton.setOnClickListener {
            application.saveTask(descriptionInputField.text.toString())
        }
    }
}
```

```
class TaskApplication {

    private object NoActionScreenCallback : (Screen) -> Unit {
        override fun invoke(screen: Screen) {}
    }

    private var screenCallback : (Screen) -> Unit = NoActionScreenCallback
    private lateinit var screen: Screen
    private val tasks = mutableListOf<String>()

    fun open() {
        screen = TaskListScreen(tasks)
    }

    fun withScreenCallback(callback: (Screen) -> Unit) {
        this.screenCallback = callback
        this.screenCallback(screen)
    }

    fun addTask() {
        screen = SaveTaskScreen()
        this.screenCallback(screen)
    }

    fun saveTask(description: String) {
        tasks.add(description)
        screen = TaskListScreen(tasks)
        this.screenCallback(screen)
    }
}
```

```
class TaskApplication {

    private object NoActionScreenCallback : (Screen) -> Unit {
        override fun invoke(screen: Screen) {}
    }

    private var screenCallback : (Screen) -> Unit = NoActionScreenCallback
    private lateinit var screen: Screen
    private val tasks = mutableListOf<String>()

    fun open() {
        screen = TaskListScreen(tasks)
    }

    fun withScreenCallback(callback: (Screen) -> Unit) {
        this.screenCallback = callback
        this.screenCallback(screen)
    }

    fun addTask() {
        screen = SaveTaskScreen()
        this.screenCallback(screen)
    }

    fun saveTask(description: String) {
        tasks.add(description)
        screen = TaskListScreen(tasks)
        this.screenCallback(screen)
    }
}
```

Persist Saved Tasks

```
@Test
fun `Given I have a task
     And I close the application
     When I open task application
     Then I see Task List screen And I see the task`() {

    val taskApplication = TaskApplication()
    taskApplication.open()
    taskApplication.addTask()
    taskApplication.saveTask("Clean up my room")
    val newTaskApplication = TaskApplication()

    newTaskApplication.open()

    newTaskApplication.withScreenCallback { screen ->
        assertEquals(true, screen is TaskListScreen)
        assertEquals(listOf("Clean up my room"), (screen as TaskListScreen).tasks)
    }
}
```

```
class TaskApplication(private val context: Context) {
    object NoActionScreenCallback : (Screen) -> Unit {
        override fun invoke(screen: Screen) {}
    }

    private var screenCallback : (Screen) -> Unit = NoActionScreenCallback
    private lateinit var screen: Screen
    private val tasks = mutableListOf<String>()

    private val taskHandler by lazy {
        val thread = HandlerThread("task")
        thread.start()
        Handler(thread.looper)
    }

    private val mainHandler by lazy {
        Handler(Looper.getMainLooper())
    }

    private val database by lazy {
        SQLiteOpenHelperDatabase(
            context,
            "task_database",
            1,
            "CREATE TABLE IF NOT EXISTS tasks (description TEXT)")
    }

    fun open() {
        taskHandler.post {
            var cursor: Cursor? = null
            try {
                cursor = database
                    .readableDatabase
                    .rawQuery("SELECT * FROM tasks", null)
                while (cursor.moveToNext()) {
                    tasks.add(cursor.getString(cursor.getColumnIndex("description")))
                }
            } finally {
                cursor?.close()
            }

            updateScreen(TaskListScreen(tasks))
        }
    }

    fun saveTask(description: String) {
        taskHandler.post {
            database
                .writableDatabase
                .execSQL("INSERT INTO tasks (description) VALUES ('$description')")

            tasks.add(description)
            updateScreen(TaskListScreen(tasks))
        }
    }

    fun withScreenCallback(callback: (Screen) -> Unit) {
        taskHandler.post {
            this.screenCallback = callback
            updateScreen(screen)
        }
    }

    fun addTask() {
        taskHandler.post {
            updateScreen(SaveTaskScreen())
        }
    }

    private fun updateScreen(screen: Screen) {
        this.screen = screen
        mainHandler.post {
            screenCallback(this.screen)
        }
    }
}
```

```
class SQLiteOpenHelperDatabase(
  context: Context,
  name: String?,
  version: Int,
  private val createSQL: String
) : SQLiteOpenHelper(context, name, null, version) {

    override fun onCreate(database: SQLiteDatabase) {
        database.execSQL(createSQL)
    }

    override fun onUpgrade(database: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    }
}
```

```
class TaskRepository(context: Context) {

    private val tasks = mutableListOf<String>()

    private val database by lazy {
        SQLiteOpenHelperDatabase(
            context,
            "task_database",
            1,
            "CREATE TABLE IF NOT EXISTS tasks (description TEXT)"
        )
    }


    fun getTasks(): List<String> {
        if (tasks.isEmpty()) {
            var cursor: Cursor? = null
            try {
                cursor = database
                    .readableDatabase
                    .rawQuery("SELECT * FROM tasks", null)
                while (cursor.moveToNext()) {
                    tasks.add(
                        cursor
                            .getString(cursor.getColumnIndex("description"))
                    )
                }
            } finally {
                cursor?.close()
            }
        }
        return tasks
    }

    fun saveTask(description: String) {
        database
            .writableDatabase
            .execSQL("INSERT INTO tasks (description) VALUES ('$description')")
        tasks.add(description)
    }
}
```

```
interface Scheduler {
    fun schedule(action: () -> Unit)
}

class HandlerScheduler(private val name: String) : Scheduler {

    private val handler by lazy {
        val handlerThread = HandlerThread(name)
        handlerThread.start()
        Handler(handlerThread.looper)
    }

    override fun schedule(action: () -> Unit) {
        handler.post {
            action()
        }
    }
}

object MainHandlerScheduler: Scheduler {

    private val mainHandler by lazy {
        Handler(Looper.getMainLooper())
    }

    override fun schedule(action: () -> Unit) {
        mainHandler.post {
            action()
        }
    }
}
```

```
class TaskApplication(
    context: Context,
    private val repository: TaskRepository = TaskRepository(context),
    private val scheduler: Scheduler = HandlerScheduler("task"),
    private val mainScheduler: Scheduler = MainHandlerScheduler
) {

    private object NoActionScreenCallback : (Screen) -> Unit {
        override fun invoke(screen: Screen) {}
    }

    private var screenCallback : (Screen) -> Unit = NoActionScreenCallback
    private lateinit var screen: Screen

    fun open() {
        scheduler.schedule {
            updateScreen(TaskListScreen(repository.getTasks()))
        }
    }

    fun withScreenCallback(callback: (Screen) -> Unit) {
        scheduler.schedule {
            this.screenCallback = callback
            updateScreen(screen)
        }
    }

    fun addTask() {
        scheduler.schedule {
            updateScreen(SaveTaskScreen())
        }
    }

    fun saveTask(description: String) {
        scheduler.schedule {
            repository.saveTask(description)
            updateScreen(TaskListScreen(repository.getTasks()))
        }
    }

    private fun updateScreen(screen: Screen) {
        this.screen = screen
        mainScheduler.schedule {
            screenCallback(this.screen)
        }
    }
}
```