
Gabriel Weis | **BINOSYS** GmbH

RxJava

Just another golden grail for Android?

Why

*“A good design is easier to change
than a bad design”*

Dave Thomas

Why Architecture

- Understandability
- Testability
- Maintainability
- Reliability

⇒ Stability in the long term

Why not

Architecture is not free

- More source code
- Less performance
- Needs reviews
- Needs experienced developers

What makes it difficult on Android to have a good architecture?

Android Flaws

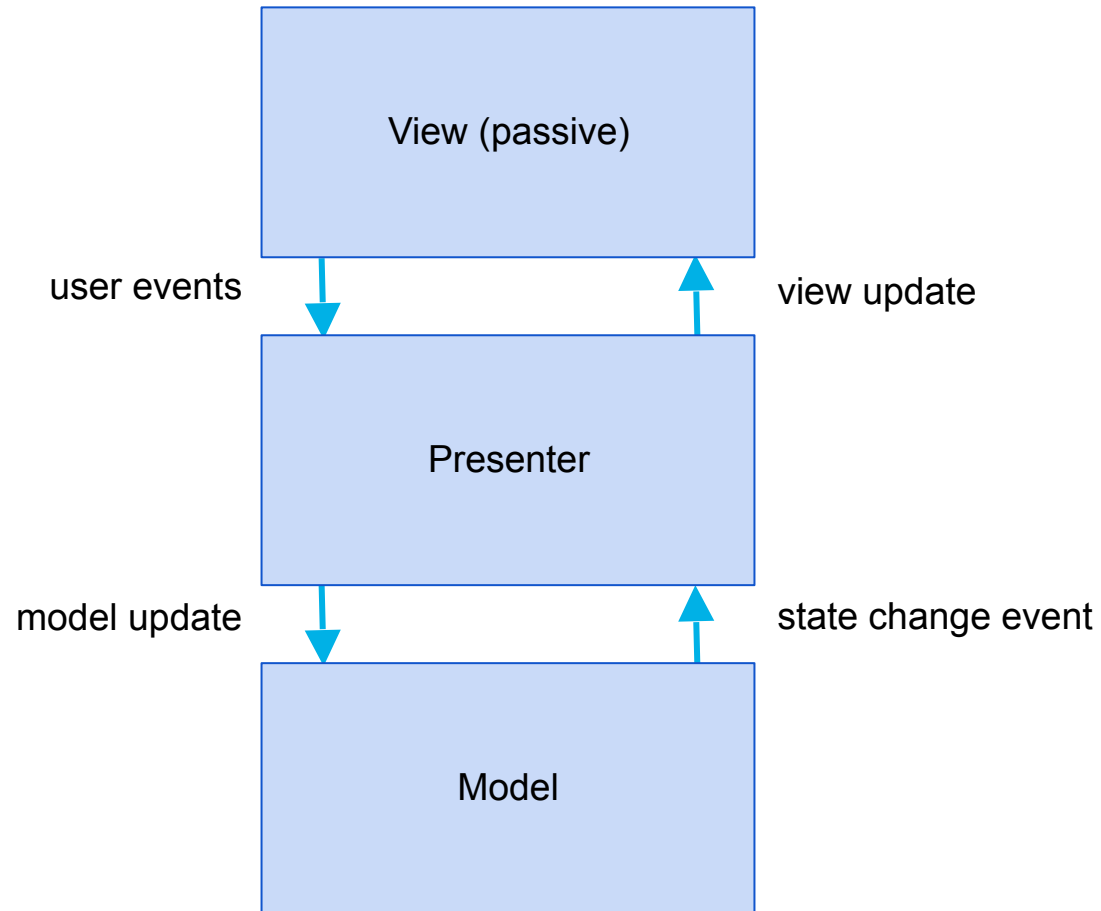
- Android violates massively SOLID
 - Single responsibility: can you say “*Context*”
 - Open/closed: we are final
 - Liskov substitution: puh, had luck
 - Interface segregation: the empty listener
 - Dependency inversion: interface what?

Android Flaws

- Lifecycle
- No build-in ModelView-Whatever
- Asynchronous execution
- No guidelines and degree of freedom

KISS[®]

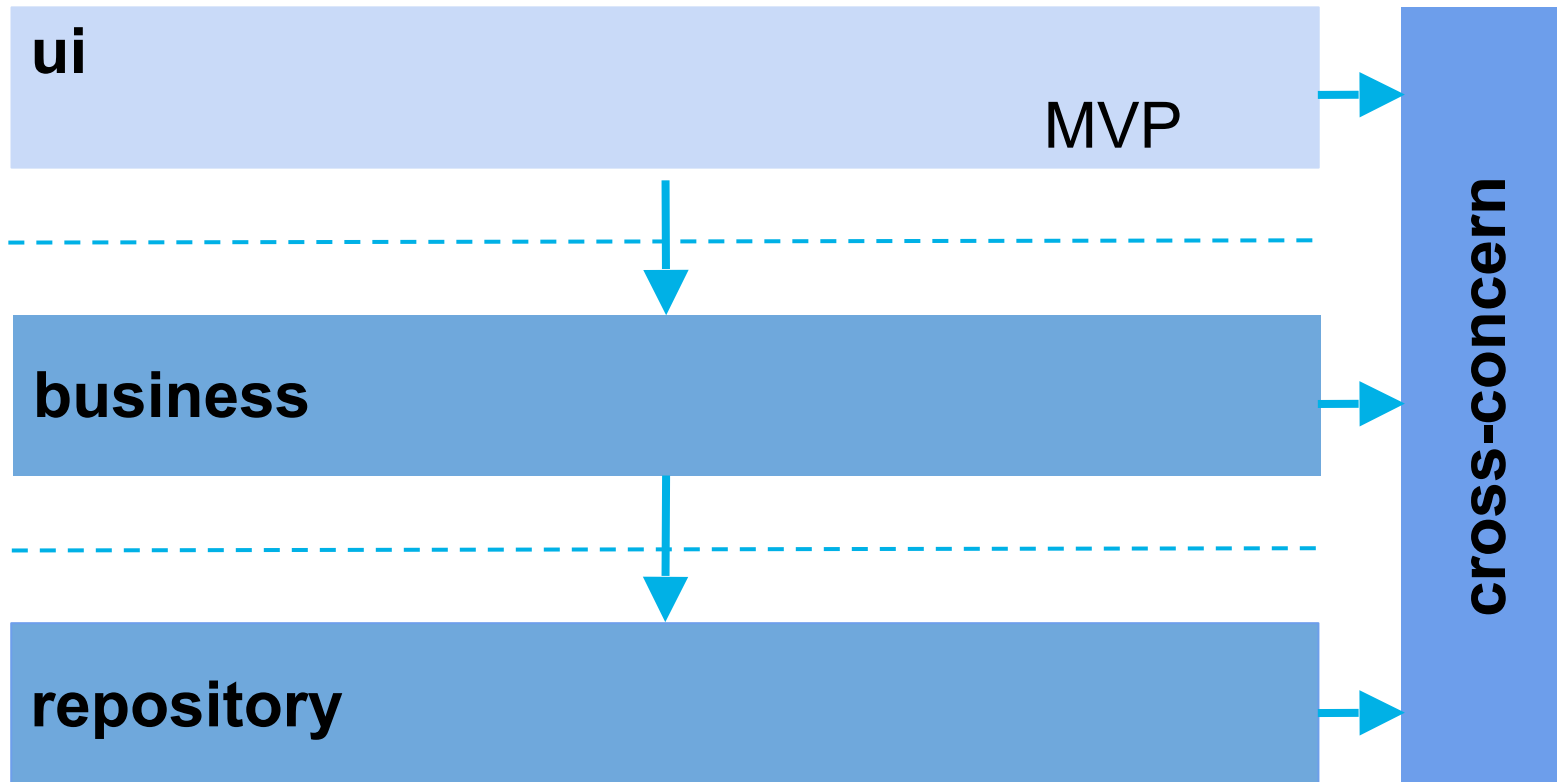
MVP



Separation of Concerns

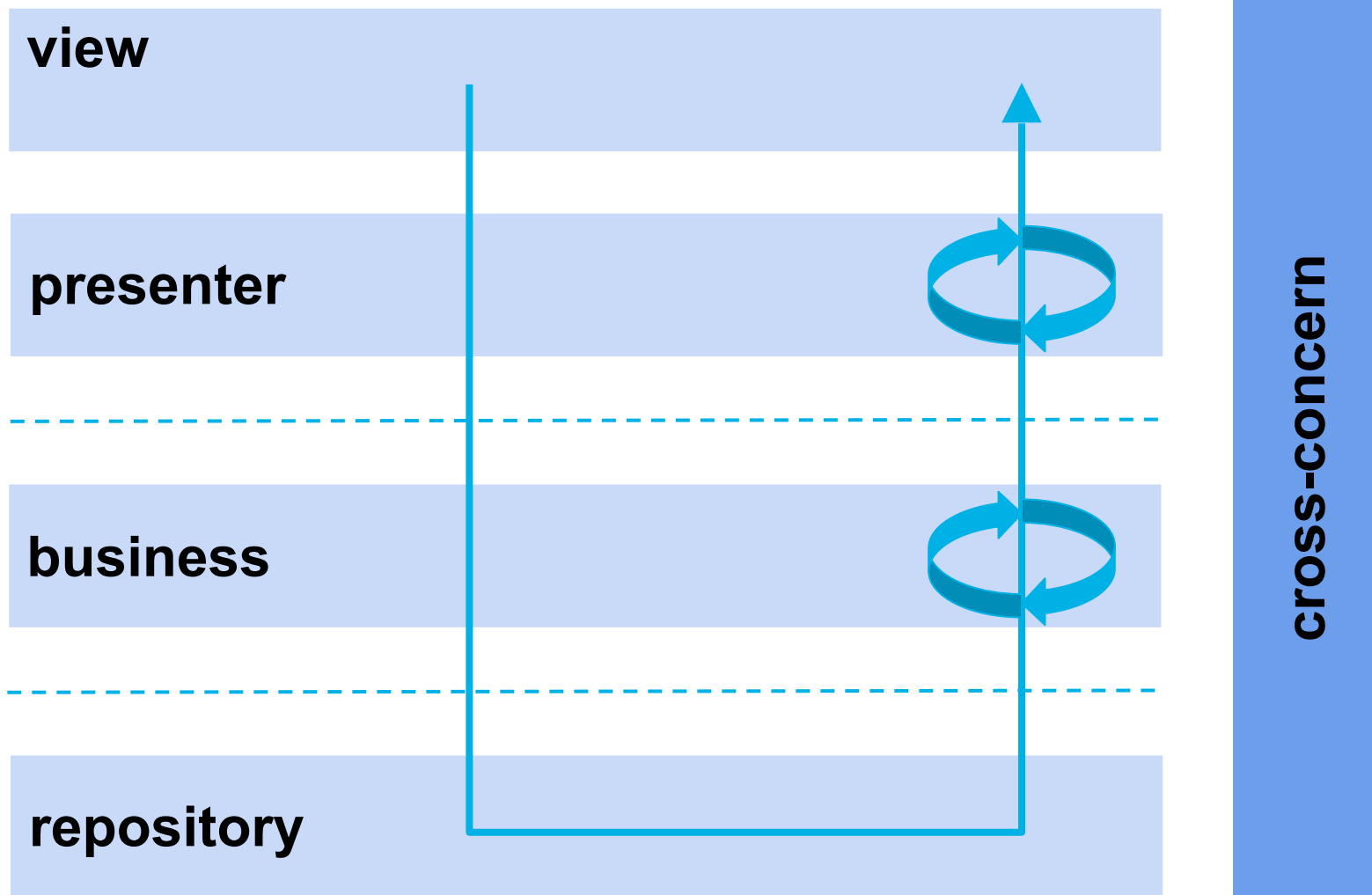


Layer & Dependencies



Dependency Rule →

Flow



RxJava over Bus?

More powerful and explicit...

... less easy to learn

3:0 for RxJava

- BUS just notifies and requires pull
 - *say data hell* to stateful events
- RxJava binds and pushes
- RxJava has built in data transformation
- solution for asynchronous tasks

+1 for extraordinary documentation

RxJava concepts

- Observable and Completable
 - (onNext)
 - onCompleted
 - onError

⇒ **Observer pattern on steroids**


RxJava concepts

- Operators
 - Creation
 - Transforming
 - Filtering
 - ...

RxJava concepts

- Schedulers
 - subscribeOn
 - observeOn
- Subscription

@Test

 public void filterMapAct() throws Exception {

// Arrange

List<String> result = new ArrayList<>();

Integer[] values = {1, 2, 3, 4, 5, 6};

```
Observable<String> observable = Observable
    .from(values)
    .filter(value -> isEven(value))
    .map(integer -> String.valueOf(integer))
    .doOnNext(string -> result.add(string));
```

// Act

observable.subscribe();

// Assert

assertEquals(3, result.size());

assertEquals("2", result.get(0));

assertEquals("4", result.get(1));

assertEquals("6", result.get(2));

}

RxJava in MVP + layers?

- data and events get observable
- emitted items can be transformed

⇒ **each layer can “decorate”**

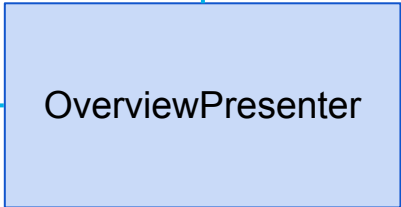
Fitsyou-34	80:44:3E:3F:A1:4A	CONNECT
Wristband	10:86:E5:E6:C1:4A	CONNECT
Goggle	32:47:23:EF:B1:4A	CONNECT
Xasd34	45:43:11:FF:E1:4A	CONNECT
Skoda-1	86:76:1E:45:A1:4A	CONNECT
MAC_GoGo	13:87:B4:E1:A1:4A	CONNECT
ASDGT_234	10:03:BF:76:D1:4A	CONNECT
123_DHSD	56:92:BE:33:D1:4A	CONNECT
FHSD	E3:01:BAE:35:31:4A	CONNECT
MalcolmX	AA:00:AE:76:41:4A	CONNECT

Fitsyou-34	80:44:3E:3F:A1:4A	CONNECT
Wristband	10:86:E5:E6:C1:4A	CONNECT
Goggle	32:47:23:EF:B1:4A	CONNECT
Xasd34	45:43:11:FF:E1:4A	CONNECT
Skoda-1	86:76:1E:45:A1:4A	CONNECT
MAC_GoGo	13:87:B4:E1:A1:4A	CONNECT
ASDGT_234	10:03:BF:76:D1:4A	CONNECT
123_DHSD	56:92:BE:33:D1:4A	CONNECT
FHSD	E3:01:BAE:35:31:4A	CONNECT
MalcolmX	AA:00:AE:76:41:4A	CONNECT

Hotspots

- generation of devices
- update of view
- connecting a device

UI



Business



OverviewFragment

OverviewPresenter

OverviewManager

OverviewActivity

IOverviewView

IOverviewPresenter

Business

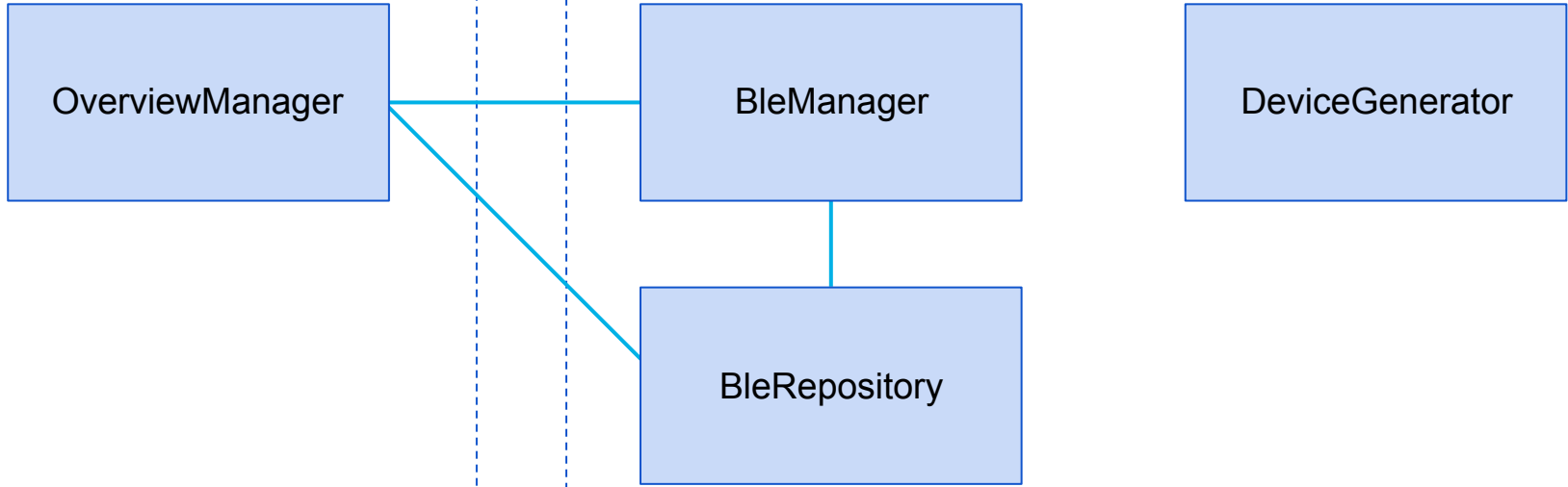
OverviewManager

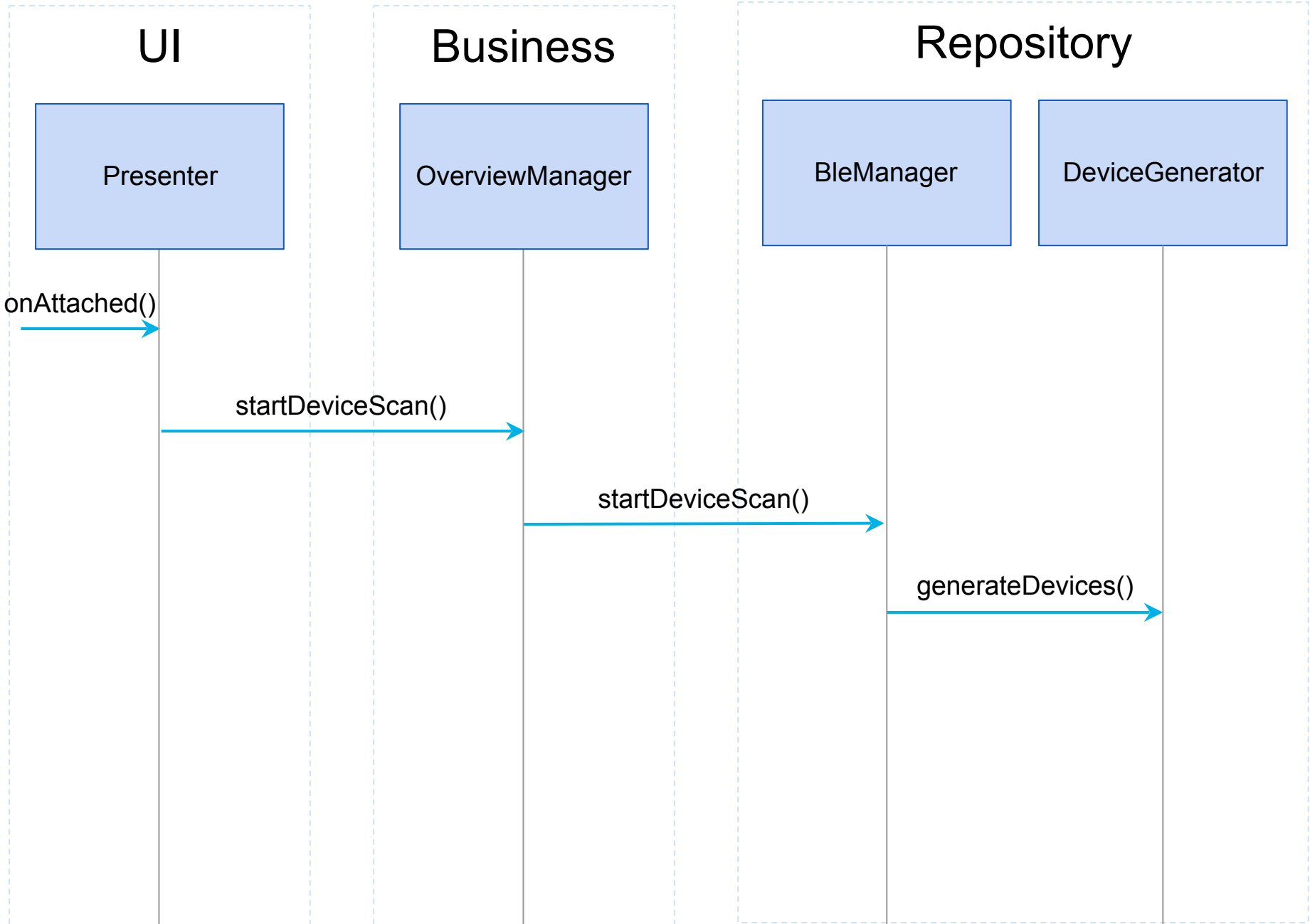
Repository <library>

BleManager

BleRepository

DeviceGenerator





```

public class DeviceGenerator {

    private static final String TAG = DeviceGenerator.class.getSimpleName();

    @Inject
    BleDeviceRepository repo;

    private int count = 0;
    private int delayInMs = 1000;

    private Subscription subscription;

    public void generateDevices() {

        if (subscription == null || subscription.isUnsubscribed()) {
            this.subscription = create()
                .subscribe();
        }
    }

    @VisibleForTesting
    Completable create() {

        return Completable
            .fromAction(this::createRandomDevice)
            .subscribeOn(Schedulers.computation())
            .observeOn(Schedulers.computation())
            .doOnError(throwable -> Log.e(TAG, "Could not create device.", throwable))
            .delay(delayInMs, TimeUnit.MILLISECONDS)
            .repeat();
    }

    private void createRandomDevice() {

        String address = addresses[count % addresses.length];
        String name = names[count % names.length];
        int rssi = (int) (Math.random() * 100);
        BleDevice bleDevice = new BleDevice(address, name, rssi, null, null);

        repo.put(bleDevice);
        count++;
    }
}

```

```
public class DeviceGeneratorTest {

    @Mock
    Handler          mockHandler;
    @Mock
    BleDeviceRepository mockRepo;

    DeviceGenerator testee;

    @Before
    public void setUp() {

        MockitoAnnotations.initMocks(this);

        testee = new DeviceGenerator();
        testee.repo = mockRepo;
    }

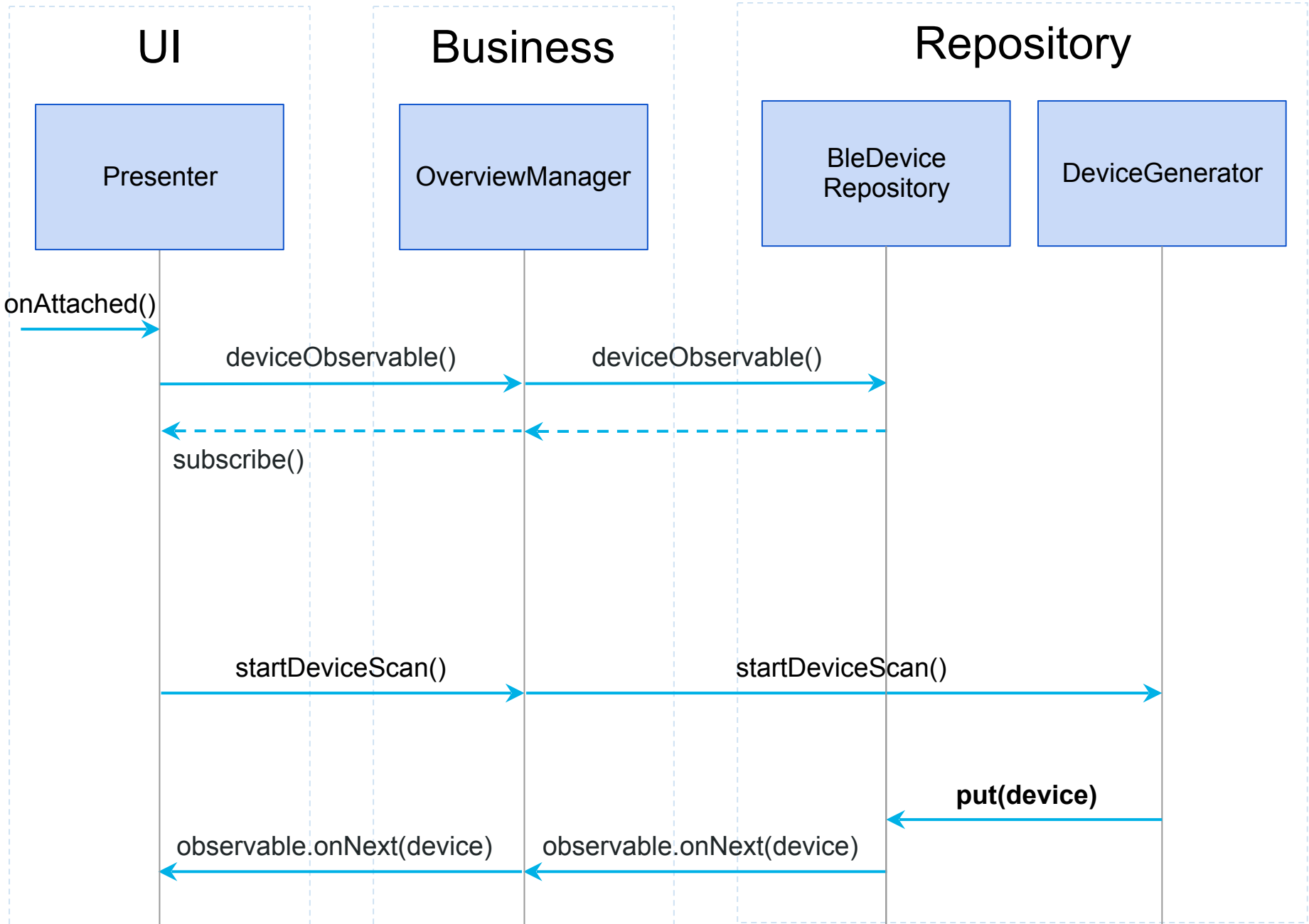
    @Test
    public void generateDevices() throws Exception {

        // Act
        testee.generateDevices();
        // Assert
        assertNotNull(testee.getSubscription());
        assertFalse(testee.getSubscription().isUnsubscribed());
    }

    @Test
    public void create_completable_and_await() {
        // Arrange
        testee.setDelayInMs(50);

        // Act
        Completable completable = testee.create();
        completable.await(120, TimeUnit.MILLISECONDS);

        // Assert
        verify(mockRepo, times(3)).put(any());
    }
}
```



```
@Singleton
```

```
public class BleDeviceRepository {
```

```
    private Map<String, BleDevice> devices = new HashMap<>();  
    ReplaySubject<BleDevice> deviceSubject = ReplaySubject.create();
```

```
@Inject
```

```
public BleDeviceRepository() {
```

```
}
```

```
/**  
 * Adds the device to the repo. If the device already exists in the repo it is overwritten.  
 *  
 * @param device  
 */
```

```
public void put(BleDevice device) {
```

```
    if (device != null) {  
        devices.put(device.getAddress(), device);  
        deviceSubject.onNext(device);  
    }
```

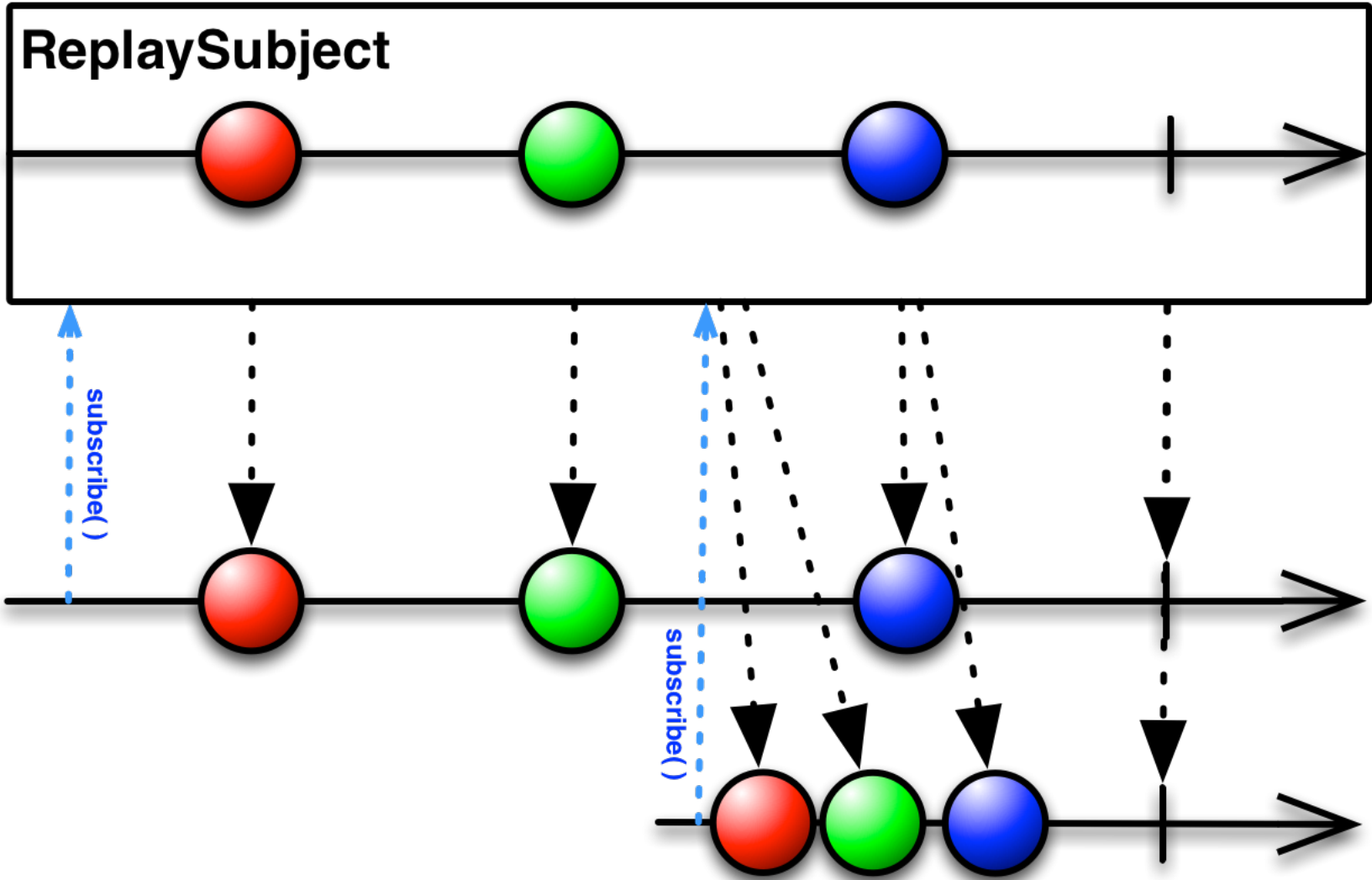
```
}
```

```
/**  
 * Getter for an observable for new {@link BleDevice}s  
 *  
 * @return  
 */
```

```
public Observable<BleDevice> deviceObservable() {
```

```
    return deviceSubject;
```

```
}
```



```
public class BleDeviceRepositoryTest {

    private BleDeviceRepository testee;

    @Before
    public void setUp() {

        testee = new BleDeviceRepository();
    }

    @Test
    public void observable_emits_items() throws Exception{

        // Arrange
        BleDevice device1 = new BleDevice("address1", "name1", 0, new byte[0], null);
        BleDevice device2 = new BleDevice("address2", "name2", 0, new byte[0], null);
        TestSubscriber<BleDevice> testSubscriber = new TestSubscriber<>();
        testee.deviceObservable().subscribe(testSubscriber);

        // Act
        testee.put(device1);
        testee.put(device2);

        // Assert
        testSubscriber.assertValues(device1, device2);
    }
}
```

```
@Singleton
public class OverviewPresenter extends RxPresenter implements IOverviewPresenter {

    @Override
    public void onViewAttached() {

        super.onViewAttached();
        observeDevices();
        overviewManager.startDeviceScan();
        initList();
    }
}
```

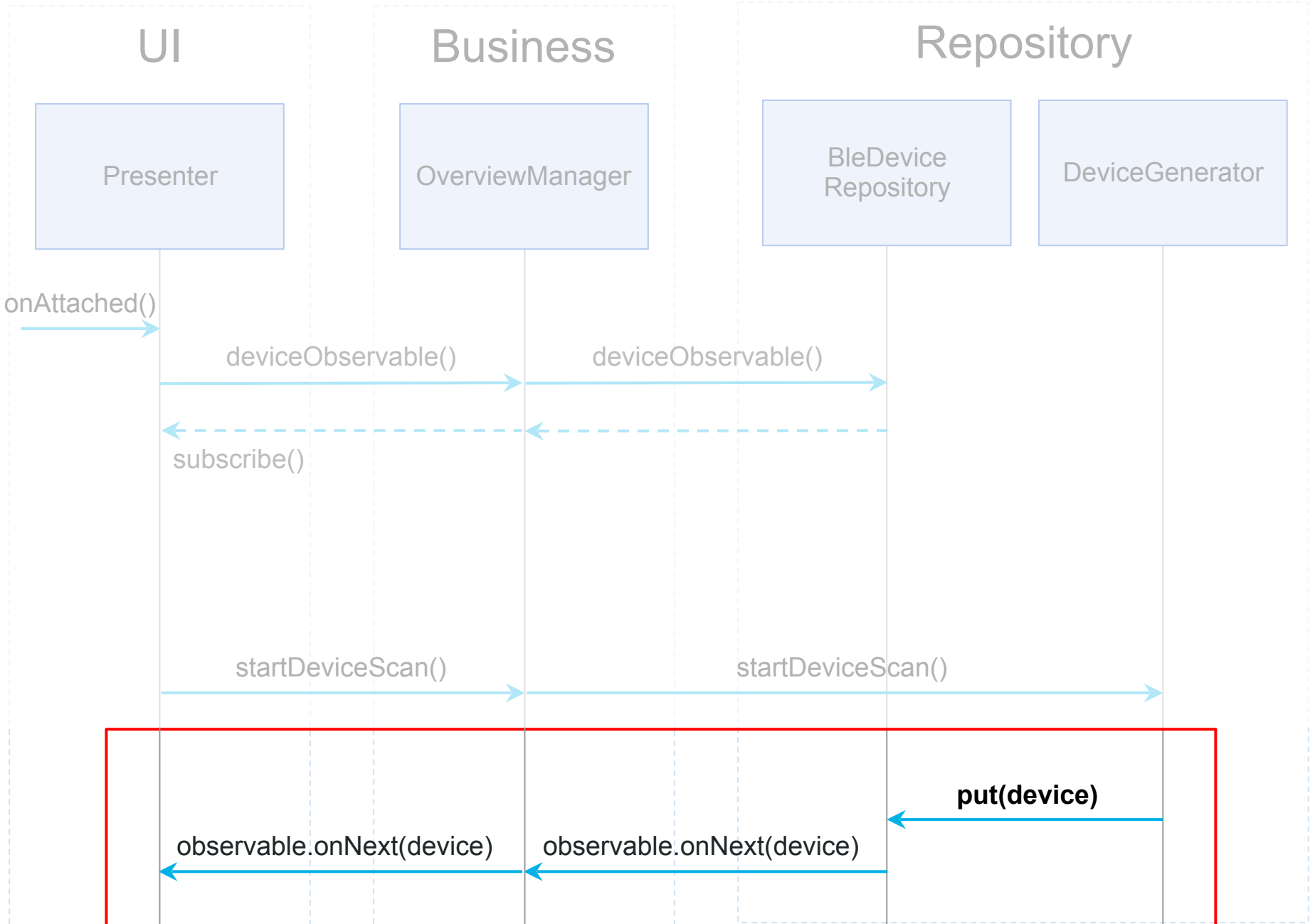
```
private void observeDevices() {

    Subscription subscription = createDeviceObserver()
        .subscribe();

    addSubscription(subscription);
}
}
```

```
@VisibleForTesting
Observable<BleDevice> createDeviceObserver() {

    return overviewManager.deviceObservable()
        .observeOn(AndroidSchedulers.mainThread())
        .doOnNext(device -> view.addDeviceItem(device));
}
```

Business

Repository

OverviewManager

BleManager

Device
Generator

Device
Repository

startDeviceScan()

put(device)

bus.post(new EventRepoAddedNewDevice)

bus.post(...)

getDevice(address)

getDevice(address)

```
@Singleton
public class OverviewPresenter extends RxPresenter implements IOverviewPresenter {
```

```
    @Inject
    OverviewManager overviewManager;
    private IOverviewView view;
```

```
    @Override
    public void onConnectButtonClick(BleDevice device) {

        overviewManager.setSelectedDevice(device);

        Subscription subscription = overviewManager
            .connect(device)
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(
                theDevice -> {},
                error -> showError(device),
                () -> showConnectedDevice(device));

        addSubscription(subscription);
    }
}
```

```
public class OverviewManager extends DeviceManager {

    private static final String TAG = OverviewManager.class.getSimpleName();

    @Inject
    BleManager bleManager;
    @Inject
    BleDeviceRepository bleDeviceRepository;
    @Inject
    MtcPreferences preferences;
```



```
public Observable<BleDevice> connect(BleDevice device) {

    return bleManager.connect(device)
        .doOnCompleted(() -> Log.i(TAG, "Connection established. "))
        .doOnError(t -> Log.e(TAG, "Error: connect failed. "))
        .retry(1);
}
```

```
public class BleManager {

    private static final long    DELAY_IN_MS = 300;
    private static final String TAG        = BleManager.class.getSimpleName();

    @Inject
    BleDeviceRepository repo;
    @Inject
    DeviceGenerator     generator;

    public Observable<Void> connect(BleDevice device) {
        return Observable
            .just((Void)null)
            .delay(DELAY_IN_MS, TimeUnit.MILLISECONDS)
            .doOnNext(d -> fakeConnect(device));
    }

    private BleDevice fakeConnect(BleDevice device) {

        if (Math.random() < 0.5) {
            throw new RuntimeException("Exception while connecting device.");
        } else {
            Log.i(TAG, "Connected with: " + device.getAddress());
        }

        return device;
    }
}
```

some more words about testing

- Passive View
- Presenter is Java-only
- Contract between View/Presenter
- Separate configuration & execution

THE UGLY TRUTH

Memory Leaks

- Remember anonymous classes?
=> reference enclosing class

Memory Leaks

- Care for your Subscription(s)
 - unsubscribe is not enough
 - CompositeSubscription.clear()

```
public class RxPresenter implements IPresenter {  
  
    protected CompositeSubscription compositeSubscription = new CompositeSubscription();  
  
    @Override  
    public void onViewAttached() {  
  
    }  
  
    @Override  
    public void onViewDetached() { compositeSubscription.clear(); }  
}
```

Sometimes strange

- async configuration surprises sometimes
- async nested commands with loops

Summary

- KISS / SoC and clarity are Key
- straighten out
- RxJava has a learning curve ;-)

Thank you!

“No rules are universal”

(except this one)

About

- **BINOSYS** GmbH
 - <http://binosys.de>
 - info@binosys.de

Links

- Memory Leaks

- <https://medium.com/@scanarch/how-to-leak-memory-with-subscriptions-in-rxjava-ae0ef01ad361#.trg5p6fn4>

- MVVM

- <https://en.wikipedia.org/wiki/Model-view-viewmodel>
- <https://medium.com/@manuelvicnt/rxjava-android-mvvm-app-structure-with-retrofit-a5605fa32c00#.x17crfmof>

- MVVM

- <https://www.infoq.com/articles/Testing-RxJava>

- RxLifecycle









- <https://github.com/trello/RxLifecycle>

Layers start with packages...

... and packages give structure to source

- ▼ java
 - ▼ de
 - ▼ company.app
 - ▶ about
 - ▶ chart
 - ▶ dashboard
 - ▶ database
 - ▶ dev
 - ▶ device
 - ▶ helper
 - ▶ howto
 - ▶ info
 - ▶ init_profil
 - ▶ profile
 - ▶ services
 - ▶ settings
 - ▶ slider
 - ▶ statistics
 - ▶ training
 - ▶ utilities
 - ⊙ AppModule
 - ⊙ BaseActivity
 - ⊙ InfoActivity
 - ⊙ MainActivity
 - ⊙ MyApp
 - ⊙ MyProfileActivity
 - ⊙ SetupActivity
 - ⊙ SplashAct



- ▼  java
 - ▼  ch/sbb/cisi/mobzk/client/android
 - ▶  business
 - ▶  crossconcern
 - ▶  repository
 - ▶  ui
 -   MobZKApplication

