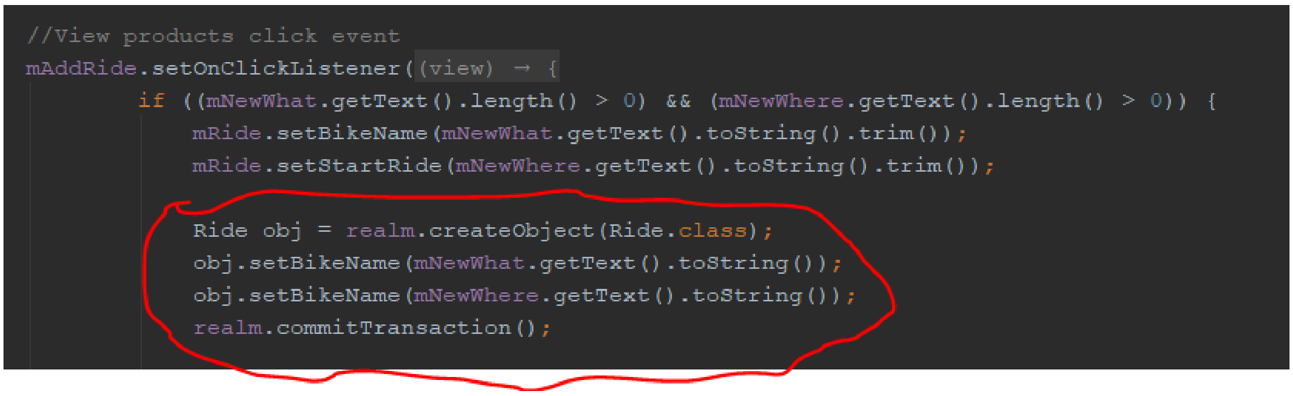
Jeppe Nannestad Dyekjær

Report assigment 2 - BikeShare

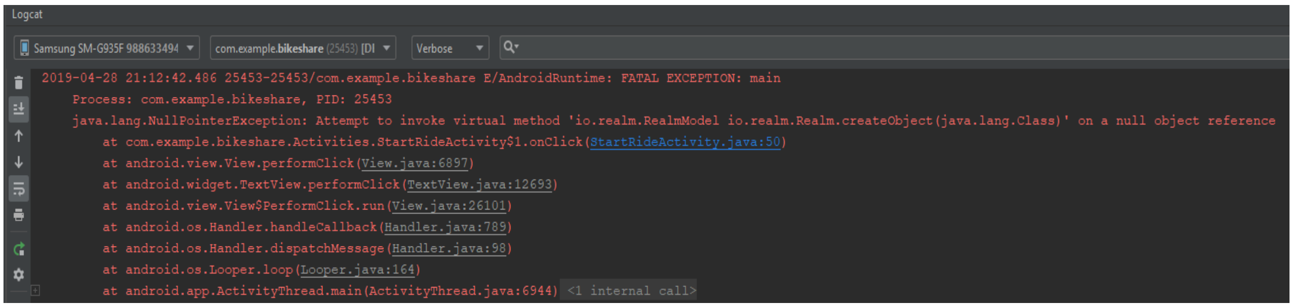
Problems (most important at the moment):
Currently I have tried to set up a Realm database but it's not working at the moment.
I'm getting this NullPointerException when trying to create the database in my onClick-method in the StartRideActivity. I'm unsure how to implement the Realm.init() and other initializations of the Realm objects in the code. Currently Realm.init() and realm = Realm.getDefaultInstance(); is implemented in BikeShareActivity right before the initialization of the FragmentManager.



```
//View products click event
mAddRide.setOnClickListener((view) → {
        if ((mNewWhat.getText().length() > 0) && (mNewWhere.getText().length() > 0)) {
            mRide.setBikeName(mNewWhat.getText().toString().trim());
            mRide.setStartRide(mNewWhere.getText().toString().trim());

            Ride obj = realm.createObject(Ride.class);
            obj.setBikeName(mNewWhat.getText().toString());
            obj.setBikeName(mNewWhere.getText().toString());
            realm.commitTransaction();
```

I have tried to implement parts of it but unfortunately it doesn't work at the moment. The program can compile but creates the below NullPointerException when I try to add the string values from the TextViews in the StartRideActivity to the database (which is shown in the above picture of the onClickListener().



```
2019-04-28 21:12:42.486 25453-25453/com.example.bikeshare E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.example.bikeshare, PID: 25453
    java.lang.NullPointerException: Attempt to invoke virtual method 'io.realm.RealmModel io.realm.Realm.createObject(java.lang.Class)' on a null object reference
        at com.example.bikeshare.Activities.StartRideActivity$1.onClick(StartRideActivity.java:50)
        at android.view.View.performClick(View.java:6897)
        at android.widget.TextView.performClick(TextView.java:12693)
        at android.view.View$PerformClick.run(View.java:26101)
        at android.os.Handler.handleCallback(Handler.java:789)
        at android.os.Handler.dispatchMessage(Handler.java:98)
        at android.os.Looper.loop(Looper.java:164)
        at android.app.ActivityThread.main(ActivityThread.java:6944) <1 internal call>
```

Most important design choices, for example, class and layout structures.
The overall design of the program is made of activities. All activity classes do also contain an activity xml-file which interact with each other. Each component in the XML-files are identified with unique id's in order to make it easier for the Java classes to refer to their components (buttons, text views etc.).

The BikeShareActivity-class is the parent activity for all the other activities (EndRideActivity and StartRideActivity). All the rides for bikes can be stored in a list which is identified in

BikeShareActivity. This list is used by both StartRideActivity and EndRideActivity to add a bike and its start location and the end location for the ride.
The BikeShareActivity have refers to a BikeShareFragment with two setOnClickListener() which can navigate to either StartRideActivity or EndRideActivity. The EndRideActivity is only able to store values in the list when the StartRideActivity has stored values in the list at first. All the components in the xml-activities are defined in linear layouts in order to split the page into more parts. As an example, the list of rides on BikeShareActivity is shown in the layout (activity_bike_share/fragment_bike_share) as a list view. All the components of the list view are added in the getView-method in the RideArrayAdapter-class with unique ids for the bike, start- and end ride. These ids are in the layout defined in another xml-file called list_item_ride.

All the more detailed layout is stored in the folder called values underneath the res-folder. Furthermore, that includes colors for background and texts and sizes of fonts. Each color and font size have got a unique name, which the XML-files refers to when creating a component in the UI.

Short explanation about the user interface of your BikeShare app:
When the app starts to run, the front page is shown. From there you can either choose to add or end a ride by pressing one of the two red buttons. Because there is no added ride when the program starts up, the user should press the “Add ride”-button. Then the user can write a name of the bike in the first text field “What bike?” and in the second text field “Where” the location of the bike. After the text fields are filled out, the user can press the “Add ride”-button. Then the user gets navigated back to the main page. Then the orange list view displays the values which have been added to the list. Afterwards the user can end the ride by pressing the “End ride”-button. From there the user has to make sure that he/she has typed in the exact same name of the added bike to be able to store an ending location of the bike which is typed in the “Ending where?”-text field. At last the user can press the “End ride”-button and the end ride location should be stored to the related bike in the recycler view. In order to view the elements in the list the user presses the “Show rides”-button. In order to delete a ride the user presses the “Delete ride”-button and simply types in the name of the bike he/she wants to delete.

Extensions compared to BikeShare app
Fragments have been implemented and work fully. A date timer for when the given bike ride has been added to the list has been added to the list do also work. Deleting a ride was also an extra feature which works.

How did you test and evaluate your app?
I was mostly using the build in debugger to check that the given values were correct when typed in some input in the given text fields. When running the app, I used my phone to test the UI and functionality.