# Project Part 6 New Test Cases: Realm-Java

**Authors: Wen-Chia Yang, Junxian Chen, Zihua Weng**

**March 15, 2020**

# ABSTRACTION

## Objectives

In this report, we code new test cases for realm-java to cover existing functionality and new functionality we contribute for the second issue.

## Source Code

Repository: https://github.com/solution-accepted/realm-java
Branch: new-test-cases

# RESEARCH

## New Understanding of realm-java

We first studied the existing test cases in RealmInMemoryTest.java and IOSRealmTests.java, they gave us a bigger picture about how in-memory Realm instances interact with each other and how Realm deals with multi-platform application. When reading the test cases, we had other assumptions about the system and simulated the possible output that Realm may return. Then we read the source code to verify our assumptions. Here, We would like to know if new data will be included if the writeCopyTo function is called in transaction. We found out that new data will be written into the destination file too. For IOSRealmTests.java we would like to test if current code could process different iOS data types and null values. And we learned the outcome by reading the source code that the code supports both Android and iOS.

Afterwards, we believe that it will be a good idea to write new test cases about these situations to test the system. Also, the test cases provide more information of how those parts of Realm work for people who read the code.

Also, we added three new test cases to test new functionalities we added to fix our second [issue # 1050 https://github.com/realm/realm-java/issues/1050].

# TEST CASES IMPLEMENTATION

## Pull Request

For the first two new test cases: https://github.com/realm/realm-java/pull/6777

For the last three new test cases: https://github.com/realm/realm-java/pull/6778

## New Test Cases

<table>
<tr><th colspan="2">writeCopyToInTransaction in RealmInMemoryTest.java</th></tr>
<tr><td><b>Purpose</b></td><td>To test if the destination database file will write the current new data if it is called within the transaction.</td></tr>
<tr><td><b>Explanation</b></td><td>The test case first creates an in-memory Realm instance and starts a new transaction. Before committing the transaction, it calls writeCopyTo to write the instance into the destination. The new data should be written into the file.</td></tr>
<tr><td><b>Test Case</b></td><td><pre>
// Tests writeCopyTo result when called in a transaction.
    @Test
    public void writeCopyToInTransaction() {
        String fileName = IDENTIFIER + ".realm";
        RealmConfiguration conf = configFactory.createConfigurationBuilder()
                .name(fileName)
                .build();

        Realm.deleteRealm(conf);

        testRealm.beginTransaction();
        Dog dog = testRealm.createObject(Dog.class);
        dog.setName("DinoDog");

        // Write copy to destination file in transaction.
        // Check if the new data would be written into the file.
        testRealm.writeCopyTo(new File(configFactory.getRoot(), fileName));
        Realm onDiskRealm = Realm.getInstance(conf);
        assertEquals(1, onDiskRealm.where(Dog.class).count());

        testRealm.commitTransaction();

        assertEquals(1, testRealm.where(Dog.class).count());
        onDiskRealm.close();

    }
</pre></td></tr>
</table>

### iOSDataTypesMixValues in IOSRealmTests.java

| | |
|---|---|
| **Purpose** | To test mix cases of values and null values |
| **Explanation** | In this test case, we test loading data from .realm file. We start from loading realm files. We then test null values of byte array and String, and we also test values with short, integer, long, float, double and date. |

**Test Case**

```
@Test
public void iOSDataTypesMixValues() throws IOException {
  for (String iosVersion : IOS_VERSIONS) {
     configFactory.copyRealmFromAssets(context,
           "ios/" + iosVersion + "-alltypes-mix.realm", REALM_NAME);
     realm = Realm.getDefaultInstance();

     IOSAllTypes obj = realm.where(IOSAllTypes.class).findFirst();
     assertEquals(null, obj.getByteCol());
     assertEquals(null, obj.getStringCol());
     assertFalse(obj.isBoolCol());
     assertEquals(11125, obj.getShortCol());
     assertEquals(15350, obj.getIntCol());
     assertEquals(773863123, obj.getLongCol());
     assertEquals((float) 0.8914557, obj.getFloatCol(), 0F);
     assertEquals(0.8702290174167451, obj.getDoubleCol(), 0D);
     assertEquals(Long.MIN_VALUE, obj.getDateCol().getTime());
  }
}
```

## SOLUTION-ACCEPTED

<table>
<tr><th colspan="2">findFirst_limit in RealmQueryTests.java</th></tr>
<tr><td><b>Purpose</b></td><td>To fix issue # 1050, we added findLast(int), findFirst(int), findRandom(int). This test case is to test findFirst(int).</td></tr>
<tr><td><b>Explanation</b></td><td>In this test case, create a new transaction. Since RealmResults is disordered, we test the length of the query result to be the same as the input limit.</td></tr>
<tr><td><b>Test Case</b></td><td>

```java
@Test
public void findFirst_limit() {
  realm.beginTransaction();

  Owner owner1 = realm.createObject(Owner.class);
  owner1.setName("Owner 1");

  Owner owner2 = realm.createObject(Owner.class);
  owner2.setName("Owner 2");

  Owner owner3 = realm.createObject(Owner.class);
  owner3.setName("Owner 3");

  Owner owner4 = realm.createObject(Owner.class);
  owner4.setName("Owner 4");

  realm.commitTransaction();

  RealmResults<Owner> owners = realm.where(Owner.class).findFirst(2);
  assertEquals(2, subQueryResult.size());
}
```

</td></tr>
</table>

### findLast_limit in RealmQueryTests.java

| | |
|---|---|
| **Purpose** | To fix issue # 1050, we added findLast(int), findFirst(int), findRandom(int). This test case is to test findLast(int). |
| **Explanation** | In this test case, create a new transaction. Since RealmResults is disordered, we test the length of the query result to be the same as the input limit. |

**Test Case**

```
@Test
public void findLast_limit() {
  realm.beginTransaction();

  Owner owner1 = realm.createObject(Owner.class);
  owner1.setName("Owner 1");

  Owner owner2 = realm.createObject(Owner.class);
  owner2.setName("Owner 2");

  Owner owner3 = realm.createObject(Owner.class);
  owner3.setName("Owner 3");

  Owner owner4 = realm.createObject(Owner.class);
  owner4.setName("Owner 4");

  realm.commitTransaction();

  RealmResults<Owner> owners = realm.where(Owner.class).findLast(2);
  assertEquals(2, subQueryResult.size());
}
```

| findRandom_limit in RealmQueryTests.java | |
|---|---|
| **Purpose** | To fix issue # 1050, we added findLast(int), findFirst(int), findRandom(int). This test case is to test findRandom(int). |
| **Explanation** | In this test case, create a new transaction. Since RealmResults is disordered, we test the length of the query result to be the same as the input limit. |

**Test Case**

```
@Test
public void findRandom_limit() {
  realm.beginTransaction();

  Owner owner1 = realm.createObject(Owner.class);
  owner1.setName("Owner 1");

  Owner owner2 = realm.createObject(Owner.class);
  owner2.setName("Owner 2");

  Owner owner3 = realm.createObject(Owner.class);
  owner3.setName("Owner 3");

  Owner owner4 = realm.createObject(Owner.class);
  owner4.setName("Owner 4");

  realm.commitTransaction();

  RealmResults<Owner> owners = realm.where(Owner.class).findRandom(2);
  assertEquals(2, subQueryResult.size());
}
```