Realm Java for Android

Gokhan Arik

Android Developer at Client Resources Inc.

www.gokhanarik.com

What is Realm?

Replacement database for SQLite and CoreData

Cross platform mobile database

Founded by Alexander Stigsen and Bjarne Christiansen (former Nokia emp.)

Known as Tight DB in the past

100 million devices in 9 months

Switching to Realm

Why Realm?

faster than SQLite (up to 10x speed up over raw SQLite)

easy to use

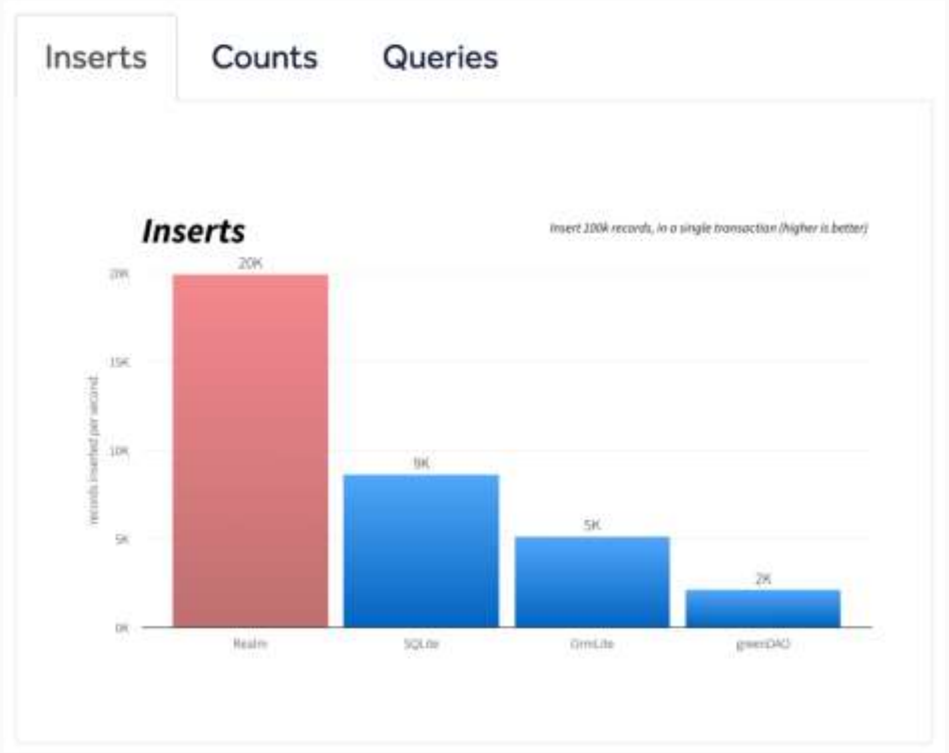
object conversion

it is free

documentation and support

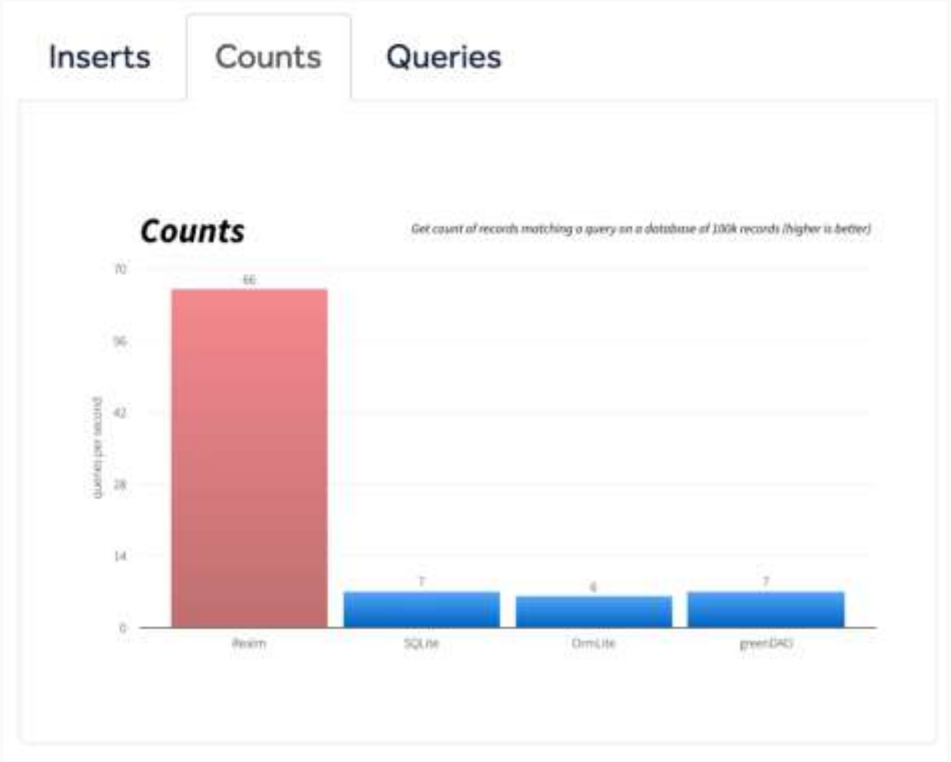
That’s why

Faster than SQLite (up to 10x speed up over raw SQLite)



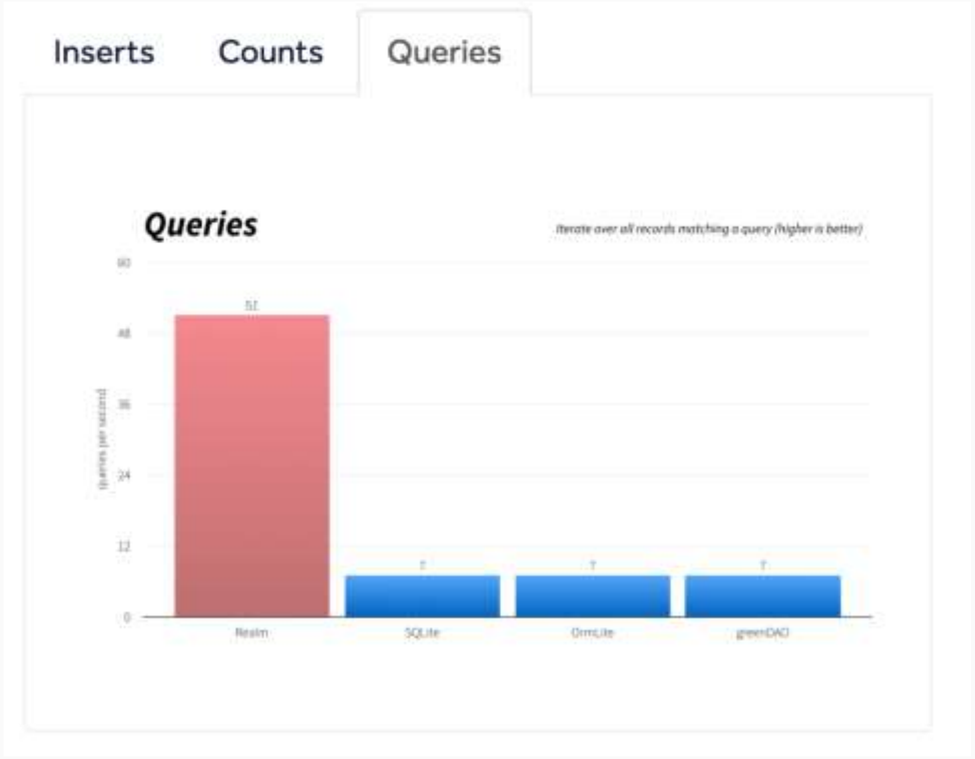
That’s why

Faster than SQLite (up to 10x speed up over raw SQLite)



That’s why

Faster than SQLite (up to 10x speed up over raw SQLite)



That’s why

Faster than SQLite (up to 10x speed up over raw SQLite)

Please check out this article about Kevin Galligan’s concerns about the performance of Realm

http://kpgalligan.tumblr.com/post/133281929963/my-talk-at-droidcon-uk

That’s why

Easy to use (not this one, next one)

```java
public class DBHelper extends SQLiteOpenHelper {

    public static final String DATABASE_NAME = "MyDBName.db";
    public static final String CONTACTS_TABLE_NAME = "contacts";
    public static final String CONTACTS_COLUMN_ID = "id";
    public static final String CONTACTS_COLUMN_NAME = "name";
    public static final String CONTACTS_COLUMN_EMAIL = "email";
    public static final String CONTACTS_COLUMN_STREET = "street";
    public static final String CONTACTS_COLUMN_CITY = "place";
    public static final String CONTACTS_COLUMN_PHONE = "phone";
    private HashMap hp;

    public DBHelper(Context context)
    {
        super(context, DATABASE_NAME , null, 1);
    }
```

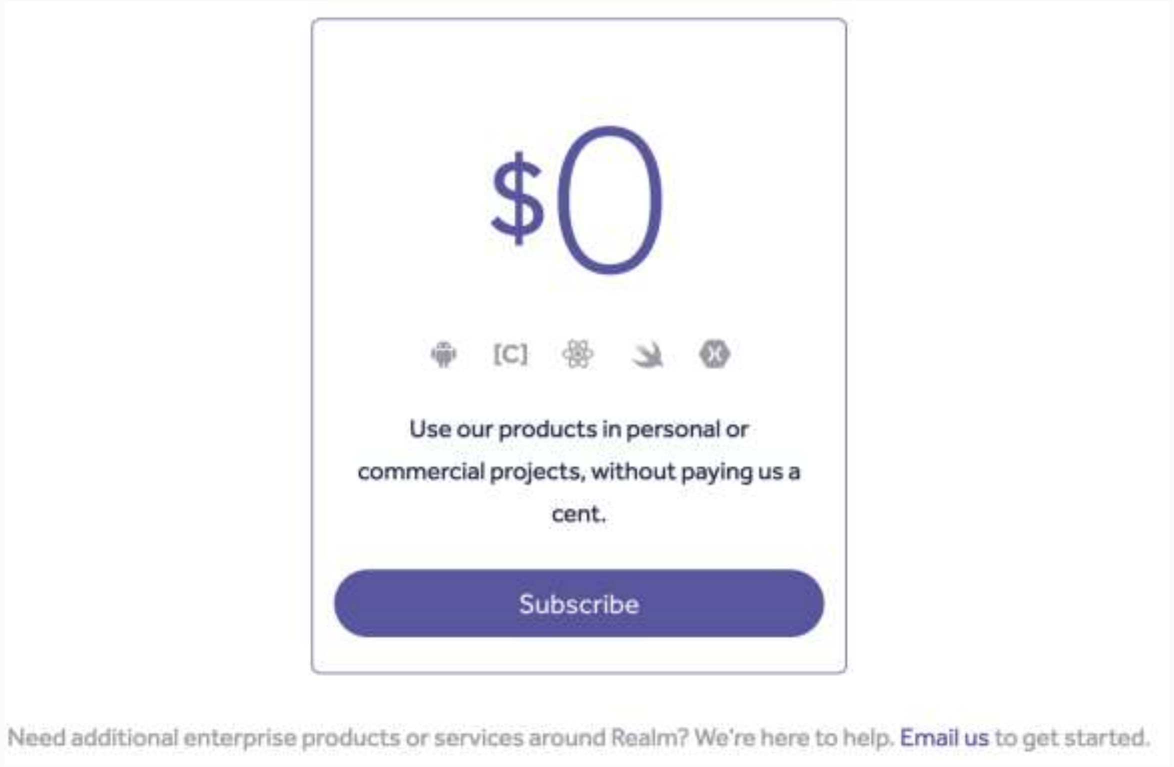
That’s why

Object conversion

```
public class Person extends RealmObject {
    @PrimaryKey
    private long id;
    private String name;
    private RealmList<Dog> dogs; // Declare one-to-many relationships

    // ... Generated getters and setters ...
}
```

That’s why

It is free



$0

[C]

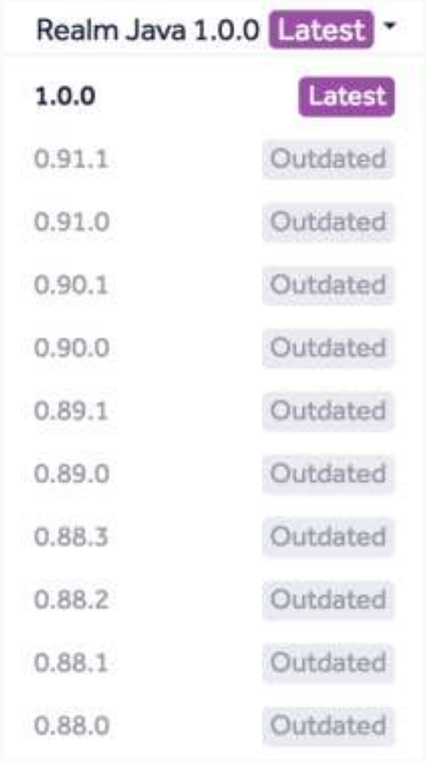
Use our products in personal or commercial projects, without paying us a cent.

Subscribe

Need additional enterprise products or services around Realm? We're here to help. Email us to get started.

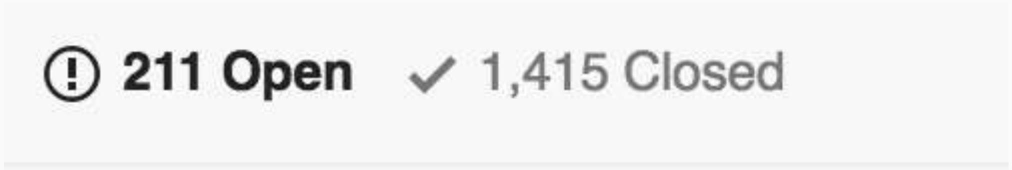
That’s why

Documentation and Support

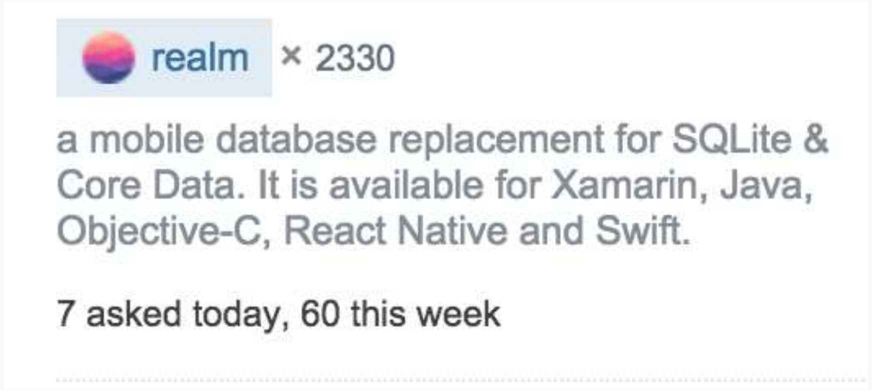


Realm Java 1.0.0 Latest

1.0.0	Latest
0.91.1	Outdated
0.91.0	Outdated
0.90.1	Outdated
0.90.0	Outdated
0.89.1	Outdated
0.89.0	Outdated
0.88.3	Outdated
0.88.2	Outdated
0.88.1	Outdated
0.88.0	Outdated



211 Open ✓ 1,415 Closed

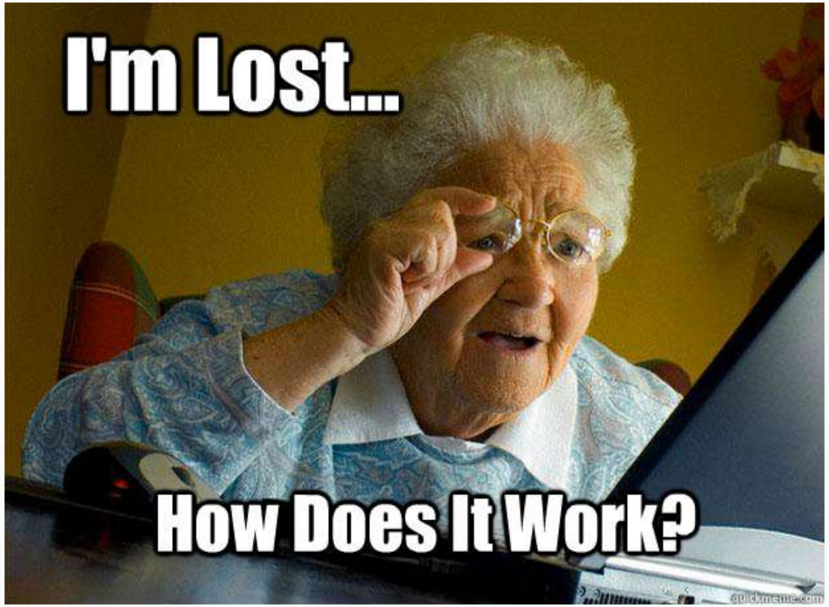


realm × 2330

a mobile database replacement for SQLite & Core Data. It is available for Xamarin, Java, Objective-C, React Native and Swift.

7 asked today, 60 this week

How does it work?



Like this (as of version 1.0.0)

No Maven or Ant support

Supports Android since API Level 9 (Android 2.3 Gingerbread & above)

As of version 1.0.0 Eclipse is not supported

No need Proguard configuration. It is included in library

Like this (as of version 1.0.0)

Step 1: Add the following class path dependency to the project level build.gradle file.

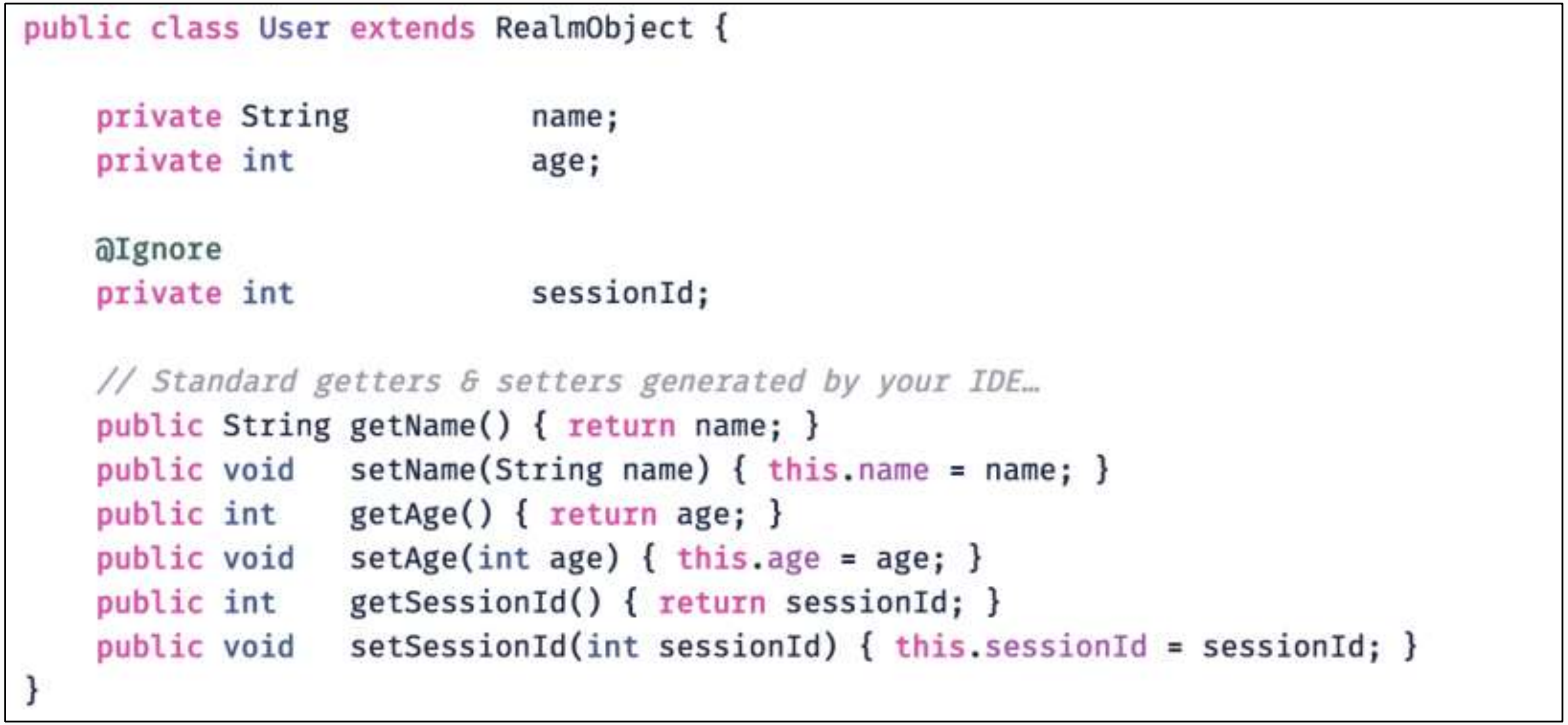
```
buildscript {
    repositories {
        jcenter()
    }
    dependencies {
        classpath "io.realm:realm-gradle-plugin:"
    }
}
```

Step 2: Apply the realm-android plugin to the top of application level build.gradle file

```
apply plugin: 'realm-android'
```

Like this (as of version 1.0.0)

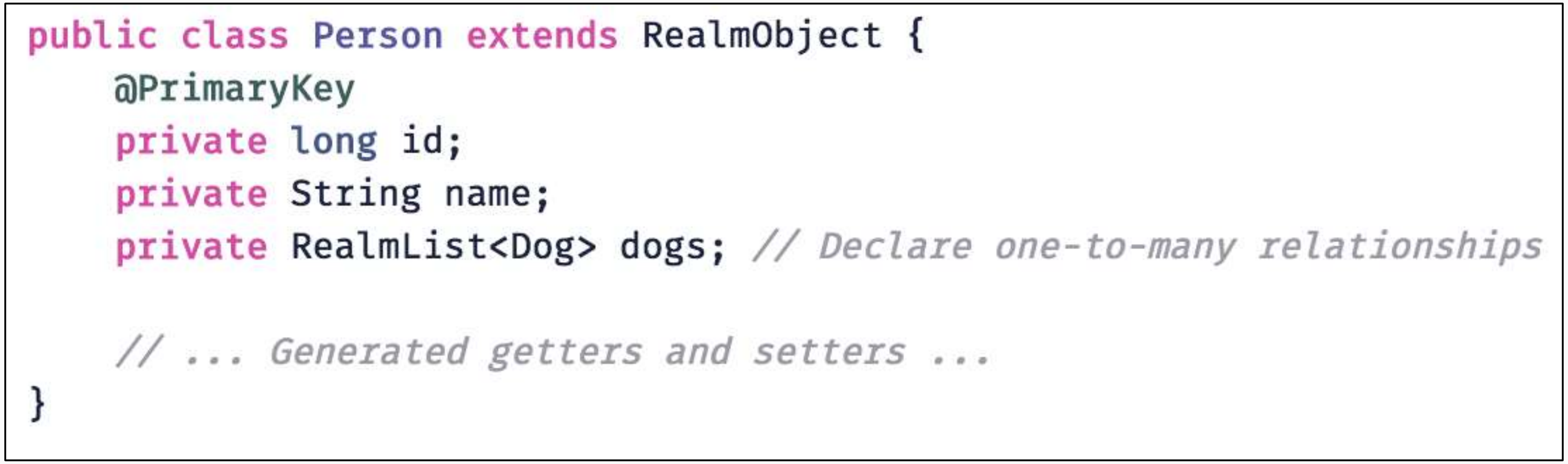
Models



```java
public class User extends RealmObject {

    private String          name;
    private int             age;

    @Ignore
    private int             sessionId;

    // Standard getters & setters generated by your IDE…
    public String getName() { return name; }
    public void   setName(String name) { this.name = name; }
    public int    getAge() { return age; }
    public void   setAge(int age) { this.age = age; }
    public int    getSessionId() { return sessionId; }
    public void   setSessionId(int sessionId) { this.sessionId = sessionId; }
}
```

Like this (as of version 1.0.0)

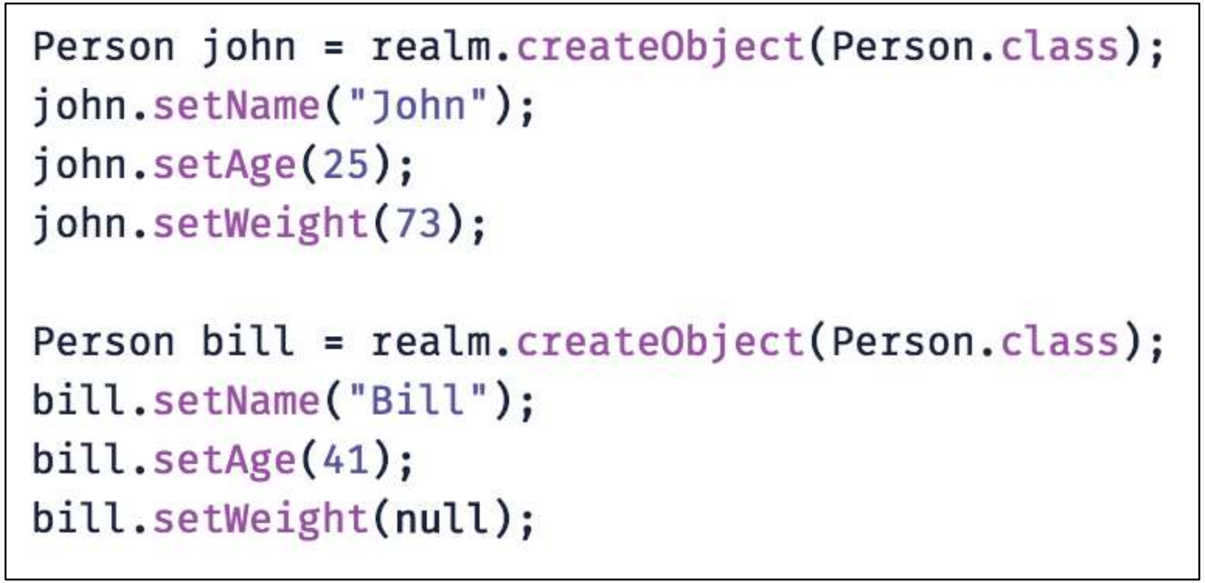
Models



```java
public class Person extends RealmObject {
    @PrimaryKey
    private long id;
    private String name;
    private RealmList<Dog> dogs; // Declare one-to-many relationships

    // ... Generated getters and setters ...
}
```

Like this (as of version 1.0.0)

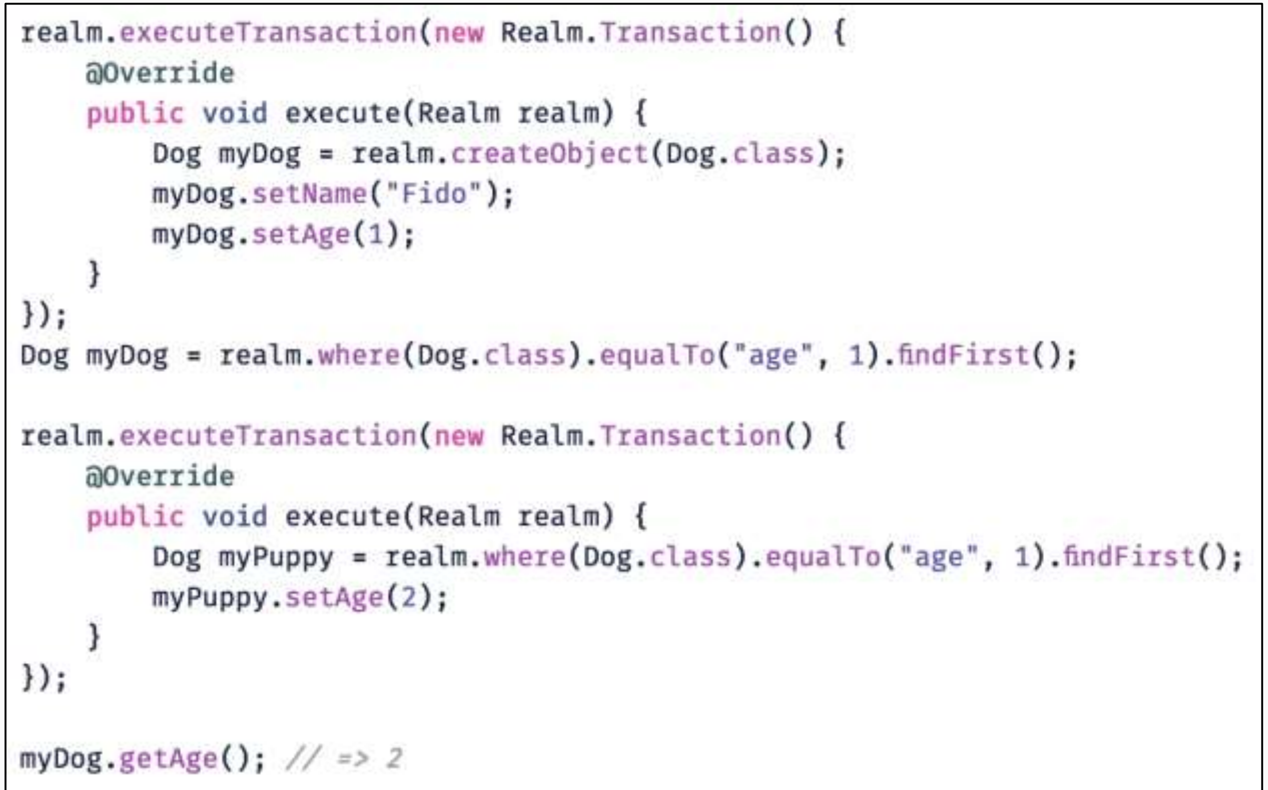
Models



```
Person john = realm.createObject(Person.class);
john.setName("John");
john.setAge(25);
john.setWeight(73);

Person bill = realm.createObject(Person.class);
bill.setName("Bill");
bill.setAge(41);
bill.setWeight(null);
```

Like this (as of version 1.0.0)

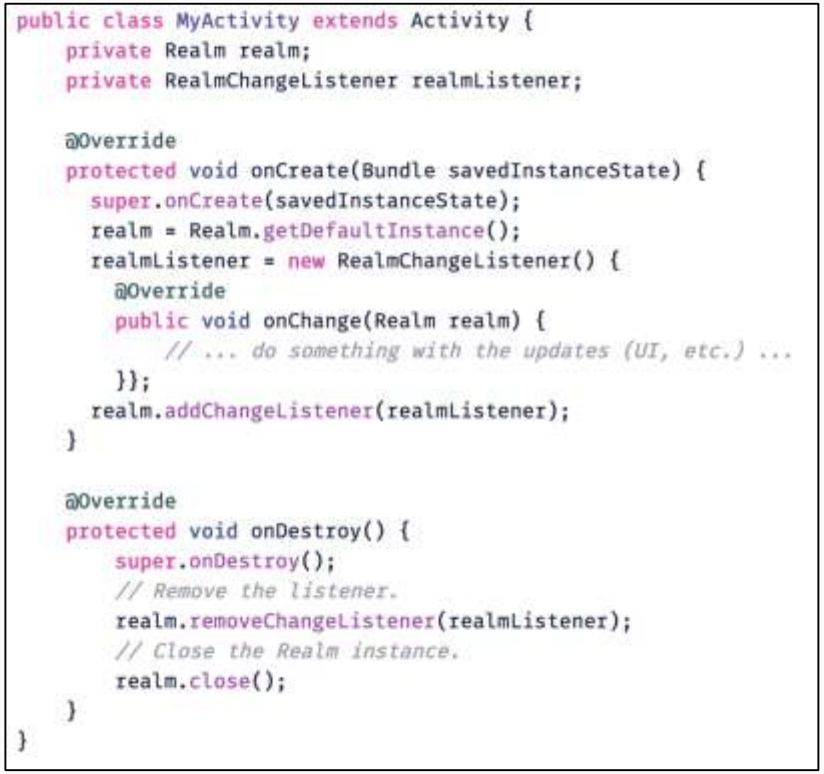
Models - Auto-Update



```java
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        Dog myDog = realm.createObject(Dog.class);
        myDog.setName("Fido");
        myDog.setAge(1);
    }
});
Dog myDog = realm.where(Dog.class).equalTo("age", 1).findFirst();

realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        Dog myPuppy = realm.where(Dog.class).equalTo("age", 1).findFirst();
        myPuppy.setAge(2);
    }
});

myDog.getAge(); // => 2
```

Like this (as of version 1.0.0)

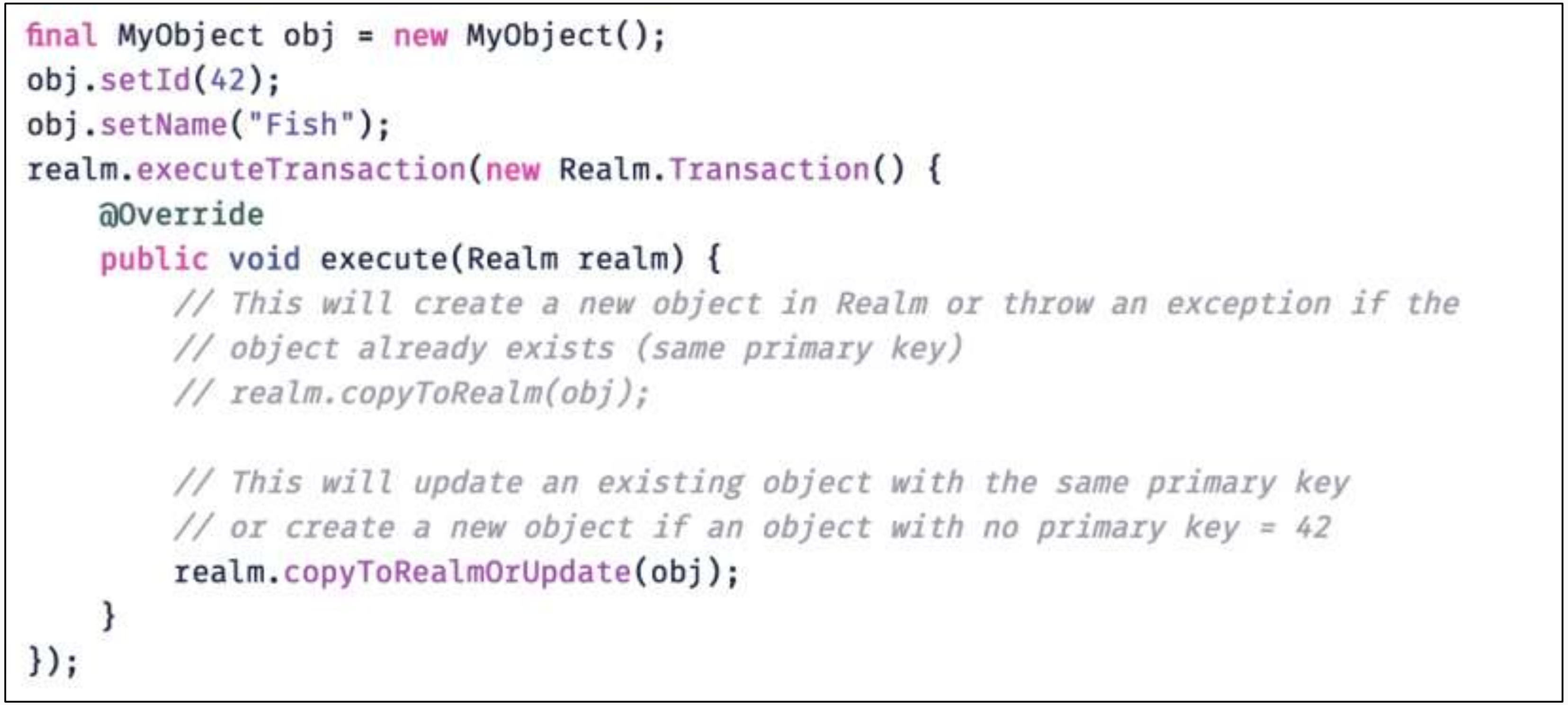
Models - Notifications



```java
public class MyActivity extends Activity {
    private Realm realm;
    private RealmChangeListener realmListener;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      realm = Realm.getDefaultInstance();
      realmListener = new RealmChangeListener() {
        @Override
        public void onChange(Realm realm) {
            // ... do something with the updates (UI, etc.) ...
        }};
      realm.addChangeListener(realmListener);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Remove the listener.
        realm.removeChangeListener(realmListener);
        // Close the Realm instance.
        realm.close();
    }
}
```

Like this (as of version 1.0.0)

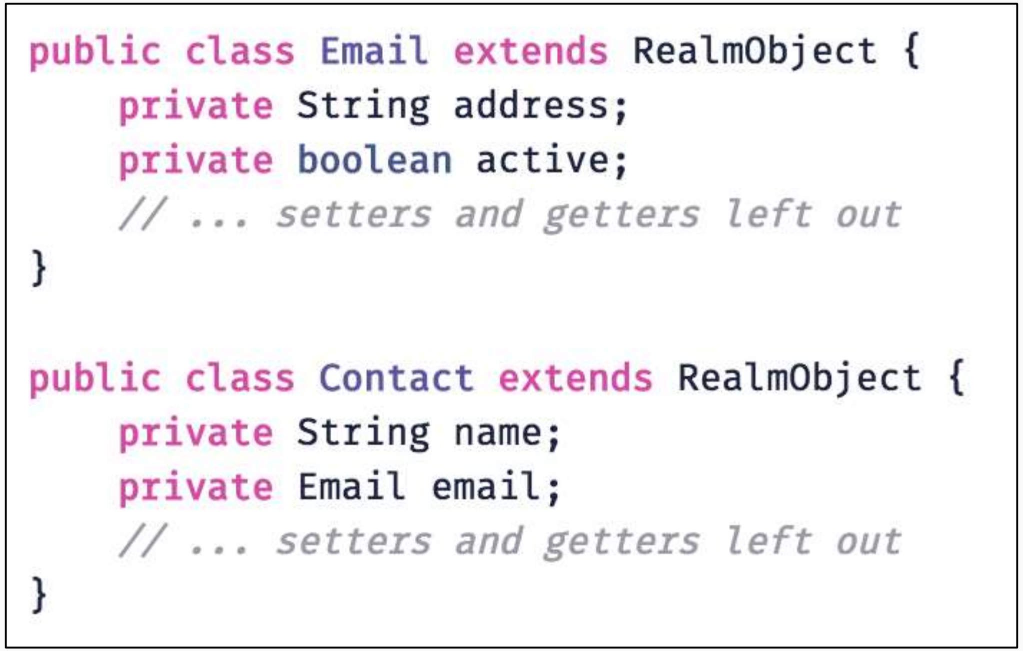
Models - Primary Keys



```java
final MyObject obj = new MyObject();
obj.setId(42);
obj.setName("Fish");
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        // This will create a new object in Realm or throw an exception if the
        // object already exists (same primary key)
        // realm.copyToRealm(obj);

        // This will update an existing object with the same primary key
        // or create a new object if an object with no primary key = 42
        realm.copyToRealmOrUpdate(obj);
    }
});
```

Like this (as of version 1.0.0)

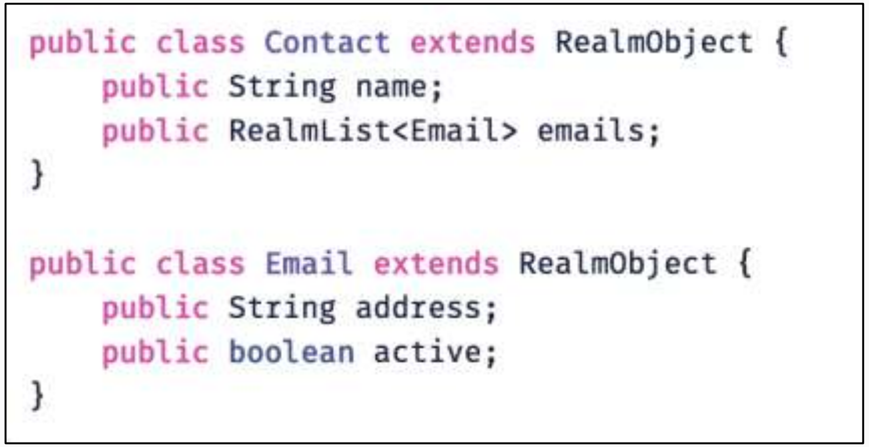
Relationships - Composition



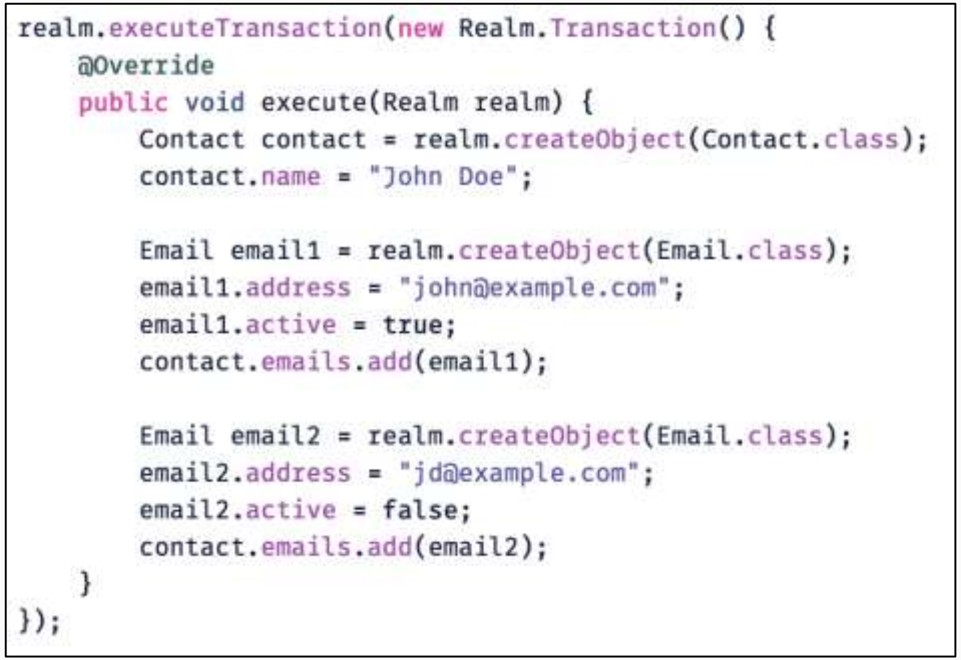
```java
public class Email extends RealmObject {
    private String address;
    private boolean active;
    // ... setters and getters left out
}

public class Contact extends RealmObject {
    private String name;
    private Email email;
    // ... setters and getters left out
}
```

Like this (as of version 1.0.0)

Relationships - Many to Many



```
public class Contact extends RealmObject {
    public String name;
    public RealmList<Email> emails;
}

public class Email extends RealmObject {
    public String address;
    public boolean active;
}
```



```
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        Contact contact = realm.createObject(Contact.class);
        contact.name = "John Doe";

        Email email1 = realm.createObject(Email.class);
        email1.address = "john@example.com";
        email1.active = true;
        contact.emails.add(email1);

        Email email2 = realm.createObject(Email.class);
        email2.address = "jd@example.com";
        email2.active = false;
        contact.emails.add(email2);
    }
});
```

Like this (as of version 1.0.0)

Writes

```
// Obtain a Realm instance
Realm realm = Realm.getDefaultInstance();

realm.beginTransaction();

//... add or update objects here ...

realm.commitTransaction();
```

```
realm.beginTransaction();
User user = realm.createObject(User.class);

//  ...

realm.cancelTransaction();
```

Like this (as of version 1.0.0)

Writes - Creating Objects

```
realm.beginTransaction();
User user = realm.createObject(User.class); // Create a new object
user.setName("John");
user.setEmail("john@corporation.com");
realm.commitTransaction();
```

```
User user = new User("John");
user.setEmail("john@corporation.com");

// Copy the object to Realm. Any further changes must happen on realmUser
realm.beginTransaction();
User realmUser = realm.copyToRealm(user);
realm.commitTransaction();
```

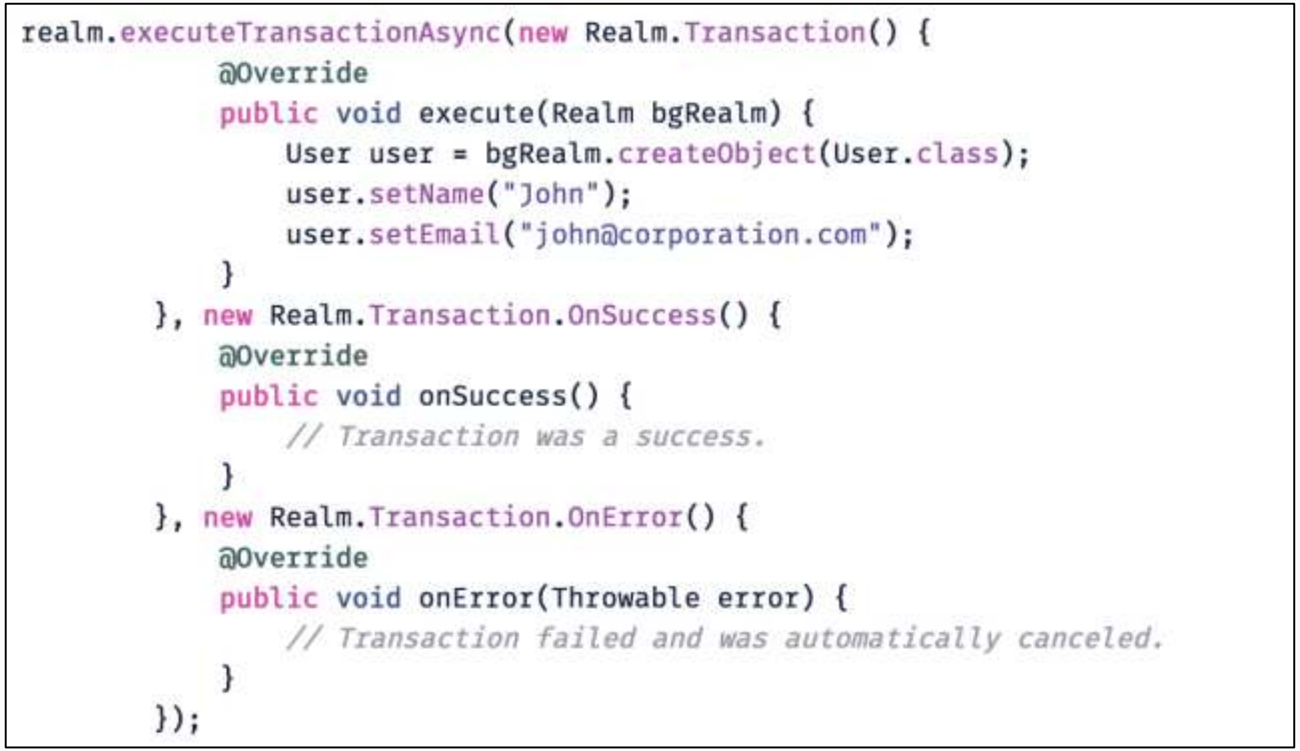
Like this (as of version 1.0.0)

Writes - Creating Objects

```
realm.executeTransaction(new Realm.Transaction() {
        @Override
        public void execute(Realm realm) {
                User user = realm.createObject(User.class);
                user.setName("John");
                user.setEmail("john@corporation.com");
        }
});
```

Like this (as of version 1.0.0)

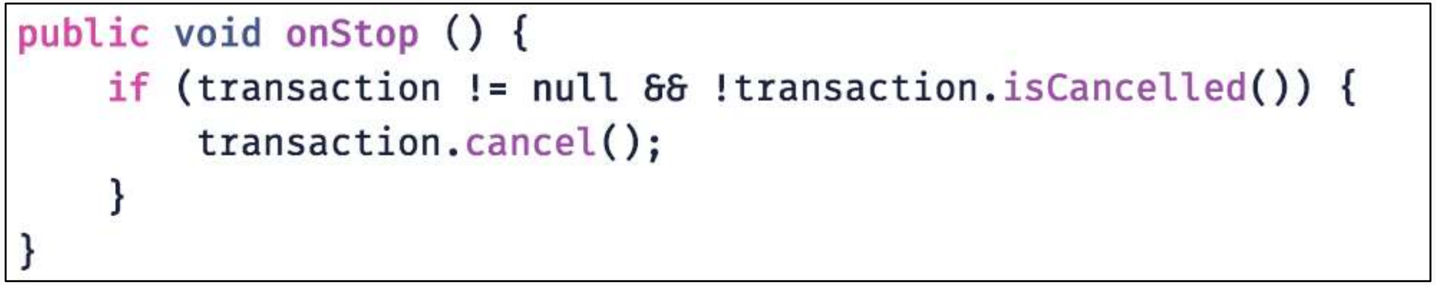
Writes - Creating Objects



```
realm.executeTransactionAsync(new Realm.Transaction() {
            @Override
            public void execute(Realm bgRealm) {
                User user = bgRealm.createObject(User.class);
                user.setName("John");
                user.setEmail("john@corporation.com");
            }
        }, new Realm.Transaction.OnSuccess() {
            @Override
            public void onSuccess() {
                // Transaction was a success.
            }
        }, new Realm.Transaction.OnError() {
            @Override
            public void onError(Throwable error) {
                // Transaction failed and was automatically canceled.
            }
        });
```

Like this (as of version 1.0.0)

Writes - Creating Objects



```java
public void onStop () {
    if (transaction != null && !transaction.isCancelled()) {
        transaction.cancel();
    }
}
```

Like this (as of version 1.0.0)

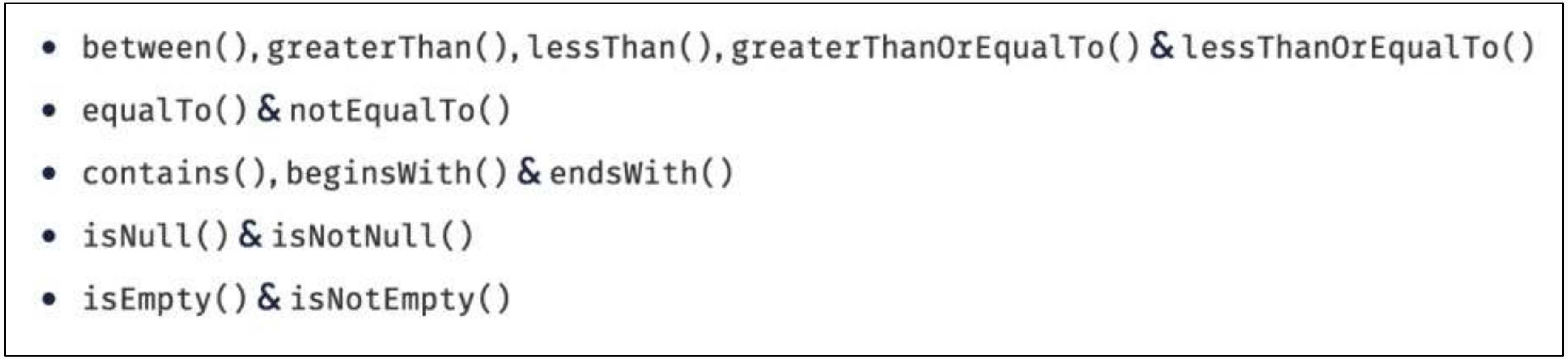
Queries

All fetches (including queries) are lazy in Realm, and the data is never copied.

```
// Build the query looking at all users:
RealmQuery<User> query = realm.where(User.class);

// Add query conditions:
query.equalTo("name", "John");
query.or().equalTo("name", "Peter");

// Execute the query:
RealmResults<User> result1 = query.findAll();

// Or alternatively do the same all at once (the "Fluent interface"):
RealmResults<User> result2 = realm.where(User.class)
                                  .equalTo("name", "John")
                                  .or()
                                  .equalTo("name", "Peter")
                                  .findAll();
```

Like this (as of version 1.0.0)

Queries - Conditions



- `between()`, `greaterThan()`, `lessThan()`, `greaterThanOrEqualTo()` & `lessThanOrEqualTo()`
- `equalTo()` & `notEqualTo()`
- `contains()`, `beginsWith()` & `endsWith()`
- `isNull()` & `isNotNull()`
- `isEmpty()` & `isNotEmpty()`

Like this (as of version 1.0.0)

Queries - Sorting

```
RealmResults<User> result = realm.where(User.class).findAll();
result = result.sort("age"); // Sort ascending
result = result.sort("age", Sort.DESCENDING);
```

```
RealmResults<User> result = realm.where(User.class).findAll();
result.sort("age"); // Sort ascending
result.sort("age", RealmResults.SORT_ORDER_DESCENDING);
```

Like this (as of version 1.0.0)

Queries - Chaining Queries

```
RealmResults<User> teenagers = realm.where(User.class).between("age", 13, 20).findAll();
User firstJohn = teenagers.where().equalTo("name", "John").findFirst();
```

Like this (as of version 1.0.0)

Queries - Async Queries

```
RealmResults<User> result = realm.where(User.class)
                                 .equalTo("name", "John")
                                 .or()
                                 .equalTo("name", "Peter")
                                 .findAllAsync();
```

```
private RealmChangeListener callback = new RealmChangeListener() {
    @Override
    public void onChange(RealmResults<User> results) {
        // called once the query complete and on every update
    }
};

public void onStart() {
    RealmResults<User> result = realm.where(User.class).findAllAsync();
    result.addChangeListener(callback);
}
```

Like this (as of version 1.0.0)

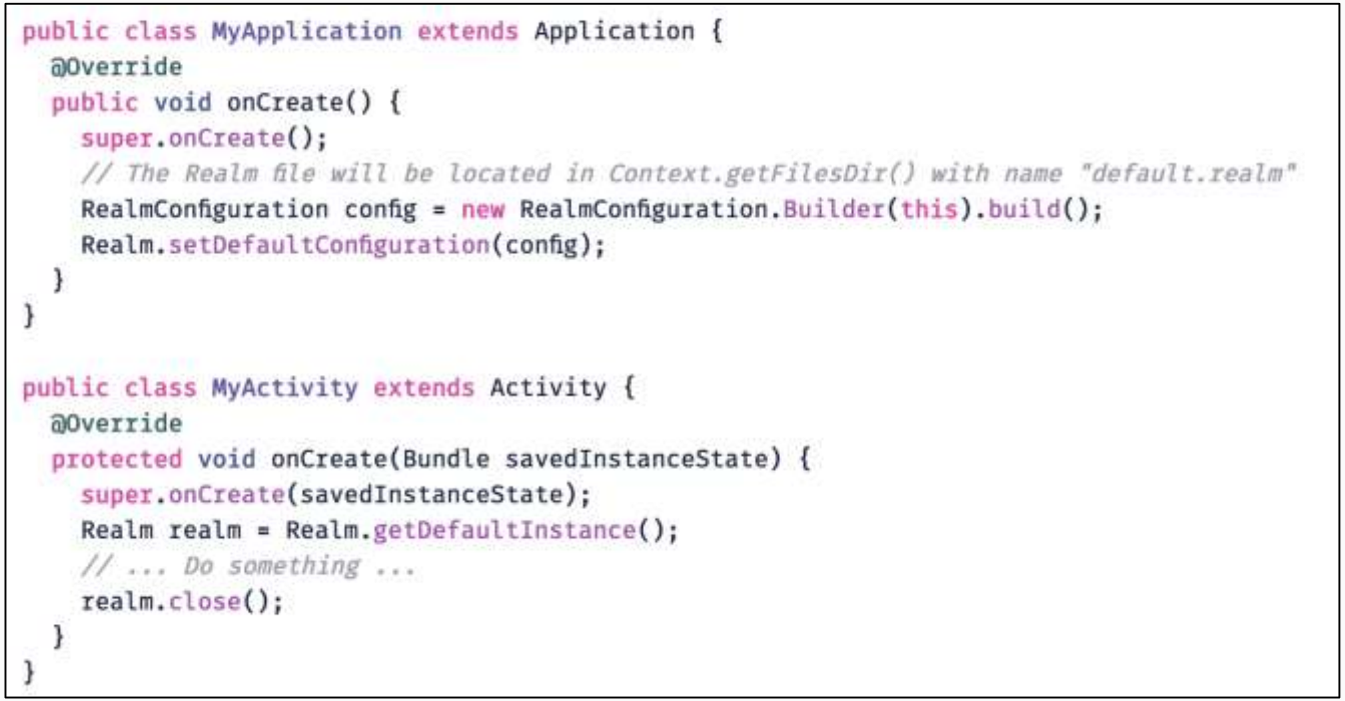
Queries - Async Queries

```
public void onStop () {
    result.removeChangeListener(callback); // remove a particular listener
    // or
    result.removeChangeListeners(); // remove all registered listeners
}
```

```
RealmResults<User> result = realm.where(User.class).findAllAsync();
if (result.isLoaded()) {
  // Results are now available
}
```

Like this (as of version 1.0.0)

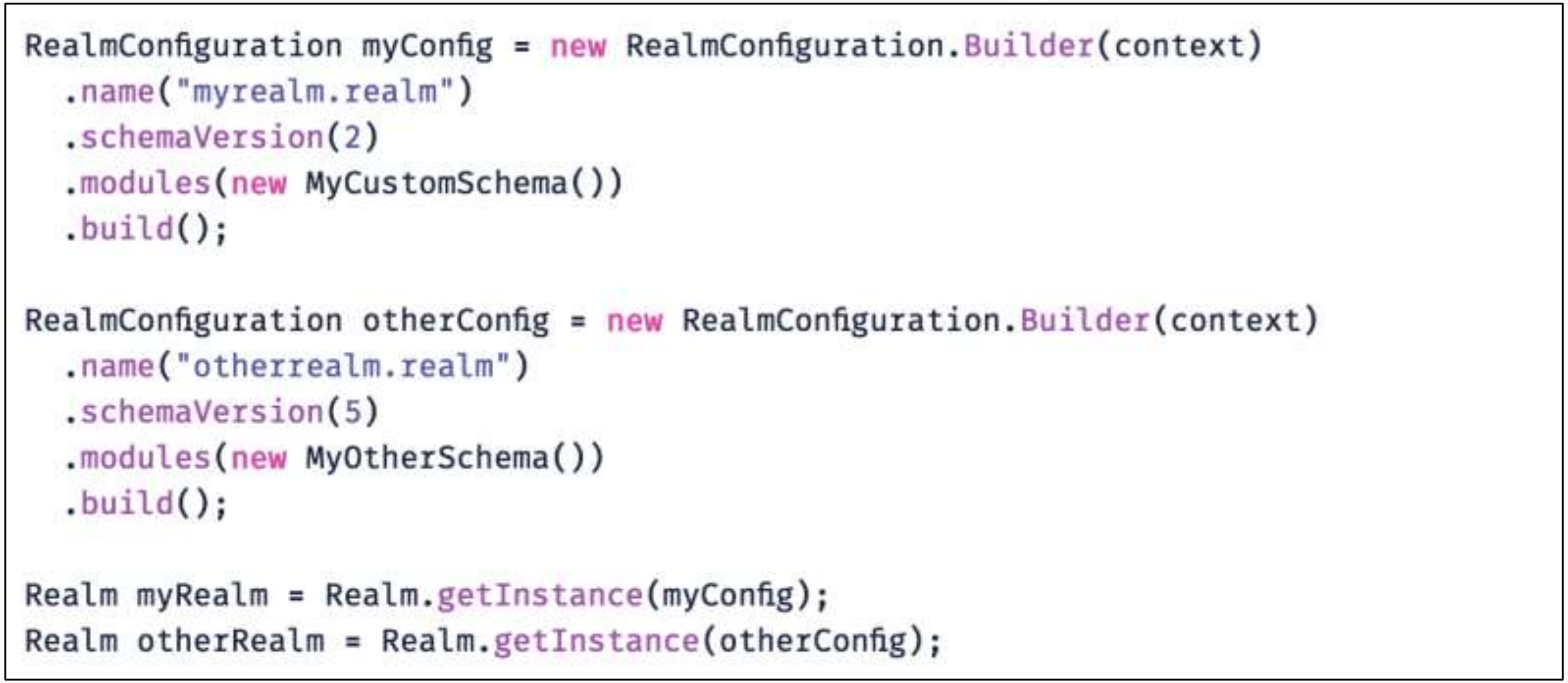
Realms - Default Configuration



```
public class MyApplication extends Application {
  @Override
  public void onCreate() {
    super.onCreate();
    // The Realm file will be located in Context.getFilesDir() with name "default.realm"
    RealmConfiguration config = new RealmConfiguration.Builder(this).build();
    Realm.setDefaultConfiguration(config);
  }
}

public class MyActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Realm realm = Realm.getDefaultInstance();
    // ... Do something ...
    realm.close();
  }
}
```

Like this (as of version 1.0.0)

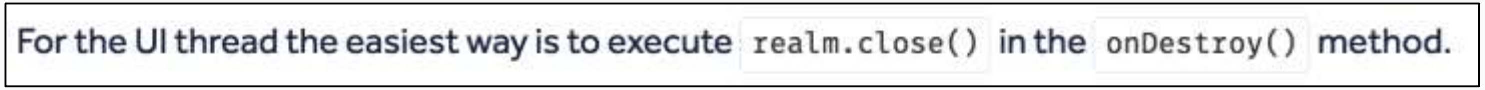
Realms - Multiple Configuration

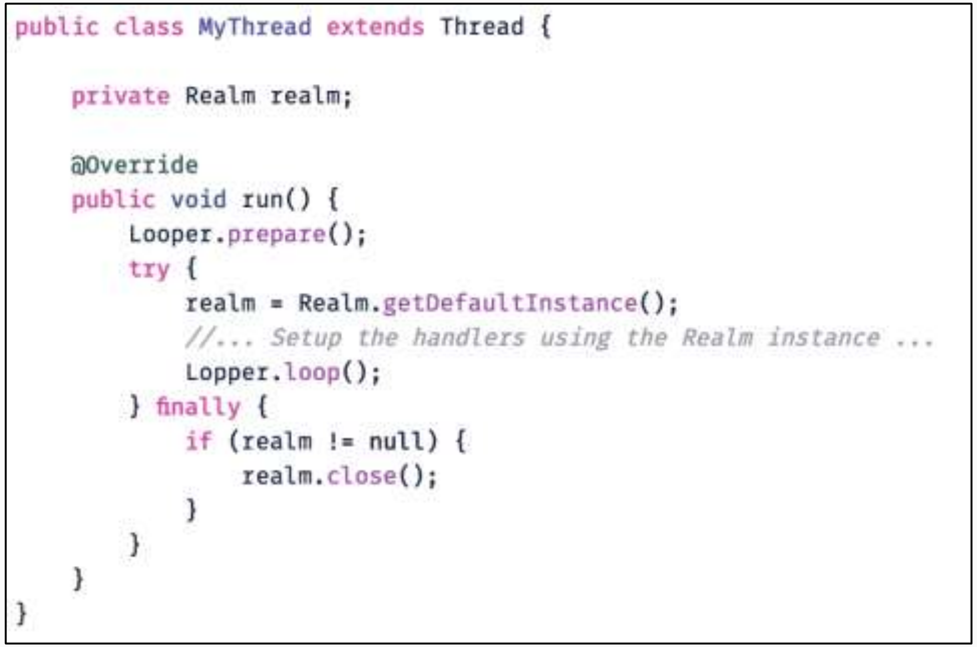


```
RealmConfiguration myConfig = new RealmConfiguration.Builder(context)
  .name("myrealm.realm")
  .schemaVersion(2)
  .modules(new MyCustomSchema())
  .build();

RealmConfiguration otherConfig = new RealmConfiguration.Builder(context)
  .name("otherrealm.realm")
  .schemaVersion(5)
  .modules(new MyOtherSchema())
  .build();

Realm myRealm = Realm.getInstance(myConfig);
Realm otherRealm = Realm.getInstance(otherConfig);
```

Like this (as of version 1.0.0)

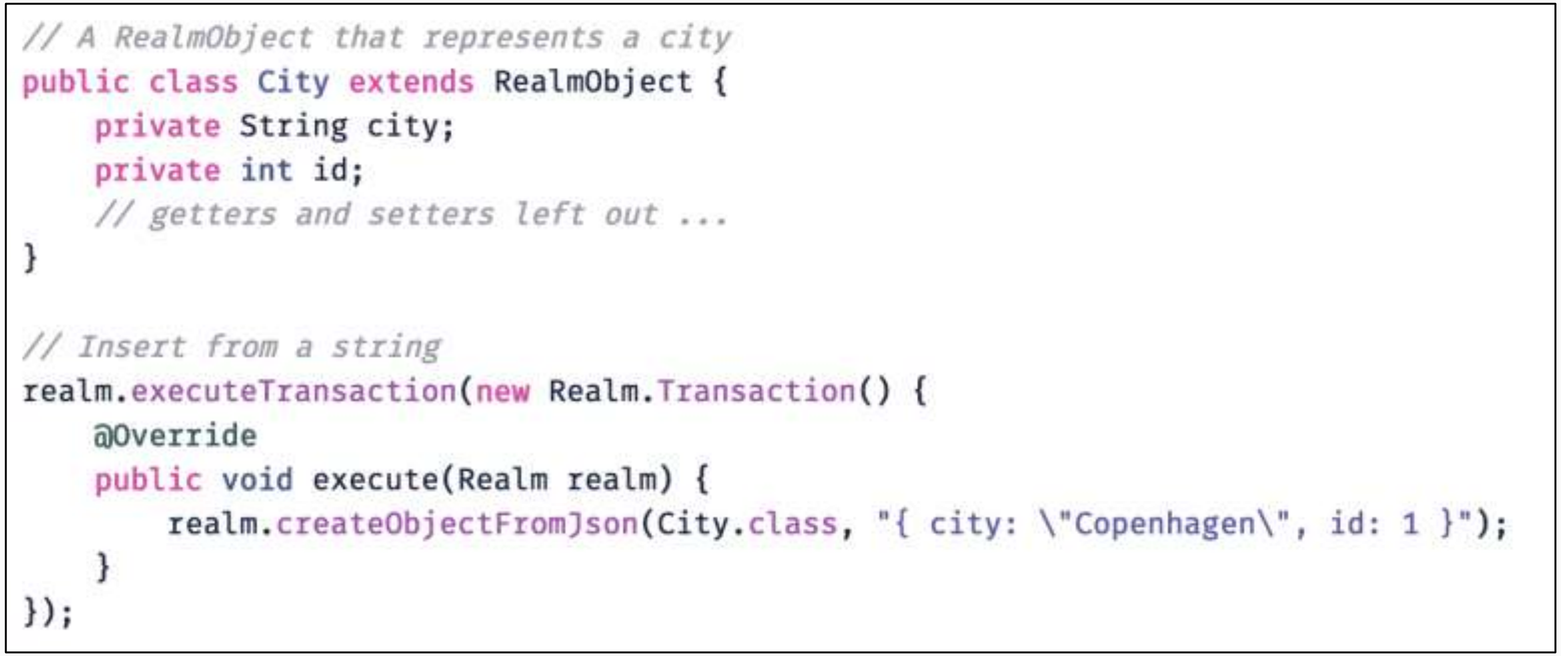
Realms - Closing Instances


For the UI thread the easiest way is to execute `realm.close()` in the `onDestroy()` method.


```
public class MyThread extends Thread {

    private Realm realm;

    @Override
    public void run() {
        Looper.prepare();
        try {
            realm = Realm.getDefaultInstance();
            //... Setup the handlers using the Realm instance ...
            Lopper.loop();
        } finally {
            if (realm != null) {
                realm.close();
            }
        }
    }
}
```

Like this (as of version 1.0.0)

JSON



```java
// A RealmObject that represents a city
public class City extends RealmObject {
    private String city;
    private int id;
    // getters and setters left out ...
}

// Insert from a string
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        realm.createObjectFromJson(City.class, "{ city: \"Copenhagen\", id: 1 }");
    }
});
```

Like this (as of version 1.0.0)

Android - Intents

```
// Assuming we had a person class with a @PrimaryKey on the 'id' field ...
Intent intent = new Intent(getActivity(), ReceivingService.class);
intent.putExtra("person_id", person.getId());
getActivity().startService(intent);
```

```
// in onCreate(), onHandleIntent(), etc.
String personId = intent.getStringExtra("person_id");
Realm realm = Realm.getDefaultInstance();
Person person = realm.where(Person.class).equalTo("id", personId).findFirst();
// do something with the person ...
realm.close();
```

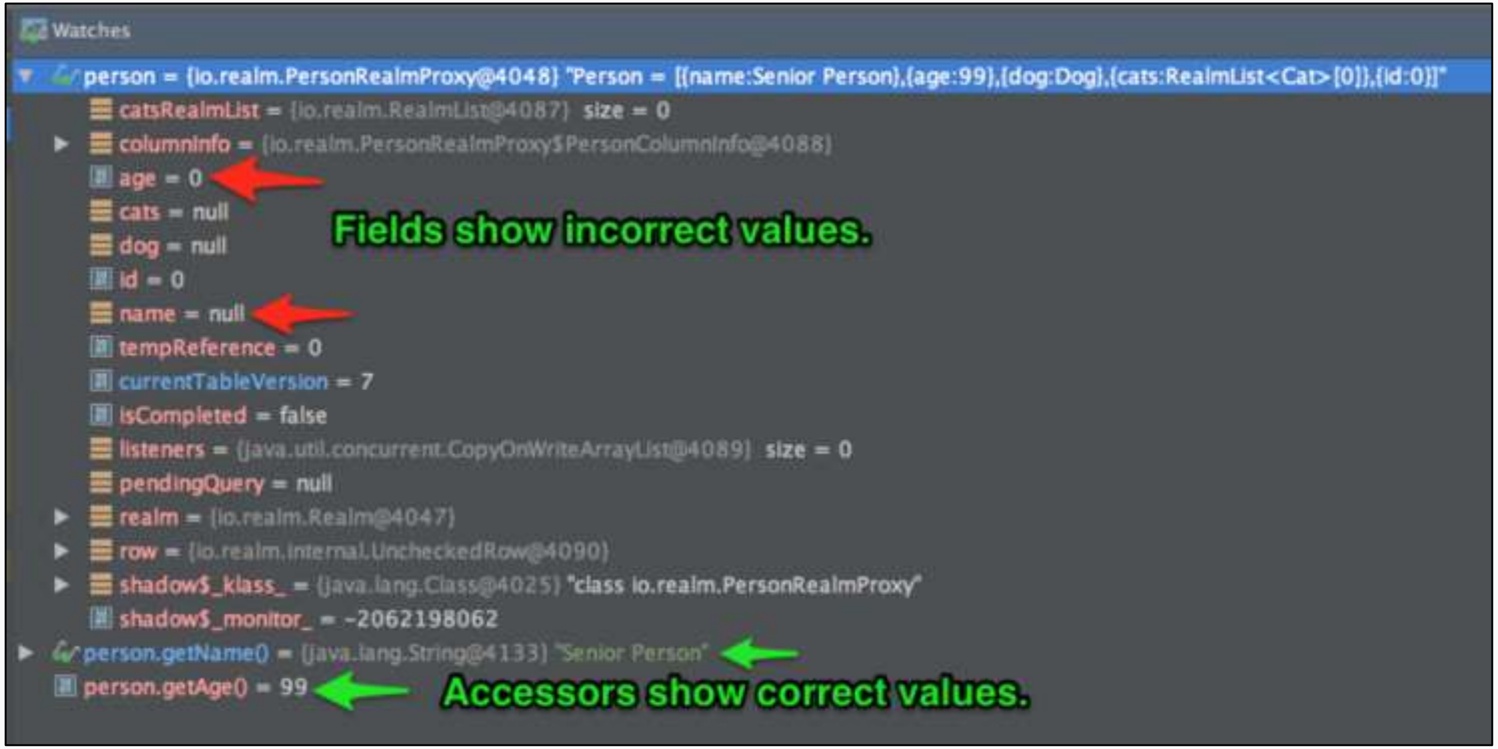
Like this (as of version 1.0.0)

Android - Async Task

```java
private class DownloadOrders extends AsyncTask<Void, Void, Long> {
    protected Long doInBackground(Void... voids) {
        // Now in a background thread.

        // Open the Realm
        Realm realm = Realm.getDefaultInstance();
        // Work with Realm
        realm.createAllFromJson(Order.class, api.getNewOrders());
        Order firstOrder = realm.where(Order.class).findFirst();
        long orderId = firstOrder.getId(); // Id of order
        realm.close();
        return orderId;
    }

    protected void onPostExecute(Long orderId) {
        // Back on the Android mainThread
        // do something with orderId such as query Realm
        // for the order and perform some operation with it.
    }
}
```

Like this (as of version 1.0.0)

Android - Debugging



```
Watches
▼ person = {io.realm.PersonRealmProxy@4048} "Person = [{name:Senior Person},{age:99},{dog:Dog},{cats:RealmList<Cat>[0]},{id:0}]"
      catsRealmList = {io.realm.RealmList@4087} size = 0
    ▶ columnInfo = {io.realm.PersonRealmProxy$PersonColumnInfo@4088}
      age = 0
      cats = null
      dog = null
      id = 0
      name = null
      tempReference = 0
      currentTableVersion = 7
      isCompleted = false
      listeners = {java.util.concurrent.CopyOnWriteArrayList@4089} size = 0
      pendingQuery = null
    ▶ realm = {io.realm.Realm@4047}
    ▶ row = {io.realm.internal.UncheckedRow@4090}
    ▶ shadow$_klass_ = {java.lang.Class@4025} "class io.realm.PersonRealmProxy"
      shadow$_monitor_ = -2062198062
▶ person.getName() = {java.lang.String@4133} "Senior Person"
  person.getAge() = 99
```

Fields show incorrect values.

Accessors show correct values.

Final Notes

perform all Realm write operations on a background thread (not Android's main thread)

RealmObjects and **RealmResults** access all the data they refer to lazily. For this reason it is important to keep the Realm instance open for as long as you want to access your Realm objects or query results

References

http://kpgalligan.tumblr.com/post/133281929963/my-talk-at-droidcon-uk

http://realm.io

https://gist.github.com/cmelchior/1a97377df0c49cd4fca9