How to Setup ORMLite to your Project

1. Setup ORMLite - you have to change gradle file and add following lines to your dependencies

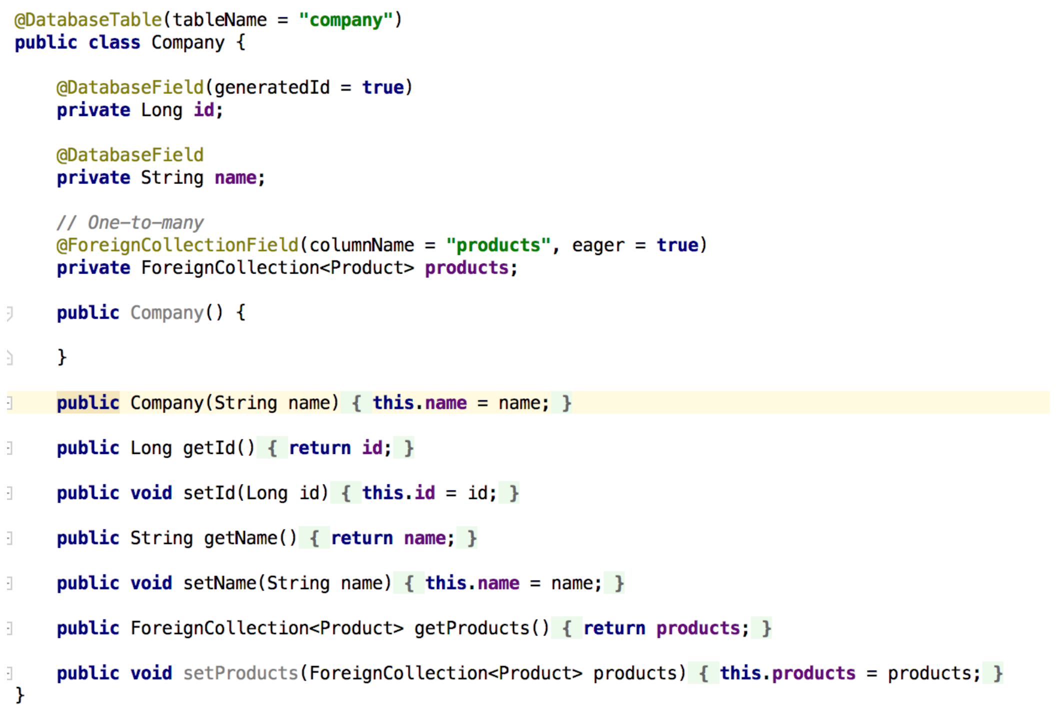
```
compile 'com.j256.ormlite:ormlite-core:4.48'
compile 'com.j256.ormlite:ormlite-android:4.48'
```

2. Sync your Gradle.

3. Create Model Class for Mapping to Database
   For example, we are creating a “Company” and “Product” in our Project.

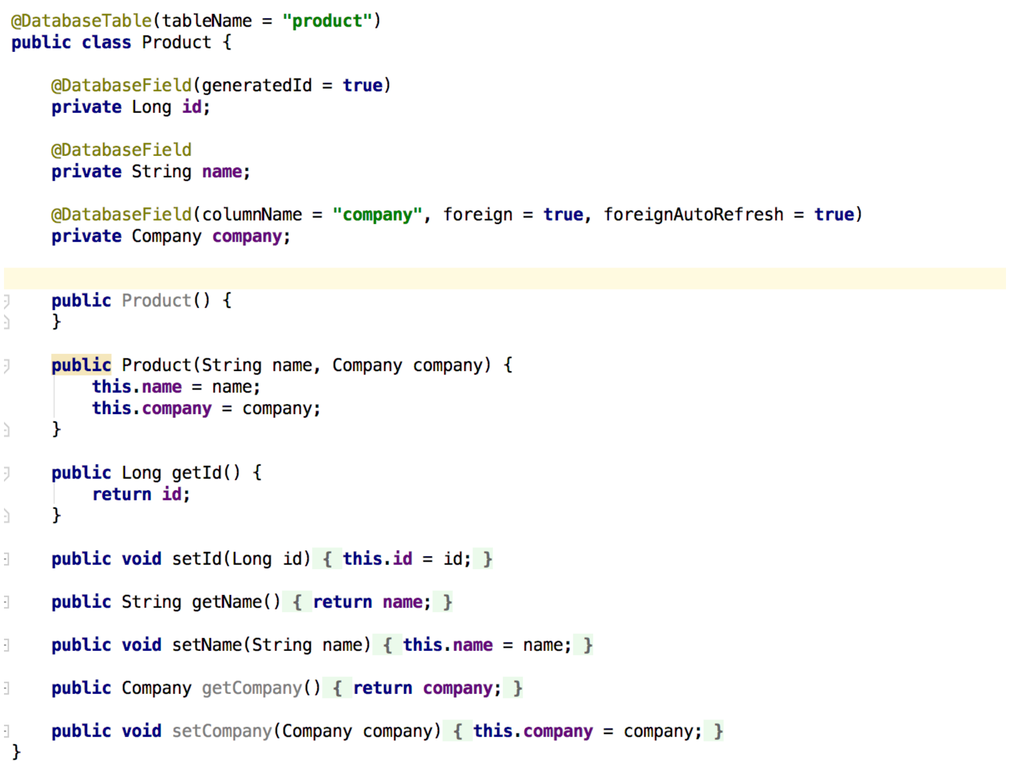
4. In ORMLite using annotation we can map our Model class to the DB.

Company Model -



```
@DatabaseTable(tableName = "company")
public class Company {

    @DatabaseField(generatedId = true)
    private Long id;

    @DatabaseField
    private String name;

    // One-to-many
    @ForeignCollectionField(columnName = "products", eager = true)
    private ForeignCollection<Product> products;

    public Company() {

    }

    public Company(String name) { this.name = name; }

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public ForeignCollection<Product> getProducts() { return products; }

    public void setProducts(ForeignCollection<Product> products) { this.products = products; }
}
```

Product Model -

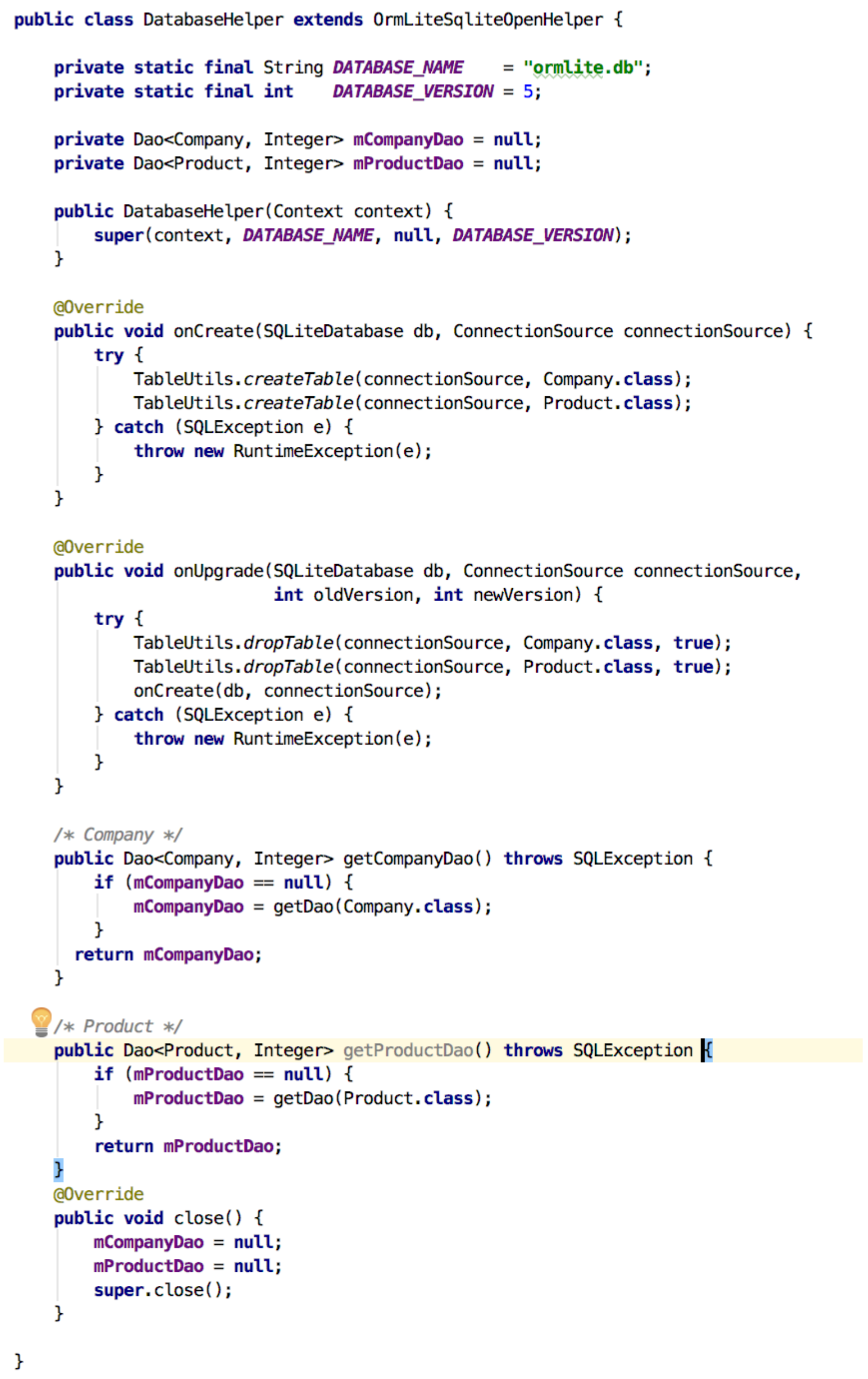


```java
@DatabaseTable(tableName = "product")
public class Product {

    @DatabaseField(generatedId = true)
    private Long id;

    @DatabaseField
    private String name;

    @DatabaseField(columnName = "company", foreign = true, foreignAutoRefresh = true)
    private Company company;


    public Product() {
    }

    public Product(String name, Company company) {
        this.name = name;
        this.company = company;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public Company getCompany() { return company; }

    public void setCompany(Company company) { this.company = company; }
}
```

a. Using *@DatabaseTable* We map our model to Database table name.
b. Auto generated primary key - *@DatabaseField(generatedId = true)*
c. For Other field just use *@DatabaseField*
d. Company produce more than one products. Using, *@ForeignCollectionField* we can specify that one-to-many relationship. (See the Company Class Above)
e. In Product class company id is Foreign key. Using following annotation, we specify foreign key.(See the Product class above)
@DatabaseField(columnName = "company", foreign = true, foreignAutoRefresh = true)

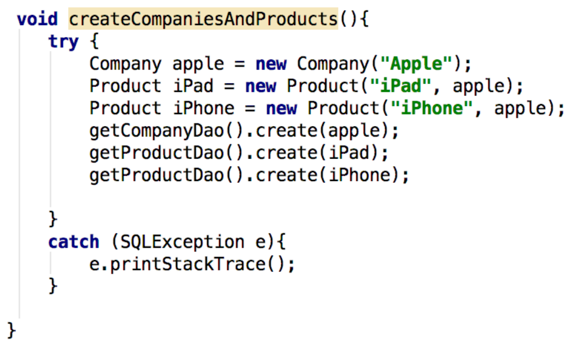
5. We need an another class which is responsible for the complete logic of database file creation, accessibility etc.
Key Points of this Class –
    a. Inherited from *OrmLiteSqliteOpenHelper* Class.
    b. Here we generally specify Database Name & version
    c. *onCreate()* method should include all the table creation statements and other first time configuration logics. onCreate() method executes only once i.e. when the application is running for the first time
    d. For update in DB we need *onUpgrade()* method

e. DAO: DAOs are the one of the most important components in ORMLite as those are the only handle to access database tables. So, each and every table should have a DAO, so application can access this table when required.



```
public class DatabaseHelper extends OrmLiteSqliteOpenHelper {

    private static final String DATABASE_NAME    = "ormlite.db";
    private static final int    DATABASE_VERSION = 5;

    private Dao<Company, Integer> mCompanyDao = null;
    private Dao<Product, Integer> mProductDao = null;

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db, ConnectionSource connectionSource) {
        try {
            TableUtils.createTable(connectionSource, Company.class);
            TableUtils.createTable(connectionSource, Product.class);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, ConnectionSource connectionSource,
                          int oldVersion, int newVersion) {
        try {
            TableUtils.dropTable(connectionSource, Company.class, true);
            TableUtils.dropTable(connectionSource, Product.class, true);
            onCreate(db, connectionSource);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }

    /* Company */
    public Dao<Company, Integer> getCompanyDao() throws SQLException {
        if (mCompanyDao == null) {
            mCompanyDao = getDao(Company.class);
        }
      return mCompanyDao;
    }

    /* Product */
    public Dao<Product, Integer> getProductDao() throws SQLException {
        if (mProductDao == null) {
            mProductDao = getDao(Product.class);
        }
        return mProductDao;
    }
    @Override
    public void close() {
        mCompanyDao = null;
        mProductDao = null;
        super.close();
    }

}
```

6. The following code snippet describes how to insert data into the Database -



```java
void createCompaniesAndProducts(){
    try {
        Company apple = new Company("Apple");
        Product iPad = new Product("iPad", apple);
        Product iPhone = new Product("iPhone", apple);
        getCompanyDao().create(apple);
        getProductDao().create(iPad);
        getProductDao().create(iPhone);

    }
    catch (SQLException e){
        e.printStackTrace();
    }

}
```