Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерный наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий.

**Отчёт по лабораторной работе №4**

**Дисциплина**: Проектирование мобильных приложений

**Тема**: RecyclerView. Вариант 13

Выполнил студент гр. 3530901/80201: ­­­­­­­­­­­­­ ­ В.Д. Петров

(подпись)

Преподаватель: ­А.Н. Кузнецов

(подпись)

Санкт-Петербург

2020

**СОДЕРЖАНИЕ**

Введение…………………………………………………………………………….3

Задача №1. Знакомство с библиотекой (unit test)…………………………………4

Задача №2. Знакомство с Recycler View…………………………………………..8

Задача №3. Бесконечный список……………………………………………..…..13

Заключение………………………………………………………………………..17

Список использованных источников…………………………………………....18

**ВВЕДЕНИЕ**

Целями данной работы является ознакомление с принципами работы adapter-based views и получение практических навыков разработки адаптеров для view. RecyclerView облегчает отображение больших сетов данных. Вы предоставляете данные и указываете, как будет выглядеть каждый элемент, а библиотека сама создает элементы, когда они необходимы. Каждый элемент в списке определяется объектом view holder, RecyclerView запрашивает эти view и связывает их с данными с помощью методов из адаптера, который определяется, наследуясь от RecyclerView.Adapter. Layout manager упорядочивает элементы в списке. При определении адаптера нужно переопределить 3 ключевых метода:

onCreateViewHolder – вызывается, когда RecyclerView нужно создать новый объект ViewHolder. В данном методе ViewHolder все еще пуст и не связан с данными.

onBindViewHolder – вызывается, когда RecyclerView нужно связать данные с объектом ViewHolder.

getItemCount – вызывается для того, чтобы RecyclerView знал, когда список должен будет закончиться.

**ЗАДАЧА №1. ЗНАКОМСТВО С БИБЛИОТЕКОЙ (UNIT TEST)**

По заданию необходимо ознакомиться с предложенной библиотекой, предоставляющей программный доступ к записям в формате bibtex. У этой библиотеки имеется два мода – strict и normal, в strict моде присутствует искусственное ограничение по количеству хранимых записей – при превышении данного ограничения 1-я запись становится невалидной. В данной библиотеке присутствует 4 класса и 2 перечисления.

Первым рассмотрим класс BibConfig – в нем находится 3 поля: флаг shuffle для перемешивания базы данных, флаг strict для установки strict мода работы и поле maxValid для установки количества валидных записей.

Затем рассмотрим класс BibEntry – в нем находится 4 поля: latestOrder для отслеживания общего числа записей, при создании нового объекта BibEntry данное поле увеличивает свое значение на 1, myOrder – порядковый номер для текущего объекта BibEntry, который она получает, изменяя значение latestOrder, delegate – объект класса BibTeXEntry, cfg – объект класса BibConfig. В нем присутствует 3 метода. Метод assertDelegateIsValid проверяет, если у поля cfg внутреннее поле strict равно true, то берется значение поля latestOrder и проверяется со значением нынешней записи – если общее число записей окажется больше, чем 20, то у нынешней записи будет брошено исключение IllegalStateException. Метод getType вызывает метод assertDelegateIsValid, после чего просматривает тип записи у объекта delegate и возвращает его. Метод getField вызывает метод assertDelegateIsValid и просматривает поле объекта delegate по заданному ключу, если оно не равно null, то возвращает данное поле в виде строки, если оно равно null, то null и возвращается.

Затем рассмотрим перечисление Types. В нем перечисляются различные форматы записей, соответствующие форматам из библиотеки BibTeX, после чего создается HashMap с ключом в виде формата BibTeX, по которому выдается соответствующее ему значение Types. Таким образом при обращении к delegate внутри класса BibEntry, можно будет привести формат записи класса BibTeXEntry к более удобному формату Types.

Затем рассмотрим перечисление Keys. В нем перечисляются поля у каждой записи – автор, название книги, название публикующей организации и т.д.

Затем рассмотри класс BibDatabase. В нем находится 2 поля – entries для списка из записей класса BibTeXEntry и cfg для объекта класса BibConfig. Без указания в конструкторе объекта для BibConfig, то будет создан новый, у которого флаг shuffle будет равен true, а флаг strict будет равен false. После этого будет вызван конструктор, в котором с помощью BibTeXParser из объекта класса Reader из некоторого файла читаются записи, они заполняют объект класса BibTeXDatabase, после чего из данного объекта записи перемещаются в поле entries и с помощью метода shuffle у класса Collections записи перемешиваются, если был установлен флаг shuffle в значение true. У данного класса есть 2 метода – getEntry, возвращающая новый объект класса BibEntry со значением, взятым из поля entries по указанному индексу и значением cfg, в также getCfg, возвращающая объект поля cfg.

Наконец, рассмотрим класс BibDatabaseTest – это класс для unit тестов написанной библиотеки. В нем есть только одно поле database для объекта класса BibDatabase. До начала теста вызывается метод setup, в котором из файла references.bib читается содержимое и на его основе создается новый объект database. В первом тесте getFirstEntry берется элемент из database по индексу 0, после чего его содержимое по ключам сравнивается с эталонным, а также сравнивается тип записи – Article. Во втором тесте normalModeDoesNotThrowException конфиг database изменяется флагом strict равным false, после чего 21 раз берется запись по индексу 0 в database – при работе в strict моде должна была произойти ошибка, но в нормальном моде такого не происходит. Последние два теста не написаны – их нужно сделать.

В тесте strictModeThrowsException было сделано следующее – в database поле strict было изменено на true, был взят элемент по индексу 0 и присвоен переменной first, после чего тот же самый элемент был взять 20 раз – в результате при вызове у переменной first метода getType будет выкинута ошибка IllegalStateException, которую мы будем отлавливать с помощью блока try-catch, а также выдано сообщение, содержание которого мы будем сравнивать с эталонным. В результате теста ошибка действительно была выкинута, сообщение полностью соответствует эталонному.

Листинг 1 – тест strictModeThrowsException

@Test  
public void strictModeThrowsException() {  
 database.getCfg().strict = true;  
 BibEntry first = database.getEntry(0);  
 for (int i = 0; i < database.getCfg().maxValid; i++) {  
 database.getEntry(0);  
 }  
 try {  
 first.getType();  
 } catch (IllegalStateException e) {  
 Assert.*assertEquals*("This object has already been invalidated. myOrder=1, latestOrder=21", e.getMessage());  
 }  
}

В тесте shuffleFlag было сделано следующее – был создан файл с 2 записями, одна из которых идентична записи в references.bib, после чего на основе данного файла была создана база данных another, у которой по умолчанию флаг shuffle будет равен true. У старой базы данных и новой сравниваются 0 элементы, если они равны, то вторая база данных снова создается с теми же параметрами и так до тех пор, пока вторая база данных не будет отличаться от первой. После этого смотрим элемент по индексу 1 у второй базы данных, он должен будет соответствовать элементу по индексу 0 у первой базы данных и проверяем что это так. По некоторой причине данный тест иногда выбрасывает ошибку IndexOutOfBoundsException, которая говорит о том, что объекту reader стал недоступен для чтения файл references1.bib, поэтому вместо пересоздания объекта another был дополнен класс BibDatabase методом shuffle, который повторяет аналогичную процедуру из конструктора класса – если флаг shuffle равен true, то тогда вызывается соответствующий метод класса Collections.

Листинг 2 – метод shuffle

public void shuffle(){  
 if (cfg.shuffle) {  
 Collections.*shuffle*(entries);  
 }  
}

В результате тест перестал падать с ошибкой и успешно проходит свое исполнение каждый раз.

Листинг 3 – тест shuffleFlag

@Test  
public void shuffleFlag() {  
 try (InputStreamReader reader = new InputStreamReader(getClass().getResourceAsStream("/references1.bib"))) {  
 BibDatabase another = new BibDatabase(reader);  
 String databaseJournal = database.getEntry(0).getField(Keys.*JOURNAL*);  
 String anotherJournal = another.getEntry(0).getField(Keys.*JOURNAL*);  
 while (databaseJournal.equals(anotherJournal)) {  
 another.shuffle();  
 anotherJournal = another.getEntry(0).getField(Keys.*JOURNAL*);  
 }  
 Assert.*assertEquals*(database.getEntry(0).getField(Keys.*JOURNAL*), another.getEntry(1).getField(Keys.*JOURNAL*));  
 }catch(IOException e){  
 *fail*();  
 }  
 }

Наконец, командой gradlew build был собран .jar файл по пути build/libs/biblib.jar, билд был собран без ошибок (Рис. 1).

Изображение выглядит как текст

Автоматически созданное описание

Рис. 1 Результаты сборки

**ЗАДАЧА №2. ЗНАКОМСТВО С RECYCLER VIEW**

Исходный код для данного и следующего заданий представлен по ссылке:

<https://github.com/ADsty/labs-android_labs/tree/master/lab4>

Необходимо написать Android приложение, которое выводит все записи из bibtex файла на экран, используя предложенную библиотеку и RecyclerView. Выбранная задача – однородный список. Были скачаны файлы с исходными данными и подключена библиотека как зависимость на прекомпилированный jar файл. Для представления одного элемента был написан класс Item, в котором есть несколько полей, выбранных мной – author, title, journal, year, volume, number, pages. Данные поля будут отображаться в RecyclerView, они были выбраны как наиболее значимые из представленных в записях bibtex формата.

Листинг 4 – класс Item

class Item {  
 var author: String = ""  
 var title: String = ""  
 var journal: String = ""  
 var year: String = ""  
 var volume: String = ""  
 var number: String = ""  
 var pages: String = ""  
  
 fun changeAuthor(text: String) {  
 this.author = text  
 }  
  
 fun changeTitle(text: String) {  
 this.title = text  
 }  
  
 fun changeJournal(text: String) {  
 this.journal = text  
 }  
  
 fun changeYear(text: String) {  
 this.year = text  
 }  
  
 fun changeVolume(text: String) {  
 this.volume = text  
 }  
  
 fun changeNumber(text: String) {  
 this.number = text  
 }  
  
 fun changePages(text: String) {  
 this.pages = text  
 }  
  
 override fun toString(): String = author  
}

Для получения данных для этих полей из файла в директории raw был написан объект GetData, которому на вход будет подаваться сам файл, а далее с помощью предложенной библиотеки создается экземпляр класса BibDatabase, а уже из него в цикле достаются все необходимые данные и присваиваются соответствующим полям экземпляра класса Item.

Листинг 5 – объект GetData

object GetData {  
  
 val items: MutableList<Item> = ArrayList()  
 private var database: BibDatabase? = null  
  
 private fun addItems() {  
 for (i in 0 *until* 5) {  
  
 val item = Item()  
  
 item.changeAuthor(database!!.getEntry(i).getField(Keys.AUTHOR) ?: "empty")  
 item.changeTitle(database!!.getEntry(i).getField(Keys.TITLE) ?: "empty")  
 item.changeJournal(database!!.getEntry(i).getField(Keys.JOURNAL) ?: "empty")  
 item.changeYear(database!!.getEntry(i).getField(Keys.YEAR) ?: "empty")  
 item.changeVolume(database!!.getEntry(i).getField(Keys.VOLUME) ?: "empty")  
 item.changeNumber(database!!.getEntry(i).getField(Keys.NUMBER) ?: "empty")  
 item.changePages(database!!.getEntry(i).getField(Keys.PAGES) ?: "empty")  
  
 items.add(item)  
 }  
 }  
  
 fun setResources(inputStream: InputStream) {  
 val config = BibConfig().*apply* **{** shuffle = false  
 strict = false  
 **}** InputStreamReader(inputStream).*use* **{** database = BibDatabase(**it**, config)  
 **}** addItems()  
 inputStream.close()  
 }  
}

Для упрощения проверки бесконечности списка из следующего задания, цикл ограничен 5 элементами. Для однородного списка был создан фрагмент OnceFragment, который использует RecyclerView c LinearLayoutManager. Каждая запись отображается с помощью виджета CardView, в котором находятся виджеты TextView под каждое из выбранных полей. В OnceFragment устанавливается LayoutManager для RecyclerView, а также адаптер – OnceRecyclerViewAdapter.

Листинг 6 – класс OnceFragment

class OnceFragment : Fragment() {  
  
 override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {  
 val view = inflater.inflate(R.layout.*fragment\_item\_list*, container, false)  
  
 if (view is RecyclerView) {  
 *with*(view) **{** *layoutManager* = LinearLayoutManager(*context*)  
 *adapter* = OnceRecyclerViewAdapter(GetData.items)  
 **}** }  
 return view  
 }  
  
}

Данный класс наследуется от RecyclerView.Adapter, ему передаются значения объекта GetData, внутри себя он определяет класс ViewHolder, а также методы onCreateViewHolder – возвращает View Holder с CardView для отображения записи, onBindViewHolder – изменяет текст внутри TextView у ViewHolder и getItemCount – возвращает размер списка передаваемых GetData записей.

Листинг 7 – класс OnceRecyclerViewAdapter

class OnceRecyclerViewAdapter(private val items: List<Item>) :  
 RecyclerView.Adapter<OnceRecyclerViewAdapter.ViewHolder>() {  
  
 private val mOnClickListener: View.OnClickListener  
  
 init {  
 mOnClickListener = View.OnClickListener **{** v **->** val item = v.*tag* as Item  
 Log.d("onClick (once)", item.author + " " + item.title + " " + item.journal)  
 **}** }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 val view = LayoutInflater.from(parent.*context*).inflate(R.layout.*item*, parent, false)  
 return ViewHolder(view)  
 }  
  
 @SuppressLint("SetTextI18n")  
 override fun onBindViewHolder(holder: ViewHolder, position: Int) {  
 val item = items[position]  
 holder.author.*text* = "author = ${item.author}"  
 holder.title.*text* = "title = ${item.title}"  
 holder.journal.*text* = "journal = ${item.journal}"  
 holder.year.*text* = "year = ${item.year}"  
 holder.volume.*text* = "volume = ${item.volume}"  
 holder.number.*text* = "number = ${item.number}"  
 holder.pages.*text* = "pages = ${item.pages}"  
  
  
 *with*(holder.view) **{** *tag* = item  
 setOnClickListener(mOnClickListener)  
 **}** }  
  
 override fun getItemCount(): Int {  
 Log.d("getItemCount (once)", items.size.toString())  
 return items.size  
 }  
  
 inner class ViewHolder(val view: View) : RecyclerView.ViewHolder(view) {  
 val author: TextView = view.author  
 val title: TextView = view.title  
 val journal: TextView = view.journal  
 val year: TextView = view.year  
 val volume: TextView = view.volume  
 val number: TextView = view.number  
 val pages: TextView = view.pages  
 }  
}

Для переключения между однородным списком и бесконечным используется шторка Navigation Drawer. В результате, после запуска приложения все записи появляются на экране в виде виджетов CardView (Рис. 2).

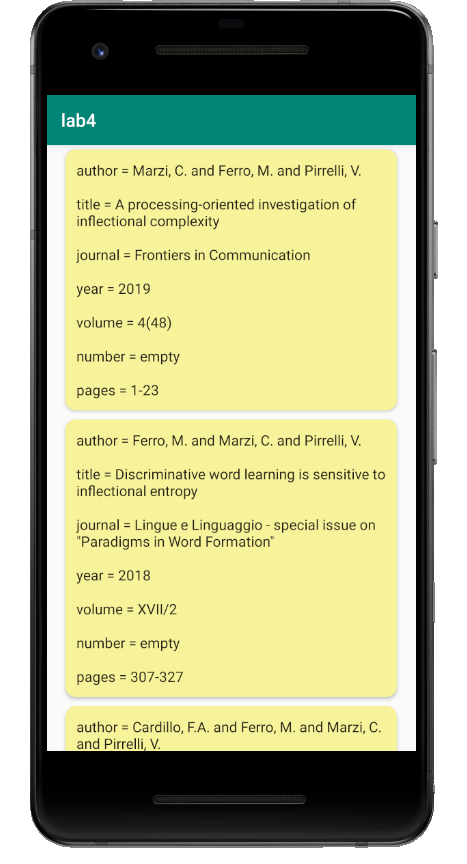


Рис. 2 Отображение записей из файла на экране

После отображения последней записи на экране (для удобства проверки их всего 5), прокрутка вниз останавливается (Рис. 3).

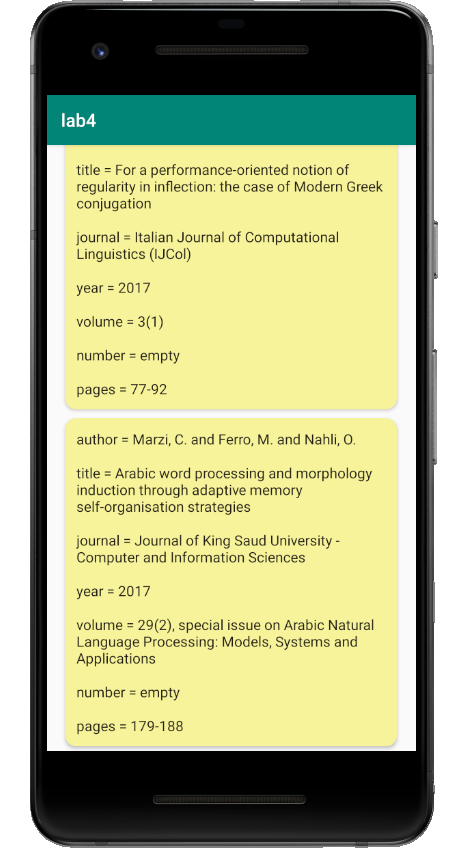


Рис. 3 Последняя запись в списке

**ЗАДАЧА №3. БЕСКОНЕЧНЫЙ СПИСОК**

В данном задании необходимо сделать список бесконечным – чтобы после отображения последней записи список начинался по новом с первой записи. Для этого был создан фрагмент InfiniteFragment, который во многом идентичен фрагменту OnceFragment за одним исключением – в нем для RecyclerView добавляется onScrollListener.

Листинг 8 – изменения в коде для InfiniteFragment

view.addOnScrollListener(object : RecyclerView.OnScrollListener() {  
 override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {  
 if (dy > 0) {  
 totalItemCount = mLayoutManager.*itemCount* pastVisiblesItems = mLayoutManager.findLastCompletelyVisibleItemPosition()  
 if (pastVisiblesItems == totalItemCount - 1) {  
 mLayoutManager.scrollToPosition(0)  
 }  
 }  
 }  
})

Теперь всякий раз, когда пользователь мотает приложение вниз (условие dy > 0), приложение просматривает номер последнего элемента, который полностью виден на экране, если этот номер равен номеру последнего элемента в списке, то LayoutManager методом scrollToPosition перемещает экран на первую запись в списке. Таким образом, список становится зацикленным, при этом его размер никак не изменяется, что гарантирует, что данный список будет бесконечен. Для демонстрации откроем приложение и перейдем в шторку Navigation Drawer (Рис. 4).

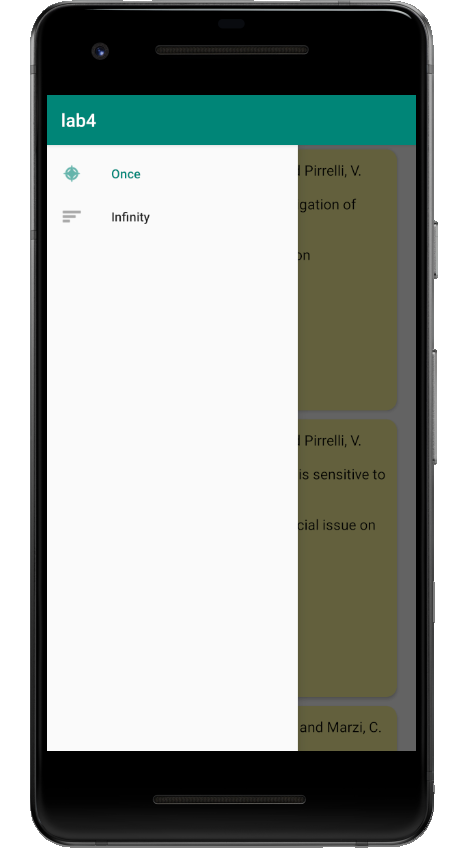


Рис. 4 Шторка Navigation Drawer

При переключении на бесконечный список дойдем до последней записи, однако необходимо оставить небольшой кусок CardView вне экрана, чтобы LayoutManager не переместил экран на первый элемент (Рис. 5).



Рис. 5 Последний элемент из списка

Чуть ниже поля pages находится нижняя грань CardView, при отображении которой сработает условие для onScrollListener и экран переместится на 0 элемент. Проверим, что это действительно так (Рис. 6).

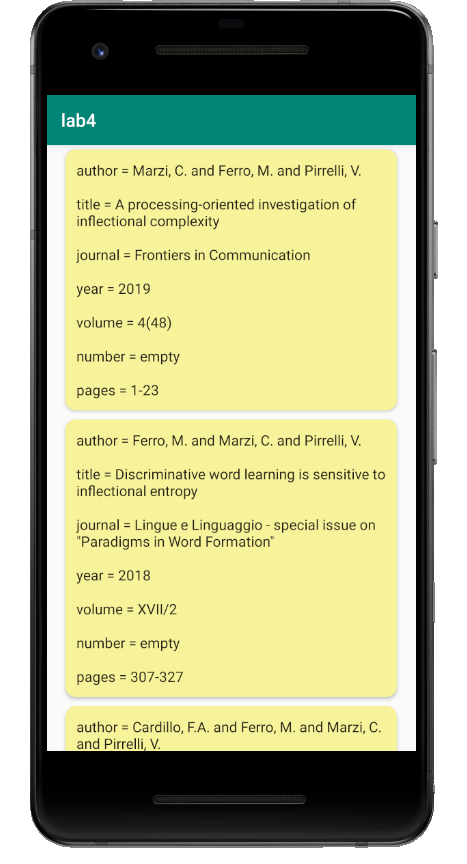


Рис. 6 Результат прокрутки экрана вниз

Как можно видеть, после прокрутки вниз экран не остановился на конце последнего элемента, а вернулся вновь к первому, таким образом бесконечный список работает именно так, как и предполагалось.

**ЗАКЛЮЧЕНИЕ**

В данной работе были изучены принципы исследования поведения сторонней библиотеки – был представлен исходный код, который необходимо было самостоятельно изучить и на основе этого сделать выводы о работе библиотеки и продемонстрировать полученные знания, дополнив junit тесты двумя новыми, подтверждающими правильное поведение библиотеки. После этого было предложено использовать данную библиотеку для составления приложения, отображающего в виде списка записи формата bibtex. Данное приложение было успешно сделано и протестирована работа как однородного списка, так и бесконечного, который возвращается к первому элементу списка после просмотра последнего. Для данного приложения был изучен и успешно использован класс RecyclerView, был написан адаптер для него.

Исходный код для лабораторных работ находится на [github](https://github.com/ADsty/labs-android_labs) (<https://github.com/ADsty/labs-android_labs> )

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

<https://developer.android.com/guide/topics/ui/layout/recyclerview>