Лабораторная работа 5.

Получение данных со страниц сайтов. Потоки.

В данной лабораторной работе научим помощника отвечать на вопросы о том, какой праздник сегодня, вчера, завтра или какого-то определенного числа, например, 4.4.2020. Для этого будем парсить html-страницы.

Будем использовать сайт с праздниками <http://mirkosmosa.ru/holiday>. На странице <http://mirkosmosa.ru/holiday/2020> информация о праздниках располагается достаточно удобно (рис. 1).

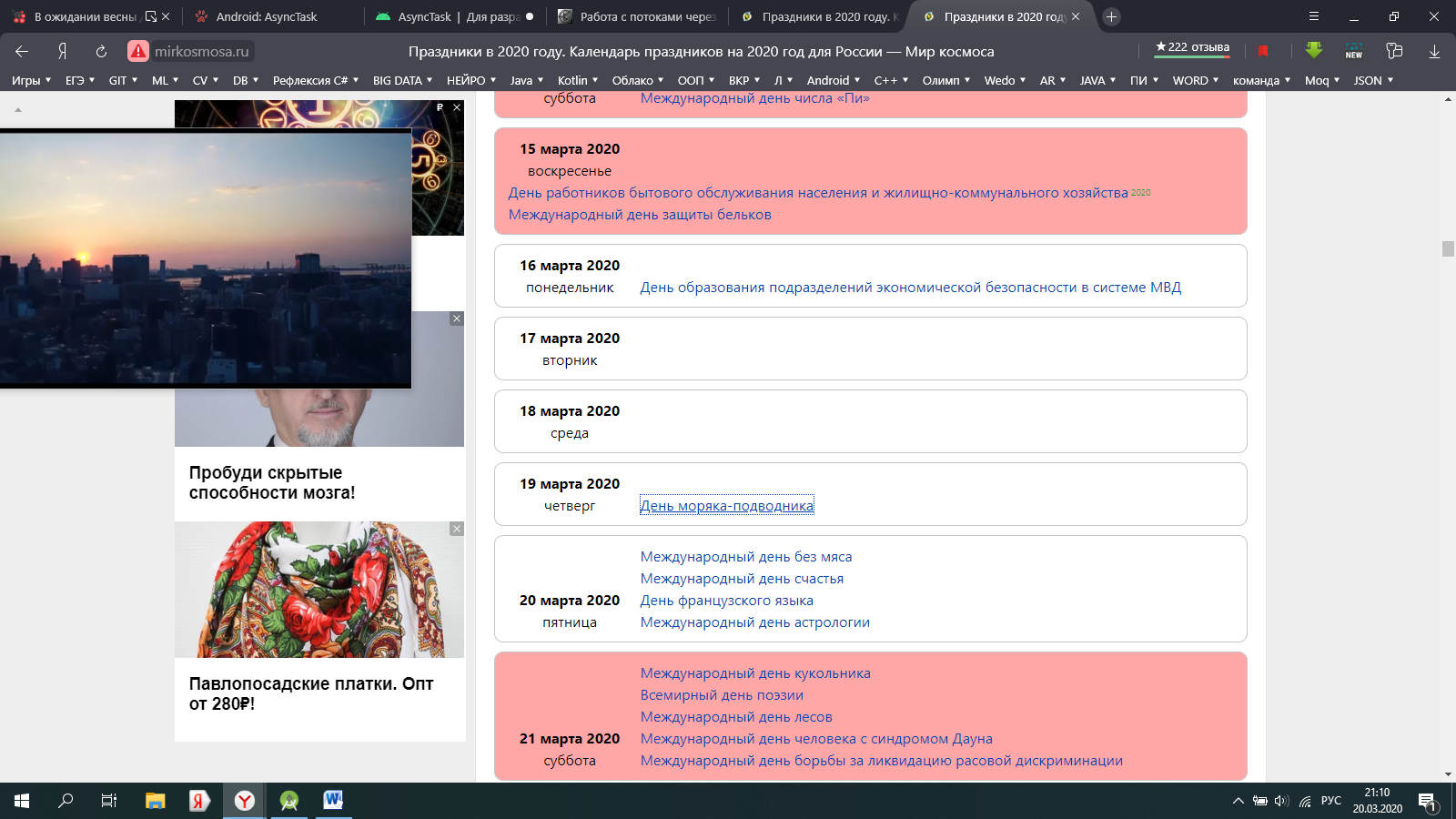


Рисунок 1.Расположение праздников на странице

Удобная разметка страницы позволяет быстро сориентироваться и организовать правильный поиск необходимого содержимого (рис.2).

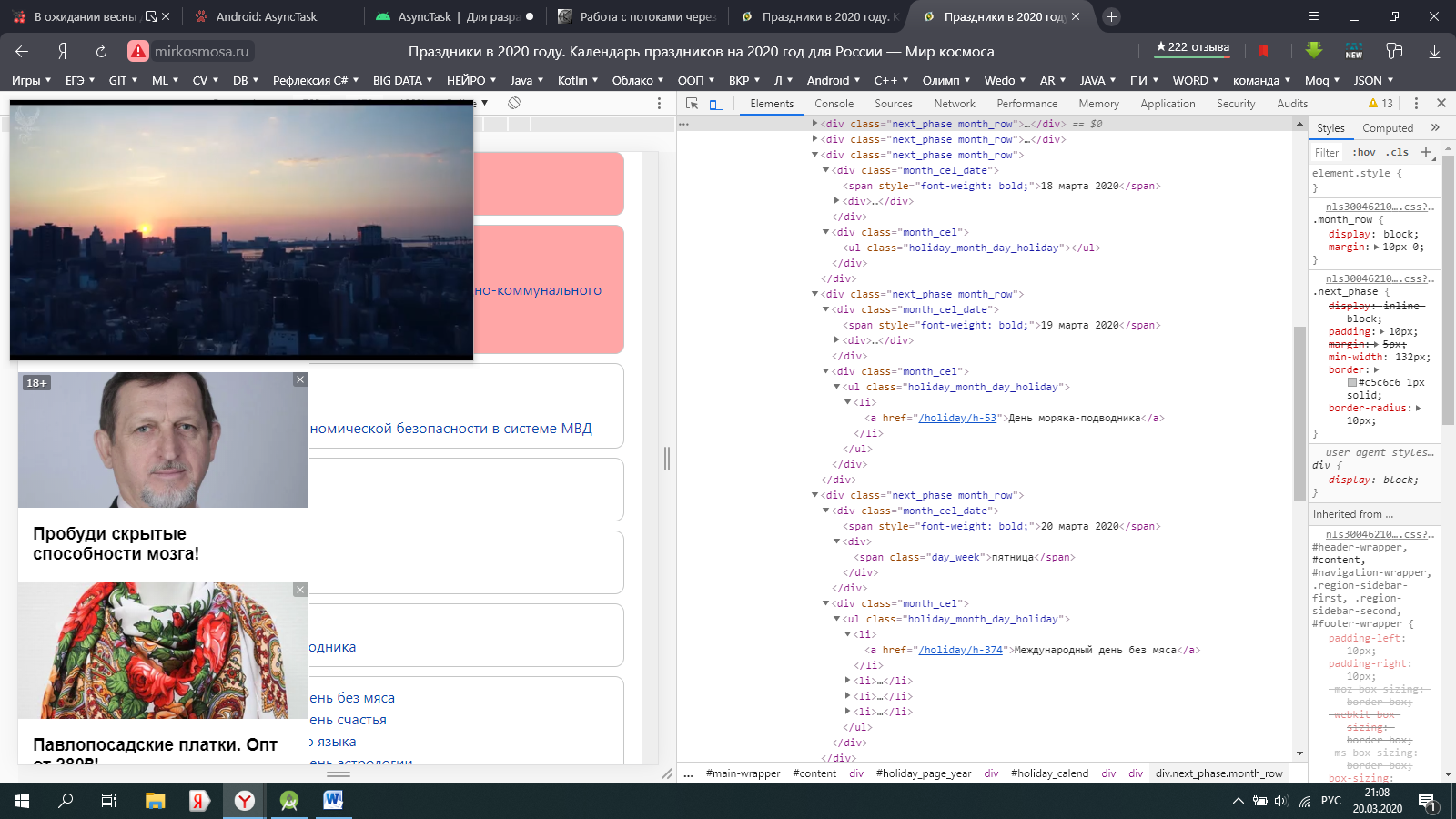


Рисунок 2. Разметка страницы

Можно организовать поиск следующим образом: искать тег div с классом month\_cel\_date. Внутри тега текст с датой и днем недели. Найдите элемент, дата которого совпадает с искомой датой. Затем можно перейти на следующий элемент. Это будет div с классом month\_cel (рис. 3).

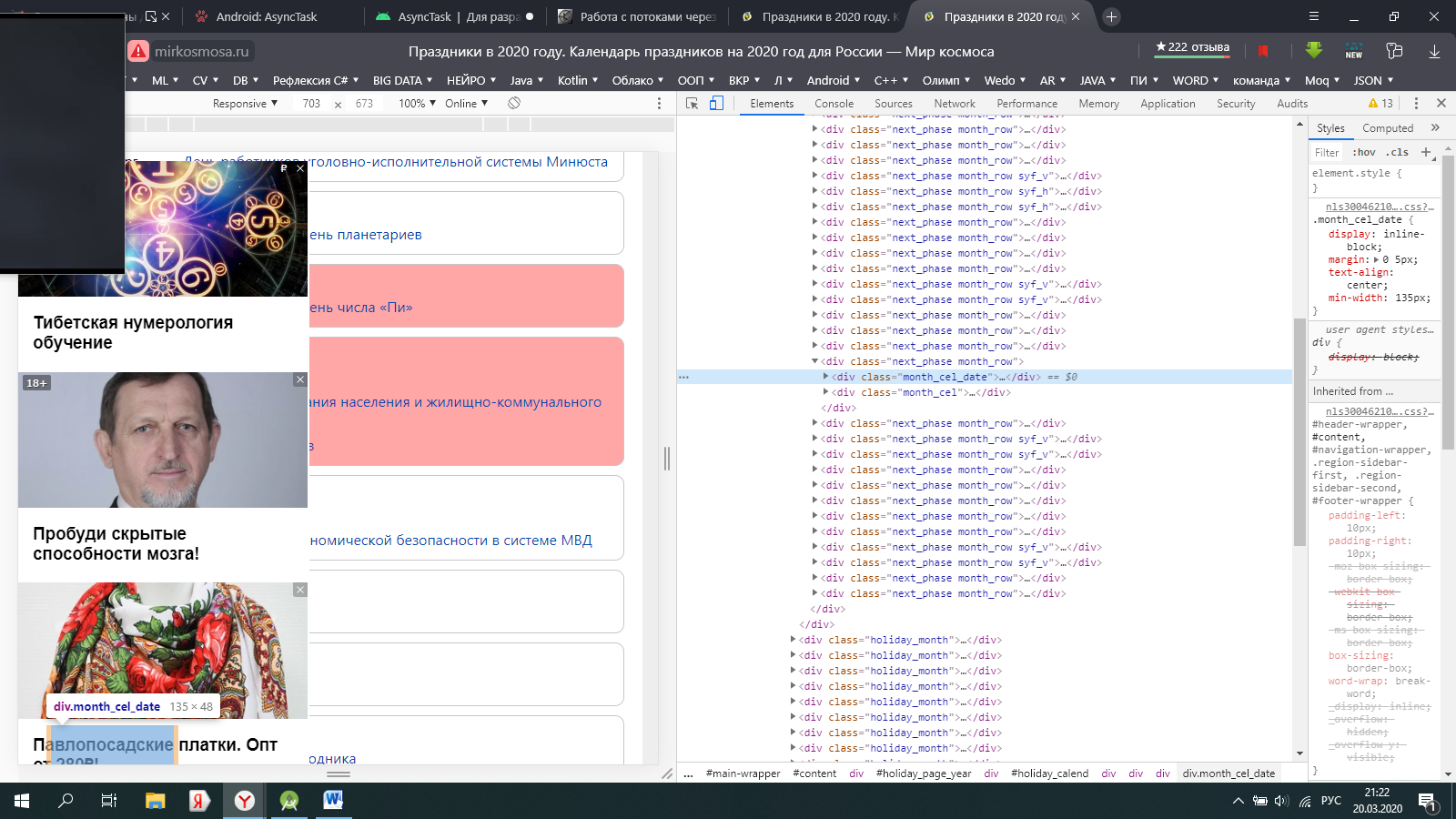


Рисунок 3. Искомые теги

Для разбора страницы будет использована легкая библиотека Jsoup. Это внешняя библиотека, на нее нужно добавить зависимость. Перейдите в файл build.gradle (Module: app) и добавьте новую зависимость в раздел dependencies:

implementation **'org.jsoup:jsoup:1.12.1'**

Объявите в классе константу для хранения адреса:

**private static final** String ***URL*** = **"http://mirkosmosa.ru/holiday/2020"**;

Создайте класс ParsingHtmlService, а в нем метод получения информации о праздниках на определенную дату. Дата передается параметром – строкой с датой в формате "dd MMMM yyyy":

**public static** String getHoliday(String date){

}

В теле метода необходимо получить объект Document:

Document document = Jsoup.*connect*(***URL***).get();

А затем получить его тело:

Element body = document.body();

Методы библиотеки возвращают либо один Element, либо коллекцию элементов Elements.

Дальше нужно реализовать поиск, описанный выше. Библиотека Jsoup предоставляет богатый выбор методов <https://o7planning.org/ru/10399/jsoup-java-html-parser-tutorial>. Среди методов удобным является метод select. Он возвращает коллекцию элементов, которую можно просматривать методом foreach. Чтобы получить тег с определенным именем класса, нужно передать в метод select строку вида "тег.класс".

Чтобы проверить, правильно ли работает данный метод, нужно использовать тесты (рис. 4).

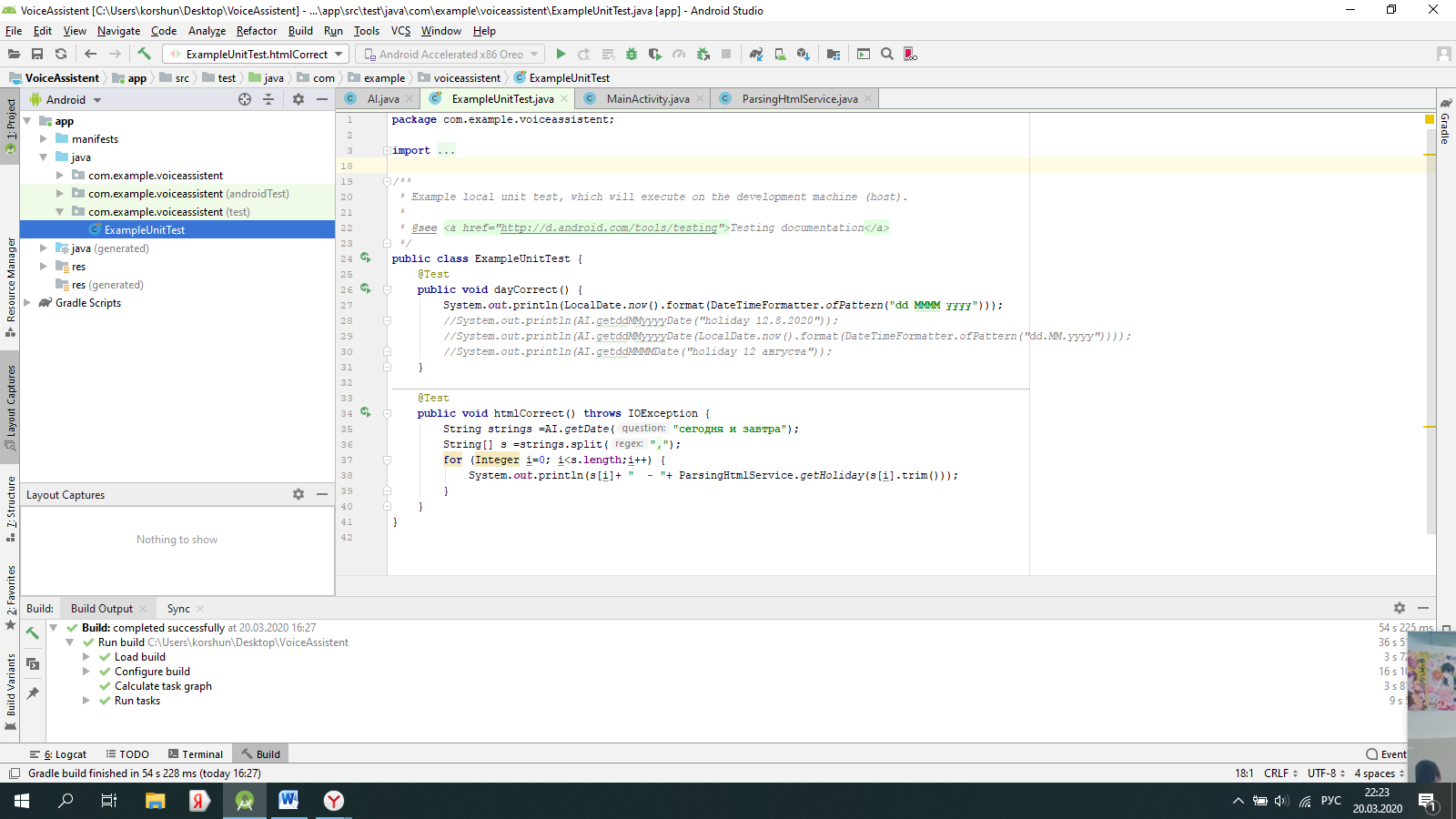


Рисунок 4. Проект приложения

В проекте уже содержится тестовый файл ExampleUnitTest и тестовый метод:

**public class** ExampleUnitTest {  
 @Test  
 **public void** тестовыйМетод() {

}  
}

Конечно, в тестовом методе используют класс Assert и его методы. Но достаточно выводить в консоль результат выполнения метода.

Проверьте, какие результаты выдает метод getHoliday класса ParsingHtmlService с параметром "19 марта 2020", "20 марта 2020".

Таким образом, помощник будет отвечать на вопрос типа: «праздник 20 марта 2020».

Далее нужно научить помощника отвечать на вопросы: праздник сегодня, праздник завтра, праздник вчера, праздник 20.03.2020 или комбинацию данных вопросов. Для этого нужно создать метод getDate в классе AI, который берет вопрос и возвращает строку с датами в нужном формате, например "10 марта 2020, 11 марта 2020, 12 апреля 2020".

Протестируйте корректность работы метода getDate на корректность получения даты в формате dd MMMM yyyy.

Осталось только вызвать описанные методы в классе AI.

Для каждого приложения операционная система выделяет свой поток. Этот поток называют основным, графическим, его предназначение – отображение activity.

Операция получения данных с сайта может занимать некоторое время. Выполнять запросы к сети в основном потоке приложения НЕЛЬЗЯ, так как сетевая операция может заблокировать основной (UI) поток! Нужно создать новый поток (асинхронный).

Для создания потока будем использовать класс AsyncTask (пока он еще жив). Понятно и с примерами про него написано в <http://developer.alexanderklimov.ru/android/theory/asynctask.php> или в <https://developer.android.com/reference/android/os/AsyncTask>.

Запрос нужно выполнять в методе getAnswer класса AI, поэтому и класс нужно описывать там же. Вернее не нужно описывать отдельный класс, создайте анонимный класс и вызовите у него метод execute.

Перейдите в класс AI. В методе getAnswer можно определить содержит ли вопрос слово «праздник». Если вопрос о празднике, то нужно получить строку с датами в формате: dd MMMM yyyy:

String findDate = *getDate*(question);

Далее создайте анонимный класс:

**new** AsyncTask<String, Integer, Void>() {  
}

Вызовите у созданного анонимного класса метод execute. Он принимает последовательность параметров через запятую:

execute(findDate.split(**","**));

Это абстрактный класс, поэтому он потребует реализовать метод doInBackground:

@Override  
**protected** Void doInBackground(String... strings) {  
 **return null**;  
}

В нем вызовите метод getHoliday от всех элементов из последовательности strings и поместите результат в ***answers***.

Далее добавьте метод onPostExecute и в нем добавьте ***answers*** в callback.

**Запуск**. Запустите приложение и протестируйте его работу.

Google грозится сделать deprecated класс AsyncTask уже в Android 11. Что делать? Популярность набирает RxJava (ReactiveX) – библиотека, позволяющая организовать множество действий, обусловленных определенными событиями в системе.

Для использования библиотеки нужно добавить зависимость в gradle:

implementation **'io.reactivex.rxjava3:rxandroid:3.0.0'**implementation **"io.reactivex.rxjava3:rxjava:3.0.0"**

Далее в методе getAnswer класса AI закомментируйте код, создающий AsyncTask. Вместо AsyncTask будем использовать класс io.reactivex.rxjava3.core.Observable.

Observable.*fromCallable*();

Так как это сетевой вызов, нужно использовать метод fromCallable, в который нужно передать экземпляр класса Callable. Можно создать анонимный класс Callable<Object> и реализовать у него метод call:

Observable.*fromCallable*(**new** Callable<Object>() {  
 @Override  
 **public** Object call() **throws** Exception {  
 **return null**;  
 }  
})

Внутрь метода call нужно поместить все, что было сделано в методе doInBackground.

НО можно сократить код!

Так как данный класс реализует только один метод call, то его можно заменить лямбдой:

Observable.*fromCallable*(()->{  
 *// здесь нужно записать необходимый код* **return null**;  
})

Вместо комментария запишите необходимую логику получения и возвращения данных. Обязательно при получении данных верните их:

**return *answers***;

После метода fromCallable нужно вызвать методы subscribeOn, observeOn и subscribe:

Observable.*fromCallable*(()->{  
 *// здесь нужно записать необходимый код и return answers;* **return null**;  
})

.subscribeOn(Schedulers.*io*())  
.observeOn(AndroidSchedulers.*mainThread*())

.subscribe((result) -> {  
});

С помощью оператора subscribeOn можно указать Scheduler (планировщик), в котором будет выполняться процесс Observable. В данном случае в нем указан планировщик Schedulers.io() для выполнения операции в потоке ввода-вывода. IO Scheduler основан на неограниченном пуле потоков. Для выполнения задачи из пула выбирается поток и затем возвращается в пул после завершения работы. Если в пуле нет свободных потоков, то создается новый поток и добавляется в пул. Этот планировщик не советуют использовать для выполнения большой вычислительной работы. В данном случае получение страницы и данных с нее – не громоздкая операция.

Оператор observeOn() используется вместе с планировщиком AndroidSchedulers.mainThread() для того, чтобы обрабатывать результат в основном потоке и передать результат в пользовательский интерфейсе приложения.

Оператор subscribe выполняет подписку. Поместите в метод (в лямбду, внутрь скобок {}) добавление данных в callback.

Дополнительно можно прочитать документацию <http://reactivex.io/documentation/operators/subscribeon.html> и статью <https://habr.com/ru/post/344016/>.