Задача:

Написать программу, которая с консоли считывает адрес (в пределах Петербурга), и выводит список всех мест, которые находятся по этому адресу (кафе, магазины и так далее).

Задача разбивается на три этапа:

1. Считать адрес.
2. Сделать запрос к серверу.
3. Распарсить ответ.
4. Вывести результат.

Первый и четвертый пункт не сильно нуждаются в пояснении, остановимся на запросе к серверу.
Эту задачу тоже можно разбить на несколько этапов:

1. Генерация запроса
2. Запрос к серверу
3. Подготовка к обработке ответа
4. Обработка ответа

Рассмотрим это подробнее:

Генерация запроса

Для получения данных по адресу будем использовать OpenStreetMap и Nominatim:
OpenStreetMap (дословно «открытая карта улиц»), сокращённо OSM — некоммерческий веб-картографический проект по созданию силами сообщества участников — пользователей Интернета подробной свободной и бесплатной географической карты мира.
https://ru.wikipedia.org/wiki/OpenStreetMap

А Nominatim, это сервис для поиска по OpenStreetMap.
http://wiki.openstreetmap.org/wiki/Nominatim

Nominatim предоставляет возможность делать поисковые запросы, без ключей и лицензий. Вот таким, примерно, образом:
nominatim.openstreetmap.org/search?street=Невский+проспект%2C+100&format=json&city=СПб

Вы можете открыть эту ссылку, и посмотреть на результат вывода.
За «базовую» ссылку можно взять
http://nominatim.openstreetmap.org/search?street=%s&format=json&city=СПб

Однако, чтобы запрос прошел удачно, следует убрать из ссылки недопустимые символы, то есть сделать Percent-encoding, он же URL Encoding.
Для этого в java можно воспользоваться статическим методом encode в классе URLEncoder, вот так:

street = URLEncoder.encode(street, "UTF-8");

Вот и все, URL готов! Осталось теперь сделать запрос к северу…

Запрос к серверу

Следует воспользоваться классом URL. Это самое простое. Просто создать, открыть соединение и получить InputStream. Нам его даже не надо читать, за нас это сделает библиотека GSON.

Подготовка к обработке ответа

Ответ к нам приходит в формате JSON. Это важно.
Но нам его не надо парсить вручную, для этого есть библиотека Gson от Google.
О том, как ей пользоваться, будет рассказано на занятии. Но, тем не менее, примеры есть тут:
https://habrahabr.ru/company/naumen/blog/228279/
http://www.javenue.info/post/gson-json-api

Нам для подготовки понадобится «обертка» вокруг сущности Place, которую возвращает сервер.
В принципе, хватит и такого:

```
public class Place {
    private String display_name,  osm_type;
    private String type;
}
```

Обработка ответа

Тут вроде все просто, но есть нюансы.
Казалось бы, мы могли бы вызвать метод gson.fromJson, передав туда полученные от URL InputStream, но мы знаем, что результат запроса к серверу это не один Place, а несколько.

Так как метод fromJson параметризован, то он должен сам внутри создать список наших объектов. Но как уже говорилось, в Java никак нельзя узнать, чем параметризован класс динамически, поэтому бесполезно вызывать метод fromJson передавая туда List<Place>. Для решения этой проблемы, GSON использует TypeToken, вот так:

```
Type listType = new TypeToken<ArrayList<Place>>() {}.getType();
```

Из экземпляра listType GSON уже «вытащит» тип и создаст внутри себя лист. Так что можно написать:

```
List<Place> places = gson.fromJson(
        new InputStreamReader(inputStreamFromServer), listType);
```

Теперь с листом places можно работать, и вывести его на экран.

Если все уже работает

Тогда следует улучшить результат. Хорошо бы, чтобы запрос к серверу выполнялся в другом потоке. Это сделать не сложно. А вот для того, чтобы узнать что поток запрос сделал и распарсил результат, можно «повесить» на него Listener, вроде такого:

```
interface DownloadListener{
    void onError();
    void onDownload(List<Place> places);
}
```

Использование аннотаций

В классе Place использованы имена, совпадающие с именами в JSON-ответе. Но что, если это не так? Вот для этого и можно использовать аннотации:

```java
@SerializedName("age")
private int dwarfAge;
```