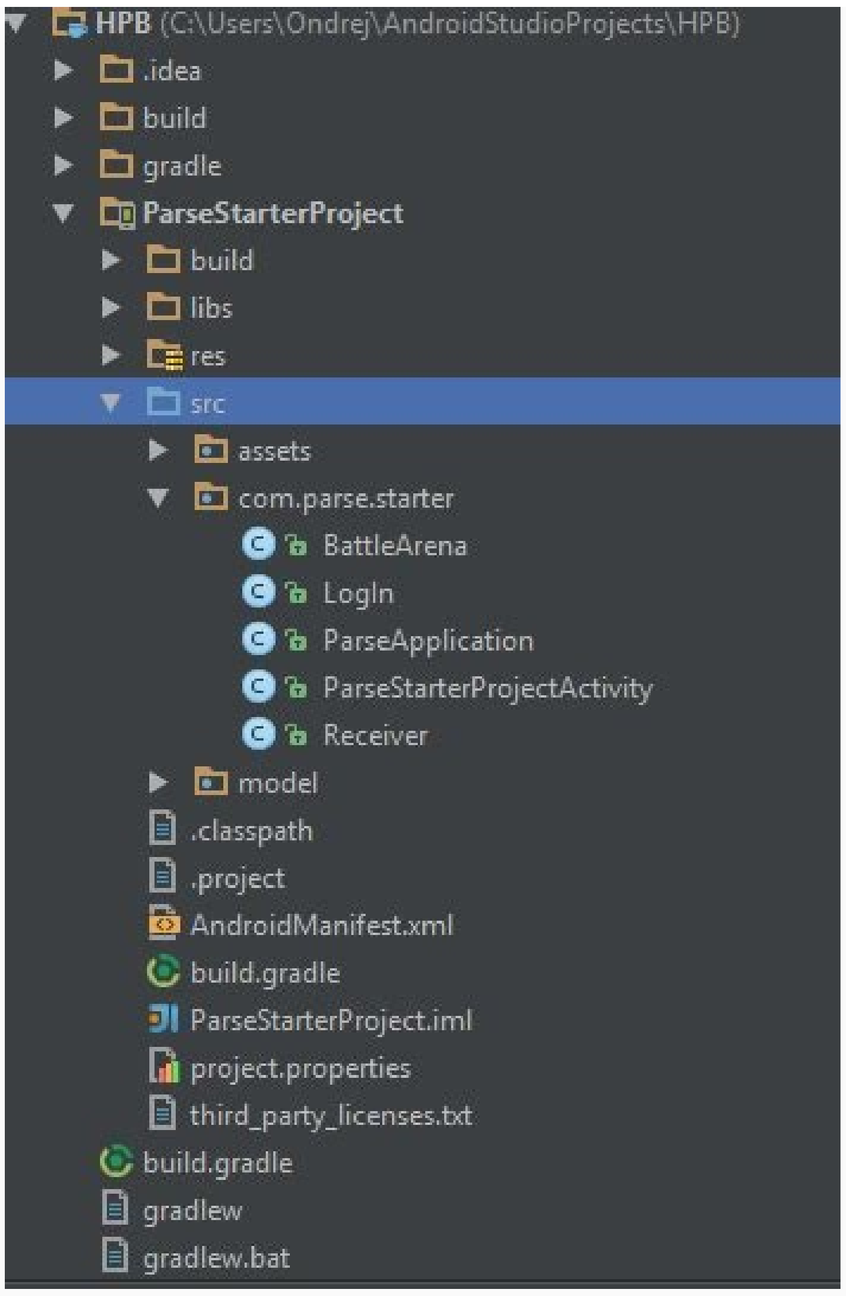
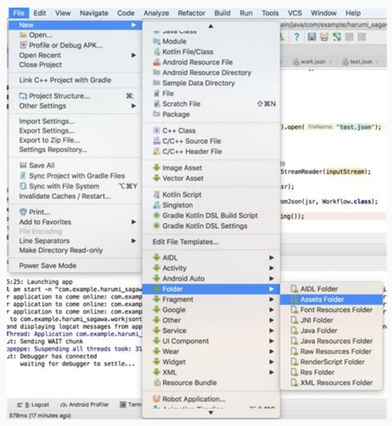
Gson android studio

Continue


HPB (C:\Users\Ondrej\AndroidStudioProjects\HPB)
.idea
build
gradle
ParseStarterProject
build
libs
res
src
assets
com.parse.starter
BattleArena
LogIn
ParseApplication
ParseStarterProjectActivity
Receiver
model
.classpath
.project
AndroidManifest.xml
build.gradle
ParseStarterProject.iml
project.properties
third_party_licenses.txt
build.gradle
gradlew
gradlew.bat


90% 10:07
halo
Halo - The Master Chief Collection - Xbox One
XBOX ONE
ALLEEN OP XBOX ONE
HALO
THE MASTER CHIEF COLLECTION
VOOR HET EERST IN DE GESCHIEDENIS DE VOLLEDIGE
1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
!#
English (UK)
.


G-SONE



Cannot resolve symbol gson android studio. Android studio unresolved reference gson. Import com.google.gson.gson error android studio. Gson dependency android studio. Gson android studio example. Android studio gson is undefined. Gson plugin android studio. Gson android studio kotlin.

Пример для Kotlin Библиотека GSON была разработана программистами Google и позволяет конвертировать объекты JSON в Java-объекты и наоборот. Домашняя страница: Гитхаб Установим зависимость в Gradle. implementation 'com.google.code.gson:gson:2.8.5' Конвертируем объект в JSON Создадим простейший класс Cat с открытыми полями. package ru.alexanderklimov.gsondemo; public class Cat { public String name; // имя public int age; // возраст public int color; // цвет // Конструктор public Cat(){ } } Попробуем сконвертировать объект созданного класса в JSON при помощи метода toJson(). Cat murzik = new Cat(); murzik.name = "Мурзик"; murzik.age = 9; murzik.color = Color.BLACK; GsonBuilder builder = new GsonBuilder(); Gson gson = builder.create(); Log.i("GSON", gson.toJson(murzik)); В логах видим строку: {"name":"Мурзик","color":-16777216,"age":9} Вот так просто можно превратить объект в строку. Это удобно при передаче данных, например, из приложения на сервер. Перепишем пример немного иначе. Cat barsik = new Cat(); barsik.name = "Барсик"; barsik.age = 8; Gson gson = new Gson(); Log.i("GSON", gson.toJson(barsik)); Смотрим на ответ. Теперь все данные данные выводятся по алфавиту. {"age":8,"color":0,"name":"Барсик"} Конвертируем JSON в объект Естественно, нам нужно уметь выполнять и обратную задачу. Допустим с сервера пришёл ответ в виде JSON-строки и мы должны из неё построить объект для работы в своём приложении. В этом случае вызывается метод fromJson(). String jsonText = "{\"name\":\"Мурзик\",\"color\":-16777216,\"age\":9}"; GsonBuilder builder = new GsonBuilder(); Gson gson = builder.create(); Cat murzik = gson.fromJson(jsonText, Cat.class); Log.i("GSON", "Имя: " + murzik.name + "Возраст: " + murzik.age); В этом примере нам пришлось экранировать кавычки. Но суть от этого не меняется. Получив строку, мы смогли создать объект murzik и узнать его имя, возраст, цвет. Сложный класс Класс Cat состоит из примитивных типов. Но иногда классы содержат объекты других классов. Усложним класс, добавив новый класс Address. // Cat.java package ru.alexanderklimov.gsondemo; public class Cat { public String name; // имя public int age; // возраст public int color; // цвет public Address address; public Cat(){ } } // Address.java package ru.alexanderklimov.gsondemo; public class Address { String street; String city; String country; public Address(String street, String city, String country){ this.street = street; this.city = city; this.country = country; } } Посмотрим, что получится. Cat murzik = new Cat(); murzik.name = "Мурзик"; murzik.age = 9; murzik.color = Color.BLACK; murzik.address = new Address("Arbat", "Moscow", "Russia"); GsonBuilder builder = new GsonBuilder(); Gson gson = builder.create(); Log.i("GSON", gson.toJson(murzik)); // {"address":{"city":"Moscow","country":"Russia","street":"Arbat"},"age":9,"color":-16777216,"name":"Мурзик"} GSON справился с заданием и показал правильный результат (опять по алфавиту). Пробуем в обратном порядке - из json-строки получим объект. String jsonText = "{\"address\":{\"city\":\"New York\",\"country\":\"USA\"," + "\"street\":\"Wall Street\"},\"age\":11,\"color\":-16777216,\"name\":\"Murzik\"}"; GsonBuilder builder = new GsonBuilder(); Gson gson = builder.create(); Cat murzik = gson.fromJson(jsonText, Cat.class); Log.i("GSON", "Имя: " + murzik.name + "Возраст: " + murzik.age + "City: " + murzik .address.city); // GSON: Имя: Murzik // Возраст: 11 // City: New York Тоже работает. Аннотации Можно использовать аннотации, чтобы помочь библиотеке разобраться с полями класса, если они не совпадают с нужным именем в json. @SerializedName("Id") public long id; Мы не рассмотрели примеры, когда объект содержит массив/список элементов, отображения (Map), Set. Пример для Kotlin На данный момент использовать библиотеку GSON в Kotlin не рекомендуется, так как она не учитывает возможности языка при работе с null. Используйте альтернативы в виде библиотек Moshi, Jackson или специального плагина kotlinx.serialization. Тем не менее приведу один пример. Для простоты создадим строку самостоятельно в формате JSON. val jsonStr = """ { "name": "Barsik", "age": 21, "isAwesome": true } """.trimIndent() Создадим класс Cat на основе этого файла. data class Cat(val name: String, val age: Int, val isAwesome: Boolean) Строим объект через метод fromJson(). val jsonStr = """ { "name": "Barsik", "age": 21, "isAwesome": true } """.trimIndent() val cat: Cat = Gson().fromJson(jsonStr, Cat::class.java) println(cat) Дополнительное чтение Реклама Can someone give me step by step guide to add the Gson library to an Android project? I tried the JSON built-in library but that seems to be a bit tedious right now. I saw a couple of examples based on Gson, and that seems really easy. In this tutorial I will discuss about how to parse a class object into json string and convert it back from json string to class object in Android Kotlin using popular Gson library.Configuring Module: app/ build.gradleTo use the Gson library you need to add following dependency in your projects module gradle file (Module: app/build.gradle) and re sync.Let’s assume, we have a data class Student,Parsing class object into json stringIf you want to convert single hierarchical class object into equivalent json string you simply need to call Gson().toJson()Parsing json string into class objectOn the other hands, you can convert a json string into class object with below code. If you do not declare SerializedName (discussed in next section), your class field name should be same to json keys.Customizing json keys while parsingSometimes server provided json keys are not neat and clean or not satisfactory with your project coding styles you follow. So you might want to make your data class field names non identical to json key names. You can do that with SerializedName annotation before each field.Parsing nested json string to nested objectWhat if the json string is not of a simple single level object rather it also have nested json object? Let’s say some json string like below one?In that case you can define nested classes and use them with GsonSkipping nested hierarchy while parsingSometime you might want to skip deep down lower hierarchy of json object because it requires creating many small nested data classes. In that case one trick is to store lower level json objects as string in the parent class field. For example,In the above json format if you don’t want to deal with city and post at that granular level so defining data class Address might be an overhead. You can simply store address object as a string in your student class. In that case you need to use JsonDeserializer.Thus you can parse a nested hierarchical json object into a less hierarchical class object,Similarly when we want to retrieve back the json string from the class object the specific field requires to be parsed directly from string representation to json object. In that case we use JsonSerializer.To call this toJason() we need to use like,Parsing between a Collection, List or ArrayGson is capable of parsing to and from an array or a list automatically into json object and vice versa. See the below code snippet,When the custom object has its setup done (discussed as previous sections i.e. SerializedName or Serilizer/Deserializer), list of custom objects can easily be parsed similarly as primitive types without doing any extra work.Validating Json while debuggingThere is a good site to validate your json. That is . TypedAdapterFactory seems like is not generated (or included) into final app.apk/META-INF/services. Could be an android build tools bug, and here is a workaround Also, those are normally generated and present: javax.ws.rs.ext.MessageBodyReader javax.ws.rs.ext.MessageBodyWriter build.gradle is configured like this: provided 'org.immutables:value:2.0.6' apt 'org.immutables:value:2.0.6' compile 'org.immutables:gson:2.0.6' apt 'org.immutables:gson:2.0.6' TypeAdapters are generated as well and I can link 'em manually.