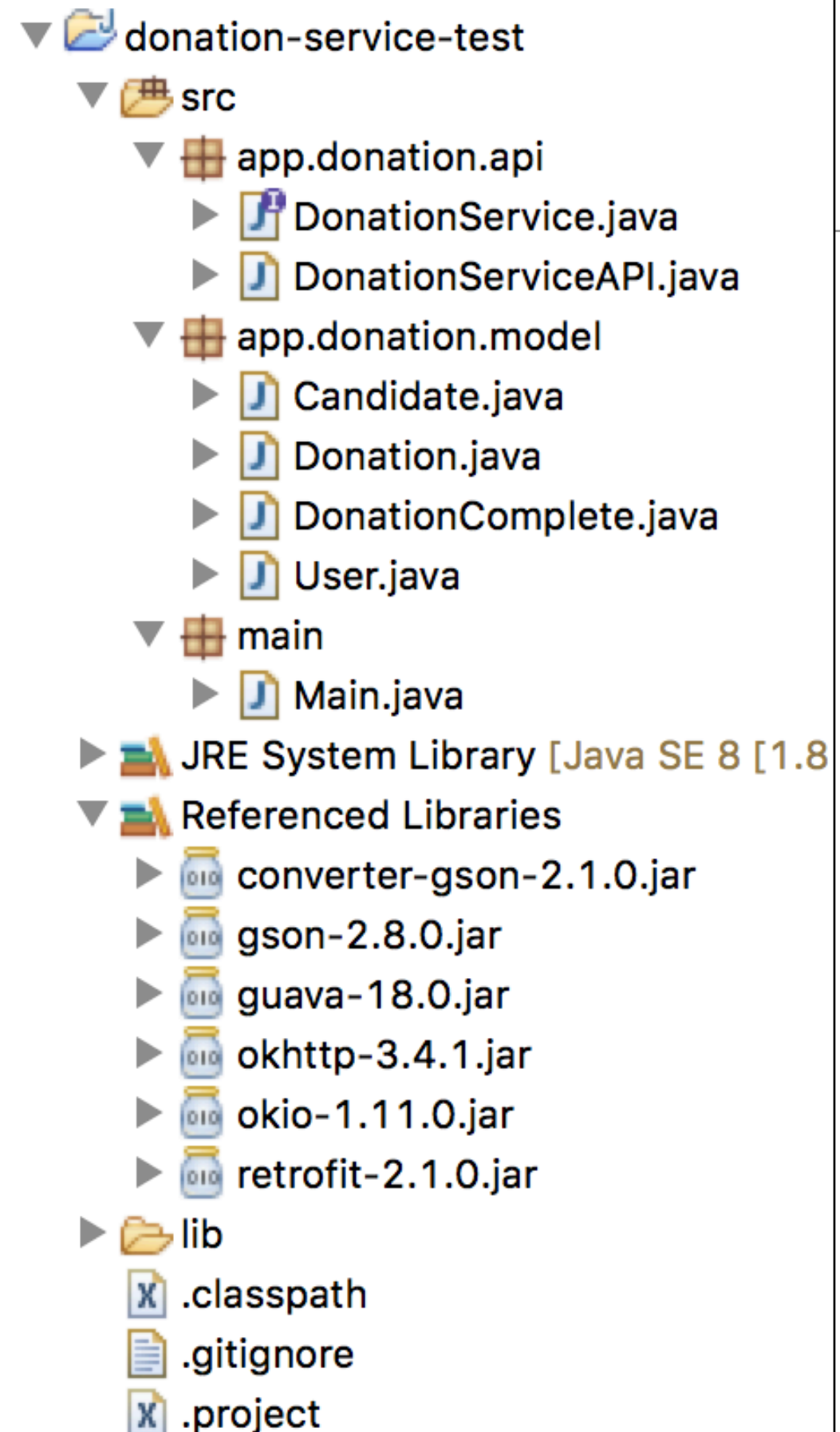
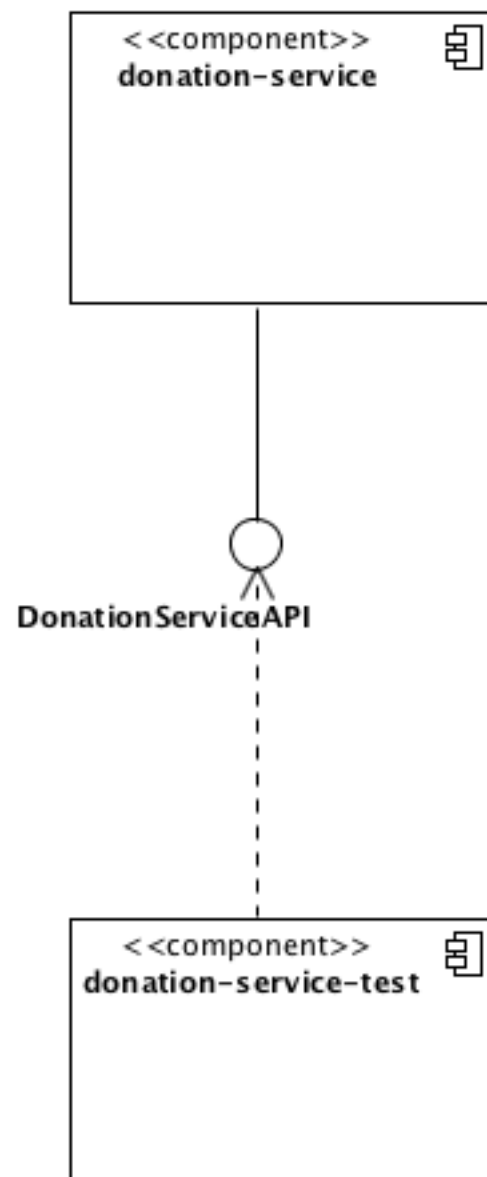


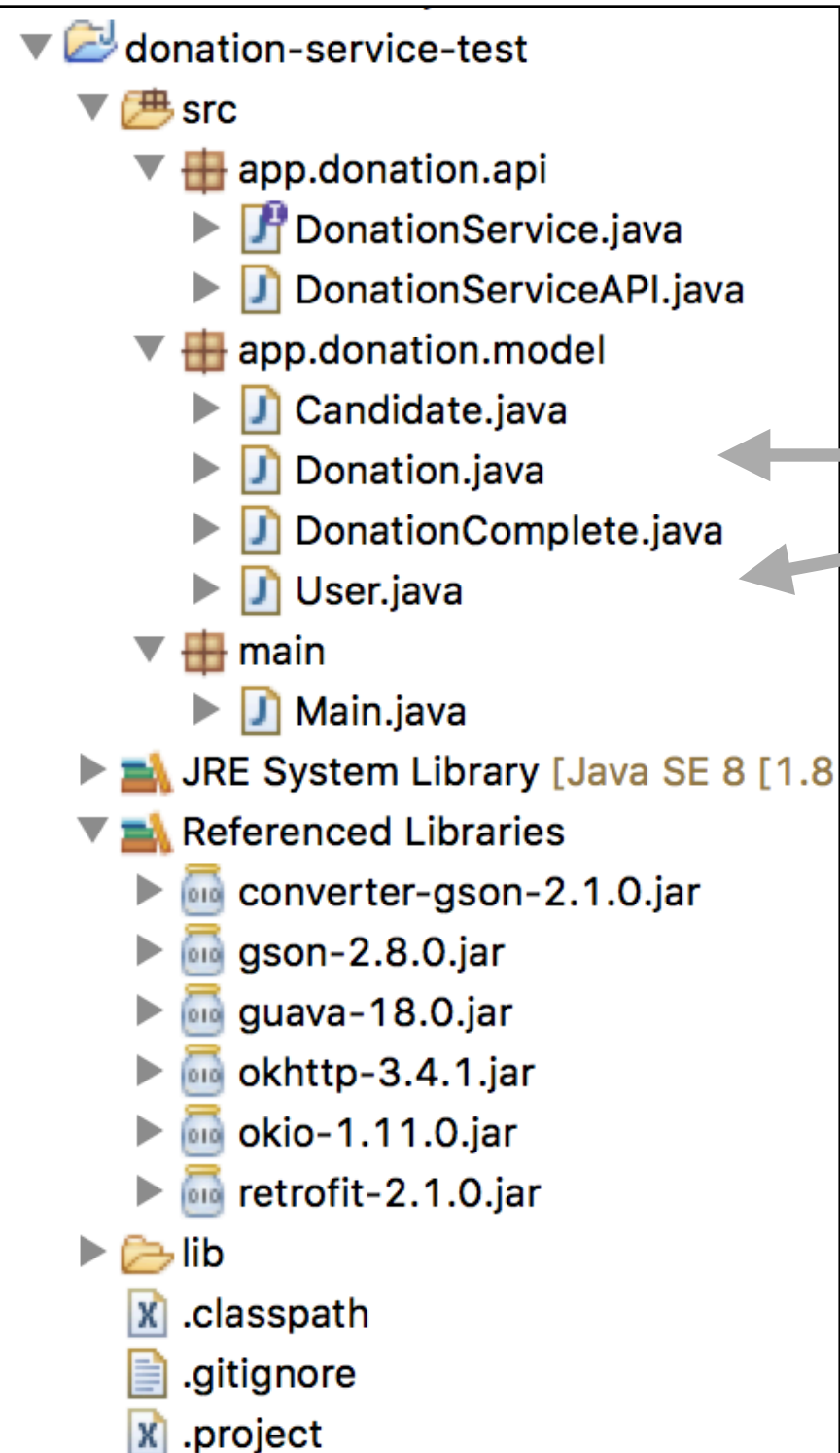
Java Rest Client

donation-service-test



Donation Models

- Java versions of Mongoose models



```
public class Candidate
{
    public String _id;
    public String firstName;
    public String lastName;
    public String office;

    public Candidate(String firstName, String lastName, String office)
    {
```

```
public class User
{
    public String _id;
    public String firstName;
    public String lastName;
    public String email;
    public String password;

    public User(String firstName, String lastName, String email, String password)
    {
```

```
public class Donation
{
    public String _id;
    public int amount;
    public String method;

    public Donation (int amount, String method)
    {
        this.amount = amount;
        this.method = method;
    }
}
```

Javascript Mongoose vs Java Models

Javascript

```
const candidateSchema = mongoose.Schema({
  firstName: String,
  lastName: String,
  office: String,
});
```

```
const donationSchema = mongoose.Schema({
  amount: Number,
  method: String,
  donor: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
  },
  candidate: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Candidate',
  },
});
```

Java

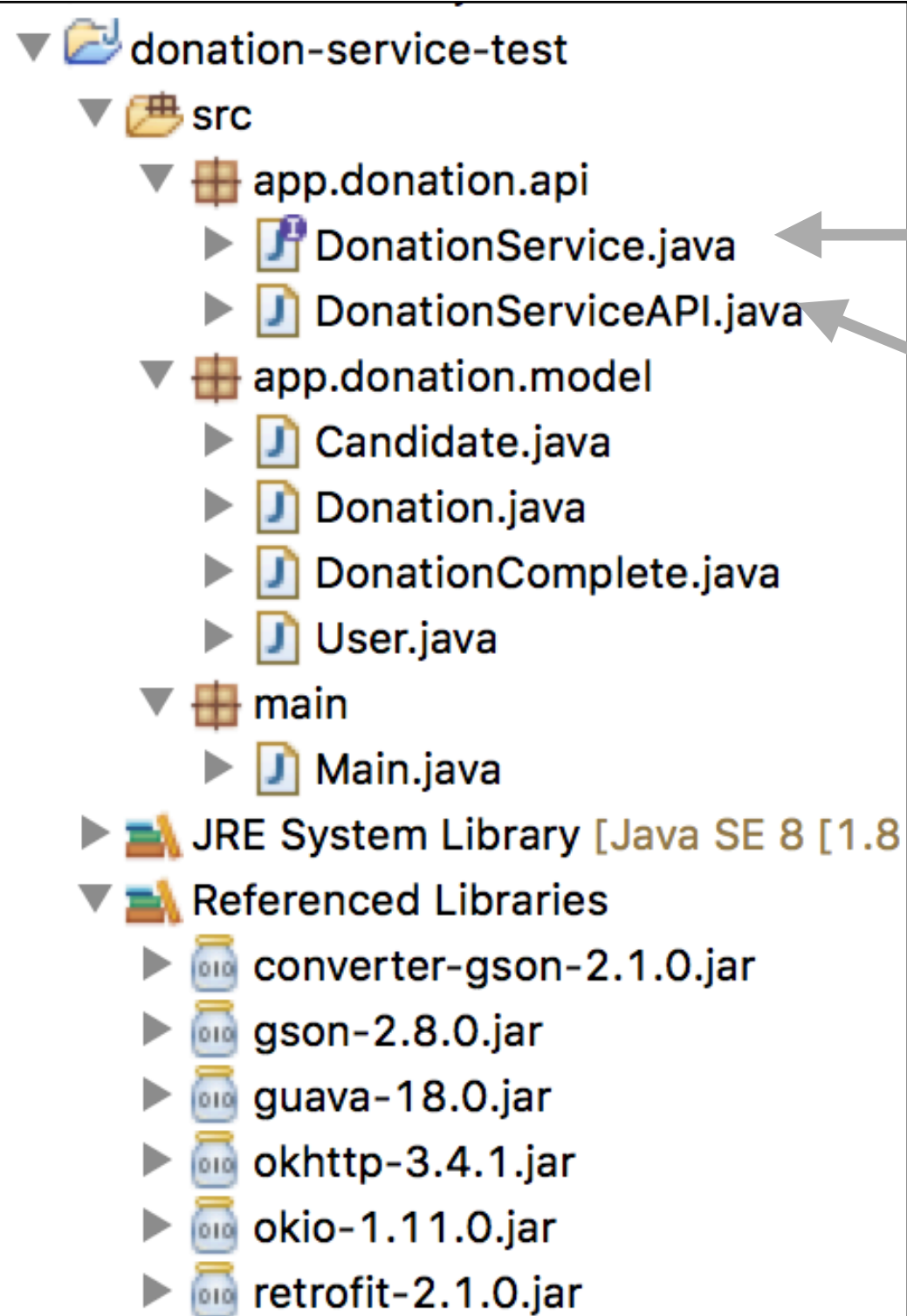
```
public class Candidate
{
    public String _id;
    public String firstName;
    public String lastName;
    public String office;

    public Candidate(String firstName, String lastName, String office)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.office = office;
    }
}
```

```
public class Donation
{
    public String _id;
    public int amount;
    public String method;

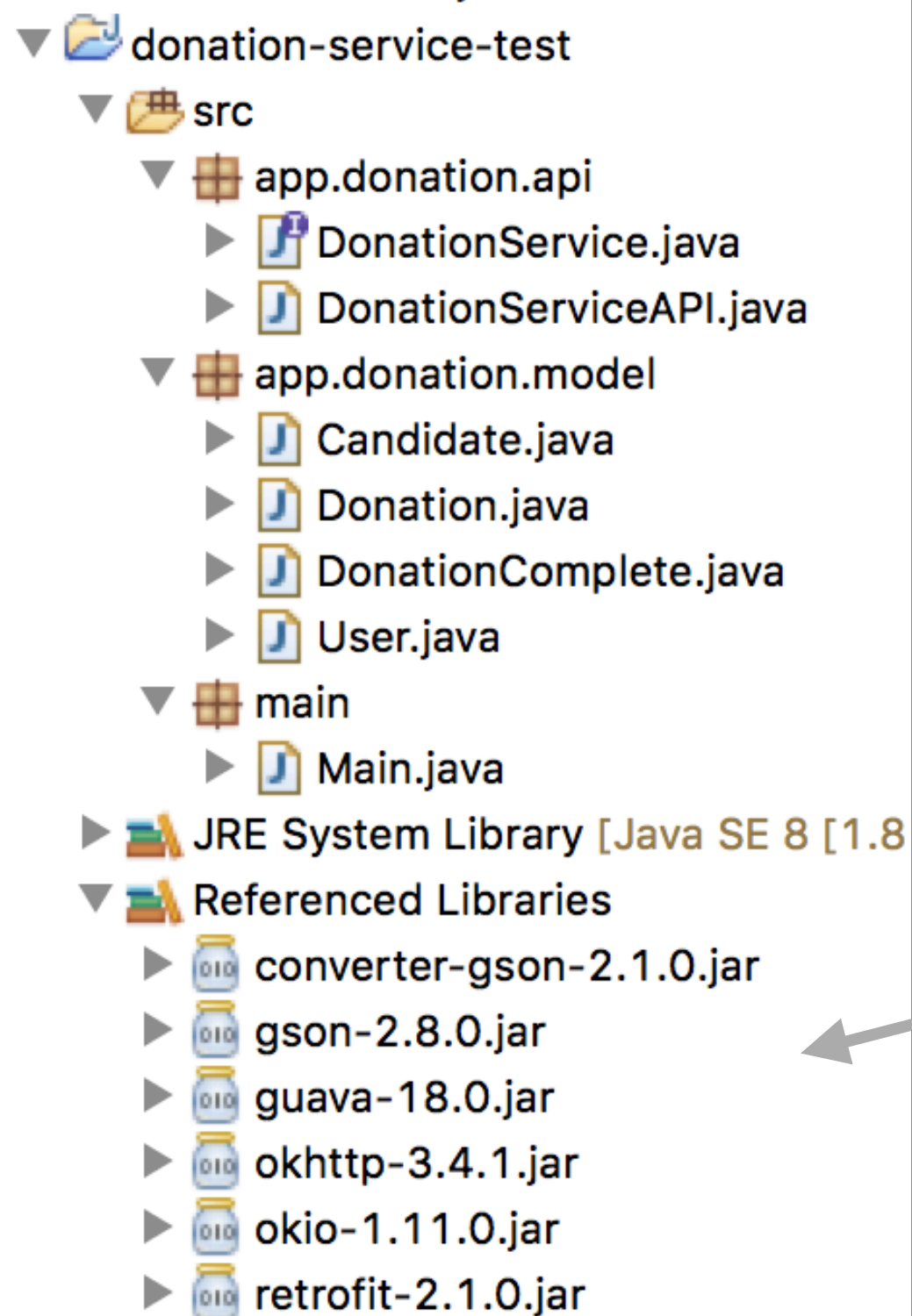
    public Donation (int amount, String method)
    {
        this.amount = amount;
        this.method = method;
    }
}
```

Donation API



- Java Wrappers to deliver a client side API.
- These class will be responsible for composing the HTTP Requests and sending them to donation-web

Donation API



- Retrofit Libraries to simplify Rest API development in Java

A type-safe HTTP client for Android and Java

Introduction

Retrofit turns your HTTP API into a Java interface.

```
public interface GitHubService {  
    @GET("users/{user}/repos")  
    Call<List<Repo>> listRepos(@Path("user") String user);  
}
```

The `Retrofit` class generates an implementation of the `GitHubService` interface.

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("https://api.github.com/")  
    .build();  
  
GitHubService service = retrofit.create(GitHubService.class);
```

Each `Call` from the created `GitHubService` can make a synchronous or asynchronous HTTP request to the remote webserver.

```
Call<List<Repo>> repos = service.listRepos("octocat");
```

Use annotations to describe the HTTP request:

- URL parameter replacement and query parameter support
- Object conversion to request body (e.g., JSON, protocol buffers)
- Multipart request body and file upload

[Introduction](#)[API Declaration](#)[Retrofit Configuration](#)[Download](#)[Contributing](#)[License](#)[Javadoc](#)[StackOverflow](#)

DonationService

▼ app.donation.api
▶ DonationService.java
▶ DonationServiceAPI.java

- Provides convenient access in a client to a remote service

```
public interface DonationService
{
    @GET("/api/users")
    Call<List<User>> getAllUsers();

    @GET("/api/users/{id}")
    Call<User> getUser(@Path("id") String id);

    @POST("/api/users")
    Call<User> createUser(@Body User User);

    @GET("/api/donations")
    Call<List<Donation>> getAllDonations();

    @GET("/api/candidates")
    Call<List<Candidate>> getAllCandidates();

    @POST("/api/candidates/{id}/donations")
    Call<Donation> createDonation(@Path("id") String id, @Body Donation donation);
}
```

```
module.exports = [
  { method: 'GET', path: '/api/users', config: UsersApi.find },
  { method: 'GET', path: '/api/users/{id}', config: UsersApi.findOne },
  { method: 'POST', path: '/api/users', config: UsersApi.create },
  { method: 'GET', path: '/api/donations', config: DonationsApi.findAllDonations },
  { method: 'GET', path: '/api/candidates', config: CandidatesApi.find },
  { method: 'GET', path: '/api/candidates/{id}/donations', config: DonationsApi.findDonations },
];
```


Creating the Client Service Object

- ‘service’
created using
the Gson
JSON library
- This object
can be used
to invoke
REST API
directly in java

```
public class DonationServiceAPI
{
    DonationService service;

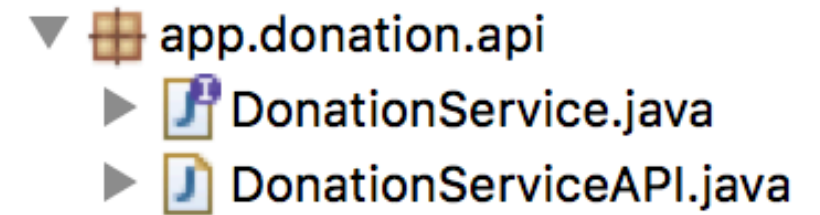
    public DonationServiceAPI(String url)
    {
        String service_url = url;

        Gson gson = new GsonBuilder().create();

        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(service_url)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build();
        service = retrofit.create(DonationService.class);
    }

    ...
}
```

DonationServiceAPI



```
public class DonationServiceAPI
{
    DonationService service;

    public DonationServiceAPI(String url)
    {
        String service_url = url;

        Gson gson = new GsonBuilder().create();

        Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(service_url)
            .addConverterFactory(GsonConverterFactory.create(gson))
            .build();
        service = retrofit.create(DonationService.class);
    }

    public List<User> getUsers() throws Exception
    {
        Call<List<User>> call = (Call<List<User>>) service.getAllUsers();
        Response<List<User>> users = call.execute();
        return users.body();
    }

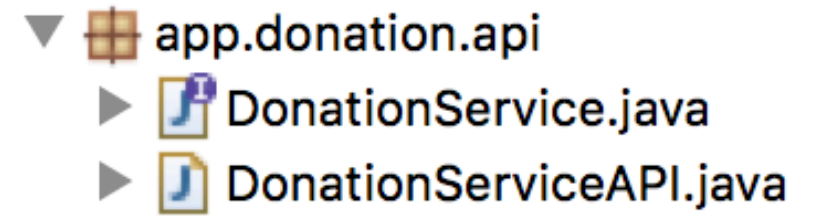
    public List<Candidate> getAllCandidates() throws Exception
    {
        Call<List<Candidate>> call = (Call<List<Candidate>>) service.getAllCandidates();
        Response<List<Candidate>> candidates = call.execute();
        return candidates.body();
    }

    public List<Donation> getAllDonations() throws Exception
    {
        Call<List<Donation>> call = (Call<List<Donation>>) service.getAllDonations();
        Response<List<Donation>> donations = call.execute();
        return donations.body();
    }

    ...
}
```

- Assemble & a HTTP request
- Translate any data from Java to JSON format
- Dispatch the request
- Wait for the response
- Translate response from JSON to Java

DonationServiceAPI

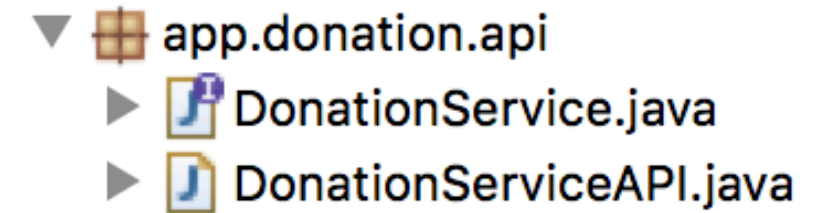


```
public List<Candidate> getAllCandidates() throws Exception
{
    Call<List<Candidate>>
        call = (Call<List<Candidate>>) service.getAllCandidates();
    Response<List<Candidate>> candidates = call.execute();
    return candidates.body();
}
```

```
public interface DonationService
{
    ...
    @GET("/api/candidates")
    Call<List<Candidate>> getAllCandidates();
    ...
}
```

- Assemble a HTTP request
- Dispatch the request & Wait for the response
- Translate response from JSON to Java

DonationServiceAPI



```
public User createUser(User newUser) throws Exception
{
    Call<User> call = (Call<User>) service.createUser(newUser);
    Response<User> returnedUser = call.execute();
    return returnedUser.body();
}
```

```
public interface DonationService
{
    ...

    @POST("/api/users")
    Call<User> createUser(@Body User User);

    ...
}
```

Using the API

```
DonationServiceAPI service = new DonationServiceAPI("http://localhost:4000");  
List<Candidate> candidates = service.getAllCandidates();  
List<User> users = service.getUsers();  
List<Donation> donations = service.getAllDonations();
```

- Simple Java API to retrieve
 - Candidates
 - Users
 - Donations
- into local model objects

Rendering Java Objects in console

- Convenient 'prettyprint' of java objects to familiar json notation

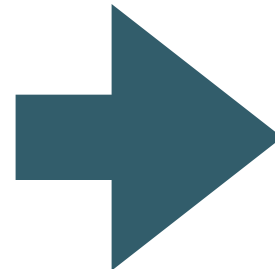
```
static Gson gson = new GsonBuilder().setPrettyPrinting().create();

public static void println(Object o)
{
    System.out.println(gson.toJson(o));
}

...
List<Donation> donations = service.getAllDonations();
println(donations);
...
```

```
public class Donation
{
    public String _id;
    public int amount;
    public String method;

    public Donation (int amount, String method)
    {
        this.amount = amount;
        this.method = method;
    }
}
```



```
[
  {
    "_id": "5815a5b10dc72a79a540cf1d",
    "amount": 40,
    "method": "paypal"
  },
  {
    "_id": "5815a5b10dc72a79a540cf1e",
    "amount": 90,
    "method": "direct"
  },
  {
    "_id": "5815a5b10dc72a79a540cf1f",
    "amount": 430,
    "method": "paypal"
  }
]
```

Object References?

```
const donationSchema = mongoose.Schema({
  amount: Number,
  method: String,
  donor: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
  },
  candidate: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Candidate',
  },
});
```

- Java Version does not include donor & candidate references
- Retrofit simply ignore these fields, and generates incomplete Donation objects in Java

```
public class Donation
{
    public String _id;
    public int amount;
    public String method;

    public Donation (int amount, String method)
    {
        this.amount = amount;
        this.method = method;
    }
}
```

```
[
  {
    "_id": "5815a5b10dc72a79a540cf1d",
    "amount": 40,
    "method": "paypal"
  },
  {
    "_id": "5815a5b10dc72a79a540cf1e",
    "amount": 90,
    "method": "direct"
  },
  {
    "_id": "5815a5b10dc72a79a540cf1f",
    "amount": 430,
    "method": "paypal"
  }
]
```


Object References - Complete Donation

```
const donationSchema = mongoose.Schema({  
  amount: Number,  
  method: String,  
  donor: {  
    type: mongoose.Schema.Types.ObjectId,  
    ref: 'User',  
  },  
  candidate: {  
    type: mongoose.Schema.Types.ObjectId,  
    ref: 'Candidate',  
  },  
});
```

```
public class DonationComplete  
{  
  public String _id;  
  public int amount;  
  public String method;  
  public User donor;  
  public Candidate candidate;  
  
  public DonationComplete (int amount, String method)  
  {  
    this.amount = amount;  
    this.method = method;  
  }  
}
```

- Merely by including references to other Java models, retrofit will generate and populate the references in Java objects

```
[  
  {  
    "_id": "5815a5b10dc72a79a540cf1d",  
    "amount": 40,  
    "method": "paypal",  
    "donor": {  
      "_id": "5815a5b00dc72a79a540cf1a",  
      "firstName": "Bart",  
      "lastName": "Simpson",  
      "email": "bart@simpson.com",  
      "password": "secret"  
    },  
    "candidate": {  
      "_id": "5815a5b00dc72a79a540cf1b",  
      "firstName": "Lisa",  
      "lastName": "Simpson",  
      "office": "President"  
    }  
  },  
],
```