Mobile Application Development

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

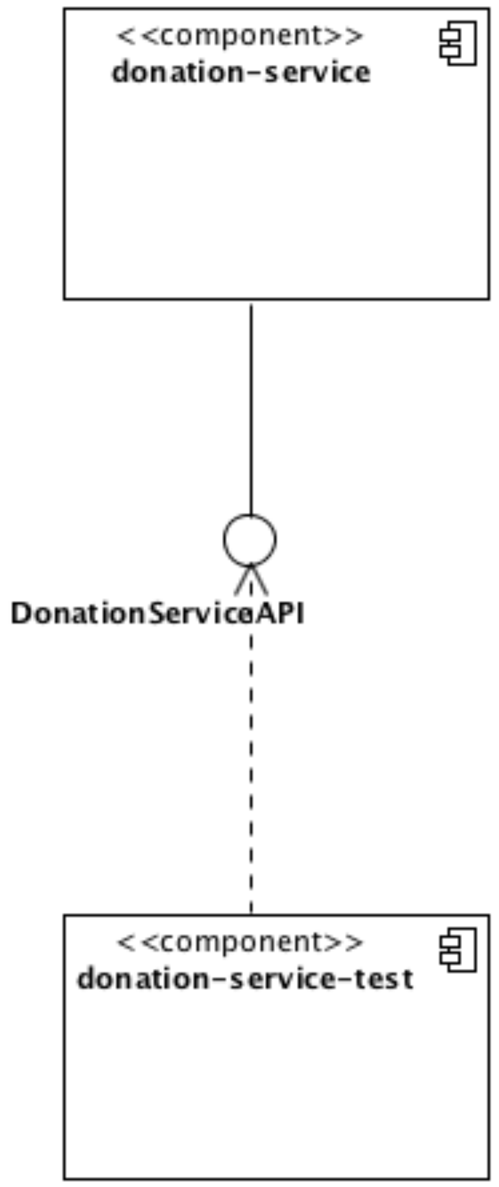
Department of Computing, Maths & Physics
Waterford Institute of Technology

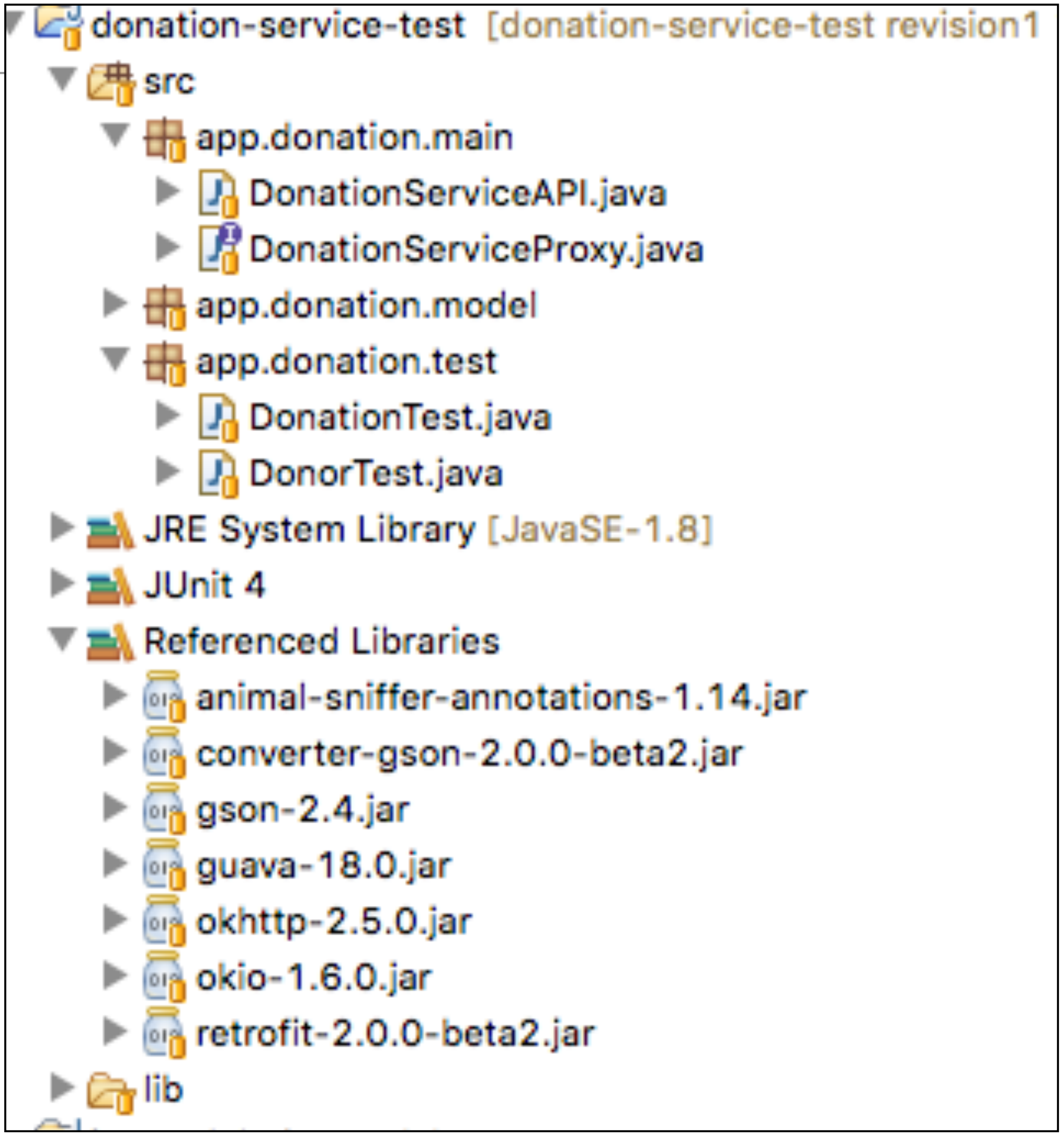
http://www.wit.ie

http://elearning.wit.ie

donation-service-test

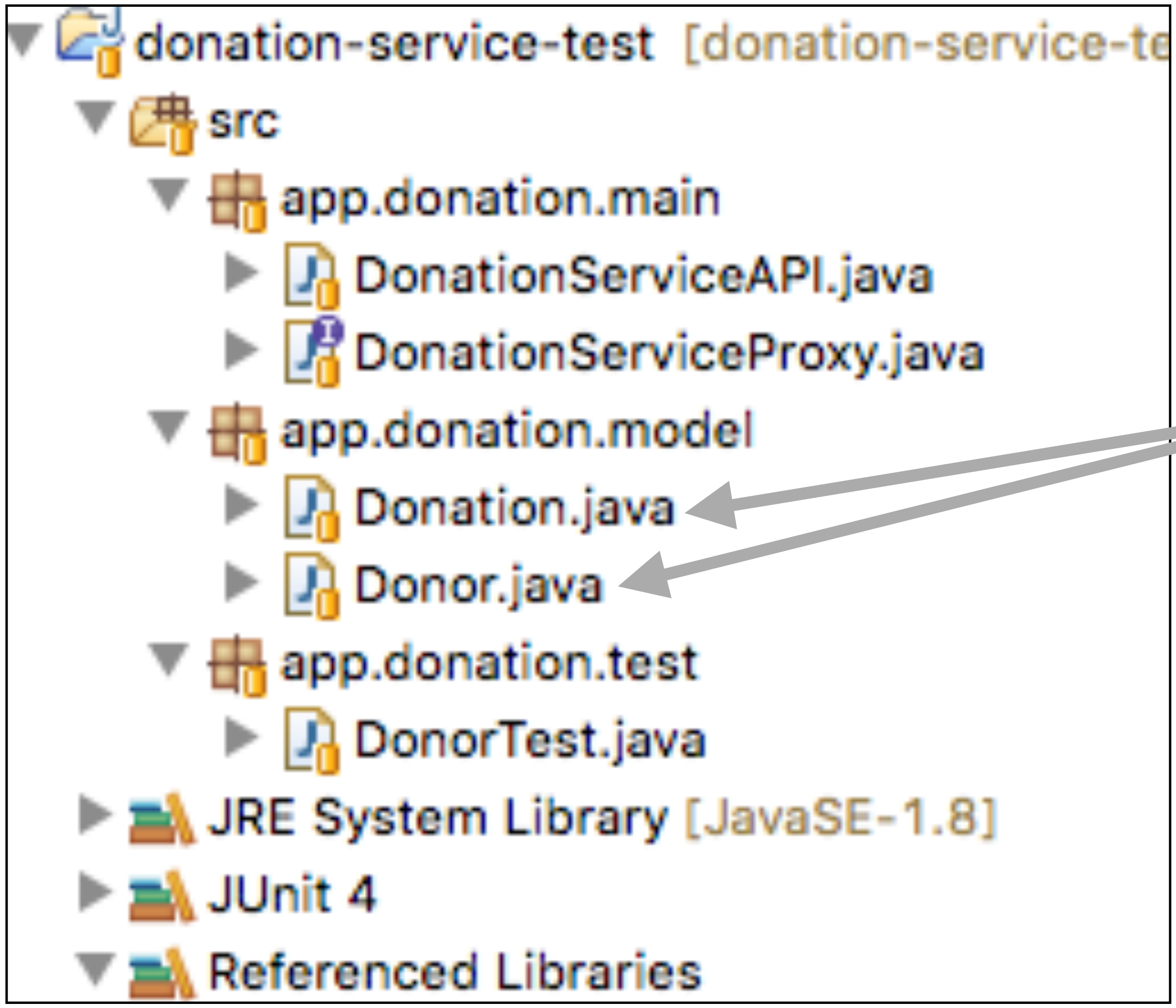
donation-service-test





```
▼ donation-service-test [donation-service-test revision1
  ▼ src
    ▼ app.donation.main
      ▶ DonationServiceAPI.java
      ▶ DonationServiceProxy.java
    ▶ app.donation.model
    ▼ app.donation.test
      ▶ DonationTest.java
      ▶ DonorTest.java
  ▶ JRE System Library [JavaSE-1.8]
  ▶ JUnit 4
  ▼ Referenced Libraries
    ▶ animal-sniffer-annotations-1.14.jar
    ▶ converter-gson-2.0.0-beta2.jar
    ▶ gson-2.4.jar
    ▶ guava-18.0.jar
    ▶ okhttp-2.5.0.jar
    ▶ okio-1.6.0.jar
    ▶ retrofit-2.0.0-beta2.jar
  ▶ lib
```

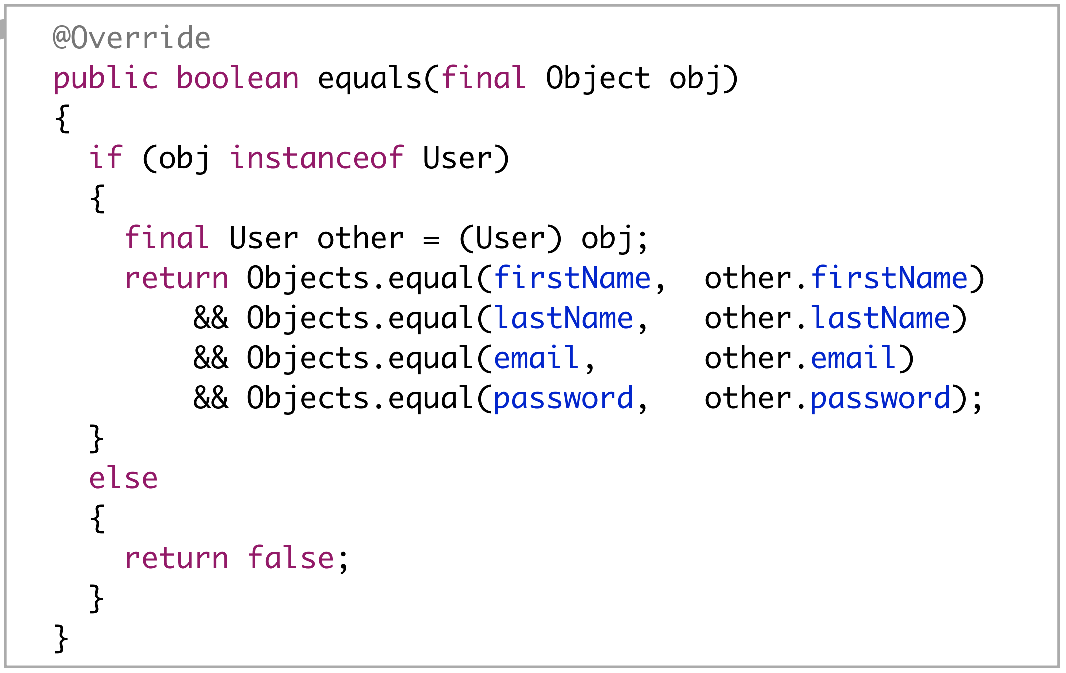
donation-service-test



donation-service-test [donation-service-te

- src
  - app.donation.main
    - DonationServiceAPI.java
    - DonationServiceProxy.java
  - app.donation.model
    - Donation.java
    - Donor.java
  - app.donation.test
    - DonorTest.java
- JRE System Library [JavaSE-1.8]
- JUnit 4
- Referenced Libraries

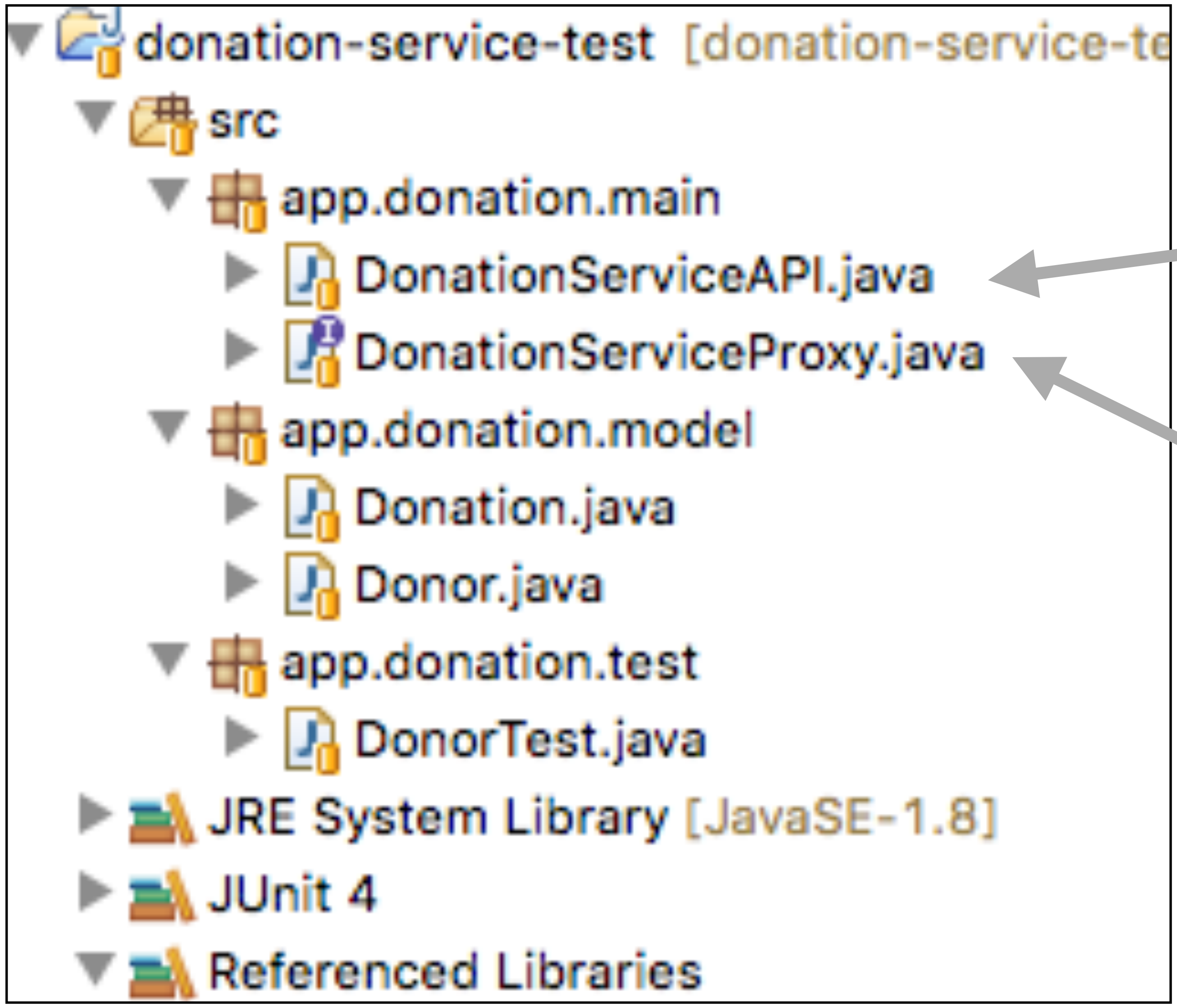
- Adapted from play versions to include equals methods



```
@Override
public boolean equals(final Object obj)
{
  if (obj instanceof User)
  {
    final User other = (User) obj;
    return Objects.equal(firstName,  other.firstName)
        && Objects.equal(lastName,   other.lastName)
        && Objects.equal(email,      other.email)
        && Objects.equal(password,   other.password);
  }
  else
  {
    return false;
  }
}
```

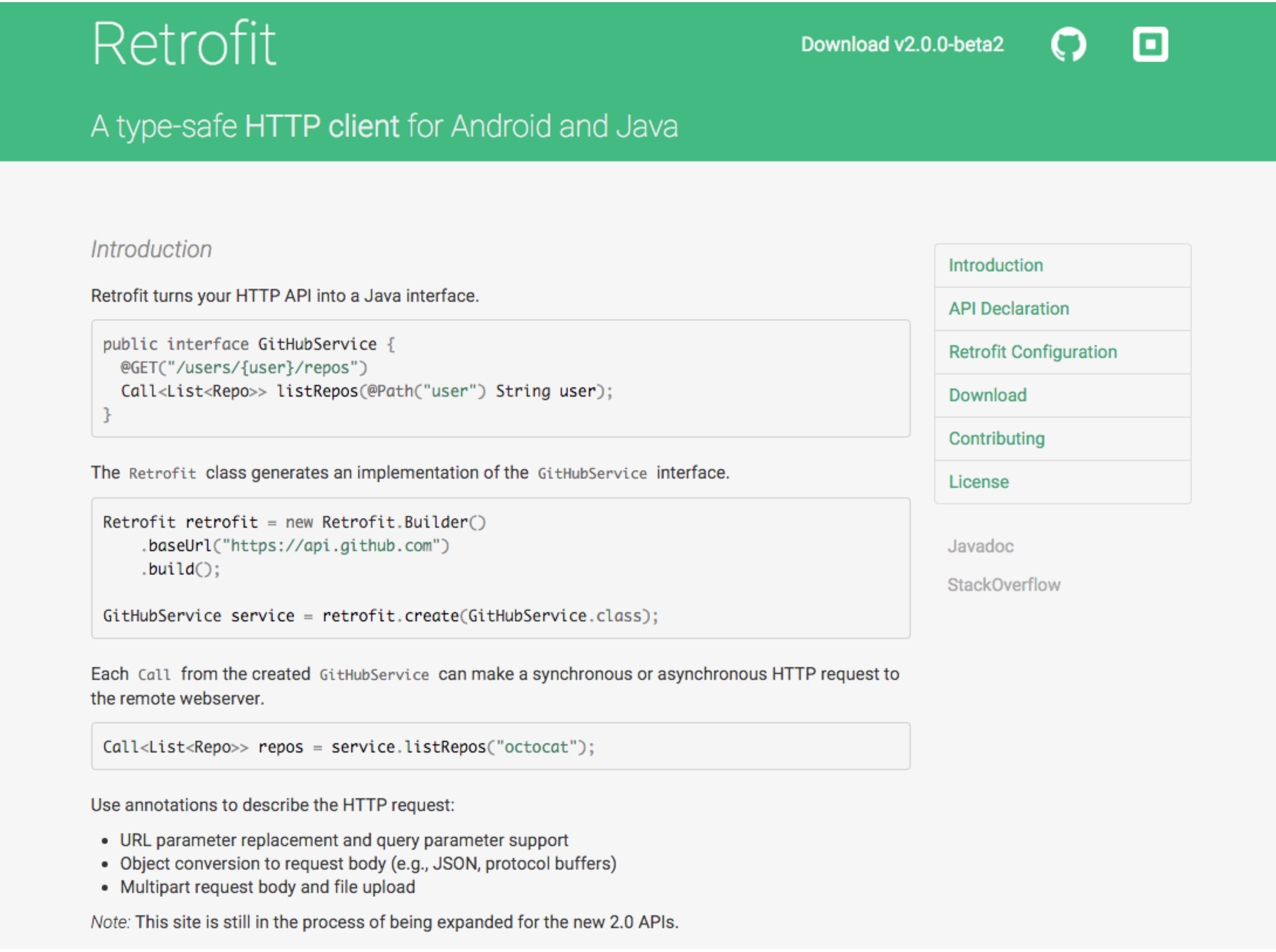
- These utility methods greatly simplify tests

donation-service-test



donation-service-test [donation-service-te
- src
  - app.donation.main
    - DonationServiceAPI.java
    - DonationServiceProxy.java
  - app.donation.model
    - Donation.java
    - Donor.java
  - app.donation.test
    - DonorTest.java
- JRE System Library [JavaSE-1.8]
- JUnit 4
- Referenced Libraries

- Wrappers to deliver a client side API.
- i.e. These class will be responsible for composing the HTTP Requests and sending them to the play service



Retrofit

Download v2.0.0-beta2

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

Note: This site is still in the process of being expanded for the new 2.0 APIs.

Javadoc

StackOverflow

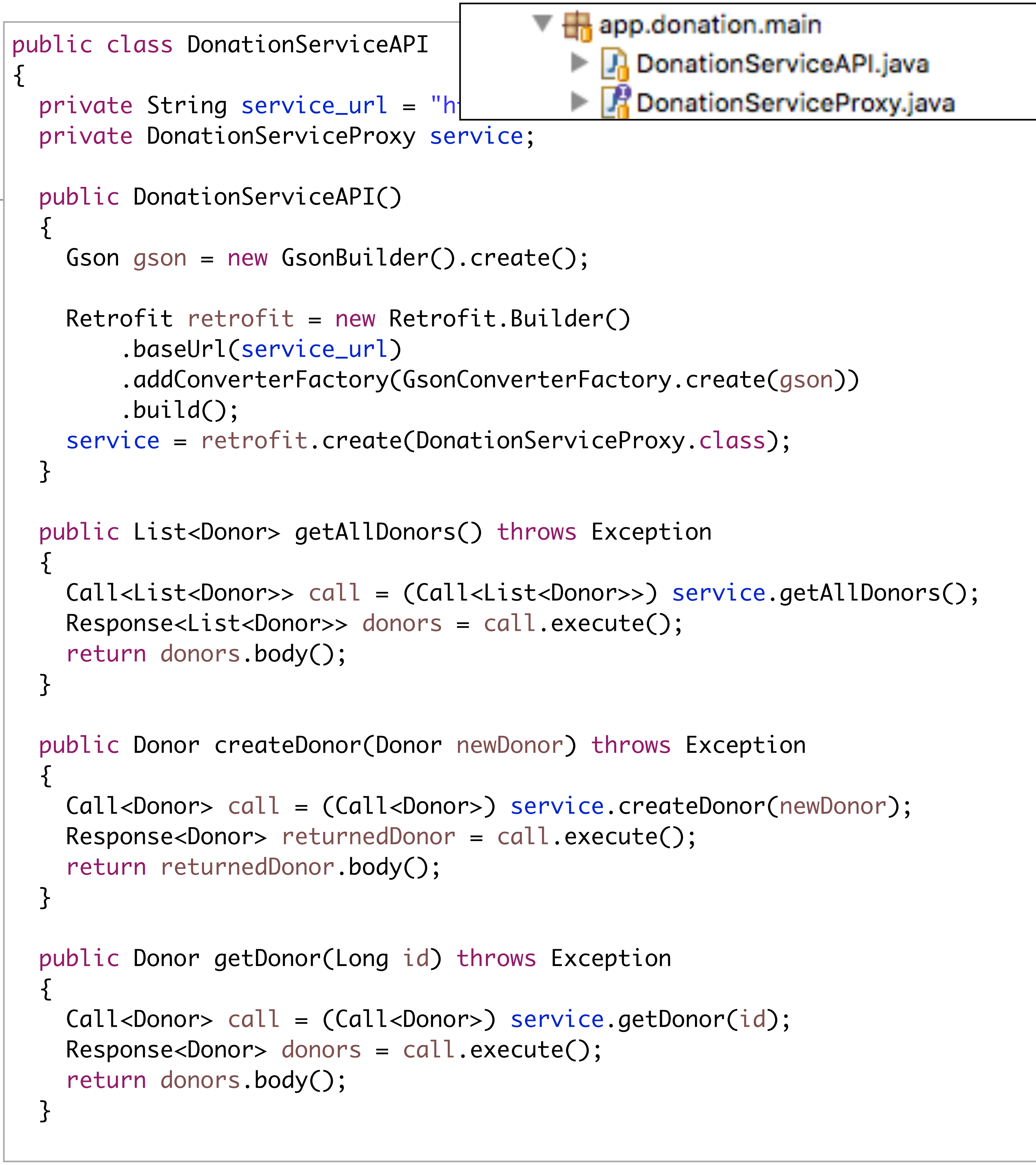
GET	/api/donors	DonorsAPI.getAllDonors
GET	/api/donors/{id}	DonorsAPI.getDonor
POST	/api/donors	DonorsAPI.createDonor
DELETE	/api/donors/{id}	DonorsAPI.deleteDonor
DELETE	/api/donors	DonorsAPI.deleteAllDonors
GET	/api/donations	DonationsAPI.getAllDonations
DELETE	/api/donations	DonationsAPI.deleteAllDonatio
GET	/api/donors/{id}/donations	DonationsAPI.getDonations
GET	/api/donors/{id}/donations/{donationId}	DonationsAPI.getDonation
POST	/api/donors/{id}/donations	DonationsAPI.createDonation
DELETE	/api/donors/{id}/donations/{donationId}	DonationsAPI.deleteDonation

```java
public interface DonationServiceProxy
{
  @GET("/api/donors")
  Call<List<Donor>> getAllDonors();

  @GET("/api/donors/{id}")
  Call<Donor> getDonor(@Path("id") Long id);

  @POST("/api/donors")
  Call<Donor> createDonor(@Body Donor Donor);

  @DELETE("/api/donors/{id}")
  Call<Donor> deleteDonor(@Path("id") Long id);

  @DELETE("/api/donors")
  Call<String> deleteAllDonors();

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

  @DELETE("/api/donations")
  Call<String> deleteAllDonations();

  @GET("/api/donors/{id}/donations")
  Call<List<Donation>> getDonations(@Path("id") Long id);

  @GET("/api/donors/{id}/donations/{donationId}")
  Call<Donation> getDonation(@Path("id") Long id, @Path("id") Long donationId);

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

  @DELETE("/api/donors/{id}/donatinos/{donationId}")
  Call<Donation> deleteDonation(@Path("id") Long id, @Path("id") Long donationId);
}
```

DonationServiceProxy

DonationServiceAPI

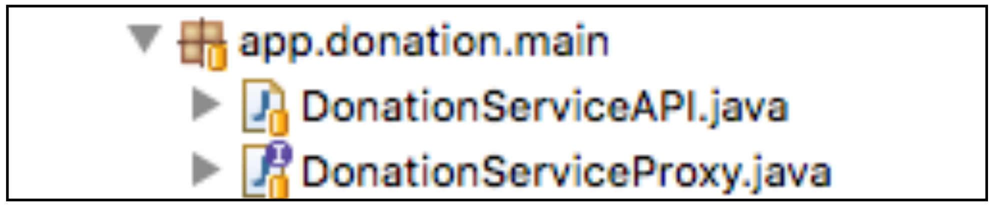
- Assemble & a HTTP request
- Translate any data from Java to JSON format
- Dispatch the request
- Wait for the response
- Translate response from JSON to Java



```
▼ app.donation.main
    ▶ DonationServiceAPI.java
    ▶ DonationServiceProxy.java
```

```java
public class DonationServiceAPI
{
  private String service_url = "h
  private DonationServiceProxy service;

  public DonationServiceAPI()
  {
    Gson gson = new GsonBuilder().create();

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

  public List<Donor> getAllDonors() throws Exception
  {
    Call<List<Donor>> call = (Call<List<Donor>>) service.getAllDonors();
    Response<List<Donor>> donors = call.execute();
    return donors.body();
  }

  public Donor createDonor(Donor newDonor) throws Exception
  {
    Call<Donor> call = (Call<Donor>) service.createDonor(newDonor);
    Response<Donor> returnedDonor = call.execute();
    return returnedDonor.body();
  }

  public Donor getDonor(Long id) throws Exception
  {
    Call<Donor> call = (Call<Donor>) service.getDonor(id);
    Response<Donor> donors = call.execute();
    return donors.body();
  }
```

DonationServiceAPI

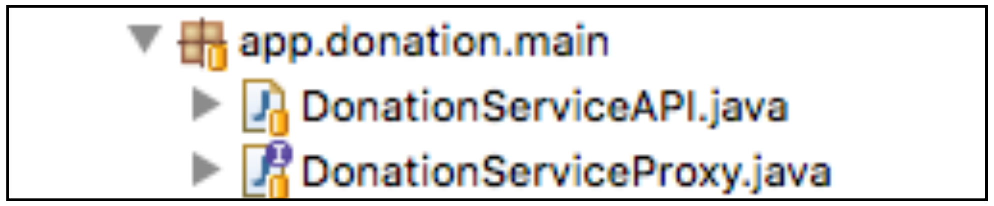

app.donation.main
DonationServiceAPI.java
DonationServiceProxy.java

```
public List<Donor> getAllDonors() throws Exception
{
  Call<List<Donor>> call
     = (Call<List<Donor>>) service.getAllDonors();

  Response<List<Donor>> donors = call.execute();

  return donors.body();
}
```

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

```
public interface DonationServiceProxy
{
  @GET("/api/donors")
  Call<List<Donor>> getAllDonors();
}
```

DonationServiceAPI

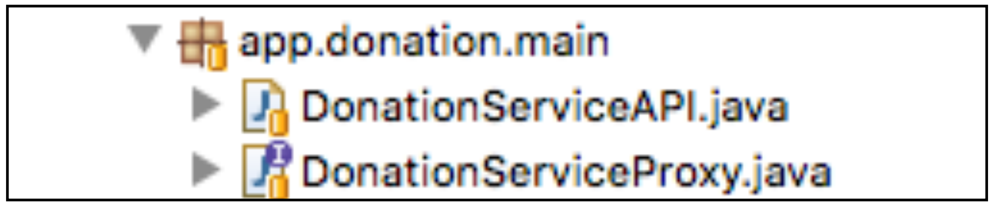

app.donation.main
DonationServiceAPI.java
DonationServiceProxy.java

```
public Donor createDonor(Donor newDonor)
{
  Call<Donor> call
    = (Call<Donor>) service.createDonor(newDonor);

  Response<Donor> returnedDonor = call.execute();

  return returnedDonor.body();
}
```

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

```
public interface DonationServiceProxy
{
  @POST("/api/donors")
  Call<Donor> createDonor(@Body Donor Donor);
}
```

DonationServiceAPI



```
▼ app.donation.main
   ▶ DonationServiceAPI.java
   ▶ DonationServiceProxy.java
```

```java
public Donor getDonor(Long id)
{
  Call<Donor> call
    = (Call<Donor>) service.getDonor(id);

  Response<Donor> donors = call.execute();

  return donors.body();
}
```

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

```java
public interface DonationServiceProxy
{
  @GET("/api/donors/{id}")
  Call<Donor> getDonor(@Path("id") Long id);
}
```

Test POST /api/donors

```
public class DonorTest
{
  private DonationServiceAPI donationServiceAPI = new DonationServiceAPI();

  @Test
  public void testCreate() throws Exception
  {
    Donor john = new Donor("john", "doe", "john@doe.com", "secret");
    Donor donor = donationServiceAPI.createDonor(john);
    assertEquals(john, donor);

    int code = donationServiceAPI.deleteDonor(donor.id);
    assertEquals (200, code);
  }
```

- Create a user object locally
- Use this to request a user be created in the donation-service
- Verify that the returned user (from the getUserRequest) contains the same values as the local object we used to create the User
- Clean up by deleting the user (from the service)

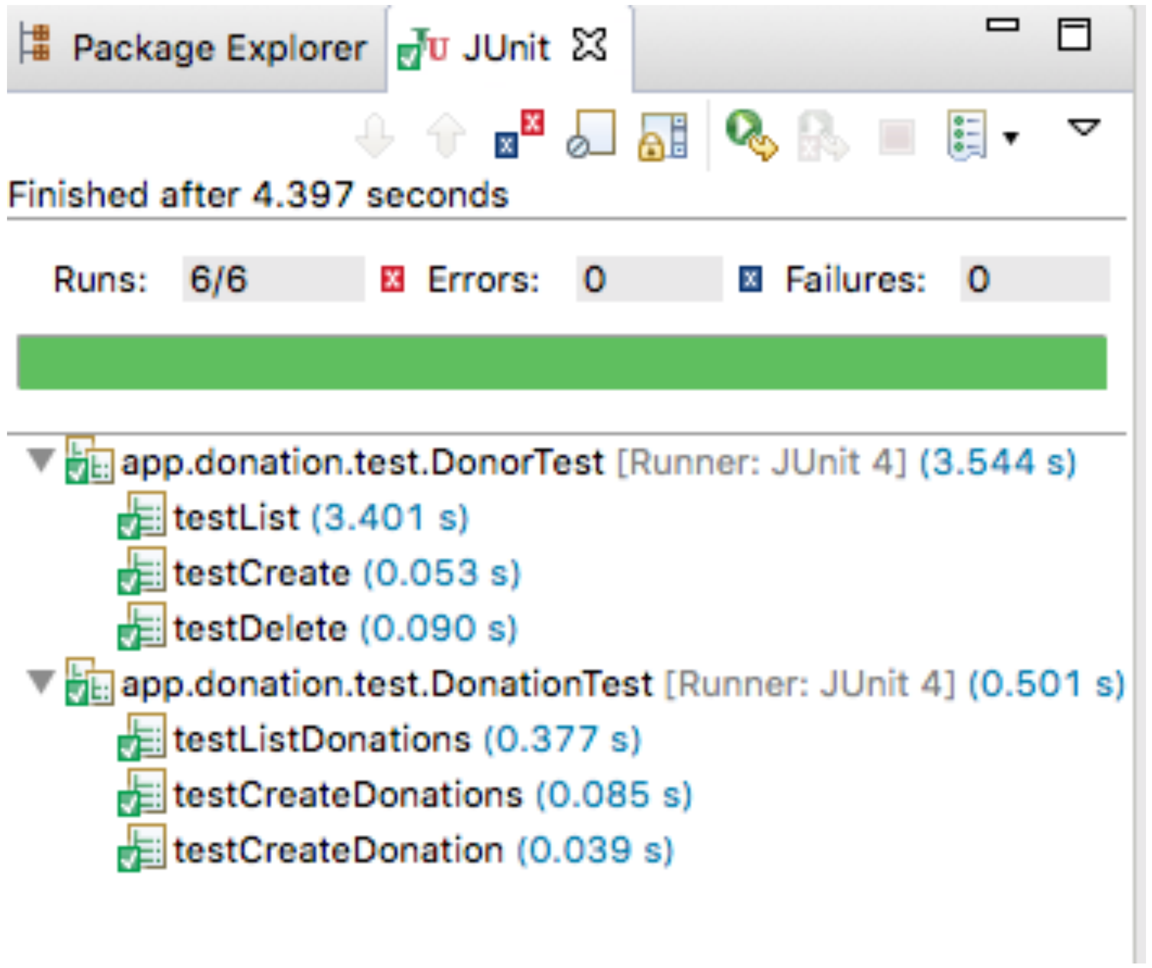
Test GET /api/donors/{id}

```
@Test
public void testGet () throws Exception
{
  Donor homer = new Donor ("homer",  "simpson", "homer@simpson.com",  "secret");
  Donor donor = DonationServiceAPI.createDonor(homer);

  User searchDonor = DonationServiceAPI.getDonor(donor.id);
  assertEquals (homer, searchDonor);
  DonationServiceAPI.deleteDonor(searchDonor);
}
```

- Having created a user, request the user by its ID, and verify that the returned user contains the expected fields

Why This Level of Tests?

- Models stored in databases using JPA need to be throughly tested.
- Specifically - complete tests for:
  - create
  - read
  - update
  - delete
- are essential.
- This is especially the case when Models are involved in relationships (OneToMany, ManyToOne etc..



```
Finished after 4.397 seconds

Runs: 6/6    Errors: 0    Failures: 0

▼ app.donation.test.DonorTest [Runner: JUnit 4] (3.544 s)
    testList (3.401 s)
    testCreate (0.053 s)
    testDelete (0.090 s)
▼ app.donation.test.DonationTest [Runner: JUnit 4] (0.501 s)
    testListDonations (0.377 s)
    testCreateDonations (0.085 s)
    testCreateDonation (0.039 s)
```

More Considered UserTest

- “Fixture” created and deleted in setup/teardown
- This fixture is a useful set of test data for many of the tests

```java
public class DonorTest
{
  static Donor donorArray [] =
  {
    new Donor ("homer",  "simpson", "homer@simpson.com",  "secret"),
    new Donor ("lisa",   "simpson", "lisa@simpson.com",   "secret"),
    new Donor ("maggie", "simpson", "maggie@simpson.com", "secret"),
    new Donor ("bart",   "simpson", "bart@simpson.com",   "secret"),
    new Donor ("marge",  "simpson", "marge@simpson.com",  "secret"),
  };
  List <Donor> donorList = new ArrayList<>();

  private DonationServiceAPI donationServiceAPI = new DonationServiceAPI();

  @Before
  public void setup() throws Exception
  {
    for (Donor donor : donorArray)
    {
      Donor returned = donationServiceAPI.createDonor(donor);
      donorList.add(returned);
    }
  }

  @After
  public void teardown() throws Exception
  {
     donationServiceAPI.deleteAllDonors();
  }
```

Tests

Because a useful fixture has been set up, these tests can then be more considered, concise and through

```
@Test
public void testCreate () throws Exception
{
  assertEquals (donorArray.length, donorList.size());
  for (int i=0; i<donorArray.length; i++)
  {
    assertEquals(donorList.get(i), donorArray[i]);
  }
}

@Test
public void testList() throws Exception
{
  List<Donor> list = donationServiceAPI.getAllDonors();
  assertTrue (list.containsAll(donorList));
}

@Test
public void testDelete () throws Exception
{
  List<Donor> list1 = donationServiceAPI.getAllDonors();

  Donor testdonor = new Donor("mark",  "simpson", "marge@simpson.com",  "secret");
  Donor returnedDonor = donationServiceAPI.createDonor(testdonor);

  List<Donor> list2 = donationServiceAPI.getAllDonors();
  assertEquals (list1.size()+1, list2.size());

  int code = donationServiceAPI.deleteDonor(returnedDonor.id);
  assertEquals (200, code);

  List<Donor> list3 = donationServiceAPI.getAllDonors();
  assertEquals (list1.size(), list3.size());
}
```

DonationTest

```
public class DonationTest
{
  private Donor marge =  new Donor ("marge",  "simpson", "homer@simpson.com",  "secret");

  private DonationServiceAPI donationServiceAPI = new DonationServiceAPI();

  @Before
  public void setup() throws Exception
  {
    marge = donationServiceAPI.createDonor(marge);
  }

  @After
  public void teardown() throws Exception
  {
    donationServiceAPI.deleteDonor(marge.id);
  }

  @Test
  public void testCreateDonation () throws Exception
  {
    Donation donation = new Donation (123, "cash");
    Donation returnedDonation = donationServiceAPI.createDonation(marge.id, donation);
    assertEquals (donation, returnedDonation);

    donationServiceAPI.deleteDonation(marge.id, returnedDonation.id);
  }
  @Test
  public void testCreateDonations () throws Exception
  {
    Donation donation1 = new Donation (123, "cash");
    Donation donation2 = new Donation (450, "cash");
    Donation donation3 = new Donation (43,  "paypal");

    Donation returnedDonation1 = donationServiceAPI.createDonation(marge.id, donation1);
    Donation returnedDonation2 = donationServiceAPI.createDonation(marge.id, donation2);
    Donation returnedDonation3 = donationServiceAPI.createDonation(marge.id, donation3);

    assertEquals(donation1, returnedDonation1);
    assertEquals(donation2, returnedDonation2);
    assertEquals(donation3, returnedDonation3);

    donationServiceAPI.deleteDonation(marge.id, returnedDonation1.id);
    donationServiceAPI.deleteDonation(marge.id, returnedDonation2.id);
    donationServiceAPI.deleteDonation(marge.id, returnedDonation3.id);
  }

  @Test
  public void testListDonations () throws Exception
  {
    Donation donation1 = new Donation (123, "cash");
    Donation donation2 = new Donation (450, "cash");
    Donation donation3 = new Donation (43,  "paypal");

    donationServiceAPI.createDonation(marge.id, donation1);
    donationServiceAPI.createDonation(marge.id, donation2);
    donationServiceAPI.createDonation(marge.id, donation3);

    List<Donation> donations = donationServiceAPI.getDonations(marge.id);
    assertEquals (3, donations.size());

    assertTrue(donations.contains(donation1));
    assertTrue(donations.contains(donation2));
    assertTrue(donations.contains(donation3));

    donationServiceAPI.deleteDonation(marge.id, donations.get(0).id);
    donationServiceAPI.deleteDonation(marge.id, donations.get(1).id);
    donationServiceAPI.deleteDonation(marge.id, donations.get(2).id);
  }
}
```