Mobile Application Development

Produced by

Eamonn de Leastar (edeleastar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

http://www.wit.ie

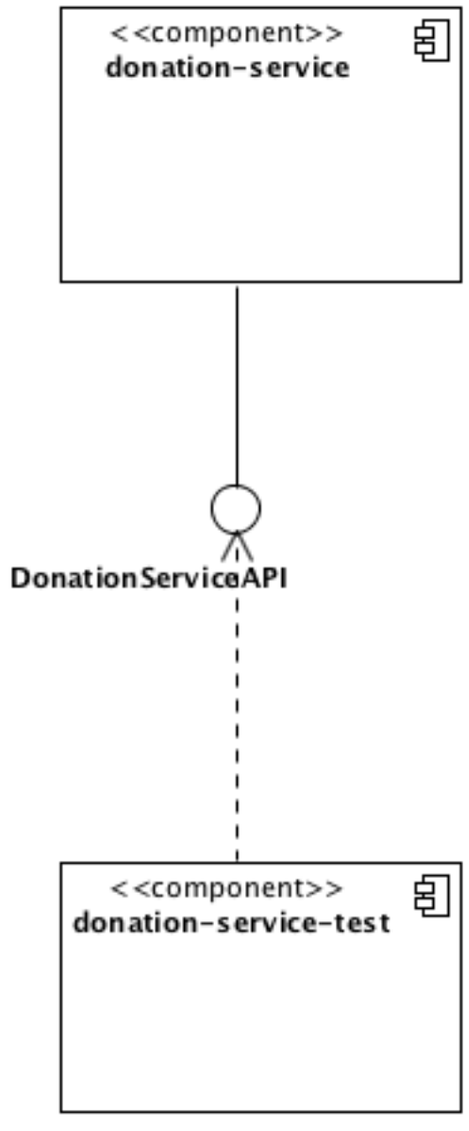
http://elearning.wit.ie

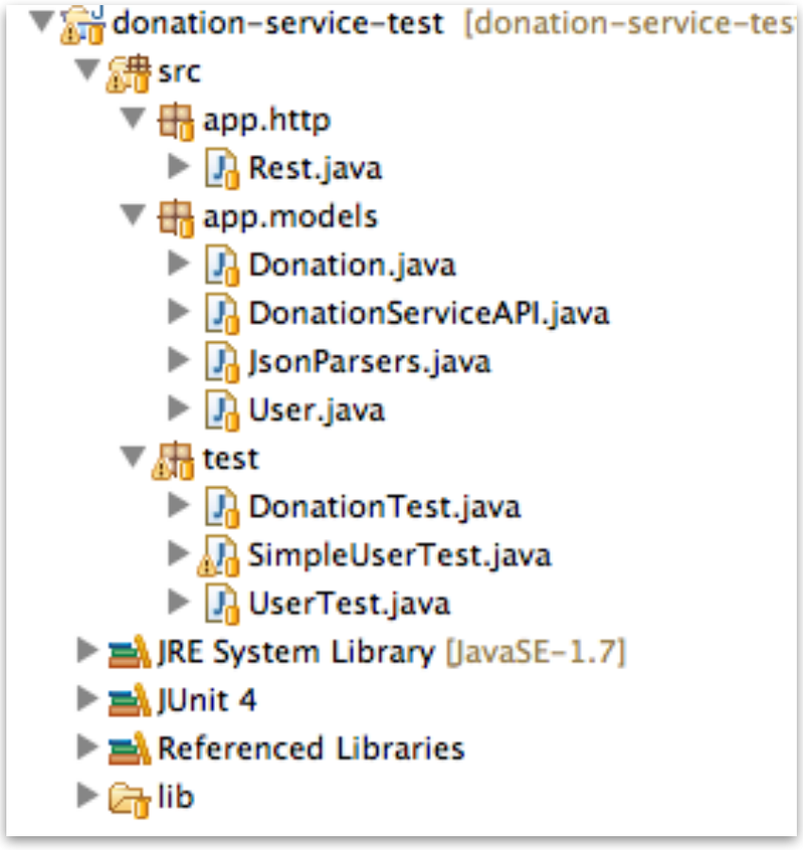
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit

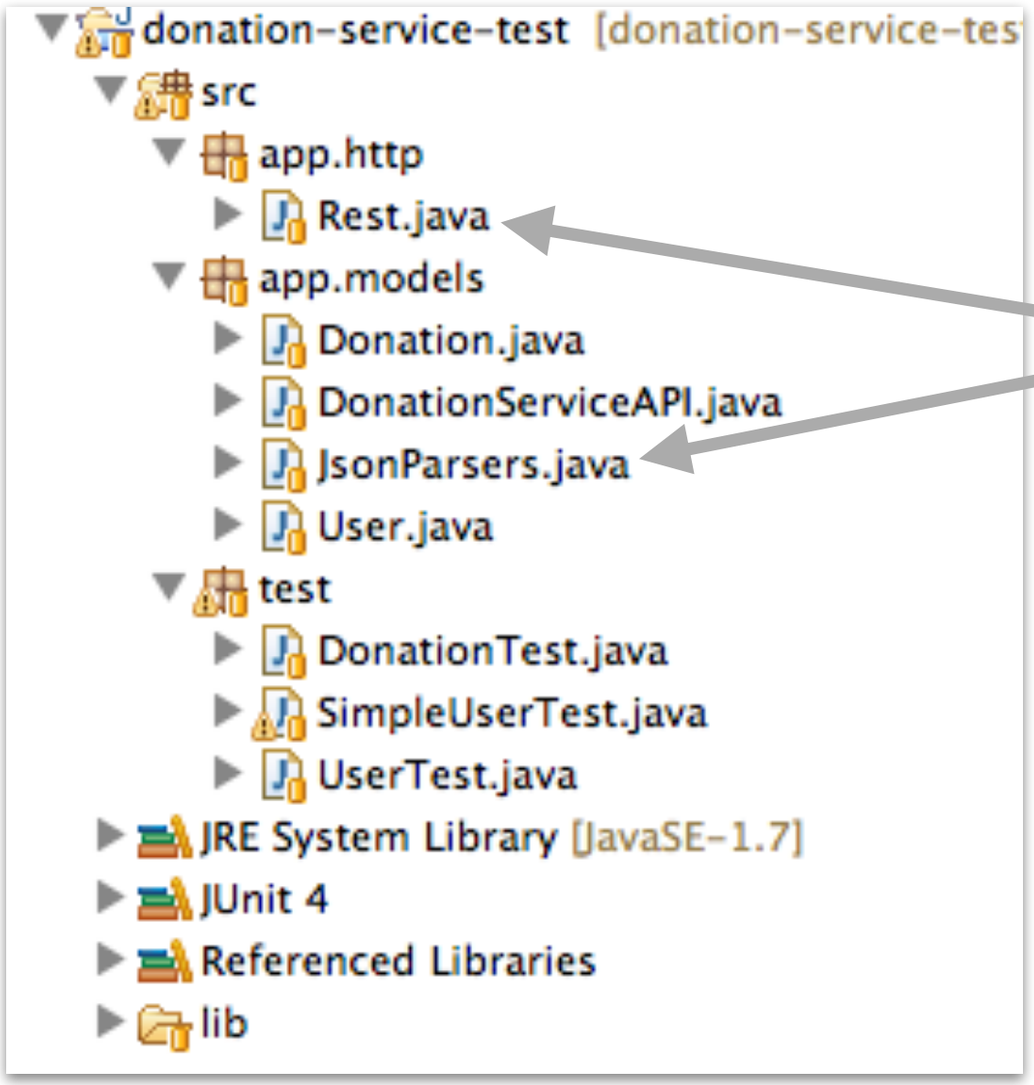
donation-service-test

donation-service-test



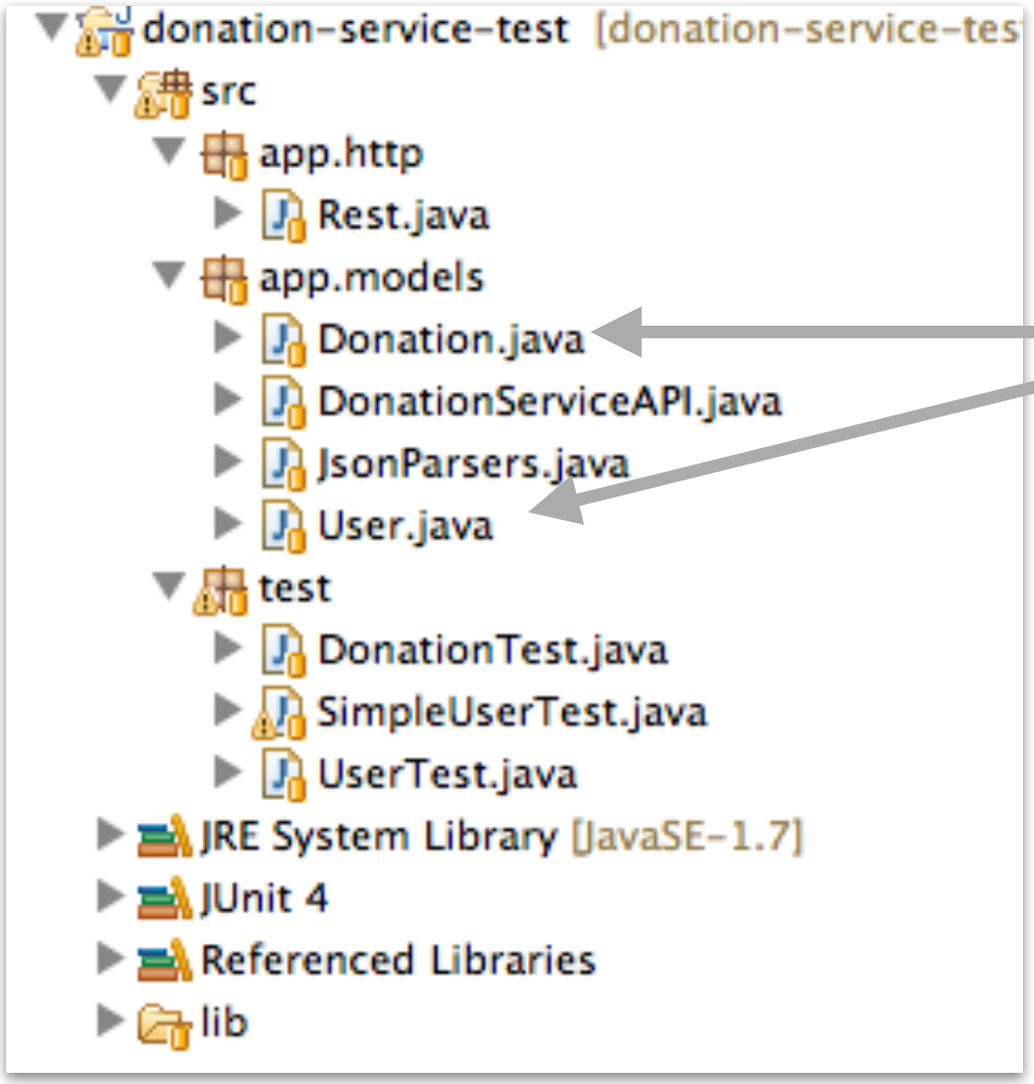


donation-service-test



- Unchanged from donation-android versions
- Testing these classes independently increases our confidence in them significantly

donation-service-test

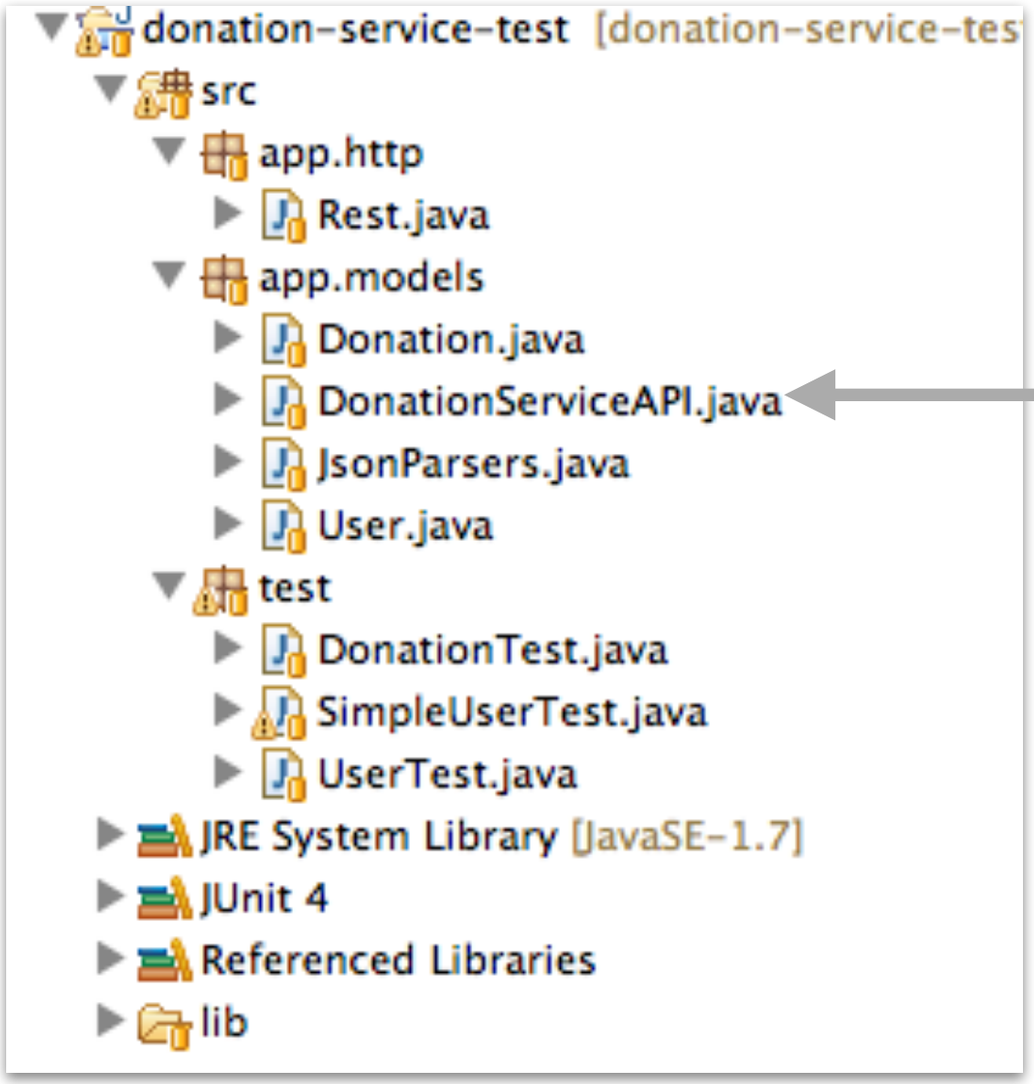


- Extended from Android versions to include equals methods

```
@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



- Simplified from donation-android version
- We do not need thread/task support in our test project
- We just wait until each request is serviced

DonationServiceAPI

- Uses the Rest class + JsonParser to provide convenient, high level interface to donation-service API

```
GET     /api/users
GET     /api/users/{id}
POST    /api/users
DELETE  /api/users/{id}

GET     /api/users/{userId}/donations
GET     /api/users/{userId}/donations/{id}
POST    /api/users/{userId}/donations
DELETE  /api/users/{userId}/donations/{id}
```

```
public class DonationServiceAPI
{
  public static List<User> getUsers() throws Exception
  {
    String response =  Rest.get("/api/users");
    List<User> userList = JsonParsers.json2Users(response);
    return userList;
  }

  public static User getUser(Long id) throws Exception
  {
    String response =  Rest.get("/api/users/" + id);
    User user = JsonParsers.json2User(response);
    return user;
  }

  public static User createUser(User user) throws Exception
  {
    String response = Rest.post ("/api/users", JsonParsers.user2Json(user));
    return JsonParsers.json2User(response);
  }

  public static void deleteUser(User user) throws Exception
  {
    Rest.delete ("/api/users/" + user.id);
  }

  public static List<Donation> getDonations(User user) throws Exception
  {
    String response =  Rest.get("/api/users/" + user.id + "/donations");
    List<Donation> donationList = JsonParsers.json2Donations(response);
    return donationList;
  }

  public static Donation createDonation(User user, Donation donation) …
  {
    String response = Rest.post ("/api/users/" + user.id + "/donations",
                                 JsonParsers.donation2Json(donation));
    return JsonParsers.json2Donation(response);
  }

  public static void deleteDonation(User user, Donation donation) …
  {
    Rest.delete ("/api/users/" + user.id + "/donations/" + donation.id);
  }
}
```

Test POST /api/users

```
public class SimpleUserTest
{
  @Test
  public void testCreate () throws Exception
  {
    User homer = new User ("homer",  "simpson", "homer@simpson.com",  "secret");
    User user  = DonationServiceAPI.createUser(homer);
    assertEquals(homer, user);
    DonationServiceAPI.deleteUser(user);
  }
}
```

- 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/users/{id}

```
@Test
public void testGet () throws Exception
{
  User homer = new User ("homer",  "simpson", "homer@simpson.com",  "secret");
  User user = DonationServiceAPI.createUser(homer);

  User searchUser = DonationServiceAPI.getUser(user.id);
  assertEquals (homer, searchUser);
  DonationServiceAPI.deleteUser(user);
}
```

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

Test GET /api/users

```
@Test
public void testList () throws Exception
{
  List<User> list1 = DonationServiceAPI.getUsers();

  User homer = new User ("homer",  "simpson", "homer@simpson.com",  "secret");
  User marge = new User ("marge",  "simpson", "homer@simpson.com",  "secret");
  User lisa  = new User ("lisa",  "simpson", "homer@simpson.com",  "secret");

  User user1 = DonationServiceAPI.createUser(homer);
  User user2 = DonationServiceAPI.createUser(marge);
  User user3 = DonationServiceAPI.createUser(lisa);

  List<User> list2 = DonationServiceAPI.getUsers();
  assertEquals (list1.size()+3, list2.size());

  DonationServiceAPI.deleteUser(user1);
  DonationServiceAPI.deleteUser(user2);
  DonationServiceAPI.deleteUser(user3);

}
```

- Create three users
- Request a list of all users
- Verify that the list is +3 users

DonationTest

`POST /api/users/{userId}/donations`

- Set up a user as a ‘fixture’
- Verify that at donation can be created

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

  @Before
  public void setup() throws Exception
  {
    marge = DonationServiceAPI.createUser(marge);
  }

  @After
  public void teardown() throws Exception
  {
    DonationServiceAPI.deleteUser(marge);
  }

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

    DonationServiceAPI.deleteDonation(marge, returnedDonation);
  }
```

ListDonations

`GET /api/users/{userId}/donations`

- Create 3 donations for a user
- Request all donations for same user
- Verify that donations are in returned list as expected

```
@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, donation1);
  DonationServiceAPI.createDonation(marge, donation2);
  DonationServiceAPI.createDonation(marge, donation3);

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

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

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

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..

Example: Donation-service v2

- This method contains a bug.
- The bug did not effect the current version of donation-android

```
public static void createDonation(Long userId, JsonElement body)
{
  User user = User.findById(userId);
  Donation donation = JsonParsers.json2Donation(body.toString());
  Donation newDonation = new Donation (donation.amount, donation.method);
  user.donations.add(donation);
  user.save();
  renderJSON (JsonParsers.donation2Json(newDonation));
}
```

- What is it?

- Inserts wrong donation into database.
- The returned donation has a ‘null’ id field

```
public static void createDonation(Long userId, JsonElement body)
{
  User user = User.findById(userId);
  Donation donation = JsonParsers.json2Donation(body.toString());
  Donation newDonation = new Donation (donation.amount, donation.method);
  user.donations.add(newDonation);
  user.save();
  renderJSON (JsonParsers.donation2Json(newDonation));
}
```

- Using the ID on the returned object will cause failures in subsequent calls
- Finding this bug on android would have been a major effort.

More Considered UserTest

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

```
public class UserTest
{
  static User userArray [] =
  {
    new User ("homer",  "simpson", "homer@simpson.com",  "secret"),
    new User ("lisa",   "simpson", "lisa@simpson.com",   "secret"),
    new User ("maggie", "simpson", "maggie@simpson.com", "secret"),
    new User ("bart",   "simpson", "bart@simpson.com",   "secret"),
    new User ("marge",  "simpson", "marge@simpson.com",  "secret"),
  };

  List <User> userList = new ArrayList<>();

  @Before
  public void setup() throws Exception
  {
    for (User user : userArray)
    {
      User returned = DonationServiceAPI.createUser(user);
      userList.add(returned);
    }
  }

  @After
  public void teardown() throws Exception
  {
    for (User user : userList)
    {
      DonationServiceAPI.deleteUser(user);
    }
  }
```

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 (userArray.length, userList.size());
  for (int i=0; i<userArray.length; i++)
  {
    assertEquals(userList.get(i), userArray[i]);
  }
}

@Test
public void testList() throws Exception
{
  List<User> list = DonationServiceAPI.getUsers();
  assertTrue (list.containsAll(userList));
}

@Test
public void testDelete () throws Exception
{
  List<User> list1 = DonationServiceAPI.getUsers();
  User testUser = new User("mark",  "simpson", "marge@simpson.com",  "secret");
  User returnedUser = DonationServiceAPI.createUser(testUser);
  List<User> list2 = DonationServiceAPI.getUsers();
  assertEquals (list1.size()+1, list2.size());
  DonationServiceAPI.deleteUser(returnedUser);
  List<User> list3 = DonationServiceAPI.getUsers();
  assertEquals (list1.size(), list3.size());
}
```