Mobile Application Development

Higher Diploma in Science in Computer Science

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

Coffeemate Application Programmer Interfaces (APIs)

APIs

- An application programming interface (API) specifies how some software components should interact with each other.

- An API often encapsulates a service, capability or system, and will ideally do so in a coherent, elegant and reliable manner.

- APIs are documented, presented and packaged for consumption by programmers.

- Programmers can then integrate new service and capabilities into their systems.

Examples - General

- Java SDK
    - collections API (java.util)
    - DateTime
    - Sockets (java.net)
- OpenGL (for graphics / Visualization / Games)
- Windows SDK

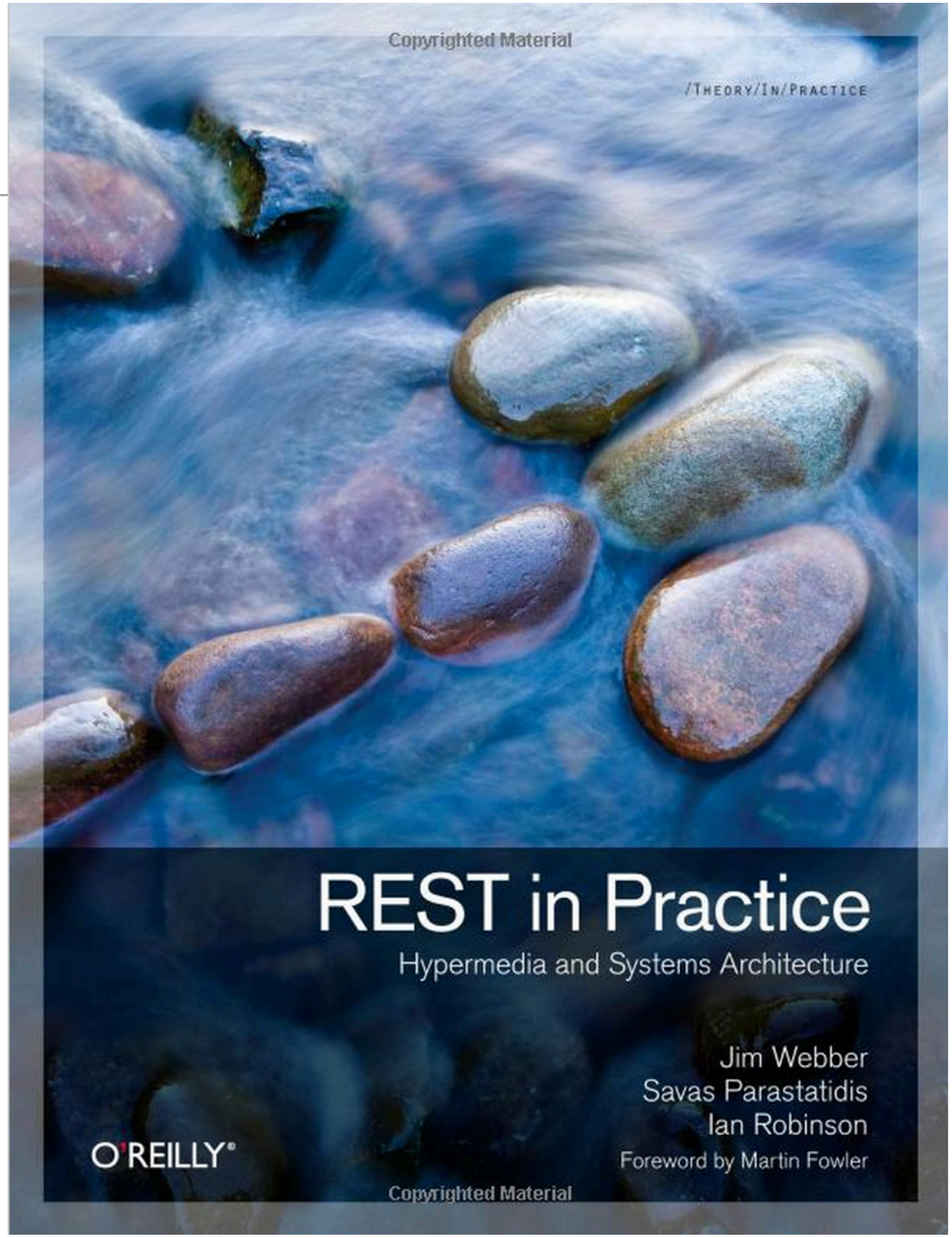
Examples 2 - REST

- Twitter API
- Google Maps
- Twillio
- Blogger.com
- App.net
- SupportFu
- Gist

- REST is an “Architectural Style” - enumerating an approach to building distributed systems.
- It embodies an approach that aims to maximize the infrastructure of http infrastructure deployed in the public internet, enabling secure, scalable distributed systems that do not require expensive, complex alternative infrastructure.

REST

Representational State Transfer (**REST**) is an architectural style that abstracts the architectural elements within a distributed hypermedia system.[1] REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.[2]REST has emerged as a predominant web API design model

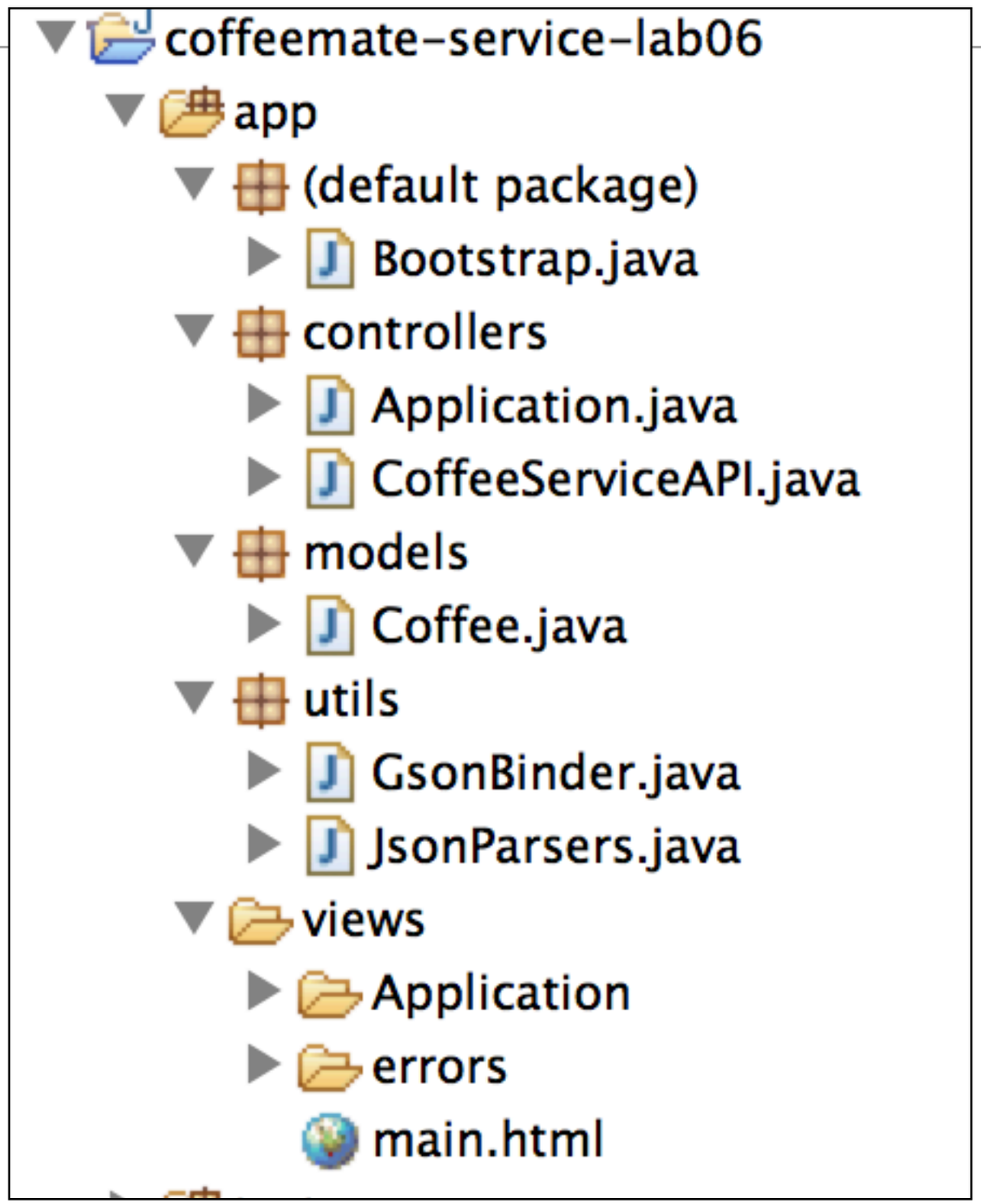


CoffeeMate

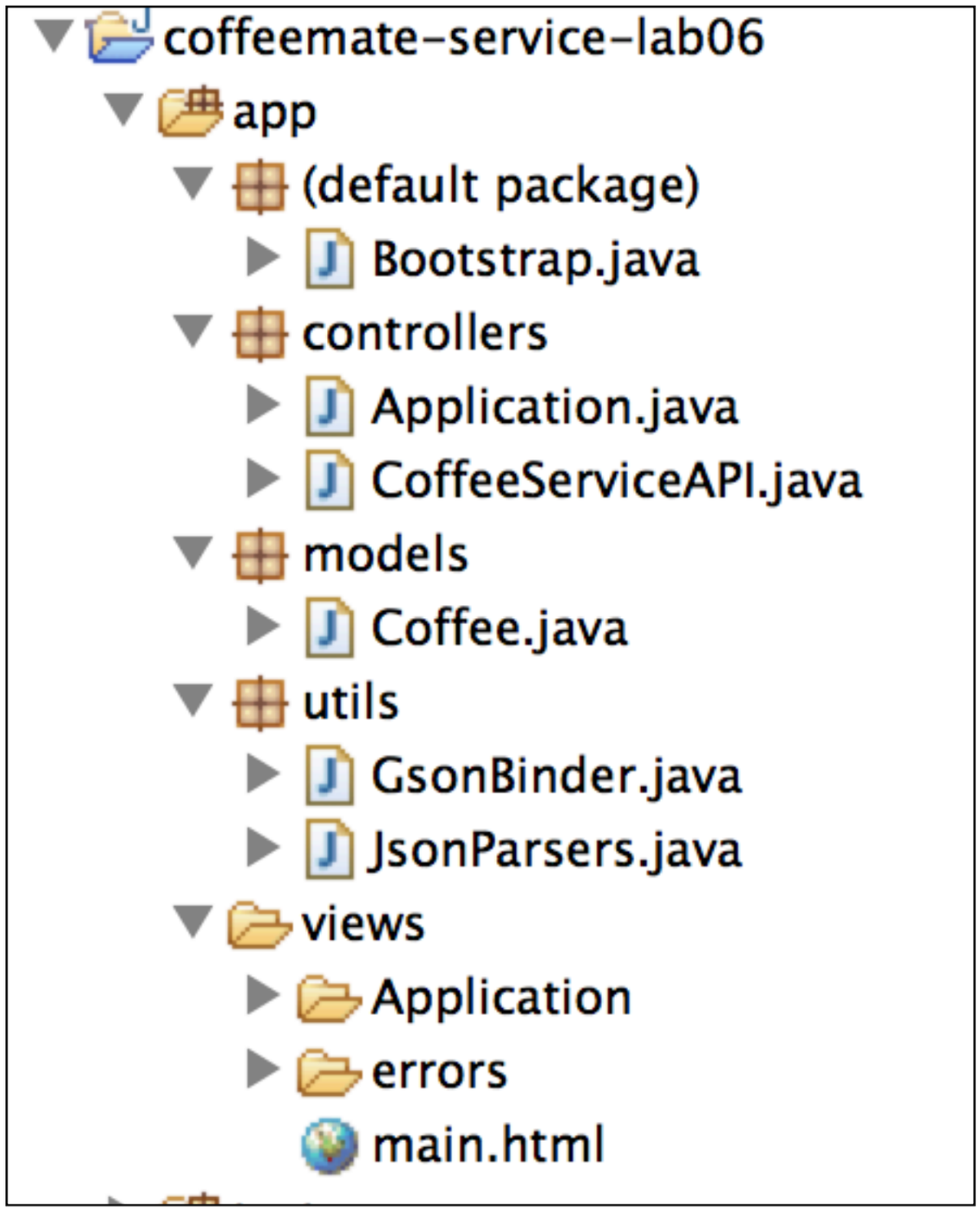
- Play Web App + API
- Android App - consumes the API to deliver enhanced UX
- ***+ A Java Test App***
  - *The Test App is to independently verify the correct operation of the API*
  - *Its purpose is to support orderly evolution of a robust service*

coffemate-service

- A standard Play application
- However:
  - No views
  - No conventional controllers



coffemate-service



- Standard ‘models’ package - containing model classes
- Utils classes to support API
- API ‘served’ by CoffeeServiceAPI controller

coffeemate-service model

```java
@Entity
public class Coffee extends Model
{
  public String name;
  public String shop;
  public double rating;
  public double price;
  public int    favourite;

  public Coffee(String coffeeName, String shop, double rating, double price, int favourite)
  {
    this.name      = coffeeName;
    this.shop      = shop;
    this.rating    = rating;
    this.price     = price;
    this.favourite = favourite;
  }

  @Override
  public boolean equals(final Object obj)
  {
    if (obj instanceof Coffee)
    {
      final Coffee other = (Coffee) obj;
      return Objects.equal(name,      other.name)
          && Objects.equal(shop,      other.shop)
          && Objects.equal(rating,    other.rating)
          && Objects.equal(price,     other.price)
          && Objects.equal(favourite, other.favourite);
    }
    else
    {
      return false;
    }
  }

  public String toString()
  {
    return Objects.toStringHelper(this)
        .add("Id", id)
        .add("name",      name)
        .add("shop",      shop)
        .add("rating",    rating)
        .add("price",     price)
        .add("favourite", favourite).toString();
  }
}
```

- All Model classes should implement toString and equals methods
- equals provides convenient comparison of model objects (very useful for testing)
- toString provides support for logging and debugging

coffeemate-service utils

```
public class JsonParsers
{
  static Gson gson = new Gson();

  public static Coffee json2Coffee(String json)
  {
    return gson.fromJson(json, Coffee.class);
  }

  public static String coffee2Json(Object obj)
  {
    return gson.toJson(obj);
  }

  public static List<Coffee> json2Coffees(String json)
  {
    Type collectionType = new TypeToken<List<Coffee>>() {}.getType();
    return gson.fromJson(json, collectionType);
  }
}
```

- JsonParsers - translate to / from Coffee Java objects and Json representations
- GsonBinder - Make Json payload of http requests available to controller/action methods.

```
public class GsonBinder implements TypeBinder<JsonElement>
{
  public Object bind(String name, Annotation[] notes, String value, Class toClass, Type toType) throws Exception
  {
    return new JsonParser().parse(value);
  }
}
```

coffeemate-service controllers

```
public class CoffeeServiceAPI extends Controller
{
  public static void coffees()
  {
    List<Coffee> coffees = Coffee.findAll();
    renderJSON(JsonParsers.coffee2Json(coffees));
  }

  public static void coffee (Long id)
  {
   Coffee coffee = Coffee.findById(id);
   renderJSON (JsonParsers.coffee2Json(coffee));
  }

  public static void createCoffee(JsonElement body)
  {
    Coffee coffee = JsonParsers.json2Coffee(body.toString());
    coffee.save();
    renderJSON (JsonParsers.coffee2Json(coffee));
  }
}
```

- Handle http requests associated with routes
- If request provides data - assume this is Json format and translate it into Java objects
- Respond to requests in Json format

```
GET     /api/coffees                    CoffeeServiceAPI.coffees
GET     /api/coffees/{id}               CoffeeServiceAPI.coffee
POST    /api/coffees                    CoffeeServiceAPI.createCoffee
```

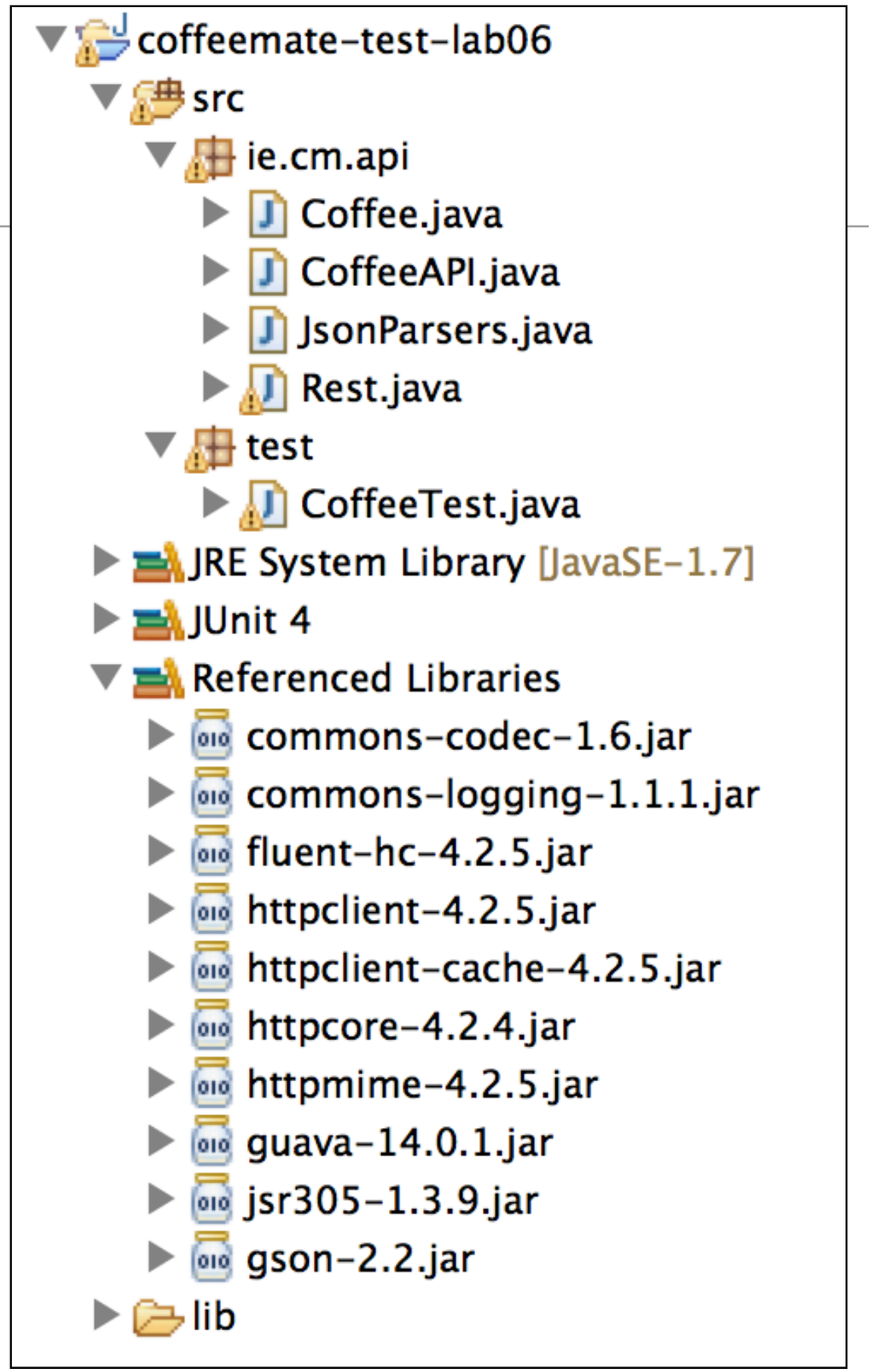
coffeemate-test

- A Standard Java Project (not a web app, not an android app)

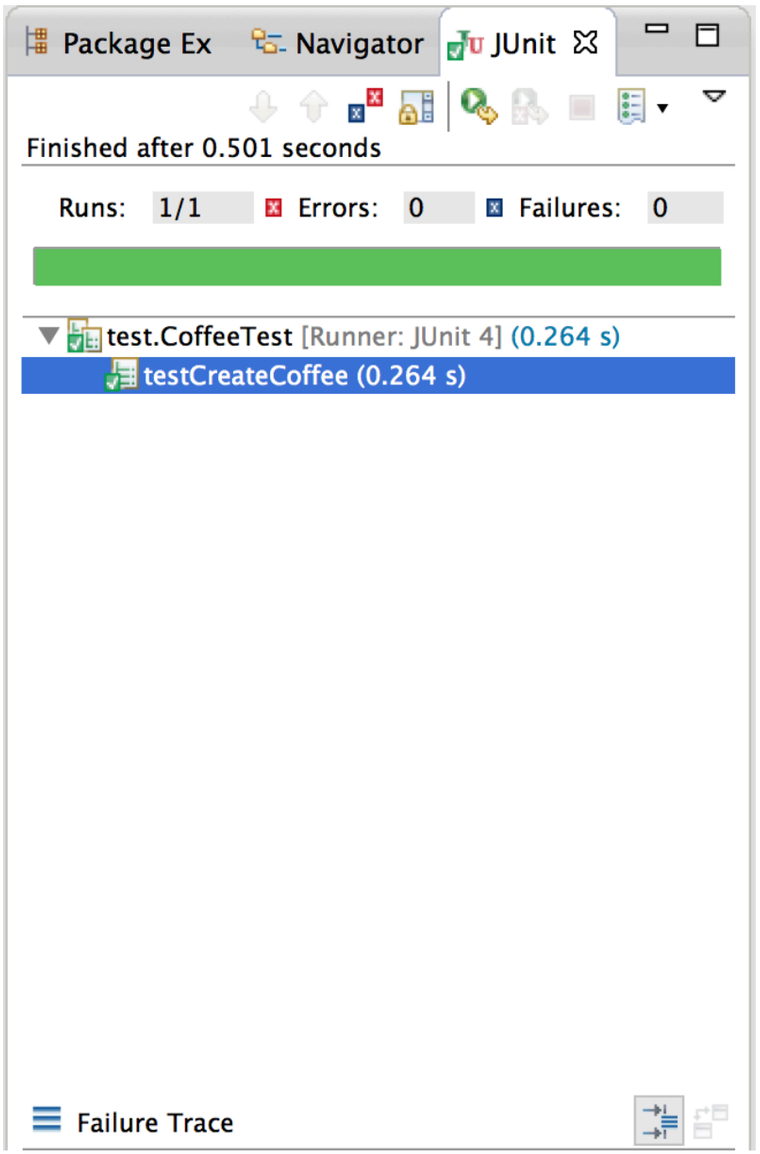
- Designed to test the coffemate-service api (CoffeeTest)

- Contains support classes to handle HTTP communication from client to server (Rest, JsonParsers, Coffee)

- Provides wrapper around these classes to simplify api calls (CoffeeAPI).



coffeemate-test




```
public class CoffeeTest
{
  private Coffee mocha;
  private long   mochaId;

  @Before
  public void setup() throws Exception
  {
    Coffee returnedCoffee;

    mocha      = new Coffee ("mocha", "costa",  4.5, 3.0, 1);
    returnedCoffee = CoffeeAPI.createCoffee(mocha);
    mochaId = returnedCoffee.id;
  }


  @After
  public void teardown()
  {
    mocha      = null;
  }

  @Test
  public void testCreateCoffee () throws Exception
  {
    assertEquals (mocha, CoffeeAPI.getCoffee(mochaId));
  }
}
```

CoffeeTest

```
public class CoffeeTest
{
  private Coffee mocha;
  private long   mochaId;

  @Before
  public void setup() throws Exception
  {
    Coffee returnedCoffee;

    mocha      = new Coffee ("mocha", "costa",  4.5, 3.0, 1);
    returnedCoffee = CoffeeAPI.createCoffee(mocha);
    mochaId = returnedCoffee.id;
  }

  @After
  public void teardown()
  {
    mocha      = null;
  }

  @Test
  public void testCreateCoffee () throws Exception
  {
    assertEquals (mocha, CoffeeAPI.getCoffee(mochaId));
  }
}
```

- Create a ‘local’ coffee object (mocha)
- As the coffeemate-service to create a new Coffee object on the server, using mocha to initialize it.
- Store the id of the created object (mochaId)
- In the test - ask the API for the objects with the id mochaID
- Compare it with the local object mocha.

JsonParsers

```
public class JsonParsers
{
  static Gson gson = new Gson();

  public static Coffee json2Coffee(String json)
  {
    return gson.fromJson(json, Coffee.class);
  }

  public static String coffee2Json(Object obj)
  {
    return gson.toJson(obj);
  }

  public static List<Coffee> json2Coffees(String json)
  {
    Type collectionType = new TypeToken<List<Coffee>>() {}.getType();
    return gson.fromJson(json, collectionType);
  }
}
```

- Exactly same class as in coffeemate-service

Coffee

- Almost exactly same class as in coffeemate-service
- Except:
  - Not derived from Model
  - Not annotated with JPA
  - Contains explicit id field

```java
public class Coffee
{
  public Long   id;
  public String name;
  public String shop;
  public double rating;
  public double price;
  public int    favourite;

  public Coffee()
  {}
  public Coffee(String coffeeName, String shop, double rating, double price, int favourite)
  {
    this.name      = coffeeName;
    this.shop      = shop;
    this.rating    = rating;
    this.price     = price;
    this.favourite = favourite;
  }
  @Override
  public boolean equals(final Object obj)
  {
    if (obj instanceof Coffee)
    {
      final Coffee other = (Coffee) obj;
      return Objects.equal(name,      other.name)
          && Objects.equal(shop,      other.shop)
          && Objects.equal(rating,    other.rating)
          && Objects.equal(price,     other.price)
          && Objects.equal(favourite, other.favourite);
    }
    else
    {
      return false;
    }
  }
  public String toString()
  {
    return Objects.toStringHelper(this)
        .add("id",        id)
        .add("name",      name)
        .add("shop",      shop)
        .add("rating",    rating)
        .add("price",     price)
        .add("favourite", favourite).toString();
  }
}
```

```
public class Rest
{
  private static DefaultHttpClient httpClient = null;
  private static final String URL = "http://localhost:9000";

  private static DefaultHttpClient httpClient()
  {
    if (httpClient == null)
    {
      HttpParams httpParameters = new BasicHttpParams();
      HttpConnectionParams.setConnectionTimeout(httpParameters, 2000);
      HttpConnectionParams.setSoTimeout(httpParameters, 2000);
      httpClient = new DefaultHttpClient(httpParameters);
    }
    return httpClient;
  }

  public static String get(String path) throws Exception
  {
    HttpGet getRequest = new HttpGet(URL + path);
    getRequest.setHeader("accept", "application/json");
    HttpResponse response = httpClient().execute(getRequest);
    return new BasicResponseHandler().handleResponse(response);
  }

  public static String post(String path, String json) throws Exception
  {
    HttpPost putRequest = new HttpPost(URL + path);
    putRequest.setHeader("Content-type", "application/json");
    putRequest.setHeader("accept", "application/json");

    StringEntity s = new StringEntity(json);
    s.setContentEncoding("UTF-8");
    s.setContentType("application/json");
    putRequest.setEntity(s);

    HttpResponse response = httpClient().execute(putRequest);
    return new BasicResponseHandler().handleResponse(response);
  }
}
```

Rest

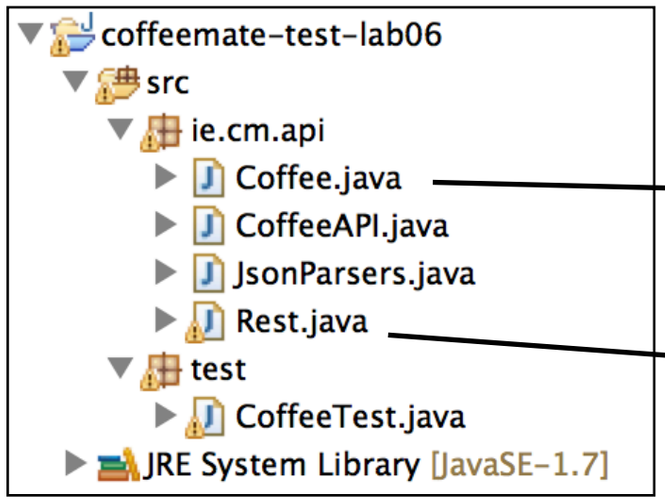
- A utility class to ‘speak’ http to a server
- Supports ‘get’ and ‘post’ Json requests
- Hard coded to http://localhost:9000
- Still need to incorporate
  - delete
  - put

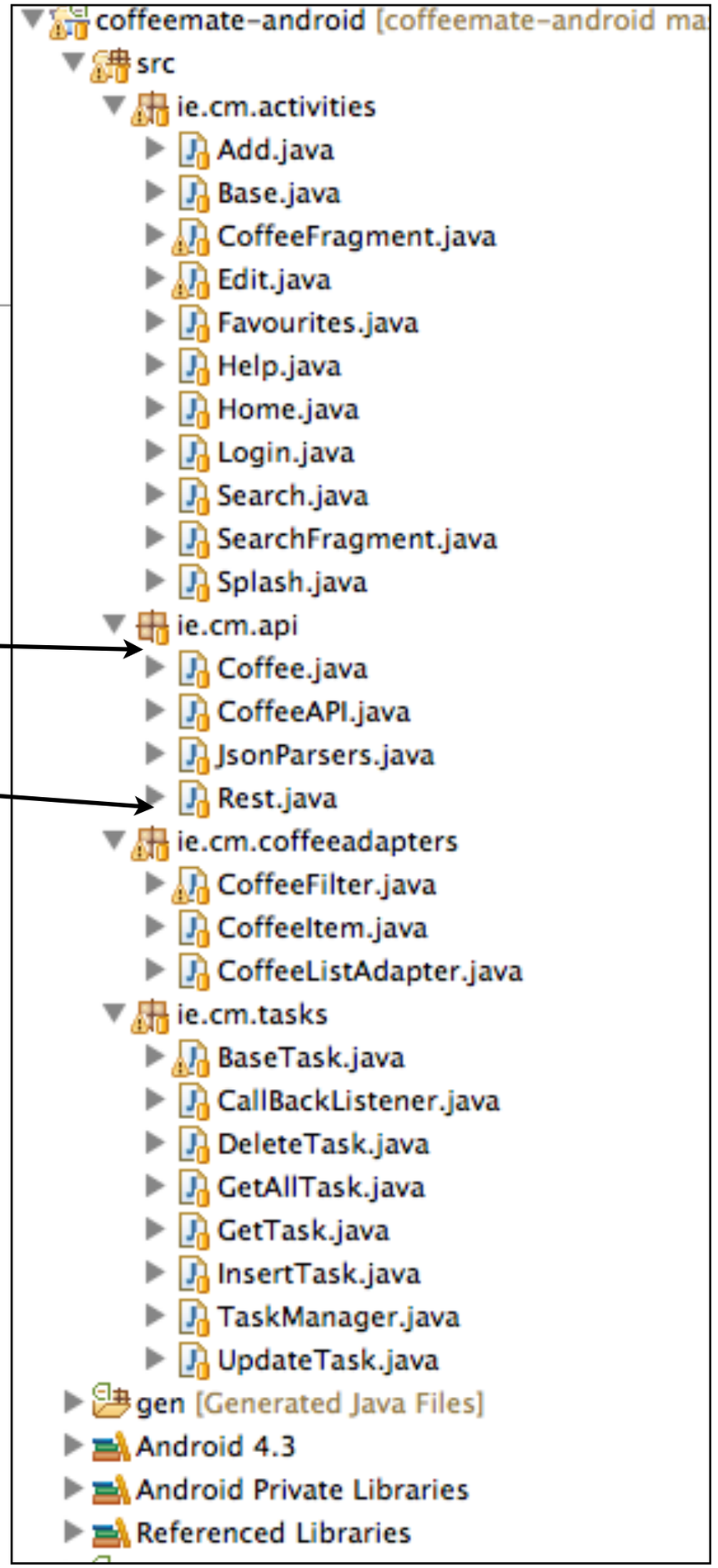
CoffeeAPI

- Client side ‘proxy’ for the coffeemate-service API
- Uses JsonParsers and Rest classes to communicate with API
- Exposes Coffee object API to local process

```
public class CoffeeAPI
{
  public static Coffee getCoffee (Long id) throws Exception
  {
    String coffeeStr = Rest.get("/api/coffees/" + id);
    Coffee coffee = JsonParsers.json2Coffee(coffeeStr);
    return coffee;
  }

  public static Coffee createCoffee(String coffeeJson) throws Exception
  {
    String response = Rest.post ("/api/coffees", coffeeJson);
    return JsonParsers.json2Coffee(response);
  }

  public static Coffee createCoffee(Coffee coffee) throws Exception
  {
    return createCoffee(JsonParsers.coffee2Json(coffee));
  }

  public static List<Coffee> getCoffees () throws Exception
  {
    String response =  Rest.get("/api/coffees");
    List<Coffee> coffeeList = JsonParsers.json2Coffees(response);
    return coffeeList;
  }
}
```

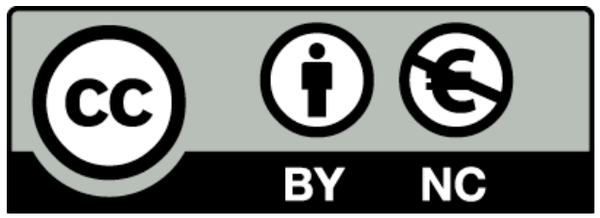
Android





- Android version of coffemate will use this ie.cm.api classes directly - unmodified

- However, we can test them exhaustively here before integrating into Android.



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

eLearning support unit