Iteração 3

FF – Falcon Framework

Thiago Meira Bernardes
Gabriel Araújo
Luis Filipe

Universidade de Brasília - UNB
Faculdade Gama – FGA

Brasília, DF - 2013

Índice

1. Introdução

O planejamento da iteração foi realizado com o auxílio da ferramenta trello. Todos os recursos presentes no https://gitlab.com/FalconTeam/artefatos/wikis/Plano-da-fase- foram alocados para essa iteração

A iteração teve Início no dia 10/05/2016 e fim no dia 30/05/2016 com a duração de 3 semanas. Mais detalhes interativos sobre a mesma podem ser encontrados nos principais repositórios utilizados.

Artefatos - https://gitlab.com/FalconTeam/artefatos

FalconAPIClientSDK Android - https://gitlab.com/FalconTeam/android_sdk

FalconServerFactory - https://gitlab.com/FalconTeam/server_generetor

2. Requisitos Planejados

UC03 - Criar código API Cliente

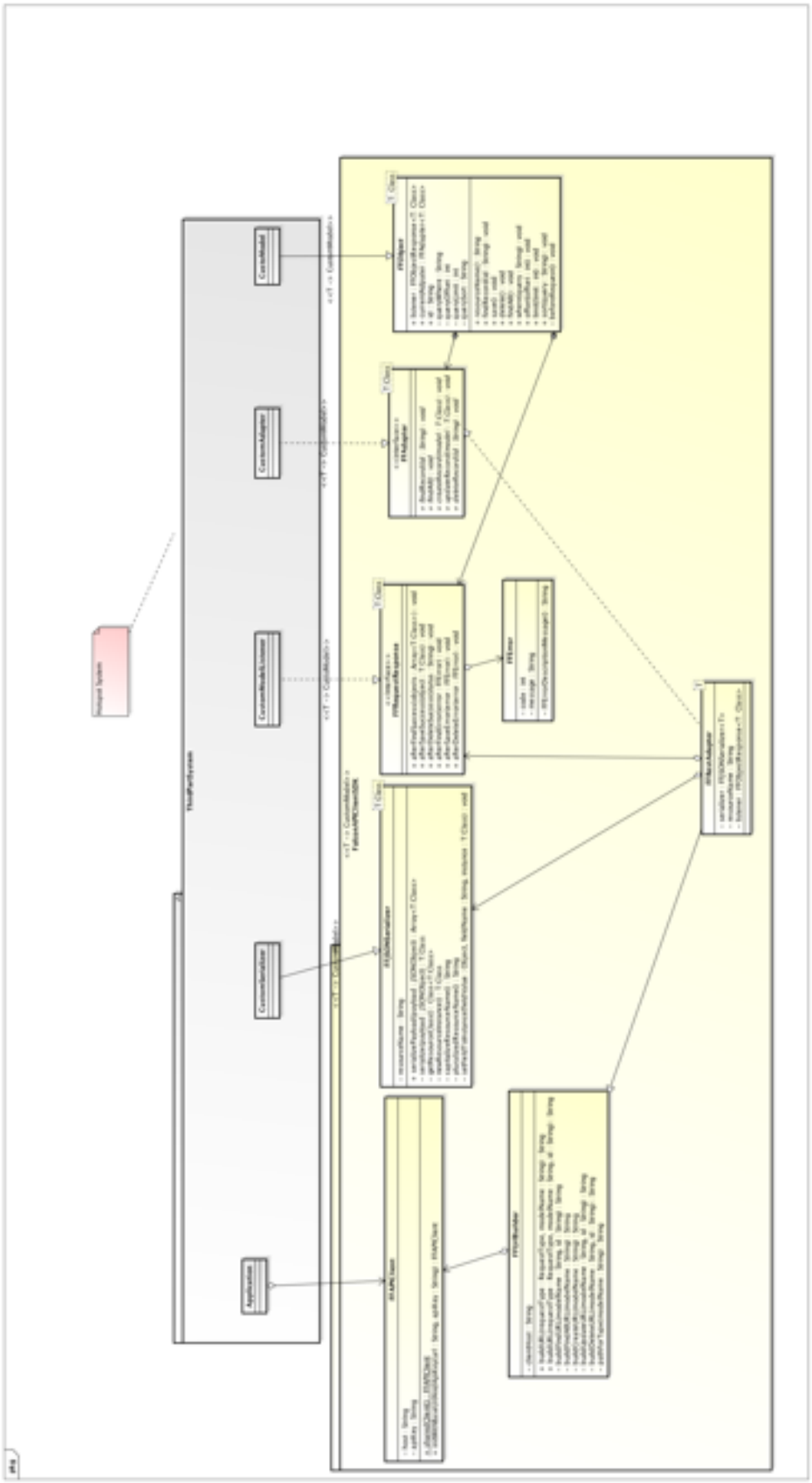
- AT06 - Criar código de POST
- AT07 - Criar código de GET
- AT08 - Criar código de PUT
- AT09 - Criar código de DELETE

3. Objetivos

- Evolouir incremento de software da interação anterior.
- Evoluir artefatos da interação anterior.
- Continuar a Implementar o caso de uso UC03.
- Refatorar com implementações de padrões de projeto
- Definir plano de métricas.
- Empacotar solução de geração automática de código de servidores.

4. Versão da arquitetura evoluída

Para melhor visualização consultar https://gitlab.com/FalconTeam/artefatos/wikis/iteracao-3



5. Principais evoluções e incrementos

- Foi empacotada em uma .gem do ruby a primeira versão beta do gerador de servidor. A gem pode ser acessada no seguinte repositório Falcon Server Factory
- A arquitetura base do FalconAPIClientSDK foi evoluída aplicando padrões GoF dando mais liberdade de customização do SDK, transformando o framework em um framework do tipo cinzento, tendo uma implementação padrão e possibilidade de ter várias implementações customizada.
- Criad o plano de métricas presente no link https://gitlab.com/FalconTeam/artefatos/wikis/Plano-de-Métricas

6. Padrões implementados

6.1. Creational

6.1.1.Singleton

A. Problema

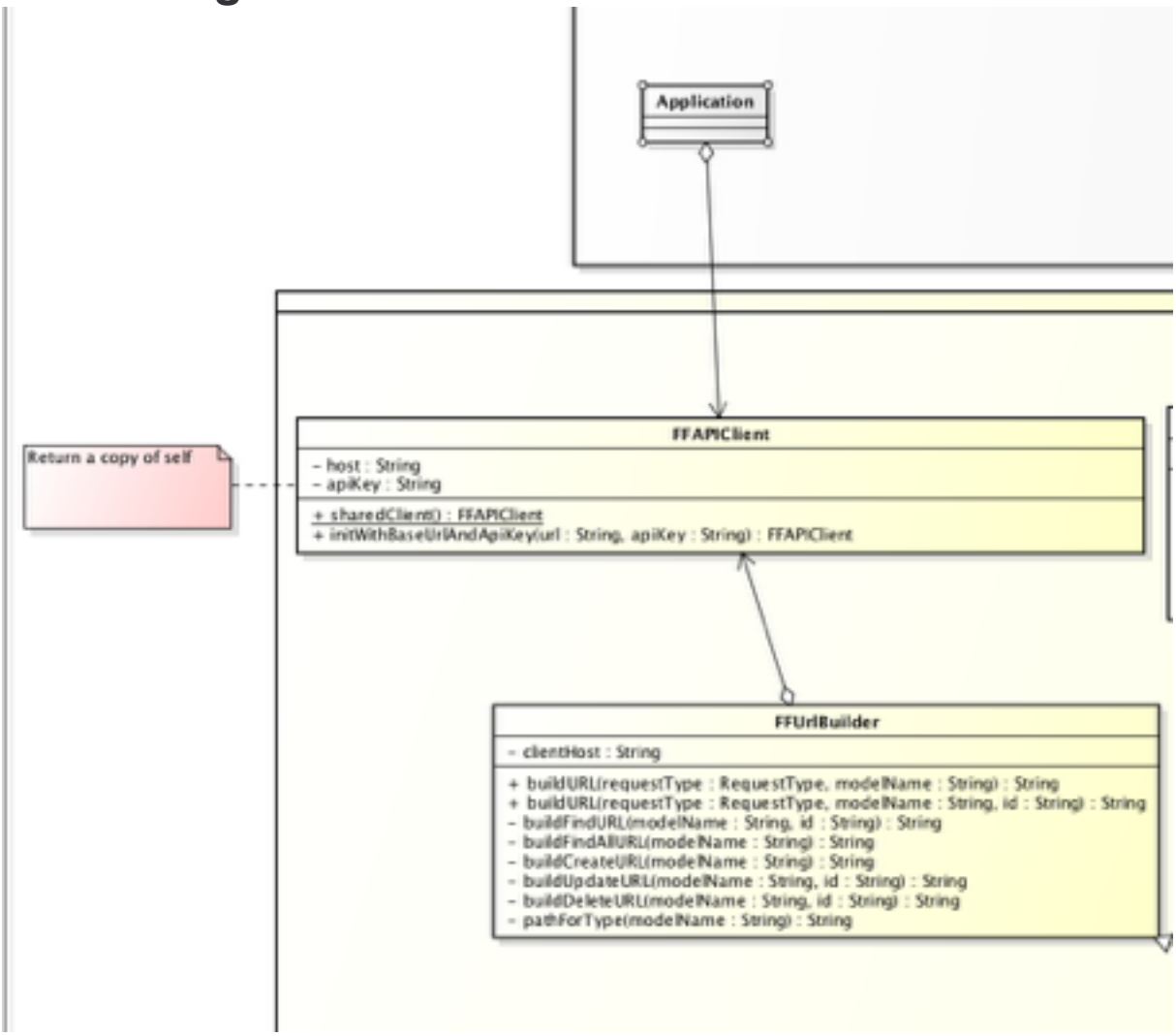
Algumas classes devem ser instanciadas apenas uma vez durante a aplicação. A definição de uma variável global permite o acesso a esse recurso, porém não limita ela a ser instanciado apenas uma vez durante a aplicação. No nosso caso, o ApiClient não necessita de mais instancias pois é quem configura o uso do SDK no android.

B. Solução

- O padrão singleton permite a classe ter apenas uma instância e cria um ponto de acesso global a ela.
- A própria classe é responsável pela manutenção da sua instância:
- Quando a instância for requisitada pela primeira vez, essa instância deve ser criada;
- Em requisições subsequentes, a instância criada na primeira vez é retornada.

- Como citado foi aplicado na classe ApiClient que implementa esse padrão.

1. Modelagem



2. Implementação

Classe singleton

```
public class FFAPIClient {

    private  String host="";
    private  String apiKey = "";

    private static FFAPIClient ourInstance = new FFAPI-
Client();

    /**
     * Returns an Singleton Instance
     *
     * @return  the static instance of this class
     */
```

```
    public static FFAPIClient sharedClient() {
        return ourInstance;
    }

    private FFAPIClient(){

    }
    /**
     * Set in Singleton Instance the parametres url and
apiKey
     *
     * @param url an api base url for client
     * @param key an api key of client in server
     */
    public FFAPIClient (String url, String key){
        sharedClient().host =
this.normalizeNakedURL(url);
        sharedClient().apiKey = key;
    }

    public String getHost() {
        return host;
    }

    public void setHost(String apiBaseUrl) {
        this.host = apiBaseUrl;
    }

    public String getApiKey() {
        return apiKey;
    }

    public void setApiKey(String apiKey) {
        this.apiKey = apiKey;
    }

    private String normalizeNakedURL(String nakedURL) {
        return "http://" + nakedURL;
    }

}
```

Classe Application que instancia o Singleton

```
public class MainActivity extends AppCompatActivity im-
plements FFRequestResponse<User>{

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Set API settings
        FFAPIClient apiSetting = new
FFAPIClient("192.168.0.21:3000", "none");
....
```

Classe URL builde que usa o singleton

```
public class FFURLBuilder {

    private FFAPIClient apiSettings;
    private String host;
    private String apiKey;

    public FFURLBuilder(){
        this.apiSettings = FFAPIClient.sharedClient();

        this.host = this.apiSettings.getHost();
        this.apiKey = this.apiSettings.getApiKey();
    }
....
```

6.1.2.Builder

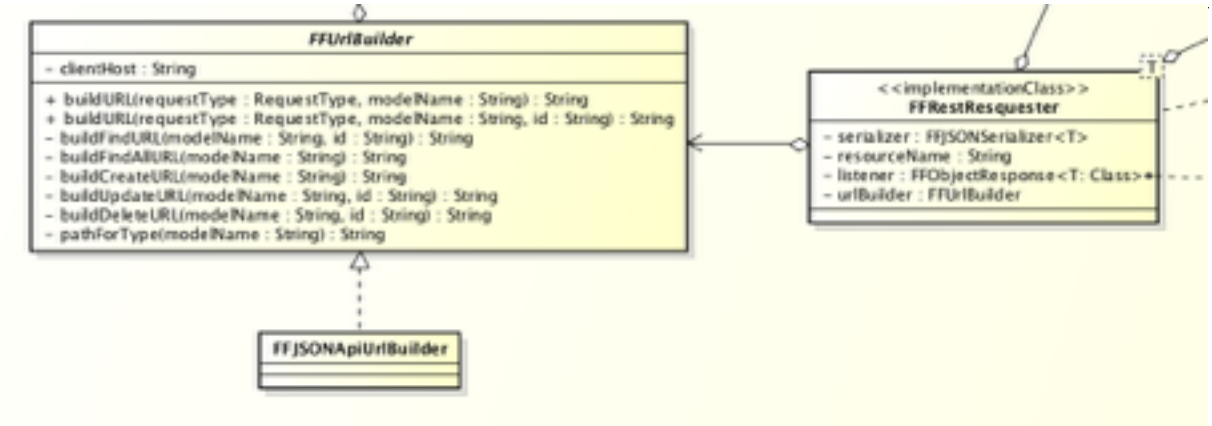
A. Problema

- Preciso separar a construção de um objeto complexo da parte que constrói o objeto e como ele é montado.
- No processo de construção preciso de representações diferentes do meu objeto, para a construção variar de acordo com o tipo de servidor que meu cliente está lidando.
- O processo de construção precisa ser separado em diferentes métodos.

B. Solução

O padrão Builder permite separar a construção de um objeto complexo de sua representação de modo que o mesmo processo de construção possa criar diferentes representações: No nosso caso o URL builder sevirá de builder e pode variar a construção de urls dependendo do tipo de servidor que estou tentando me conectar.

1. Modelagem



2. Implementação

Classe Builder Abstrata

```
public abstract class FFURLBuilder {

    protected FFAPIClient apiSettings;
    protected String host;
    protected String apiKey;
    /**
     * Builds a URL for a given type and optional ID.
     * By default, it pluralizes the type's name (for ex-
ample, 'post' becomes 'posts' and 'person' becomes 'peo-
ple').
     * If an ID is specified, it adds the ID to the path
generated for the type, separated by a /.
     */
    public abstract String buildURL(String requestType,
String modelName);
    public abstract String buildURL(String requestType,
String modelName, String id);

}
```

Classe Builder Concreta

```
public class FFJSONApiURLBuilder extends FFURLBuilder {


    public FFJSONApiURLBuilder(){
        this.apiSettings = FFAPIClient.sharedClient();
        this.host = this.apiSettings.getHost();
        this.apiKey = this.apiSettings.getApiKey();
    }

    @Override
    public String buildURL(String requestType, String
modelName) {
```

```
        switch (requestType) {
            case "findAll":
                return this._buildURL(modelName);
            case "query":
                return this._buildURL(modelName);
            case "createRecord":
                return this._buildURL(modelName);
            default:
                return this._buildURL(modelName);
        }
    }

    @Override
    public String buildURL(String requestType, String
modelName, String id) {
        switch (requestType) {
            case "findRecord":
                return this._buildURL(modelName, id);
            case "updateRecord":
                return this._buildURL(modelName, id);
            case "deleteRecord":
                return this._buildURL(modelName, id);
            default:
                return this._buildURL(modelName);
        }
    }

    private String _buildURL(String modelName) {
        String path = this.pathForType(modelName);
        String host = this.host;

        return TextUtils.join("/", new
ArrayList<String>(Arrays.asList(host, path)));
    }

    private String _buildURL(String modelName, String id)
{
        String encodedID = null;
        try {
            encodedID = URLEncoder.encode(id, "UTF-8");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return TextUtils.join("/", new
ArrayList<String>(Arrays.asList(this._buildURL(modelName)
, encodedID)));
```

```
    }

    private String pathForType(String modelName) {
        return English.plural(modelName);
    }
}
```

Classe Builder Construtor

```
public class FFRestRequester<T extends FFObject>  implements FFRequester<T> {


    private AsyncHttpClient asyncHttp = new AsyncHttpClient();
    private FFJSONSerializer<T> serializer;
    private String resourceName;
    private FFRequestResponse<T> requestResponse;
    private FFURLBuilder urlBuilder;

public FFRestRequester(String resourceName, FFRequestResponse<T> requestResponse) {
        this.resourceName = resourceName;
        this.requestResponse = requestResponse;
        serializer =  new FFJSONSerializer<>(resourceName);

        switch (FFAPIClient.sharedClient().getServerPattern()) {
            case NONE:
                this.urlBuilder = null;
                break;
            case JSONAPI:
                this.urlBuilder = new FFJSONApiURLBuilder();
                break;
        }
    }
....
```

6.2. Structural

6.2.1.Bridge

A. Problema

A implementação lógica de uma classe abstrata precisa variar, para que cada nova classe concreta precisa ter variados tipos de implementação em tempo de execução. A classe abstrata precisa ser

protegida para não se alterar sua implementação abstrata tornando-a retrocompatível com qualquer implementação que o usuário do framework defina.

B. Solução

O padrão bridge, desacopla a abstração de sua implementação para que ambas possam variar sem dependências diretas. Uma abstração pode ter muitas extensões e hierarquias, e o padrão permite uma hierarquia de chamadas mais generalizada.

1. Modelagem

2. Implementação

Classe Abstrata

```
public abstract class FFObject<T> {
```

```
    public Integer id;
    public FFRestRequester requester;
    public FFRequestResponse<T> requestResponse;

    public abstract String resourceName();

......
```

Classe Abstração Concreta

```
import FalconAPIClientSDK.FFObject;

public class User extends FFObject {

    public String name;
    public String email;
    public Integer age;

    @Override
    public String resourceName() {
        return "user";
    }
}
....
```

Classe Implementador

```
import FalconAPIClientSDK.FFObject;

public class User extends FFObject {

    public String name;
    public String email;
    public Integer age;

    @Override
    public String resourceName() {
        return "user";
    }
}
....
```

Classe Implementador Concreto

```
public class FFRestRequester<T extends FFObject> extends
FFURLBuilder implements FFRequester<T> {


    private AsyncHttpClient asyncHttp = new AsyncHttp-
Client();
    private FFJSONSerializer<T> serializer;
```

```
    private String resourceName;
    private FFRequestResponse<T> requestResponse;

    public FFRestRequester(String resourceName, FFRe-
questResponse<T> requestResponse) {
        this.resourceName = resourceName;
        this.requestResponse = requestResponse;
        serializer =  new FFJSONSerializer<>(resource-
Name);
    }

    /**
     * Called by the FFResource in order to fetch the
JSON for a given type and ID.
     * The findRecord method makes an Asynchronous re-
quest to a URL computed by buildURL,
     * and returns a promise for the resulting payload.
     * This method performs an HTTP GET request with the
id provided as part of the query string.
     */
    @Override
    public void findRecord(String id) {
        String url = this.buildURL("findRecord", this.re-
sourceName, id);
        final FFRestRequester self = this;
        this.asyncHttp.get(url, new JsonHttpResponseHan-
dler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{
                ArrayList<T> response = (ArrayList<T>)
self.serializer.serializePayload(jsonObject);
                self.requestResponse.afterFindSuccess(re-
sponse);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

    /**
```

```
     * Called by the FFResource in order to fetch a JSON
array for all of the records for a given type.
     * The findAll method makes an Asynchronous (HTTP
GET) request to a URL computed by buildURL,
     * and returns a promise for the resulting payload.
     */
    @Override
    public void findAll() {
        String url = this.buildURL("findAll", this.re-
sourceName);
        final FFRestRequester self = this;

        this.asyncHttp.get(url, new JsonHttpResponseHan-
dler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{
                ArrayList<T> response = (ArrayList<T>)
self.serializer.serializePayload(jsonObject);
                self.requestResponse.afterFindSuccess(re-
sponse);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

//    /**
//     * Called by the FFResource in order to fetch a
JSON array for the records that match a particular query.
//     * The query method makes an Asynchronous (HTTP
GET) request to a URL computed by buildURL, and returns a
//     * promise for the resulting payload.
//     * The query argument is a simple Map object that
will be passed directly to the server as parameters.
//     */
//    @Override
//    public void query(String modelName, String query) {
//
//    }

    /**
```

```
     * Called by FFResource when a newly created record
is saved via the save method on a model
     * record instance. The createRecord method serial-
izes the record and makes an Asynchronous
     * (HTTP POST) request to a URL computed by buildURL.
     * See serialize for information on how to customize
the serialized form of a record.
     */
    @Override
    public void createRecord(final T model) {
        String url = this.buildURL("createRecord",
this.resourceName);
        final FFRestRequester self = this;

        RequestParams params = this.serializer.deserial-
ize(model);

        this.asyncHttp.post(url, params, new JsonHttpRe-
sponseHandler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{
                System.out.println(jsonObject);
                try {
                    model.id =
jsonObject.getJSONObject(self.resourceName).getInt("id");
                } catch (JSONException e) {
                    e.printStackTrace();
                }


self.requestResponse.afterSaveSuccess(model);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

    /**
     * Called by the FFResource when an existing record
is saved via the save method on a model record instance.
```

```
     * The updateRecord method serializes the record and
makes an Asynchronous (HTTP PUT) request to
     * a URL computed by buildURL.
     * See serialize for information on how to customize
the serialized form of a record.
     */
    @Override
    public void updateRecord(final T model) {
        String url = this.buildURL("updateRecord",
this.resourceName, model.id.toString());

        final FFRestRequester self = this;

        RequestParams params = this.serializer.deserial-
ize(model);

        this.asyncHttp.put(url, params, new JsonHttpRe-
sponseHandler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{

self.requestResponse.afterSaveSuccess(model);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

    /**
     * Called by the FFResource when a record is deleted.
     * The deleteRecord method makes an Asynchronous
(HTTP DELETE) request to a URL computed
     * by buildURL.
     */
    @Override
    public void deleteRecord(String id) {
        String url = this.buildURL("deleteRecord",
this.resourceName, id);
        final FFRestAdapter self = this;
```

```
        this.asyncHttp.delete(url, new JsonHttpResponse-
Handler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{

self.requestResponse.afterDeleteSuccess("204");
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }
....
```

6.2.2.Facade

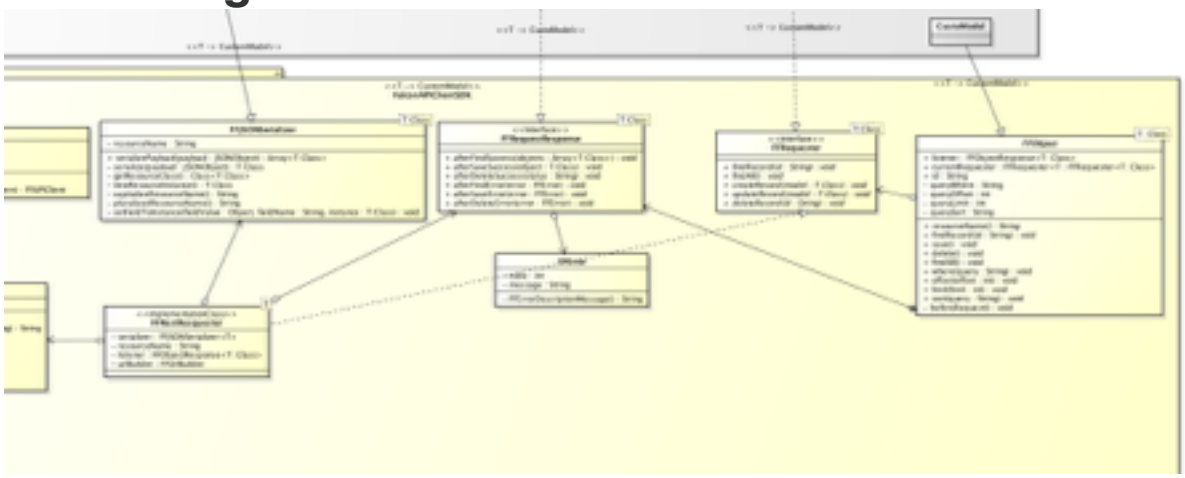
A. Problema

- Fornecer uma interface unificada para um conjunto de interfaces em um subsistema, mantendo mais fácil o uso de subsistemas.
- O Framework possui muitas interfaces e grande complexidade, devemos tornar a interação com o mesmo mais fácil para o usuário.

B. Solução

- O padrão Facade permite minimizar a comunicação do cliente com os subsistemas do Framework, fornecendo uma interface simplificada através de uma "fachada" permitindo um uso mais fácil dos recursos do mesmo.

1. Modelagem



2. Implementação

Classe Fachada

```
public abstract class FFObject<T> {

    public Integer id;
    public FFRestRequester currentRequester;
    public FFRequestResponse<T> requestResponse;

    public abstract String resourceName();

    public FFRequestResponse<T> getRequestResponse() {
        return requestResponse;
    }

    public void setRequestResponse(FFRequestResponse<T>
requestResponse) {
        this.requestResponse = requestResponse;
    }

    public void findAll() {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();
        this.currentRequester.findAll();
    }

    public void findRecord(String id) {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();
        this.currentRequester.findRecord(id);
    }

    private void beforeRequest() {
        if (this.currentRequester == null) {
            this.currentRequester = new
FFRestRequester(this.resourceName(), this.requestRe-
sponse);
        }
    }

    public void save() {
        if (this.requestResponse == null) {
```

```
            return;
        }
        this.beforeRequest();

        if (this.id != null) {
            //update
            this.currentRequester.updateRecord(this);
        } else {
            //create
            this.currentRequester.createRecord(this);

        }
    }

    public void delete() {
        this.beforeRequest();
        if (this.id != null) {
            System.out.println(this.id);
            this.currentRequester.deleteRecord(String.-
valueOf(this.id));
        }
    }
}
```

Classe Cliente

```
public class User extends FFObject {

    public String name;
    public String email;
    public Integer age;

    @Override
    public String resourceName() {
        return "user";
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }
```

```
    public void setEmail(String email) {
        this.email = email;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}
```

Classe Que usa a Fachada

```
public class MainActivity extends AppCompatActivity im-
plements FFRequestResponse<User>{

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Set API settings
        FFAPIClient apiSetting = new
FFAPIClient("192.168.0.21:3000", "none", ServerPattern.J-
SONAPI);
        User user = new User();
        user.setRequestResponse(this);
        user.findRecord("1");
    }

    @Override
    public void afterFindSuccess(ArrayList<User> objects)
{
        User u = objects.get(0);
        u.setRequestResponse(this);
        System.out.println("######## Velho ##########");
        System.out.println(u.name);
        u.delete();
    }

....
```

6.2.3.Adapter

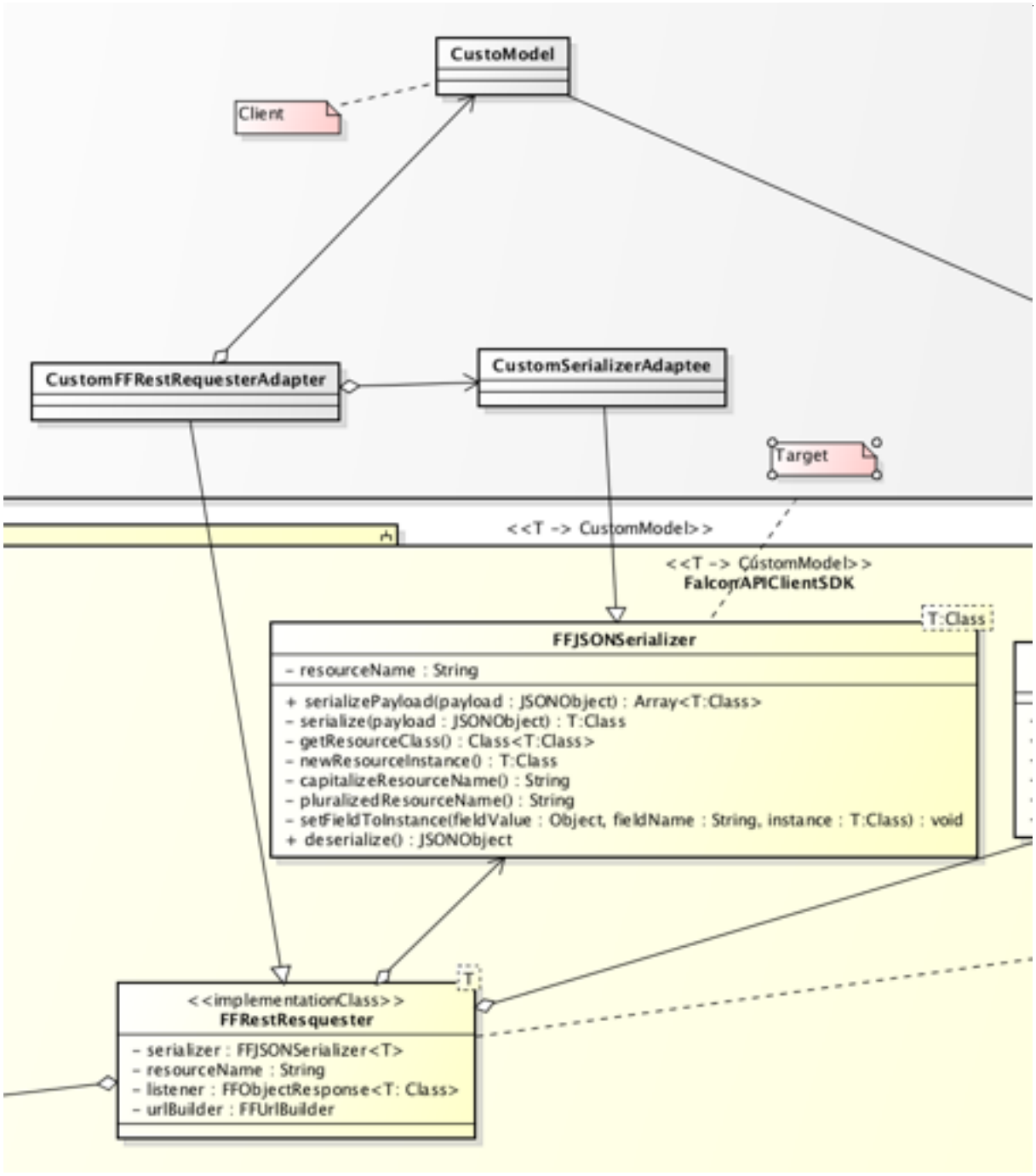
A. Problema

- Quero criar uma CustomRequester próprio e o FFJSONSerializer atual não serializa os objetos de acordo com o meu payload customizado, assim meu Cliente não está compatível com o FFJSONSerializer.

B. Solução

- É necessário converter a interface da classe FFJSONSerializer em outra interface, esperada pelo CustomRequester, para que desse modo o Cliente consiga criar um serializer compatíve,l com a sua necessidade, porém o FFObject precisa que esse objeto seja do tipo FFJSONSerializer, para poder continuar usando o Framework normalmente. O padrão Class Adapter permite exatamente isso.

1. Modelagem



2. Implementação

Classe Client

```
public abstract class FFObject<T> {

    public Integer id;
    public FFRestRequester currentRequester;
    public FFRequestResponse<T> requestResponse;

    public abstract String resourceName();

    public FFRequestResponse<T> getRequestResponse() {
```

```
        return requestResponse;
    }

    public void setRequestResponse(FFRequestResponse<T>
requestResponse) {
        this.requestResponse = requestResponse;
    }

    public void findAll() {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();
        this.currentRequester.findAll();
    }

    public void findRecord(String id) {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();
        this.currentRequester.findRecord(id);
    }

    private void beforeRequest() {
        if (this.currentRequester == null) {
            this.currentRequester = new
FFRestRequester(this.resourceName(), this.requestRe-
sponse);
        }
    }

    public void save() {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();

        if (this.id != null) {
            //update
            this.currentRequester.updateRecord(this);
        } else {
            //create
            this.currentRequester.createRecord(this);

        }
```

```
    }

    public void delete() {
        this.beforeRequest();
        if (this.id != null) {
            System.out.println(this.id);
            this.currentRequester.deleteRecord(String.-
valueOf(this.id));
        }
    }
}
```

Classe Target

```
public class FFJSONSerializer<T> {

//    private final Class<T> type;

    private String resourceName;

    public String getResourceName() {
        return resourceName;
    }

    private void setResourceName(String resourceName) {
        this.resourceName = resourceName;
    }

    public FFJSONSerializer(String resourceName) {
        setResourceName(resourceName);
    }

    public ArrayList<T> serializePayload(JSONObject pay-
load) {

        ArrayList<T> serializedPayload = new
ArrayList<T>();


        if (payload.has(this.pluralizedResourceName())) {
            JSONArray p = null;
            try {
                p = payload.getJSONArray(this.pluralized-
ResourceName());
            } catch (JSONException e) {
                e.printStackTrace();
            }
```

```
            for (int i = 0; i < p.length(); i++) {
                try {

serializedPayload.add(this.serialize(p.getJSONObject(i)))
;
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }

        } else {
            try {
                serializedPayload.add(this.serialize(pay-
load.getJSONObject(this.resourceName)));
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }

        return serializedPayload;
    }

    private T serialize(JSONObject payload) {
        T newResource = this.newResourceInstance();

        try {
            payload = payload.getJSONObject(this.re-
sourceName);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        Iterator<?> keys = payload.keys();

        while(keys.hasNext()) {
            String key = (String)keys.next();
            try {
                Object value = payload.get(key);
                this.setFildToInstace(value, key, newRe-
source);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }

        return  newResource;
```

```
    }

    private Class<T> getResourceClass() {
        Class<T> resourceClass = null;
        try {
            String className = "falconframework.sample-
falconsdk.Models." + this.capitalizedResourceName();
            resourceClass =
(Class<T>)Class.forName(className);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }

        return resourceClass;
    }

    private T newResourceInstance() {
        Class<T> resourceClass = this.getResourceClass();
        T newResourceInstace = null;

        try {
            newResourceInstace = resourceClass.newIn-
stance();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }

        return newResourceInstace;
    }

    private String capitalizedResourceName() {
        return this.resourceName.substring(0, 1).toUpper-
Case() + this.resourceName.substring(1);
    }

    private String pluralizedResourceName() {
        return English.plural(this.resourceName);
    }

    private void setFildToInstace(Object fieldValue,
String fieldName, T instance) {
        Field field = null;

        try {
```

```
            field = instance.getClass().getField(field-
Name);
            field.setAccessible(true);
            try {
                field.set(instance, fieldValue);
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        }
    }

    public RequestParams deserialize(T model) {
        Field[] fields = this.getResourceClass().get-
Fields();

        RequestParams params = new RequestParams();

        Map<Object, Object> map = new HashMap<Object, Ob-
ject>();

        for (Field field : fields ) {
            try {
                map.put(field.getName(), "" +
field.get(model));
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }

        params.put(this.resourceName, map);

        return params;
    }

}
```

Classe Adapter

Será implementada pelo usuário do framework

....

Classe Adaptee

Será implementada pelo usuário do framework

....

6.3. Behavioral

6.3.1.Iterator

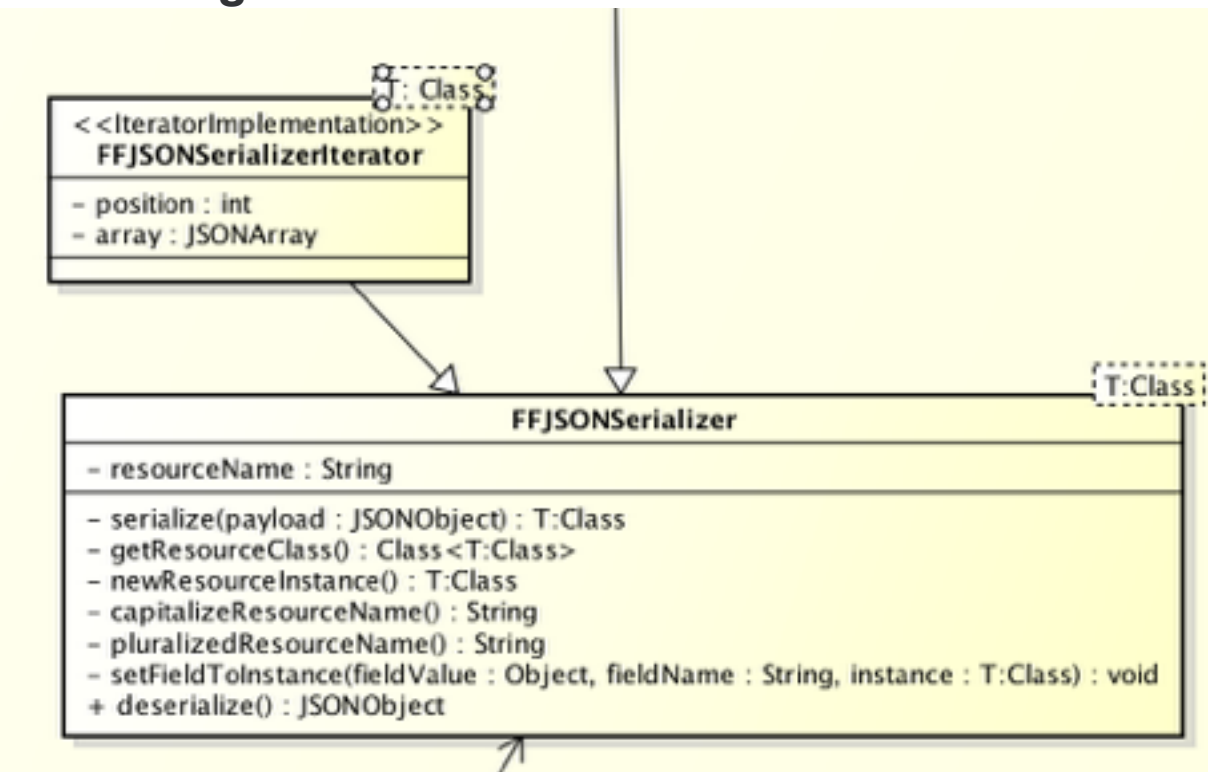
A. Problema

- Quero percorrer JSONObjects em um JSONArray e serializar cada json object que retornar, após serializar adiciona-lo em um array de objetos serializados.
- Essa operação está deixando a classe muito poluída e lotada de try catch.

B. Solução

- O padrão Iterator resolve isso permitindo a partir da interface Iterator, criar uma classe que realiza essa operação de recuperar o objeto json e serializar retornando o objeto serializado em cada iteração.

1. Modelagem



2. Implementação

Classe que implementa o Iterator

```
public class FFJSONSerializerIterator<T>  extends FFJSON-
Serializer<T> implements Iterator<T> {

    JSONArray array;
    int position = 0;
```

```
    public FFJSONSerializerIterator(String
resourceName,JSONArray jsonArray) {
        super(resourceName);
        this.array = jsonArray;
    }

    @Override
    public boolean hasNext() {
        try {
            if (position >= array.length() ||
array.get(position) == null) {
                return false;
            } else {
                return true;
            }
        } catch (JSONException e) {
            e.printStackTrace();
            return false;
        }    }

    @Override
    public T next() {
        try {
            T object = this.serialize(array.getJSONOb-
ject(this.position));
            position++;
            return object;
        }catch (JSONException e) {
            e.printStackTrace();
            return null;
        }

    }

    @Override
    public void remove() {

    }
}
```

Classe que usa o Iterator implementado

```
 public class FFJSONSerializer<T> {

    //    private final Class<T> type;

        private String resourceName;
```

```
        public String getResourceName() {
            return resourceName;
        }

        private void setResourceName(String resourceName)
{
            this.resourceName = resourceName;
        }

        public FFJSONSerializer(String resourceName) {
            setResourceName(resourceName);
        }

        public ArrayList<T> serializePayload(JSONObject
payload) {

            ArrayList<T> serializedPayload = new Array-
List<T>();


            if
(payload.has(this.pluralizedResourceName())) {
                JSONArray payloadArray = null;
                try {
                    payloadArray =
payload.getJSONArray(this.pluralizedResourceName());
                } catch (JSONException e) {
                    e.printStackTrace();
                }
                Iterator<T> jsonArrayIterator  = new FFJ-
SONSerializerIterator<>(this.resourceName, payloadArray);
                while (jsonArrayIterator.hasNext()) {
                    T object = jsonArrayIterator.next();
                    serializedPayload.add(object);
                }
...
```

6.3.2.Observer

Problema

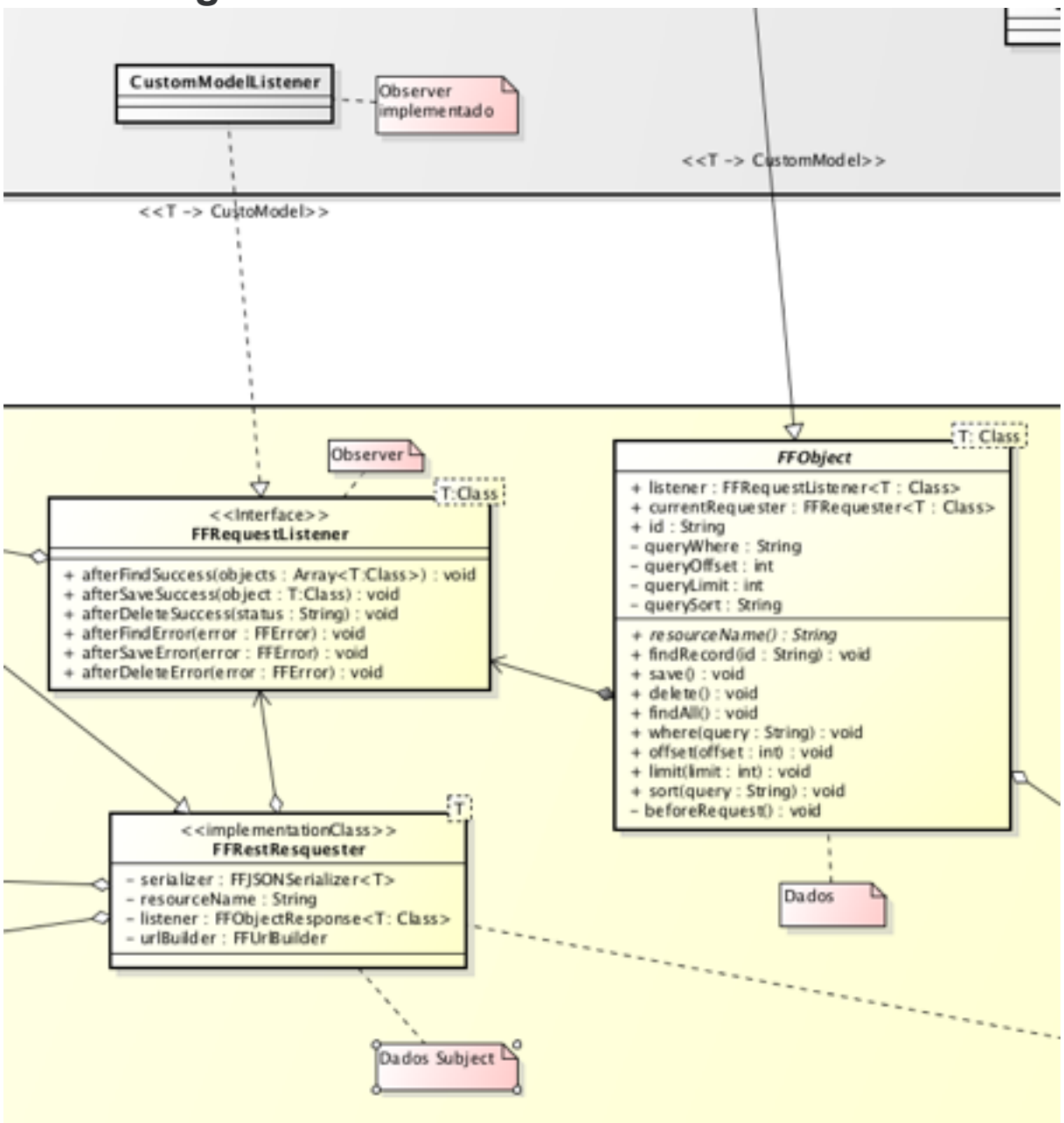
- Sempre que uma nova requisição for realizada no servidor, os assinantes devem receber essa requisição e tratar da maneira que quizerem, serja para listar a model em uma tabela, seja

para excluir todas as models listadas, ou para mostrar um aviso ao usuário, que foi requisitado com sucesso.

B. Solução

- O padrão observer permite isso, através de assinantes(observadores) e notificadores de assuntos, os assinantes podem receber e tratar os dados sempre que o download ou upload da requisição for completado(com sucesso ou não).

1. Modelagem



2. Implementação

Classe com o Dado

```
public abstract class FFObject<T> {
```

```
    public Integer id;
    public FFRestRequester currentRequester;
    public FFRequestListener<T> requestResponse;

    public abstract String resourceName();

    public FFRequestListener<T> getRequestResponse() {
        return requestResponse;
    }

    public void setRequestResponse(FFRequestListener<T>
requestResponse) {
        this.requestResponse = requestResponse;
    }

    public void findAll() {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();
        this.currentRequester.findAll();
    }

    public void findRecord(String id) {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();
        this.currentRequester.findRecord(id);
    }

    private void beforeRequest() {
        if (this.currentRequester == null) {
            this.currentRequester = new
FFRestRequester(this.resourceName(), this.requestRe-
sponse);
        }
    }

    public void save() {
        if (this.requestResponse == null) {
            return;
        }
        this.beforeRequest();
```

```
        if (this.id != null) {
            //update
            this.currentRequester.updateRecord(this);
        } else {
            //create
            this.currentRequester.createRecord(this);

        }
    }

    public void delete() {
        this.beforeRequest();
        if (this.id != null) {
            System.out.println(this.id);
            this.currentRequester.deleteRecord(String.-
valueOf(this.id));
        }
    }
}
```

Classe que notifica as novidades

```
public class FFRestRequester<T extends FFObject>  imple-
ments FFRequester<T> {


    private AsyncHttpClient asyncHttp = new AsyncHttp-
Client();
    private FFJSONSerializer<T> serializer;
    private String resourceName;
    private FFRequestListener<T> requestResponse;
    private FFURLBuilder urlBuilder;
    public FFRestRequester(String resourceName, FFRe-
questListener<T> requestResponse) {
        this.resourceName = resourceName;
        this.requestResponse = requestResponse;
        serializer =  new FFJSONSerializer<>(resource-
Name);

        switch (FFAPIClient.sharedClient().getServerPat-
tern()) {
            case NONE:
                this.urlBuilder = null;
                break;
            case JSONAPI:
                this.urlBuilder = new FFJSONApiURL-
Builder();
                break;
```

```
        }
    }

    /**
     * Called by the FFResource in order to fetch the
JSON for a given type and ID.
     * The findRecord method makes an Asynchronous re-
quest to a URL computed by buildURL,
     * and returns a promise for the resulting payload.
     * This method performs an HTTP GET request with the
id provided as part of the query string.
     */
    @Override
    public void findRecord(String id) {
        String url = this.urlBuilder.buildURL("find-
Record", this.resourceName, id);
        final FFRestRequester self = this;
        this.asyncHttp.get(url, new JsonHttpResponseHan-
dler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{
                ArrayList<T> response = (ArrayList<T>)
self.serializer.serializePayload(jsonObject);
                self.requestResponse.afterFindSuccess(re-
sponse);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

    /**
     * Called by the FFResource in order to fetch a JSON
array for all of the records for a given type.
     * The findAll method makes an Asynchronous (HTTP
GET) request to a URL computed by buildURL,
     * and returns a promise for the resulting payload.
     */
    @Override
    public void findAll() {
```

```
        String url = this.urlBuilder.buildURL("findAll",
this.resourceName);
        final FFRestRequester self = this;

        this.asyncHttp.get(url, new JsonHttpResponseHan-
dler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{
                ArrayList<T> response = (ArrayList<T>)
self.serializer.serializePayload(jsonObject);
                self.requestResponse.afterFindSuccess(re-
sponse);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

//    /**
//     * Called by the FFResource in order to fetch a
JSON array for the records that match a particular query.
//     * The query method makes an Asynchronous (HTTP
GET) request to a URL computed by buildURL, and returns a
//     * promise for the resulting payload.
//     * The query argument is a simple Map object that
will be passed directly to the server as parameters.
//     */
//    @Override
//    public void query(String modelName, String query) {
//
//    }

    /**
     * Called by FFResource when a newly created record
is saved via the save method on a model
     * record instance. The createRecord method serial-
izes the record and makes an Asynchronous
     * (HTTP POST) request to a URL computed by buildURL.
     * See serialize for information on how to customize
the serialized form of a record.
     */
```

```
    @Override
    public void createRecord(final T model) {
        String url = this.urlBuilder.buildURL("createRe-
cord", this.resourceName);
        final FFRestRequester self = this;

        RequestParams params = this.serializer.deserial-
ize(model);

        this.asyncHttp.post(url, params, new JsonHttpRe-
sponseHandler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{
                System.out.println(jsonObject);
                try {
                    model.id =
jsonObject.getJSONObject(self.resourceName).getInt("id");
                } catch (JSONException e) {
                    e.printStackTrace();
                }


self.requestResponse.afterSaveSuccess(model);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

    /**
     * Called by the FFResource when an existing record
is saved via the save method on a model record instance.
     * The updateRecord method serializes the record and
makes an Asynchronous (HTTP PUT) request to
     * a URL computed by buildURL.
     * See serialize for information on how to customize
the serialized form of a record.
     */
    @Override
    public void updateRecord(final T model) {
```

```
        String url = this.urlBuilder.buildURL("updateRe-
cord", this.resourceName, model.id.toString());

        final FFRestRequester self = this;

        RequestParams params = this.serializer.deserial-
ize(model);

        this.asyncHttp.put(url, params, new JsonHttpRe-
sponseHandler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{

self.requestResponse.afterSaveSuccess(model);
            }

            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }

    /**
     * Called by the FFResource when a record is deleted.
     * The deleteRecord method makes an Asynchronous
(HTTP DELETE) request to a URL computed
     * by buildURL.
     */
    @Override
    public void deleteRecord(String id) {
        String url = this.urlBuilder.buildURL("deleteRe-
cord", this.resourceName, id);
        final FFRestRequester self = this;

        this.asyncHttp.delete(url, new JsonHttpResponse-
Handler() {
            @Override
            public void onSuccess(JSONObject jsonObject)
{

self.requestResponse.afterDeleteSuccess("204");
            }
```

```
            @Override
            public void onFailure(int statusCode, Throw-
able throwable, JSONObject error) {
                System.out.println(error);
            }
        });
    }


}
```

Classe de interface de observação

```
public interface FFRequestListener<T> {

    /**
     * This method is for running a code after operation
     * find become success
     *
     * @param objects list of return objects if find suc-
cess
     */
    void afterFindSuccess(ArrayList<T> objects);

    /**
     * This method is for running a code after operation
     * save become success
     *
     * @param object a return object if save success
     */
    void afterSaveSuccess(T object);

    /**
     * This method is for running a code after operation
     * delete become success
     *
     * @param status an actual status if delete success
     */
    void afterDeleteSuccess(String status);

    /**
     * This method is for running a code after operation
     * find return an error.
     *
     * @param error an error that causes find operation
fail
     */
    void afterFindError(FFError error);
```

```
    /**
     * This method is for running a code after operation
     * save return an error.
     *
     * @param error an error that causes save operation
fail
     */
    void afterSaveError(FFError error);

    /**
     * This method is for running a code after operation
     * delete return an error.
     *
     * @param error an error that causes delete operation
fail
     */
    void afterDeleteError(FFError error);
}
```

Classe que implementa a observação

```
public class MainActivity extends AppCompatActivity im-
plements FFRequestListener<User> {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Set API settings
        FFAPIClient apiSetting = new
FFAPIClient("192.168.0.21:3000", "none", ServerPattern.J-
SONAPI);
        User user = new User();
        user.setRequestResponse(this);
        user.findRecord("1");
    }

    @Override
    public void afterFindSuccess(ArrayList<User> objects)
{
        User u = objects.get(0);
        u.setRequestResponse(this);
        System.out.println("######## Velho ##########");
        System.out.println(u.name);
        u.delete();
    }
```

```
    @Override
    public void afterSaveSuccess(User object) {
        System.out.println("######## Novo ##########");
        System.out.println(object.name);
    }

    @Override
    public void afterDeleteSuccess(String status) {

    }

    @Override
    public void afterFindError(FFError error) {

    }

    @Override
    public void afterSaveError(FFError error) {

    }

    @Override
    public void afterDeleteError(FFError error) {

    }
}
```