Chiappelloni Nicolas
Projet d’analyse et de conception
3 ème année
Professeur: Mr. Ninforge-ory
ECI

20 octobre 2019

TOLL MANAGER

Agile manifest

Table of contents

Application Architecture

The application is sliced into 2 big parts, the graphic part and the business part linked by the *applicationLifTime* interface, in this interface we put every use case which the graphic application uses to deal with the business code (see the ** CQS borrow model **).

Business Layer

In the business layer we have 3 main sections.

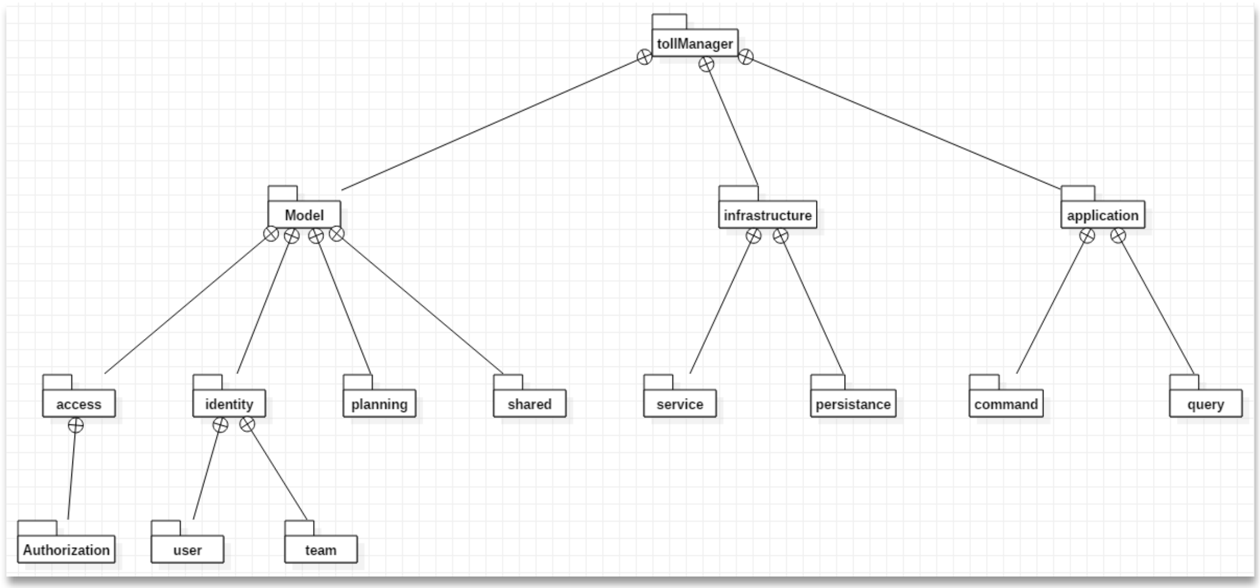


Figure 1business layer

Model

In model section we have the Domain of the application where each sub domain is packaged.

Each sub domain must be agnostic from other sub domain for example the sub domain “access” doesn’t care about the sub domain “identity” or “planning”, But if for some measured reason something needed to be shared between two subdomains the package *shared* exists, but the reason to do that should be precise (example an interface observable or class responsible to generate id, …).

(don’t confuse model from MVC and here from DDD it’s same purpose but in DDD the model has some responsibility an rules than MVC doesn’t precise).

Infrastructure

Here we put everything that depends of a specific architecture, like the repository(postgres, mysql, …) or service like cryptography(md5,sha-1,…)

Application

Here we put what is concerned by the launching of the application like *Command* or *Query* for CQS or The ApplicationLifeTime implementation.

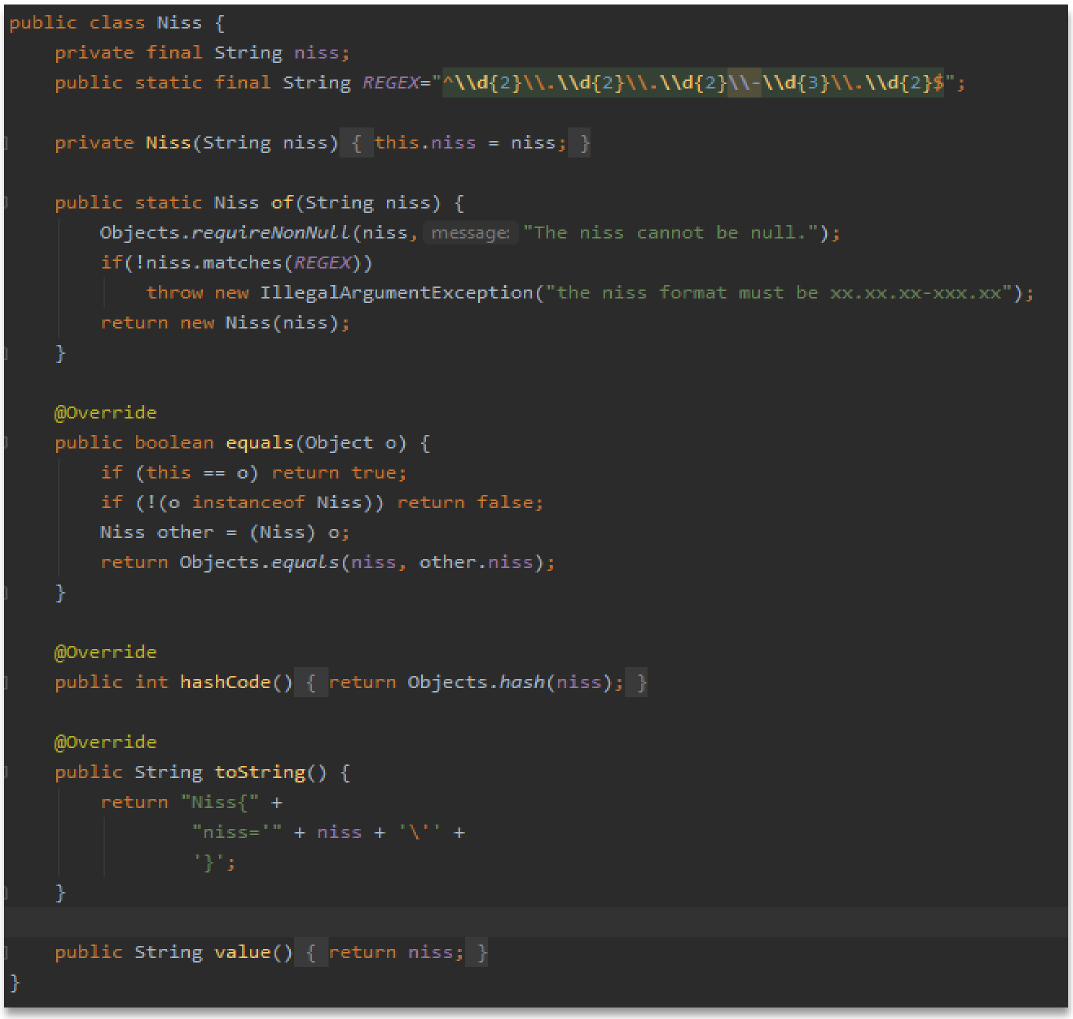
The application layer is like the cement that link all part of the model

Domain Driven Development

Some of strategies are borrow from the domain driven design, we don't respect every concept (cause too complex) but some interesting concepts are used.

Value Object

When you need something like the name of a person don't use a string to represent the name but use a value object, A small object **immutable** that represents a simple entity whose equality is not based on identity.



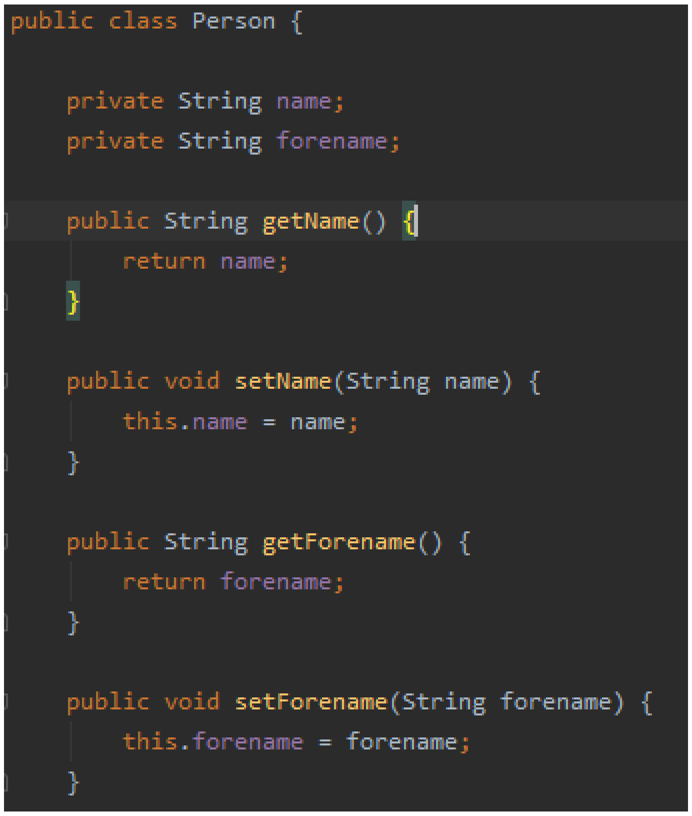
```java
public class Niss {
    private final String niss;
    public static final String REGEX="^\\d{2}\\.\\d{2}\\.\\d{2}\\-\\d{3}\\.\\d{2}$";

    private Niss(String niss) { this.niss = niss; }

    public static Niss of(String niss) {
        Objects.requireNonNull(niss, message: "The niss cannot be null.");
        if(!niss.matches(REGEX))
            throw new IllegalArgumentException("the niss format must be xx.xx.xx-xxx.xx");
        return new Niss(niss);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Niss)) return false;
        Niss other = (Niss) o;
        return Objects.equals(niss, other.niss);
    }

    @Override
    public int hashCode() { return Objects.hash(niss); }

    @Override
    public String toString() {
        return "Niss{" +
                "niss='" + niss + '\'' +
                '}';
    }

    public String value() { return niss; }
}
```

Figure 2: example of value object

Setter is not allowed! if you want to edit a value object then you need to create a new value object! (avoid side effect).

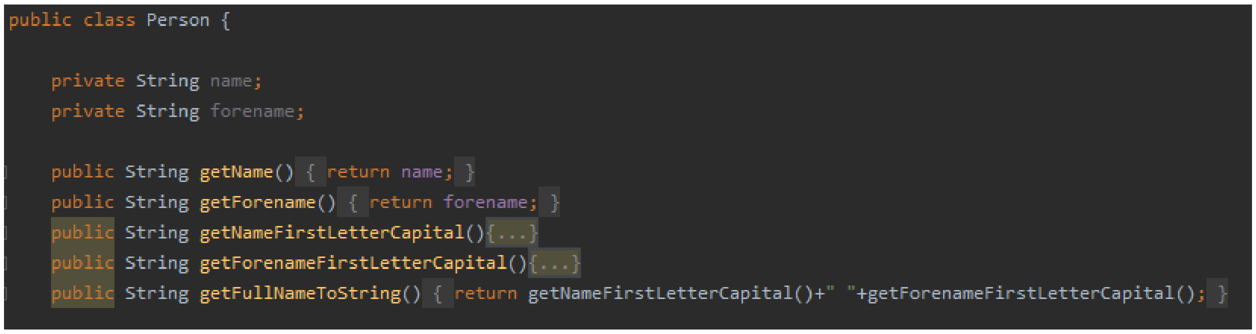
No anemic object

An object anemic is an object does nothing and let other object like service to do for its, it’s not an OOP practice, an object is created to do something!



```java
public class Person {

    private String name;
    private String forename;

    public String getName() {
        return name;
    }

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

    public String getForename() {
        return forename;
    }

    public void setForename(String forename) {
        this.forename = forename;
    }
```

Figure 3 anemic model



```java
public class Person {

    private String name;
    private String forename;

    public String getName() { return name; }
    public String getForename() { return forename; }
    public String getNameFirstLetterCapital(){...}
    public String getForenameFirstLetterCapital(){...}
    public String getFullNameToString() { return getNameFirstLetterCapital()+" "+getForenameFirstLetterCapital(); }
```

Figure 4 no anemic model

As you can see on the picture 1, class does nothing just getter and setter against the model on figure 2 non anemic model where the object does something.

Null Object Pattern

Avoid null instead use the Null object pattern.

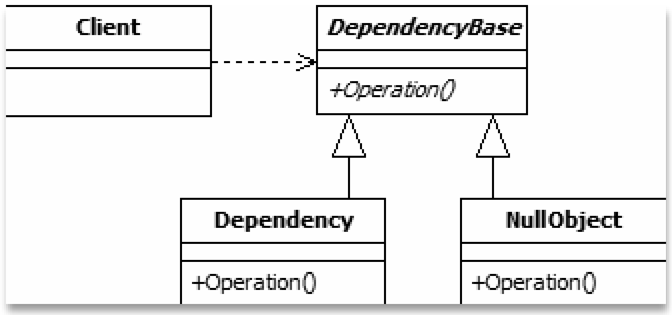
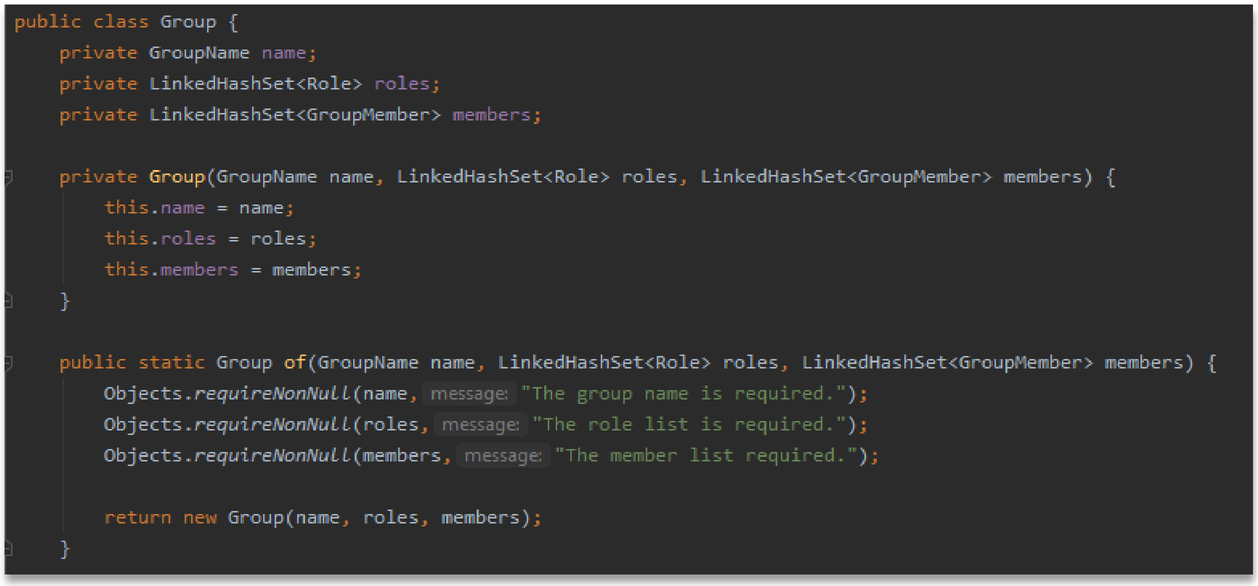


Figure 5null object pattern

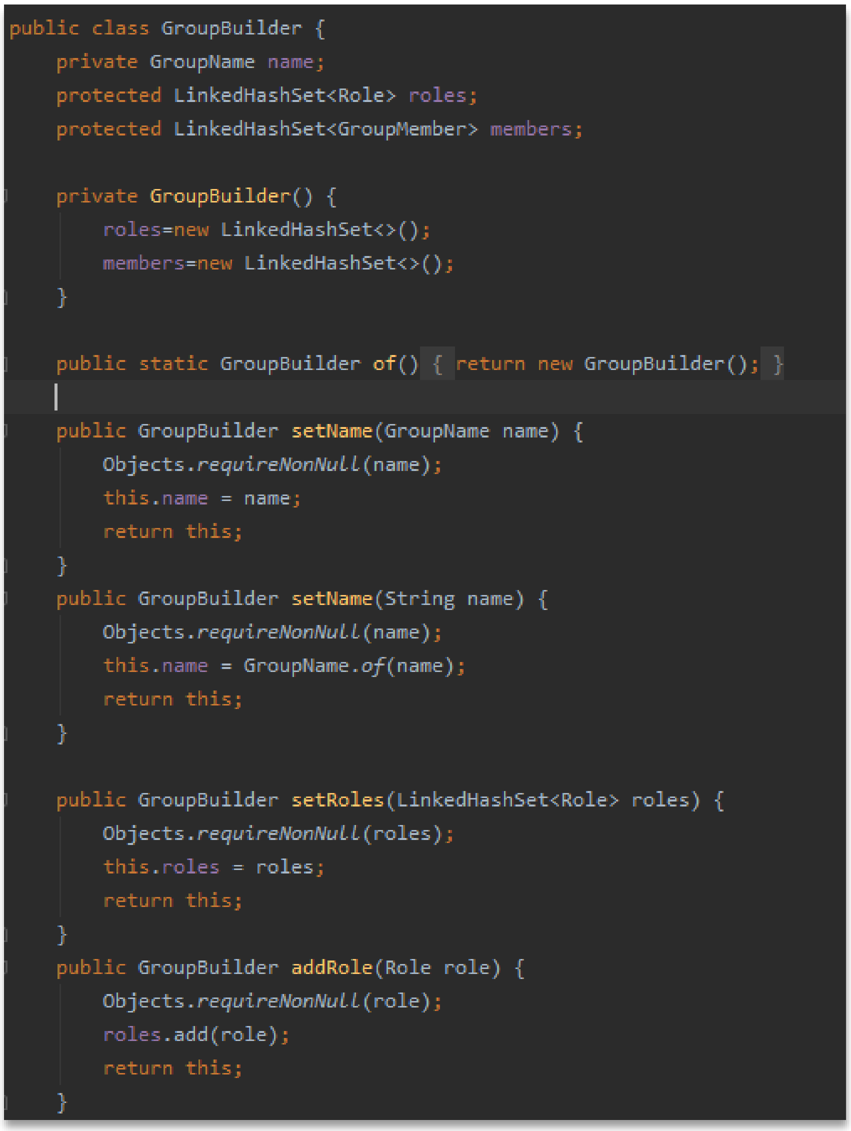
Factory method

In the business code, constructor is not allowed, we need to use the factory method pattern and for complex object, the Builder pattern



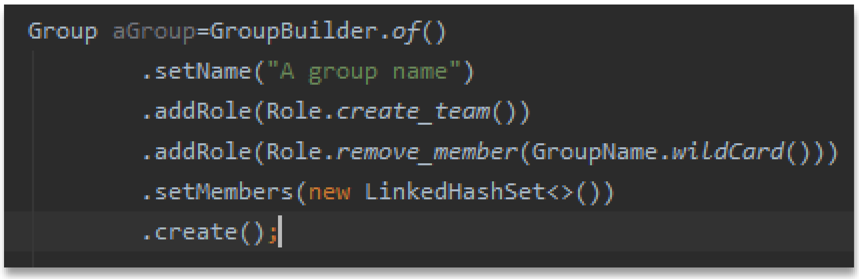
```
public class Group {
    private GroupName name;
    private LinkedHashSet<Role> roles;
    private LinkedHashSet<GroupMember> members;

    private Group(GroupName name, LinkedHashSet<Role> roles, LinkedHashSet<GroupMember> members) {
        this.name = name;
        this.roles = roles;
        this.members = members;
    }

    public static Group of(GroupName name, LinkedHashSet<Role> roles, LinkedHashSet<GroupMember> members) {
        Objects.requireNonNull(name, message: "The group name is required.");
        Objects.requireNonNull(roles, message: "The role list is required.");
        Objects.requireNonNull(members, message: "The member list required.");

        return new Group(name, roles, members);
    }
```

Figure 6factory method instead constructor



```java
public class GroupBuilder {
    private GroupName name;
    protected LinkedHashSet<Role> roles;
    protected LinkedHashSet<GroupMember> members;

    private GroupBuilder() {
        roles=new LinkedHashSet<>();
        members=new LinkedHashSet<>();
    }

    public static GroupBuilder of() { return new GroupBuilder(); }

    public GroupBuilder setName(GroupName name) {
        Objects.requireNonNull(name);
        this.name = name;
        return this;
    }
    public GroupBuilder setName(String name) {
        Objects.requireNonNull(name);
        this.name = GroupName.of(name);
        return this;
    }

    public GroupBuilder setRoles(LinkedHashSet<Role> roles) {
        Objects.requireNonNull(roles);
        this.roles = roles;
        return this;
    }
    public GroupBuilder addRole(Role role) {
        Objects.requireNonNull(role);
        roles.add(role);
        return this;
    }
```

Figure 7 GroupBuilder



```java
Group aGroup=GroupBuilder.of()
        .setName("A group name")
        .addRole(Role.create_team())
        .addRole(Role.remove_member(GroupName.wildCard()))
        .setMembers(new LinkedHashSet<>())
        .create();
```

Figure 8use a builder

Domain separation

A domain it’s a group of classes that can work together without other object extern to the domain, for example the planning domain don’t care about the access, but when the ApplicationLifeTime will run, then the application can deal with access and planning together. See sub domain like a brick and the Application layer like the wall.

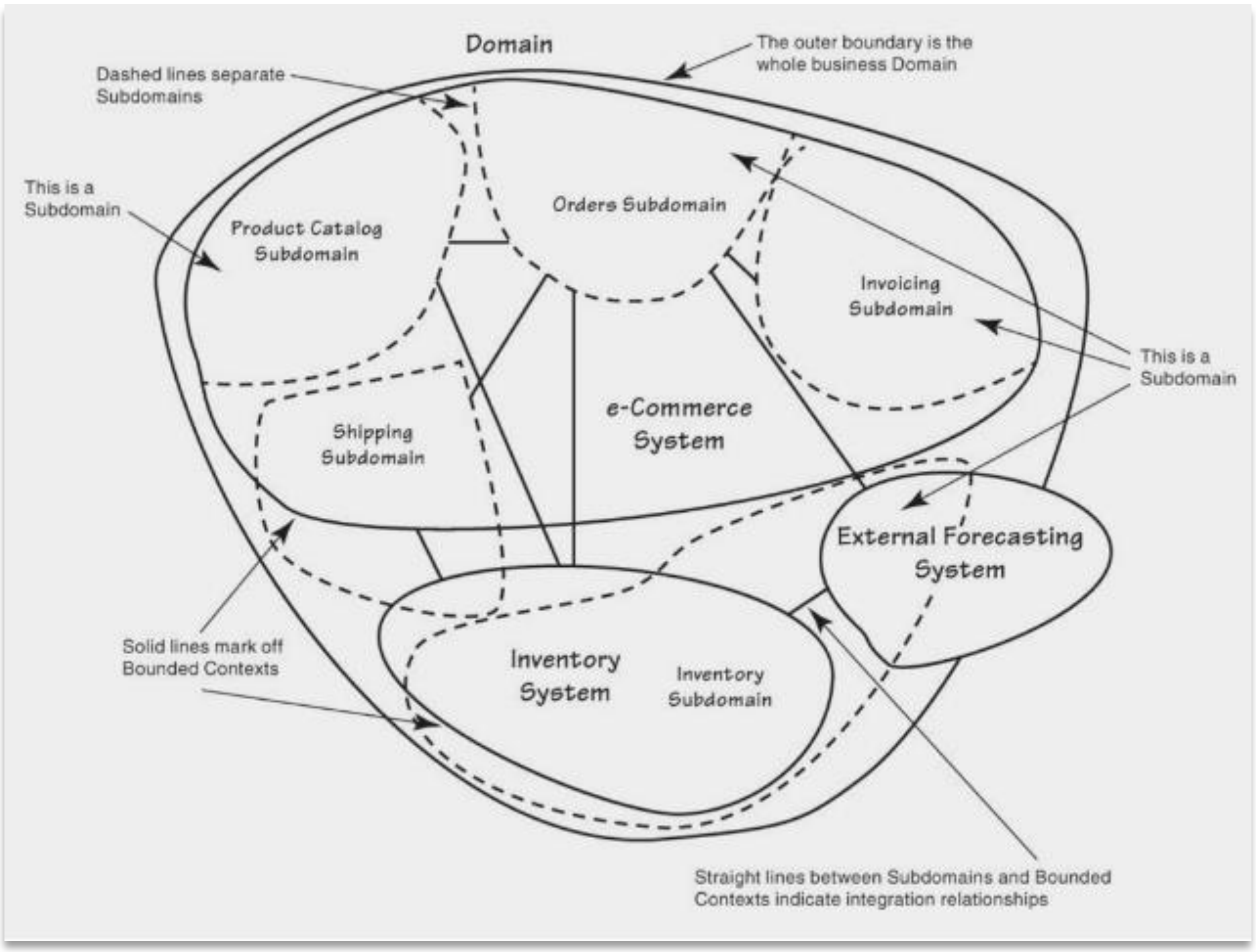

Domain
The outer boundary is the whole business Domain
Dashed lines separate Subdomains
This is a Subdomain
Product Catalog Subdomain
Orders Subdomain
Invoicing Subdomain
This is a Subdomain
Shipping Subdomain
e-Commerce System
External Forecasting System
Solid lines mark off Bounded Contexts
Inventory System
Inventory Subdomain
Straight lines between Subdomains and Bounded Contexts indicate integration relationships

Figure 9Domain example

Test Driven Development

Perhaps the most important part of this Manifest, No one nano octet of code should be writing without a test before into the business layer ! The 100% code coverage is not required (mean don't test toString and getter method) but It's very important to assure one piece of code appended or edited don't make a side effect on de business code !

As a reminder, when you want to write code, write the test before, launch the test to failed it, make code to pass test to green and refactor as far as it's possible.

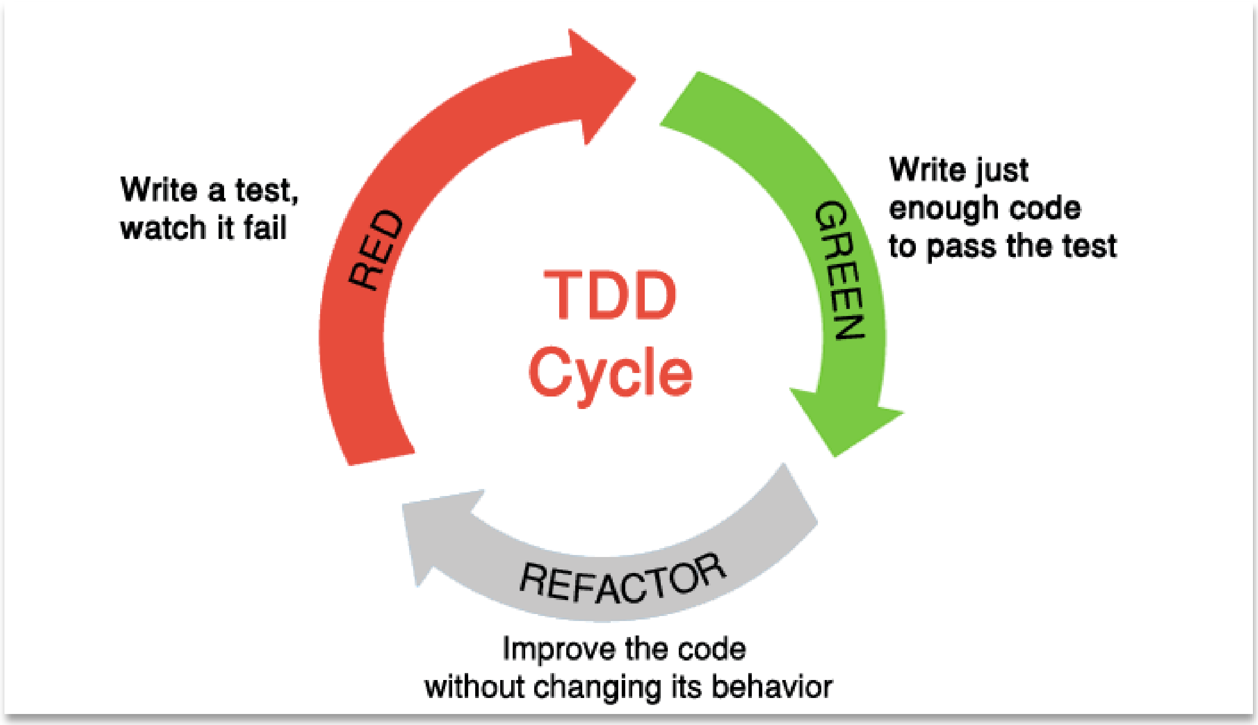
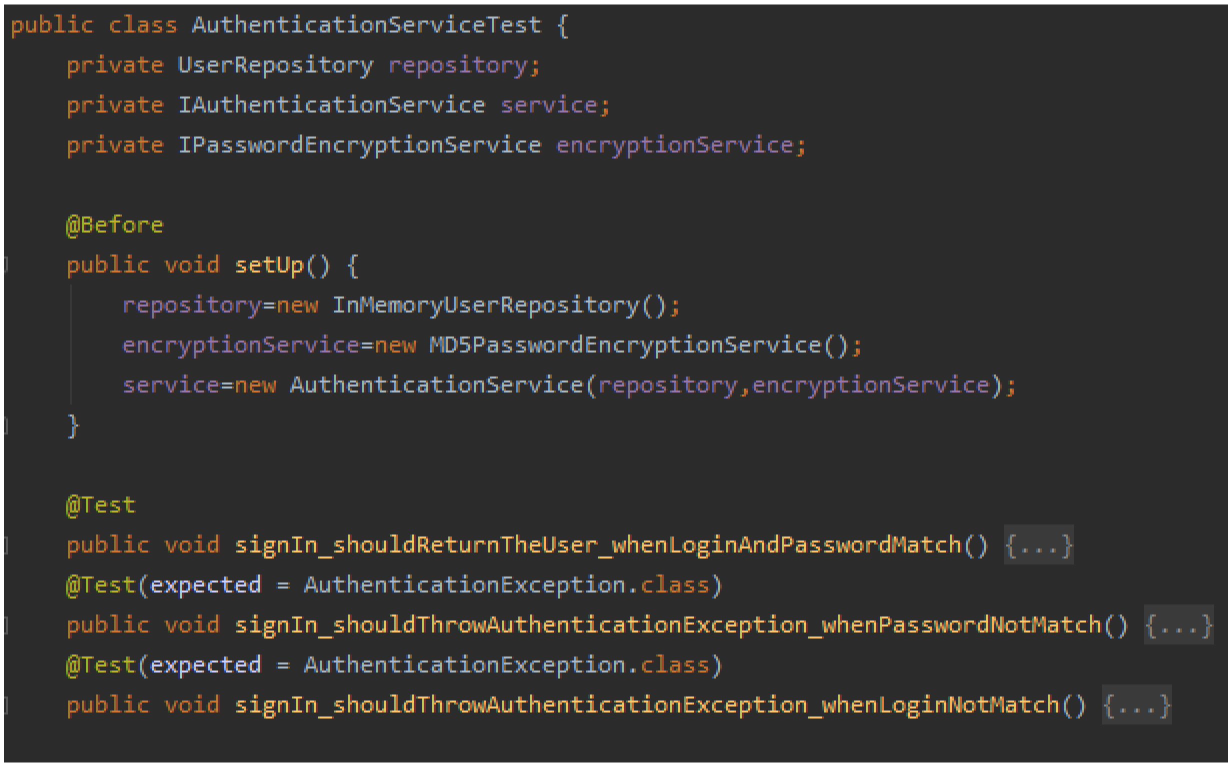


Figure 10tdd cycle



```java
public class AuthenticationServiceTest {
    private UserRepository repository;
    private IAuthenticationService service;
    private IPasswordEncryptionService encryptionService;

    @Before
    public void setUp() {
        repository=new InMemoryUserRepository();
        encryptionService=new MD5PasswordEncryptionService();
        service=new AuthenticationService(repository,encryptionService);
    }

    @Test
    public void signIn_shouldReturnTheUser_whenLoginAndPasswordMatch() {...}
    @Test(expected = AuthenticationException.class)
    public void signIn_shouldThrowAuthenticationException_whenPasswordNotMatch() {...}
    @Test(expected = AuthenticationException.class)
    public void signIn_shouldThrowAuthenticationException_whenLoginNotMatch() {...}
```

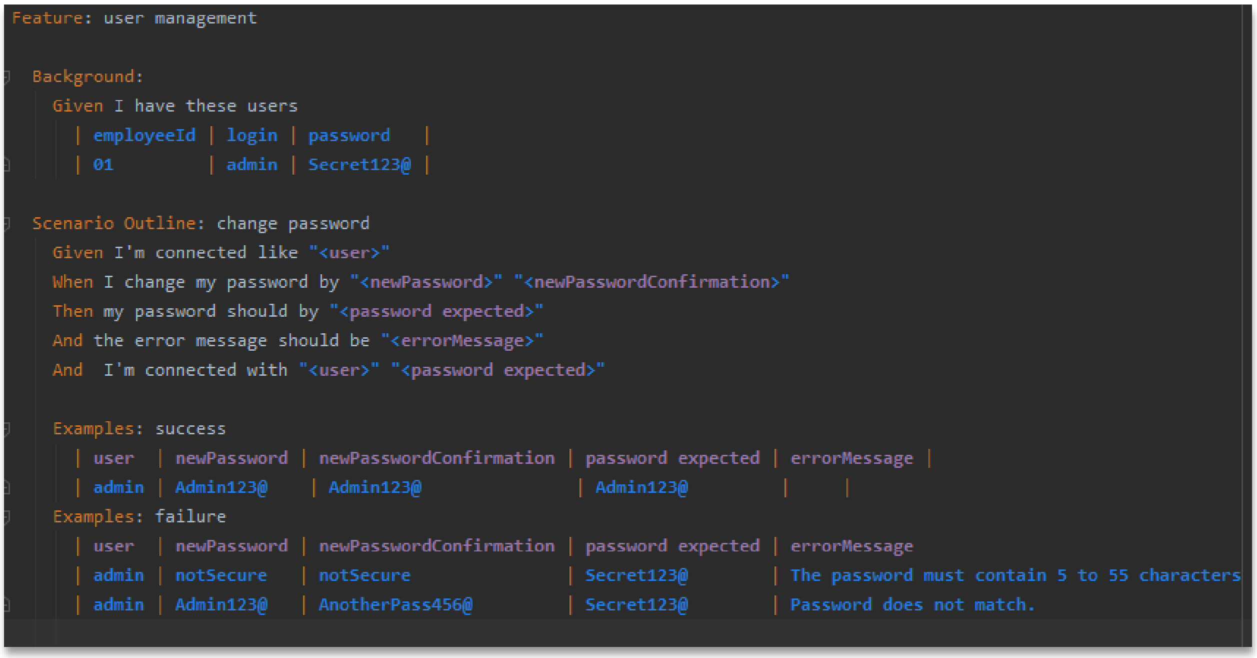
Figure 11: TDD class example

As you can see, I began by test the simple thing, return user match login and password, when it's pass, I create a new test where I want to throw exception, I pass the test and assure my first test stay green, … If I need to make one modification to my code a launch my test that assure I haven't a side effect…

Behavior Driven Development

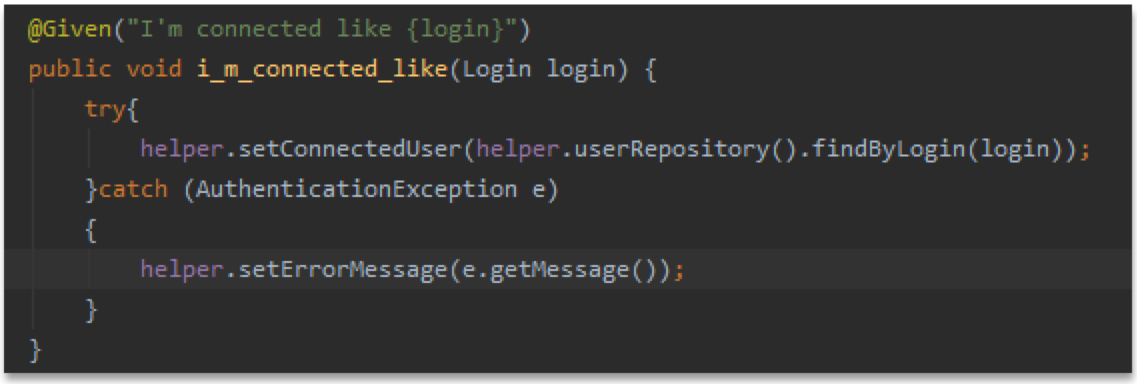
Another big part of the manifest it’s the BDD, when you want to append a feature to the Business code Then it’s very important to write a Cucumber file that explains the feature! If you cannot write a feature maybe that mean you don’t need to it…

Example of feature:



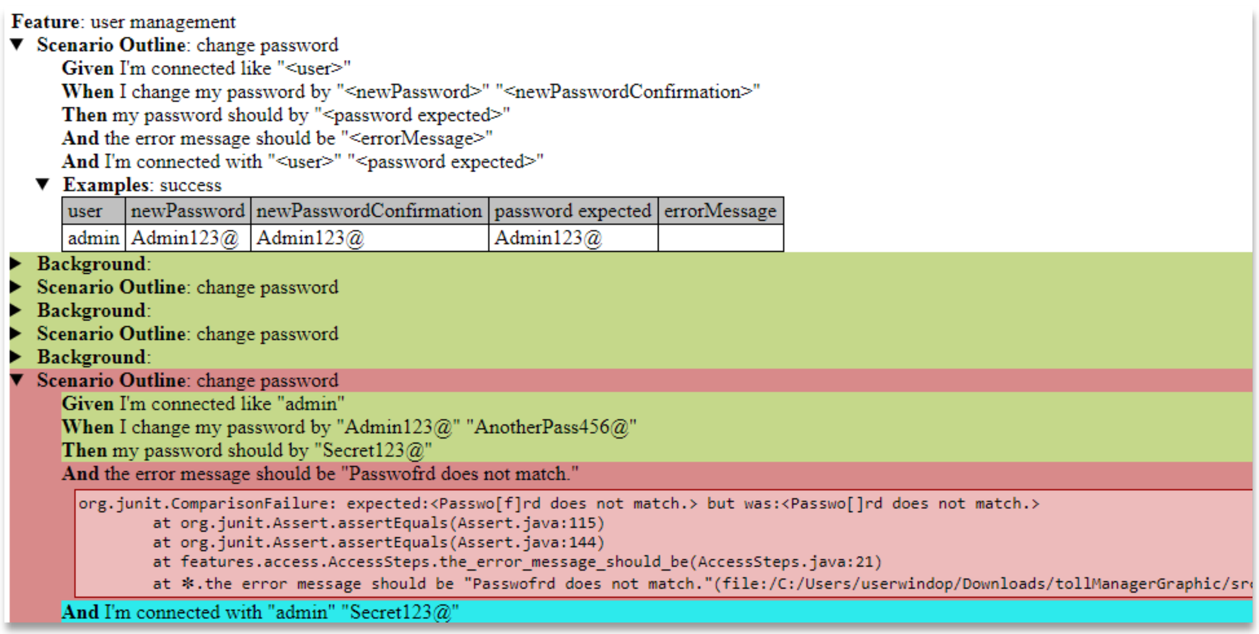
```
Feature: user management

  Background:
    Given I have these users
      | employeeId | login | password   |
      | 01         | admin | Secret123@ |

  Scenario Outline: change password
    Given I'm connected like "<user>"
    When I change my password by "<newPassword>" "<newPasswordConfirmation>"
    Then my password should by "<password expected>"
    And the error message should be "<errorMessage>"
    And  I'm connected with "<user>" "<password expected>"

    Examples: success
      | user  | newPassword | newPasswordConfirmation | password expected | errorMessage |
      | admin | Admin123@    | Admin123@               | Admin123@         |      |
    Examples: failure
      | user  | newPassword | newPasswordConfirmation | password expected | errorMessage
      | admin | notSecure   | notSecure               | Secret123@        | The password must contain 5 to 55 characters
      | admin | Admin123@   | AnotherPass456@         | Secret123@        | Password does not match.
```

Figure 12:BDD example



```java
@Given("I'm connected like {login}")
public void i_m_connected_like(Login login) {
    try{
        helper.setConnectedUser(helper.userRepository().findByLogin(login));
    }catch (AuthenticationException e)
    {
        helper.setErrorMessage(e.getMessage());
    }
}
```

Figure 13first step in java



Feature: user management
▼ **Scenario Outline**: change password
Given I'm connected like "<user>"
When I change my password by "<newPassword>" "<newPasswordConfirmation>"
Then my password should by "<password expected>"
And the error message should be "<errorMessage>"
And I'm connected with "<user>" "<password expected>"
▼ **Examples**: success

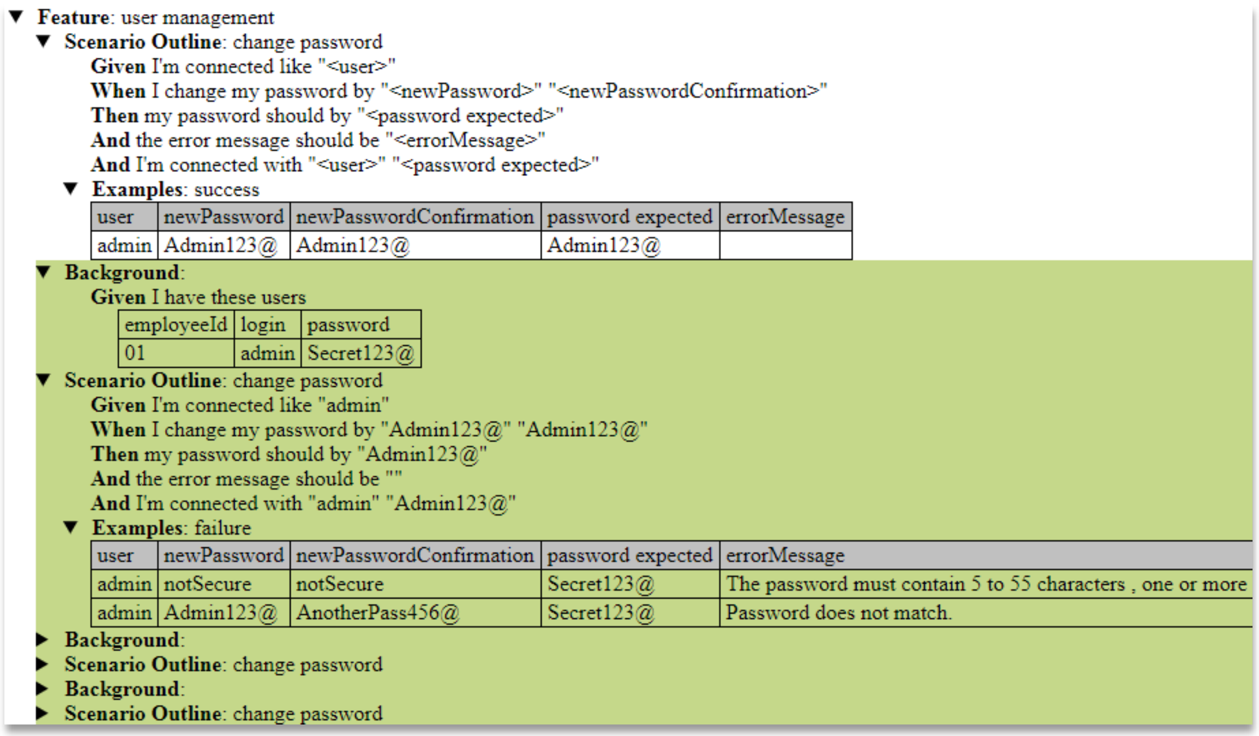
user	newPassword	newPasswordConfirmation	password expected	errorMessage
admin	Admin123@	Admin123@	Admin123@	

▶ **Background**:
▶ **Scenario Outline**: change password
▶ **Background**:
▶ **Scenario Outline**: change password
▶ **Background**:
▼ **Scenario Outline**: change password
Given I'm connected like "admin"
When I change my password by "Admin123@" "AnotherPass456@"
Then my password should by "Secret123@"
And the error message should be "Passwofrd does not match."

```
org.junit.ComparisonFailure: expected:<Passwo[f]rd does not match.> but was:<Passwo[]rd does not match.>
	at org.junit.Assert.assertEquals(Assert.java:115)
	at org.junit.Assert.assertEquals(Assert.java:144)
	at features.access.AccessSteps.the_error_message_should_be(AccessSteps.java:21)
	at ✻.the error message should be "Passwofrd does not match."(file:/C:/Users/userwindop/Downloads/tollManagerGraphic/sr
```

And I'm connected with "admin" "Secret123@"

Figure 14BDD report with one error, step not finish



▼ **Feature**: user management
▼ **Scenario Outline**: change password
Given I'm connected like "<user>"
When I change my password by "<newPassword>" "<newPasswordConfirmation>"
Then my password should by "<password expected>"
And the error message should be "<errorMessage>"
And I'm connected with "<user>" "<password expected>"
▼ **Examples**: success

user	newPassword	newPasswordConfirmation	password expected	errorMessage
admin	Admin123@	Admin123@	Admin123@	

▼ **Background**:
Given I have these users

employeeId	login	password
01	admin	Secret123@

▼ **Scenario Outline**: change password
Given I'm connected like "admin"
When I change my password by "Admin123@" "Admin123@"
Then my password should by "Admin123@"
And the error message should be ""
And I'm connected with "admin" "Admin123@"
▼ **Examples**: failure

user	newPassword	newPasswordConfirmation	password expected	errorMessage
admin	notSecure	notSecure	Secret123@	The password must contain 5 to 55 characters , one or more
admin	Admin123@	AnotherPass456@	Secret123@	Password does not match.

▶ **Background**:
▶ **Scenario Outline**: change password
▶ **Background**:
▶ **Scenario Outline**: change password

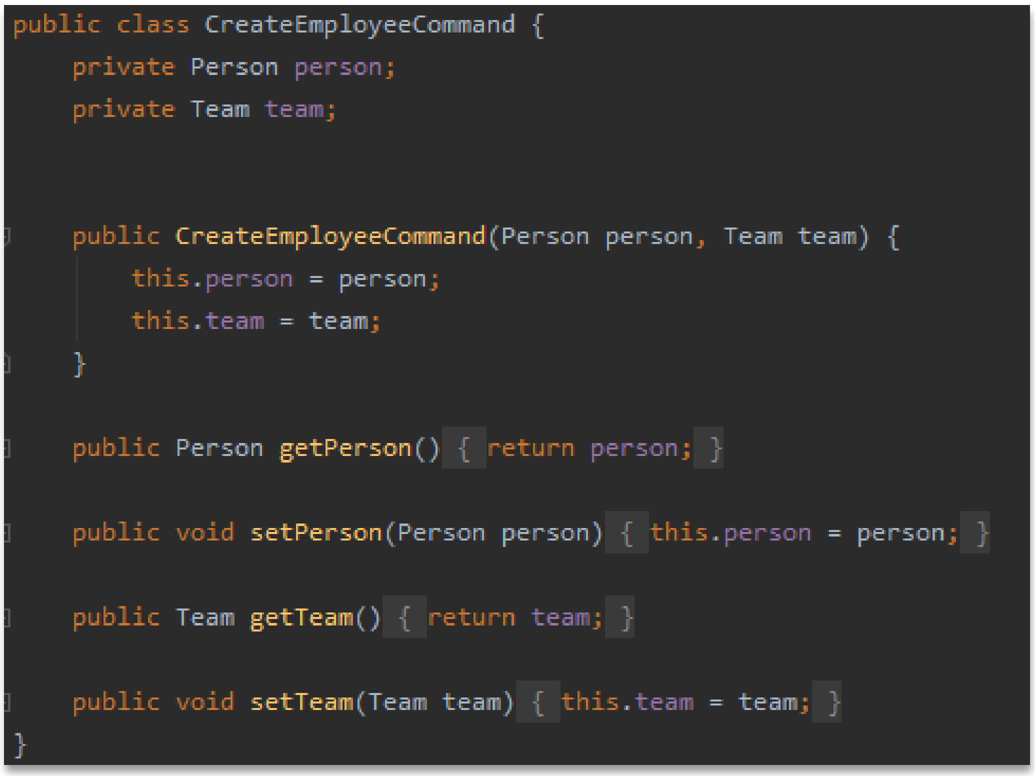
Figure 15BDD report, step finish

Communicate with graphic

To communicate with the graphic (means out of the business layer), we use a technic borrow to the *CQS*, the Command and Query.

Command:

A command is an action like ‘create an employee’, ‘change my hair color’, …

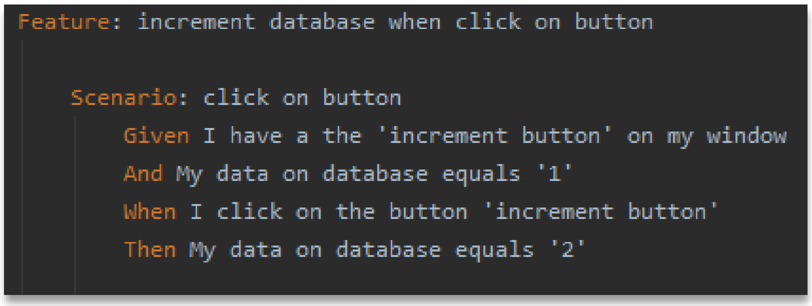


```java
public class CreateEmployeeCommand {
    private Person person;
    private Team team;


    public CreateEmployeeCommand(Person person, Team team) {
        this.person = person;
        this.team = team;
    }

    public Person getPerson() { return person; }

    public void setPerson(Person person) { this.person = person; }

    public Team getTeam() { return team; }

    public void setTeam(Team team) { this.team = team; }
}
```

Figure 16Command example

example:

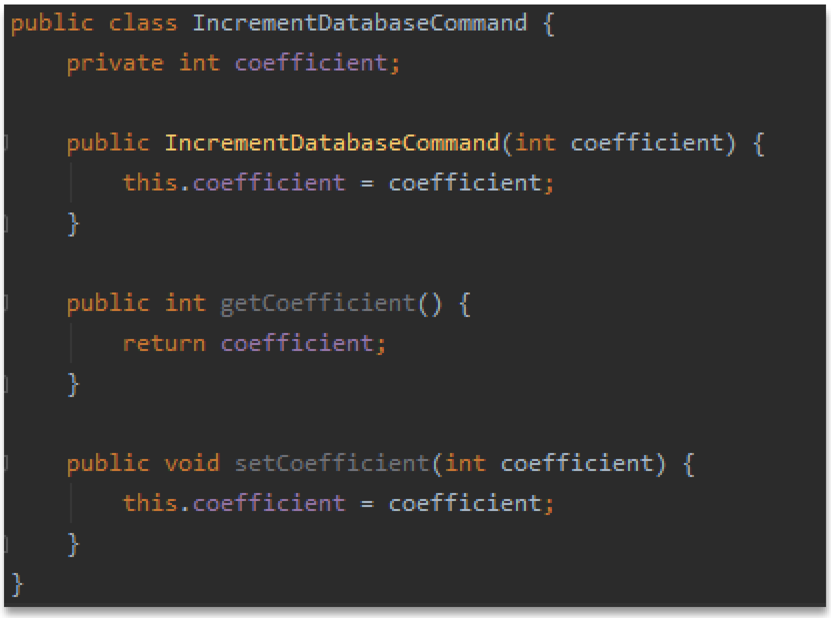
A simple feature, when I click on the button the data is incremented.



```
Feature: increment database when click on button

  Scenario: click on button
    Given I have a the 'increment button' on my window
    And My data on database equals '1'
    When I click on the button 'increment button'
    Then My data on database equals '2'
```

Figure 17: feature example command

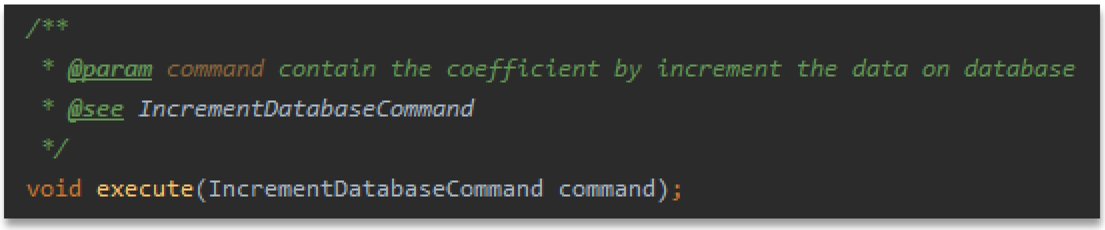
To make an action into the database I create a class where the end of the name is *Command* and the class is located on tollManager.application.command package.



```
public class IncrementDatabaseCommand {
    private int coefficient;

    public IncrementDatabaseCommand(int coefficient) {
        this.coefficient = coefficient;
    }

    public int getCoefficient() {
        return coefficient;
    }

    public void setCoefficient(int coefficient) {
        this.coefficient = coefficient;
    }
}
```

Figure 18 IncrementDataBaseCommand

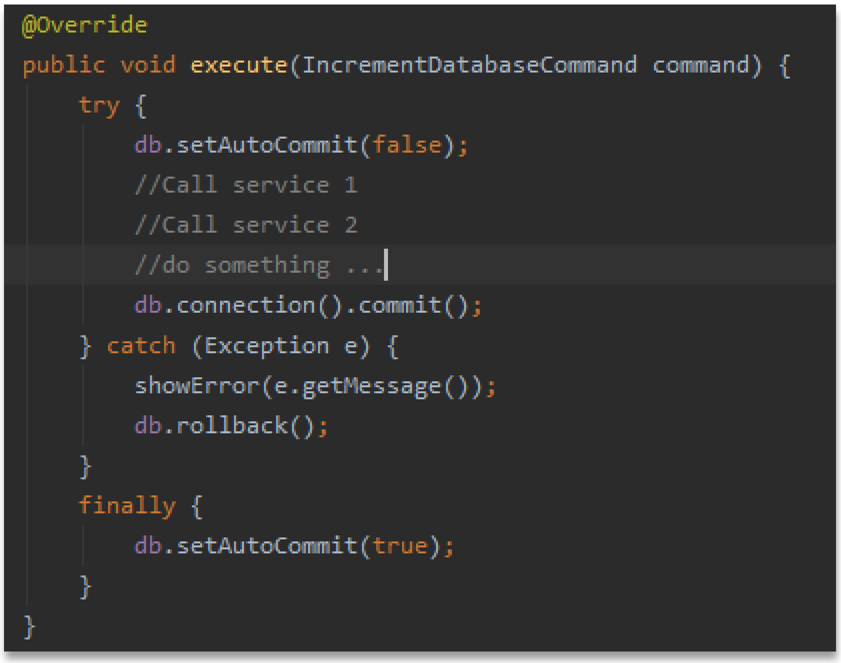
After the class creation, I create the function *execute* into the *ApplicationLifeTime* that take as argument the *IncrementDataBaseCommand*



```
/**
 * @param command contain the coefficient by increment the data on database
 * @see IncrementDatabaseCommand
 */
void execute(IncrementDatabaseCommand command);
```

Figure 19 Execute function on ApplicationLifeTime

and in my implementation of ApplicationLifeTime, I call the concerned business.



```
@Override
public void execute(IncrementDatabaseCommand command) {
    try {
        db.setAutoCommit(false);
        //Call service 1
        //Call service 2
        //do something ...
        db.connection().commit();
    } catch (Exception e) {
        showError(e.getMessage());
        db.rollback();
    }
    finally {
        db.setAutoCommit(true);
    }
}
```

Figure 20 Execute implementation

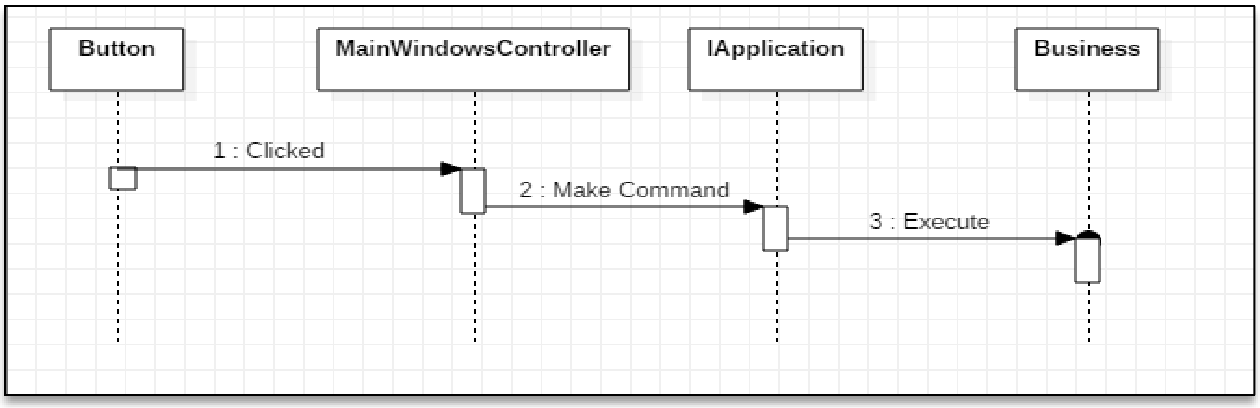
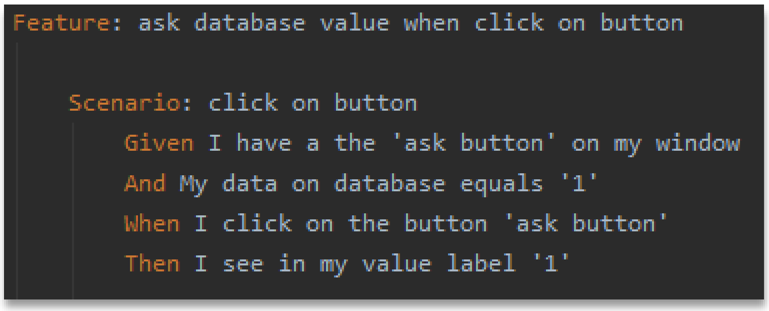


Figure 21: Command sequence

Query

A Query is an ask on the database or on the business layer, it does nothing , the business should reply something.

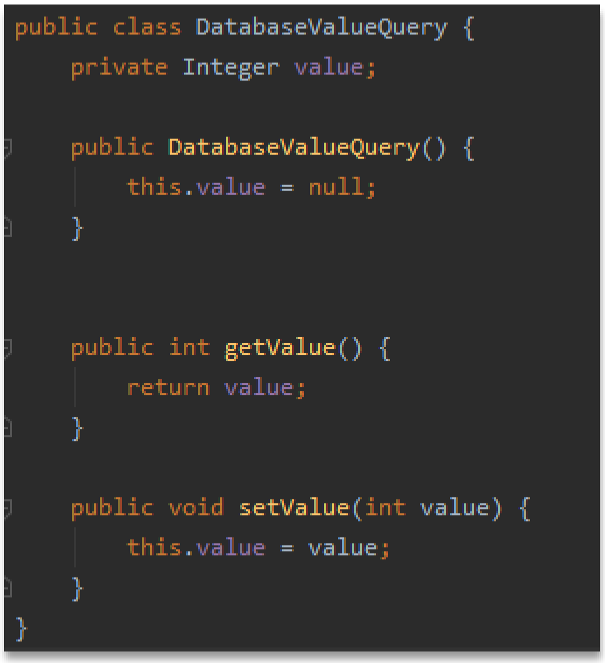
example:



```
Feature: ask database value when click on button

  Scenario: click on button
    Given I have a the 'ask button' on my window
    And My data on database equals '1'
    When I click on the button 'ask button'
    Then I see in my value label '1'
```

Figure 22query feature

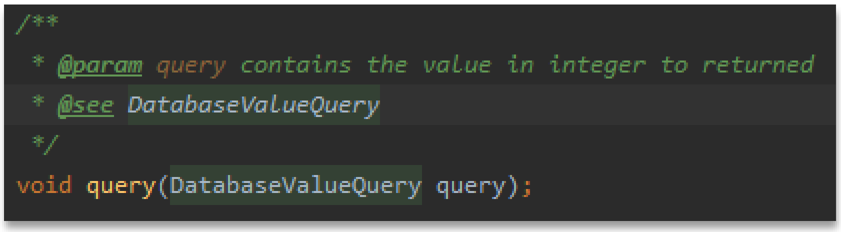
To make a query to the database I create a class where the end of the name is *Query* and the class is located on tollManager.application.Query package.



```
public class DatabaseValueQuery {
    private Integer value;

    public DatabaseValueQuery() {
        this.value = null;
    }


    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }
}
```

Figure 23DatabaseValueQuery

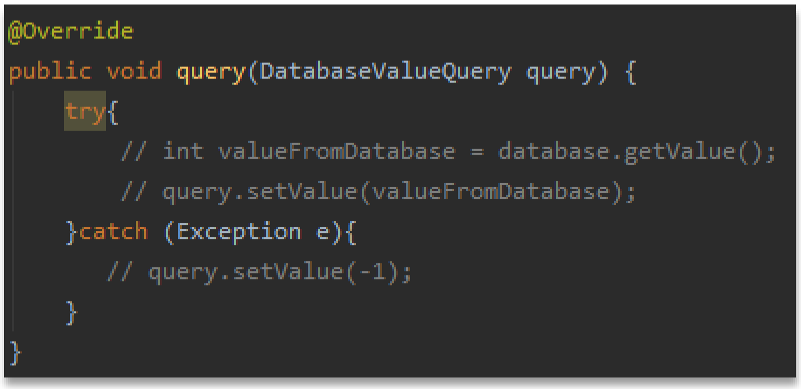
After the class creation, I create the function *query* into the *ApplicationLifeTime* that take as argument the DatabaseValueQuery



```java
/**
 * @param query contains the value in integer to returned
 * @see DatabaseValueQuery
 */
void query(DatabaseValueQuery query);
```

Figure 24query in ApplicationLifeTime

and in my implementation of ApplicationLifeTime, I call the concerned business, and I put the value in the query in argument.



```java
@Override
public void query(DatabaseValueQuery query) {
    try{
        // int valueFromDatabase = database.getValue();
        // query.setValue(valueFromDatabase);
    }catch (Exception e){
       // query.setValue(-1);
    }
}
```

Figure 25Query implementation

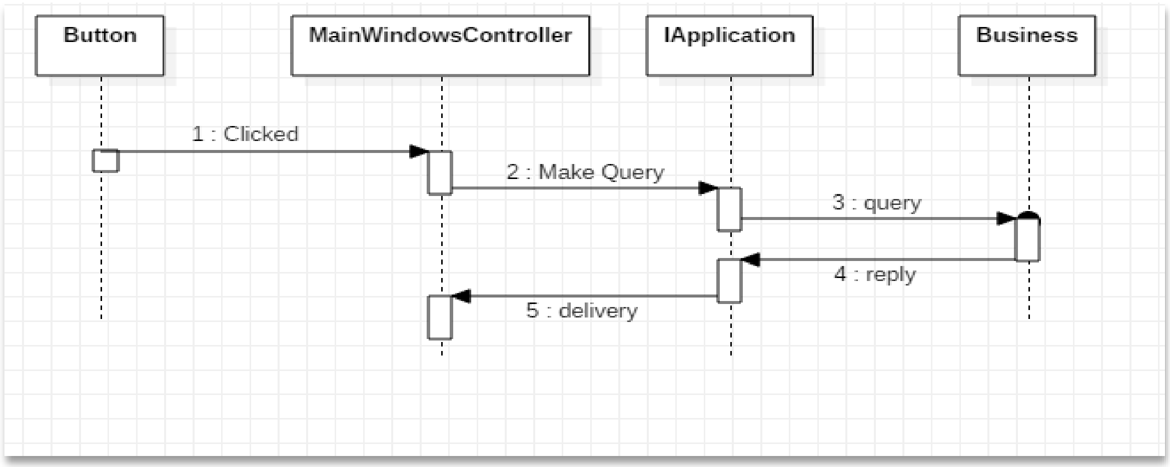


Figure 26: query execution

The advantage to use Command an Query it's a common way to deal between graphic and business without dependency from graphic or business!

Also, a big advantage It's Command and Query are incremental, so we can build a complex Query & Command during their journey.

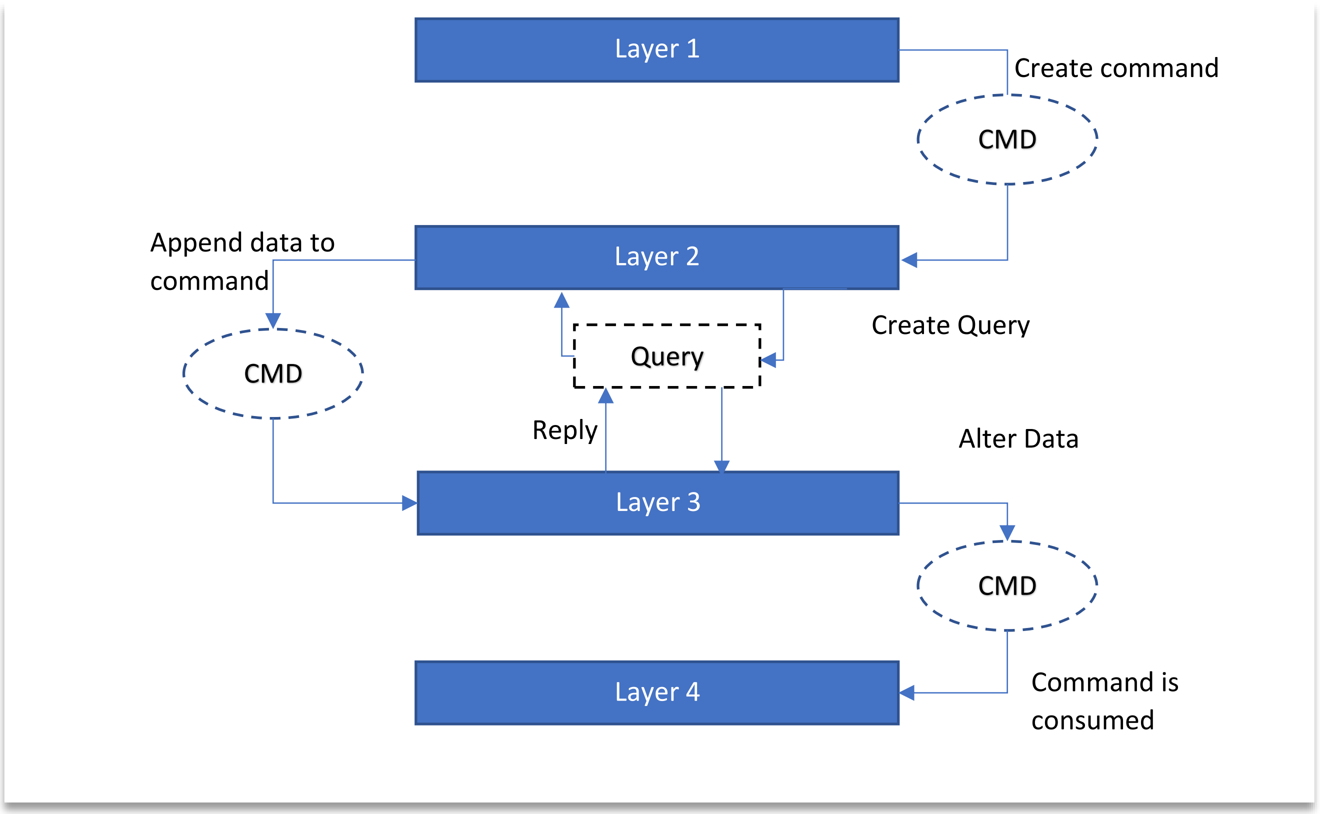


Figure 27Example of complex path of a command