
epam
BDD

Imperative style

The imperative style uses highly reusable granular steps which outlines much of the user interface.

```
Story: Animal Submission

  As a Zoologist
  I want to add a new animal to the site
  So that I can share my animal knowledge with the community

  Scenario: successful submission
  Given I'm on the animal creation page

  When I fill in Name with 'Alligator'
  And select Phylum as 'Chordata'
  And fill in Animal Class with 'Sauropsida'
  And fill in Order with 'Crocodilia'
  And fill in Family with 'Alligatoridae'
  And fill in Genus with 'Alligator'
  And check Lay Eggs
  And click the Create button

  Then I should see the notice 'Thank you for your animal submission!'
  And the page should include the animal's name, phylum, animal class, order, family, and genus
```

Declarative style

Declarative style is more aligned with User Stories in the agile sense having more of the "token for conversation" feel to it.

The first thing that you should observe about this style is how much smaller it is than the imperative one.

The imperative style tends to produce noisy scenarios that drown out the signal. With the declarative style the goal of the scenario remains clear.

When a new field is added to the form the scenario does not have to be modified.

Declarative style

```
Story: Animal Submission

  As a Zoologist
  I want to add a new animal to the site
  So that I can share my animal knowledge with the community

  Scenario: successful submission
  Given I'm on the animal creation page

  When I add a new animal

  Then I should see the page for my newly created animal
  And the notice 'Thank you for your animal submission!'
```

Create Object

Feature:

- As End User
- I want have possibility to create objects in my systems
- So that I would be able to add object based on my needs

Scenario:

- 1. Login to Equity Institutional application.
- 2. Under Deal menu, click on Create Deal.
- 3. Click Create Issuer button.
- 4. "Enter Issuer Name: <today's date (mm/dd/yyyy)> QA WFA EQ Smoke1 <build no.>
- Select Industry: Financial Services - U.S. Banks
- Enter Ticker: QA <today's date (mmddyyyy)> EQ1"
- 5. Click Create.

Create Object

- 6. Select/Verify Deal Type as Common Stock
- 7. Enter the Deal code: QA<today's date (mmddyyyy)>
- 8. Select Deal Class: IPO
- 9. Select/Verify Deal Class dropdown: Retail
- 10. Verify Industry: Financial Services - U.S. Banks
- 11. Enter Description: Test Deal Description <Deal Name>
- 12. Verify default selection for Restrictions: Exempt
- 13. Enter Expected Offering Date 'EnterTodays Date + 1' in the Editbox
- 14. Enter File Date - Enter Todays date -1
- 15. Enter Launch Date - Enter Today date
- 16. Enter File Range low '11' to high '13'
- 17. Enter 9mm in Lead Tranche Initial File Size field.

Create Object

- 18. "Base Product Section:

- Select 'Common Shares' from Base Product Security Type drop-down"
- Select 'USD' from Base Product Deal Currency drop-down
- 19. Enter 'NYSE' in Exchange drop-down
- 20. Enter CUSIP 'A12345678' in CUSIP Editbox (9 digits)
- 21. Select Pubilc radio button under 'Registration' Section
- 22. Select 'S-3' from Registration drop-down

- 23. "Lead Tranche/Syndicate Information section:
- Select 'United States' from Lead Tranche Name drop-down"
- 24. Select 'Wachovia Securities, LLC' from Lead Tranche Subsidiary drop-down
- 25. Select 'Lead Manager/Bookrunner' from Lead Tranche Syndicate Role drop-down

Create Object

- 26. "Deal Level Access Controls section:
- Select 'Announced' from Deal Access Deal State drop-down"
- 27. "Lead Tranche/Syndicate Information section:
- 28. Select ""Book Open"": Yes Radiobutton "
- 29. Select ' Internal' radiobutton Under Deal Access Controls section.
- 30. Click on Create button.
- 31. Navigate to Deal > Operations sub menu
- 32. Click the “Trade Processing Accounts+” link in the “Account Codes” section.
- 33. Enter Region 'RG1, RG2, RG3, BLRG1' and Account No '5111,5111,5111,5111'
- 34. Click Save.
- 35. Click on “Save” on the Operations page.
- 36. Under Syndicate menu, click “Syndicate Participation” .

Create Object

- 37. "Edit Syndicate Participants section:
- Enter number of shares ""3,000,000"" in RETAIL RETENTION box next to the Lead Manager/ Bookrunner."
- 38. Click Save.
- 39. Click “Access Controls” under “Deal” menu.
- 40. Select “Retail” checkbox under the “Calendar” section OR confirm that "Retail" check box is CHECKED and saved.
- 41. Click Save on Access Controls Page
- 42. Click “Send to Retail” button.
- 43. Logout from Equity Institutional Application.

Create Object with min param\specific

Feature:

- As End User
- I want have possibility to create objects in my systems
- So that I would be able to add object based on my needs

Scenario:

- Given I have an object
- When I Update object
- Then Object is updated and saved in the system

Update Object

Feature:

- As End User
- I want have possibility to Update objects from my systems
- So that I would be able to change needed for me information

Scenario:

- Given I have an object
- When I Update object
- Then Object is updated and saved in the system

Delete Object

Feature:

- As End User
- I want have possibility to delete objects from my systems
- So that all Unnecessary object will be removed from system

Scenario:

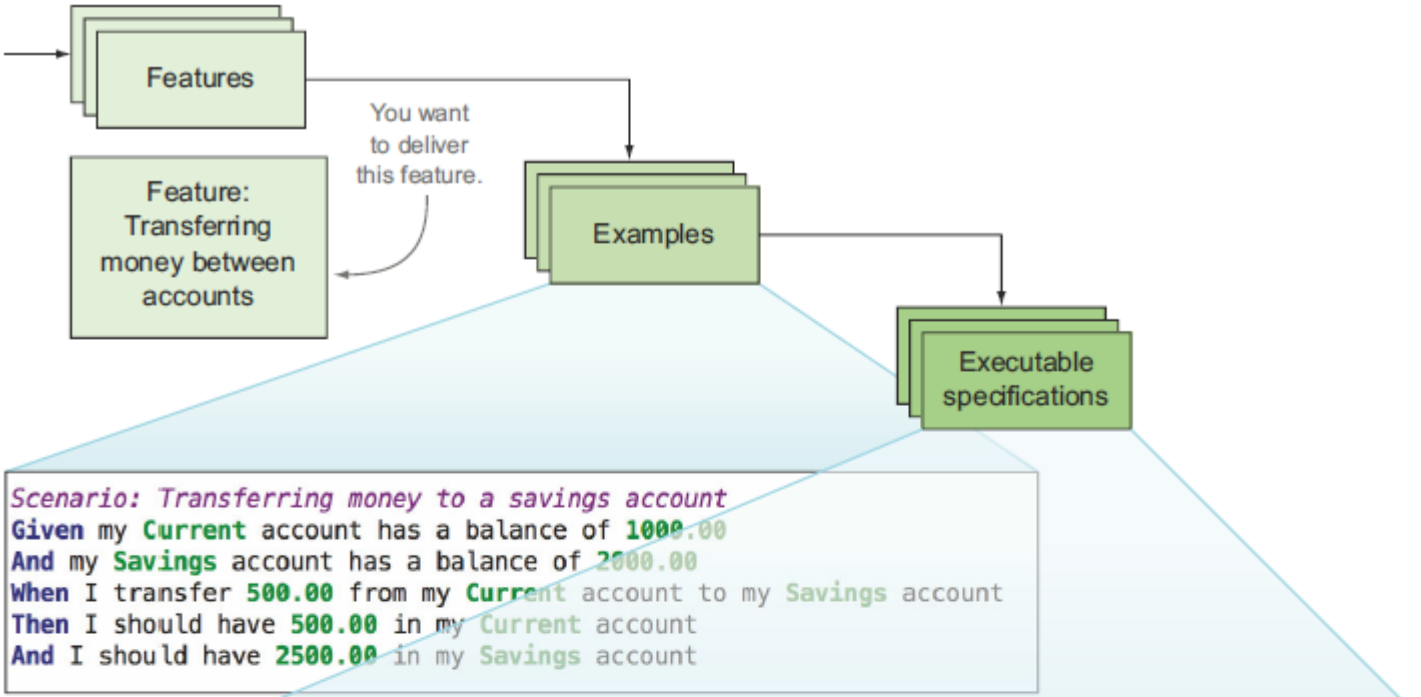
- Given I have an object
- When I Delete object
- Then Object is delated from the system

BDD for Bank Account

```
Feature: Transferring money between accounts
  In order to manage my money more efficiently
  As a bank client
  I want to transfer funds between my accounts whenever I need to

  Scenario: Transferring money to a savings account
    Given my Current account has a balance of 1000.00
    And my Savings account has a balance of 2000.00
    When I transfer 500.00 from my Current account to my Savings account
    Then I should have 500.00 in my Current account
    And I should have 2500.00 in my Savings account
  Scenario: Transferring with insufficient funds
    Given my Current account has a balance of 1000.00
    And my Savings account has a balance of 2000.00
    When I transfer 1500.00 from my Current account to my Savings account
    Then I should receive an 'insufficient funds' error
    Then I should have 1000.00 in my Current account
    And I should have 2000.00 in my Savings account
```

BDD for Bank Account




```
Scenario: Transferring money to a savings account
Given my Current account has a balance of 1000.00
And my Savings account has a balance of 2000.00
When I transfer 500.00 from my Current account to my Savings account
Then I should have 500.00 in my Current account
And I should have 2500.00 in my Savings account
```

The acceptance criteria for the feature take the form of concret examples.

```
@Given("my $accountType account has a balance of $amount")
public void setupInitialAccount(AccountType accountType, double amount) {
    Account account = Account.ofType(accountType).withInitialBalance(amount);
    accountService.create(account);
    myAccounts.put(accountType, account.getAccountNumber());
}

@When("I transfer $amount from my $source account to my $destination account")
public void transferAmountBetweenAccounts(double amount,
                                          AccountType source,
                                          AccountType destination) {
    Account sourceAccount = accountService.findByNumber(myAccounts.get(source)).g
    Account destinationAccount = accountService.findByNumber(myAccounts.get(desti
    accountService.transfer(amount).from(sourceAccount).to(destinationAccount);
}
```

You turn the examples into "executable specifications."

Transferring money to a savings account 0.17s

Mapping

```
Scenario: Transferring money to a savings account
Given my Current account has a balance of 1000.00
And my Savings account has a balance of 2000.00
When I transfer 500.00 from my Current account to my Savings account
Then I should have 500.00 in my Current account
And I should have 2500.00 in my Savings account
```

```
@Given("my $accountType account has a balance of $amount")
public void setupInitialAccount(AccountType accountType, double amount) {
    Account account = Account.ofType(accountType).withInitialBalance(amount);
    accountService.create(account);
    myAccounts.put(accountType, account.getAccountNumber());
}

@When("I transfer $amount from my $source account to my $destination account")
public void transferAmountBetweenAccounts(double amount,
                                          AccountType source,
                                          AccountType destination) {
    Account sourceAccount = accountService.findByNumber(myAccounts.get(source)).g
    Account destinationAccount = accountService.findByNumber
    accountService.transfer(amount).from
```

Step definitions call application code to implement steps in the acceptance criteria.

Low-level executable specifications (*unit tests*) help design the detailed implementation.

```
class WhenCreatingANewAccount extends Specification {

    def "account should have a number, a type and an initial balance"() {
        when:
            Account account = Account.ofType(Savings)
                                     .withInitialBalance(100)
        then:
            account.accountType == Savings
            account.balance == 100
    }
```

Good \ Bad examples

Try to minimize each scenario and test one thing at a time

Try to limit to 1 Given and 1 Then per scenario

BAD

```
Scenario: Test A
    Given I login
    When I go to the main screen
    Then I can start
    Given I have started
    When I create x
    Then it is verified
```

GOOD

```
Scenario: Verify that I can start

Given I have logged in

When I have started

Then it is verified
```

Good \ Bad examples

Test scenarios should consist of 3 max 10 conditions.

BAD:

```
Story: Animal Submission

  As a Zoologist
  I want to add a new animal to the site
  So that I can share my animal knowledge with the community

  Scenario: successful submission
  Given I'm on the animal creation page

  When I fill in Name with 'Alligator'
  And select Phylum as 'Chordata'
  And fill in Animal Class with 'Sauropsida'
  And fill in Order with 'Crocodilia'
  And fill in Family with 'Alligatoridae'
  And fill in Genus with 'Alligator'
  And check Lay Eggs
  And click the Create button

  Then I should see the notice 'Thank you for your animal submission!'
  And the page should include the animal's name, phylum, animal class, order, family, and genus
```

GOOD:

```
Story: Animal Submission

  As a Zoologist
  I want to add a new animal to the site
  So that I can share my animal knowledge with the community

  Scenario: successful submission
  Given I'm on the animal creation page

  When I add a new animal

  Then I should see the page for my newly created animal
  And the notice 'Thank you for your animal submission!'
```

Good \ Bad examples

No more than 5 columns and 10 rows for example, else call in from Excel

BAD: Scenario outline: Free delivery Given the threshold for free delivery is 10 books And the customer is <type> with <books> in the cart When checking out Then free delivery is <free delivery>

Examples

| type | books | free delivery |

| VIP | 9 | not offered |

| VIP | 10 | offered |

| VIP | 11 | offered |

| Normal | 10 | not offered |

| Normal | 9 | not offered |

| Normal | 10 | offered |

| VIP | 11 | offered |

| Normal | 10 | not offered |

| STD| 9 | not offered |

| STD | 10 | offered |

| STD | 11 | offered |

| Normal | 10 | not offered |

Good \ Bad examples

Scenario should run independently without any dependencies on other scenarios.

BAD:

```
Scenario: Test A
    Given I login
    When I go to the main screen
    Then I can start

Scenario: Test B
    Given I have started
    When I create x
    Then it is verified
```

GOOD:

```
Scenario: Verify that I can start
    Given I have logged in
    When I have started
    Then it is verified
```

Hook Order*

Hook Order*

BeforeTestRun

BeforeFeature

BeforeScenario or Before

BeforeScenarioBlock

BeforeStep

AfterStep

AfterScenarioBlock

AfterScenario or After

AfterFeature

AfterTestRun

Hook Order*

Hook Order*

- BeforeTestRun - called before any Features are executed, cannot be used with tags
- BeforeFeature - called before the Feature is executed
- BeforeScenario or Before - called before the Scenario or an Example in a Scenario Outline is executed
- BeforeScenarioBlock - called before each Given, When, and Then block
- BeforeStep - called before each sentence
- AfterStep - called after each sentence
- AfterScenarioBlock - called after each Given, When, and Then block
- AfterScenario or After - called after each Scenario or an Example in a Scenario Outline is executed
- AfterFeature - called after the Feature is executed
- AfterTestRun - called after all Features have executed, cannot be used with tags

The order of Hooks with the same tag is undefined