

TRABALHO PRÁTICO 2 - ESII - GRUPO1213

TEST DESIGN SPECIFICATION

Version 7.0
21/01/2022

Histórico de Versões

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	12/01/2022	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	12/01/2022	Análise testes caixa preta para o módulo transações
2.0	Bruno Ferreira Gonçalo Oliveira	13/01/2022	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	13/01/2022	Correção de pormenores nas tabelas de casos de teste
3.0	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	14/01/2022	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	14/01/2022	Correção de alguns erros identificados
4.0	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	15/01/2022	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	15/01/2022	Adicionar casos de testes identificados na análise testes de caixa branca do módulo de transações
5.0	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	17/01/2022	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	17/01/2022	Análise testes caixa preta para o módulo de custos de envio
6.0	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	18/01/2022	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	18/01/2022	Adicionar casos de testes identificados na análise testes de caixa branca do módulo de cálculo de custos
7.0	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	21/01/2022	Bruno Ferreira Jorge Correia Gonçalo Oliveira Nuno Castro	21/01/2022	Análise testes caixa preta para o módulo de expedição

Tabela de Conteúdos

1. Introdução	4
1.1. Identificador do documento	4
1.2. Âmbito	4
1.3. Repositório	4
1.4. Referências	4
1.5. Glossário	5
2. Features/Itens a testar	5
3. Detalhes da abordagem aos testes	7
4. Identificação dos Testes	7
1. Gestão Encomendas	8
1.1 addEncomenda	8
1.2 cancelarEncomenda	9
1.3 findEncomenda	10
2. BasicLinhaTransação	11
2.1 addTransactionLine	11
3. Estatísticas	12
3.1 getValorMedioTransacoes	12
3.2 getNumMedioProdutosPorTransacao	13
3.3 getNumMedioEncomendasPorDia	14
3.4 getTotalCustosEnvio	15
3.5 getValorMedioVendasComprasPorDistrito	16
4. Custos	17
4.1 calculaCustoEnvio	17
5. Expedição	18
5.1 assignEncomendas	18
5. Critérios de passagem ou falha das features	19

1. Introdução

1.1. Identificador do documento

TestCaseSpecification_Grupo1213

1.2. Âmbito

Este relatório de Test Case Specification está a ser realizado no âmbito do Trabalho Prático 2 da disciplina de Engenharia de Software II em que, e tal como o nome indica, o objetivo deste relatório é especificar os testes BlackBoxTests bem como, toda a documentação referente a abordagens efetuadas e resultados esperados.

Ao longo do desenvolvimento do módulo foram feitos testes unitários aos métodos que, na nossa opinião, se identificavam como cruciais para o bom funcionamento da aplicação.

Estão explicitados neste relatório todas as informações necessárias para a realização dos testes, bem como as ferramentas e técnicas usadas para tal.

1.3. Repositório

A hiperligação para o repositório onde foi realizado o trabalho é a seguinte:

<https://gitlab.estg.ipp.pt/esii.grupo1213/esii-grupo1213-tp2>

1.4. Referências

Como referência para a realização deste documento foram usados documentos como:

- Slides disponibilizados no Moodle da Unidade Curricular
- IEEE Standard for Software Unit Testing

1.5. Glossário

Testes de software - Um teste de software é um software que executa outro software, validando se os resultados são os esperados (teste de estado) ou se executa a sequência de eventos esperado (teste de comportamento).

Resumidamente, os testes de software permitem aos programadores verificar se a lógica do programa desenvolvido está de acordo com os requisitos.

A execução automática de testes permite identificar “bugs” resultantes de mudanças no código fonte.

ECP - Técnica destinada a reduzir o número de testes necessários dividindo o domínio de entrada (ou saída) em classes de dados em que os casos de teste podem ser derivados para cada operação, o “tester” deve identificar as classes de equivalência dos argumentos e os estados dos objetos

BVA - Técnica baseada na observação de bugs que ocorrem frequentemente em valores fronteira. É focada em testar valores especiais (null, 0, etc) e limites do domínio de entrada (ou saída) imediatamente acima e abaixo (além de ou em vez de valores intermédios)

JUnit - Framework open-source para realizar testes unitários para código em Java.

Permite a execução de testes de forma:

- Fácil
- Regular
- Fiável

Contém várias funcionalidades para testing entre elas a capacidade de testar cada componente de um programa de forma independente do resto do programa

Gradle - É uma ferramenta que permite integrar e automatizar várias tarefas relacionadas com o processo de desenvolvimento de software em várias linguagens de programação.

O Gradle determina quais os componentes do projeto que estão atualizados, evitando a recompilação de todo o projeto.

2. Features/Itens a testar

Item a testar	Descrição	Requisitos	Responsabilidade
addEncomenda	Adicionar uma encomenda ao sistema que, ainda não se encontre no mesmo, para que esta possa ser gerida devidamente.	-----	Bruno Ferreira
cancelarEncomenda	Cancelar uma encomenda que esteja adicionada e ainda não tenha sido feito o pagamento.	-----	Jorge Correia
findEncomenda	Verifica se uma determinada encomenda se encontra registada.	-----	Gonçalo Oliveira
addTransactionLine	Adicionar linhas de transação a uma transação. Uma linha de transação contém informação sobre o produto e sobre a própria linha de transação.	-----	Nuno Castro
getValorMedioTransacoes	Métrica para calcular o número médio de transações.	-----	Bruno Ferreira
getNumMedioProdutosPorTransacao	Métrica para obter o número médio de produtos que uma transação possui.	-----	Bruno Ferreira
getNumMedioEncomendasPorDia	Métrica para obter o número médio de encomendas por dia	-----	Gonçalo Oliveira
getTotalCustosEnvio	Métrica para obter o total custo de envio gasto num determinado tempo.	-----	Gonçalo Oliveira
calculaCustoEnvio	Calcular o custo de envio de uma determinada encomenda para que posteriormente seja calculado o custo total da encomenda	-----	Jorge Correia
assignEncomendas	Atribuir encomendas a contentores e posteriormente exportar documento JSON que contém a informação dos contentores e das suas encomendas	-----	Bruno Ferreira
getValorMedioVendasComprasPorDistrito	Métrica para calcular o valor médio de vendas e compras por distrito..	-----	Bruno Ferreira

3. Detalhes da abordagem aos testes

Para assegurar a qualidade do software foram realizados testes de forma a tornar o software robusto e resistente a falhas. Tendo isto em conta foram utilizadas técnicas como ECP (equivalency class partitioning) e BVA (boundary value analysis).

Inicialmente foram identificados os métodos mais importantes de todo o projeto e foram realizadas tabelas tendo em conta as técnicas identificadas acima, em seguida passou-se para a codificação dos testes utilizando a framework JUnit.

No caso da técnica ECP foram realizadas tabelas identificando classes de equivalência válidas e inválidas para cada método e foram realizados casos de teste baseados nessas classes, no caso do BVA foram identificadas entradas em que as pré-condições são asseguradas bem como o contrário.

Para finalizar será criada uma tabela com casos de teste para cada método, identificando o ECP em que se insere, os casos limite (BVA), o resultado esperado, o resultado obtido e o resultado do teste.

Pretende-se com esta abordagem detetar o maior número de erros executando o menor número de casos de teste.

4. Identificação dos Testes

Neste tópico estão identificadas todas as tabelas ECP e BVA para cada método, realizadas pelo grupo.

1. Gestão Encomendas

1.1 addEncomenda

addEncomenda			
Pre - condição	A encomenda passada como parâmetro ainda não foi adicionada.		
ECP			
Critérios	Classe de equivalência válida	Classe de equivalência inválida	
nº de entradas	1	< 1 ou >1	
nome do Input	encomenda	encomenda	
tipo do Input	Encomenda	!Encomenda	
valor específico x	Encomenda(Transaction transactionEnvio)	! Encomenda(Transaction transactionEnvio)	
identificação do ECP	ECP 1	ECP 2	
Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Transaction transactionEnvio; Encomenda encomenda = new BasicEncomenda (transactionEnvio); addEncomenda (encomenda);			
addEncomenda ("encomenda");			
addEncomenda (1)			
addEncomenda (encomenda1, encomenda2)			
addEncomenda (true)			
addEncomenda (null)			
addEncomenda ()			

BVA (addEncomenda)
<p>Aceita uma encomenda válida, sendo a classe inválida um objeto que não seja instância de Encomenda, consequentemente, a classe válida é uma instância de Encomenda.</p>

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomenda	Requisito: Adicionar encomenda Use Case: Módulo Transações Funcionalidade: addEncomenda()	Pré-condição: Assume-se que a encomenda não foi adicionada. Estados Iniciais: É necessário criar um objeto encomenda que seja criado dentro das normalidades, ou seja, do tipo Envio, com um sender e receber válidos.	"true"
	addEncomenda (encomenda)			
#2	encomenda	Requisito: Adicionar encomenda Use Case: Módulo Transações Funcionalidade: addEncomenda()	Pré-condição: Assume-se que a encomenda já foi adicionada. Estados Iniciais: É necessário criar um objeto encomenda que seja criado dentro das normalidades, ou seja, do tipo Envio, com um sender e receber válidos.	"false"
	addEncomenda (encomenda)			
#3	addEncomenda (null)	Requisito: Adicionar encomenda Use Case: Módulo Transações Funcionalidade: addEncomenda()	--	Throw IllegalArgumentException
#4	addEncomenda("encomenda")	Requisito: Adicionar encomenda Use Case: Módulo Transações Funcionalidade: addEncomenda()	--	Erro de Sintaxe

1.2 cancelarEncomenda

cancelarEncomenda			
Pre - condição	É necessário que a encomenda enviada como parâmetro exista para que esta possa ser inserida.		
ECP			
Critérios	Classe de equivalência válida	Classe de equivalência inválida	
nº de entradas	1	< 1 ou >1	
nome do Input	encomenda	encomenda	
tipo do Input	Encomenda	!Encomenda	
valor específico x	Encomenda(transaction transactionEnvio)	! Encomenda(transaction transactionEnvio)	
identificação do ECP	ECP 1	ECP 2	
Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Transaction transactionEnvio; Encomenda encomenda = new BasicEncomenda (transactionEnvio); addEncomenda (encomenda); cancelarEncomenda(encomenda);			
cancelarEncomenda ("encomenda");			
cancelarEncomenda (1)			
cancelarEncomenda (encomenda1, encomenda2)			
cancelarEncomenda (true)			
cancelarEncomenda (null)			
cancelarEncomenda ()			

BVA (cancelarEncomenda)
Aceita uma encomenda válida, sendo a classe inválida um objeto que não seja instância de Encomenda, consequentemente, a classe válida é uma instância de Encomenda.

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomenda	Requisito: Cancelar encomenda Use Case: Módulo Transações Funcionalidade: cancelar Encomenda()	Pré-condição: Assume-se que a encomenda enviada como parâmetro já foi anteriormente adicionada. Estados Iniciais: Considera-se que o objeto encomenda enviada como parâmetro foi criada dentro das normalidades.	"true"
	cancelarEncomenda (encomenda)			
#2	encomenda	Requisito: Cancelar encomenda Use Case: Módulo Transações Funcionalidade: addEncomenda()	Pré-condição: Assume-se que a encomenda enviada como parâmetro não foi anteriormente adicionada. Estados Iniciais: Considera-se que o objeto encomenda enviada como parâmetro foi criada dentro das normalidades.	"false"
	cancelarEncomenda (encomenda)			
#3	cancelarEncomenda (null)	Requisito: Cancelar encomenda Use Case: Módulo Transações Funcionalidade: addEncomenda()	--	Throw IllegalArgumentException
#4	encomenda	Requisito: Cancelar encomenda Use Case: Módulo Transações Funcionalidade: addEncomenda()	--	Erro de Sintaxe
	cancelarEncomenda("encomenda")			

1.3 findEncomenda

findEncomenda			
Pre - condição	N/D		
	ECP		
Crítérios	Classe de equivalência válida	Classe de equivalência	
nº de entradas	1	< 2 ou >2	
nome do Input	id	id	
tipo do Input	String	!String	
valor específico x	< max	> max	
identificação do ECP	ECP 1	ECP 2	
Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
findEncomenda ("20210101_01")			
findEncomenda (1)			
findEncomenda ("idEncomenda")			
findEncomenda (true)			
findEncomenda (null)			
findEncomenda ()			
findEncomenda ("20210101_01", "20210101_01")			

BVA (findEncomenda)
<p>Aceita uma string, sendo a classe inválida um tipo de dados que não seja uma string, consequentemente, a classe válida é uma string.</p>

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomenda	Requisito: N/D Use Case: N/D Funcionalidade: findEncomenda()	Pré-condição: N/D Estados Iniciais: Considera-se que existem algumas encomendas armazenadas na classe BasicGestaoEncomendas, sendo que uma delas tem o id igual ao enviado como parametro na função.	Encomenda(id, ...)
	findEncomenda ("20220112_01")			
#2	encomenda	Requisito: N/D Use Case: N/D Funcionalidade: findEncomenda()	Pré-condição: N/D Estados Iniciais: Considera-se que existem algumas encomendas armazenadas na classe BasicGestaoEncomendas, sendo que nenhuma delas tem o id igual ao enviado como parametro na função.	null
	findEncomenda ("20220112_100")			
#3	encomenda	Requisito: N/D Use Case: N/D Funcionalidade: findEncomenda()	--	null
	findEncomenda ("")			
#4	encomenda	Requisito: N/D Use Case: N/D Funcionalidade: findEncomenda()	--	Erro de Sintaxe
	findEncomenda(123)			
#5	encomenda	Requisito: N/D Use Case: N/D Funcionalidade: findEncomenda()	--	Throw IllegalArgumentException
	findEncomenda(null)			

2.BasicLinhaTransação

2.1 addTransactionLine

addTransactionLine		
Pre - condição	A trasactionLine passada como parâmetro ainda não foi adicionada.	
ECP		
Critérios	Classe de equivalência válida	Classe de equivalência inválida
nº de entradas	1	< 1 ou >1
nome do Input	transactionPagamento	transactionPagamento
tipo do Input	TransactionLine	!TransactionLine
valor específico x	LinhaTransacao	! LinhaTransacao
identificação do ECP	ECP 1	ECP 2

Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Entity sender,receiver; TipoTrasacao tipoTrasaction; LinhaTransacao linhaTransacao; TransactionEncomenda encomenda = new BasicTransactionEncomenda (sender, receiver, tipoTrasaction); addTransactionLine (linhaTransacao);			
addTransactionLine ("");			
addTransactionLine (1);			
addTransactionLine (linhaTransacao1, linhaTransacao2);			
addTransactionLine (true);			
addTransactionLine (null);			
addTransactionLine ();			

BVA (addTransactionLine)
<p>Aceita uma LinhaTransacao válida, sendo a classe inválida um objeto que não seja instância de LinhaTransacao, consequentemente, a classe válida é uma instância de LinhaTransacao.</p>

ID Coso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	linhaTransacao	Requisito: -- Use Case: Módulo Transações Funcionalidade: addTransactionLine()	Pré-condição: Assume-se que a linhaTransacao não foi adicionada. Estados Iniciais: É necessário criar um objeto BasicLinhaTransacao que seja criado dentro das normalidades, ou seja, o Produto e a quantidade validos.	"true"
	addTransactionLine (LinhaTransacao)			
#2	linhaTransacao	Requisito: -- Use Case: Módulo Transações Funcionalidade: addTransactionLine()	Pré-condição: Assume-se que a linhaTransacao já foi adicionada. Estados Iniciais: É necessário criar um objeto BasicLinhaTransacao que seja criado dentro das normalidades, ou seja, o itemDescription, a quantidade e o unitPrice validos.	Throw IllegalArgumentException
	addTransactionLine (TransactionLine)			
#3	linhaTransacao	Requisito: -- Use Case: Módulo Transações Funcionalidade: addTransactionLine()	Pré-condição: -- Estados Iniciais: É necessário criar um objeto BasicLinhaTransacao que seja criado dentro das normalidades, ou seja, o itemDescription, a quantidade e o unitPrice validos.	Throw IllegalArgumentException
	addTransactionLine (TransactionLine)			
#4	linhaTransacao	Requisito: -- Use Case: Módulo Transações Funcionalidade: addTransactionLine()	--	Throw IllegalArgumentException
	addTransactionLine (null)			
#5	linhaTransacao	Requisito: -- Use Case: Módulo Transações Funcionalidade: addTransactionLine()	--	Erro de Sintaxe
	addTransactionLine ("linhaTrasacao")			

3.Estatísticas

3.1 getValorMedioTransacoes

getValorMedioTransacoes			
Pre - condição		---	
ECP			
Critérios	Classe de equivalência válida	Classe de equivalência inválida	
nº de entradas	1	< 1 ou >1	
nome do Input	encomendas	encomendas	
tipo do Input	Iterator<Encomenda>	! Iterator<Encomenda>	
valor específico x	Iterator<Encomenda>	! Iterator<Encomenda>	
identificação do ECP	ECP 1	ECP 2	
Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Iterator<Encomenda> encomendas = new Iterator< >(); getNumMedioProdutosPorTransacao (encomendas)			
getNumMedioTransacoes("encomendas");			
getNumMedioTransacoes(1)			
Iterator<Encomenda> encomendas = new Iterator< >(); getNumMedioTransacoes(encomendas, encomendas)			
getNumMedioTransacoes(true)			
getNumMedioTransacoes(null)			
getNumMedioTransacoes ()			

BVA (getValorMedioTransacoes)
<p>Aceita um Iterator que percorre elementos instância Encomenda, sendo a classe inválida um Iterator que percorre elementos que não são instância Encomenda e, conseqüentemente, a classe válida é um Iterator que percorre elementos instância Encomenda.</p>

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomendas getValorMedioTransacoes(encomendas)	Requisito: Calcular valor médio de transações Use Case: Módulo Transações Funcionalidade: Calcular o valor médio das transações	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foram realizadas 2 encomendas e que o valor total das 2 encomendas é de 30€	15
#2	encomendas getValorMedioTransacoes(encomendas)	Requisito: Calcular valor médio de transações Use Case: Módulo Transações Funcionalidade: Calcular o valor médio das transações	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foram realizadas 3 encomendas e que o valor total das 3 encomendas é de 80€	26.6
#3	encomendas getValorMedioTransacoes(null)	Requisito: Calcular valor médio de transações Use Case: Módulo Transações Funcionalidade: Calcular o valor médio das transações	--	Throw IllegalArgumentExcepti on
#4	encomendas getValorMedioTransacoes("encomenda")	Requisito: Calcular valor médio de transações Use Case: Módulo Transações Funcionalidade: Calcular o valor médio das transações	--	Erro de Sintaxe

3.2 getNumMedioProdutosPorTransacao

getNumMedioProdutosPorTransacao			
Pre - condição		---	
ECP			
Critérios	Classe de equivalência válida	Classe de equivalência inválida	
nº de entradas	1	< 1 ou >1	
nome do Input	encomendas	encomendas	
tipo do Input	Iterator<Encomenda>	! Iterator<Encomenda>	
valor específico x	Iterator<Encomenda>	! Iterator<Encomenda>	
identificação do ECP	ECP 1	ECP 2	
Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Iterator<Encomenda> encomendas = new Iterator< > (); getNumMedioProdutosPorTransacao (encomendas)			
getNumMedioProdutosPorTransacao ("encomendas");			
getNumMedioProdutosPorTransacao (1)			
Iterator<Encomenda> encomendas = new Iterator< > (); getNumMedioProdutosPorTransacao (encomendas, encomendas)			
getNumMedioProdutosPorTransacao (true)			
getNumMedioProdutosPorTransacao (null)			
getNumMedioProdutosPorTransacao ()			

BVA (getNumMedioProdutosPorTransacao)

Aceita um Iterator que percorre elementos
instância Encomenda, sendo a classe inválida
um Iterator que percorre elementos que não
são instância Encomenda e, consequentemente,
a classe válida é um Iterator que percorre
elementos instância Encomenda.

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foram encomendados 6 produtos num total de 2 encomendas.	3
	getNumMedioProdutosPorTransacao (encomendas)			
#2	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foram encomendados 7 produtos num total de 3 encomendas.	2.3
	getNumMedioProdutosPorTransacao (encomendas)			
#3	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	--	Throw IllegalArgumentExcepti on
	getNumMedioProdutosPorTransacao (null)			
#4	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	--	Erro de Sintaxe
	getNumMedioProdutosPorTransacao ("encomenda")			
#5	encomenda	Requisito: Calcular valor médio de transações Use Case: Módulo Transações Funcionalidade: Calcular o valor médio das transações	Pré-condição: É necessário enviar o iterator vazio.	0
	getNumMedioProdutosPorTransacao (encomendas)			

3.3 getNumMedioEncomendasPorDia

getNumMedioEncomendasPorDia			
Pre - condição		O iterador passado como parâmetro é válido e do tipo de dados Encomenda.	
ECP			
Critérios	Classe de equivalência válida	Classe de equivalência inválida	
nº de entradas	1	< 1 ou >1	
nome do Input	encomendas	encomendas	
tipo do Input	Iterator<Encomenda>	! Iterator<Encomenda>	
valor específico x	Iterator<Encomenda>	! Iterator<Encomenda>	
identificação do ECP	ECP 1	ECP 2	
Casos de Teste baseados nas classes de equivalência (ECP)			
	nº de entradas	tipo do Input	valor específico x
Iterator<Encomenda> encomendas = new Iterator< >(); getNumMedioEncomendasPorDia (encomendas)			
getNumMedioEncomendasPorDia ("encomendas");			
getNumMedioEncomendasPorDia(1)			
Iterator<Encomenda> encomendas = new Iterator< >(); getNumMedioEncomendasPorDia (encomendas, encomendas)			
getNumMedioEncomendasPorDia (true)			
getNumMedioEncomendasPorDia (null)			
getNumMedioEncomendasPorDia ()			

BVA (getNumMedioEncomendasPorDia)

Aceita um Iterator que percorre elementos instância Encomenda, sendo a classe inválida um Iterator que percorre elementos que não são instância Encomenda e, consequentemente, a classe válida é um Iterator que percorre elementos instância Encomenda.

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomendas getNumMedioEncomendasPorDia(encomendas)	Requisito: Calcular o número médio de encomendas por dia Use Case: Módulo Transações Funcionalidade: Calcular o número médio de encomendas por dia	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foram enviadas 3 encomendas e que a encomenda que tem associada a data mais antiga é de dia 02-01-2022. (Considerando que o dia atual é 12-01-2022)	0.27
#2	encomendas getNumMedioEncomendasPorDia(encomendas)	Requisito: Calcular o número médio de encomendas por dia Use Case: Módulo Transações Funcionalidade: Calcular o número médio de encomendas por dia	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foi enviada uma encomenda e que essa encomenda tem associada a data de dia 12-01-2022. (Considerando que o dia atual é 12-01-2022)	1
#3	encomendas getNumMedioEncomendasPorDia(null)	Requisito: Calcular o número médio de encomendas por dia Use Case: Módulo Transações Funcionalidade: Calcular o número médio de encomendas por dia	--	Throw IllegalArgumentException
#4	encomendas getNumMedioEncomendasPorDia ("encomenda")	Requisito: Calcular o número médio de encomendas por dia Use Case: Módulo Transações Funcionalidade: Calcular o número médio de encomendas por dia	--	Erro de Sintaxe
#5	encomenda getNumMedioEncomendasPorDia (encomendas)	Requisito: Calcular valor médio de transações Use Case: Módulo Transações Funcionalidade: Calcular o valor médio das transações	Pré-condição: É necessário enviar o Iterator vazio.	0

3.4 getTotalCustosEnvio

getTotalCustoEnvio						
Pre - condição		---				
ECP						
Critérios	Classe de equivalência válida			Classe de equivalência inválida		
nº de entradas	3			< 3 ou >3		
nome do Input	encomendas	dataInicio	dataFim	encomendas	dataInicio	dataFim
tipo do input	Iterator<Encomenda>	LocalDate	LocalDate	!= Iterator<Encomenda>	!= LocalDate	!= LocalDate
valor específico x	Iterator<Encomenda>	<= LocalDate.now	<= LocalDate.now	!= Iterator<Encomenda>	>= LocalDate.now	>= LocalDate.now
identificação do ECP	ECP 1			ECP 2		

Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do input	valor específico x
getTotalCustoEnvio (encomendas, LocalDate, LocalDate)			
getTotalCustoEnvio (encomendas, "LocalDate", LocalDate)			
getTotalCustoEnvio (1, LocalDate, LocalDate)			
getTotalCustoEnvio (encomendas, LocalDate)			
getTotalCustoEnvio (true, LocalDate, LocalDate)			
getTotalCustoEnvio (encomendas, null, LocalDate)			
getTotalCustoEnvio ()			

BVA (getTotalCustosEnvio)

Aceita um conjunto de encomendas válidas e um período de tempo, sendo esta a classe válida, sendo que a classe inválida é um parâmetro que não seja um Iterator de encomendas ou alguma das datas recebidas seja maior que o dia atual, sendo assim conclui-se que, a classe válida é um Iterator de encomendas com datas menores que o dia atual

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomendas dataInicio dataFim getTotalCustosEnvio(encomenda, dataInicio, dataFim)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	Pré-condição: Estados Iniciais: Assume-se que são enviadas duas encomendas em que as entidades envolvidas são do distrito do Porto e Aveiro. * A primeira encomenda enviada foi criada no dia 16-01-2022 e possui 1 produto com um volume 5 e peso 2.5. * A segunda encomenda enviada foi criada no dia 17-01-2022 e 2 produtos com um volume 1.65 e peso 0.50. * A terceira e última encomenda enviada foi criada no dia 15-01-2022 e possui um produto de volume 0.41 e peso 0.10 e outro com volume 5 e peso 2.5 O período de tempo enviado é de 16-01-2022 a 17-01-2022.	66.87
#2	encomendas dataInicio dataFim getTotalCustosEnvio(encomenda, dataInicio, dataFim)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	Pré-condição: Estados Iniciais: Assume-se que são enviadas três encomendas em que as entidades envolvidas são do distrito do Porto e Aveiro. * A primeira encomenda enviada foi criada no dia 16-01-2022 e possui um produto com volume 5 e peso 2.5. * A segunda encomenda enviada foi criada no dia 17-10-2022 e possui dois produtos com volume 1.65 e peso 0.5. * A terceira e última encomenda enviada foi criada no dia 15-01-2022 e possui um produto de volume 0.41 e peso 0.10 e outro com volume 5 e peso 2.5. O período de tempo enviado é de 15-01-2022 a 16-01-2022.	97.19
#3	encomendas dataInicio dataFim getTotalCustosEnvio(null, dataInicio, dataFim)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	Pré-condição: Estados Iniciais: Assume-se que as datas que definem o período de tempo são válidas.	Throw IllegalArgumentExcepti on
#4	encomendas dataInicio dataFim getTotalCustosEnvio(encomenda, null, dataFim)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	Pré-condição: Assume-se que a encomenda e data de fim, pertencente ao período de tempo, são válidas. Estados Iniciais:	Throw IllegalArgumentExcepti on
#5	encomendas dataInicio dataFim getTotalCustosEnvio(encomenda, dataInicio, null)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	Pré-condição: Assume-se que a encomenda e data de inicio, pertencente ao período de tempo, são válidas. Estados Iniciais:	Throw IllegalArgumentExcepti on
#6	encomendas dataInicio dataFim getTotalCustosEnvio(encomenda, dataInicio, dataFim)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	Pré-condição: Assume-se que a encomenda e data de inicio, pertencente ao período de tempo, são válidas. Estados Iniciais: Assume-se que a data de inicio é 17-01-2022 e a data de fim é 15-01-2022, ou seja, a data de inicio é superior à data de fim.	Throw IllegalArgumentExcepti on
#7	encomendas dataInicio dataFim getTotalCustosEnvio(null, null, null)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	--	Throw IllegalArgumentExcepti on
#8	encomendas dataInicio dataFim getTotalCustosEnvio("encomendas", dataInicio, dataFim)	Requisito: Calcular o total de custos de envio num determinado período de tempo Use Case: Módulo de cálculo de custos Funcionalidade: getTotalCustosEnvio()	--	Erro de sintaxe

3.5 getValorMedioVendasComprasPorDistrito

getValorMedioVendasComprasPorDistrito			
Pre - condição	---		
ECP			
Critérios	Classe de equivalência válida	Classe de equivalência inválida	
nº de entradas	1	< 1 ou >1	
nome do Input	encomendas	encomendas	
tipo do Input	Iterator<Encomenda>	! Iterator<Encomenda>	
valor específico x	Iterator<Encomenda>	! Iterator<Encomenda>	
identificação do ECP	ECP 1	ECP 2	
Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Iterator<Encomenda> encomendas = new Iterator<>(); getNumMedioVendasComprasPorDistrito (encomendas)			
getNumMedioVendasComprasPorDistrito ("encomendas");			
getNumMedioVendasComprasPorDistrito (1)			
Iterator<Encomenda> encomendas = new Iterator<>(); getNumMedioVendasComprasPorDistrito (encomendas, encomendas)			
getNumMedioVendasComprasPorDistrito (true)			
getNumMedioVendasComprasPorDistrito (null)			
getNumMedioVendasComprasPorDistrito ()			

BVA (getValorMedioVendasComprasPorDistrito)

Aceita um Iterator que percorre elementos instância Encomenda, sendo a classe inválida um Iterator que percorre elementos que não são instância Encomenda e, consequentemente, a classe válida é um Iterator que percorre elementos instância Encomenda.

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foram feitas encomendas 3 encomendas. * A primeira encomenda foi enviada por uma entidade de Aveiro e recebida por uma entidade do Porto. Esta encomenda tem um preço total de 14.68 * A segunda encomenda foi enviada por uma entidade do Porto e recebida por uma entidade de Aveiro. Esta encomenda tem um preço total de 72.63 * A terceira encomenda foi enviada por uma entidade do Porto e recebida por uma entidade de Aveiro. Esta encomenda tem um preço total de 59.62	matrix [4][0] = 14.68 matrix [4][1] = 66.13 matrix [6][0] = 66.13 matrix [6][1] = 14.68
	getValorMedioProdutosPorTransacao (encomendas)			
#2	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que foram feitas encomendas 5 encomendas. * A primeira encomenda foi enviada por uma entidade de Aveiro e recebida por uma entidade do Porto. Esta encomenda tem um preço total de 14.68 * A segunda encomenda foi enviada por uma entidade do Porto e recebida por uma entidade de Aveiro. Esta encomenda tem um preço total de 72.63 * A terceira encomenda foi enviada por uma entidade do Porto e recebida por uma entidade de Aveiro. Esta encomenda tem um preço total de 59.62 * A quarta encomenda foi enviada por uma entidade de Portalegre e recebida por uma entidade do Porto. Esta encomenda tem um preço total de 213.25 * A quinta encomenda foi enviada por uma entidade de Aveiro e recebida por uma entidade de Portalegre. Esta encomenda tem um preço total de 158.0	matrix [4][0] = 86.34 matrix [4][1] = 66.13 matrix [6][0] = 66.13 matrix [6][1] = 113.97 matrix [13][0] = 213.25 matrix [13][1] = 158.0
	getValorMedioProdutosPorTransacao (null)			
#3	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	--	Throw IllegalArgumentException
#4	encomendas	Requisito: Calcular número médio de produtos por transação Use Case: Módulo Transações Funcionalidade: Calcular número médio de produtos por transação	--	Erro de Sintaxe

4.Custos

4.1 calculaCustoEnvio

calculaCustoEnvio		
Pre - condição	É necessário que a encomenda enviada como parâmetro exista para que esta possa ser inserida.	
ECP		
Critérios	Classe de equivalência válida	Classe de equivalência inválida
nº de entradas	1	< 1 ou >1
nome do Input	encomenda	encomenda
tipo do Input	Encomenda	!Encomenda
valor específico x	Encomenda(Transaction transactionEnvio)	! Encomenda(Transaction transactionEnvio)
identificação do ECP	ECP 1	ECP 2

Casos de Teste baseados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Transaction transactionEnvio; Encomenda encomenda = new BasicEncomenda (transactionEnvio); calculaCusto(encomenda);			
calculaCusto ("encomenda");			
calculaCusto (1)			
calculaCusto (encomenda1, encomenda2)			
calculaCusto (true)			
calculaCusto (null)			
calculaCusto ()			

BVA (calculaCustoEnvio)

Aceita uma encomenda válida, sendo a classe inválida um objeto que não seja instância de Encomenda, consequentemente, a classe válida é uma instância de Encomenda.

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomenda	Requisito: Calcular custo de envio Use Case: Calcular custo de envio Funcionalidade: calculaCustoEnvio()	Pré-condição: Considera-se que o objeto encomenda enviado como parâmetro foi criada dentro das normalidades. Estados Iniciais: Assume-se que as entidades são do distrito de Aveiro e Porto, respetivamente, e que a encomenda possui 2 produtos com um volume 1.65 e peso 0.10	4.245
	calculaCustoEnvio(encomenda)			
#2	encomenda	Requisito: Calcular custo de envio Use Case: Calcular custo de envio Funcionalidade: calculaCustoEnvio()	Pré-condição: Considera-se que o objeto encomenda enviado como parâmetro foi criada dentro das normalidades. Estados Iniciais: Assume-se que as entidades são do distrito de Aveiro e Porto, respetivamente, e que a encomenda possui 2 produtos com um volume 1.65 e peso 0.50 e, 1 produto com volume 1.65 e peso 0.10	21.3675
	calculaCustoEnvio(encomenda)			
#3	encomenda	Requisito: Calcular custo de envio Use Case: Calcular custo de envio Funcionalidade: calculaCustoEnvio()	--	Throw IllegalArgumentException
	calculaCustoEnvio(null)			
#4	encomenda	Requisito: Calcular custo de envio Use Case: Calcular custo de envio Funcionalidade: calculaCustoEnvio()	--	Erro de Sintaxe
	calculaCustoEnvio("encomenda")			

5.Expedição

5.1 assignEncomendas

assignEncomendas			
Pre - condição	---		
ECP			
Critérios	Classe de equivalência válida		Classe de equivalência inválida
nº de entradas	2		< 2 ou >2
nome do Input	encomendas	path	encomendas
tipo do Input	Iterator<Encomenda>	String	! Iterator<Encomenda> String
valor específico x	Iterator<Encomenda>	String	! Iterator<Encomenda> String
identificação do ECP	ECP 1		ECP 2
Casos de Teste basados nas classes de equivalência (ECP)	nº de entradas	tipo do Input	valor específico x
Iterator<Encomenda> encomendas = new Iterator< >(); assignEncomendas(encomendas, "../libs/Contentores.json")			
assignEncomendas("encomendas", "../libs/Contentores.json");			
assignEncomendas(1)			
Iterator<Encomenda> encomendas = new Iterator< >(); assignEncomendas(encomendas, encomendas, " ../libs/Contentores.json")			
assignEncomendas(true, "../libs/Contentores.json")			
assignEncomendas(null, "../libs/Contentores.json")			
assignEncomendas()			

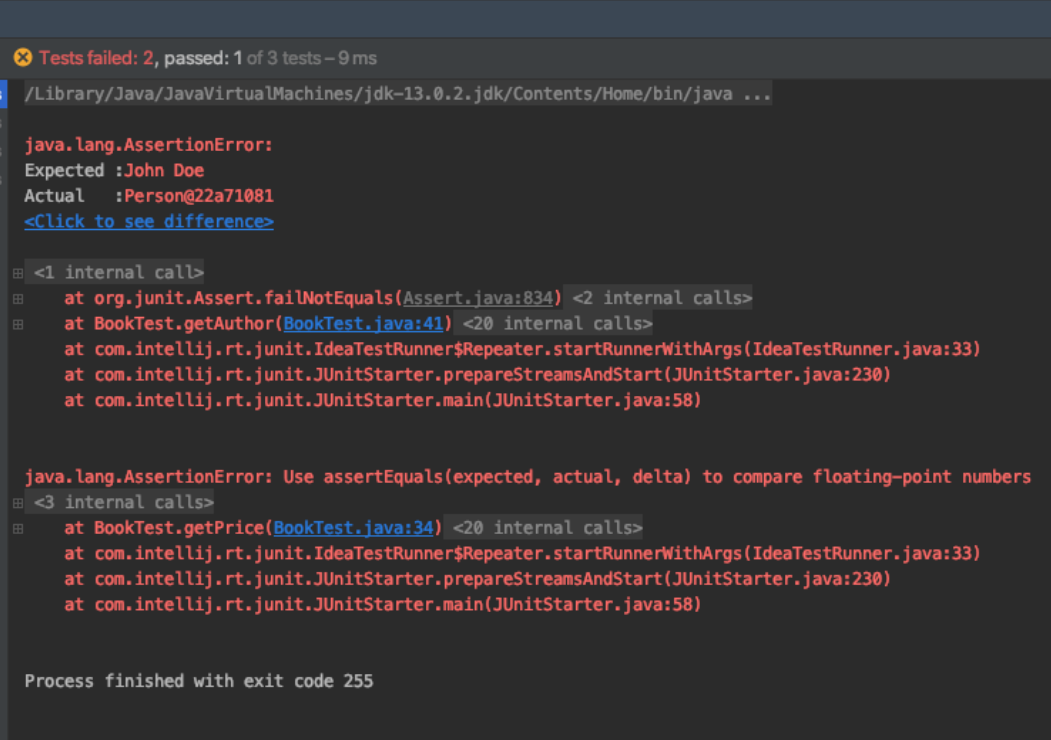
BVA (assignEncomendas)
Aceita uma encomenda válida, sendo a classe inválida um objeto que não seja instância de Encomenda, consequentemente, a classe válida é uma instância de Encomenda.

ID Caso de Teste	Input	Requisito/Use Case/ Funcionalidade	Pré-condições/Estados Iniciais	resultado esperado
#1	encomendas	Requisito: Atribuir encomendas a contentores Use Case: Módulo Expedições Funcionalidade: Associar encomendas a contentores	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que os contentores não foram exportados devidamente	"false"
	assignEncomendas(encomendas, "")			
#2	encomendas	Requisito: Atribuir encomendas a contentores Use Case: Módulo Expedições Funcionalidade: Associar encomendas a contentores	Pré-condição: Assume-se que o Iterator<Encomendas> é válido. Estados Iniciais: Assume-se que os contentores foram exportados devidamente	"true"
	assignEncomendas(encomendas, "../libs/Contentores.json")			
#3	encomendas	Requisito: Atribuir encomendas a contentores Use Case: Módulo Expedições Funcionalidade: Associar encomendas a contentores	Pré-condição: Assume-se que o Iterator<Encomendas> contém encomendas inválidas. Estados Iniciais:	"false"
	assignEncomendas(encomendas, "../libs/Contentores.json")			
#4	encomendas	Requisito: Atribuir encomendas a contentores Use Case: Módulo Expedições Funcionalidade: Associar encomendas a contentores	--	Throw IllegalArgumentException
	assignEncomendas(null, "../libs/Contentores.json")			
#5	encomendas	Requisito: Atribuir encomendas a contentores Use Case: Módulo Expedições Funcionalidade: Associar encomendas a contentores	--	Erro de sintaxe
	assignEncomendas("encomendas", "../libs/Contentores.json")			

5. Critérios de passagem ou falha das features

Através do IDE IntelliJ e do plugin JUnit podemos definir um resultado esperado e, ao correr todos os testes, ser feita a comparação entre o resultado obtido e o resultado esperado. Caso sejam iguais, o teste é identificado como “Passado”, caso contrário será identificado como “Falhado”.

Os critérios que serão usados para a determinação dos testes que passaram e falharam será através da informação fornecida pelo Plugin JUnit que, automaticamente compara o resultado esperado com o obtido e caso estes sejam diferentes é notificada a falha nos testes. Tal como se pode observar na imagem seguinte:



```
✖ Tests failed: 2, passed: 1 of 3 tests - 9 ms
/Library/Java/JavaVirtualMachines/jdk-13.0.2.jdk/Contents/Home/bin/java ...

java.lang.AssertionError:
Expected :John Doe
Actual   :Person@22a71081
<Click to see difference>

<1 internal call>
at org.junit.Assert.failNotEquals(Assert.java:834) <2 internal calls>
at BookTest.getAuthor(BookTest.java:41) <20 internal calls>
at com.intellij.rt.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:33)
at com.intellij.rt.junit.JUnit4TestRunner.prepareStreamsAndStart(JUnit4TestRunner.java:230)
at com.intellij.rt.junit.JUnit4TestRunner.main(JUnit4TestRunner.java:58)

java.lang.AssertionError: Use assertEquals(expected, actual, delta) to compare floating-point numbers
<3 internal calls>
at BookTest.getPrice(BookTest.java:34) <20 internal calls>
at com.intellij.rt.junit.IdeaTestRunner$Repeater.startRunnerWithArgs(IdeaTestRunner.java:33)
at com.intellij.rt.junit.JUnit4TestRunner.prepareStreamsAndStart(JUnit4TestRunner.java:230)
at com.intellij.rt.junit.JUnit4TestRunner.main(JUnit4TestRunner.java:58)

Process finished with exit code 255
```