LEI_ESII2021_TP1_GRUPOU

TEST CASE OUTLINE

Version 1.0

07/12/2020

Histórico de Versões

Version #	Implemented By	Revision Date	Approved By	Approval Date	Reason
1.0	*8160207 – Nuno Matos;* *8190305 – Rafael Dias;*	*07/12/2020*	*8160207 – Nuno Matos;* *8190305 – Rafael Dias;*	*07/12/2020*	First and final draft;

Tabela de Conteúdos

1. Introdução

1.1. Identificador do documento

TCO_DOC

1.2. Âmbito

Todos os casos de uso, testes, e a devida documentação foram realizados para projeto PackingAPI, em prol da unidade curricular de Engenharia de Software II, licenciatura de Engenharia Informática.

1.3. Referencias

Ao longo deste documento aparecem uma série de tabelas de testes que contém o identificador dos casos de uso que são abordados. Estes mesmo podem ser consultados no documento intitulado “Casos de Uso”.

1.4 Glossário

Exemplo – palavras sublinhadas correspondem a instâncias de classes;

Exemplo – palavras em itálico correspondem a variáveis de instância;

Exemplo() – palavras em itálico seguidas de parênteses correspondem a métodos de classes;

2. Detalhes

Test Case ID	TC#1.1	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter uma instância válida de Position.				
Procedimento				
1. Invocar o método *setX()* com A como parâmetro. 2. Verificar se valor de A é o mesmo que agora é retornado pelo método *getX()*. 3. Invocar o método *setX()* com B como parâmetro. 4. Verificar se valor de B é o mesmo que agora é retornado pelo método *getX()*. 5. Invocar o método *setX()* com C como parâmetro. 6. Verificar se valor de C é o mesmo que agora é retornado pelo método *getX()*. 7. Invocar o método *setX()* com D como parâmetro. 8. Verificar se valor de D é o mesmo que agora é retornado pelo método *getX()*.				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0001	A, B, C, D;	A= 0; B=5; C=15; D=50;	--Nenhum--	PASS

Test Case ID	TC#1.2	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter uma instância válida de Position.				
Procedimento				
1. Verificar se ao invocarmos o *setX()* com o valor A como parâmetro nos é atirado uma PositionException; 2. Verificar se ao invocarmos o *setX()* com o valor B como parâmetro nos é atirado uma PositionException;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0001	A, B;	A= -1; B= -15;	**throws** PositionException;	PASS

Test Case ID	TC#2.1	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter uma instância válida de Position.				
Procedimento				
1. Invocar o método *setY()* com A como parâmetro. 2. Verificar se valor de A é o mesmo que agora é retornado pelo método *getY()*. 3. Invocar o método *setY()* com B como parâmetro. 4. Verificar se valor de B é o mesmo que agora é retornado pelo método *getY()*. 5. Invocar o método *setY()* com C como parâmetro. 6. Verificar se valor de C é o mesmo que agora é retornado pelo método *getY()*. 7. Invocar o método *setY()* com D como parâmetro. 8. Verificar se valor de D é o mesmo que agora é retornado pelo método *getY()*.				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0002	A, B, C, D;	A= 0; B=5; C=15; D=50;	--Nenhum--	PASS

Test Case ID	TC#2.2	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	FAIL	Ao inserirmos valores negativos no método *setX()*, nenhuma PositionException é atirada.

Pré-condições				
Devemos ter uma instância válida de Position.				
Procedimento				
1. Verificar se ao invocarmos o *setY()* com o valor A como parâmetro nos é atirado uma PositionException; 2. Verificar se ao invocarmos o *setY()* com o valor B como parâmetro nos é atirado uma PositionException;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0002	A, B;	A= -1; B= -15;	**throws** PositionException;	FAIL

Test Case ID	TC#3.1	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter uma instância válida de Position.				
Procedimento				
1. Invocar o método *setZ()* com A como parâmetro. 2. Verificar se valor de A é o mesmo que agora é retornado pelo método *getZ()*. 3. Invocar o método *setZ()* com B como parâmetro. 4. Verificar se valor de B é o mesmo que agora é retornado pelo método *getZ()*. 5. Invocar o método *setZ()* com C como parâmetro. 6. Verificar se valor de C é o mesmo que agora é retornado pelo método *getZ()*. 7. Invocar o método *setZ()* com D como parâmetro. 8. Verificar se valor de D é o mesmo que agora é retornado pelo método *getZ()*.				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0003	A, B, C, D;	A= 0; B=5; C=15; D=50;	--Nenhum--	PASS

Test Case ID	TC#3.2	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter uma instância válida de Position.				
Procedimento				
1. Verificar se ao invocarmos o *setZ()* com o valor A como parâmetro nos é atirado uma PositionException; 2. Verificar se ao invocarmos o *setZ()* com o valor B como parâmetro nos é atirado uma PositionException;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0003	A, B;	A= -1; B= -15;	**throws** PositionException;	PASS

Test Case ID	TC#4.1	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item e Container.				
Procedimento				
1. Calculamos o valor do volume que será ocupado pelos itens caso todos sejam corretamente adicionados ao Container; 2. Calculamos o valor do volume livre no Container caso todos os itens sejam corretamente inseridos; 3. Verificamos se ao invocarmos o método *addItem()* com os inputs A como parâmetro nos é retornado *true*, que significa que foi devidamente adicionado; 4. Verificamos se ao invocarmos o método *addItem()* com os inputs B como parâmetro nos é retornado *true*, que significa que foi devidamente adicionado; 5. Verificamos se ao invocarmos o método *addItem()* com os inputs C como parâmetro nos é retornado *true*, que significa que foi devidamente adicionado; 6. Verificamos se o volume anteriormente calculado como o ocupado no Container após a adição dos itens corresponde ao agora retornado pelo método *getOccupiedVolume();* 7. Verificamos se o volume anteriormente calculado como o que resta por ocupar no Container após a adição dos itens corresponde ao agora retornado pelo método *getRemainingVolume();* 8. Cria-se uma cópia do input A; 9. Verifica-se se é rejeitada a adição da cópia pelo método *addItem().*				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0004	A, B, C, D;	A=(itemA, positionA, Color.aqua); B=(itemB, positionB, Color.blue); C=(itemC, positionC, Color.gray); D=(copyOfItemA, new Position(0, 0, 50), Color.black);	A, B, C – true; D – false;	PASS

Test Case ID	TC#4.2	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	FAIL	Ao usarmos um valor *null* no terceiro paramêtro do método *addItem()* não nos é atirado uma ContainerException.

Pré-condições
Devemos ter instâncias válidas de Position, Item e Container.
Procedimento
1. Verificamos se ao colocar-mos um valor *null* no primeiro paramêtro, nos é lançado uma ContainerException; 2. Verificamos se ao colocar-mos um valor *null* no segundo paramêtro, nos é lançado uma ContainerException; 3. Verificamos se ao colocar-mos um valor *null* no terceiro paramêtro, nos é lançado uma ContainerException; 4. Adicionamos um Item válido a um Container. 5. Fechamos o Container. 6. Verificamos se ao tentarmos adicionar um novo Item a um Container fechado, nos é atirado uma ContainerException.

Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0004	A, B, C, D;	A=(null, positionA, Color.aqua); B=(itemA, null, Color.aqua); C=(itemA, positionA, null); D=(itemB, positionB, Color.blue);	A, B, C, D – **throws** ContainerException;	FAIL

Test Case ID	TC#5.1	**Dependências**	TC#4.1; TC#4.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições
Devemos ter instâncias válidas de Position, Item e Container.
Procedimento
1. Guardamos o valor original correspondente ao volume ocupado de um Container; 2. Guardamos o valor original correspondente ao volume livre de um Container; 3. Adicionamos um Item válido a um Container; 4. Removemos o Item previamente adicionado ao Container (input A)e verificamos se nos é devidamente retornado um valor *true;* 5. Verificamos se o volume atualmente ocupado do Container corresponde ao volume ocupado antes do Item ser adicionado; 6. Verificamos se o volume atualmente livre do Container corresponde ao volume livre antes do Item ser adicionado; 7. Verificamos se ao tentarmos remover um Item que não se encontra dentro do Container (input B) nos é retornado um valor *false;*

Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0005	A, B;	A=itemA; B=itemB;	A - *true*; B – *false*;	PASS

Test Case ID	TC#5.2	**Dependências**	TC#4.1; TC#4.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	FAIL	Ao tentarmos remover um item depois de fecharmos um Container, não nós é atirado a devida ContainerException.

Pré-condições
Devemos ter instâncias válidas de Position, Item e Container.
Procedimento
1. Verificamos se ao usarmos um valor *null* como parâmetro no método *removeContainer()* nos é atirado uma ContainerException; 2. Adicionamos um Item válido a um Container; 3. Encerramos o Container; 4. Verificamos se ao tentarmos remover o Item previamente adicionado depois de fecharmos o Container, nos é atirado uma ContainerException;

Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0005	A, B;	A=*null*; B=itemA;	A, B – **throws** ContainerException;	FAIL

Test Case ID	TC#6.1	**Dependências**	TC#4.1; TC#4.2; TC#8.1; TC#8.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item e Container.				
Procedimento				
1. Adicionamos um Item válido a um Container. 2. Fechamos o Container;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0006	--NENHUM--	--NENHUM--	--NENHUM--	PASS

Test Case ID	TC#6.2	**Dependências**	TC#4.1; TC#4.2; TC#8.1; TC#8.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	FAIL	Ao adicionarmos um Item fora dos limites do Container e procedermos a seu encerramente, não nos é atirada a devida PositionException.

Pré-condições
Devemos ter instâncias válidas de Position, Item e Container.
Procedimento
1. Criamos uma instância de Container válida; 2. Criamos uma instância de Container válida; 3. Criamos um Item que é ultrapassa os limites dos Containers (longItem), mas não ultrapassa o volume; 4. Criamos um Item que ultrapassa o volume dos Containers (fatItem); 5. Adicionamos um Item válido a um Container; 6. Adicionamos um outro Item na mesma posição do anterior ao mesmo Container; 7. Verificamos se ao tentarmos fechar o Container anterior, nos é atirado uma PositionException; 8. Adicionamos um Item que se encontra fora dos limites do Container. 9. Verificamos se ao tentarmos fechar o Container anterior, nos é atirado uma PositionException; 10. Adicionamos o longItem a um Container. 11. Verificamos se ao tentarmos fechar o Container anterior, nos é atirado uma PositionException; 12. Adicionamos o fatItem a um Container. 13. Verificamos se ao tentarmos fechar o Container anterior, nos é atirado uma ContainerException;

Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0006	--NENHUM--	--NENHUM--	- **throws** PositionException; - **throws** ContainerException;	FAIL

Test Case ID	TC#7.0	**Dependências**	TC#4.1; TC#4.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:00	8190305 – Rafael Dias	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item e Container.				
Procedimento				
1. Adicionamos um Item válido a um Container. 2. Adicionamos um Item válido a um Container. 3. Verificamos se, ao utilizarmos o método *getItem()* com o valor A como parâmetro, nos é retornado a devida instância de Item; 4. Verificamos se, ao utilizarmos o método *getItem()* com o valor B como parâmetro, nos é retornado a devida instância de Item;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0007	A, B;	A - itemA.getReference(); B - itemB.getReference();	A – itemA; B – itemB;	PASS

Test Case ID	TC#8.1	**Dependências**	TC#4.1; TC#4.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item e Container.				
Procedimento				
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Validamos o Container anteriormente utilizado;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0008	--NENHUNS--	--NENHUNS--	--NENHUNS--	PASS

Test Case ID	TC#8.2	**Dependências**	TC#4.1; TC#4.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	FAIL	Ao adicionarmos um Item fora dos limites do Container e procedermos á sua validação, não nos é atirada a devida PositionException.

Pré-condições
Devemos ter instâncias válidas de Position, Item e Container.
Procedimento
1. Criamos uma instância de Container válida; 2. Criamos uma instância de Container válida; 3. Criamos um Item que é ultrapassa os limites dos Containers (longItem), mas não ultrapassa o volume; 4. Criamos um Item que ultrapassa o volume dos Containers (fatItem); 5. Adicionamos um Item válido a um Container; 6. Adicionamos um outro Item na mesma posição do anterior ao mesmo Container; 7. Verificamos se ao tentarmos validar o Container anterior, nos é atirado uma PositionException; 8. Adicionamos um Item que se encontra fora dos limites do Container; 9. Verificamos se ao tentarmos validar o Container anterior, nos é atirado uma PositionException; 10. Adicionamos o longItem a um Container; 11. Verificamos se ao tentarmos validar o Container anterior, nos é atirado uma PositionException; 12. Adicionamos o fatItem a um Container; 13. Verificamos se ao tentarmos validar o Container anterior, nos é atirado uma ContainerException;

Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0008	--NENHUM--	--NENHUM--	- **throws** PositionException; - **throws** ContainerException;	FAIL

Test Case ID	TC#9.1	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#14.1; TC#14.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Passamos o *status* da ShippingOrder para *"IN_TREATMENT";* 4. Encerramos o Container anterior (input A); 5. Criamos uma cópia do Container anterior (input B); 7. Verificamos se, ao invocarmos o método *addContainer()* com o input A como parâmetro, nos é retornado o valor *true*; 8. Verificamos se, ao invocarmos o método *addContainer()* com o input B como parâmetro, nos é retornado o valor *false*;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0009	A, B;	A – containerA; B – copyOfContainerA;	A – *true*; B – *false;*	PASS

Test Case ID	TC#9.2	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#14.1; TC#14.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	FAIL	Quando temos uma ShippingOrder num *status* que não seja *"IN_TREATMENT"*, e adicionamos um novo Container ao mesmo, não nós é atirada uma OrderException.

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 5. Verificamos se, ao invocarmos o método *addContainer()* com o input A como parâmetro, nos é atirada uma ContainerException; 6. Verificamos se, ao invocarmos o método *addContainer()* com o input B como parâmetro, nos é atirada uma ContainerException; 7. Encerramos o Container correspondente ao input B; 8. Passamos o *status* da ShippingOrder para *"CANCELLED"*; 9. Verificamos se, ao invocarmos o método *addContainer()* com o input B como parâmetro, nos é atirada uma OrderException;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC0009	A, Bx2;	A – *null*; B – containerA;	A, B - **throws** ContainerException; B – **throws** OrderException;	FAIL

Test Case ID	TC#10.0	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#14.1; TC#14.2; TC#9.1; TC#9.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Adicionamos um Item válido a um Container. 2. Adicionamos um Item válido a um Container. 3. Adicionamos um Item válido a um Container. 4. Encerrar o Container anteriormente utilizado (input A). 5. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 6. Adicionamos o Container à ShippingOrder. 7. Verificamos se, ao invocarmos o método *existsContainer()* como o input A como parâmetro, nos é retornado um valor *true*; 8. Verificamos se, ao invocarmos o método *existsContainer()* como o input B como parâmetro, nos é retornado um valor *false*;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00010	A, B;	A – containerA; B – containerB;	A – *true*; B – *false*;	PASS

Test Case ID	TC#11.0	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#14.1; TC#14.2; TC#9.1; TC#9.2

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Encerrar o Container anteriormente utilizado (containerA); 5. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 6. Adicionamos o Container à ShippingOrder; 7. Adicionamos um Item válido a um Container; 8. Adicionamos um Item válido a um Container; 9. Adicionamos um Item válido a um Container; 10. Encerrar o Container anteriormente utilizado (containerB); 11. Adicionamos o Container à ShippingOrder; 12. Verificamos se, ao invocarmos o método *findContainer()* com o input A como parâmetro, nos é retornado o valor 0; 13. Verificamos se, ao invocarmos o método *findContainer()* com o input A como parâmetro, nos é retornado o valor 1;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00011	A, B;	A – containerA.getReference(); B – containerB.getReference()	A – 0; B – 1;	PASS

Test Case ID	TC#12.0	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#14.1; TC#14.2; TC#9.1; TC#9.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Encerrar o Container anteriormente utilizado (containerA); 5. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 6. Adicionamos o Container à ShippingOrder; 7. Validamos os Containers inseridos em ShippingOrder;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00012	--NENHUNS--	--NENHUNS--	--NENHUNS--	PASS Nota: Fluxo Alternativo não é possível ser testado;

Test Case ID	TC#13.1	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#14.1; TC#14.2; TC#9.1; TC#9.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Encerrar o Container anteriormente utilizado (input A); 5. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 6. Adicionamos o Container à ShippingOrder; 7. Verificamos se, ao invocarmos o método *removeContainer()* com o input A como parâmetro, nos é retornado o valor *true*; 8. Verificamos se, ao invocarmos o método *removeContainer()* com o input B como parâmetro, nos é retornado o valor *false*;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00013	A, B;	A – containterA; B – containerB;	A – *true*; B – *false*;	PASS

Test Case ID	TC#13.2	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#14.1; TC#14.2; TC#9.1; TC#9.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Encerrar o Container anteriormente utilizado (input B); 5. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 6. Adicionamos o Container à ShippingOrder; 7. Passamos o *status* de uma ShippingOrder para *"CANCELLED"*; 8. Verificamos se, ao invocarmos o método *removeContainer()* com o input A como parâmetro, nos é atirada uma OrderException; 9. Verificamos se, ao invocarmos o método *removeContainer()* com o input B como parâmetro, nos é atirada uma OrderException;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00013	A, B;	A – *null*; B – containerA;	A, B – **throws** OrderException;	PASS

Test Case ID	TC#14.1	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#9.1; TC#9.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	FAIL	Quando uma ShippingOrder não contêm nenhum Container não devia ser possível passar a ter o *status* "*CLOSED*", porem não se verifica esta restrição.

Pré-condições				
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.				
Procedimento				
1. Criamos uma instância válida de ShippingOrder (empty); 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Adicionamos um Item válido a um Container; 5. Encerrar o Container anteriormente utilizado (containerA); 6. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 7. Verificamos se o *status* atual da ShippingOrder corresponde a *"IN_TREATMENT"*; 8. Adicionamos o Container à ShippingOrder; 9. Passamos o *status* de uma ShippingOrder para *"CLOSED"*; 10. Verificamos se o *status* atual da ShippingOrder corresponde a *"CLOSED"*; 11. Passamos o *status* de uma ShippingOrder para *"SHIPPED"*; 12. Verificamos se o *status* atual da ShippingOrder corresponde a *"SHIPPED"*; 13. Passamos o *status* de empty para *"AWAITS_TREATMENT"*; 14. Passamos o *status* de empty para *"IN_TREATMENT"*; 15. Verificamos se o *status* atual de empty corresponde a *"IN_TREATMENT"*; 16. Passamos o *status* de empty para *"CLOSED"*; 17. Verificamos se o *status* atual da ShippingOrder não mudou para *"CLOSED", e permanece em "SHIPPED"*;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00014	--NENHUNS--	--NENHUNS--	--NENHUNS--	FAIL

Test Case ID	TC#14.2	**Dependências**	TC#4.1; TC#4.2; TC#6.1; TC#6.2; TC#9.1; TC#9.2;

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	FAIL	Quando o *status* da ShippingOrder se encontra como *"IN_TREATMENT"* e colocamos *"AWAITS_TREATMENT"* como parâmetro do método *setStatus()* não nos é atirada nenhuma OrderException.

Pré-condições
Devemos ter instâncias válidas de Position, Item, Container e ShippingOrder.
Procedimento
1. Adicionamos um Item válido a um Container; 2. Adicionamos um Item válido a um Container; 3. Adicionamos um Item válido a um Container; 4. Encerrar o Container anteriormente utilizado; 5. Passamos o *status* de uma ShippingOrder para *"AWAITS_TREATMENT"*; 6. Verificamos se, ao invocarmos o método *setStatus()* com o input A como parâmetro, nos é atirada uma OrderException; 7. Verificamos se, ao invocarmos o método *setStatus()* com o input B como parâmetro, nos é atirada uma OrderException; 8. Passamos o *status* de uma ShippingOrder para *"IN_TREATMENT"*; 9. Verificamos se, ao invocarmos o método *setStatus()* com o input C como parâmetro, nos é atirada uma OrderException; 10. Verificamos se, ao invocarmos o método *setStatus()* com o input B como parâmetro, nos é atirada uma OrderException; 11. Adicionamos o Container anteriormente utilizado à ShippingOrder; 12. Passamos o *status* de uma ShippingOrder para *"CLOSED"*; 13. Verificamos se, ao invocarmos o método *setStatus()* com o input C como parâmetro, nos é atirada uma OrderException; 14. Verificamos se, ao invocarmos o método *setStatus()* com o input D como parâmetro, nos é atirada uma OrderException;

Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00014	A, Bx2, CX2, D;	A – OrderStatus.CLOSED; B – OrderStatus.SHIPPED; C – OrderStatus.AWAITS_TREATMENT; D – OrderStatus.IN_TREATMENT;	A, B – **throws** OrderException;	FAIL

Test Case ID	TC#15	**Dependências**	--Nenhumas--

Data de Execução	**Hora de Execução**	***Tester***	**Pass/Fail**	**Resumo da Falha**
07/12/2020	19:30	8160207 - Nuno Matos	PASS	

Pré-condições				
Devemos ter ficheiros JSON criados.				
Procedimento				
1. Criamos uma instância de PackingGUI; 2. Verificamos se, ao invocarmos o método *validate()* usando o input A como parâmetro, deve-nos ser retornado um valor *true*; 3. Verificamos se, ao invocarmos o método *validate()* usando o input A como parâmetro, deve-nos ser retornado um valor *false*;				
Caso de Uso	**Inputs**	**Valores do Inputs**	**Resultados Esperados**	**Pass/Fail/Untested**
UC00015	A, B;	A – caminho/nome de um ficheiro JSON válido; B – caminho/nome de um ficheiro JSON inválido;	A – *true*; B – *false*;	PASS