LEI_ESII2021_TP1_GRUPOU

CASOS DE USO

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;

CASOS DE USO

ID: UC0001

NOME: Alterar a coordenada *X* de uma Position.

DESCRIÇÃO: Alterar o valor da coordenada *X* de uma Position.

PRECONDIÇÃO: Devemos ter uma instância de Position criada.

FLUXO NORMAL: Invocamos o método *setX()* da classe Position, com um valor inteiro como parâmetro, verifica-se se esse valor é maior ou igual a zero, e, se tal é observado, é substituida a coordenada *X*.

FLUXO ALTERNATIVO: Se invocarmos o método *setX()*, o valor passado como parâmetro for inferior a zero, é lançado uma PositionException.

ID: UC0002

NOME: Definir a coordenada *Y* de uma Position.

DESCRIÇÃO: Alterar o valor da coordenada *Y* de uma Position.

PRECONDIÇÃO: Já devemos ter uma instância de Position criada.

FLUXO NORMAL: Invocamos o método *setY()* da classe Position, com um valor inteiro como parâmetro, verifica-se se esse valor é maior ou igual a zero, e, se tal é observado, é substituída a coordenada *Y*.

FLUXO ALTERNATIVO: Se ao invocarmos o método *setY(),* o valor passado como parâmetro for inferior a zero, é lançado uma PositionException.

ID: UC0003

NOME: Definir a coordenada *Z* de uma Position.

DESCRIÇÃO: Alterar o valor da coordenada *Z* de uma Position.

PRECONDIÇÃO: Já devemos ter uma instância de Position criada.

FLUXO NORMAL: Invocamos o método *setZ()* da classe Position, com um valor inteiro como parâmetro, verifica-se se esse valor é maior ou igual a zero, e, se tal observado, é definida a coordenada *Z*.

FLUXO ALTERNATIVO: Se ao invocarmos o método *setZ()*, o valor passado como parâmetro for inferior a zero é lançado uma PositionException.

ID: UC0004

NOME: Adicionar um ItemPacked a um Container.

DESCRIÇÃO: Adicionar um ItemPacked a um Container já existente.

PRECONDIÇÃO: Já devemos ter uma instância de Container criada.

FLUXO NORMAL: Invocamos o método *addItem()* com 3 parâmetros, o Item a ser guardado, a Position em que o ItemPacked vai ser colocado e a Color usada para o item.

1. Se o ItemPacked a ser criado já existir no Container, o método retorna *false*.
2. Se o ItemPacked é inserido no Container, o volume do Container é atualizado, e no caso de não existir espaço suficiente na coleção esta deve ser ajustada de forma a caberem mais instâncias de ItemPacked. Por fim o método retorna *true*.

FLUXO ALTERNATIVO: Se ao invocarmos o método *addItem()* algum dos parâmetros for *null* ou o Container estiver *"closed"*, é lançada uma ContainerException.

ID: UC0005

NOME: Remover um ItemPacked de um Container;

DESCRIÇÃO: Remover um ItemPacked específico de dentro de um Container se este se encontrar lá inserido.

PRECONDIÇÃO: Já devemos ter uma instância de Item e de Container criadas e devidamente preenchidas.

FLUXO NORMAL: Invocando a função *removeItem()*, usamos uma instância de Item como parâmetro, que corresponde ao que queremos que seja eliminado do Container. A função vai procurar o Item mencionado, e depois temos dois caminhos possíveis:

1. Se encontrar o Item dentro do Container, procede á sua remoção de dentro do mesmo, o volume é atualizado, e no fim retorna *true*.
2. Se o Item não for localizado então não elimina nada, e simplesmente retorna *false*.

FLUXO ALTERNATIVO: Se ao invocarmos a função o Container se encontrar "*closed",* ou se o parâmetro introduzido na sua invocação for um *null* então é lançada um ContainerException.

ID: UC0006

NOME: Fechar um Container.

DESCRIÇÃO: Encerrar um Container já existente.

PRECONDIÇÃO: Já devemos ter uma instância de Container criada e devidamente preenchida.

FLUXO NORMAL: Invocando o método *close()*, é invocada o método *validate()* de modo a validar o Container. Passada a validação, o Container é fechado.

FLUXO ALTERNATIVO: Se ao invocarmos o método *close():*

1. Durante a validação o volume ocupado for maior que o volume total do Container, é lançada uma ContainerException.
2. Se algum Item se encontrar fora dos limites do Container, ou se estender para fora desses limites, ou se um Item estiver com uma posição que o ponha em conflito com outro, então é-nos lançado um PositionException.

ID: UC0007

NOME: Obter um Item específico.

DESCRIÇÃO: Através da sua referência, obter uma instância de um Item.

PRECONDIÇÃO: Já devemos ter uma instância de Container criada e devidamente preenchida.

FLUXO NORMAL: Invocamos o método *getItem()*, utilizando uma String correspondente à referência do Item que queremos obter. Depois de encontrado no Container, é nos retornado o dito Item.

Nota: Não é especificado o que acontece com o método caso o Item não seja detetado.

FLUXO ALTERNATIVO: —Nenhum—

ID: UC0008

NOME: Validar um Container.

DESCRIÇÃO: Verificar se um Container se encontra devidamente estruturado.

PRECONDIÇÃO: Já devemos ter uma instância de Container criada e devidamente preenchida.

FLUXO NORMAL: Ao invocarmos o método *validate()* é nos analisada toda a estrutura e informação do Container, verificando se o volume ocupado é menor ou igual ao volume total, se todos os ItemPackeds se encontram dentro do Container, sem nenhum ocupar posições conflituosas. Nenhuma exceção deve ser atirada.

FLUXO ALTERNATIVO: Ao invocarmos o método *validate()*:

1. Se o volume ocupado for maior que o volume total do Container é lançada uma ContainerException.
2. Se algum ItemPacked estiver posicionado fora dos limites do contentor, ou estiver em sobreposição com outro, é lançada uma ContainerException.

ID: UC0009

NOME: Adicionar um Container a uma ShippingOrder.

DESCRIÇÃO: Adicionar um novo Container válido a uma ShippingOrder já existente. O Container a ser introduzido não pode já existir na ShippingOrder.

PRECONDIÇÃO: Já devemos ter uma instância de ShippingOrder e de Container criadas e preenchidas.

FLUXO NORMAL:

1. Invocamos o método *addContainer()* da classe ShippingOrder, com uma instância de Container válida como parâmetro, verifica-se se o Container já existe na ShippingOrder, e se este não for o caso, adicionamo-lo e retornamos *true*.
2. Se invocarmos o método *addContainer()* com um Container já existente na ShippingOrder, o método ao verificar se ele já lá existe através do método *existsContainer()*, vai detetá-lo, não o vai introduzir outra vez, e retorna *false*.

FLUXO ALTERNATIVO: Se ao invocarmos o método *addContainer()* sem nenhum parâmetro, ou o Container estiver aberto, é-nos lançado uma ContainerException. Se o status da ShippingOrder for diferente de *"IN_TREATMENT"* é lançado uma OrderException.

ID: UC0010

NOME: Verificar se um Container existe numa ShippingOrder.

DESCRIÇÃO: Verificar se um Container específico já se encontra dentro de uma ShippingOrder ou não.

PRECONDIÇÃO: Já devemos ter uma instância válida de Container e de ShippingOrder criadas e preenchidas.

FLUXO NORMAL: Invocamos o método *existsContainer()*, usando uma instância de Container como parâmetro. Comparamos as instâncias de Container existentes na ShippingOrder com a instância introduzida como parâmetro.

1. Se o Container é detetado dentro da ShippingOrder retorna-se *true*;
2. Se o Container não é detetado, retorna-se *false*.

FLUXO ALTERNATIVO: —Nenhum—

ID: UC0011

NOME: Encontrar a posição de um Container.

DESCRIÇÃO: Usando a sua referência, procurar um Container específico numa ShippingOrder, e, se encontrado, devolver o número correspondente à sua posição na ShippingOrder.

PRECONDIÇÃO: Já devemos ter uma instância válida de ShippingOrder e preenchida.

FLUXO NORMAL: Ao invocarmos o método *findContainer()*, introduzimos como parâmetro uma String correspondente à referência de um Container. Percorremos a ShippingOrder, e se encontrarmos um Container com a referência igual à introduzida como parâmetro, devolvemos um int que corresponde à sua posição.

Nota: Não é especificado o que acontece com o método caso o Container não seja detetado.

FLUXO ALTERNATIVO: --Nenhum--

ID: UC0012

NOME: Validar todos os Containers.

DESCRIÇÃO: Verificar se todos os Containers dentro de uma ShippingOrder são válidos ou não.

PRECONDIÇÃO: Já devemos ter uma instância válida de ShippingOrder criada e preenchida.

FLUXO NORMAL: Ao invocarmos a função *validate()* não devemos obter nenhuma exceção.

FLUXO ALTERNATIVO:

1. Se o volume ocupado de um Container for maior que o volume total do mesmo, então é-nos lançado um ContainerException.
2. Se algum Item se encontrar fora dos limites do Container, ou se estender para fora desses limites, ou se um Item estiver com uma posição que o ponha em conflito com outro, então é-nos lançado um PositionException.

ID: UC0013

NOME: Remover um Container;

DESCRIÇÃO: Remover um Container específico de dentro de uma ShippingOrder se este se encontrar lá inserido.

PRECONDIÇÃO: Já devemos ter uma instância de ShippingOrder e de Container criadas e devidamente preenchidas.

FLUXO NORMAL: Invocando a função *removeContainer()*, usamos uma instância de Container como parâmetro, que corresponde ao que queremos que seja eliminado da ShippingOrder. A função vai invocar a função *existsContainer()*, tendo depois duas possibilidades de execução:

3. Se encontrar o Container dentro da ShippingOrder, procede á sua remoção de dentro do mesmo, e no fim retorna *true*.
4. Se o Container não for localizado então não elimina nada, e simplesmente retorna *false*.

FLUXO ALTERNATIVO: Se ao invocarmos a função o *status* da ShippingOrder não for *"IN_TREATMENT"* então é lançada uma OrderException. Se o parâmetro introduzido na sua invocação for um *null* é lançada uma ContainerException.

ID: UC0014

NOME: Alterar o estado de uma ShippingOrder.

DESCRIÇÃO: Alterar o estado de uma ShippingOrder existente.

PRECONDIÇÃO: Já devemos ter uma instância de ShippingOrder criada e preenchida.

FLUXO NORMAL: Ao invocarmos a função *setStatus()*, usamos um variável do tipo OrderStatus como parâmetro, que corresponde ao *status* que queremos que a ShippingOrder tome.

1. Se o *status* de parâmetro for *"IN_TREATMENT"* e o *status* atual da ShippingOrder for *"AWAITS_TREATMENT"* então o *status* muda.
2. Se o *status* de parâmetro for *"CLOSED"* e o *status* atual da ShippingOrder for *"IN_TREATMENT"*, se tivermos pelo menos um Container inserido, e se a ShippingOrder for validada corretamente, então o *status* muda.
3. Se o *status* de parâmetro for *"SHIPPED"* e o *status* atual da ShippingOrder for *"CLOSED"* então o *status* muda.

FLUXO ALTERNATIVO: Se ao invocarmos a função e o *status* usado como parâmetro não for compatível com o *status* atual da ShippingOrder então é lançado uma OrderException. Se ao validarmos os Containers se verificar que um tem um volume ocupado maior que o seu volume total é lançado uma ContainerException, e se algum Item se encontrar fora dos limites do Container, ou se estender para fora desses limites, ou se um Item estiver com uma posição que o ponha em conflito com outro, então é-nos lançado um PositionException.

ID: UC0015

NOME: Validar um ficheiro JSON.

DESCRIÇÃO: Verificar se um ficheiro JSON correspondente a uma ShippingOrder se encontra devidamente formado ou não para que o possamos depois visualizar num formato gráfico.

PRECONDIÇÃO: Devemos ter um ficheiro JSON correspondente a uma ShippingOrder já criado e pronto para validar.

FLUXO NORMAL: Ao invocarmos a função *validate()*, inserimos como parâmetro uma String que corresponde à localização e nome de um ficheiro JSON. Este ficheiro será analisado para se verificar se pode ser usado na criação de uma representação gráfica ou não.

1. Se o ficheiro se encontrar devidamente estruturado, a função retorna *true*.
2. Se o ficheiro não se encontra apto para a representação gráfica, retorna-se *false.*

FLUXO ALTERNATIVO: —Nenhum—
