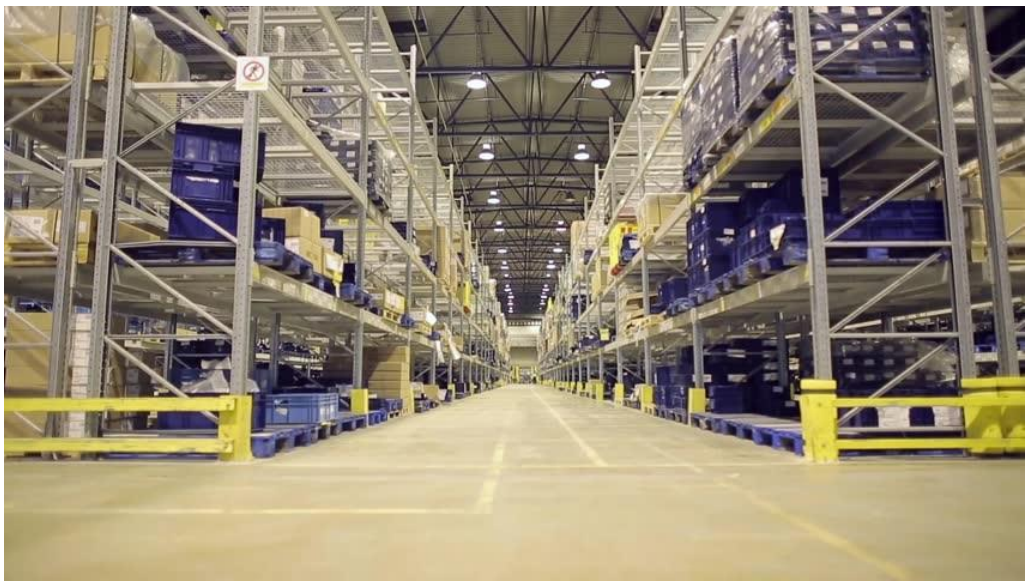
IPS – Escola Superior de Tecnologia de Setúbal

Programação Orientada para Objetos

2022/2023

PROJETO

Fase 2



Joana Silva **202100190**

Laura Costa **202100242**

Docente: **André Sanguinetti**

Licenciatura em Engenharia Informática

ÍNDICE

DESCRIÇÃO DO PROGRAMA

O programa consiste na implementação de uma simulação de um centro de distribuição de produtos utilizando Java.

O centro de distribuição é um espaço retangular dividido em posições, com locais de armazenamento, veículos AGV e paredes. Os produtos são embalados e transportados para os locais de armazenamento, utilizando diferentes tipos de embalagens e veículos. A simulação ocorre com a definição do armazém, adição de veículos e locais de armazenamento, entrega e recolha de produtos. O programa enfatiza o uso correto dos conceitos de POO e visa uma modelagem bem pensada para facilitar o desenvolvimento e manutenção do sistema.

Na segunda fase, foi desenvolvida uma interface gráfica em JavaFX que permite visualizar o centro de distribuição com todos os elementos, incluindo as paredes, locais de entrega e recolha de produtos, prateleiras e veículos AGV. A interface gráfica mostra o armazém e permite ao utilizador movimentar os veículos clicando neles, é possível também guiar um veículo até a uma prateleira e descarregar a sua carga para a mesma. Também é exibido o número de produtos armazenados e o peso total dos produtos armazenados, com atualização dinâmica desses valores.

MANUAL TÉCNICO

Classes

Classe Product

A classe **Product** representa um produto. Ela contém informações sobre o nome do produto, seu peso, tipo de produto e informações relacionadas ao empacotamento.

Principais atributos:

- **ID:** Um inteiro estático que representa o ID do produto.
- **name:** Uma string que representa o nome do produto.
- **weight:** Um valor decimal que indica o peso do produto em quilogramas.
- **productType:** Um objeto do tipo ProductType que representa o tipo de produto.
- **isPackaged:** Um booleano que indica se o produto está empacotado ou não.
- **packagingType:** Um objeto do tipo PackagingType que representa o tipo de embalagem do produto.

Principais métodos:

- **generateRandomNumber(double min, double max):** Gera um número aleatório no intervalo especificado.
- **isPackaged():** Retorna um indicador se o produto está embalado ou não.
- **setWeights():** gera pesos para cada produto com base no seu tipo.

Classe Bag

A classe **Bag** representa uma sacola usada para empacotar produtos. Ela possui informações sobre o peso máximo que a sacola pode suportar, os produtos presentes na sacola, o peso atual da sacola e um código identificador.

Principais atributos:

- **maxWeight:** Um valor decimal que representa o peso máximo que a sacola pode suportar.
- **productsInTheBag:** Uma lista de objetos do tipo Product que representa os produtos presentes na sacola.
- **currentWeight:** Um valor decimal que indica o peso atual da sacola.
- **code:** Um número inteiro que serve como código identificador para a sacola.
- **Loaded:** Valor booleano que indica se a mala foi carregada em algum veículo.

Principais métodos:

- **isFull():** Verifica se a sacola está cheia.
- **pack(Product p):** Empacota um produto na sacola.

Classe Box

A classe **Box** representa uma caixa usada para empacotar produtos. Ela possui informações sobre o peso máximo que a caixa pode suportar, o peso atual da caixa, a lista de produtos presentes na caixa e um código identificador.

Principais atributos:

- **maxWeight:** Um valor decimal que representa o peso máximo que a caixa pode suportar.
- **currentWeight:** Um valor decimal que indica o peso atual da caixa.
- **productsInTheBox:** Uma lista de objetos do tipo Product que representa os produtos presentes na caixa.
- **code:** Um número inteiro que serve como código identificador para a caixa.
- **Loaded:** Valor booleano que indica se a caixa foi carregada em algum veículo.

Principais métodos:

- **pack(Product p):** Empacota um produto na caixa.
- **isFull():** Verifica se a caixa está cheia.

Classe CardBox

A classe **CardBox** representa uma caixa de cartão usada para empacotar produtos. Ela possui informações sobre o peso máximo que a caixa de cartão pode suportar, o peso atual da caixa, a lista de produtos presentes na caixa, o número máximo de produtos que a caixa pode conter, o número atual de produtos na caixa e um código identificador.

Principais atributos:

- **maxWeight:** Um valor decimal que representa o peso máximo que a caixa de cartão pode suportar.
- **currentWeight:** Um valor decimal que indica o peso atual da caixa de cartão.
- **productsInTheBox:** Uma lista de objetos do tipo Product que representa os produtos presentes na caixa de cartão.
- **maxProducts:** Um número inteiro que indica o número máximo de produtos que a caixa de cartão pode conter.
- **currentProducts:** Um número inteiro que indica o número atual de produtos na caixa de cartão.
- **code:** Um número inteiro que serve como código identificador para a caixa de cartão.
- **Loaded:** Valor booleano que indica se a caixa de cartão foi carregada em algum veículo.
- **putIntoPallet:** Valor booleano que indica se a caixa de cartão foi colocada em alguma pallet.

Principais métodos:

- **pack(Product p):** Empacota um produto na caixa de cartão.

Classe Pallet

A classe **Pallet** representa um palete usado para embalar caixas de cartão. Possui as seguintes variáveis principais:

Principais atributos:

- **maxWeight:** Um double que representa o peso máximo do palete.
- **currentWeight:** Um double que representa o peso atual do palete.
- **maxBoxes:** Um int que representa o número máximo de caixas que o palete pode conter.
- **currentBoxes:** Uma lista de objetos CardBox que representa as caixas atualmente no palete.
- **code:** Um int que representa o código associado ao palete.
- **Loaded:** Valor booleano que indica se a pallet foi carregada em algum veículo.

Principais métodos

- **pack(Product p):** Um método que tenta embalar um produto no palete. Como essa classe representa um palete para embalar caixas de cartão, não é possível embalar produtos individualmente no palete. Portanto, esse método sempre retorna false.
- **pack(CardBox b):** Um método que tenta embalar uma caixa de cartão no palete. Ele verifica se o número de caixas atual no palete é menor que o número máximo permitido e se o peso atual mais o peso da caixa não excede o peso máximo permitido. Se as condições forem atendidas, a caixa é embalada no palete, o peso atual é atualizado e o método retorna true. Caso contrário, o método retorna false.

Classe DistributionCenter

A classe **DistributionCenter** representa um centro de distribuição. Ela possui vários atributos e funções relacionadas ao gerenciamento do centro de distribuição, como criação do layout, embalagem de produtos, posicionamento de veículos, carga de veículos, entre outros.

Principais atributos:

- **positionsList**: Uma lista de posições no centro de distribuição.
- **vehicles**: Uma lista de veículos presentes no centro de distribuição.
- **products**: Uma lista de produtos presentes no centro de distribuição.
- **shelves**: Uma lista de prateleiras presentes no centro de distribuição.

Principais métodos:

- **packProducts(List<Product> products):** Recebe uma lista de produtos e realiza a embalagem dos produtos de acordo com suas características. Retorna uma lista de itens embalados.
- **createDistributionCenter():** Cria o layout do centro de distribuição, preenchendo a matriz de posições com as diferentes áreas (paredes, prateleiras, etc.).
- **positionMatrixToList(Position position[][]):** Converte a matriz de posições em uma lista de posições.
- **printStorageMatrix(Position[][] matrix):** Imprime o layout do centro de distribuição na forma de uma matriz.
- **setVehiclesStartingPositions(List<Vehicle> vehiclesList, List<Position> positionsList):** Define as posições iniciais dos veículos no centro de distribuição. (Cada posição “Collect” do centro de distribuição pode ter no máximo 2 veiculos).
- **findVehicles(List<Vehicle> vehicles, String vehicleType):** Devolve uma lista de veículos que correspondem ao tipo de veículo procurado.
- **loadVehicles(List packedItems, List<Vehicle> vehicles):** Coloca os objetos embalados nos veículos disponíveis.

- **addToCardBoxes(List<Object> packedItems, Product p):** Adiciona um objeto Product a uma CardBox existente na lista ou cria uma nova CardBox para embalar o objeto Product, atualizando a lista de itens embalados.
- **addCardBoxesToPallets(List<Object> packedItems):** Organiza as CardBoxes em paletes, criando paletes conforme necessário e movendo as CardBoxes para os paletes disponíveis, atualizando a lista de itens embalados.

Classe Position

A classe **Position** representa uma posição dentro de um centro de distribuição. Ela possui atributos como as coordenadas x e y, o nome da posição, a capacidade máxima de veículos que podem ocupar a posição e uma lista de veículos atualmente presentes nessa posição.

Principais atributos:

- **MAX_VEHICLES**: A capacidade máxima de veículos que podem ocupar a posição.
- **vehicleInPosition**: Guarda o objeto Veículo que se encontra na posição.

Principais métodos:

- **addToPosition(Vehicle vehicle):** Adiciona um veículo à posição.

Classe ProductsCSV

A classe **ProductsCSV** implementa a interface CSV e é responsável por criar um arquivo CSV contendo informações de produtos, bem como realizar a leitura desse arquivo e retornar uma lista de objetos Product.

Principais métodos:

- **createCSV():** Implementação do método da interface CSV. Cria um arquivo CSV chamado "produtos.csv" e escreve nele informações sobre produtos. Gera aleatoriamente um tipo de produto para cada item e escreve o nome do item e o tipo no arquivo CSV.
- **readFromCSV():** Lê o arquivo CSV "produtos.csv" e retorna uma lista de objetos Product com base nas informações contidas no arquivo. Ele conta o número de linhas e colunas do arquivo, cria uma matriz para armazenar os dados, lê as linhas do arquivo CSV, extrai as informações de nome do produto e tipo de produto, cria objetos Product com essas informações e os adiciona à lista de produtos.

Classe Shelf

A classe **Shelf** permite gerenciar a posição e os produtos de uma prateleira. É possível adicionar produtos à prateleira desde que estejam embalados e verificar se a prateleira está vazia. A lista de produtos pode ser obtida para fins de visualização ou processamento adicional.

Principais atributos:

- **position**: Uma instância da classe Position que representa a posição da prateleira.
- **products**: Uma lista de objetos que representa os produtos armazenados na prateleira.

Principais métodos:

- **addToShelf(List products):** Adiciona um produto embalado à prateleira.
- **isEmpty():** Verifica se a prateleira está vazia. Retorna true se a lista de produtos estiver vazia e false caso contrário.
- **getProducts():** Retorna a lista de produtos da prateleira.
- **getWeight():** Retorna o peso atual que se encontra na prateleira.

Classe Vehicle

A classe **Vehicle** é a superclasse para todos os outros veículos. Ela representa um veículo genérico com funcionalidades básicas, como movimentação, carga e descarga. Os veículos específicos herdam dessa classe para adicionar comportamentos adicionais. A classe possui métodos para obter a quantidade de carga, verificar a disponibilidade, mover-se em diferentes direções e gerenciar posições.

Principais atributos:

- **currentPosition:** Variável que armazena a posição atual em que se encontra o veículo.
- **destination:** Variável que armazena a posição para a qual o veículo terá de se deslocar.
- **available**: Um booleano que indica se o veículo está disponível ou não.

Principais métodos:

- **moveUpwards(List<Position> positions)**Recebe uma lista de posições como parâmetro e verifica se a posição atual do veículo e a lista de posições são nulas. Em seguida, percorre a lista de posições em busca de uma posição acima da posição atual. Realiza várias verificações para determinar se o movimento é válido ou se há obstáculos, como prateleiras ou outros veículos. Se o movimento for possível, atualiza a posição do veículo e retorna a nova posição. Caso contrário, retorna null.
- **moveDownwards(List<Position> positions)** Verifica se a posição atual do veículo e a lista de posições são nulas e percorre a lista de posições em busca de uma posição abaixo da posição atual. Realiza as mesmas verificações de validade do movimento e obstáculos. Se o movimento for válido, atualiza a posição do veículo e retorna a nova posição. Caso contrário, retorna null.
- **loadShelf(Shelf shelf)** Adiciona a carga atual do veículo à prateleira, define o conteúdo do veículo como nulo e redefine o peso do veículo como zero. Atualiza o estado do veículo e da prateleira após o carregamento.

- **moveRight(List<Position> positions):** Verifica se a posição atual do veículo e a lista de posições são nulas. Em seguida, percorre as posições e verifica se há uma posição à direita da posição atual. Se a posição for válida e não houver obstáculos, o método atualiza a posição do veículo e retorna a nova posição. Caso contrário, retorna null.
- **moveLeft(List<Position> positions):**Verifica se a posição atual do veículo e a lista de posições são nulas. Em seguida, percorre as posições e verifica se há uma posição à esquerda da posição atual. Se a posição for válida e não houver obstáculos, o método atualiza a posição do veículo e retorna a nova posição. Caso contrário, retorna null.
- **isAvailable():** Verifica se o veículo está disponível.
- **int getWeight():** Retorna o peso atual do veículo.

Classe AGC

A classe **AGC** representa um Carrinho Guiado Automaticamente (AGC, na sigla em inglês). Ela é responsável por gerir a posição atual, local de recolha e local de entrega do AGC, bem como o transporte de carga.

Principais variáveis:

- **MAX_WEIGHT:** Constante que define o peso máximo que o AGC pode transportar.
- **currentWeight:** Variável que armazena o peso atual da carga do AGC.
- **currentCargo:** Lista que armazena a carga atual do AGC.

Principais métodos:

- **addBag(Bag bag):** Adiciona uma bolsa à carga do AGC. Verifica se o peso da bolsa somado ao peso atual do AGC não excede o limite máximo de peso. Retorna true se a bolsa for adicionada com sucesso e false caso contrário.
- **addBox(Box box):** Adiciona uma caixa à carga do AGC. Verifica se o peso da caixa somado ao peso atual do AGC não excede o limite máximo de peso. Retorna true se a caixa for adicionada com sucesso e false caso contrário.

Classe DeliveryCart

A classe **DeliveryCart** representa um carrinho de entrega. Ela é responsável por gerir a posição atual, o peso e a carga do carrinho.

Principais variáveis:

- **MAX_WEIGHT:** Constante que define o peso máximo que o carrinho de entrega pode transportar.
- **currentWeight:** Variável que armazena o peso atual da carga do carrinho.
- **isTugged:** Um booleano que indica se o carrinho está atrelado.
- **currentCargo:** Lista que armazena a carga atual do carrinho.

Principais métodos:

- **addBag(Bag bag):** Adiciona uma bolsa à carga do carrinho de entrega. Verifica se o peso da bolsa somado ao peso atual do carrinho não excede o limite máximo de peso. Retorna true se a bolsa for adicionada com sucesso e false caso contrário.
- **addBox(Box box):** Adiciona uma caixa à carga do carrinho de entrega. Verifica se o peso da caixa somado ao peso atual do carrinho não excede o limite máximo de peso. Retorna true se a caixa for adicionada com sucesso e false caso contrário.

Classe TugVehicle

A classe **TugVehicle** representa um veículo de reboque. Ela é responsável por gerir os carrinhos de entrega rebocados.

Principais variáveis:

- **towedAGC:** Variável que armazena o carrinho de entrega que se encontra atrelado ao veículo.

Principais métodos:

- **putDCIntoTugVehicle(List<Vehicle> vehicles):** Recebe uma lista de veículos e coloca os carrinhos de entrega nessa lista no veículo de reboque. Verifica se o veículo é um carrinho de entrega e se está disponível. Se houver carrinhos de entrega com carga e o veículo de reboque não estiver a rebocar nenhum carrinho no momento, o carrinho de entrega é “atrelado” ao veículo.

Classe ULC

A classe **ULC** representa um Unidade de Transporte de Carga (ULC). Ela é responsável por gerir o caminho, a carga atual e a posição atual da ULC.

Principais variáveis:

- **path:** Lista que armazena o caminho da ULC, representado por uma lista de posições.
- **currentWeight:** Variável que armazena o peso atual no veículo.
- **currentCargo:** Lista que armazena a carga atual da ULC, representada por uma lista de paletes.
- **MAX_PALLETS:** Constante que define o número máximo de paletes que a ULC pode transportar.
- **currentPosition:** Variável que armazena a posição atual da ULC.

Principais métodos:

- **addPallet(Pallet p):** Adiciona um palete à carga atual da ULC. Verifica se a quantidade de paletes na carga atual é menor que o máximo permitido. Se for, o palete é adicionado à carga e o método retorna verdadeiro; caso contrário, o método retorna falso.

Classe VehiclesCSV

A classe **VehiclesCSV** implementa a interface CSV e é responsável por criar um arquivo CSV de veículos e também realizar a leitura desse arquivo.

Principais métodos:

- **createCSV():** Implementa o método da interface CSV para criar o arquivo CSV de veículos. Ele gera um arquivo "veiculos.csv" e escreve os cabeçalhos e informações dos veículos aleatórios no arquivo. Os tipos de veículos são definidos em um array vehicleType, e cada veículo é gerado com um nome aleatório e um tipo aleatório a partir desse array.
- **readFromCSV():** Implementa o método da interface CSV para ler o arquivo CSV de veículos. Ele lê o arquivo "veiculos.csv", conta o número de linhas e colunas, cria uma matriz data com as dimensões adequadas e, em seguida, itera sobre os dados lidos para criar os objetos Vehicle correspondentes aos tipos de veículos encontrados. Os veículos são adicionados a uma lista vehicles que é retornada no final.

Classe WarehouseCSV

A classe **WarehouseCSV** implementa a interface CSV e é responsável por criar um arquivo CSV para armazenar informações sobre o armazém, bem como realizar a leitura desse arquivo

Principais métodos:

- **createCSV():** Implementa o método da interface CSV para criar o arquivo CSV do armazém. Ele gera um arquivo chamado "armazem.csv" e escreve as dimensões do armazém no arquivo. As dimensões do armazém, como comprimento e largura, são definidas e escritas no arquivo.
- **readFromCSV():** Implementa o método da interface CSV para ler o arquivo CSV do armazém. Ele lê o arquivo "armazem.csv" e extrai as dimensões do armazém. Essas dimensões são armazenadas em um array size e são retornadas no final. O método utiliza a classe Scanner para ler o arquivo linha por linha e obter as informações necessárias.

Classe Camera

A classe **Camera** representa um sensor de câmera. Ela é uma subclasse da classe Sensor e possui os seguintes atributos:

- **range:** Representa a faixa de alcance da câmera.
- **angle:** Representa o ângulo de visão da câmera.

Principais métodos:

- **getRange():** Retorna a faixa de alcance da câmera.
- **setRange(int range):** Define a faixa de alcance da câmera.
- **getAngle():** Retorna o ângulo de visão da câmera.
- **setAngle(int angle):** Define o ângulo de visão da câmera.

Classe Lidar

A classe **Lidar** representa um sensor Lidar. Ela é uma subclasse da classe Sensor e possui os seguintes atributos:

- **range:** Representa a faixa de alcance do sensor Lidar.
- **angle:** Representa o ângulo de visão do sensor Lidar.

Principais métodos:

- **getRange():** Retorna a faixa de alcance do sensor Lidar.
- **setRange(int range):** Define a faixa de alcance do sensor Lidar.
- **getAngle():** Retorna o ângulo de visão do sensor Lidar.
- **setAngle(int angle):** Define o ângulo de visão do sensor Lidar.

Classe Sensor

A classe Sensor representa um sensor genérico. Ela possui os seguintes atributos:

- **type:** Representa o tipo do sensor.

Principais métodos:

- **Sensor(String type):** Construtor da classe Sensor que inicializa o sensor com o tipo especificado.
- **getType():** Retorna o tipo do sensor.
- **setType(String type):** Define o tipo do sensor.

Classe UltrasonicSensor

A classe UltrasonicSensor representa um sensor ultrassônico. Ela estende a classe Sensor e possui os seguintes atributos adicionais:

- **range:** Representa o alcance do sensor ultrassônico.
- **angle:** Representa o ângulo do sensor ultrassônico.

Principais métodos:

- **getRange():** Retorna o alcance do sensor ultrassônico.
- **setRange(int range):** Define o alcance do sensor ultrassônico.
- **getAngle():** Retorna o ângulo do sensor ultrassônico.
- **setAngle(int angle):** Define o ângulo do sensor ultrassônico.

Interfaces

Packaging

A interface **Packaging** representa uma interface de embalagem. Ela define um único método:

- **boolean pack(Product p):** Um método que empacota um produto na embalagem. Ele recebe um objeto Product como parâmetro e retorna true se o produto for empacotado com sucesso na embalagem, caso contrário, retorna false.

CSV

A interface **CSV** define o contrato para a criação de um arquivo CSV. Ela possui um único método:

- **void createCSV():** Um método que cria um arquivo CSV. Esse método não possui parâmetros e não retorna nenhum valor.

Enumeradores

PackagingType

A enumeração **PackagingType** representa os tipos de embalagens disponíveis. Ela possui os seguintes valores:

- **BAG:** Saco
- **BOX:** Caixa
- **CARD_BOX:** Caixa de papelão
- **PALLET:** Pallet

Principais métodos:

- **int getCode():** Retorna o código associado ao tipo de embalagem. Esse método utiliza uma estrutura switch para retornar o código correspondente a cada tipo de embalagem. Os códigos são os seguintes:
  - **BAG:** 1001
  - **BOX:** 2002
  - **CARD_BOX:** 3003
  - **PALLET:** 4004

ProductType

A enumeração **ProductType** representa os tipos de produtos disponíveis. Ela possui os seguintes valores:

- **CLOTHING:** Vestuário
- **ACCESSORY:** Acessório
- **SMALL_TOY:** Brinquedo pequeno
- **LARGE_TOY:** Brinquedo grande
- **SMALL_ELECTRONIC_EQUIPMENT:** Equipamento eletrônico pequeno
- **LARGE_ELECTRONIC_EQUIPMENT:** Equipamento eletrônico grande
- **BOOK:** Livro