Curso LEIC

<LEIC_924016_POO>

Ano letivo 2024/2025

Projeto:

<Projeto_NAUTILUS>

Alunos:

<Gonçalo Taborda, Nº14065>

<Henrique Domingos, Nº14064>

<Manuel Roque, Nº14061>

Paço d'Arcos, 18/01/2025

Índice

Introdução:

Este projeto tem como objetivo principal desenvolvermos o máximo de capacidades utilizando os conhecimentos que aprendemos ao longo do semestre na respetiva UC.

Simula um sistema de portos onde no mesmo é possível fazer imensas ações com os respetivos marinheiros / embarcações que o frequentem. Para a realização do trabalho, decidimos implementar três diferentes menus interativos, com a possibilidade de visualizar todas as funções que foram desenvolvidas ao longo do nosso código e assim percebermos, como as mesmas funcionam em termos práticos.

Descrição dos Algoritmos:

➔ Irei descrever as funções mais relevantes que foram criadas ao longo deste projeto de forma sucinta.

➢ Classe Porto:

- public void salvarDados(String caminhoFicheiro);

  - Propósito Geral: Salva o estado do objeto num ficheiro.
  - Parâmetro: caminhoFicheiro (String).
  - Retorno: void, sem retorno.
  - Algoritmo: Usa try com recursos, escreve o estado, exibe mensagem.
  - Uso: Salvar dados antes de fechar, criar pontos de restauração, exportar dados.

- public static Porto carregarDados(String caminhoFicheiro);

  - Propósito Geral: Carrega o estado do objeto Porto de um ficheiro.
  - Parâmetro: caminhoFicheiro (String).
  - Retorno: Porto, o objeto carregado ou null se ocorrer um erro.
  - Algoritmo: Usa try com recursos, lê o objeto, exibe mensagem de sucesso ou erro.
  - Uso: Recuperar o estado do objeto Porto previamente salvo em ficheiro.

- public void exportarEmbarcacoes(String nomeFicheiro);

  - Propósito Geral: Exporta uma lista de embarcações para um ficheiro.
  - Parâmetro: nomeFicheiro (String).
  - Retorno: void, sem retorno.
  - Algoritmo: Usa try com recursos, escreve informações das embarcações atracadas e em missão, exibe mensagem de sucesso ou erro.
  - Uso: Salvar uma lista detalhada de embarcações atracadas e em missão num ficheiro.

- public void registarEmbarcacoes(Embarcacao embarcacao);

  - Propósito Geral: Registar uma embarcação no porto.
  - Parâmetro: embarcacao (Embarcacao).
  - Retorno: void, sem retorno.
  - Algoritmo: Verifica se a embarcação já está registada usando procurarEmbarcacao, lança uma exceção se já estiver registada, caso contrário, adiciona a embarcação a embarcacoesAtracadas e exibe uma mensagem de confirmação.
  - Uso: Registar novas embarcações no porto, garantindo que não haja duplicadas.

- public void registarMarinheiro(Marinheiro marinheiro);

  - Propósito Geral: Registar um marinheiro no porto.
  - Parâmetro: marinheiro (Marinheiro).
  - Retorno: void, sem retorno.

- Algoritmo: Verifica se o marinheiro já está registado usando procurarMarinheiro, lança uma exceção se já estiver registado, caso contrário, adiciona o marinheiro a marinheiros e exibe uma mensagem de confirmação.
- Uso: Registar novos marinheiros no porto, garantindo que não haja duplicados.

- public void ativarRadar() | public void desativar Radar();

  - Propósito Geral: Ativar ou desativar o radar
  - Parâmetro: Nenhum
  - Retorno: void, sem retorno.
  - Algoritmo: Verifica se o radar está ligado ou desligado, respetivamente, exibindo sempre uma mensagem com formato respetivo a ação do radar.
  - Uso: Ligar ou desligar o radar.

- public List <Embarcacao> detectarEmbarcacoes();

  - Propósito Geral: Detetar todas as embarcações (atracadas e em missão) utilizando o radar.
  - Parâmetro: Nenhum.
  - Retorno: List<Embarcacao>, lista de embarcações detetadas..
  - Algoritmo: Verifica se o radar está ligado, adiciona embarcações atracadas e em missão à lista de detetadas, exibe os detalhes das embarcações detetadas.
  - Uso: Detetar e listar todas as embarcações quando o radar está ativado.

- public List <Embarcacao> detectarEmbarcacoes();

  - Propósito Geral: Detetar todas as embarcações (atracadas e em missão) utilizando o radar.
  - Parâmetro: Nenhum.
  - Retorno: List<Embarcacao>, lista de embarcações detetadas..
  - Algoritmo: Verifica se o radar está ligado, adiciona embarcações atracadas e em missão à lista de detetadas, exibe os detalhes das embarcações detetadas.
  - Uso: Detetar e listar todas as embarcações quando o radar está ativado.

➢ Classe BarcoPatrulha:

- public void ligarHolofote() | public void desligarHolofote()

  - Propósito Geral: Ligar ou desligar o holofote.
  - Parâmetro: Nenhum.
  - Retorno: void, sem retorno.
  - Algoritmo: Verifica se o holofote está ligado ou desligado e exibe uma mensagem com formato respetivo da ação do holofote.
  - Uso: Acender ou desligar o holofote.

- public void ativarRadar() | public void desativar Radar();

  - Propósito Geral: Ativar ou desativar o radar

- Parâmetro: Nenhum
- Retorno: void, sem retorno.
- Algoritmo: Verifica se o radar está ligado ou desligado, respetivamente, exibindo sempre uma mensagem com formato respetivo a ação do radar.
- Uso: Ligar ou desligar o radar.

- public String toString()

  - Propósito Geral: Retorna uma representação em string de um objeto Barco de Patrulha.
  - Parâmetro: Nenhum.
  - Retorno: String, a representação formatada do objeto.
  - Algoritmo: Converte listas de motores e marinheiros em strings, remove vírgulas finais, formata e retorna a string com todas as informações do objeto.
  - Uso: Obter uma string detalhada e formatada representando um objeto Barco de Patrulha.

➢ Classe Embarcacao:

- public void adicionarMotores(List<Motor> novosMotores)

  - Propósito Geral: Adicionar novos motores ao barco.
  - Parâmetro novosMotores (List<Motor>).
  - Retorno: void, sem retorno.
  - Algoritmo: Inicializa a lista de motores se necessário, copia a lista de novos motores, adiciona motores que não estejam já na lista.
  - Uso: Adicionar motores ao barco evitando duplicatas.

- public void adicionarMarinheiros(Marinheiro marinheiro)

  - Propósito Geral: Adicionar um marinheiro à tripulação.
  - Parâmetro： marinheiro (Marinheiro).
  - Retorno: void, sem retorno.
  - Algoritmo: Verifica se o marinheiro não está alocado nem registado em outra tripulação, adiciona-o à tripulação, atualiza o estado do marinheiro, ou exibe mensagens se já estiver na tripulação ou em outra tripulação.
  - Uso: Adicionar marinheiros à tripulação evitando duplicatas e garantindo que não estejam alocados em outra embarcação.

- public void removeMarinheiros(Marinheiro marinheiro)

  - Propósito Geral: Remover um marinheiro da tripulação.
  - Parâmetro： marinheiro (Marinheiro).
  - Retorno: void, sem retorno.
  - Algoritmo: Remove o marinheiro da tripulação, atualiza o estado do marinheiro.
  - Uso: Remover marinheiros da tripulação e atualizar seu estado.

- Classe Idade:

  - public static Idade calcularIdade(LocalDate dataNascimento)

    - Propósito Geral: Calcular a idade baseada na data de nascimento.
    - Parâmetro: dataNascimento (LocalDate).
    - Retorno: Idade, instância com a quantidade de anos e meses.
    - Algoritmo: Obtém a data atual, calcula o período entre a data de nascimento e a data atual, retorna a idade em anos e meses.
    - Uso: Determinar a idade de uma pessoa ou objeto a partir da data de nascimento.

- Classe LanchaRapida:

  - public void ligarHolofote() | public void desligarHolofote()

    - Propósito Geral: Ligar ou desligar o holofote.
    - Parâmetro: Nenhum.
    - Retorno: void, sem retorno.
    - Algoritmo: Verifica se o holofote está ligado ou desligado e exibe uma mensagem com formato respetivo da ação do holofote.
    - Uso: Acender ou desligar o holofote.

- Classe Marinheiro:

  - public boolean isPatenteValida(Patente patente)

    - Propósito Geral: Verificar se uma patente é válida.
    - Parâmetro: patente (Patente).
    - Retorno: boolean, true se a patente for válida, caso contrário false.
    - Algoritmo: Itera através dos valores de Patente, retorna true se encontrar um valor correspondente, caso contrário, retorna false.
    - Uso: Validar se uma patente específica está entre os valores permitidos.

- Classe Motor:

  - public void abastecer()

    - Propósito Geral: Abastecer o tanque do motor.
    - Parâmetro: nenhum.
    - Retorno: void, sem retorno
    - Algoritmo: Exibe mensagens sobre o estado do combustível, simula um atraso de abastecimento usando Thread.sleep, chama o método abastecer do tanque e lida com possíveis interrupções durante o processo.

- Uso: Realizar o abastecimento de combustível e simular o tempo necessário para a operação.

- public String toString()

  - Propósito Geral: Retornar uma representação em string do objeto Motor.
  - Parâmetro: nenhum.
  - Retorno: String, a representação formatada do objeto Motor.
  - Algoritmo: Retorna uma string formatada com a cilindrada, potência e informações do tanque.
  - Uso: Obter uma string detalhada e formatada representando um objeto Motor.

➢ Classe NavioSuporte:

- public void ligarHolofote() | public void desligarHolofote()

  - Propósito Geral: Ligar ou desligar o holofote.
  - Parâmetro: Nenhum.
  - Retorno: void, sem retorno.
  - Algoritmo: Verifica se o holofote está ligado ou desligado e exibe uma mensagem com formato respetivo da ação do holofote.
  - Uso: Acender ou desligar o holofote.

- public void ativarRadar() | public void desativar Radar();

  - Propósito Geral: Ativar ou desativar o radar
  - Parâmetro: Nenhum
  - Retorno: void, sem retorno.
  - Algoritmo: Verifica se o radar está ligado ou desligado, respetivamente, exibindo sempre uma mensagem com formato respetivo a ação do radar.
  - Uso: Ligar ou desligar o radar.

- public String toString()

  - Propósito Geral: Retornar uma representação em string do objeto Navio de Suporte.
  - Parâmetro: Nenhum
  - Retorno: String, a representação formatada do objeto.
  - Algoritmo: Utiliza String.format para criar uma string com detalhes do navio, incluindo ID, nome, marca, modelo, data de fabricação, zona, capacidade de carga, número de camas, estado de missão, motores e tripulação.
  - Uso: Obter uma string detalhada e formatada representando um objeto Navio de Suporte.

➢ Classe TanqueCombustivel:

- public String toString()

- Propósito Geral Retornar uma representação em string do objeto, mostrando a capacidade e o tipo de combustível.
- Parâmetro: Nenhum.
- Retorno: String, a representação formatada do objeto.
- Algoritmo: Utiliza String.format para criar uma string com a capacidade e o tipo de combustível.
- Uso: Obter uma string detalhada e formatada representando as características do objeto.

➢ Enumeração TipoCombustivel:

- Propósito Definir os tipos de combustível disponíveis.
- Parâmetro: Nenhum.
- Retorno: enum, enumeração dos tipos de combustível.
- Algoritmo: Declara os tipos de combustível possíveis: GASOLINA, GASOLEO, ETANOL.
- Uso: Utilizada para especificar e restringir os tipos de combustível aceitos em várias partes do programa.

➢ Enumeração Patente:

- Propósito Definir os tipos de patentes disponíveis.
- Parâmetro: Nenhum.
- Retorno: enum, enumeração dos tipos de patentes.
- Algoritmo: Declara os tipos de combustível possíveis: OFICIAL, SARGENTO, PRACA.
- Uso: Utilizada para especificar e restringir os tipos de patentes aceitos nas descrições dos marinheiros.

➢ Enumeração Zona:

- Propósito Definir os tipos de Zona disponíveis.
- Parâmetro: Nenhum.
- Retorno: enum, enumeração dos tipos de Zona.
- Algoritmo: Declara os tipos de combustível possíveis: NORTE, SUL, LESTE, OESTE, nenhuma das anteriores (“ZONA INVÁLIDA”).
- Uso: Utilizada para especificar e restringir os tipos de zonas aceitos em várias partes do programa.

(“MAIN”)

➢ Classe Projeto:

- public void menu()

  - Propósito Geral: Exibir o menu principal interativo para o sistema do porto.

- Parâmetro: Nenhum.
- Retorno: void, sem retorno.
- Algoritmo:

    1. Exibe opções do menu.
    2. Lê a escolha do utilizador.
    3. Executa a ação correspondente à opção escolhida (modo de manutenção, modo de utilização, sair, ou exibe uma mensagem para opção inválida).

- Uso: Navegar entre diferentes modos de operação do sistema do porto.

- public void menuManutencao()

    - Propósito Geral: Exibir o menu de manutenção interativo para o sistema do porto.
    - Parâmetro: Nenhum.
    - Retorno: void, sem retorno.
    - Algoritmo (continuação do menu anterior):

        4. Exibe opções do menu.
        5. Lê a escolha do utilizador.
        6. Executa a ação correspondente à opção escolhida (modo de manutenção, modo de utilização, sair, ou exibe uma mensagem para opção inválida).

    - Uso: Navegar entre diferentes funcionalidades de manutenção do sistema do porto.

- public void menuUtilizacao ()

    - Propósito Geral: Exibir o menu de utilização interativo para o sistema do porto.
    - Parâmetro: Nenhum.
    - Retorno: void, sem retorno.
    - Algoritmo (continuação do menu anterior):

        1. Exibe opções do menu.
        2. Lê a escolha do utilizador.
        3. Executa a ação correspondente à opção escolhida (ordenar/terminar missão, detetar embarcações, ativar/desativar radar, alocar/desalocar marinheiro, exibir marinheiros, total missões, exportar embarcações, recuperar/gravar estado, ou sair).

    - Uso: Navegar entre diferentes funcionalidades de utilização do sistema do porto.

- public void menuGerirEmbarcacoes (Scanner scanner)

    - Propósito Geral: Exibir o menu de gerenciamento de embarcações
    - Parâmetro: Scanner scanner, ler entradas;
    - Retorno: void, sem retorno.
    - Algoritmo (continuação do menu anterior):

        1. Exibe opções do menu.
        2. Lê a escolha do utilizador.
        3. Executa a ação correspondente à opção escolhida (adicionar, editar ou remover);

    - Uso: Navegar entre diferentes funcionalidades de gerir embarcações do sistema do porto.

Resultados Obtidos:

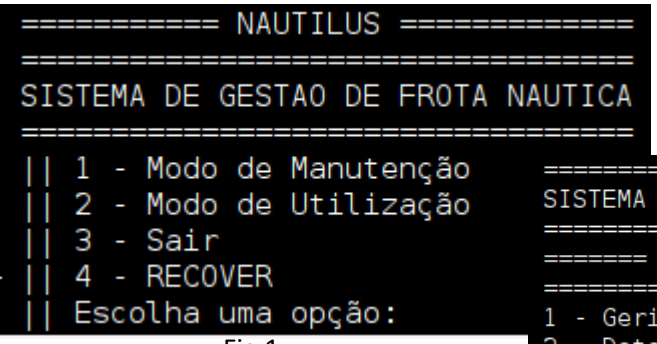

Fig.1

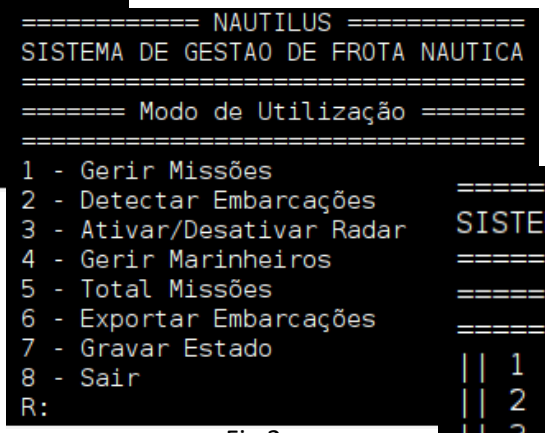

Fig.2

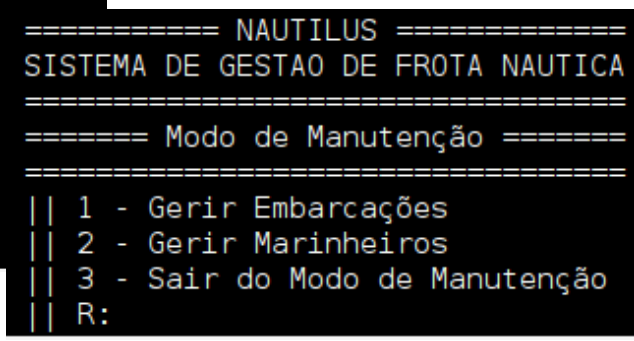

Fig.3

➔ Aqui podemos observar os resultados de cada menu que foi implementado no respetivo "main" (projeto.java):

- No menu principal, o utilizador seleciona um dos menus secundários, que pretende utilizar.

- Caso o utilizador, digite 1, ele irá ser redirecionado para o menu de manutenção onde, somente se encontram especificações de marinheiros e de embarcações.

- Caso o utilizador, digite 2, ele irá ser redirecionado para o menu de Utilização onde, pode ordenar / terminar uma missão, ativar / desativar radar, desalocar marinheiros, exibir marinheiros, exportar embarcações, entre outras tantas funções.

- Caso o utilizador, digite 3, ele acaba por sair do programa, assim encerrando-o.

- Caso o utilizador, digite 4, o programa vai fazer "RECOVER" através de um ficheiro indicado pelo utilizador.

Discussões/ Conclusão:

➔ Iremos abordar vários pontos fulcrais no desenvolvimento deste trabalho:
- O que mais nos custou executar / realizar
- O que mais gostamos de desenvolver
- Se os resultados obtidos, foram os que desejávamos / os que previámos

1.

Na nossa opinião o mais complicado de desenvolver foi a Classe "Porto", pois esta tinha imensas condições que iriam ter de ser verificadas/ alteradas ao longo do nosso projeto. Para além de ser bastante extensa, tem muitas funções importantes para a lógica do nosso código.

2.

Sem dúvida que o que gostamos mais de realizar foi a parte do main e das verificações respetivas às embarcações e aos marinheiros, pois foi com essa parte que o nosso projeto ganhou "vida", e assim desenvolvemos uma ideia mais estruturada daquilo que tínhamos planeado inicialmente, através do documento onde escrevemos o nosso plano de projeto.

3.

Os resultados que obtivemos foram bastante gratificantes pois aplicamos muito tempo de estudo e de trabalho a este projeto, pensando em tudo ao pormenor. Conseguimos desenvolver tudo o que o professor nos pediu de caracter obrigatório e ainda tivemos criatividade para desenvolver mais funções de caracter opcional.

Em suma, foi um trabalho muito bem conseguido por parte de todos, pois todos colaborámos para conseguirmos alcançar um excelente resultado, aplicando todo o conhecimento que fora adquirido.