# PROJETO PRÁTICO – PROGRAMAÇÃO ORIENTADA A OBJETOS

**Objetivos:**

- Elaborar um programa que cumpra os pré-requisitos enumerados no enunciado.
- Comentários no código-fonte são encorajados e alvo de avaliação.
- Otimização é alvo de avaliação.
- **É obrigatório o programa executar** (sem erros de compilação).
- Deve ser usada linguagem Java para a resolução do presente projeto prático.
- Deve ser entregue na tarefa do Microsoft Teams. Pasta do Projeto comprimida (.zip) com o nome TP_POO_(nome_formando)

## EXERCÍCIOS

Após ter terminado o curso de **Software Developer**, foi contactado por um estúdio de videojogos para desenvolver o novo RPG (Role-Playing Game). Este jogo trata-se de uma Personagem que deve percorrer um caminho e fazer escolhas de modo a concluir um labirinto cheio de obstáculos e inimigos. Para isso:

- Crie uma classe abstrata "**Entidade**" com os atributos:
    - nome (String)
    - vida / hp (int)
    - força (int).
  - Crie o método mostrarDetalhes que escreva na consola todos os detalhes da entidade.

- Crie as suas subclasses **Heroi** (também será abstrata) e **NPC** (Non-playable character).
- A classe **Heroi** deverá ter atributos:
    - nível (int)
    - ouro (int)
    - armaPrincipal (ArmaPrincipal)
    - ArrayList<Consumivel> inventario
  - Desenvolva o método abstrato atacar: recebe um **NPC** como argumento, esta classe deverá ter uma implementação diferente em cada uma das suas subclasses, no entanto, a finalidade é a mesma, confrontar o herói com um NPC numa luta até que um fique sem vida.

- A classe NPC deverá ter os atributos:
    - ArrayList<ItensHeroi> inventarioNPC
    - ouro (int)

- Desenvolva a classe abstrata **ItemHeroi** que tem como atributos:
    - nome (String)
    - preço em moedas de ouro (int)
    - ArrayList<String> heroisPermitidos que irá guardar o tipo de heróis que podem/sabem usar oito item.
  - Desenvolva também o método mostrarDetalhes.

- Desenvolva duas subclasses de **ItemHeroi**:
  - **ArmaPrincipal**
  - **Consumivel (abstrata)**

- A classe **Arma** deve ter os atributos:
    - Ataque (int)
    - AtaqueEspecial (int)

- A classe **Consumivel** deve ter as subclasses:
  - A classe **ConsumivelCombate** deve ter os atributos:
    - AtaqueInstantaneo (int)
  - A classe **Poção** deve ter os atributos:
    - vida a curar (int)
    - aumento de força (int)
  - Deve implementar o método mostrarDetalhes que imprime na consola as especificações de cada item.

- Crie a classe **Vendedor** que terá atributos:
    - ArrayList<ItemHeroi> loja, que representa os itens que o Herói poderá comprar durante o seu jogo. Sejam Armas Principais, Consumíveis de Combate ou Poções.
  - Deve conter o método imprimirLoja que imprime aleatoriamente 10 itens em stock, assim como as suas especificações. Mesmo que o vendedor tenha um inventário maior, apenas 10 devem ser mostrados de forma aleatória.
  - Deve conter o método vender( ) que recebe o Herói como parâmetro, e verifica se a compra pode ser efetuada, caso tal compra seja possível, deve acrescentar ao inventário do herói o item (se for armaPrincipal será uma substituição) e decrementar o seu ouro.

- A classe **Heroi** deverá ter, no mínimo, três subclasses indicando o tipo de herói com que o utilizador poderá jogar. Por exemplo: **Cavaleiro**, **Feiticeiro** e **Arqueiro**. (Poderá divergir de acordo com o contexto do seu jogo, no entanto, todos eles devem ter comportamentos semelhantes (mesmos métodos) aos demonstrados a seguir).
- Deve ter o método usarPocao( ) que imprime o inventário de poções do Herói e pergunta qual quer usar, seguidamente incrementa na vida/força do Herói os respetivos valores da poção.
- Cada uma destas subclasses deve implementar o método atacar de acordo com as seguintes regras (se escolheu divergir com o contexto, poderá também escolher métodos de ataques diferentes. Obrigatório serem diferentes para cada subclasse). Exemplo:
    - **Cavaleiro**: Recebe o NPC como argumento. Esta é a única classe onde o inimigo ataca primeiro. O inimigo ataca (é subtraído à vida do herói a força do inimigo, mas neste herói, devido à sua armadura, o inimigo só tem 80% da força inicial), seguidamente ataca o herói (é subtraído à vida do inimigo a força do herói somado com o ataque da sua arma).
    - **Feiticeiro**: Recebe o NPC como argumento. O Herói ataca primeiro (é subtraído à vida do inimigo a força do herói somado com o ataque da sua arma). Seguidamente o inimigo ataca (é subtraído à vida do herói a força do inimigo).
    - **Arqueiro**: Recebe o NPC como argumento. O Herói ataca primeiro (é subtraído à vida do inimigo a força do herói somado com o ataque da sua arma). Seguidamente o inimigo ataca (é subtraído à vida do herói a força do inimigo, devido à falta de proteção, este herói recebe 10% mais dano).

    - Este método ataque( ) funciona por turnos, ou seja, o turno começa com a primeira entidade a atacar e termina com a primeira entidade a ser atacada. Antes do Herói atacar, deve ser possível escolher na consola o tipo de ataque:
        - Ataque Normal: Subtrai na vida do inimigo a força do herói somado com o ataque da sua Arma Principal.
        - Ataque Especial: Subtrai na vida do inimigo a força do herói somado com o ataque especial da sua Arma Principal. (Só pode ser usado uma vez por luta)
        - Ataque Consumível: quando selecionado deve imprimir na consola a lista de consumíveis de combate do Heroi. Seguidamente pergunta qual dos consumíveis quer usar e, caso um válido seja selecionado, subtrai na vida do inimigo o ataque instantâneo. Assim sendo, deve remover do inventario do herói. Também deve ser possível cancelar este tipo de ataque e voltar ao menu de seleção.

    - O método atacar termina quando um dos envolventes na luta morrer, ou seja, a sua vida seja igual ou menos que 0. A partir daí deve devolver a entidade vencedora ou boolean.
    - Se o vencedor for o Herói: este sobe de nível, ganha 10 pontos de vida, 1 ponto de força e fica com o inventário do NPC (o ouro acrescenta automáticamente): para cada item encontrado deve perguntar ao Herói se quer aceitar (se for Arma Principal trocará a atual, se for consumível acrescenta ao ArrayList).

- Elabore uma Classe **Jogo** com um método **criarPersonagem** que permita criar uma personagem através de feedback da consola (pode criar métodos auxiliares nas respetivas classes).
    - Inicialmente deve perguntar qual o tipo de herói com que o utilizador deseja jogar (**Cavaleiro, Feiticeiro** ou **Arqueiro**).
    - Seguidamente é perguntada a dificuldade, podendo ser Fácil ou Difícil.
    - Depois o utilizador pode distribuir pontos de criação de personagem entre vida e força da sua personagem: sabendo que cada ponto de vida vale um ponto de criação de personagem e cada ponto de força vale 5 pontos de criação de personagem, assim o utilizador deve distribuir corretamente de modo a ficar com 0 pontos de criação da personagem.
    - Se a dificuldade for fácil tem direito a 300 pontos, se for difícil só tem direito a 220 pontos. A seguir é também atribuído ouro à personagem, se a dificuldade for fácil tem direito a 20 moedas de ouro, se for difícil apenas a 15 moedas de ouro. (exemplo demonstrativo, pode alterar os valores e ainda acrescentar outros graus de dificuldade)
    - Quando distribuir corretamente os pontos, deve ser instanciado o Herói do tipo correto.

Na Classe Jogo crie o método (nome do jogo), como por exemplo, aventuraMágica( ), guerraDasGalaxias( ) ou labirintoMortal( ). Recebe o Herói criado como parâmetro. Este será o método de "jogo", ou seja, terá um ArrayList<Sala> que guarda todas as salas do labirinto/mapa.

Esta Classe Sala deve ter os seguintes atributos:

- Nome (String)
- Id (int)
- ArrayList<int> idSalaConectada que representa todos os ids das salas que podem ser acedidas através desta mesma sala. (não precisa de ser bidirecional, ou seja, eu posso aceder a sala 2 a partir da sala 1, mas não o contrário).
- ArrayList<NPC> inimigosNaSala
- Vendedor vendedorPresente
- ArrayList<ItemHeroi> itemPerdido
- int danoArmadilha;
- int ouroPerdido.

Note que a sala só pode ter uma ação, ou seja, só deve ter inimigos ou vendedor ou itemPerdido ou Armadilha ou ouroPerdido.
Quando o Herói entra na sala, deve invocar o método boasVindas( ) que imprime na consola a mensagem de entrada.

Quando o Herói está prestes a sair da sala, deve invocar o método saída( ) que recebe como parâmetro a próxima sala escolhida pelo utilizador e imprime na consola uma mensagem personalizada. Por exemplo: está no rioSelvagem e pretende ir para montanhaAlta, imprime: "Depois de se abastecer com água e recuperar o fôlego, o (nome do herói) começa a sua subida para a Montanha Alta". Mas se o herói estiver no rioSelvagem e preferir continuar para o pantanoNegro, deve imprimir: "Depois de se abastecer com água e recuperar o fôlego, o (nome do herói) continua a sua jornada pela beira do rio até chegar ao Pantano Negro".
Com estas mensagens iremos criar a narrativa.

- Assim, voltando ao método (nome do jogo): Instancie, no mínimo, 14 itens, os quais serão adicionados a uma instância de Vendedor. (Note que cada item pode ser usado por uma ou mais categorias de Heróis, deve ter, no mínimo, uma poção de vida que cura 25 pontos de vida que seja possível usar por todas as categorias).

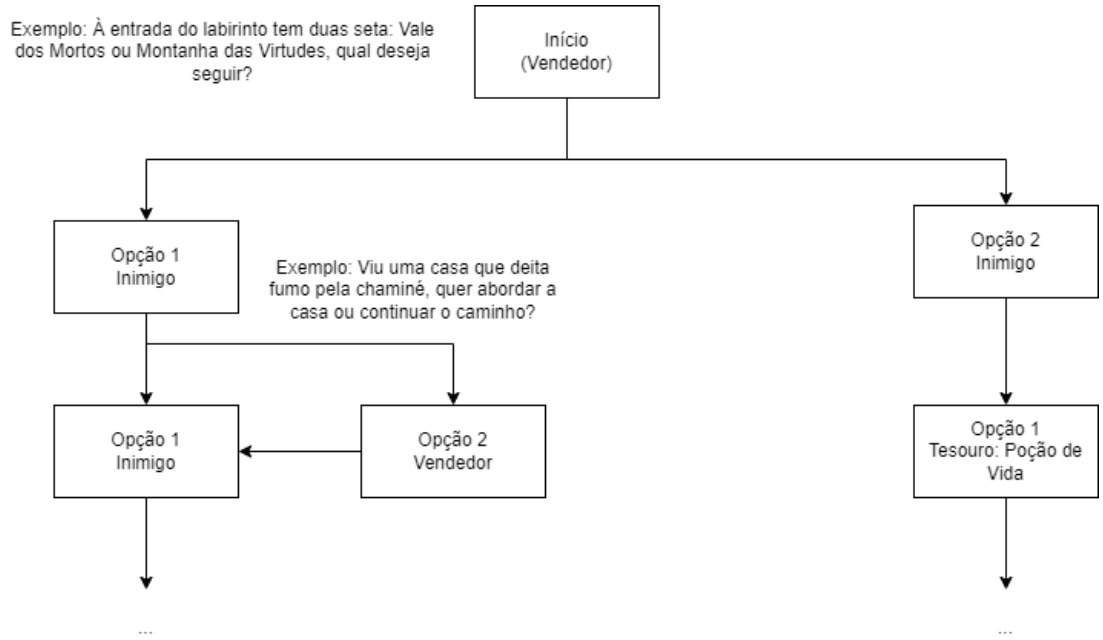
- Deve instanciar, sem conhecimento do utilizador, os Inimigos.

- Devem também ser instanciadas e adicionadas ao Array as salas. Verifique que todas as salas sejam de possível acesso. E adicione os respetivos inimigos a cada sala.

- A aventura começa com uma explicação do porque do herói ter de enfrentar o jogo (Por exemplo: Labirinto dos Perigos ou Galáxia do Vazio, etc…).

- Seguidamente no vendedor que se encontra no início da jornada, este terá itens à venda, a primeira ação é imprimir todos os itens na consola. Seguidamente o utilizador pode escolher qual quer comprar. Deve ser da classe de Herói certa e ter o ouro suficiente para fazer. Pode comprar os que quiser, ou simplesmente avançar sem itens para poupar ouro.

- O labirinto pode ter o seguinte aspeto, ficando à sua escolha a organização do mesmo: (Dica: Desenhar o labirinto e as ligações de todas as salas)




- Deve ter, pelo menos 16 salas e, no mínimo, 6 situações de escolhas diferentes. Pode ser criativo uma vez que está a criar uma narrativa.
- Ao fim de cada sala, o Herói pode usar, no máximo 3 das poções ou avançar. Dessa forma, sempre que o herói terminar uma sala, deve imprimir na consola todo o seu inventário de Poções e perguntar qual deseja usar (pode usar no máximo 3 de uma só vez). Atenção: Se a vida máxima atual do Herói for 100hp e ele tiver 90hp, se escolher uma poção que cure 25hp, só deve curar até o máximo de 100hp. (quando acontecer este fenómeno de excesso deve avisar o jogador da quantidade de excesso e apresentar pergunta de confirmação)
- Pode criar salas sem inimigos ou vendedores, podendo ter itensPerdidos, ouro ou armadilhas.
- Pode ter a sala de vendedor onde o vendedor é o mesmo do início e o utilizador pode comprar itens, para os quais não tinha ouro inicialmente.
- Cada sala que tenha um inimigo ou mais, deve invocar o método atacar pela ordem do ArrayList, seguidamente verificar qual o vencedor. Se for o utilizador pode continuar a jogar, caso seja o inimigo o jogo deve terminar. Quando o utilizador perde, no main deve apresentar a mensagem de derrota e perguntar de deseja **jogar novamente com a mesma personagem, criar uma nova, ou fechar o programa**.
- Ao fim de cada sala, devem aparecer as escolhas possíveis que o utilizador tem para avançar no jogo, aparecendo o nome das respetivas salas ligadas à sala atual.

- Todas as classes devem estar organizadas em devidos packages:
    a. Pacote Entidades: Entidade, Hero, NPC, (Cavaleiro, Feiticeiro, Arqueiro), Vendedor.
    b. Pacote Itens: ItemHeroi, Arma, Consumivel, ConsumivelCombate, Pocao.
    c. Pacote Jogo: Jogo
- Pode implementar métodos ao seu critério, para além dos acima mencionados.
- O jogo deve ter, pelo menos, uma maneira possível de ganhar.
- Deve usar a biblioteca Random pelo menos em dois contextos/mecânicas diferentes. (Por exemplo: sala que tem um totem amaldiçoado e tem 50% chance de dar 20 ouro ou de matar o jogador, podendo o jogador usar o totem as vezes que quiser)(Outro Exemplo: sala com item escondido que só tem 10% de hipótese de ser detetado)(Outro Exemplo: sala com armadilha que tira pontos de vida aleatórios ao jogador, entre 1 e 30)(Outro Exemplo: Sala que tem 50% hipótese de ter 1 inimigo, 30% de hipótese de ter 2 inimigos e 20% de hipótese para ter 3 inimigos)
- Pode criar métodos/classes adicionais, incluindo métodos específicos para salas específicas.
- Deve gerar o **JAVADOC**.

**Bom trabalho!** ☺