Universidade Estadual de Campinas
Instituto de Computação

Tarefa 2

RPG - Jogo Narrativo

MC322 - Programação Orientada a Objetos

Professor: Marcelo da Silva Reis
PEDs: Athyrson Machado Ribeiro / Giorgio de Moraes Rossa
PADs: Gustavo Miller / Pedro Lucas / Vinicius Leme

---

1. Descrição Geral

Seguindo a construção do nosso RPG, nesta tarefa iremos acrescentar mais três partes: a definição de mundo virtual no formato de diferentes fases, a progressão de dificuldade e nível do herói e armas a serem utilizadas pelos heróis e pelos monstros.

As diferentes fases utilizadas vão servir para ambientação inicial de forma simples e definição de inimigos presentes. Cada fase precisa ter uma dificuldade definida, de forma que o herói tenha cenários mais desafiadores a cada etapa, e que sejam igualmente mais recompensadores. Ao derrotar monstros, nosso herói ganhará experiência e poderá aumentar de nível, aumentando suas capacidades. Nosso sistema de itens começa a ser montado, utilizando atualmente armas para a batalha!

2. Objetivos

Os principais objetivos deste laboratório são:

- Estender os conhecimentos fundamentais de Programação Orientada a Objetos, como classes, atributos, métodos, heranças, classes abstratas e encapsulamento;
- Aplicação dos conceitos vistos nas últimas aulas, como coleção de dados (ArrayList) e geração de número aleatórios;
- Utilização de métodos privados e métodos estáticos de acordo com sua necessidade;
- Utilização de aleatoriedade para definir diferentes comportamentos do sistema;

3. Classes

Esta seção descreve a implementação principal do laboratório. As classes a seguir representam a estrutura mínima esperada, mas sinta-se encorajado(a) a expandi-la. As classes utilizadas na Tarefa anterior serão expandidas e utilizadas novamente. As diferenças nessas classes serão demonstradas em negrito (mudanças de comportamento, novos atributos ou novos métodos).

As seguintes diretrizes gerais devem ser observadas:

- **Liberdade Criativa:** Fique à vontade para criar classes auxiliares, atributos e métodos extras que julgar necessários para enriquecer seu projeto. O tema do RPG também é de sua livre escolha (ex: fantasia medieval, Pokémon, dinossauros, ficção científica, etc.).
- **Autonomia Técnica:** Este enunciado é intencionalmente vago em alguns aspectos, pois parte da atividade é a tomada de decisão de design. É esperado que você:
  - Identifique os tipos de variáveis mais adequados para cada atributo (ex: `int`, `String`, `double`).
  - Determine quais parâmetros são os mais adequados para cada método, com base na sua função.

3.1. Novas Classes

3.1.1. Classe `Fase`

Esta classe representará uma fase específica do jogo. A fase deve possuir um cenário e uma lista de monstros, que devem ter sua dificuldade aumentada (pontos de vida e força) de acordo com o nível da fase.

- **Atributos:**
  - `nivel` - Nível da fase;
  - `ambiente` - Nome ou breve descrição do cenário da fase;
  - `monstros` - Uma lista de monstros a serem derrotados.
- **Métodos:**
  - Construtor para inicializar os atributos;

3.1.2. Classe `ConstrutorDeCenário`

Esta classe será utilizada para guardar os métodos estáticos necessários para a execução de partes do nosso RPG relacionadas à definição dos cenários onde a história se passa.

- **Métodos:**
  - `gerarFases(int nFases)` **(Método Estático)**- Cria "n"fases de acordo com o parâmetro do método. Retorna uma lista de Fases com dificuldade crescente (nível da fase).

3.1.3. Classe Abstrata `Arma`

Esta classe representa uma Arma que pode ser utilizada por um Personagem[1].

- **Atributos:**
  - `dano` - Quanto de dano extra esta arma da ao Personagem.
  - `minNivel` - Nível mínimo que o Herói precisa ter para utilizar a Arma (note que monstros não possuem nível definido, podendo usar qualquer arma).

3.1.4. Classe Concreta de Arma

Um tipo de arma, herda de `Arma`.

- Implemente pelo menos três classes concretas de Arma Ex: `Espada, Machado, Arco etc.`.

3.2. Classes já definidas e suas mudanças

3.2.1. Classe Abstrata `Personagem`

Esta classe representará a base para qualquer entidade viva no mundo do jogo. Sendo abstrata, não pode ser instanciada diretamente.

- **Atributos:**
  - `nome` - Nome do personagem;
  - `pontosDeVida` - A saúde atual do personagem;
  - `forca` - O atributo de força, usado em combate.
  - `arma` **- Atributo do tipo Arma, usado em combate.**
- **Métodos:**
  - Construtor para inicializar os atributos;
  - `receberDano()` - Reduz os pontos de vida com base no dano recebido;
  - `exibirStatus()` - Imprime as informações do personagem (nome, vida, etc.).
  - `atacar(Personagem alvo)` **(Método Abstrato)** - Um contrato que obriga todas as classes filhas a implementar sua própria lógica de ataque.

[1] Caso o cenário imaginado pelo grupo na Tarefa 1 não combine com a descrição de armas, utilizem outro equipamento de auxilio de ataque relativo ao universo escolhido (por exemplo, se o cenário anterior era uma batalha com balões cheios de água, cada cor de balão pode mudar o quanto o alvo se molha).

3.2.2. Classe Abstrata `Heroi`

Esta classe herda de `Personagem` e serve como base para todas as classes jogáveis.

- **Atributos Adicionais:**
  - `nivel` - O nível atual do herói;
  - `experiencia` - Pontos de experiência acumulados.
  - `expProximoNivel` - Pontos de experiência necessários para aumentar para o próximo nível.
  - `sorte` - Valor entre 0 e 1 que define a atual sorte do Herói para algumas ações do jogo.
- **Métodos Adicionais:**
  - Construtor que utiliza o `super()` para inicializar os atributos da classe `Personagem`;
  - `ganharExperiencia()` - Adiciona pontos de experiência ao héroi;
  - `exibirStatus()` - Sobrescreve (*override*) o método da classe pai para incluir o nível e a experiência.
  - `usarHabilidadeEspecial(Personagem alvo)` **(Método Abstrato)** - Contrato que obriga as classes concretas de heróis a terem uma habilidade única. **A habilidade especial deve, necessariamente, ter um efeito relacionado à sorte do Herói. Tal comportamento pode ser uma chance de erro ou de acerto crítico. Usem criatividade!**;
  - `subirDeNível()` **Método privado - Método para aumentar o nível atual do herói. Deve ser chamado apenas de forma interna na classe (privado). Subir de nível deve atualizar a experiência necessária para o próximo nível, assim como fortalecer os atributos atuais do herói de alguma forma**;
  - `equiparArma(Arma novaArma)` - **Método para trocar a arma do Herói, respeitando o nível mínimo requirido pela Arma.**;

3.2.3. Classes Concretas de Heróis

Estas são as classes jogáveis que herdam de `Heroi`.

- Implemente duas classes concretas de héroi.
- Adicione um atributo único para cada uma (ex: `furia` para um Guerreiro, `precisao` para um Arqueiro).
- Forneça a implementação concreta para os métodos abstratos `atacar()` e `usarHabilidadeEspecial()`. Nesse lab a implementação pode ser um simples `System.out.println` descrevendo a ação.

3.2.4. Classe Abstrata `Monstro`

Esta classe herda de `Personagem` e serve como base para todos os inimigos do jogo.

- **Atributos Adicionais:**
  - `xpConcedido` - Experiência que o monstro concede ao ser derrotado.
  - `listaDeArmasParaLargar` **- Lista de possíveis Armas a serem largadas pelo monstro ao ser derrotado.**
- **Métodos Adicionais:**
  - Construtor que utiliza o `super()` para inicializar os atributos da classe `Personagem`;
  - `exibirStatus()` - Sobrescreve o método para incluir o XP que será concedido;
  - `largaArma()` **- Retorna uma arma aleatória dentro da lista de armas listaDeArmasParaLargar.**

3.2.5. Classe Concreta de Monstro

Um tipo de inimigo que herda de `Monstro`.

- Implemente pelo menos duas classes concretas de Monstro Ex: `Goblin, Esqueleto, Fantasma etc.` com atributos e comportamentos únicos entre si.
- Forneça a implementação concreta para o método abstrato `atacar()`.

3.3. Classe `Main` (Cenário de Sobrevivência)

A classe `Main` simulará um desafio de sobrevivência. O objetivo é criar um cenário onde um único herói deve enfrentar e sobreviver a diferentes Fases.

- **Criação das Fases:**
  - Utilize o método estático da classe `ConstrutorDeCenário` para gerar pelo menos 3 fases. Tal método deve criar os Monstros dentro de cada fase, como descrito anteriormente.
- **Criação dos Personagens:**
  - Crie uma instância de um herói de sua escolha (ex: um `Guerreiro`).
- **Apresentação do Desafio:**
  - Imprima uma mensagem inicial apresentando o desafio dependendo do ambiente da fase, como O HERÓI ENTRA NA(O)[Nome do ambiente da Fase] PARA ENFRENTAR [Numéro de monstros da Fase]! – Cada ambiente define uma mensagem de apresentação diferente, baseado nos cenários que vocês imaginaram.
  - Exiba o status inicial do herói de uma forma elegante (usando o método exibirStatus()).
- **Simulação da aventura em Loops:**
  - Crie um laço de repetição (*loop*) que execute cada uma das Fases criadas. Como nosso herói é perspicaz, ao entrar em uma Fase ele prontamente se posiciona de forma a enfrentar apenas um monstro por vez;
  - Anuncie os detalhes da fase e do herói a cada início de fase;
  - Outro laço de repetição deve ser criado, executando um combate para cada Monstro da Fase até o herói ter derrotado todos os monstros da fase ou ter sido derrotado;
  - Dentro do laço de combate:
    - Anuncie a chegada do monstro.
    - Enquanto tanto o herói quanto o monstro estiverem vivos:
      - O herói ataca o monstro.
      - Verifiquem se o monstro sobreviveu.
      - O monstro ataca o herói de volta.
    - Verifique se o herói sobreviveu. Se os pontos de vida do herói chegarem a zero ou menos, os laços devem ser interrompidos e uma mensagem de "Game Over"deve ser exibida.
    - Caso o monstro tenha sido derrotado, testa a sorte do herói (por exemplo, testando se um valor aleatório é maior ou menor que a sorte definida). Utilize o método `largaArma` do monstro de acordo com a sorte do herói.
    - A decisão de equipar ou não a arma pode ser definida conforme o caso de vocês.
    - Implementar uma forma de decidir se o herói vai trocar a arma atual dele pela arma derrubada pelo monstro.
- **Conclusão do Desafio:**
  - Se o herói sobreviver a todas as fases, imprima uma mensagem de vitória no final.

4. Avaliação

Atenção para os seguintes pontos que serão considerados na avaliação:

- **Diferenciação e Criatividade:** A criatividade será observada na implementação das classes concretas e cenários imaginados. Armas e monstros criados, uso da sorte em cada habilidade única e descrição das ações são pontos de atenção.

```
/tarefa1/
     |-- src/
        |-- ...
/tarefa2/
     |-- src/
        |-- classe1.java
```

```
        |-- classe2.java
        |-- classe3.java
        \__ main.java

```

- **Clareza do Código:** Use comentários para descrever o propósito de cada classe e para explicar trechos de lógica que não sejam triviais.
- **Escopo do Laboratório:** Não utilize entrada de dados pelo teclado neste laboratório; todos os dados devem ser pré-definidos no código.
- **Apresentação do Resultado:** A clareza e a organização da saída de texto no console são fundamentais. Use cabeçalhos (ex: — INÍCIO DO TURNO —), espaçamento e mensagens descritivas para que a simulação seja fácil de ler e acompanhar.
- **Funcionalidade e Execução:** O projeto entregue deve compilar e executar sem erros no ambiente de correção especificado (Java 21). É responsabilidade do aluno garantir a compatibilidade e testar seu código antes da submissão. Projetos que não executarem devido a erros de código ou incompatibilidade de versão serão severamente penalizados.

5. Entrega

A entrega do laboratório deve ser feita via repositório no GitHub.

- **A entrega do Laboratório é realizada exclusivamente via Github**[2] Para a submissão no Github, gere um release (tag) com a identificação do laboratório.
- Gere um release no Github com a tag no formato: **tarefaX-RA1-RA2**.
- Se vocês optarem por deixar seus repositórios privados adicionem os PEDs da disciplina como colaboradores do projeto: **Athyrson - @athyrson06** e **Giorgio - @giorgioDeMoraesRossa**.
- O prazo de submissão da tarefa é até o dia **11/09/2025** às **23:59**.
- Entregas atrasadas serão penalizadas em **25 %** da nota recebida.
- Utilize os horários de laboratório e atendimentos para tirar eventuais dúvidas de submissão e também relacionadas ao desenvolvimento do laboratório.

6. Dicas

Durante a avaliação os PEDs utilizarão os seguintes comandos para compilar e executar seus projetos:

```
cd tarefa2
javac -d bin $(find src -name "*.java")
java -cp bin Main
```

Certifique-se de que a classe `main` (ou equivalente, caso utilize pacotes) contenha o método `public static void main(String[] args)` e esteja organizada de acordo com a estrutura de diretórios definida. Dessa forma, a compilação e a execução ocorrerão corretamente no ambiente de avaliação.

[2] Você deve criar um link e enviar no Google Classroom