Universidade Estadual de Campinas
Instituto de Computação

Laboratório 4

Simulador de Robôs

MC322 - Programação Orientada a Objetos

Professora: Esther Colombini
PEDs: Angelica Cunha dos Santo / Athyrson Machado Ribeiro /
Wladimir Arturo Garces Carrillo

---

1. Descrição Geral

Você fará parte da equipe de engenharia de um laboratório de robótica onde diversos robôs inteligentes precisam interagir, mover-se e realizar tarefas cooperativas. Cada robô possui comportamentos distintos (como sensores, mobilidade ou comunicação) e devem seguir contratos de funcionalidade definidos via interfaces. Além disso, você deverá garantir que os robôs sejam utilizados corretamente, evitando erros em tempo de execução com tratamento de exceções.

Neste Laboratório 4, vamos continuar fazendo nosso simulador. Vamos fixar os conceitos vistos até então e apresentar novos conceitos como:

1. **Reforçar o uso de enumerações**
2. **Interfaces e polimorfismo**
3. **Exceções**
4. **Herança múltipla**

2. Objetivos

Os objetivos principais do Laboratório 4 são os seguintes:

- Consolidação dos conteúdos vistos nos labs anteriores;
- Praticar interfaces e polimorfismo.
- Introduzir o uso de exceções personalizadas para controle de fluxo.
- Reforçar o uso de enumerações.
- Aplicar alocação dinâmica com coleções e entrada do usuário.
- Explorar herança múltipla via interfaces.

3. Descrição das classes

Neste Laboratório 4, utilizaremos a mesma base de classes desenvolvidas nos laboratórios anteriores. Contudo, será necessário aprimorar e expandir essas classes para atingir os novos objetivos, especialmente o uso de interfaces, polimorfismo e exceções personalizadas.

3.1. Enumeração `TipoEntidade`

A enumeração `TipoEntidade` representa os diferentes tipos de entidades que podem existir no ambiente. Ela é utilizada para fins de identificação, controle de fluxo e visualização.

- **Valores possíveis:**
  - `VAZIO`
  - `ROBO`
  - `OBSTACULO`

- `DESCONHECIDO`

- **Exemplo de uso:**

```
Entidade e = mapa[x][y][z];
if  (e.getTipo() == TipoEntidade.ROBO) {
    // Executa ação específica
}
```

ou

```
private void inicializarMapa() {
    for (int x = 0; x < largura; x++) {
        for (int y = 0; y < profundidade; y++) {
            for (int z = 0; z < altura; z++) {
                mapa[x][y][z] = TipoEntidade.VAZIO;
            }
        }
    }
}
```

- **Nota:** Você pode adicionar outros tipos conforme a necessidade da sua implementação. Lembre-se de refatorar os exemplos conforme sua necessidade e implementação.

3.2. Interface `Entidade`

Todas as entidades colocadas no ambiente devem implementar esta interface. Ela define um contrato mínimo de comportamento.

- **Métodos esperados:**
  - `int getX();` → Retorna a coordenada X da entidade.
  - `int getY();` → Retorna a coordenada Y da entidade.
  - `int getZ();` → Retorna a coordenada Z da entidade.
  - `TipoEntidade getTipo();` → Retorna o tipo da entidade.
  - `String getDescricao();` → Retorna uma descrição textual da entidade.
  - `char getRepresentacao();` → Retorna o caractere que representa a entidade visualmente.

3.3. Classe `Ambiente`

A classe `Ambiente` representa o espaço tridimensional onde os robôs operam, o plano de coordenadas do `Ambiente` pode ser visto na **Figura 1**. Agora, além de armazenar as entidades dinamicamente, o ambiente também mantém uma representação explícita do espaço: um mapa tridimensional de ocupação.

- **Atributos:**
  - `largura`, `profundidade`, `altura` (int) → Dimensões do ambiente nos eixos X, Y e Z.
  - `entidades` (ArrayList<Entidade>) → Lista polimórfica com todos os elementos do ambiente.
  - `mapa` (TipoEntidade[][][]) → Mapa tridimensional que indica a ocupação do ambiente. [**NOVO**]
- **Métodos principais:**
  - `inicializarMapa()`
  - `adicionarEntidade(Entidade e)`
  - `removerEntidade(Entidade e)`
  - `dentroDosLimites(int x, int y, int z)` → Pode lançar `ColisaoException`.
  - `estaOcupado(int x, int y, int z)` [**NOVO**]

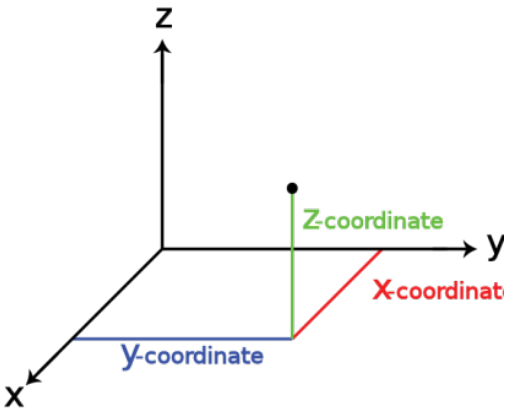

Figura 1: Representação do plano de coordenadas do `Ambiente`

- `moverEntidade(Entidade e, int novoX, novoY, novoZ)`
- `executarSensores()`
- `verificarColisoes()` → Pode lançar `ColisaoException`.
- `visualizarAmbiente()` → Imprime em consola uma vista no plano X,Y do ambiente. **[NOVO]**

- **Nota:** Os elementos do `Ambiente` agora são tratados como `Entidades`. Adapte suas classes conforme esse padrão.

3.4. Classe `Robo`

Classe base para todos os robôs do simulador. Uma nova interpretação dessa classe deve implementar a interface `Entidade` e conter atributos comuns, como posição, tipo e estado. Além disso tera que incluir um método para ações especificas, por exemplo, executar um comportamento para chegar num outro punto.

- **Atributos:**
  - `id` (String) → Identificador único.
  - `estado` (EstadoRobo) → Estado atual do robô.
  - `tipoEntidade` (TipoEntidade)
  - `x, y, z` (int) → Coordenadas atuais no ambiente.
- **Métodos principais:**
  - `moverPara(int x, int y, int z)` → Move o robô.
  - `ligar(), desligar()` → Altera o estado do robô. Pode usar enumerações para os estados.
  - `executarTarefa()` → Método abstrato para ações específicas.

3.5. Classe `CentralComunicacao`

Classe responsável por registrar e disponibilizar o histórico de mensagens trocadas entre robôs que implementam `Comunicavel`.

- **Atributos:**
  - `mensagens` (ArrayList<String>) → Lista de mensagens trocadas.
- **Métodos:**
  - `registrarMensagem(String remetente, String msg)` → Armazena uma nova mensagem.
  - `exibirMensagens()` → Exibe todas as mensagens.

3.6. Interfaces `Sensoreavel` e `Comunicavel`

Estas interfaces definem comportamentos opcionais que podem ser adotados por certos robôs.

- `Sensoreavel`:
  - Método: `acionarSensores()` → Pode lançar `RoboDesligadoException`.
- `Comunicavel`:
  - Métodos:
    - `enviarMensagem(Comunicavel destinatario, String mensagem)` → Pode lançar `RoboDesligadoException`.
    - `receberMensagem(String mensagem)` → Pode lançar `RoboDesligadoException`.

3.7. Exceções Personalizadas

- `RoboDesligadoException`: lançada quando um robô tenta executar uma ação sem estar ligado.
- `ColisaoException`: lançada quando há tentativa de ocupação de uma posição já ocupada no mapa.
- `ErroComunicacaoException`: lançada quando há um erro na comunicação (ex. o receptor não tem capacidade de comunicação).

3.8. Relações entre elementos

- **Composição:** A classe `Ambiente` contém uma coleção de objetos do tipo `Entidade`, bem como um mapa tridimensional de ocupação.
- **Herança:** A classe `Robo` pode ser estendida por tipos específicos de robôs, que herdam seus atributos e comportamentos básicos, podendo sobrescrever ou adicionar funcionalidades via polimorfismo.
- **Implementação de interfaces:**
  - `Robo` implementa `Entidade`.
  - Robôs específicos podem também implementar `Sensoreavel` e/ou `Comunicavel`, adicionando funcionalidades especializadas.
- **Polimorfismo:** A lista de entidades no ambiente (`ArrayList<Entidade>`) pode armazenar qualquer objeto que implemente a interface `Entidade`, permitindo o uso de diferentes tipos de robôs e objetos de forma genérica.
- **Tratamento de erros:**
  Através de exceções personalizadas (`RoboDesligadoException`, `ColisaoException`, etc), o sistema pode lidar de forma estruturada com situações excepcionais durante a execução das ações dos robôs.

4. Atividades

Este laboratório deve ser desenvolvido com base nas estruturas anteriores, integrando os novos conceitos apresentados neste enunciado. As atividades estão divididas em cinco partes principais:

1. Implementação das Classes
   - Adapte os robôs específicos que você já criou anteriormente para implementar as interfaces `Sensoreavel` e `Comunicavel`, quando aplicável.
   - Crie pelo menos **3 novas interfaces funcionais** além das propostas neste enunciado, com comportamentos distintos (ex: `Autonomo`, `Explorador`, `Atacante`).
   - Garanta que cada robô herde de uma `classe abstrata base` e implemente múltiplas interfaces conforme seu comportamento. **(Herança múltipla via interfaces)**
   - Implemente um **sistema de mapa tridimensional** utilizando uma matriz do tipo `TipoEntidade[][][]` para representar o estado do ambiente.
   - Crie **pelo menos 3 exceções personalizadas adicionais** além das sugeridas (`RoboDesligadoException`, `ColisaoException`). Exemplos: `ForaDosLimitesException`, `AcaoNaoPermitidaException`.
   - Defina **tarefas específicas** para cada tipo de robô no ambiente (ex: buscar um ponto, explorar uma região, monitorar obstáculos, enviar alerta, etc.).

2. **Funcionalidades do Menu Interativo**:
   O menu interativo, implementado via `Scanner`, deve conter as seguintes opções:

   - Listar todos os robôs por tipo ou estado (ligado/desligado).
   - Escolher um robô para interagir.
   - Visualizar o status de um robô selecionado e do ambiente.
   - Visualizar o mapa do ambiente no console (visão 2D de cima), com todos os elementos posicionados.
   - Listar e executar as funcionalidades do robô selecionado:
     - Comunicação (via `Comunicavel`).
     - Leitura de sensores (via `Sensoreavel`).
     - Outras funcionalidades que você criou.
   - Controlar a movimentação básica do robô:
     - Frente, trás, direita, esquerda, cima, baixo.
   - Ativar ou desligar robôs.
   - Listar mensagens trocadas entre os robôs (via `CentralComunicacao`).

3. **O que não deve ser incluído no menu interativo**:

   - Criação do ambiente.
   - Instanciação de robôs.
   - Métodos altamente complexos que exigem entrada textual extensa.
   - Uso exagerado de entrada de `String`.

4. **O que colocar na `Classe Main`**

   - Inicialize o ambiente e os robôs manualmente.
   - Instancie as entidades necessárias e adicione ao ambiente.
   - Implemente o menu interativo descrito acima.
   - Realize testes das funcionalidades criadas.

5. Atualização do `README.md`

   Inclua no arquivo `README.md` do repositório:

   - Descrição das principais mudanças realizadas neste laboratório.
   - Diagrama UML atualizado com as interfaces e heranças utilizadas.
   - Lista das interfaces criadas e onde foram implementadas.
   - Lista das exceções personalizadas implementadas e onde são lançadas.
   - Instruções para compilação e execução.

5. Entrega

Para a entrega do trabalho considere:

1. **A entrega é realizada exclusivamente via Github** (crie um link da release e submeta (como link) no Google Classroom).
2. Gere um release no Github com a tag no formato *"lab04-RA1-RA2"*.
3. **Prazo de Entrega:** 28/05/25 - 23h59.

6. Avaliação

Critérios de avaliação:

- Implementação correta das classes e seus relacionamentos;
- Execução do projeto sem erros ou warnings;
- Output do projeto, o que incluí formatação da saída de dados e menu de leitura de dados;

- Organização e documentação do código e projeto;
- Criatividade na criação de classes e métodos;
- Uso adequado da classe Main para testagem;
- Submissão dentro das normas.