Universidade Estadual de Campinas
Instituto de Computação

Laboratório 3

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

Um simulador de robôs é um ambiente computacional que permite modelar, testar e analisar o comportamento de robôs em diferentes cenários sem a necessidade de hardware físico. Esses simuladores são amplamente utilizados na pesquisa, na indústria e na educação para desenvolver e validar algoritmos de controle, navegação, inteligência artificial e interação com o ambiente antes da implementação real.

Neste Laboratório (03), vamos fixar os conceitos vistos até então e apresentar novos conceitos como:

1. **Relacionamentos entre Classes:** Tipos de relacionamentos (Herança, Associação, Agregação e Composição) assim como Cardinalidade dos relacionamentos;
2. **Operações entre classes**;
3. **Uso de Enum**;
4. **Leitura de dados do teclado**;

2. Objetivos

Os objetivos principais do Laboratório 3 são os seguintes:

- Consolidação dos conteúdos vistos nos labs anteriores;
- Capacidade de refatorar um projeto;
- Aplicação de diferentes relacionamentos entre classes;
- Capacidade de abstração de como aplicar os conceitos de orientação objetos em face das funcionalidades requeridas.

3. Descrição das classes

Nesse laboratório focaremos em aprimorar a classe Ambiente, implementar uma classe Obstaculo e uma classe Sensor. Essas classes devem devem interagir com as classes de robôs criadas no laboratório anterior.

3.1. Classe `Ambiente`

A classe `Ambiente` representa o espaço onde os robôs se movimentam e onde os obstáculos existem:

- **Atributos:**
  - `largura` (int) → Largura do ambiente.
  - `altura` (int) → Altura do ambiente.
  - `robos` (ArrayList<Robo>) → Lista de robôs no ambiente.
  - `obstaculos` (ArrayList<Obstaculo>) → Lista de obstáculos.
- **Métodos:**
  - `adicionarRobo(Robo r)` → Adiciona robô ao ambiente.

- `removerRobo(Robo r)` → Remove robô do ambiente.
- `dentroDosLimites(int x, int y, int altitude)` → Verifica se posição está dentro dos limites.
- `detectarColisoes()` → Verifica colisões entre robôs e obstáculos (Pode ser substituído por um sensor).

3.2. Classe `Obstaculo`

Nova classe para representar obstáculos no ambiente. Utilize um Enum para representar o tipo de obstáculo e padronizar as propriedades dos tipos de obstáculo.

- **Atributos:**
  - `posicaoX1`, `posicaoY1`, `altura` (`int`)
  - `posicaoX2`, `posicaoY2` (`int`)
  - `tipo` (`TipoObstaculo`) → Tipo enumerado que define o tipo de obstáculo
- **Enum `TipoObstaculo`:**
  - `PAREDE`, `ARVORE`, `PREDIO`, `BURACO`, `ETC`
- **Relação com Ambiente:** Composição (1 `Ambiente` tem muitos `Obstaculo`)

Enum `TipoObstaculo`

Exemplo da implementação da enumeração responsável por definir os tipos de obstáculo, junto com suas propriedades padrão (altura e bloqueio de passagem). Implemente sua própria versão!

```java
public enum TipoObstaculo {
    PAREDE(3, true),
    ARVORE(5, true),
    PREDIO(10, true),
    BURACO(0, true),
    OUTRO(-1, false); // Altura -1 representa valor variavel

    private final int alturaPadrao;
    private final boolean bloqueiaPassagem;

    TipoObstaculo(int alturaPadrao, boolean bloqueiaPassagem) {
        this.alturaPadrao = alturaPadrao;
        this.bloqueiaPassagem = bloqueiaPassagem;
    }

    public int getAlturaPadrao() {
        return alturaPadrao;
    }

    public boolean isBloqueiaPassagem() {
        return bloqueiaPassagem;
    }
}
```

Listing 1: Enum TipoObstaculo com construtor

3.3. Classe `Sensor`

O Robo poderá utilizar sensores para explorar melhor o Ambiente.

- **Atributos:**
  - `raio (de alcance máximo)` (double)
- **Métodos:**
  - `monitorar()` → Utiliza a função de monitoramento do sensor.

- **Relação com Robôs:** Composição (1 Robô tem um ou muitos Sensores)

Implemente pelo menos duas classes herdadas da classe Sensor. Cada classe deve representar um tipo de sensor disponível para os robôs. Por exemplo: sensor de altitude*, posição*, temperatura**, proximidade**, colisão*, umidade**, ...

- **Atenção:**
  - * Dificuldade menor, menos criativo
  - ** Dificuldade maior, mais criativo

3.4. Relações entre Classes

- **Herança:** RoboTerrestre e RoboAereo herdam de Robô
- **Composição:** Ambiente contém Robôs e Obstáculos
- **Agregação:** Robôs podem ter sensores (classe separada)
- **Dependência:** Sensores dependem de Robôs para existirem

4. Atividades

1. **Implementação das Classes:**
   - Implemente todas as classes conforme descrito, incluindo as relações entre elas.
   - Crie pelo menos 2 classes derivadas para Sensor.
   - Atualize a classe Robo para interagir com a classe Sensor.
   - Utilize variáveis finais para certos atributos das classes Ambiente e Obstaculo.
   - Crie um menu interativo, bonito e simples para interações com os robôs e ambientes. Mas instancie todas as classes manualmente.
2. **O que colocar no menu interativo:**
   - Visualizar status de robôs e ambiente;
   - Controlar movimentação básica;
   - Utilizar/relatar uso dos sensores.
   - Instruções para uso de cada entrada de dados;
   - Etc.
3. **O que não colocar no menu interativo:**
   - Instanciação das classes;
   - Uso de métodos altamente complexos;
   - Uso exagerado de leitura de Strings;
4. **O que colocar na Classe `Main`:**
   - Instancie todas as classes e faça testes manualmente no código (sem leitura de teclado) de métodos não abrangidos pelo teclado.
   - Implementação do menu interativo;
5. **Esboçe um diagrama de classes contendo as relações entre todas as suas classes implemetadas:**
   - Deve exibir atributos e métodos;
   - Deve exibir os tipos dos atributos;
   - Exiba os principais parâmetros dos métodos;
   - Descreva brevemente sua linha de raciocínio na construção do diagrama.
6. **Aprimore o Read.me do seu repositório:**
   - Adicione IDE usada e versão do Java usada;
   - Descreva brevemente o conteúdo do repositório;
   - Adicione uma seção descrevendo como executar cada laboratório;
   - Exiba o diagrama de classes criado e sua descrição.

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 *"lab03-RA1-RA2"*.
3. **Prazo de Entrega:** 25/04/25 - 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.