Notas Práticas e Exercícios 5

Disciplina: Programação Orientada a Objetos

{*Anotações para uso pessoal}

1 Exercício Execeções

Implementação do exercício descrito nos slides da aula de exceções.

```java
public class OperacaoInvalidaException extends Exception {
    public OperacaoInvalidaException(String message) {
        super(message);
    }
}

public class Calculator {
    public double calcular(double a, double b, char operacao) throws
        ArithmeticException,OperacaoInvalidaException {
        switch (operacao) {
            case '+':
                return a + b;
            case '-':
                return a - b;
            case '*':
                return a * b;
            case '/':
                if (b == 0) {
                    throw new ArithmeticException("Divisão por zero.");
                }
                return a / b;
            default:
                throw new OperacaoInvalidaException("Operação inválida: " +
                    operacao);
        }
    }
}

public class ExemploCalculadora {
    public static void main(String[] args) {
        double a = 10;
        double b = 5;
        char operacao = '#'; // Operação inválida

        Calculator calculator = new Calculator();
```

```
34
35          try {
36              double resultado = calculator.calcular(a, b, operacao);
37              System.out.println("Resultado: " + resultado);
38          } catch (OperacaoInvalidaException e) {
39              System.out.println("Erro: " + e.getMessage());
40          } catch (ArithmeticException e) {
41              System.out.println("Erro: " + e.getMessage());
42          }
43      }
44  }
```

2 Exercícios Práticos

1. Exercício: Interface e Implementação de Formas Geométricas

   Crie uma interface chamada `FormaGeometrica` com os seguintes métodos:

   - `double calcularArea();`
   - `double calcularPerimetro();`

   Implemente duas classes que representam formas geométricas, `Circulo` e `Retangulo`, que implementem a interface `FormaGeometrica`. Ambas as classes devem ter construtores que aceitem os parâmetros necessários para calcular suas respectivas áreas e perímetros.

2. Exercício: Classe Abstrata e Herança de Animais

   Crie uma classe abstrata chamada `Animal` com os seguintes atributos e métodos:

   - Atributo `String nome;`
   - Método `void emitirSom();`

   Derive duas classes a partir de `Animal`: `Cachorro` e `Gato`. Implemente o método `emitirSom()` em ambas as classes de forma que cada uma exiba o som característico do animal.

3. Exercício: Interface, Classe Abstrata e Sistema de Controle de Estoque

   Crie uma interface chamada Produto com os seguintes métodos:

   - `String getNome();`
   - `double getPreco();`
   - `int getQuantidade();`
   - `void setQuantidade(int quantidade);`
   - `String getId();`

   Crie uma classe abstrata chamada ProdutoAbstrato que implemente a interface Produto e adicione os seguintes atributos e métodos:

   - Atributo `String nome;`
   - Atributo `double preco;`
   - Atributo `int quantidade;`
   - Atributo `String id;`

- Método abstrato `double calcularDesconto();`

Implemente duas classes que herdem de ProdutoAbstrato: ProdutoPerecivel e ProdutoNaoPerecivel. Ambas as classes devem implementar o método calcularDesconto() de acordo com as regras abaixo:

- `ProdutoPerecivel`: Implemente um atributo `LocalDate dataValidade` e um método `boolean produtoVencido()`. O desconto deve ser calculado com base na proximidade da data de validade: 30
- `ProdutoNaoPerecivel`: Implemente um atributo `String categoria` e um método `boolean categoriaEspecial()`. O desconto deve ser de 5

Crie uma classe chamada ControleEstoque com os seguintes atributos e métodos:

- Atributo `List<Produto> produtos;`
- Método `void adicionarProduto(Produto produto);`
- Método `void removerProduto(String id);`
- Método `void listarProdutos();`
- Método `Produto buscarProduto(String id);`
- Método `void aplicarDesconto(String id);`

Os métodos devem interagir com a lista de produtos, permitindo adicionar, remover, listar, buscar e aplicar desconto nos produtos de acordo com suas regras.

4. Exercício: Interface, Classe Abstrata e Sistema de Reservas de Passagens Aéreas

   Crie uma interface chamada Passagem com os seguintes métodos:

   - `String getCodigo();`
   - `double getPreco();`
   - `LocalDateTime getDataHora();`
   - `String getOrigem();`
   - `String getDestino();`

   Crie uma classe abstrata chamada PassagemAbstrata que implemente a interface Passagem e adicione os seguintes atributos e métodos:

   - Atributo `String codigo;`
   - Atributo `double preco;`
   - Atributo `LocalDateTime dataHora;`
   - Atributo `String origem;`
   - Atributo `String destino;`
   - Método abstrato `double calcularPrecoFinal();`

   Implemente duas classes que herdem de PassagemAbstrata: PassagemEconomica e PassagemExecutiva. Ambas as classes devem implementar o método calcularPrecoFinal() de acordo com as regras abaixo:

   - `PassagemEconomica`: O preço final deve ser igual ao preço base.
   - `PassagemExecutiva`: O preço final deve ser o preço base multiplicado por 1.5.

Crie uma classe chamada SistemaReservas com os seguintes atributos e métodos:

- Atributo `List<Passagem> passagens;`
- Método `void adicionarPassagem(Passagem passagem);`
- Método `void removerPassagem(String codigo);`
- Método `void listarPassagens();`
- Método `Passagem buscarPassagem(String codigo);`
- Método `double calcularPrecoFinal(String codigo);`

Os métodos devem interagir com a lista de passagens, permitindo adicionar, remover, listar, buscar e calcular o preço final das passagens de acordo com suas regras.

5. Exercício: Sistema de Cadastro de Alunos com Exceções

   Crie uma classe chamada Aluno com os seguintes atributos e métodos:

   - Atributo `String matricula;`
   - Atributo `String nome;`
   - Atributo `double nota;`
   - Método `void validarNota(double nota);` (lança exceções personalizadas)

   Crie as seguintes exceções personalizadas para tratar erros e situações específicas:

   - `NotaInvalidaException`: lançada quando uma nota fornecida é inválida (menor que 0 ou maior que 10).
   - `MatriculaInvalidaException`: lançada quando uma matrícula fornecida é inválida (por exemplo, uma string vazia ou com tamanho inadequado).

   Implemente o método `validarNota` na classe `Aluno` para lançar a exceção `NotaInvalidaException` quando a nota fornecida for inválida. Adicione um construtor que aceite a matrícula e o nome do aluno e valide a matrícula, lançando a exceção `MatriculaInvalidaException` quando a matrícula for inválida.

   Crie uma classe chamada CadastroAlunos com os seguintes atributos e métodos:

   - Atributo `List<Aluno> alunos;`
   - Método `void adicionarAluno(Aluno aluno);`
   - Método `void removerAluno(String matricula);` (lança exceção personalizada)
   - Método `void listarAlunos();`
   - Método `Aluno buscarAluno(String matricula);` (lança exceção personalizada)

   Os métodos `removerAluno` e `buscarAluno` devem lançar a exceção `AlunoInexistenteException` quando não for possível encontrar um aluno com a matrícula fornecida. Adapte a classe `CadastroAlunos` para tratar as exceções lançadas e fornecer feedback apropriado ao usuário.

   Por exemplo, você pode criar um método para adicionar alunos que solicita os dados do aluno e valida a entrada:

   - Método `void adicionarAlunoInterativo();`

O método `adicionarAlunoInterativo` deve solicitar os dados do aluno, como nome, matrícula e nota. Em seguida, ele deve tentar criar um objeto `Aluno` com os dados fornecidos e adicioná-lo à lista de alunos. Se ocorrerem exceções (`MatriculaInvalidaException` ou `NotaInvalidaException`), o método deve exibir uma mensagem de erro e solicitar a correção dos dados.

Exemplo de implementação do método `adicionarAlunoInterativo`:

```java
public void adicionarAlunoInterativo() {
    Scanner scanner = new Scanner(System.in);
    boolean adicionadoComSucesso = false;

    while (!adicionadoComSucesso) {
        try {
            System.out.print("Digite o nome do aluno: ");
            String nome = scanner.nextLine();

            System.out.print("Digite a matricula do aluno: ");
            String matricula = scanner.nextLine();

            System.out.print("Digite a nota do aluno: ");
            double nota = scanner.nextDouble();
            scanner.nextLine(); // Consumir a quebra de linha restante

            Aluno aluno = new Aluno(matricula, nome);
            aluno.validarNota(nota);
            aluno.setNota(nota);
            adicionarAluno(aluno);
            adicionadoComSucesso = true;
            System.out.println("Aluno adicionado com sucesso!");
        } catch (MatriculaInvalidaException | NotaInvalidaException e) {
            System.err.println("Erro: " + e.getMessage());
            System.out.println("Por favor, insira os dados corretos.");
        }
    }
}
```