Universidade de Brasília- UnB
Faculdade Gama- FGA
Orientação a objetos

Trabalho prático final

Alunos:
180127667 Matheus Augusto M Ribeiro
190039166 Vinicius Alves F Livramento
170038971 Leonardo Sampaio Barros
170108996 Lucas Pereira Pires
190118059 Vinicius Angelo de Brito Vieira
202045802 João Victor Soares de Moura Costa
Professor orientador: André Lanna

Novembro
2021

Conteúdo

1 Introdução

Existem inúmeros modos de se programar um código, e estes modos são chamados de paradigmas de programação, sendo a programação orientada a objetos (POO) um desses. Como alternativa a programação estruturada, este modelo apresenta dois conceitos chaves principais: classes e objetos. A partir destes dois, outros conceitos são ramificados compondo assim este padrão de desenvolvimento de software amplamente utilizado em linguagens de programação atuais como python, C#, java, entre outras.

Pode-se entender o conceito de classes como um conjunto de características e comportamentos que definem quais objetos podem pertencer a essa classe. Assim, a classe de certa forma pode ser descrita como um molde, que toma forma ao se criar um objeto para esta. Esse processo de criação de objetos é chamado de instanciação de classe. Assim, de exemplo, podemos construir uma classe "carro" e usar como objeto "hb20", "uno" e assim por diante.

A partir desses dois conceitos iniciais, podemos derivar até os quatros conceitos pilares da POO: Encapsulamento, Herança, Polimorfismo e Abstração. Estes serão destrinchados e exemplificados mais adiante. Para entender melhor estes conceitos, foi desenvolvido um programa em linguagem Java que tem por objetivo o gerenciamento das despesas de alunos que residem em uma república. O código foi desenvolvido utilizando os fundamentos de POO e as aplicações serão destacadas ao longo deste relatório.

2 Desenvolvimento

O código deve realizar as seguintes funções: Cadastrar alunos residentes da república registrando nome, e-mail e rendimentos em um documento .txt chamado de alunos.txt; Cadastrar as despesas mensais da república registrando a descrição da despesa, a categoria a qual ela pertence e o valor da despesa em um documento .txt chamado de despesas¡Mês¿¡ano¿.txt; Cadastro de categorias e subcategorias para as despesas; Divisão do valor de despesas mensais de forma igualitária ou proporcional, a critério de escolha dos usuários.

Além disso, o programa deve apresentar exceções diante das seguintes situações: cadastro incompleto de aluno; cadastro de rendimento inválido; cadastro incompleto de despesa.

O programa foi desenvolvido em cima dessas diretrizes e do paradigma de POO.

2.1 Classes e objetos

Para começar, foi criado um diagrama de classes para que pudesse ser definido quais classes seriam criadas e quais objetos pertenceriam a cada classe. O digrama construído pode ser visualizado na figura 1.

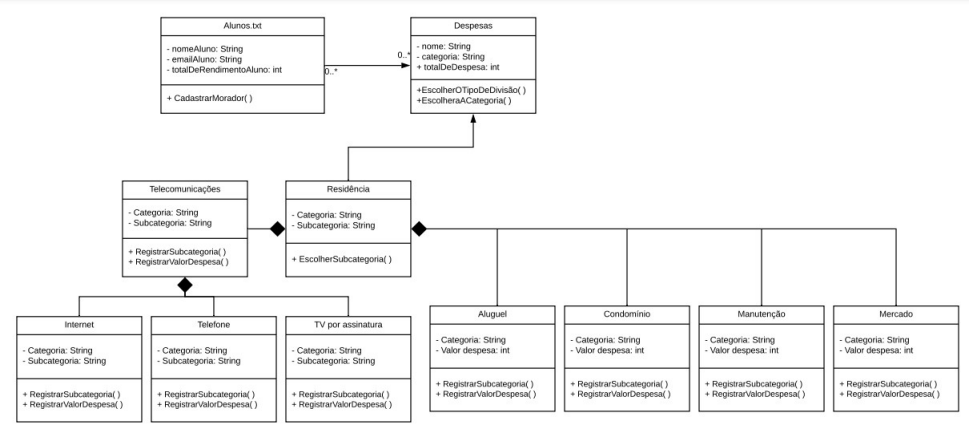

Figura 1: Diagrama de Classes

A partir deste diagrama, foram criados dois pacotes. O pacote Registro contém as classes CalculoIgualitario, CalculoProporcional, Categoria, Despesas, Alunos e Republica. O pacote UI contém apenas a classe Main_App. Assim, cada uma dessas classes pode ser instanciada para se criar objetos, como por exemplo na classe Alunos, serão instanciados os moradores da república, na classe Categoria, as categorias de despesa como telecomunicações ou residência, e assim em diante.

2.2 Abstração

O primeiro passo para realizar a POO é a abstração, e ela consiste basicamente em 3 pontos.

O primeiro ponto seria a atribuição de uma identidade ao objeto a ser criado. Essa identidade deve ser única para que não haja conflitos dentro do código.

Em seguida, deve-se estabelecer as características do objeto que será criado. Como supracitado, dentro da POO essas características recebem o nome de propriedades, e são equivalentes as variáveis presentes na programação estruturada, porém agrupadas em classes.

Por fim, deve-se definir quais ações o objeto criado irá executar. Como também já citado, essas ações recebem o nome de métodos. Assim, por exemplo, um objeto "gato" pode ter como propriedades tamanho, cor e peso, e como método "miar()".

2.3 Encapsulamento

Dentro de um código programado estruturalmente é comum que variáveis e funções se estendam ao longo do código por repetidas vezes, o que pode tornar a interpretação do código cada vez mais complexa. Portanto, para resolver isso, utiliza-se o conceito de encapsulamento, que nada mais é que o agrupamento destas funções e variáveis que podem ser logicamente unidas dentro de classes, a fim de organizar e reutilizar o código de uma melhor forma. Ao realizar esse agrupamento dentro de uma classe, as variáveis passam a ser chamadas de propriedades e as funções de métodos.

Assim, dentro do código desenvolvido pelos autores, temos o exemplo de propriedades para a classe Alunos na figura 2, e de um método para a mesma classe na figura 3.

```
public alunos(String nomeAluno, String emailAluno, String totalDeRendimentos) {
        this.nomeAluno = nomeAluno;
        this.emailAluno = emailAluno;
        this.totalDeRendimento = totalDeRendimentos;
        this.numeroDeAlunos = 0;
}
```

Figura 2: Exemplos de propriedades da classe Alunos

```
public void cadastrarAluno() {
        nomeAluno = JOptionPane.showInputDialog("Informe o nome do aluno");
        emailAluno = JOptionPane.showInputDialog("Informe o email do aluno");
        String totalDeRendimento = JOptionPane.showInputDialog("Informe o total de rendimento do aluno");
        float rendimentoDoAluno = Float.parseFloat(totalDeRendimento);
        totalR= rendimentoDoAluno + totalR;
        numeroDeAlunos = numeroDeAlunos + 1;

        alunos a = new alunos(nomeAluno, emailAluno, totalDeRendimento);

        aluno.add(a);

        System.out.println(numeroDeAlunos);
        System.out.println(totalR);


}
```

Figura 3: Exemplo de método da classe Aluno

2.4 Herança

Como o próprio nome sugere, o conceito de herança dentro da POO surge quando uma classe herda características de uma outra classe, e acrescenta suas próprias características depois. Isto é utilizado para que uma classe não fique muito concentrada e complexa, visto que muitas vezes classes compartilham características em comum, o que faz com que acrescentar apenas o que difere na classe filha uma estratégia de desenvolvimento muito mais eficiente.

No código desenvolvido, um exemplo de herança pode ser visto na imagem 4. Nela, é possível observar que a classe Despesas herda as características da classe Categoria, ao mesmo tempo em que possui suas próprias características (propriedades e métodos). Assim, a classe Categoria é a classe "pai" nessa relação, e a classe Despesas é a classe "filho".

```
public class Despesas extends Categoria{
        private String descricao;
        private String categoria;
        private String totalDeDespesa;
        private float despesaF;
        public float despesafinal;
        int ano;
        String mes;
        String nomeArquivo = "despesas" + "_" + mes + "_" + ano + ".txt";
        List<Despesas> despesas;

        public Despesas(String descricao, String categoria, String totalDeDespesa) {
                this.descricao = descricao;
                this.categoria = categoria;
                this.totalDeDespesa = totalDeDespesa;

        }
```

Figura 4: Exemplo de herança entre classes

2.5 Polimorfismo

O conceito de polimorfismo surge a partir da ideia de que um objeto pode assumir várias formas. Assim, quando uma classe filha herda as características de uma classe pai, ela pode sobrescrever qualquer uma dessas características com um novo valor que se adeque melhor. Desta forma evita-se a criação de uma mesma característica duas vezes para classes distintas.

3 Conclusão

Após o desenvolvimento do programa, foram realizados testes para verificar tanto o efetivo funcionamento do código, quanto o antendimento de todos os requisitos solicitados. Após validar o funcionamento, observou-se também a utilização de todos os pilares de programação orientada a objetos, e compreendeu-se a função de cada um na elaboração do código.

Bibliografia

[1]Acessado em 30 de outubro de 2021
https://balta.io/blog/orientacao-a-objetos
[2]Acessado em 30 de outubro de 2021
https://www.devmedia.com.br/os-4-pilares-da-programacao-orientada-a-objetos/9264
[3]Link para github dos desenvolvedores
https://github.com/joaodito/ProjetoOO_Unb