

Classes Abstratas e Interfaces

Capítulo VII e VIII

Classes Abstratas

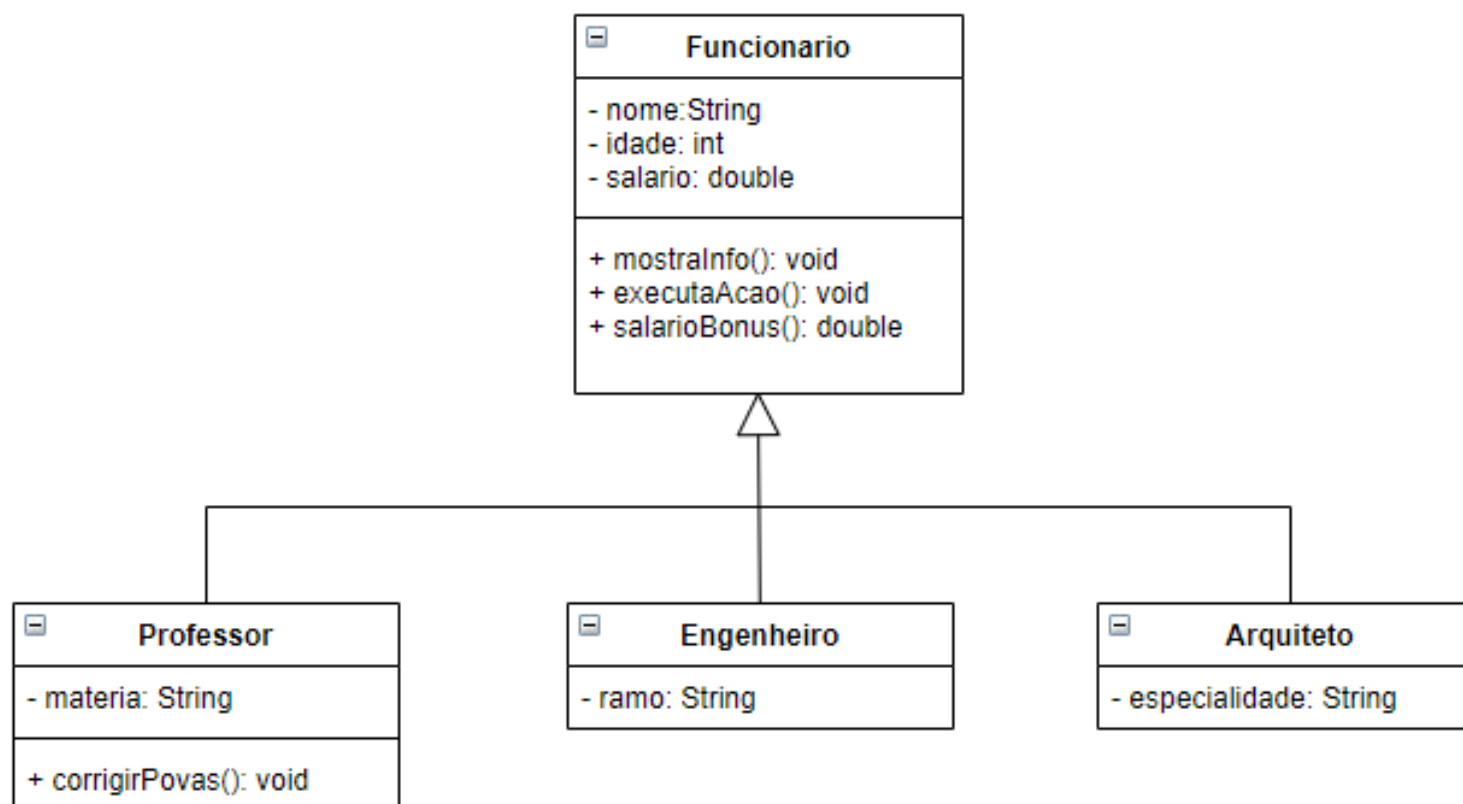
Para forçarmos o programador a criar códigos que fazem mais sentido, utilizaremos o conceito de classe Abstrata.

As Classes Abstratas não podem ser instanciadas, são Classes que servem de modelo para as classes filhas.

Métodos das Classes Abstratas

Tornamos um método abstrato, assim garantimos que ele sempre será reescrito pelas suas classes filhas.

Utilizando o exemplo da aula anterior



Utilizando o exemplo da aula anterior

```
public static void main(String[] args) {  
  
    Funcionario func = new Funcionario();  
  
    Funcionario[] funcionarios = new Funcionario[5];  
  
    Engenheiro eng = new Engenheiro("João", 23, 700.8, "Telecom");  
    Arquiteto arq = new Arquiteto("Maria", 34, 600.7, "Design");  
    Professor prof = new Professor("Joaquim", 56, 800, "Análise de dados");  
  
    funcionarios[0] = eng;  
    funcionarios[1] = arq;  
    funcionarios[2] = prof;  
}
```

Faz sentido instanciarmos um objeto Funcionario na main?

Neste caso, não. Funcionário é muito genérico e não faz sentido colocá-lo no array com os outros objetos, por exemplo. Ou seja, Funcionario é apenas utilizado para reaproveitarmos de código e realizar o polimorfismo.

Utilizando o exemplo da aula anterior

Mas como impedir que um objeto Funcionario seja instanciado?

Utilizamos a palavra reservada **abstract** para tornar a classe Funcionario em abstrata. Isso quer dizer que nenhum objeto Funcionario pode ser instanciado.

```
public abstract class Funcionario {  
  
    private String nome;  
    private int idade;  
    private double salario;  
  
    public void fazAlgo() {  
        System.out.println(nome + " está fazendo algo");  
    }  
}
```

```
5  
6 public static void main(String[] args) {  
7  
8  
9  
10  
11  
    Funcionario func = new Funcionario();  
}
```

Funcionario is abstract; cannot be instantiated

(Alt-Enter shows hints)

Utilizando o exemplo da aula anterior

Mesmo a classe sendo abstrata, ainda podemos criar uma referência para ela:

```
public static void main(String[] args) {
```

```
    Funcionario func;
```

Referência para um objeto

```
    Funcionario[] funcionarios = new Funcionario[5];
```

Referência para 5 objetos

```
Funcionario[] funcionarios = new Funcionario[5];
```

```
Engenheiro eng = new Engenheiro("João", 23, 700.8, "Telecom");
```

```
Arquiteto arq = new Arquiteto("Maria", 34, 600.7, "Design");
```

```
Professor prof = new Professor("Joaquim", 56, 800, "Análise de dados");
```

```
funcionarios[0] = eng;
```

```
funcionarios[1] = arq;
```

```
funcionarios[2] = prof;
```

Recapitulando:

- Se Funcionario não pode ser instanciado, por que não criar somente Engenheiro, Arquiteto ou Professor?
- Resposta:
 - Reuso de código
 - Polimorfismo: A classe genérica vai nos permitir tratar de forma fácil todo tipo de produto e colocar todos os tipos de produto em uma só coleção.

Utilizando o exemplo da aula anterior

```
public abstract class Funcionario {  
  
    private String nome;  
    private int idade;  
    private double salario;  
  
    public void fazAlgo() {  
        System.out.println(nome + " está fazendo algo");  
    }  
}
```

Na classe Funcionario temos o método fazAlgo() que é reescrita nas classes que herdam de Funcionario.

```
@Override  
public void fazAlgo() {  
    System.out.println(this.getNome() + " está programando");  
}
```

Engenheiro

```
@Override  
public void fazAlgo() {  
    System.out.println(this.getNome() + " está projetando uma casa");  
}
```

Arquiteto

```
@Override  
public void fazAlgo() {  
    System.out.println(this.getNome() + " está dando aula de " + materia);  
}
```


Professor

Utilizando o exemplo da aula anterior

Devido ao fato deste método ser reescrito em todas as classes filhas, e pelo fato de agora a classe Funcionario ser abstrata, podemos tornar o método fazAlgo() em abstrato:

```
public abstract class Funcionario {  
  
    private String nome;  
    private int idade;  
    private double salario;  
  
    public abstract void fazAlgo();  
}
```

Um método abstrato não possui corpo, declaramos apenas seu escopo para ele ser reescrito depois.



```
public class Arquiteto extends Funcionario{  
  
    private String especialidade;  
  
    public Arquiteto(String nome, int idade, double salario, String especialidade){  
        this.setNome(nome);  
        this.setIdade(idade);  
        this.setSalario(salario);  
        this.especialidade = especialidade;  
    }  
  
    @Override  
    public void fazAlgo() {  
        System.out.println(this.getNome() + " está projetando uma casa");  
    }  
}
```

Interface

Utilizando de uma **Interface**, temos a garantia de que toda classe que implementá-la, terá de implementar os métodos desta Interface.

```
public interface GerenciaProjeto {  
  
    public void gerenciar();  
  
}
```

Utilizaremos a Interface GerenciaProjeto, que será implementada pelas classes Arquiteto e Engenheiro

Interface

```
public class Engenheiro extends Funcionario implements GerenciaProjeto{

    private String ramo;

    public Engenheiro(String nome, int idade, double salario, String ramo){
        this.setNome(nome);
        this.setIdade(idade);
        this.setSalario(salario);
        this.ramo = ramo;
    }

    @Override
    public void gerenciar() {
        System.out.println(this.getNome() + " está gerenciando a criação de um novo aplicativo");
    }
}
```

```
public class Arquiteto extends Funcionario implements GerenciaProjeto{

    private String especialidade;

    public Arquiteto(String nome, int idade, double salario, String especialidade){
        this.setNome(nome);
        this.setIdade(idade);
        this.setSalario(salario);
        this.especialidade = especialidade;
    }

    @Override
    public void gerenciar() {
        System.out.println(this.getNome() + " gerenciou a ambientação de uma casa");
    }
}
```

Utilizamos a palavra reservada **implements** para dizer que uma classe implementa uma interface.

Interface

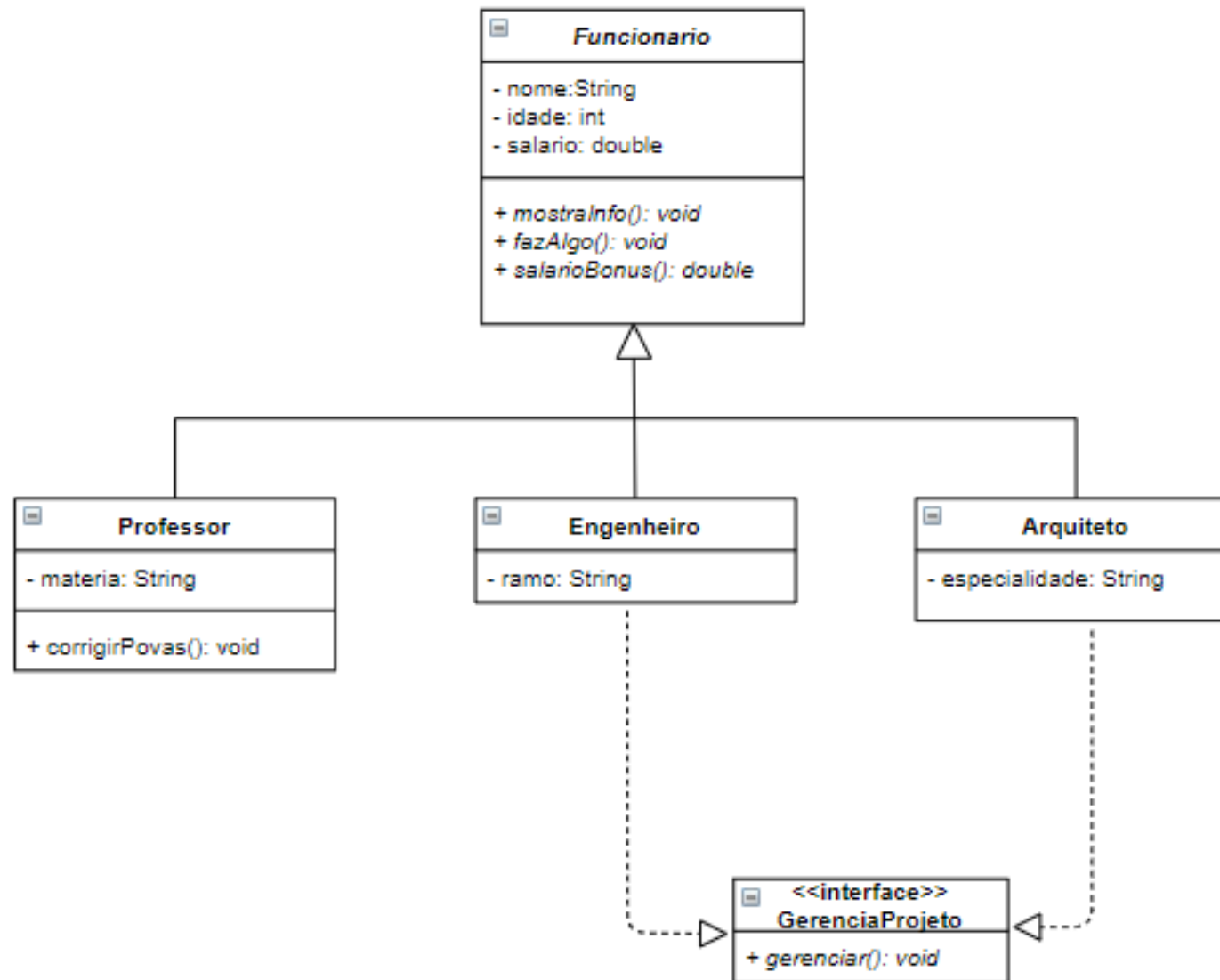
```
for (int i = 0; i < funcionarios.length; i++) {  
    if(funcionarios[i] != null){  
        if(funcionarios[i] instanceof Engenheiro){  
            System.out.println("Esta posição contém um engenheiro");  
            Engenheiro e = (Engenheiro)funcionarios[i];  
            e.fazAlgo();  
            e.mostraInfo();  
            e.gerenciar();  
        }else if(funcionarios[i] instanceof Professor){  
            System.out.println("Esta posição contém um professor");  
            Professor p = (Professor)funcionarios[i];  
            p.corrigiProvas();  
            p.fazAlgo();  
            p.mostraInfo();  
        }else{  
            System.out.println("Esta posição contém um arquiteto");  
            Arquiteto a = (Arquiteto)funcionarios[i];  
            a.fazAlgo();  
            a.mostraInfo();  
            a.gerenciar();  
        }  
    }  
}
```

Interface:

- **É um tipo que define apenas os métodos que uma classe deve implementar.**
- **Declaramos apenas o escopo desses métodos**
- **A interface estabelece um contrato com a classe.**

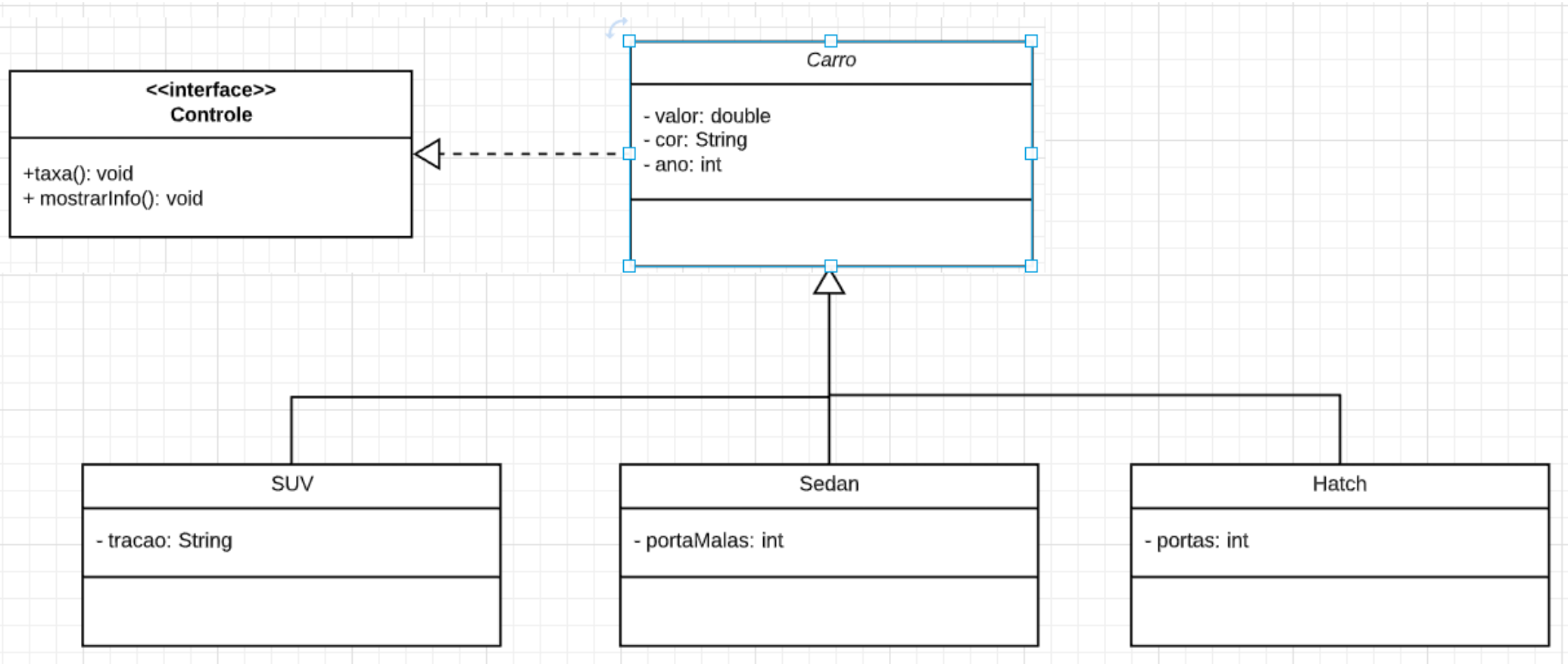
- **Pra que serve?**
 - **Criar sistemas com baixo acoplamento e flexíveis.**

Interface



Exercícios

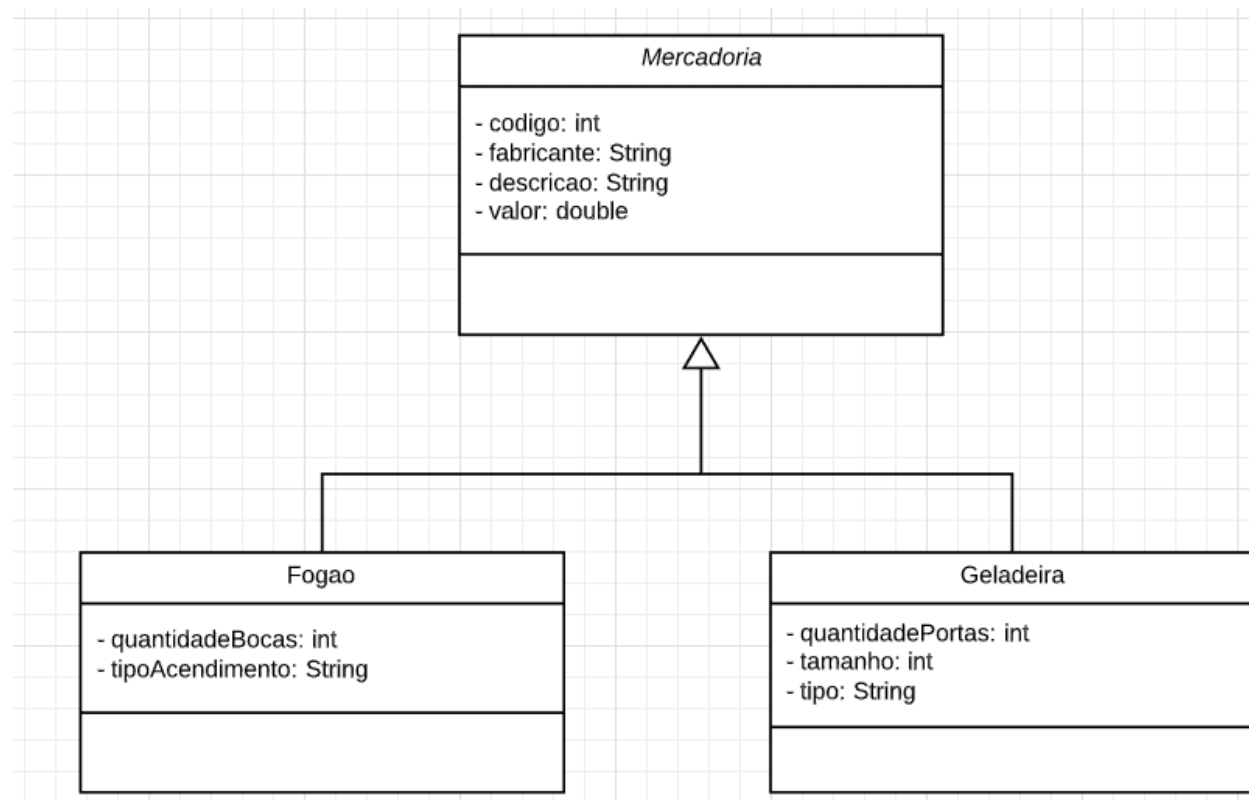
1. Desenvolva um sistema para a concessionária CarroFeliz



Exercícios

2. Você trabalha em um grande varejista, e foi designado para criar um sistema de cadastro das mercadorias da empresa. Neste primeiro momento serão inseridas as informações dos eletrodomésticos abaixo:

1. Mercadoria: código, fabricante, descrição e valor
2. Fogão: quantidade de bocas, tipo de acendimento (manual ou automático).
3. Geladeira: quantidade de portas, tamanho em litros e se ela é inox ou não



Obrigado!