Nome do Sistema: iDinner

Nome dos alunos: JOSÉ FIRMINO VERAS NETO, PEDRO JOSÉ VIANA DE LIMA GUEDES, IVANICE FRAGOSO MARTINS CUNHA E PEDRO AFONSO LEITE DE ANDRADE

25 de março de 2022

1. Descrição do sistema

O iDinner é um projeto que permite ao cliente de um restaurante específico fazer o seu pedido diretamente pelo seu dispositivo móvel, não exigindo que o cliente espere algum atendente do estabelecimento, melhorando a experiência do consumidor e evitando sobrecarregar os trabalhadores do local. O cliente, ao chegar no estabelecimento, abre o aplicativo do iDinner, confere as opções do cardápio e os preços direto pelo app, seleciona quais forem de sua preferência e após isso, o aplicativo imprime o pedido, com as opções selecionadas por ele, o seu nome e o número da sua mesa.

2. Especificação do Sistema

O sistema iDinner foi programado em Java, utilizando as bibliotecas:

- javax.swing.JOptionPane;
- java.util.ArrayList.

Dentro do sistema, primeiramente, foram usados quatro ArrayList de Strings, armazenados em uma classe específica, abstrata, chamada **Cardápio**, onde cada um deles armazena diferentes tipos de opções para o cliente (bebidas, pedido principal, sobremesa e acompanhamento). Os ArrayList são administrados através de outra classe específica que herda ela, a classe **GerenciarCardápio**, onde as opções de menu são inseridas manualmente pelos técnicos do aplicativo. Outra classe foi criada, a classe **Pedido**, que herda todos os ArrayList da classe cardápio, porém, diferente da classe de gerenciar, a classe pedido possui variáveis próprias para a criação de um objeto. Ela possui 5 variáveis inteiras: *pratoPrincipal*, *bebida*, *acompanhamento*, *sobremesa* e *mesa*.

A classe cardápio possui um método próprio de interface, onde ela exibe o cardápio para o cliente. Após exibir o menu, o sistema solicita que o cliente insira as informações de quais números do cardápio ele vai querer, esses números são armazenados nas variáveis inteiras de

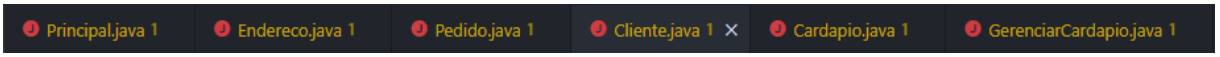
pedido e aplicadas aos ArrayList da classe cardápio.

Existem outras classes auxiliares que facilitam a experiência do usuário e podem melhorar o uso do aplicativo em visitas futuras, como as classes **Endereços** e **Cliente**, onde ele pode cadastrar o seu endereço e o seu perfil.

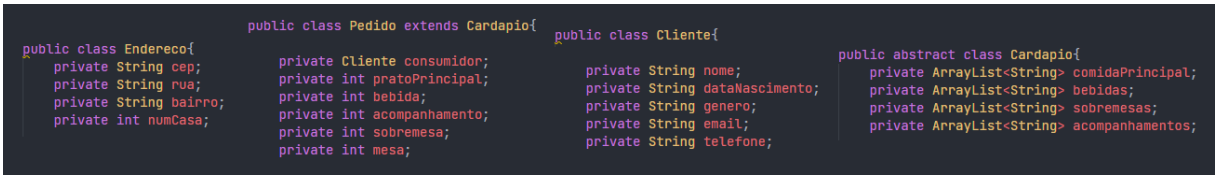
3. Conteúdos usados no projeto

**** Classe, objetos, métodos assessores, toString;***

Classes usadas até agora para o projeto:



Definição de parâmetros nas classes para criação de objetos:



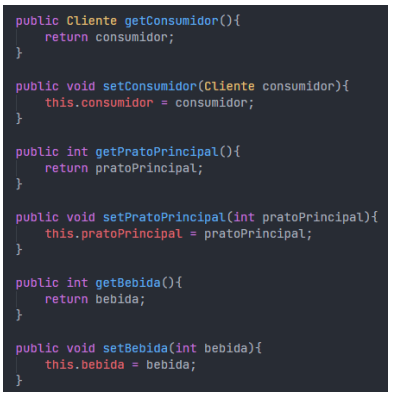
```java
public class Endereco{
    private String cep;
    private String rua;
    private String bairro;
    private int numCasa;
```

```java
public class Pedido extends Cardapio{
    private Cliente consumidor;
    private int pratoPrincipal;
    private int bebida;
    private int acompanhamento;
    private int sobremesa;
    private int mesa;
```

```java
public class Cliente{
    private String nome;
    private String dataNascimento;
    private String genero;
    private String email;
    private String telefone;
```

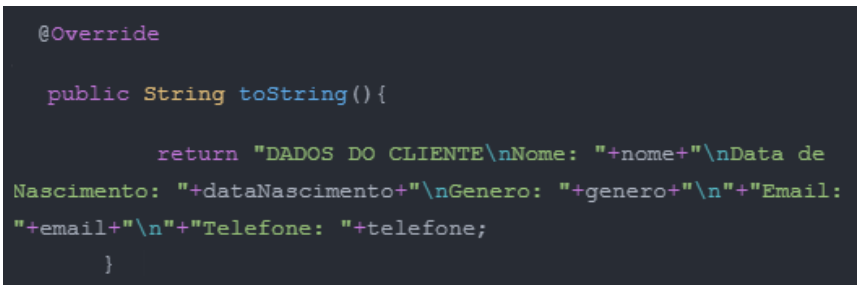
```java
public abstract class Cardapio{
    private ArrayList<String> comidaPrincipal;
    private ArrayList<String> bebidas;
    private ArrayList<String> sobremesas;
    private ArrayList<String> acompanhamentos;
```

Métodos assessores (Get e Set), exemplo tirado da classe Pedido:



```java
public Cliente getConsumidor(){
    return consumidor;
}

public void setConsumidor(Cliente consumidor){
    this.consumidor = consumidor;
}

public int getPratoPrincipal(){
    return pratoPrincipal;
}

public void setPratoPrincipal(int pratoPrincipal){
    this.pratoPrincipal = pratoPrincipal;
}

public int getBebida(){
    return bebida;
}

public void setBebida(int bebida){
    this.bebida = bebida;
}
```

Trecho de código após o cliente concluir seu cadastro no iDinner:



```java
@Override

 public String toString(){

          return "DADOS DO CLIENTE\nNome: "+nome+"\nData de
Nascimento: "+dataNascimento+"\nGenero: "+genero+"\n"+"Email:
"+email+"\n"+"Telefone: "+telefone;
      }
```

*** Encapsulamento;**

Todos os parâmetros para as classes foram encapsulados.

```java
public class Endereco{
    private String cep;
    private String rua;
    private String bairro;
    private int numCasa;
```

```java
public class Pedido extends Cardapio{

    private Cliente consumidor;
    private int pratoPrincipal;
    private int bebida;
    private int acompanhamento;
    private int sobremesa;
    private int mesa;
```

```java
public class Cliente{

    private String nome;
    private String dataNascimento;
    private String genero;
    private String email;
    private String telefone;
```

```java
public abstract class Cardapio{
    private ArrayList<String> comidaPrincipal;
    private ArrayList<String> bebidas;
    private ArrayList<String> sobremesas;
    private ArrayList<String> acompanhamentos;
```

*** Uso de construtores;**

Foram aplicados os construtores em todas as classes. O de armazenamento, e vazio, para inicializar as variáveis:

```java
public Cliente(String nome, String dataNascimento, String genero, String email, String telefone) {
    this.nome = nome;
    this.dataNascimento = dataNascimento;
    this.genero = genero;
    this.email = email;
    this.telefone = telefone;
}

    public Cliente() {
    this.nome = "-";
    this.dataNascimento = "-";
    this.genero = "-";
    this.email = "-";
    this.telefone = "-";
}
```

*** Objetos e métodos estáticos;**

Métodos estáticos usados na classe principal:

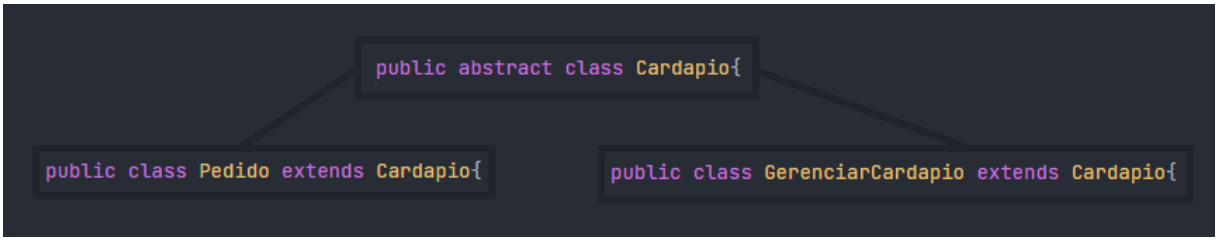
```java
public class Principal{
    Run | Debug
    public static void main(String[] args) {
        System.out.println("Bem-vindo ao iDinner. Aqui voce pode fazer seu pedido direto pelo nosso aplicativo!\n");
        System.out.println("Primeiro, vamos criar o seu registro de cliente!");

        Cliente c = new Cliente();
        c.registrarCliente();

        int aux=0;
```

*** Composição;**

Composição usada na construção das classes:

*** Herança;**

Herança de classes usadas no projeto:



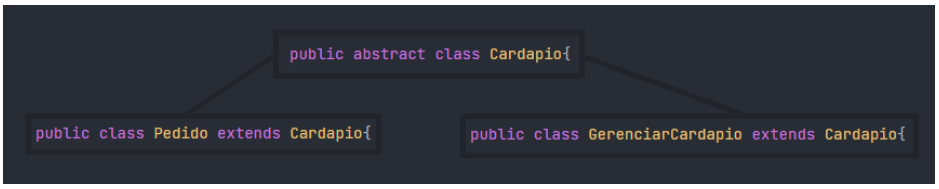
*** Coleções;**

Coleções usadas para construção do projeto:

```
import javax.swing.JOptionPane;
import java.util.ArrayList;
```

*** Classes e métodos abstratos / Interface**

A classe abstrata cardápio possui o método imprimir, usado para imprimir as informações de outras classes, como cliente, pedido, etc.




```
public abstract void imprimir();
```

*** Polimorfismo;**

Polimorfismo sendo usados através da função imprimir, sendo passada de uma classe para outra:

```
public void imprimir(){
    System.out.println("CLIENTE: "+cliente.getNome);
    System.out.println(comidaPrincipal(pratoPrincipal-1));
    System.out.println(bebidas(bebida-1));
    System.out.println(sobremesas(sobremesa-1));
    System.out.println(acompanhamentos(acompanhamento-1));
    System.out.println("MESA: "+pedido.getMesa);
}
```

*** Downcasting e Upcasting;**

Não foi usado.

Tópicos:

O programa começa dando as boas vindas ao usuário e registrando o cliente no sistema do iDinner:

```java
public class Principal{
    Run | Debug
    public static void main(String[] args) {
        System.out.println("Bem-vindo ao iDinner. Aqui voce pode fazer seu pedido direto pelo nosso aplicativo!\n");
        System.out.println("Primeiro, vamos criar o seu registro de cliente!");
```

Após isso, o programa chama o método responsável por registrar o cliente:

```java
Cliente c = new Cliente();
c.registrarCliente();
```

Após registrar o cliente, o sistema pede a entrada de um número inteiro, onde, dependendo desse valor, o programa irá realizar determinada função. Ele pode visualizar o cardápio, fazer o pedido direto ou apenas fechar o aplicativo. Para cada número, uma condicional diferente é acionada, responsável por chamar determinadas funções que serão responsáveis por realizar o pedido e imprimir o mesmo.

```java
int aux=0;

while(aux!=3){
    aux = JOptionPane.showInputDialog("Insira o que voce deseja fazer:\n 1 - Visualizar cardapio.\n2 - Fazer Pedido.\n3 -
    if(aux==1){
        Pedido p = new Pedido();
        p.fazerPedido();
        p.pedido.imprimir();
    }
    else if(aux==2){
        Cardapio ca = new Cardapio();
        ca.imprimirCardapio();
    }
    else{
        System.out.println("Obrigado e volte sempre!");
    }
}
```

As maiores dificuldades do grupo foi conciliar o projeto com as outras cadeiras e o tempo corrido. Além da dificuldade para fazer o código rodar, pois como não tivemos muito tempo para treinar esse período, foi muito comum ocorrer erros de sintaxe e colocações erradas.

4. Conclusão

Foi bem interessante desenvolver um projeto como esse pela primeira vez, desde estruturar uma ideia do zero até o ponto de dividir as funções e começar a escrever o programa em si. Como já dito anteriormente, o projeto não foi feito da forma ideal pois todos estávamos com o calendário apertado e não conseguimos nos dedicar o necessário para um projeto perfeito.

Porém, mesmo com todos os contratempos, foi possível absorver vários conhecimentos e como funciona a POO de uma forma geral, além de também aprender a usar as ferramentas que facilitam e poupam tempo na hora de programar, como gerar os getters e setters de forma automática. Os dois professores foram ótimos, sempre claros e fazendo o possível para ajudar os alunos nas aulas.