**Herança e Polimorfismo**

**Hierarquia de Classes**

Mapa de interações: <https://app.diagrams.net/#G1Hpet-8gVTOSvnAHQNI6kkiQGZ1ERZ3Lx>

Em nosso trabalho, o sistema de agendamento e monitoramento do Ru tem como base a classe Usuario onde são definidos os principais dados de cada usuário do sistema.

* Nome
* Identificador
* Email
* Senha
* Telefone
* Senha
* Horários

Em se tratando dos métodos da classe Usuario, temos as ações que dizem respeito *exclusivamente* aos usuários e que assim devem atuar como métodos dessa classe. Essa ações que poderão ser realizadas são:

* Cadastrar/Alterar/Visualizar horários
* Login

Como nosso sistema de informação considera 3 tipos principais de usuários, sendo eles os Discentes, os Docentes e os TAEs, e como a única diferença entre eles é o identificador, faz sentido que a classe Usuario atue como superclasse para as classes citadas. Assim, as classes Discentes, Docentes e TAEs atuarão como subclasses da superclasse Usuario e herdarão seus métodos e atributos. Caso haja necessidade, cada subclasse criada poderá sobrescrever métodos e atributos herdados da classe de origem.

public class Usuario {

String nome;

String identificador;

String email;

String telefone;

String senha;

int[][] horários = new int[15][5] //matriz que armazena os horários do usuário

cadastrarHorarios();

alterarHorários();

visualizarHorários();

login();

}

Visão da matriz de horários

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Segunda | Terça | Quarta | Quinta | Sexta | Sábado |
| 7:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14:00 | 1 | 1 | 1 | 1 | 1 | 1 |
| 15:00 | 1 | 1 | 1 | 1 | 1 | 1 |
| 16:00 | 1 | 1 | 1 | 1 | 1 | 1 |
| 17:00 | 1 | 1 | 1 | 1 | 1 | 1 |
| 18:00 | 1 | 1 | 1 | 1 | 1 | 1 |
| 19:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21:00 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22:00 | 0 | 0 | 0 | 0 | 0 | 0 |

Matriz 16x6

A tabela será preenchida com zeros e durante o cadastro dos horários pelo usuário, será inserido o valor “1” no horário cadastrado.

**Herança das Subclasses**

public class Discentes extends Usuario {

// atributos definidos igualmente à classe Usuario

}

public class Docentes extends Usuario {

// atributos definidos igualmente à classe Usuario

}

public class TAEs extends Usuario {

// atributos definidos igualmente à classe Usuario

}

*Superclasse*

Usuario

*Subclasses*

Discentes

Docentes

TAEs

Existem outras ações que poderão ser realizadas pelos usuários mas que não dizem respeito somente a sua própria classe. São elas:

* Monitorar lotação
* Acessar cardápio
* Agendar refeição
* Encomendar refeição
* Consultar e-Ticket

Para a realização dessas ações pelo usuário, uma sugestão é a criação das seguintes classes:

Obs: Uma possibilidade é considerarmos o agendamento e a encomenda de refeições na mesma classe, mas em métodos diferentes.

Assim, seria criado um método para agendar e outro para encomendar, além de um método de cancelamento para cada umas dessas ações

Public class Refeicao() {

String tipoRefeicao;

String turnoRefeicao;

Date data;

agendar ();

encomendar();

cancelarAgendamento();

cancelarEncomenda();

}

}

Public class Refeicao() {

String tipoRefeicao;

String turnoRefeicao;

Date data;

agendar ();

cancelar();

}

Também, podemos considerar que uma encomenda é uma refeição porém sobrescrevendo o método agendar.

Assim, Encomendas herdarão os métodos e atributos da classe Refeição

Public class Encomendas extends Refeicao() {

@Overwrite

// sobrescrever método agendar

}

Public class Encomendas() {

String tipoRefeicao;

String turnoRefeicao;

Date data;

encomendar();

cancelar();

}

A existência da classe AgendaRu é necessária para se armazenar todos os agendamentos em uma única variável. É importante separar a lista de agendamentos de refeições da lista de encomendas, sendo necessária a criação de duas listas.

Para cada agendamento ou encomenda, é adicionado na lista o identificador do usuário e a data do agendamento. Através dessas listas será possível separar os agendamentos por turno e por dia, facilitando a execução do método monitorar, que é o método presente nessa classe

Public class AgendaRU {

Map<String,String> agendaRefeicao = new HashMap<>()

// Ex: agendaRefeicao.put(“201469023C”, data)

Map<String,String> agendaEncomendas = new HashMap<>()

// Ex: agendaEncomendas.put(“201469023C”, data)

monitorar();

}

Quanto ao cardápio das refeições, faz sentido criar uma classe exclusiva para essa opção onde o cardápio de cada semana será armazenado em uma matriz. O método acessaCardapio apenas acessa essas informações no dia e turno solicitados pelo usuário.

Public class Cardapio () {

String diaSemana;

String turno;

String[][] cardapioCafeDaManha = new String[5][5]

String[][] cardapioAlmoco = new String[5][5]

String[][] cardapioJanta = new String[5][5]

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Almoço | Segunda | Terça | Quarta | Quinta | Sexta | Sábado |
| Prato Principal | Strogonoff de Frango | … | … | … | … | … |
| Opção Vegetariana | Strogonoff de Soja | … | … | … | … | … |
| Guarnição | Batata palha | … | … | … | … | … |
| Arroz | Branco e Integral | … | … | … | … | … |
| Feijão | Carioca | … | … | … | … | … |
| Sobremesa | Gelatina | … | … | … | … | … |

acessarCardapio();

}

Para armazenar os agendamentos e encomendas de cada usuário, pode ser criado um dicionário para cada turno de refeição. O usuário ao realizar um agendamento, por exemplo, terá sua ação registrada no dicionário correspondente ao turno. Essas informações poderão ser acessadas para a comprovação do agendamento no momento da retirar da encomenda ou de entrada no Ru com o horário agendado.

Public class eTicket() {

String nomeUsuario;

String identificador

Date data

Map<String,date> cafedamanha = new HashMap<>()

// Ex: cafedamanha.put(“Agendamento”, data)

Map<String,String> almoco = new HashMap<>()

// Ex: almoco.put(“Encomenda”, data)

Map<String,String> janta = new HashMap<>()

// Ex: janta.put(“Encomenda”, data)

acessarETicket();

}