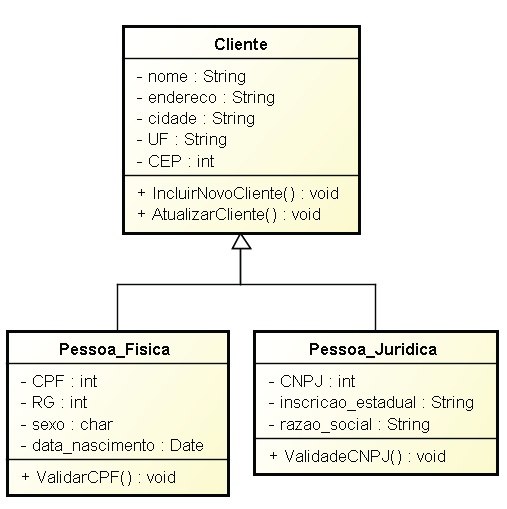
**Funções e Classes em um Código**

* Explique com suas palavras os conceitos de herança e polimorfismo.

Herança - é a possibilidade que um objeto filho tem de herdar do objeto pai atributos e métodos, promovendo assim a reutilização de códigos.

Polimorfismo – é a capacidade que é conferida ao objeto filho de modificar o comportamento de métodos adquiridos.

* O diagrama UML abaixo possui um dos princípios de orientação a objetos, análise e descreva o seu entendimento sobre a necessidade da utilização nesse cenário.



Visto que todo cliente, quer seja ele pessoa física ou jurídica, tem que ter um nome e estar estabelecido em algum lugar, a classe Cliente pode fornecer estes atributos como herança para as classes Pessoa\_Fisica e Pessoa\_Juridica incluindo também a possibilidade de inserir e atualizar novos registros, que são os métodos que as classes Pessoa\_Fisica e Pessoa Juridica também herdam.

* Você foi contratado para desenvolver uma lista de contatos, suas tarefas incluem:
  + Definir quais classes vão atender o requisito.

Classes: Contato – Contato\_Profissional – Contato\_Pessoal

* + Identificar quais atributos devem existir em cada classe.

Classe Contato: - nome, - email, - celular, - redes sociais, - notas;

Classe Contato\_Profissional: - cargo, - empresa;

Classe Contato\_Pessoal: - aniversário, - casamento

* + Identificar quais métodos são necessários desenvolver.

+ Adicionar\_Contato()

+ Remover\_Contato()

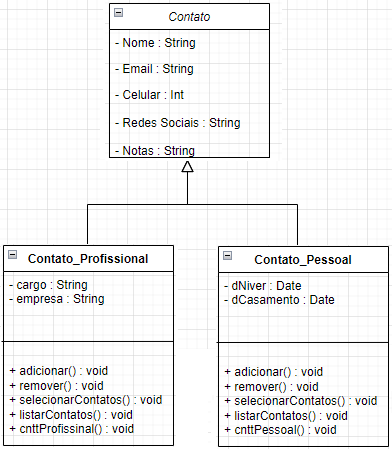
+ Listar\_Contatos()

+ Selecionar Contatos()

+ Contato Profissional()

+ Contato Pessoal()

* + Produzir um diagrama UML representando sua solução proposta.



* + Implementar a solução proposta.

const prompt = require('prompt-sync')();

class Contato{

constructor(nome,email,celular,rdSocial,notas){

this.nome = nome;

this.email = email;

this.celular = celular;

this.rdSocial = rdSocial;

this.notas = notas;

}

}

class CProfissional extends Contato{

constructor(nome,cargo,empresa,email,celular,rdSocial,notas,){

super(nome,email,celular,rdSocial,notas);

this.cargo = cargo;

this.empresa =empresa;

this.listaPro = [];

}

adicionar(contato){

this.listaPro.push(contato);

}

remover(contato){

this.listaPro.shift(contato);

}

selecionarContatos(){

this.listarContatos()

var cnttSelecionar = parseInt(prompt("Selecione o Contato: "))

console.log(`Contato desejado: \n

${this.listaPro[cnttSelecionar].cnttProfissional()}`) //Só funciona no browser.

}

listarContatos(){

this.listaPro.forEach(function(contato,index){

console.log(`${index} - ${contato.cnttProfissional()}`);

})

}

cnttProfissional(){

return `Nome: ${this.nome}

Cargo: ${this.cargo}

Empresa: ${this.empresa}

Email: ${this.email}

Celular: ${this.celular}

Rede Social: ${this.rdSocial}

Notas: ${this.notas}\n`;

}

}

class CPessoal extends Contato{

constructor(nome,email,celular,rdSocial,notas,dNiver,dCasamento){

super(nome,email,celular,rdSocial,notas);

this.dNiver = dNiver;

this.dCasamento = dCasamento;

this.listaPess = [];

}

adicionar(contato){

this.listaPess.push(contato);

}

remover(contato){

this.listaPess.shift(contato);

}

selecionarContatos(){

this.listarContatos()

var cnttSelecionar = parseInt(prompt("Selecione o Contato: "))

console.log(`Contato desejado: \n

${this.listaPess[cnttSelecionar].cnttPessoal()}`)

}

listarContatos(){

this.listaPess.forEach(function(contato,index){

console.log(`${index} - ${contato.cnttPessoal()}`);

})

}

cnttPessoal(){

return `Nome: ${this.nome}

Email: ${this.email}

Celular: ${this.celular}

Rede Social: ${this.rdSocial}

Data do aniversário: ${this.dNiver}

Data do Casamento: ${this.dCasamento}

Notas: ${this.notas}\n`;

}

}

var lima = new CProfissional("Luiz","Diretor","Minha S.A",'aaaa@bb.com',"31 7777888889","github", 'Gente boa');

var ro = new CProfissional("Rosangela","Programadora","Meninas S.A","bbb@bb.com", "38 999333333","github","Da melhor qualidade");

var joao = new CProfissional("João","Vice Diretor","Minha S.A",'yyyy@bb.com',"31 9900001111","github", 'Workaholic');

var lima2 = new CPessoal("Luiz",'aaaa@bb.com',"31 7777888889","whatsapp", "melhor qualidade","21/02/11", "01/09/19");

var ro2 = new CPessoal("Rosangela",'bbb@bb.com',"38 999333333","whatsapp", "Gente boa","21/02/98", "01/09/72");

var contPro = new CProfissional();

contPro.adicionar(lima);

contPro.adicionar(ro);

contPro.adicionar(joao);

//contPro.listarContatos();

//contPro.remover(lima)

contPro.listarContatos();

contPro.selecionarContatos();

contPess = new CPessoal();

contPess.adicionar(lima2)

contPess.adicionar(ro2);

contPess.listarContatos();

contPess.selecionarContatos();

* Seu restaurante favorito solicitou o seu serviço como desenvolvedor para criar um menu eletrônico, esse menu tem como objetivo facilitar a apresentação dos pratos servidos no local. Para isso, você deve analisar e construir:
  + A representação de um prato contendo os atributos para armazenar o nome, descrição, imagem e o valor.
  + O restaurante possui um número indefinido de pratos que precisam ser registrados.
  + O menu deve apresentar todos os pratos do restaurante e apresentar as informações do prato quando ele for selecionado.

const prompt = require('prompt-sync')();

class Prato{

constructor(nome,descricao,imagem,valor){

this.nome = nome;

this.descricao = descricao;

this.imagem = imagem;

this.valor = valor;

}

apresentarPrato(){

return `Nome: ${this.nome} - Valor: ${this.valor}`

}

getNome(){

return this.nome;

}

}

class Menu{

constructor(){

this.pratos = [];

}

addPrato(prato){

this.pratos.push(prato);

}

apresentarPratos(){

this.pratos.forEach(function (prato, index){

console.log(`${index} = ${prato.apresentarPrato()}`);

});

}

selecao(){

this.apresentarPratos();

var pratoSelct = parseInt(prompt(

"Selecione o prato desejado: "));

console.log(`Prato Selecionado: {this.pratos[pratoSelct].getNome()}`);

}

}

var doDia = new Prato("Tropeiro",'Bife de carne','', 59);

var frango = new Prato('Frango','Frango a xadrez','', 40);

var menu1 = new Menu();

menu1.addPrato(doDia);

menu1.addPrato(frango);

menu1.selecao();

Você deve seguir o conceito de orientação a objetos e pensar em uma implementação que separe as responsabilidades de cada necessidade.