UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO
DISCIPLINA: Programação Orientada a Objetos **2025/1**

PROVA DE SUFICIÊNCIA

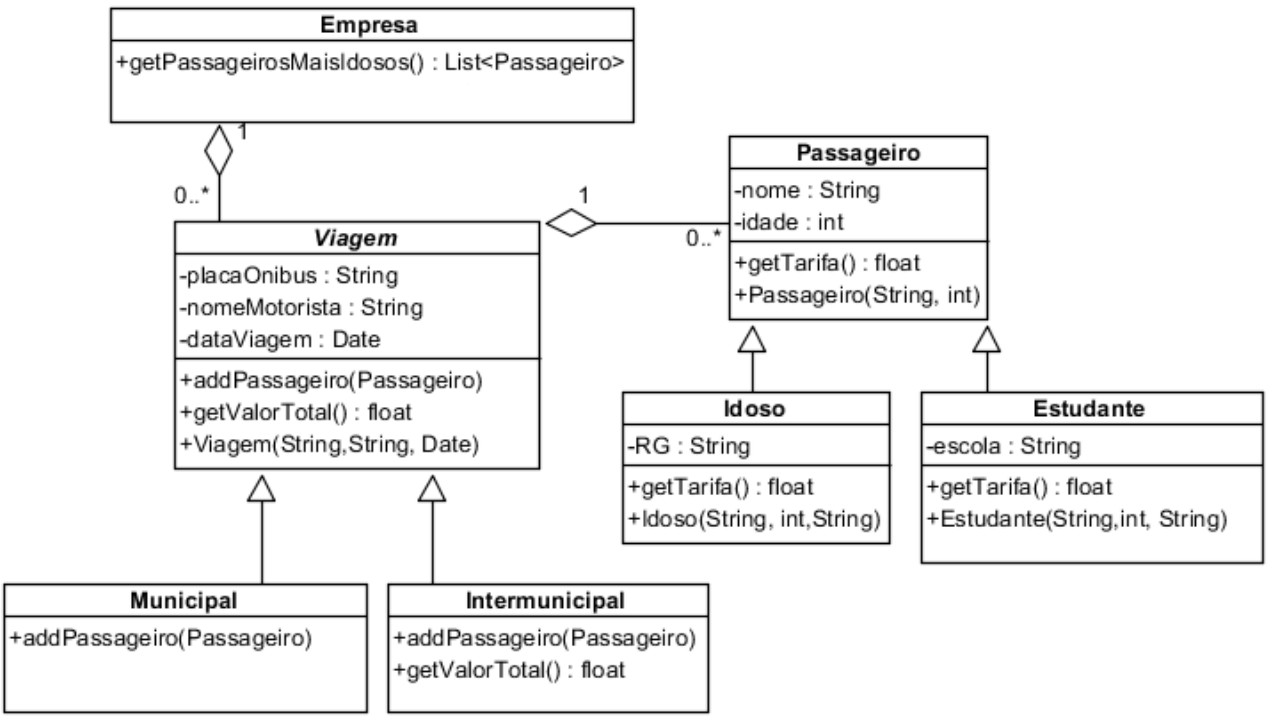
Esta prova de suficiência é composta de duas partes:

- Desenvolvimento do código-fonte
- Arguição oral e presencial do código-fonte entregue

Após finalizar o código-fonte deve enviar somente os arquivos .java para o e-mail abizon@furb.br com o assunto "Prova de Suficiência POO – SEU NOME"

QUESTÃO 1: (6,0)

Uma empresa de ônibus deseja informatizar seu sistema, registrando os passageiros em cada viagem. Os passageiros podem ser de três tipos: regulares, que pagam a tarifa inteira da viagem; estudantes, que pagam metade da tarifa; e idosos, que não pagam passagem. Obrigatoriamente um idoso deve ter 60 anos de idade ou mais (Estatuto do Idoso). A seguir é apresentado o diagrama de classes do sistema (os métodos *getters* e *setters* foram ocultados no diagrama, mas devem ser implementados).



Deve-se considerar que esta empresa pode atuar em linhas municipais e intermunicipais. As regras para os dois tipos de viagem são:

- o valor da tarifa inteira é de R$5,00 para ambos os casos. Implemente como um atributo de classe em `Passageiro` (`tarifaInteira`);
- viagens municipais: permitem que passageiros viagem em pé, com um limite total de 65 passageiros, sendo 30 passageiros sentados e 35 em pé (ignore a situação de desembarque, ou seja, considere apenas a entrada de passageiros no ônibus e não se importe se o passageiro está sentado ou não);
- viagens intermunicipais: não permitem que passageiros viagem em pé (também ignore o desembarque), ou seja, até 45 passageiros sentados. Também cobra uma taxa de embarque (R$ 3,15), de igual valor para todos os passageiros. Portanto, além da tarifa, cada passageiro deverá pagar o valor integral desta taxa, independentemente do seu tipo.

O sistema deve permitir a realização das seguintes operações:

1. quanto foi arrecadado em cada viagem (`float getValorTotal()` na classe `Viagem`);
2. retornar o(s) passageiro(s) transportado(s) com a maior idade identificada em relação a todas as viagens (`getPassageirosMaisIdosos() : List<Passageiro>` na classe `Empresa`);

Todos os atributos e métodos identificados no diagrama acima devem ser implementados, observando as assinaturas especificadas. Contudo, de acordo com a necessidade, novos métodos podem ser criados.

Implemente as classes do domínio da aplicação, assim como a interface com usuário (com Java Swing). Estas classes devem estar preparadas para instanciar objetos (recebendo os dados necessários na construção). Exceções devem ser lançadas ao menos nos seguintes casos: (i) quando tentar criar um idoso com menos de 60 anos; (ii) quando exceder o limite de passageiros no ônibus.

Pontuação:
Interface Gráfica: 20%
Classes de domínio: 80%

QUESTÃO 2 : (4,0)

Implementar a classe **`ProvaSuficiencia`** contendo um método com assinatura:

```
public static void serializar(String arqOrigem, String arqDestino)
```

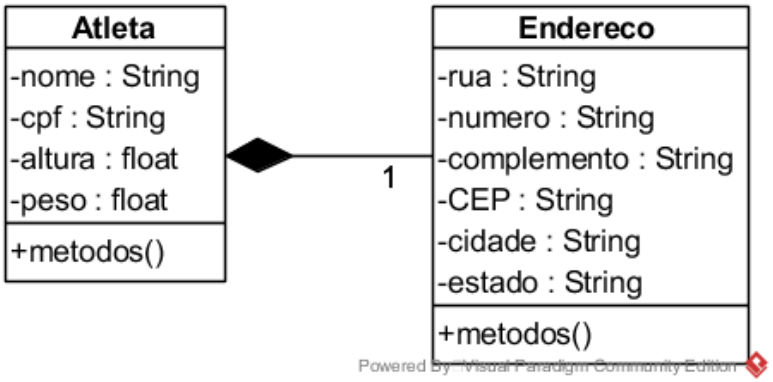
A classe `ProvaSuf` deve possuir o método `serializar`, cuja finalidade é criar um arquivo binário de objetos (seu nome é o parâmetro arqDestino), a partir dos dados existentes em um arquivo do tipo texto (seu nome é o parâmetro arqOrigem).

Os dados no arquivo texto origem referem-se a atletas de um clube esportivo. Cada linha possui os dados de um atleta, usando o formato CSV (*Comma-Separated Values* – valores separados por vírgula). Um arquivo exemplo será entregue pelo professor com os seguintes dados:

José dos Santos,123456789-00,1.92,98.3,Av. Brasil,123,Apto.203,89020-120,Blumenau,SC
Aparecida Maria,321654987-11,1.70,65.7,Rua João Pessoa,500,Apt 1001,89010-200,Blumenau,SC
João da Silva,987654321-99,1.89,87.6,Rua das Hortências,535,,89080-890,Blumenau,SC

A partir dos dados lidos de um atleta, deve-se instanciar um objeto de Atleta (classe a ser desenvolvida conforme o diagrama a seguir) e inseri-lo numa estrutura de dados da JCF. Deve-se selecionar uma estrutura de coleção, que garanta que

i. o objeto é único na coleção, usando CPF como critério de unicidade;
ii. ao serem percorridos os objetos (*Iterator*), eles estejam ordenados por CPF.



Após criados todos os objetos de Atleta e inseridos na coleção, a estrutura de dados deve ser serializada – o nome do arquivo é o parâmetro arqDestino.

Caso ocorra algum problema na leitura dos dados do arquivo origem (formato incorreto, erro de leitura, etc), o método deve lançar uma exceção do tipo `EArquivoOrigemIncorreto.` Você deve criar esta classe de exceção que deve ser subclasse de `IOException.`

Observações:

1. a prova é individual e com consulta. A interpretação do enunciado faz parte da avaliação;
2. esta é uma prova prática e a avaliação será feita sobre os programas-fonte entregues à banca examinadora e serão consideradas a racionalidade e lógica da solução;
3. coloque seu nome como comentário no início de cada programa-fonte;
4. você pode desenvolver as classes em qualquer ambiente de desenvolvimento (Netbeans, Eclipse, IntelliJ, etc);
5. coloque a questão 1 em um projeto e a questão 2 em outro projeto e envie somente os .JAVA;
6. quando a avaliação estiver encerrada, compacte os projetos e entregue-os ao professor.

Bons códigos! 😉

GABARITO

Parte teórica

1.Assinale as semelhanças entre objetos de uma mesma classe:

(x) possuem os mesmos atributos

(x) possuem os mesmos métodos

2. No fragmento de programa em uma linguagem hipotética abaixo,...

(x) 40

3. Qual dos modelos a seguir melhor representa o seguinte trecho de código?

(x) Opção 2

4. Considere as seguintes afirmações no contexto da Programação Orientada a Objetos.
(x) Somente a afirmativa I está correta.

5. Qual dos trechos de código abaixo melhor implementa o que está representado no modelo da figura?

(x) Opção 2

Parte prática

Questão 1 – 12 itens

Corrigir como se valesse 10,0 e depois ponderar para a nota da questão (4,5)

Hierarquia de Passageiro (2,5)	
Se Passageiro for abstract	Descontar 0,5
Construtor de Passageiro e Estudante	0,5
Construtor de Idoso (consistência 60 anos+exceção)	0,5
getTarifa() (3 classes)	1,0
Atributo de classe *tarifaInteira* na classe Passageiro	0,5
Hierarquia de Viagem (4,5)	
Viagem – abstract	0,5
Viagem agrega Passageiro	0,5
Construtores (3 classes)	0,5
addPassageiro() (3 classes, gerando exceção quando passa do limite)	1,5
getValorTotal() (Viagem e Intermunicipal, usando polimorfismo)	1,5
Empresa (2,0)	
addViagem	0,5
getPassageirosMaisIdosos() : List<Passageiro>	1,5
Compilação	1,0

Questão 2 – 8 itens

Corrigir como se valesse 10,0 e depois ponderar para a nota da questão (4,0)

- **Lançar exceção EArquivoOrigemIncorreto. = 0,5**

  Lançou outra exceção existente = - 0,2 Não é subclasse de IOException = -0,2

- **Leitura arquivo de texto = 1,5**

  Abrir arquivo BufferedReader (0,3) – caso não seja BufferedReader = -0,5

  Ler linha (0,4)

  Laço linhas até o fim (0,5)

  Fechar o arquivo (0,3)

- **Separar os dados de uma linha (split) = 0,5**
- **Criar objetos de Atleta e Endereco = 2,0**

  Duas classes implementadas (2 x 0,5)

  Objetos sendo criados e setados (0,5)

  Objetos Endereco agregado em Atleta (0,5)

- **Colocar objetos de Atleta na estrutura da JCF = 2,5**

  Utilizou TreeSet (1,0)

  Reimplementou o método equals do Atleta para comparar CPF (0,5)

  Implementou a interface Comparable para Atleta (1,0)

- **Classe Atleta e Endereco implements Serializable = 2 x 0,5**
- **Gravar arquivo de objetos = 2,0**

  Abrir arquivo ObjectOutputStream (0,3) - se criou arquivo a cada Atleta= -0,5

  Gravar coleção (1,4)

  – se gravou individualmente (dentro do laço) = -0,7

  - se deu write em Atleta e em Endereco -0,5

  Fechar arquivo (0,3)