1 - Os exercícios devem ser realizados INDIVIDUALMENTE.

2 - As resoluções dos exercícios devem estar em ARQUIVOS DIFERENTES. Uma pasta.zip com todos os arquivos deverá ser entregue até as 23:59 do dia 30/11/25 pelo Google Classroom.

3 - Nas descrições das questões, todos os atributos, métodos, classes e/ou interfaces em negrito devem ser explicitamente criados com o MESMO NOME nas soluções. Do contrário, você é livre para escolher qual a assinatura dos mesmos.

4 - Os tipos de entrada e saída explicitados devem ser respeitados.

5 - É permitido criar atributos, métodos, classes e/ou qualquer estrutura auxiliar que você considerar necessária para a resolução do problema. Porém, tudo que for pedido deverá ser implementado, obrigatoriamente.

6 - Os métodos que forem sobrescritos devem ter a anotação @Override.

7 - A correção da lista será feita através da análise individual de cada código, o principal aspecto não se baseia no output correto, mas em uma arquitetura de solução condizente com os princípios da programação orientada a objeto. Logo utilize getters, setters, modificações de visibilidade e todos os demais conceitos estudados em sala de aula na medida que julgarem necessário para a resolução do problema.

8 - Qualquer dúvida ou inconsistência em relação às questões, entrar em contato urgente com os monitores.

9 - Boa sorte e atentem-se ao prazo! 🙂

Questão 1 – Viagem para a Praia

A agência de viagens **“Partiu Praia!”** está organizando um pacote especial para o verão. Para isso, eles precisam de um sistema para gerenciar diferentes meios de transporte usados para levar os turistas.

Cada transporte precisa calcular o **valor da viagem** e **o tempo estimado de chegada**, porém cada um faz isso de maneira diferente.
Você foi contratado para modelar o sistema usando **herança e sobrescrita de métodos (override)**.

Requisitos:

Você deverá implementar o sistema orientado a objetos abaixo:

Interface: `Transporte`

Define o comportamento geral de qualquer meio de viagem.:

Métodos

- `public double calcularValor()`
- `public String tempoDeViagem()`

Classe base: `Viagem`

Representa uma viagem genérica.

Atributos

- `String destino`
- `double distanciaEmKm`
- `double precoBase` *(valor mínimo da viagem)*

Construtor

- recebe destino, distância e preço base

Esta classe implementa a interface `Transporte`.

Métodos implementados (pode ser comportamento padrão)

- `calcularValor()` → retorna `precoBase`
- `tempoDeViagem()` → retorna `"Tempo desconhecido"`

⚠ Estes métodos deverão ser **sobrescritos** nas subclasses.

Subclasses (devem sobrescrever os métodos)

`Ônibus`

- adicional de R$0,30 por km
- tempo de viagem = distância / 80 km/h
  (arredondar para inteiro e retornar: ex: "8 horas")

`Avião`

- adicional de R$1,00 por km
- tempo de viagem = distância / 800 km/h
  (arredondar para 1 casa decimal e retornar: "0.5 horas")

`Carro`

- adicional de R$0,50 por km
- tempo de viagem = distância / 100 km/h
  (arredondar para inteiro e devolver: "6 horas")

Cada classe deve sobrescrever:

```
@Override
public double calcularValor()

@Override
public String tempoDeViagem()
```

Classe `Main`

No `main`, crie uma viagem para Salvador (900 km) usando:

Ônibus
Avião
Carro

Para cada um, imprimir:

```
Transporte: <tipo>
Destino: Salvador
Preço da viagem: R$xxxx
Tempo estimado: xxxx
```

Questão 2 – Sistema de Ranking Inteligente da ATP

A ATP está testando um novo sistema de atualização de ranking, em que pontos podem ser adicionados ou removidos remotamente conforme jogos acontecem ao redor do mundo.
O problema é que as atualizações estão acontecendo simultaneamente, causando inconsistência no ranking final do jogador.

Você foi contratado para resolver o problema utilizando **métodos sincronizados**.

Requisitos do Sistema

Você deverá implementar um sistema composto por:

Classes e Interfaces

Classe `JogadorATP`

Representa um jogador no ranking da ATP.

Atributos

- `private double pontos;`
- `private String nome;`

Métodos

- `public synchronized void adicionarPontos(double valor)`
    - Aumenta os pontos do jogador
    - Imprime qual thread adicionou pontos e o total final

- `public synchronized void removerPontos(double valor)`

    - Se o valor for maior que os pontos, lançar `PontosInsuficientesException`

    - Imprime qual thread removeu pontos e o total final

- `public double getPontos()`

✔ *Ambos os métodos devem ser sincronizados, utilizando o monitor do próprio objeto (`this`).*

Classe AtualizacaoRanking (Thread)

Representa uma atualização de ranking (ganho ou perda de pontos).

Atributos

- `private JogadorATP jogador;`
- `private boolean ehGanho; // true = adiciona pontos, false = remove pontos`
- `private double valor;`

Método

- No `run()`, executar a operação correspondente.

Exceções

`PontosInsuficientesException`

Lançada quando há uma tentativa de remover mais pontos do jogador do que ele possui atualmente.

Classe `Main`

No `main`:

- Criar um jogador com 100 pontos

- Criar múltiplas threads simulando ganhos e perdas simultâneas
- Testar:
    - ganhos concorrentes
    - perdas concorrentes
    - tentativa de perder pontos além do limite → exceção
    - 5 ganhos e 5 perdas simultâneos → ranking consistente

Questão 3 – Javatouille

Você é responsável por desenvolver um sistema de simulação que representa a cozinha de determinado restaurante. O sistema deve coordenar o trabalho entre Aprendizes e Chefes usando um Balcão compartilhado, onde os aprendizes preparam os ingredientes a serem utilizados pelos Chefes em suas receitas. O sistema é implementado seguindo o paradigma orientado a objetos e concorrência, sendo composto por 4 classes principais: Ingrediente, Balcao, Aprendiz e Chefe.

Ingrediente: Representa os ingredientes manipulados

- Atributos:
  string nome: Representa o nome do ingrediente

Balcao: Classe que representa o local compartilhado para a disposição dos ingredientes (o buffer).

- Atributos:
  string nome: Representa o nome do balcão
  int capacidade: Representa a quantidade de espaço para ingredientes na bancada
  É necessário também implementar uma fila para os ingredientes que estão no balcão

- Métodos:
  public void colocar(Ingrediente ingrediente) throws InterruptedException: método utilizado pelos aprendizes para colocar os ingredientes prontos para uso no balcão

public void retirar() throws InterruptedException:
método utilizado pelos chefes para retirar os ingredientes prontos para uso do balcão

Aprendiz:

- Atributos:
  Balcao balcao: Referência ao **Balcao** utilizado na cozinha
  int desempenho: Representa a quantidade de ingredientes preparados pelo aprendiz por iteração.

- Métodos:
  Implementa a interface Runnable e simula a produção de ingredientes

Chefe:

- Atributos:
  Balcao balcao: Referência ao **Balcao** utilizado na cozinha
  int desempenho: Representa a quantidade de ingredientes prontos consumidos pelo Chefe por iteração.

- Métodos:
  Implementa a interface Runnable e simula o consumo dos ingredientes

Main:

Implementar casos

- Caso de balcão com pouco espaço
- Caso de capacidade de produção dos aprendizes muito superior a de chefes
- Caso de capacidade de consumo dos chefes muito superior à dos aprendizes

Questão 4 – Java Souls

Você ficou responsável por implementar o sistema de descanso em um jogo, onde há diversas opções de locais de descanso para o Personagem, sendo eles **Fogueira** para vida, **FonteMagica** para mana e **Acampamento** para estamina. O sistema é composto pelos seguintes elementos:

Personagem
Classe que representa o personagem que efetuará o descanso

- Atributos:
  double vidaAtual: Representa a quantidade de vida que o personagem está
  double vidaMaxima: Representa a capacidade máxima de vida do personagem
  double energiaAtual: Representa a quantidade de energia que o personagem está
  double energiaMaxima: Representa a capacidade máxima de energia do personagem
  TipoEnergia tipo: Representa o tipo de energia utilizado pelo personagem

- Métodos:
  gets e sets para **vidaAtual e energiaAtual**
  get para **vidaMaxima e energiaMaxima**

TipoEnergia
Enumerador que define o tipo de energia do personagem

- **MANA**: Utiliza e recupera apenas mana
- **ESTAMINA**: Utiliza e recupera apenas estamina

Restauracao
Interface que define os efeitos e durações de um descanso

- Métodos:

- **public void restaurar(Personagem personagem, double quantidadeRecuperada)**: recupera o personagem com o tipo correspondente de restauração

- **public void calcularDuracao(double quantidadeRecuperada)**: exibe a quantidade de vida efetiva que foi recuperada assim como a duração desse descanso

Fogueira
Classe que representa um local de restauração de vida

- Atributos:
  double eficiencia: Representa a taxa de recuperação de vida por segundo proporcionada pela fogueira

- Métodos:
  implementa a interface de **Restauracao**

FonteMágica
Classe que representa um local de restauração de mana

- Atributos:
  double eficiencia: Representa a taxa de recuperação de mana por segundo proporcionada pela fonte mágica

- Métodos:
  implementa a interface de **Restauracao**

Acampamento
Classe que representa um local de restauração de estamina

- Atributos:
  double eficiencia: Representa a taxa de recuperação de estamina por segundo proporcionada pelo acampamento

- Métodos:
  implementa a interface de **Restauracao**

EnergiaIncompatívelException
Deve estender a classe Exception. Chamada quando o personagem tentar restaurar sua energia usando um tipo incompatível.

Main

Implementar testes:

- Restauração de vida
- Restauração de energia bem sucedido
- Restauração de energia incompatível