

PRONTUÁRIO: INDICAR NO ZIP

Dia: 17/06/2025

Horário: 8h00 – 11h30

✓ A prova é individual e sem consulta.	✓ A prova deve ser nomeada da seguinte forma: PRONTUARIO_P3, com o "SC".
✓ Não coloque nome no projeto.	✓ Não envie apenas as classes, mas todo o projeto via zip no Moodle.
✓ Atribui-se nota zero à prova em desacordo com o item acima.	

Considere o problema especificado a seguir:

Você foi contratado para desenvolver um software para gestão de um estacionamento. No software, os clientes (`Customers`) são identificados pelo número da placa (`plate`), telefone (`phoneNumber`) e tipo do veículo (`vehicleType`), que pode ser um carro (`CAR`) ou uma moto (`MOTORCYCLE`). Ao entrar no estacionamento, é criado no sistema um registro (`Ticket`) que contém um ID do tipo UUID, o cliente (`customer`), o dia e a hora da entrada (`entry`) e o dia e a hora da saída (`exit`). No momento da saída, é registrado o dia e a hora em que o cliente deixou o estacionamento.

O valor a ser pago é calculado a partir da permanência em horas no estacionamento, considerando faixas de valores por período. Cada período de horas possui um valor diferente. Inicialmente, existem planos para 1h a R\$7,00, 2h a R\$10,00 e 12 horas por R\$30,00. No sistema, o cliente sempre paga o menor valor possível. Por exemplo, se permanecer por 7 horas, pagará R\$30,00 (12 horas), mas se ficar apenas 5 horas, pagará R\$27,00 (2x2h + 1h). Entretanto, o dono do estabelecimento é extremamente inflexível com atrasos: se o cliente se atrasar, mesmo que por um milissegundo, será cobrado por uma hora completa.

Como o valor e a forma de cobrança podem mudar ao longo do tempo, o custo por faixa de permanência é armazenado de forma flexível em um banco de dados. O DBA do seu time já construiu uma classe chamada `DatabaseBuilder` (pacote `persistence`) que cria e povoa, com dados de teste, um banco de dados SQLite chamado `database.db` (raiz). Os métodos dessa classe são invocados no início do método `main()` da classe `Main` (pacote `main`). Assim, um banco de dados novo é criado sempre que a aplicação é executada.


Execute as atividades a seguir para a implementação do exercício em Java. Para a atribuição da nota, será levada em conta não apenas a funcionalidade, mas também a qualidade, adequação e pertinência de cada solução. Bom senso faz parte da prova

Dica 1: para salvar datas com horário no banco de dados, converta-as em `String` por meio do método `toString()`. Para recriá-las a partir de uma `String`, utilize o método `LocalDateTime.parse()`.

Dica 2: para salvar UUIDs no banco de dados, converta-os em `String` por meio do método `toString()`. Para recriá-las a partir de uma `String`, utilize o método `UUID.fromString()`.

#	Descrição	Pont.
1	Adicione pacotes <code>costs</code> , <code>customer</code> e <code>ticket</code> . Nos respectivos pacotes, crie interfaces DAO para cada tabela do banco. Na interface <code>PeriodCostDao</code> , crie um método <code>findAll()</code> , que retorna uma lista de <code>PeriodCostDto(int hours, double fee)</code> . Na interface <code>CustomerDao</code> , crie um método <code>save()</code> e outro <code>findOne()</code> , que salvam e recuperam objetos do tipo <code>CustomerDto(String</code>	1,5 pt

	plate, String phone, String type), respectivamente. Na interface TicketDao, crie os métodos save() e findOpenTicket(), que salvam e recuperam objetos do tipo EntryTicketDto(String id, String plate, String entry). Crie também o método updateExit(), que atualiza um ticket a partir do objeto ExitTicketDto(String plate, String exit, double fee). O método findOpenTicket() só deve encontrar um ticket se ele estiver aberto, pois podem existir outros registros de estacionamento já concluídos para o mesmo veículo. Utilize records e Optional para obter a pontuação máxima na questão.	
2	Crie classes PeriodCostDaoImpl, CustomerDaoImpl e EntryTicketDtoImpl no pacote persistence e forneça implementações concretas para cada uma das interfaces. Para organizar o acesso ao banco de dados, crie uma classe ConnectionFactory, que utiliza apenas uma conexão para toda a aplicação e também gerencia a instanciação de objetos PreparedStatement.	1,5 pt
3	Implemente as classes Customer e Ticket, colocando-as nos seus respectivos pacotes. Na classe Customer, utilize um enum para representar o VehicleType. Na classe Ticket, crie dois construtores. O primeiro deve receber apenas o customer, criando o id aleatoriamente (UUID.randomUUID()) e atribuindo o dia e hora atual a variável entry. O segundo deve receber essas três informações por parâmetro e as atribuir às variáveis da classe. Na classe Ticket, crie o método exit() para registrar a hora de saída a partir do horário atual. Crie também o método parkingDuration(), para calcular e retornar a duração do veículo no estacionamento. Em ambas as classes, crie métodos getters, toString(), equals() e hashCode().	1,0 pt
4	Crie uma exceção não verificada chamada EntityAlreadyExistsException e a coloque no pacote persistence.	0,5 pt
5	Crie uma classe chamada RegisterCustomerService no pacote customer, que receba um CustomerDao por injeção de dependência. Em um método register(), receba a placa, o telefone e o tipo do veículo do cliente para salvá-los no banco de dados. Valide o parâmetros do método e dispare exceções se for pertinente. Caso o registro já exista no banco, lance uma EntityAlreadyExistsException.	1,0 pt
6	Crie uma classe chamada RegisterEntryService no pacote ticket, que receba CustomerDao e TicketDao por injeção de dependência. Em um método register(), receba a placa do veículo do cliente. Se o cliente não existir no banco de dados, dispare uma NoSuchElementException. Se o cliente existir, crie um ticket e salve-o no banco de dados. Como as interfaces Dao lidam com objetos Dtos e não entidades, crie um método auxiliar na conversão entre os tipos.	1,0 pt
7	Crie uma classe chamada RegisterExitService no pacote ticket, que receba CustomerDao, TicketDao e PeriodCostDao por injeção de dependência. Em um método register(), receba a placa do veículo do cliente. Se o cliente não existir no banco de dados, dispare uma NoSuchElementException. Obtenha o ticket	1,0 pt

	aberto relativo a placa do carro no banco de dados. Se não houver um ticket aberto para a placa, dispare uma <code>IllegalStateException</code> . Como as interfaces <code>Dao</code> lidam com objetos <code>Dtos</code> e não entidades, crie métodos auxiliares para a conversão entre os tipos. De posse do ticket, registre a saída e obtenha o tempo de permanência do carro no estacionamento. Utilize o método <code>calculateFee()</code> da Questão 8 para calcular o custo do estacionamento. Atualize o ticket no banco de dados com o horário de saída e o valor pago. Retorne o valor pago como resultado do método.	
8	Implemente o método <code>calculateFee()</code> , que recebe o número de horas de permanência no estacionamento e calcula o menor valor com base na tabela de permanências e os valores disponível no banco de dados.	1,5 pt
9	Na classe <code>Main</code> , crie objetos dos três services e realize a injeção de dependência dos repositórios. Em seguida, registre um <code>customer</code> , uma entrada e uma saída. Como não é possível testar parmanências maiores do que <code>millisseguros</code> a partir dos services, existem registros de entrada de sete horas incompletas e quinze horas incompletas já cadastrados no banco de dados, respectivamente para as placas "POO0007" e "POO0015". Você pode incluir outros dados de teste na classe <code>DatabaseBuilder</code> , mas não pode modificar os dados existentes.	1,0 pt
	Utilizar programação imperativa nos pontos do código que poderia usar programação declarativa com a API de Stream.	-1,0pt

Glossário SQL:

INSERT INTO <table> (<column>, ..., (<column>) VALUES (<value>, ..., (< value >)

SELECT <columns / * > FROM <table> WHERE <criteria> AND <criteria> (ex.: <Column> IS NULL)

UPDATE bulletin SET <column> = <value>, ..., <column> = <value> WHERE <criteria>

Prêmio Usain Bolt: O aluno que terminar todas as atividades corretamente primeiro ganha 1,0pt adicional para usar em outras provas. Você está voando?

*** Boa sorte! ***