Programação Orientada a Objetos
Projeto Orientado a Objetos
Valor: 70 pts
Última atualização: 22/05/2023

1. Orientações Gerais

O trabalho deve ser realizado em grupos de até cinco alunos e deverá ser entregue no moodle de forma incremental a cada semana (sprint), sempre no domingo até às 23:59. É permitida a discussão de estratégias com os colegas, porém a modelagem e implementação somente deve ser feita pelos integrantes do grupo. Cópias de modelos e programas (ainda que parciais) não serão aceitas. Entregas com atraso poderão ser aceitas, porém serão penalizadas com perda de pontos pelo grupo. Deve ser postado APENAS um arquivo compactado por grupo contendo todos os entregáveis da semana. Apenas um aluno deve realizar a entrega informando o nome dos integrantes do grupo. Os grupos não devem ser alterados ao longo do semestre.

2. Objetivo

Neste trabalho prático desenvolveremos um modelo de classes e implementaremos um sistema de gestão para um Aeroporto. O projeto será desenvolvido em sprints semanais, que envolverão a modelagem das classes para atender ao escopo da sprint, e também a implementação dessas classes em PHP. O código-fonte das classes deverá ser testado através da chamada de métodos das mesmas.

A dinâmica de trabalho se dará da seguinte forma:

- A sprint se inicia em uma quinta-feira, na qual serão apresentados os requisitos a serem atendidos;
- A modelagem das classes em linguagem UML deverá ser entregue via moodle até o domingo subsequente em formato PDF;
- Na terça-feira da semana seguinte, a implementação das classes será iniciada na aula na sala 312 do bloco 1 (CCE).
- O código-fonte gerado deverá ser enviado via moodle também até o domingo subsequente.

A pontuação se dará da seguinte forma:

- 6 pts para a entrega do modelo de classes a cada sprint, totalizando 42 pts ao final do semestre.
- O código-fonte, apesar de entregue a cada sprint, somente será avaliado e pontuado na entrega da versão final, totalizando 28 pts.

2.1. Sprint 1

Nesta sprint, vamos iniciar o desenvolvimento do sistema, que deverá atender aos seguintes requisitos:

- O sistema deve permitir o cadastro de companhias aéreas, contendo nome, código, razão social, CNPJ e sigla.
    - A sigla deve ser formada por duas letras.
- O sistema deve permitir o cadastro de aeronaves, contemplando os seguintes atributos:
    - Fabricante (Ex.: Boeing)
    - Modelo (Ex.: A-320)
    - Capacidade de passageiros
    - Capacidade de carga em kg
    - Registro da aeronave
        - Composto pelo prefixo, que contém duas letras
        - Um hífen
        - Seguido de três letras
        - (Ex.: PR-GUO)
    - No Brasil, somente são permitidos para voos comerciais os prefixos PT, PR, PP, PS, que devem ser validados.
- Toda aeronave pertence a uma companhia aérea.
- O sistema deve permitir o cadastro de voos, que consistem em um deslocamento entre dois aeroportos em um dado horário de partida, realizado por uma companhia aérea.
    - A frequência dos voos também deve ser modelada. Existem voos diários e voos em apenas alguns dias da semana.
    - Um voo costuma ser executado por uma mesma aeronave, que pode executar mais de um vôo.
    - A duração estimada do voo também deve ser armazenada.
- A execução de um voo, ou seja, sua ocorrência e os respectivos detalhes, também deve ser armazenada.
    - Por exemplo: Voo AD4127 da Azul, entre CNF e CGH, executado no dia 13/03/2023.
    - Os horários de partida e de chegada devem ser armazenados.
    - Ocasionalmente, uma aeronave diferente da aeronave cadastrada pode executar um voo.
- Os aeroportos também devem ser cadastrados no sistema com sua sigla, que possui três letras, e a cidade e estado do aeroporto.

2.2. Sprint 2

Nesta sprint devem ser modeladas e implementadas classes para atendimento dos seguintes requisitos:

- O sistema deve permitir a definição de um código para cada vôo, caso ainda não tenha sido modelado, seguindo a seguinte regra de validação:
    - O código deve ser composto de duas letras seguidas de quatro números;
    - As duas letras iniciais devem coincidir com a sigla da companhia responsável por executar o vôo.
- Em conversas com os usuários, convencionou-se que o termo *viagem* deve ser utilizado para representar uma execução de um vôo em uma data específica. Esse termo passará a ser utilizado a partir desta sprint visando facilitar a comunicação entre desenvolvedores, PO e usuários.
- O sistema deve realizar a programação de viagens de vôos ativos por um período de até 30 dias a partir da data atual, visando permitir a venda de passagens.
- O sistema deve permitir a compra antecipada de passagens por clientes, identificados pelo nome e sobrenome, além de um documento de identificação válido (RG ou Passaporte).
    - Toda passagem deve ter um aeroporto de origem e um aeroporto de destino;
    - O valor da tarifa deve ser também modelado, assim como a ocupação e disponibilidade de assentos em cada viagem;
    - Também deve ser permitida a escolha de um assento, opcional e sem custo para o passageiro;
    - O cliente pode optar por adquirir até no máximo 3 franquias de bagagens de até 23kg, cujo valor unitário deve ser definido por cada companhia aérea.
- Caso seja solicitada uma passagem entre dois aeroportos que não possuem vôos diretos, o sistema deve compor a passagem utilizando dois. No momento vamos permitir apenas uma conexão.
    - Nesse caso as informações de ambas as viagens devem ser armazenadas;
    - O valor total da passagem deve ser a soma dos vôos.

2.3. Sprint 3

Nesta sprint devem ser modeladas e implementadas classes para atendimento dos seguintes requisitos:

- O sistema deve permitir o cadastro de passageiros, que serão de dois tipos:
    - Passageiro Comum – que possui os campos abaixo e nenhum benefício especial:
        - Nome e sobrenome

        - Documento de identificação válido (RG ou Passaporte)
        - CPF (se brasileiro)
        - Nacionalidade
        - Data de Nascimento
        - E-mail
    - Passageiro Vip – que, além das informações do Passageiro Comum, possui os seguintes benefícios:
        - Alteração e cancelamento de vôo sem custo
        - 1 franquia de passagem gratuita por viagem
        - Franquias adicionais com desconto de 50%
- Os campos CPF, e-mail e Data de Nascimento dos passageiros devem ser validados.
- Passageiros devem estar associados às suas passagens.
- Deve ser possível acessar o histórico de vôos de um passageiro em ordem cronológica.
- O sistema deve permitir o check-in de passagens já adquiridas em um período compreendido entre 48h e 30 minutos do horário de partida do primeiro vôo. Em caso de não realização do check-in o sistema deve registrar o NO SHOW, que indica o não comparecimento do passageiro.
- Além do check-in o sistema deve permitir o registro de embarque (ou não) do passageiro. Caso este não embarque, também deve ser registrado NO SHOW.
- Todos os status do passageiro em um vôo devem ser passíveis de registro. São eles:
    - Passagem adquirida
    - Passagem cancelada
    - Check-in realizado
    - Embarque realizado
    - NO SHOW

2.4. Sprint 4

Nesta sprint devem ser atendidos os seguintes requisitos:

- O Passageiro Vip deve ter os seguintes atributos adicionados:

    - Número de registro e programa de milhagem favorito a ser cadastrado e mantido no sistema (Ex.: LatamPass, Smiles, Tudo Azul, etc.)

    - Categoria dentro do programa de milhagem- também cadastrável no sistema. (Ex.: Tudo Azul Diamante, Tudo Azul Safira, etc)

        - *Upgrade*: Cada categoria possui uma quantidade mínima de pontos acumulados para ser adquirida. Por exemplo a categoria Tudo Azul Diamante exige 5.000pts, enquanto que a Tudo Azul Safira exige 10.000

pts.

        - Caso a quantidade mínima de pontos acumulados nos últimos 12 meses de uma categoria não seja mantida, o passageiro vip tem um *downgrade* e retorna à categoria anterior.

    - Data de início e data de expiração na categoria atual

    - Quantidade de pontos acumulados

- O sistema deve permitir o cadastrado da tripulação de uma viagem, compreendendo um piloto e um co-piloto, além de no mínimo 2 comissários de bordo. Devem ser armazenadas os seguintes dados dos tripulantes:

    - Nome e sobrenome
    - Documento de identificação válido (RG ou Passaporte)
    - CPF (se brasileiro)
    - Nacionalidade
    - Data de Nascimento
    - E-mail
    - Número do Certificado de Habilitação Técnica (CHT)
    - Endereço completo
        - Logradouro, número, bairro, CEP, cidade, estado.
    - Companhia Aérea
    - Aeroporto base

- A cada vôo/viagem concluída, todos os passageiros Vip devem ter seus pontos acumulados com base no valor de milhagem concedido por aquela viagem.

2.5. Sprint 5

A fim de permitir a organização do transporte de tripulantes para o aeroporto, será necessário modelar no sistema o cadastro de veículos utilizados no deslocamento para atendimento de uma *viagem*. A empresa dispõe de uma frota de microônibus com capacidade variável entre eles. Deverá ser permitido o cadastro de veículos.

Cada veículo deve ter uma *rota* definida com base nos endereços dos tripulantes que deve transportar. Deve ser informada a sequência em que cada residência deve ser visitada pelo veículo.

Tendo como referência as coordenadas geográficas tanto dos tripulantes quanto do *aeroporto base*, o sistema deve ser capaz de calcular a distância percorrida por meio de uma integração com uma API de roteirização. **Esta funcionalidade não será implementada nesta sprint***. Como forma de simular o comportamento desse método, vamos utilizar a distância euclidiana para calcular a distância aproximada entre duas coordenadas geográficas (lat/long) em Km, conforme código abaixo*.

```
float calculaDistancia (float x1, float y1, float x2, float y2) {

        return 110.57 * sqrt( pow(x2-x1,2) + pow(y2-y1, 2) );

}
```

Baseado na distância total da rota, o tempo de percurso deve ser calculado, bem como o ***horário previsto para embarque*** de cada um dos tripulantes da rota, sabendo que os mesmos devem chegar ao aeroporto 90 minutos antes da decolagem da primeira viagem, e que o mesmo se desloca a uma velocidade média de 18 Km/h.

> * A integração com a API do Google Maps tanto para cálculo da rota (distância) quanto para georreferenciamento dos endereços (conversão de endereço em lat/long), será pontuada em 5 pts adicionais na nota final do trabalho.
>
> ** O código de exemplo está implementado em C++. Você deve fazer a implementação em PHP.

2.6. Sprint 6

Após a realização do *checkin* de uma passagem, é necessário que sejam gerados e armazenados os cartões de embarques de todas as viagens, contendo as seguintes informações:

- Nome e Sobrenome do passageiro
- Origem e destino do voo
- Horário de embarque e horário da viagem
- Assento

O horário de embarque é sempre 40 minutos antes do horário da decolagem.

Deve ser fornecido um método para realizar uma consulta/pesquisa por viagens tendo os parâmetros a seguir:

- Aeroporto de origem e destino
- Data da viagem
- Número de passageiros

Essa consulta deve retornar uma lista de possíveis viagens, inclusive com conexões, para atender o deslocamento total. O valor total de uma opção de trajeto deverá ser exibido, além dos horários de embarque das viagens. A disponibilidade de assentos deve ser validada.

O sistema deve permitir a alteração ou cancelamento de uma passagem conforme as seguintes regras:

- Em caso de solicitação de alteração, uma multa deve ser paga pelo passageiro. O valor da multa deve ser definido por viagem.
- Alterações em passagens somente podem ser feitas em até 4h horas antes do horário de partida da primeira viagem.
- Em caso de cancelamento, um valor de multa deve ser abatido do ressarcimento ao cliente.
- Passageiros VIP's não pagam multa por cancelamento ou alteração, apenas quando voarem nas companhias do seu programa de milhagem favorito. Nos demais casos, comportam-se como passageiros comuns.

2.7. **Sprint 7**

Nesta sprint devem ser modeladas e implementadas classes para atendimento dos seguintes requisitos:

- O sistema deve permitir o cadastro de usuários contendo login, senha e e-mail.
- Deve ser criado um método que permita o login de um usuário, que deverá ser realizado antes da chamada de qualquer outra funcionalidade do sistema. Ou seja, para utilizar qualquer funcionalidade do sistema, primeiro deve ser realizado o login de um usuário previamente cadastrado.
- Somente deve ser permitido um usuário logado no sistema por vez.
- Todas as ações realizadas pelo usuário no sistema devem ter um registro de log para posterior auditoria.
- Devem existir dois tipos de registro: logs de leitura e de escrita.
- Qualquer operação de acesso a dados do sistema deve gerar um registro de log de leitura contendo as seguintes informações:
    - Usuário da operação
    - Data e hora
    - Entidade acessada (*nome da classe*)
    - Informação acessada (*nome do atributo*)
- Todas as operações realizadas que alteram dados de entidades do sistema devem gerar um registro de um log de escrita. Este deve armazenar as seguintes informações:
    - Usuário da operação

- Data e hora
- Entidade alterada (*nome da classe*)
- Objeto serializado antes da operação (utilize a função *serialize()* do PHP)
- Objeto serializado após a operação

Com objetivo de tornar o projeto mais elegante, uma classe chamada *log* ***abstrata*** deve ser implementada, servindo de classe base para as demais classes da modelagem.

Bom trabalho!