Trabalho Final – Spotflix

Programação para Design – 2018.1

Prof. Anderson Lemos

1. Objetivo

Desenvolver um sistema chamado ***Spotflix***. Esse sistema deverá gerenciar a venda de filmes, séries, músicas e podcasts, com interface de iteração com o usuário (não necessariamente gráfica, pode ser em console). O sistema será capaz de vender os produtos aos clientes cadastrados, vender planos, listar produtos, buscar produtos, cadastrar novos produtos e permitir três tipos de usuário: o cliente, o gerente e o operador do sistema.

2. Classes

2.1 – Produtos

Os produtos disponíveis no **SpotFlix** são Vídeos, Áudios ou Pacotes. Os Vídeos podem ser Filmes ou Séries. Os Áudios podem ser Músicas ou Podcasts.

Um Produto é uma Classe Abstrata que deve ter os seguintes atributos: *codigo* (string), *nome* (string), *genero* (string), *preco* (number). Além disso, um Produto declara um método abstrato chamado **executar() : void**.

Um Vídeo **é-um** Produto e tem os seguintes atributos: *anoLancamento* (number) e *faixaEtaria* (number). Vídeo também é uma classe abstrata.

Filmes e Séries **são** Vídeos e são classes concretas. Um Filme tem uma *duracao* (number, em minutos). Uma Série tem um *numeroDeEpisodios* (number) e uma *duracaoMediaDeEpisodio* (number, em minutos).

Áudio **é-um** Produto. Assim como Vídeo, também é uma classe Abstrata. Tem como atributos *autor* (string) e *duracao* (number).

Músicas e Poscasts são filhos de Áudio, e são classes concretas. Música tem como atributo *album* (string) e Poscast tem como atributo *tematica* (string).

Um Pacote **é-um** Produto e é uma classe concreta. Tem como atributos *produtos* (Array<Produto>), uma *validade* (***Data***, faça uma classe Data com *dia*, *mes* e *ano*, todos do tipo number) e um *precoMensal* (number). Como ele é um Produto, também terá o atributo *preco* de Produto (sem precisar sem escrito explicitamente). Assim, o *preco* do Pacote é a soma dos preços de cada produto em *produtos*. O *precoMensal* de um Pacote é 15% do *preco* do Pacote.

2.2 - Atores

Os atores envolvidos no sistema são: o **Cliente**, o **Gerente**, e o **Operador do Sistema**. Iremos criar uma classe **Usuário** para generalizar seus atributos.

A Classe **Usuário** é abstrata e deverá ter os seguintes atributos: *nome* (string), *email* (string) e *senha* (string). A Classe **Funcionário** é filha de Usuário, também é abstrata e deve ter os atributos *matricula* (string) e *salario* (number). A classe **Cliente** é filha de Usuário, mas **Gerente** e **OperadorDeSistema** são filhas de Funcionário, e as três são classes concretas. **Cliente** tem um *endereco* (string), uma *data de nascimento* (Data), uma lista de *produtos* (Array<Produto>) e um *cartão de crédito* (**CartaoDeCredito**, faça uma classe **CartaoDeCredito** com *numero* (string), *validade* (Data), *codigoCCV* (number) e *limite* (number)).

Um **Gerente** deve ter as seguintes ações:

- Adicionar Operadores.
- Remover Operadores.
- Listar os Clientes, Produtos e Operadores.
- Procurar Cliente, Produto e Operadore por *email* ou *codigo*.

Um **Operador de Sistema** deve ter as seguintes ações:

- Adicionar Produtos.
- Remover Produtos.
- Listar os Clientes e Produtos.
- Procurar Cliente e Produto por *email* ou *codigo*.

Um **Cliente** deve ter as seguintes ações:

- Se cadastrar no sistema.
- Comprar um produto.
- Executar uma mídia.

2.3 – Repositórios

Os repositórios servem para armazenar a nossa Base de Dados **em memória**. Vão existir dois tipos de repositórios: o de Produtos à venda e o de Usuários. O aluno deve criar uma classe para cada Repositório e armazenar os objetos da seguinte forma:

1. No caso dos Usuários, usar um **Map** onde a chave é o *e-mail* do usuário e o valor é o objeto em si.
2. No caso de Produtos, também usar um **Map** onde a chave é o *código* do Produto e o valor é o objeto do tipo produto.

Os repositórios terão funções em comum que são as de adicionar, remover e retornar um objeto em sua coleção. Fica a cargo do aluno o nome destas funções. Usem padrões de convenção de código.

3. A Interface

Abaixo são mostradas sugestões para implementação do sistema com interface no console, abusando do readline.question. Veja o Exemplo:

**** Bem Vindo ****
1 – Logar
2 – Cadastrar-se como cliente
3 – Sair
Digite a opção: _<ENTER>

Se o usuário digitar 2, devem ser recebidas via readline.question as informações do cliente para que ele possa ser cadastrado. Não deve ser possível cadastrar um usuário sem todos os seus atributos (se algum campo for vazio, não deverá ser possível cadastrar). Não deve ser possível cadastrar um usuário se seu e-mail já estiver cadastrado.

Se o usuário digitar 1, ele deve ser levado a tela de Login:

****Faça seu Login****
Digite seu login:_ <ENTER>
Digite sua senha:_ <ENTER>

O sistema irá verificar se o login e senha correspondem com um Usuário (do repositório de

usuários), e se existir, verificar se é Cliente, Gerente ou Operador (com o **instanceof**). Caso dê tudo certo, as telas das opções serão mostradas para o perfil desejado:

Para o Gerente:

****Olá Gerente <Nome do Gerente>*****
1 – Cadastrar Operador
2 – Remover Operador
3 – Listar Produtos
4 – Listar Clientes
5 – Listar Operadores
6 – Procurar Produto
7 – Procurar Cliente
8 – Procurar Operador
9 – Sair
Digite a opção:_<ENTER>

Para o Operador

****Olá Operador <Nome do Operador>****
1 – Cadastrar Produto
2 – Remover Produto
3 – Listar Produtos
4 – Listar Clientes
5 – Procurar Produto
6 – Procurar Cliente
7 – Sair
Digite a opção:_<ENTER>

Para o Cliente

****Olá <Nome do Cliente>****
1 – Listar Produtos da Loja
2 – Listar meus Produtos
3 – Comprar Produto
4 – Play
5 – Sair
Digite a opção:_<ENTER>

Gerente:

Na tela de opção 1 pro Gerente, o sistema irá receber, via readline.question, todos os dados do Operador e inseri-lo no repositório de Usuários. Não deve ser possível cadastrar um usuário sem todos os seus atributos (se algum campo for vazio, não deverá ser possível cadastrar). Não deve ser possível cadastrar um usuário se seu e-mail já estiver cadastrado. Na tela de opção 2 do Gerente, é lido com readline.question uma matrícula de Operador e apaga o Operador com essa matrícula. Nas telas de opção 3, 4 e 5 pro Gerente, serão mostrados os dados referentes ao objeto escolhido um por um. Use um laço e imprima na tela as informações de cada objeto. Nas telas de 6, 7 e 8 pro Gerente, apenas peça o e-mail ou o código e mostre o Produto ou Usuário correspondente caso ele **exista.** Para isto, o sistema deve acessar os Repositórios. Mas quem cadastra o gerente? Você mesmo! Crie no código um usuário padrão que é do tipo Gerente com o e-mail: ***gerente@admin.com*** e senha ***admin***, e insira no repositório de usuários, sempre que o programa for iniciado. Assim, sempre haverá pelo menos um gerente no sistema.

Operador:

Na tela de opção 1 pro Operador, o sistema irá receber, via readline.question, todos os dados do Produto e inseri-lo no repositório de Produtos. Na tela de opção 2 do Operador, é lido com

readline.question um código do Produto e apaga o Produto com esse código.
Nas telas de opção 3 e 4 pro Operador, serão mostrados os dados referentes ao objeto escolhido um por um. Use um laço e imprima na tela as informações de cada objeto. Nas telas de 5 e 6 pro Operador, apenas peça o e-mail ou o código e mostre o Produto ou Usuário correspondente caso ele **exista.** Para isto, o sistema deve acessar os Repositórios.

Cliente:

Na tela de opção 1 pro Cliente, serão mostrados os dados referentes aos produtos que estão no sistema e que o cliente ainda não comprou, um por um. Use o repositório de produtos. Na tela de opção 2 pro Cliente, serão mostrados os dados referentes aos produtos que o cliente comprou, um por um. Produtos que estiverem em Pacotes vencidos, não devem ser mostrados. Nas duas primeiras opções, use um laço e imprima na tela as informações de cada objeto. Na tela de opção 3 do Cliente, o sistema irá receber, via readline.question, o código do produto que se quer comprar, e se há limite no cartão de crédito do cliente, compra aquele produto e adiciona nos produtos do cliente. Se o produto for um pacote, deverá ser possível escolher entre comprar o pacote ou fazer uma assinatura mensal. Caso seja uma assinatura mensal, a data de validade deverá ser dita, caso contrário, a data de validade deverá ser "infinita" (utilize o valor *null* na data de validade para representar isso). Se não houver limite no cartão de crédito, deverá ser lançada uma exceção ***NoLimitCreditCardError*** (crie essa classe, lance quando necessário e trate a exceção). Na tela de opção 4 do Cliente, o sistema irá receber, via readline.question, o código do produto que se quer executar, e o produto será executado chamando o método executar do produto (aqui, cada produto deverá executar de uma forma, através do polimorfismo). No método executar do produto, só imprima "executando produto <nome do produto> que é um(a) <tipo do produto>", onde <tipo do produto> é se é uma série, um filme, uma música ou um podcast. Obviamente, só pode ser possível executar produtos que o cliente comprou.

4. Considerações Finais

1. As equipes serão formadas por no máximo **2 (DOIS)** alunos.
2. As notas serão diferenciadas por aluno no momento da apresentação do trabalho em sala.
3. A nota do trabalho fará parte da nota final da disciplina, valendo um terço da nota total.
4. O trabalho deverá ser implementado em TypeScript.

5. Dicas

1. Sejam criativos! Façam um sistema que vocês gostariam de usar. Funcionalidades a mais serão muito bem vistas, por exemplo:
    - Salvar os dados em arquivos para não serem perdidos quando o programa for fechado e serem carregados quando o programa for aberto.
    - Interface gráfica.
    - Algo muito ousado, como realmente executar a mídia comprada.
2. A interpretação do trabalho é uma forma de avaliação. Se você acha que algo não ficou bem explicado, tente entender como seria a maneira certa de fazê-lo e resolva o problema. Esse é um problema que vocês vão ter que lidar diariamente no mercado de trabalho.
3. Deixem a parte mais difícil por último, concentrem-se em implementar as classes de acordo com a orientação a objetos. As funcionalidade podem ser implementadas depois.
4. Procurem dividir o trabalho entre os integrantes.
5. Sintam-se livres para implementar da forma que for mais conveniente. No entanto, respeitem a O.O e convenções de código. Tornem o programa funcional.
6. Façam o trabalho por partes: leiam com cuidado este documento e vão implementando à medida que forem entendendo.
7. Tirem dúvidas com o professor, entre vocês e pela Internet.
8. Utilizem os conceitos vistos em sala: herança, polimorfismo, abstração, encapsulamento, tratamento de erros, coleções, classes abstratas, etc.