Exercício TravelApp

1. Crie um aplicativo que gerencia a venda de passagens de uma companhia aérea. Para isso, deveremos ter uma função construtora para o objeto LinhaAerea que possui um código identificador, uma cidade origem, uma cidade destino, uma data de partida, uma data de chegada e um valor de passagem. Tal objeto possui também uma lista de assentos, identificados por um identificador numérico único para o voo e uma variável que indica se o respectivo assento está ou não disponível para venda. Essa função construtora deve receber como parâmetros de entrada os valores necessários para os atributos definidos acima (código identificador, uma cidade origem, uma cidade destino, uma data de partida, uma data de chegada, um valor de passagem e a quantidade de assentos para esse voo). Para as datas de partida e chegada verifique o uso do objeto Date.

2. Para a função construtora acima, crie os seguintes métodos:

   a. getValor(): Retorna o valor da passagem;

   b. getDestino(): Retorna o destino do vôo;

   c. getOrigem(): Retorna a origem do vôo;

   d. getPartida(): Retorna a data de partida;

   e. getChegada(): Retorna a data de chegada;

   f. getAssentos(): Retorna o status de todos os assentos com o uso do console.log(), na forma:

      identificador_do_assento – disponibilidade_do_assento(livre ou ocupado)

   g. getAssentosLivres(): Retorna uma lista de identificadores de assentos que estão livres para compra.

   h. setValor(valor): Altera o valor da passagem aérea de acordo com o valor passado como argumento;

   i. atrasar(minutos): altera a data de partida atrasando a mesma de acordo com o número de minutos passados como argumento desse método. Note que um atraso na partida deve representar um atraso na chegada de tempo igual a este atraso;

   j. comprar(id): Compra uma passagem para o assento identificado por id. Note que ao comprar um assento, o mesmo deve ter sua disponibilidade recebendo o valor false, retornando o número do assento caso este possa ser comprado (está disponível) ou false caso não seja possível comprar esse assento (já tenha sido ocupado anteriormente)

3. Altere o método comprar() para que, caso não seja repassado nenhum id, seja comprado o primeiro assento livre disponível na lista.

4. Crie uma validação na entrada do atributo valor de modo que o mesmo deva ser sempre maior que 0. Caso seja inserido um valor menor ou igual a 0 deverá ser atribuído o valor 1,00 e uma mensagem no console deverá ser gerada informando essa situação.

5. Crie uma validação de modo a garantir que a data de chegada não possa ser anterior a data de partida. Caso isso ocorra, deverá ser gerada uma mensagem no indicando essa ocorrência, e a data de chegada deverá ser alterada para 24 horas após a data de partida.

6. Crie uma validação que garanta que a data de partida não possa ser anterior a data atual do sistema. Para isso, pesquise a chamada new Date();

Exercício Música para todos os gostos

7. Crie a função construtora Musica, que possui como atributos o nome, o artista, o álbum, o tempo de duração, o gênero, a posição da música e o atributo playing, que indica se a música está sendo tocada nesse instante. Uma música deve ser criada atribuindo o valor a todos esses atributos, incluindo false para o atributo playing.

8. Para a função construtora acima, crie os métodos play e pause. O método play deve atribuir true ao playing e apresentar uma mensagem console.log com a frase “Tocando a música nome_da_música”. Por sua vez, o botão pause deve atribuir false ao atributo playing da música

9. Crie uma função construtora para gerar objetos Playlist. O objeto Playlist é formado por um nome, um array composto por objetos Musica e um atributo musicaAtual, que nada mais é do que um indicador que mostra o índice no array de objetos Musica com a música a ser tocada nessa playlist. Um objeto playlist também deve possuir os seguintes métodos:

    a. mostraPlaylist(), que apresenta, com o auxílio da console.log, uma lista numerada com os nomes das Músicas que compõe essa playlist

    b. adicionaMusica(musica), que adiciona no final do array de músicas um novo objeto Musica, passado como parâmetro desse método.

    c. removeMusica(indice), que remove a música apresentada nesse índice.

    d. getMusicaCorrente(), que retorna o índice da música atualmente indicada como música atual da playlist.

    e. setMusicaCorrente(indice), que indica o índice da música que passa a ser tratada como música atual da playlist.

    Observações:

    a. Note que, para remover uma música da playlist, é esperado que o usuário visualize as músicas dessa playlist e então repasse o número da música indicado por essa função.

b. É esperado que o atributo musicaAtual receba algum tipo de tratamento caso a música setada como musicaAtual seja removida da lista.

c. Quando a música corrente é alterada, antes de efetivar essa troca de música, caso a música corrente esteja com o atributo playing como true, esse deve ser passado para false. Da mesma forma, a nova música corrente deve ter o atributo playing setado para true. (Não esqueça de utilizar os métodos definidos anteriormente para a questão)

10. Para os objetos Playlist definidos anteriormente, crie os controles de play(), pause(), next() e previous(), que devem simular o controle de músicas a serem tocadas nessa playlist.

11. Crie o objeto player que possui uma lista de playlists. Para isso use os objetos Playlist e Musica definidos anteriormente. Tal objeto deve ser capaz de gerenciar múltiplas playlists, permitindo que o usuário adicione, selecione ou remova uma playlist, bem como coordene a execução das músicas dessa playlist. Note que uma mesma música poderá estar presente em mais de uma playlist.