UPskill – Java+.NET

Programação Orientada a Objetos – Tratamento de exceções

UPskill
Digital Skills & Jobs

P.PORTO

EXERCÍCIO SISTEMA DE FATURAÇÃO PARA UMA EMPRESA DE TELECOMUNICAÇÕES

1. A ficha de trabalho consiste em desenvolver um sistema de faturação para uma empresa de telecomunicações que oferece vários serviços, como Internet, TV por cabo e telefone. O sistema precisa lidar com diferentes tipos de clientes, cada um com planos e detalhes de cobrança específicos. O objetivo é implementar um programa em C# que incorpore o tratamento de exceções e aproveitando os conhecimentos sobre os princípios de abstração, polimorfismo e agregação/composição adquiridos. Considere que:
    a. A empresa tem clientes que podem ser: Regular ou Premium (`CustomerType`[1]). Esta caraterização será utilizada para o cálculo do valor de faturação.
    b. Os clientes devem implementar a interface `ICustomer`[1].
    c. Os clientes podem ser clientes de televisão (`TVCustomer`), internet (`InternetCustomer`) ou telefone (`PhoneCustomer`).
    d. Os `InternetCustomer` consideram:
        i. Os clientes são faturados pelo seu consumo de dados.
        ii. O consumo de dados pode ser atualizado, devendo lançar uma exceção do tipo `ArgumentOutOfRangeException` caso o valor seja negativo.
        iii. O valor por unidade de consumo de dados é 0.1€.
        iv. O valor de faturação deve considerar um desconto de 20% caso o cliente seja premium.
    e. Os PhoneCustomer consideram:
        i. Os clientes são faturados pelo seu consumo de minutos de chamadas e mensagens enviadas.
        ii. O consumo de minutos e mensagens pode ser atualizado, devendo lançar uma exceção do tipo `ArgumentOutOfRangeException` caso o valor seja negativo.
        iii. O valor por unidade de consumo de minutos é 0.35€ e de mensagens 0.25€.

[1] Disponível no final deste documento.

iv. O valor de faturação deve considerar um desconto de 25% caso o cliente seja premium e 10% caso seja regular.

f. Os TVCustomer consideram:

i. Os clientes são faturados pelo número de canais subscritos.

ii. O número máximo de canais é de 100.

iii. Os clientes aderem a um pacote que pode ser: Básico, Standard ou Premium[2]. Cada pacote assume um mínimo de canais garantidos, respetivamente, 25, 50 e 100. Por defeito, o cliente recebe o número de canais que o pacote à qual subscreve garante.

iv. O número de canais à qual um cliente subscreve pode ser alterado, devendo lançar uma exceção do tipo `ArgumentOutOfRangeException` caso o valor seja negativo. No caso de o número de canais ser inferior ao garantido pelo seu pacote, uma `InvalidLowerNumberOfChannelThanDefaultException` deverá ser lançada.

v. O pacote à qual um cliente subscreve pode ser alterado, devendo garantir que o número mínimo de canais que o pacote garante. Se o cliente tiver mais canais que os garantidos pelo pacote, este valor não pode ser alterado pela alteração de pacote.

vi. O valor por canal de dados é 0.5€.

vii. O valor de faturação deve considerar um desconto de 10% caso o cliente seja premium.

g. Os clientes `InternetCustomer` e `PhoneCustomer` devem implementar a interface `IResetable`[1] que permite que os seus dados de consumo sejam reiniciados a `0`.

h. O sistema `BillingSystem` deve conter os seguintes métodos:

i. Método[3] `ChangeData` que recebe como argumento um tipo de cliente e efetua a leitura dos seus dados de consumo. Este método deve tratar as exceções lançadas pelas classes, informando o utilizador dos erros e requisitando os valores até que seja introduzido um valor correto.

---

[2] Considere o use de enumerações.
[3] Pode precisar mais do que um dado existirem vários tipos de clientes (i.e., internet, TV e telefone).

ii. Método `AddCustomer(ICustomer customer)` que adiciona um cliente à lista de listes do sistema de faturação. Este método lança as seguintes exceções:
    1. `ArgumentNullException` - quando o cliente fornecido é nulo.
    2. `ArgumentException` - quando o cliente já existe no sistema de faturação.
iii. Método `DisplayAllCustomers` que mostra todas as informações dos clientes.
iv. Método `CalculateTotalRevenue` retorna o total de faturação considerando todos os clientes do sistema.
v. Método `ResetAll` que reinicia os dados de consumo dos clientes que implementam a interface `IResetable` e retorna quantos foram reiniciados.

2. Teste de modo abrangente o código desenvolvido.
3. De modo a verificar a sua implementação e os conhecimentos adquiridos, analise o código disponibilizado. Preste atenção na estrutura, nas declarações de classe e nos métodos.

Notas para os estudantes: Reserve um tempo para ler e entender cuidadosamente cada parte do código. Sinta-se à vontade para experimentar modificações ou funcionalidades adicionais para aprofundar a sua compreensão. Esta ficha de trabalho foi concebida para encorajar os alunos a trabalhar ativamente com o código fornecido, identificar conceitos-chave e aplicar os seus conhecimentos para alargar o projeto. O seu objetivo é reforçar a compreensão das classes abstratas, do polimorfismo e da herança num contexto prático.

Código:

```
/// <summary>
/// Represents the distinct characterizations of customers in the system.
/// </summary>
public enum CustomerType {
    Regular,
    Premium
}
```

```
/// <summary>
/// Interface for customers in the billing system.
/// </summary>
public interface ICustomer {

     /// <summary>
     /// Gets or sets the name of the customer.
```

```
    /// </summary>
    string Name { get; set; }

    /// <summary>
    /// Gets or sets the type of the customer.
    /// </summary>
    CustomerType Type { get; set; }

    /// <summary>
    /// Calculates the bill for the customer.
    /// </summary>
    /// <returns>The calculated bill amount.</returns>
    double CalculateBill();

    /// <summary>
    /// Displays information about the customer.
    /// </summary>
    void DisplayInfo();

    /// <summary>
    /// Determines whether the specified object is equal to the current customer.
    /// </summary>
    /// <param name="obj">The object to compare with the current customer.</param>
    /// <returns>True if the name and customer type are equal to the
    /// current customer; otherwise, false.</returns>
    bool Equals(object? obj);

    /// <summary>
    /// Serves as the default hash function.
    /// </summary>
    /// <returns>A hash code for the current customer.</returns>
    int GetHashCode();
}
```

```
/// <summary>
/// Represents an interface for objects that can be reseted.
/// </summary>
public interface IResetable {
    /// <summary>
    /// Resets the client consumption.
    /// </summary>
    void reset();
}
```