**Teoria:**

1. **Defina Classes, atributos e métodos**

Uma classe é um modelo ou gabarito para a definição de objetos.  
Então um objeto não existe sem a definição de uma classe.  
Através da definição de uma classe, podemos descrever:

* **quais propriedades e/ou atributos o objeto vai ter;**
* **qual o comportamento que o objeto da classe terá;**

A classe seria então o modelo a partir do qual criamos os objetos da classe e fazemos isso criando instâncias da classe.  
Criar um objeto de uma classe significa criar uma instância da classe.

1. **O que significa encapsulamento**O encapsulamento é o processo de ocultar ou esconder os membros de uma classe do acesso exterior usando modificadores de acesso.

O encapsulamento fornece uma maneira de preservar a integridade do estado dos dados. Ao invés de definir campos públicos devemos definir campos de dados privados.  
  
A classe bem encapsulada deve ocultar seus dados e os detalhes de implementação do mundo exterior. Isso é denominado programação *caixa preta*. Usando o encapsulamento, a implementação do método pode ser alterada pelo autor da classe sem quebrar qualquer código existente fazendo uso dela.

1. **Qual a função dos operadores set e get**

**Set => Atribuição de conteúdo**

**Get => Obtenção de conteúdo**

1. **Explique o Principio SRP do SOLID**

**“Uma classe só deve ter um único motivo para mudar.”**

Uma classe deve somente uma responsabilidade, deve fazê-la bem e deve fazer somente ela.   
Se uma classe tem mais de um motivo para ser alterada, ela não segue este princípio.   
Se ao se referir a uma determinada classe, você diz, por exemplo: “minha classe tem as informações

do cliente e salva o mesmo no banco de dados” perceba que o “e” na frase, indica mais de

uma responsabilidade, ferindo assim o SRP( single responsability principle ).

Aplicar o princípio da responsabilidade única é importante para uma arquitetura madura e

sustentável. Quando começamos a dar valor a princípios como este, vemos que estamos

amadurecendo e fazendo melhor o que gostamos de fazer: software.

“Cada responsabilidade deve ser codificada em uma classe separada, pois cada responsabilidade é eixo para mudança.”

1. **Qual é a função da classe Exception?**As exceções (Exceptions) são mecanismos primários para comunicar condições de erros.  
   As exceções possuem um grande poder e um grande poder trás também grandes responsabilidades.
2. **Defina para que serve cada palavra reservada da linguagem: **

**Namespace (Espaço de nomes)  
Namespaces** **organizam os objetos em um assembly;** assim , um assembly pode conter um ou mais *namespaces* , e estes namespaces pode conter um ou mais *namespaces.*   
Desta forma os **Namespaces** **evitam a ambiguidade e organizam referências** quando são usados grande grupos de objetos como as libraries de classes.

**private**

O modificador de acesso **Private** impede que uma classe exponha suas variáveis de membros e funções de membros a outras funções e objetos (em outro contexto lógico – um IF por exemplo). **Um membro** **private** **não poderia** ser acessado de fora da classe.

**public**O modificador de acesso **Public** permite que uma classe exponha suas variáveis de membros e funções de membros a outras funções e objetos.  
Qualquer membro público pode ser acessado de fora da classe.

**sealed**Trata-se de um modificador de acesso de classe, que garante que a mesma não poderá ser herdada por outra classe.

**Using  
Forma 1) Como uma diretiva**A **diretiva** using tem três usos:

* Para permitir o uso de tipos em um namespace para que você não precise qualificar o uso de um tipo nesse namespace:

using System.Text;

* Para permitir que você acesse membros estáticos e tipos aninhados de um tipo sem precisar qualificar o acesso com o nome do tipo.

using static System.Math;

* Para criar um alias para um namespace ou um tipo. Isso é chamado de uma *diretiva alias de using*.  
  using Project = PC.MyCompany.Project;

**Forma 2) Como uma instrução**using (Font font1 = new Font("Arial", 10.0f))

{byte charset = font1.GdiCharSet;}

**void**

Indica que um método não retorna valor ao em sua execução

**try/catch**É a estrutura de controle de execução, que permite a captura de exceções caso ocorram.   
 exemplo abaixo explica o que ocorre dentro do exemplo citado acima sobre **using (Forma 2)**:

{

Font font1 = new Font("Arial", 10.0f);

try

{

byte charset = font1.GdiCharSet;

}

finally

{

if (font1 != null) ((IDisposable)font1).Dispose();

}

}

<https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/using-statement>

**set/get**

**Set => Atribuição de conteúdo**

**Get => Obtenção de conteúdo**