UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE PROCESSAMENTO DE ENERGIA ELÉTRICA

Abstração e encapsulamento

Curso de Engenharia de Controle e Automação

DPEE1090 - Programação orientada a objetos para automação

Prof. Renan Duarte

1º semestre de 2024

Sumário

Abstração e encapsulamento

- Conceito de abstração e sua aplicação em classes
- Conceito de Interface e implementação
- Exploração do conceito de encapsulamento
- Utilização de modificadores de acesso em atributos e métodos
- Implementação de *getters, setters* e propriedades

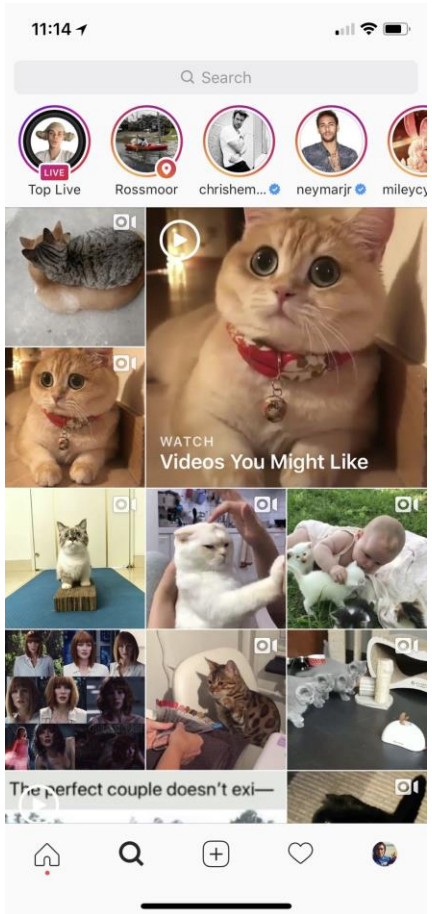
Abstração

Pense uma atividade que você faz todo dia: Checar o Instagram

Ao fazer isso, você se preocupa com qual servidor do Meta está fornecendo os dados para você?

Quais os ajustes precisam ser feitos ao layout do app para melhor se adaptar a tela do seu celular?

Qual o próximo conteúdo que precisa ser carregado do servidor à medida que você vê o feed?



Abstração

Pense uma atividade que você faz todo dia: Checar o Instagram

Do seu ponto de vista, como usuário final, os únicos dados que importam são o número de curtidas na sua última foto e quantas pessoas viram seus stories.

Todos os demais detalhes inerentes ao funcionamento do Instagram são irrelevantes para você.

Você naturalmente abstrai essas informações.

Abstração

Definição

Abstração é o processo utilizado na análise de uma situação real, na qual observa-se a realidade e busca-se determinar os aspectos e fenômenos considerados essenciais, excluindo-se todos os outros considerados irrelevantes ou secundários.

Na programação orientada a objetos, abstração é o princípio fundamental que envolve a criação de modelos simplificados de sistemas complexos, representando apenas as características essenciais e relevantes para o contexto atual, enquanto omite detalhes internos e desnecessários.

Abstração

Níveis de abstração

Quanto mais detalhes são omitidos, mais se está utilizando a abstração, ou seja, maior é o nível de abstração em relação a determinado problema ou conceito.

No caso do Instagram, se você é um desenvolvedor do Meta, os detalhes importantes passam a ser outros. Os comportamentos internos do sistema são relevantes para você. Assim, seu nível de abstração com relação ao funcionamento do Instagram é maior.

Abstração

Níveis de abstração

Por exemplo, considerando uma viagem de avião, o passageiro tem um nível de abstração muito mais baixo do que o piloto no que diz respeito ao funcionamento do processo:

- Horário de partida
- Número do assento
- Localização do banheiro

- Modelo do avião
- Nível de combustível
- Temperatura
- Clima
- Quantidade de carga
- Procedimentos padrão
- Cheklists

...

Abstração

Níveis de abstração

O nível de abstração também é utilizado como parâmetro para classificar as linguagens de programação:

- **Linguagem de baixo nível** → Baixo nível de abstração. Sintaxe muito diferente da linguagem humana e muito mais próxima da linguagem de máquina

- **Linguagem de alto nível** → Alto nível de abstração. Sintaxe próxima da linguagem humana e mais distante da linguagem de máquina (programador não precisa conhecer características específicas do hardware como instruções e registradores → as características foram abstraídas)

Abstração

Níveis de abstração

Exemplo:

Assembly	C

.MODEL SMALL
.STACK
.CODE
 	
mov ah,2a
mov dl, ‘A’
int 0x21
mov ax, 0x4c00
int 0x21
 	
.END
	#include <stdio.h>
	
	int main ()
	{
	printf(“A”);
	return 0;
	}

Abstração

Uso da abstração na POO

- Exibir apenas o essencial para o mundo externo
- Permite que se trabalhe de forma incremental → Desacoplamento
- Desenvolvimento de software colaborativo
- Permite trabalhar com conceitos em um nível mais alto sem se preocupar com os detalhes de implementação
- **Sempre que possível, deve-se depender da interface e não da implementação**

Interface e implementação

- Interface é a forma como seções do código interagem
- Implementação se refere a como estes métodos são escritos

Abstração

Interface e implementação

Por exemplo, considerando a calculadora desenvolvida na última aula para o cálculo de uma circunferência para um dado raio:

```
public static double Circunferencia(double raio)
{
   return 2 * Pi * raio;
}
```

Interface

Implementação

Se modificarmos a implementação de $2\pi r$ para $\int_0^{2\pi} r d\theta$ por exemplo, o usuário final que usa a interface Circunferencia(double raio) não notaria nenhuma modificação no comportamento do código

Encapsulamento

Definição

É o princípio que consiste em esconder detalhes de implementação de um componentes, expondo apenas operações seguras e que o mantenham em um estado consistente.

Diferenças entre encapsulamento e abstração

- Abstração esconde complexidade, fornecendo uma imagem mais abstrata, enquanto o encapsulamento oculta o trabalho interno
- A abstração está focada principalmente no **que** deve ser feito, enquanto o encapsulamento está focado em **como** deve ser feito.
- A abstração resolve problemas no nível de design, enquanto o encapsulamento resolve problemas no nível de implementação

Encapsulamento

Uso na POO

Um objeto deve estar sempre em um estado consistente e a própria classe que o criou deve garantir isso

Códigos externos à nossa classe não devem ser capazes de alterar o estado interno dos objetos criados a partir dessa classe.

Data hidding

- Permite o controle de acesso aos dados
- Cada objeto controla seu próprio estado
- Evita que o programa assuma estados imprevistos
- Permite fazer validação de dados

Encapsulamento

https://bit.ly/3JnxHrj

Exemplo 2 – Implementação sem encapsulamento

Em uma conta bancária é preciso fazer um depósito para aumentar o valor do saldo disponível. Para retirar dinheiro da conta, pode-se fazer um saque, mas somente se o valor a ser retirado é menor ou igual ao saldo disponível.

O que acontece se o atributo “saldo” tiver acesso público?

Nessa situação, pode-se aumentar o saldo da conta sem fazer um depósito ou definir o saldo como negativo, fazendo com que o estado consistente do nosso objeto “conta” seja perdido.

Encapsulamento

Modificadores de acesso

Para garantir esse comportamento, podemos utilizar modificadores de acesso. Os principais que veremos nesse curso são:

- *Public* → Não impõe restrições ao dado da classe, permitindo que ele seja acessado e modificado diretamente onde e quando necessário
- *Private* → Permite o acesso ao dado apenas através da própria classe (por exemplo, uma função da classe)
- *Protected* → O dado é acessível dentro de sua classe e por instâncias derivadas dessa classe (subclasses)

https://learn.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/access-modifiers

Encapsulamento

https://bit.ly/3VXZ96H

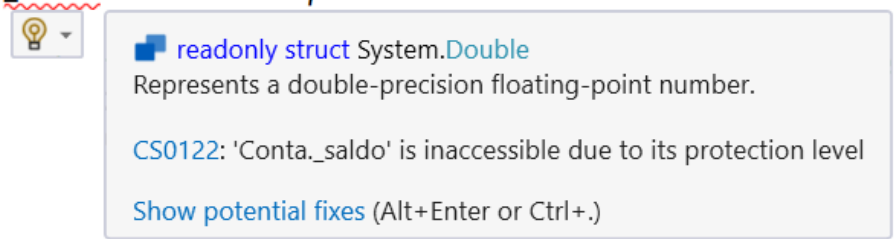
Exemplo 3 – Implementação com encapsulamento

Vamos modificar o exemplo anterior para proteger o atributo "saldo" de forma que ele só possa ser alterado através dos métodos "Deposito" e "Saque":

```
private double _saldo; // Convenção: _camelCase p/ privados
```

Agora, a tentativa de alteração do saldo fora da classe resulta em um erro:

```
// Erro pois o atribulo "saldo" é privado
conta._saldo = 100000000;
```



Encapsulamento

Getters e *setters*

Quando definimos um atributo como privado, o acesso ao seu conteúdo fora da classe também fica proibido:

```
// Inválido pois _saldo "não existe" fora da classe
Console.WriteLine(conta._saldo);
conta._saldo = 100000000;
```

Para ler o conteúdo de um atributo privado precisamos de um método que nos retorne seu valor → *Getter*

Para alterar o valor de um atributo privado precisamos de um método que faça isso dentro da classe → *Setter*

Encapsulamento

Exemplo 4 – *Getters* e *Setters*

Inclusão de *getter* e *setter* para o atributo "_saldo"

```
// Getter para o valor do saldo
public double GetSaldo()
{
   return _saldo;
}

// Setter para o valor do saldo (apenas para ver sintaxe)
public void SetSaldo(double valor)
{
  _saldo = valor;
}
```

Encapsulamento

Getters e *setters*

Pode-se ainda usar os getters e setters para fazer formatação e validação dos dados.

Exemplos:

```
// Getter para o valor do saldo em forma de string
public string GetSaldo()
{
   return _saldo.ToString("F2");
}
// Validação - valor deve ser maior ou igual a zero
public void SetSaldo(double valor)
{
   if (valor > = 0)
      _saldo = valor;
}
```

Encapsulamento

Getters e *setters*

Não é necessário implementar ambos *getters* e *setters*. Se apenas a visualização do dado precisa ser feita de forma externa, somente o *getter* é necessário por exemplo.

De forma análoga, se apenas a alteração do dado precisa ser feita de forma externa, somente o *setter* precisa ser implementado.

Encapsulamento

Propriedades

Existe uma sintaxe alternativa mais flexível para definir o acesso de atributos de uma classe. É a chamada "propriedade" ou "*propertie*".

As propriedades podem ser usadas como se fossem membros de dados públicos, mas são **métodos** especiais chamados acessadores.

https://learn.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/properties

Encapsulamento

Exemplo 5 - Propriedades

A sintaxe de uma propriedade encapsula o *getter* e o *setter* e é acessada da mesma forma que um atributo público:

```
// Propriedade Saldo
public double Saldo {
   // Getter da propriedade
   Get {
      return _saldo;
   }

   // Setter da propriedade
   set
   {
      if (value >= 0)
         _saldo =  value;
   }
}
```

```
// Alteração do saldo
conta.Saldo = 1000;

// Exibição do saldo
Console.WriteLine(conta.Saldo);
```

Encapsulamento

https://bit.ly/4d36c3Q

Propriedades autoimplementadas

São uma forma simplificada de declarar propriedades que não necessitam de lógicas particulares para as operações *get* e *set*.

Com elas, eliminados a necessidade de declarar um atributo privado e métodos getters e setters. Todos esses parâmetros são encapsulados em uma única entidade, acessada da mesma forma que se acessa um atributo público

```
// Propriedade autodeimplementada Saldo
// Apenas get é acessível fora da classe.
// Alteração do valor não é permitida
public double Saldo { get; private set; }
```

https://learn.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties

Encapsulamento

Exemplo 6 – Propriedades autoimplementadas

Vamos alterar o exemplo da conta bancária para que apenas a leitura do valor do atributo “Saldo” seja possível fora da classe. A alteração do valor deve ser privada.

Para isso, usaremos uma propriedade autoimplementada.

```
// Propriedade autodeimplementada Saldo
// Apenas get é acessível fora da classe.
// Alteração do valor não é permitida
public double Saldo { get; private set; }
```

Revisão

Próxima aula

Exercícios

- Lógica de programação em C#
- Classes
- Objetos
- Abstração e encapsulamento

Dúvidas?

renan.duarte@gedre.ufsm.br

GEDRE – Prédio 10 – CTLAB