# Algoritmos e Estruturas de Dados I
# Aula 13 - Introdução à Orientação a Objetos

Prof. Felipe Lara

# Definições

- **Classe**

  Uma classe é um tipo definido pelo usuário que contém o molde, a especificação para os objetos, assim como o tipo inteiro contém o molde para as variáveis declaradas como inteiros.

- **Objeto**

  Um objeto é uma instância de uma classe, ou seja, é a materialização de uma construção feita com base na forma/classe definida. Uma variável declarada de um tipo A (classe) seria um objeto da classe A.

# Atributos e Métodos

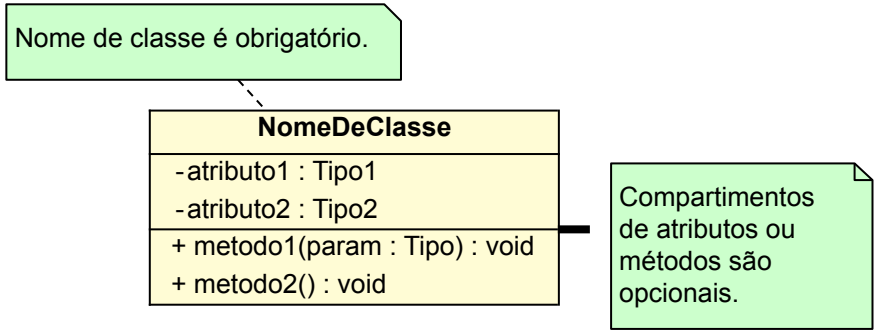
Uma struct possui atributos. Ex. Struct Cliente. Atributos: id, nome, salário.

**Uma classe possui atributos e métodos.** Os métodos realizam operações sobre os atributos.

Exemplo: uma classe Produto para um sistema de gerenciamento de estoque.

- Atributos:
  - descricao : char[50]
  - preco : float
  - quant : int
- Métodos:
  - emEstoque() : bool
  - incializaProduto (char[50], float , int )

# Representação Visual de classes com UML




| Produto |
| --- |
| + descricao : char[50]<br>+ preco : float<br>+ quant : int |
| + inicializaProduto(descricao : char[50], preco : float, quant : int) : void<br>+ emEstoque() : bool |

UML (*Unified Modeling Language*) permite representar classes e objetos para fins de modelagem de dados.

# Definindo a classe Produto: C++

```
#include <iostream>
#include <string.h>
using namespace std;

class Produto{
  public:
    string descricao;
    float  preco;
    int quant;

  bool emEstoque ( )  {
    return (quant > 0);
  }
  void inicializaProduto(string d, float p, int q){
    descricao= d;
    preco = p ;
    quant = q ;
  }
};
```

# Usando a classe Produto: Main em C++

```
int main() {
  Produto p;
  p.descricao = "Curso de Engenharia”;
  p.preco = 10.99F;
  p.quant = 200 ;
  cout << " Produto :  " << p.descricao << " \n " ;
  cout << " Preço :  " << p.preco << " \n " ;
  cout << " Estoque :  " << p.quant << " \n " ;

  if (p.emEstoque ( ))
    cout << " Produto em estoque . " << " \n " ;
  return 0 ;

}
```

# Construindo um objeto

- Objetos são instâncias de uma classe:
  - Lê-se instância como sendo um elemento com o tipo da classe e um estado corrente individual.
- Exemplo:
  - Classe → Produto (tipo com descricao, preco e quantidade)
  - Objeto de Produto → p = (Engenharia; 1.99; 200)
- Ao criar um objeto sua memória é inicializada.

# Criação de Objetos em C++

- Declaração de um Objeto:
  - nomeClasse nomeObjeto;
  - Objeto é inicializado no momento da declaração e um estado lhe é atribuído.
- Declaração de apontadores:
  - nomeClasse *nomeObjeto;
  - Cria-se o apontador mas não se cria o objeto. Criação do objeto através da cláusula new.
  - Apontador pode apontar para qualquer objeto.

# Definição

- Em C++, se não for definido um modo de inicialização o compilador atribui lixo para os atributos da classe.
- Construtores são usados para inicializar objetos com valores diferentes do padrão.

- Construtores:
  - Possuem o mesmo nome da classe. Não possuem valores de retorno.
- Uma classe pode ter de 0 a muitos construtores.

# Construtores

Razões para se usar construtores especializados: Algumas classes não possuem estado inicial aceitável sem parâmetros

- Fornecer um estado inicial é conveniente e aceitável
- Construir um objeto aos poucos pode ser desgastante
- Um construtor que não é público restringe quem irá criar objetos utilizando-o. Pode-se assim restringir o uso de sua classe

# Exemplo de Construtor

```
class  Produto  {
    . . .
    Produto ( string desc ,  float p , int q ) {
        i f  (desc.length ( ) ) >= 3 )
            descricao = desc;
        i f  ( p > 0 )
            preco = p ;
        i f  ( q >= 0 )
            quant = q ;
    }

    Produto ( )  {
        descricao = “Novo Produto”;
        preco = 0 . 01F ;
        quant = 0 ;
      }
}
```

# Usando construtores da classe Produto em C++

```
i n t main ( )  {

    / / Criando objeto - Exemplo 1
    Produto p1 ;

    / / Criando objeto - Exemplo 2
     Produto p2 ( " Xulambs " , 1 . 99F ,  200 ) ;

    cout << " Produto :  " << p1 . descricao << " \ n " ;
    cout << " Preço :  " << p1 . preco << " \ n " ;
    cout << " Estoque :  " << p1 . quant << " \ n " ;

    cout << " Produto :  " << p2 . descricao << " \ n " ;
    cout << " Preço :  " << p2 . preco << " \ n " ;
    cout << " Estoque :  " << p2 . quant << " \ n " ;
}
```

# Usando construtores da classe Produto em C++

Em C++ os construtores podem ser chamados usando o comando new ou na declaração de um objeto de uma certa classe.

```
int main ( ) {

    Produto p1 ( " Descricao produto 1 " , 1 . 99F ,  200 ) ;
    Produto * p2 = new Produto ( " Descricao produto 2 " , 1 . 99F ,  200 ) ;

    cout << " Produto :  " << p1 . descricao << " \ n " ;
    cout << " Preço :  " << p1 . preco << " \ n " ;
    cout << " Estoque :  " << p1 . quant << " \ n " ;

    cout << " Produto :  " << p2->descricao << " \ n " ;
    cout << " Preço :  " << p2->preco << " \ n " ;
    cout << " Estoque :  " << p2->quant << " \ n " ;
}
```

# Exercícios

1) Quais atributos e métodos são importantes para uma classe aluno? Implemente essa classe em C++ criando os atributos e os métodos definidos.

2) Quais atributos e métodos são importantes para uma classe loja? Implemente essa classe em C++ criando os atributos e os métodos definidos.

PUC Minas

# Encapsulamento

# Encapsulamento: ocultando informações

- Objetiva separar aspectos visíveis de um objeto ou classe de seus detalhes de implementação
- Interface:
  - tudo aquilo que o usuário do objeto vê/acessa.

# Encapsulamento: ocultando informações

- Permite alterar a implementação de um objeto sem impactos em outros módulos do sistema.
- Permite que seus dados sejam protegidos de acesso ilegal.
- Em geral, desejamos ocultar determinados dados e/ou métodos do cliente/usuário da aplicação.

# Ocultando informações

Exemplo:

- Acessar o campo quant e definir um estoque negativo pode invalidar o Produto.

Solução:

- Encapsulamento.

# Modificadores de acesso

- Modificadores de acesso controlam a visibilidade dos componentes na aplicação.
- Nível da classe: **public**
  - Classe declarada como public é visível a todas as classes do programa.
  - Classe sem modificador de acesso é visível apenas em seu pacote.
- Nível dos membros (atributos e métodos): **public, private, protected**

# Modificadores de acesso

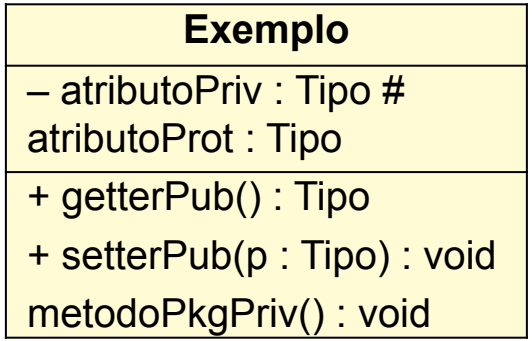
O C++ possui essencialmente 3 modificadores de acesso ao nível dos membros:

- private: membros declarados com acesso privado são acessíveis apenas na própria classe.
- protected: membros declarados com acesso protegido são acessíveis às classes do pacote e adicionalmente por suas subclasses.
- public: membros declarados com acesso público são acessíveis de qualquer lugar do programa.

# Princípios da ocultação de informação

- Use o nível de acesso mais restrito e que faça sentido para um membro particular.
- Use private a menos que haja uma boa razão para não fazê-lo.
- Evite campos public exceto para constantes. Campos públicos aumentam o acoplamento em relação a uma implementação específica e reduz a flexibilidade do sistema a mudanças.

# Encapsulamento na UML




```
class  Exemplo  {
    private Tipo  atributoPriv;
    protected Tipo  atributoProt;

    public Tipo  getterPub ( )  {
     . . .
    }
    public void  setterPub ( Tipo p )  {
     . . .
    }

    void metodoPkgPriv ( )  {
     . . .
    }
}
```

# Classe Produto: encapsulamento em C++

```
class  Produto  {

private :
    string descricao;
    f l o a t  preco ;
    i n t  quant ;
public :
    Produto ( string d ,  float  p ,  int  q) {
        i f  ( d.length() >= 3 )
            descricao = d  ;
        i f  ( p > 0 )
            preco = p ;
        i f  ( q >= 0 )
            quant = q ;
    }

    Produto ( ) {
        strcpy (descricao , " Novo Produto " ) ;
        preco = 0 . 01F ;
        quant = 0 ;
    }
    bool emEstoque ( ) {
        return  ( quant > 0 ) ;
    }

}
```

# Métodos de acesso (getters e setters)

- **Métodos get:** acessam o valor de um atributo privado.
  - Valores podem ser tratados antes de serem exibidos.
  - Ex: atributo booleano sendo exibido como V ou F atributo numérico e seu correspondente string.
- **Métodos set:** atribuem um valor a um atributo privado.
  - Valores devem ser validados/tratados antes de serem atribuídos.
  - Ex: número do dia numa classe Data depende do atributo mes.

# Classe Produto: métodos de acesso (getters e setters)

```
public :
    char * get Descricao ( ) { return descricao ; }
    float get Preco ( ) { return preco ; }
    int getQuant ( ) { return quant ; }

    void setDescricao ( char d [ ] ) {
        i f ( strlen( d ) >= 3 )
            strcpy( descricao , d ) ;
    }
    void set Preco (float p ) {
        i f ( preco > 0 ) preco = p ;
    }
    void setQuant ( int q ) {
        i f ( quant >= 0 ) quant = q ;
    }
    Produto ( char d [ ] , float p ,int q ) {
         set Descricao ( d ) ;
        set Preco ( p ) ;
        setQuant ( q ) ;
    }}
```

# Classe Produto: acessando membros encapsulados

```
i n t main ( ) {
    Produto p1 ;
    Produto p2 ( " Produto 2 " , 1 . 99F , 200 ) ;

    p1 . set Descricao ( " Produto 1 " ) ; p1 . set Preco ( 2 . 4 9 ) ;
    p1 . setQuant ( 1 0 ) ;

    cout << " Produto : " << p1 . get Descricao ( ) << " \ n " ; cout << " Preço : " << p1 . get Preco ( ) << " \ n " ;
    cout << " Estoque : " << p1 . getQuant () < < " \ n " ;

    cout << " Produto : " << p2 . get Descricao ( ) << " \ n " ; cout << " Preço : " << p2 . get Preco ( ) << " \ n " ;
    cout << " Estoque : " << p2 . getQuant () < < " \ n " ;

    return  0 ;
}
```

# Exemplos para Classe Conta

```
class Conta {
private :
    double limite ;
    double saldo ;
public :
    double getSaldo ( ) {
        return  saldo ;
    }

    void set Saldo ( double saldo ) {
        this ->saldo = saldo ;
    }
    double getLimite( ) {
        return limite;
    }
    void setLimite ( double limite ) {
        this -> limite = limite;
    }
} ;
```

```
class Conta {
private :
    double saldo ;
    double limite ;
public :
    double get Saldo ( ) {
        return  saldo ;
    }
    Conta ( double limite ) {
        this -> limite = limite ;
    }
    void depositar ( double x ) {
            saldo += x ;
    }
    void sacar ( double x ) {
        if ( saldo + limite >= x )
                saldo -= x ;
        else
            cout << " Fundos insuficientes.\n" ;
}
```

# Exercícios

1 - Crie uma classe chamada Pessoa com dois atributos: nome e idade. Escreva um programa que instancie um objeto dessa classe, atribua valores e imprima-os no terminal.

2 - Altere a classe Pessoa do exercício 1 para incluir um construtor que receba nome e idade como parâmetros. Adicione também um método chamado exibirDados() que mostre as informações da pessoa.

3 - Modifique a classe Pessoa do exercício 2 para que os atributos sejam privados. Crie métodos get e set para acessar e modificar os dados da pessoa com segurança.

PUC Minas

# Dúvidas?