

Orientação a Objetos

Prof. Isaac

Orientação a Objetos: Vantagens e Desvantagens

Por que Orientação a Objetos?

- Aproximação do sistema criado ao que é observado no mundo real:
 - Representação de elementos como objetos (classes)
- Reutilização do código:
 - Menos linhas
 - Melhor manutenção
- Organização

Desvantagens da POO

- Pode ser mais lenta quando comparada a programação estruturada / imperativa / procedimental (procedural) ...
- Porém, com a tecnologia atual, essa execução mais lenta quase sempre não é sentida.

Algumas Linguagens Orientadas a Objetos





Breve História

- Criado por James Gosling, Mike Sheridan e Patrick Naughton na Sun Microsystems no início da década de 90 (O nome inicial do Java era Oak);
- A ideia era criar aplicativos para televisão interativa... Mas o projeto era muito avançado para a televisão da época.



James Gosling

Breve História

- Gosling desenvolveu o Java com estilo de sintaxe do C/C++
- Oficialmente o Java foi lançado pela Sun em 1995.
- Em 2009-10 a Sun foi comprada pela Oracle.
- Ainda em 2010, James Gosling deixa a Oracle
- Por que o símbolo do Java é uma xícara???!!!



Objetivos da criação do Java

Existiam 5 objetivos principais na criação do Java:

- It must be “simple, object-oriented, and familiar”.
- It must be “robust and secure”.
- It must be “architecture-neutral and portable”.
- It must execute with “high performance”.
- It must be “interpreted, threaded, and dynamic”.

Características do Java

- Não faz uso do tipo *ponteiro*
- Não possui ciclo de pré-processamento
- Java roda em todo tipo de dispositivo: notebooks, desktops, video games, celulares, supercomputadores etc.
- Roda em uma máquina virtual - Java Virtual Machine (JVM)
- “Write Once, Run Anywhere” (WORA)
- Enorme quantidade de bibliotecas gratuitas para realizar os mais diversos trabalhos

Java Virtual Machine (JVM):

- Faz o gerenciamento automático da memória (Garbage Collection)
- Permite que um código feito em Java seja multiplataforma
- Hotspot: detecta pontos em que o código é muito executado e os compila para instruções nativas da plataforma.
 - JIT: Just in Time Compiler

Conceitos Básicos de Orientação a Objetos

Classes e Objetos

Classes

- Representam itens do mundo real:
 - Exemplos:
 - Pessoas
 - Veículos
 - Robôs
- São Compostas de:
 - **Atributos** (variáveis de instância)
 - **Métodos** (funções-membro)

Objetos

- Todo objeto pertence a uma **classe**
- Representam **instâncias** de entidades no mundo real
- São criados a partir das **classes**
- São **instâncias** das **classes**
- Os objetos associam valores específicos aos **atributos**

Instanciar (Informática)

- Instanciar é criar um objeto, ou seja, alocar um espaço na memória, para posteriormente poder utilizar os métodos e atributos que o objeto dispõe.
- Em informática instância é usada com o sentido de exemplar. No contexto da orientação ao objeto, instância significa a concretização de uma classe.

Classes e Objetos

Quando definimos uma classe de objetos, estamos, na verdade, definindo que propriedades e métodos o objeto possui!

Exemplo: Diferença entre Classe e Objeto

- Classe:

- É um modelo;
- De maneira mais prática, é como se fosse a **planta de uma casa**;

- Objeto:

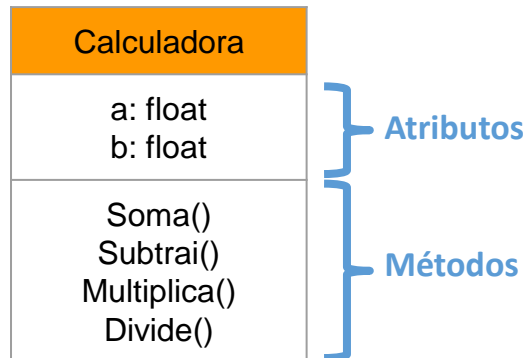
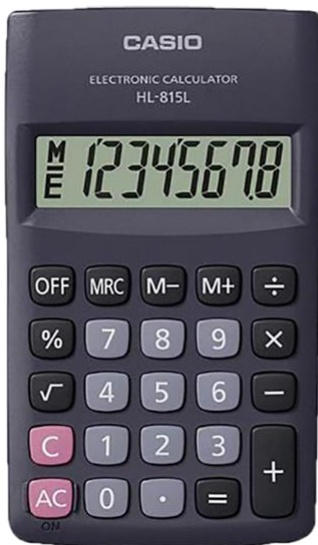
- É criado a partir da classe;
- É como se fosse a **própria casa** construída
- Pode-se construir várias casas a partir da mesma planta, assim como podemos instanciar vários objetos de uma só classe

Estrutura de uma Classe

Nome da Classe
- Atributos
- Métodos

- **Atributos** são **variáveis** que armazenam informações do objeto.
- **Métodos** são as operações (**funções**) que o objeto pode realizar.

Exemplo de Classe



Atributos

- Variáveis de instância
- Atributos são **variáveis** em que o **objeto** armazena **informações**.
- Lembram:
 - os campos de uma *struct* em C.

Métodos

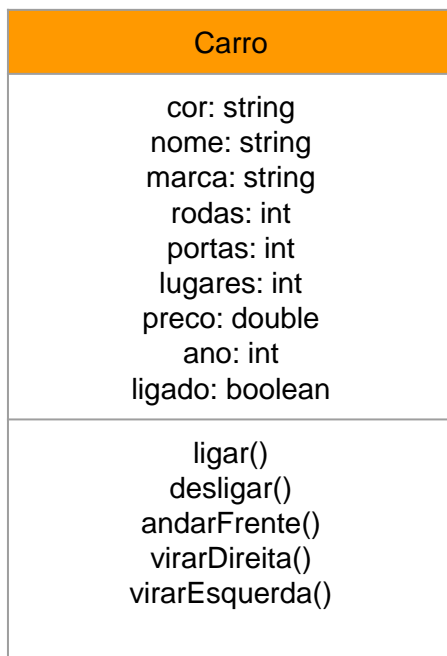
- São sequências de **declarações** e **comandos** executáveis **encapsulados** como se fossem um mini-programa.
- Similares:
 - sub-rotinas
 - procedimentos
 - funções

Exemplo de Classe e Objetos

Como você modelaria um carro?

Quais atributos e métodos você incluiria na sua classe Carro?

Exemplo de Classe e Objetos



Exemplo de Classe e Objetos

Carro
<p>cor: string nome: string marca: string rodas: int portas: int lugares: int preco: double ano: int ligado: boolean</p>
<p>ligar() desligar() andarFrente() virarDireita() virarEsquerda()</p>



Exemplo de Classe e Objetos

Carro
<p>cor: string nome: string marca: string rodas: int portas: int lugares: int preco: double ano: int ligado: boolean</p>
<p>ligar() desligar() andarFrente() virarDireita() virarEsquerda()</p>



Exemplo de Classe e Objetos

Cat
size: float color: string positionX: float positionY: float
moveForward() moveBackward() moveUP() moveDown()

Cat garfield;
Cat tom;
Cat felix ;
Cat scratchy;



Exemplo de Classe e Objetos

Robot

positionX: float
positionY: float
direction: float

moveForward()
moveBackward()
turnLeft()
turnRight()

Robot b1;
Robot b2;
Robot b3;



Exemplo de Classe e Objetos

Movie
name: string storyline: string runtime: float
play() stop() pause()

Movie poderosoChefao;
Movie senhorDosAneis;
Movie forrestGump;



Code Conventions

- Nome das *Classes* deverão ser substantivos, para nome composto use letra maiúscula no início de cada palavra. Tente manter o nome da sua classe simples e descritivo. Evite usar acrônimos ou abreviações.
- Exemplos:
 - **class Movie**
 - **class Calculadora**
 - **class ImageSprite**

Code Conventions

- *Métodos* deverão ser verbos, iniciado com letra minúscula, e com letra maiúscula no início das demais palavras.
- Exemplos:
 - `run();`
 - `runFast();`
 - `getBackground();`

Code Conventions

- *Todos os atributos das classes deverão iniciar com letra minúscula e usar letra maiúscula no inicio das demais palavras.*
- Exemplos:
 - myGradeBook;
 - application;
 - testScope;

Orientação a Objetos no



Definindo a Classe

- A definição da classe indica ao compilador que **métodos** e **atributos** pertencem à **classe**.
- A classe é iniciada pela palavra-chave **class** seguida do nome da classe.
- O corpo da classe é colocado entre chaves **{ }**.

Exemplo de classe

```
1 // Fig. 3.1: GradeBook.java
2 // Declaração de Classe com um método.
3
4 public class GradeBook
5 {
6     // exibe uma mensagem de boas-vindas para o usuário GradeBook
7     public void displayMessage()
8     {
9         System.out.println( "Welcome to the Grade Book!" );
10    } // termina o método displayMessage
11
12 } // fim da classe GradeBook
```


Exemplo: Criando um Objeto da Classe

1. Declarando um Objeto:

- <nome_da_classe> <nome_do_objeto>
- **GradeBook myGradeBook**

2. Instanciando um Objeto:

- **myGradeBook = new GradeBook()**

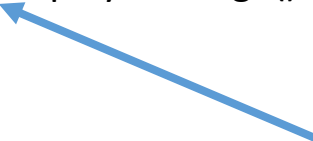


a palavra-chave **new**
cria um novo objeto da
classe especificada

Operador ponto (.)

- É usado para acessar variáveis de instância e métodos a partir de um objeto.
- Exemplo:

- `myGradeBook.displayMessage()`



Chama o método
displayMessage do
objeto `myGradeBook`

Exemplo de classe

```
1// Fig. 3.1: GradeBook.java
2// Declaração de Classe com um método.
3
4public class GradeBook
5{
6    // exibe uma mensagem de boas-vindas para o usuário GradeBook
7    public void displayMessage()
8    {
9        System.out.println( "Welcome to the Grade Book!" );
10    } // termina o método displayMessage
11
12} // fim da classe GradeBook
```

palavra-chave **new** cria um novo objeto da classe especificada à direita da palavra-chave

```
1// Fig. 3.2: GradeBookTest.java
2// Cria um objeto GradeBook e chama seu método displayMessage.
3
4public class GradeBookTest
5{
6    // método main inicia a execução de programa
7    public static void main( String args[] )
8    {
9        // cria um objeto GradeBook e o atribui a myGradeBook
10        GradeBook myGradeBook = new GradeBook();
11
12        // chama método displayMessage de myGradeBook
13        myGradeBook.displayMessage();
14    } // fim de main
15
16} // fim da classe GradeBookTest
```

ponto separador: acessa membros/variáveis a partir de um objeto

Encapsulamento

- Encapsular significa separar o programa em partes independentes, o mais isoladas possível.
- O propósito do encapsulamento é o de organizar os dados que sejam relacionados, encapsulando-os em objetos / classes.
- O uso do modificador de acesso *private* nos membros, combinado com o uso dos métodos *get* e *set* é uma parte do encapsulamento.

Encapsulamento - Modificadores de Acesso

- *public:*

- Indica que um método ou atributo é acessível a outras funções e funções-membro de outras classes.

- *private:*

Torna um membro de dados ou uma função-membro acessível apenas a funções-membro da classe.

- *protected:*

Torna o membro acessível às classes do mesmo pacote ou através de herança.

Encapsulamento - Modificadores de Acesso

- *package-private* (modificador padrão):
 - Se nenhum modificador for utilizado, todas as classes do mesmo pacote têm acesso ao atributo, construtor, método ou classe.

Encapsulamento - Modificadores de Acesso

Regra Geral:

- atributos (variáveis de instância) devem ser declarados como *private*
- funções-membro (métodos) devem ser declaradas como *public*

Encapsulamento - Funções *set* e *get*

Se as *variáveis de instância* devem ter especificador de acesso *private*,
como então elas serão acessadas pelos objetos que já foram instanciados?

Encapsulamento - Funções *set* e *get*

Para isso utilizamos as funções (com acesso *public*):

set : para atribuir valores

get : para obter valores

```

1 // Fig. 3.7: GradeBook.java
2 // classe GradeBook que contém uma variável de instância courseName
3 // e métodos para configurar e obter seu valor.
4
5 public class GradeBook
6 {
7     private String courseName; // nome do curso para esse GradeBook
8
9     // método para configurar o nome do curso
10    public void setCourseName( String name )
11    {
12        courseName = name; // armazena o nome do curso
13    } // termina o método setCourseName
14
15    // método para recuperar o nome do curso
16    public String getCourseName()
17    {
18        return courseName;
19    } // termina o método getCourseName
20
21    // exibe uma mensagem de boas-vindas para o usuário GradeBook
22    public void displayMessage()
23    {
24        // essa instrução chama getCourseName para obter o
25        // nome do curso que esse GradeBook representa
26        System.out.printf( "Welcome to the grade book for\n%s!\n",
27            getCourseName() );
28    } // termina o método displayMessage
29
30 } // fim da classe GradeBook

```

Atributos vs. Variáveis Locais

- Atributos (Variáveis de Instância):
 - Existem por toda a vida do **objeto**
 - São representados como **membros de dados**
 - Todo **objeto** de classe mantém sua própria **cópia de atributos**
- Variáveis Locais:
 - Variáveis declaradas no corpo de uma definição de **método**
 - Não podem ser utilizadas fora do corpo deste método
 - São conhecidas apenas por este método
 - Quando o método **termina**, os valores das respectivas **variáveis locais são perdidos (escopo local)**

```

1 // Fig. 3.8: GradeBookTest.Java
2 // Cria e manipula um objeto GradeBook.
3 import java.util.Scanner; // programa utiliza Scanner
4
5 public class GradeBookTest
6 {
7     // método main inicia a execução de programa
8     public static void main( String args[] )
9     {
10         // cria Scanner para obter entrada a partir da janela de comando
11         Scanner input = new Scanner( System.in );
12
13         // cria um objeto GradeBook e o atribui a myGradeBook
14         GradeBook myGradeBook = new GradeBook();
15
16         // exibe valor inicial de courseName
17         System.out.printf( "Initial course name is: %s\n\n",
18             myGradeBook.getCourseName() );
19
20         // solicita e lê o nome do curso
21         System.out.println( "Please enter the course name:" );
22         String theName = input.nextLine(); // lê uma linha de texto
23         myGradeBook.setCourseName( theName ); // configura o nome do curso
24         System.out.println(); // gera saída de uma linha em branco
25
26         // exibe mensagem de boas-vindas depois de especificar nome do curso
27         myGradeBook.displayMessage();
28     } // fim de main
29
30 } // fim da classe GradeBookTest

```

Exercício 1 - Crie a classe Pessoa

Variáveis:

- *cpf* e nome, ambos *private* e do tipo *String*.
- idade *private* e *int*

Métodos:

- *set* e *get* para cada um dos atributos

main():

Classe *TestePessoa*

- crie três objetos (p1, p2 e p3) do tipo Pessoa
- obtenha pelo teclado o valor de cpf, nome e idade para p1, p2 e p3
- inicialize os atributos de p1, p2 e p3 com os métodos *set*
- exiba o conteúdo dos atributos de p1, p2 e p3 utilizando o método *get*

Exercício 2 - Crie a classe Swapper

Variáveis:

- *x* e *y*, ambos *private* e do tipo *float*.

Métodos:

- *set* e *get* para cada um dos atributos
- *void swap()* que troca os valores de *x* e *y*

main():

- classe *SwapperDemo*
- crie um objeto, chamado *troca*, do tipo *Swapper*
- obtenha pelo teclado o valor de *x* e *y* para o objeto *troca*
- inicialize os atributos de *troca* com os métodos *set*
- utilize o método *swap()* para trocar os valores de *x* e *y*
- exiba os valores trocados utilizando os métodos *get*

Fim

Material Complementar



Definindo a Classe

- A definição da classe indica ao compilador que **métodos** e **atributos** pertencem à **classe**.
- A classe é iniciada pela palavra-chave **class** seguida do nome da classe.
- O corpo da classe é colocado entre chaves { }.

Exemplo de Classe

Início do
corpo da
classe

Fim do
corpo da
classe

```
1// Fig. 3.1: fig03_01.cpp
2// Define a classe GradeBook com uma função membro displayMessage;
3// Cria um objeto GradeBook e chama sua função displayMessage.
4#include <iostream>
5
6using namespace std;
7
8// Definição da classe GradeBook
9class GradeBook
10{
11public:
12    // função que exibe uma mensagem de boas-vindas ao usuário do GradeBook
13    void displayMessage()
14    {
15        cout << "Welcome to the Grade Book!" << endl;
16    } // fim da função displayMessage
17}; // fim da classe GradeBook
```

class inicia a classe
chamada GradeBook

```

1// Fig. 3.1: fig03_01.cpp
2// Define a classe GradeBook com uma função membro displayMessage;
3// Cria um objeto GradeBook e chama sua função displayMessage.
4#include <iostream>
5
6using namespace std;
7
8// Definição da classe GradeBook
9class GradeBook
10{
11public:
12    // função que exibe uma mensagem de boas-vindas ao usuário do GradeBook
13    void displayMessage()
14    {
15        cout << "Welcome to the Grade Book!" << endl;
16    } // fim da função displayMessage
17}; // fim da classe GradeBook

```

C++

```

1// Fig. 3.1: GradeBook.java
2// Declaração de Classe com um método.
3
4public class GradeBook
5{
6    // exibe uma mensagem de boas-vindas para o usuário GradeBook
7    public void displayMessage()
8    {
9        System.out.println("Welcome to the Grade Book!");
10    } // termina o método displayMessage
11
12} // fim da classe GradeBook

```

Java

Exemplo: Instanciando um Objeto da Classe

objeto
myGradeBook criado

```
20 // a função main inicia a execução do programa
21 int main()
22 {
23     GradeBook myGradeBook; // cria um objeto GradeBook chamado myGradeBook
24     myGradeBook.displayMessage(); // chama a função displayMessage do objeto
25     return 0; // indica terminação bem-sucedida
26 } // fim de main
```

myGradeBook chama o
método displayMessage()

Encapsulamento Funções *set* e *get*

```
1 // Fig. 3.5: fig03_05.cpp
2 #include <iostream>
3 #include <string> // o programa utiliza classe de string padrão C++
4
5 using namespace std;
6
7 // Definição da classe GradeBook
8 class GradeBook
9 {
10 public:
11     void setCourseName(string name)
12     {
13         courseName = name;
14     }
15
16     string getCourseName() .....
17     { .....
18         return courseName;
19     } .....
20
21     void displayMessage()
22     {
23         cout << "Welcome to the grade book for\n" << getCourseName() << "!"
24         << endl;
25     }
26
27 private:
28     string courseName;
29 };
```