



Quem se prepara, não para.

Programação Orienta a Objetos

3º período

Professora: Michelle Hanne

Sumário

- ✓ Fundamentos Java
- ✓ IDE Eclipse
- ✓ Conceitos Fundamentais:
 - ✓ Classe
 - ✓ Atributo
 - ✓ Método
 - ✓ Objeto
- ✓ Encapsulamento
- ✓ Mensagem

- ***Diferente da maior parte das linguagens de programação, a compilação de um programa java não gera um executável, mas um bytecode a ser executado na JVM (Java Virtual Machine).***
- Programa fonte em java tem a extensão **.java**.
- Programa Java compilado tem a extensão **.class**.

Simples	Java tem um conjunto de recursos conciso e coeso que a torna fácil de aprender e usar.
Segura	Java fornece um meio seguro de criar aplicativos de Internet.
Portável	Os programas Java podem ser executados em qualquer ambiente para o qual houver um sistema de tempo de execução Java.
Orientada a objetos	Java incorpora a moderna filosofia de programação orientada a objetos.
Robusta	Java incentiva a programação sem erros por ser fortemente tipada e executar verificações de tempo de execução.
Várias threads	Java fornece suporte integrado à programação com várias threads.
Neutra quanto à arquitetura	Java não tem vínculos com uma determinada máquina ou arquitetura de sistema operacional.
Interpretada	Java dá suporte a código para várias plataformas com o uso do bytecode.
Alto desempenho	O bytecode Java é altamente otimizado para obtenção de velocidade de execução.
Distribuída	Java foi projetada visando o ambiente distribuído da Internet.
Dinâmica	Os programas Java carregam grandes quantidades de informações de tipo que são usadas na verificação e resolução de acessos a objetos no tempo de execução.

A programação orientada a objetos (OOP, *object oriented programming*) é a essência de Java. A metodologia orientada a objetos é inseparável da linguagem, e todos os programas Java são, pelo menos até certo ponto, orientados a objetos

- Java possui os seguinte tipos básicos de dados:
 - boolean: valores booleanos *true* e *false*;
 - byte: inteiro de 8 bits;
 - short: inteiro de 16 bits;
 - **int: inteiro de 32 bits**
 - Números inteiros que começam com “0” são octais. Ex.: 077
 - Número inteiros que começam com “0x” são hexadecimais: Ex.: 0xA34
 - long: inteiro de 64 bits
 - **float: real de 32 bits**
 - Para indicar que uma constante é *float* deve-se colocar *f* ou *F* no final dela. Ex.: 35.5f
 - double: real de 64 bits;
 - char: caracteres.

- Em Java, **String** é um classe pré-definida.
- Cada *string* utilizada no programa é um objeto do tipo **String**.
- Métodos principais da classe String:
 - `public char charAt(int index):` devolve o caractere da posição *index*.

- **public boolean contains(CharSequence s):** devolve **true** se a string contém a sequência *s* de caracteres
- **public boolean equals(Object obj):** devolve **true** se a string é igual a *obj* (quando *obj* é uma string)
- **public int compareTo(String s):** retorna um valor negativo se a string precede *s*; devolve um valor negativo de *s* precede a string; devolve zero, se a string é igual a *s*
- **public int compareToIgnoreCase(String s):** igual ao anterior, porém ignora diferença entre maiúscula e minúsculas

OPERADOR	SIGNIFICADO	ASSOCIATIVIDADE	EXEMPLO
+	Adição	Esquerda	a + b
-	Subtração	Esquerda	a - b
*	Multiplicação	Esquerda	a * b
/	Divisão	Esquerda	a / b
%	Resto da divisão inteira	Esquerda	a % b
-	Sinal negativo (- unário)	Direita	- a
+	Sinal positivo (+ unário)	Direita	+a
++	Incremento unitário	Esquerda/Direita	++a ou a++
--	Decremento unitário	Esquerda/Direita	--a ou a--

Tabela dos Operadores Aritméticos

OPERADOR	SIGNIFICADO	ASSOCIATIVIDADE	EXEMPLO
= =	Igual	Esquerda	a == b
! =	Diferente	Esquerda	a != b
>	Maior que	Esquerda	a > b
>=	Maior ou igual a	Esquerda	a >= b
<	Menor que	Esquerda	a < b
<=	Menor ou igual a	Esquerda	a <= b

Tabela dos Operadores Relacionais

Resumo Geral Java:

<https://medium.com/@claudiobernardo/guia-de-estudo-sintaxe-java-5fe1a0d997a0>

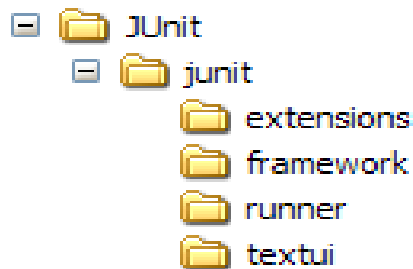
- Em Java classes são organizadas em pacotes.
- Um pacote é um conjunto de classes relacionadas.
- A palavra *package* indica o pacote ao qual a classe pertence.

```
package rh;
```

```
public class Funcionario {  
    //corpo da classe funcionario  
}
```

(A classe `Funcionario` está dentro de um pacote chamado `rh`. No Windows, um pacote corresponde a uma pasta onde ficam armazenadas as suas classes).

- Exemplo: o JUnit (uma biblioteca *open source* para realizar testes em software Java) possui os pacotes *extensions*, *framework*, *runner* e *textui*.



Quando uma classe necessita utilizar uma outra classe que não esteja em seu pacote é necessário importar o pacote da classe a ser utilizada.

- Isso é feito incluindo um comando **import** no início do código do arquivo .java.
- Exemplo: se quisermos utilizar a classe Date da API de Java, temos que importar o seu pacote.

```
import java.util.*;
```

Cada ferramenta IDE tem uma estrutura particular para armazenar os arquivos de um projeto.

Por exemplo, no Eclipse organiza os arquivos de acordo com a estrutura a seguir:

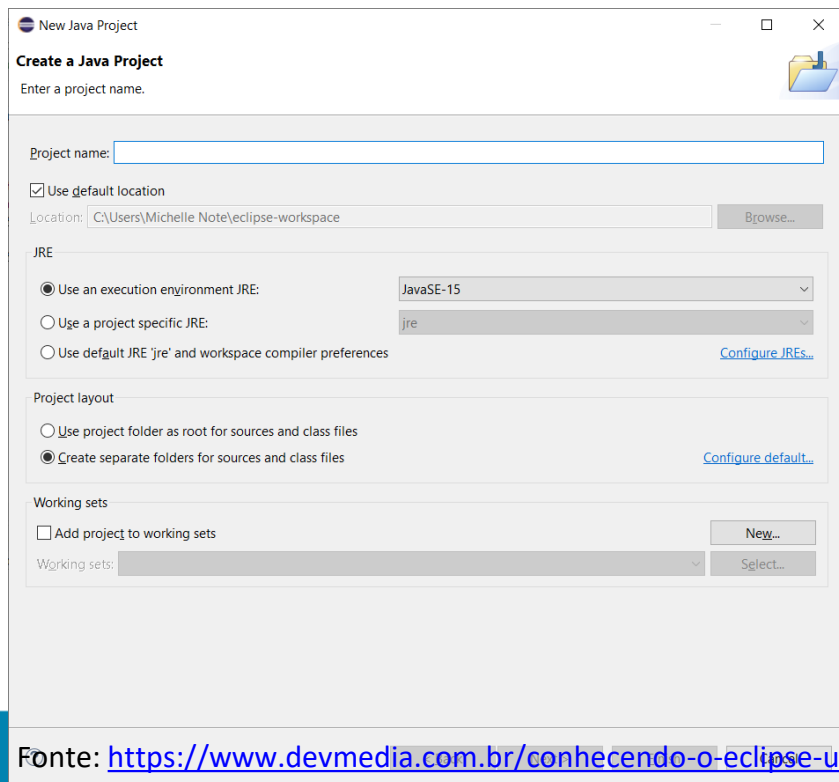
bin: contém os arquivos bytecodes compilados (.class) organizados em pacotes

settings: contém arquivos de configuração gerados pelo Eclipse.

src: onde ficam os arquivos fontes (.java) organizados em pacotes

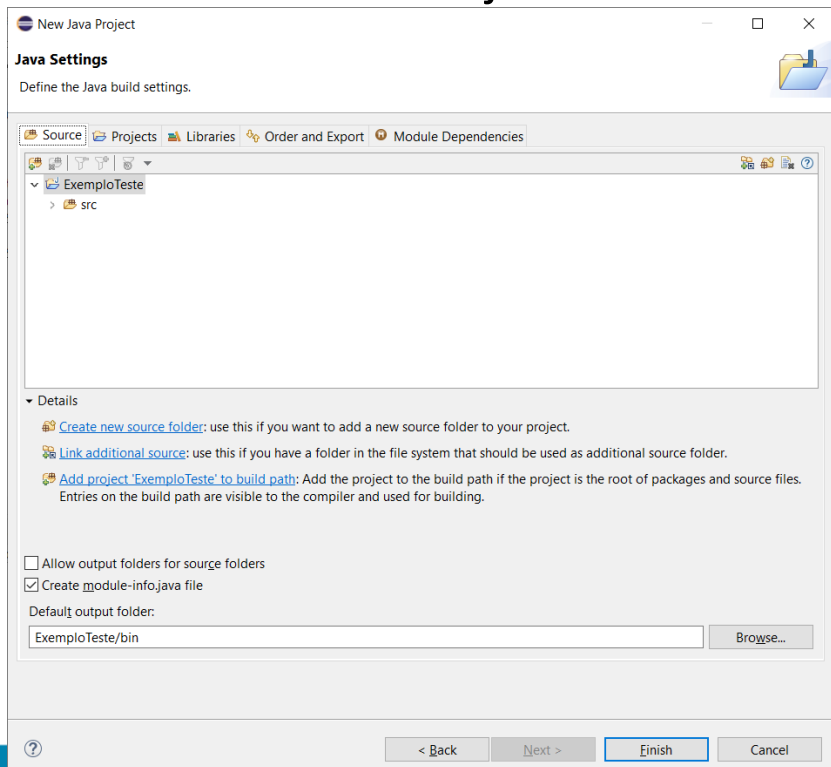
Criação do projeto

File-> New-> Java Project



- **Project Name** (Nome do Projeto) – Exemplo: `ComunicacaoBanco`.
- **Location** (Localização) – Define qual local da Workspace que será armazenado.
- **JRE (Java Runtime Environment)** – Ambiente de execução do Java.
- **Projeto layout** - Define a estrutura do projeto.
- **Working sets** – Organiza o projeto, podendo criar categorias para separar as Workspaces adicionadas, mais é uma maneira de organização.

Criação do projeto File-> New-> Java Project



- Para adicionar uma biblioteca pode-se instalar através da janela Libraries e dependências em Module Dependencies.

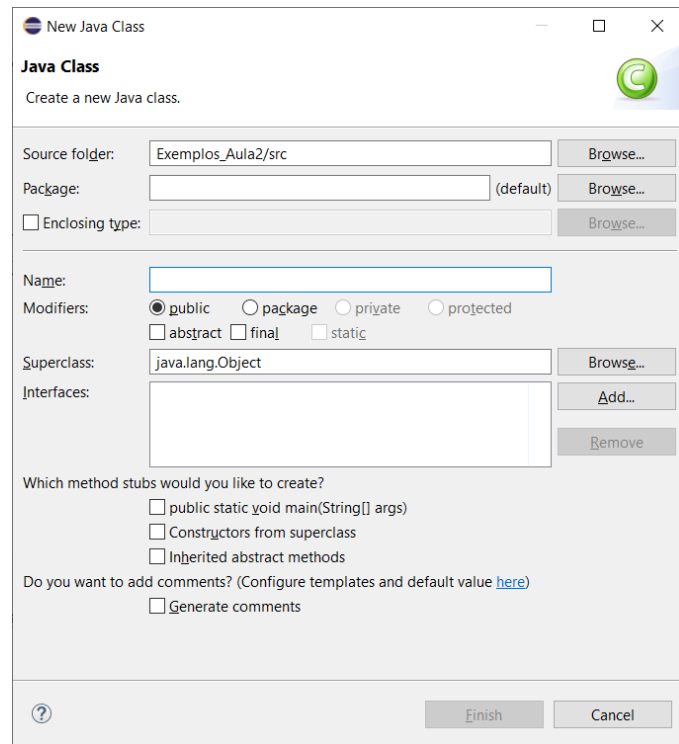
A pasta bin armazena os **bytecodes (.class)**, que são os arquivos que a Máquina Virtual Java (JVM), usa para rodar os programas Java, também podem ser lidos pela JVM de qualquer Sistema Operacional.

Criação do pacotes

- Os packages também conhecidos por pacotes são criados para armazenar as classes facilitando a organização do projeto e ajuda na reutilização de código.
- Para criar uma package clique com o botão direito do mouse em cima do projeto vá em New > Package então coloque o nome e clique no botão Finish.

Criação de Classes

- A Classe armazena as características e ação que definem seu estado e comportamento.
- Clique com o botão direito do mouse em cima do projeto vá em New > Class então coloque o nome e clique no botão Finish.
- Para criar uma classe de execução escolha a opção:
public static void main(String[] args)



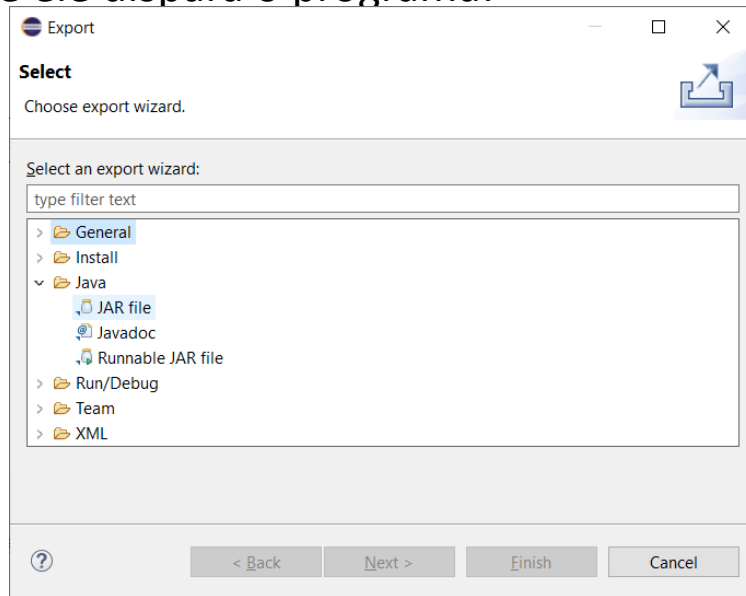
The screenshot shows the 'New Java Class' dialog box in Eclipse. The 'Source folder' is set to 'Exemplos_Aula2/src'. The 'Package' field is empty with a '(default)' hint. The 'Enclosing type' checkbox is unchecked. The 'Name' field is empty. Under 'Modifiers', the 'public' radio button is selected, and the 'abstract', 'final', and 'static' checkboxes are unchecked. The 'Superclass' is set to 'java.lang.Object'. The 'Interfaces' list is empty. Under 'Which method stubs would you like to create?', the 'public static void main(String[] args)' checkbox is selected, while 'Constructors from superclass' and 'Inherited abstract methods' are unchecked. At the bottom, the 'Do you want to add comments?' section has the 'Generate comments' checkbox unchecked. The 'Finish' and 'Cancel' buttons are at the bottom right.

Gerando arquivo .jar e javadoc

Um arquivo **.jar (Executable Jar File)**, tem o mesmo funcionamento de um arquivo executável .exe, ele esconde a implementação (código) do usuário, e roda o programa sozinho, basta clicar duas vezes sobre o mesmo, que ele dispara o programa.

O **javadoc** refere-se a documentação do seu projeto. Selecione o menu Project e, depois, a opção Generate Javadoc.

Botão direito em cima do Projeto, escolher Export



- **F11 - Debug**
- **CTRL + F11 – Run Eclipse**
- **CTRL + SHIFT + W - Fechar todas as abas;**
- CTRL + O - Localizar método / atributo em uma classe;
- CTRL + SHIFT + R - Localizar arquivos no workspace;
- CTRL + SHIFT + T - Descobrir onde estão as classes em um workspace mesmo que dentro de um jar;
- **CTRL + SHIFT + L - Lista de atalhos do Eclipse;**
- CTRL + F6 - Navegar entre os arquivos abertos;
- CTRL + F7 - Navegar entre as abas da perspectiva;
- CTRL + F8 - Navegar pelas perspectivas abertas;
- CTRL + H - Pesquisa onde está sendo utilizado determinado método, basta posicionar o cursor no método desejado;
- CTRL + SHIFT + F - Formata o código conforme os padrões setados nas preferências do eclipse;

Teclas de Atalho do Elipse

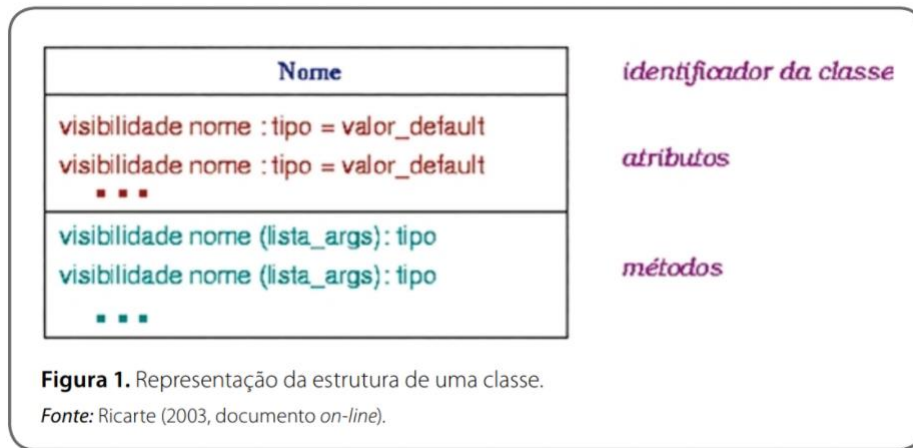
- CTRL + SHIFT + O - Organiza os imports da classe;
- CTRL + ALT + H - Mostra todas as ocorrências de um atributo ou método dentro de uma classe;
- ALT + SHIFT + L - Extrai o que estiver selecionado para uma variável;
- ALT + SHIFT + M - Extrai o que estiver selecionado para um método;
- CTRL + SHIFT + C - Comentar / Descomentar um bloco de código;
- CTRL + SHIFT + / - Comentar um grande bloco de código com /* */;
- CTRL + SHIFT + \ - Descomentar um grande bloco de código com /* */;
- CTRL + I - Indenta corretamente o código conforme preferências do eclipse;
- CTRL + SHIFT + B - Inserir / excluir breakpoint na linha onde está o cursor;
- CTRL + M - Maximizar a tela de edição de código;
- CTRL + ESPAÇO - Autocompleta;
- CTRL + 1 - Correções quando há erros de compilação, para novas classes, ou até mesmo quando precisar importar pacotes;
- CTRL + 3 - Busca um comando ou uma opção de menu baseado no que você escreve;

Conceitos Fundamentais

“uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica” (Booch, Rumbaugh e Jacobson (2006, p. 45) apud Silva et al (2019)).

As classes são utilizadas para expressar todo o vocabulário e escopo do sistema a ser desenvolvido.

- O paradigma de programação orientada a objetos envolve a identificação e abstração de entidades, de acordo com o escopo de um sistema. Essas entidades formam o vocabulário do sistema.
- Por exemplo, se estamos desenvolvendo um sistema para um consultório médico, entidades como prontuário, paciente, médico, etc., fazem parte do vocabulário e serão implementadas pelas classes.



Segundo Lima (2014, p. 49) apud Silva (2019) , “uma classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica”.

Cada classe deve possuir um nome que a diferencie das demais. Por padrão, o nome da classe sempre começa com letra maiúscula.

Um **atributo** é uma **propriedade** de uma classe e também são chamados de **variáveis** ou **campo**. É pelos atributos que podemos **definir o estado de um objeto**, fazendo com que esses valores possam ser alterados. Um atributo é, portanto, **uma abstração do tipo de dado ou de estados que os objetos da classe podem abranger**.

```
public class Cachorro{  
    public String nome;  
    public int peso;  
    public String corOlhos;  
    public int quantPatas;  
}
```

Uma **operação** ou **método** é a **implementação** de um **serviço** que pode ser **acionado** por algum **objeto da classe**, ou seja, são as operações que podem determinar o comportamento dos objetos. Um método pode também ser definido como **uma abstração de algo que pode ser feito com determinado objeto** e que pode ser **compartilhado** por todos os **objetos da classe**.

```
class Cachorro {  
    int tamanho;  
    String nome;  
    void latir() {  
        if(tamanho > 60)  
            System.out.println("Woof, Woof!");  
        else if(tamanho > 14)  
            System.out.println("Ruff!, Ruff!");  
        else  
            System.out.println("Yip!, Yip!");  
    }  
}
```


As classes definem reponsabilidades. Uma reponsabilidade é o conceito de que uma **classe deve executar uma única responsabilidade no sistema** e essa responsabilidade deve ser executada por esta única classe, sem ser repetida por outra classe do sistema.

Segundo Tucker e Noonan (2009, p. 275) apud Silva (2019), “existem três tipos de relacionamento especialmente importantes”, conforme vemos a seguir.

- **Dependências:** representam os relacionamentos de utilização entre classes.
- **Generalizações:** representam o relacionamento entre classes mães e suas classes herdeiras.
- **Associações:** representam os relacionamentos estruturais entre objetos:
 - **agregações:** tipo de associação na qual o objeto dependente pode existir mesmo sem o objeto pai;
 - **composição:** tipo de associação na qual a existência do objeto dependente só faz sentido se o objeto pai existe.

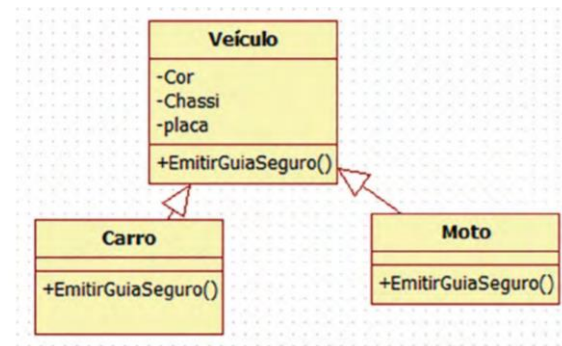


Figura 3. Relacionamento de generalização entre classes.

Fonte: Adaptada de Vasconcelos (2011).

- **abstract**: método abstrato, sem corpo.
- **final**: método não pode ser redefinido.
- **public**: método pode ser acessado por outras classes.
- **private**: método só pode ser acessado pela própria classe.
- **protected**: método pode ser acessado por classes dentro do mesmo pacote ou pelas subclasses.
- **static**: método compartilhado por todos os objetos da classe, com acesso a apenas campos estáticos.

- **final**: atributo é uma constante.
- **public**: atributo pode ser acessado por outras classes.
- **private**: atributo só pode ser acessado pela própria classe.
- **protected**: atributo pode ser acessado por classes dentro do mesmo pacote ou pelas subclasses.
- **static**: atributo compartilhado por todos os objetos da classe.

- **Empacotadoras:** há uma classe empacotadora para cada tipo primitivo em Java
 - boolean: Boolean
 - byte: Byte
 - short: Short
 - char: Character
 - int: Integer
 - long: Long
 - float: Float
 - double: Double

- Serviços das classes empacotadoras:

- Contrutor: `public Integer(int value)`

- `Integer numero = new Integer(5);`

- Contrutor: `public Integer(String s)`

- `Integer numero = new Integer("5");`

- Se o valor passado estiver no formato incorreto, por exemplo uma letra, é lançada uma exceção *NumberFormatException*.

- `public static Integer valueOf(String s)`

Método que retorna um objeto `Integer` que empacota um inteiro cujo valor é dado por `s`.

- `obj.intValue()`

Retorna um valor do tipo primitivo *int* empacotado pelo objeto `obj`, que é do tipo `Integer`.

- `public static int parseInt("5")`

Retorna um valor do tipo primitivo *int* que corresponde à `String` passada. É lançada uma exceção *NumberFormatException* se a `String` passada não for do formato de um número.

Classes Empacotadores (Wrappers) - Exemplo

```
public void testaWrapper() {  
    boolean b = true;  
    byte bt = 5;  
    char c = 'k';  
    short s = 10;  
    long l = 50;  
    int i = 20;  
    float f = 3.4f;  
    double d = 5.8;  
  
    Boolean B = new Boolean(b);  
    Byte BT = new Byte(bt);  
    Character C = new Character(c);  
    Short S = new Short(s);  
    Long L = new Long(l);  
    Integer I = new Integer(i);  
    Float F = new Float(f);  
    Double D = new Double(d);  
}
```


Classes Empacotadores (Wrappers) - Exemplo

```
//imprime o valor inteiro armazenado em I
System.out.println("Valor inteiro de I: " + I.intValue());

//imprime o valor inteiro armazenado em I2
Integer I2 = Integer.valueOf("700");
System.out.println("Valor inteiro de I2: " + I2.intValue());

//imprime o valor de i3
int i3 = Integer.parseInt("500");
System.out.println("Valor de i3: " + i3);

try{
    int i4 = Integer.parseInt("cinco");
    System.out.println("Valor de i4: " + i4);
}
catch (NumberFormatException e){
    System.out.println("Formato numérico inválido");
}
}
```

Propriedade de um módulo esconder ou tornar não manipulável uma parte do código.

O mecanismo de encapsulamento permite agrupar componentes que juntos fornecem um serviço específico e tornar apenas os aspectos relevantes visíveis para os clientes.

TAD é “uma estrutura de programação na qual uma determinada estrutura de dados é conhecida somente via as operações realizadas sobre os seus elementos de dados, sem que se identifique como a estrutura é codificada”

A programação orientada por objetos é o resultado do uso da abstração de dados no desenvolvimento de softwares.

- “Uma classe é um conceito OO que encapsula as abstrações de dados e procedimentos necessários para descrever o conteúdo e o comportamento de alguma entidade do mundo real”.

Pressman, 2002.

- *Uma classe é a implementação do objeto, seus serviços e propriedades destes serviços.*
- **Uma classe corresponde a um TAD (tipo abstrato de dados)**

Uma classe descreve uma categoria genérica.

Um **objeto** é uma instância de uma classe.

Objeto é uma estrutura computacional que representa um objeto do mundo real.

Uma instância de uma classe é uma estrutura de dados que representa um membro específico da categoria.

- Exemplo: classe: Aluno
- São objetos da classe Aluno: João, Paulo, Sílvia, Marina,
- Jurema, Felício.

Uma classe possui:

- **Atributos:** também denominados membros de dados, campos ou variáveis de instância.
 - Representam as **características** que os objetos da classe possuem.
- **Métodos:** também denominados membros de função ou operações.
 - Representam o **comportamento** que os objetos da classe possuem.

Um programa OO é constituído basicamente por

- definição das classes de objetos;
- um programa principal que dispara a comunicação entre os objetos.

Em Java, o programa principal encontra-se dentro de determinada classe e é representado pelo método **main**.

```
public class Principal {  
    public static void main (String[] args) {  
        //Código do programa principal  
    }  
}
```

Seja o contexto de automação bancária. Identificam-se as seguintes classes neste contexto: **cliente**, **agência**, **conta**, **conta corrente**, **conta poupança**, dentre outras.

A figura a seguir mostra a estrutura da classe `ContaCorrente`.

Atributos: numero, agência e saldo.

Métodos: depositar, sacar, consultar saldo.

Conta Corrente
- Numero : long - Agencia : int - Saldo : double
+ Depositar() : void + Sacar() : void + ConsultarSaldo() : double

O código a seguir mostra uma implementação possível para esta classe em Java.

```
public class ContaCorrente {  
    private long numero;  
    private int agencia;  
    private double saldo;  
  
    public ContaCorrente(long n, int ag) {  
        numero = n;  
        agencia = ag;  
        saldo = 0.0;  
    }  
}
```

```
public void sacar(double valor){  
    if (valor > 0)  
        saldo = saldo - valor;  
}  
public void depositar(double valor){  
    if (valor > 0)  
        saldo = saldo + valor;  
}  
  
public double consultarSaldo(){  
    return (saldo);  
}  
}
```

No exemplo, o método *ContaCorrente* (de mesmo nome da classe) é denominado construtor.

Um método construtor é utilizado para determinar o estado inicial do objeto.

Em Java, objetos são criados utilizando-se a palavra reservada `new`.

```
1 ContaCorrente minhaConta;  
2 minhaConta = new ContaCorrente(12345, 236);
```

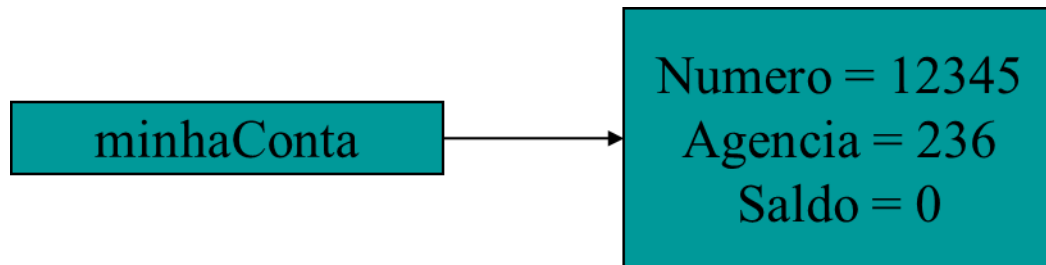
A linha 1 cria uma área na memória que é uma referência para um objeto da classe `ContaCorrente`.



minhaConta

```
1 ContaCorrente minhaConta;  
2 minhaConta = new ContaCorrente(12345, 236);
```

A linha 2 cria um objeto da classe *ContaCorrente* e o atribui a *minhaConta*;



- Programa orientados por objetos são constituídos por objetos que trocam mensagens entre si.
- O envio de uma *mensagem* a um objeto corresponde a **invocar um método** de tal objeto.

Em

```
minhaConta.depositar(350.00);
```

O método *depositar* do objeto *minhaConta* é invocado. Em outras palavras, é enviada uma mensagem para o objeto *minhaConta* para que este realize a operação *depositar*.

Referências

SILVA, Fabricio Machado da. **Paradigmas de programação**. SAGAH, 2019. ISBN digital: 9788533500426

Barnes, David e Kölling, M. **Programação Orientada a Objetos com Java**. São Paulo: Pearson Prentice Hall, 2004.

Deitel, H. M.; Deitel, P. J. **Java - Como Programar**. 6. ed. Prentice-Hall, 2005. Capítulo 4 e 5.

PRESSMAN, Roger S. **Engenharia de Software**. Rio de Janeiro: MacGraw Hill, 2002.