

Paradigmas

Módulo Básico





PARADIGMAS

PROGRAMAÇÃO ORIENTADA A OBJETOS

Objetivos

- ***Gerais:***
 - ***Apresentar a orientação a objetos utilizando uma linguagem de programação.***
- ***Específicos:***
 - ***Disseminar os princípios da Orientação a Objetos. Ao término deste módulo, o aluno deverá demonstrar compreensão dos aspectos fundamentais do paradigma.***

Orientação a Objetos

- ***Orientação a Objetos***

- Um ***modelo*** de ***programação*** ou ***paradigma*** de ***programação*** é um ***conjunto*** de ***princípios, ideias, conceitos*** e ***abstrações*** ***utilizado*** para o ***desenvolvimento*** de uma ***aplicação***



Orientação a Objetos

- ***Analogia***

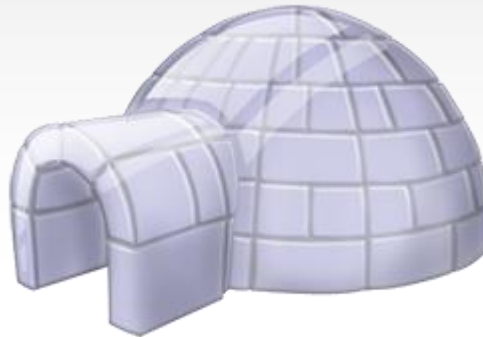
- Para ***entender melhor*** o que são os ***modelos de programação***, podemos ***compará-los*** com ***padrões arquiteturais utilizados*** por ***diferentes civilizações*** para ***construção de casas***
- As ***características ambientais definem*** quais ***técnicas devem*** ser ***adotadas*** para a ***construção das moradias***
- ***Analogamente, devemos escolher o modelo de programação*** mais ***adequado*** às ***necessidades*** da ***aplicação*** que ***queremos desenvolver***

Orientação a Objetos

- ***Analogia***



Cabana de Índio



Iglu



Casa Ocidental

Orientação a Objetos

- **Modelo de Programação**

- O *modelo de programação* mais *adotado* no *desenvolvimento de sistemas corporativos* é o *modelo orientado a objetos*
- Esse *modelo* é *utilizado* com o *intuito* de *obter alguns benefícios específicos* (um dos principais é *facilitar a manutenção das aplicações*)



Orientação a Objetos

- ***Modelo de Programação***

- *Em geral, os conceitos do modelo de POO diminuem a complexidade do desenvolvimento de sistemas que possuem as seguintes características:*

- *Sistemas com grande quantidade de funcionalidades desenvolvidos por uma equipe*
 - *Sistemas que serão utilizados por um longo período de tempo e sofrerão alterações constantes*



LÓGICA

- ***O que é um Programa?***

- ***Um dos maiores benefícios da utilização de computadores é a automatização de processos realizados manualmente por pessoas***

- **Exemplo:**

- ***Quando as apurações dos votos das eleições no Brasil eram realizadas manualmente, o tempo para obter os resultados era alto e havia probabilidade de uma falha humana***
 - ***Esse processo foi automatizado e hoje é realizado por computadores***
 - ***O tempo para obter os resultados e a chance de ocorrer uma falha humana diminuíram drasticamente***

- ***O que é um Programa?***
 - ***Basicamente, os computadores são capazes de executar instruções matemáticas mais rapidamente do que o homem***
 - ***Essa simples capacidade permite que eles resolvam problemas complexos de maneira mais eficiente***
 - ***Entretanto, eles **não** possuem a inteligência necessária para definir quais instruções devem ser executadas para resolver uma determinada tarefa***
 - ***Por outro lado, os seres humanos possuem essa inteligência***

- ***O que é um Programa?***

- *Dessa forma, uma **pessoa precisa definir** um **roteiro** com a **sequência** de **comandos necessários** para **realizar** uma **determinada tarefa** e **depois passar** para um **computador executar** esse **roteiro***
- *Formalmente, esses **roteiros** são **chamados** de **programas***



- ***Linguagem de Programação***

- As ***linguagens de programação*** tentam se ***aproximar*** das ***linguagens humanas***
- ***Confira*** o ***trecho*** de um ***código escrito*** com a ***linguagem*** de ***programação Java***

```
1 class OlaMundo {  
2     public static void main(String[] args) {  
3         System.out.println("Olá Mundo");  
4     }  
5 }
```

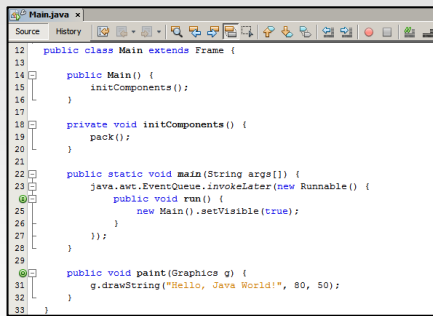
Código Java: OlaMundo.java

- Um ***arquivo contendo*** as ***instruções*** de um ***programa*** em ***linguagem de programação*** é chamado ***arquivo fonte***

- **Compilador**

- Os *computadores* *processam* *apenas* *instruções* em *linguagem de máquina*
- As *pessoas* *definem* as *instruções* em *linguagem de programação*
- *Torna-se necessário traduzir* o *código* *escrito* em *linguagem de programação* para um *código* em *linguagem de máquina* para que um *computador* *possa* *processar*
- *Essa tradução é realizada* por *programas* *especiais* chamados *compiladores*

- **Compilador**



```
12 public class Main extends Frame {  
13  
14     public Main() {  
15         initComponents();  
16     }  
17  
18     private void initComponents() {  
19         pack();  
20     }  
21  
22     public static void main(String args[]) {  
23         java.awt.EventQueue.invokeLater(new Runnable() {  
24             public void run() {  
25                 new Main().setVisible(true);  
26             }  
27         });  
28     }  
29  
30     public void paint(Graphics g) {  
31         g.drawString("Hello, Java World!", 80, 50);  
32     }  
33 }
```

Código Fonte

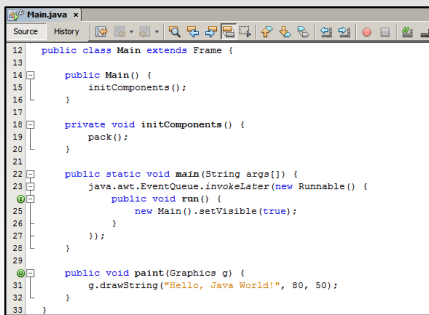


Processador



Não executa

- **Compilador**

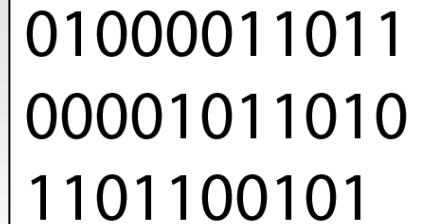


```
12 public class Main extends Frame {  
13  
14     public Main() {  
15         initComponents();  
16     }  
17  
18     private void initComponents() {  
19         pack();  
20     }  
21  
22     public static void main(String args[]) {  
23         java.awt.EventQueue.invokeLater(new Runnable() {  
24             public void run() {  
25                 new Main().setVisible(true);  
26             }  
27         });  
28     }  
29  
30     public void paint(Graphics g) {  
31         g.drawString("Hello, Java World!", 80, 50);  
32     }  
33 }
```

Código Fonte



Compilador

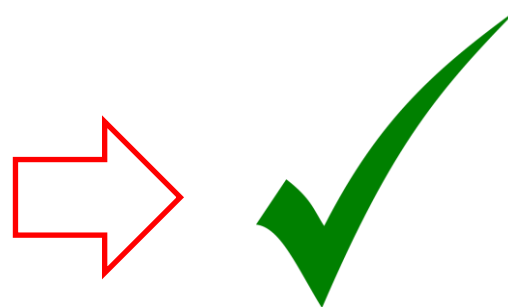


```
01000011011  
00001011010  
1101100101
```

Código de Máquina



Processador



Executa

- ***Máquinas Virtuais***

- A ***linguagem de máquina*** de um ***computador*** é ***definida*** pela ***arquitetura do processador*** desse ***computador***
- ***Há diversas arquiteturas diferentes (Intel, ARM, PowerPC, etc.) e cada uma delas define uma linguagem de máquina distinta***
- Um ***programa*** pode ***não executar*** em ***computadores*** com ***processadores de arquiteturas distintos***

- **Máquinas Virtuais**

- Os computadores são controlados por um **sistema operacional** que **oferece diversas bibliotecas necessárias** para o **desenvolvimento** das **aplicações** que podem ser **executadas através dele**
- Um **programa** pode **não executar** em **computadores** com **sistemas operacionais distintos**
- Para **determinar** se um **código** em **linguagem de máquina** **pode** ou **não** ser **executado** por um **computador**, devemos **considerar a arquitetura do processador** e o **SO**

- ***Máquinas Virtuais***



Programa 1



Plataforma 1



Executa



Programa 2



Plataforma 3

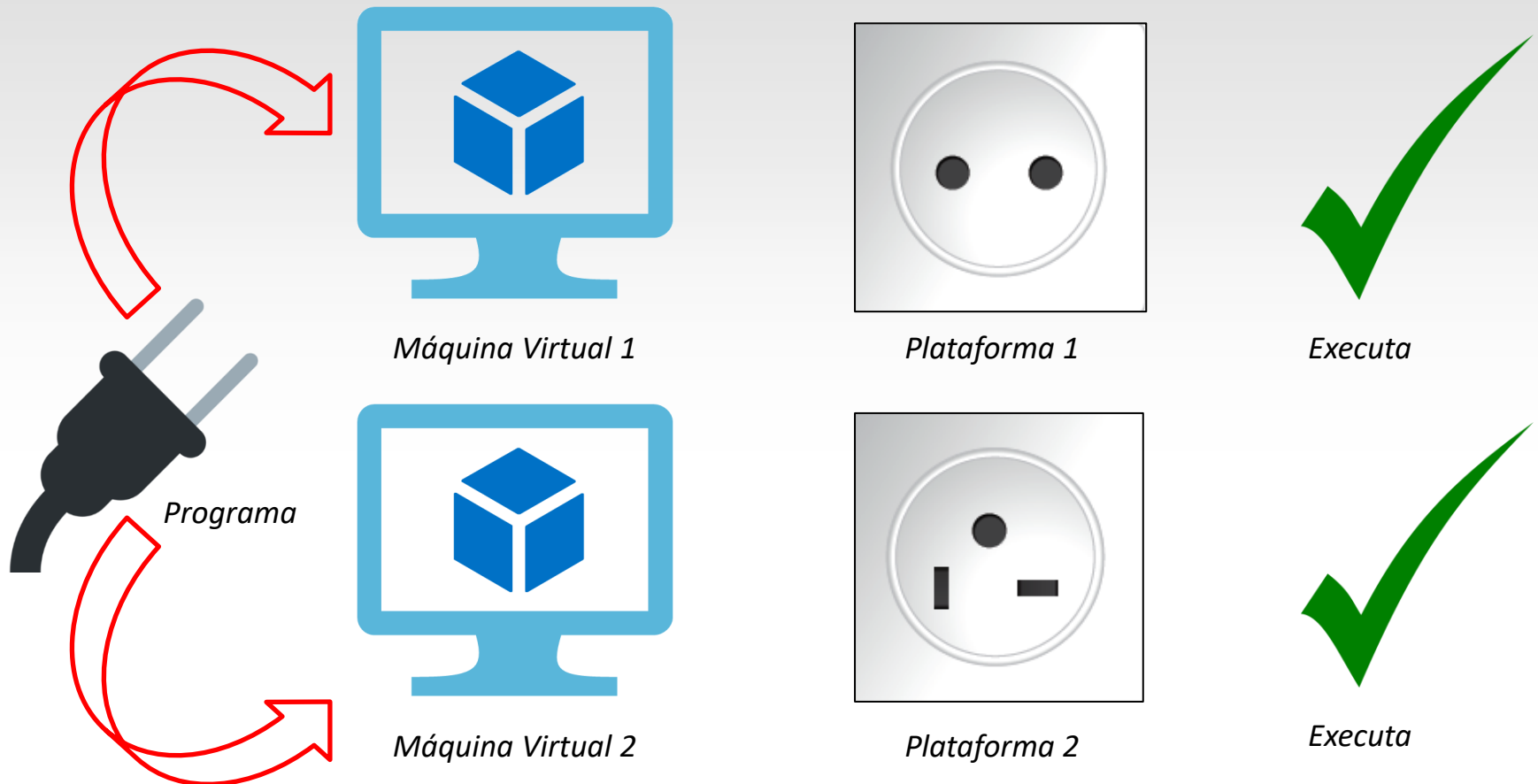


Não Executa

- **Máquinas Virtuais**

- Para tentar *solucionar* o *problema* do *desenvolvimento* de **aplicações multiplataforma**, surgiu o *conceito* de **máquina virtual**
- Uma *máquina virtual funciona* como uma *camada* a *mais* *entre* o *código compilado* e a *plataforma*
- Quando *compilamos* um *código fonte*, estamos *criando* um *executável* que a *máquina virtual* saberá *interpretar* e ela é *quem deverá traduzir* as *instruções* do seu *programa* para a *plataforma*

- ***Máquinas Virtuais***



- ***Máquinas Virtuais***

- ***Desvantagens:***

- ***Redução de performance***, já que a ***própria máquina virtual consome recursos do computador***
 - As ***instruções do programa*** são ***processadas primeiro pela máquina virtual e depois pelo computador***



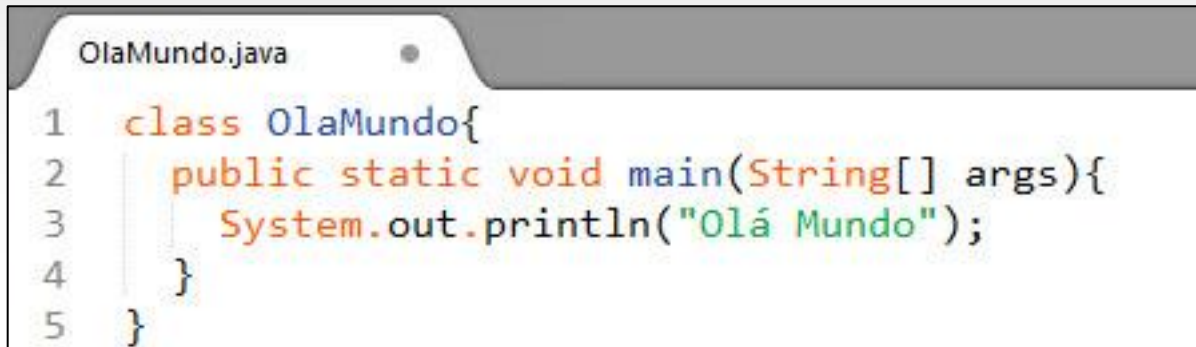
- ***Máquinas Virtuais***

- As *máquinas virtuais* podem ***aplicar otimizações*** que ***aumentam a performance*** da ***execução*** de um ***programa***
- Essas ***otimizações*** podem ***considerar informações*** geradas ***durante a execução***
- ***Exemplos:***
 - ***Quantidade de uso de memória RAM e do processador***
 - ***Quantidade de acesso ao disco rígido***
 - ***Quantidade de chamadas de rede***
 - ***Frequência de execução de um determinado trecho do programa***

- ***Máquinas Virtuais***

- Normalmente, as ***máquinas virtuais*** utilizam uma ***estratégia*** de ***compilação*** chamada ***Just-in-time compilation*** (JIT)
- Nessa abordagem, o ***código de máquina*** pode ser ***gerado diversas vezes durante*** o ***processamento*** de um ***programa*** com o ***intuito*** de ***melhorar*** a ***utilização*** dos ***recursos disponíveis*** em um ***determinado instante*** da ***execução***

- **Exemplo: Programa Java**
 - **Observe o código do exemplo de um programa escrito em Java que *exibe* uma *mensagem* na *tela***

A screenshot of a code editor window titled 'OlaMundo.java'. The code is written in Java and consists of five lines. Line 1: 'class OlaMundo{' (class keyword in orange, OlaMundo in blue, { in black). Line 2: ' public static void main(String[] args){' (public in orange, static in orange, void in orange, main in blue, String in orange, [] in black, args in blue, { in black). Line 3: ' System.out.println("Olá Mundo");' (System.out in orange, println in blue, "Olá Mundo" in green, ; in black). Line 4: ' }' (closing brace for main method). Line 5: '}' (closing brace for class).

```
OlaMundo.java
1  class OlaMundo{
2      public static void main(String[] args){
3          System.out.println("Olá Mundo");
4      }
5  }
```

Código Java: OlaMundo.java

- **Exemplo: Programa Java**
 - O *código fonte Java* deve ser *colocado* em *arquivos* com a *extensão* **.java**
 - *Não é necessário entender todo o código do exemplo*
 - *Basta saber que toda aplicação Java precisa ter um método especial chamado* **main** *para* **executar**
 - O *próximo passo é compilar o código fonte*, para *gerar* um **executável** que possa ser *processado* pela *máquina virtual do Java*

- **Exemplo: Programa Java**
 - O *compilador padrão* da *plataforma Java* (**javac**) pode ser *utilizado* para *compilar* esse *arquivo*
 - O *compilador* pode ser *executado* pelo *terminal*

```
C:\P00>javac OlaMundo.java
C:\P00>dir
O volume na unidade C não tem nome.
O Número de Série do Volume é AAAB-2973

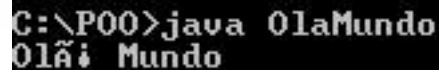
Pasta de C:\P00

13/02/2018  16:49    <DIR>          .
13/02/2018  16:49    <DIR>          ..
13/02/2018  16:49                422 OlaMundo.class
13/02/2018  16:44                105 OlaMundo.java
                        2 arquivo(s)          527 bytes
                        2 pasta(s)    105.985.036.288 bytes disponíveis
```

Terminal: Compilando

- **Exemplo: Programa Java**

- O **código gerado** pelo **compilador Java** é **armazenado** em **arquivos** com a **extensão .class**
- O **programa gerado** pelo **compilador** é **colocado** em um **arquivo** chamado **OlaMundo.class**, o qual **pode ser executado através** do **terminal**



```
C:\P00>java OlaMundo
Olá! Mundo
```

A screenshot of a Windows command prompt window. The prompt is 'C:\P00>'. The user has entered 'java OlaMundo' and pressed enter. The output is 'Olá! Mundo'.

Terminal: Executando

- **Método Main – Ponto de Entrada**

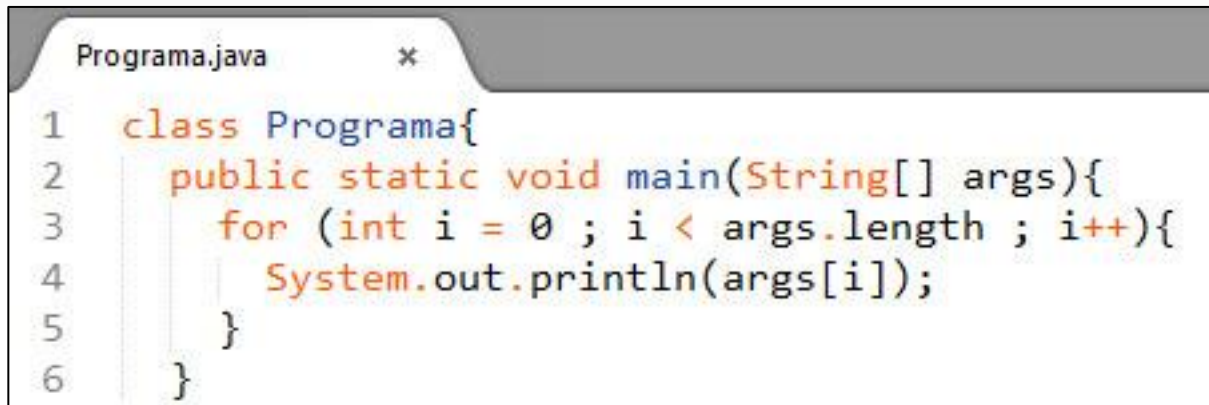
- Para um *programa Java executar*, é *necessário definir* um *método especial* para *ser o ponto de entrada do programa*
- O método **main** precisa ser **public**, **static**, **void** e *receber* um *array de strings* como *argumento*
- *Algumas das possíveis variações da assinatura do método main:*

```
1  static public void main ( String [] args )  
2  public static void main ( String [] args )  
3  public static void main ( String args [] )  
4  public static void main ( String [] parametros )
```

Código Java: Variações da Assinatura do Método Main

- **Método Main – Ponto de Entrada**

- Os *parâmetros* do método **main** são *passados* pela *linha de comando* e *podem* ser *manipulados dentro* do *programa*
- O *código abaixo* *exibe* cada *parâmetro recebido* em *uma linha diferente*

A screenshot of a code editor window titled 'Programa.java'. The code is written in Java and consists of a class named 'Programa' with a 'main' method. The 'main' method takes an array of strings 'args' as a parameter. It uses a 'for' loop to iterate over each element in 'args' and prints it to the console using 'System.out.println'. The code is as follows:

```
1 class Programa{
2     public static void main(String[] args){
3         for (int i = 0 ; i < args.length ; i++){
4             System.out.println(args[i]);
5         }
6     }
```

Código Java: Exibindo os parâmetros da linha de comando

- **Método Main – Ponto de Entrada**
 - *Os parâmetros devem ser passados imediatamente após o nome do programa*

```
C:\P00>javac Programa.java

C:\P00>dir
O volume na unidade C não tem nome.
O Número de Série do Volume é AAAB-2973

Pasta de C:\P00

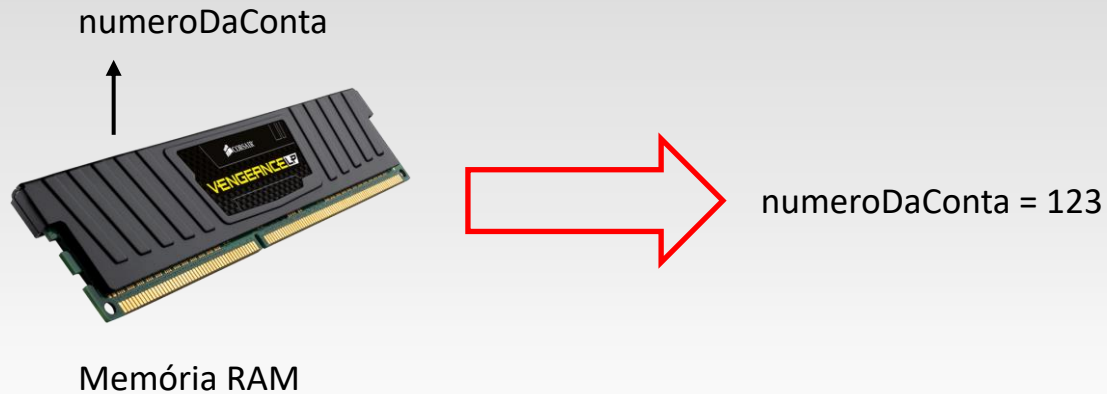
13/02/2018  17:10    <DIR>          .
13/02/2018  17:10    <DIR>          ..
13/02/2018  16:49             422 OláMundo.class
13/02/2018  17:08             105 OláMundo.java
13/02/2018  17:13             458 Programa.class
13/02/2018  17:10             161 Programa.java
                4 arquivo(s)          1.146 bytes
                2 pasta(s)    105.980.706.816 bytes disponíveis

C:\P00>java Programa Nome1 Nome2 Nome3
Nome1
Nome2
Nome3
```


- ***Variáveis***

- Basicamente, o que ***um programa faz*** é ***manipular dados***
- Normalmente, esses ***dados*** são ***armazenados*** em ***variáveis localizadas*** na ***memória RAM*** do ***computador***
- Uma ***variável*** pode ***armazenar dados*** de ***vários tipos***: ***números***, ***textos***, ***booleanos*** (***verdadeiro*** ou ***falso***), ***referências de objetos***
- ***Toda variável*** possui ***um nome*** que é ***utilizado quando*** a ***informação dentro*** da ***variável precisa*** ser ***manipulada*** pelo ***programa***

- ***Variáveis***



Processo de atribuição do valor numérico 123 à variável numeroDaConta

- **Declaração**

- Em Java, as **variáveis** devem ser **declaradas** para que **possam** ser **utilizadas**
- A **declaração** de **uma variável** envolve definir um **nome único** (*identificador*) **dentro** de um **escopo** e um **tipo de valor**
- As **variáveis** são **acessadas** pelos **nomes** e **armazenam valores compatíveis** com o seu **tipo**

```
1 // Uma variável do tipo int chamada numeroDaConta
2 int numeroDaConta;
3
4 // Uma variável do tipo double chamada precoDoProduto
5 double precoDoProduto;
```

- **Declaração**

- Em Java, as **variáveis** devem ser **declaradas** para que **possam** ser **utilizadas**

- A declaração de uma variável envolve definir um nome **de**

Uma linguagem de programação é dita **estaticamente tipada** quando ela exige que os tipos das variáveis sejam definidos **antes** da compilação.

Uma linguagem de programação é dita **fortemente tipada** quando ela exige que os valores armazenados em uma variável sejam compatíveis com o tipo da variável. **em**

```
3  
4 // Uma variável do tipo double chamada precoDoProduto  
5 double precoDoProduto;
```

- ***Declaração***

- Na ***convenção*** de ***nomes*** da ***linguagem Java***, os ***nomes*** das ***variáveis*** ***devem seguir*** o ***padrão*** ***camel case*** com a ***primeira letra minúscula*** (***lower camel case***)

- ***Exemplos:***

- nomeDoCliente
 - numeroDeAprovados

- ***Declaração***

- A ***declaração*** de ***uma variável*** pode ser ***realizada*** em ***qualquer linha*** de ***um bloco***
- ***Não*** é ***necessário declarar todas*** as ***variáveis*** no ***começo*** do ***bloco*** como ***acontece*** em ***algumas linguagens*** de ***programação***



- **Declaração**
 - **Exemplo (01):**
 - **Declaração de variáveis**

```
1 // Declaração com Inicialização
2 int numero = 10;
3
4 // Uso da variável
5 System.out.println(numero);
6
7 // Outra declaração com inicialização
8 double preco = 137.6;
9
10 // Uso da variável
11 System.out.println(preco);
```

Declaração em qualquer linha de um bloco

- ***Declaração***
 - ***Exemplo (02):***
 - ***Erro de compilação***

```
1 // Declaração
2 int numero = 10;
3
4 // Erro de compilação
5 int numero = 10;
```

Duas variáveis com o mesmo nome no mesmo bloco

- **Inicialização**

- *Toda variável deve ser inicializada antes de ser utilizada pela primeira vez*
- *Se a inicialização **não** for realizada, ocorrerá um **erro** de compilação*
- *A inicialização é realizada através do operador de atribuição = (igualdade)*

atribuição
↑
numero = 110; → valor
↓
identificador

- **Inicialização**

- **Exemplo:**

```
1 // Declarações
2 int numero;
3 double preco;
4
5 // Inicialização
6 numero = 10;
7
8 // Uso correto
9 System.out.println(numero);
10
11 // Erro de compilação
12 System.out.println(preco);
```

Código Java: Inicialização

- ***Tipos Primitivos***

- A **linguagem Java define** um **conjunto** de **tipos básicos** de **dados** que **são chamados tipos primitivos**

Tipo	Descrição	Tamanho
byte	Valor inteiro entre -128 e 127 (inclusivo)	1 byte
short	Valor inteiro entre -32.768 e 32.767 (inclusivo)	2 bytes
int	Valor inteiro entre -2.147.483.648 e 2.147.483.647 (inclusivo)	4 bytes
long	Valor inteiro entre -9.223.372.036.854.775.808 e 9.223.372.036.854.775.807 (inclusivo)	8 bytes
float	Valor com ponto flutuante entre $1,40129846432481707 \times 10^{-45}$ e $3,40282346638528860 \times 10^{38}$ (positivo ou negativo)	8 bytes

- ***Tipos Primitivos***

- A **linguagem Java define** um **conjunto** de **tipos básicos** de **dados** que **são chamados tipos primitivos**

Tipo	Descrição	Tamanho
double	Valor com ponto flutuante entre $4,94065645841246544 \times 10^{-324}$ e $1,79769313486231570 \times 10^{308}$ (positivo ou negativo)	8 bytes
boolean	true ou false	1 bit
char	Um único caractere Unicode de 16 bits. Valor inteiro e positivo entre 0 (ou '\u0000') e 65.535 (ou '\uffff')	2 bytes

Tipos primitivos de dados em Java (Parte 2)

- ***Operadores***

- ***Para manipular os valores das variáveis de um programa, devemos utilizar os operadores oferecidos pela linguagem***
- ***A linguagem Java possui diversos operadores e os principais são categorizados da seguinte forma:***
 - ***Aritmético*** (+ , - , * , / , %)
 - ***Atribuição*** (= , += , -= , /= , %=)
 - ***Relacional*** (== , != , < , <= , > , >=)
 - ***Lógico*** (&& , ||)

- **Aritmético**

- *Os operadores aritméticos funcionam de forma muito semelhante aos operadores na matemática*

- *Soma (+)*
 - *Subtração (-)*
 - *Multiplicação (*)*
 - *Divisão (/)*
 - *Módulo (%)*

- **Aritmético**

- **Exemplo:**

```
1  int umMaisUm = 1 + 1;           // umMaisUm = 2
2  int tresVezesDois = 3 * 2;       // tresVezesDois = 6
3  int quatroDivididoPor2 = 4 / 2;  // quatroDivididoPor2 = 2
4  int seisModuloCinco = 6 % 5;     // seisModuloCinco = 1
5  int x = 7;
6  x = x + 1 * 2;                   // x = 9
7  x = x - 3;                       // x = 6
8  x = x / (6 - 2 + (3 * 5) / (16 - 1) ); // x = 2
```

Código Java: Exemplo de uso dos operadores aritméticos

- **Atribuição**
 - *Os operadores de atribuição são:*
 - *Simples (=)*
 - *Incremental (+=)*
 - *Decremental (-=)*
 - *Multiplicativa (*=)*
 - *Divisória (/=)*
 - *Modular (%=)*

- **Atribuição**

- **Exemplo:**

```
1  int valor = 1;    // valor = 1
2  valor += 2;       // valor = 3
3  valor -= 1;       // valor = 2
4  valor *= 6;       // valor = 12
5  valor /= 3;       // valor = 4
6  valor %= 3;       // valor = 1
```

Código Java: Exemplo de uso dos operadores de atribuição

- **Exemplo:**

```
1  int valor = 1;    // valor = 1
2  valor = valor + 2; // valor = 3
3  valor = valor - 1; // valor = 2
4  valor = valor * 6; // valor = 12
5  valor = valor / 3; // valor = 4
6  valor = valor % 3; // valor = 1
```

Código Java: O mesmo exemplo anterior, usando os operadores aritméticos

- **Relacional**

- **Utilizado** para **determinar** a **relação existente entre** uma **variável** ou **valor** com **outra variável** ou **valor**
- As **operações realizadas** com os **operadores relacionais** **devolvem valores** do **tipo primitivo** “**boolean**”
- **Os operadores relacionais são:**
 - **Igualdade** (**==**)
 - **Diferença** (**!=**)
 - **Menor** (**<**)
 - **Menor ou igual** (**<=**)
 - **Maior** (**>**)
 - **Maior ou igual** (**>=**)

- **Relacional**

- **Exemplo:**

- **Manipulação de operadores relacionais**

```
1  int valor = 2;
2  boolean t = false ;
3  t = ( valor == 2); // t = true
4  t = ( valor != 2); // t = false
5  t = ( valor < 2);  // t = false
6  t = ( valor <= 2); // t = true
7  t = ( valor > 1);  // t = true
8  t = ( valor >= 1); // t = true
```

Código Java: Exemplo de uso dos operadores relacionais em Java

- **Lógico**
 - A *linguagem Java* permite verificar duas ou mais condições por meio de **operadores lógicos**
 - Os **operadores lógicos** devolvem valores do tipo primitivo **“boolean”**
 - Os **operadores lógicos** são:
 - “E” lógico (**&&**)
 - “OU” lógico (**||**)

- **Lógico**

- **Exemplo:**

- **Uso dos operadores lógicos em Java**

```
1  int valor = 30;
2  boolean teste = false ;
3  teste = valor < 40 && valor > 20;    // teste = true
4  teste = valor < 40 && valor > 30;    // teste = false
5  teste = valor > 30 || valor > 20;    // teste = true
6  teste = valor > 30 || valor < 20;    // teste = false
7  teste = valor < 50 && valor == 30;  // teste = true
```

Código Java: Exemplo de uso dos operadores lógicos em Java

- ***If-Else***

- ***if***: *permite verificar uma determinada condição e, decidir qual bloco de instruções deve ser executado*
- ***Exemplo***:

```
1  if ( preco < 0 ) {  
2      System.out.println ("O preço do produto não pode ser negativo");  
3  } else {  
4      System.out.println ("Produto cadastrado com sucesso");  
5  }
```

Código Java: Comando if

- ***If-Else***

- O comando ***if*** permite que valores booleanos sejam ***testados***
- Se o ***valor passado*** como ***parâmetro*** para o comando ***if*** for ***true***, o ***bloco*** do ***if*** é ***executado***
- Caso ***contrário***, o ***bloco*** do ***else*** é ***executado***
- O ***parâmetro passado*** para o comando ***if*** deve ser um ***valor booleano***, caso ***contrário*** o ***código não compila***
- O comando ***else*** e o seu ***bloco*** são ***opcionais***

- **While**

- *Permite definir quantas vezes um determinado trecho de código deve ser executado pelo computador*

- **Exemplo:**

- *Uso da estrutura de repetição “while”*

```
1  int contador = 0;
2
3  while ( contador < 100 ) {
4      System.out.println ("Bom Dia");
5      contador ++;
6  }
```

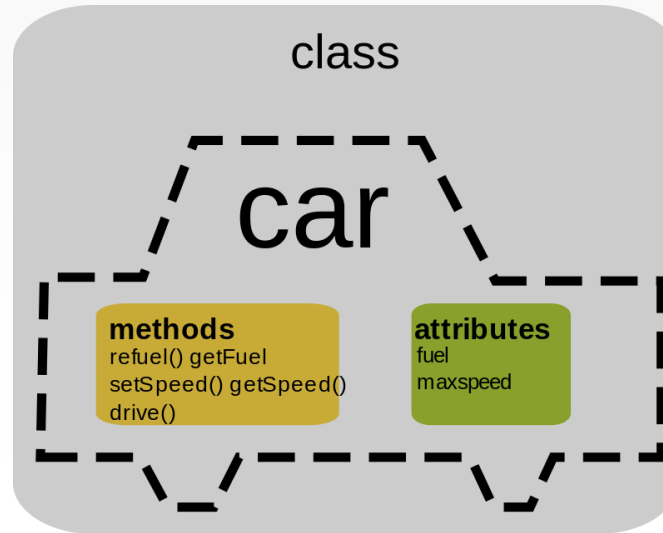
Código Java: Comando while

- **For**

- O comando **for** é análogo ao **while**
- A diferença entre esses dois comandos é que o **for** recebe três argumentos (**inicialização**, **condição** e **incremento/decremento**)
- **Exemplo:**
 - **Uso da estrutura de repetição “for”**

```
1  for ( int contador = 0; contador < 100; contador ++ ) {  
2      System.out.println ("Bom Dia");  
3  }
```

Código Java: Comando for



ORIENTAÇÃO A OBJETOS

Orientação a Objetos

- ***Domínio e Aplicação***

- **Domínio:**

- É ***composto*** pelas ***entidades, informações e processos relacionados*** a um ***determinado contexto***

- **Aplicação:**

- Pode ser ***desenvolvida*** para ***automatizar*** ou ***tornar factível*** as ***tarefas*** de ***um domínio***
 - Uma ***aplicação*** é, ***basicamente***, o “***reflexo***” de um ***determinado domínio***

Orientação a Objetos

- **Exemplo:**

- *Imagine que estamos interessados em desenvolver uma aplicação para facilitar as tarefas do cotidiano de uma instituição financeira*
- *Podemos identificar clientes, funcionários, agências e contas como entidades desse domínio (dessa forma, é possível identificar as informações e os processos relacionados a essas entidades)*



Orientação a Objetos

- *Exemplo:*



Orientação a Objetos

- **Observação:**

- A **identificação** dos **elementos** de um **domínio** é uma **tarefa complexa**, sobretudo por **depende fortemente** do **conhecimento** das **entidades**, **informações** e **processos** que o **compõem**
- Em geral, essas **pessoas** que possuem esse **conhecimento** ou parte dele estão em **contato constante** com o **domínio** e **não** possuem **conhecimentos técnicos** para **desenvolver** uma **aplicação**
- **Desenvolvedores** de **software** buscam **constantemente mecanismos** para tornar mais **eficiente** o **entendimento** dos **domínios** para os quais eles **devem desenvolver aplicações**

Orientação a Objetos

- ***Objetos, Atributos e Métodos:***
 - As ***entidades*** identificadas no ***domínio*** devem ser ***representadas*** de ***alguma forma dentro*** da ***aplicação correspondente***
 - Nas ***aplicações orientadas a objetos***, as ***entidades*** são ***representadas*** por ***objetos***
 - Uma ***aplicação orientada a objetos*** é ***composta*** por ***objetos***
 - Um ***objeto representa*** uma ***entidade*** do ***domínio***

Orientação a Objetos

- **Exemplo (1):**
 - Imagine que no **domínio** da **instituição financeira** exista um **cliente** chamado “**João**”
 - Dentro de uma **aplicação orientada a objetos** correspondente a esse **domínio**, **deverá existir** um **objeto** para **representar** esse **cliente**
 - **Suponha** que **algumas informações** do **cliente** “**João**” como **nome**, **data de nascimento** e **sexo** são **importantes** para a **instituição financeira**

Orientação a Objetos

- **Exemplo (2):**

- Já que esses **dados** são **relevantes** para o **domínio** da **aplicação**, o **objeto** que **representa** esse “**cliente**” deverá **possuir** essas **informações**
- Esses **dados** são **armazenados** nos **atributos** do **objeto** que **representa** o “**João**”
 - Um **atributo** é uma **variável** que **pertence** a um **objeto**
 - Os **dados** de um **objeto** são **armazenados** nos seus **atributos**

Orientação a Objetos

- **Exemplo (3):**
 - O **próprio objeto** deve **realizar operações** de **consulta** ou **alteração** dos **valores** de seus **atributos**
 - Essas **operações** são **definidas** nos **métodos** do **objeto**
 - Os **métodos** permitem **interações entre** os **objetos** de uma **aplicação**



Orientação a Objetos

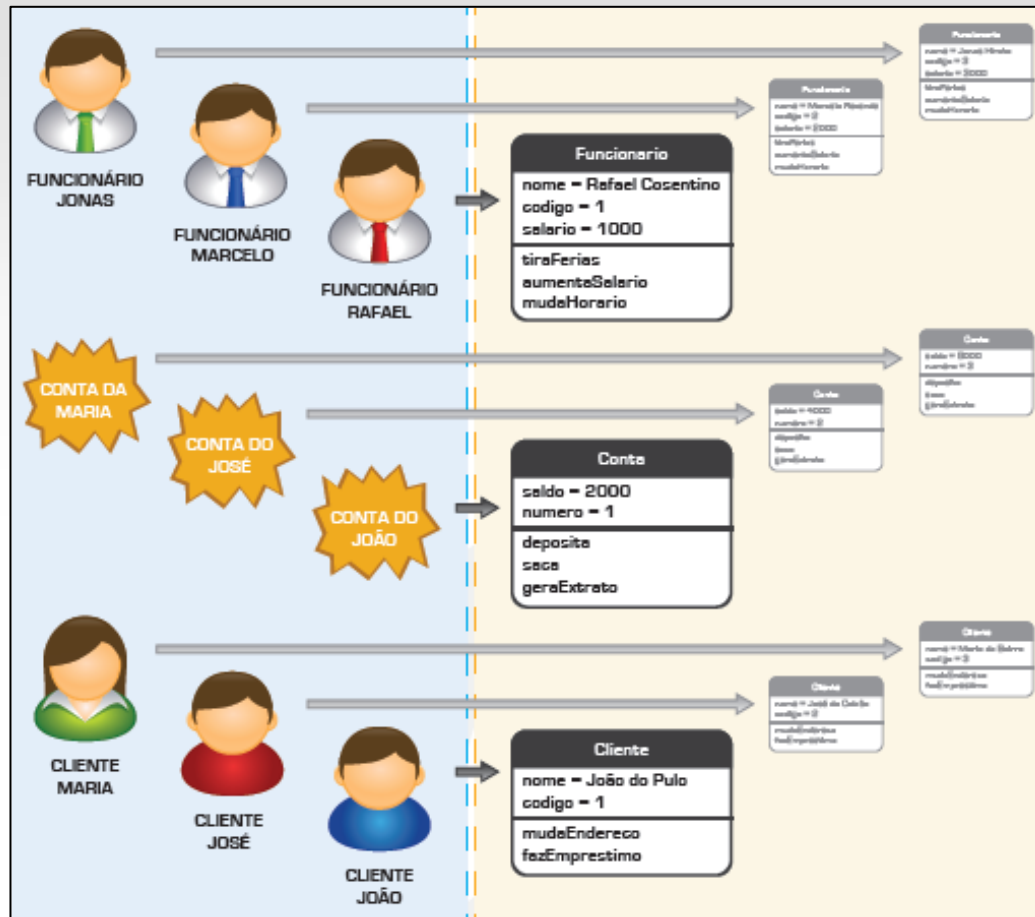
- **Exemplo (4):**
 - Quando um **cliente** *requisita* um **saque** *através* de um **caixa eletrônico**, o **objeto** que *representa* o **caixa eletrônico** deve *interagir* com o **objeto** que *representa* a **conta do cliente**
 - As **tarefas** que um **objeto** pode *realizar* são *definidas* pelos seus **métodos**
 - Um **objeto** é *composto* por **atributos** e **métodos**

Orientação a Objetos

- Exemplo (5):

Domínio Bancário

Aplicação



Orientação a Objetos

- **Observação:**

- Os **objetos** **não** **representam** apenas **coisas concretas** como os **clientes** da **instituição financeira**
- Eles também **devem** ser **utilizados** para **representar coisas abstratas** como uma **conta** de um **cliente** ou um **serviço** que a **instituição financeira** ofereça



Orientação a Objetos

- **Classes:**

- Antes da **criação** de um **objeto**, é necessário definir quais serão os seus **atributos** e **métodos**
- Essa **definição** é realizada através de uma **classe** elaborada por um desenvolvedor
- A partir de uma **classe**, podemos **construir objetos** na **memória** do **computador** que executa a nossa aplicação

Carro
+Marca: String +Cor: String +Placa: String +Velocidade_atual: Inteiro +Marcha_atual: Caractere +Freio_de_mao_puxado: booleano +chave_virada: booleano
+Ligar() +Acelerar_ate(velocidade: Inteiro) +Mudar_Marcha(marcha: Inteiro) +Parar()

Orientação a Objetos

- **Classes:**

- Podemos **representar** uma **classe** através de **diagramas da UML**
- O **diagrama UML** de uma **classe** é **composto** pelo **nome** da mesma e pelos **atributos** e **métodos** que ela define
- Todos os **objetos** criados a **partir** da **classe** “**Conta**” terão **atributos** e **métodos**, conforme apresentados pelo diagrama

Conta
numero
saldo
limite
saca()
deposita()
imprimeExtrato()

Diagrama UML da classe Conta

Orientação a Objetos

- ***Classes (Analogia):***

- Um ***objeto*** é como se ***fosse*** uma ***casa*** ou um ***prédio***
- Para ser ***construído***, ***precisa*** de um ***espaço físico***
- No caso dos ***objetos***, esse ***espaço físico*** é algum trecho vago da ***memória*** do ***computador*** que ***executa*** a ***aplicação***
- No caso das ***casas*** e dos ***prédios***, o ***espaço físico*** é ***algum terreno vazio***



Orientação a Objetos

- ***Classes (Analogia):***

- *Um **prédio** é construído a partir de uma **planta** criada por um engenheiro ou arquiteto*
- *Para criar um **objeto**, é necessário algo semelhante a uma **planta** para que sejam “**desenhados**” os **atributos** e **métodos** que o **objeto** deve possuir*
- *Em **orientação a objetos**, a “**planta**” de um **objeto** é o que chamamos de **classe***

Orientação a Objetos

- **Classes (Analogia):**
 - Uma **classe** funciona como uma “**receita**” para **criar objetos**
 - Vários **objetos** podem ser **criados** a partir de uma **única classe**



Diversas casas construídas a partir da mesma planta

Orientação a Objetos

- ***Classes em Java:***
 - Criando a classe **Conta**

```
1  
2  
3  class Conta {  
4      public double saldo;  
5      public double limite;  
6      public int numero;  
7  }
```

Código Java: Conta.java

Orientação a Objetos

- ***Classes em Java:***

- A classe Java **Conta** é declarada utilizando a palavra reservada **class**
- No **corpo** dessa **classe**, são declaradas **três variáveis** que são os **atributos** que os **objetos** possuirão
- Como a linguagem Java é estaticamente tipada, os **tipos** dos **atributos** são **definidos** no **código**
- O **modificador** **public** é **adicionado** em **cada atributo** para que eles possam ser **acessados** a **partir** de qualquer **ponto** do **código** (**escopo**)

Orientação a Objetos

- ***Criando Objetos em Java:***
 - *Posteriormente, definir a classe “Conta”, podemos criar objetos a partir dela*
 - *Esses objetos devem ser alocados na memória RAM do computador*
 - *Todo o processo de alocação do objeto na memória é gerenciado pela máquina virtual*

Orientação a Objetos

- ***Criando Objetos em Java:***
 - Criando **novos objetos** com o comando “**new**”

```
1  
2  
3 class TestaConta {  
4     public static void main(String[] args) {  
5         //criando objetos  
6         new Conta(); → 1º  
7         new Conta(); → 2º  
8         new Conta(); → 3º  
9     }  
10 }
```

Código Java: TestaConta.java

Orientação a Objetos

- **Referências:**

- Todo **objeto** possui uma **referência**
- A **referência** de um **objeto** é a **única maneira** de **acessar** os seus **atributos** e **métodos**
- Dessa forma, devemos **guardar** as **referências** dos **objetos** que **desejamos utilizar**
- **Analogia:**
 - Podemos **comparar** a **referência** de um **objeto** com o **endereço** de **memória** desse **objeto**
 - Uma **referência** é o **elemento** que **permite** que um **determinado objeto** seja **acessado**

Orientação a Objetos

- *Referências em Java:*

- Ao *utilizar* o comando “**new**”, um *objeto* é *alocado* em *algum lugar* da *memória*
- Para *acessar* esse *objeto*, é *necessário utilizar* sua *referência*
- O comando “**new**” *retorna* a *referência* do *objeto* que foi *criado*
- Para *guardar* as *referências retornadas* pelo comando “**new**”, é *necessário utilizar variáveis não primitivas*

Orientação a Objetos

- *Referências em Java:*

Conta **referencia** = new Conta();

- A *variável* “**referencia**” *receberá* a *referência* do *objeto criado* pelo *comando* “**new**”
- *Essa variável* é do *tipo Conta* (isso significa que ela *só pode armazenar referências de objetos do tipo Conta*)

Orientação a Objetos

- ***Manipulando Atributos:***

- É possível **alterar** ou **acessar** os valores armazenados nos atributos de um objeto se tivermos a referência a esse objeto
- Os atributos são acessados pelo **nome**
- Em Java, a **sintaxe** para **acessar** um atributo utiliza o operador “.” (ponto)

Orientação a Objetos

- ***Manipulando Atributos:***

```
Conta referencia = new Conta();  
  
referencia.saldo = 1000.00;  
referencia.limite = 500.00;  
referencia.numero = 1;  
  
System.out.println("Saldo: " + referencia.saldo);  
System.out.println("Limite: " + referencia.limite);  
System.out.println("Número: " + referencia.numero);
```

- Os atributos “*saldo*”, “*limite*” e “*numero*” foram **atribuídos** com os **valores** *1000.00*, *500.00* e *1*, respectivamente
- Na sequência, foram apresentados na tela pelo comando *System.out.println*

Orientação a Objetos

- **Valores Padrão:**

- É permitido *instanciar* um **objeto** e *utilizar* seus **atributos** *sem inicializa-los explicitamente* (os **atributos** são *inicializados* com **valores padrão**)
- Os **atributos** de *tipos numéricos* são *inicializados* com **0** (zero), os **atributos** do *tipo boolean* são *inicializados* com **false** e os *demais atributos* com **null** (referência vazia)

Orientação a Objetos

- **Valores Padrão:**

- A **inicialização** dos **atributos** com os **valores padrão** ocorre na **instanciação**, ou seja, quando o **comando** “**new**” é **utilizado**
- Dessa forma, todo **objeto** “**nasce**” com os **valores padrão**
- Em alguns casos, é necessário **trocar** esses **valores**
- Para **trocar** o **valor padrão** de um **atributo**, **devemos inicializa-lo** na **declaração**

Orientação a Objetos

- **Valores Padrão:**

- **Exemplo (1):**

- **Suponha** que o **limite padrão** das **contas** de um **banco** seja **R\$ 500,00**

```
1  
2  
3  class Conta {  
4      public double limite = 500;  
5  }
```

Código Java: Conta.java

Orientação a Objetos

- **Valores Padrão:**

- Exemplo (2):

- **Suponha** que o **limite padrão** das **contas** de um **banco** seja **R\$ 500,00**

```
1
2
3  class TestaConta {
4      public static void main(String[] args) {
5          Conta referencia = new Conta();
6
7          System.out.println("Limite: " + referencia.limite);
8      }
9  }
```

Código Java: TestaConta.java

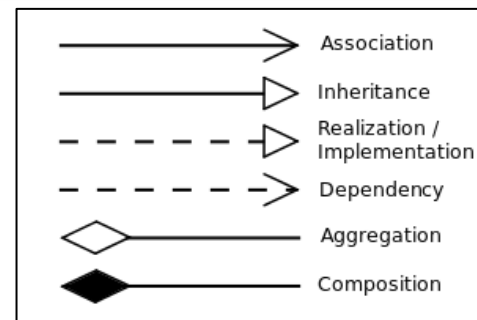
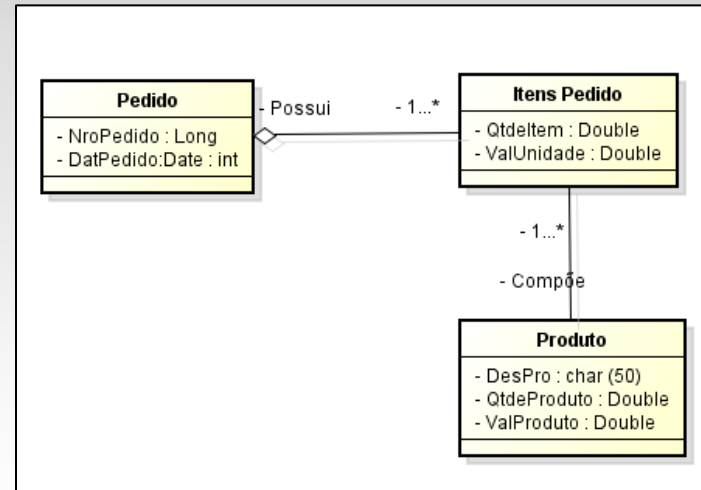
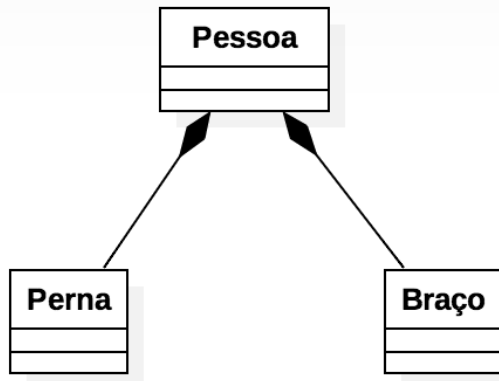


RELACIONAMENTOS

Agregação

- **Relacionamentos:**

- **Agregação**
- **Associação**
- **Composição**



- **Exemplo:**

- *Imagine* que um **determinado cliente adquira** um **cartão de crédito**
- *Dentro de um sistema bancário, deverá existir* um **objeto** que **represente** o **cliente** e outro que **represente** o **cartão de crédito**
- *Para expressar a relação entre o cliente e o cartão de crédito, algum vínculo entre esses dois objetos deve ser estabelecidos*

Agregação

- Exemplo:

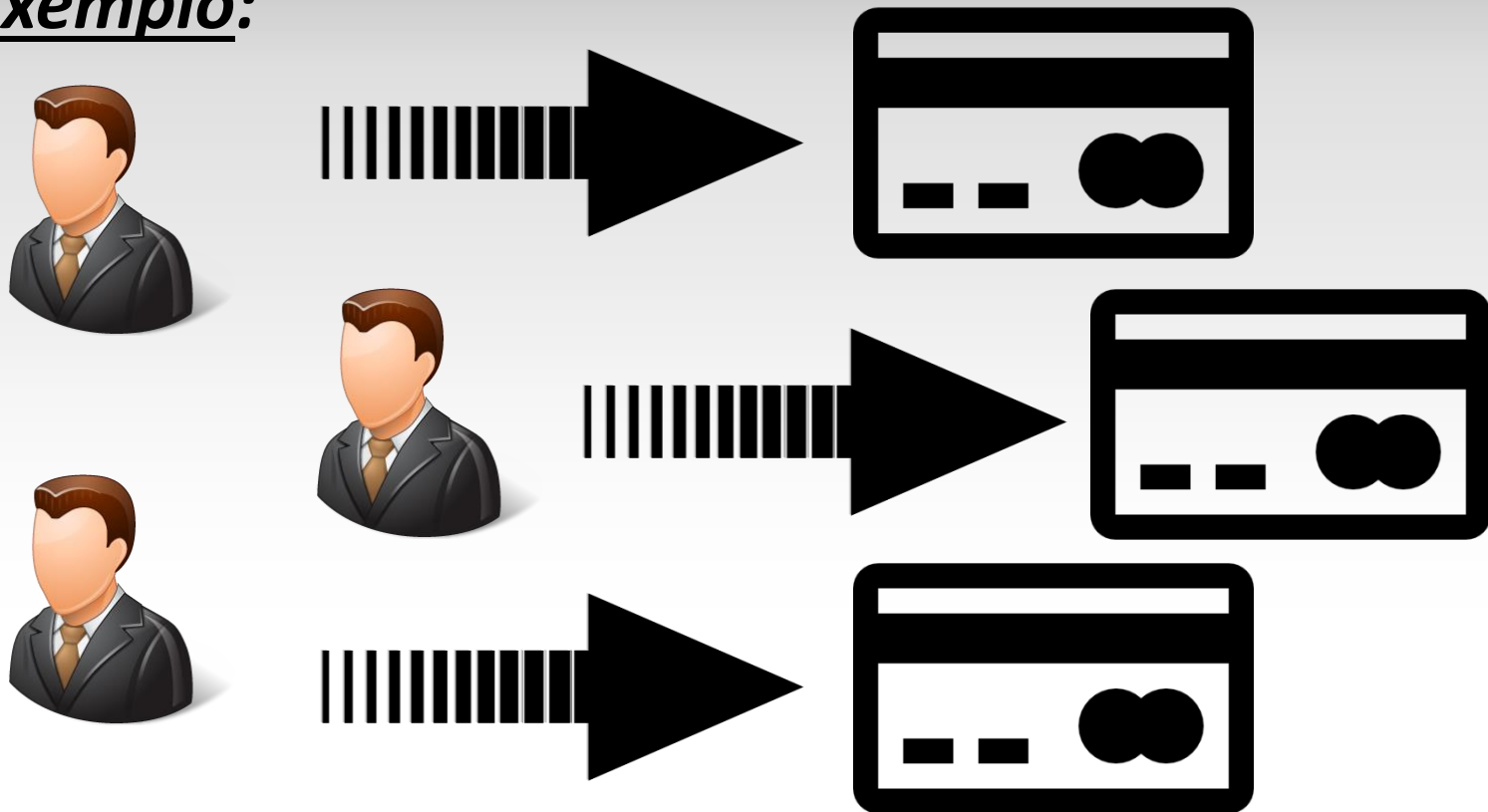


Figura 1: Clientes e cartões de crédito

Agregação

- **Exemplo:**

- Dessa maneira, **duas classes** devem ser **criadas**:

- Uma para definir os **atributos** e **métodos** dos **clientes**
 - Uma para definir os **atributos** e **métodos** dos **cartões de crédito**



Agregação

- Exemplo (Agregação):

- Para exemplificar o **relacionamento existente** entre **cliente** e **cartão de crédito**, adicionamos um **atributo tipo Cliente** na **classe CartaDeCredito**:

```
1  
2  
3 public class Cliente {  
4     public String nome;  
5 }
```

Código Java: Cliente.java

```
1  
2  
3 public class CartaoDeCredito {  
4     public int numero;  
5     public String dataDeValidade;  
6     public Cliente cliente;  
7 }
```

Código Java: CartaoDeCredito.java

Agregação

- **Agregação:**
 - Esse *tipo de relacionamento* é chamado de **Agregação**
 - Há uma *notação gráfica* na *linguagem UML* para *representar* uma *agregação*

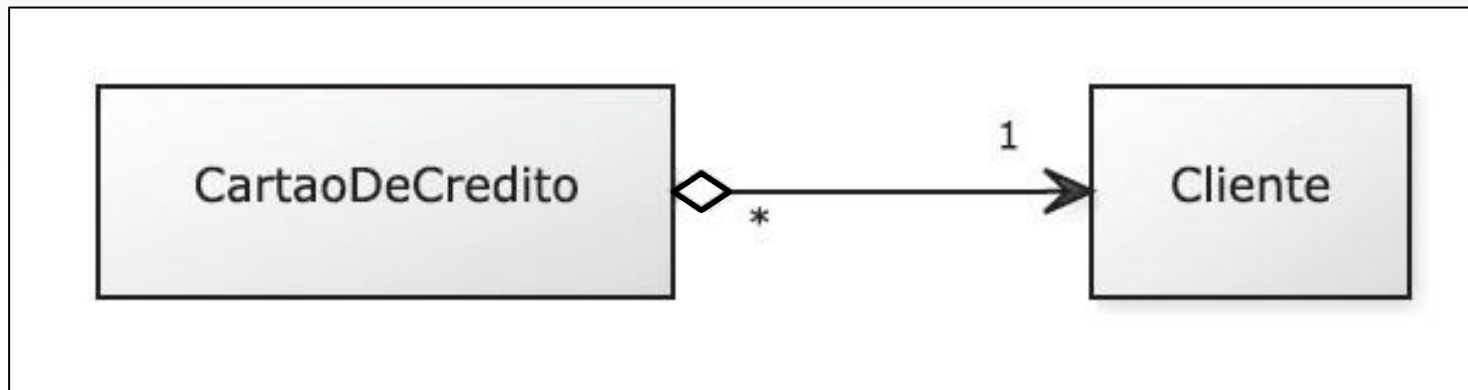


Figura 2: Agregação entre clientes e cartões de crédito

- **Exemplo (Agregação):**
 - O *relacionamento* entre *um objeto* da classe **Cliente** e *um objeto* da classe **CartaoDeCredito** apenas é *concretizado* quanto a *referência* do *objeto* **Cliente** é *armazenada* no *atributo* **cliente** do *objeto* da classe **CartaoDeCredito**
 - Depois de *relacionarmos*, podemos *acessar*, *indiretamente*, os *atributos* do *cliente* por meio da *referência* do *objeto* da classe **CartaoDeCredito**

Aggregação

- *Exemplo (Aggregação):*

```
1  
2  
3 public class TestaAggregacao {  
4     public static void main(String[] args) {  
5         // Criando um objeto de cada classe  
6         CartaoDeCredito cdc = new CartaoDeCredito();  
7         Cliente c = new Cliente();  
8  
9         // Vinculando os objetos  
10        cdc.cliente = c;  
11  
12        // Acessando o nome do cliente  
13        cdc.cliente.nome = "Luis Inacio";  
14    }  
15 }
```

Código Java: Criando uma agregação

Agregação

- Exemplo (Agregação):**

Cartão
numero = 123
dataValidade = 01/2020
cliente = null
Métodos

Cliente
nome = Luis Inacio
cpf = 123.456.789-00
Métodos

Cartão
numero = 123
dataValidade = 01/2020
cliente
Métodos

Cliente
nome = Luis Inacio
cpf = 123.456.789-00
Métodos



Figura 3: Conectando um cliente e um cartão

Referências

SANTOS, Rafael; Introdução à programação orientada a objetos usando Java / Campus; Rio de Janeiro, 2003.

ALVES, William Pereira; Java 2 - Programação Multiplataforma. – Ed. Érica, 2006.

HORSTMANN, Cay; Conceitos de Computação com o Essencial de Java. Ed. Bookman – Porto Alegre, 2005.