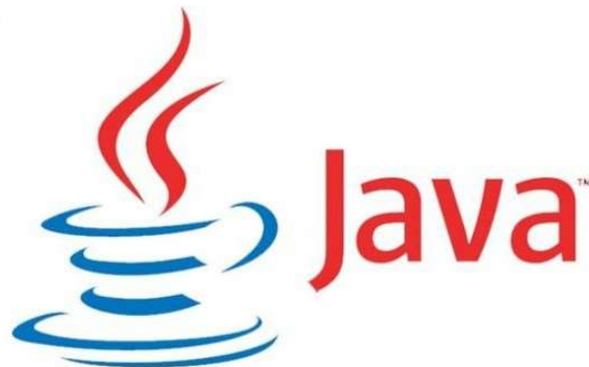

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
DEPARTAMENTO DE COMPUTAÇÃO
TÉCNICAS DE PROGRAMAÇÃO I CMP 1048
PROF. MSC. ANIBAL SANTOS JUKEMURA



PROGRAMAÇÃO ORIENTADA A OBJETOS [JAVA]



Agenda



- Orientação a Objetos com Java
- Métodos
- Classes / Objetos
- Exemplo
- Exercícios

Paradigma de Orientação a Objetos



- Orientação a objetos é uma maneira de programar que ajuda na organização e resolve muitos problemas enfrentados pela programação procedural.
- O problema do paradigma procedural é que não existe uma forma simples de criar conexão forte entre dados e funcionalidades. No paradigma orientado a objetos é muito fácil ter essa conexão através dos recursos da própria linguagem.

Paradigma de Orientação a Objetos



- Orientação a objetos vai te ajudar em muito em se organizar e escrever menos, além de concentrar as responsabilidades nos pontos certos, flexibilizando sua aplicação, **encapsulando** a lógica de negócios.
- Outra enorme vantagem, onde você realmente vai economizar montanhas de código, é o **polimorfismo das referências**.

Introdução: Dicionário OO

- Objeto: pode ser definido como uma unidade de *software* constituído de atributos (dados) e de métodos (códigos de funções) que atuam sobre os dados, sendo os representantes das classes.
- Atributos: qualquer propriedade, qualidade ou característica que possa ser atribuída, podendo ser acessado pelo serviços.
- Funções: Blocos de códigos para especificar comportamentos e/ou ações

Introdução: Dicionário OO

- Serviços: atividade executada para permitir o acesso a alguns recursos, em OO é um comportamento específico que um objeto deve exibir.
- Encapsulamento: restrição de escopo ou visibilidade dos dados do aplicativo que poderão ser acessados pelos serviços.

Introdução: Dicionário OO

- Passagem de mensagem: Possibilita a comunicação entre objetos.
- Herança: mecanismo para trabalhar com similaridades entre as classes como, por exemplo, mamíferos são os homens e o macacos, podendo considerá-los como herdeiros dos atributos da classe mamíferos.

Introdução: Dicionário OO

- **Classes:** Em linguagem Java, a unidade de programação é a classe, a partir da qual os objetos são **instanciados** (criados).



Introdução: Dicionário OO

- Superclasse: Classe base para as subclasses (Mamíferos).
- Subclasse: herdeiros das superclasses (Homens e Macacos)
- Superclasse Abstrata: Finalidade apenas de determinação de uma superclasse, não permitindo o manuseio pelas subclasse.

Introdução: Dicionário OO

- Associação: Conexão de idéias através das classes como, por exemplo, diferenças finitas podem resolver problemas de eletromagnetismo.
- Polimorfismo: Relaciona as diferentes formas de um objeto.

Janela ()	
Janela (1 x 2, 2)	
Janela(1x2, 2, Azul)	

Paradigma de Orientação a Objetos



```
6 package javaapplication1;
7
8  /**...4 linhas */
12
13 class OlaMundo{
14     public void mensagem()
15     {
16         System.out.println("HELLO WORLD!");
17     }
18 }
19
20 public class JavaApplication1 {
21
22     public static void main(String[] args) {
23         OlaMundo objOlaMundo = new OlaMundo();
24         objOlaMundo.mensagem();
25     }
26
27 }
```



MÉTODOS

MÉTODOS INDEPENDENTES

- Métodos (bem como classes) ajudam a modularizar um programa separando suas tarefas em unidades autocontidas.
- As instruções no corpo dos métodos são escritas apenas uma vez, permanecem ocultas de outros métodos e podem ser reutilizadas a partir de várias localizações em um programa.
- Motivação: modularizar para adotar a abordagem dividir para conquistar
- Outra é a capacidade de reutilização de software — o uso de classes e métodos existentes como blocos de construção para criar novos programas.

STATIC

- Métodos podem realizar uma tarefa que não depende de um objeto. Esse método se aplica à classe em que é declarado como um todo e é conhecido como método **static** ou **método de classe**.
- É comum que as classes contenham métodos **static** convenientes para realizar tarefas corriqueiras.
- Para declarar um método como **static**, coloque a palavra-chave **static** antes do tipo de retorno na declaração do método.

STATIC

Declarações:

public static void metodo1 ()

() : esse método não possui
parâmetros de entrada

void: esse método não
retorna valores

static: método
independente de objeto

Modificador de acesso

STATIC

Declarações:

public static int metodo1 (char a)

Esse método possui um parâmetro char a

int: esse método deve retornar um valor int.
Use: return

static: método independente de objeto

Modificador de acesso

Paradigma de Orientação a Objetos

STATIC

Exemplo:



```
public class Somatorio {  
  
    public static int Somar (int total)  
    {  
        int i;  
        int s=0;  
        for (i=1;i<=total;i++)  
            s = s + i;  
        return s;  
    }  
  
    public static void main(String[] args) {  
        int n=100;  
        int soma=0;  
  
        System.out.println("Soma dos " + n + " primeiros numeros naturais:" );  
        soma=Somar(n);  
        System.out.println("Soma dos 100 =" + soma);  
    }  
}
```

Método static Somar

Como chamar (executar) um método, a partir de outro método

MÉTODOS INDEPENDENTES

Por que o método main é declarado static?

- Ao executar a Java Virtual Machine (JVM) com o comando java, a JVM tenta invocar o método **main** da classe que você especifica.
- Declarar main como static permite que a JVM invoque main sem criar uma instância da classe.

MÉTODOS INDEPENDENTES

EM SALA: métodos com múltiplos parâmetros

- Criar um método chamado máximo. Esse método receberá três parâmetros double x, y, z. Use o método para definir e retornar qual dos três parâmetros é o maior.

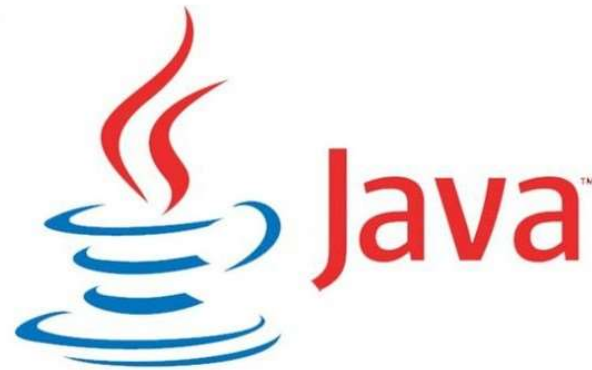


MÉTODOS INDEPENDENTES

EM SALA: métodos para um Menu

- Usar métodos independentes para simular um menu.





NÚMEROS ALEATÓRIOS

RANDOM

PACOTE:

```
import java.util.Random
```

DECLARAÇÃO:

```
Random gerador = new Random();
```

USO:

```
gerador.nextBoolean();  
gerador.nextInt();  
gerador.nextDouble();  
gerador.nextFloat();  
gerador.nextLong();
```

SECURERANDOM

PACOTE:

```
import java.security.SecureRandom;
```

DECLARAÇÃO:

```
SecureRandom gerador = new SecureRandom();
```

USO:

```
gerador.nextBoolean();  
gerador.nextInt();  
gerador.nextDouble();  
gerador.nextFloat();  
gerador.nextLong();
```

RANDOM x SECURERANDOM

EM SALA:

- Criar um método que simule uma jogada de dados. Informe por parâmetro, a quantidade de jogadas desejadas. O método não precisa retornar valor (void), mas deve mostrar os lances gerados com Random ou SecureRandom.



Exemplo



Problema: Objeto Aluno. O que é?

- Um aluno tem, por exemplo:
- Nome
- Nota
- Matrícula
- Mensalidade
- Desconto



Problema: listagem de alunos

- Uma certa instituição de ensino está incentivando os seus alunos a participarem de eventos acadêmicos em troca de créditos para obter desconto nas mensalidades.
 - Para participar, o aluno deve comparecer em algum dos eventos cadastrados na instituição e depois escrever um relatório sobre o conteúdo apresentado no evento. Este relatório será avaliado por um professor e receberá uma pontuação de 0.0 a 10.0.
 - A instituição quer manter uma listagem dos alunos que entregaram relatórios. Cada relatório entregue por um aluno corresponde a uma entrada na lista.
- Notas de 3.0 a 5.0 recebem desconto de 10% na mensalidade.
 - Notas de 5.1 a 7.0 recebem desconto de 30% na mensalidade.
 - Notas de 7.1 a 10.0 recebem desconto de 50% na mensalidade.

Exemplo



Problema: listagem de alunos

???



Aluno	Nota
Rafael	10
Ana	7
Paulo	5.5
Douglas	4.5
José	6
Mauro	8
Sérgio	9
Daniel	6.5
Maria	8.5
Pedro	3.5

Problema: como fazer um programa em Java que represente cada desconto da lista de alunos avaliada pelo professor?

Exemplo



Solução Homer Simpson !

- Usar VETORES!
- **Vetor Alunos** = guarda o nome de cada aluno
- **Vetor Notas** = guarda nota de cada aluno
- **Vetor Desconto** = guarda desconto para cada um conforme as notas



Exemplo



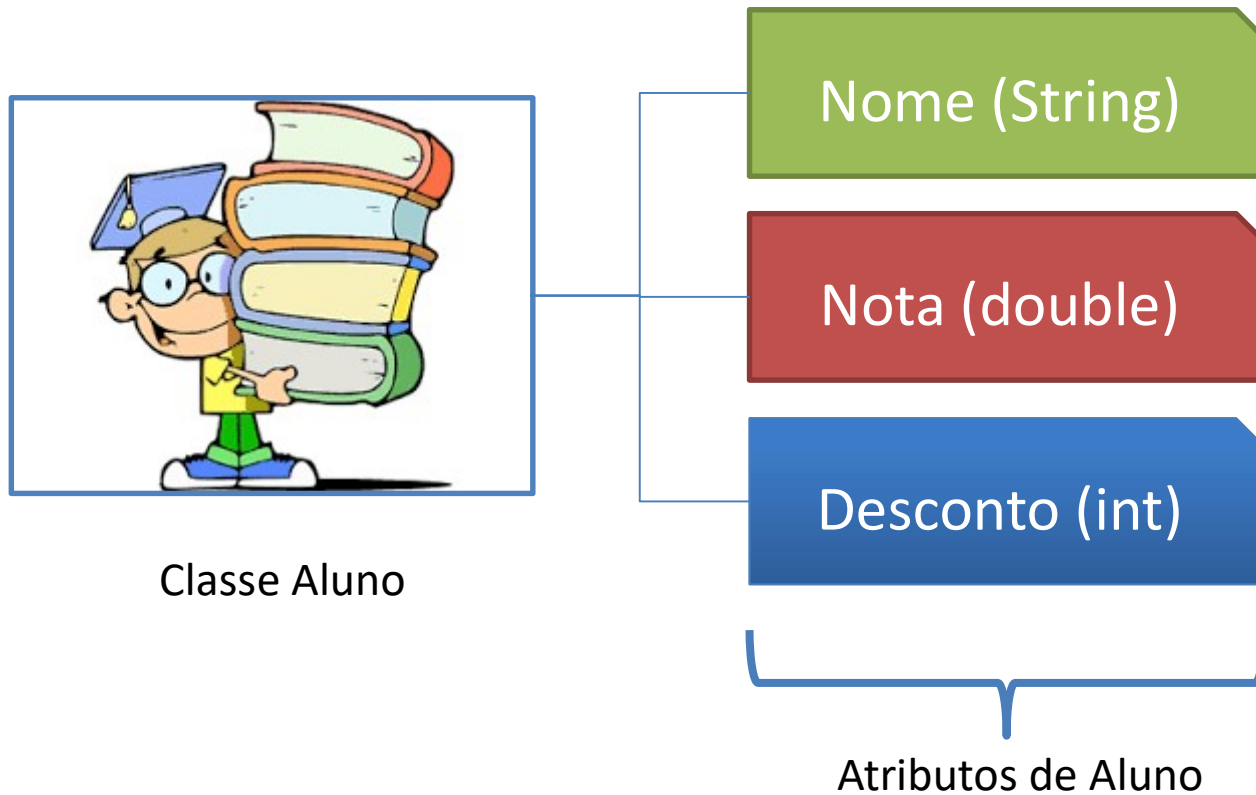
Solução Homer Simpson !

```
public class JavaApplication1 {  
  
    public static void main(String[] args) {  
        String Nome[] = {"Rafael", "Ana", "Paulo", "Douglas", "Jose", "Mauro",  
                        "Sergio", "Daniel", "Maria", "Pedro"};  
  
        double Nota[] = new double[] {10, 7, 5.5, 4.5, 6, 8, 9, 6.5, 8.5, 3.5};  
  
        int Desconto[] = new int[10];  
  
        int i;  
        for (i=0; i<10; i++)  
        {  
            if (Nota[i] >= 3 && Nota[i] <= 5)  
                Desconto[i] = 10;  
            else if (Nota[i] > 5 && Nota[i] <= 7)  
                Desconto[i] = 30;  
            else if (Nota[i] > 7 && Nota[i] <= 10)  
                Desconto[i] = 50;  
        }  
        System.out.println("Relação de Descontos");  
        System.out.println("Aluno\tNota\tDesconto");  
        for (i=0; i<10; i++) {  
            System.out.println(Nome[i] + "\t" + Nota[i] + "\t" + Desconto[i]);  
        }  
    }  
}
```



Exemplo

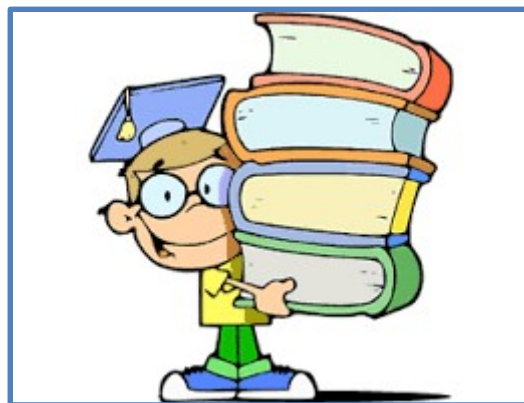
Solução através de Orientação a Objetos:



Exemplo

Solução através de Orientação a Objetos:

Classe Aluno e seus Métodos



Nome (String)

Gravar nome:
setNome (String name)

Ler nome:
String getNome ()

Nota (double)

Gravar nota:
setNota (double nota)

Ler nota:
double getNota ()

Desconto (int)

Gravar desconto:
setDesconto (int desconto)

Ler desconto:
int getDesconto ()

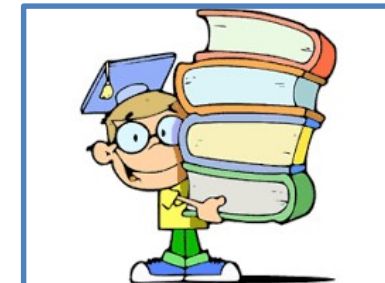
Exemplo

Solução através de Orientação a Objetos:

```
public class Aluno{  
    private String Nome;  
    private double Nota;  
    private int Desconto;  
  
    public void setNome(String nome) {  
        this.Nome=nome;  
    }  
    public void setNota(double valor){  
        this.Nota=valor;  
    }  
    public void setDesconto(int desconto){  
        this.Desconto=desconto;  
    }  
    public String getNome(){  
        return this.Nome;  
    }  
    public double getNota(){  
        return this.Nota;  
    }  
    public int getDesconto(){  
        return this.Desconto;  
    }  
}
```

Atributos

Métodos

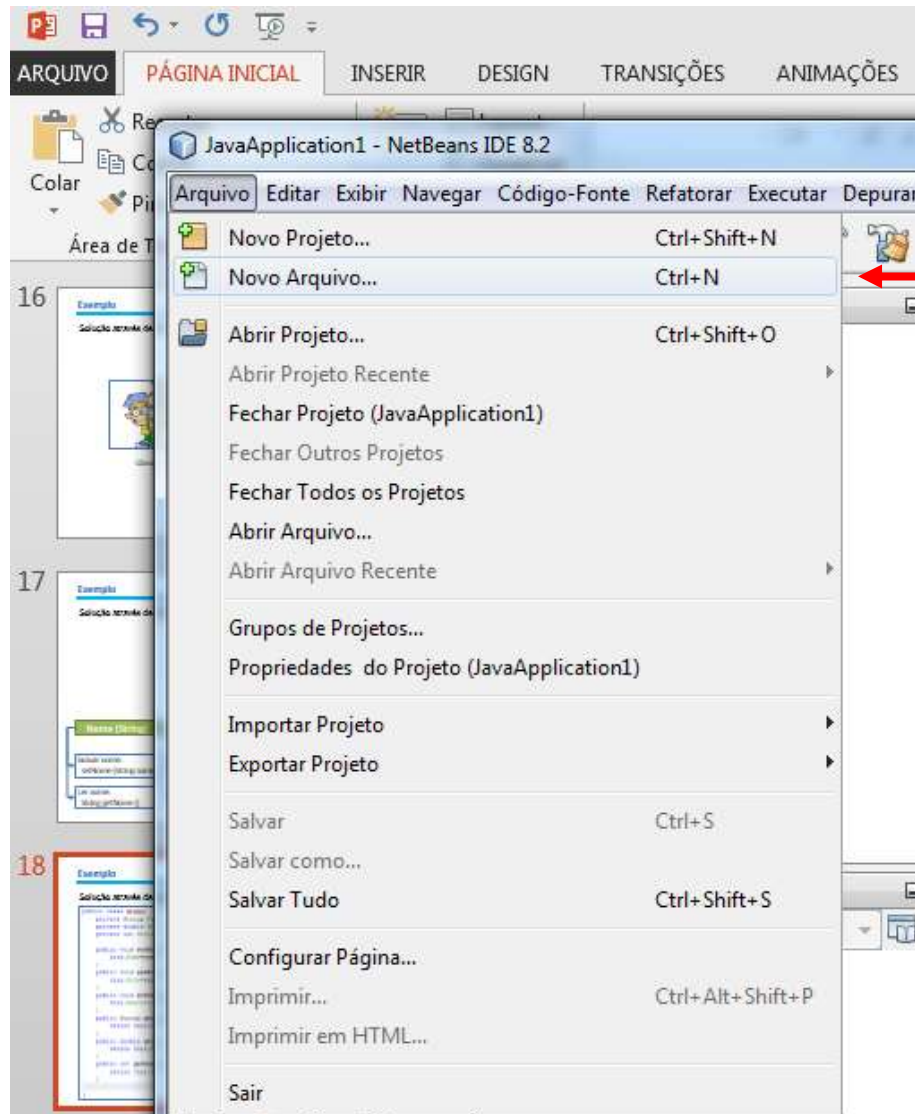


Classe Aluno

Exemplo



Solução através de Orientação a Objetos:



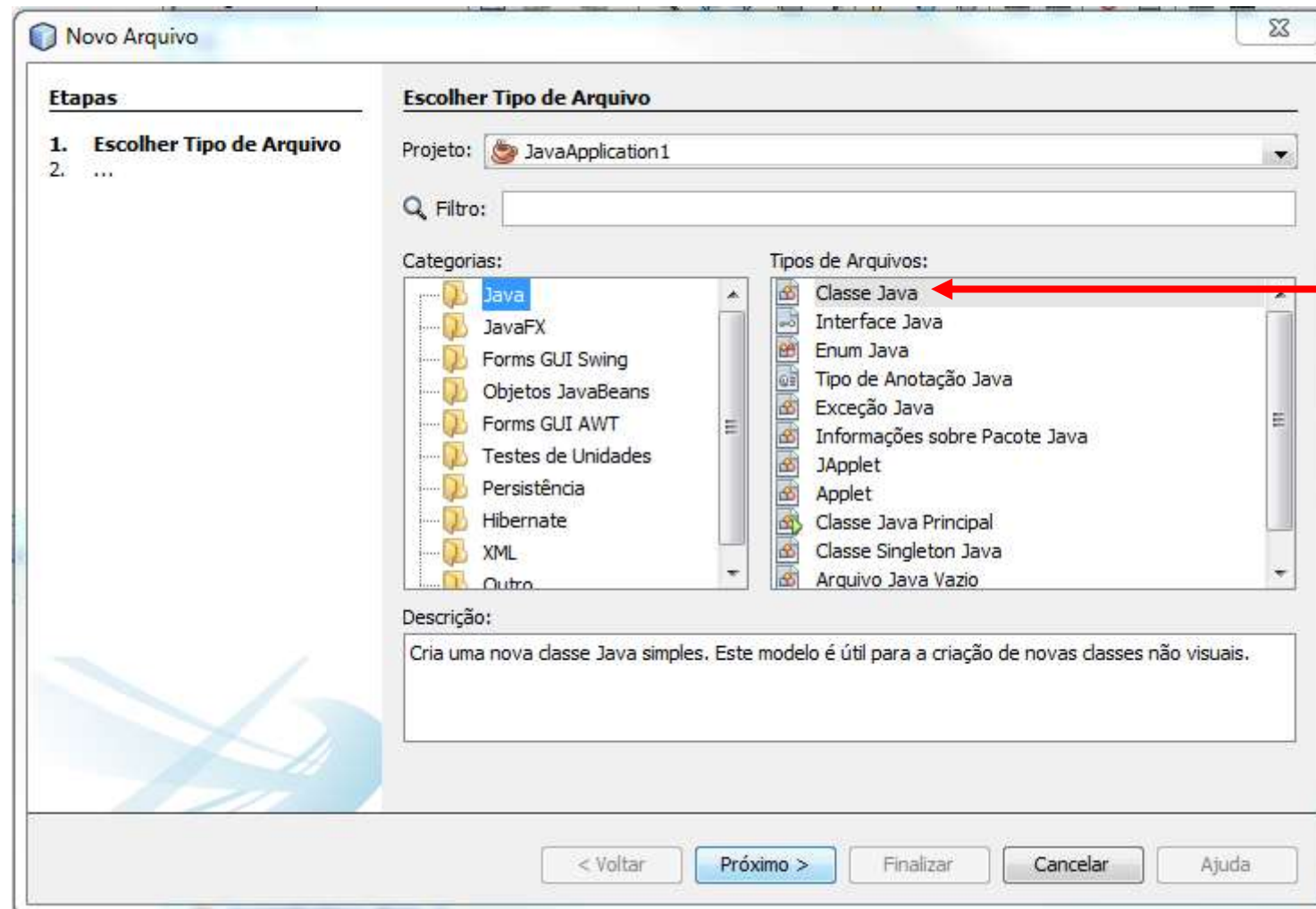
Clicar em Arquivo

↳ Novo Arquivo

Exemplo



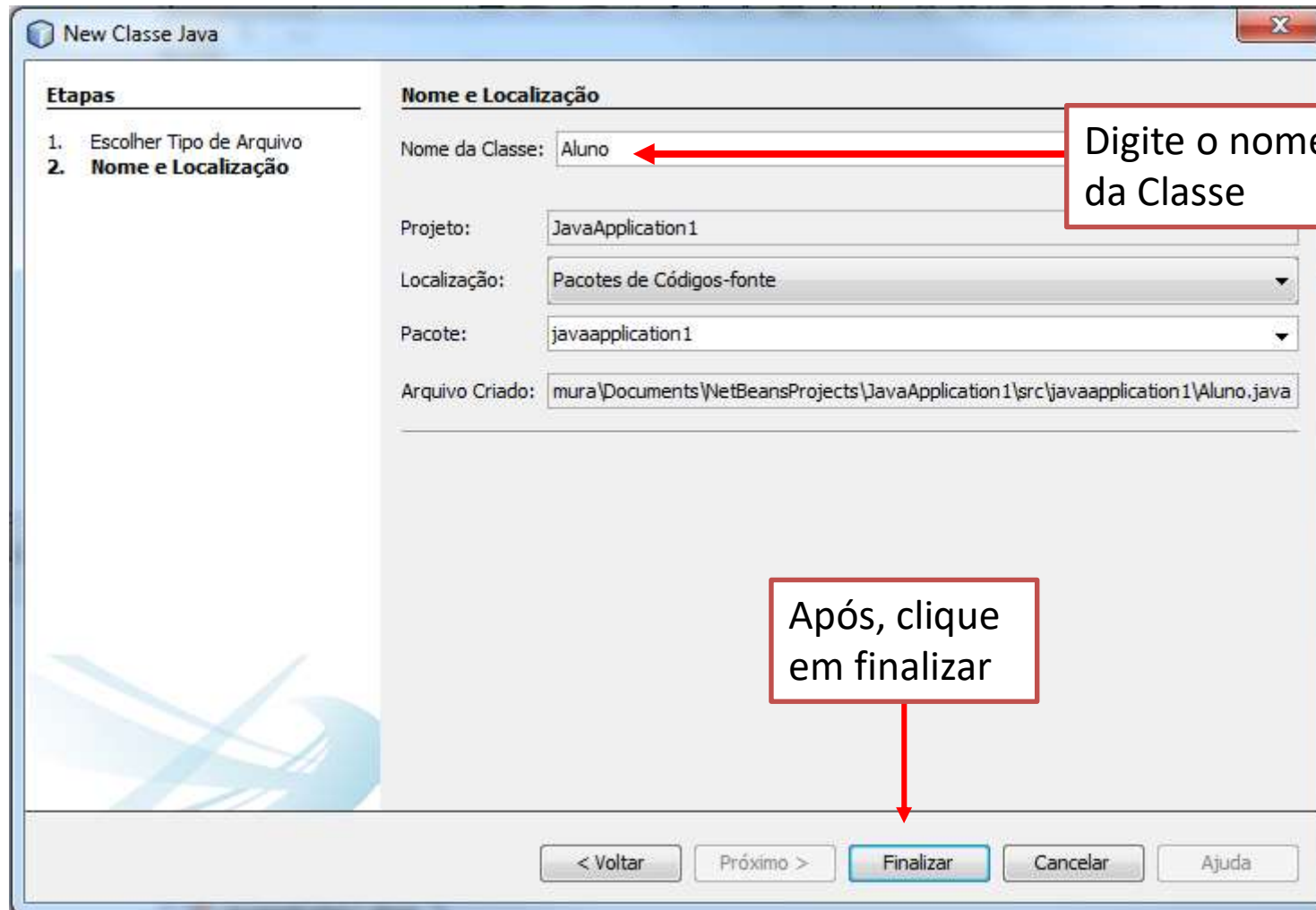
Solução através de Orientação a Objetos:



Clicar em
Classe Java

Exemplo

Solução através de Orientação a Objetos:



New Classe Java

Etapas

1. Escolher Tipo de Arquivo
2. **Nome e Localização**

Nome e Localização

Nome da Classe:

Projeto:

Localização:

Pacote:

Arquivo Criado:

Após, clique em finalizar

< Voltar Próximo > **Finalizar** Cancelar Ajuda

Exemplo

Solução através de Orientação a Objetos:



The screenshot displays an IDE with the following components:

- Project Explorer (Left):** Shows a project named 'JavaApplication1'. Under 'Pacotes de Códigos-fonte', there is a package 'javaapplication1' containing 'Aluno.java' (highlighted with a red arrow) and 'JavaApplication1.java'. Below are 'Pacotes de Teste', 'Bibliotecas', and 'Bibliotecas de Testes'.
- Editor (Right):** Displays the source code of 'Aluno.java'. The code defines a class 'Aluno' with three private attributes: 'Nome' (String), 'Nota' (double), and 'Desconto' (int). It includes a constructor 'Aluno()' that initializes these attributes to empty string, 0, and 0 respectively. There are three setter methods: 'setNome(String nome)', 'setNota(double valor)', and 'setDesconto(int desconto)'. There are also three getter methods: 'getNome()', 'getNota()', and 'getDesconto()'. A red bracket on the right side of the code block groups the setter and getter methods.
- setNome - Navegador (Bottom Left):** Shows the 'Membros' (Members) of the 'Aluno' class. The list includes the constructor 'Aluno()', the three getter methods ('getDesconto() : int', 'getNome() : String', 'getNota() : double'), the three setter methods ('setDesconto(int desconto)', 'setNome(String nome)', 'setNota(double valor)'), and the three private attributes ('Desconto : int', 'Nome : String', 'Nota : double').

```
13     private String Nome;
14     private double Nota;
15     private int Desconto;
16
17     Aluno () {
18         this.Nome="";
19         this.Nota=0;
20         this.Desconto=0;
21     }
22
23     public void setNome(String nome) {
24         this.Nome=nome;
25     }
26     public void setNota(double valor) {
27         this.Nota=valor;
28     }
29     public void setDesconto(int desconto) {
30         this.Desconto=desconto;
31     }
32     public String getNome() {
33         return this.Nome;
34     }
35     public double getNota() {
36         return this.Nota;
37     }
38     public int getDesconto() {
39         return this.Desconto;
40     }
41 }
```

Exemplo

Solução através de Orientação a Objetos:



Construtor da Classe Aluno

Também conhecidos pelo inglês **constructors**, os construtores são os responsáveis por criar o objeto em memória, ou seja, instanciar a classe que foi definida. Eles são obrigatórios e são declarados conforme declaração:

```
public class Carro{  
  
    /* CONSTRUTOR DA CLASSE Carro */  
    public Carro(){  
        //Faça o que desejar na construção do objeto  
    }  
  
}
```



```
Aluno () {  
    this.Nome="";  
    this.Nota=0;  
    this.Desconto=0;  
}
```


Exemplo

Solução através de Orientação a Objetos:
Usando a classe Aluno para um aluno



```
public class JavaApplication1 {  
  
    public static void main(String[] args) {  
        Aluno aluno1=new Aluno();  
  
        aluno1.setNome("Rafael");  
        aluno1.setNota(10);  
        if (aluno1.getNota()>=3.0 && aluno1.getNota()<=5.0)  
            aluno1.setDesconto(10);  
        else if (aluno1.getNota()>5.0 && aluno1.getNota()<=7.0)  
            aluno1.setDesconto(30);  
        else if (aluno1.getNota()>7.0 && aluno1.getNota()<=10.0)  
            aluno1.setDesconto(50);  
  
        System.out.println("Nome: " + aluno1.getNome());  
        System.out.println("Nota: " + aluno1.getNota());  
        System.out.println("Desconto: " + aluno1.getDesconto());  
    }  
}
```

Exemplo



Solução através de Orientação a Objetos: Lista de Alunos

```
public class JavaApplication1 {  
  
    public static void main(String[] args) {  
        Scanner ler=new Scanner(System.in);  
        Aluno[] alunos = new Aluno[10];  
        int i;  
        for (i=0;i<10;i++){  
            alunos[i]=new Aluno();  
        }  
        for(i=0;i<10;i++){  
            System.out.print("Digite o nome do aluno " + (i+1) + ": ");  
            alunos[i].setNome(ler.next());  
            System.out.print("Digite a nota do aluno " + (i+1) + ": ");  
            alunos[i].setNota(ler.nextDouble());  
        }  
  
        for(i=0;i<10;i++){  
            if (alunos[i].getNota()>=3.0 && alunos[i].getNota()<=5.0)  
                alunos[i].setDesconto(10);  
            else if (alunos[i].getNota()>5.0 && alunos[i].getNota()<=7.0)  
                alunos[i].setDesconto(30);  
            else if (alunos[i].getNota()>7.0 && alunos[i].getNota()<=10.0)  
                alunos[i].setDesconto(50);  
        }  
  
        for (i=0;i<10;i++){  
            System.out.println("Nome: " + alunos[i].getNome()+"\t"+"Nota: " +  
                alunos[i].getNota()+"\t"+"Desconto: " + alunos[i].getDesconto());  
        }  
    }  
}
```

Declaração de um
Vetor de Aluno

Inicialização dos
objetos

Exercícios:

Exercício 01 - Escreva uma aplicação que demonstre o uso de instâncias da classe **ContaBancaria** que deve ser criada com o atributo: saldo. Demonstre como a transferência de valores de uma instância da classe para outra pode ser feita através de chamadas aos métodos **deposita** e **retira**. Crie um método **getSaldo** para retornar o saldo corrente da conta. Tente fazer com que os dados que serão usados nas classes sejam lidos do teclado. Use a classe com dois vetores para indicar duas agências bancárias diferentes.



BOAS PRÁTICAS DE PROGRAMAÇÃO



Observação de engenharia de software 6.2

Para promover a capacidade de reutilização de software, todos os métodos devem estar limitados à realização de uma única tarefa bem definida, e o nome do método deve expressar essa tarefa efetivamente.



Dica de prevenção de erro 6.1

Um método que realiza uma única tarefa é mais fácil de testar e depurar do que aquele que realiza muitas tarefas.



Observação de engenharia de software 6.3

Se não puder escolher um nome conciso que expresse a tarefa de um método, seu método talvez tente realizar tarefas em demasia. Divida esse método em vários menores.



Dica de prevenção de erro 6.2

Ao chamar um método que retorna um valor que indica se o método realizou sua tarefa com sucesso, certifique-se de verificar o valor de retorno desse método e, se esse método não foi bem-sucedido, lide com a questão de forma adequada.

BOAS PRÁTICAS DE PROGRAMAÇÃO



Observação de engenharia de software 6.5

Métodos podem retornar no máximo um valor, mas o valor retornado poderia ser uma referência a um objeto que contém muitos valores.



Observação de engenharia de software 6.6

Variáveis devem ser declaradas como campos da classe somente se forem utilizadas em mais de um método da classe ou se o programa deve salvar seus valores entre chamadas aos métodos da classe.



Erro comum de programação 6.1

Declarar parâmetros de método do mesmo tipo como `float x, y` em vez de `float x, float y` é um erro de sintaxe — um tipo é requerido para cada parâmetro na lista de parâmetros.



Erro comum de programação 6.4

Declarar um método fora do corpo de uma declaração de classe ou dentro do corpo de um outro método é um erro de sintaxe.

BOAS PRÁTICAS DE PROGRAMAÇÃO



Erro comum de programação 6.5

Redeclarar um parâmetro como uma variável local no corpo do método é um erro de compilação.



Erro comum de programação 6.6

Esquecer de retornar um valor em um método que deve retornar um valor é um erro de compilação. Se um tipo de retorno além de void for especificado, o método deve conter uma instrução return, que retorna um valor consistente com o tipo de retorno do método. Retornar um valor de um método cujo tipo de retorno foi declarado como void é um erro de compilação.

Referência Bibliográfica Principal

- DEVMEDIA. Disponível em <https://www.devmedia.com.br> . Acessado em Julho de 2019.
- DEITEL, Harvey M.; [Java : Como Programar](#) - 10. ed. Caps, 01, 02, 03, 06; 2015.