

AULA 01:

Introdução a

Orientação a Objetos

Agradecimentos ao material do Profº **Bruno Nogueira**



Apresentação do Curso

Conteúdo

Introdução ao Paradigma de Programação Orientado a Objetos.

Motivo

Preparação para o Hackatruck, que em breve estará na UFMS. O conhecimento do paradigma Orientado a Objetos é extremamente importante para o desenvolvimento de aplicações na linguagem Swift, que será abordada no Hackatruck.

www.facom.ufms.br
<https://hackatruck.com.br/>

Lecionadores

Kaio Mitsuharu Lino Aida

kaiomudkt@gmail.com

Mateus Ragazzi Balbino

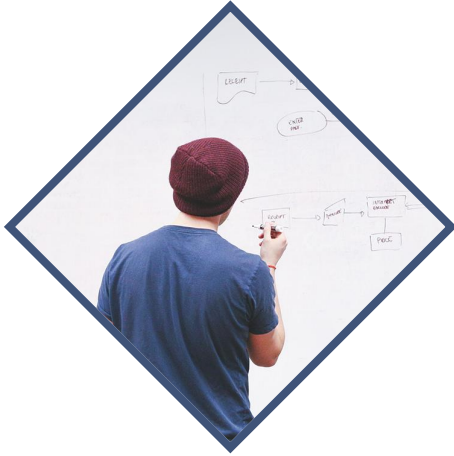
mateusragazzi.b@gmail.com

Acadêmicos de Sistemas de Informação

Mário de Araújo Carvalho

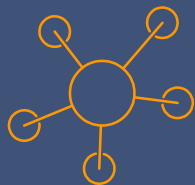
mariodearaujocarvalho@gmail.com

Acadêmico de Ciência da Computação.



1

Aula



Introdução a conceitos de OO



Introdução

- Programação é baseada em abstração de conceitos
- Todas as linguagens de programação nos fornecem níveis de abstração
 - Assembly fornece uma abstração da máquina
 - Linguagens imperativas (Fortran, Basic, C, ...) nos fornecem abstrações dos conceitos de Assembly
 - Tanto Assembly quanto linguagens imperativas são focadas na **abstração da máquina e não na abstração do problema que queremos resolver**



O que é um paradigma

Paradigma é um modelo ou padrão a seguir.

Serve de modelo ou exemplo a ser seguido em determinada situação.

São as normas orientadoras de um grupo que estabelecem limites e que determinam como um indivíduo deve agir dentro desses limites.

O paradigma seria o conjunto de elementos linguísticos que podem ocorrer no mesmo contexto ou ambiente.



O que é um paradigma de programação

Um paradigma de programação fornece e determina a **visão** que o programador possui **sobre a estruturação e execução do programa**.

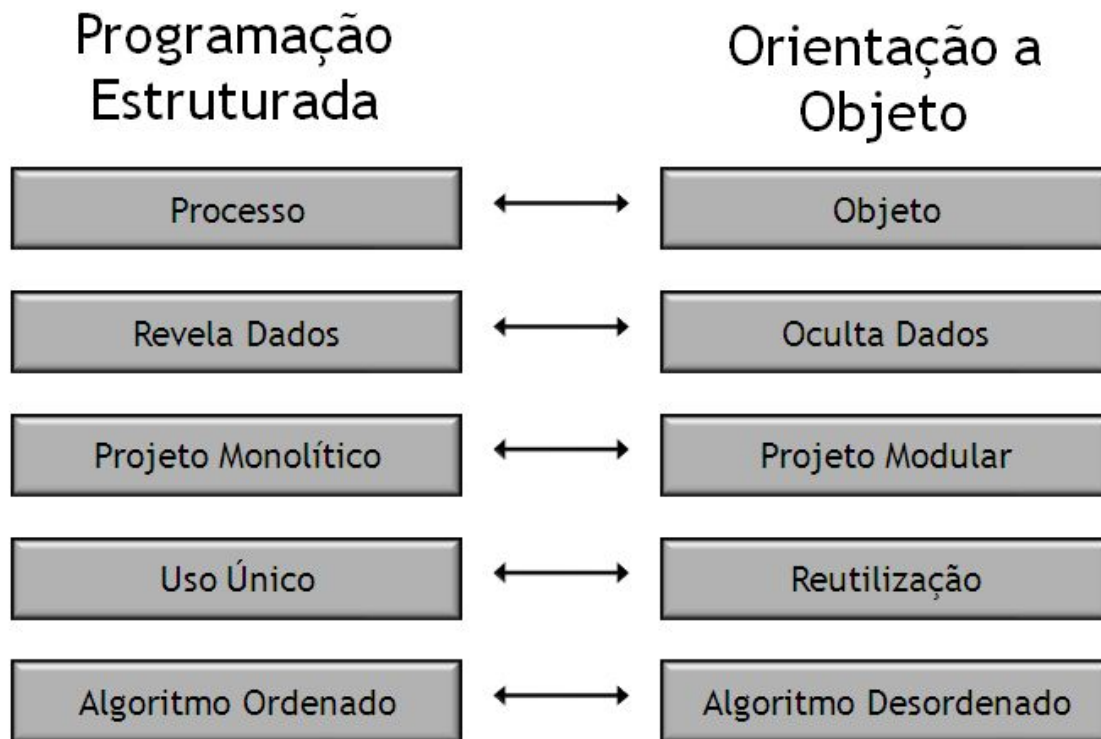


OO e Programação funcional

- Por exemplo, em **programação funcional** os programadores abstraem o programa como uma **sequência de funções executadas de modo empilhado**.
- Enquanto em programação **orientada a objetos**, os programadores podem abstrair um programa como uma **coleção de objetos** que interagem entre si.



Programação estruturada e OO



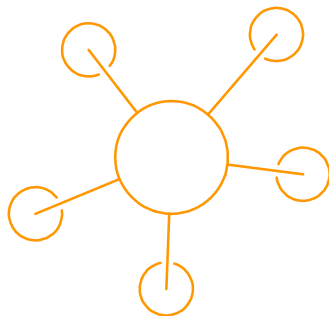


Exemplos de paradigmas de programação

- Funcional
- Lógico
- Declarativo
- Imperativo
- Orientado a objetos
- Orientado a eventos

Paradigma

Orientação a objetos





Conceitos de Orientação a Objetos

- A abordagem orientada a objetos provê uma **solução** mais **genérica**.
- **Elementos no espaço de problemas** são conhecidos como objetos.



Conceitos de Orientação a Objetos

- Programa é **adaptado** ao problema por meio da **criação de novos objetos** – resolução dos problemas mais flexível
- **Linguagem não mais voltada à modelagem da máquina, mas sim do problema a ser solucionado**



Cinco características do Smalltalk

Uma das primeiras linguagens orientadas a objetos na qual Java se baseou, por Allan Kay:

1. Tudo é um objeto
2. Um programa é um grupo de objetos dizendo uns aos outros o que fazer por meio do envio de mensagens
3. Cada objeto tem a sua própria memória, composta por outros objetos
4. Todo objeto tem um tipo
5. Todos os objetos de um mesmo tipo podem receber as mesmas mensagens



Princípios da OO

- **Abstração:**
 - ▷ Representação de um objeto do mundo real somente com suas partes relevantes.
- **Encapsulamento:**
 - ▷ Define a visibilidade de cada elemento.
- **Herança:**
 - ▷ Reuso de código baseado em características semelhantes.
- **Polimorfismo:**
 - ▷ Alteração do funcionamento de um método.
- **Modularidade:**
 - ▷ Procedimentos agrupados em módulos, com namespaces distintos



Abstração

- Consiste em obter do domínio do problema **somente os aspectos importantes** e representá-los na linguagem de programação
 - ▶ Classes são abstrações de conceitos



mario : Personagem

- nome : String
- habilidades : Habilidades[]
- estaGrande : boolean

- + pular() : void
- + correr() : void



Encapsulamento

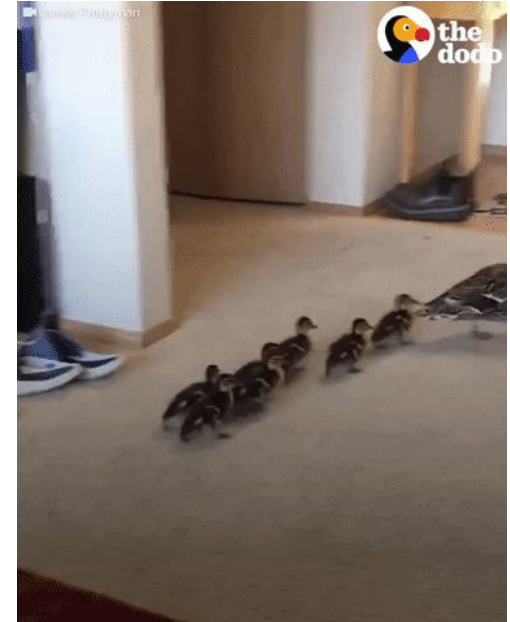
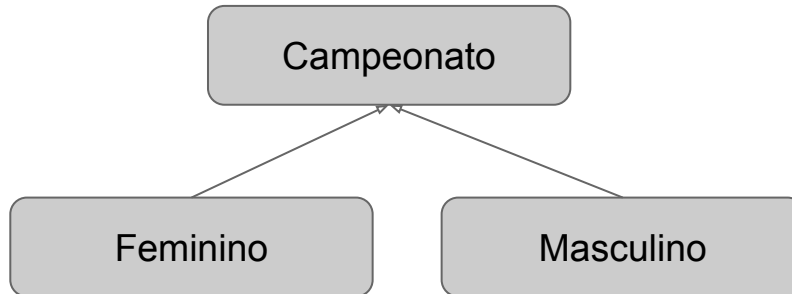
- Dados e procedimentos relacionados a uma classe devem ser implementados nessa mesma classe
 - ▷ Um objeto não deve **nunca manipular diretamente** os dados internos de outros objetos
 - ▷ Manipulação de dados de outras classes deve ser feito **por meio de métodos**





Herança

- Estabelece relação de especialização entre as classes
 - A classe filha **herda características (atributos e métodos)** da classe pai





Polimorfismo

- Um objeto que pode ter várias maneiras de instanciação, assumindo várias **formas distintas**

```
Object o = new Object();  
Object p = new Pessoa();
```

Memória

Pessoa : p

Object : o



Modularidade

- Dividir o software em partes distintas ou utilizar implementações genéricas
 - Exemplo: biblioteca de acesso à banco de dados MySQL





Mensagens

- Objetos se comunicam por meio de envio e recebimento de mensagens
 - Uma mensagem contém alguma forma de **informação**



mario : Personagem



yoshi : Personagem

→
andar





Apresentação da IDE

- Recomendamos a IDE NetBeans.
- <https://www.oracle.com/technetwork/pt/java/javase/downloads/jdk-netbeans-jsp-3413153-ptb.html>
- Outras IDE: IntelliJ IDEA e Eclipse.
- Ame sua IDE!

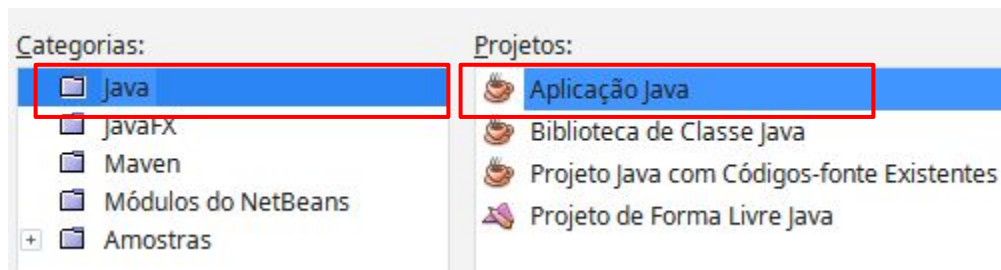
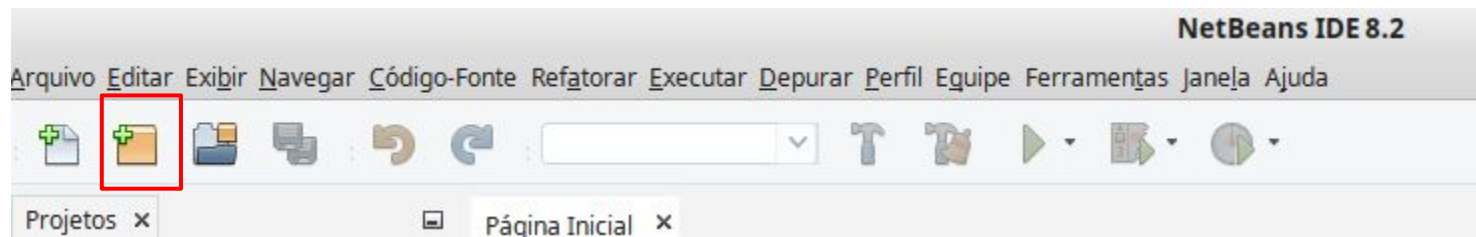




Criando projeto



NetBeans



Nome do projeto

Nome e Localização

Nome do Projeto:

OlaMundo

Localização do Projeto:

/home/kaio/NetBeansProjects

Procurar...

Pasta do Projeto:

/home/kaio/NetBeansProjects/OlaMundo

☐ Usar Pasta Dedicada para Armazenar Bibliotecas

Pasta Bibliotecas:

Procurar...

Usuários e projetos diferentes podem compartilhar as mesmas bibliotecas de compilação (consulte a Ajuda para obter detalhes).

☒ Criar Classe Principal

olamundo.OlaMundo



Primeiro programa em Java

Crie dentro do método principal “**main**” um “print” em java, com o comando:

```
System.out.println("Olá Mundo!");
```



Primeiro programa em Java

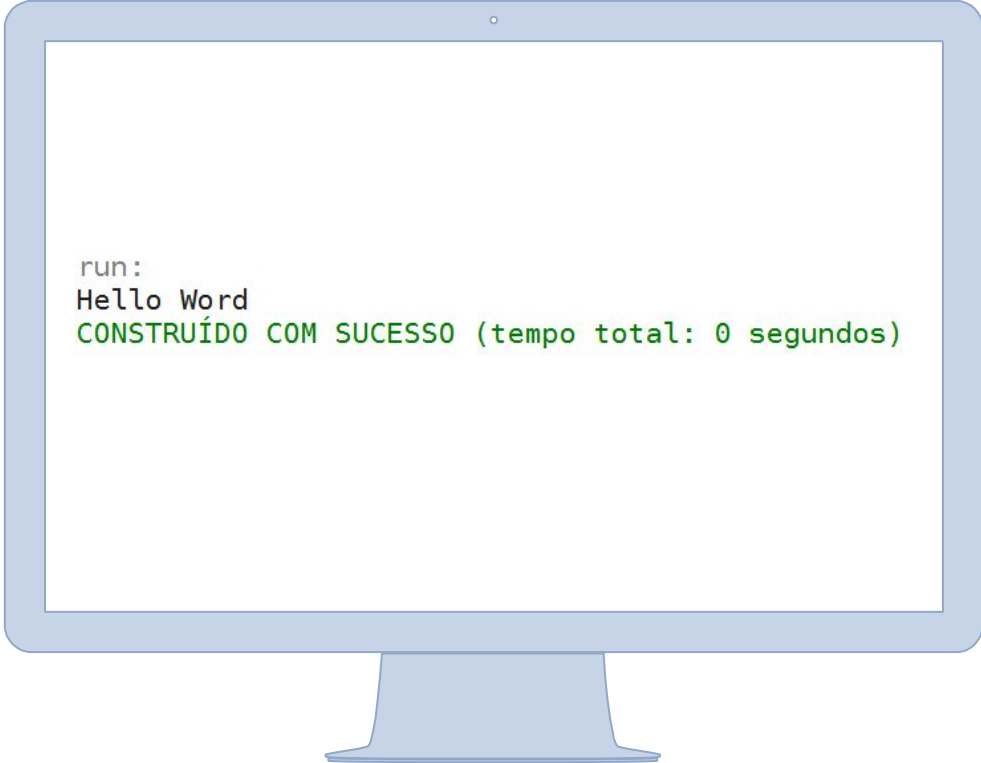
```
HelloWord.java x
Código-Fonte  Histórico
1 package teste;
2 public class HelloWord {
3     public static void main(String[] args) {
4         System.out.println("Olá Mundo!");
5     }
6 }
7
```



Executa projeto



Saída



```
run:  
Hello Word  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



Java é case-sensitive

É um modo de escrever que se refere a um tipo de análise tipográfica da informática. Em língua portuguesa, significa algo como "sensível à caixa das letras" ou "**sensível a maiúsculas e minúsculas**". Diz-se que um software é case-sensitive ou possui "case sensitivity" quando ele é capaz de analisar uma cadeia de caracteres, avaliar a existência de caixa alta e caixa baixa e comportar-se de diferentes maneiras em função disso.



Classe

```
public class HelloWord {  
  
}
```

- Sintaxe
- Objetivo



Método main

```
public static void main(String[] args) {  
  
}
```

- Sintaxe
- Objetivo



Prática de 00

Crie um projeto chamado de “Instancia”.

Nome do Projeto:

Instancia

Desmarque esta opção



Criar Classe Principal

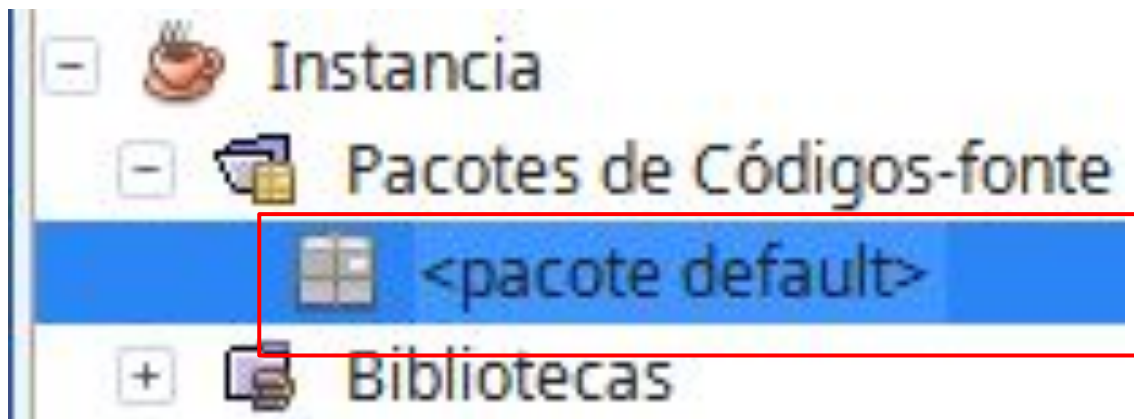
instancia.Instancia



Crie as três classes dentro do <pacote default>

- **Filho**
- **Pai**
- **Teste**

Lembre que nome de classe sempre inicia com letra maiúscula.





Crie essas variáveis para cada uma dessas duas classes

```
public class Pai {  
    String nome;  
    int idade;  
}
```

```
public class Filho {  
    String nome;  
    int idade;  
}
```



Crie o método construtor na classe Pai

```
public class Pai {  
    String nome;  
    int idade;  
  
    public Pai(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }  
}
```



Método construtor

- 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 como no exemplo anterior.
- Em Java, se não criar o construtor, será gerado um padrão, sem parâmetros.



Crie o método construtor na classe Filho

```
public class Filho {  
    String nome;  
    int idade;
```

```
    public Filho(String nome, int idade) {  
        this.nome = nome;  
        this.idade = idade;  
    }
```

```
}
```



Classe teste

Crie o método principal (main) dentro da classe
Teste

```
public class Teste {  
    public static void main(String[] args) {  
  
    }  
}
```



Instancie a classe Pai

```
public class Teste {  
    public static void main(String[] args) {  
        Pai pai = new Pai("Pedro", 45);  
    }  
}
```



Instanciação (new)

Classe **Nome do novo objeto**

Pai pai;

pai = new Pai("Pedro", 45);

Construtor da classe

Cria novo objeto

Tipo do novo objeto



Instanciação (new)

```
Pai pai = new Pai("Pedro", 45);
```

Classe

Cria novo objeto

Construtor da classe

Nome do novo objeto

Tipo do novo objeto



Instancie a classe filho

```
public class Teste {  
    public static void main(String[] args) {  
        Pai pai = new Pai("Pedro", 45);  
        Filho filho = new Filho("João", 20);  
    }  
}
```

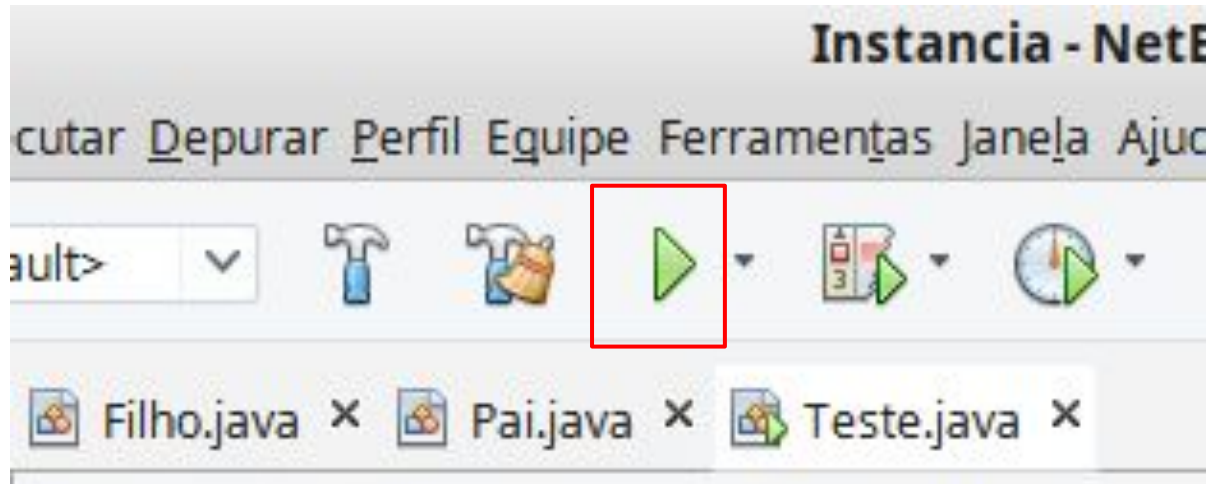


Print valores dos atributos destes objetos

```
public class Teste {  
    public static void main(String[] args) {  
        Filho filho = new Filho("Pedro", 45);  
        Pai pai = new Pai("Jão", 20);  
  
        System.out.println("O filho de "+pai.nome+" de "+pai.idade+" anos: ");  
        System.out.println("É "+filho.nome+" de "+filho.idade+" anos.");  
    }  
}
```



Teste





Saída

Saída - Instancia (run) - Editor

Saída - Instancia (run) x



run:



0 filho de Pedro de 45 anos:

É João de 20 anos.



CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)



Mude os valores das variáveis destes objetos

```
public class Teste {  
    public static void main(String[] args) {  
        Filho filho = new Filho("Pedro", 45);  
        Pai pai = new Pai("Jão", 20);  
        pai.idade = pai.idade + 1;  
        filho.idade = filho.idade + 1;  
  
        System.out.println("O filho de "+pai.nome+" de "+pai.idade+" anos: ");  
        System.out.println("É "+filho.nome+" de "+filho.idade+" anos.");  
    }  
}
```



Nova saída

Saída - Instancia (run) - Editor

Saída - Instancia (run) x



run:



0 filho de Pedro de 46 anos:

É João de 21 anos.



CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)



Exercício

Crie vários filhos!





Crie novos objetos da classe Filho

```
public class Teste {  
    public static void main(String[] args) {  
        Pai pai = new Pai("Pedro", 45);  
        Filho filho = new Filho("Jão", 20);  
  
        Filho filho2 = new Filho("Guilherme", 19);  
        Filho filho3 = new Filho("Paulo Henrique", 18);  
        Filho filho4 = new Filho("Mathews", 17);  
    }  
}
```



Print os novos filhos

```
public class Teste {  
    public static void main(String[] args) {  
        ...  
        System.out.println("Os filhos de "+pai.nome+" de "+pai.idade+" anos, são: ");  
        System.out.println(filho.nome+" de "+filho.idade+" anos.");  
        System.out.println(filho2.nome+" de "+filho2.idade+" anos.");  
        System.out.println(filho3.nome+" de "+filho3.idade+" anos.");  
        System.out.println(filho4.nome+" de "+filho4.idade+" anos.");  
        ...  
    }  
}
```



Saída

Saída - Instancia (run) - Editor

Saída - Instancia (run) x



run:



Os filhos de Pedro de 45 anos, são:



João de 20 anos.



Guilherme de 19 anos.

Paulo Henrique de 18 anos.

Mathews de 17 anos.

CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)



Encapsulamento

- Um dos conceitos de OO apresentados é:
 - ▷ Os atributos podem ser **alterados** apenas por **métodos**.
- Além disso, o JAVA possui 4 modificadores de visibilidade:
 - ▷ **Public**
 - ▷ **Protected**
 - ▷ **Private**
 - ▷ **Default**



Encapsulamento

- **Public:**
 - ▷ Visível a todos elementos do projeto
 - ▷ Deixa o acesso direto aberto
- **Protected**
 - ▷ Visível a todos os elementos do pacote
- **Private**
 - ▷ O acesso fica restrito aos métodos GET e SET
- **Default:**
 - ▷ Visível a todos os elementos do pacote



Encapsule os atributos das classes Pai e Filho

- Utilize o modificador de acesso “**private**”
- Sintaxe: `private tipo` variavel;

```
String nome;  
int idade;
```



O que aconteceu na classe **Teste**?

Modifique sua classe **Teste** para funcionar corretamente.



Métodos de acesso

Para acessar um atributo privado de uma outra classe, precisamos **criar os métodos** que permitem o **acesso**.



Criando métodos de acesso (Getter e Setter)

```
public int getIdade() {  
    return idade;  
}  
  
public void setIdade(int idade) {  
    this.idade = idade;  
}
```

```
public String getNome() {  
    return nome;  
}
```

```
public void setNome(String nome) {  
    this.nome = nome;  
}
```



Por que usar métodos de acesso?

```
public void setIdade(int idade) {  
    if( idade >= 0 ) {  
        this.idade = idade;  
    }  
}
```

Impõem **regras**
De acesso e modificação



Editar a classe Teste.java para funcionar corretamente

```
System.out.println("Os filhos de "+pai.getNome()+" de "+pai.getIdade()+"  
anos, são: ");
```

```
System.out.println(filho.getNome()+" de "+filho.getIdade()+" anos.");
```

```
System.out.println(filho2.getNome()+" de "+filho2.getIdade()+" anos.");
```

```
System.out.println(filho3.getNome()+" de "+filho3.getIdade()+" anos.");
```

```
System.out.println(filho4.getNome()+" de "+filho4.getIdade()+" anos.");
```



Editar os valores das idades

Utilize os métodos de acesso para adicionar mais um ano de idade para todos.



Utilização dos metodos getters e setters

```
pai.setIdade(pai.getIdade()+1);
```

```
filho.setIdade(filho.getIdade()+1);
```

```
filho2.setIdade(filho2.getIdade()+1);
```

```
filho3.setIdade(filho3.getIdade()+1);
```

```
filho4.setIdade(filho4.getIdade()+1);
```



Nova saída

Saída - Instancia (run) - Editor

Saída - Instancia (run) x

```
run:  
Os filhos de Pedro de 46 anos, são:  
João de 21 anos.  
Guilherme de 20 anos.  
Paulo Henrique de 19 anos.  
Mathews de 18 anos.  
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
```



Dúvidas?

Kaio Mitsuharu Lino Aida

kaiomudkt@gmail.com

Mateus Ragazzi Balbino

mateusragazzi.b@gmail.com

Mário de Araújo Carvalho

mariodearaujocarvalho@gmail.com