INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Bagé

Orientação a Objetos com Java

Rafael Bastos

Roteiro

Fundamentos da Programação OO

Criação de classes em Java

Modificadores de acesso

Herança, encapsulamento e polimorfismo

Atividade

Roteiro

Fundamentos da Programação OO

Criação de classes em Java

Modificadores de acesso

Herança, encapsulamento e polimorfismo

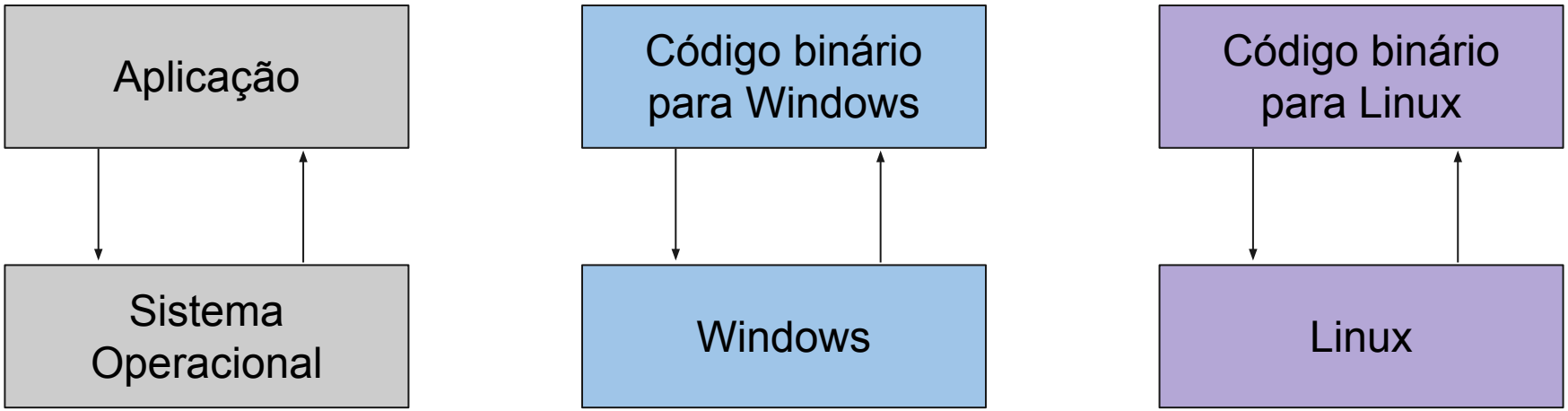
Atividade

Fundamentos de Programação OO

- Máquina virtual
- Conceitos básicos
- Classes
- Objetos
- Atributos
- Métodos

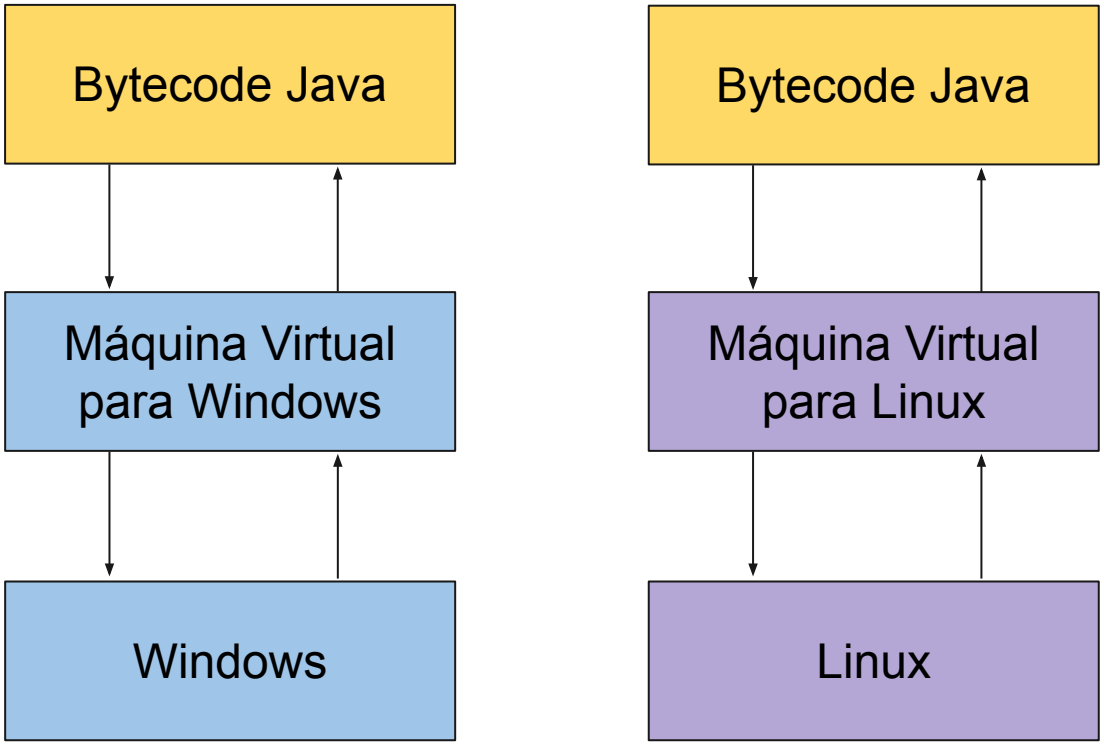
Fundamentos de Programação OO

Abordagem tradicional



Fundamentos de Programação OO

Máquina virtual



Fundamentos de Programação OO

JVM - Java Virtual Machine

JRE - Java Runtime Environment

JDK - Java Development Kit

Fundamentos de Programação OO

Preparando o ambiente

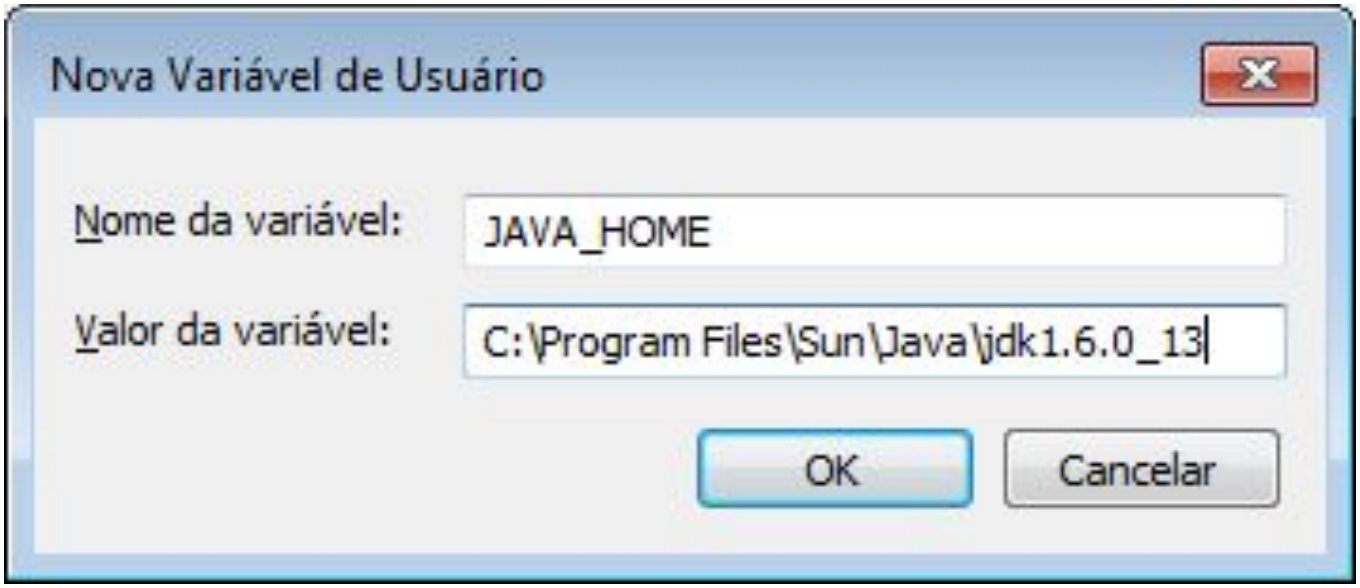
JDK + variáveis de ambiente

- JAVA_HOME
- PATH

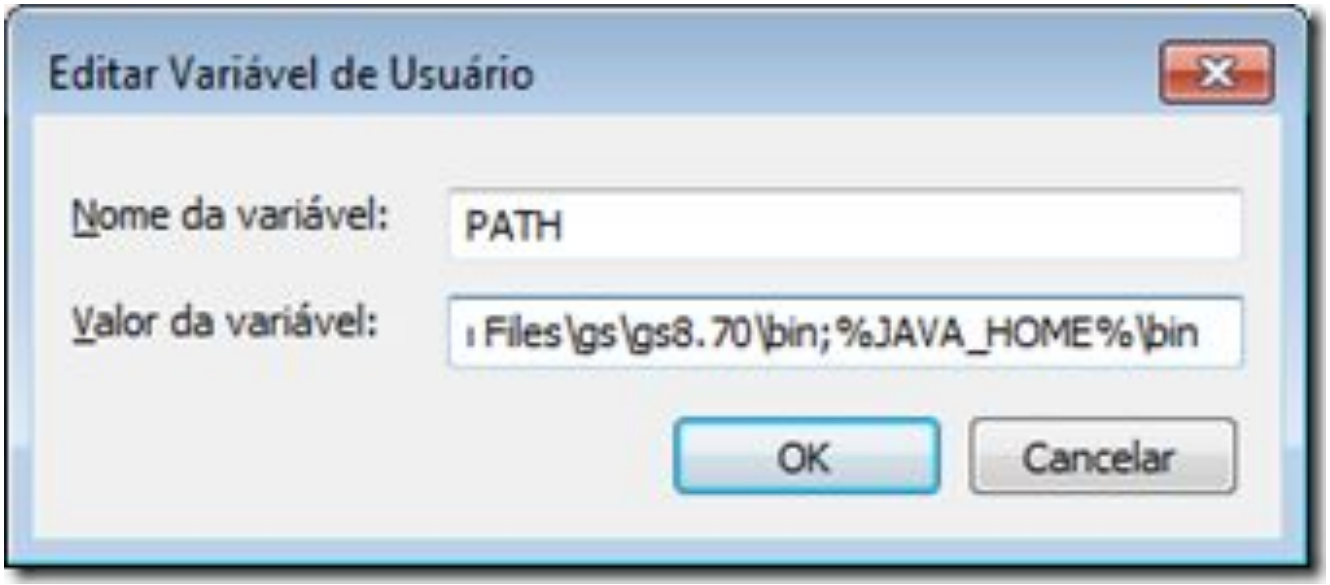
Notepad++, Eclipse, Notepad …

Fundamentos de Programação OO

Preparando o ambiente



Preparando o ambiente



Fundamentos de Programação OO

Exemplo

```
public class pooj_01 {
   public static void main(String[] args) {
      System.out.println("Primeiro programa!");
   }
}
```

Fundamentos de Programação OO

Exemplo

pooj_01.java

```
javac pooj_01.java
```

pooj_01.class

```
java pooj_01
```

Fundamentos de Programação OO

Variáveis primitivas

```
int idade; // idade = 23;
double preco; // preco = 16.95;
boolean teste; // teste = true;
char letra; // letra = 'a';
String nome; // nome = 'Rafael';
```

Fundamentos de Programação OO

Operadores

Aritméticos (+, -, *, /, %)
Atribuição (=, +=, -=, *=, /=, %=)
Relacionais (==, !=, <, <=, >, >=)
Lógicos (&&, ||)

Fundamentos de Programação OO

Casting

```
double d = 5;
float f = 3;
float x = f + (float) d;
```

Fundamentos de Programação OO

Casting

PARA: / DE:	byte	short	char	int	long	float	double
byte	----	*Impl.*	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
short	(byte)	----	(char)	*Impl.*	*Impl.*	*Impl.*	*Impl.*
char	(byte)	(short)	----	*Impl.*	*Impl.*	*Impl.*	*Impl.*
int	(byte)	(short)	(char)	----	*Impl.*	*Impl.*	*Impl.*
long	(byte)	(short)	(char)	(int)	----	*Impl.*	*Impl.*
float	(byte)	(short)	(char)	(int)	(long)	----	*Impl.*
double	(byte)	(short)	(char)	(int)	(long)	(float)	----

Converção String x outros tipos

```
String variavel = Integer.toString(15);
String variavel2 = Double.toString(15.99);

int variavel3 = Integer.valueOf("16");
double variavel4 = Double.valueOf("16.998");
```

Fundamentos de Programação OO

if - else

```
if (condicao [&&, ||]) {
   codigo;
}

if (media >= 7.5) {
   System.out.println("Aprovado!");
} else {
   System.out.println("Reprovado!");
}
```

Fundamentos de Programação OO

while

```
while (condicao) {
   codigo;
}

int x = 1;
while (x <= 10) {
   System.out.println(x);
   x = x + 1;
}
```

Fundamentos de Programação OO

for

```
for (inicializacao; condicao; incremento) {
   codigo;
}

for (int x = 1; x <= 10; x = x + 1) {
   System.out.println("Número " + x);
}
```

Fundamentos de Programação OO

Classe

Estrutura que abstrai um conjunto de objetos com características similares.

Define o comportamento de seus objetos através de métodos e os estados possíveis destes objetos através de atributos.

Descreve os serviços providos por seus objetos e quais informações eles podem armazenar.

Fundamentos de Programação OO

Objeto

É uma referência a um local da memória que possui um valor.

Pode ser uma variável, função, ou estrutura de dados.

A instância de uma classe.

Fundamentos de Programação OO

Atributo

Elementos que definem a estrutura de uma classe. Também são conhecidos como variáveis de classe.

Podem ser divididos em dois tipos básicos: atributos de instância e de classe.

Fundamentos de Programação OO

Método

Determinam o comportamento dos objetos de uma classe e são análogos às funções ou procedimentos da programação estruturada.

Exemplo - pooj_02.java

```
import java.util.Scanner;
class pooj_02 {
	public static void main(String[] args)  {
		Scanner scanner = new Scanner(System.in);
		System.out.println("Qual seu nome?");
		String nome = scanner.nextLine();
		System.out.println("Quantos anos você tem?");
		int idade = scanner.nextInt();
		System.out.println(nome + ", você tem " + idade + " anos");
	}
}
```

Fundamentos de Programação OO

Exercício

Crie um programa em java que receba duas notas e informe a média aritmética destas duas notas na tela.

Fundamentos de Programação OO

Exercício

Crie um programa em java que receba duas notas e informe a média aritmética destas duas notas na tela.

calcMedia.java

Roteiro

Fundamentos da Programação OO

Criação de classes em Java

Modificadores de acesso

Herança, polimorfismo e encapsulamento

Atividade

Criação de classes em Java

Antes de um objeto ser criado, deve-se definir quais serão os seus atributos e métodos.

A partir de uma classe, pode-se construir objetos na memória do computador que executa a aplicação.

Criação de classes em Java

Exemplo - Aluno.java

```
class Aluno {
   String nome;
   double nota1, nota2;
}
```

Criação de classes em Java

Exemplo - Aluno.java

```
class Aluno {
   String nome;
   double nota1, nota2;

   public double calcMedia() {
      return (this.nota1 + this.nota2)/2;
   }
}
```

Roteiro

Fundamentos da Programação OO

Criação de classes em Java

Modificadores de acesso

Herança, polimorfismo e encapsulamento

Atividade

Modificadores de acesso

```
public
private (métodos e atributos)
protected
default (padrão)
final
abstract (classes)
static
```

Modificadores de acesso

`public`

Uma declaração com o modificador *public* pode ser acessada de qualquer lugar e por qualquer entidade que possa visualizar a classe a que ela pertence.

`private`

Os elementos da classe definidos como *private* não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.

Modificadores de acesso

`protected`

O modificador *protected* torna o elemento acessível às classes do mesmo pacote ou através de herança, seus elementos herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

`default`

A classe e/ou seus elementos são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador.

Modificadores de acesso

`final`

Quando é aplicado na classe, não permite estendê-la, nos métodos impede que o mesmo seja sobrescrito (overriding) na subclasse, e nos valores de variáveis não pode ser alterado depois que já tenha sido atribuído um valor.

`abstract`

Uma classe abstrata não pode ser instanciada, ou seja, não pode ser chamada pelos seus construtores. Se houver alguma declaração de um método como *abstract*, a classe também deve ser marcada como abstract.

Modificadores de acesso

`static`

É usado para a criação de uma variável que poderá ser acessada por todas as instâncias de objetos desta classe como uma variável comum, ou seja, a variável criada será a mesma em todas as instâncias e quando seu conteúdo é modificado numa das instâncias, a modificação ocorre em todas as demais. E nas declarações de métodos ajudam no acesso direto à classe, portanto não é necessário instanciar um objeto para acessar o método.

Roteiro

Fundamentos da Programação OO

Criação de classes em Java

Modificadores de acesso

Herança, polimorfismo e encapsulamento

Atividade

Herança

Permite a criação de novas classes a partir de outras previamente criadas.

Essas novas classes são chamadas de subclasses e as classes já existentes, que deram origem às subclasses, são chamadas de superclasses.

Uma subclasse herda métodos e atributos de sua superclasse, mas pode rescrevê-los para uma forma mais específica de representar o comportamento do método herdado.

Exemplo - Pessoa.java

```
public class Pessoa {
    public String nome;
    public String cpf;

    public Pessoa(String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
    }
}
```

Exemplo - Aluno2.java

```
public class Aluno2 extends Pessoa {
    public Aluno2(String nome, String cpf) {
        super(nome, cpf);
    }
    public String matricula;
}
```

Exemplo - Professor.java

```
public class Professor extends Pessoa {
    public Professor(String nome, String cpf) {
        super(nome, cpf);
    }
    public double salario;
}
```

Exemplo - Aluno2.java

```
public static void main(String[ ] args) {
    Aluno2 aluno = new Aluno2("Jose", "123.456.789-00");
    aluno.matricula = "BG.12345";

    System.out.println("Nome: " + aluno.nome);
    System.out.println("CPF: " + aluno.cpf);
    System.out.println("Matrícula: " + aluno.matricula);
}
```

Polimorfismo

Princípio a partir do qual as classes derivadas de uma superclasse são capazes de invocar os métodos que, embora apresentem a mesma assinatura, comportam-se de maneira diferente para cada uma das classes derivadas.

Mecanismo por meio do qual selecionamos as funcionalidades utilizadas de forma dinâmica por um programa no decorrer de sua execução.

Exemplo - Aluno3.java

```
public class Aluno3 {
	String nome;
	double media;

	public Aluno3(String nome) {
		this.nome = nome;
	}
	public Aluno3(String nome, double media) {
		this.nome = nome;
		this.media = media;
	}
}
```

Encapsulamento

Técnica utilizada para esconder detalhes internos para o usuário, tornando partes do sistema mais independentes possível.

Em um processo de encapsulamento os atributos das classes são do tipo private.

Para acessar esses tipos de modificadores, é necessário criar métodos setters e getters.

Encapsulamento

```
public class AlunoGS {
   private String nome;
   private double nota1, nota2;

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

Roteiro

Fundamentos da Programação OO

Criação de classes em Java

Modificadores de acesso

Herança, polimorfismo e encapsulamento

Atividade

Atividade

Crie um programa em Java que implemente uma classe **Funcionario** com os seguintes atributos: nome, matricula, cpf, cargo, salario.

Esta classe deve possuir os seguintes métodos:

* Consulta ao nome, cargo e salario;
* Troca de cargo, recebendo o novo cargo;
* Aumento de salário, recebendo o novo salário.

Uma segunda classe deve realizar a criação do objeto e manipulação.

INSTITUTO FEDERAL
Sul-rio-grandense
Câmpus Bagé

Orientação a Objetos com Java

github.com/rafaelrodriguesbastos/SACI2024

Rafael Bastos