Programação 2

Introdução a classes, objetos e métodos

Prof. Domingo Santos

domingos.santos@upe.br

UNIVERSIDADE DE PERNAMBUCO

Resumo

- Introdução a classes, objetos e métodos
  - Introdução
  - métodos set e métodos get
  - Tipos primitivos versus tipos por referência
  - Construtor da Classe
  - Exemplo de uma classe
  -

- Métodos, um exame mais profundo:
  - Considerações iniciais
  - Métodos static, campos static e classe Math
  - Métodos com múltiplos parâmetros
  - Promoção e coerção de argumentos
  - Escopo das declarações
  - Sobrecarga de método

Conteúdo baseado no capítulo 3 e 6 do livro de Deitel

UNIVERSIDADE DE PERNAMBUCO

Introdução a classes, objetos e métodos

Introdução à tecnologia de objetos

- Forte demanda para desenvolvimento de software de maneira rápida, correta e econômica

- Programas orientados a objetos são muitas vezes mais fáceis de entender, corrigir e modificar

- **Classe**:
  - Uma classe é um modelo para criar objetos. Define os atributos (variáveis) e métodos (funções) que os objetos dessa classe podem ter
  - Capacidade de reutilização

- **Método**: Um método é uma função associada a uma classe que define o comportamento dos objetos dessa classe

- **Objeto**:
  - Um objeto é uma instância de uma classe. Possui características específicas definidas pela classe
  - Nesse sentido podemos criar objetos de data/hora, objetos áudio, objetos vídeo, objetos automóvel, objetos, pessoas, quase qualquer substantivo

Introdução à tecnologia de objetos

Exemplos:

- Classe Gato:
  - Atributos:
    - cor
    - tamanho
    - idade
    - nome
    - raça
    - tipoRação
  - Métodos:
    - tempoVida
    - exibirNomeIdade
- Classe Computador:
  - Atributos:
    - cor
    - tamanho
    - isNotebook?
    - marca
    - tamanhoTela
  - Métodos:
    - setMarca
    - getMarca
    - exibirInfo
- Classe Apartamento:
  - Atributos:
    - tamanho
    - cpfDono
    - IsAlugado
    - valorAluguel
  - Métodos:
    - isInadimplente
    - calcularICMS

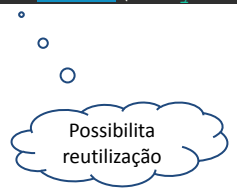
Introdução à tecnologia de objetos

Exemplo: Classe Carro quais são os atributos deste?

- marca
- modelo
- ano

Instância do objeto em java

```java
Carro meuCarro = new Carro("Toyota", "Corolla", 2022);
```



Classe em java

```java
public class Carro {
    // Atributos
    private String marca;
    private String modelo;
    int ano;
    // Método construtor
    public Carro(String marca, String modelo, int ano) {
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }
    // Método para exibir informações do carro
    public void exibirInfo() {
        System.out.println("Marca: " + marca);
        System.out.println("Modelo: " + modelo);
        System.out.println("Ano: " + ano);
    }
}
```

Variáveis de instância, métodos set e métodos get

Criação de um arquivo com Classe Account:

- Essa classe não será a main

```java
package com.example.helloworld;

public class Account {

    private String name; // variável de instancia

    //metodo para definir o nome no objeto
    public void setName(String name) {
        this.name = name; // armazena o nome
    }

    //metodo que recupera o nome do objeto

    public String getName() {
        return name; // retorna o valor do nome para o chamador
    }

}
```

Criação de um arquivo com Classe AccountTest

- Essa classe é a main
- que utiliza os métodos da classe Account

```java
package com.example.helloworld;
import java.util.Scanner;

public class AcountTest {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        Account myAccount = new Account();

        // exibe o valor inicial do nome (null)
        System.out.printf("Initial name is: %s%n%n", myAccount.getName());

        // solicita e lê o nome
        System.out.println("Please enter the name:");
        String theName = input.nextLine(); // lê uma linha de texto
        myAccount.setName(theName);

        System.out.println(); // gera saída de uma linha em branco

        System.out.printf("Name in object myAccount is:%n%s%n", myAccount.getName());

        input.close();
    }

}
```

- Métodos e variáveis
  - boa prática: declarar no início da classe
  - Variável local e global (this.)

- Private x public
- Tipos de retorno
- palavra chave new, cria uma nova instância

Variáveis de instância, métodos set e métodos get

acesso proibido à variável privada

```
System.out.printf("Name in object myAccount is:%n%s%n", myAccount.name);
```

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The field Account.name is not visible

        at com.example.helloworld.AcountTest.main(AcountTest.java:22)
```

Por que devemos usar métodos get e set para retornar e alterar a variável:

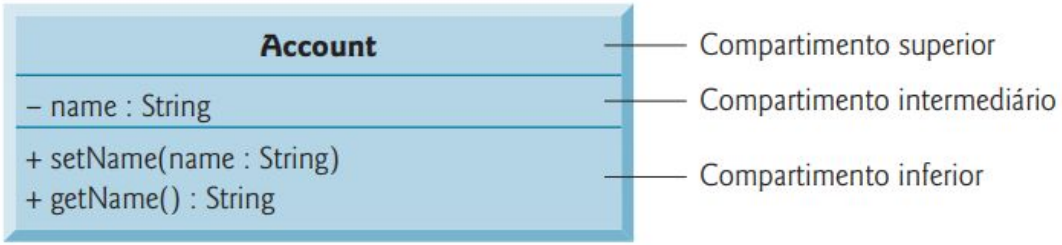
- Validar tentativas de modificações nos dados private
- Controlar como os dados são apresentados para o chamador
- Rejeitar qualquer tentativa de definir os dados como valores ruins:
  - altura negativa;
  - formatação para o CPF
  - …

Classe Account

```
package com.example.helloworld;

public class Account {

    private String name; // variável de instancia

    //metodo para definir o nome no objeto
    public void setName(String name) {
        this.name = name; // armazena o nome
    }

    //metodo que recupera o nome do objeto

    public String getName() {
        return name; // retorna o valor do nome para o chamador
    }

}
```

Variáveis de instância, métodos set e métodos get

Diagrama de classe UML para Account



Classe Account

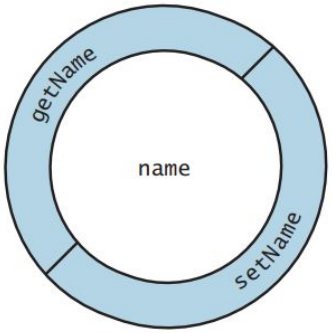
```java
package com.example.helloworld;

public class Account {

    private String name; // variável de instancia

    //metodo para definir o nome no objeto
    public void setName(String name) {
        this.name = name; // armazena o nome
    }

    //metodo que recupera o nome do objeto

    public String getName() {
        return name; // retorna o valor do nome para o chamador
    }

}
```

Interpretação do diagrama de classe UML para:

- private (-), public (+)
- Compartilhamento superior: nome da classe em negrito
- Compartimento intermediário: atributos da classe no formato sinal (+ ou -) nome : tipo do atributo
- Compartimento inferior: operações da classe, possuindo o formato, sinal (+ ou -) nome(nome : tipo do atributo): Tipo do retorno

Visualização conceitual de um objeto Account com dados encapsulados

A variável de instância private chamada name permanece oculta no objeto e protegida por uma camada externa de métodos public



UNIVERSIDADE DE PERNAMBUCO

Tipos primitivos versus tipos por referência

- Primitivos:
  - int, boolean, byte, char, short, long, float e double
  - Uma variável de tipo primitivo pode armazenar exatamente um valor de seu tipo declarado por vez. Caso exista o anterior, é perdido
  - Variáveis de instância de tipo primitivo são inicializadas por padrão — dos tipos byte, char, short, int, long, float e double como 0, e as do tipo boolean como false
  - É possível especificar um valor inicial particular, exemplo: private int numberOfStudents = 10
  - Não podem ser usadas para chamar métodos
- Por referência:
  - Classes que especificam os objetos são por referência
  - Variáveis locais não são inicializadas por padrão, o valor padrão é *null*
  - Exemplo:
    - Account myAccount = new Account() -> myAccount uma referência ao objeto Account

Construtor da Classe

- Variável name é inicializada como null, como poderíamos permitir algo diferente disso?

- Por meio do **construtor:**
  - O Java requer uma chamada de construtor para cada objeto que é desenvolvido,
  - então esse é o ponto ideal para inicializar variáveis de instância de um objeto
  - **Construtores não podem retornar valores**

Classe Account

```java
package com.example.helloworld;

public class Account {

    private String name; // variável de instancia

    //metodo para definir o nome no objeto
    public void setName(String name) {
        this.name = name; // armazena o nome
    }

    //metodo que recupera o nome do objeto

    public String getName() {
        return name; // retorna o valor do nome para o chamador
    }

}
```

Chamada para a classe sem construtor

```java
Account myAccount = new Account();
```

Classe Account com construtor

```java
package com.example.helloworld;

public class Account {

    private String name; // variável de instancia

    public Account(String name) {
        this.name = name;
    }
    //metodo para definir o nome no objeto
    public void setName(String name) {
        this.name = name; // armazena o nome
    }

    //metodo que recupera o nome do objeto

    public String getName() {
        return name; // retorna o valor do nome para o chamador
    }

}
```

Chamada para a classe com construtor

```java
Account myAccount = new Account("Namy");
```

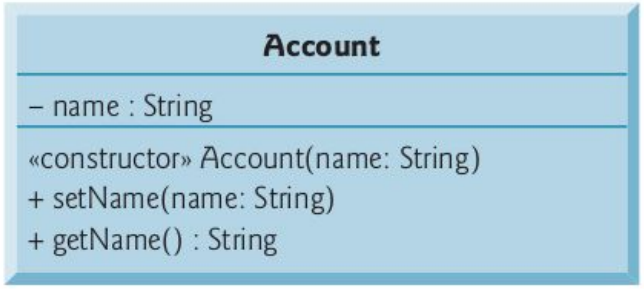
Façam as alterações e olhem o novo resultado

Construtor da Classe

- Novo Diagrama de classe para Account
  - entre aspas francesas (« e »)

Classe Account com construtor

```java
package com.example.helloworld;

public class Account {

    private String name; // variável de instancia

    public Account(String name) {
        this.name = name;
    }
    //metodo para definir o nome no objeto
    public void setName(String name) {
        this.name = name; // armazena o nome
    }

    //metodo que recupera o nome do objeto

    public String getName() {
        return name; // retorna o valor do nome para o chamador
    }

}
```



A classe Account com um saldo; números de ponto flutuante

- E se agora fosse preciso manter o saldo da Account, além do nome.
  - seria possível utilizar o int?
  - Não! precisamos representá-los por números de ponto flutuante: **Float ou Double**

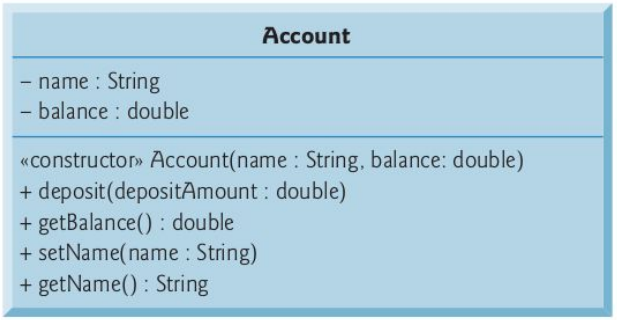
Classe Account

```java
package com.example.helloworld;

public class Account {

    private String name; // variável de instancia

    public Account(String name) {
        this.name = name;
    }
    //metodo para definir o nome no objeto
    public void setName(String name) {
        this.name = name; // armazena o nome
    }

    //metodo que recupera o nome do objeto

    public String getName() {
        return name; // retorna o valor do nome para o chamador
    }

}
```

A classe Account com um saldo; números de ponto flutuante

Adição da variável balance na classe Account

Novo Diagrama UML da Classe




```java
package com.example.helloworld;
public class Account {

	private String name; // variável de instancia
	private double balance; // variável de instancia

	public Account(String name, double balance) {
		this.name = name;
		//validacao para alterar a variavel, caso nao seja
		// maior que 0, eh mantido valor default 0
		if (balance>0.0) {
			this.balance = balance;
		}
	}

	public void deposit(double depositAmount) {
		if (depositAmount>0.0) {
			balance = balance + depositAmount;
		}
	}

	public double getBalance() {
		return balance;
	}
	//metodo para definir o nome no objeto
	public void setName(String name) {
		this.name = name; // armazena o nome
	}

	//metodo que recupera o nome do objeto

	public String getName() {
		return name; // retorna o valor do nome para o chamador
	}

}
```

A classe Account com um saldo; números de ponto flutuante

A classe AccountTest para utilizar a classe Account

Input nextDouble

"%.2f": precisão de um ponto flutuante com duas casas decimais

```
package com.example.helloworld;
import java.util.Scanner;
public class AcountTest {
        public static void main(String[] args) {

                double depositAmount;

                Scanner input = new Scanner(System.in);

                Account account1 = new Account("Jane Green", 50.00);
                Account account2 = new Account("John Blue", -7.53);

                // exibe saldo inicial de cada objeto
                System.out.printf("%s balance: $%.2f \n",
                account1.getName(),account1.getBalance());

                System.out.printf("%s balance: $%.2f \n\n",
                account2.getName(),account2.getBalance());

                System.out.print("Enter deposit amount for account1: "); // prompt
                depositAmount = input.nextDouble(); // obtém a entrada do usuário

                System.out.printf("\nadding %.2f to account1 balance\n\n",
                depositAmount);
                account1.deposit(depositAmount); // adiciona o saldo de account
                // exibe os saldos
                System.out.printf("%s balance: $%.2f \n",
                account1.getName(),account1.getBalance());
                System.out.printf("%s balance: $%.2f \n\n",
                                account2.getName(),
                                account2.getBalance());

                System.out.print("Enter deposit amount for account2: "); // prompt
                depositAmount = input.nextDouble(); // obtém a entrada do usuário
                System.out.printf("\nadding %.2f to account2 balance\n\n",
                depositAmount);

                account2.deposit(depositAmount); // adiciona ao saldo de account2

                // exibe os saldos
                System.out.printf("%s balance: $%.2f \n",
                account1.getName(),account1.getBalance());
                System.out.printf("%s balance: $%.2f \n\n",
                account2.getName(),account2.getBalance());
        }
}
```

parte 1

parte 2

parte 3

A classe Account com um saldo; números de ponto flutuante

- Desafio !
- Ajuste para ajustar códigos repetidos em AccountTest:
  - possibilidade criar problemas de manutenção de código quando ele precisa ser atualizado

Mesmo código usamos em lugares diferentes

```
// exibe os saldos
System.out.printf("%s balance: $%.2f \n",
                  account1getName(),
                  account1getBalance());
System.out.printf("%s balance: $%.2f \n\n",
                  account2.getName(),
                  account2.getBalance());
```

Resolução

Adicionar método **displayAccount** em Account

```
public void displayAccount() {
        System.out.printf("%s balance: $%.2f \n\n",
        this.getName(),this.getBalance());
}
```

Em AccountTest, trocar a chamada anterior por

```
// exibe os saldos
account1.displayAccount();
account2.displayAccount();
```

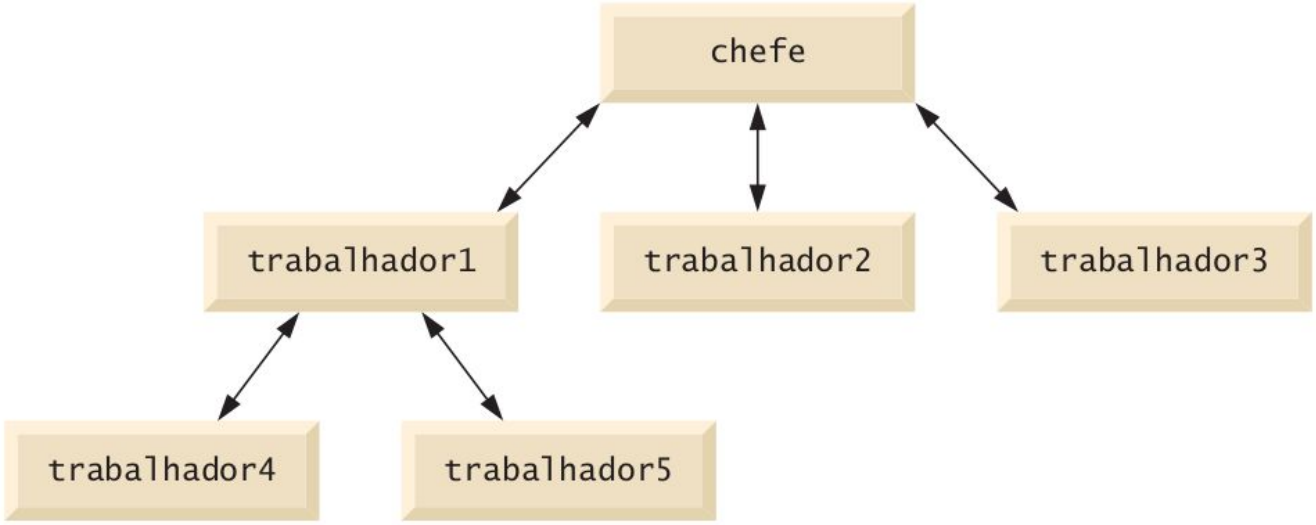
Métodos: um exame mais profundo

Considerações iniciais

- Boa maneira de desenvolver e manter um programa grande é **construí-lo a partir de pequenos e simples pedaços**, ou módulos.
- **Dividir para conquistar**
- 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
- Torna o desenvolvimento de **programas mais gerenciável**, construindo programas a partir de peças mais simples e menores
- Java API ou Biblioteca de classes Java

Considerações iniciais

- Relacionamento hierárquico
- Um chefe (o chamador) solicita que um trabalhador (o método chamado) realize uma tarefa e informe (retorne) os resultados depois de completar a tarefa.
- O chamador espera um resultado
- o método é que define o “como”



Métodos static, campos static e classe Math

Método static

- Um método realiza uma tarefa que não depende de um objeto
- Não sendo necessário de criar um objeto

```
Account account1 = new Account("Jane Green", 50.00);
```

- É comum que as classes contenham métodos static convenientes para realizar tarefas corriqueiras.
- Exemplo:

```
public static int somar(int a, int b) {
      return a + b;
}
```

Métodos static, campos static e classe Math

Os métodos da classe Math

- Permite realizar cálculos matemáticos comuns
- Lista de métodos estáticos disponíveis:

Exemplo de uso

```
System.out.println(Math.sqrt(900.0));
```

Método	Descrição	Exemplo
`abs(x)`	valor absoluto de *x*	`abs(23.7)` é 23.7 `abs(0.0)` é 0.0 `abs(-23.7)` é 23.7
`ceil(x)`	arredonda *x* para o menor inteiro não menor que *x*	`ceil(9.2)` é 10.0 `ceil(-9.8)` é -9.0
`cos(x)`	cosseno trigonométrico de *x* (*x* em radianos)	`cos(0.0)` é 1.0
`exp(x)`	método exponencial e^x	`exp(1.0)` é 2.71828 `exp(2.0)` é 7.38906
`floor(x)`	arredonda *x* para o maior inteiro não maior que *x*	`floor(9.2)` é 9.0 `floor(-9.8)` é -10.0
`log(x)`	logaritmo natural de *x* (base *e*)	`log(Math.E)` é 1.0 `log(Math.E* Math.E)` é 2.0
`max(x,y)`	maior valor de *x* e *y*	`max(2.3, 12.7)` é 12.7 `max(-2.3, -12.7)` é -2.3
`min(x,y)`	menor valor de *x* e *y*	`min(2.3, 12.7)` é 2.3 `min(-2.3, -12.7)` é -12.7
`pow(x,y)`	*x* elevado à potência de *y* (isto é, x^y)	`pow(2.0, 7.0)` é 128.0 `pow(9.0, 0.5)` é 3.0
`sin(x)`	seno trigonométrico de *x* (*x* em radianos)	`sin(0.0)` é 0.0
`sqrt(x)`	raiz quadrada de *x*	`sqrt(900.0)` é 30.0
`tan(x)`	tangente trigonométrica de *x* (*x* em radianos)	`tan(0.0)` é 0.0

Métodos static, campos static e classe Math

Variável static

- Uma variável estática em Java é uma variável que pertence à classe em vez de pertencer a instâncias individuais dessa classe.
- Isso significa que, independentemente de quantas instâncias da classe você criar, haverá apenas uma cópia da variável estática compartilhada por todas as instâncias.
- Constantes, exemplo: Math.PI (3,141592653589793)
- public **static** void main: permite que a JVM invoque main sem criar uma instância da classe.

exemplo de uso

```
public class ExemploVariavelStatic  {
    // Variável estática para contar o número de instâncias criadas
    public static int contadorInstancias = 0;
    // Construtor da classe
    public ExemploVariavelStatic () {
        // Incrementa o contador de instâncias ao criar uma nova
instância
        contadorInstancias++;
    }
    public static void main(String[] args) {
        // Criando algumas instâncias da classe
        ExemploVariavelStatic  obj1 = new ExemploVariavelStatic ();
        ExemploVariavelStatic  obj2 = new ExemploVariavelStatic ();
        ExemploVariavelStatic  obj3 = new ExemploVariavelStatic ();
        // Imprimindo o número de instâncias criadas
        System.out.println("Número de instâncias criadas: "  +
ExemploVariavelStatic .contadorInstancias);
    }
}
```

Métodos com múltiplos parâmetros

- Exemplo:

Utiliza um método chamado maximum para determinar e retornar o maior dos três valores double

Outras possibilidades:

- Criar classe com gets e sets
- Usar método do Math

```
Math.max(number3,
        Math.max(number1,
                 number2)
)
```

```
package methods;
import java.util.Scanner;
public class MaximumFinder {


	public static void main(String[] args) {
		Scanner input = new Scanner(System.in);

		System.out.print(
		"Enter three floating-point values separated by enter or space: "  );
		double number1 = input.nextDouble(); // lê o primeiro double
		double number2 = input.nextDouble(); // lê o segundo double
		double number3 = input.nextDouble(); // lê o terceiro double

		double result = maximum(number1, number2, number3);
		// exibe o valor máximo
		System.out.println("Maximum is: " + result);

	}

	// retorna o máximo dos seus três parâmetros de double
	public static double maximum(double x, double y, double z)
	{
		double maximumValue = x; // supõe que x é o maior valor inicial

		// determina se y é maior que maximumValue
		if (y > maximumValue) {
			maximumValue = y;
		}


		// determina se z é maior que maximumValue
		if (z > maximumValue) {
			maximumValue = z;
		}

		return maximumValue
	}
}
```

Promoção e coerção de argumentos

Regras que especificam quais conversões são autorizadas, isto é, quais conversões podem ser realizadas sem perda de dados

Exemplo:

- no método sqrt é esperado um double, mas permite int

```
System.out.println(Math.sqrt(4));
```

Essas conversões podem levar a erros de compilação se as regras de promoção do Java não forem satisfeitas

Conversões que não existe perda de dados

Regras de promoção

Tipo	Promoções válidas
double	None
float	double
int	long, float ou double
char	int, long, float ou double
boolean	Nenhuma (os valores boolean não são considerados números em Java)

Promoção e coerção de argumentos

Coerção Implícita

ocorre quando o tipo de destino é maior ou mais preciso que o tipo de origem. Não há perda de dados na conversão.

Exemplo:

```
int number = 10;
float number_float = number;
```

Coerção Explícita

ocorre quando o tipo de destino é menor ou menos preciso que o tipo de origem. **Pode resultar em perda** de dados se o valor não puder ser representado no tipo de destino.

Sintaxe: (tipoDestino) valor

Exemplo:

```
float number = 10.23;
int number_int = (int) number;
```

Escopo das declarações

O escopo de uma declaração é a parte do programa que pode referenciar a entidade declarada pelo seu nome

Variável global e local

Palavra-chave this e um ponto (.)

Exemplo ao lado ->

```
package methods;
public class Scope
{
	// campo acessível para todos os métodos dessa classe
	private static int x = 1;
	// O método main cria e inicializa a variável local x
	// e chama os métodos useLocalVariable e useField
	public static void main(String[] args)
	{
		int x = 5;
		System.out.printf("local x in main is %d%n", x);
		useLocalVariable();
		useField();
		useLocalVariable();
		useField();
		System.out.printf("%nlocal x in main is %d%n", x);
	}

	public static void useLocalVariable ()
	{
		int x = 25; // inicializada toda vez que useLocalVariable é chamado
		System.out.printf(
		"%nlocal x on entering method useLocalVariable is %d%n" , x);
		++x; // modifica a variável local x desse método
		System.out.printf(
		"local x before exiting method useLocalVariable is %d%n" , x);
	}
	// modifica o campo x da classe Scope durante cada chamada
	public static void useField ()
	{
		System.out.printf(
		"%nfield x on entering method useField is %d%n" , x);
		x *= 10; // modifica o campo x da classe Scope
		System.out.printf(
		"field x before exiting method useField is %d%n" , x);
	}
} // fim da classe Scope
```

UNIVERSIDADE DE PERNAMBUCO

Sobrecarga de método

- métodos com o mesmo nome podem ser declarados na mesma classe,
- contanto que tenham diferentes conjuntos de parâmetros (determinados pelo número, tipos e ordem dos parâmetros)
- comumente utilizada para criar vários métodos com o mesmo nome que realizam as mesmas tarefas, ou tarefas semelhantes, mas sobre tipos diferentes ou números diferentes de argumentos.
- Por exemplo, os métodos Math abs, min e max:
  - Uma com dois parâmetros double.
  - Uma com dois parâmetros float.
  - Uma com dois parâmetros int.
  - Uma com dois parâmetros long.
- podem ter diferentes tipos de retorno se os métodos tiverem diferentes listas de parâmetro.
- Além disso, métodos sobrecarregados não precisam ter o mesmo número de parâmetros.
- Se tiver o mesmo nome e conjunto de parâmetros será apresentado um erro

Sobrecarga de método

```java
package methods;
public class MethodOverload
{
	// teste de métodos square sobrecarregados
	public static void main(String[] args)
	{
		System.out.printf("Square of integer 7 is %d%n", square(7));
		System.out.printf("Square of double 7.5 is %f%n", square(7.5));
	}
	// método square com argumento de int
	public static int square(int intValue)
	{
		System.out.printf("%nCalled square with int argument: %d%n",
		intValue);
		return intValue * intValue;
	}
	// método square com argumento double
	public static double square(double doubleValue)
	{
		System.out.printf("%nCalled square with double argument: %f%n",
		doubleValue);
		return doubleValue * doubleValue;
	}
} // fim
```

```
Called square with int argument: 7
Square of integer 7 is 49

Called square with double argument: 7,500000
Square of double 7.5 is 56,250000
```