53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

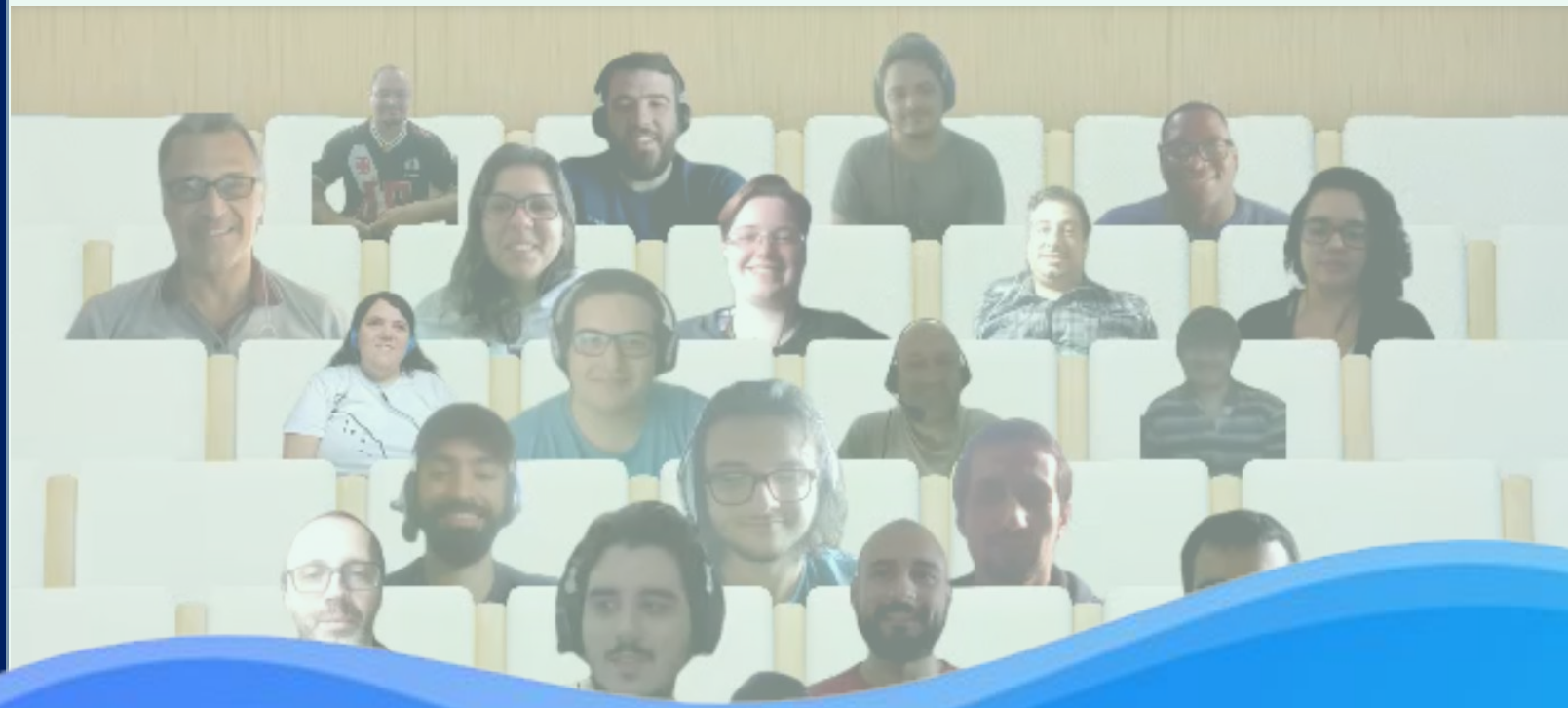
RESIDÊNCIA DE

SOFTWARE

CAPACITAR
TREINAR
EMPREGAR
TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana



Gerir objetos e classes: acesso, getters, setters, construtores, static

11/04/2022

JAVA

- Encapsulamento
- Construtor
- Atributos e métodos estáticos
- Escopo de variável

EXERCÍCIOS

Revisando aula anterior

1) Crie uma classe com o nome **Veiculo** com os atributos e métodos abaixo:

atributos:

- placa
- tipo
- valor

- Construa dois objetos em uma outra classe com o nome **MainVeiculo** com os seguintes dados:

  LZX9090, “Flex”,20000.

  KYZ1080, “Gás”,40600.

Métodos

- Crie um método para calcular o ipva para **Flex 4%** e para **Gás** 1.5**%**

RESOLUÇÃO EXERCÍCIO 1

```java
public class Veiculo {
    String placa;
    String tipo;
    double valor;

    public double calculaIPVA() {
        if (tipo.equals("Flex") || tipo.equals("Gasolina")) {
            return valor * 0.04;
        } else {
            return valor * 0.015;
        }
    }
}
```

RESOLUÇÃO EXERCÍCIO 1

```java
public class TesteVeiculo {

    public static void main(String[] args) {
        Veiculo veiculo = new Veiculo();
        veiculo.placa = "KYZ9034";
        veiculo.tipo = "Flex";
        veiculo.valor = 20000;

        System.out.println("O valor do IPVA é:" + veiculo.calculaIPVA());

    }

}
```

ENCAPSULAMENTO

Encapsular significa isolar, separar em partes um programa, além de esconder como funcionam os métodos, protegendo assim o acesso direto aos atributos e métodos de uma classe. Caso outros programadores acessem nossas classes garantimos que erros por mau uso não ocorram. Para isso se faz necessário o uso de **modificadores de acesso** mais restritivos nos atributos da classe. Esses atributos são manipulados indiretamente com o uso de métodos específicos.

Modificadores de Acesso

- Facilita checar valores inválidos, modificação e a implementação de correções
- Proteção contra acesso não autorizado
- Não é comum deixarmos os atributos de uma classe como **public**
- Para acessarmos os atributos utilizamos **getters** e **setters**

MODIFICADORES DE ACESSO UTILIZADOS

Public
Fica visível a classe, subclasses e pacotes do projeto Java.

Private
Deixa o atributo visível apenas para a classe em que o mesmo se encontra.

Protected
O atributo fica visível para todas as outras classes e subclasses que pertencem ao mesmo pacote.

Sem Modificador (Padrão)
Permite acesso apenas ao pacote em que o membro se encontra.

Modificador	Classe	Pacote	Sub-Classe	Global
public	SIM	SIM	SIM	SIM
private	SIM	NÃO	NÃO	NÃO
protected	SIM	SIM	SIM	NÃO
padrão	SIM	SIM	NÃO	NÃO

APLICANDO OS MODIFICADORES DE ACESSO

```java
package aulas;

public class Pessoa {
    int idPessoa;
    String nome;
    double peso;
    double altura;

    public double calculaImc() {
        double imc = peso / (altura * altura);
        System.out.println(imc);
        return imc;
    }

    public String resultado() {
        String situacao;
        if (calculaImc() < 18.5) {
            return situacao = "Abaixo do Peso";
        } else {
            if (calculaImc() >= 18.5 && calculaImc() <= 24.9) {
                return situacao = "Peso Normal";
            } else {
                return situacao = "Acima do Peso";
            }
        }

    }
}
```

Modificador de acesso padrão

APLICANDO OS MODIFICADORES DE ACESSO

```java
package aulas;

public class TestePessoa {

    public static void main(String[] args) {

        Pessoa pessoa = new Pessoa();

        pessoa.idPessoa = 1;
        pessoa.nome = "Mariana";
        pessoa.peso = 70;
        pessoa.altura = 1.70;

        System.out.println(pessoa.nome + " você está:" + pessoa.resultado());

    }

}
```

Problema:
Se atribuirmos um peso ou altura inválidos, negativo por exemplo?

Uma forma de resolver o problema seria colocando um **if** para testarmos se é um número válido.

```java
if (pessoa.altura > 0 && pessoa.peso > 0) {
    System.out.println(pessoa.nome + " você está:" + pessoa.resultado());
} else {
    System.out.println("O peso ou a altura estão inválidos !");
}
```

Se tivermos mais de um “Pessoa” este código vai se repetindo pelo programa.

APLICANDO OS MODIFICADORES DE ACESSO

Vamos alterar o modificador dos atributos para private na classe **Pessoa**

```
public class Pessoa {
    private int idPessoa;
    private String nome;
    private double peso;
    private double altura;
```

Modificador de acesso privado é o mais utilizado na maioria das classes

Ao tentar executar novamente a classe **TestePessoa**
O código retorna erro pois os atributos estão em modo **private** visível somente para a classe **Pessoa**

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field Pessoa.idPessoa is not visible
        The field Pessoa.nome is not visible
        The field Pessoa.peso is not visible
        The field Pessoa.altura is not visible
        The field Pessoa.altura is not visible
        The field Pessoa.peso is not visible
        The field Pessoa.nome is not visible
```

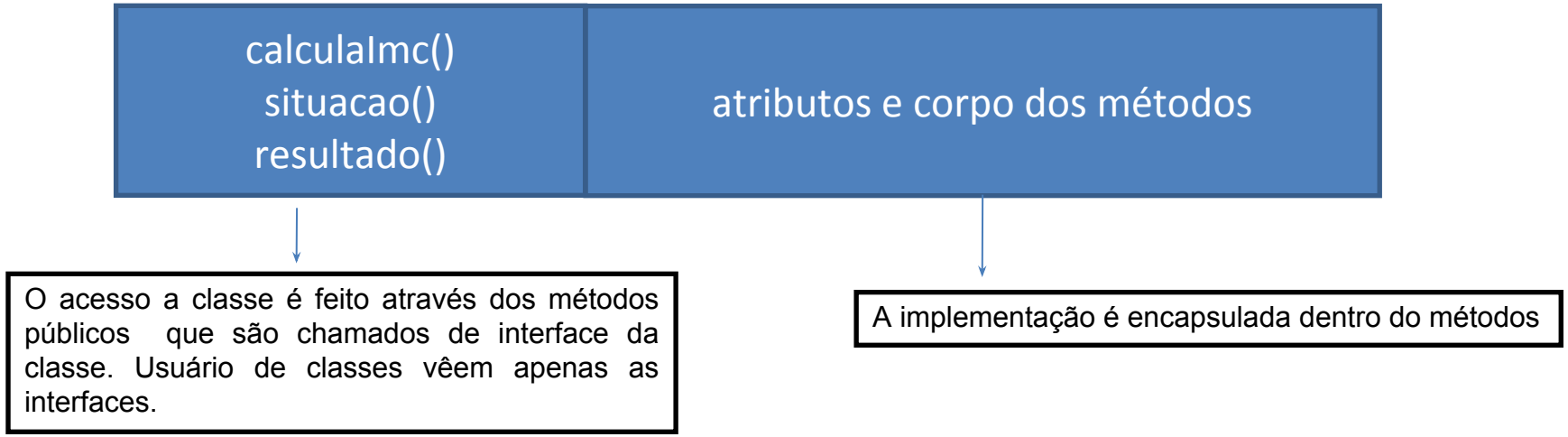
ABSTRAÇÃO DE DADOS

As classes, normalmente, ocultam os detalhes de implementação dos seus usuários. Isso se chama **ocultamento de informações.**

Podemos usar como exemplo um carro. O motorista faz uso do veículo mas não sabe do funcionamento interno do motor. Estamos preocupados com a funcionalidade que o carro oferece e não como o motor funciona isto é conhecido como abstração de dados.

Encapsulando

Fazendo o encapsulamento estamos escondendo os membros de uma classe e como funcionam os métodos do nosso sistema.



METODOS GETTERS E SETTERS

Para Manipular atributos privados utilizamos os **getters e setters**

set – define o valor do atributo
get – retorna o valor do atributo

Padrão usado na definição:

setNomeAtributo( )

getNomeAtributo( )

METODOS GETTERS E SETTERS

Pressionar ALT+SHIFT +S para Inserir Getters and Setters

```java
public class Pessoa {
    private int idPessoa;
    private String nome;
    private double peso;
    private double altura;

    public int getIdPessoa() {
        return idPessoa;
    }


    public void setIdPessoa(int idPessoa) {
        this.idPessoa = idPessoa;
    }
```

É sempre desejável que uma classe sempre esteja valores corretos, isto é, com os atributos sempre válidos. Para isto ao invés de permitir que o programador ou usuário atribua qualquer valor para os atributos, limita-se o acesso a eles por métodos, que fazem uma checagem dos dados antes de modificar os atributos. Outra utilidade é para se fazer atributos “somente leitura”, disponibilizando só o getter.

```java
public class TestePessoa {

    public static void main(String[] args) {

        Pessoa pessoa = new Pessoa();
        pessoa.setIdPessoa(1);
        pessoa.setNome("Mariana");
        pessoa.setPeso(70);
        pessoa.setAltura(1.70);

        if (pessoa.getAltura() > 0 && pessoa.getPeso() > 0) {
            System.out.println(pessoa.getNome() + " você está:" + pessoa.resultado());
        } else {
            System.out.println("O peso ou a altura estão inválidos !");
        }

    }
}
```

CONSTRUTOR

Um construtor permite que um determinado trecho de código seja executado toda vez que um objeto é criado, sempre que o comando **new** é utilizado. Para os construtores são passados argumentos que são obrigatórios na criação do objeto. Construtores não são métodos, não tem retorno e tem o mesmo nome da classe.

Exemplo 1:

*Produto.java

```
package aula;

public class Produto {
    private int cod;
    private String descricao;
    private double valor;

    public Produto(String descricao) {
        this.descricao = descricao;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String decricao) {
        this.descricao = decricao;
    }

}
```

Construtor criado. O atributo descricao é obrigado a ser informado na criação do objeto.

TestaProduto.java

```
package aula;

public class TestaProduto {
    public static void main(String[] args) {
        Produto p = new Produto("arroz");
        System.out.println(p.getDescricao());
    }

}
```

CONSTRUTOR PADRÃO

As classes dos exercícios nas aulas anteriores não possuíam nenhum construtor. Quando não declaramos nenhum construtor em uma classe, o compilador Java cria um construtor padrão. Esse construtor é o construtor default, ele não recebe nenhum argumento e o corpo dele é vazio. A partir do momento que declaramos um construtor, o construtor padrão não é mais fornecido.

Quero ter a opção de informar ou não a descrição como fazer?

*Produto.java

```
package aula;

public class Produto {
    private int cod;
    private String descricao;
    private double valor;

    public Produto() {
    }

    public Produto(String descricao) {
        this.descricao = descricao;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String decricao) {
        this.descricao = decricao;
    }

}
```

Criamos um construtor padrão (vazio).

TestaProduto.java

```
package aula;

public class TestaProduto {
    public static void main(String[] args) {
        Produto p = new Produto();
        p.setDescricao("Arroz");
        System.out.println(p.getDescricao());
    }

}
```

Temos agora opção de não informar a descrição na criação do objeto.

SOBRECARGA DE CONSTRUTOR (OVERLOAD)

Abaixo um novo construtor foi criado na classe Produto para termos a opção de receber todos os campos

```
public Produto() {

}

public Produto(String descricao) {
    this.descricao = descricao;
}

public Produto(int cod, String descricao, double valor) {
    this.cod = cod;
    this.descricao = descricao;
    this.valor = valor;
}
```

Insira o construtor no eclipse usando
ALT+SHIFT+S

Quando definimos mais de um construtor temos o que chamamos de sobrecarga de construtores acima temos 3 opções de usar um construtor na criação de um objeto.

TestaProduto.java

```
package aula;

public class TestaProduto {
    public static void main(String[] args) {
        Produto p = new Produto(1,"arroz",2.5);
        System.out.println(p.getCod()+ " "
                + p.getDescricao()
                + " "+p.getValor());
    }

}
```

UM CONSTRUTOR FAZENDO UMA CHAMADA PARA OUTRO CONSTRUTOR

Insira o código em destaque na classe Produto

```
public Produto() {
    System.out.println("Construtor vazio");
}

public Produto(String descricao) {
    this();
    this.descricao = descricao;
    System.out.println("Construtor descricao");
}

public Produto(int codigo, String descricao, double valor) {
    this(descricao);
    System.out.println("Construtor com todos atributos");
    this.codigo = codigo;
    this.descricao = descricao;
    this.valor = valor;
}
```

O construtor padrão é executado

O segundo construtor é executado primeiro.

O comando this() é usado para chamarmos outro construtor

ATRIBUTOS E MÉTODOS ESTÁTICOS

São usados através da classe e não pelos objetos quando são criados. Todos os objetos podem compartilhar os métodos e atributos estáticos.

```
public class Produto {
    private int cod;
    private String descricao;
    private double valor;
    private static int totalProdutos;
```

Insira o atributo totalProdutos na classe Produto. Este atributo pertence a classe

```
public static double getTotalProdutos() {
    return totalProdutos;
}
```

Insira o método getTotalProdutos ALT+SHIFT+S na classe Produto. Este método é compartilhado por toda classe e não para cada objeto independente.

```
public Produto(int cod, String descricao, double valor) {
    this.cod = cod;
    this.descricao = descricao;
    this.valor = valor;
    Produto.totalProdutos += 1;
}
```

Acrescente a linha em destaque no construtor para que quando for criado um objeto é incrementado o total de produtos.

```
Produto p1 = new Produto(8,"Feijao",1.8);
System.out.println(p1.getCod()+ " "
        + p1.getDescricao()
        + " "+p1.getValor());

System.out.println("Total de Produtos:" + Produto.getTotalProdutos());
```

Insira um novo objeto na classe TestaProduto e faça o teste

EXERCÍCIOS

1) Criar um novo projeto com o nome **aula3.** Criar um pacote com o nome **aulas.**

Criar uma classe com o nome **Medico**
- atributos: crm, nome e salario e valor da consulta.

- Métodos da classe **Medico:**
- Crie um método **pagamentoDinheiro** para pagamentos em dinheiro.
- Crie o método **pagamentoPlano** para pagamentos com plano de saúde

Obs: O médico receberá 70% do valor da consulta no plano de saúde.

- Construa dois objetos em uma outra classe com o nome **TestaMedico** com os seguintes dados:
- Crie um construtor vazio e outro com todos os dados da conta na classe **Medico**. Passe os dados na construção dos objetos.

crm: 12345
nome: Ana Maria
salario: 0
valorConsulta: 250

crm: 456789
nome: Antônio
salario: 0
valorConsulta: 300

- Fazer uma consulta com pagamento em dinheiro
- Fazer uma consulta com pagamento com plano de saúde
- Exiba na tela os dados dos médicos.
-Exiba o número total de médicos.

RESOLUÇÃO

```java
public class Medico {
    private int crm;
    private String nome;
    private double salario;
    private double valorConsulta;
    private static int totalMedicos;

    public Medico(int crm, String nome, double salario, double valorConsulta) {
        this.crm = crm;
        this.nome = nome;
        this.salario = salario;
        this.valorConsulta = valorConsulta;
        totalMedicos += 1;
    }

    public String getNome() {
        return nome;
    }

    public double getSalario() {
        return salario;
    }

    public static int getTotalMedicos() {
        return totalMedicos;
    }

    public static void setTotalMedicos(int totalMedicos) {
        Medico.totalMedicos = totalMedicos;
    }

    public void pagamentoDinheiro(double valorConsulta) {
        salario = salario + valorConsulta;
    }

    public void pagamentoPlano(double valorConsulta) {
        salario = salario + valorConsulta * 0.7;
    }

}
```

RESOLUÇÃO

```
public class TesteMedico {

    public static void main(String[] args) {
        Medico medico1 = new Medico(12345,"Ana Maria",0,250);
        Medico medico2 = new Medico(456789,"Antônio",0,300);

        medico1.pagamentoDinheiro(250);
        medico2.pagamentoPlano(300);

        System.out.println(medico1.getNome() +"\n" + medico1.getSalario());
        System.out.println(medico2.getNome() +"\n" + medico2.getSalario());
        System.out.println("Total de Médicos:" + Medico.getTotalMedicos());

    }

}
```

ESCOPO DE VARIÁVEL

Existem três tipos de escopo de variáveis: escopo local, escopo de instância, e escopo de classe.

- **Variáveis locais**

A variável local está limitada ao escopo local ,isto é, está limitada a um bloco de código. Escopos locais podem ser métodos, ifs, construtores, loops...

- **Variáveis de instância**

Variáveis de instâncias são variáveis que existem enquanto existir uma instância da classe onde a variável foi declarada. Variáveis de instância são declaradas fora de construtores ou métodos, são membros de uma classe.

- **Variáveis de classe**

Variáveis de classe são as variáveis estáticas que não precisam de uma instância para existir e são compartilhada entre todas a instancias de uma classe. Essas variáveis estáticas são carregadas junto com a classe quando a classe é carregada em memória.

ESCOPO DE VARIÁVEL

Variáveis locais

```
if(a==null) {
    int c = 0;
    System.out.println(c);
}
```

Variáveis de classe

```
public class Pessoa {
    static int idade;
    static double peso;
}
```

Variáveis de instância

```
public class Imovel {

    int codImovel;
    String bairro;
    String tipo;
    double valor;

    public void calcularReajuste() {
        if (this.tipo == "casa") {
            this.valor *= 1.05;
        } else {
            this.valor *= 1.08;
        }
    }
}
```

OPERADORES TERNÁRIOS

O operador ternário é um recurso condicional parecido com o do **if/else**, mas que é codificado em apenas uma linha.
Ao avaliar a expressão booleana, caso ela seja verdadeira,valor 1, declarado após o ponto de interrogação (?) será executado; caso contrário, será executado o valor 2, declarado após os dois pontos (:).

Sintaxe do operador ternário:
(expressão booleana) ? valor 1 : valor 2;

```
public class Ternario {
    public static void main(String[] args) {
        int a=10,  b=5;

        if (a==b) {
            System.out.println("Verdadeiro");
        }else{
            System.out.println("Falso");
        }


        System.out.println((a==b) ? "verdadeiro":"Falso");
    }
}
```

OPERADORES TERNÁRIOS

```java
public class Ternario2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Qual seu dia da semana favorito?");
        System.out.println("Escolha de 1 a 7:");
        int numero = sc.nextInt();

        String dia = (numero == 1) ? "segunda":
                     (numero == 2) ? "terça":
                     (numero == 3) ? "quarta":
                     (numero == 4) ? "quinta":
                     (numero == 5) ? "sexta":
                     (numero == 6) ? "sabado":
                     (numero == 7) ? "domingo":
                                     "dia inválido";
        System.out.println("Dia escolhido:" + dia);
    }
}
```

```java
String dia;
switch (numero) {
case 1:
    dia = "segunda";
    break;
case 2:
    dia = "terça";
    break;

default:
    dia ="inválido";
    break;
}
}
```

O conteúdo acima pode ser substituído por um switch Conforme exemplo ao lado

Outro Exemplo

```java
double salario = 900;
double bonus = salario* (salario < 1000? salario * 0.15:salario * 0.10);
System.out.println(salario);
```

EXERCÍCIOS

1) Criar uma classe com nome **Funcionario** com os seguintes atributos:
   - **nome**
   - **salario**

Encapsular os atributos.

Inserir o construtor com todos os atributos.

Criar um método com o nome **calcularInss**. Este método deverá retornar um **double** sobre 11% do salário.
Criar um método com o nome **calcularValeTransporte**. Este método deverá retornar um **double** sobre 6% do salário.

Criar uma classe com o main e instanciar um funcionário.
Exibir o salário líquido do funcionário conforme imagem abaixo.

EXERCÍCIOS

2) Criar uma classe com nome **Produto** com os atributos abaixo:
- **descricao**
- **valor**
- **quantidade**

Encapsular os atributos.
Inserir o construtor com todos os atributos.

Criar um método com o nome **calcularIcms**. Este método deverá retornar um **double** sobre 12% do total.

Criar uma classe com o método **main**

Instanciar os produtos até a resposta digitada for igual a '**S'** conforme imagem abaixo.

```
Produto:
Arroz
Valor:
25
Quantidade:
2
Descrição:Arroz
Valor:25.0
Total:50.0
ICMS:6.0

Deseja encerrar o programa? (S/N)
```

EXERCÍCIOS

3) Crie uma classe com o nome **Calculadora.** Esta classe deverá conter um método para calculo das operações básicas e retornar um valor como **double**.

3.1)Criar uma classe **MainCalculadora** com o método main com um menu com 5 opções:

1 – soma
2 – subtração
3 – multiplicação
4 – divisão
5 – sair

Entrar com os dois valores via console ou **JOptionPane** e exibir o resultado da operação.

RESOLUÇÃO EXERCÍCIO 2

```java
public class Calculadora {
    public double calcular(double a, double b, int operacao) {
        double resultado = 0;
        switch (operacao) {
        case 1:
            resultado = a + b;
            break;
        case 2:
            resultado = a - b;
            break;
        case 3:
            resultado = a * b;
            break;
        case 4:
            resultado = a / b;
            break;

        default:
            break;

        }
        return resultado;

    }
}
```

```java
import javax.swing.JOptionPane;

public class TesteCalculadora {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        int opcao;
        double resultado;
        String menu = "Calculadora\n\n" + "1-Somar\n" + "2-Subtrair\n" +
        "3-Multiplicar\n" + "4-Dividir\n" + "5-Finalizar\n\n";
        opcao = Integer.parseInt(JOptionPane.showInputDialog(null, menu,
                "Calculadora - JAVA", JOptionPane.QUESTION_MESSAGE));

        while (opcao != 5) {
            String numero1 = JOptionPane.showInputDialog("Valor 1:");
            String numero2 = JOptionPane.showInputDialog("Valor 2:");
            resultado = calc.calcular(Double.parseDouble(numero1), Double.parseDouble(numero2), opcao);
            JOptionPane.showMessageDialog(null, resultado);
            opcao = Integer.parseInt(JOptionPane.showInputDialog(null, menu,
                    "Calculadora - JAVA", JOptionPane.QUESTION_MESSAGE));
        }

    }
}
```