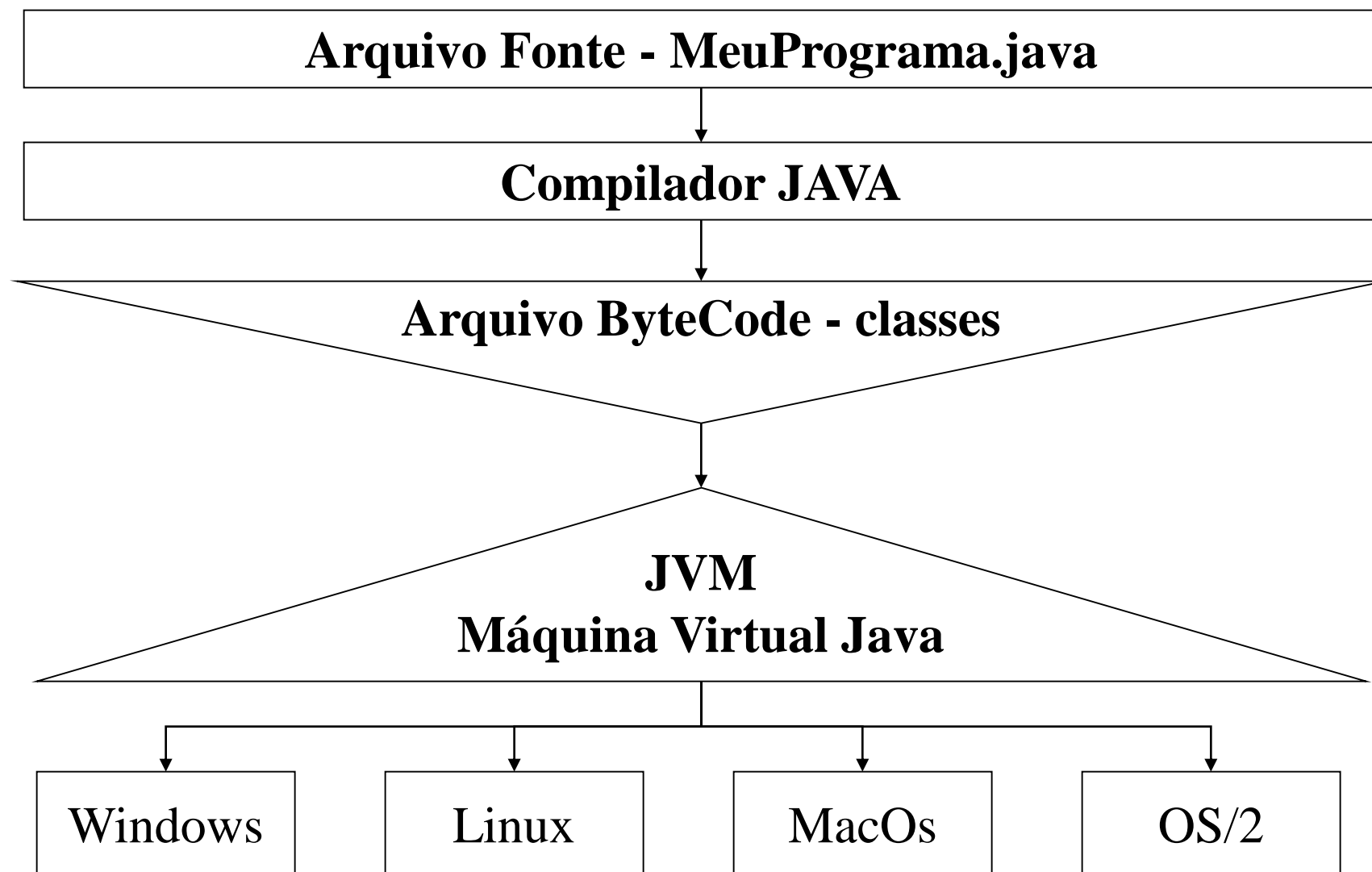
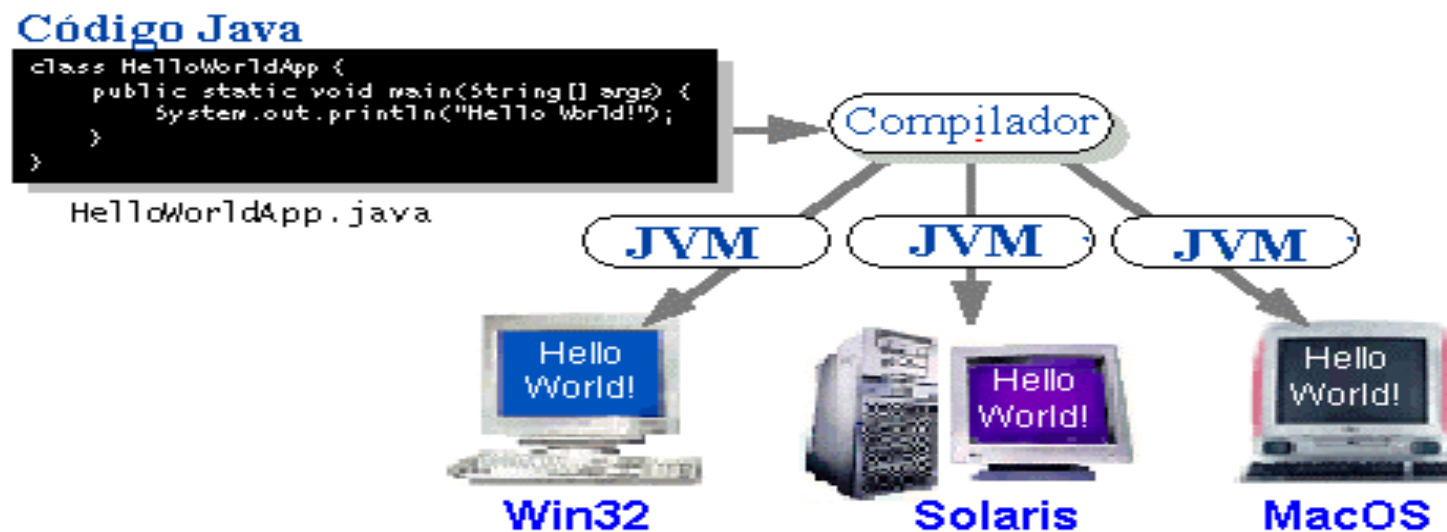


Tópico 1 – Programação Orientada a Objetos (POO)

- **Introdução ao JAVA;**
- **Máquina Virtual;**
- **Garbage-collection**
- **IDE Netbeans – criação de um projeto**



- Máquina Virtual



- **Garbage-collection**

É um mecanismo de controle automático de alocação e liberação de memória.

- **Tipos de Variáveis**

Para tipos numéricos com valores inteiros, positivos ou negativos, temos:

| Nome | Memória | Mínimo | Máximo |
|-------|---------|--------|---------|
| byte | 1 byte | -128 | 127 |
| short | 2 bytes | -32768 | 32767 |
| int | 4 bytes | -231 | 231 – 1 |
| long | 8 bytes | -263 | 263 – 1 |

Para tipos numéricos com valores fracionários, positivos ou negativos, temos:

| Nome | Memória |
|--------|---------|
| float | 4 bytes |
| double | 8 bytes |

- **Declarando uma variável**

[modificador] tipo indentificador = valor.

Onde:

Modificadores: podem ser public ou private. Modificadores são opcionais.

Tipo: é o tipo de dado que a variável irá receber

Identificador: o nome que identifica a variável

Exemplos:

```
public int base=0;
```

```
public char sexo='F';
```

```
public double nota=4,5
```

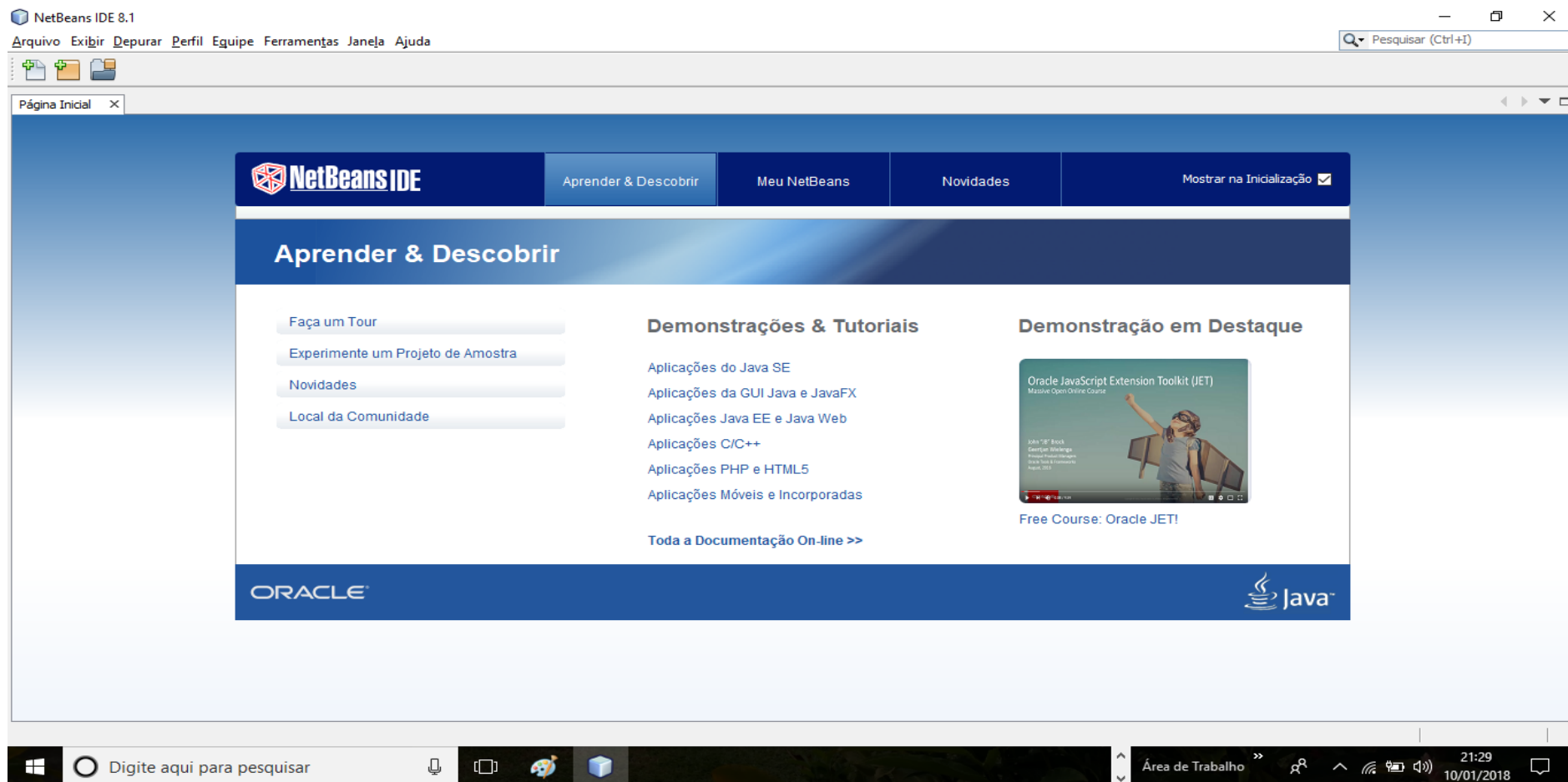
```
public int idade=20;
```

Considerações para Nomes de variáveis

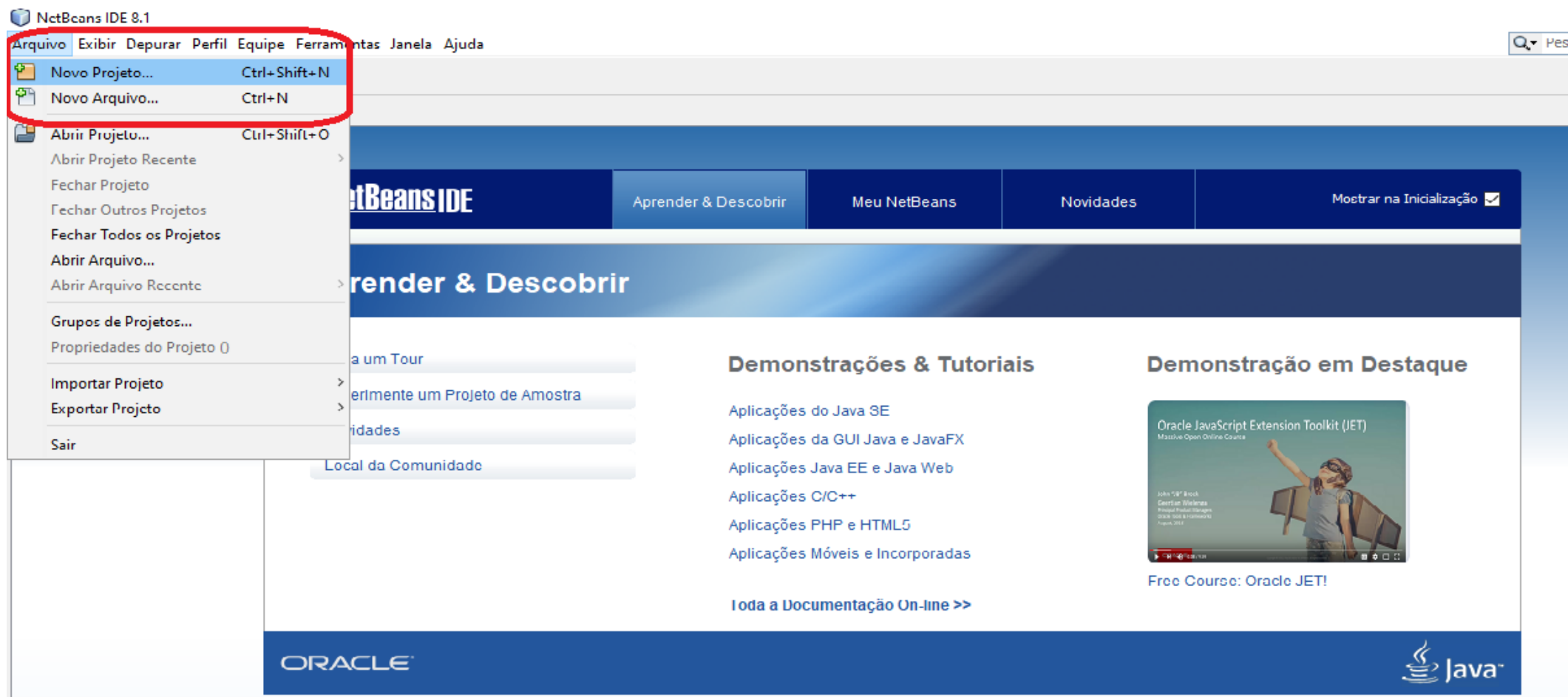
- Devem ser simples mas descritivos
- Não pode utilizar palavras reservadas na linguagem java.
- Deve iniciar com uma letra do alfabeto.
- O caracter underscore (sublinhado)
- O caracter dolar \$
- Nome de variáveis podem ser acentuados, mas não se recomenda acentos.
- Não pode conter espaço entre palavras.
- Variáveis são case sensitivity. NOME <> nome

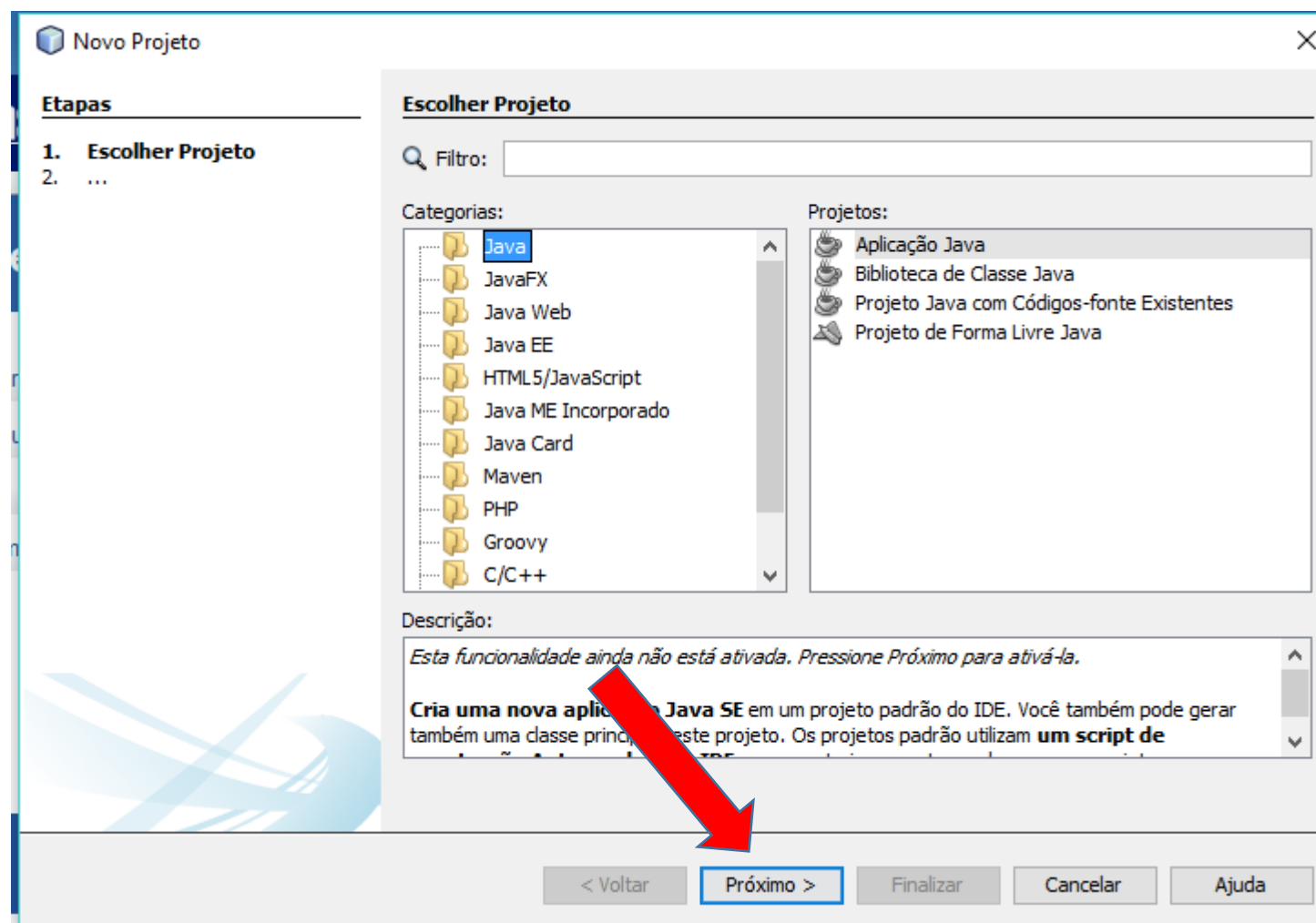
| CERTO | ERRADO |
|---------------|---------------|
| _nome | Nome do aluno |
| \$nome | -nome |
| Nome_do_aluno | 1ºnome |
| nomeDoAluno | |
| Nome23 | |
| NOME | |
| Nome | |
| @nome | |

- IDE (*Integrated Development Environment*) ou Ambiente Integrado de Desenvolvimento - NetBeans



- Criando um projeto





- Inserir o nome do Projeto no campo **Nome do Projeto**;

Novo Aplicação Java

Etapas

1. Escolher Projeto
2. **Nome e Localização**

Nome e Localização

Nome do Projeto:

Localização do Projeto: Procurar...

Pasta do Projeto:

☐ 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

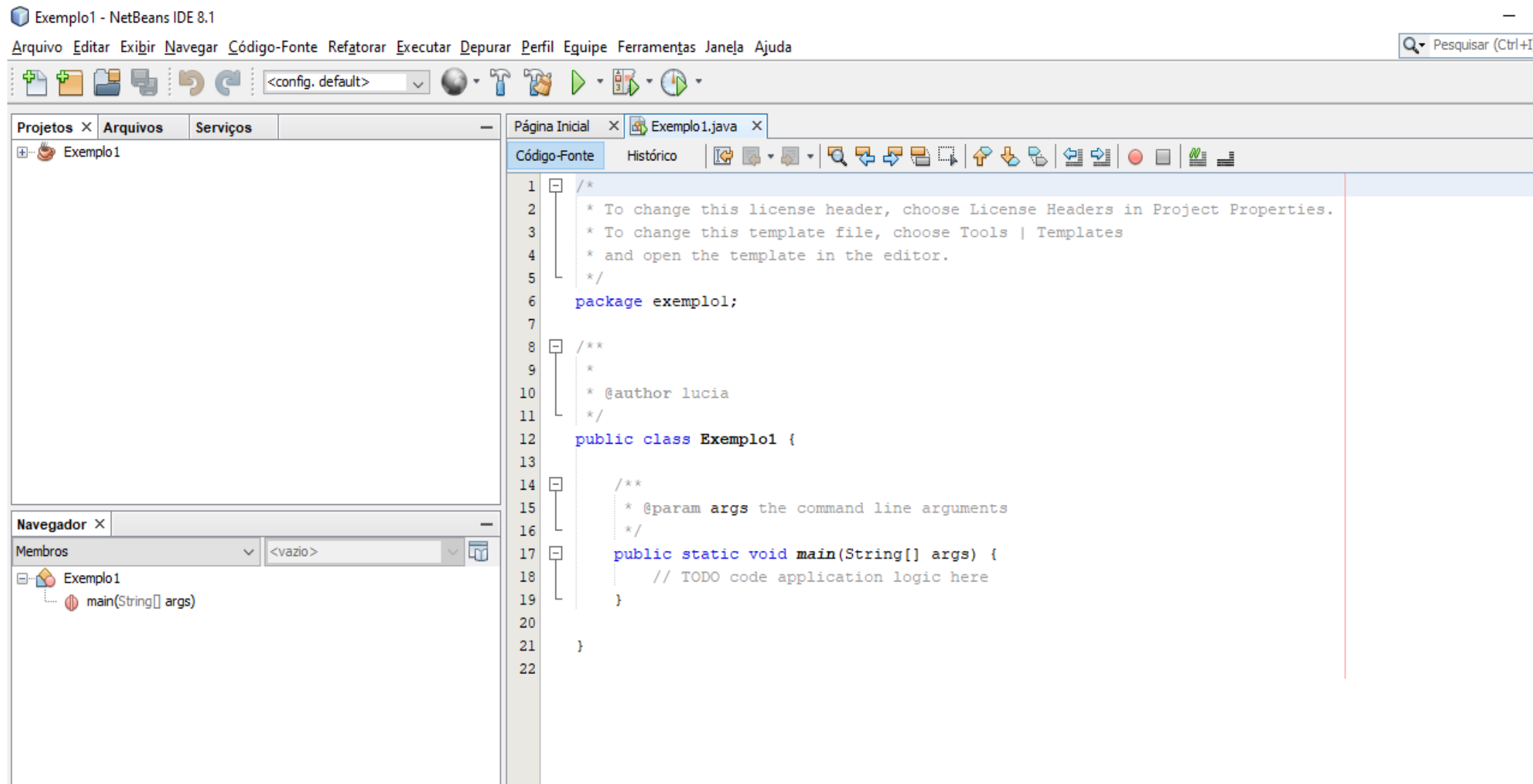
< Voltar Próximo > Finalizar Cancelar Ajuda

- No campo **Localização do Projeto**, poderá escolher uma pasta (diretório), onde você deseja que seu projeto seja criado.
- No campo **Pasta do Projeto**, você verá o caminho completo, onde estará localizado seu novo projeto. Neste campo você poderá observar também, que será criada uma pasta com o nome que você deu ao seu projeto, e esta pasta será criada dentro da última pasta do caminho que você escolheu no campo **Localização do Projeto** (campo imediatamente acima do campo **Pasta do Projeto**).

E para finalizar,

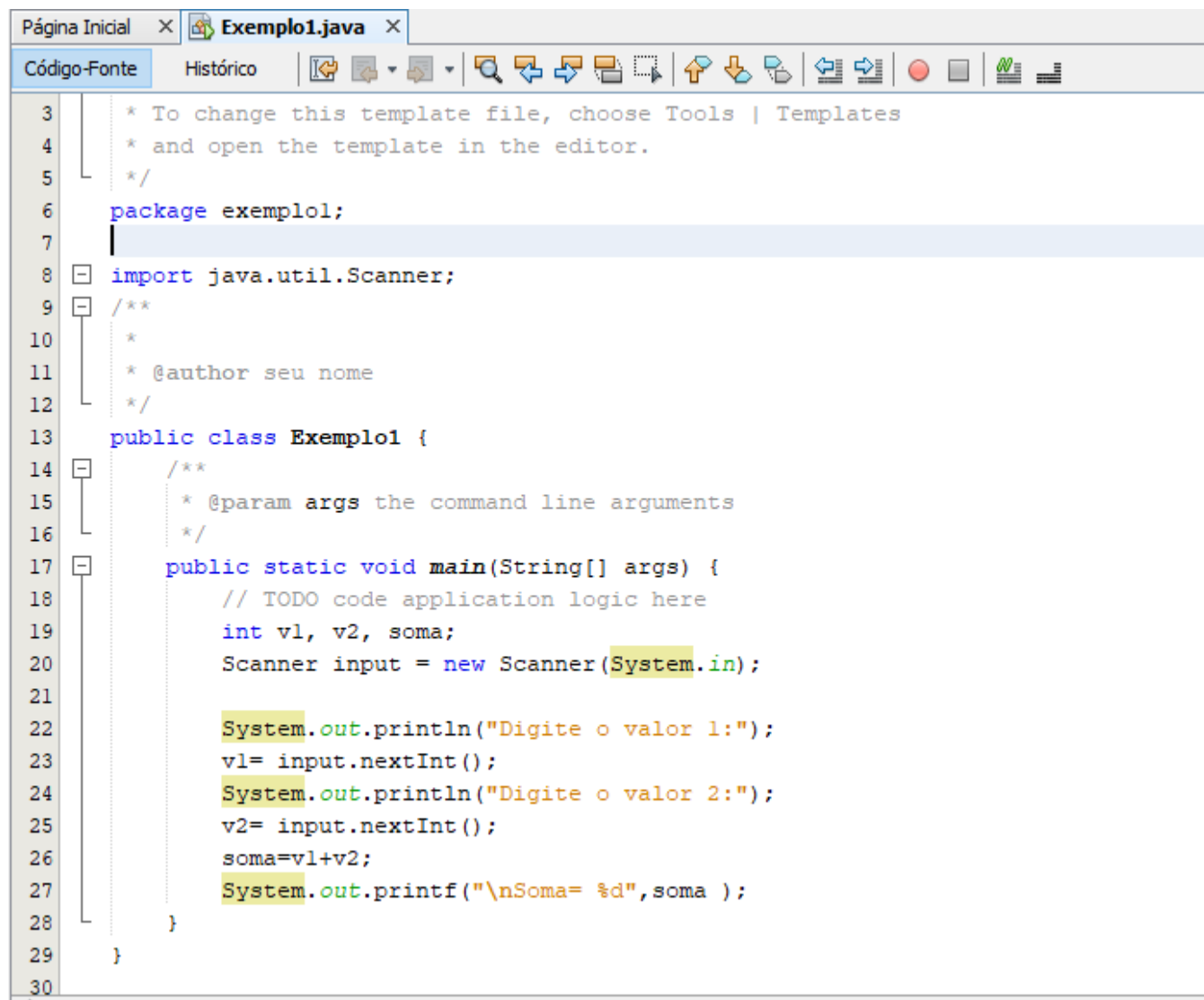
- temos o campo **Criar classe principal**, que possui um Check Box e, quando este estiver marcado, então a classe Main será criada com o nome que você escolher. Caso você não escolha um nome específico, será utilizado o nome que você deu ao projeto;
- Tudo preenchido, então aperte o botão **Finalizar**. Será mostrado uma tela com um novo projeto já iniciado

- Projeto criado



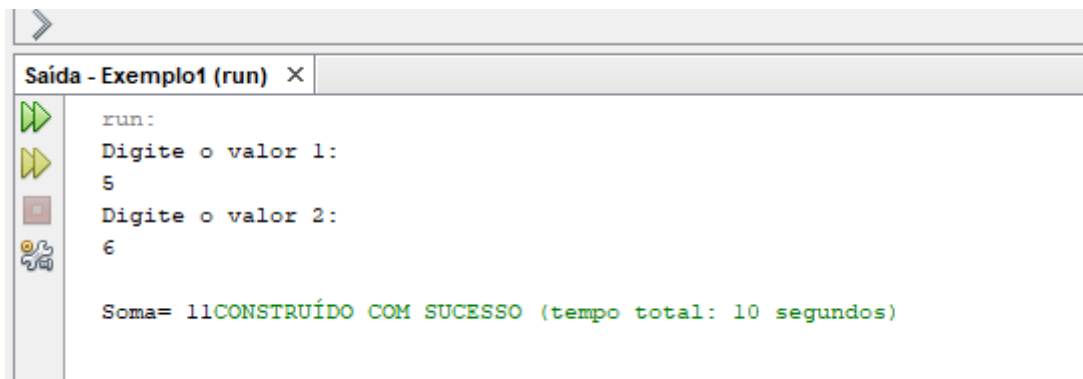
- A IDE, cria um projeto com o nome que você escolheu;
- Cria também um **package** com o nome que você escolheu para o projeto.
- Cria ainda, uma classe chamada Main, isto se você não desmarcou e tampouco alterou o nome no campo **Criar classe principal**,
- E para finalizar, a IDE cria também um método (função) especial, chamado “*main*”.

- Complete o código com as instruções abaixo:



```
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package exemplo1;
7
8  import java.util.Scanner;
9
10 /**
11  * @author seu nome
12  */
13 public class Exemplo1 {
14     /**
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         // TODO code application logic here
19         int v1, v2, soma;
20         Scanner input = new Scanner(System.in);
21
22         System.out.println("Digite o valor 1:");
23         v1= input.nextInt();
24         System.out.println("Digite o valor 2:");
25         v2= input.nextInt();
26         soma=v1+v2;
27         System.out.printf("\nSoma= %d",soma );
28     }
29 }
30
```

- Após ter escrito o código-fonte, então compile-o e execute-o. Para compilação e execução em sequencia, basta pressionar a tecla **F6**.
- Saída após a execução:



```
run:
Digite o valor 1:
5
Digite o valor 2:
6

Soma= 11CONSTRUÍDO COM SUCESSO (tempo total: 10 segundos)
```


Entendendo o código

- **package** – Mecanismo para organização de classes Java, dentro de um mesmo *namespace*. Esta organização pode-se dar de diversas formas: similaridade, funcionalidade e etc.
- **import** – Semelhante ao comando *#include da linguagem C. Através do comando import*, pode-se utilizar outras classes e seus métodos, dentro da classe que está sendo construída.
- **public** – Modificador de acesso, refere-se a acessibilidade que se pode ter em relação a um método, classe ou variável.

Acessibilidade de variáveis e métodos depende dos modificadores de acesso, que se coloca (ou não) diante da variável em sua declaração, método ou classe em sua definição:

- ☐ **public** : *Dá acessibilidade completa à variável. Esta pode ser acessada a partir da própria classe e também por outras classes, estando estas classes no mesmo package ou não.*
- ☐ **protected**: *Dá acessibilidade para todas as classes que estão no mesmo package.*
- ☐ **private**: *Só se pode acessar desde a própria classe.*
- ☐ **static** – **O modificador *static* nos garante que somente haverá uma, e não mais que uma**, referência para determinada variável ou método disponível em memória. Em outras palavras, declarando alguma coisa como *static*, *todas as instâncias da classe compartilharão a mesma* cópia da variável ou método. Declarar algo como *static* também *permite* você acessar métodos e atributos diretamente, ou seja, sem precisar criar uma instância da classe.
- ☐ **void** – **Palavra-chave utilizada na declaração de funções que não precisam retornar um valor.**

int – Palavra-chave utilizada na declaração/criação de variáveis do tipo inteiro, ou seja, criação de espaços na memória do computador, capazes de armazenar valores do tipo inteiro.

public static void main(String[] args) - Assinatura do método principal.

{ - Início do corpo do método (função).

} - Fim do corpo do método.

“int v1,v2,soma” - Criação de variável do tipo inteiro. Ou seja, criação de um espaço na memória do computador, onde pode-se guardar valores do tipo inteiro.

// Este é um comentário de uma só linha .

/* Este é um comentário de uma ou mais linhas. ***/**

/** Este é um comentário de documentação. ***/**

Comentário - O que estiver como comentário não é executado pelo compilador. Utilizado para documentar o *software*.

System.out.print - Comando para impressão de *strings na tela do computador*.

System.out.printf - Comando para impressão de *strings, associadas a valores em variáveis*, na tela do computador.

new – Comando utilizado para instanciação de objetos.

Scanner – Classe utilizada para captação de dados via teclado.

Entrada de Dados

Para obtermos uma entrada de dados do usuário, utilizaremos uma instrução chamada **Scanner**.

Primeiro dizemos ao computador que queremos pegar entrada do usuário com:

```
Scanner leitor = new Scanner(System.in);
```

o objeto '**leitor**' será usado para ler entradas do sistema.

Depois requisitamos as entradas com:

```
variável = leitor.nextDouble();    // para dados do tipo double;
```

```
variável = leitor.nextInt();       // para dados do tipo int;
```

```
variável = leitor.nextFloat();     // para dados do tipo float;
```

System.out – Objeto de Saída de Dados em Java

Dentro desse objeto existem métodos para gerar saídas de **Strings**, dentre eles:

- **print()**,
- **println()**,
- **printf()**

System.out.print()

```
System.out.print("Primeiro exemplo de saída de dados!");
```

Exemplos:

```
System.out.print("João");
```

```
System.out.print("da Silva");
```

`System.out.println()`

```
System.out.println("Segundo exemplo de saída de dados!");
```

O '**ln**' de '**println**' é de '**line**', pois essa função imprime uma linha, e linha inclui uma quebra de linha (ou newline, ou `\n`, ou [enter], ou parágrafo).

Ou seja, a função '**print**' não inclui essa quebra. Como colocar essa quebra no '**print**'?

Existe um símbolo especial para isso, é o '`\n`'.

```
System.out.print("Segundo exemplo de saída de dados!\n");
```

System.out.printf()

O argumento do método printf é uma **String de formato** que pode consistir em texto fixo e **especificadores de formato**. A letra “f” no final da palavra “print” significa “**formatted**”, ou seja, exibe os dados formatados.

Os **especificadores de formato** são como marcadores de lugares para um valor, especificando o tipo da saída dos dados que iniciam com um sinal de porcentagem (%) seguido por um caractere representando seu tipo de dado. Na tabela abaixo alguns especificadores de formato:

| | |
|------------|------------------------------------|
| %d | representa números inteiros |
| %f | representa números floats |
| %2f | representa números doubles |
| %b | representa valores booleanos |
| %c | representa valores char |
| %s | representa Strings |

Exemplos:

```
System.out.printf("%s\n %s\n", "Fatec", "Zona Sul");
```

```
System.out.printf("Soma das variáveis num1 e num 2 = %d",soma);
```