Técnicas de Programação II

Aula 01 – Introdução à Linguagem Java

Edirlei Soares de Lima
<edirlei.lima@uniriotec.br>

Paradigmas de Programação

- Um paradigma de programação consiste na filosofia adotada na construção de softwares:
  - Imperativo ou Procedural (C, PHP, Fortran, ...);
  - Declarativo ou Lógico (Prolog, ...);
  - Funcional (Lisp, Haskell, ...);
  - **Orientado a Objetos** (Java, C++, C#, ...);

Paradigma Orientado a Objetos

- Consiste em um paradigma de **análise**, **projeto** e **programação** de sistemas baseado na composição e interação entre diversas unidades de software chamadas de **objetos**.

- Sugere a diminuição da distância entre a **modelagem computacional** e o **mundo real**:
  - O ser humano se relaciona com o mundo através de conceitos de objetos;
  - Estamos sempre identificando qualquer objeto ao nosso redor;
  - Para isso lhe damos nomes, e de acordo com suas características lhes classificamos em grupos;

Paradigma Orientado a Objetos

- O paradigma orientado a objetos é uma forma de entender e representar **sistemas complexos** como **estruturas hierárquicas de objetos** que se **relacionam**.

- **Vantagens:**
  - Organização do código;
  - Aumenta a reutilização de código;
  - Reduz tempo de manutenção de código;
  - Reduz complexidade através da melhoria do grau de abstração do sistema;
  - Aumenta qualidade e produtividade, oferecendo maiores facilidades ao desenvolvedor;
  - Ampla utilização comercial;

Linguagem Java

- Java é uma linguagem de programação **orientada a objetos** desenvolvida na década de 90

- Baseia-se na sintaxe da linguagem **C/C++**

- Programas desenvolvidos em Java são **compilados para bytecode** e executados pela **Maquina Virtual Java**
  - Liguagens compiladas vs linguagens interpretadas;

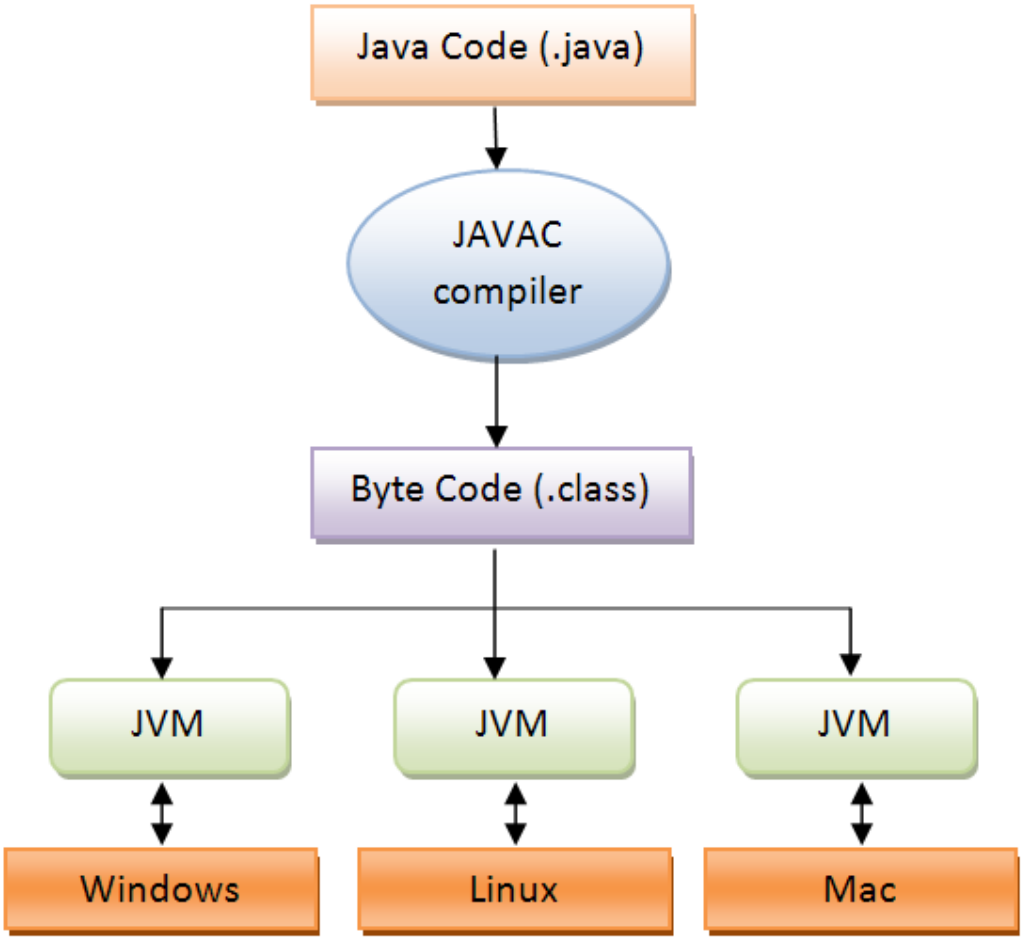
- Segunda linguagem mais utilizada atualmente (Agosto, 2014)
  - http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Linguagem Java

- **Portabilidade:** escreva uma vez, execute em qualquer lugar;
  - Windows, Linux, Mac, Celulares…
  - A portabilidade é obtida pelo fato da linguagem ser interpretada, ou seja, o compilador gera um código independente de máquina chamado **bytecode**
  - No momento da execução, este bytecode é interpretado por uma **máquina virtual** instalada no sistema

- **Máquina Virtual Java** (Java Virtual Machine - JVM) é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina.
  - Permite que os programas escritos em Java possam funcionar em qualquer plataforma que possua uma versão da JVM.

Máquina Virtual Java



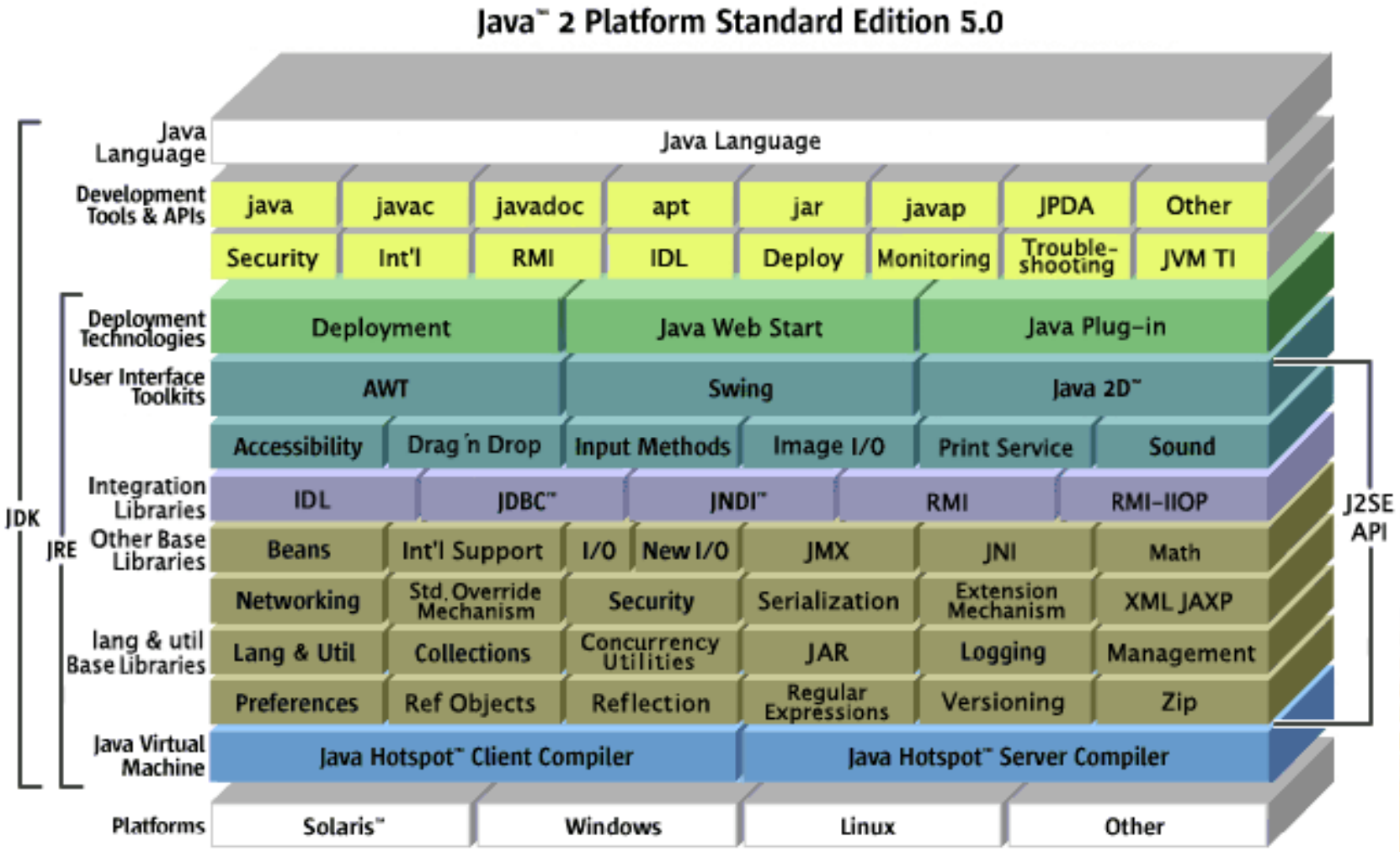
JDK e JRE

- **Edições:**
  - Java Standard Edition (Java SE)
  - Java Enterprise Edition (Java EE)
  - Java Mobile Edition (Java ME)

- **Java Development Kit (JDK)**
  - Ferramentas de desenvolvimento

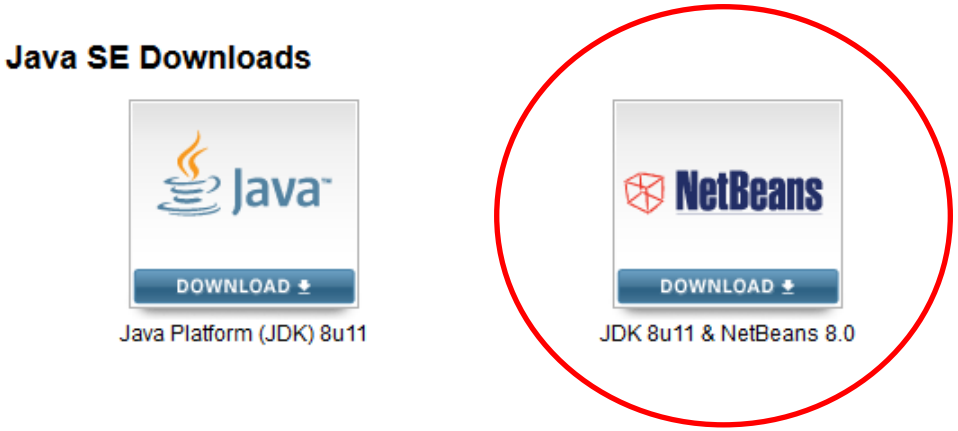
- **Java Runtime Environment (JRE)**
  - Necessária para rodar programas Java (bytecodes compilados);
  - É a única parte da plataforma Java que os clientes precisam instalar;

JDK e JRE


Java™ 2 Platform Standard Edition 5.0
Java Language
Java Language
Development Tools & APIs
java
javac
javadoc
apt
jar
javap
JPDA
Other
Security
Int'l
RMI
IDL
Deploy
Monitoring
Trouble-shooting
JVM TI
Deployment Technologies
Deployment
Java Web Start
Java Plug-in
User Interface Toolkits
AWT
Swing
Java 2D™
Accessibility
Drag 'n Drop
Input Methods
Image I/O
Print Service
Sound
Integration Libraries
IDL
JDBC™
JNDI™
RMI
RMI-IIOP
Other Base Libraries
Beans
Int'l Support
I/O
New I/O
JMX
JNI
Math
Networking
Std. Override Mechanism
Security
Serialization
Extension Mechanism
XML JAXP
lang & util Base Libraries
Lang & Util
Collections
Concurrency Utilities
JAR
Logging
Management
Preferences
Ref Objects
Reflection
Regular Expressions
Versioning
Zip
Java Virtual Machine
Java Hotspot™ Client Compiler
Java Hotspot™ Server Compiler
Platforms
Solaris™
Windows
Linux
Other
JDK
JRE
J2SE API

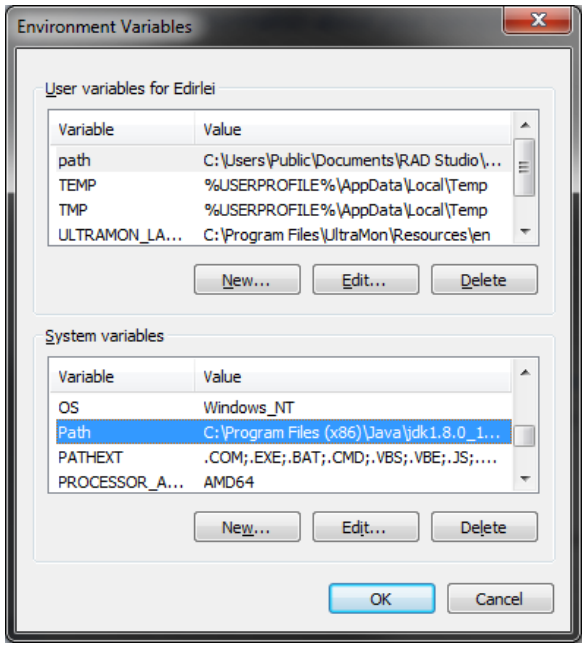
JDK & NetBeans - Download

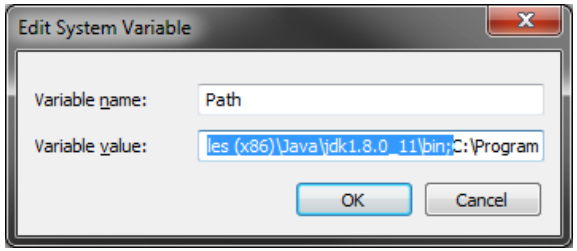
- http://www.oracle.com/technetwork/java/javase/downloads/index.html



Configuração - Windows

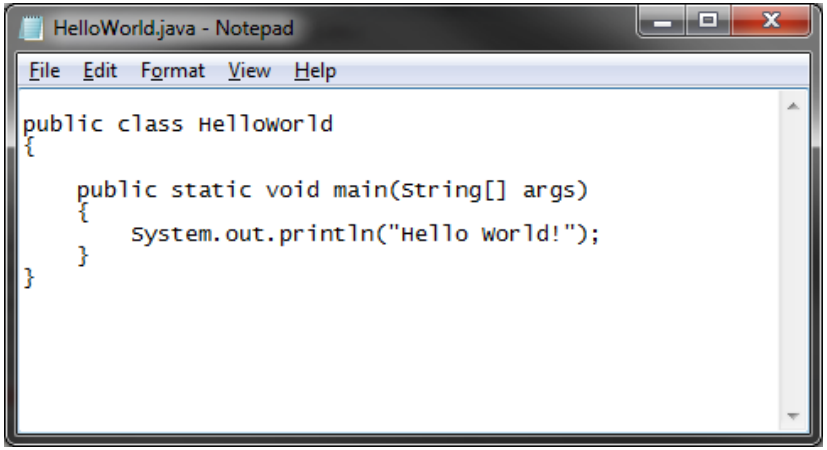
- Após a instalação, é importante adicionar o caminho para o diretório bin do JDK no path do sistema.
  - Control Panel -> System ->Advanced-> Environment Variables
  - Adicione o diretório “C:\Program Files (x86)\Java\jdk1.8.0_11\bin” a variável “Path” do sistema.





Hello World em Java

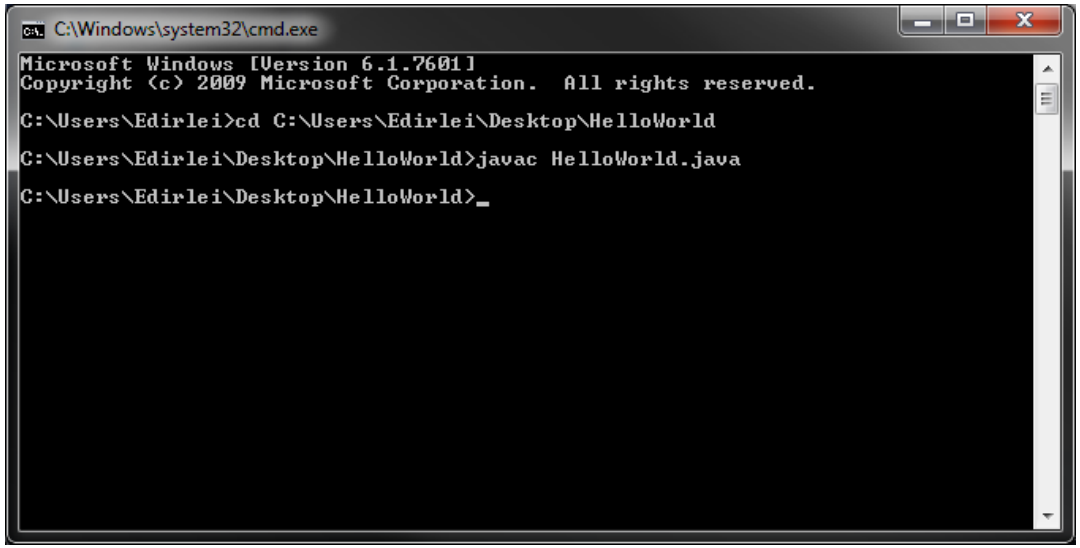
1) Crie no bloco de notas o seguinte programa:



2) Salve o programa como HelloWorld.java (lembre-se salva com a extensão .java).

Hello World em Java

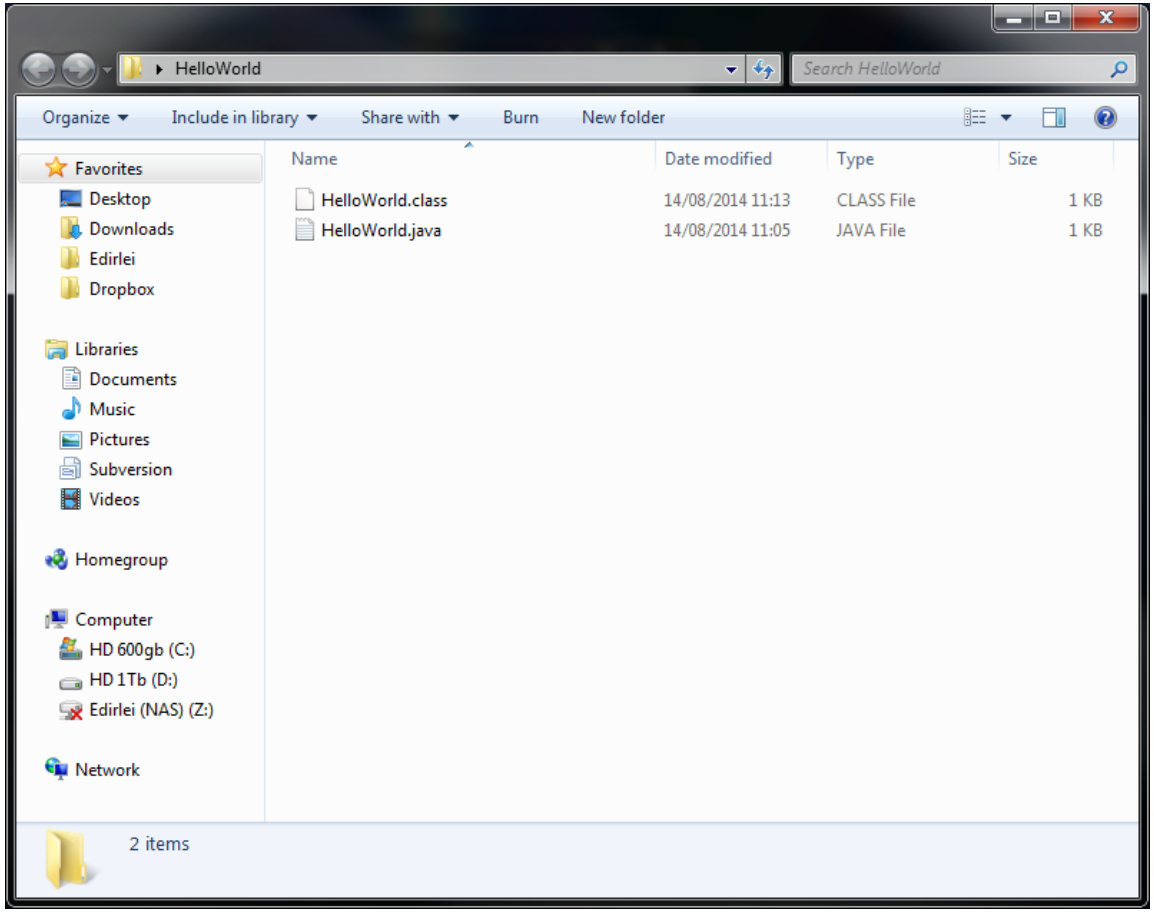
3) Usando a linha de comando do Windows, execute o seguinte comando no diretório onde o arquivo “HelloWorld.java” foi salvo: **javac HelloWorld.java**



Importante: se o Windows não reconhecer o comando javac, verifique se o diretório bin do JDK está incluído no path do sistema.

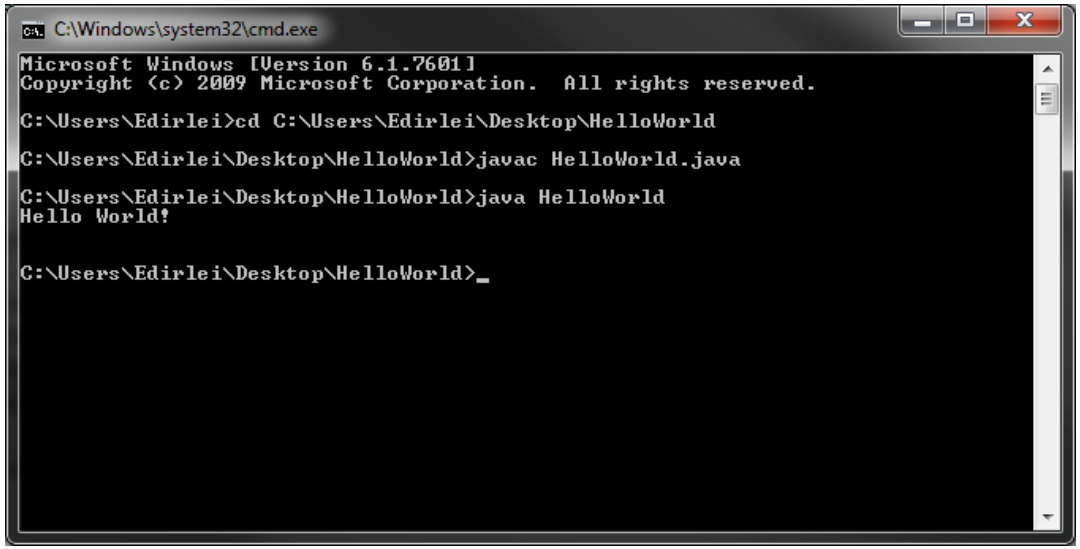
Hello World em Java

4) O arquivo “HelloWorld.class” será criado:



Hello World em Java

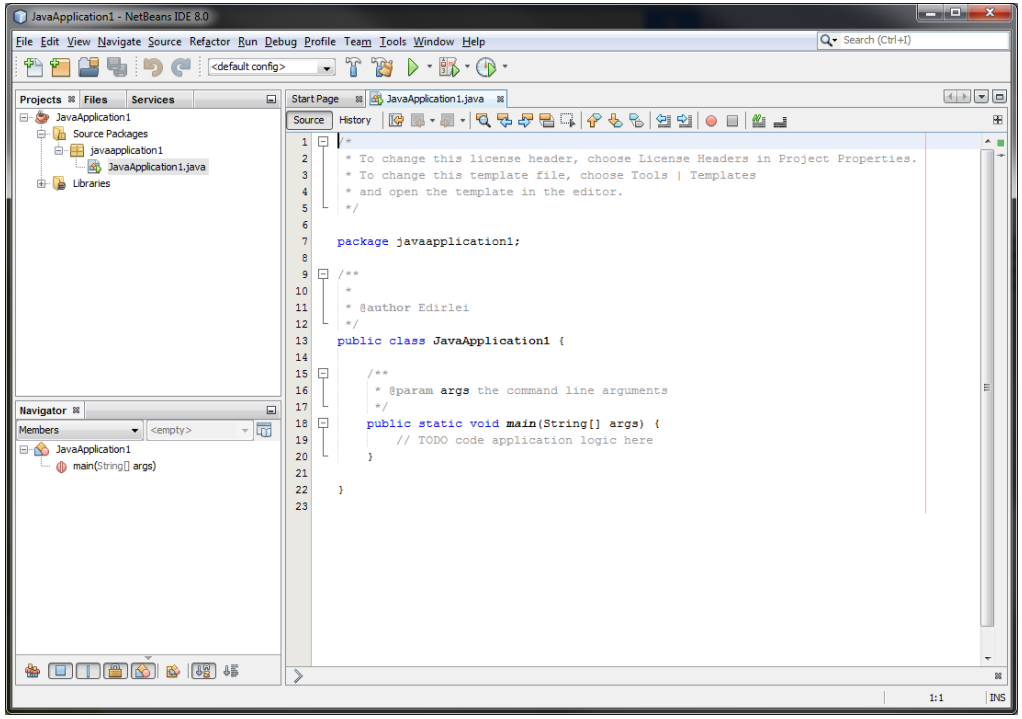
5) Para executar o programa Java, execute o seguinte comando: **java HelloWorld**



IDE Java

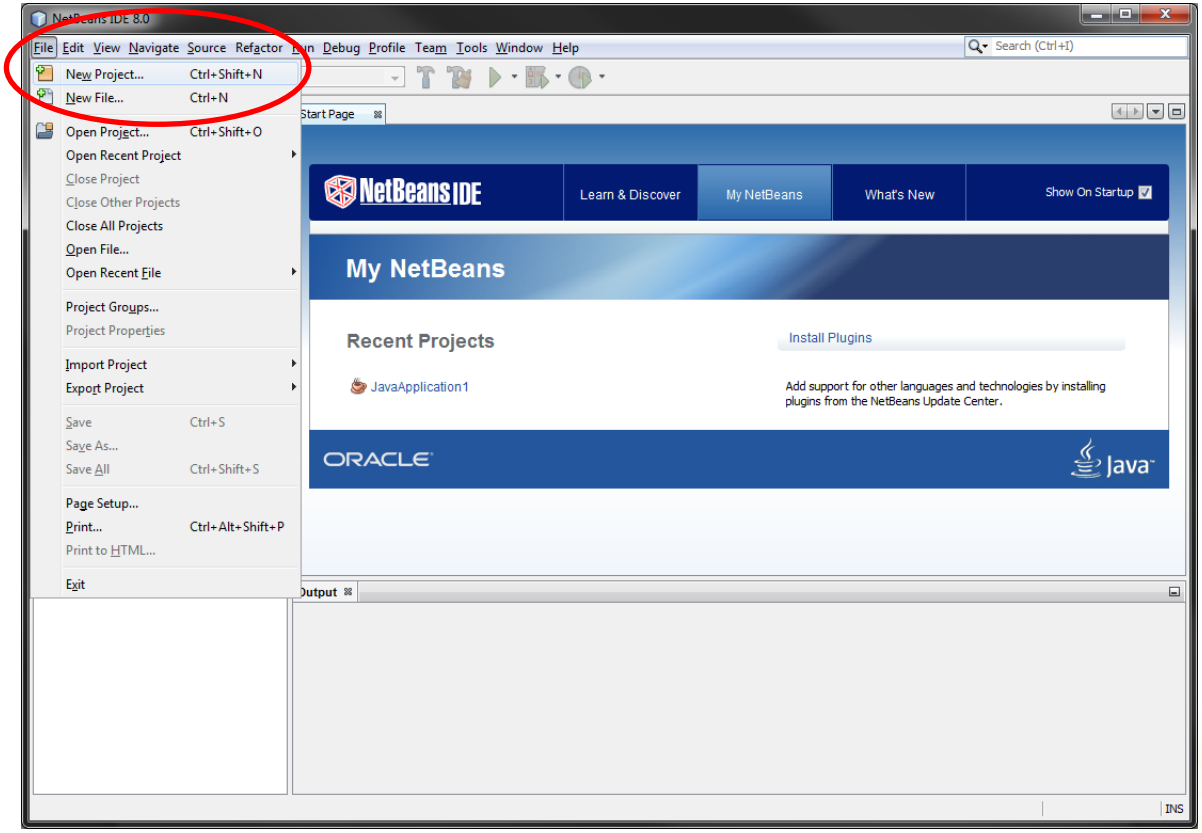
- **Integrated Development Environment (IDE)**

- Exemplos de IDE Java: **NetBeans**, Eclipse...



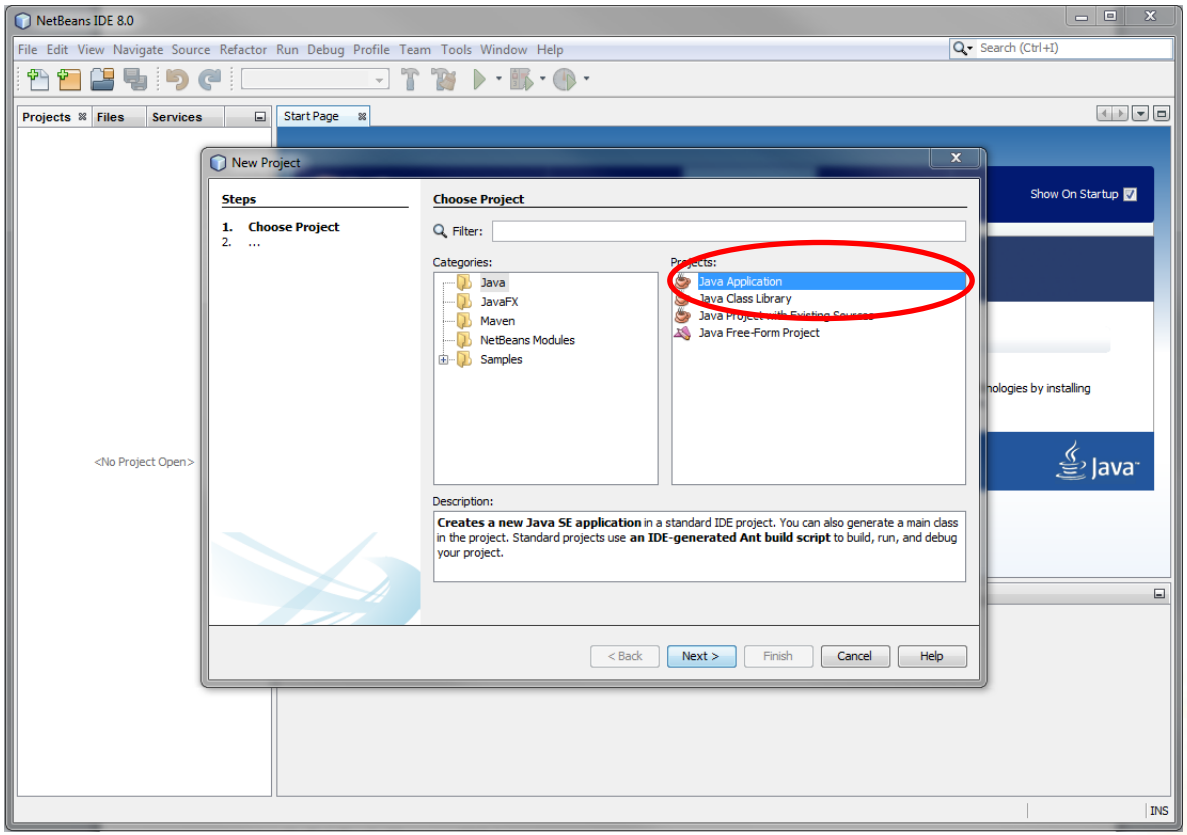
Criando um Projeto no NetBeans

1) Acesse o menu File->New Project...



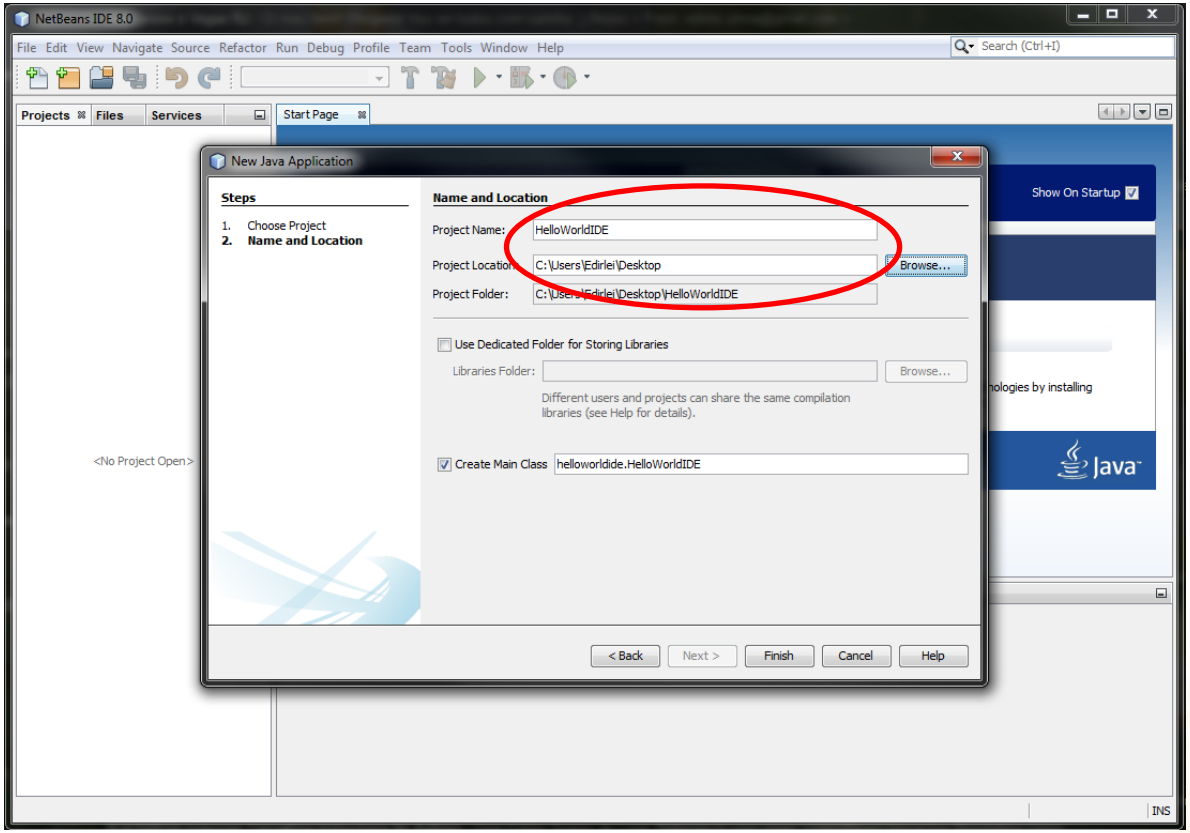
Criando um Projeto no NetBeans

2) Selecione o tipo de projeto “Java Application” e em seguida clique em “Next”:



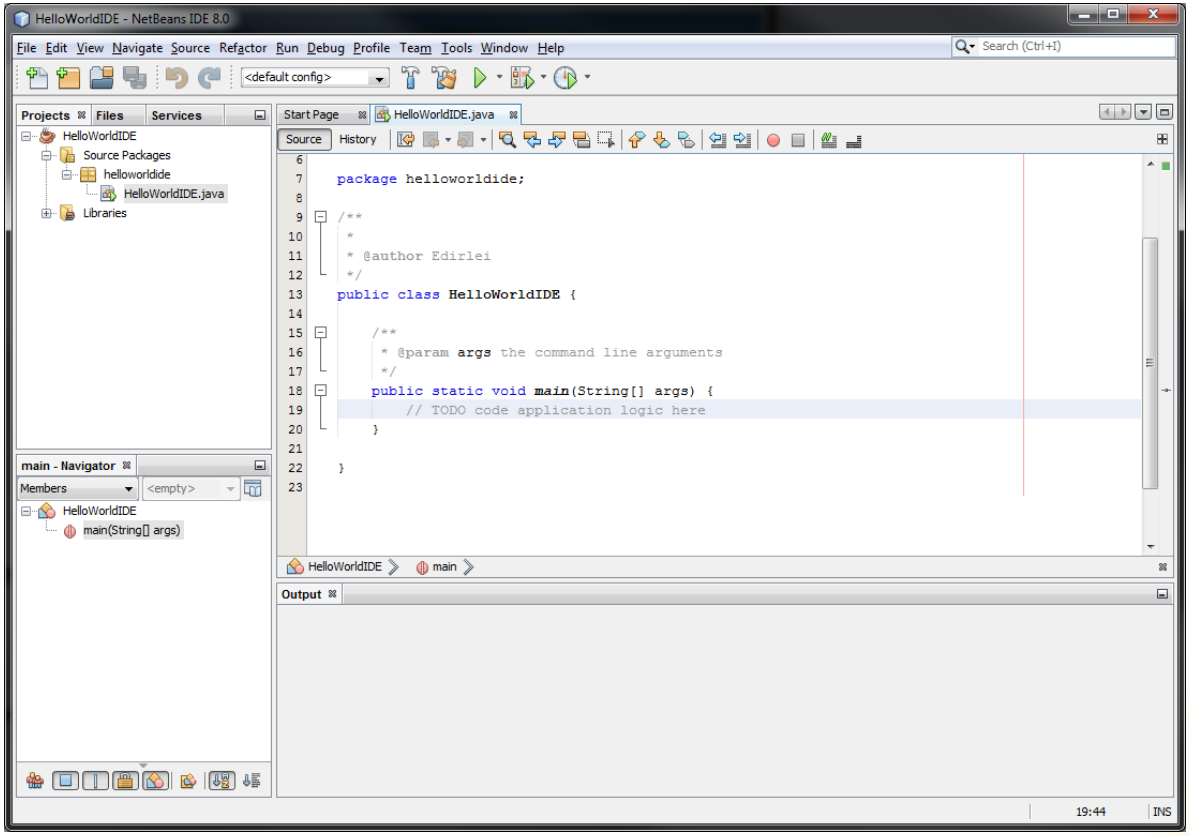
Criando um Projeto no NetBeans

3) De um nome para o projeto e selecione o local onde ele será salvo. Em seguida clique em “Finish”:



Criando um Projeto no NetBeans

4) O projeto e a classe principal do programa serão criados:



Estrutura Básica de Um Programa Java

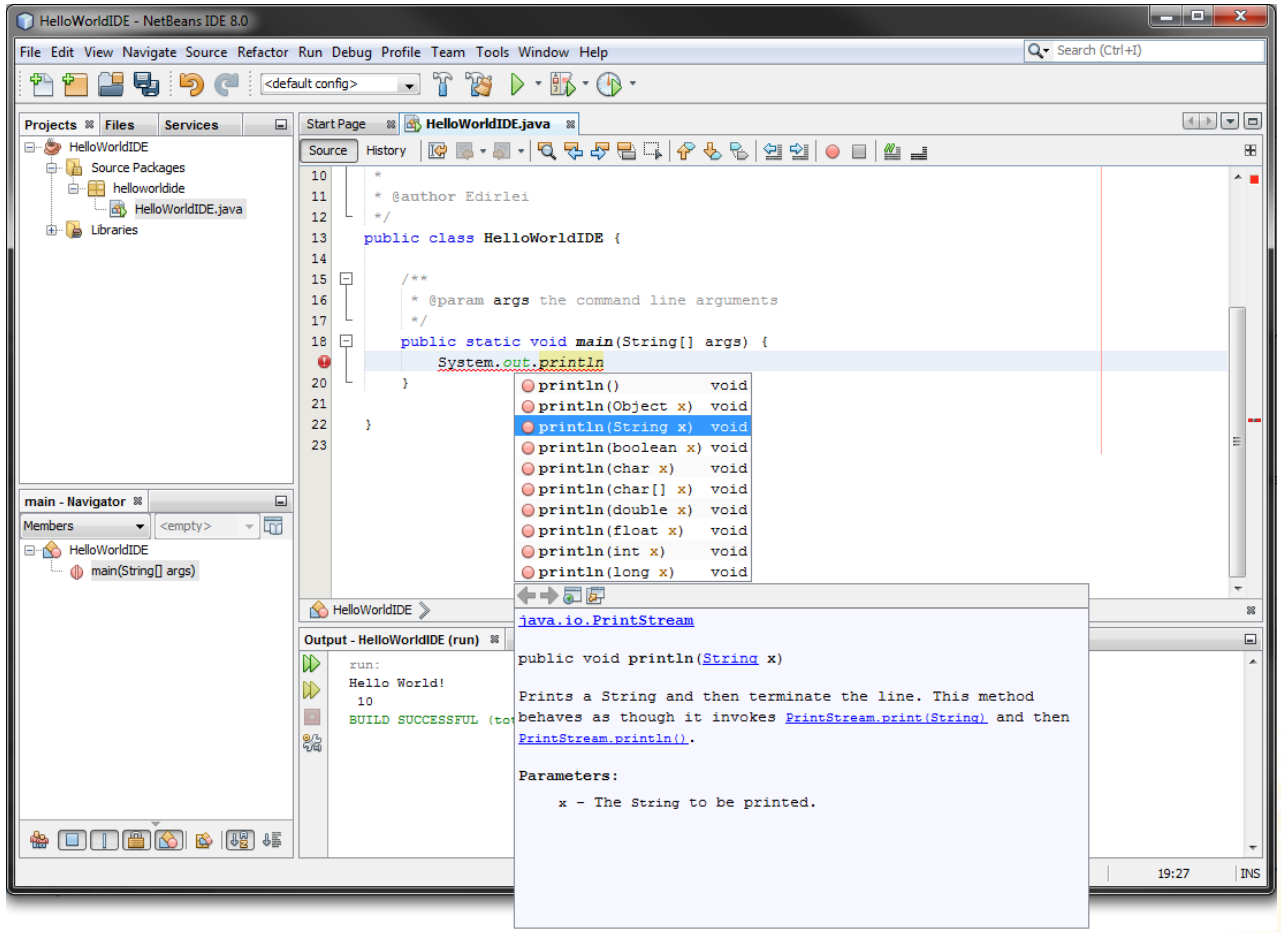
```
package helloworldide;

/**
 * @author Edirlei
 */
public class HelloWorldIDE
{
    /**
     * @param args the command line arguments
     */
     public static void main(String[] args)
     {
         // TODO code application logic here
     }

}
```

Autocomplete no NetBeans

- Para ativar pressione Ctrl+Espaço:

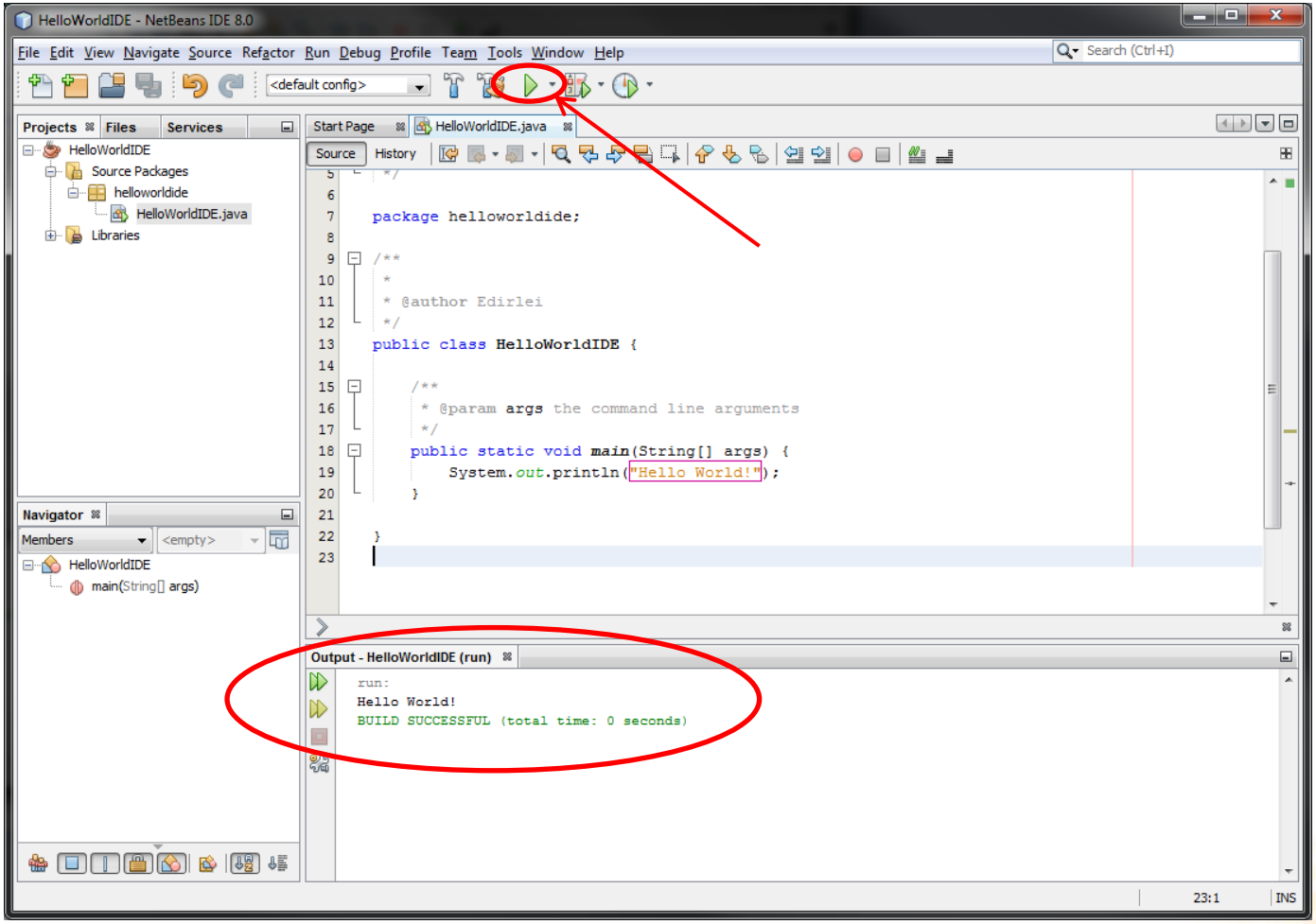


Hello World no NetBeans

```
package helloworldide;

/**
 *
 * @author Edirlei
 */
public class HelloWorldIDE
{
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Executando o Projeto no NetBeans



O Método `main`

- Toda aplicação (não toda classe!) deve possuir um método `public static void main(String[] args) {...}`

- A assinatura do método contém três modificadores:
  - public - pode ser invocado por qualquer objeto;
  - static - método de classe (pode ser acessado globalmente);
  - void - o método não retorna valor;

- Quando o interpretador executa a aplicação, começa por chamar o método `main`, que, por sua vez, chama os outros métodos existentes na aplicação.

Variáveis em Java

- **Variável** é um espaço reservado na memória do computador para armazenar um tipo de dado.

- Devem receber **nomes** para poderem ser referenciadas e modificadas quando necessário.

- Toda variável tem:
  - um nome
  - um tipo de dado
  - um valor

Variáveis em Java

- **Tipos Inteiros:**

Tipo	Descrição
byte	Pode assumir valores entre $-2^7 = -128$ e $2^7 = +128$.
short	Pode assumir valores entre -2^{15} e 2^{15}
int	Pode assumir valores entre -2^{31} e 2^{31}
long	Pode assumir valores entre -2^{63} e 2^{63}

- **Tipos Reais:**

Tipo	Descrição
float	• O menor valor positivo representável por esse tipo é 1.40239846e-46 e o maior é 3.40282347e+38. • 4 bytes de tamanho e 23 dígitos binários de precisão.
double	• O menor valor positivo representável é 4.94065645841246544e-324 e o maior é 1.7976931348623157e + 308. • 8 bytes de tamanho e 52 dígitos binários de precisão.

Declaração de Variáveis em Java

- A declaração de variáveis em Java segue um padrão semelhantes a linguagem C/C++

- Exemplos:

```
int a;      /* declara uma variável do tipo int */
float b;    /* declara uma variável do tipo float */
double c;   /* declara uma variável do tipo double */
int d, e;   /* declara duas variáveis do tipo int */
int f = 10;/* declaração e inicialização da variável */
```

Operadores e Expressões

- **Operadores aritméticos** são usados para se realizar operações aritméticas com as variáveis e constantes.

Operação	**Símbolo**
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto da Divisão	%

Exemplos:

operador de atribuição

```
total = preco * quantidade;
media = (nota2 + nota2)/2;
resultado = 3 * (1 - 2) + 4 * 2;
resto = 4 % 2;
```

Operadores e Expressões

- Operadores de atribuição:

= , += , -= , *= , /=

```
i += 2;              é equivalente a     i = i + 2;
x *= y + 1;          é equivalente a     x =  x * (y +1);
```

- Operadores de incremento e decremento:

++ , --

```
n++    incrementa n uma unidade
n--    decrementa n uma unidade
```

Funções Matemáticas

- **Classe Math:**
  - `double sqrt(double a);`
  - `double pow(double a, double b);`
  - `double cos(double a);`
  - `double sin(double a);`
  - `double log(double a);`
  - `int round(float a);`
  - `...`
- Exemplo:

```
double x = 64;
double r = Math.sqrt(x);
```

Lista das funções disponíveis na classe Math:
http://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

Conversão de Tipos e Type Casting

- A linguagem Java não suporta atribuições arbitrárias entre variáveis de tipos diferentes:

```
double x = 64.2;
int y = x;
```

incompatible types: possible lossy conversion from double to int

- Type Casting:

```
double x = 64.2;
int y = (int)x;
```

Entrada e Saída em Java

- A leitura e a escrita de dados via console é feita pelos objetos:
  - `System.in`
  - `System.out`

- Os principais métodos de `System.out` são:
  - `print(...)`

    Imprime o conteúdo de uma variável ou expressão

  - `println(...)`

    Imprime o conteúdo de uma variável ou expressão, e uma quebra de linha

Entrada e Saída em Java

- A forma mais simples para a leitura de dados do buffer `System.in` é utilizando a classe `java.util.Scanner`
- Para utilizar a classe `java.util.Scanner` é necessario instanciar um novo objeto do tipo `java.util.Scanner`:

```
int num1;
float num2;
Scanner entrada = new Scanner(System.in);

System.out.println("Digite um numero inteiro:");
num1 = entrada.nextInt();

System.out.println("Digite um numero real:");
num2 = entrada.nextFloat();
```

Exemplo 01

```
package exemplos;

import java.util.Scanner;

public class Exemplo01
{
  public static void main(String[] args)
  {
    int num1, num2, resultado;
    Scanner entrada = new Scanner(System.in);

    System.out.println("Digite o primeiro numero:");
    num1 = entrada.nextInt();
    System.out.println("Digite o segundo numero:");
    num2 = entrada.nextInt();

    resultado = num1 + num2;

    System.out.println("Resultado: " + resultado);
  }
}
```

Criando Funções em Java

```
tipo_de_retorno nome_da_funcao(parametros)
{
  variaveis locais

  instrucoes em java
}
```

Se uma função não tem retorno colocamos *void*.

Se uma função não tem uma lista de parâmetros colocamos apenas o ().

Consiste no bloco de comandos que compõem a função.

Exemplo 02

```
package exemplos;
import java.util.Scanner;

public class Exemplo02
{
    public static double celsius_fahrenheit(double tc)
    {
      double f;
      f = 1.8 * tc + 32;
      return f;
    }

    public static void main(String[] args)
    {
        double cels, fahr;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Digite a temperatura em celsius:");
        cels = entrada.nextDouble();
        fahr = celsius_fahrenheit(cels);
        System.out.println("Temperatura em Fahrenheit: " + fahr);
    }
}
```

Estruturas Condicionais em Java

- Em Java, uma estrutura condicional é construída através do comando ***if***:

```
if (expressao_logica)
{
   /* Bloco de comandos */
}
```

- Exemplo:

```
if (nota < 5.0)
{
   System.out.println("Reprovado");
}
```

Os comandos do **bloco de comandos** somente são executados se a **expressão lógica** for verdadeira

Estruturas Condicionais em Java

- Também é possível usar o comando **else** para executar algo quando a expressão lógica não é verdadeira:

```
if (expressao_logica)
{
   /* Bloco de comandos */
}
else
{
   /* Bloco de comandos */
}
```

Exemplo:

```
if (nota < 5.0)
{
  System.out.println("Reprovado");
}
else
{
  System.out.println("Aprovado");
}
```

Estruturas Condicionais em Java

- Também é possível criar sequencias de comandos **if-else** para a verificação exclusiva de varias condições:

```
if ( _condição_1_ )
{
   /* Bloco de comandos 1 */
}
else if ( _condição_2_ )
{
   /* Bloco de comandos 2 */
}
else if ( _condição_3_ )
{
   /* Bloco de comandos 3 */
}
```

Se a primeira condição resultar em *verdadeiro*, apenas o primeiro bloco de comandos é executado, e as outras condições não são sequer avaliadas.
Senão, se a segunda condição resultar em *verdadeiro*, apenas o segundo bloco de comandos é executado, e assim por diante.

Expressões Booleanas em Java

- Uma expressão booleana é construída através da utilização de **operadores relacionais:**

Descrição	Símbolo
Igual a	==
Diferente de	!=
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=

Exemplos:

X = 10 e Y = 5

Expressão	Resultado
(X == Y)	Falso
(X != Y)	Verdadeiro
(X > Y)	Verdadeiro
(X < Y)	Falso
(X >= Y)	Verdadeiro
(X <= Y)	Falso

Expressões Booleanas em Java

- Expressões booleanas também podem ser combinadas através de **operadores lógicos**.

Operador	Significado	Símbolo em Java
Conjunção	E	&&
Disjunção	OU	||
Negação	NÃO	!

Exemplos:

Expressão	Resultado
(X > 0) && (X == Y)	Falso
(X > 0) || (X == Y)	Verdadeiro
!(Y < 10)	Falso

X = 10
Y = 5

```
package exemplos;
import java.util.Scanner;

public class Exemplo03{
    public static void main(String[] args){
        float nota;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Digite a nota:");
        nota = entrada.nextFloat();
        if ((nota <= 10) && (nota >= 0))
        {
            if (nota >= 9.0)
                System.out.println("A");
            else if (nota >= 8.0)
                System.out.println("B");
            else if (nota >= 7.0)
                System.out.println("C");
            else if (nota >= 5.0)
                System.out.println("D");
            else
                System.out.println("F");
        }
        else
            System.out.println("Nota Invalida!");
    }
}
```

```
package exemplos;
import java.util.Scanner;
public class Exemplo04{
    public static void main(String[] args){
        int mes;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Digite o mes:");
        mes = entrada.nextInt();
        switch (mes){
            case 1: System.out.println("Janeiro");
                    break;
            case 2: System.out.println("Fevereiro");
                    break;
            case 3: System.out.println("Marco");
                    break;
            case 4: System.out.println("Abril");
                    break;
            case 5: System.out.println("Maio");
                    break;
            default:
                    System.out.println("Outro");
                    break;
        }
    }
}
```

Estruturas de Repetição em Java

- Em Java, uma das formas de se trabalhar com repetições é através do comando **while**:

```
...
while(expressao_logica)
{
   /* Bloco de comandos */
}
...
```

Enquanto a "**expressão_lógica**" for verdadeira, o "bloco de comandos" é executado!

Depois, a execução procede nos comandos subsequentes ao bloco while.

- Exemplo:

```
int x = 0;
while (x <= 100)
{
  System.out.println(x);
  x = x + 1;
}
```

Estruturas de Repetição em Java

- Outra forma de se trabalhar com repetições em Java é através do comando **for**:

```
...
for(expressão_inicial; expressão_lógica; expressão_atualização)
{
   /* Bloco de comandos */
}
...
```

- Exemplo:

```
int x;

for (x = 0; x <= 100; x++)
{
  System.out.println(x);
}
```

Estruturas de Repetição em Java

- A linguagem Java oferece uma terceira construção de laços através do comando **do-while**:

```
...
do
{
   /* Bloco de comandos */
}while(expressão_lógica);
...
```

- Exemplo:

```
int x = 0;

do{
  System.out.println(x);
  x++;
}while (x <= 100);
```

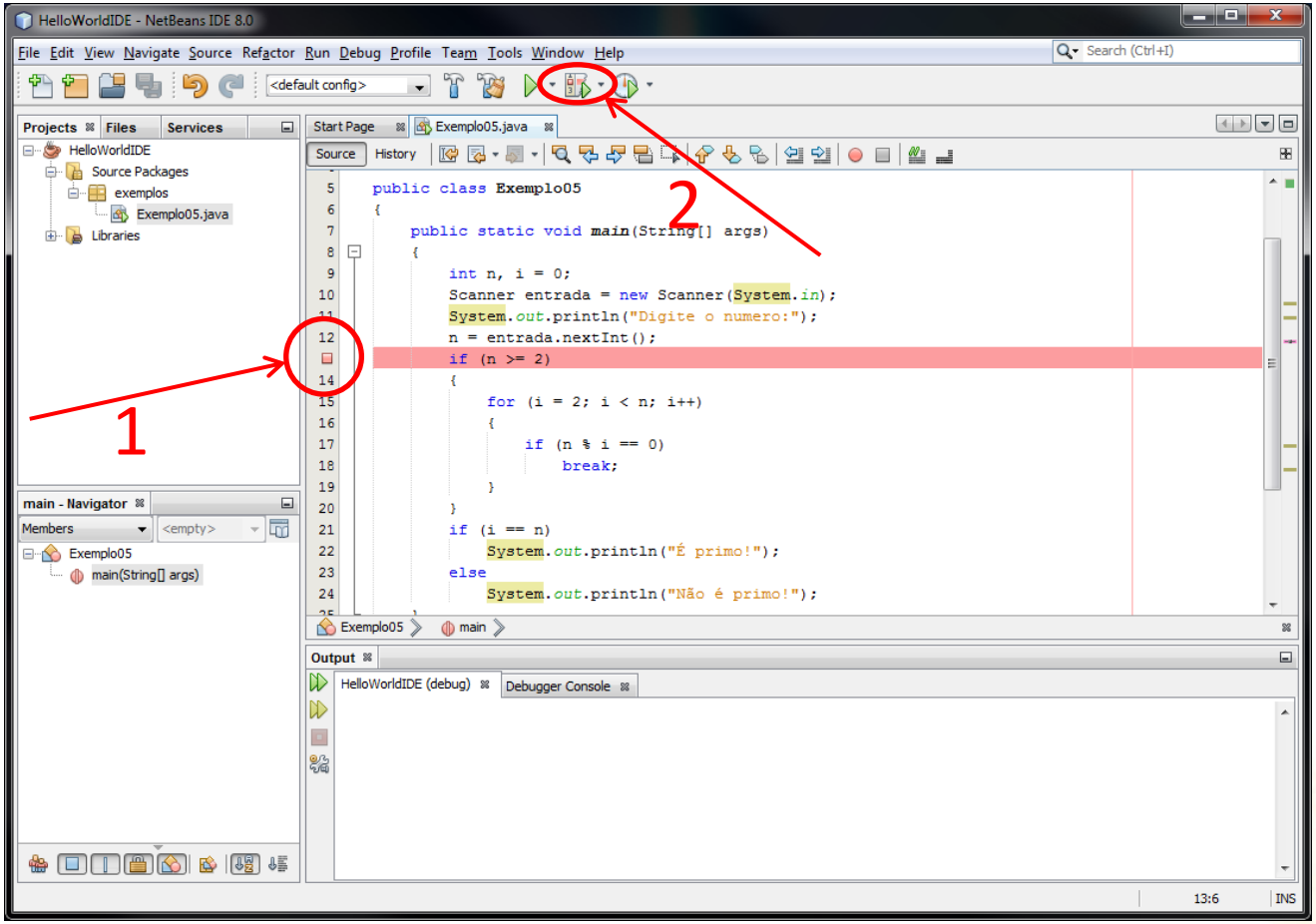
```
package exemplos;
import java.util.Scanner;

public class Exemplo05
{
    public static void main(String[] args)
    {
        int n, i = 0;
        Scanner entrada = new Scanner(System.in);

        System.out.println("Digite o numero:");
        n = entrada.nextInt();
        if (n >= 2)
        {
            for (i = 2; i < n; i++)
            {
                if (n % i == 0)
                    break;
            }
        }
        if (i == n)
            System.out.println("É primo!");
        else
            System.out.println("Não é primo!");
    }
}
```

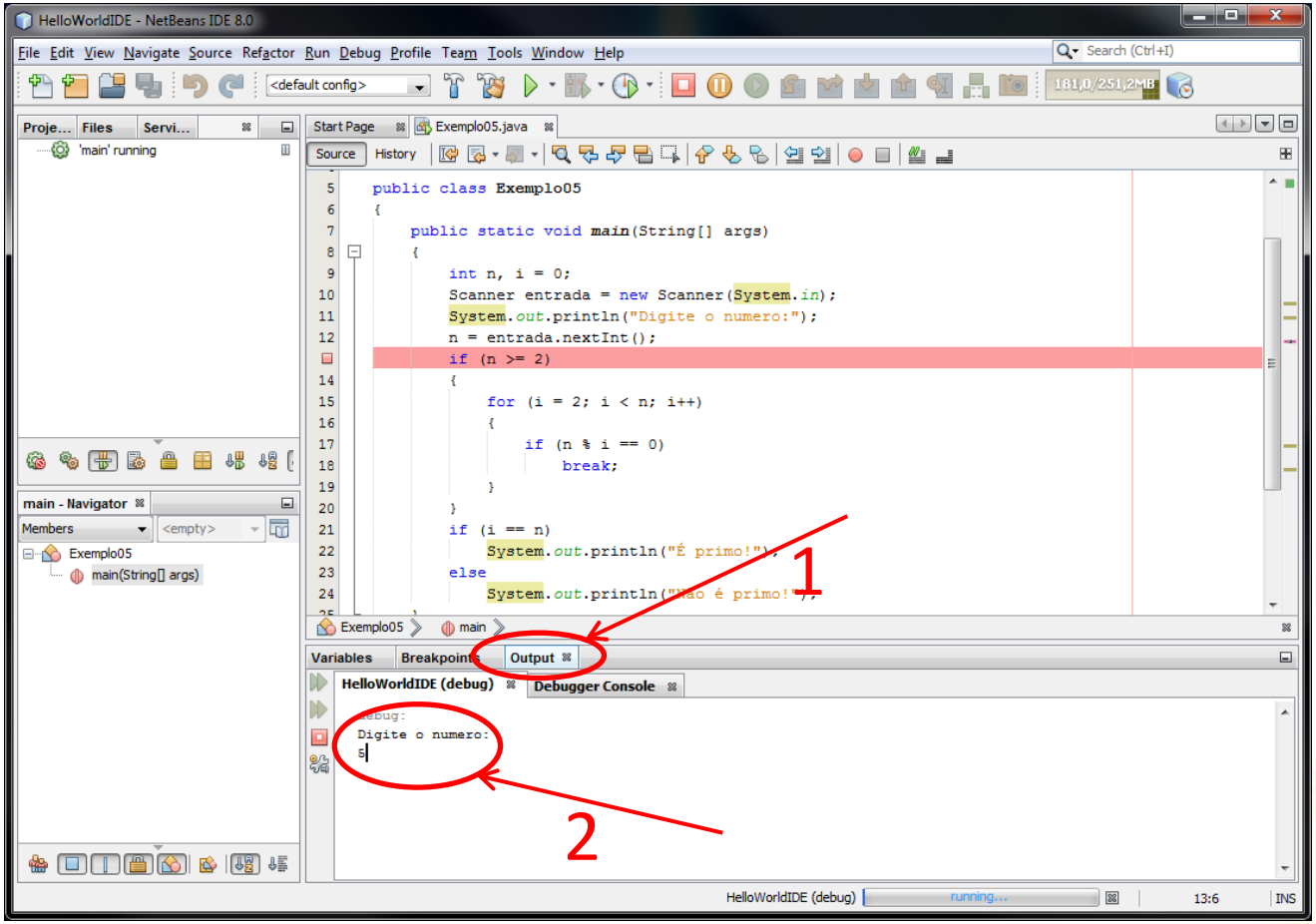
Depuração no NetBeans

1) Defina um breakpoint e inicie a depuração:



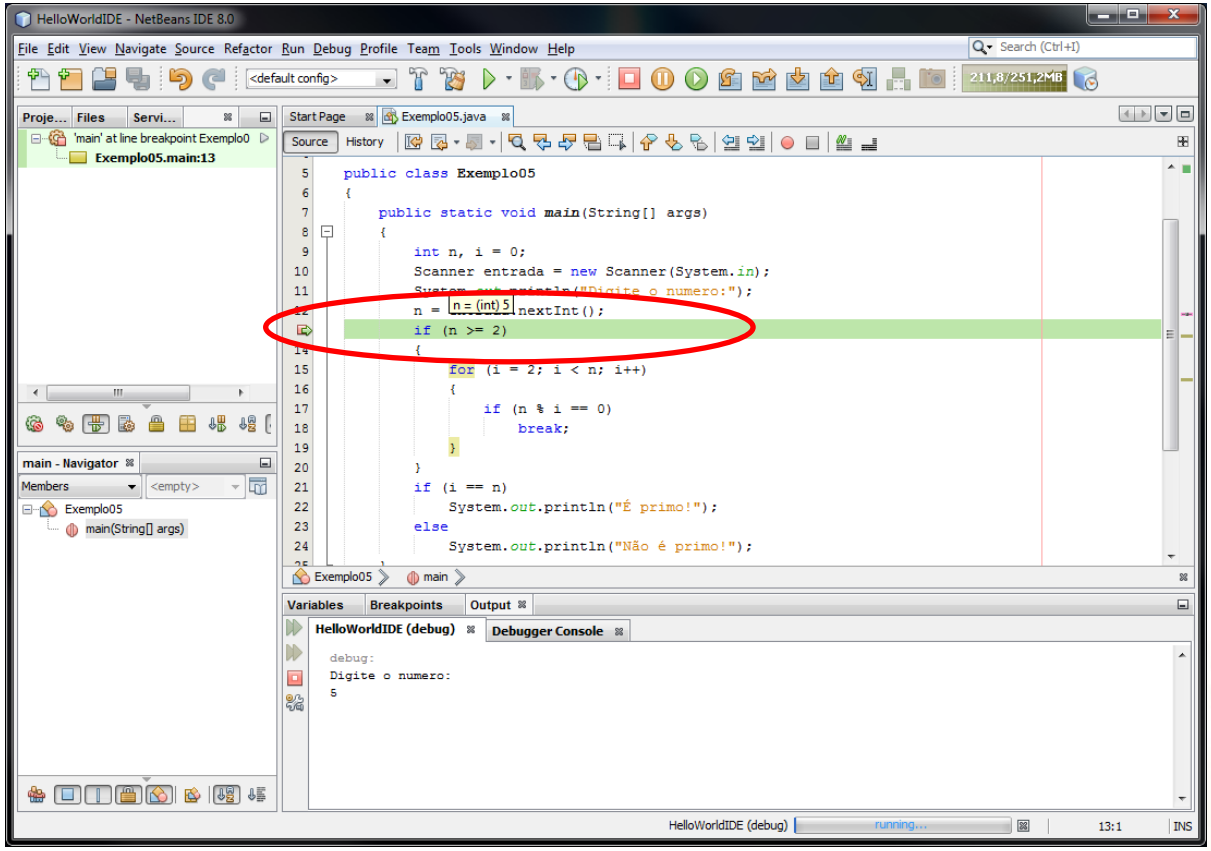
Depuração no NetBeans

2) Clique na aba output e digite um número:



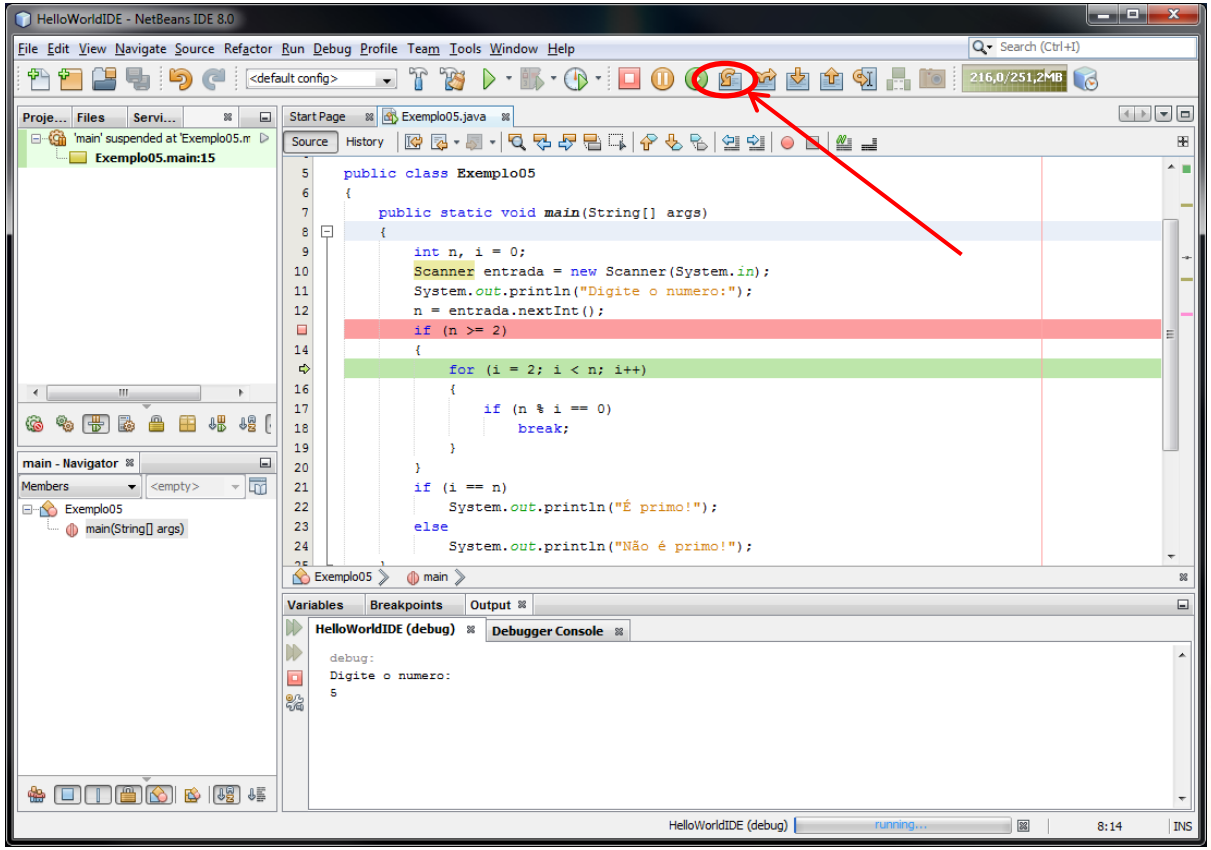
Depuração no NetBeans

3) A execução para ao chegar no breakpoint. Coloque o mouse sobre as variáveis para avaliar o seu valor:



Depuração no NetBeans

4) Clique em “Step Over” para avançar para as próximas linhas e acompanhar a execução do programa:



Exercícios

Lista de Exercícios 01 – Introdução Java

http://uniriodb2.uniriotec.br