BandTec
DIGITAL SCHOOL

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

Linguagem de Programação

Programação em Java - Nivelamento

Sumário

IDE e ambiente de desenvolvimento Java

A criação de sistemas em Java exige o esclarecimento sobre alguns conceitos que causam certa confusão. A seguir, os conceitos básicos sobre Java.

Java: plataforma de desenvolvimento

O termo “Java” em computação pode ter vários entendimentos. Por exemplo, para usuários que não trabalham com TI, como um contador, uma enfermeira ou um professor de história: se perguntar “o que é Java?”, provavelmente vão falar “é aquilo que instalo para poder usar meu internet banking” ou “é aquilo que instalo para poder usar o programa da declaração de imposto de renda”.

Para um profissional de desenvolvimento de software que nunca trabalhou com Java, a resposta provavelmente seria “é uma linguagem de programação”. A verdade é que Java é mais que um programa ou uma linguagem de programação: É uma **plataforma de desenvolvimento**, ou seja, é um conjunto de ferramentas e bibliotecas que permitem a criação e/ou execução de programas. É possível ver como essa plataforma é composta na Figura 1.

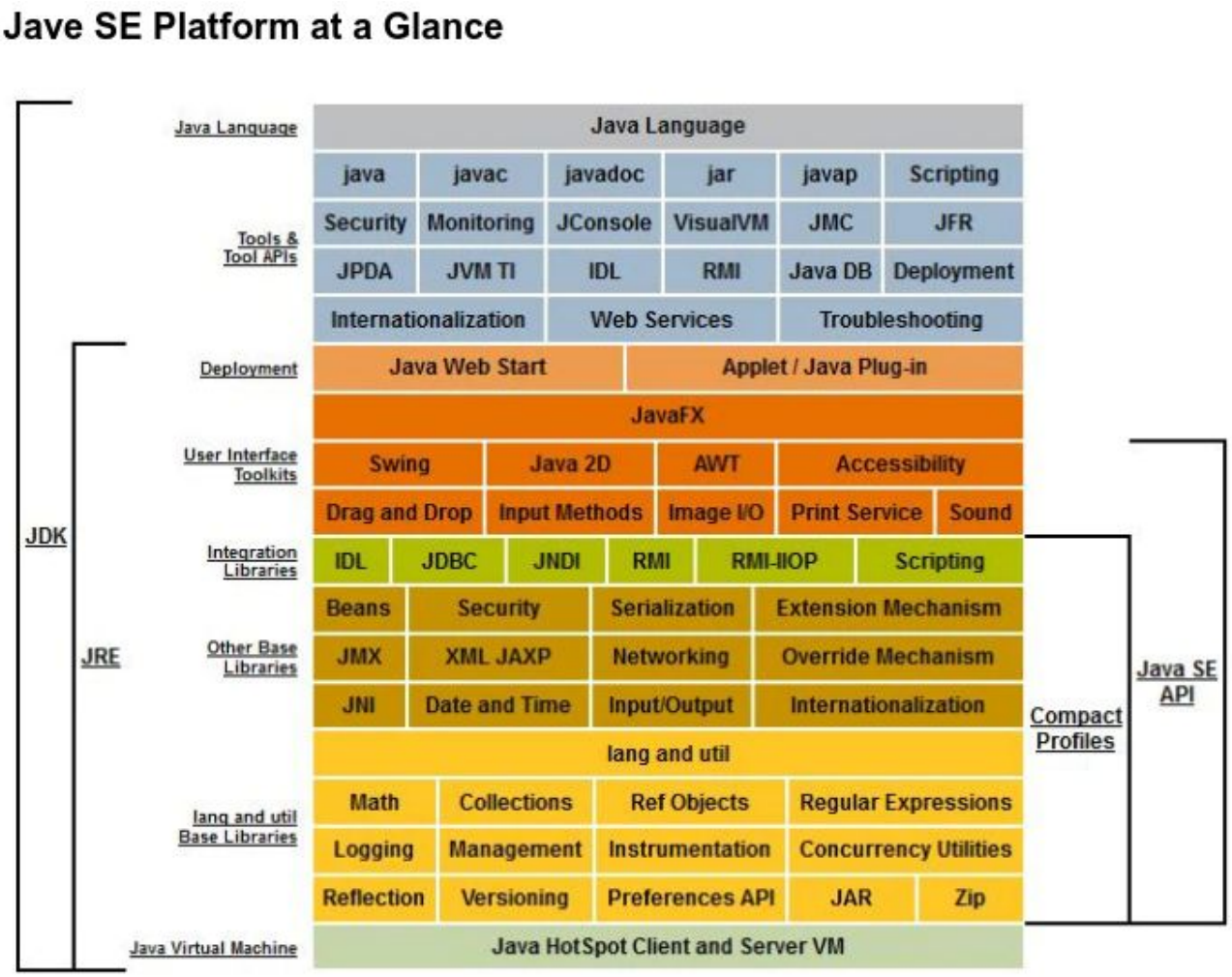

Figura 1: Plataforma de desenvolvimento Java
Fonte: https://www.oracle.com/technetwork/java/javase/tech/index.html

A plataforma Java é de código aberto, porém é de propriedade da **Oracle Corporation** desde 2010, quando ela comprou a **Sun Microsystems**, empresa que criou e popularizou a tecnologia Java.

JRE - Java Runtime Environment

O conjunto de ferramentas e bibliotecas que permite que um programa criado para a plataforma Java seja executado é a **JRE - Java Runtime Environment** (“Ambiente de tempo de execução Java” em tradução livre). É isso que, na verdade, um usuário que não trabalha com desenvolvimento de software instala quando diz “instalei o Java no meu computador”. Exemplos práticos: O programa de declaração anual do imposto de renda também é um programa para a plataforma Java, por isso é necessário ter a JRE instalada no computador para poder executá-lo; A maioria dos internet banking precisam da JRE, pois usam um programa desenvolvido para a plataforma Java e que precisa ser executado nos navegadores.

JVM - Java Virtual Machine

A **JVM - Java Virtual Machine** (“Máquina Virtual Java” em tradução livre), é como uma máquina virtual dessas que usamos para iniciar um sistema operacional em outro. Pense como se

houvesse um sistema operacional chamado “Java” que inicia em poucos segundos. Ela faz parte da **JRE** e sempre é iniciada para permitir a execução de um programa da plataforma Java. Existem diferentes versões de JVM, uma para cada um dos principais sistemas operacionais da atualidade.

Para entender o motivo de sua existência é preciso entender que os programas da plataforma Java **não são compilados para executáveis nativos**. Isso significa que não existem arquivos executáveis diretamente pelo sistema operacional criados na plataforma Java (por exemplo, arquivos **.exe** para Windows). Programas compilamos um programa criado para a plataforma Java, é gerado um pseudocódigo, um *“Byte Code”*, que é um arquivo com a extensão **.class**. Esses arquivos só podem ser executados por uma **JVM**. Esse funcionamento está ilustrado na Figura 2.

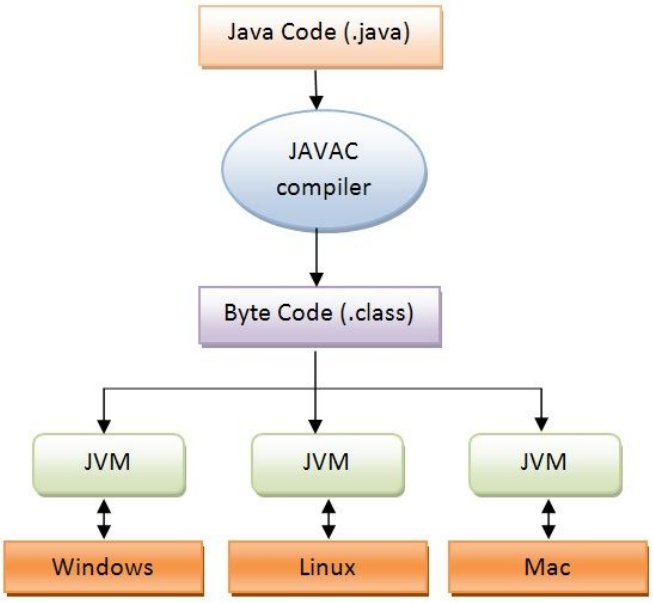

Figura 2: Funcionamento da JVM
Fonte: https://www.devmedia.com.br/introducao-ao-java-virtual-machine-jvm/27624

Por que isso? Porque assim, um mesmo programa escrito para a plataforma Java não precisa gerar executáveis para diferentes sistemas operacionais (afinal, mesmo sistemas de uma mesma família possuem suas peculiaridades). Basta gerar para o “sistema operacional” Java, que é iniciado rapidamente pela **JVM** instalada junto de uma **JRE**. Isso favorece os desenvolvedores, que não precisam se preocupar com detalhes dos diferentes sistemas operacionais. Quem se preocupa com isso é a Oracle, que cria várias versões diferentes da **JRE/JVM** para diferentes sistemas operacionais.

JDK - Java Development Kit

O conjunto de ferramentas e bibliotecas que permite que a criação de um programa para a plataforma Java é a **JDK - Java Development Kit** (“Kit para o desenvolvimento Java”, em tradução livre). Com isso instalado num computador, um desenvolvedor pode criar programas para usando apenas um editor de texto ou uma IDE. No momento da instalação da JDK, quase sempre é instalada uma JRE da mesma versão simultaneamente.

Java: Linguagem de programação

Desenvolvedores de software de outras linguagens normalmente pensam que Java é apenas uma linguagem de programação. Como acabamos de ver, Java também é o nome de uma plataforma de desenvolvimento. Porém, sim, existe a **linguagem de programação Java**.

O importante é saber que Java **não é a única linguagem de programação** para a plataforma Java. Lembra que falamos sobre a questão dos arquivos **.class** executáveis pela **JVM**? Isso significa que, teoricamente, qualquer linguagem pode ser usada para criar programas para a plataforma Java contanto que exista um compilador que, a partir gere um **.class** válido. As linguagens de programação mais populares para a plataforma Java, fora a Java, são: **Groovy**, **Kotlin**, **Scala** e **Clojure**, conforme o estudo anual "**State of Java**" (Vide Figura 3).

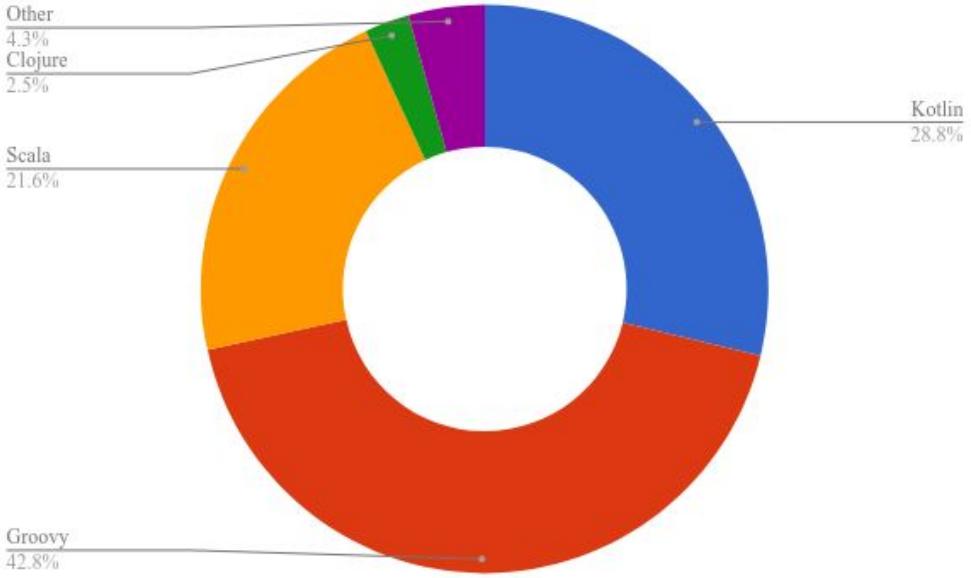

Figura 3: Adoção de outras linguagens da plataforma Java em 2018
Fonte: https://www.baeldung.com/java-in-2018

A linguagem **Groovy**, por exemplo, é de uso geral, porém muito usadas para scripts de automação e criação de testes automatizados (com o Spock framework, por exemplo). A **Kotlin** vem se tornando muito popular por estar sendo muito adotada na criação de aplicativos para Android.

IDE - Integrated Development Environment

Uma **IDE - Integrated Development Environment** ("Ambiente de desenvolvimento integrado" em tradução livre), é um programa ou conjunto de programas usados para programar. A vantagem no uso de **IDEs** é que abstraem alguns detalhes do uso de SDKs padrão e facilitam tarefas no dia-a-dia, como renomear um método que é usado em centenas de arquivos diferentes de um projeto ou ainda executar um programa em "*modo debug*", por exemplo.

Toda a **IDE** para a plataforma Java precisa de uma **JDK** instalada, embora algumas tenham a opção de usarem uma **JDK embarcada**.

Não existe uma IDE oficial para a plataforma Java. Aliás, isso é um item bem democrático da plataforma, que possui 3 principais IDEs: **IntelliJ**, **Eclipse** e **Netbeans**. A **IntelliJ** é a única das 3 que é proprietária, porém possui uma versão 100% gratuita.

Ambiente de desenvolvimento com a IDE NetBeans

Lembrando que você deve, antes de executar o NetBeans, ter uma **JDK** instalada. Em nosso curso, usaremos a versão mais recente da **JDK família 8**. Para Windows, seu instalador pode ser encontrado em **https://www.oracle.com/technetwork/es/java/javase/downloads/jdk8-downloads-2133151.html**. Para Linux e Mac é melhor usar a versão disponível no pacotes de cada distribuição/versão.

Em nosso curso, a IDE usada será o **NetBeans 11**, que pode ser obtido em **http://ftp.unicamp.br/pub/apache/incubator/netbeans/incubating-netbeans/incubating-11.0/incubating-netbeans-11.0-bin.zip**.

Ela não possui instalador. Você baixa um arquivo compactado, descompacta ele onde preferir e executa o arquivo "**bin/netbeans**" (no caso de SO Linux ou Mac) ou o "**bin/netbeans64.exe**" (no caso de SO Windows). Claro, é preferível criar um atalho para esse executável.

Criando Projetos Java no NetBeans

Na plataforma Java, trabalhamos com o conceito de **Projeto** e não de arquivo. Um programa, por mais simples que seja, deve ser criado a partir de um **Projeto** e não de um arquivo Java. Um **Projeto Java** é um diretório com vários arquivos com a extensão **.java** (que são as **classes**) e arquivos de vários outros tipos que servem de apoio no projeto. A seguir, vejamos como criar um Projeto Java no NetBeans.

O primeiro passo é solicitar a criação de um novo Projeto. Claro que, para um desenvolvedor "raiz" é melhor usar o atalho de teclado, que é **Ctrl + Shift + N**. Porém, isso pode ser feito clicando-se no botão com uma pasta laranja no canto superior esquerdo, como mostra a Figura 4.

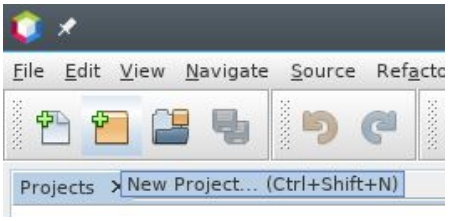

Figura 4: Botão laranja para criação novo Projeto no NetBeans

Outra opção é ir no menu **File -> New Project**, como mostra a Figura 5.

Figura 5: Menu para criação de novo Projeto no NetBeans

Em seguida, inicia-se o assistente de criação de projetos. Como é possível ver na Figura 6, o NetBeans permite criar vários tipos de projetos. Porém, vamos de **Java** -> **Java Application**. Depois, clicamos em **Next**.

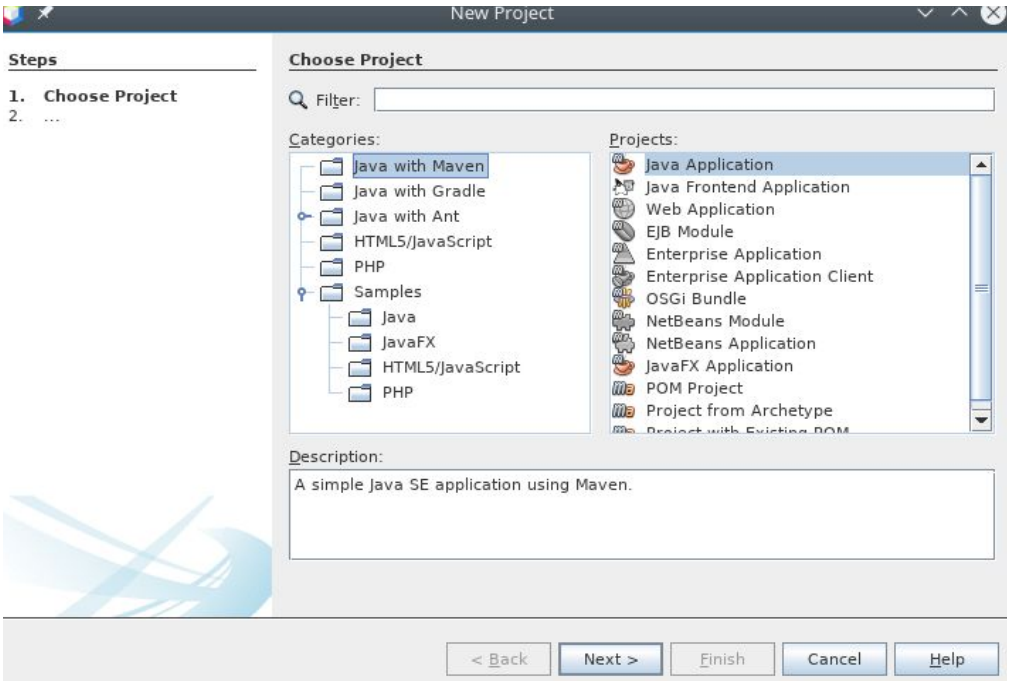

Figura 6: Assistente de criação de novo Projeto no NetBeans - primeiro passo

O próximo e último passo do assistente é definir o nome e local do projeto. Para os primeiros projetos, quando ainda está se aprendendo Java e NetBeans, é importante deixar marcada a opção ***Create Main Class***. Com tudo definido, podemos clicar em **Finish** a aguardar a criação do projeto. Vide Figura 7.

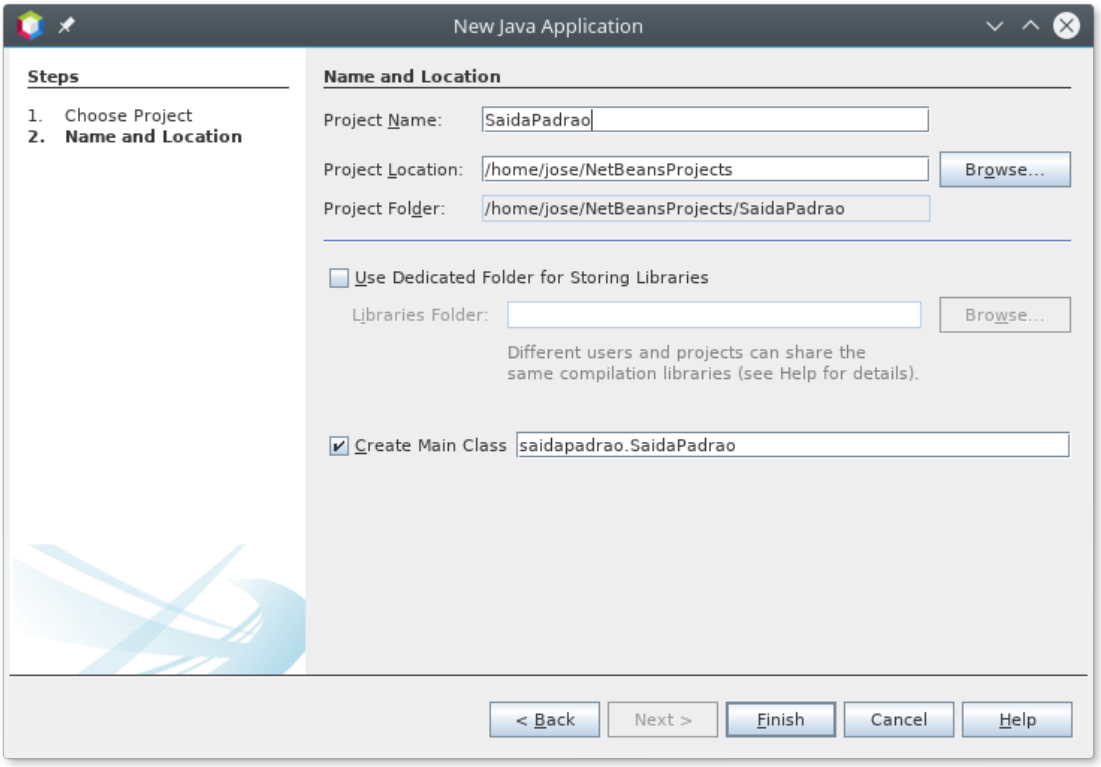

Figura 7: Assistente de criação de novo Projeto no NetBeans - Nome e local do projeto

Caso a criação do Projeto ocorra com sucesso, você verá sua estrutura num painel chamado ***Projects*** e o código fonte da classe executável aberto no editor no painel maior, à direita. Vide Figura 8.

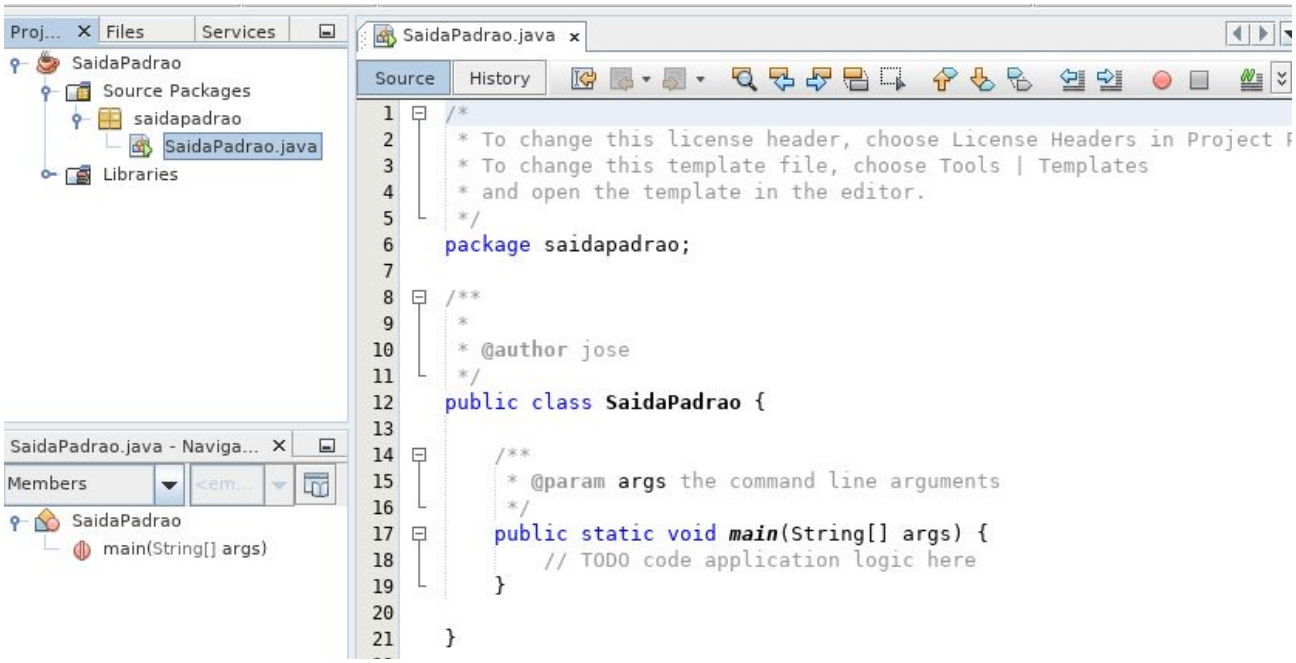

Figura 8: Projeto Java recém criado no NetBeans

Execução de um Projeto Java

Método Main

Na verdade, não é o Projeto Java que é executado e sim alguma de suas classes. Para que uma classe Java possa ser executada pela IDE e/ou pelo SO, ela precisa apenas ter o chamado **método main**. É nada mais que uma estrutura de código padronizada, como na Figura 9.

```
public static void main(String[] args) {
    // aqui vai o código que será executado
}
```

Figura 9: Método main válido numa classe Java.

Como já vimos anteriormente, o NetBeans já cria uma classe com esse **método main** em novos projetos. Porém, caso você precise transformar uma classe em executável, pode criá-lo manualmente. Não se desespere! Não precisa decorar todo esse código. Basta, dentro de uma classe, digitar **psvm** e teclar **TAB** e... *Voila*! O método main é criado pelo NetBeans!

Execução de uma classe com método main

Para executar uma classe executável, ou seja, que contém o método main, podemos usar o atalho de teclado **Shift + F6** ou, clicar com o botão direito do mouse em qualquer parte da classe e escolher a opção **Run File**.

Atenção!!! Existe um botão “Play” (uma seta verde) na parte superior do NetBeans. Ele não é confiável para a execução de sua classe. Evite a tentação de ficar executando sua classe por ele, ok?

Tipos de dados e variáveis em Java

Existem virtualmente infinitos tipos de dados em Java, afinal, quando você cria uma classe ela passa a ser um tipo válido em seu projeto. Porém, existem os tipos básicos que já estão presentes na JDK, são eles:

Tipos primitivos

São tipos que não são classes. Seus valores só funcionam **literalmente como valores**, pois não são tratados como objetos pelo Java. São eles:

- **short**: Número inteiro curto. Aceita valores de -2^{15} até 2^{15} (ou seja, de -32,768 a 32,768).
- **int**: Número inteiro. Aceita valores de -2^{31} até $2^{31}-1$.
- **long**: Número inteiro longo. Aceita valores de -2^{63} até $2^{63}-1$.
- **float**: Número real (ou número de ponto flutuante). Aceita valores de 2^{-149} até $(2-2^{-52})\cdot 2^{127}$. Detalhe: números float precisam da letra f ou F ao final. Ex: **8.5f** ou **12.3F**.
- **double**: Número real de dupla precisão. Aceita valores de 2^{-1074} até $(2-2^{-52})\cdot 2^{1023}$. É o tipo numérico real padrão do Java. Portanto, use double, sempre que possível.
- **byte**: Byte único. Aceita valores de -2^{7} até $2^{7}-1$.
- **boolean**: Boleana (ou lógica). Aceita apenas os valores **true** (verdadeiro) ou **false** (falso).
- **char**: Caractere único, delimitado com aspas simples. Exemplos: ‘a’; ‘b’; ‘w’.

Uma característica importante dos tipos primitivos, é que, como não são objetos, não podem ser nulos (**null**). Devem, obrigatoriamente, possuir um valor antes de serem usados. Por conta disso, todos possuem um **valor padrão**, que são:

- **short, int, long, float, double e byte**: 0 (zero).
- **boolean**: false.
- **char**: ‘\u0000’ (caractere unicode que significa “vazio”).

Casas decimais nos numéricos reais

Os tipos usados para números reais, caso precisem de casas decimais, estas devem ser separadas com **ponto (.)** e **não vírgula (,)**. Exemplos: “um e meio” é **1.5** e “oito ponto setenta e cinco” é **8.75**.

Peculiaridade de float

O tipo primitivo **float** possui a peculiaridade de exigir a letra **f** ou **F** logo após o valor numérico, caso seja indicado diretamente no código fonte. Por exemplo, para declarar uma variável chamada peso do tipo float com o valor 80,5, o código seria **float peso = 80.5f**. (ou **80.5F**).

Peculiaridade de long

Caso tenha o valor indicado diretamente no código fonte, uma número **long** pode vir a precisar da letra **l** ou **L** logo após o número. Essa necessidade é apenas caso o valor numérico exceda o máximo valor de **int**. Exemplos:

long idadeCivilizacao = 25000; (não precisa da letra ao final, pois 25000 pode ser **int**)

long bacterias = 9808974289L; (precisa da letra ao final, pois o valor é maior que **int**)

Classes Wrapper

Como vimos há pouco, os tipos primitivos: não são considerados objetos pelo Java e não podem ser nulos. Isso pode trazer algumas desvantagens em algumas situações, como conversão de tipos e possibilidade de não existir valor (ou seja, de aceitar **null**).

Para resolver essas desvantagens, é possível usar algumas classes da JDK que “imitam” os tipos primitivos, mas, por serem classes, geram objetos com métodos e atributos e permitem o valor nulo (**null**). São elas:

- **Short**: pode ser usada no lugar do primitivo **short**.
- **Integer**: pode ser usada no lugar do primitivo **int**.
- **Long**: pode ser usada no lugar do primitivo **long**.
- **Float**: pode ser usada no lugar do primitivo **float**.
- **Double**: pode ser usada no lugar do primitivo **double**.
- **Byte**: pode ser usada no lugar do primitivo **byte**.
- **Boolean**: pode ser usada no lugar do primitivo **boolean**.
- **Character**: pode ser usada no lugar do primitivo **char**.

Os Wrappers numéricos possuem os mesmos limites de valor de seus respectivos pares primitivos

O interessante é que se um método possui como argumento um tipo primitivo, as ser invocado você pode passar um objeto de um tipo Wrapper, contanto que não esteja nulo (**null**). E se você tiver um argumento definido como um tipo Wrapper, pode passar um tipo primitivo na invocação do método sem problemas.

Para entender de maneira prática a diferença entre primitivos e wrappers, basta pôr um ponto **(.)** após uma variável de qualquer um dos tipos e pressionar **CTRL+Espaço** no teclado. Após a variável primitiva o NetBeans não sugere nada, mas após a Wrapper, ela sugere os métodos e atributos públicos disponíveis (Vide Figura 10). Essa combinação de teclas, aliás, deve ser usada sem moderação. Ela faz abrir uma janela com tudo que você pode escrever naquele na posição do código em que está.

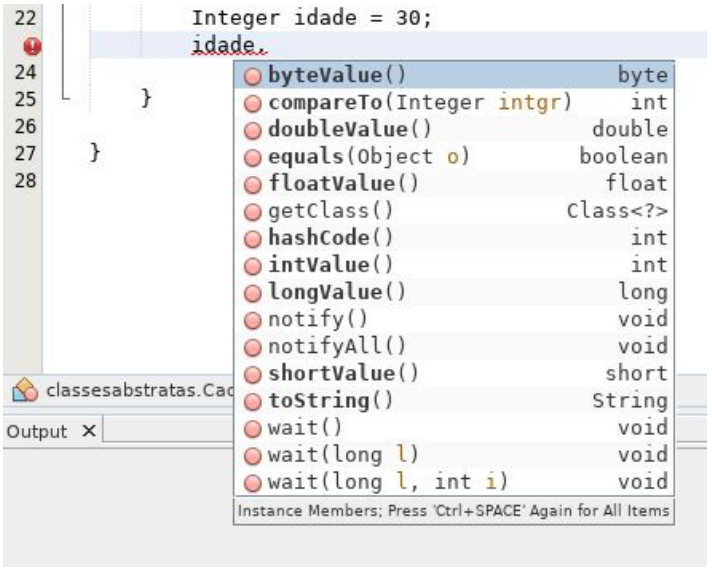

Figura 10: Sugestões de métodos públicos a partir de um objeto do tipo Integer

Na Figura 10 vemos o que acontece quando criamos uma variável do tipo Wrapper Integer e pedimos para o NetBeans sugerir alguma ação a partir dela.

Tipo String para alfanuméricos

Uma classe extremamente importante em Java é a **String**. Ela é usada para manipular textos, ou seja, valores alfanuméricos. Por exemplo, para criar uma variável String que armazene o nome de um bairro, o código seria:

String bairro = “Itaim bibi”;

Ou para armazenar um CEP:

String cep = “55055-000”;

Note que o valor **sempre** deve estar entre **aspas duplas**! Aspas simples não são aceitas para String em Java.

Como dito, String é uma classe, portanto uma variável String possui métodos públicos que podem ser usados usando-se um ponto (**.**) após a variável. A seguir, alguns dos principais métodos da String.

- **length()** - Retorna a quantidade de caracteres da String em um número inteiro (int)
- **toLowerCase()** - Retorna uma versão da String com todos os caracteres em caixa baixa (vulgo “minúsculo”). **Importante:** esse método **NÃO** altera a String original, apenas devolve para quem a invocou uma outra String.
- **toUpperCase()** - Retorna uma versão da String com todos os caracteres em caixa alta (vulgo “maiúsculo”). **Importante:** esse método **NÃO** altera a String original, apenas devolve para quem a invocou uma outra String.
- **equals(String outraString)** - Retorna um boolean **true** caso a String do argumento seja 100% idêntica (até nas maiúsculos e minúsculas) ou **false** em caso contrário.
- **equalsIgnoreCase(String outraString)** - Retorna um boolean **true** caso a String do argumento seja idêntica (mesmo que diferencie nos maiúsculos e minúsculas) ou **false** em caso contrário.

Praticando a String

Na classe executável do projeto, crie uma **String** com o nome e conteúdo que preferir (de preferência um conteúdo com mais de 5 letras, usando tanto letras maiúsculas quanto minúsculas). Use o **System.out.println(...)** para imprimir no **Console** os resultados dos métodos aqui explicados ou qualquer outro que queira experimentar. Exemplo:

```
String fruta = “Mamão Papaya”;

System.out.println(“Quantas letras: “ + fruta.length());
```

Conversão de tipos com os Wrappers

Outra vantagem dos Wrappers é que possuem métodos e construtores que permitem a conversão de tipos (converter um texto em número ou vice versa, por exemplo). São centenas, por isso vamos falar dos principais a seguir:

Conversão de texto em número

Vamos supor que você obteve um texto valor que espera que seja um número. Você precisa então convertê-lo para número para que possa realizar operações matemáticas com ele. Vejamos como fazer isso com os Wrappers numéricos por meio dos exemplos a seguir:

- **Short**

```
Short numero = new Short(“0”);
Short numero = Short.valueOf(“0”);
Short numero = Short.parseShort(“0”);
```

short numero = new Short(“0”);
short numero = Short.valueOf(“0”);
short numero = Short.parseShort(“0”);

- **Integer**:
  Integer numero = new Integer(“0”);
  Integer numero = Integer.valueOf(“0”);
  Integer numero = Integer.parseInt(“0”);
  int numero = new Integer(“0”);
  int numero = Integer.valueOf(“0”);
  int numero = Integer.parseInt(“0”);

- **Long**:
  Long numero = new Long(“0”);
  Long numero = Long.valueOf(“0”);
  Long numero = Long.parseLong(“0”);
  long numero = new Long(“0”);
  long numero = Long.valueOf(“0”);
  long numero = Long.parseLong(“0”);

- **Float**:
  Float numero = new Float(“0”);
  Float numero = Float.valueOf(“0”);
  Float numero = Float.parseFloat(“0”);
  float numero = new Float(“0”);f
  float numero = Float.valueOf(“0”);
  float numero = Float.parseFloat(“0”);

- **Double**:
  Double numero = new Double(“0”);
  Double numero = Double.valueOf(“0”);
  Double numero = Double.parseDouble(“0”);
  double numero = new Double(“0”);
  double numero = Double.valueOf(“0”);
  double numero = Double.parseDouble(“0”);

- **Byte**:
  Byte numero = new Byte(“0”);
  Byte numero = Byte.valueOf(“0”);
  Byte numero = Byte.parseByte(“0”);
  byte numero = new Byte(“0”);

```
byte numero = Byte.valueOf(“0”);
byte numero = Byte.parseByte(“0”);
```

Em todos os esses exemplos, o resultado é o mesmo: o texto, se contiver caracteres numéricos, é convertido para um número de verdade.

Obtendo um número de outro tipo

Notou que os tipos numéricos possuem como que uma “ordem de grandeza”? Onde o **byte/Byte** é o menor e o **double/Double** é o maior? Usando os Wrappers, a partir de qualquer um pode obter um dos outros tipos. Todo Wrapper de número possui os seguintes métodos:

- **byteValue()**: Obtém o **byte** possível a partir do objeto original. Por exemplo, um inteiro 0 retornaria o byte 0.
- **shortValue()**: Obtém o **short** possível a partir do objeto original. Por exemplo, um inteiro 80 retornaria o short 0.
- **intValue()**: Obtém o **int** possível a partir do objeto original. Por exemplo, um real 9.5 retornaria o inteiro 9.
- **longValue()**: Obtém o **long** possível a partir do objeto original. Por exemplo, um real 2020.99 retornaria o long 2020.
- **floatValue()**: Obtém o **float** possível a partir do objeto original. Por exemplo, um real 12.4 retornaria o float 12.4f.
- **doubleValue()**: Obtém o **double** possível a partir do objeto original. Por exemplo, um inteiro 99 retornaria o double 99.0.

Bibliografia

DEITEL, Paul J. Java como programar - 8ª edição. Campinas: Pearson, 2015.

ORACLE. Java™ Platform, Standard Edition 8 - API Specification. Disponível em <https://docs.oracle.com/javase/8/docs/api/>. Acesso em: 15 de novembro de 2018.