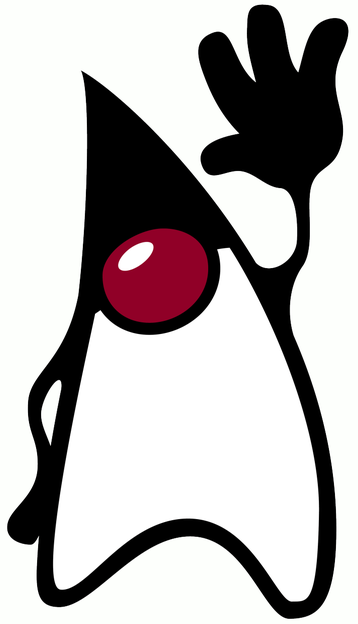
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Introdução ao Java

QXD0007 - Programação Orientada a Objetos

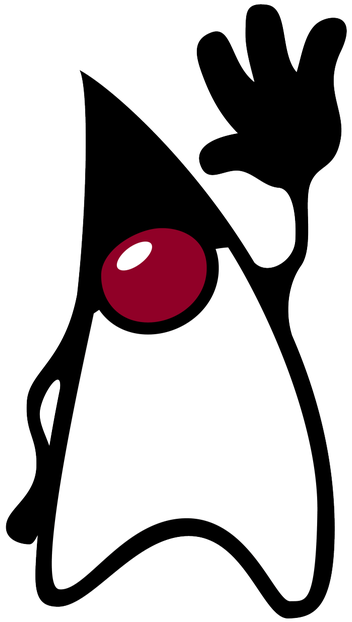


Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Conteúdo

- Introdução
- A Java Virtual Machine - JVM
- Princípios básicos
- Tipos de dados
- Classes e Objetos
- Declarando e usando variáveis
- Operadores
- Lendo e Imprimindo dados
- Estruturas de controle de fluxos

Introdução



Introdução

- A Sun criou um time (conhecido como Green Team) para desenvolver inovações tecnológicas em 1992
- Esse time foi liderado por James Gosling, considerado o pai do Java
- A ideia de criar um interpretador (já era uma máquina virtual, veremos o que é isso mais a frente) para dispositivos eletrônicos inteligentes
  - Facilitar a reescrita de software para aparelhos, como vídeo cassete, televisão e aparelhos de TV a cabo
- Decidiu-se criar um plataforma de desenvolvimento usando C++

Introdução

- James Gosling, coordenador do projeto, decidiu criar uma nova linguagem
  - Nasce então o projeto Oak
- O sucesso se deu ao seu uso no desenvolvimento web
  - Ao perceber o potencial do projeto para a web, a Sun Microsystems lançou o Java em 1995
  - Java ganhou popularidade rapidamente graças à sua portabilidade e capacidade de executar código em múltiplas plataformas

“Write Once, Run Anywhere”

Introdução

Versões

- Java 1.0 a 1.4 (1996-2002):
  - A primeira versão oficial, Java 1.0, que introduziu a biblioteca AWT (Abstract Window Toolkit) para construção de interfaces gráficas.
  - Versões subsequentes como Java 1.2 trouxeram a Swing API (para GUI), além de adicionar funcionalidades como collections
- Java 5 (2004):
  - Marcou uma grande atualização, adicionando recursos como Generics, Annotations, enum e varargs
- Java 6 (2006):
  - Focou na melhoria da JVM e da biblioteca padrão.

Introdução

Versões

- Java 7 (2011):
  - Trouxe novos recursos, como o uso de try-with-resources para gerenciamento automático de recursos e a API ForkJoinPool para programação paralela
- Java 8 (2014):
  - Um marco no desenvolvimento do Java, introduzindo Lambda Expressions e o Stream API, que facilitaram a programação funcional e o processamento de dados.
  - A nova API de Data e Hora substituiu a antiga Date e Calendar, oferecendo maior precisão e facilidade de uso.
- Java 9 (2017):
  - Adicionou o JShell (um REPL), facilitando a experimentação com o código

Introdução

Versões

- Java 10 a 16 (2018-2021):
  - Trouxeram novos recursos, como a inferência de tipo local (var) em Java 10 e a sintaxe aprimorada do switch em Java 12
- Java 17 (2021):
  - Versão LTS (Long-Term Support) que consolidou muitas funcionalidades introduzidas em versões anteriores
  - Melhorias na sintaxe, como sealed classes e pattern matching com switch
- Java 21 (2023):
  - Introduziu novos recursos de otimização de desempenho e segurança, além de funcionalidades experimentais.

Introdução

Características

- Orientação a Objetos
  - Atualmente é paradigma mais utilizado para o desenvolvimento de software
- Portabilidade
  - Permite a portabilidade de código entre diferentes sistemas operacionais.
  - Altamente versátil para desenvolvimento multiplataforma
- Robustez e Gerenciamento de Memória Automático
  - Garbage Collector, que automatiza a limpeza de objetos não referenciados, minimizando vazamentos de memória e falhas por alocação incorreta

Introdução

Características

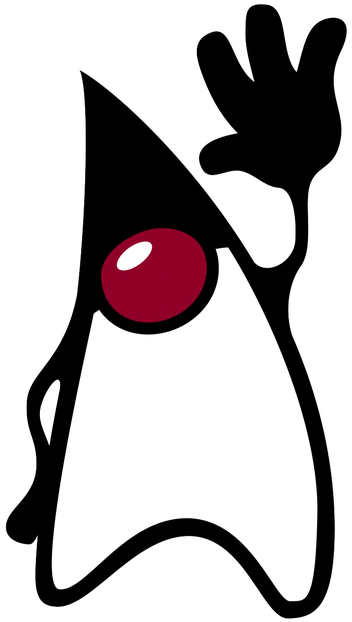
- Segurança
  - Oferece um ambiente seguro para execução de código, com proteções contra código malicioso
  - A JVM inclui uma série de verificações de segurança antes da execução do bytecode
- Multithreading e Concorrrência
  - A API de threads que permite que várias partes de um programa rodem simultaneamente
  - Essencial para aplicativos modernos que precisam executar várias tarefas ao mesmo tempo
- Sintaxe Similar a C/C++
  - Facilitando a aprendizagem para quem já tem experiência com essas linguagens

Introdução

Características

- Desempenho (JIT Compilation)
  - O uso do JIT (Just-In-Time) Compiler permite que a JVM otimize o bytecode em tempo de execução, aproximando seu desempenho ao de linguagens compiladas
- Grande Biblioteca Padrão (Java Standard Library)
  - A biblioteca padrão do Java, inclui um vasto conjunto de classes prontas para uso, desde manipulação de strings e coleções de dados até APIs de rede, entrada/saída, e manipulação de datas e horas
- Comunidade Ativa
  - Possui uma grande comunidade de desenvolvedores, documentação abundante e um ciclo de atualizações consistente

A Java Virtual Machine - JVM



A Java Virtual Machine - JVM

- A JVM é o núcleo da portabilidade e do desempenho do Java
- Funciona como um ambiente de execução que permite que o código Java seja executado em qualquer sistema operacional
- Para entendermos o que é essa máquina virtual e como ela funciona é importante entender a diferença entre:
  - Linguagens compiladas
  - Linguagens interpretadas
  - Linguagens híbridas

A Java Virtual Machine - JVM

Linguagens compiladas

- O código-fonte é transformado (traduzido) em código de máquina específico do sistema operacional e hardware onde será executado
  - Esse processo é feito por um compilador antes da execução do programa
- Geralmente, oferecem um desempenho muito alto
  - O código é otimizado para o sistema em que está rodando.
- O código compilado para uma plataforma geralmente não funciona em outra sem recompilação

A Java Virtual Machine - JVM

Linguagens Interpretadas

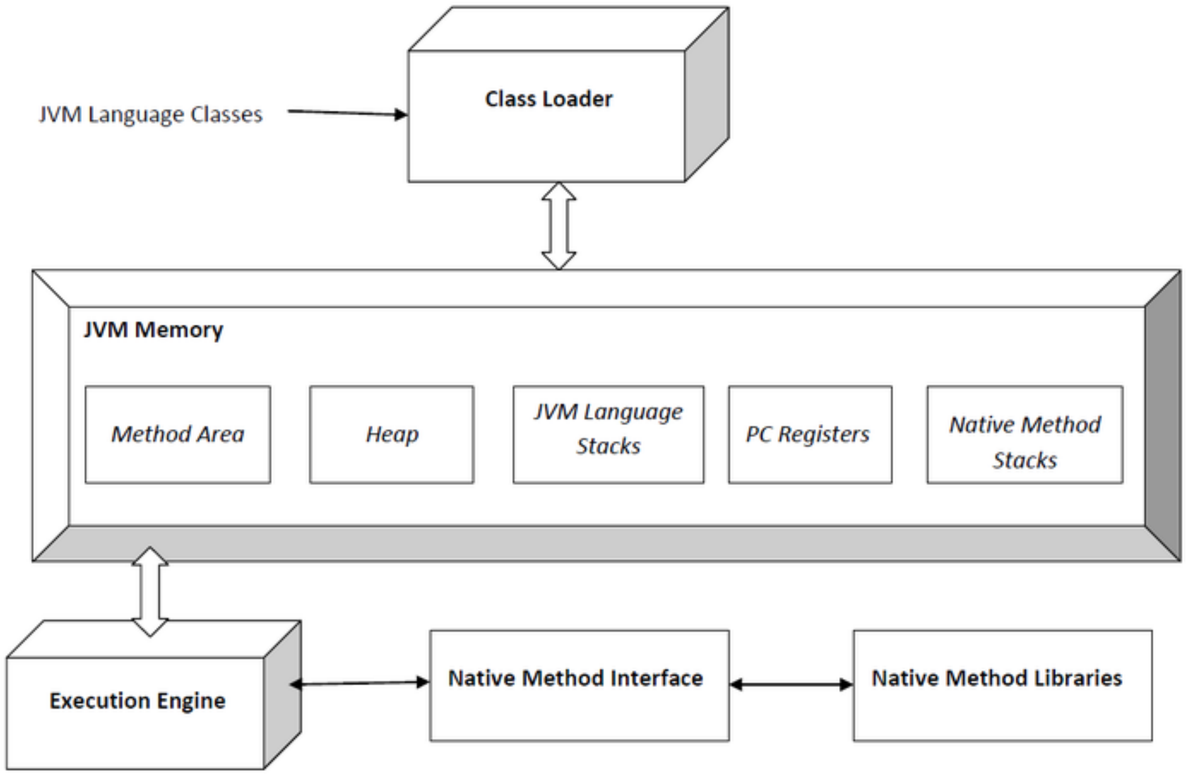
- O código-fonte é interpretado linha a linha em tempo de execução por um interpretador
  - Não há necessidade da etapa de compilação (tradução)
- Maior flexibilidade e portabilidade
- O código pode ser executado em qualquer sistema que tenha o interpretador
- Geralmente, o desempenho é inferior ao de linguagens compiladas

A Java Virtual Machine - JVM

Linguagens Híbridas

- Combina características de linguagens compiladas e interpretadas.
- O código é compilado para uma linguagem intermediária
    - Em Java: o compilador (javac), traduz o código Java para bytecode
    - Esse bytecode é então interpretado pela JVM
- O mesmo bytecode pode rodar em qualquer sistema que tenha uma JVM
- Esse modelo permite ao Java manter a portabilidade e ainda obter ótimo desempenho com o uso do JIT (Just-In-Time Compiler).

A Java Virtual Machine - JVM

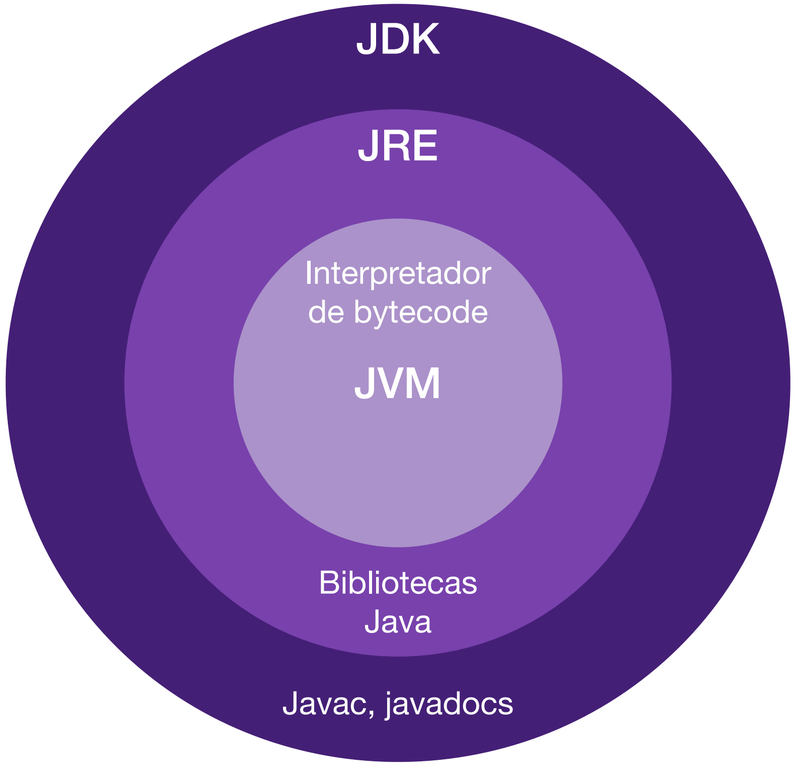


Fonte: Wikipedia

A Java Virtual Machine - JVM

JVM, JRE, JDK?

- JVM : A virtual machine, esse download não existe, ela sempre vem acompanhada
- JRE = Java Runtime Environment
  - Ambiente de execução Java
  - Formado pela JVM e bibliotecas necessárias para executar uma aplicação Java
- JDK = Java Development Kit
  - É formado pela JRE somado a ferramentas de desenvolvimento, como o compilador



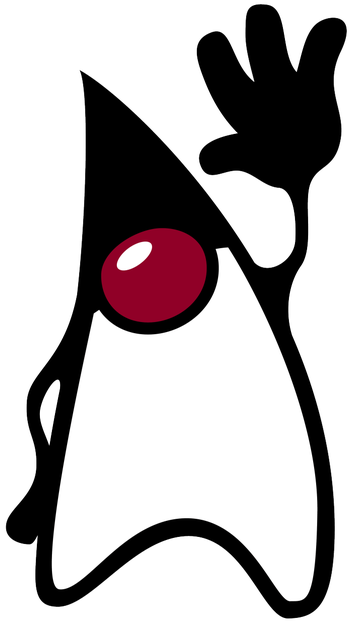
A Java Virtual Machine - JVM

JVM, JRE, JDK?

- JSE = Java Standard Edition
  - Ferramentas e APIs essenciais para qualquer aplicação Java
  - É suficiente para desenvolver aplicações desktop com ou sem interface gráfica
- JEE = Java Enterprise Edition
  - Ferramentas e APIs para o desenvolvimento de aplicações distribuídas
  - Engloba tecnologias tais como RMI, EJB, CORBA, JMS, etc.
- ~~JME = Java Micro Edition~~
  - ~~Ferramentas e APIs para o desenvolvimento de aplicações para aparelhos portáteis (palms, celulares, eletrodomésticos)~~

Devemos instalar o JavaSE

Princípios básicos



Princípios básicos

Características

- Sintaxe baseada em C e C++, familiar para vários programadores
- Elimina várias redundâncias de C++
- Simples para algumas aplicações, desde que se conheça alguns pacotes

Princípios básicos

Orientada a objetos

- Objetos e Classes
- Encapsulamento (dados e operações)
- Subtipos e Herança
- Polimorfismo
- Ligações dinâmicas (dynamic binding)
- Criação e remoção dinâmica de objetos

- ~~Variáveis e funções globais~~
- ~~Ponteiros~~
- ~~goto, struct e union~~
- ~~Tipos fracos~~
- ~~Remoção programática de objetos (liberação de memória)~~

Princípios básicos

- É case-sensitive
- As classes, métodos ou blocos sempre estarão delimitados por { }
- Um comando deve ser finalizado por ; sempre
- Nomes de variáveis, classes e métodos sempre devem começar por letras
- Uma aplicação em Java é caracterizada por possuir o método main( )

Princípios básicos

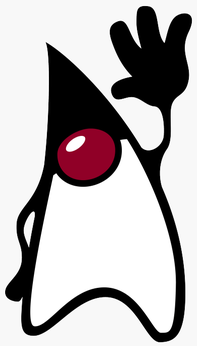
O método main

- É um método especial pois representa o ponto de entrada para a execução de um programa em Java
  - É o primeiro método que o interpretador chamará
  - Controla o fluxo de execução do programa
  - Executa qualquer outro método necessário para a funcionalidade da aplicação
- Nem toda classe terá um método main

Princípios básicos

Seu primeiro programa em Java

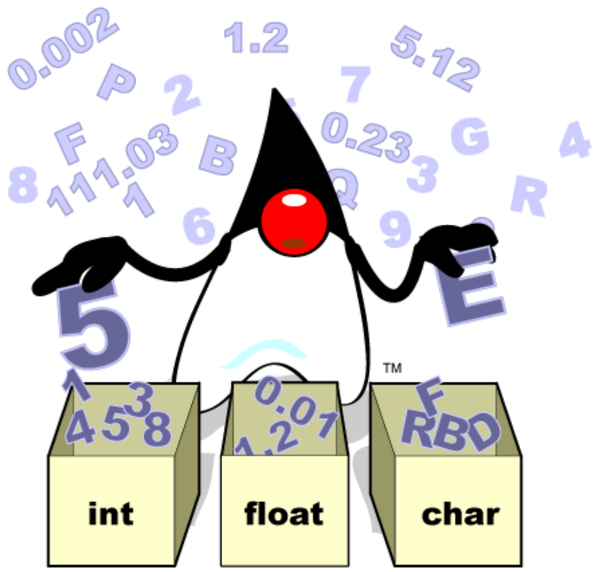
```
class OlaMundo {
    public static void main(String[] args) {
        System.out.println("Olá Mundo!");
    }
}
```



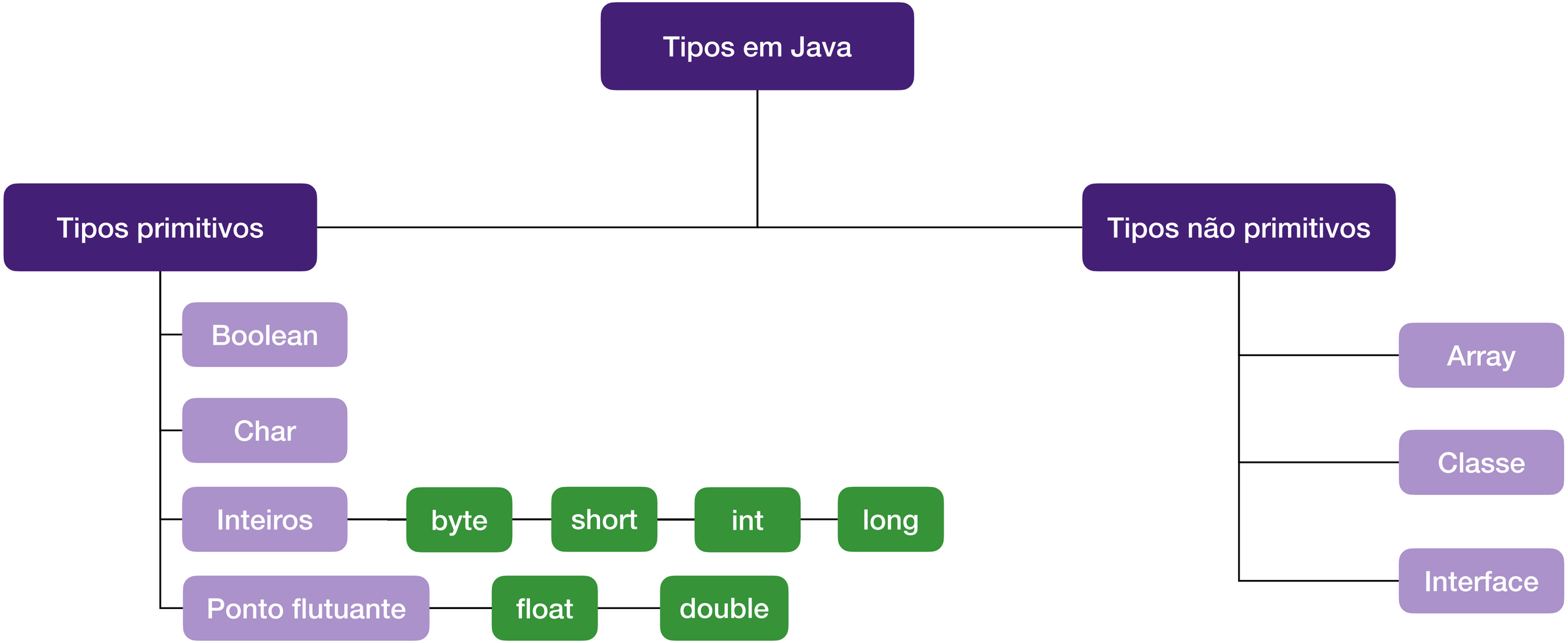
Compilando e executando

1. javac OlaMundo.java
2. java OlaMundo

Tipos de dados



Tipos de dados



Tipos de dados

Categoria	Tipo	Tamanho	Intervalo
Inteiros	byte	1 byte	[-128, 127]
	short	2 bytes	[-32.768, 32.767]
	int	4 bytes	[-2.147.483.648, 2.147.483.647]
	long	8 bytes	[-9.223.372.036.854.775.808, 9233.372.036.854.775.807]
Ponto Flutuantes	float	4 bytes	
	double	8 bytes	
Caractere	char	16 bits	[0, 256]
Booleano	boolean	1 bit	true / false

Tipos de dados

Array

Declaração

Instanciação

```
int meuVetor[] = new int[5];
meuVetor[0] = -1;
```

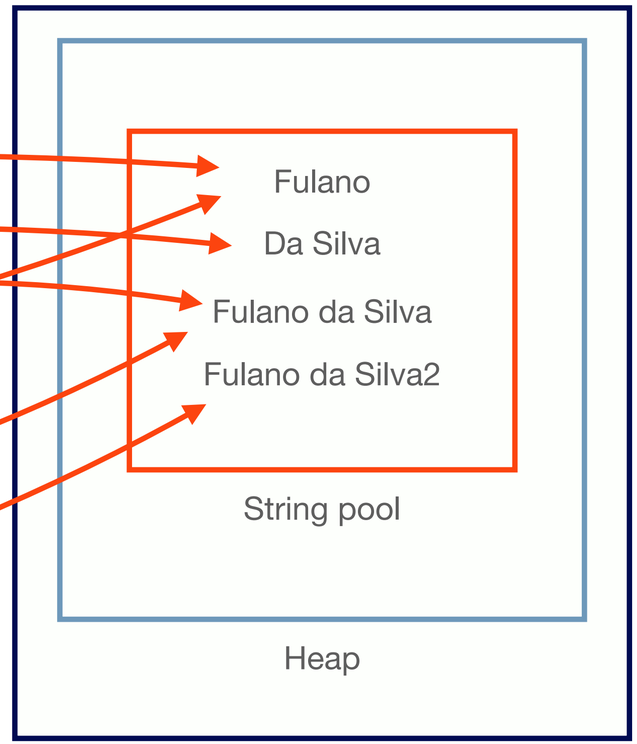
Inicialização

Tipos de dados

String

- Não são um tipo primitivo, são objetos
- São imutáveis

```
String nome = "Fulano";
String sobrenome = "da Silva";
System.out.println(nome + " " + sobrenome); // Fulano da Silva
String aux = "Fulano";
System.out.println(nome.length()); //6
System.out.println(nome.equals(aux)); //true
nome = nome + " " + sobrenome;
System.out.println(nome.equals(aux)); //false
System.out.println(nome + 2); // Fulano da Silva2
```



Memória da JVM

Classes e Objetos

Classes e objetos

- Representa um conjunto de objetos com características afins
- Define
  - Quais estados eles podem alcançar (através seus atributos)
  - O comportamento dos seus objetos (através de seus métodos)

Classes e objetos

Criando uma classe

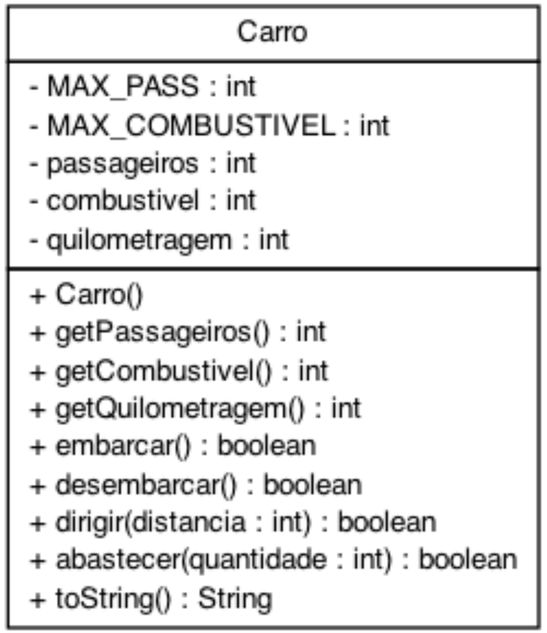
```
package <nome do pacote>
import <nome do pacote/classe>

<modificador de acesso> class <nome da classe>
{
     <Declaração das Variáveis de Instância (Atributos)>
     <Declaração de Métodos>

public static void main( String args[] ) {
        //corpo principal do programa
     }
}
```

Classes e objetos

Criando uma classe




```java
public class Carro {

    private int passageiros;
    private int combustivel;
    private int quilometragem;

    public int getPassageiros() {}

    public int getCombustivel() {}

    public int getQuilometragem() {}

    public boolean embarcar() {}

    public boolean desembarcar() {}

    public boolean dirigir(int distancia) {}

    public boolean abastecer(int quantidade) {}

    public String toString() {}

}
```

Classes e objetos

Criando objetos a partir de uma classe

```
public static void main(String[] args) {

    Carro carro1 = new Carro();
    Carro carro2 = new Carro();
    Carro carro3 = new Carro();
    ...
    Carro carroN = new Carro();
}
```

Declarando variáveis

Declarando variáveis

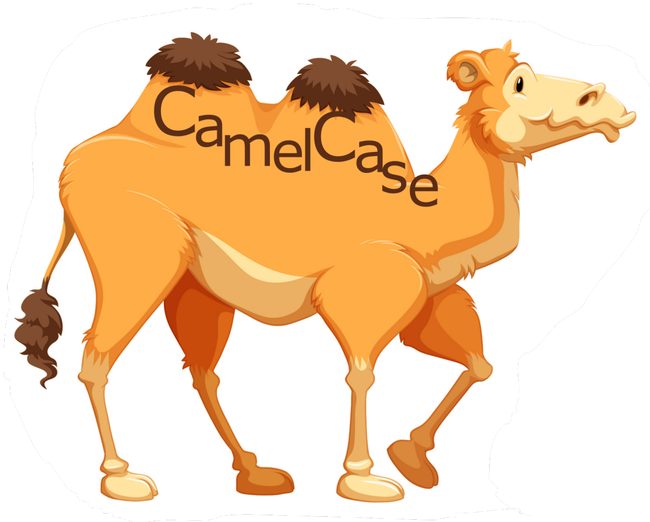
```
//Declarando variáveis
//<tipo> <variavel>
int a, b;
float c;
Carro carro;

//Declarando e inicializado variáveis
//<tipo> <variavel> = <expressao>
double d = 1.0;
long z = 123L;
Carro carro3 = new Carro();
```

Declarando variáveis

Convenções

Tipo de declaração	Regra	Exemplos
Classes e Interfaces	• Primeira letra maiúscula	• class MinhaClasse • interface MinhaInterface
Métodos e variáveis	• Primeira letra minúscula	• double salario = 0.0; • void sacar(float montante)
Constantes	•Todas as letras em maiúsculo	• MAX_PASSAGEIRO

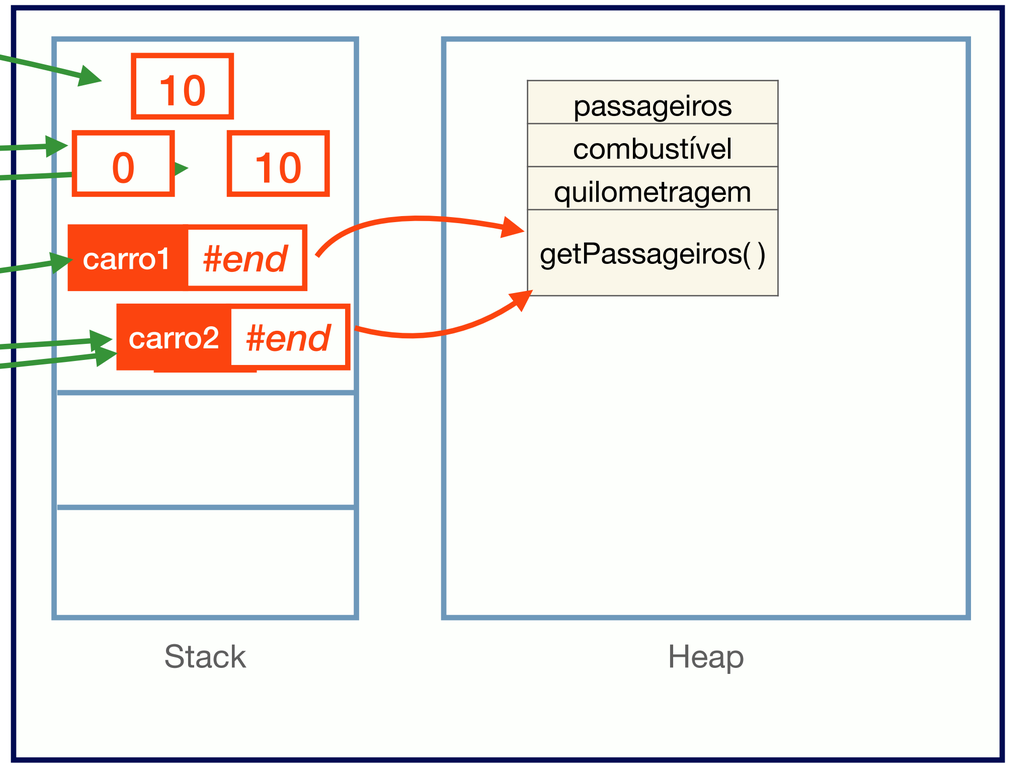


Declarando variáveis

Referência vs Tipo primitivo

```
int a = 10;
int b;
b = a;

Carro carro1 = new Carro();
Carro carro2;
carro2 = carro1;
```



Memória da JVM

Operadores

Operadores

Aritméticos

Operador	Operação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto da divisão
++	Incremento
--	Decremento

Operadores

Relacionais

Operador	Operação
>	Maior
<	Menor
<=	Menor ou igual
>=	Maior ou igual

Operadores

Lógicos

Operador	Operação
==	Igualdade
!=	Diferente
!	Negação
&&	E
||	Ou

Operadores

Binários

Operador	Operação
Complemento binário	~
And	&
Or	|
Xor	^
Deslocamento à esquerda	<<
Deslocamento à direita	>>

Lendo e Imprimindo dados

Lendo e imprimindo dados

Escrevendo dados na saída padrão (monitor)

```
System.out.print(“dados”); // Imprime e continua na mesma linha
System.out.println(“dados”); // Imprime e salta de linha
System.out.println( x ); // (x é uma variável)
```

Lendo e imprimindo dados

Lendo dados da entrada padrão (teclado)

```
import java.util.Scanner;

class Exemplo {


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

      System.out.print( "digite uma linha: " );
      String linha = scanner.nextLine() ; // lê a linha
      System.out.print( "digite um numero: " );
      int i = scanner.nextInt() ; // lê um inteiro
      System.out.print( "digite um numero: " );
      double d = scanner.nextDouble();//lê um ponto-flutuante
    }
}
```

Estruturas de controle de fluxo

Estruturas de controle de fluxo

If / else

```
if (condicaoBooleana) {
    // seu cogigo aqui
} else {
    // mais codigo aqui
}
```

- Uma condição booleano é qualquer expressão que retorna true ou false

Estruturas de controle de fluxo

Switch

```
switch ( cmd ){
    case 0:
        System.out.println(“Item do menu 1”);
        menu = ++cmd;
        break;
    case 1:
        System.out.println(“Item do menu 2”);
        menu = ++cmd;
        break;
    default:
        System.out.println(“Comando invalido!”);
}
```

- Aceita um char, byte, short int, String, tipos enumerados e algumas outra classes especiais
- A palavra reservada break deve vir dentro do bloco case para evitar que os blocos case seguintes sejam executados
- O caso default não é obrigatório, mas é uma boa prática sempre utilizá-lo

Estruturas de controle de fluxo

While / Do While

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    i = i + 1;
}
```

```
do {
    salario = salario * 0.5;
    i--;
} while ( i != 0 );
```

- while: O trecho de código será executado enquanto a condição permanecer verdadeira
- do/while: O bloco é executado pelo menos uma vez
  - Após a primeira execução a condição é testada

Estruturas de controle de fluxo

For

```
for (int i = 0; i < 10; i = i + 1) {
    System.out.println("olá!");
    System.out.println(i);
}
```

- Isola um espaço para inicialização de variáveis e o modificador dessas variáveis
- Realiza o controle da variável de iteração automaticamente

Estruturas de controle de fluxo

Controlando os laços

- Podemos decidir parar o loop por algum motivo especial sem que o resto do laço seja executado

```
for (int i = x; i < y; i++) {
    if (i % 19 == 0) {
        System.out.println("Achei um número divisível por 19 entre x e y");
        break;
    }
}
```

Estruturas de controle de fluxo

Controlando os laços

- Da mesma maneira, é possível obrigar o loop a executar o próximo laço

```
for (int i = 0; i < 100; i++) {
    if (i > 50 && i < 60) {
        continue;
    }
    System.out.println(i);
}
```

Estruturas de controle de fluxo

For-each

- Da mesma maneira, é possível obrigar o loop a executar o próximo laço

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Por hoje é só