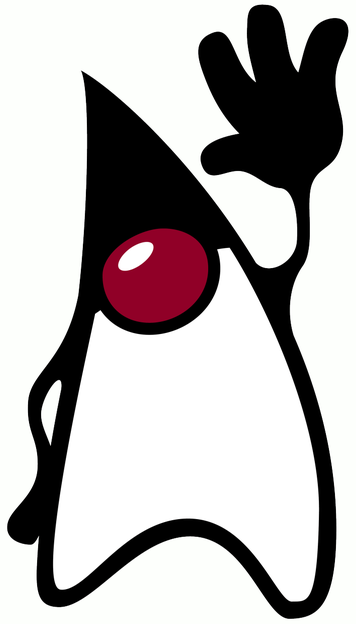
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Introdução ao Java

QXD0007 - Programação Orientada a Objetos

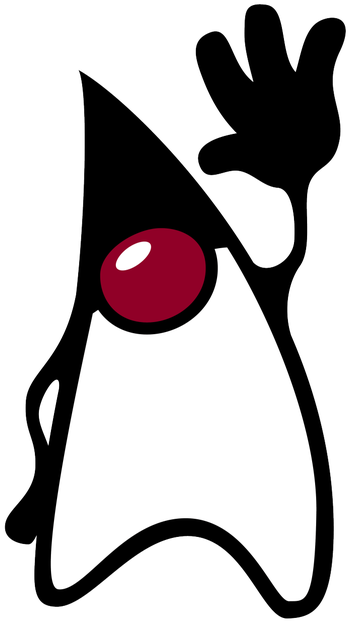


Prof. Bruno Góis Mateus (brunomateus@ufc.br)

Conteúdo

- Princípios básicos
- Tipos de dados
- Classes e Objetos
- Declarando e usando variáveis
- Operadores
- Lendo e Imprimindo dados
- Estruturas de controle de fluxos

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
- A declaração do método deve ser : `public static void main(String[] args)`

Princípios básicos

Estrutura básica de um programa em Java

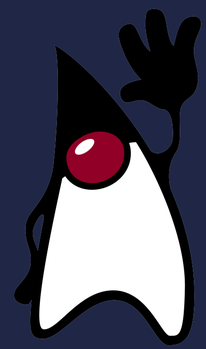
- Sempre precisaremos de pelo menos UMA classe
- A definição de uma deve respeitar a seguinte sintaxe

```
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
     }
}
```

Princípios básicos

Seu primeiro programa em Java

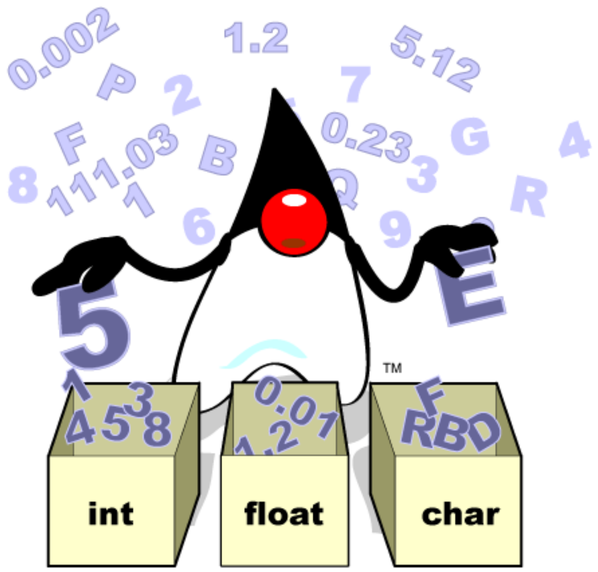
```
class OlaMundo {
    public static void main(String[] args) {
        System.out.println("Olá Mundo!");
    }
}
```



Compilando e executando

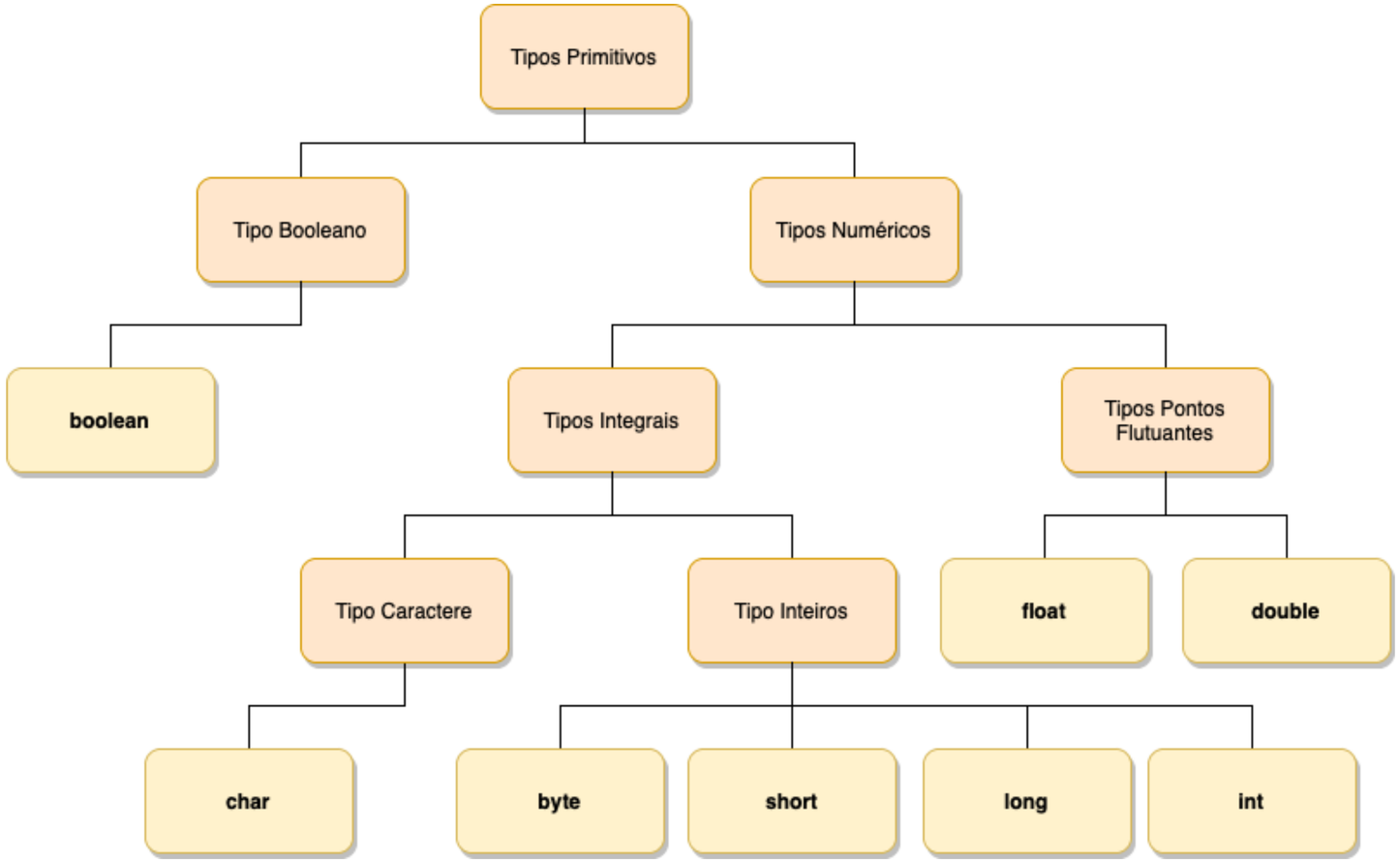
1. javac OlaMundo.java
2. java OlaMundo

Tipos de dados



Tipos de dados

- Uma variável pode ser de um dos seguintes tipos:
  - Tipo Primitivos
  - Array
  - Classe
  - Interface



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;
```

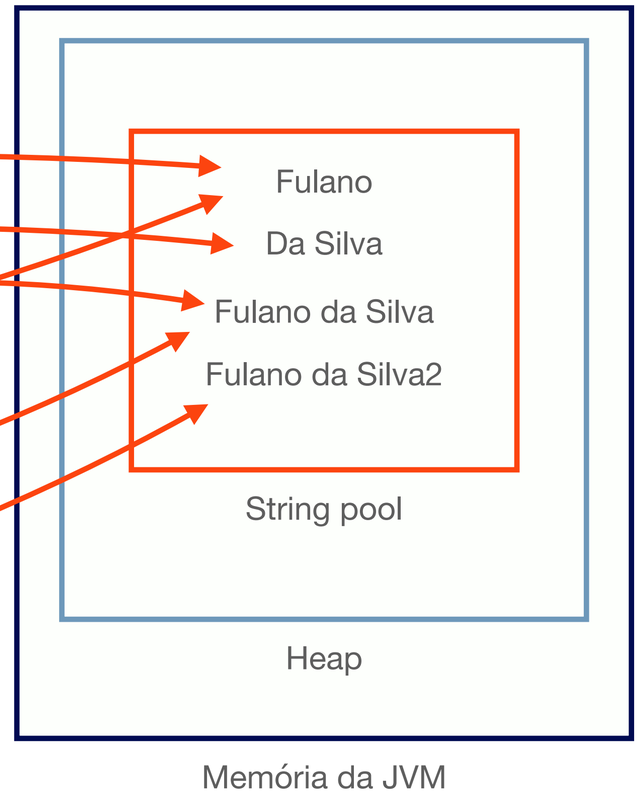
Inicialização

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
```



Classes e Objetos

Classes e objetos

Classe de objetos

- Representa um conjunto de objetos com características afins. Uma classe
- Define o comportamento dos seus objetos (através das suas operações/ métodos) e quais estados eles podem alcançar (através das suas propriedades/atributos).

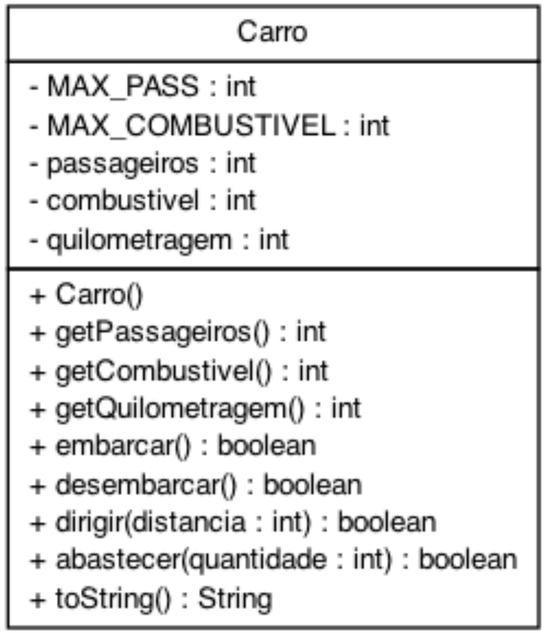
Classes e objetos

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 boolean 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

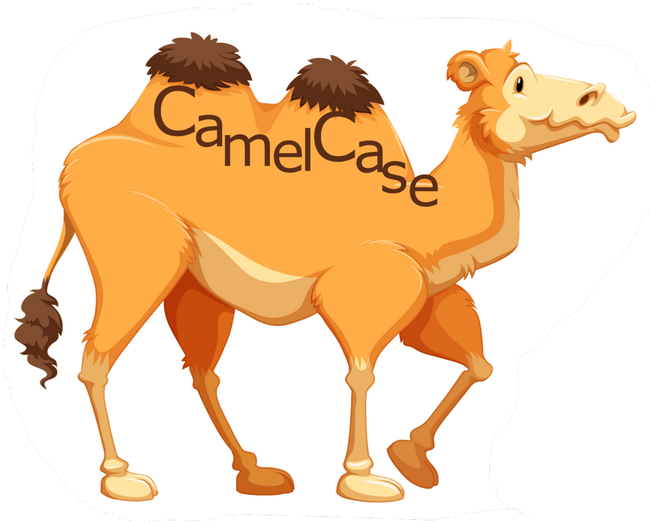
Declarando variáveis

```
//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

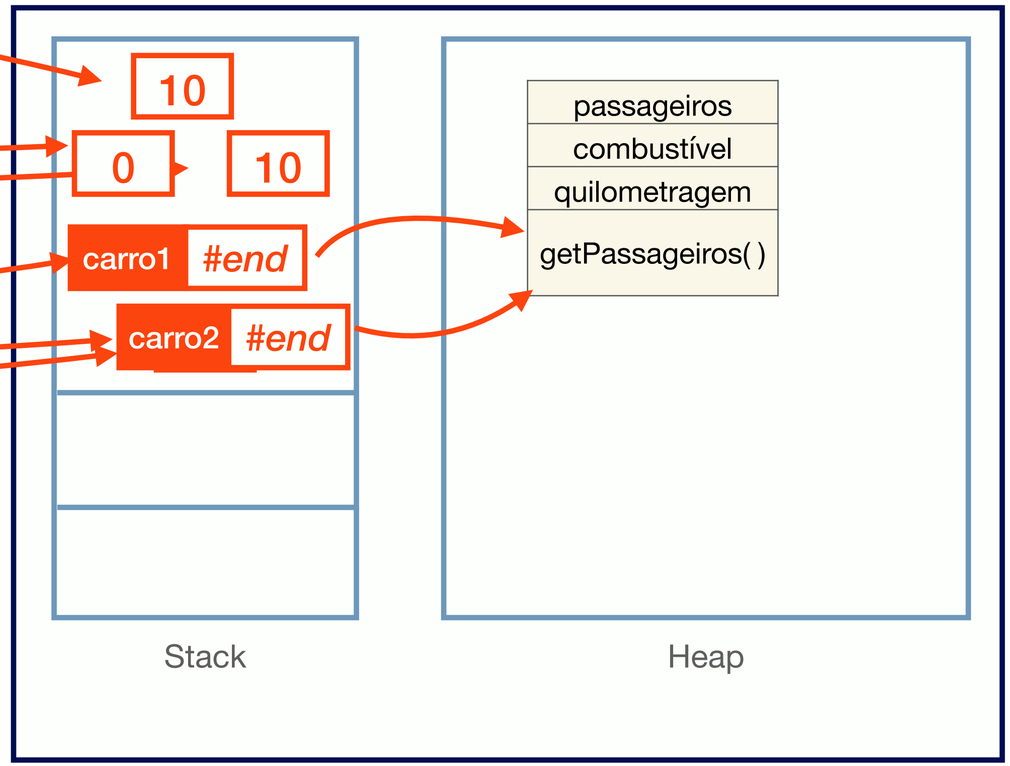
Tipo de declaração	Regra	Exemplos
Classes e Interfaces	• Primeira letra maiúscula	• class MinhaClasse • interface MinhaInterface
Métodos e variáveis	• Primeira letra minúscula	• double salario = 0.0; • void sacar(float montante)
Constantes	•Todas as letras em maiúsculo	• MAX_PASSAGEIRO



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;
    case 2:
        System.out.println(“Item do menu 3”);
        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 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);
}
```

Por hoje é só