

# Aula 01 – Introdução

## MAC0321 - Laboratório de Programação Orientada a Objetos

Professor: Marcelo Finger (mfinger@ime.usp.br)

Departamento de Ciência da Computação  
Instituto de Matemática e Estatística



# Tópicos

1. Apresentação do curso
2. Início do processo de POO
3. Primeiros passos em Java

# 1. Apresentação do Curso



# Professor

- Marcelo Finger
  - Engenheiro de Elétrico (Poli-USP)
  - Mas professor do IME-USP
  - Sala 210C (Bloco C do IME)
  - [mfinger@ime.usp.br](mailto:mfinger@ime.usp.br)

# Objetivos

- Conceitos de Orientação a Objetos (OO)
- Aspectos básicos de programação
  - Qualidade do código
  - Programação defensiva e tratamento de erros

## **Objetivos secundários**

- Aprendizado da linguagem Java
- **Testes de Unidades**
- Depuração
- Uso de um sistema de controle de versão

# Programa do Curso

- Ver página do curso

# Programa

Aula	Assunto
1	Visão geral e padrões de código
2	Conceitos básicos de Orientação a Objetos
3	Encapsulamento
4	Ciclo de vida de um objeto
5	Interface e diagrama de classes da UML
<b>P1</b>	
6	Herança
7	Classe abstrata e polimorfismo
7B	Exercício Longo
8	Outros conceitos de herança e polimorfismo
9	Pacotes e biblioteca padrão do Java
?? Sem aula (semana de provas)	
10	Programação defensiva e exceção

# Organização

- **Teoria**  $\approx$  1,5 hora / **Prática**  $\approx$  2 horas
  - Exercícios entregues até 19h (*Menos hoje!*)
- Aprovação: MEP  $\geq$  5 e E  $\geq$  5

MEP =	EP1 + 2*EP2
	3

- E: média aritmética dos exercícios feitos em aula
- EP1: Exercício Programa (meio do curso)
- EP2: Exercício Programa Final
- Média Final =  $(3*MEP + E)/4$ , se MEP, E  $\geq$  5;  $\min(\text{MEP}, E)$ , caso contrário
- Não tem SUB



# Bibliografia básica

- BUDD, T. **An Introduction to Object-Oriented Programming**. 3rd Edition. Addison-Wesley. 2001.
- GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley. 1995.
- ORACLE. **The Java Tutorials**. Disponível em: <<http://docs.oracle.com/javase/tutorial/>>, 2022.
- **Outras Recomendações**
  - SIERRA, K.; BATES, B. **Head First Java**. 2<sup>nd</sup> Edition. O'Reilly. 2005.
  - FREEMAN, E.; FREEMAN, E. **Head First Design Patterns**. O'Reilly. 2004.

# Organização

- Material e entregas no eDisciplinas
- <https://edisciplinas.usp.br/course/view.php?id=116619>

## 2. Visão Geral de POO



# Interesse de POO

- Modelo mais próximo a realidade
- Facilidade na reutilização de código
  - bibliotecas de classes
  - componentes
- construção de código reutilizável
  - mais difícil !!!
- facilidade de evolução
  - interfaces bem definidas
  - baixo acoplamento entre as partes do sistema
- Permite a construção de GRANDES SISTEMAS

# Pequeno histórico de POO

- Smalltalk: primeira linguagem
  - Smalltalk-76, Smalltalk-80
  - Simula-67
- Linguagens OO mais usadas:
  - C++ derivada de C, com suporte para OO
  - Java linguagem OO pura (?)
  - Python: tudo é um objeto, mas..  
(quase) ninguém programa **orientado** a objeto em Python. Não é parte da cultura da linguagem

# POO (pura): princípios

- Tudo é um **objeto** → variável especial que guarda dados e tem um *comportamento* determinado
- Um programa é um conjunto de objetos que se comunicam
- Cada objeto tem a sua memória composta por outros objetos
- Cada objeto tem um tipo **(classe)** ← Determina quais chamadas estão disponíveis
- Todos os objetos de um mesmo tipo podem responder às mesmas mensagens

# POO (pura): características

- Encapsulamento  
(abstração de dados)
- Herança  
(novas classes são derivadas de classes existentes)
- Polimorfismo  
(o método depende dos objetos parâmetros)

# Encapsulamento

- Objeto = Dados + Métodos  
[Memória + Comportamento]
- Interação com os objetos : mensagens
  - Uma mensagem é uma requisição de serviço
- Cada mensagem corresponde a um método
  - A mensagem causa a execução do método
  - A mensagem é o nome do método, possivelmente seguido de uma lista de parâmetros



# POO fundamentos

## Vantagens do encapsulamento:

- esconder detalhes: não é necessário saber como foram feitas as funções membro para usá-las;
- pode-se trocar a implementação do objeto sem prejuízo ao resto do programa;
- pode-se prever métodos que podem ser úteis e implementá-los no futuro;
- os objetos podem ser testados individualmente.

# POO: Java vs Python

- Java: encapsulamento **fortemente** imposto:
  - Manutenção mais fácil; testagem mais difícil
  - Facilita desenvolvimento de grandes sistemas
- Python: encapsulamento voluntário, sempre pode ser violado
  - Mais fácil de testar
  - Fácil de interfacear com bibliotecas de outras linguagens
  - Sistemas: aglutinamento de bibliotecas

# Desenvolvimento OO

- maior dificuldade  $\implies$  pensar em bons objetos
- antes de começar a escrever o código, fazer a análise detalhada do problema:
  - Quais serão os objetos ?
  - Qual a interface deles ?

Atenção: nem sempre é possível definir totalmente os objetos antes de começar a codificação

Para isto as membros abstratos podem ser muito úteis

# 3. Visão Geral do Java



# Linguagens de Programação

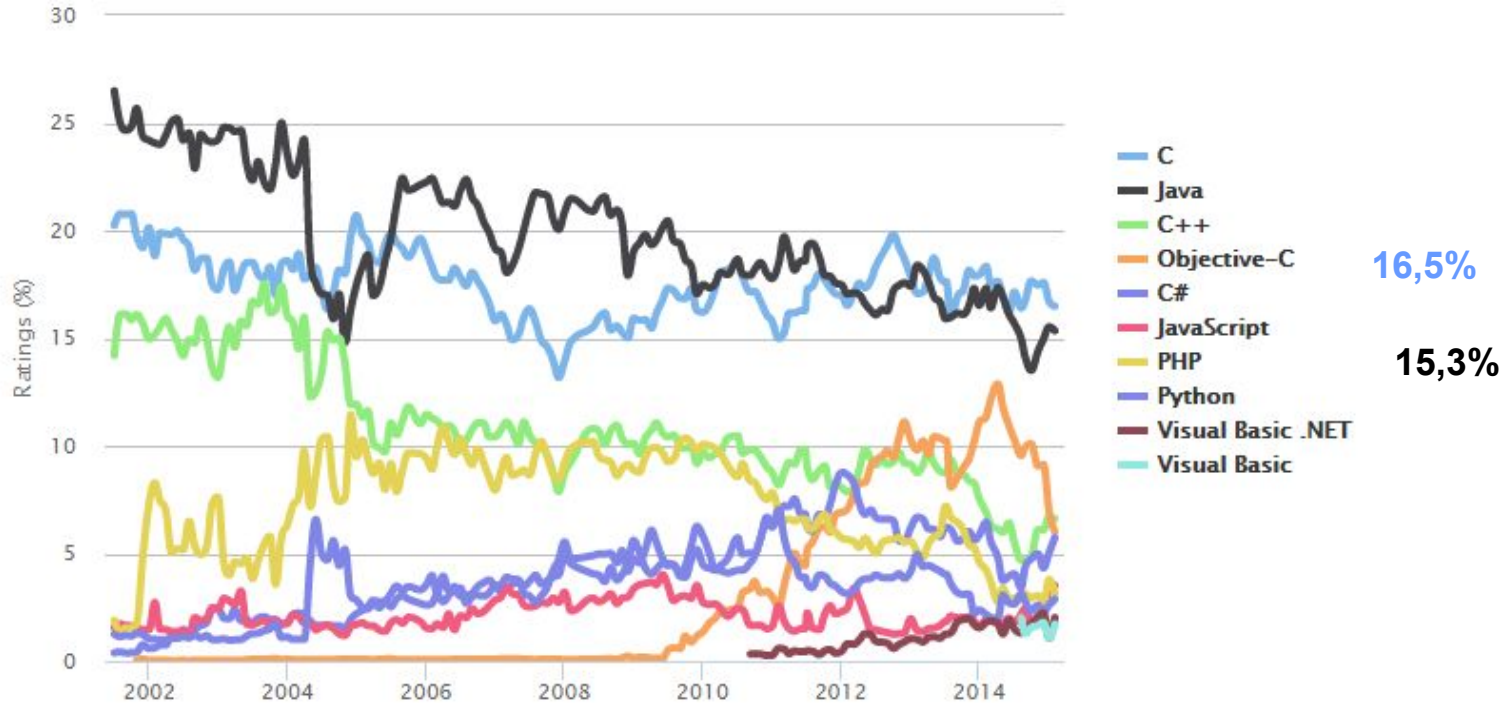
- Meados da década de 1940 – Linguagem de Máquina  
0100 0011 1010 1111 0000 1010 0001 0011
- 1948 – Linguagem de Montagem (Assembly)  
ADD AX, BX
- Linguagens de Alto Nível
  - FORTRAN (1957)
  - LISP (1958)
  - Simula-67, Smaltalk-80
  - C, C++, C#
  - Java, Python, Ruby, Javascript, Clojure, Go, Hack, Haskell ...

# Java

- Linguagem de programação
  - Orientada a Objetos (*quase tudo é objeto*)
- Ambiente para rodar programas
- Criada em 1995
- Desenvolvida pela *Sun Microsystems*
  - A Sun foi adquirida pela Oracle em 2009
- Baseada no C++
  - Sintaxe e semântica parecidas



# Popularidade de Java



Fonte: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

# Java

- Exemplos de uso do Java
  - Sites\*
    - Banco do Brasil, TAM, Garmin, Levi's (EUA) e Gap (EUA)
  - Software
    - IRPF, Eclipse, IntelliJIDEA, OpenOffice\* e Vuze
  - Android
    - Java é a linguagem de desenvolvimento



# Tecnologias Java

## Java SE

- Ambiente para aplicações em desktops e servidores

## Java EE

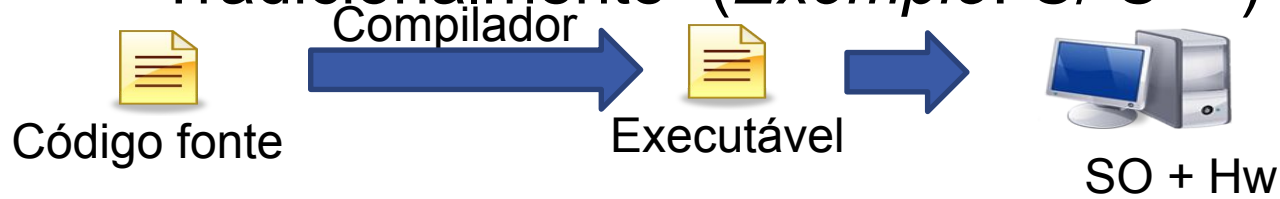
- Ambiente para aplicações distribuídas em servidores

## Java ME

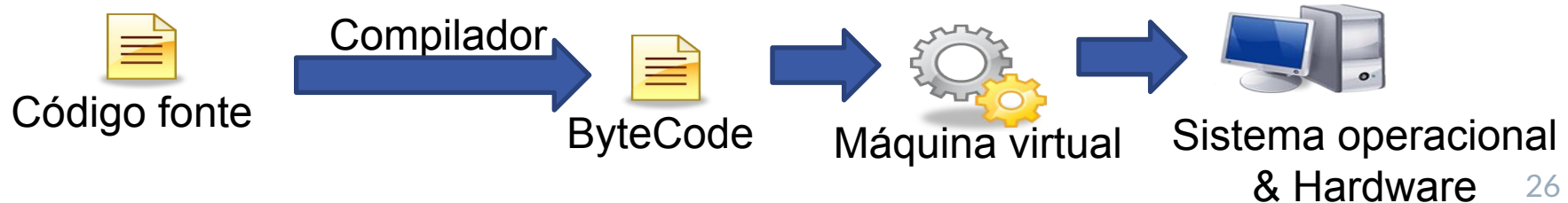
- Ambiente para aplicações que rodam em dispositivos como telefones, PDAs, televisões etc. (Obs: Android não é J2ME)

# Características

- Independente de *plataforma*
  - Windows, Linux, Mac, Solaris...
- "Tradicionalmente" (*Exemplo: C/ C++*)



- Java (linguagem Híbrida, como Python, etc)



# Independente de plataforma

- Comportamento comum em qualquer plataforma
- Pseudocódigo interpretado
- **Problemas**
  - Desempenho
    - Interpretação em tempo de execução
  - Não aproveita vantagens da plataforma
    - *Exemplo*: telas gráficas e otimizações

# Características

- Código aberto
  - Não é software livre
  - “Semi-proprietária”
    - *Java Community Process*
    - Evolução da linguagem é gerenciada pela *Oracle*
- Comunidade Java
  - <http://www.oracle.com/technetwork/java/>
  - <http://www.apache.org>
  - <http://www.sourceforge.net>
  - ChatGPT !!!

# Ambientes de programação (IDE)

- DrJava
  - <http://www.drjava.org/>
- Eclipse
  - <http://www.eclipse.org>
- IntelliJIDEA (JetBrains)
  - <https://www.jetbrains.com/idea/>
- NetBeans (apoio da Oracle)
  - <http://www.netbeans.org>
- Rational Software Architect (IBM)
  - <http://www.ibm.com/>



# Zerésimo exemplo

- Imprime um texto no console
  - Crie um arquivo HelloWorldNao.java

```
public class HelloWorldNao {  
    public static void main (String[] args) {  
        System.out.println("Qualquer coisa menos Hello World!");  
    }  
}
```

- Compile (abra um *prompt* de comando: cmd)

```
javac HelloWorldNao.java
```

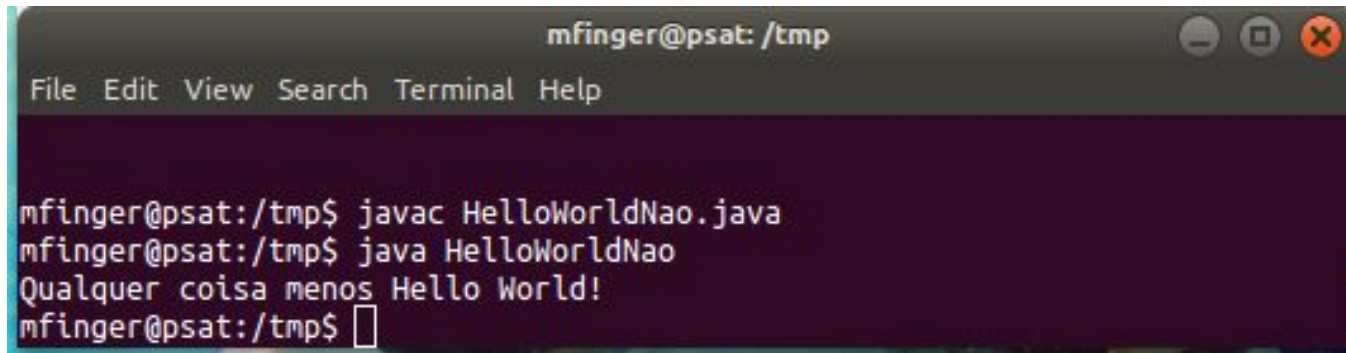
- Execute

```
java HelloWorldNao
```

- Cuidado com as letras maiúsculas e minúsculas!!!

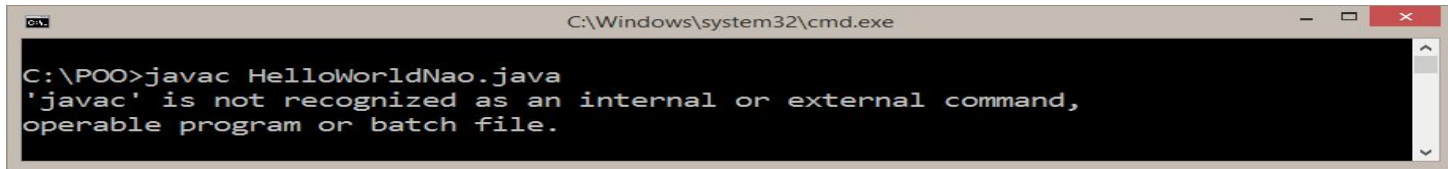
# Zerésimo exemplo

- Imprime um texto no console

A screenshot of a macOS Terminal window. The title bar shows 'mfinger@psat: /tmp' and standard window controls. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal content shows the following commands and output:

```
mfinger@psat:/tmp$ javac HelloWorldNao.java
mfinger@psat:/tmp$ java HelloWorldNao
Qualquer coisa menos Hello World!
mfinger@psat:/tmp$
```

# Problemas



```
C:\Windows\system32\cmd.exe
C:\POO>javac HelloWorldNao.java
'javac' is not recognized as an internal or external command,
operable program or batch file.
```

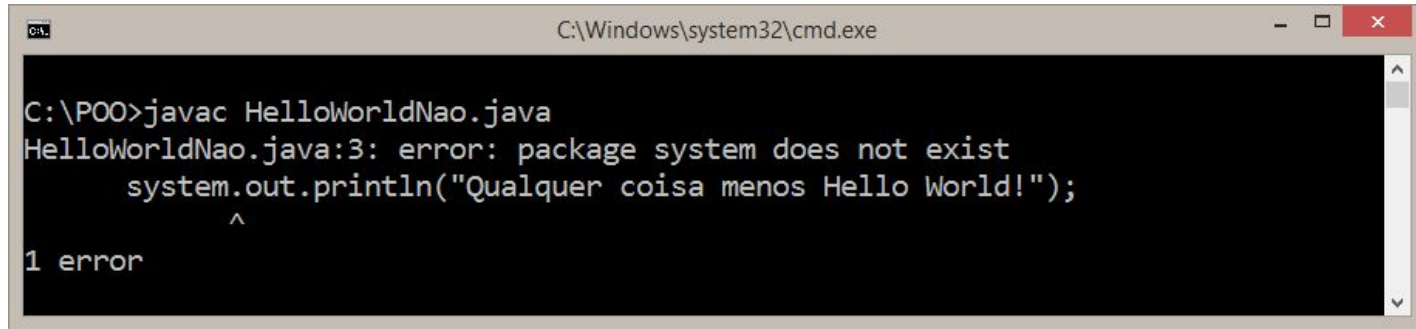
- **Motivo:** JDK não configurado adequadamente
  - Variável de ambiente PATH no Windows
- **Solução:** escreva no *prompt* de comando
  - **Dica:** use o tab para autocompletar a pasta no prompt

```
SET PATH=%PATH%;"C:\Program Files\Java\jdk1.8.0_31\bin"
```

**Observação:** Toda vez que o *prompt* for aberto é necessário refazer isso ou deve-se configurar as variáveis de ambiente



# Problemas



```
C:\Windows\system32\cmd.exe

C:\POO>javac HelloWorldNao.java
HelloWorldNao.java:3: error: package system does not exist
    system.out.println("Qualquer coisa menos Hello World!");
    ^
1 error
```

- **Motivo:** erro de compilação
- **Solução:** reveja o código!

# Zerésimo exemplo

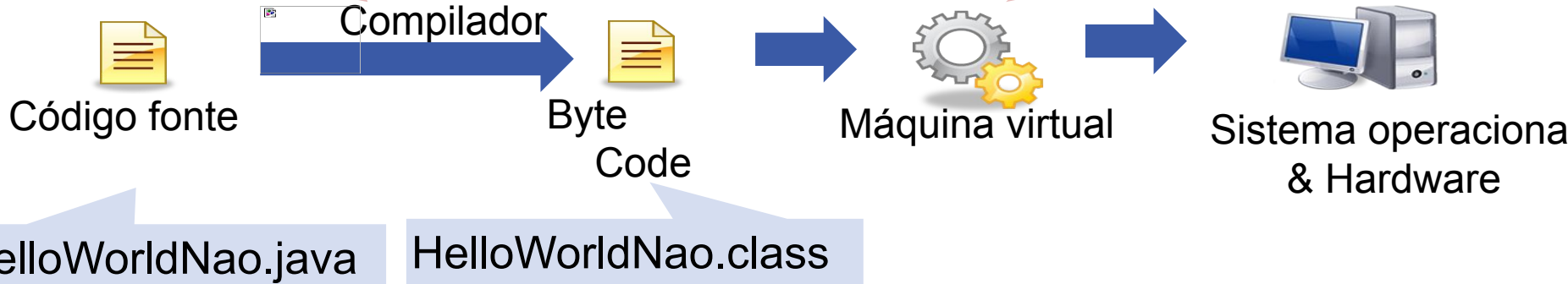
- Extensão `.java`
  - Arquivo com o código fonte do programa Java
- Extensão `.class`
  - Arquivo com o *ByteCode* Java
  - Executado pela Máquina Virtual para rodar o programa

# Zerésimo exemplo

- Comandos e arquivos gerados

```
javac HelloWorldNao.java
```

```
java HelloWorldNao
```



# Instalação do Eclipse

- [Eclipse Downloads | The Eclipse Foundation](#)

- Em downloads, escolha a arquitetura
- Baixar, descompactar e rodar “eclipse-installer”
- Selecionar “Eclipse IDE for Java Developers”
- Clicar em Install, aceitar os termos



# 4. Elementos Básicos da Linguagem Java



# Variáveis

- Declaração
  - Tipo, identificador e valor (*opcional*)

```
int numeroDePessoas;  
boolean confirmado = true;  
int maior = 100, menor = 0;  
double x, y = 50.0;
```

## Observações sobre os identificadores

- Há diferença entre letras maiúsculas e minúsculas
- A primeira letra não pode ser um dígito
- Não pode conter espaços
- Pode usar acentuação (mas às vezes há problemas nas ferramentas por causa da codificação usada)
- Variáveis podem ser declaradas em qualquer parte do bloco
- Bloco: conjunto de comandos entre "{" e "}"

# Tipos primitivos

Tipo	Valores	Bits	Faixa de valores	Exemplo
boolean	Booleano	1	Verdadeiro / Falso	true, false
char	Caractere	16	Unicode 16 bits	'a', ',', 125
byte	Byte	8	-128 a 127	-125, 3
short	Número	16	$-2^{15}$ a $2^{15} - 1$	0, -1, 15000
int	Número	32	$-2^{31}$ a $2^{31} - 1$	0, -1, 15000
long	Número	64	$-2^{63}$ a $2^{63} - 1$	0, -1, 1E10
float	Número com ponto flutuante	32	$\pm 1.4 \cdot 10^{-45}$ a $\pm 3.4 \cdot 10^{38}$	-1.45E-30
double	Número com ponto flutuante	64	$\pm 4.9 \cdot 10^{-324}$ a $\pm 1.8 \cdot 10^{308}$	1.9E100

# Condição e laço

- Condição

```
if (x == 0) {  
    // ...  
} else if (x > 0) {  
    // ...  
} else {  
    // ...  
}
```

- Laços

- While

```
while (x > 0) {  
    // ...  
}
```

```
do {  
    // ...  
} while (x > 0);
```

```
for (int i = 0; i < 10; i++) {  
    // ...  
}
```



# Operadores lógicos

- Principais operadores lógicos

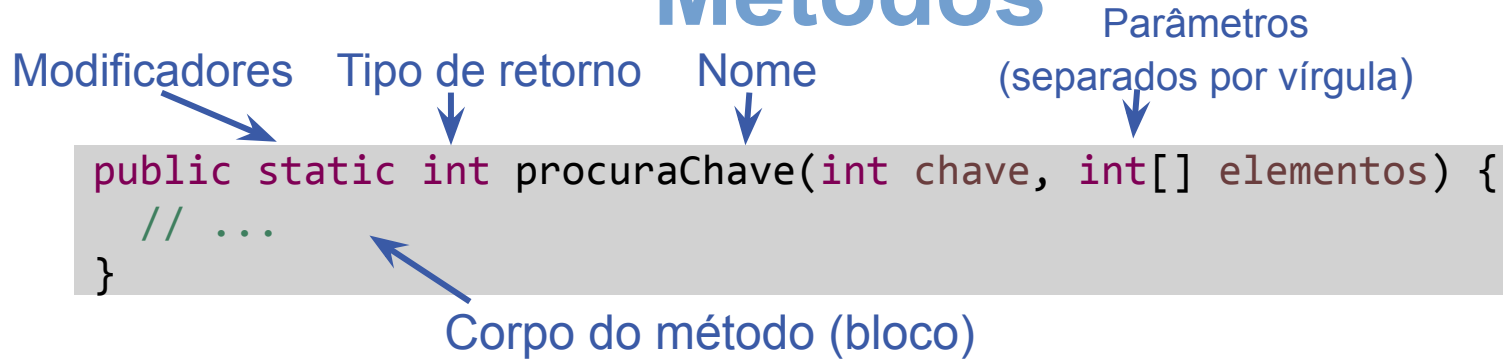
Operador	Descrição
&&	E lógico
	Ou lógico
!	Negação
==	Comparação

- Exemplo*

```
boolean encontrado = false;
int x = 0, y = 0;
// ...

if (!encontrado && (x > 0 || y > 5)) {
    // ...
}
```

# Métodos



- Chamada de um método

```
retorno = procuraChave(10, vetor);
```

- Retorno de valores

```

public static void f() {
    // ...
    return;
    // ...
}
  
```

Sem retorno

```

public static int g() {
    // ...
    return 1;
    // ...
}
  
```

Com retorno (inteiro)

# Comentários

- Dois tipos de comentários

```
x++; // O resto da linha é comentado
```

- Comenta do “//” em diante até o fim da linha

- /\* e \*/

- Comenta o texto entre os /\* e \*/

```
/*  
  Exercício 1  
  Autor: Meu nome  
  Data: 25/02/2015  
*/
```

```
/* Comentário */ x++;
```

# String

- Tipo definido pela biblioteca padrão
  - Não é um *tipo primitivo*

```
String nome;  
String endereco = "Rua X";
```

- Concatenação de Strings
  - Operador “+”

"texto 1" + "texto 2" é equivalente a "texto 1texto 2"

- Concatenação de Strings com outros tipos
  - *Converte* o tipo para String

"texto A" + 1 é equivalente a "texto A1"

# Vetor

- É um conjunto ordenado de variáveis de um mesmo tipo

x[0]	15
x[1]	-29
x[2]	54
x[3]	-1
x[4]	3

Índice

Vetor x

# Vetor

- Declaração do vetor

```
int[] numeros;
```



Declara um vetor (variável desse tipo)

**Cuidado:** Apesar de declarado, o vetor não está criado! Ainda não é possível usá-lo normalmente!

- Declaração x criação do vetor
  - *Qual é o tamanho?*
  - *(Quais são os valores?)*

# Vetor

- Criação do vetor

```
int[] numeros = new int[5];
```



Declara um vetor e o cria com espaço para 5 elementos inteiros

```
int[] numeros = {56, 78};
```



Declara um vetor e o cria com dois elementos: o valor 56 e o valor 78

- Em Java é possível usar variáveis para definir o tamanho do vetor

- *Exemplo:* o tamanho do vetor é definido pela variável “tamanho”

```
int[] numeros = new int[tamanho];
```

# Vetor

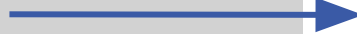
- Acesso aos elementos do vetor

```
numeros[0] = 10;
```



Atribui o valor 10 à posição 0 do vetor (1ª posição)

```
x = numeros[5];
```



Atribui o valor da posição 5 do vetor à variável x

- Tamanho do vetor

- O vetor tem “associado” a ele a informação do tamanho

```
x = numeros.length;
```



Atribui à variável x o tamanho do vetor numeros

**Observação:** não é necessário (nem elegante) guardar em uma variável o "numeros.length". Use-o diretamente.



# Vetor

- Existe um for específico para varrer um vetor
  - (Funciona para outros tipos de *conjuntos*)

```
String[] nomesDosAlunos = {"Fabio", "Joao", "Maria"};
```

Em cada iteração, um valor de `nomesDosAlunos` é colocado em `nome`

```
for (String nome : nomesDosAlunos) {  
    System.out.println(nome);  
}
```



Equivalente a

```
for (int i = 0; i < nomesDosAlunos.length; i++) {  
    String nome = nomesDosAlunos[i];  
    System.out.println(nome);  
}
```

# Programa Java

## ▪ Programa Java básico

```
import java.util.Date;
import java.util.Random;

public class NomeDaClasse {

    public static int teste() {
        return 0;
    }

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

Classes importadas (exemplo)  
(por enquanto não é necessário)

Nome da classe

Métodos

- Os métodos têm que estar dentro de uma classe!
- O nome do arquivo deve ser igual ao nome da classe
  - Com extensão .java

# Programa Java

- O método `main` é o que inicia a execução do programa
  - Ponto de entrada do programa Java
  - Opcional e no máximo um `main` por classe
  - Precisa ter exatamente a seguinte assinatura
    - (A menos do nome do argumento...)

```
public static void main(String[] args) {  
    // Código!  
}
```

# Saída

- Formas de se imprimir na tela

Comando	Saída
<code>System.out.print("Texto");</code>	Texto
<code>System.out.print(1);</code>	1
<code>System.out.print("Número: " + i);</code>	Número: 1

→ Valor de i

- `System.out.println`
  - Imprime o dado e pula uma linha
    - (Equivalente a um `System.out.print` com `"\r\n"` no final)
- `System.out.printf`
  - Semelhante ao `printf` da linguagem C

# Entrada

- (Ainda não veremos a *entrada padrão*)

- Entrada através de chamada de parâmetros  
`java Teste parametro1 parametro2`

- Os parâmetros são acessíveis no vetor de Strings declarado no método **main**

```
public static void main(String[] args) {  
    System.out.println(args[0]);  
    System.out.println(args[1]);  
}
```

→ Imprime *parametro1*  
→ Imprime *parametro2*

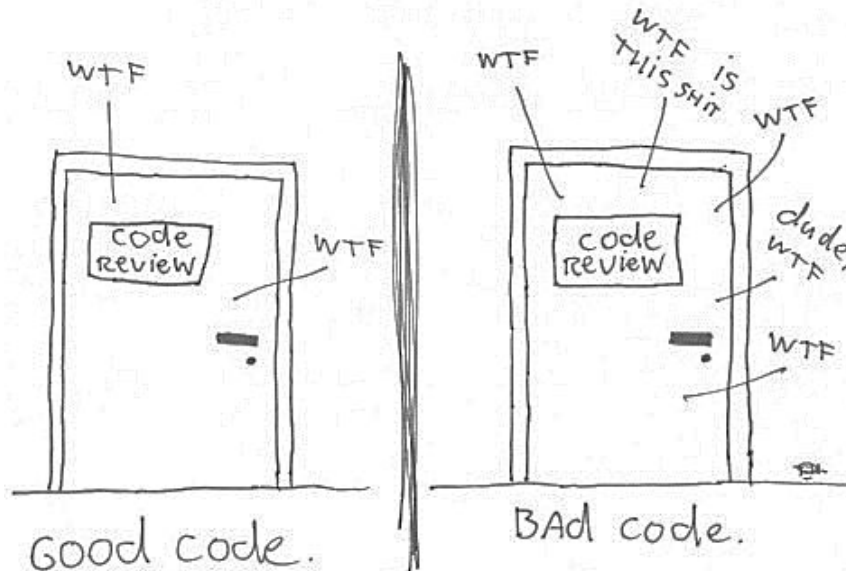
- O tamanho do vetor (`.length`) é igual ao número de parâmetros passados

# Qualidade do código

- O que é um *bom código*?
  - Correto
    - Faz o que se espera
  - Eficiente
    - Não desperdiça recursos (memória e processamento)
  - Elegante
    - Simples, limpo, bonito e sem enfeites
  - Testável
    - Fácil de procurar defeitos

# Qualidade do código

The ONLY valid MEASUREMENT  
OF code QUALITY: WTFs/minute



# Qualidade do código

```
public class Exemplo1 {
    public static void main(String[] args) {
        int i, j;
        int k;
        int l;
        int[] m;
        k = args.length; m = new int[k];
        for (i = 0; i < k; i++) {
            m[i] = Integer.parseInt(args[i]);
            l = 0;
        }
        for (j = 0; j < k; j++) {
            if (m[j] > 0)
                l = l + m[j];
        }
        System.out.println("O valor é: " + l);
    }
}
```

Alguns problemas:


- Nome das variáveis
- Indentação
- Espaçamento
- Blocos não claros



# Qualidade do código

```
public class Exemplo2 {  
    public static void main(String[] args) {  
        if (args.length == 0) {  
            System.out.println("Informe uma lista de números.");  
            return;  
        }  
  
        int somaDosPositivos = 0;  
        int valorAtual;  
  
        for (int i = 0; i < args.length; i++) {  
            valorAtual = Integer.parseInt(args[i]);  
            if (valorAtual > 0)  
                somaDosPositivos += valorAtual;  
        }  
        System.out.println("O valor é: " + somaDosPositivos);  
    }  
}
```

# Nome de variáveis

- Use nomes representativos
  - O nome deve revelar o significado da variável!
- Evite nomes redundantes
  - *Exemplo*: `int valorInteiro;`
- Evite abreviações 
  - *Exemplo*: `double med;`      `double media;`
  - ...evite nomes muito longos...
  - (Bons IDEs ajudam a preencher nomes longos)
- Use nomes do domínio de aplicação

# Nome de variáveis

- Convenção do Java: *CamelCase*
  - Variáveis e métodos
    - Primeira palavra com letra minúscula e as demais com só a 1ª letra em maiúscula
    - *Exemplo*: `i`, `peso`, `maiorNumero` e `pedidosAtrasados`
  - Nomes de arquivos
    - Cada palavra com a 1ª letra maiúscula e as demais minúsculas
    - *Exemplo*: `HelloWorld.java`, `Exemplo1.java`, `ListaLigada.java`

# Comentários

- **Comentários não melhoram um código ruim**
  - *Melhore o código ao invés de comentá-lo!*
- **Evite comentários desnecessários**
  - O ideal é o código ser autoexplicativo
    - Ou seja, escreva um código que *não precisa de comentários!*
  - O comentário deve explicar algo que não é possível explicitar no código
- **Apague código que não é mais usado**
  - (Existem formas melhores de manter um histórico)

# Organização geral

- Declaração de variável no início do bloco
  - Bloco real (*if*, *for*, etc.) ou *lógico* (organização do código)
- Declare somente a variável controladora do *for* no *for*
  - *Exemplo:*

```
for (int i = 0, z; i < 10; i++)
```



```
int z;  
for (int i = 0; i < 10; i++);
```



# Espaçamento

- Use um espaço para separar uma palavra da seguinte
  - =, <=, *while*, *if*, *for* etc. são palavras
- Deixe um espaço depois, mas não antes, de cada sinal de pontuação
  - *Exemplo*: `int i, j;`
- Parênteses
  - Deixe um espaço antes, mas não depois, de abrir um parêntese
  - Deixe um espaço depois, mas não antes, de fechar um parêntese
  - *Exemplo*: `for (int i = 0; i < 5; i++) j++;`

# Espaçamento

- Exceções
  - `x[i]` e não `x [i]`
  - `x++` e não `x ++`
  - Para funções e métodos
    - `calculoComplexo(x)` e não `calculoComplexo (x)`
  - `"."` não é sinal de pontuação em Java!

# Chaves (bloco)

- O { deve ficar na mesma linha do comando que define o bloco, com um espaço antes

- ```
if (x > 5) {
```

 ✓

- ```
if (x > 5){
```

 ✗

- ```
if (x > 5)  
{
```

 ✗

- ```
if (x > 5)  
{
```

 ✗

- O } deve ficar em uma linha separada, mas alinhado com o comando que definiu o bloco

- *Exemplo*

```
if (x > 5) {  
    ...  
}
```

 ✓



# Blocos

- O conteúdo de um bloco deve ficar tabulado (um “tab”, normalmente)
  - Deve ser possível diferenciar o que está dentro ou fora do

```
for (int i = 0; i < 5; i++) {  
    if (entrada[i] > 5) {  
        System.out.println(entrada[i]);  
    }  
}
```



```
for (int i = 0; i < 5; i++) {  
if (entrada[i] > 5) {  
    System.out.println(entrada[i]);  
}}
```



# Recomendações

- `if`, `else` e `else if` devem ficar alinhados

```
if (x > 5) {  
    // ...  
} else if (x < 0) {  
    // ...  
} else {  
    // ...  
}
```



- As chaves podem ser omitidas se o bloco tiver apenas uma linha
  - Indente ou deixe na mesma linha o comando

```
if (x > 5)  
    i++;
```



```
if (x > 5) i++;
```



# Outras recomendações

- *Evite copy-paste*
  - *Reorganize* o código
- Evite processamento desnecessário
- Evite excesso de otimização
  - *Código ilegível, mas eficiente!*
    - Dificuldade de manutenção
  - (a menos que otimização seja fundamental)
- (*The International Obfuscated C Code Contest*)
  - <http://www.ioccc.org>
- **O fundamental é ser consistente**

- **Primeira Aplicação Java:**  
*“Hello, World”*

- Arquivo fonte `HelloWorld.java`:

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```

Note que o nome do arquivo fonte é o nome da classe, mais a extensão `.java`

- Para compilar e executar:

```
javac HelloWorld.java           (gera HelloWorld.class)
```

```
java HelloWorld                 (roda HelloWorld.class)
```

# Segunda Aplicação Java

- Arquivo fonte HelloWorldDate.java:

```
public class HelloWorldDate {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
        System.out.println(new Date());  
    }  
}
```

## Observações:

- Nomes das classes **XxxxXxxx**
- **Public** keyword - acessível para todos
- **Static** keyword - função sem objeto

# Terceira Aplicação Java

- Achar o n-ésimo número primo
  - Vamos escrever o mesmo programa em C e em Java
  - Motivação:
    - Comparar a sintaxe das duas linguagens
    - Comparar o desempenho
  - Algoritmo:
    - Vai obtendo cada primo, em sequência, e guardando num vetor
    - Para verificar se um certo número é primo, tenta dividi-lo pelos primos menores que ele (esses já estão no vetor)

# N-ésimo Número Primo em C

```
#include <stdio.h>
#include <stdlib.h>

static int *primes;
static int len;

static int is_prime(int k)
{
    int i;
    for (i = 0; i < len; i++) {
        if (2 * primes[i] > k)
            return 1;
        else if ((k % primes[i]) == 0)
            return 0;
    }
    return 1;
}
```



# N-ésimo Número Primo em C (cont.)

```
int main(int argc, char *argv[])
{
    int n;
    int i;

    if (argc != 2) {
        fprintf(stderr, "Usage: nth prime n\n");
        exit(1);
    }
    n = atoi(argv[1]);
    if (n <= 0) {
        fprintf(stderr,
                "n must be greater than zero\n");
        exit(1);
    }
}
```



# N-ésimo Número Primo em C (cont.)

```
primes = malloc(n * sizeof(int));
if (primes == NULL) {
    fprintf(stderr, "Not enough memory\n");
    exit(1);
}
len = 0;
for (i = 2; len < n; i++) {
    if (is_prime(i))
        primes[len++] = i;
}
printf("%d\n", primes[len - 1]);
free(primes);
}
```

# N-ésimo Número Primo em Java

Arquivo fonte `NthPrime.java`:

```
public class NthPrime {  
  
    private static int[] primes;  
    private static int len;  
  
    private static boolean isPrime(int k) {  
        for (int i = 0; i < len; i++) {  
            if (2 * primes[i] > k)  
                return true;  
            else if ((k % primes[i]) == 0)  
                return false;  
        }  
        return true;  
    }  
}
```

- **N-ésimo Número Primo em Java (cont.)**

```
public static void main(String[] args) {
    if (args.length != 1) {
        System.err.println("Usage: java NthPrime n");
        System.exit(1);
    }
    int n = Integer.parseInt(args[0]);
    if (n <= 0) {
        System.err.println("n must be "
            + "greater than zero");
        System.exit(1);
    }
}
```

# N-ésimo Número Primo em Java (cont.)

```
    primes = new int[n];
    len = 0;
    for (int i = 2; len < n; i++) {
        if (isPrime(i))
            primes[len++] = i;
    }
    System.out.println(primes[len - 1]);
}                                     // end of main method

}                                     // end of class NthPrime
```

- Para compilar e executar:

`javac NthPrime.java` (gera `NthPrime.class`)

`java NthPrime 4` (acha o quarto primo)

`java NthPrime 1000` (acha o milésimo primo)

# Observações

- Essa aplicação Java não tem nada de OO
- Ela tem a mesma estrutura que o programa escrito em C
- Java não força você a programar OO
- A linguagem incentiva, mas não impõe o paradigma de programação OO

- **Exercício**

- Fazer um programa enésimo primo OO em espírito e forma.

# Bibliografía complementar

- **MARTIN, R. C. Clean Code: A Handbook of Agile Software Craftsmanship.** Prentice Hall. 2008. Capítulo 2.

# MAC321

## Lab POO

- ▶ Professor: Marcelo Finger  
E-mail: [mfinger@ime.usp.br](mailto:mfinger@ime.usp.br)