MC322 - Programação orientada a objetos
Aula 2
Introdução à Java

Prof. Marcos M. Raimundo
Instituto de Computação - UNICAMP

Características

A Linguagem Java: Características

- Sintaxe similar a C++
- Elementos básicos da linguagem são: objetos, classes e interfaces
- Bytecode interpretado sobre máquina virtual: Independência de ambiente operacional
- Herança simples e múltiplas interfaces: Simulação de herança múltipla através das interfaces

A Linguagem Java: Características

- Simples, Familiar e Orientada a Objetos
  - Os conceitos fundamentais são simples o suficiente para que sejam facilmente absorvidos pelos programadores.
  - A semelhança com outras linguagens como C++ torna a transição para Java mais fácil e familiar.
  - A necessidade crescente de sistemas distribuídos e complexos, baseados em cliente-servidor, coincide com os conceitos de linguagem orientada a objetos, especialmente a possibilidade de encapsulamento fornecida pela linguagem.
- Robusta e Segura
  - Construída para permitir o desenvolvimento de aplicativos de alta confiabilidade.
  - O gerenciamento de memória é bastante simples: não há ponteiros explícitos (criados pelo programador) e as memórias que deixaram de ser utilizadas pelo aplicativo são automaticamente liberadas pelo coletor de lixo (garbage collector).
  - Ela provê extensivas checagens de segurança, em tempo de compilação, para impedir invasão externa de sistemas, invasão de arquivos e criação de vírus.

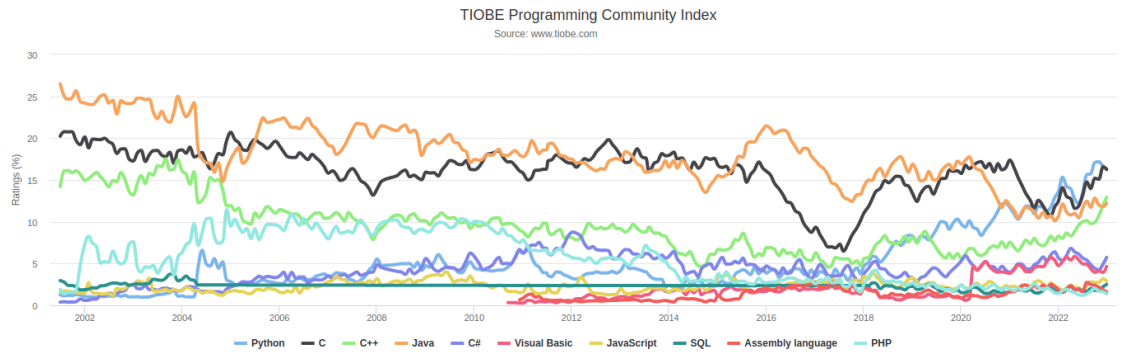
A Linguagem Java: Características

- Independente de arquiteturas e Portável
  - As aplicações Java suportam plataformas, hardwares e redes heterogêneas.
  - Compilada para bytecodes, que são interpretadas por uma máquina virtual.
  - A portabilidade também é alcançada pela convenção dos tamanhos dos tipos de dados e dos comportamentos dos operadores aritméticos.
- Interpretada e Paralelizável
  - Os bytecodes podem ser interpretados em qualquer máquina contanto que a máquina virtual esteja devidamente instalada.
  - A plataforma Java permite multithreading com a adição de métodos para sincronizaão. Muitas bibliotecas em Java já foram elaboradas para suportar **multithreading** e **concorrência**.
- Bom desempenho
  - Compatível com outras linguagens.
  - O garbage collector é executado somente como uma thread de baixa prioridade.
  - Aplicações que requerem melhor desempenho podem ter parte de seus códigos reescritos em linguagem de máquina (Hot Spot compiler).

A Linguagem Java: Características

- Tipos fortes, com ligação dinâmica.
- Coleta de lixo.
- Sem aritmética de ponteiros.
- Bibliotecas portáveis.
- Threads e monitores.
  Programação concorrente.
- Integração com C/C++.
  Métodos nativos.

A Linguagem Java: Características



Fonte: TIOBE Index – www.tiobe.com

Executando um código em Java

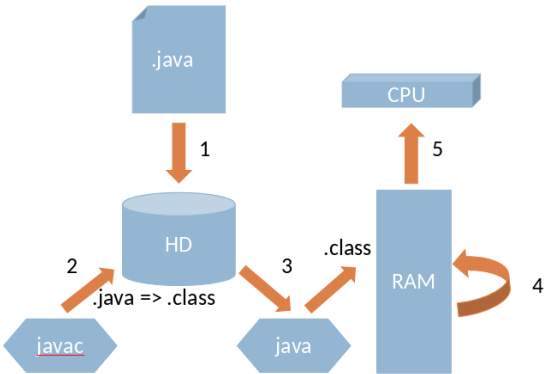
Java: uma tecnologia

- JVM (Java Virtual Machine) - virtual machine
- JRE (Java Runtime Environment) - ambiente de execução Java, formado pela JVM e bibliotecas. Contém os elementos para executar uma aplicação Java.
- JDK (Java Development Kit) – necessário para desenvolvedores. Inclui o compilador java e kit de desenvolvimento
- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)

A Linguagem Java: Fases

Programas Java passam por cinco fases até sua execução:

1. Edição
2. Compilação
3. Carregamento
4. Verificação
5. Execução



A Linguagem Java: Fases

1. Edição (código fonte)
   - Código-fonte
   - Salvo em disco com extensão .java
   - Edição
2. Compliação (código fonte => Bytecode)
   - Nesta fase o compilador Java é executado para o código-fonte utilizando o comando javac em um terminal:
     javac Teste.java
   - Resultado: código em bytecodes no arquivo Teste.class
3. Carregamento (carregar o .class na memória)
   - JVM carrega o código em bytecode na memória
     - java Teste.class
     - São carregados o Teste.class e todas as APIs (http://docs.oracle.com/javase/7/docs/api/) usadas
   - Resultado: código carregado na memória principal

A Linguagem Java: Fases

4. Verificação (bytecode)
   - Nesta fase, enquanto as classes são carregadas, o verificador de bytecodes da JVM examina os bytecodes para assegurar que são válidos e não violam restrições de segurança
   - Os códigos Java passam por diversas certificações de segurança para reduzir ataques a arquivos e sistemas por vírus de computador
5. Execução (código fonte $=>$ Bytecode)
   A JVM executa as intruções definidas pelos bytecodes.
   - Nas primeiras versões de Java, a JVM era simplesmente um interpretador de bytecodes, o que deixava a execução dos programas lenta
   - As atuais JVMs executam bytecodes combinando interpretação e compilação Just-In-Time (JIT). Buscam Hotspots
   - Na compilação JIT, a JVM analisa os bytecodes enquanto são interpretados, procurando por hot spots, parte dos bytecodes executados com frequência

A Linguagem Java: Convenções de Arquivo

A Linguagem Java: Convenções de Arquivo

A declaração de uma classe em Java é composta por:

- Arquivo fonte Java
  - NomeDaClasse.java
  - Cada arquivo de origem Java contém uma única classe pública ou interface.
  - Quando as classes privadas e interfaces estão associados a uma classe pública, você pode colocá-los no mesmo arquivo de origem como a classe pública.
  - A classe pública deve ser a primeira classe ou interface no arquivo.
- Arquivo bytecode Java
  - NomeDaClasse.class

O arquivo fonte Java deve conter, nesta ordem:

- Cometários iniciais
- Declarações de pacote e de importação
- Declarações de classe e de interface

A Linguagem Java: Convenções de Arquivo

```java
/*
 * ContaBancaria.java
 *
 *   Itima   modificação: 20/08/2022
 *
 * Material usado na disciplina MC322 — Programação orientada a objetos.
 */

package java.awt;
import java.awt.peer.CanvasPeer;
/**
 * Essa classe contém a estrutura de implementação de uma Conta Bancária
 */
public class ContaBancaria{
    /* Inicio da Classe */
```

A Linguagem Java: Convenções de Arquivo

Ordem	Parte da Classe / Interface	Observação
1	Comentário de documentação da Classe/interface (/**...*/)	
2	Declaração da classe ou interface	
3	Comentário de implementação da Classe/interface (/**...*/)	Opcional
4	Variáveis de classe (static)	Primeiro public, depois protected, depois package e depois private.
5	Propriedades	Primeiro public, depois protected, depois package e depois private
6	Construtores	
7	Métodos	Agrupados por funcionalidade

A Linguagem Java: Ferramentas de Compreensão de Código

Bons exemplos de quebra de linha

```
alpha = (aLongBooleanExpression) ? beta : gamma;

alpha = (aLongBooleanExpression) ? beta
                                 : gamma;

alpha = (aLongBooleanExpression)
        ? beta
        : gamma;

longName1 = longName2 * (longName3 + longName4 - longName5)
            + 4 * longName6; //PREFER
```

A Linguagem Java: Comentários

- Comentários de linha única

  devem ser precedidos de uma linha em branco

```
if (condicao) {

    /*Comentário */
    ...
}
```

- Comentários à direita

```
if (a==2) {
    return true;            /* caso especial */
} else {
    return isPrime(a);      /* funciona apenas para impar*/
}
```

A Linguagem Java: Comentários

Comentários de final de linha

- O delimitador // pode ser usado para comentar uma linha completa ou apenas uma linha parcial
- Ele não deve ser usado em várias linhas consecutivas para comentários de texto
- No entanto, ele pode ser utilizado em várias linhas consecutivas para comentar a seções de código.

```
if (foo > 1) {
    // Faca algo
} else {
    return false;       // Explique o porque aqui
}
// if (bar >1){
//
//      // faca algo
//      ...
//}
//else{
//  return false;
//}
```

A Linguagem Java: Comandos e Modificadores

A Linguagem Java: Palavras Reservadas

Palavras reservadas em Java

abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

Palavras não usadas atualmente

const goto

A Linguagem Java: Modificadores de Visibilidade

private

- Aplicado à declaração de um método ou variável, o modificador private torna este elemento acessível apenas ao código da classe onde está declarado
- Aplicado à declaração de uma classe ou interface torna esta classe ou interface inacessível fora do arquivo onde está declarada

protected

- Aplicado à declaração de um método ou variável de uma classe, torna este elemento acessível apenas às classes que pertencem ao mesmo pacote desta classe, e ao código das classes que herdam desta primeira
- Aplicado à declaração de uma classe ou interface torna esta classe ou interface visível apenas no escopo do pacote onde está declarada

A Linguagem Java: Modificadores de Visibilidade

public

- Aplicado à declaração de um método, variável, classe ou interface elimina qualquer restrição de visibilidade a este elemento

default (sem modificador)

- Aplicado à declaração de um método, variável, classe ou interface torna o elemento acessível no pacote onde foi declarado

Modificador	**Classe**	**Pacote**	**Subclasse**	**Mundo**
public	S	S	S	S
protected	S	S	S	N
default	S	S	N	N
private	S	N	N	N

A Linguagem Java: Primeiro programa

A Linguagem Java: Primeiro programa

- Arquivo Main.java

```
// Programa Java
public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Olá Mundo!");
    } // fim do método main
} // fim da classe Main
```

```
>> Olá Mundo!
```

- Todo programa Java possui pelo menos uma classe definida pelo programador
- A palavra-chave class introduz uma declaração de classe e é imediatamente seguida pelo nome da classe (Main)
- Convenção: O nome de classe deve começar por letra maiúscula assim como a inicial de cada palavra incluída (Main)
- O arquivo .java deve possuir o mesmo nome da classe declarada

A Linguagem Java: Primeiro programa

- Arquivo Main.java

```
// Programa Java
public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Olá Mundo!");
    } // fim do método main
} // fim da classe Main
```

```
>> Olá Mundo!
```

- Não existe executável em Java sem um método main. O main é o ponto de partida das aplicações Java
- O main precisa ser static, ou seja, ele é um método da classe (mais detalhes a seguir...)
- Não pode haver mais de um método main em um programa Java
- Os argumentos do main estão dentro dos ()
- O método main recebe um vetor de Strings

A Linguagem Java: Primeiro programa

- Arquivo Main.java

```
// Programa Java
public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Olá "+args[0]+"!");
    } // fim do método main
} // fim da classe Main
```

```
$ java Main MC322
> Olá MC322!
```

- Usando os argumentos do main:
- O main recebe um vetor de Strings
- Este argumento pode ser acessado pela variável args
- Se senhum argumento for passado, args tem tamanho 0

A Linguagem Java: Primeiro programa

- Arquivo Main.java

```
// Programa Java
public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Olá "+args[0]+"!");
    } // fim do método main
} // fim da classe Main
```

```
$ java Main MC322
> Olá MC322!
```

- Saída de dados:
  - A classe System fornece ao programador acesso à saída padrão
  - System.out é o objeto de saída padrão, que permite a exeibição de caracteres no terminal
  - O método System.out.println imprime uma cadeia de caracteres (fornecida como argumento) no terminal

A Linguagem Java: Tipos de dados

A Linguagem Java: Tipos de dados

- A linguagem Java é fortemente tipada
  - As variáveis precisam ter o seu tipo declarado antes de serem usadas
  - O tipo da variável não muda ao longo do programa
- Tipos de dados de Java:
  - Tipos básicos ou primitivos
    São pré-definidos pela linguagem
  - Tipos referenciados
    São usados para referenciar vetores ou tipos definidos pelo usuário

A Linguagem Java: Tipos de dados primitivos

Tipo	Descrição
boolean	Pode assumir o valor true ou o valor false
char	Caractere em notação Unicode de 16 bits. Serve para armazenar dados alfanuméricos. Também pode ser usado como um dado inteiro com valores na faixa entre 0 e 65535
byte	Inteiro de 8 bits em notação de complemento de dois. Intervalo: $[-2^7 = -128, 2^7 - 1 = 127]$
short	Inteiro de 16 bits em notação de complemento de dois. Intervalo: $[-2^{-15} = -32.768, 2^{15} - 1 = 32.767]$
int	Inteiro de 32 bits em notação de complemento de dois. Intervalo: $[-2^{31} = 2.147.483.648, 2^{31} - 1 = 2.147.483.647]$
long	Inteiro de 64 bits em notação de complemento de dois. Intervalo: $[-2^{63}, 2^{63} - 1]$
float	Representa números em notação de ponto flutuante normalizada em precisão simples de 32 bits em conformidade com a norma IEEE 754-1985. Intervalo: [1.40239846e-46, 3.40282347e+38]
double	Representa números em notação de ponto flutuante normalizada em precisão dupla de 64 bits em conformidade com a norma IEEE 754-1985. Intervalo: [4.94065645841246544e-324, 1.7976931348623157e+308]

A Linguagem Java: Tipos de dados primitivos

```java
// Programa Java
class HelloWorld{
    public static void main(String[] args){

        String b1 = "Bom dia!";
        String b2 = "Odeio acordar cedo!";
        boolean opcao = true;
        int numVezes = 5;
        char c = 'A';
        float f = 1.25f;
        double b = 1.25

        System.out.println(b1);
        for (int i=1; i<numVezes; ++i) {
            if (opcao)
                System.out.println(b2);
        }
        System.out.println("O valor de c é: " + c);
        System.out.println("O valor de f é: " + f);
        System.out.println("O valor de d é: " + d);
    }
}
```

```
> Bom dia!
> Odeio acordar cedo!
> Odeio acordar cedo!
> Odeio acordar cedo!
> Odeio acordar cedo!
> Odeio acordar cedo!
> O valor de c é: A
> O valor de c é: 1.25
> O valor de c é: 1.25
```

A Linguagem Java: Variáveis

A Linguagem Java: Variáveis

- Variáveis são espaços de memória capazes de armazenar um dado de um determinado tipo
- Em Java, as variáveis podem ser:
  - Variáveis de instância
  - Variáveis de classe
  - Variáveis locais
  - Parâmetros
- As variáveis tem escopo, ou seja, existem dentro de um determinado contexto

A Linguagem Java: Variáveis de Instância

- São os atributos dos objetos que não são estáticos, ou seja, que não são declarados utilizando a palavra-chave static
- São usados para caracterizar um objeto
  Ex.: o atributo cor de um carro
  O valor deste atributo pode ser diferente para cada carro criado

A Linguagem Java: Variáveis de Classe

- São os atributos da classe, declarados por meio da palavra-chave static
- São usadas para armazenar informações da classe que são comuns a todos os objetos
  Ex.: a variável que armazena o número de carros criados até o momento
  - O valor desta variável é o mesmo para todos os objetos que estiverem na memória uma vez que o mesmo não é copiado para todos
  - Esta variável é acessada por meio do nome da classe onde foi declardo
    X.varClasse;

A Linguagem Java: Variáveis Locais

- Uma variável local é aquela declarada dentro do escopo de um método
- Para variáveis locais, apenas o seu tipo precisa ser informado visto que a mesma possui visibilidade local restrita o método
- São as variáveis de uso geral do método, definidas internamente

```
public float celsiusToFahrenheit(float celsius)
{
    float conversao = (celsius * 1.8) + 32;
    return conversao;
}
```

A Linguagem Java: Parâmetros

- Os parâmetros são variáveis utilizadas para a passagem de argumentos de um método são variáveis que não têm a denominação de atributo

```
public void setNome(String n)
{
    nome = n;
}
```

A Linguagem Java: Operações

Java: Operadores Aritméticos

Operador	Uso	Descrição
+	a + b	Soma a e b
-	a - b	Subtrai b de a
-	-a	Nega aritmeticamente a
*	a * b	Multiplica a e b
/	a / b	Divide a por b
%	a % b	Retorna o resto da divisão de a por b (o operador de módulo)
++	a++	Incrementa a de 1; usa o valor de a antes de incrementar
++	++a	Incrementa a de 1; usa o valor de a depois de incrementar
–	a–	Decrementa a de 1; usa o valor de a antes de incrementar
–	–a	Decrementa a de 1; usa o valor de a depois de incrementar
+=	a += b	Abreviação de a = a + b
-=	a -= b	Abreviação de a = a - b
*=	a *= b	Abreviação de a = a * b
/=	a /= b	Abreviação de a = a / b
%=	a %= b	Abreviação de a = a % b

Java: Operadores Lógicos e Relacionais

Operador	Uso	Retorna true se...
>	a >b	a for maior que b
>=	a >= b	a é maior que ou igual a b
<	a <b	a é menor que b
<=	a <= b	a é menor que ou igual a b
==	a == b	a é igual a b
!=	a != b	a é diferente a b
&&	a && b	a e b são true. Condicionalmente avalia b (se a for false, b não será avaliado)
||	a || b	a ou b é true; condicionalmente avalia b (se a for true, b não será avaliado)
!	!a	a é false
&	a & b	a e b são true (para a e b booleanas); sempre avalia b. Se a e b numéricos, usado para operações bitwise.
|	*a|b*	a ou b for true (para a e b booleanas); sempre avalia b. Se a e b numéricos, usado para operações bitwise.
^	a ^ b	a e b são diferentes (para a e b booleanas); Se a e b numéricos, usado para operações bitwise.

Java: Operadores Lógicos e Relacionais

```
boolean i=true, j=false
if (i & j)
    System.out.println("Entrei no if: " + (i & j));
else:
    System.out.println("Entrei no if: " + (i & j));
```

```
> Entrei no else: false
```

```
int i=0x02, j=0x03
if (i & j)
    System.out.println("Entrei no if: " + (i & j));
else:
    System.out.println("Entrei no if: " + (i & j));
```

```
> Entrei no if: 2
```

```
byte i=0x02, j=0x03
System.out.println("Bitwise i & j: " + (i & j));
```

```
> Bitwise i & j: 2
```

Java: Instruções de Controle

- Instruções de desvio condicional
  if, if-else e switch
- Instruções de repetição
  while, do-while, for
- Instruções de desvio incondicional
  break e continue

A Linguagem Java: Instruções de Controle

Java: Instruções de Controle

Instrução de desvio condicional **if-else**

- Caso a condição1 seja verdadeira
  os comandos do Bloco 1 serão executados
- Caso a condição1 seja falsa
  - Caso a condição2 seja verdadeira
    os comandos do Bloco 2 serão executados
  - Caso a condição2 seja falsa
    os comandos do Bloco 3 serão executados
- Caso todas as condições anteriores falhem
  os comandos do Bloco 3 será executado

```
if (condicao1) {
    // Bloco 1
} else if (condicao2){
    // Bloco 2
} else {
    // Bloco 3
}
```

- Condição precisa estar entre ( )
- Caso haja mais de um comando no bloco, é preciso colocar os comandos entre
- O else final é opcional

Operador ternário

```
(condicao) ? fazSeVerdadeiro : fazSeFalso
```

- Caso a condição seja verdadeira
  Executa o comando na sequência da ?
- Caso a condição seja falsa
  Executa o comando na sequência do :

```
class Main{
    public static void main(String[] args){
        int x=0;
        int z=1;
        System.out.println("x antes: "+ x);
        System.out.println("z antes: "+ z);
        x = ((x>2) ? z++ : z--);
        System.out.println("x antes: "+ x);
        System.out.println("z antes: "+ z);
    }
}
```

```
> x antes: 0
> z antes: 1
> x depois: 1
> z depois: 0
```

Java: Instruções de Controle

```
class InstrucoesControleSwitch{
    public static void main(String[] args){
        int numDias=0;
        int mes=1;
        switch(mes) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 10:
            case 12:
                numDias = 31;
                break;
            case 4:
            case 9:
            case 11:
                numDias = 30;
                break;
            default:
                numDias = 28; // Não vamos tratar o ano bissexto
                break;
        }
        System.out.println("O mes " + mes + " tem " + numDias + "
             dias.");
    }
}
```

Instrução de desvio condicional **switch**

- fornece um conjunto de caminhos possíveis
- funciona com os tipos primitivos byte, short, char e int e com tipos enumerados, classe String, e classes empacotadoras Character, Byte, Short e Integer
- default é opcional

Instrução de repetição **while**

- Enquanto a condição for verdadeira, os comandos “comando1, comando2, ...” serão executados
- Note a importância de se incluir o controle da condição, em alguma parte do bloco (inclusive na própria condição!) O controle assegura a finalização do loop

```
while (condicao)
{
    comando1;
    comando2;
    ...
    controle;
}
```

Instrução de repetição **do-while**

- Enquanto a condição for verdadeira, os comandos "comando1, comando2, ..." serão executados
- Note a importância de se incluir o controle da condição, em alguma parte do bloco (inclusive na própria condição!) O controle assegura a finalização do loop

```
do
{
    comando1;
    comando2;
    ...
    controle;
}while (condicao);
```

Os exemplos abaixo produzem o mesmo resultado, mas não são equivalentes! Por quê?

```java
int a;
a=10;
while (a<14)
{
    System.out.println("a: " + a);
    a++;
}
```

```java
int a;
a=10;
do
{
    System.out.println("a: " + a);
    a++;
}while (a<14);
```

```
> a: 10
> a: 11
> a: 12
> a: 13
```

Os exemplos abaixo produzem o mesmo resultado, mas não são equivalentes! Por quê?

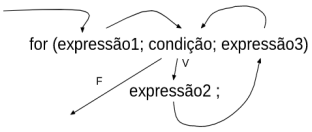
```java
int a;
a=14;
while (a<14)
{
    System.out.println("a: " + a);
    a++;
}
```

```
>
```

```java
int a;
a=14;
do
{
    System.out.println("a: " + a);
    a++;
}while (a<14);
```

```
> a: 14
```

Instrução de repetição **for**



- expressão1: uma expressão ou qualquer comando ou chamada de função. Normalmente uma atribuição;
- expressão2: qualquer comando ou chamada de função;
- expressão3: uma expressão ou qualquer comando ou chamada de função.
- Todas as expressões, assim como a condição, são opcionais.

Instrução de repetição **for**

```
for (int a=10; a<14; ++a)
    System.out.println("a: " + a)
```

```
> a: 10
> a: 11
> a: 12
> a: 13
```

```
int a;
a=10;
for (; a<14; ++a)
    System.out.println("a: " + a)
```

```
> a: 10
> a: 11
> a: 12
> a: 13
```

Instrução de desvio incondicional **break**

O break pode ser usado de duas formas:

- Sem rótulo:
  - Para terminar os laços for, while e do-while, além da instrução switch
  - Neste caso, a instrução break termina o laço mais próxima, ou seja, a instrução mais interna que contém o break
- Com rótulo:
  - Um break com rótulo pode terminar qualquer uma dessas instruções que o contém, mesmo a mais externa
  - Para isso, basta atribuir um rótulo à instrução que se deseja terminar e chamar o break com esse rótulo

Java: Instruções de Controle

```
class InstrucoesControleSwitch{
    public static void main(String[] args){
        int numDias=0;
        for (int mes=1; mes<13; ++mes){
            switch(mes) {
                case 1:
                case 3:
                case 5:
                case 7:
                case 10:
                case 12:
                    numDias = 31;
                    break;
                case 4:
                case 9:
                case 11:
                    numDias = 30;
                    break;
                default:
                    numDias = 28; // Não vamos tratar o ano bissexto
                    break;
            }
            System.out.println("O mes " + mes + " tem " + numDias + "
                 dias.");
        }
    }
}
```

Instrução de desvio incondicional **break**. Sem rótulo (apenas o switch foi encerrado):

```
> O mes 1 tem 31 dias.
> O mes 2 tem 28 dias.
> O mes 3 tem 31 dias.
> O mes 4 tem 30 dias.
> O mes 5 tem 31 dias.
> O mes 6 tem 30 dias.
> O mes 7 tem 31 dias.
> O mes 8 tem 31 dias.
> O mes 9 tem 30 dias.
> O mes 10 tem 31 dias.
> O mes 11 tem 30 dias.
> O mes 12 tem 31 dias.
```

Java: Instruções de Controle

Instrução de desvio incondicional **break**. Com rótulo:

```
class InstrucoesControleSwitch{
    public static void main(String[] args){
        int i=0, j=0;
        chega:
            for (i=0; i<10; ++i){
                for (j=0; j<10; ++j){
                    if (i*j > 6)
                        break chega;
                    else
                        System.out.println("i: " + i + " j: " + j)
                }
        }
        System.out.println("Acabei em i: " + i + " j: " + j);
    }
}
```

```
> i: 0 j: 0
> i: 0 j: 1
> i: 0 j: 2
> i: 0 j: 3
> i: 0 j: 4
> i: 0 j: 5
> i: 0 j: 6
> i: 0 j: 7
> i: 0 j: 8
> i: 0 j: 9
> i: 1 j: 0
> i: 1 j: 1
> i: 1 j: 2
> i: 1 j: 3
> i: 1 j: 4
> i: 1 j: 5
> i: 1 j: 6
> Acabei em i: 1 j: 7.
```

Instrução de desvio incondicional **continue**

O continue pode ser usado de duas formas:

- Sem rótulo:
  - Para pular a iteração corrente de laços for, while e do-while
  - Neste caso, a instrução continue pula a iteração corrente do laço mais próxima, ou seja, a instrução mais interna que o contém
- Com rótulo:
  - Um continue com rótulo pode pular qualquer uma das instruções que o contém, mesmo a mais externa
  - Para isso, basta atribuir um rótulo à instrução que se deseja pular e chamar o continue com esse rótulo

Instrução de desvio incondicional **break**. Com rótulo:

```
class InstrucoesControleSwitch{
    public static void main(String[] args){
        int i=0, j=0;
        chega:
            for (i=0; i<10; ++i){
                for (j=0; j<10; ++j){
                    if (i*j > 6)
                        break chega;
                    else
                        System.out.println("i: " + i + " j: " + j)
                }
        }
        System.out.println("Acabei em i: " + i + " j: " + j);
    }
}
```

```
> i: 0 j: 0
> i: 0 j: 1
> i: 0 j: 2
> i: 0 j: 3
> i: 0 j: 4
> i: 0 j: 5
> i: 0 j: 6
> i: 0 j: 7
> i: 0 j: 8
> i: 0 j: 9
> i: 1 j: 0
> i: 1 j: 1
> i: 1 j: 2
> i: 1 j: 3
> i: 1 j: 4
> i: 1 j: 5
> i: 1 j: 6
> Acabei em i: 1 j: 7.
```

Instrução de desvio incondicional **continue**. Com rótulo:

```
class InstrucoesControleSwitch{
    public static void main(String[] args){
        int i=0, j=0;
        pulaPar:
            for (i=0; i<5; ++i){
                for (j=0; j<5; ++j){
                    if (i%2 == 0)
                        continue;
                    else
                        System.out.println("i: " + i + " j: " + j)
                }
            }
        System.out.println("Acabei em i: " + i + " j: " + j);
    }
}
```

```
> i: 0 j: 0
> i: 0 j: 1
> i: 0 j: 2
> i: 0 j: 3
> i: 0 j: 4
> i: 3 j: 0
> i: 3 j: 1
> i: 3 j: 2
> i: 3 j: 3
> i: 3 j: 4
> Acabei em i: 5 j: 5.
```

Instrução de desvio incondicional **break**

Com rótulo:

```
class InstrucoesControleContinue {
    public static void main(String[] args) {
        int i=0, j=0;
        pulaIPar:
            for (i=0; i<5; ++i) {
                for (j=0; j<5; ++j) {
                    if (i % 2 == 0)
                        continue pulaIPar;
                    else
                        System.out.println("i: " + i + " j: " + j);
                }
            }
        System.out.println("Acabei em i: " + i + " j: " + j);
    }
}
```

```
i: 1 j: 0
i: 1 j: 1
i: 1 j: 2
i: 1 j: 3
i: 1 j: 4
i: 3 j: 0
i: 3 j: 1
i: 3 j: 2
i: 3 j: 3
i: 3 j: 4
Acabei em i: 5 j: 0
```

A Linguagem Java: Entrada e Saída de dados

Java: Entrada de dados

O pacote java.util da API de Java possui uma classe para leitura de dados via entrada padrão (e outros mecanismos): a classe Scanner

- Primeiro, é preciso importar esta classe através de:
  import java.util.Scanner;
- Antes de realizar a leitura dos dados, precisamos informar a fonte dos dados de leitura, no nosso caso, a entrada padrão (System.in)
  Scanner entrada = new Scanner (System.in);
- A partir deste ponto, podemos ler dados via a variável entrada

Java: Entrada de dados

```
class EntradaDados{
    public static void main(String[] args){
        int i, leitura;
        Scanner entrada = new Scanner(System.in);
        for (i=0; i<5; ++i){
            System.out.print("Informe um numero inteiro: ")
            leitura = entrada.nextInt();
            System.out.printf("Acabei de ler %d\n", leitura);
        }
    }
}
```

```
> Informe um numero inteiro: 10
 Acabei de ler 10
> Informe um numero inteiro: 20
 Acabei de ler 20
> Informe um numero inteiro: 30
 Acabei de ler 30
> Informe um numero inteiro: 40
 Acabei de ler 40
> Informe um numero inteiro: 50
 Acabei de ler 50
```

Java: Saída de dados

```
class EntradaDados{
    public static void main(String[] args){
        int i, leitura;
        Scanner entrada = new Scanner(System.in);
        for (i=0; i<5; ++i){
            System.out.print("Informe um numero inteiro: ")
            leitura = entrada.nextInt();
            System.out.printf("Acabei de ler %d\n", leitura);
        }
    }
}
```

```
> Informe um numero inteiro: 10
Acabei de ler 10
> Informe um numero inteiro: 20
Acabei de ler 20
> Informe um numero inteiro: 30
Acabei de ler 30
> Informe um numero inteiro: 40
Acabei de ler 40
> Informe um numero inteiro: 50
Acabei de ler 50
```

Para imprimir na saída padrão, podemos usar:

- System.out.print (imprime String)
- System.out.println (imprime String e pula linha)
- System.out.printf (imprime de acordo com a formatação indicada)

Atividades extra-classe

Exercícios:

- Construa um programa que receba 10 valores informados pelo usuário e os apresente na tela
- Construa um programa que apresente os n primeiros números primos, onde n é um valor informado pelo usuário
- Quais as fases de construção de um programa em Java? O que elas implicam na qualidade/desempenho do código final? Compare com o funcionamento de C.
- Qual a diferença entre argumento e parâmetro?

MC322 - Programação orientada a objetos
Aula 2
Introdução à Java

Prof. Marcos M. Raimundo
Instituto de Computação - UNICAMP