Universidade de São Paulo
Brasil

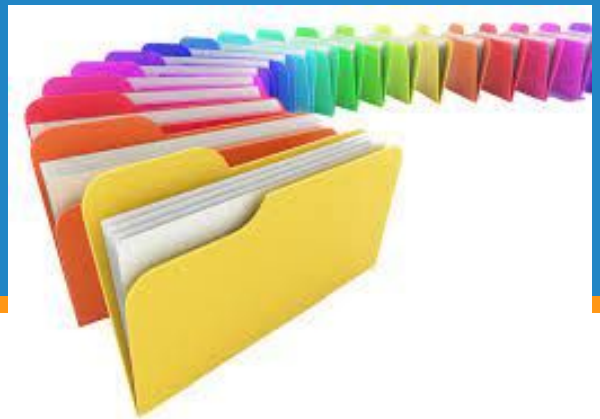
Aula 02 – Instruções e Operandos Java

MAC0321 - Laboratório de Programação Orientada a Objetos

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

Departamento de Ciência da Computação
Instituto de Matemática e Estatística

1. Como escrever os meus primeiros programas?



Variáveis

- Java é uma linguagem OO
  - Todas as variáveis são objetos
    - Objetos devem ser criados durante a inicialização
- Também existem tipos primitivos
  - São variáveis tradicionais

Tipos primitivos

- As variáveis **devem** ser inicializadas
  - Caso contrário, erro de compilação!
    - Em outras linguagem erros por variáveis não inicializadas são difíceis de encontrar !!
- Para cada um dos tipos primitivos existe um objeto correspondente
  - Ex: Para o int a classe é a Integer
    - Contém apenas um campo, inteiro, e métodos

Tipos primitivos

boolean	True ou false	
char	16 bits	
byte	8 bits	-128 a 127
short	16 bits	-2^{15} a $2^{15}-1$
int	32 bits	-2^{31} a $2^{31}-1$
long	64 bits	-2^{63} a $2^{63}-1$
float	32 bits	IEEE 754
double	64 bits	IEEE 754

Discussão: como as variáveis são armazenadas na memória?

Objeto Integer

```
java.lang
 Class Integer

java.lang.Object
    |+--java.lang.Number
          | +--java.lang.Integer

 All Implemented Interfaces: Comparable, Serializable
```

Alguns métodos:

`byte byteValue()` Returns the value of this Integer as a byte.

`int compareTo(Integer anotherInteger)`

Compares two Integers numerically.

`static int parseInt(String s)`

Parses the string argument as a signed decimal integer.

Outros objetos

- BigInteger
  - Números inteiros com precisão arbitrária
- BigDecimal
  - Números com uma quantidade fixa de dígitos após a vírgula e precisão arbitrária
    - Útil para uso monetário
    - Em float ou double, 0.3 é uma dízima!

Operadores em Java

- Além dos operadores convencionais:

  +, -, *, /, =, == (igualdade) e != (diferença)

  Operadores compostos: +=, *=, -= e /=

  `a=a+2;` é equivalente a `a+=2;`

  Operadores de incremento e decremento: ++ e --

  `a=a+1;` é equivalente a `a++;`

  O operador + também se aplica a objetos String

```
String nome;
nome = "Joaquim" + " Cruz";
```

Em Java se usa bem menos sobrecarga que em C++

Dicas operadores

- Existe o operador para cálculo do resto da divisão:
  - O operador % deve ser usado com tipos inteiros
    - Com números reais o resultado é “estranho”
- Use parênteses quando necessário:
  - Qual o valor de 5*4/2*2 ? 5*(4/2)*2 ou (5*4)/(2*2).
- Para guardar números reais use sempre `double`
  - `Com reais use sempre ponto: 5.0`
  - `Float é para poupar memória!`

Operações com objetos

O que é impresso no programa seguinte ?

```
class Number { int i;}

public class Assignment {
  public static void main(String[] args) {
    Number n1 = new Number();
    Number n2 = new Number();
    n1.i = 9;
    n2.i = 47;
    System.out.println("1: n1.i: "+n1.i+", n2.i: "+n2.i);
    n1 = n2;
    System.out.println("2: n1.i: "+n1.i+", n2.i: "+n2.i);
    n1.i = 27;
    System.out.println("3: n1.i: "+n1.i+", n2.i: "+n2.i);
  }
} ///:~
```

Operações com objetos

O que é impresso no programa seguinte ?

```
public class Equivalence {
  public static void main(String[] args) {
    Integer n1 = new Integer(47);
    Integer n2 = new Integer(47);
    System.out.println(n1 == n2);
    System.out.println(n1 != n2);
  }
}
```

Como resolver isto ?

Operações com objetos

Método presente em todos os objetos

- `equals`

```
//: EqualsMethod.java

public class EqualsMethod {
  public static void main(String[] args) {
    Integer n1 = new Integer(47);
    Integer n2 = new Integer(47);
    System.out.println(n1.equals(n2));
  }
} ///:~
```

Operações com objetos

Mas atenção ao comportamento padrão do método `equals`

```
class Value {
  int i;
}

public class EqualsMethod2 {
  public static void main(String[] args) {
    Value v1 = new Value();
    Value v2 = new Value();
    v1.i = v2.i = 100;
    System.out.println(v1.equals(v2));
  }
}
```

Mudança de tipos

Uma variável (ou objeto) pode ter seu tipo mudado usando-se cast.

```
int a;
double x = 2.5;
a = (int) x;
```

Quando uma variável (objeto) é atribuido a algo mais genérico, o cast não é necessário

```
byte b = 10;
double z;
z = b;
```

Comparações compostas

- condição1 `&&` condição2
  - verdadeiro apenas se as duas condições são true
- condição1 `||` condição2
  - verdadeiro se pelo menos uma das condições é true
- `!` condição
  - verdadeiro se condição == false, e falso caso contrário

Dica: short-circuit

ex: `(x>0)&&(1/x<2)` // a segunda condição só é // testada quando x>0

Controle da execução

- if – else pode ser usado de duas formas:

```
if (expressão)
   statement
```

ou

```
if (expressão1)
   statement1
else
   statement2
```

(expressão) = expressão booleana
statement = instrução;

ou

```
{
    instruçao1;
    instrução2;

    instrução n;
}
```

Controle de execução

- keyword **return**
  - serve para retornar algo por uma função ou método
  - provoca também a saída da função ou método

Controle de execução

- keyword while

```
while (expressão)
   statement
```

- keyword for

```
for (inicialização; expressão; passo)
   Statement

for( tipo var: vars)
  Statement
```

Arrays

- Declaração:

```
int[] listaDeInteiros;
int listaDeInteiros[];
```

- listaDeInteiros.length

Arrays (cont)

- Criação e inicialização:

```
int[] listaDeInteiros = new int[10]; //valores default
```

Arrays (cont)

- Criação e inicialização:

```
int[] listaDeInteiros = new int[10]; //valores default
int[] listaDeInteiros = new int[] {1,2,3,4,5}; /* tamanho implícito */
```

Arrays (cont)

- Criação e inicialização:

```
int[] listaDeInteiros = new int[10]; //valores default
int[] listaDeInteiros = new int[] {1,2,3,4,5}; /* tamanho
                                                  implícito */
int[] listaDeInteiros = {1,2,3,4,5}; /* tipo e tamanho
                                        implícitos  -
                    desta forma só junto com a declaração */
```

Arrays (cont)

- Criação e inicialização:

```
int[] listaDeInteiros = new int[10]; //valores default
int[] listaDeInteiros = new int[] {1,2,3,4,5}; /* tamanho
                                    implícito */
int[] listaDeInteiros = {1,2,3,4,5}; /* tipo e tamanho
                        implícitos  -
        desta forma só junto com a declaração */
Point[] pontos =    {circulo1.getCenterPoint(),
                        circulo2. getCenterPoint()}
      //inicialização em tempo de execução
```

Arrays (cont)

- Uso:

```
public static void main(String[] args)
{
    for (int i = 0; i < args.length; i++)
        System.out.println(args[i]);
}
```

- IndexOutOfBoundsException

Arrays multidimensionais

```
int[][] tabuada = {     {0,0,0,0,0,0},
                        {0,1,2,3,4,5},
                        {0,2,4,6,8,10},
                        {0,3,6,9,12,15},
                        {0,4,8,12,16,20},
                        {0,5,10,15,20,25}
                       };

int produto = tabuada[3][5];
```

Como funciona a memória ?

- Objetos
  - Alguma semelhança com ponteiros?
  - Passagem sempre por referência
- Arrays

Dicas de programação

- Identação
- Nomes de variáveis e métodos
- Comentários
- Code Smells

Lista de exercícios

No computador com o Eclipse

Entrega até o final do dia

MAC321
Lab POO

- Professor: Marcelo Finger

  E-mail: mfinger@ime.usp.br