

MC322 – Programação Orientada a Objetos

Laboratório 01 – 1s2020

Leonardo Montecchi (Professor)
leonardo@ic.unicamp.br

Iury Cleveston (PED)
iurycl@gmail.com

Matheus Rotta (PAD)
matheusrotta7@gmail.com

Matheus Mazon (PAD)
matheusmazon@outlook.com

10/03/2020

1 Objetivos

O objetivo deste laboratório é introduzir a linguagem Java apresentando alguns conceitos básicos (entrada, saída e compilação) e de apresentar a plataforma **Eclipse** (ambiente para desenvolvimento Java) e suas funcionalidades. Não se preocupe, por enquanto, em aplicar o paradigma de programação a objetos.

2 Introdução

2.1 Primeiro Programa - Hello World

O código de um Hello World em Java é ilustrado no Algoritmo 1. A primeira linha do código define a classe *HelloWorld*. Já na linha 2, o método *main* é definido e é o ponto inicial de todo programa Java. Note que o método *main* sempre deve ser declarado da mesma forma que foi ilustrado no exemplo.

```
1 public class HelloWorld {  
2     public static void main (String args[]) {  
3         System.out.println("Hello World!");  
4     }  
5 }
```

Algoritmo 1: Hello World em Java.

Quatro características do método *main* devem ser destacadas:

1. O vetor de *String* no argumento representa os parâmetros de entrada que foram utilizados ao executar o programa;
2. O método não retorna nenhum tipo de valor (*void*);
3. O método é estático (*static*);
4. O método é público (*public*).

Também é importante mencionar que todo método deve estar localizado no escopo de alguma classe (no exemplo, o método *main* está no escopo da classe *HelloWorld*).

Por enquanto, não fique preocupado em entender os conceitos das palavras-chaves *class*, *public* e *static* pois serão discutidas em aulas/laboratórios futuros.

2.2 Compilação

Para compilar o código de Hello World primeiro abra um editor de texto qualquer e salve o código como **HelloWorld.java**. Após isso, abra o terminal e navegue até a localização do código e execute o comando:

```
1 javac HelloWorld.java
```

Tal comando irá transformar o código-fonte em um código de *byte-code* (instruções interpretáveis pela máquina virtual do Java) armazenado no arquivo **HelloWorld.class**. Finalmente, para executar o programa basta utilizar o comando:

```
1 java HelloWorld
```

2.3 Entrada e Saída

Existem diversas formas de realizar uma leitura em Java, uma delas é criando um objeto do tipo *Scanner* conforme pode ser notado na linha 6 do Algoritmo 2. Note que é necessário importar (linha 1) a classe *Scanner*. Neste laboratório não será explicado detalhes sobre a classe *Scanner*. No momento, basta saber que ao invocar os métodos *nextInt*, *nextFloat*, *nextDouble*, etc, será requisitado uma leitura a partir do teclado.

```
1 import java.util.Scanner;
2
3 public class EntradaSaida {
4
5     public static void main(String[] args) {
6         Scanner entrada = new Scanner(System.in);
7         System.out.print("Digite um numero: ");
8         int num = entrada.nextInt();
9         System.out.println("Numero: " + num);
10        System.out.printf("Numero: %d \n", num);
11    }
12 }
```

Algoritmo 2: Exemplo de entrada e saída em Java.

Para imprimir algo na tela, os seguintes métodos podem ser utilizados:

1. **System.out.print**: Imprime o valor na tela;
2. **System.out.println**: Imprime o valor na tela com quebra de linha no final;
3. **System.out.printf**: Imprime o valor na tela com formatação similar a linguagem C.

Note que a concatenação de *String* em java é feita utilizando o operador + (linha 9).

2.4 Controle de Fluxo

Laços em Java podem ser feitos utilizando as palavra-chaves *for*, *while* ou *do-while*. Enquanto que estruturas de decisões podem ser feitas utilizando as palavra-chaves *if* ou *switch*. As sintaxes de cada um dos comandos são ilustradas nos Algoritmos 3 e 4.

```
1 for(int i = 0; i < n; i++) {  
2     ...  
3 }  
4 while(i < n) {  
5     ...  
6 }  
7 do{  
8     ...  
9 }while(i < n);
```

Algoritmo 3: Estruturas de repetição em Java.

```
1 if(i < n) {  
2     ...  
3 } else {  
4     ...  
5 }  
6  
7 switch(i) {  
8     case 1:  
9         ...  
10        break;  
11     case 2:  
12        ...  
13        break;  
14     default:  
15        ...  
16 }
```

Algoritmo 4: Estruturas de decisão em Java.

2.5 Plataforma Eclipse

Para facilitar o desenvolvimento, pode-se utilizar a IDE Eclipse. Nos laboratórios do IC, a plataforma Eclipse para desenvolvimento Java encontra-se com o nome **eclipse-jee**.

Primeiro, deve ser criado um projeto Java como a seguir:

1. File > New > Java Project (ou File > New > Project e selecione Java Project).
2. Digite o nome do projeto.
3. Na aba JRE, selecione a versão JavaSE-1.8.

4. Finalize a criação (*Finish*). Caso seja solicitado a criação de um *module-info*, não crie (é uma funcionalidade nova incorporada no Java 9, mas que não é conteúdo deste laboratório).

Para criação de classes, os seguintes passos podem ser efetuados:

1. Clique com o botão direito sobre o projeto na aba do *Package Explorer* e escolha New > Class.
2. Digite o nome da classe e clique em finalizar (os conceitos das opções que aparecem no menu de criação de classe serão abordadas futuramente).

2.6 Utilizando o Debug do Eclipse

A função de **Debug** é uma ótima opção para examinar a execução do código passo a passo, sendo possível visualizar o estado atual do código (e.g., valores de variáveis).

Para isso, primeiro é necessário inserir *breakpoints* no local onde você gostaria de verificar a execução passo a passo. Para inserir os *breakpoints*, basta clicar duas vezes com o botão esquerdo do mouse do lado esquerdo dos números das linhas, ou clicar com o botão direito na linha para abrir o menu conforme ilustrado na Figura 1.

Agora, basta clicar no ícone *bug* (inseto) localizado no menu superior. No momento em que a execução do código chegar em um *breakpoint*, você poderá continuar a execução do código passo a passo (entrando em métodos ou pulando algumas partes do código).

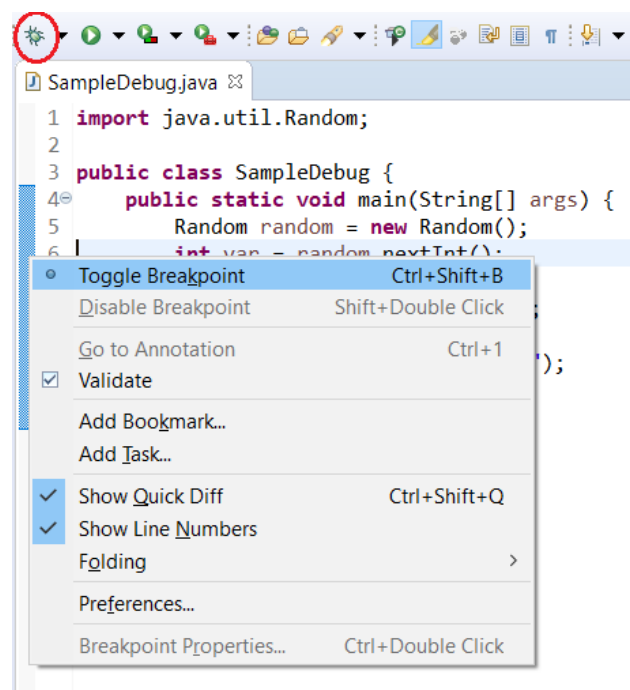
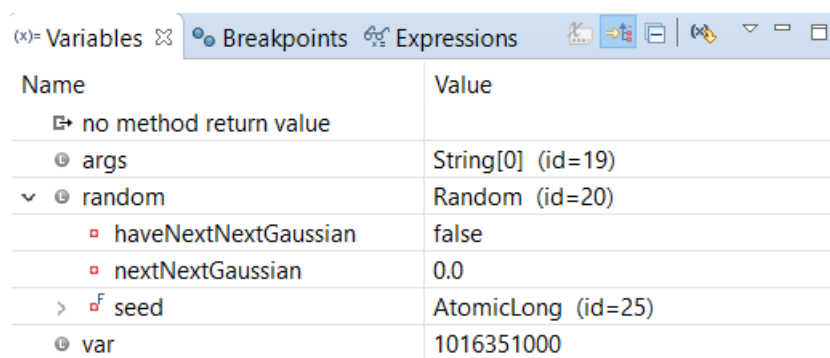


Figura 1: Inserindo um *breakpoint* no código.

Ao executar o Debug, você poderá consultar as informações detalhadas das variáveis e também criar expressões que podem auxiliar no teste. Tais funcionalidades podem ser visualizadas no painel conforme ilustrado na Figura 2.



Name	Value
no method return value	
args	String[0] (id=19)
random	Random (id=20)
haveNextNextGaussian	false
nextNextGaussian	0.0
seed	AtomicLong (id=25)
var	1016351000

Figura 2: Painel de visualização das informações do Debug.

Por último, os principais comandos ao executar o Debug são:

- Step Into (F5): Executa a linha do código redirecionando para o escopo do método se houver um;
- Step Over (F6): Executa a linha do código, mas não redireciona para o método se houver um;
- Step Return (F7): Retorna um escopo acima do atual;
- Resume (F8): Executa todo o programa;
- Run To Line (Ctrl + R): Move a execução do código para aonde o cursor do mouse está localizado.

2.7 Boas Práticas em Java

A medida do possível sempre que conceitos novos forem apresentados também serão apresentadas boas práticas em relação a tais conceitos. Portanto, até o momento as seguintes convenções devem ser aplicadas:

1. A indentação padrão de Java segue conforme foi ilustrado no exemplo (abertura de chave na mesma linha da declaração e fechamento de chave alinhada com a linha de declaração;
2. O nome do arquivo fonte deve ser sempre igual ao nome da classe que representa tal arquivo (no exemplo, a classe foi nomeada de **HelloWorld** e, portanto, o arquivo também deve ser nomeado como **HelloWorld**).

2.8 Documentação do Java

A documentação oficial do Java pode ser encontrada no site da Oracle, em:

<https://docs.oracle.com/en/java/javase/13/docs/api/index.html>

Um tutorial sobre conceitos fundamentais pode ser encontrado em: <https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html>

3 Exercícios

3.1 Desenvolvendo uma Calculadora em Java

Desenvolva uma calculadora que tenha como entrada, via terminal, um número representando a operação desejada pelo usuário; os demais números necessários para o cálculo. O menu deverá fornecer as seguintes opções ao usuário:

1. Digite 1 para somar;
2. Digite 2 para subtrair;
3. Digite 3 para multiplicar;
4. Digite 4 para dividir;
5. Digite 5 para calcular fatorial;
6. Digite 6 para verificar se um número é primo;
7. Qualquer outro valor para sair do programa.

Sempre que uma função for executada, é necessário perguntar ao usuário se é desejado executar outra função.

3.2 Conserte os erros utilizando a função Debug

Utilize a função **Debug** presente no Eclipse para acompanhar a execução do programa abaixo e conserte os erros necessários para a correta ordenação dos elementos no vetor. Existe um total de **5 erros** presentes no código abaixo. Ao final, o algoritmo deveria imprimir na tela 10 números ordenados em ordem crescente.

```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4  
5         int quantity = 10;  
6         int[] vector = new int[quantity];  
7  
8         for (int i = 0; i < vector.length - 1; i++) {  
9             vector[i] = (int) (Math.random()*100);  
10        }  
11  
12        sort(vector);  
}
```

```

13
14     for (int i = 1; i < vector.length; i++) {
15         System.out.println(vector[i]);
16     }
17
18 }
19
20 private static void sort(int vector[]){
21
22     boolean switched = true;
23     int aux;
24     while (switched) {
25         switched = false;
26         for (int i = 0; i < vector.length + 1; i++) {
27             if (vector[i] > vector[i + 1]) {
28                 aux = vector[i];
29                 vector[i] = vector[i + 1];
30                 vector[i - 1] = aux;
31                 switched = false;
32             }
33         }
34     }
35 }
36
37 }

```

Algoritmo 5: Algoritmo de ordenação com 5 erros.

4 Submissão

Submeta o trabalho no link de entrega na página do Classroom da disciplina, em formato de arquivo compactado (zip). Envie o arquivo com o nome **{ra}_Lab01.zip**. Arquivos a serem submetidos:

- Código fonte da calculadora;
- Código fonte do algoritmo de ordenação corrigido.

Este laboratório **não vale** nota.