MC322 – Programação Orientada a Objetos
Laboratório 01 – 2s2021

Leonardo Montecchi (Professor) Lucas Borges Rondon Thales Eduardo Nazatto
Ângelo Renato Pazin Malaguti Leonardo de Sousas Rodrigues

Para perguntas ou dúvidas usem o Discord (`https://discord.gg/Xzpz9epNTG`)

1 Objetivos e submissão

Objetivos

- Introduzir a linguagem Java apresentando alguns conceitos básicos (entrada, saída e compilação).
- Apresentar a plataforma **Eclipse** (ambiente para desenvolvimento Java) e suas funcionalidades.

Data de entrega

- **22/08/2021** até às **23h59**.

Submissão

- Ao criar o projeto Java, selecionar a versão **JavaSE-11** do JRE, ou "Use default JRE".
- IMPORTANTE: Nomear o projeto na forma **RA_Lab01** e o pacote base na forma **com.unicamp.mc322.lab01**. Substitua RA com o seu *Registro Acadêmico* (matrícula).
- Submeta o trabalho no link de entrega na página do Classroom da disciplina, em formato de arquivo compactado (zip) com o nome **RA_Lab01.zip**. Substitua RA com o seu *Registro Acadêmico* (matrícula).
- Arquivos importantes a serem submetidos no projeto:
  - Código fonte da calculadora.
  - Código fonte do algoritmo de ordenação corrigido.
- Não enviar o código compilado (*.class)

Critérios de avaliação

- Este laboratório **não vale** nota.

2 Exercícios

2.1 Desenvolvendo uma Cálculadora em Java

Desenvolva uma calculadora que tenha como entrada, via terminal, um número representando a operação desejada pelo usuário e 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.

2.2 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** lógicos presentes no código abaixo. Ao final, o algoritmo devera imprimir na tela 10 números ordenados em ordem crescente.

```
package com.unicamp.mc322.lab01;

public class Algorithm {

    public static void main(String[] args) {

        int quantity = 10;
        int[] vector = new int[quantity];

        for (int i = 0; i < vector.length - 1; i++) {
            vector[i] = (int) (Math.random()*100);
        }

        sort(vector);

        for (int i = 1; i < vector.length; i++) {
            System.out.println(vector[i]);
        }

    }

    private static void sort(int[] vector){

        boolean switched = true;
        int aux;
        while (switched) {
            switched = false;
            for (int i = 0; i < vector.length + 1; i++) {
                if (vector[i] > vector[i + 1]) {
                  aux = vector[i];
                  vector[i] = vector[i + 1];
                  vector[i - 1] = aux;
                  switched = false;
                }
            }
        }
    }

}
```

Código Fonte 1: Algoritmo de ordenação com 5 erros.

3 Documentação de ajuda

3.1 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 uma abertura de chave na mesma linha da declaração e fechamento de chave alinhada com a linha de declaração (Código Fonte 2);
2. O nome do arquivo fonte deve ser sempre igual ao nome da classe que representa tal arquivo (no Código Fonte 2, a classe foi nomeada de **HelloWorld** e, portanto, o arquivo também deve ser nomeado como **HelloWorld**).

3.2 Primeiro programa - Hello World

O código de um Hello World em Java é ilustrado no Código Fonte 2. A primeira linha define o pacote *com.unicamp.mc322.lab01* o qual serve para organizar um conjunto de classes. A terceira linha do código define a classe *HelloWorld*. Já na linha 5, 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 package com.unicamp.mc322.lab01;
2
3 public class HelloWorld {
4
5     public static void main (String[] args) {
6         System.out.println("Hello World!");
7     }
8
9 }
```

Código Fonte 2: 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 Código Fonte 2, 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.

3.3 Compilação no terminal

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 `javac`

```
# Command: javac
# Usage: javac <options> <source files>
user@cpu:/path-to-your-code$ 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 `java`

```
# Command: java
# Usage: java [options] <mainclass> [args...] (to execute a class)
user@cpu:/path-to-your-code$ java HelloWorld
```

3.4 Entrada e saída no terminal

Existem diversas formas de realizar uma leitura em Java, uma delas é criando um objeto do tipo *Scanner* conforme pode ser notado na linha 8 do Código Fonte 3. Note que é necessário importar (linha 3) 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  package com.unicamp.mc322.lab01;
2
3  import java.util.Scanner;
4
5  public class InputOutput {
6
7      public static void main(String[] args) {
8          Scanner input = new Scanner(System.in);
9          System.out.print("Digite um numero: ");
10         int num = input.nextInt();
11         System.out.println("Numero: " + num);
12         System.out.printf("Numero: %d \n", num);
13     }
14 }
```

Código Fonte 3: 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 11).

3.5 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 Códigos Fontes 4 e 5.

```java
for(int i = 0; i < n; i++) {
  ...
}
while(i < n) {
  ...
}
do{
 ...
}while(i < n);
```

Código Fonte 4: Estruturas de repetição em Java.

```java
if(i < n) {
  ...
} else {
  ...
}

switch(i) {
  case 1:
    ...
    break;
  case 2:
    ...
    break;
  default:
    ...
}
```

Código Fonte 5: Estruturas de decisão em Java.

3.6 Plataforma Eclipse

Eclipse é uma IDE para facilitar o desenvolvimento de código Java, porém suporta várias outras linguagens a partir de plugins como C/C++, PHP, Python e Kotlin. Eclipse não e a única IDE com suporte para Java, também temos outros IDEs (Intellij IDEA, NetBeans e Visual Studio Code) com as mesmas ou similares características do que o Eclipse.

Instalação

- Pode usar o Eclipse Installer para instalar uma versão do Eclipse.
  - `https://www.eclipse.org/downloads/packages/installer`

Criação do Projeto

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-11** ou "Use default JRE".
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).

Criação dos Pacotes

1. Clique com o botão direito sobre o projeto na aba do *Package Explorer* e escolha New > Package.
2. Digite o nome do pacote.

Criação das Classes

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

3.7 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).

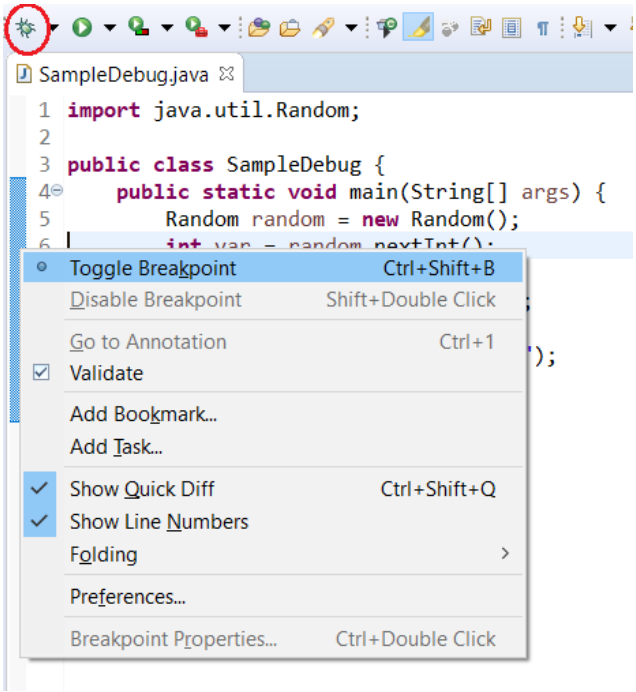

Figura 1: Inserindo um *breakpoint* no código.

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.

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

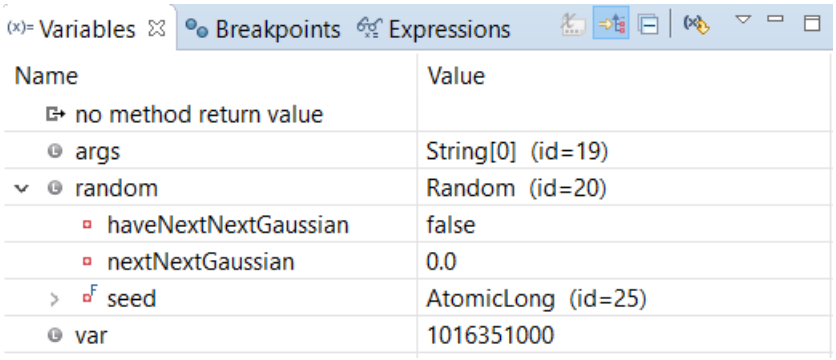

Figura 2: Painel de visualização das informações do Debug.

- 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.

É importante lembrar que às vezes os atalhos no Eclipse mudam de acordo com o computador, sistema operacional, configuração do teclado o versão do Eclipse. A seção Keys do Eclipse mostra os atalhos no Eclipse (Windows > Preferences > Keys)

3.8 Guia de instalação do Java

- Instalar o **Java SE Development Kit (JDK) 11**
  - `https://www.oracle.com/lad/java/technologies/javase-jdk11-downloads.html`
  - Tipicamente, as distribuições Linux fornecem uma versão open source da JDK entre os pacotes. No Ubuntu/Debian instalar o pacote `openjdk-11-jdk`.
- Guia de instalação do JDK no Ubuntu
  - `https://www.digitalocean.com/community/tutorials/how-to-install-java-with-apt-on-ubuntu-20-04-pt`
- Guia de instalação do JDK no Mac
  - `https://java.tutorials24x7.com/blog/how-to-install-java-11-on-mac`
- Guia de instalação do JDK no Windows
  - `https://java.tutorials24x7.com/blog/how-to-install-java-11-on-windows`

3.9 Documentação do Java

- A documentação oficial do Java pode ser encontrada no site da Oracle
  - `https://docs.oracle.com/en/java/javase/11/docs/api/index.html`
- Um tutorial sobre conceitos fundamentais pode ser encontrado em:
  - `https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html`