JAVA TECHNOLOGY

(503111)

LAB 1

Lab Objectives: Review the knowledge of the Java programming language learned in previous courses such as basic syntax, object-oriented programming, how to compile and run programs using the command line, using libraries externally, manage libraries with Maven tools, use new APIs in java such as Lambda Expression, Stream API, etc.

EXERCISE 1

An exercise on how to compile, run programs using the command line and how to read parameters passed from outside environment to the program through the command line.

Requirements: Write a program that takes a mathematical expression from the argument passed from the command line, then computes and prints the result of that expression. Assuming the program is written in a file named Program.java, some examples of commands and outputs printed on the console are as follows:

- When typing "`java Program 15 / 2`", the program outputs: **7.5**
- When typing "`java Program 2 x 8`", the program outputs: **16** (don't use * for multiplication because java will handle the * symbol differently).
- When typing "`java Program 2 ^ 3`", the program outputs: **8**
- When typing "`java Program 2 # 3`", the program outputs: **Unsupported operator**
- When typing "`java Program 2`" or "`java Program 32 + 8`", the program outputs: **Invalid expression**

Other constraints you need to follow:

- The program needs to be compiled and run from the command line (using javac and java) on the Windows Command Prompt or the macOS/Linux Terminal.
- Do not create packages in this exercise (if you don't know what a package is, that's fine)
- Expressions needs to be passed from outside the program through command line arguments. Do not read data from the keyboard through the Scanner, do not assign values to expressions directly from the source code.
- The output printed to the console must have the format shown above.
- The program only supports 5 operations including: addition (+), subtraction (-), multiplication (x), division (/) and exponentiation (^).

Instructions:

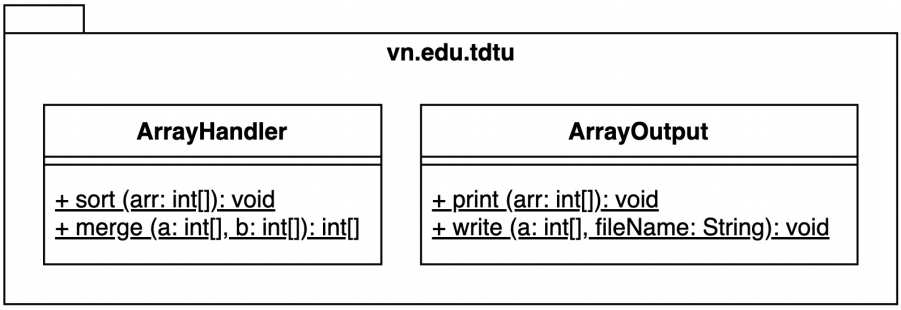
- Use `javac <filename.java>` command to compile java files into *.class files, or use `javac *.java` command to compile all java files into class files.
- Use `java <classname>` command to run the program where `<classname>` is the name of the class containing the main method.
- When using the "java Program 32 + 8" command, we are implicitly understanding that the main method is written in the Program.java file, the main method has the `main(String args[])` syntax. Where `args[]` is a String array of 3 elements, equivalent to the elements passed in from the command line argument, separated by spaces: `{"32", "+", "8"}`.
- Use `Integer.parseInt()`, `Double.parseInt()` and try-catch statement to handle input.

What to submit to the e-learning: A file named **Program.java**

EXERCISE 2

An exercise on how to use external libraries as *jar files; compile and run the program using the command line when using an external library; learn the concept of class path and the environment variable CLASSPATH.

Requirements: Given the file **lib.jar** is a library containing one-dimensional array processing functions that have been compiled and packaged into a jar file. Descriptions of this library can be summarized in the class diagram below:



This library can be understood as follows: There are two classes named ArrayHandler and ArrayOutput placed in the package named vn.tdtu.edu. The methods in both classes are static methods. The sort() method takes an int array as an argument and returns void. This method sorts the elements in the array in ascending order.

Write a program (Program.java) that uses the api provided in the library to perform operations on a one-dimensional array, specifically as follows:

- Initialize two arbitrary one-dimensional arrays and name the variables **a** and **b**.
- Print these two arrays to console (using the print() method in the ArrayOutput class).
- Merge these two arrays and assign the result to array c then print array c to the console.
- Sort the array c in ascending order then print the result to the console.
- Write the contents of the array c to the file output.txt

Other constraints you need to follow:

- Do not define the array handling functions (sort, merge, print, write), you must use the functions provided in the lib.jar library.
- Compile and run the program using the command line. Do not use tools like Eclipse, Intellij Idea to do this exercise.

Instructions:

- The source code of your program needs to use two classes provided in the library, so a suitable import statement is required: `import vn.edu.tdtu.*;` or `import vn.edu.tdtu.ArrayHandler` and `import vn.edu.tdtu.ArrayOutput;`

```
1  import vn.edu.tdtu.ArrayOutput; // đưa thư viện vào sử dụng
2
3  public class Program
4  {
       Run | Debug
5      public static void main(String[] args) {
6
7          int a[] = {12,2,34,5,6};
8          ArrayOutput.print(a); // gọi chức năng print() để in mảng
9
```

Example of using the print() function in ArrayHandler to print an array. When coding, the IDE may display the error message like "ArrayOutput cannot be resolved", this is normal because the IDE has not set up the correct classpath to the library. As long as we set a valid classpath when we compile and run the program, this error will be solved.

- When compiling and running the program, we still use the same java and javac commands as before, but now we need to set the classpath parameter to determine where *.class files and *.jar files are saved.
- For example, we use `javac -cp ".;./lib.jar" Program.java` to compile the program and use the command `java -cp ".;./lib.jar" Program` to run the program. Where `-cp` stands for *classpath*, this parameter is used to set the class path location. `".;./lib.jar"` is a list of one or more directory paths of the classpath where class or jar files are stored, which can be either absolute or relative path. If there are multiple paths, they must be separated by a **;** on Windows or by the **:** on macOS and Linux.

- In the above example, the `.` represents the path of the current working directory, and `./lib.jar` represents the lib.jar file located inside the current working directory.

What to submit to the e-learning: the files named **Program.java** and **lib.jar**, they should be placed in the same directory**.**

EXERCISE 3

An exercise on new APIs in java 8 such as Lambda Expression, Method Reference, Stream API, Map, Recude, Filter, etc.

Requirements: Based on the source code given in the **ex3.zip** file, modify or write additional code at the locations marked with @TODO to complete the exercise. The given source code includes the following files:

- **Student.java**: file containing Student class definition, **do not** modify this file.
- **StudentUtils.java**: modify/add code at locations marked with @TODO.
- **Program.java**: modify/add code at locations marked with @TODO.

```
/**
 * @TODO
 * Chuyển đổi cách viết sử dụng new Comparator... sang sử dụng Lambda Expression
 */
public static void sortByName(List<Student> list) {
    Collections.sort(list, new Comparator<Student>() {  //  <--- thay đổi bằng lambda expression
        @Override
        public int compare(Student o1, Student o2) {
            return o1.name.compareTo(o2.name);
        }
    });
}
```

An example of the requirements of the exercise

In the code above, the sortByName() method is a method that sorts the list of students by name in ascending order. This method is implemented in the usual way, using the object created from the Comparator interface. With this todo requirement, you need to modify the source code, replacing it with a lambda expression to maintain the same functionality but make the source code more concise.

What to submit to the e-learning: files named **Program.java** and **StudentUtils.java.**

EXERCISE 4

A maven exercise: create java maven project, manage dependency, build and package project with the Maven build tool.

Requirements: Write a Java program (using the maven console project) that takes a command line parameter as the url to the file to be downloaded to the local computer. If a valid url is passed, the file will be downloaded and stored locally with the corresponding name in the url. When the information is not entered correctly or an invalid url is entered, the appropriate message should be displayed. The program and all dependencies must be packaged into a single jar file (using maven's package command).

Some examples of commands when running the program are as follows:

- `java -jar Program.jar`: the program outputs "*Please specify an URL to a file*".
- `java -jar Program.jar http://abc`: the program outputs "*This is not a valid URL*".
- `java -jar Program.jar http://abc.com/document.doc`: the program downloads the document.doc file and saves it to the current working directory of the command line. If any error is encountered during the file download, display the corresponding error message.

You must to use the libraries from the maven repository to solve the problems in this exercise, specifically:

- Check the validity of the URL: Using the class **UrlValidator** in Apache Commons Validator library: *https://mvnrepository.com/artifact/commons-validator/commons-validator*
- Download files from the internet: use the mehtod **FileUtils.copyURLToFile()** in Apache Commons IO: *https://mvnrepository.com/artifact/commons-io/commons-io*

Instructions:

- Create a maven project using supported tools such as Visual Studio Code, Intellij Idea or Eclipse, using an archetype of: **maven-archetype-quickstart**.
- Modify the pom.xml file in the project to add the necessary libraries.

- Use the maven **package** command to compile and package the program into a *.jar file, the results will be stored in the project's target directory.

```
<groupId>org.example</groupId>
<artifactId>untitled</artifactId>
<version>1.0-SNAPSHOT</version>

<dependencies>
    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.11.0</version>
    </dependency>
</dependencies>
```

Set up the apache-commons library in the pom.xml file, see more at https://mvnrepository.com/artifact/commons-io/commons-io/2.11.0

- If you get the error "no main manifest attribute..." when you run the program, you need to add the xml highlighted in **green** below to the maven-jar-plugin plugin tag. Where com.mvm.App is the full path of the file containing the main method in your program.

```
<plugin>
  <artifactId>maven-jar-plugin</artifactId>
  <version>3.0.2</version>
  <configuration>
    <archive>
      <manifest>
        <addClasspath>true</addClasspath>
        <classpathPrefix>lib/</classpathPrefix>
        <mainClass>com.mvm.App</mainClass>
      </manifest>
    </archive>
  </configuration>
</plugin
```

- If you get an error like below then you need to use the maven-shade-plugin plugin, put it in the <plugins> tag inside the <build> tag like the example below. It should be noted that the added <plugins> tag must be outside the <pluginManagement> tag.

"NoClassDefFoundError: org/apache/commons/validator/routines/UrlValidator"

```
<build>
  <pluginManagement>
        // do not change this tag
  </pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <finalName>Program-${artifactId}-${version}</finalName>
      </configuration>
    </plugin>
  </plugins>
</build>
```