Lab 1 - Java

ESS 201 Programming II

International Institute of Information Technology – Bangalore

Part A (Submission: LMS by 1 Sep 11:00:00)

To be worked on in the Lab session and a pdf file need to be submitted with screenshot and writeup.

Task 1:

Submit one screenshot

As a first step, ensure you have an appropriate JDK installed on your machine. It is preferable that you use Java 8 or Java 11 or Java 17. Please make sure you are not using any features introduced after Java 8. Note that the .class files compiled with a later version of Java may not work with previous versions of the JVM.

Submit screenshot of java version after running below command:

```
> java --version
```

Task 2

Submit six screenshot and one writeup

This part consists of a set of short programs that are intended to help you get familiar with aspects of Java - basic language features, compiler and runtime environment. Screenshots need to be submitted on LMS.

A set of Java source files are available under Course Activities/Java/Lab 1 in LMS. You should try to compile and run each of them, and use them to help get started in this course. DO NOT USE AN IDE - work on the command line to invoke the compiler and to run the program

Copy each of the source files to your machine and try out the following as indicated. In each case, and each step, try to compare what you observe to similar functionality (or compiler/runtime behaviour) in C or Python. Keep track of the different warnings, compile errors and runtime errors. We will discuss all these in detail in subsequent lectures.

For each of these cases, compile and then run the program as follows :

```
> javac <filex>.java (where <filex>.java is one of the files below)
> java <filex>
```

1. HelloWorld.java:

A minimal Java program, just to get started. Just to make sure your environment is set up correctly. Don't worry about the specific syntax/keywords for now. Experiment with the following:

- delete either or both of the keywords "public" in this file

- change the name of the file, but retain the class name as HelloWorld within the file

2. HelloAgain.java

Just a little more than the first HelloWorld.

System.out.println(...) is roughly equivalent to fprintf(stdout, …) in C.

Note how you can combine strings and integers as arguments to this method.

3. TestArray.java

Compile and run this file by uncommenting one section at a time. Some of them will produce errors. Identify if these are compile-time or run-time errors. Can you fix the errors before moving on to the next step?

4. PrintArray.java

Here, the main invokes another method (or function) print.

Note that the loop to iterate over the elements of the array can be written without hardwiring the size of the array or any additional parameter

5. MemTest.java

Java manages memory differently from C or C++. How large can you make N and/or M before the program crashes or doesn't compile - in Step1 and then in Step 2? If you keep M fixed and keep increasing N, what do you notice about the running time of the

Program? Is there a way to increase the heap size available for the program?

6. Try the following on any of the above files. (optional)

When you compile using javac, the compiler writes out a file with the same class name as the input file, and with the extension .class. Copy this .class file to a different OS (say from Windows to Linux or vice versa), or a machine with a different architecture, and try running the file (without recompiling):

```
> java <classname>
```

Note that you are able to essentially move “object files” across OS/machines.

Try to summarize your understanding of the features of the Java language you have observed. How do these compare to C or Python (or C++ if you are familiar with that language)? Do you see any benefits of such features? (Note write answer to this question in the file and submit)

Part B (Submission: Codetanra by 8 Sep 11:00:00)

The source code for this exercise should be submitted by the due date.

In this assignment, you will complete the code for the **BankAccount** class to simulate a bank account. The class already has some basic methods and attributes defined, but you need to implement the functionality for depositing money, withdrawing money, transferring funds, changing the account holder's name, checking for overdraft, applying interest, and getting the account holder's name.

Instructions:

1. Open the provided **BankAccount** class, which has a skeleton structure with attributes and method signatures.
2. Implement the following methods within the **BankAccount** class:
    - **deposit(double amount)** - Adds the specified amount to the account balance.
    - **withdraw(double amount)** - Subtracts the specified amount from the account balance.
    - **transfer(BankAccount recipient, double amount)** - Transfers funds from this account to another account (recipient).
    - **changeAccountHolderName(String newHolderName)** - Changes the account holder's name to the new name provided.
    - **isOverdraft()** - Checks if the account is in overdraft (i.e., the balance is negative).
    - **applyInterest(double interestRate)** - Applies interest to the account balance based on the provided interest rate.
    - **getAccountHolder()** - Returns the account holder's name.
3. In the **Main** class, create two **BankAccount** objects for testing purposes. You may use the provided code as a reference.
4. Display the initial account information for both accounts.
5. Perform various operations on **account1**, such as depositing, withdrawing, transferring funds, changing the account holder's name, and applying interest.
6. Display the updated account information for both accounts after the operations.
7. Check if **account1** is in overdraft and print an appropriate message.

Note: Check Main.java file for the skeleton code.