Hands-on Session on Java
COL106 (Sem I, 2019-20)

July 23, 2019

This practice session module has been developed by Chinmay Narayan, Ismi Abdi and Prof. Amitabha Bagchi. Thanks to Prof. Amitabha Bagchi for sharing with us.

This practise session is to help students to get familiarized with Java programming. We will be starting with basic syntax and features of Java. This module is by no means complete reference of Java as it is nearly impossible to squeeze a book's worth of material into 2-3 lab session modules. Students are advised to compile and execute each of them.

There are two type of exercises: (a) self-help, and (b) programming.

- The self-help exercises are marked yellow in the manual. For each self-help exercise, one file is given in the exercise directory: intro.java. The intro.java file contains the code required to understand the exercise.
- The programming exercises are marked blue in the manual. For each programming exercise, three files are given in the exercise directory: checker.java, program.java, Makefile. You are supposed to modify the test() function in program.java. checker.java will call the test function of program.java, and it should return the expected output. You should not modify the signature of the test function (a function signature includes the function's return type, the number of arguments, and the types of arguments). Type make to compile and execute the exercise. **DO NOT MODIFY** checker.java. This file contains the test cases which should pass with your program. At the time of demo, we will change the test cases. You are expected to complete the exercises on your own.

Example for self-help exercise:

```
$ tar -xvf JavaModuleExercises.tar.gz
$ cd JavaLabModule/exercise1
$ javac intro.java
$ ls
```

Example for programming exercise:

```
$ tar -xvf JavaModuleExercises.tar.gz
$ cd JavaLabModule/exercise9
..... Open program.java, and un-comment line 12 .....
$ make
```

If the make command is not working, replace it with following commands:

```
$ javac program.java
$ javac checker.java
$ java checker
```

In imperative programming languages like C, a program is made of a set of functions which are invoked by a main function in some order to perform a task. In some sense, the most basic unit of computation in these languages is function and data. In Java, or in any other Object oriented programming language (OOP), the most basic unit of computation is an object or a class.

Class For any variable x in a programming language, the *type* of x defines what kind of operations are valid on that variable. Java, like every programming language, has a set of *primitive types* to represent *integer*, *character*, *real* etc. Java also allows user to create new types by means of class declaration. A class is an encapsulation of data and the methods operating on this data in a single unit of type. The notion of **Class** in Java is somewhat similar to **Type** in **SML** programming language and **structure** in **C**. A class in Java is written as below.

Listing 1: An empty java class.

```
class intro
{

}
```

A class in Java is called a *public class* if the keyword public is added before the keyword class in a class definition. For example the code of listing 2 creates a public class **newintro**.,

Listing 2: An empty public java class.

```
public class newintro
{

}
```

Object Object is an instantiation of a class type. For the example class of listing 1, to *declare* an object of type intro, we write

    intro var;

where var is any string. Above syntax declares that var is a *reference/handle* of type intro. It **is to be noted that the above syntax only creates a handle/reference to an intro object. Actual object is not created by this syntax.** Java provides a keyword new to create a new object of a given type and assign it to a reference of that type.

    intro var = new intro( );

Above syntax declares a reference variable var of type intro as well as initialize it with a newly created intro object. In absence of new intro() part, reference variable is only initialized to null and any operation on such uninitialized reference will generate run-time error. **Therefore we must make sure that every reference is properly initialized before using it.**

Fields and Methods A class consists of a set of variables, also called *fields*, and a set of functions, also called *methods*. For an object of a given class, the fields and methods of that class are accessed using . operator. A field of a class is of the following form **[Qualifier] type variablename** where **Qualifier** is optional and can take following values,

- public: It means that this field can be accessed from outside the class by just using *objectname.fieldname*.
- static: It means that to access this field you do not need to create an object of this class. Therefore this field is also sometimes called as class *variable/field*.

For the example program in listing 3, credits is a public static variable and hence can be accessed from outside this class as intro.credits. On the other hand, age is only a pubic variable and hence can only be accessed after creating an object of intro type as below.

```
intro var = new intro( );
var.age = 10;
```

Above code first creates an object of **intro** class and then assign value 10 to the field age of this object. It is to be noted that every object of a class, contains different copies of non-static variable (e.g. age in listing 3) and the same copy of static variable (credits in listing 3). You do not require to create an object to access the static members of a class.

Listing 3: field qualifiers

```
class intro
{
 public int age ;
 public static int credits ;
 }
```

methods A method/function is defined inside a class as **[Qualifier] return-type function-name (arguments)**. A method in Java also specifies *exception* signature which we will discuss in detail in *error handling*. Two important qualifiers for methods are same as for fields,

- public: A public method can be invoked from outside the class using the object of that class.
- static: A static method can be called without creating an object of that class.

Following restrictions are applied on invocation of static methods.

- static function can not call non-static functions of that class.
- static function can not access non-static variables/fields of that class.
-

Every class in Java has a public method by default, it is called *Constructor* of that class. The name of the constructor function is same as the name of the class without any return type, but it can take a set of arguments like any method. This method is called when an object of this class is created using *new* operator as shown earlier.

Listing 4: Filling up methods and fields

```
class intro
{
  public int age ;
  public static int credits ;
  public intro ( int a )
  {
    age = a ;
  }
  public void setage ( int newage )
  {
    age = newage ;
  }
  public static int getcredit ()
  {
    return credits ;
  }
}
```

In the example program of listing 4, static function getcredit returns the value of static field credits. Function setage takes an integer argument and set the value of the field age to this value. Return type of setage is void as it does not return any value, while the return value of getcredit is int. We also have a constructor in this class which takes an integer and assign it to field age of that object. Constructors are mainly used to make sure that all fields of an object are initialized properly at the time of creation. To create an object of class intro, we use following syntax,

```
intro handler = new intro (4);
```

Notice that we pass an integer (4 in this case) while creating an object with new operator. This is because the constructor of class intro takes an integer argument. Therefore the constructor of a class, when defined, also tells us what all arguments must be passed while creating an object of that class. If no constructor is defined explicitly, java itself puts a **zero argument constructor** in that class.

Having covered the most essential aspect of a class definition in Java, now let us look at the process of compiling and executing a Java program.

File naming convention for java programs

- Any java program must be saved in a file with extension ".java".
- A java file can contain at most one public class.

- If a java file contains a public class then the name of the file must be same as the name of the public class.
- If a java file contains no public class then the name of the file must be same as any of the non-public class present in that file.
- Smallest non-empty java file, which is compilable, must have at least one class declaration.

Compiling a java program The compiler of Java is a program called "javac". To compile a java program which is saved in a file, say "first.java", execute the following command on shell/command line.

```
javac first.java
```

As a result of this compilation a set of ".class" files are generated, one for each class declared in "first.java". These files are like compiled object files of "C".

Exercise 1: Compile the code of listing 4. What file name will you use to save and compile this program? What all new files are generated by compilation?

Executing a java program Once you get a set of class files after compiling a java program, you execute the program with the following command.

```
java first
```

where "first" is the name of the public class, if any, which was declared in "first.java". It is to be noted that **to execute a java program it must contain at least one public class with a public static method main declared in it**, as below.

```
public static void main ( String args [])
```

Why a method main is needed? Method main is like entry point for a java program. It must be static so that the java runtime do not need to instantiate any object to executed it. It must be public so that it can be invoked by java runtime.

Exercise 2: Try to execute the program of listing 4 and check what error do you get? Now add function main (of same syntax as given above) in the class intro of listing 4 and again try to compile and execute it.

Using library functions/helper functions defined elsewhere In C, helper functions or library functions (like printf, scanf etc.) are used by including appropriate header file with **# include** syntax. In Java, the equivalent of a header file is *Package* which is included in your java code using **import** syntax. A package name, say X.Y.Z denotes a directory structure where the topmost directory is named as X, inside it there is a subdirectory named Y which in turn contains a compiled java file Z.class. The syntax import X.Y.Z in a java program makes all public functions of class

Z.class available to this program. One most important package which contain the helper functions for console input/output (like printf and scanf) is java.lang. This package is include by default in every java program.

Now we are ready to write, compile and execute our first java program which prints a string on command line/ console.

Listing 5: Printing “hello world” on screen

```java
public class intro
{
   public static void main ( String args [])
   {
       System . out . println ( " Hello world " );
    }
}
```

Exercise 3: Compile and execute the program of listing 5.

We can modify the program of listing 5 as following

- Create a public function in class intro which prints Hello world on screen.
- Create an object of this class in function main and invoke/call the function define above on this object.

Along the same line,

Exercise 4: Complete the program of listing 6 (in place of · · ·) such that it prints ”Hello world” on console.

Listing 6: Modified hello world program

```java
public class intro
{
   public void printinfo ()
   {
      System . out . println ( " Hello world " );
   }
    public static void main ( String args [])
    {
       ...
    }
}
```

Reading from console To read from command line we use the class **Scanner** which is defined in the package java.util.Scanner. Following program takes one string (name) and an integer (age) from the command line and store it in two fields.

Listing 7: Reading from command line

```
1  import java . util . Scanner ;
2  public class intro
3  {
4    public String name ;
5    public int age ;
6    public static void main ( String args [])
7    {
8       Scanner s = new Scanner ( System . in );
9       intro obj = new intro ();
10      System . out . println ( " Enter your name " );
11      obj . name = s . nextLine ();
12      System . out . println ( " Enter your age " );
13      obj . age = s . nextInt ();
14      s . close ();
15      System . out . println ( " Name : " + obj . name );
16      System . out . println ( " Age : " + obj . age );
17    }
18 }
```

- In line 1, we import the package *java.util.Scanner* which contain the class **Scanner** and associated methods.
- We have two public fields, name and age, declared in class intro.
- In main, we create a **Scanner** object with argument **System.in** which represents standard input. We will see later in *file handling* section that how the same scanner class, just by changing this input stream parameter, can be used to read data from files.
- We create an object of intro class in line 9.
- As it is clear from the names, **nextLine()** and **nextInt()** stops at command prompt and allows user to enter a string and integer respectively.
- Notice the + operator in **System.out.println**. This operator is used to concatenate two strings. Even though obj.age is of integer type, java implicitly converts it into string.

Exercise 5: Compile the code of listing 7 and check the output.

loop and condition constructs Java has three constructs for implementing loops,

- do{Statements} while(boolean condition);,
- while(boolean condition){Statements}, and
- for(initialization; terminating condition; post-operation){Statements}.

Listing 8 shows a program to print all numbers from 10 to 100 using all three looping constructs.

Listing 8: Different looping constructs in Java

```
public class intro
{
  public static void main ( String args [])
  {
    int i =10;
    System . out . println ( " Using do - while " );
    do {
       System . out . println ( i );
       i ++;
     } while (i <100);
     for ( i =10; i < 100; i ++){
       System . out . println ( i );
     }
     i =10;
     while (i <100){
       System . out . println ( i );
       i ++;
      }
   }
}
```

For selection (if then else), java provides two constructs.

- if(condition) {Statements} else {Statements}
- Switch-case. break is used to break the flow of program once a case is satisfied.

An example program to check if a given number is even or odd is implemented using both conditional constructs in listing 9.

Listing 9: Different conditional constructs in Java

```
import java . util . Scanner ;
public class intro
{
   public static void main ( String args [])
   {
      Scanner s = new Scanner ( System . in );
      System . out . println ( " Enter a number " );
      int inp = s . nextInt ();
      System . out . println ( " Using if - else construct " );
      if ( inp % 2 == 0)
        System . out . println ( inp + " is even " );
      else
        System . out . println ( inp + " is odd " );
15
16         System . out . println ( " Using switch case construct " );
17         switch ( inp % 2){
18           case 0:
19             System . out . println ( inp + " is even " );
20             break ;
21           case 1:
22             System . out . println ( inp + " is odd " );
23             break ;
24         }
25     }
26 }
```

Exercise 6: Compile and execute the program of listing 9. After that remove the break of line 20 and again execute the program. What is the difference?

Passing arguments In Java, all arguments are **passed by value**. What does it mean for non-primitive types? As we saw earlier that when we create a variable of a non-primitive type (abstract data type/ class) then effectively we only create a reference to an object of that type. For example, the syntax

```
intro introobj;
```

only creates a handle/reference **introobj** of the type **intro**. No object is created as a result of this declaration. To create an object and assign that object to this reference, we use new keyword.

```
introobj = new intro( );
```

When we pass an object as an argument to another function, we only pass reference/handle of that object. This reference is also passed as value but because this is a reference hence any change in the called function using this reference changes the original object as well.

It is to be noted that the declaration of primitive type (like int, float etc.) variables does not create reference/handle but creates the actual storage for that variable. When a primitive type variable is passed as an argument to another function, it is passed by value. But because actual value is copied to the called function, instead of the reference, hence any change done to this copy is not reflected back in the original copy of this variable in calling function. Let us try to understand it with the help of an example in listing 10.

Listing 10: Parameters are always passed by value; Objects are no exception

```
import java . util . Scanner ;
class holder {
  public int old ;
}
public class intro
{
    public static void main ( String args [])
    {
        holder h ;
        h = new holder ();
        h . old = 20;
        System . out . println ( " Value of old " + h . old );
        update (h ,30);
        System . out . println ( " Value of old " + h . old );
    }

    static void update ( holder k , int newval )
    {
        k . old = newval ;
    }
}
```

Exercise 7: Compile and execute the program of listing 10.

Arrays In java arrays are declared as

Listing 11: Array declaration in Java

```
int [] record ;
float [] points ;
Student [] sectionb ;
```

Notice that you can not specify number of elements in array by only declaring the array variable. A declaration as in listing 11 only declares that the variable is an array of given type.

```
int[] record = new int[5];
```

declares that the variable record is an array of integer type and also allocates the space of 5 integers in this array. Similarly,

```
int[] record = {1,2,3};
```

declares that the variable record is an array of integer type and contains elements 1,2 and 3. Its size has been fixed by this declaration to three elements. Similarly,

```
Student[] sectionb = new Student[5];
```

declares that the variable sectionb is an array of Student type and also allocates the space of 5 Student objects in this array. **Notice that this declaration does not create individual object for each of these 5 slots. You must create 5 Student objects by new keyword and assign them to these slots explicitly, otherwise these slots will remain uninitialized and therefore contain null instead of Student objects.** Given an array variable, say record, the length of the array is given by the field length, as in listing 12.

Listing 12: Manipulating arrays in Java

```
1  import java . util . Scanner ;
2  public class intro
3  {
4     public static void main ( String args [])
5     {
6         Scanner s = new Scanner ( System . in );
7         int n = s . nextInt ();
8         int [][] data = new int [ n ][ n +2];
9         System . out . println ( " number of rows = " + data . length
10                  + " and number of columns = " + data [0]. length );
11        for ( int i =0 ; i < data . length ; i ++){
12            for ( int j =0; j < data [0]. length ; j ++)
13                data [ i ][ j ] = 1;
14    }
15
16    int [][][] datam = new int [ n ][ n +2][ n +4];
17    System . out . println ( " Length along outermost indices = "
18         + datam . length );
19     System . out . println ( " Length along middle indices = "
20         + datam [0]. length );
21     System . out . println ( " Length along innermost indices = "
22         + datam [0][0]. length );
23
24    for ( int i =0 ; i < datam . length ; i ++){
25        for ( int j =0; j < datam [0]. length ; j ++)
26            for ( int k =0; k < datam [0][0]. length ; k ++)
27               datam [ i ][ j ][ k ] = 1;
28     }
29   }
30 }
```

- Program of listing 12 reads an integer from user and create a two dimensional integer array data at line 9 with number of rows equal to n and number of columns equal to n+2.

- It is important to notice the use of length field of array variable for finding out the length along different dimensions of a multi-dimensional array.
- data.length gives length of the array variabledata along outermost dimension.
- You keep on adding index 0 to array variable to get the reference along that dimension. Field length on that reference gives the length along that dimension as shown in lines 18, 20 and 22. This point is important when you want to traverse along different dimensions of an array as shown in the loops of line 12 and 25.

Exercise 8: Compile and execute the program of listing 12.

Strings and associated helper functions In java strings are declared as below.

```
String name;
```

where name is a reference to an object of string type. To create actual string object and assign that to the reference name, we need to do the following,

```
name = "CSL 201";
```

Initialization can be done either at the time of declaration of reference or later.

Listing 13: Manipulating Strings in Java

```
import java . util . Scanner ;
public class stringex
{
   public static void main ( String args [])
   {
      Scanner s = new Scanner ( System . in );
      String line = s . nextLine ();
      String newline = " " ;
      for ( int i =0; i < line . length (); i ++)
      {
         if ( line . charAt ( i )!= ’ ’)
         newline = newline + line . substring (i , i +1);
      }
      System . out . println ( " String after removing all spaces : " +
                    newline );
   }
}
```

Consider an example of listing 13. It reads a line from console and store it in a string variable **line**. Then it iterates over all characters of this string and check if the character at any index (using **charAt** function) is equal to whitespace (’ ’). If it is not equal to whitespace then it concatenates the string representation of that character (obtained by **substring** function) in a temporary string **newline**. When loop terminates, **newline** points to a string which is same as the original one without any whitespace. Function **substring** takes two arguments, starting index and end index, and return the substring present in between them. It does not include the character present at the end index.

To get more useful functions on **String** class of Java, please refer to the following link.

http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/String.html

This link/site host the documentation of all classes and associated functions of Java language. If you want to know about any function of any utility class, say **Date, File, String** etc., just go to the above link and look for the appropriate class name and its member functions.

Exercise 9: Leap year- Given a year, return true if it is a leap year otherwise return false. Please note that years that are multiples of 100 are not leap years, unless they are also multiples of 400.

Exercise 10: Same last digit- Given two non-negative integers, return true if they have the same last digit, such as with 27 and 57. Note that the % ”mod” operator computes remainder, so 17%10 is 7.

Exercise 11: Least common multiple- Given two integers n and m, the objective is to compute the LCM (least common multiple) of n and m. LCM is the smallest number that is divisble by both n amd m. For e.g. is n is 12 and m is 14, the LCM is 84. If n is 32 and m is 16, the LCM is 32.

Exercise 12: Roots of polynomial- Write a Java program that given b and c, computes the roots of the polynomial x*x+b*x+c. You can assume that the roots are real valued and need to be return in an array.

Exercise 13: Average of numbers- Given an array of integers finds the average of all the elements. For e.g. for 4,7,9,4 the average is 6 and for 1,3,8,5 the average is 4.25. Please note that if the array has no elements, the average should be 0.

Exercise 14: Remove zeros- Given an array of integers return an array in the same order with all 0’s removed. For e.g. is the input is 1,2,3,4,5,0,1,2,0,0,2 the expected output is 1,2,3,4,5,1,2,2. If the input is 0,0,1,2 the output is 1,2. If the input is 0,0,0,0 the expected output is .

Exercise 15: Hex to binary- Given a string representing a number in hexadecimal format, convert it into its equivalent binary string. For e.g. if the input if ”1F1” then its binary equivalent is ”111110001”. If the input is “13AFFFF”, the output should be “1001110101111111111111111”.

Exercise 16: Java files- You have been given the list of the names of the files in a directory. You have to select Java files from them. A file is a Java file if it's name ends with ".java". For e.g. "File- Names.java" is a Java file, "FileNames.java.pdf" is not. If the input is "can.java", "nca.doc", "and.java", "dan.txt", "can.java", "andjava.pdf",the expected output is "can.java", "and.java", "can.java"

Exercise 17: Most frequent digit- Given a number, the objective is to find out the most frequently occuring digit in the number. If more than 2 digits have the same frequency, return the smallest digit. The number is input as a string and the output should be the digit as an integer. For e.g. if the number is 12345121, the most frequently occuring digit is 1. If the number is 9988776655 the output should be 5 as it is the smallest of the digits with the highest frequency.

Exercise 18: Matrix addition- Given two matrices M1 and M2, the objective to add them. Each matrix is provided as an int[][], a 2 dimensional integer array. The expected output is also 2 dimensional integer array.

Exercise 19: String concat- Given two strings s1,s2 and two indices m,n return a string having chars from index m to end of s1 and then 0 to n of s2 (both m and n are inclusive). For eg, if s1="hello", s2="world", m=3, n=0, then answer is "low"

Exercise 20: Score Marks- The "key" array is an array containing the correct answers to an exam, like 'a','a','b','c'. The "answers" array contains a student's answers, with ' ?' representing a question left blank. The two arrays are not empty and are the same length. Return the score for the provided array of answers, giving a +4 for each correct answer, -1 for each incorrect answer and 0 for each blank answer. For e.g. key = 'a','c','d','b' answers = 'c','c',' ?','b' then score is -1 + 4 + 0 + 4 =7