

**Department of Computer Science
Faculty of Physical Sciences
Ahmadu Bello University, Zaria**

COSC 211 : Object Oriented Programming I - LAB00

Contents

- [What is Java Development Kit \(JDK\)](#)
- How to Download JDK [Windows](#) / [Mac OS](#)
- How to Install JDK [Windows](#) / [Mac OS](#)
- My First Program [On Windows](#) / [On Mac OS](#)

What is JDK

Java Development Kit (JDK), officially named "Java Platform Standard Edition (Java SE)", which is freely available from Sun Microsystems (now part of Oracle), is needed for writing Java programs. For more details go to <http://www.oracle.com/technetwork/java/index.html>. JRE (Java Runtime Environment) is also needed for running Java programs but once installed the JDK it comes with JRE.

How to Download JDK (Windows)

Step 1: Un-install Older Version(s) of JDK/JRE or Go to Step 1

It's recommended that you should install only the latest JDK. Although you can install multiple versions of JDK concurrently but is messy. If you have already installed older version(s) of JDK/JRE, un-install All of them.

Step 2: Download JDK (If already have it go to Installation guides)

- Go to Java SE download site <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Under "Java Platform, Standard Edition" ==> "Java SE 8u{xx}", where {xx} is the latest update of number ==> Click "Java Download" button.
- Read and check "Accept License Agreement".
- Choose your operating system platform, e.g., "Windows x64 (for 64-bit Windows OS) or "Windows x86 (for 32-bit windows OS). You can check whether your windows OS is 32-bit or 64-bit via "Control Panel" ==> "System" ==> Under "System Type".

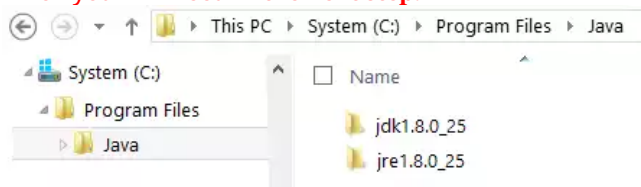
How to Install JDK 8 (on Windows)

Step 1: Install JDK and JRE

Run the downloaded installer (e.g., "jdk-8u{xx}-windows-x64.exe"), which install both the JDK and JRE. By Default, the JDK will be installed in directory "C:\Program Files\Java\jdk1.8.0_xx", where xx denotes the latest upgrade number; and JRE in "C:\Program Files\Java\jre1.8.0_xx".

For novices, accept the defaults, Follow the screen instruction to install the JDK and JRE

Check the JDK installed directory by inspecting these folders by going to "C: drive" ==> "Program Files" ==> "Java". Take note of your JDK installed directory, which you will need in the next step.



Step 2: Include JDK's "bin" Directory in the PATH

Windows OS searches the current directory and the directories listed in the PATH environment variable for executable programs. JDK's programs (such as Java compiler javac.exe and Java runtime java.exe) reside in directory "\bin" (where denotes

the JDK installed directory). You need to include "\\bin" in the PATH to run the JDK programs.

To edit the PATH environment variable in Windows XP/Vista/7/8/10:

- a. Launch "Control Panel" ==> "System" ==> Click "Advanced system settings".
- b. Switch to "Advanced" tab ==> "Environment Variables".
- c. Under "System Variables", scroll down to select "Path" ==> "Edit...".
- d. **(CAUTION: Read this paragraph 3 times before doing this step! There is no UNDO)**

For Windows 10: You see a table listing the existing PATH entries. Click "New" ==> Enter the JDK's binary directory "c:\Program Files\Java\jdk1.8.0_xx\bin" (Replace xx with your installation's upgrade number!!!) ==> Select "Move Up" to move it all the way to the top.

Prior to Windows 10: In "Variable value" field, INSERT "C:\Program Files\Java\jdk1.8.0_xx\bin" (Replace xx with your installation upgrade number!!!) IN FRONT of all the existing directories, followed by a semi-colon (;) which separates the JDK's binary directory from the rest of the existing directories. DO NOT DELETE any existing entries; otherwise, some existing applications may not run.

Variable name : PATH

Variable value : C:\Program Files\Java\jdk1.8.0_xx\bin; [existing entries...]

Step 3: Verify the JDK Installation

Launch a CMD shell (Click "Start" button ==> run... ==> enter "cmd"; OR from "Start" button ==> All Programs ==> Accessories ==> Command Prompt).

- a. Issue "path" command to list the contents of the PATH environment variable. Check to make sure that your \bin is listed in the PATH.
// Display the PATH entries
prompt>path
PATH=c:\Program Files\Java\jdk1.8.0_xx\bin;[other entries...]
prompt>_

Don't type prompt>, which denotes the command prompt!!! Key in the command (highlighted) only.

- b. Issue the following commands to verify that JDK/JRE are properly installed and display their version:
Display the JRE version and the JDK version
prompt> java -version
java version "1.8.0_xx"
Java(TM) SE Runtime Environment (build 1.8.0_xx-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.5-b02, mixed mode)
prompt> javac -version
javac 1.8.0_xx
prompt>_

My First Program on Windows

Step 1: Write a Hello-World Java Program

- a. Create a directory to keep your works, e.g., "C:\myProject", or any directory of your choice. The directory name shall not contain blank or special characters. Use meaningful but short name as it is easier to type.
- b. Launch a programming text editor (such as NotePad, TextPad or any text editor recommended by your lab instructor). Begin with a new file and enter the following source code. Save the file as "Hello.java", under your work directory (e.g., C:\myProject).

```
/*
 * My first Java program to say Hello
 */
// Save as "Hello.java" under "C:\myProject"
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Step 2: Compile and Run the Hello-World Java Program

To compile the source code "Hello.java":

- a. Start a CMD Shell (Click the "Start" button ==> "run..." ==> Enter "cmd").
- b. Set the Current Drive to the drive where you saved your source file "Hello.java". For example, suppose that your source file is saved in drive "C", enter "C:" as follow:

```
prompt>C:
C:\Users\user-pc>_
C:\Users\user-pc>cd ..
C:\Users>cd ..
C:\>_
```

Don't enter prompt>, which denotes the command prompt.
- c. Set the Current Working Directory to the directory that you saved your source file via the cd (Change Directory) command. For example, suppose that your source file is saved in directory "C:\myProject".

```
C:\>cd \myProject
C:\myProject>_
```
- d. Invoke the JDK compiler "javac" to compile the source code "Hello.java".

```
D:\myProject>javac Hello.java
```

The compilation is successful if the command prompt returns. Otherwise, error messages would be shown. Correct the errors in your source file and re-compile. Check "Common JDK Installation Errors", if you encounter problem compiling your program.

- e. The output of the compilation is a Java class called "Hello.class". Issue a dir (List Directory) command again to check for the output.

```
D:\myProject>dir
```

```
.....
xx-xxx-xx  01:53 PM                416 Hello.class
xx-xxx-xx  06:25 PM                277 Hello.java
.....
```

To run the program, invoke the Java Runtime "java":

```
D:\myProject>java Hello
Hello, world!
```

How to Download JDK (Mac OS)

Step 1: Check if JDK has been Pre-Installed

In some Mac systems (earlier than Mac OS X 10.7 Lion), JDK has been pre-installed. To check if JDK has been installed, open a "Terminal" (Finder ==> Go ==> Utilities ==> Terminal) and issue this command:

```
javac -version
```

- If a JDK version number is returned (e.g., JDK 1.x.x), then JDK has already been installed. If the JDK version is prior to 1.7, proceed to Step 2 to install the latest JDK; otherwise, proceed to "Step 3: Write a Hello-world Java program".
- If message "command not found" appears, JDK is NOT installed. Proceed to the "Step 2: Install JDK".
- If message "To open javac, you need a Java runtime" appears, select "Install" and follow the instructions to install JDK. Then, proceed to "Step 3: Write a Hello-world Java program".

Step 2: Download JDK

1. Goto Java SE download site @ <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. Under "Java Platform, Standard Edition" ==> "Java SE 8u{xx}" ==> Click the "JDK Download" button
2. Check "Accept License Agreement"
3. Choose your operating platform, e.g., "Mac OS X" (jdk-8u{xx}-macosx-x64.dmg). Download the installer.

Install JDK/JRE (on Mac OS)

- a. Double-click the downloaded Disk Image (DMG) file. Follow the screen instructions to install JDK/JRE.
- b. Eject the DMG file.
- c. To verify your installation, open a "Terminal" and issue these commands.

Display the JDK version

```
javac -version
javac 1.x.x_xx
```

Display the JRE version

```
java -version
java version "1.x.x_xx" Java(TM) SE Runtime Environment (build 1.x.x_xx-xxx)
Java HotSpot(TM) Client VM (build 22.1-b02, mixed mode, sharing)
```

Display the location of Java Compiler

```
which javac /usr/bin/javac
```

Display the location of Java Runtime

```
which java /usr/bin/java
```

My First Program on Windows

Step 1: Write a Hello-World Java Program

- Create a directory called "myProject" under your home directory (Launch "Finder" ==> "Go" ==> "Home"; Select "File" ==> "New Folder" ==> "myProject"). In Mac OS, the home directory of the current login user is denoted as "~". Hence, this new directory is represented as "~/myProject".
- Use a programming text editor (such as jEdit, gedit, Sublime Text or any text editor recommended by your lab instructor) to input the following source code and save as "Hello.java" under the directory "~/myProject". [If you use Mac OS's default text editor "TextEdit" (NOT recommended), you need to open a new file ==> choose "Format" ==> "Make Plain Text" ==> Enter the source code ==> Save as "Hello.java".]

```
/*
 * My First Java program to say Hello
 */
// Save as "Hello.java" under "~/myProject"
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Step 2: Compile and Run the Hello-World Java Program

- To compile the source code "Hello.java", open a new "Terminal" ("Go" ==> "Utilities" ==> "Terminal") and issue these commands (as illustrated):

Change Directory (cd) to where "Hello.java" resides

```
cd ~/myProject
```

Check if "Hello.java" exists using list (ls) command

```
ls
Hello.java .....
```

Compile "Hello.java" using JDK compiler "javac"

```
javac Hello.java
```

If error message appears, correct your source code and re-compile

Check for the compiled output "Hello.class"

```
ls  
Hello.class  
Hello.java .....
```

b. To run the Hello-world, invoke the Java Runtime "java" as follows:

Run "Hello.class"

```
java Hello  
Hello, world!
```

**Department of Computer Science
Faculty of Physical Sciences
Ahmadu Bello University, Zaria**

COSC 211 : Object Oriented Programming I - LAB01

Objectives

To gain experience with:

- [Printing Programs Output](#)
- [Escape Sequences](#)
- [Assignment](#)

Printing Output

A print statement is actually a call to the *print* or *println* method of the [System.out](#) object. The print method takes exactly one argument or no arguments. However, it can take string or numeric value

There are three kinds of print statements in java:

1. [System.out.print\(argument\)](#) just print out its argument.

Example1: Type and run the following program and observe the output

```
//Printing.java
public class Printing{
    public static void main(String[] args){
        System.out.print("Computer");
        System.out.print("Science");
        System.out.print("Department");
    }
}
```

2. [System.out.println\(argument\)](#) prints out its argument and ends the line.

Example2: Modify the example1 program above, save as PrintLine.java, and compile, run, and observe the output

Note that "print" is to be replaced with "println" method

```
//PrintingLine.java
public class PrintingLine{
    public static void main(String[] args){
        System.out.println("Computer");
        System.out.println("Science");
        System.out.println("Department");
    }
}
```

3. [System.out.printf\(format, argument\)](#), gives more control over how things are printed.

Format String::

Composed of literals and format specifiers. Arguments are required only if there are format specifiers in the format string. Format specifiers include: flags, width, precision, and conversion characters in the following sequence:

% [flags] [width] [.precision] conversion-character (square brackets denote optional parameters)

Flags:

- : left-justify (default is to right-justify)
- + : output a plus (+) or minus (-) sign for a numerical value
- 0 : forces numerical values to be zero-padded (default is blank padding)
- , : comma grouping separator (for numbers > 1000)
- : space will display a minus sign if the number is negative or a space if it is positive

Width:

Specifies the field width for outputting the argument and represents the minimum number of characters to be written to the output. Include space for expected commas and a decimal point in the determination of the width for numerical values.

Precision:

Used to restrict the output depending on the conversion. It specifies the number of digits of precision when outputting floating-point values or the length of a substring to extract from a String. Numbers are rounded to the specified precision

Conversion-Characters:

- d : decimal integer [byte, short, int, long]
- f : floating-point number [float, double]
- c : character Capital C will uppercase the letter
- s : String Capital S will uppercase all the letters in the string
- h : hashcode A hashcode is like an address. This is useful for printing a reference
- n : newline Platform specific newline character- use %n instead of \n for greater compatibility

Example3: Study, type, run and compile the following program and observe the output

```
//PrintingFormat.java
public class PrintingFormat{
    public static void main(String[] args){
        System.out.printf("%d",60);
        System.out.println();

        System.out.printf("%03d%n",9);

        System.out.printf("PI %.3f",3.141527);
        System.out.println();

        System.out.printf("Total is: N%,.2f%n", 84785.8978);

        System.out.printf("%-10S%n","Department of Computer Science");

        System.out.printf("% .2f",-67.8917);
        System.out.println();

        System.out.printf("% .2f",67.8917);
        System.out.println();

        System.out.printf("%C",'a');
        System.out.println();
    }
}
```

}

Escape Sequence

A character preceded by a backslash (\) is an escape sequence and has a special meaning to the compiler.

Following table shows the Java escape sequences -

Escape Sequence	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point..
\f	Inserts a form feed in the text at this point.
'	Inserts a single quote character in the text at this point.
"	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly.

Example: Study, Type, compile and run the following program and observe the output

```
// EscapeSequence.java
public class EscapeSequence {

    public static void main(String args[]) {
        System.out.println("ABU said \"To pass, you must work hard!\".\n");
        System.out.println("Two Sided Triangles\n*\t*****\n**\t ****\n***\t ***\n****\t **\n*****\t *\n");
        System.out.println("A Diamond \n *\n\r ***\n\r*****\n ****\n *\n");
        System.out.println("A Letter S\n *\n\b* *\n * \n* *\n");
        System.out.println(" E = 46.12 * 6.9 - 2.5 * 10\n\t\b_____ \n\t 90.5 * 11.6");
    }
}
```

Assignment

1. Write a program that displays the following pattern



2. Write a program that displays the following table

	1	2	3	4
1	1	2	3	4
2	2	4	6	16
3	3	6	9	12
4	4	8	12	16

Department of Computer Science

Faculty of Physical Sciences

Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB02

Objectives:

To gain experience with:

- Creating objects from classes
- Fundamental Data Types
- The String class

1. Creating Objects from classes

Java is an **object-oriented** programming language, meaning problems are solved by interaction among objects. An **Object** is a software-entity that models some real-world entity – e.g. A particular rectangle, a particular student, etc.

Each object has a state (represented by its **fields**) and behaviors (represented by its **methods**).

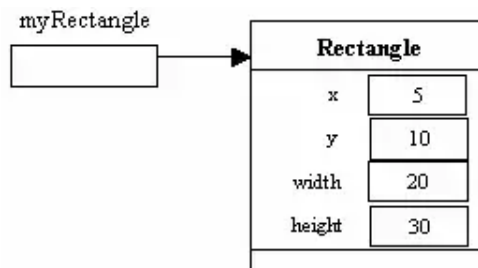
Objects are created from **classes**, thus to create an object, we need first to design a class. We can also use classes from the Java library. The Java library consists of hundreds of classes organized into related groups called **packages**.

A **class** is a general blueprint or specification for a set of objects in which fields and methods are defined.

To create an object from a class, we use the **new** operator as follows:

```
Rectangle myRectangle = new Rectangle(5,10, 20, 30);
```

In the above, **Rectangle** is a class provided in the Java library under **java.awt** package. When creating a rectangle object, we need to provide values for the top-left corner (x,y), width and height of the rectangle object. The **new** operator creates the rectangle object and returns the address of the object in memory which is then assigned to the **reference** variable **myRectangle**.



Once an object is created, we can call its methods using the **dot** operator. For example, to call the **translate** method, we write:

```
myRectangle.translate(10, 15).
```

The following is a complete program showing the above points. Notice that if we use a Java library class that is not in the **java.lang** package, we must import the class using the **import** statement.

Example1:

```
import java.awt.Rectangle;

/* Creates a rectangle object, calls its translate method and then print it. */
public class MoveRectangle {
    public static void main(String[] args) {
```

```

        Rectangle myRectangle=new Rectangle(5, 5, 10, 10);

        System.out.println("Before Translation: "+myRectangle);

        myRectangle.translate(15,25);

        System.out.println("After Translation: "+myRectangle);

    }

}

```

2. The String Class.


A string is a sequence of characters enclosed in double quotes.
 String processing is one of the common applications of computers. Java recognizes this fact and therefore provides special support for string processing through a class called **String** in the java.lang package, which has many useful methods.
 To create a string object from the string class we can use the new operator as follows:

```
String s = new String("the content of the string in quotes");
```

However, because of its common use, Java allows strings objects to be created without using the new operator as in:

```
String greeting = "Hello, World!";
```

In a string object, the characters are indexed starting from 0 as shown below:

<i>greeting</i>														
address of object		H	e	l	l	o	,		W	o	r	l	d	!
		0	1	2	3	4	5	6	7	8	9	10	11	12

The table below shows some of the most frequently used methods of the String class:

method	description	String myString = 'examples'
int length()	returns the number of characters in this String.	int i = myString.length(); // i = 8
String toUpperCase()	returns a new String, representing the Upper-case equivalent of this String.	String upper=myString.toUpperCase(); // upper == "EXAMPLES"
String toLowerCase()	returns a new String, representing the lower-case equivalent of this String.	String lower=myString.toLowerCase(); // lower == "examples"
boolean equals (String s)	returns true if the argument has the same length, and same characters as this String.	if (myString.equals("EXAMPLES")) // false
Boolean equalsIgnoreCase (String s)	the second version ignores the case.	
int compareTo (String s)	returns positive number, 0, or negative number if this String is greater than, equal to or less than the argument respectively	if (myString.compareTo("Yes")>0) // true
int compareToIgnoreCase (String s)		
char charAt (int index)	returns the char in this String, at the index position passed to the method. Note: Index positions always start at 0	char c = myString.charAt(6); // c = 'e'
int indexOf (int ch)	finds the first / last occurrence of a character or substring within this string, and returns the index. If the substring or char is not found -1 is returned.	int i = myString.indexOf("amp"); // i == 2
int lastIndexOf (int ch)		
//also overloaded to accept String parameter		if (myString.indexOf("Y") > 0) // returns -1 and is false
String substring (int from)	returns a part of the original String starting from the fromIndex to the end	String s=myString.substring(5) // returns "ample" and "les"

String substring (int from, int to)	of the string or up to but not including toIndex if one is given.	String s=myString.substring(2, 7); // returns "les"
String trim()	returns a String, produced by removing any whitespace characters from the beginning and end of this string.	String s = " 125 "; s = s.trim(); //returns "125"
static String valueOf (any primitive type)	is a static method, that is overloaded for all the primitive data types, and returns a String representation of the argument.	String s = String.valueOf(3.14F); // numbers to Strings "3.14"
String concat (String s) //equivalent to + symbol	append the argument string at the end of this string.	String s = myString.concat(" below") // returns "examples below"

Example 2:

The following example generates a password for a student using his initials and age.(note: do not use this for your actual passwords).

```
/* generates a password for a student using his initials and age */

public class MakePassword {

    public static void main(String[] args) {

        String firstName = "Amr";

        String middleName = "Samir";

        String lastName = "Al-Ibrahim";

        int age = 20;

        //extract initials

        String initials =

            firstName.substring(0,1)+

            middleName.substring(0,1)+

            lastName.substring(3,4);

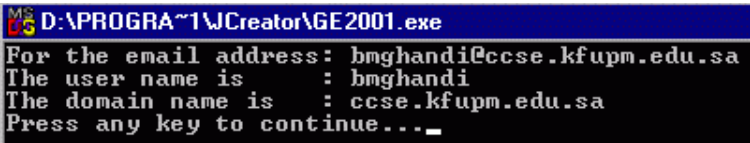
        //append age

        String password = initials.toLowerCase()+age;

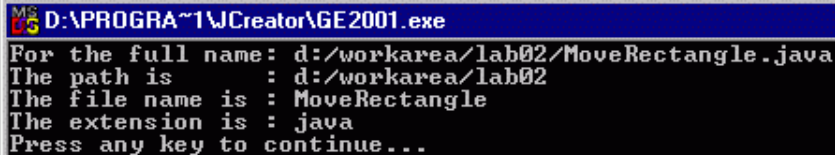
        System.out.printf("Your Password =%-20s",password);

    }
}
```

3. Assignments

- Write a program that will do the following:
 - Declare a double variable named miles with initial value of 10.5, increase its value by 1 using the increment operator. Show the result in the form:
 "The current value of miles is now: ... "
 - Declare X, Y, Z as doubles, a as an integer equal to 1, and d as a double equal to 1.0. Compute the values of the following expressions.
 $X = d + 43 \% 5 * (23 * 3 \% 2),$
 $Y = 1.5 * 3 + (++a),$ and
 $Z = 3 + d * d + 4.$
 - Declare R and p to be doubles with p storing the value 3.758. Compute and display the value of R, where $R = \log p.$
- Use either Math.PI or declare PI as constant and use it below:
 Write a program that will display the following:
 "The value of Cos 30° is ...",
 "The value of exp(15) is ..."
 "The value of Sin 30° is: ..."
 "The absolute value of k is ..."
 "The value of tan 30° is ..."
 where $k = 2 / 4 * 4 \% 3,$ Note also that $30^\circ = (PI * 30)/180$ radian
- Write a program that breaks a given email address into username and domain name


```

D:\PROGRAMS\1\Creator\GE2001.exe
For the email address: bmghandi@ccse.kfupm.edu.sa
The user name is      : bmghandi
The domain name is    : ccse.kfupm.edu.sa
Press any key to continue..._
  
```
- Write a program that breaks a given full file name into path, filename and extension.


```

D:\PROGRAMS\1\Creator\GE2001.exe
For the full name: d:/workarea/lab02/MoveRectangle.java
The path is       : d:/workarea/lab02
The file name is  : MoveRectangle
The extension is  : java
Press any key to continue...
  
```

4. Home Work

- The class Random of the java.util package can be used to create an object which can then be used to generate a random integer number from 0 to a given maximum.

Example:

```
Random rand = new Random();
int randomInt = rand.nextInt(1000);
```

Improve the password example by generating a random number, multiply the random number with the age before appending it to the initials.

- Write a program to evaluate and print the value of the following mathematical function :

$$f(x, y) = 2x^2 - y + 3$$

Vary the values of x and y according to the following table and prints the values of f(x,y) as shown in the table.

x	y	f(x,y)
3.0	3.0	
2.0	3.0	
-4.0	2.0	

Very important guidelines:

1. You should submit both printed copy and soft copy in a flash drive before the next lab.
2. All your programs files should be saved in a folder HW1 on your flash drive.
3. Code your programs according to the Java naming conventions -- check this out on my home page.
4. Indent your work so that content of a class and methods are pushed inside by a tab or at least three spaces.
5. Use comments at the beginning of each program to explain its purpose. Also include your name and ID number as part of the comment.
6. Use comments to explain each variable whose purpose cannot be determined from its name.

Department of Computer Science
Faculty of Physical Sciences
Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB03

Objectives:

To gain experience with:

- Reading input (characters, strings and numbers) from the keyboard

1. Reading Input

There are various ways to read input from the keyboard, the *java.util.Scanner* class is one of them.

The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace by default. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression.

There is a list of commonly used Scanner class methods:

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Example 1: Study, compile, run and observe the output of the following program

```
//ReadingInput.java
import java.util.Scanner;
import java.io.IOException;

public class ReadingInput {
    public static void main(String[] args) throws IOException {
        Scanner input = new Scanner(System.in);

        System.out.print("Please enter your name: ");
        String name = input.nextLine();

        System.out.print("Now enter your age: ");
        int age = input.nextInt();
    }
}
```

```

        System.out.print("Enter amount to donate: ");
        double amount = input.nextDouble();

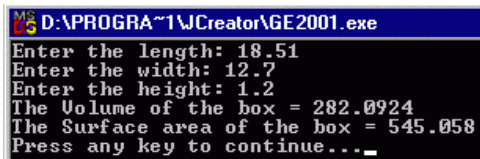
        System.out.printf("Mr. %s after one year you will be %d years\n",name, (age+1));
        System.out.printf("This is the amount you donated $%,.2f",amount);
    }
}

```

3. Assignments

1. Write a Java program that prompts for and reads the length, width, and height (in centimeters) of closed box. The program should then compute and display

- (a) the volume of the box.
- (b) the surface area of the box.

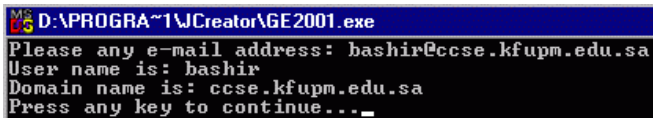


```

D:\PROGRAMS\1\JCreator\GE2001.exe
Enter the length: 18.51
Enter the width: 12.7
Enter the height: 1.2
The Volume of the box = 282.0924
The Surface area of the box = 545.058
Press any key to continue..._

```

2. Write a program that prompts for and read e-mail address a user. The program then prints the user name and the domain name on different lines. (Hint: use indexOf() and substring() method)



```

D:\PROGRAMS\1\JCreator\GE2001.exe
Please any e-mail address: bashir@ccse.kfupm.edu.sa
User name is: bashir
Domain name is: ccse.kfupm.edu.sa
Press any key to continue..._

```

2.

Write a program that prompts user to read the sides and compute the area, perimeter, longest, and smallest side of triangle. Compute and display the remainder when the longest side is divided by the smallest side. Print and format the output as

```

The perimeter of triangle is 11.00
The area of triangle with sides a=6, b=7, and c=9 is 25.69unit square
The longest side is 9
The smallest side is 6
The remainder is 3

```

Department of Computer Science

Faculty of Physical Sciences

Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB04

Objectives:

- Learn how to create classes that can be used in creating objects
- Understand the purpose of constructors and how they differ from methods
- Differentiate between static and instance variables (and methods)
- Differentiate between accessor and mutator methods

1. Classes as factory of objects

So far, all the classes we have been written are applications containing the main method. Another type of class is that which is written not necessarily to be executed as an application, but to be used in creating objects. Such type of classes are designed such that they describe the state and methods for a set of objects. This is achieved by using instance variables, instance methods and constructors.

Instance Variables: These are variables declared without using the **static** keyword. Such variables are only accessed through an object of the class and each object has its own copy of the variables – hence the name instance. Instance variables are used to store the state of an object and are usually declared as **private** so that they cannot be accessed directly by other objects.

Instance methods: These are methods declared without using the **static** keyword. They are normally used to describe the behavior of objects of a particular class. Instance methods are normally declared as **public** so that they could be accessed by other objects.

Constructor: These are used to initialize the fields (variables) of an object at the time the object is being created. Constructors must have the same name as the class, they have no return type and they are implicitly public.

Example 1: The following shows two classes. A class, Box, which specifies the fields, a constructor and methods of a box object, and a class, BoxDemo, which creates two Box objects and calls their methods.

```
public class Box {  
  
    private double length , width , height; //instance variables  
  
    public Box(double boxLength , double boxWidth , double boxHeight) { //constructor  
        length = boxLength;  
        width = boxWidth;  
        height = boxHeight;  
    }  
  
    public double volume() { //instance methods  
        return length * width * height;  
    }  
  
    public double surfaceArea() {  
        return 2*(length*width + length*height + width*height);  
    }  
}
```

```
public class BoxDemo {
```



```

public static void main(String[] args) {
    // Create two Box objects
    Box box1 = new Box(20.0, 10.0, 15.0);
    Box box2 = new Box(6.0, 4.0, 2.0);

    double volume, area;

    // Get and display volume and surfacearea of box1
    volume = box1.volume();
    area = box1.surfaceArea();
    System.out.println("The volume ofbox1 is " + volume + " cubic cm");
    System.out.println("The surfacearea of box1 is "+area+" square cm\n");

    // Get and display volume of surfacearea box2
    System.out.println("The volume ofbox2 is " + box2.volume() + " cubic cm");
    System.out.println("The surfacearea of box2 is "+area+" square cm\n");
}
}

```

2. Difference between static and instance variables

If a variable is declared as **static** then there will be only one copy of the variable, which can be access though the class name, even without creating any object of the class. Thus, such variables are sometimes called **class variables**. Such a variable cannot be used to store the state of an object since each object has its own state.

Example 2: The following example shows the difference between static and non-static variables. Notice that both the class Circle and the class CircleDemo are stored in the same file. In this case, only one of the classes should be declared as public – the one containing the main method. The file should also have the same name as this public class.

```

// Demonstrating the use of static variables.

class Circle {
    private static int numberOfCircles = 0;    //a static variable

    private double radius;                    //instance variables

    public Circle(double circleRadius) {
        numberOfCircles++;
        radius = circleRadius;
    }

    public double area() {
        return Math.PI * radius * radius;
    }

    public double circumference() {
        return 2.0 * Math.PI * radius;
    }

    public static int getNumberOfCircles() {
        return numberOfCircles;
    }
}

public class CircleDemo {
    public static void main(String[] args) {

        // The static method getNumberOfCircles can be called before creating any circle
        object
        System.out.println("The number ofcircles is " + Circle.getNumberOfCircles());

        Circle circle1 = new Circle(8.5);
        System.out.println("The number ofcircles is " + Circle.getNumberOfCircles());

        Circle circle2 = new Circle(5.0);
        System.out.println("The number ofcircles is " + Circle.getNumberOfCircles());
    }
}

```

```

        System.out.println("The area of thefirst circle is \t" + circle1.area() + "
square cm");
        System.out.println("The circumference of the first circle is\t" +
circle1.circumference() + " cm");

        System.out.println("The area of thesecond circle is\t" + circle2.area() + "
square cm");
        System.out.println("The circumference of the second circle is\t" +
circle2.circumference() + " cm\n\n");
    }
}

```

3. Accessor and Mutator methods

It is a good programming principle to always declare instance variables as private. However, it is O.K. to allow indirect access to these variables by providing public methods. Some of these method only access but do not change the variables, hence they are called **accessor** methods. Those that do not only access, but also make changes to these variables are called **mutator** methods.

Example 3: The follows shows example of accessor and mutator methods.

```

// Demonstrating using accessor and mutator methods.
class Rectangle {
    private double length , width;

    public Rectangle(double rectangleLength ,double rectangleWidth) {
        length = rectangleLength;
        width = rectangleWidth;
    }
    public double area() {
        return length * width;
    }

    // Accessor methods
    public double getLength() {
        return length;
    }
    public double getWidth() {
        return width;
    }

    // Mutator methods
    public void setLength(double newLength) {
        length = newLength;
    }
    public void setRectangleWidth(floatnewWidth) {
        width = newWidth;
    }
}

public class RectangleDemo {
    public static void main(String[] args) {
        // Create one Rectangle object
        Rectangle rectangle = new Rectangle(20.0, 10.0);

        // Get and display the area of rectangle
        double area = rectangle.area();
        System.out.println("The area of therectangle is\t" + area + " square cm");

        // Get and display the length ofrectangle
        //Note: we cannot use rectangle.length
        System.out.println("The length ofthe rectangle is\t" + rectangle.getLength() +
" cm");

        // Modify the length of rectangle1 to12.0
    }
}

```

```

        //We must use a mutator method to change the value of a private instance
variable
        rectangle.setLength(12.0);

        // Get and display the new length of the rectangle
        System.out.println("The new length of the rectangle is " +
rectangle.getLength() + " cm");

        // Get and display the new area of the rectangle
        System.out.println("The new area of the rectangle is " + rectangle.area() + "
square cm\n\n");
    }
}

```

4. Assignment.

- Open the folder **lab04** , Study, compile, and then execute each of the three demo applications:
 - BoxDemo
 - CircleDemo
 - RectangleDemo
- Write a java program containing two classes: BankAccount and BankAccountDemo. The class BankAccount must contain the following:
 - an instance variable **accountNumber** of type int.
 - an instance variable **customerName** of type String
 - an instance variable **balance** of type double.
 - a **constructor** that initializes each of the instance variables.
 - a method **deposit**
 - a method **withdraw**
 - a method **getBalance**

The class BankAccountDemo must:

· Prompt for and read the account number for the

 D:\PROGRA~1\VCrator\GE2001.exe

customer.

- Prompt for and read the name of the customer.
- Prompt for and read the initial balance for the customer.
- Create a bankAccount object for the customer.
- Prompt for and read an amount to be deposited.
- Display the current balance for the customer after the deposit.
- Prompt for and read an amount to be withdrawn.
- Display the current balance of the customer after the withdrawal.

```
Enter Account Number: 998877
Enter Customer name: Bashir
Enter initial balance: 2000

Enter amount to deposit: 500
New balance is: 2500.0

Enter amount to withdraw: 1000
New balance is: 1500.0
Press any key to continue...
```

3. Write two java classes, **Student** and **StudentDemo** in separate files. The class **Student** should contain the following:

- an instance variable **name** of type String.
- an instance variable **iDNumber** of type int
- three instance variables **quiz1**, **quiz2**, **quiz3** of type double.
- a **constructor** that initializes each of the instance variables.
- a **get method** for each of the instance variables.
- three **set method**, one for each of quiz1, quiz2 and quiz3
- a method **average** to return the average of the three quizzes
- a method **printDetails** to print the details of a student object in the following format:

Student Name: ??????????

Student ID: ??????????

Quiz Grades: ??????? ?????????? ??????????

The class StudentDemo must:

- Create an object of type Student, supplying values for name, ID number and three quizzes in the constructor call.

- calls the printDetails method to print the details of the student
- calls the average method to get the average and print it.
- Prompts for and read the new grade for quiz3
- calls the setQuiz3() method to change the value of quiz3 to the value entered by the user.
- prints the details and the average again.

```

MS-DOS D:\PROGRAMS\1\Creator\GE2001.exe
Student Name: Tareg
Stundet ID: 996753
Quiz Grades: 15.0      17.0      12.0
Average : 14.666666666666666

Enter new grade for quiz3: 20

Student Name: Tareg
Stundet ID: 996753
Quiz Grades: 15.0      17.0      20.0
Average : 17.333333333333332
Press any key to continue...
  
```

**Department of Computer Science
Faculty of Physical Sciences
Ahmadu Bello University, Zaria**

COSC 211 : Object Oriented Programming I - LAB05

Objectives:

To gain experience with:

- Selection Statement
- Iteration

1. Selection Statement

Three types of selection statements.

if statement.

- Performs an action, if a condition is true; skips it, if false.
- Single-selection statement—selects or ignores a single action (or group of actions).

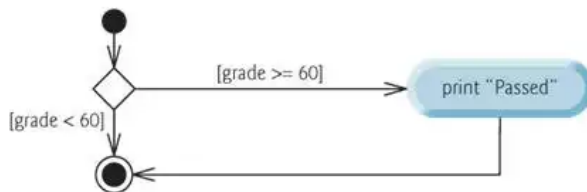
if...else statement:

- Performs an action if a condition is true and performs a different action if the condition is false.
- Double-selection statement—selects between two different actions (or groups of actions).

switch statement:

- Performs one of several actions, based on the value of an expression.
- Multiple-selection statement—selects among many different actions (or groups of actions).

if Single-Selection Statement.



Pseudocode:

*If student's grade is greater than or equal to 60
Print "Passed"*

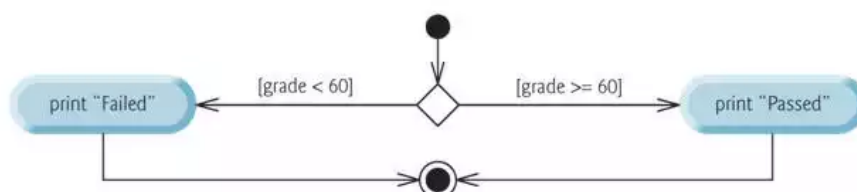
Java Code:

```
if ( studentGrade >= 60 )
```

```
System.out.println(  
"Passed" );
```

If the condition is false, the Print statement is ignored, and the next pseudocode statement in order is performed.

if Single-Selection Statement.



Java Code:

Pseudocode:	if (grade >= 60)
If student's grade is greater than or equal to 60	System.out.println(
Print "Passed"	"Passed");
Else	else
Print "Failed"	System.out.println(
	"Failed");
Note that the body of the else is also indented.	

Conditional operator (?:)—shorthand if...else.

Ternary operator (takes three operands)

boolean expression ? the value if the boolean expression is true : the value if the boolean expression evaluates to false

`System.out.println(studentGrade >= 60 ? "Passed" : "Failed");`

Example 1:

<pre>import java.util.Scanner; public class Tester { public static void main(String [] args) { int score ; char grade; Scanner input = new Scanner(System.in); System.out.println("Enter your score"); score = input.nextInt(); if (score >=90) grade = 'A'; else if (score >=80) grade = 'B'; else if (score >=70) grade = 'C'; else if (score >=60) grade = 'D'; else grade = 'F'; System.out.println("Your test score is "+ score + ", which is equivalent to the grade " + grade + "."); } }</pre>	<pre>import java.util.Scanner; public class Tester { public static void main(String [] args) { int score ; char grade; Scanner input = new Scanner(System.in); System.out.println("Enter your score"); score = input.nextInt(); if (score >=90) grade = 'A'; if (score >=80) grade = 'B'; if (score >=70) grade = 'C'; if (score >=60) grade = 'D'; else grade = 'F'; System.out.println("Your test score is "+ score + ", which is equivalent to the grade " + grade + "."); } }</pre>
Study, run, compile and run the above codes and observe the output	

[Click for nested if statements](#)

1. Iteration (Repetition)

Three repetition statements (also called looping statements) Perform statements repeatedly while a loop-continuation condition remains true.

while and *for* statements perform the action(s) in their bodies zero or more times. if the loop-continuation condition is initially false, the body will not execute

The *do...while* statement performs the action(s) in its body one or more times

3. Assignments

- (a) Write an application that will prompt user the to enter the first term, common difference, and the number of terms of an arithmetic progression (AP). It should compute the nth term of the series and the sum of the first n terms. Your code should ensure that the number of terms, n, is positive:
[Hint: use $T_n = a + (n - 1)d$, and $S_n = (n/2)(a + T_n)$, where a is the first term, n is the number of terms, d is the common difference, T_n is the nth term of the series, and S_n is

the sum of the firstn terms.]

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

2. A quadratic equation of the form $ax^2 + bx + c = 0$ has roots:

(a) Write a program that efficiently determines the values of the roots (*root1* and *root2*) of the given equation. Assume that all the variables have been declared as type *double*. Note that the roots are real and distinct, real and equal, or complex according to whether the discriminant ($b^2 - 4ac$) is positive, zero or negative, respectively. ;

Format the output in the following ways (where *root1* and *root2* are the calculated roots of the equation).

The roots are real and distinct :*root1,root2*

The roots are real and equal:*root1*

The roots are complex

(b) Re-write the example 1 above using *switch* statement

3. Write a program using the following pseudocode:

(a)

```
1 Set total to zero
2 set grade counter to one
3
4 While grade counter is less than or equal to zero
5   Prompt the user to enter the next grade
6   Input the next grade
7   Add the grade into the total
8   Add one to the grade counter
9
10 Set the class average to the total divided by ten
11 Print the class average
```

(b)

```
1 Initialize total to zero
2 Initialize counter to zero
3
4 Prompt the user to enter the first grade
5 Input the first grade (possibly the sentinel)
6
7 While the user has not yet entered the sentinel
8   Add this grade into the running total
9   Add one to the grade counter
10  Prompt user to enter the next grade
11  Input the next grade (Possibly the sentinel)
12
13 If the counter is not equal to zero
14   Set the average to the total divided by the counter
15   Print the average
16 else
17   Print "No grades were entered!"
```

(c)

```
1 Initialize passes to zero
2 Initialize failures to zero
3 Initialize student counter to one
4
5 While student counter is less than or equal to 10
6   Prompt the user to enter the exam result
7   Input the next exam result
8
9   If the student passed
10    Add one to passes
11  Else
12    Add one to failures
13
14  Add one to student counter
15
16 Print the number of passes
17 Print the number of failures
18
19 If more than eight students passed
20   Print "Excellent to instructor"
```

4. Home Work

1. The value of an investment of P nain after t years at an interest rate of $r\%$ compounded yearly is given by $P(1 + r/100)^t$. Write a program that will ask the user to input P , t and r , and will calculate and display the value of the investment. This should be done in a sentinel-controlled loop so that many such calculations can be performed.
2. The first term of a GP is 1.5, and its common ratio is 2. Write a program that will calculate the sum of ten terms. Do not use a formula.

Very important guidelines:

1. You should submit both printed copy and soft copy in a flash drive before the next lab.
2. All your programs files should be saved in a folder HW1 on your diskette.
3. Code your programs according to the Java naming conventions -- check this out on my home page.
4. Indent your work so that content of a class and methods are pushed inside by a tab or at least three spaces.
5. Use comments at the beginning of each program to explain its purpose. Also include your name and ID number as part of the comment.
6. Use comments to explain each variable whose purpose cannot be determined from its name.

Department of Computer Science

Faculty of Physical Sciences

Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB06

Objectives:

- Learn how to **overload** constructors and methods
- Learn how to use the **this** keyword
- Learn how to write and use the **toString()** method

1. Overloading

Overloading means having more than one constructor in a class or having more than one method with the same name in a class. In the case of constructor, the purpose of overloading is to allow the user to have as many options as possible when creating an object of the class thus making the class more flexible to use. In the case of methods, overloading allows the same name to be used for methods that perform similar tasks – Imagine having two plus operators, one for integer addition and another for double addition! The condition for overloading is that the overloaded methods and/or constructors must have different signatures. The signature of a constructor is determined by the **number**, the **type** and the **order** of its parameters.

Example 1: The following example implements the Employee class. Notice how the constructors are overloaded. Also notice the overloading of the deductions methods.

```
public class Employee1 {
    private int iDNumber;
    private String name;
    private double salary;

    public Employee1(int iD, String employeeName, double employeeSalary){
        iDNumber = iD;
        name = employeeName;
        salary = employeeSalary;
    }
    public Employee1(String employeeName, int iD, double employeeSalary){
        iDNumber = iD;
        name = employeeName;
        salary = employeeSalary;
    }
    public Employee1(int iD, String employeeName){
        iDNumber = iD;
        name = employeeName;
        salary = 0.0;
    }
    public Employee1(String employeeName, int iD){
        iDNumber = iD;
        name = employeeName;
        salary = 0.0;
    }
    public void setSalary(double employeeSalary) {
```

```

        salary = employeeSalary;
    }
    public int getIDNumber() {
        return iDNumber;
    }
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }
    public void deductions(double telephoneBills){
        salary -= telephoneBills;
    }
    public void deductions(double telephoneBills, double medicalBills){
        salary -= (telephoneBills + medicalBills);
    }
    public void raiseSalary(double percentIncrease) {
        salary += salary * percentIncrease/100;
    }
    public void printDetails() {
        System.out.println("\nID Number: "+iDNumber+"\nName: "+name+"\nSalary: "+salary);
    }
}

```

The following shows how the constructors and the overloaded methods in the above example may be used:

```

import java.util.Scanner;
public class TestEmployee1 {
    public static void main(String[] args){
        Scanner input = new Scanner(System.in);

        int number;
        String name;
        double salary;

        System.out.print("Enter Name for Employee 1: ");
        name = input.nextLine();
        System.out.print("Enter ID Number for Employee 1: ");
        number = input.nextInt();
        System.out.print("Enter Salary for Employee 1: ");
        salary = input.nextDouble();

        //any of the following constructors be used to create the object
        Employee1 emp1 = new Employee1(number, name, salary);
        // or Employee1 emp1 = new Employee1(name, number, salary);
        System.out.print("\nEnter Name for Employee 2: ");
        name = input.nextLine();
        System.out.print("Enter ID Number for Employee 2: ");
        number = input.nextInt();

        //if we do not know the salary, we can use one of the following constructors
        Employee1 emp2 = new Employee1(number, name);
        //or Employee1 emp2 = new Employee1(name, number);
        emp2.setSalary(emp1.getSalary());
        emp1.deductions(50);
        emp2.deductions(60, 40);
        emp1.printDetails();
        emp2.printDetails();
    }
}

```

2. The *this* keyword

this is an implicit reference variable (i.e variable that doesn't need to be declared)

that refers to the current object. At the point of defining the class, it is used for two purposes as follows:

- To refer to the instance variables of the class, especially when their names happen to be the same with parameters or local variables of a method or constructor.
- To call a constructor from within another constructor of the same class.

The advantages of using **this** as can be seen from the following example is that the program becomes shorter and that we do not have to think of different names for the parameters of constructors and methods.

Example 2: The following example modifies the above by using the **this** keyword

```
public class Employee2 {
    private int iDNumber;
    private String name;
    private double salary;

    public Employee2(int iDNumber, String name, double salary) {
        this.iDNumber = iDNumber;
        this.name = name;
        this.salary = salary;
    }
    public Employee2(String name, int iDNumber, double salary) {
        this(iDNumber, name, salary);
    }
    public Employee2(int iDNumber, String name) {
        this(iDNumber, name, 0.0);
    }

    public Employee2(String name, int iDNumber) {
        this(iDNumber, name, 0.0);
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public int getIDNumber() {
        return iDNumber;
    }
    public String getName() {
        return name;
    }
    public double getSalary() {
        return salary;
    }
    public void deductions(double telephoneBills) {
        salary -= telephoneBills;
    }
    public void deductions(double telephoneBills, double medicalBills) {
        salary -= (telephoneBills + medicalBills);
    }
    public void raiseSalary(double percentIncrease) {
        salary += salary * percentIncrease/100;
    }
    public void printDetails() {
        System.out.println("\nID Number: "+iDNumber+"\nName: "+name+"\nSalary: "+salary);
    }
}
```

To test the above class, you need to run the file **TestEmployee2.java** which is the same as **TestEmployee1.java** except of the creation of objects of type **Employee2** class instead of objects of class **Employee1**

3. The **toString()** method

Sometimes we would like to print the values of the instance variable of an object. One way of doing this is to have a method such as the **printDetails()** method in the above examples. However, this is not a good idea since

in java, there are different output targets such as graphical windows which uses `drawString()` method of `Graphics/Graphics2D` object or output files which require a different object instead of `System.out`. Thus, what is normally done is to provide `toString()` method which returns a representation of the object as a `String`. The application can then use the string returned by this method as it wishes. the name `toString()` is very special in that it does not need to be called explicitly like other methods. The java system automatically calls the `toString()` method whenever the object reference variable is used in an operation that requires a string.

Example 3: The file `Employee3.java` is the same as `Employee2.java` except for the replacement of the `printDetails()` method with `toString()` method.

```
public String toString() {  
    return "\nID Number: " + iDNumber + "\nName: " + name + "\nSalary: " + salary;  
}
```

The file `TestEmployee3.java` accordingly modifies `TestEmployee2.java` to print the two employee objects by implicitly calling the `toString()` method as follows.

```
. . . .  
System.out.println(emp1);  
System.out.println(emp2);  
. . . .
```

4. Assignment.

1. Download the folder **lab06**.

- a) Open the files `Employee1.java` and `TestEmployee1.java`, study them to understand what each is doing, then compile and execute `TestEmployee1.java`.
- b) Try the alternative constructor calls in the `TestEmployee1.java` by removing the comments on them and commenting the lines before each of them, then compile and execute the program again. You should notice no difference.
- c) Open the files `Employee2.java` and `TestEmployee2.java`, study them, then compile and execute `TestEmployee2.java`.
- d) Open the files `Employee3.java` and `TestEmployee3.java`, study them, then compile and execute `TestEmployee3.java`.

2. (a) Design a class called *Author* that contains the following.

- (i) Three private instance variables: *name (String)*, *email (String)*, and *gender (char of either 'm' or 'f')*.
- (ii) One constructor to initialize name, email and gender with given values.

(iii) Public getters and setters: `getName()`, `getEmail()`, `setEmail()`, and `getGender()`.

There are no setters for name and gender, as these attributes cannot be changed.

(iv) A `toString()` method that returns *"author-name (gender) at email"*, eg, *"Aliyu Garba (m) at galiyu@abu.edu.ng"*.

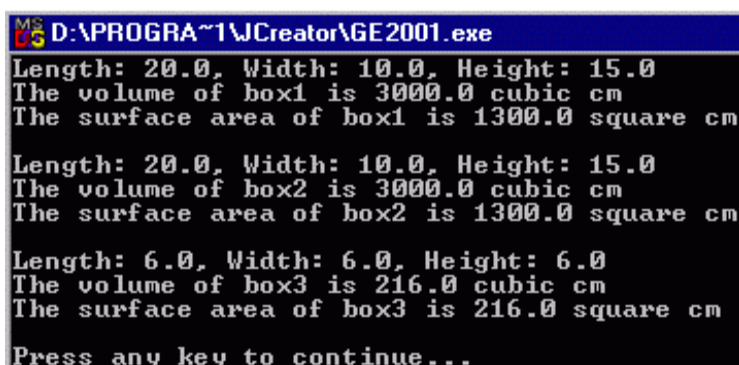
(b) Write a test program called *TestAuthor* to test the constructor and the public methods.

Try changing the email of an author.

3. Modify the *Box.java* and *BoxDemo.java* of Example 1 of lab04 as follows:

- a) add methods `getLength()`, `getWidth()` and `getHeight()` that returns the length, width and height of the box object.
- b) add another constructor which receives only the length. It then calls the other constructor supplying this length for all of length, width and height (i.e. it forms a cube).
- c) add another constructor that receives another Box object as parameter and then uses its length, width and height to initialize the current box object (i.e it creates a box object with same dimension as the one it receives as parameter)
- d) add a `toString()` method that returns the length, width, and height as a string that prints on one line as shown in the figure below.

The class *BoxDemo.java* should be modified to create three Box objects using each of the three constructors and then prints it (Note: no need to read input). Use the first box object to create the second.



```
MS D:\PROGRAMS\1\Creator\GE2001.exe
Length: 20.0, Width: 10.0, Height: 15.0
The volume of box1 is 3000.0 cubic cm
The surface area of box1 is 1300.0 square cm

Length: 20.0, Width: 10.0, Height: 15.0
The volume of box2 is 3000.0 cubic cm
The surface area of box2 is 1300.0 square cm

Length: 6.0, Width: 6.0, Height: 6.0
The volume of box3 is 216.0 cubic cm
The surface area of box3 is 216.0 square cm

Press any key to continue...
```

Department of Computer Science

Faculty of Physical Sciences

Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB07

Objectives:

To gain experience with:

- using arrays
- cloning arrays
- writing methods that receives arrays as parameters or have array as their return type
- using array of objects

1. Brief Review of Arrays

An array is a contiguous list of memory cells that can be identified using a single variable.

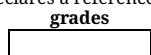
An array is declared as shown below

```
int[] grades = new int[10];
```

The above can also be broken into two steps:

1. Declaration of the array: `int[] grades;`

This declares a reference variable for an integer array (a variable that can hold the address of an integer array).



2. Creation of the array object: `grades = new int[10];`

This creates the array object in the memory (just like any object) and stores its address in the reference variable `grades`.



Notice that the index of an array begins from zero and ends one less than the size. Any attempt to access elements outside this range will make the Java interpreter throw `ArrayIndexOutOfBoundsException`.

Once an array is created, its individual elements can be accessed by using the array name followed by the index in square brackets. Example, to assign 15 to element number 6, we write:

```
grades[5] = 15;
```

Each array has an instance final variable `length` that holds the size of the array. This is very useful if we need to access each element of the array. For example, the following for loop initializes each element of the array to its index:

```
for (int i = 0; i < grades.length; i++)  
    grades[i] = i;
```

If we have the values to be assigned to an array are known, then an array can be created using an initializer list as follows:

```
char[] vowels = {'a', 'e', 'i', 'o', 'u'};
```

This declares and creates a character array of five elements and initializes it with the vowel characters. Notice that in this case, the size is not specified and the new operator is not used. The compiler automatically counts the elements, creates an array of the appropriate size and fills the array with the elements.

We can also create an array of objects. For example, the following declares an array to hold 10 student objects:

```
Student[] student = new Student[10];
```



However, unlike an array of primitive types where primitive values are stored directly into the array cells, an array of objects is used to store the references to the objects. For example, to store a reference to a student object in the first cell of the array, we create the student object as follows:

```
student[i] = new Student(. . .);
```

2. Example programs

Example 1: The following creates two arrays and prints their vector and dot products. Notice that the `vectorProduct()` method receives two array parameters and returns another array – the vector product of the two arrays.

```
public class Vector {  
  
    public static void main(String[] args) {  
  
        double[] x = {5, 2, -3, 4, -2};  
  
        double[] y = {2, 3, 5, -4, -6};  
  
        double[] z;  
  
  
        z = vectorProduct(x, y);  
  
    }  
}
```

```

System.out.println("The vector product is: ");
for(int k = 0; k < z.length; k++)
    System.out.print(z[k] + " ");
System.out.println("\n\nThe dot product is: "+dotProduct(x,y));
}

public static double[] vectorProduct(double[] a, double[] b) {
    double[] t = new double[a.length];
    for(int i = 0; i < a.length; i++)
        t[i] = a[i] * b[i];

    return t;
}

public static double dotProduct(double[] a, double[] b) {
    double sum=0;
    for(int i = 0; i < a.length; i++)
        sum = a[i] * b[i];

    return sum;
}
}

```

Example 2: The demonstrates array cloning.

```

public class ArrayCloning {
    public static void main(String[] args) {
        int k;
        double[] x = {5, 2, -3, 4, -2};
        double[] y = new double[5];
        double[] z = {10, 25, 30, 45};
        double[] w;

        y = x; // y and x refer to the same object. The object that was referenced by y is lost

        System.out.println("The array referenced by y is: ");
        for(k = 0; k < y.length; k++)
            System.out.print(y[k] + " ");

        x[0] = 200;
        x[4] = 66;
        System.out.println("\n\nThe array referenced by y is: ");
        for(k = 0; k < y.length; k++)
            System.out.print(y[k] + " ");

        w = (double[])z.clone(); // w and z refer to different objects

        z[0] = 88;
        z[2] = -99;
        System.out.println("\n\nThe array referenced by w is: ");
        for(k = 0; k < w.length; k++)
            System.out.print(w[k] + " ");
    }
}

```

Example 3: The following examples shows that we can have array as an instance variable. It also shows how array instance variable may be initialized by a constructor. Notice that the Student class is in its own separate file.

```

class Student {

```



```

private int iDNumber;

double[] quiz;

public Student(int iDNumber, double[] quiz) {
    this.iDNumber = iDNumber;
    this.quiz = quiz;
}

public int getID() {
    return iDNumber;
}

public double getQuiz(int quizNumber) {
    if(quizNumber >= 1 && quizNumber <= quiz.length)
        return quiz[quizNumber - 1];
    else
        return -1.0;
}

public void setQuiz(int quizNumber, double quizGrade) {
    if(quizNumber >= 1 && quizNumber <= quiz.length)
        quiz[quizNumber - 1] = quizGrade;
}

public double sum() {
    double sum = 0;
    for(int k = 0; k < quiz.length; k++)
        sum += quiz[k];

    return sum;
}

public double average() {
    return sum() / quiz.length;
}

public String toString() {
    String s = ""+iDNumber;
    for (int i = 0; i<quiz.length; i++)
        s += "\t"+quiz[i];
    s += "\t"+average();

    return s;
}
}

```

```

import java.util.Scanner;

public class TestStudent {

    public static void main(String[] args)    {
        double[] quiz = {50, 30, 60, 55};
        Student student = new Student(980000, quiz);

        /* destroy the object referenced by quiz, so that the only way to obtain quiz information is from the object referenced by student
        */
        quiz = null;

        System.out.println("Student information before changing grade:");
        System.out.println(student);

        student.setQuiz(3, 90);
        student.setQuiz(4, 75);

        System.out.println("\nStudent information after changing grades:");
    }
}

```

```

        System.out.println(student);
    }
}

```

Example 4: The following example shows how to use **array of objects**. It uses the same Student class as in example 3 above to create an **array** of three students objects, each one having ID number and three quizzes. The Program then prints the **average** of each students and the **overall** average for all students.

```

import java.util.Scanner;

public class TestStudent2 {
    static Scanner stdin = new Scanner(System.in);
    static final int STUDENT_COUNT = 3, QUIZZES_COUNT = 3;

    public static void main(String[] args) {
        Student[] student = new Student[STUDENT_COUNT];

        double sum=0;
        for(int i = 0; i < student.length; i++) {
            System.out.println("\nCreating student #" + (i+1));
            student[i] = createStudent();
        }

        System.out.println("\nID\tAverage");
        for(int i = 0; i < student.length; i++) {
            System.out.println(student[i].getID()+"\t"+student[i].average());
            sum += student[i].sum();
        }
        System.out.println("\nOverall average is: " + sum/(STUDENT_COUNT*QUIZZES_COUNT));
    }

    static Student createStudent() {
        int id;
        double[] quiz = new double[QUIZZES_COUNT];
        System.out.print("Enter student ID# ");
        id = stdin.nextInt();

        System.out.println("\nEnter three quizzes for this student");
        for (int i=0; i<quiz.length; i++) {
            System.out.print("Quiz#" + (i+1)+ " ");
            quiz[i] = stdin.nextDouble();
        }

        Student s = new Student(id, quiz);
        return s;
    }
}

```

3. Assignments

1. Modify example 3 so that after the **array** is created (as in example 3), it displays a menu that allows the operations shown in the following figure to be performed in a loop until the Quit option is chosen.

```

*****
*   Quiz Management System   *
*****
1. Show a quiz grade
2. Show all quizzes
3. Change a quiz grade
4. Show average
5. Quit
Your choice? : 2
980000 50.0   30.0   60.0   55.0   48.75
*****
*   Quiz Management System   *
*****
1. Show a quiz grade
2. Show all quizzes
3. Change a quiz grade
4. Show average
5. Quit
Your choice? :

```

Note: If the first option is chosen, your program should prompt for and read a quiz number. It then calls the `getQuiz()` method to print it. If option 3 is chosen, your program should prompt for and read the quiz number and the new grade for that quiz. It then calls the `setQuiz()` method to change the grade and then display the updated student information.

2. Modify example 4 as follows:
 - Add another method, `static double quizAverage(Student[] student, int k)` that receives an array of student objects and a quiz number as parameters. It then returns the average for that quiz.
 - Now modifies the main method so that it prints :
 - the ID number, grades and average of each student in a tabular form
 - the average of each quiz
 - the overall average for all students.

```

Creating student #1
Enter student ID# 984652

Enter three quizzes for this student
Quiz#1 10
Quiz#2 12
Quiz#3 14

Creating student #2
Enter student ID# 563745

Enter three quizzes for this student
Quiz#1 15
Quiz#2 14
Quiz#3 16

Creating student #3
Enter student ID# 764509

Enter three quizzes for this student
Quiz#1 20
Quiz#2 19
Quiz#3 18

ID      Quiz1  Quiz2  Quiz3  Average
984652  10.0   12.0   14.0   12.0
563745  15.0   14.0   16.0   15.0
764509  20.0   19.0   18.0   19.0
AVG:    15.0   15.0   16.0

Overall average is: 15.333333333333334
Press any key to continue...

```

3. Ahmadu Bello University Press wants to know the maximum, minimum, total, and average profit gained from years 2005 to 2014. Besides that, they are interested in knowing the difference between the maximum and minimum profit and those profits that are above average. The profits are as follows.

Year	Profit(₦)
2005	5,000,000.34
2006	2,005,000.00
2007	3,020,000.97
2008	5,057,800.20
2009	4,500,000.67
2010	5,000,000.00
2011	3,048,900.56
2012	4,800,000.50
2013	2,980,000.71
2014	4,909,000.80

- (a) Create a class called `Profit` that has the following members
 - i. Two array fields to store the profit and year named `profit` and `year` using appropriate data types.
 - ii. A method called `getMaxProfit()` that will return the maximum profit.
 - iii. A method called `getMinProfit()` that will return the minimum profit.
 - iv. A method called `getSumOfProfit()` that will return the total profit.
 - v. A method called `getAverageProfit()` that will return the average profit.
 - vi. A method called `getRange()` that will return the difference between the maximum and minimum profits.
 - vii. A method called `showProfitBelowAverage()` that will display all profits that are below average.
- (b) Create a test class that will instantiate the `Profit` class and display the required results with suitable headings and labels

Department of Computer Science

Faculty of Physical Sciences

Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB08

Objectives:

To gain experience with:

- Declaring, creating and accessing both regular and jagged 2-D arrays
- Using 2-D arrays in methods (as parameters and as return type)
- Cloning 2-D arrays

1. Brief Review of 2-D Arrays

Why 2D-arrays?

A 2D-array is used to store information that would otherwise require parallel 1D-arrays to store.

ID	quiz1	quiz2	quiz3	quiz4	quiz5	quiz6	quiz7
900000	50.5	40.0	60.0	0.0	55.0	30.0	48.0
920000	70.0	60.0	75.0	90.0	66.5	75.0	80.0
930520	65.0	70.0	65.0	80.0	78.0	50.0	69.0
940000	80.0	90.0	95.0	85.0	100.0	88.0	92.0
953478	40.0	30.0	50.0	55.0	45.0	35.0	0.0
972893	60.0	50.0	39.0	70.0	55.9	70.0	59.0

What is a Java 2D-array?

A Java 2D-array is an array of arrays of possibly different sizes.

Declaring 2D-arrays:

A 2D-array in which each row has the same number of elements is declared as:

```
type[ ][ ] arrayName = new type[numberOfRows][numberOfColumns];
```

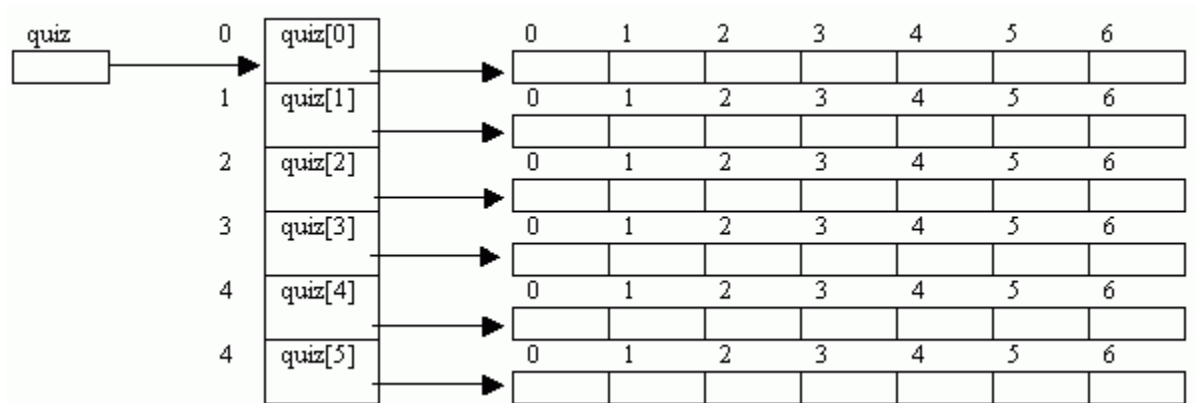
Example:

```
double[ ][ ] quiz = new double[6][7];
```

The above declaration is equivalent to either of the following:

<pre>double[][] quiz; quiz = new double[6][7];</pre>	<pre>double[][] quiz = new double[6][7]; for(int k = 0; k < quiz.length; k++)</pre>
---	---

quiz[0] = new double[7];	quiz[k] = new double[7];
quiz[1] = new double[7];	
quiz[2] = new double[7];	
quiz[3] = new double[7];	
quiz[4] = new double[7];	
quiz[5] = new double[7];	



Notes:

It is obvious from the above declarations that each row of a 2-D array, `quiz[i]` is an independent 1-D array. Thus, we could use it wherever a 1-D array is expected. For example, if there is a method `average` with the following header:

```
public static double average(double[] a) {
    . . .
}
```

Then to get the average of the i^{th} row, we could call it as:

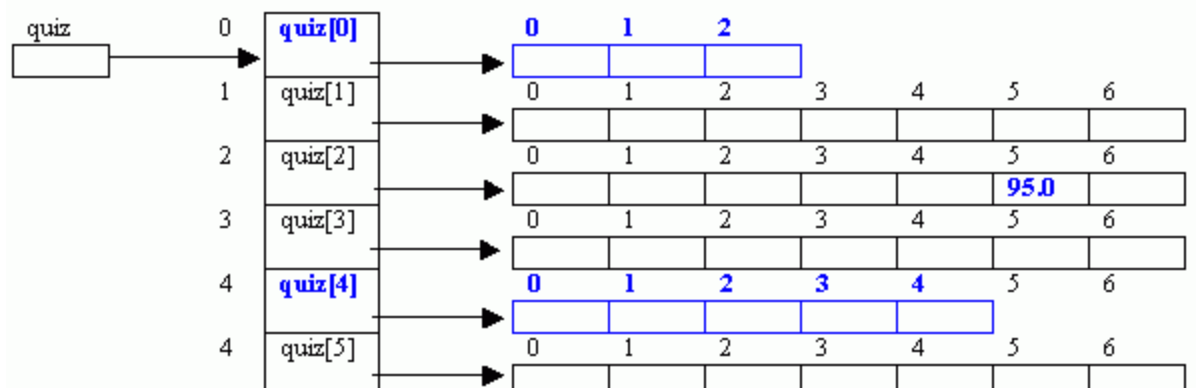
```
double avg = average(quiz[i]);
```

Declaring **rugged** 2D-arrays:

Because each row in a 2-D array is an independent 1-D array, it follows that the rows of a 2-D array do not need to be all of the same size. For example, if in the above declaration, we change the declaration of rows 0 and 4 as follows,

```
quiz[0] = new double[3];
quiz[4] = new double[5];
```

Then we have the following **rugged** array.



Accessing individual elements:

Individual element of a 2-D array is accessed by specifying the index of row and column. For example

```
quiz[2][5] = 95.0;
```

Declaring, Creating and Initializing a 2D-array in one step:

If we have the values, we could directly declare, create and initialize a 2-D array in one step. For example:

```
int[][] a = { {1, 0, 12, -1},
              {7, -3, 2, 5 },
              {-5, -2, 2, 9} };
```

Again, the rows do not have to be all of the same size. For example:

```
int[][] a = { {1, 0, 12, -1},
              {7, 5 },
              {-5, -2, 2, 9} };
```

Example 1: The following reads a rugged array one row at a time and then prints it.

```
import java.util.Scanner;
import java.util.StringTokenizer;

public class Rugged2DArray {
    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);

        int[][] a = new int[3][];
        a[0] = new int[5];
        a[1] = new int[3];
        a[2] = new int[4];

        int row, column;
        for(row = 0; row < a.length; row++) {
            System.out.println("Enter " + a[row].length + " elements for row#" + (row + 1));
            String inputLine = stdin.nextLine();
            StringTokenizer tokenizer = new StringTokenizer(inputLine);
            for(column = 0; column < a[row].length; column++)
                a[row][column] = Integer.parseInt(tokenizer.nextToken());
        }

        System.out.println("\nThe array elements are:\n");
    }
}
```

```

        for(row = 0; row < a.length; row++) {
            for(column = 0; column < a[row].length; column++)
                System.out.print(a[row][column] + "    ");
            System.out.println();
        }
        System.out.println("\n\n");
    }
}

```

2. Using 2-D arrays in methods

Like 1-D array, 2D-array can be passed as parameter to a method. Also, a method can return a 2-D array.

Example 2: The demonstrates array cloning.

```

import java.util.Scanner;
import java.util.StringTokenizer;

public class Matrix {
    static Scanner stdin = new Scanner(System.in);

    public static double[][] sum(double[][] a, double[][] b) {
        int row = a.length;
        int col = a[0].length;
        double[][] c = new double[row][col];

        for (int i=0; i<row; i++)
            for (int j=0; j<col; j++)
                c[i][j] = a[i][j] + b[i][j];

        return c;
    }

    public static double[][] createArray(int row, int col) {
        double[][] array = new double[row][col];

        System.out.println("Enter a "+row+" by "+col+" matrix row-wise");
        for (int i=0; i<row; i++) {

```

```

        String input = stdin.nextLine();
        StringTokenizer st = new StringTokenizer(input);
        for (int j=0; j<col; j++)
            array[i][j] = Double.parseDouble(st.nextToken());
    }
    return array;
}

public static void print(double[][] a) {
    int row = a.length;
    int col = a[0].length;

    for (int i=0; i<row; i++) {
        for (int j=0; j<col; j++)
            System.out.print(a[i][j]+ "\t");
        System.out.println();
    }
}

public static void main(String[] args) throws IOException {
    int row, column;
    System.out.print("Enter number of rows: ");
    row = Integer.parseInt(stdin.readLine());
    System.out.print("Enter number of columns: ");
    column = Integer.parseInt(stdin.readLine());

    double[][] a = createArray(row, column);
    double[][] b = createArray(row, column);
    double[][] c = sum(a,b);
    System.out.println("The sum of the two arrays is");
    print(c);
}
}

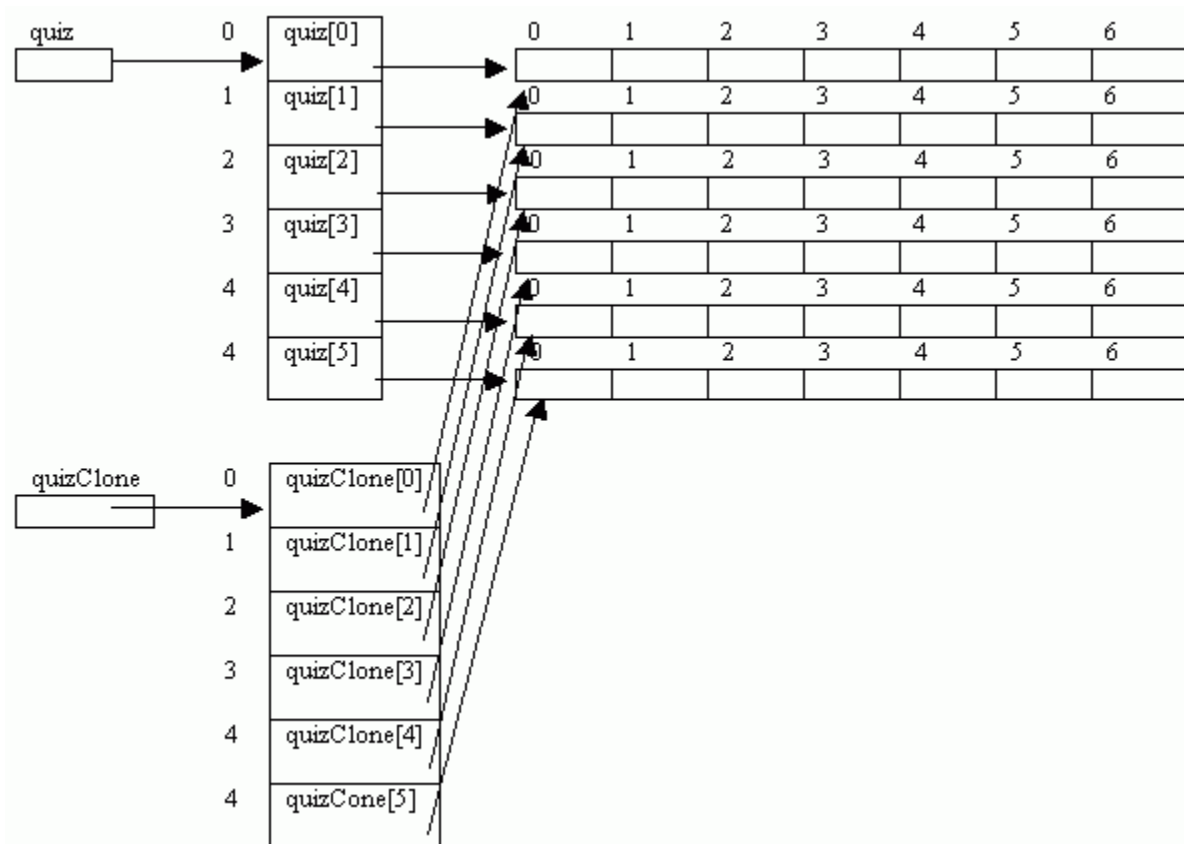
```

3. Cloning 2-D arrays

Like 1-D array, we can make a copy of an array using the clone method. For example, the 2D-array quiz can be clone using:


```
double[][] quizClone = (double[][]) quiz.clone;
```

However, the above statement only copies the references to the rows of quiz. The values of the actual elements are not copied. That is, any changes made to quiz will affect quizClone. This is **called shallow** cloning. The following figure shows the effect of shallow cloning:



To actually make a complete copy of quiz – called **deep cloning**, we need to also clone each row of quiz and assigned to the corresponding row of quizClone. Using the statements of the form: **`quizClone[i] = (double[][]) quiz[i].clone();`**

Example 3: The following shows shallow cloning.

```
public class ShallowCloning {
    public static void main(String args[]) {

        double[][] a = {{1, 2, 3},
                        {4, 5, 6},
                        {7, 8, 9}};

        System.out.println("Printing array a");
        Matrix.print(a);

        double[][] b = (double[][])a.clone();
```

```

System.out.println("Printing array b - clone of a");
Matrix.print(b);

b[1][1] = 20;
System.out.println("Printing array b after making chages to it");
Matrix.print(b);

System.out.println("Printing array a after changing its clone");
Matrix.print(a);
}
}

```

Example 4: The following shows deep cloning:

```

public class DeepCloning {
    public static void main(String args[]) {

        double[][] a = {{1, 2, 3},
                        {4, 5, 6},
                        {7, 8, 9}};

        System.out.println("Printing array a");
        Matrix.print(a);

        double[][] b = (double[][])a.clone();
        for (int i=0; i<b.length; i++)
            b[i] = (double[])a[i].clone();

        System.out.println("Printing array b - clone of a");
        Matrix.print(b);

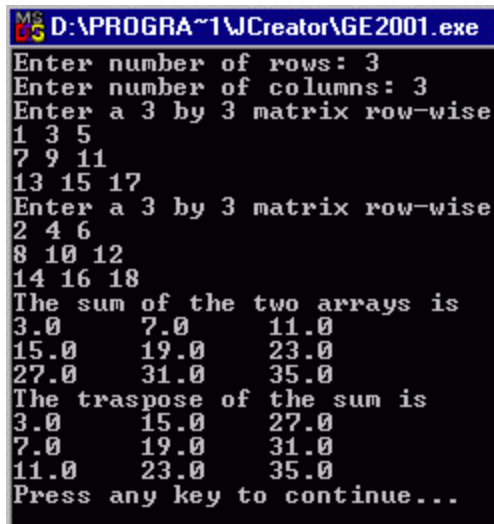
        b[1][1] = 20;
        System.out.println("Printing array b after making chages to it");
        Matrix.print(b);

        System.out.println("Printing array a after changing its clone");
        Matrix.print(a);
    }
}

```

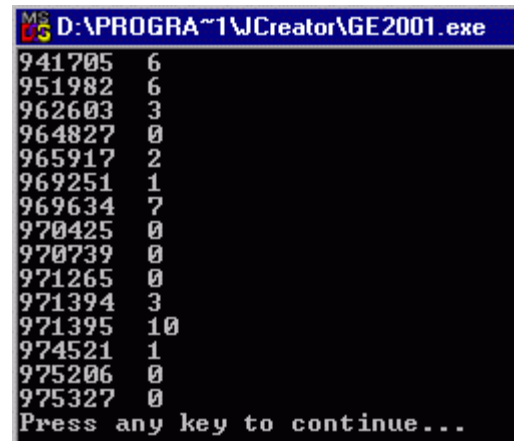
4. Assignments

1. Modify the Matrix class of Example2 by adding a static method **transpose**, that returns the transpose of a matrix it receives as parameter. Now call this method from the main method to get the transpose of the array c (the sum of the two matrices) and then print it. Fig 1 below shows the sample run of the modified program.



```
D:\PROGRA~1\JCreator\GE2001.exe
Enter number of rows: 3
Enter number of columns: 3
Enter a 3 by 3 matrix row-wise
1 3 5
7 9 11
13 15 17
Enter a 3 by 3 matrix row-wise
2 4 6
8 10 12
14 16 18
The sum of the two arrays is
3.0    7.0    11.0
15.0   19.0   23.0
27.0   31.0   35.0
The traspose of the sum is
3.0    15.0   27.0
7.0    19.0   31.0
11.0   23.0   35.0
Press any key to continue...
```

fig 1



```
D:\PROGRA~1\JCreator\GE2001.exe
941705 6
951982 6
962603 3
964827 0
965917 2
969251 1
969634 7
970425 0
970739 0
971265 0
971394 3
971395 10
974521 1
975206 0
975327 0
Press any key to continue...
```

fig 2

2. The file [attend.txt](#) contains attendance of 15 students in 20 lectures, with 1 representing present and 0 representing absent. Write a program that reads the data from the file into a 2-D integer array, **absences**, of size 15 by 21, storing the ID numbers in the first column and the absences in the remaining 20 columns. Your program should count the number of absences of each student and store the result in a 2-D integer array, **numberOfAbsences** of size 15 by 2, storing the ID in the first column and the number of absences in the second. Finally print the content of the array **numberOfAbsences**. Fig 2 above shows the sample run of the program.
3. Modify assignment 2 above so that the program prints only those students whose number of absences is more than 5. Your program must define and make use of the following methods:
 1. **public static int[][] createArray(String fname, int rows, int cols)**: That receives name of the file and the number of rows and columns as parameter, then opens the file and uses it to create a 2-D array of int and returns it.
 2. **public static int[][] countAbsences(double[][] absences)** to counts the absences of each student.

4. Home Work

A square matrix is said to be a magic square if it satisfies the following conditions:

The elements are unique

adding the elements in each row each column and each of the two diagonals gives the same result. For example the following is a magic square:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Write a program that first reads the number size of a square matrix, then reads elements of the matrix and prints one of the messages "IS A MAGIC SQUARE" or "IS NOT A MAGIC SQUARE" accordingly

Your program must include at least the following methods:

static int sumRow(int[][] a, int i) : That returns the sum of row i

static int sumColumn(int[][] a, int i) That returns the sum of columns i

static int sumDiagonal(int[][] a, boolean mainDiag) That returns the sum of the main diagonal if mainDiag is true or the sum of the minor diagonal if mainDiag is false.

static boolean isUnique(int[][] a): That returns true if all the elements are unique

static boolean isMagic(int[][] a) That returns true if the square is magic and false otherwise.

Department of Computer Science

Faculty of Physical Sciences

Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB09

Objectives:

To gain experience with:

- Reading input from a text file
- Writing/Appending output to text files
- String processing (breaking a string into tokens)

1. Reading Input from a file.

To read input from a file, we make use of the following classes all of which are in the **java.io** package:

FileReader: This is used to create an object that can be used to read one character at a time from a file. The format is:

```
FileReader reader = new FileReader("inputfile.txt");
```

Notes:

- If the file does not exist, a **FileNotFoundException** is thrown.
- If the file is not in the current folder, the full path of the file must be specified.
- When specifying the full path of the file, it is better to use forward slashExample:
"C:/workarea/inputfile.txt"
- The back slash can also be used, but double slash must be used for each slashExample:
"C:\\workarea\\inputfile.txt"

However, if the file name is being read from the user using a string variable, then only single slash is used.

- The important methods of the **FileReader** object are:

read()	returns an integer corresponding to the character read or -1 if there is no more data in the file.
close()	this closes the file. Always close a file after you finish reading from it.

Example 1: The following example reads one character at a time from a file and prints it on the screen.

```
import java.io.*;
class TestFileReader {
    public static void main(String[] args) throws IOException {
        char c;
        int input;
        FileReader in = new FileReader("inputfile.txt");
        while ((input=in.read()) != -1) {
            c = (char) input;
            System.out.print(c);
        }
        System.out.println();
        in.close();
    }
}
```

BufferedReader: This can be used together with a **FileReader** object to read one line at a time from a text file.

```
BufferedReader in = new BufferedReader(new FileReader("inputfile.txt"));
```

As we all know, the `BufferedReader` object has **`readLine()`** that returns a string. When used to read a file, the `readLine()` method returns a whole line. It returns null if there is no more data in the file.

Example 2: The following example reads one line at a time from a file and prints it on the screen.

```
import java.io.*;
class TestBufferedReader {
    public static void main(String[] args) throws IOException {
        String s;

        BufferedReader in = new BufferedReader(new FileReader("inputfile.txt"));
        while ((s=in.readLine()) != null)
            System.out.println(s);

        in.close();
    }
}
```

2. Writing/Appending output to text files.

To write or append output to a file, we make use of the following classes of the **`java.io`** package:

FileWriter: This is used to create an object that can be used to write one character at a time to a file. The format is:

```
FileWriter writer = new FileWriter("outputfile.txt"); //for writing
or
FileWriter writer = new FileWriter("outputfile.txt",true); //for appending
```

Notes:

- Unlike the case of `FileReader` object, If the file does not exist, it is created automatically. However, if there is no writing access to the file or its folder, **`FileNotFoundException`** is thrown.
- The main methods of the `FileWriter` object are:

<code>write(int c)</code>	writes the given character to the file.
<code>close()</code>	this closes the file. It is even more important to close an output file after its use.

Example 3: The following example reads one character at a time from an input file and prints it to an output file.

```
import java.io.*;
class TestFileReader {
    public static void main(String[] args) throws IOException {
        char c;
        int input;

        FileReader in = new FileReader("inputfile.txt");
        FileWriter out = new FileWriter("outputfile.txt");
        while ((input=in.read()) != -1) {
            c = (char) input;
            out.write(c);
        }
        System.out.println();
        in.close();
        out.close();
    }
}
```

PrintWriter: The problem with the **`write()`** method of the `FileWriter` object is that it can only write one character at a time. To be able to write/append strings, we create a `PrintWriter` object making use of a `FileWriter` object as a parameter:

```
PrintWriter out = new PrintWriter(new FileWriter("result.txt"));
or
PrintWriter out = new PrintWriter(new FileWriter("result.txt", true));
```

The `PrintWriter` object has **`print()`** and **`println()`** methods that behave in exactly the same manner as those of `System.out` object.

Example 4: The following example reads one line at a time from an input file and prints it to an output file.

```
import java.io.*;
```

```

class TestPrintWriter {
    public static void main(String[] args) throws IOException {
        String s;

        BufferedReader in = new BufferedReader(new FileReader("inputfile.txt"));
        PrintWriter out = new PrintWriter(new FileWriter("outputfile.txt"));
        while ((s=in.readLine()) != null)
            out.println(s);

        in.close();
        out.close();
    }
}

```

3. String Processing (StringTokenizer class).

The **readLine()** method of the **BufferedReader** object reads an entire line at a time. However, there are many situations where we would like to break the line into individual **tokens** for further processing. For example, if we read the string:

“996502 10 15 20”

we would like to break the string into the following:

“996502” “10” “15” “20”

so that we can call the appropriate parser method to convert the **tokens** to numbers.

To achieve this, Java provides the class **StringTokenizer** in the **java.util** package. To use this class, we need to first create its object using one of its constructors.

The following table shows the constructors and the important methods of the **StringTokenizer** class.

constructors and methods	description
StringTokenizer (String str)	Assumes white space as delimiters(“ \t\n\r”)
StringTokenizer (String str, String delim)	Uses the characters in the second argument as delimiters.
StringTokenizer (String str, String delim, boolean returnDelims)	If the third argument is true, include the delimiters are counted as part of the tokens.
String nextToken()	Returns the next token from this tokenizer's string
boolean hasMoreTokens()	Tests if there are more tokens from this tokenizer's string
int countTokens()	Returns the number of tokens remaining in this tokenizer's string.

Example 5: The following example reads ID numbers, names and grades of students in three quizzes contained in a file, **quizdata.txt** and computes the average of each student. It prints the output on both the screen and the file, **result.txt**

```

import java.io.*;
import java.util.StringTokenizer;

class ProcessQuiz {
    static final int QUIZ_COUNT = 3;
    public static void main(String[] args) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader("quizdata.txt"));
        PrintWriter out = new PrintWriter(new FileWriter("result.txt"));

        String s, name, idNumber;
        double[] quiz = new double[3];
        StringTokenizer st;

        while ( (s = in.readLine()) != null) {

```

```

        st = new StringTokenizer(s, "|");
        iDNumber = st.nextToken();
        name = st.nextToken();
        System.out.print(iDNumber+"\t"+name+"\t");
        out.print(iDNumber+"\t"+name+"\t");

        double sum = 0;
        for (int i = 0; i<quiz.length; i++) {
            quiz[i] = Double.parseDouble(st.nextToken().trim());
            sum += quiz[i];
            System.out.print(quiz[i]+"\\t");
            out.print(quiz[i]+"\\t");
        }

        System.out.println(sum/QUIZ_COUNT);
        out.println(sum/QUIZ_COUNT);
    }
    in.close();
    out.close();
}
}

```

Example 6: The following example uses the data from the file **quizdata.txt**, to create an array of Student objects. It then sends the output in two files **good.txt** containing students whose average is equal or greater than the overall average and **poor.txt** containing students whose grades is below the overall average. The Student class is the same as the one used in lab09, except it has been updated to include an instance variable **name**.

```

import java.io.*;
import java.util.StringTokenizer;

class GoodAndPoor {
    static final int QUIZ_COUNT = 3, MAX_STUDENT_COUNT = 30;

    public static void main(String[] args) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader("quizdata.txt"));
        PrintWriter good = new PrintWriter(new FileWriter("good.txt"));
        PrintWriter poor = new PrintWriter(new FileWriter("poor.txt"));
        Student[] student = new Student[MAX_STUDENT_COUNT];
        double [] quiz;

        String s, name;
        int iDNumber, actualStudentCount = 0;
        StringTokenizer st;
        double sum=0, average;

        while ( (s = in.readLine()) != null){
            st = new StringTokenizer(s, "|");
            iDNumber = Integer.parseInt(st.nextToken().trim());
            name = st.nextToken();
            quiz = new double[QUIZ_COUNT];
            for (int i=0; i<quiz.length; i++)
                quiz[i] = Double.parseDouble(st.nextToken().trim());

            student[actualStudentCount] = new Student(iDNumber, name, quiz);
            sum += student[actualStudentCount].average();
            actualStudentCount++;
        }
        average = sum/actualStudentCount;

        good.println("The class average is: "+average);
        good.println("\\r\\nStudents with grades equal or above average are:\\r\\n");

        poor.println("The class is: "+average);
        poor.println("\\r\\nStudents with grades equal or above average are:\\r\\n");

        for (int i=0; i<actualStudentCount; i++)
            if (student[i].average() >= average)
                good.println(student[i]);
            else

```



```

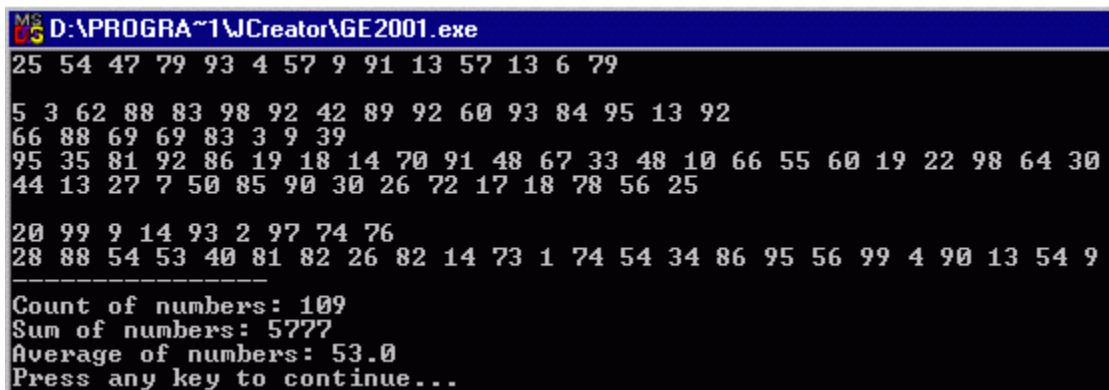
        poor.println(student[i]);

    in.close();
    good.close();
    poor.close();
}
}

```

4. Assignments

1. The file **integerdata.txt** contains integer numbers randomly arranged. Write a program that reads the data and prints the numbers as they appear in the file both on the screen and in the file **result2.txt**. Your program should also print the count of the numbers, their sum and their average

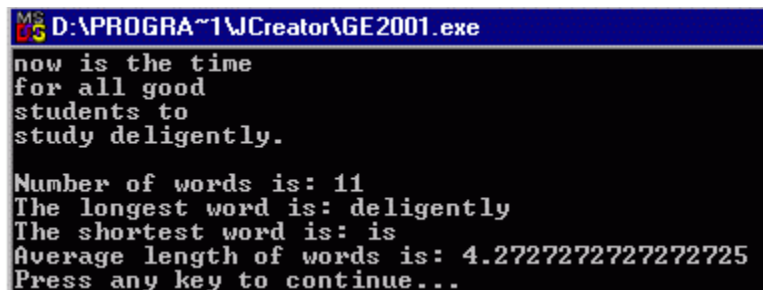


```

D:\PROGRA~1\JCreator\GE2001.exe
25 54 47 79 93 4 57 9 91 13 57 13 6 79
5 3 62 88 83 98 92 42 89 92 60 93 84 95 13 92
66 88 69 69 83 3 9 39
95 35 81 92 86 19 18 14 70 91 48 67 33 48 10 66 55 60 19 22 98 64 30
44 13 27 7 50 85 90 30 26 72 17 18 78 56 25
20 99 9 14 93 2 97 74 76
28 88 54 53 40 81 82 26 82 14 73 1 74 54 34 86 95 56 99 4 90 13 54 9
-----
Count of numbers: 109
Sum of numbers: 5777
Average of numbers: 53.0
Press any key to continue...

```

2. Write a program that reads data from the file **message.txt** and display on the screen, the content of the file, the number of words, the longest word, the shortest word and the average length of the words in the file. You should not count the punctuation characters “.,:?! ” as part of the a word.

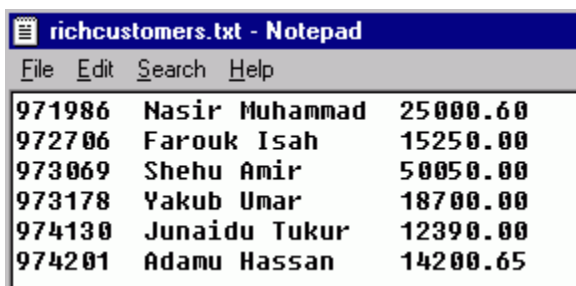


```

D:\PROGRA~1\JCreator\GE2001.exe
now is the time
for all good
students to
study diligently.
Number of words is: 11
The longest word is: diligently
The shortest word is: is
Average length of words is: 4.27272727272725
Press any key to continue...

```

3. Use the class **BankAccount** to write a program that creates an array of **BankAccount** objects using the data contained in the file **accounts.txt**. Your program then prints all those customers whose balance is greater than 10,000.00 into a file **richcustomers.txt**



File	Edit	Search	Help
971986	Nasir Muhammad	25000.60	
972706	Farouk Isah	15250.00	
973069	Shehu Amir	50050.00	
973178	Yakub Umar	18700.00	
974130	Junaidu Tukur	12390.00	
974201	Adamu Hassan	14200.65	

Department of Computer Science

Faculty of Physical Sciences

Ahmadu Bello University, Zaria

COSC 211 : Object Oriented Programming I - LAB10

Objectives:

- Learn to think recursively
- Learn how to write simple recursive methods

1. Thinking Recursively

It is natural to think of task that involves repetition in terms of loops. For example that task of computing the factorial of a number n , could be thought of as finding cumulative product of numbers from 1 to n . That is using the formula”

$$n! = 1 * \dots * n \text{ or } 1 \text{ if } n = 0.$$

Another approach to solving problems that involves repetition is **divide-and-conquer** approach or **recursion**. In this approach, the solution is defined in terms of a smaller version of the problem. This simplification of the problem is repeated until the smallest form of the problems is reached for which the solution is obvious. For example, the factorial of n could be defined as:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n > 0 \end{cases}$$

Some examples of problems that can be thought of in this way are described below:

$$\text{sumFromOneTo}(n) = \begin{cases} 0 & \text{if } n \leq 1 \\ \text{sumFromOneTo}(n-1) + n & \text{if } n > 1 \end{cases}$$

$$\text{arrayMinimum}(\text{array}, \text{start}) = \begin{cases} \text{array}[\text{start}] & \text{if } \text{start} = \text{length} - 1 \\ \min(\text{array}[\text{start}], \text{arrayMinimum}(\text{array}, \text{start} + 1)) & \text{if } \text{start} < \text{length} - 1 \end{cases}$$

$$\text{isMember}(\text{element}, \text{array}, \text{start}) = \begin{cases} \text{false} & \text{if } \text{start} \geq \text{length} \\ \text{true} & \text{if } \text{array}[\text{start}] == \text{element} \\ \text{isMember}(\text{element}, \text{array}, \text{start} + 1), & \text{if } \text{array}[\text{start}] != \text{element} \end{cases}$$

$$\text{printRange}(\text{start}, \text{stop}, \text{step}) = \begin{cases} \text{print nothing} & \text{if } \text{start} > \text{stop} \\ \text{print start; printRange}(\text{start} + \text{step}, \text{stop}, \text{step}) & \text{if } \text{start} \leq \text{stop} \end{cases}$$

2. Writing recursive methods in Java

One of the beauty of recursive algorithms is that they can be implemented almost directly in Java. The code is generally shorter than the loop version and it usually requires less number of variables and assignments. The following examples implements the algorithms described above.

Example 1: Computes the sum of integers from 1 to a given number, n.

Recursive	Iterative
<pre>import java.io.*; public class SumFromOneTo { public static void main(String args[]) { System.out.println(sumFromOneTo(10)); } public static int sumFromOneTo(int n) { if (n <= 0) return 0; else return sumFromOneTo(n-1)+n; } }</pre>	<pre>import java.io.*; public class IterativeSumFromOneTo { public static void main(String args[]) { System.out.println(sumFromOneTo(10)); } public static int sumFromOneTo(int n) { int sum=0; for (int i=1; i<=n; i++) sum+=i; return sum; } }</pre>

Example 2: Find the minimum element from an array.

Recursive
<pre>import java.util.Scanner; public class ArrayMinimum { public static void main(String[] args) { Scanner stdin = new Scanner(System.in); double[] number = new double[10]; for(int k = 0; k < number.length; k++) { System.out.print("Please enter element#" + (k + 1)+" ":); number[k] = stdin.nextDouble(); } System.out.println("The minimum value in the array is " + arrayMinimum(number, 0)); System.out.println(); } public static double arrayMinimum(double[] x, int start) { if (start == x.length-1) return x[x.length-1]; else</pre>

```

        return Math.min(x[start], arrayMinimum(x, start+1));
    }
}

```

Iterative

```

public static double arrayMinimum(double[] x) {
    double min = x[0];
    for (int i=1; i<x.length; i++)
        if (x[i] < min)
            min = x[i];
    return min;
}

```

Example 3: Check if an element is contained in an array.

Recursive

```

import java.util.Scanner;

public class Member {
    public static void main(String[] args) {
        Scanner stdin = new Scanner(System.in);

        double[] number = new double[10];
        for(int k = 0; k < number.length; k++) {
            System.out.print("Please enter element#" + (k + 1)+" : ");
            number[k] = stdin.nextDouble();
        }

        System.out.print("\nNow enter element to search for: ");
        double element = stdin.nextDouble();

        if (isMember(element, number, 0))
            System.out.println(element+ " is contained in the array");
        else
            System.out.println(element+ " is NOT contained in the array");
    }

    public static boolean isMember(double e, double[] x, int start) {
        if(start >= x.length)
            return false;
        else if (e == x[start])
            return true;
        else
            return isMember(e, x, start+1);
    }
}

```

Iterative

```

public static boolean isMember(double e, double[] x) {
    for (int i=0; i<x.length; i++)
        if (x[i] == e)

```

```

        return true;

    return false;

}

```

Example 4: Print numbers from a given starting value to a given stopping value incrementing using a given stepping value.

Recursive

```

import java.util.Scanner;

public class PrintRange {

    public static void main(String args[]) throws IOException {

        Scanner stdin = new Scanner(System.in);

        System.out.print("Enter starting value: ");

        int start = stdin.nextInt();

        System.out.print("Enter stoping value: ");

        int stop = stdin.nextInt();

        System.out.print("Enter stepping value: ");

        int step = stdin.nextInt();

        printRange(start, stop, step);

    }

    public static void printRange(int start, int stop, int step) {

        if (start <= stop) {

            System.out.println(start);

            printRange(start+step, stop, step);

        }

    }

}

```

Iterative

```

public static void printRange(int start, int stop, int step) {

    for (int i=start; i<stop; i+=step)

        System.out.println(i);

}

```

4. Assignments

- Write a recursive method that returns the sum of elements in an array of double. Test your method by writing a main method that creates an array of double of size 10, initialize it by reading data from the user and calling your method to compute the sum and print it. See fig 1 below:

```

MS-DOS Batch File D:\PROGRA~1\JCreator\GE2001.exe
Please enter element#1: 1
Please enter element#2: 4
Please enter element#3: 3
Please enter element#4: 2
Please enter element#5: 7
Please enter element#6: 6
Please enter element#7: 5
Please enter element#8: 7
Please enter element#9: 8
Please enter element#10: 9
The array sum is 52.0
Press any key to continue...

```

fig 1

```

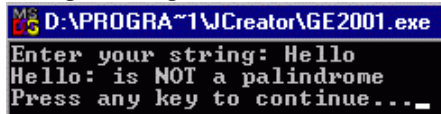
MS-DOS Batch File D:\PROGRA~1\JCreator\GE2001.exe
Please enter element#1: 1
Please enter element#2: 2
Please enter element#3: -5
Please enter element#4: -7
Please enter element#5: 2
Please enter element#6: -6
Please enter element#7: 9
Please enter element#8: 3
Please enter element#9: -5
Please enter element#10: 8
The array sum is 2.0
Number of positive elements is 6
Press any key to continue...

```

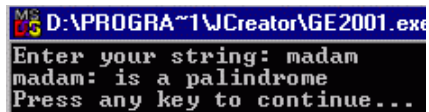
fig 2

- Improve your program in 1 above by writing another recursive method that counts the number of positive elements in the array. add a statement inside the main method to call this method so that you program prints both the sum and the number of positive elements in the array. See figure 2 above.

3. Write a recursive method that returns true if a string passed to it as parameter is a palindrome (palindrome is a string that reads the same both forward and backward). Test your method by writing a main method that reads a string from the user and then calls your method to know if the string is a palindrome and prints an appropriate message. See fig 3 below:

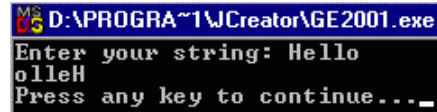


```
MS-DOS D:\PROGRAMS\1\Creator\GE2001.exe
Enter your string: Hello
Hello: is NOT a palindrome
Press any key to continue...
```



```
MS-DOS D:\PROGRAMS\1\Creator\GE2001.exe
Enter your string: madam
madam: is a palindrome
Press any key to continue...
```

fig 3



```
MS-DOS D:\PROGRAMS\1\Creator\GE2001.exe
Enter your string: Hello
olleH
Press any key to continue...
```

fig 4

4. Write a recursive method that prints a string passed to it as parameter in reverse order. Test your program by writing a main method that reads a string from the user and calls your method to print it in reverse order. See the fig 4. above.