LABORATORY MANUAL

CE2002 / CZ2002 : Object-Oriented Design & Programming

SCHOOL OF COMPUTER SCIENCE & ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

Lab 1
WINDOWS ENVIRONMENT, MY FIRST JAVA PROGRAM & FUNDAMENTAL PROGRAMMING

1. OBJECTIVE

The objectives of Lab 1 are (1) to learn the basic IDE editor ; (2) to edit a program, (3) to compile and execute a program and (4) write working fundamental programs. This aims to help students to familiarize themselves with the working programming environment and fundamental Java syntax.

You can use either Eclipse IDE to develop your Java programs.
Appendix A & B give a guide on using the Eclipse IDE.

2. My First Java Program

In this exercise, one is going to write his/her first Java program by typing in a sample Java program *MyFirstProgram.java*. You may save your works in your preferred directory. Refer to Appendix B as a guide.

2.1 Create a Java Source File

Type in the following program *MyFirstProgram.java* in the editor window and then save the program in your home directory. Note that the program must be saved as *MyFirstProgram.java.*
[[In Java, the file name (at this instance, MyFirstProgram) must be the same as the class name]]

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

     System.out.println("Hello! This is my first program.");
     System.out.println("Bye Bye!")

  }
}
```

Figure 1

3. *Your Tasks for this LAB*

3.1 Write a program that reads a character from the user and then uses a *switch* statement to achieve what the following *if* statement does.

```
if  ((choice == 'A')  || (choice == 'a'))
       printf("Action movie fan\n");
else if  ((choice == 'C')  || (choice == 'c'))
       printf("Comedy movie fan\n");
else if  ((choice == 'D')  || (choice == 'd'))
       printf("Drama movie fan\n");
else
       printf("Invalid choice\n");
```

Important: Remember to name the source code of this program as **P1.java** and name the compiled class code as **P1.class** inside the sub-directory *lab1*.

Test cases: 'a', 'A', 'c', 'C', 'd', 'D', 'b', 'B'.

Expected outputs: 'a', 'A' – Action movie fan; 'c', 'C' – Comedy movie fan; 'd', 'D' – Drame movie fan; 'b', 'B' – Invalid choice.

3.2 The salary scheme for a company is given as follows:

Salary range for grade A: $700 - $899
Salary range for grade B: $600 - $799
Salary range for grade C: $500 - $649

A person whose salary is between $600 and $649 is in grade C if his merit points are below 10, otherwise he is in grade B. A person whose salary is between $700 and $799 is in grade B if his merit points are below 20, otherwise, he is in grade A. Write a program to read in a person's salary and his merit points, and displays his grade.

Important: Remember to name the source code of this program as **P2.java** and name the compiled class code as **P2.class** inside the sub-directory *lab1*.

Test cases: (1) salary : $500, merit : 10; (2) salaray : $610, merit : 5; (3) salary : $610, merit : 10; (4) salary : $710, merit : 15; (5) salary : $710, merit : 20; (6) salary : 800, merit : 30.

Expected outputs: (1) salary : $500, merit : 10 – Grade C; (2) salaray : $610, merit : 5 – Grade C; (3) salary : $610, merit : 10 – Grade B; (4) salary : $710, merit : 15 – Grade B; (5) salary : $710, merit : 20 – Grade A; (6) salary : 800, merit : 30 – Grade A.

3.3 Write a program to generate tables of currency conversions from Singapore dollars to US dollars. Use title and column headings. Assume the following conversion rate:

1 US dollar(US$) = 1.82 Singapore dollars (S$)

Allow the user to enter the starting value, ending value and the increment between lines in S$. The starting value, ending value and the increment are all integer values. Generate three output tables using the following loops with the same input data from the user:

1. Use a *for* loop to generate the first table;
2. Use a *while* loop to generate the second table; and
3. Use a *do/while* loop to generate the third table.

Place all the codes in the main() method.

Important: Remember to name the source code of this program as **P3.java** and name the compiled class code as **P3.class** inside the sub-directory *lab1*.

Test cases: (1) starting : 1, ending : 5, increment : 1; (2) starting : 0, ending : 40, increment: 5; (3) starting : 40, ending : 0, increment: 5 (treat this case as an error).

Expected outputs:
(1) starting : 1, ending : 5, increment : 1;

```
US$          S$
--------------
1            1.82
2            3.64
3            5.46
4            7.28
5            9.1
```

(2) starting : 0, ending : 40, increment: 5;

```
   US$       S$
   --------------
   0         0.0
   5         9.1
   10        18.2
```

```
15        27.3
20        36.4
25        45.5
30        54.6
35        63.7
40        72.8
```

(3) starting : 40, ending : 0, increment: 5 (treat this case as an error) – Error input!!

4.4 Write a program that reads the height from a user and prints a pattern with the specified height. For example, when the user enters height = 3, the following pattern is printed:

AA
BBAA
AABBAA

If the height is 7, then the following pattern is printed:

AA
BBAA
AABBAA
BBAABBAA
AABBAABBAA
BBAABBAABBAA
AABBAABBAABBAA

Important: Remember to name the source code of this program as **P4.java** and name the compiled class code as **P4.class** inside the sub-directory *lab1*.

Test cases: 0, 3, 7

Expected outputs: (1) height = 0 – Error input!! (2) height = 3 & (3) height = 7 – same as the sample patterns.

LAB 2: METHODS

1. OBJECTIVE

The objectives of Lab2 are to practise on Java methods.

2. INTRODUCTION

A method is an independent collection of source code designed to perform a specific task. By dividing a problem into sub_problems and solve the sub_problems by methods, we obtain a program which is better structured, easier to test, debug and modify. We will practise on writing methods in Java in this lab.

3. *Your Tasks for this LAB*

3.1 In your preferred directory with sub-directory *lab2*, creatre and save the source code into the file Lab2p1.java, and generate the compiled class code as Lab2p1.class.

You may use the program template in Figure 1 to test your methods developed in this lab. The program contains a **main()** which includes a switch statement so that the following methods can be tested by the user. Write the code for each method and use the suggested test cases to test your code for correctness.

```
import java.util.Scanner;
public class Lab2p1 {
   public static void main(String[] args)
   {
      int choice;
      Scanner sc = new Scanner(System.in);
      do {
            System.out.println("Perform the following methods:");
            System.out.println("1:  miltiplication test");
            System.out.println("2:  quotient using division by subtraction");
            System.out.println("3:  remainder using division by subtraction");
            System.out.println("4:  count  the  number of digits");
            System.out.println("5:  position of a digit");
            System.out.println("6:  extract all odd digits");
            System.out.println("7:  quit");
            choice = sc.nextInt();

            switch (choice) {
               case 1: /* add mulTest() call */
                     break;
               case 2: /* add divide() call */
                     break;
               case 3: /* add modulus() call */
                     break;
               case 4: /* add countDigits() call */
                     break;
               case 5: /* add position() call */
                     break;
               case 6: /* add extractOddDigits() call */
                     break;
```

```
                case 7: System.out.println("Program terminating ….");
            }
        } while (choice < 7);
    }

    /* add method code here */

}
```

Figure 1: Program template for Lab 2.

3.2 Write a method that is to test students ability to do multiplication. The method will ask a student 5 multiplication questions one by one and checks the answers. The method prints out the number of correct answers given by the student. The method *random()* from the *Math* class of the Java library can be used to produce two positive one-digit integers (i.e. 1,2,3,4, …) in each question. A sample screen display when the method is called is given below:

How much is 6 times 7? *42*
How much is 2 times 9? *18*
How much is 9 times 4? *36*
……..
4 answers out of 5 are correct.

The input which is underlined is the student's answer to a question. The method header is:

```
public static void mulTest()
```

Test cases: (1) give 5 wrong answers; (2) give 1 correct answer; (3) give more than 1 correct answer.

Expected outputs: straightforward.

3.3 Write the method divide() which does division by subtraction and returns the quotient of dividing m by n. Both m and n are positive integers (i.e. 1,23,4,…). Division by subtraction means that the division operation is achieved using the subtraction method. For example, divide(12,4) will be performed as follows: 12-4=8, 8-4=4, and then 4-4=0, and it ends and returns the result of 3 as it performs three times in the subtraction operation. No error checking on the parameters is required in the method. The method header is given below:

```
public static int divide(int m, int n)
```

Test cases: (1) m = 4, n = 7; (2) m = 7, n = 7; (3) m = 25, n = 7.

Expected outputs: (1) 4/7 = 0; (2) 7/7 = 1; (3) 25/7 = 3.

3.4 Write the method modulus() which does division by subtraction and returns the remainder of dividing m by n. Both m and n are positive integers. No error checking on the parameters is required in the method. The method header is given below.

```
public static int modulus(int m, int n)
```

Test cases: (1) m = 4, n = 7 (2) m = 7, n = 7 (3) m = 25, n = 7.

Expected outputs: 4 % 7 = 4; (2) 7 % 7 = 0; (3) 25 % 7 = 4.

3.5 Write a method to count the number of digits for a positive integer (i.e. 1,2,3,4,…). For example, 1234 has 4 digits. The method countDigits() returns the result. The method header is given below:

public static int countDigits(int n)

Test cases: (1) n : -12 (give an error message); (2) n : 123; (3) n : 121456;

Expected outputs: (1) n : -12 - Error input!! (2) n : 123 - count = 3; (3) n : 121456 - count = 6.

3.6 Write the method position() which returns the position of the first appearance of a specified digit in a positive number n. The position of the digit is counted from the right and starts from 1. If the required digit is not in the number, the method should return -1. For example, position(12315, 1) returns 2 and position(12, 3) returns -1. No error checking on the parameters is required in the method. The method header is given below:

public static int position(int n, int digit)

Test cases: (1) n : 12345, digit : 3; (2) n : 123, digit : 4; (3) n : 12145, digit : 1;

Expected outputs: (1) position = 3; (2) position = -1; (3) position = 3.

3.7 Write a method extractOddDigits() which extracts the odd digits from a positive number n, and combines the odd digits sequentially into a new number. The new number is returned back to the calling method. If the input number n does not contain any odd digits, then returns -1. For examples, if n=1234567, then 1357 is returned; and if n=28, then –1 is returned. The method header is given below:

public static long extractOddDigits(long n)

Test cases: (1) n : 12345; (2) n : 54123; (3) n : 246; (4) n : -12 (give an error message)

Expected outputs: (1) oddDigits = 135; (2) oddDigits = 513; (3) oddDigits = -1; (4) oddDigits = Error input!!

Lab 3 : CLASSES & OBJECTS

1. OBJECTIVE

The objective of Lab3 is to practise on processing array of objects.

2. INTRODUCTION

Very often a program needs to process information in an array of objects in several ways. We will practise on writing array of objects in this lab.

Before you start your task, below is a brief introduction to Java syntax and constructs for arrays :

- **Arrays in Java (and C/C++) are indexed from *0* to SIZE-1.**

- **Creating** arrays with 12 elements of integer type that stores the number of days in each month (initialized to 0):

```
int [ ] days ;
days = new int[12] ;     // 2 lines code
```

OR

```
int [ ] days = new int[12];  // one line code
```

- **Assign a value to an array element :**

```
e.g.    days[0] = 31 ;
        days[1] = 28 ;
```

OR

```
int[ ] days = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } ;
```

- **Accessing array elements:**

```
Ex 1.  sales[0] = 143.50 ;

Ex 2.  if ( sales[23] == 50.0 ) ...

Ex 3.  sales[8] = sales[5] – sales[2] ;

Ex 4.  while ( sales[364] != 0.0 ) {...}

Ex 5.  for ( int i=0 ; i < 365 ; i++ )
           sales[ i ] = 0 ;
```

- **Java stores the size of the array automatically in an instance variable named *length***

e.g.

```
public class PrintingDays
{
  public static void main( String[] args )
  {
    int i;
    int[] days = {31,28,31,30,31,30,31,31,30,31,30,31};
    // print the number of days in each month
    for ( i=0 ; i < days.length ; i++ ) // traverse array
      System.out.println( "Month " + (i+1) +
        " has " +  days[i] + " days." );
  }
}
```

```
Program Output
Month 1 has 31 days.
Month 2 has 28 days.
...
Month 12 has 31 days.
```

- **Creating an array of *objects*:**

e.g. Rectangle[ ] rectArray = new Rectangle[5]; // allocate 5 spaces to hold the elements
rectArray[0] = new Rectangle(15,10); // assign Rectangle object to the space
rectArray[1] = new Rectangle(10,20); //
……

3. *Your Tasks for this LAB*

The Colossus Airlines fleet consists of one plane with a seating capacity of 12. It makes one flight daily. In this experiment, you are required to write a seating reservation application program. The problem specification is given below:

A. Write a class *PlaneSeat* that has the following features. Each *PlaneSeat* object should hold a seat identification number (seatId), a marker (assigned) that indicates whether the seat is assigned and the customer number (customerId) of the seat holder. The class diagram is given below:

PlaneSeat
- seatId: int - assigned: boolean - customerId: int
+ PlaneSeat(seat_id: int) + getSeatID(): int + getCustomerID(): int + isOccupied(): boolean + assign(cust_id: int): void + unAssign(): void

where

PlaneSeat() - is the constructor for the class.
getSeatID() – a get method that returns the seat number.
getCustomerID() – a get method that returns the customer number.
isOccupied() – a method that returns a boolean on whether the seat is occupied.
assign() – a method that assigns a seat to a customer.
unAssign() – a method that unassigns a seat.

Implement the class PlaneSeat.

B. Write a class *Plane* that comprises 12 seats. The class should create an array of 12 objects from the class *PlaneSeat*.

The class diagram is given below:

Plane
- seat: PlaneSeat[ ] - numEmptySeat: int
+ Plane() - sortSeats() : PlaneSeat[ ] + showNumEmptySeats(): void + showEmptySeats(): void + showAssignedSeats(bySeatId : boolean): void + assignSeat(seatId : int, cust_id : int): void + unAssignSeat(seatId : int): void

where

seat – instance variable containing information on the seats in the plane. It is declared as an array of 12 seat objects.
numEmptySeat – instance variable containing information on the number of empty seats.
Plane() – a constructor for the class Plane.
sortSeats() – a method to sort the seats according to ascending order of customerID.
A copy of the original seat array is used for sorting instead of the original.
showNumEmptySeats() – a method to display the number of empty seats.
showEmptySeats() – a method to display the list of empty seats.
showAssignedSeat() – a method to display the assigned seats with seat ID and customer ID.
If ***bySeatId*** is true, the order will be by seatID, else order is by customerID.
assignSeat() – a method that assigns a customer ID to an empty seat .
unAssignSeat() – a method that unassigns a seat.

Implement the class Plane.

C. Write an application class *PlaneApp* that implements the seating reservation program.

The class *PlaneApp* should be able to support the following:
(1) Show the number of empty seats
(2) Show the list of empty seats
(3) Show the list of customers together with their seat numbers in the order of the seat numbers
(4) Show the list of customers together with their seat numbers in the order of the customer ID
(5) Assign a customer to a seat
(6) Remove a seat assignment

The menu should also contain option (7) (i.e. quit) for terminating the program. After the user selects a particular option, the corrsponding operation will be executed. If the selected option is not (7), then the program shows the menu for user selection again. This application does not need to save data into a file between runs.

Important:
Remember to do all the programming inside the sub-directory *lab6* and name the source codes as **PlaneSeat.java**, **Plane.java** and **PlaneApp.java** and name the compiled codes as **PlaneSeat.class**, **PlaneSeat.class** and **PlaneApp.class**.

Test Data:
Test your application program with the following data:
1. Assign a customer to a seat with SeatID=10, CustomerID = 10001.
2. Assign a customer to a seat with SeatID=12, CustomerID = 10002.
3. Assign a customer to a seat with SeatID=8, CustomerID = 10003.
4. Show the the list of customers together with their seat numbers in the order of the seat numbers.
5. Show the number of empty seats.
6. Show the list of empty seats.
7. Assign (attempt) a customer to a seat with any existing CustomerID, and SeatID. (Should give a warning message!)
8. Remove the seat assignment with SeatID=10.

9. Assign (attempt) a customer to a seat with SeatID = 12.
10. Remove the seat assignment with SeatID=12.
11. Show the list of customers together with their seat numbers in the order of the seat numbers.
12. Show the number of empty seats.
13. Show the list of empty seats.
14. Quit

Expected outputs:

```
(1) Show number of empty seats
(2) Show the list of empty seats
(3) Show the list of seat assignments by seat ID
(4) Show the list of seat assignments by customer ID
(5) Assign a customer to a seat
(6) Remove a seat assignment
(7) Exit

  Enter the number of your choice: 5
Assigning Seat ..
  Please enter SeatID: 10
  Please enter Customer ID: 10001
Seat Assigned!

  Enter the number of your choice: 5
Assigning Seat ..
  Please enter SeatID: 12
  Please enter Customer ID: 10002
Seat Assigned!

  Enter the number of your choice: 5
Assigning Seat ..
  Please enter SeatID: 8
  Please enter Customer ID: 10003
Seat Assigned!

   Enter the number of your choice: 3
The seat assignments are as follow:
SeatID 8 assigned to CustomerID 10003.
SeatID 10 assigned to CustomerID 10001.
SeatID 12 assigned to CustomerID 10002.

   Enter the number of your choice: 4
The seat assignments are as follow:
SeatID 10 assigned to CustomerID 10001.
SeatID 12 assigned to CustomerID 10002.
SeatID 8 assigned to CustomerID 10003.
  Enter the number of your choice: 1
There are 9 empty seats

  Enter the number of your choice: 2
The following seats are empty:
SeatID 1
SeatID 2
SeatID 3
SeatID 4
SeatID 5
SeatID 6
SeatID 7
SeatID 9
SeatID 11

  Enter the number of your choice: 5
Assigning Seat ..
  Please enter SeatID: 8
  Please enter Customer ID: 10004
Seat already assigned to a customer.

  Enter the number of your choice: 6
  Enter SeatID to unassign customer from: 10
Seat Unassigned!
```

```
  Enter the number of your choice: 5
Assigning Seat ..
  Please enter SeatID: 12
  Please enter Customer ID: 10005
Seat already assigned to a customer.

  Enter the number of your choice: 6
  Enter SeatID to unassign customer from: 12
Seat Unassigned!

  Enter the number of your choice: 3
The seat assignments are as follow:
SeatID 8 assigned to CustomerID 10003.

   Enter the number of your choice: 1
There are 11 empty seats

  Enter the number of your choice: 2
The following seats are empty:
SeatID 1
SeatID 2
SeatID 3
SeatID 4
SeatID 5
SeatID 6
SeatID 7
SeatID 9
SeatID 10
SeatID 11
SeatID 12

  Enter the number of your choice: 7
```

Lab 4 : INHERITANCE & POLYMORPHISM

1. OBJECTIVE

The objective of Lab 4 is to practise on class inheritance and polymorphism.

Notes:
It is intended to be a short lab so that teams can get together to discuss/work on the assignment project which would have been published.

2. INTRODUCTION

Inheritance and Polymorphism are 2 important concepts in Object-Oriented Design and Programming. In this lab, you will get to learn more about the concepts in action. In one the tasks, you will need to decide whether ***concrete class, abstract class or interface*** is appropriate for the task required. In addition, you will also need to decide on the appropriate ***'is a'*** **or** ***'has a'*** relationship (inheritance/generalization VS delegation/object composition).

3. *Your Tasks for this LAB*

3.1 Polymorphic Sorting

Hints :
This is lab does not need a lot of coding, it is just a matter of where needs to be changed or added additional codes.

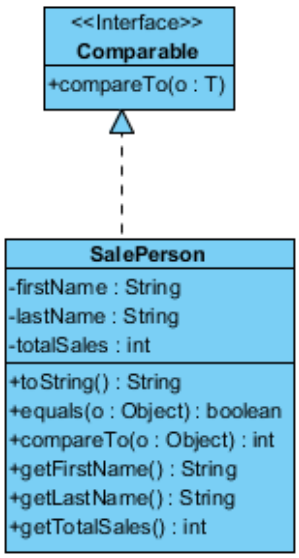
The file *Sorting.java* contains the Sorting class (in attachment). This class implements both the selection sort and the insertion sort algorithms for sorting any array of *Comparable* objects in ascending order. In this exercise, you will use the Sorting class to sort several different types of objects. (For details on the *Compareble* interface, refer to Java API doc http://docs.oracle.com/javase/7/docs/api/)

1. The file Numbers.java (in attachment) reads in an array of integers, invokes the selection sort algorithm to sort them, and then prints the sorted array. Save Sorting.java and Numbers.java to your directory. Numbers.java won't compile in its current form. Study it to see if you can figure out why.

2. Try to compile Numbers.java and see what the error message is. The problem involves the difference between primitive data and objects. Change the program so it will work correctly (note: you don't need to make many changes - the *autoboxing* feature of Java 1.5 (or higher) will take care of most conversions from int to Integer). You are to do research in the internet and understand better *autoboxing*.

3. Write a program Strings.java, similar to Numbers.java, that reads in an array of String objects and sorts them. You may just copy and edit Numbers.java.

4. Modify the insertionSort algorithm so that it sorts in descending order rather than ascending order. Change Numbers.java and Strings.java to call insertionSort rather than selectionSort. Run both to make sure the sorting is correct.



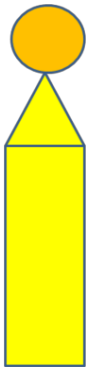
5. The class diagram on the right defines the SalePerson class that represents a sale person. The sale person has a first name, last name, and a total number of sales (an int).
   - The *toString* method will return the name of the sale person and total sales in the formal :
     <lastName> , <firstName> : <totalSales>
   - The *equals* method will check whether the first and last names of Object are the same as the current sale person.
   - The *compareTo* method make the comparison based on total sales; that is, return a negative number if the executing object has total sales less than the other object and return a positive number if the sales are greater. ***Use the name of the sales person's last name to break a tie (in ascending alphabetical order).***
   - ***Create and Write the SalePerson class***

6. The file WeeklySales.java (in attachment) contains a driver for testing the compareTo method and the sorting . Compile and run it. Make sure your compareTo method is correct. The sales staff should be listed in **the order of sales from most to least**. If the sale staffs have the same number of sales, they are listed in ascending alphabetical order of their last names.

3.2 Calculate Surface Area of a Figure.

By using the concepts of inheritance and **polymorphism**, you are required to design a program that calculate the total surface area of a figure. The following are the requirements an constraints :

- You should have a Class/Interface called *Shape and decide its appropriate attributes and behaviours*
- You should have basic shapes like *Square*, *Rectangle*, *Circle* and *Triangle*.
- The program will request the user to :
  - enter the total number of shapes
  - choose the shape and enter the required dimension/s for the selected shape
  - choose the type of calculation (for now, we will just calculate *Area,* with future plan to calculate *Volume* as well).
- The calculation/s should be done upon user's request and *NOT when dimensions are entered*.



1. For a start, use the 2-D figure on the right to verify your program. The figure consists of a Circle (radius=10), a Triangle (height=25, base =20) and a Rectangle (length=50, breadth = 20) . ***Calculate the total area of the 2- D figure***.
   (You will create an Application class **Shape2DApp.java** for this purpose)

2. We will now expand and extend your design to cater to 3-D figures. Imagine the figure on the right is turn into a 3-D figure – Circle becomes Sphere, Triangle becomes a square-based Pyramid and the Rectangle is a cubiod. ***Calculate the total surface area of the 3- D figure***. [Note : You need to think whether 'is a' or 'has a' relationship is more appropriate and relevant for between 2D and 3D shapes.
   (You will create an Application class **Shape3DApp.java** for this purpose)

3. We will include more Shapes. The square-based Pyramid will be replaced with a Cone and the Cubiod is replaced with a Cylinder. ***Calculate the total surface area of the new 3- D figure***.
   (You will **reuse** the Application class **Shape3DApp.java** with appropriate selection)

Continue on next page…..

IMPORTANT :

For this task, you are encouraged to work in pair. At the end of this task, you or the pair will demonstrate to your lab sup the working of your program.

Note :

The solutions for 3D Figures implementation can be many, simple or complex (like considering the overlapping regions). But the main objectives is to considered the idea and concept of polymorphism in the design and implementation. To build an application like AutoCAD, it can be just a discussion with your lab supervisor.

Lab 5 : OO implementation in C++

1. OBJECTIVE

The objective of Lab 5 is to practise on C++ syntax and implementation as well as Object-oriented features in C++.

2. INTRODUCTION

Focusing mainly on the Object-oriented features in C++, you start from building and defining a simple class, observing the constructor and destructor mechanisms in C++ as well a implementing inheritance relationships. At the later part of the lab, you will define abstract classes and put polymorphism to work. A brief guide on the use of the C++ IDE is provided in *Appendix C*.

3. *Your Tasks for this LAB*

You should save your project workspace in your own working directory.

3.1 Define a Class and Object Instantiation : Create the *Animal* class

1. Write the following **Animal class** and save it in your own directory as **Animal.cpp**

```
#include <iostream>
#include <string>

using namespace std ;

enum COLOR { Green, Blue, White, Black, Brown } ;

class Animal {
 public :
        Animal() : _name("unknown") {
              cout << "constructing Animal object "<< _name << endl ;
        }
        ~Animal() {
              cout << "destructing Animal object "<< _name << endl ;
        }
        void speak() const {
              cout << "Animal speaks "<< endl ;
       }
        void move() const { }

 private :
        string _name;
        COLOR _color ;
};

int main() {

       Animal a;
       a.speak() ;
       cout << "Program exiting …. "<< endl ;
    return 0;
}
```

a) Build the code into an executable, eg *OODP.exe*

b) In order to view the destructor at work, you will need to run the executable in a window console.
c) Click on the Windows **Start** at the left bottom corner and select **"Run…"**
d) Type **"cmd"** to open a command prompt console.
e) Change directory to your working directory where the .exe is by using **cd** command**.**
- Eg, **> cd E:\tmp\OODP\Debug**
- This will change the directory to **E** drive with path **\tmp\OODP\Debug**

f) Type the executable name without the '.exe' to run your program**.**
g) Note and understand the sequences where the Animal constructor and destructor are called**.**

2. Overload the Animal constructor by writing another constructor which will ***initialize the name and color*** of the Animal object.
a) The signature of the constructor is **Animal(string n, COLOR c)**
b) Print the name and color in the constructor.
c) Change or modify the **main** function to use the new constructor and provide required parameter values.
d) Do step 1(a)-(g) again

3.2 Inheritance : Create *Animal* subclasses

1. In the same **Animal.cpp** file, write a **Mammal class** to **inherit** from the **Animal class**.
a. Hint **Mammal: public Animal**
b. Create a public method **void eat() const** for the **Mammal Class**

```
void eat() const {
        cout << "Mammal eat " << endl ;
 }
```

c. Create a **constructor** which will **initialize the name and color** of the **Mammal object.**
[you should **make use** of the Animal constructor **]**
d. Create a **destructor** for the **Mammal class.**
e. Create a **Mammal object** in **main** and called the **speak** method**.**
f. Build and run the application

[***Note*** : to pause the console display, you can add the code **system("PAUSE"); OR cin.get() ;**]

2. In the same **Animal.cpp** file, write a **Dog class** to **inherit** from the **Mammal class**.
a. Create an attribute **string owner** for the **Dog Class**
b. Create a **constructor** which will **initialize the name, color and owner** of the **Dog object.**
[you should **make use** of the **Mammal** constructor **]**
c. Create a **destructor** for the **Dog class.**
d. Create a **Dog object** in **main** and called the **speak** method**.**
e. Build and run the application
[you should also observe the sequences where the Animal, Mammal and Dog constructor and destructor are called]

3. Override the **speak** method for Dog to "Woof"
a. Put the "**virtual**" keyword in the appropriate class and method and write the code for Dog object to speak appropriately.
b. Build and run the application.
c. Is the output as you have expected?

4. Pure Method
a. declare the **move** function in the **Animal** class as a **pure** method using the "**virtual**" keyword and "**= 0**"
b. Build the application
c. Understand and resolve the error/s you encountered.
[You should not make **Mammal** class as abstract]
d. Add the code in **main** for **Dog** object to called the **move** method.

3.3 **Polymorphism :** Build a Zoo

1. Using the abstract class
   a. The **Animal** class is now an **abstract** class
   b. **Abstract** class cannot be instantiated but it can be used as a reference type
   c. Add the following code in **main**
   ```
   Animal *animalPtr  = new Dog("Lassie", White, "Andy");
   ```
   d. Use the **`animalPtr`** to call the **speak** and **move** methods.
   [remember to also *delete* **`animalPtr`** before exiting the program**]**
   e. Build and run the application.

2. Without **virtual**
   a. In the **Animal** class, delete the **virtual** keyword from the **speak** method**.**
   b. Build and run the application.
   c. Note the difference in output as compared to (1).
   d. Build and run the application.
   e. Undo the deleting of the **virtual** keyword in (a).

3. **virtual** function magic
   a. Add the below codes in the **main** function

```
Dog dogi("Lassie", White, "Andy");
Mammal *aniPtr = &dogi ;
Mammal &aniRef = dogi ;
Mammal aniVal = dogi ;

aniPtr->speak() ;
       aniRef.speak() ;
              aniVal.speak() ;
```

   b. Build and run the application.
   c. Is the output as you have expected?

4. Build a Zoo
   a. In this task, you will write your own code.
   b. The requirements are :
      - Create 2 more subclasses **Cat** and **Lion** inheriting from **Mammal class**
        [you should add in the relevant **methods – constructors & destructor]**
      - Override the **move** and **speak** methods to perform a simple printout of the mammal, eg Cat meow, Lion Roar.
      - Create a program to ask user to select the animals from the list
        Eg display :
        Select the animal to send to Zoo :
        (1) Dog (2) Cat (3) Lion (4) Move all animals (5) Quit
      - You should use an **array** to store the selected ***Mammal*** **object.**
      - When **"Move all animals"** is selected, you should traverse the array and make the **animals *move, speak and eat***.
      - Will your code still work if it is an **array of *Animal* objects ?**

5. Header and Implementation files
   a. Separate the declaration code and Implementation code – create a header file/s (.h) for the declaration code and implementation file/s (.cpp) for the implementation code.
   Eg, *animal.h* for Animal and Mammal class, *childAnimal.h* for Dog, Cat, etc, subclasses and a *ZooMain.cpp* for implementation. *Make sure you #include the header file/s accordingly.*
   b. You may also selectively put simple implementation code in the header file.
   c. Build and execute the application to make sure all is still well as in (4).
   d. You will submit this version.

[Note : *You may want to google* ***'include guard"*** *if you faced compilation error/s due to #include* ***]***