Kuwait University
College of Life Sciences
Department of Information Science

Lab Manual

ISC 241 Data Structures

Prepared by

Professor Mostafa Abd-El-Barr

Professor Jehad Al Dallal

Eng. Aisha Al-Houti

Revised 2018

Table of Contents

Laboratory Hardware and Software/Tools Requirements 3
Laboratory Schedule 4
Laboratory Policy 5
Laboratory Grading Policy 6
Introduction 7
Familiarity with Lab Hardware and Software tools 7
Laboratory Tools Setup 7
Laboratory #1 – Classes, Objects, and Methods 8
Laboratory #2– Object-Oriented Design-I 10
Laboratory #3 – Object-Oriented Design-II 13
Laboratory #4 – Arrays 20
Laboratory #5 – Single-Linked Lists 23
Laboratory #6 – Doubly-Linked Lists 26
Laboratory #7– Stacks and Queues 28
Laboratory #8 – Trees 31
Laboratory #9 – Binary Search Trees and Arithmetic evaluation 34
Laboratory #10 – Graphs (1) 39
Laboratory #11 – Graphs (2) 45
Laboratory #12 – Searching 48
Laboratory #13 – Sorting 50
Appendix A: Rules to fallow by Computer Lab Users 53
Appendix B: Endorsement 54

Laboratory Hardware and Software/Tools Requirements

In this lab the students will be using the same hardware and software tools to which they we introduced in course ISC 240 Programming and Problem Solving. Java compiler and an Integrated Development Environment are required to be installed. JCreator with the Java Development Kit (JDK) are typically used for this purpose.

Laboratory Schedule

#	Lab Title	Lab activity
1	Classes, Objects, and Methods	
2	Object-Oriented Design-I	Exercise # 1
3	Object-Oriented Design-II	Exercise # 2
4	Arrays	Exercise # 3
5	Single-Linked Lists	Exercise # 4
6	Double-Linked Lists	Exercise # 5
7	Stacks and Queues	Quiz # 1
8	Trees	Exercise # 6
9	Binary Search Trees and Arithmetic evaluation	Exercise # 7
10	Graphs I	Exercise # 8
11	Graphs II	Quiz # 2
12	Searching	Exercise # 9
13	Sorting	Exercise # 10
14	Final	Lab Final

Laboratory Policy

- Fallow the laboratory rules listed in appendix “A”
- To pass this course, the student must pass the lab-component of the course.
- Cheating in whatever form will result in F grade.
- Attendance will be checked at the beginning of each Lab.
- Number of absence hours will be combined with the absence hours of the course and they are subject for applying the university absence regulations.
- Cheating in Lab Work or Lab Project will result F grade in Lab.
- There will be no make-up for any Quiz/Exam/Lab.
- Hard work and dedication are necessary ingredients for success in this course.

Laboratory Grading Policy

Activity	Weight
Lab Work (10 x 1%)	10%
Lab Quizzes (2 x 2.5%)	5%
Lab Final Exam	10%
Total	25%

Introduction

This lab is an integral part of the course ISC 241 Data Structures. The main objectives of the lab are to experience the applications of the different data structures taught in the class.

Familiarity with Lab Hardware and Software tools

In this lab the students will be using the same hardware and software tools to which they we introduced in course ISC 240 Programming and Problem Solving.

Laboratory Tools Setup

Install Java compiler and an Integrated Development Environment. Setting up JCreator with the Java Development Kit (JDK)

1. Installation of the Java Development Kit:
    1. Version 9.0 or any higher version: http://www.oracle.com/technetwork/java/javase/downloads/index.html , click **Java Platform (JDK) 9**.
    2. You will be forwarded to the download page.
    3. Accept License Agreement. Then download the executable file
    4. Install the JDK in the default directory

2. After the installation, start JCreator:
    1. Follow the steps of the setup wizard to set the paths "C:\..\jdk6.0_02\" and "C:\..\jdk6.0_02\docs"

3. If you have skipped the setup wizard
    1. Open the options window via the configure menu.
    2. Select the JDK Profiles pane.
    3. Select the default item from the list and press the edit button.
    4. Set the JDK Home path by pressing the browse button next to it.
    5. Browse to the root directory of the JDK installation.
    6. On the Profile Settings dialog box, check to see if the Name field contains the version of the selected JDK directory.
    7. Select the documentation tab and check if the list includes the path to "C:\..\ jdk9***\docs"
    8. Close the windows

4. Create a test project.
    1. Open the project wizard via the menu File > New
    2. Select the template "Basic Java Application"
    3. Compile and run the project by pressing F7 and F5 on the keyboard.

Laboratory #1 – Classes, Objects, and Methods

1. Laboratory Objective

The objective of this introductory laboratory is to reinforce students for some of the basic concepts in Java Object Oriented Programming, such as Objects, Classes, and Methods.

2. Laboratory Learning Outcomes: After conducting this laboratory students will be able to:

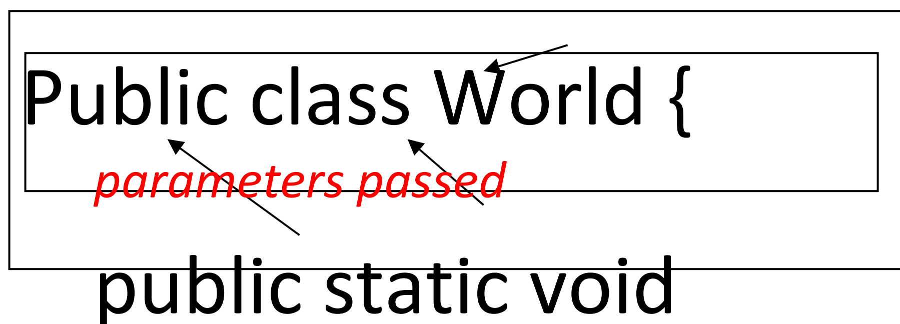
a. Type, edit, compile, execute and debug a class and its driving main program.
b. Write, edit, compile, execute and debug a method in a class.
c. Write, edit, compile, execute and debug a constructor for a class.

3. Laboratory Introductory Concepts

Complex numbers have the form *realPart + imaginaryPart * i* where *i* is $\sqrt{-1}$. Assuming that we have two complex numbers X= *a + b i* and Y= *c + d i*. The following operations can be performed on the two numbers X and Y.

i. Add the real parts of each complex number, and add the imaginary parts of each complex number.
ii. Subtract two `Complex` numbers. Subtract the real part of the right operand from the real part of the left operand, and subtract the imaginary part of the right operand from the imaginary part of the left operand.
iii. Multiply two `Complex` numbers. The real part is equal to ***ac – bd*** and the imaginary part is equal to ***ad + cb***.
iv. Divide two `Complex` numbers: the numerator and the denominator should be multiplied by the conjugate of the denominator. The real part is equal to $\frac{ac+bd}{c^2+d^2}$ and the imaginary part is equal to $\frac{cb-ad}{c^2+d^2}$.

Example: Class/Method Declaration



Syntax for creating an object:

Calss_name object_name = new class_name();

4. Laboratory Problem Description

In this laboratory, you are required to create a *class* called ***`Complex`*** for performing arithmetic operations on complex numbers. Use floating-point variables to represent the data in the class. Provide a *constructor* that enables an object of this class to be initialized when it is declared. Provide a no-argument constructor with default values in case no initialization is provided. Provide methods that perform the following operations:

a) Add two **`complex`** numbers.
b) Subtract two complex numbers.
c) Multiply two complex numbers.
d) Divide two complex numbers.
e) Print **`Complex`** numbers in the form *a + b i*, where **a** is a real part and **b** is the imaginary part.
f) Determine whether a number is a Complex or Real number and return the result.

You are required to write a **`main`** program to test your class.

5. Laboratory Instructions

(a) Good Programming Practices: Here are some "Good" programming practices:

i. List the fields of a class first, so that, as you read the code, you see the names and types of the variables before you see them used in the methods of the class. It is possible to list the class's fields anywhere in the class outside its method declarations, but scattering them tends to lead to hard-to-read code.
ii. Place a blank line between method declarations to separate the methods and enhance program readability.

(b) Common Programming Errors: Here are some of the programming errors that must be avoided:

i. Declaring more than one public class in the same file is a compilation error.
ii. Compilation error occurs if the number of arguments in a method call does not match the number of parameters in the method declaration.
iii. Compilation error occurs if the types of arguments in a method call are not consistent with the types of the corresponding parameters in the method declaration.

Laboratory #2– Object-Oriented Design-I

1. Laboratory Objective

The objective of this laboratory is to reinforce for students a number of Object Oriented Programming Principles, such as inheritance, class hierarchy, super-classes and subclasses.

2. Laboratory Learning Outcomes: After conducing this laboratory students will be able to:

a. Apply the concepts of inheritance and class hierarchy.
b. Apply the concepts of superclasses and subclasses.

3. Laboratory Introductory Concepts

- **Formulas to use in connection with circular shapes**

Circle	Sphere	Cylinder
Area = πr^2 Circumference = $2 \pi r$	Surface Area = $4 \pi r^2$ Volume = $4/3 \pi r^3$	Surface Area = $2 \pi r h$ Volume = $\pi r^2 h$

- **Syntax for creating a subclass**

```
class MountainBike extends Bicycle {

  // new fields and methods defining a mountain bike would go here

}
```

4. Laboratory Problem Description

The main emphasis in this laboratory is to practice using super and subclasses in Java. You are required to create a hierarchy for `class CircularShape` as shown in Figure 1.

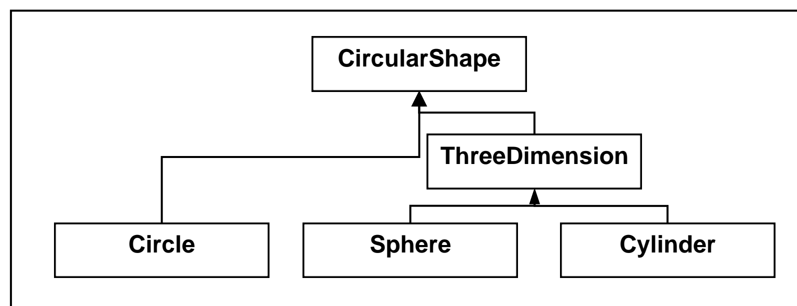


Figure 1

The class attributes and methods of each class are:

- **`CirculareShape`** should have **`radius`** (instance variable) of type integer and **`pi`** (class variable and constant) of type double. It should have three methods the first method to set the radius (**`void setRadius(int )`**), the second to get the radius (**`int getRadius()`**) and the third method (**`static void print()`**) should be a class method that prints the following statement: **`Circular shapes are Circle, Sphere and Cylinder`**.

- **`Circle`** should have the **`x`** and **`y`** coordinates of the center both of type integer. It should also have five methods: **`void setCenter(int , int )`**, **`int getX()`**, **`int getY()`**, **`double findArea()`**, and **`double findCircumference()`**.

- **`ThreeDimension`** should have **`type`** (String) of the shape. It has also two methods: **`void setType(String )`** and **`String getType()`**.

- **`Sphere`** should have a constructor (**`Sphere()`**) that sets the type of the sphere and two methods to find the surface area (**`double findSurfaceArea()`**) and the volume (**`double findVolume()`**).

- **`Cylinder`** should have **`height`** of type integer, a constructor (**`Cylinder()`**) that sets the type of the cylinder and three methods one to set the height (**`void setHeight(int )`**) and the others to find the surface area (**`double findSurfaceArea()`**) and the volume (**`double findVolume()`**).

You are required to write a program to test your class hierarchy according to the following:

1. Create objects for classes **`Circle`**, **`Sphere`** and **`Cylinder`**.
2. Use the method **`static void print()`** from class **`CircularShape`**.
3. Set the radius of the circle (radius = 2) and set x and y coordinate of the center (the center is the origin). Print the radius, center, area and circumference of the circle.
4. Set the radius of the sphere (radius = 2). Print the shape type, surface area and volume of the sphere.
5. Set the radius and the height of the cylinder (radius = 2, height = 3). Print the shape type, surface area and volume of the cylinder.

Output:

```
Circular shapes are Circle, Sphere and Cylinder

Shape is Circle

radius = 2
```

```
Center ( 0 , 0 )

Area = 12.56636

Circumference = 12.56636

Shape is Sphere

Surface Area = 50.26544

Volume = 33.51029333333333

Shape is Cylinder

Surface Area = 37.699079999999995

Volume = 37.699079999999995
```

5. Laboratory Exercises

Create class RandomeGuess that has the following methods:

- **`generateNumber`** that returns a random number between 0 and 100.
- **`main`** that will cal method **`generateNumber`** to have a random number and then it will ask the user to guess the number by entering a number between 0 and 100.
- You should have a sentinel value loop that will check whether the number entered by the user is > or < the random number generated by the method accordingly the loop will print the appropriate message and reads the number again until the guessed number is equal to the generated number.

Laboratory #3 – Object-Oriented Design-II

1. Laboratory Objective

The objective of this laboratory is to train students on the use of class extension and the reuse of methods.

2. Laboratory Learning Outcomes: After conducting this laboratory students will be able to:

a. Practice the use the keyword **`extends and super.`**

b. Practice the reuse of methods and instance variables from other classes.

3. Laboratory Introductory Concepts

Example superclass/subclass declaration

```
public abstract class car {


 }

 public class car1 extends car {


      {………}

 }

 public class car2 extends car {

           {……….}

 }

 public class car3 extends car {

           {………}

 }
```

4. Laboratory Problem Description

The main emphasis of this laboratory is to practice creating subclasses from a given superclass in Java. You are required to write class **Car** which will be **your Exercise # 1** the class has the following:

- Provide three **private** instance variables **brand** of type **String**, **model** of type **String** and **year** of type **int**.
- Provide **set** and **get** methods for each of these variables.
- Provide a **default constructor** which sets the **brand** to **"Generic"**, **model** to **1**, and **year** to **1900**.
- Provide another **constructor** that sets the instance variables to the passed parameters.
- Provide **toString** method which returns the **brand**, **model**, and **year**.

Write a class **Mercedes** which is **inherited** from class **Car** And has the following:

- Provide one **private** instance variable **serialNumber**.
- Override all the **set** methods in class **Car** to have the following:
    - **brand** must be always **"Mercedes"**,
    - **model** must be either **C280**, **E280** or **S280** (default **C280**), and
    - **year** must be between **1910** and **2010** (default is **2010**).
- Provide a **default constructor** which sets the **brand** to **"Mercedes"**, **model** to **C280**, and **year** to **2010**.
- Provide another **constructor** that sets **brand** to **"Mercedes"**, and sets **model** and **year** (using class **Mercedes set** methods) according to the passed parameters.
- Do not provide a **set** method for **serialNumber** but instead increment the last **serialNumber**.
  [Hint: declare **static private** instance variable to store the last serial number, initialize it to **10001**, and then increment it by one each time a new **Mercedes** object is created].

- Provide **toString** method which returns class **Car's toString** concatenated with the **serialNumber**.

Write a class **CarTest**, to test both of your classes.

Template

```
public class Car

{

	// declare the instance variables

	// write a three argument constructor that initializes the

	// instance variables with the passed parameters
```

```
        // write a default no argument constructor that initializes the
        // instance variables with the following
        // brand = "Genaric"
        // model = "1"
        // year = 1900
        // write getBrand method
        // write setBrand method
        // write getModel method
        // write setModel method
        // write getYear method
        // write setYear method
        // write toString method that will return a string as follows
        // Car Brand:  the brand
        // Car Model: the model
        // Car Year: the year
}
public class Mercedes extends Car
{
        // declare the instance variable
        // declare and initialize the last serial number sn=10001
        // write a default no argument constructor initializes the
        // instance variables by using the set methods to
        // brand = "Mercedes"
        // model = "C280"
        // year = 2010
        // increment sn by one
```

```
        // write a two argument constructor (model and year)
        // that initializes the instance variables
        // using the set methods to the passed parameters
        // with default brand to "Mercedes"
        // increment sn by one
        // write setBrand method
        // using class car super.setBrand("Mercedes")
        // write setModel method check if the model is equal to C280 or
        // E280 or S280 then set it using class car super.setModel(model)
        // else set it to default using class car super.setModel("C280")
        // write setYear method if the year is between 1910 and 2010
        // then set it using class car super.setYear(year)
        // else set it to default using class car super.setYear(2010)
        // write toString method that will return a string use class
        // Car super.toString() and concatenated with
        // Serial Number: serial number
}
public class CarTest
{
        public static void main(String[] args)
        {
                Mercedes c1 = Mercedes("E280",2005);
                Mercedes c2 = Mercedes("S280",1999);
                Mercedes c3 = Mercedes("10",1001);
                Mercedes c4 = Mercedes();
                System.out.println(c1.toString());
                System.out.println(c2.toString());
```

```
        System.out.println(c3.toString());

        System.out.println(c4.toString());

    }

}
```

Sample Output

```
Car Brand: Mercedes

Car Model:  E280

Car Year: 2005

Serial Number: 10002


Car Brand: Mercedes

Car Model:  S280

Car Year: 1999

Serial Number: 10003


Car Brand: Mercedes

Car Model:  C280

Car Year: 2010

Serial Number: 10004


Car Brand: Mercedes

Car Model:  C280

Car Year: 2010

Serial Number: 10005
```

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Declaring **private** instance variables helps programmers test, debug and correctly modify systems.
- Use inheritance to avoid duplicating code in situations where you want one class to "absorb" the instance variables and methods of another class.

You may want to avoid the following pitfalls

- It is a syntax error to override a method with a more restricted access modifier – a **public** method of the superclass cannot become a **private** or a **protected** method in the subclass; a **protected** method of the superclass cannot become a **private** method in the subclass. Doing so would break the "is-a" relationship in which it is required that all subclass objects be able to respond to methods calls that are made to **public** methods declared in the superclass. If a **public** method could be overridden as a **protected** or **private** method, the subclass objects would not be able to respond to the same method calls as superclass objects. Once a method is declared **public** in a superclass, the method remains **public** for all that class's direct and indirect subclasses.
- A compilation error occurs if a subclass constructor calls one of its superclass constructors with arguments that do not match exactly the number and types of parameters specified in one of the superclass constructor declarations.
- When a superclass method is overridden in a subclass, the subclass version often calls the superclass version to do portion of the work. Failure to prefix the superclass method name with the keyword **super** and a dot (**.**) separator when referencing the superclass's method causes the subclass method to call itself, creating an error called infinite recursion.

6. Laboratory Exercises

You are required to write class **Car** which will be **your Exercise # 1** the class has the following:

- Provide three **private** instance variables **brand** of type **String**, **model** of type **String** and **year** of type **int**.
- Provide **set** and **get** methods for each of these variables.
- Provide a **default constructor** which sets the **brand** to **"Generic"**, **model** to **1**, and **year** to **1900**.
- Provide another **constructor** that sets the instance variables to the passed parameters.
- Provide **toString** method which returns the **brand**, **model**, and **year**.

Laboratory #4 – Arrays

1. Laboratory Objective

The objective of this laboratory is to train students on how to use arrays as a data structure in the context of object oriented programming.

2. Laboratory Learning Outcomes: After conducting this laboratory students will be able to:

a. Apply basic operations on array data structure, such as searching and sorting.
b. Apply Object Oriented basic concepts in dealing with arrays.

3. Laboratory Introductory Concepts

- **Example Array Declaration & Initialization:**
  int [] primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
  type (int) **array name** (primes) **initial values** ({2, 3, 5, 7, 11, 13, 17, 19, 23, 29})

Primes	**2**	**3**	**5**	**7**	**11**	**13**	**17**	**19**	**23**	**29**
index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

4. Laboratory Problem Description

Create the following classes:

1. **Class name: `College` Exercise # 3 you should write this class:**
   - **Attributes:**
     a. **`name`** of type String.
     b. An array of Department objects (**`dept`**).
     c. A class variable **`noOfDepts`** of type integer with final value of 5.
   - **Constructor: `College(String, int)`** that initializes the name and creates the **`dept`** array.
   - **Methods:**
     a. **`void addDept(Department)`** to add a department object if the array is not full. If it is full, print a message.
2. **Class name: `Department`**
   - **Attributes:**
     a. **`name`** of type String.
     b. An array of Course objects (**`crs`**).
     c. A class variable **`noOfCourses`** of type integer with final value 10.

❖ **Constructor:** **`Department(String)`** that initializes the name and creates the array **`crs`**.

❖ **Methods:**
  a. **`void addCourse(Course)`** creates and adds the course if the course with id passed as parameter does not exist in the array and the array is not full. If it exists or array full, print a message.
  b. **`void deleteCourse(int)`** to delete a course if it exists in the array. If it does not exist, print a message.
  c. **`void printCourses()`** that prints all the courses in the array.

3. Class name: `Course`

❖ **Attributes:**
  a. **`id`** of type integer.
  b. **`name`** of type string.

❖ **Constructor:** `Course(int,String)` that initializes the course name and id.

❖ **Methods:**
  a. **`int getId()`** that returns the course id.
  b. **`String getName()`** that returns the course name.

4. Class name: `CollegeTest`

❖ **Method:** **`main`** method that performs the following:

i. Create object of class College for CFW.
ii. Add ISC department to the college using **`addDept(Department)`** method, **NOTE** that this method takes an object of type **`department`**.
iii. Add three courses to the ISC department using **`addCourse(Course)`** method, **NOTE** that this method takes an object of type **`course`**.
iv. Print all the course information for ISC department.
v. Delete one of the courses using **`deleteCourse(int)`** method.
vi. Print all the course information for ISC department.

Sample Output

```
240   Java Programming

241   Data Structures

363   Computer Organization
```

```
Courses after deleting


241   Data Structures

Course 240 does not exist

363   Computer Organization

Serial Number: 10005
```

5. Laboratory Instructions

☒ **Common programming errors**:

- Using a value of type long as an array index results in a compilation error. An index must be an int value or a value of a type that can be promoted to int—namely, byte, short or char, but notlong.
- In an array declaration, specifying the number of elements in the square brackets of the declaration (e.g., int c[12];) is a syntax error.
- Declaring multiple array variables in a single declaration can lead to subtle errors. Consider the declaration int[] a, b, c;. If a, b and c should be declared as array variables, then this declaration is correct—placing square brackets directly following the type indicates that all the identifiers in the declaration are array variables. However, if only a is intended to be an array variable, and b and c are intended to be individual int variables, then this declaration is incorrect—the declaration int a[], b, c; would achieve the desired result.
- Assigning a value to a constant after the variable has been initialized is a compilation error.
- Attempting to use a constant before it is initialized is a compilation error.

❒ **Good Programming Practices:**

- For readability, declare only one variable per declaration. Keep each declaration on a separate line, and include a comment describing the variable being declared.
- Constant variables also are called named constants or read-only variables. Such variables often make programs more readable than programs that use literal values (e.g., 10)—a named constant such as ARRAY_LENGTH clearly indicates its purpose, whereas a literal value could have different meanings based on the context in which it is used.

6. Laboratory Exercises

You may want to practice more with arrays using the following exercises.

1. Write an algorithm and a Java program that reads 100 values from a file, stores them in an array, and determines the number of values that are greater than or equal to the average value of the values in the array.
2. Write a Java program that computes the transpose of a 2-dimensional matrix.
3. Write a Java program that computes the multiplication of two 2-dimensional matrices and stores the results in a third 2-dimensional matrix.

Laboratory #5 – Single-Linked Lists

1. Laboratory Objective

The objective of this laboratory is to train students on how to use singly-linked lists data structure in the context of object oriented programming.

2. Laboratory Learning Outcomes: After conducting this laboratory students will be able to:

a. Apply basic operations on singly-linked lists data structure, such as creating a list, adding a node, deleting a node, hoping through a list, and counting the number of nodes in a list.

b. Apply Object Oriented basic concepts in dealing with singly-linked lists.

3. Laboratory Introductory Concepts

- **Example Node Declaration**

```
class Node {
  public int key;
  public Node next;
  Node (int n, Node p) {
      key = n;
      next = p;
  }
 };
```

- **Example hopping down a list stopping upon some condition C (p.key == k)**

```
 for (Node p = head; p != null; p = p.next)
      if ( p.key == k ) {
          // do something to the node that p refers to
                         k++;
          break;    // if you want to stop the chaining along
      }
```

- **Example Finding the length of a list**

```
int length ( ) {
  int i=0;
  for (Node p = head; p != null; p = p.next)
      ++i;
  return (i);
}
```

- **Example Printing a list**

```
void print ( ) {
    for (Node p = head; p != null; p = p.next)
       system.out.println (p.next);
    }
```

4. Laboratory Problem Description

Create the following classes:

1. **Class:** `Department`
   - **Attributes:**
     i. **`name`** of type string.
     ii. a reference to **`Course`** object (**`head`**).
   - **Constructor:** **`Department(String)`** to initialize the **`name`** and set the **`head`** to **`null`**.

   - **Methods:**
     i. **`boolean exist(int)`** that checks whether the course object with id passed as parameter exists in the singly linked list or not.
     ii. **`void addCourse(int, String)`** creates and adds the course if the course with id passed as parameter does not exist in the singly linked list.
     iii. **`void deleteCourse(int)`** to delete a course if it exists in the singly linked list.
     iv. **`void printCourses()`** is a non-recursive method that prints all the courses in the singly linked list.
     v. **`void printCourses(Course)`** is a recursive method that prints all the courses in the singly linked list.
     vi. **`void print()`** is a non-recursive method that calls the recursive **`printCourses`** and pass the **`head`** as an argument.
2. **Class name:** `Course`
   - **Attributes:**
     i. **`id`** of type integer.
     ii. **`name`** of type string.
     iii. a reference to **`Course`** object (**`next`**).
   - **Constructor:** **`Course(int,String)`** that initializes the course **`name`** and **`id`** and set the **`next`** to **`null`**.
   - **Methods:** the accessor and modifier methods for **`id`**, **`name`** and **`next`**.

3. **Class:** `Lab5Test`

   - **Method:** **`main`** method that performs the following:
     - Inputs the department name and creates the department object.
     - Uses a loop to input the course id and name and add the course to the singly linked list. If the user inputs -1 the loop terminates.

- Prints the courses using the recursive and the non-recursive methods.
- Inputs a course id to be deleted and delete the course.
- Prints the courses.

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Declare the node of the single linked list in a separate file.
- To travel through the linked list in order to reach the end or a specific node its wise to use a while loop with another variable that is initially equals to the head of the list.

You may want to avoid the following pitfalls

- When dealing with list do not move the head from its place or otherwise you will lose the list, instead make a copy from the head with another variable.

6. Laboratory Exercises

Add the following two methods to class **`Department`**:

1. **`int countCourses()`** is a non-recursive method that counts the number of courses in the singly linked list and return the number of courses to the calling method.
2. **`void findMiddleCourse()`** is a non-recursive method that calls **`countCourses`** method to find the middle course. It prints the course **`id`** and **`name`** for the middle course.

In the main method, you should call findMiddleCourse method to test your methods.

Laboratory #6 – Doubly-Linked Lists

1. Laboratory Objective

The objective of this laboratory is to train students on how to use doubly-linked lists data structure in the context of object oriented programming.

2. Laboratory Learning Outcomes: After conducting this laboratory students will be able to:

a. Apply basic operations on doubly-linked lists data structure, such as creating a list, adding a node, deleting a node, hoping through a list, and counting the number of nodes in a list.
b. Apply Object Oriented basic concepts in dealing with doubly-linked lists.

3. Laboratory Introductory Concepts

- **Example Doubly Linked List Node Declaration**

```
Public class Node-D {
  protected String element;
  protected Node-D next, prev;
  public Node-D (String e, Node-D p, Node-D n){
  element = e;
  prev = p;
  next = n;
{
```

4. Laboratory Problem Description

In the previous lab, we have created the **`Department`** and **`Course`** classes along with their attributes and methods. We have implemented the abovementioned classes using singly linked lists. In this lab, you should implement the same classes using doubly linked list.

You need to open your previous codes and make some changes to adapt to the concept of doubly linked list.

1. **Class:** `Department`
   a. **Attributes:**
      1. **`name`** of type string.
      2. a reference to **`Course`** object (**`head`**).
   b. **Constructor:** `Department(String)` to initialize the **`name`** and set the **`head`** to **`null`**.
   c. **Methods:**
      1. **`boolean exist(int)`** that checks whether the course object with id passed as parameter exists in the doubly linked list or not.
      2. **`void addCourse(int, String)`** creates and adds the course if the course with id passed as parameter does not exist in the doubly linked list. The new course should be added at the head of the doubly linked list.

3. **`void deleteCourse(int)`** to delete a course if it exists in the doubly linked list.
4. **`void printForward()`** prints the doubly linked list forward (from head to tail).
5. **`void printBackward()`** prints the doubly linked list backward (from tail to head).

2. Class name: `Course`

a. Attributes:

1. **`id`** of type integer.
2. **`name`** of type string.
3. two references to **`Course`** objects (**`next`** and **`prev`**).

b. **Constructor: `Course(int, String, Course, Course)`** that initializes the course **`name`**, **`id`**, **`next`** and **`prev`**.

c. **Methods:** the accessor and modifier methods for **`id`**, **`name`**, **`next`** and **`prev`**.

3. Class: `Lab6Test`

❖ **Method: `main`** method that performs the following:

- Create **`ISC`** object of class **`Department`**.
- Add the following three courses to **`ISC`** department: **`240 (Java Programming)`**, **`241 (Data Structures`**), and **`363 (Computer Organization)`** using **`addCourse(int, String)`** method.
- Print the course id and name of the three courses you have already added to **`ISC`** department using the **`printForward()`** and **`printBackward()`** methods.
- Delete course **`363 (Computer Organization)`** using **`deleteCourse(int)`** method.
- Print the courses again. The deleted course will not be printed.

5. Laboratory Instructions

You may want to make use of the following good programming practices

- o Declare the node of the double linked list in a separate file.
- o To travel through the linked list in order to reach the end or a specific node its wise to use a while loop with another variable that is initially equals to the head of the list.

You may want to avoid the following pitfalls

- o When dealing with list do not move the head from its place or otherwise you will lose the list, instead make a copy from the head with another variable.

6. Laboratory Exercises

. Your exercise should be to write the **`printBackward()`** method

Laboratory #7– Stacks and Queues

1. Laboratory Objective

The objective of this laboratory is to train students on how to use the Stacks and Queues data structures in the context of object oriented programming.

2. Laboratory Learning Outcomes: After conducting this laboratory students will be able to:

a. Apply basic operations on stack data structure, such as creating a stack, adding a node, deleting a node, counting the number of nodes, finding a node with certain key, etc.
b. Apply basic operations on queue data structure, such as creating a queue, adding a node, deleting a node, counting the number of nodes, finding a node with certain key, etc.

3. Laboratory Introductory Concepts

- Example algorithms for conducting some stack operations

Algorithm size():
  return t+1
Algorithm isEmpty():
  return (t<0)
Algorithm top():
  if isEmpty() then
  throw a EmptyStackException
  else return S[t]
Algorithm push (o):
  if t= S.length -1 then
  throw a FullStackException
  else t ← t+1
  S[t] ← o
Algorithm pop():
  if isEmpty() then
  throw a EmptyStackException
  else return S[t]
 t ← t-1

4. Laboratory Problem Description

Stacks

- Class **`Node`**:
  - This class keeps String value **`element`** and one reference; **`next`**.
  - It has a constructor that initializes the instance variables.
  - It has the **`set`** and **`get`** methods for its attributes.

- Class **`Stack`**:

- This class keeps integer value **noOfElements** and two references; **head** and **top**.
- It has a constructor that initializes the instance variables.
- It has the following methods:
  - **size**() that returns the size of the stack.
  - **isEmpty**() that returns **true** if the stack is empty or **false** otherwise.
  - **getTop**() that returns a reference to the **top** of the stack.
  - **push**(**Node**) that pushes a **Node** object to the **top** of the stack.
  - **pop**() that pops the **top** object and returns it.

❖ Class **StackTest**:

- This class includes **main** method to test your classes.
  - Create object of class **Stack.**
  - Check if the stack is empty and print a message that says the stack is empty
  - Push three elements n1, n2, and n3 to the stack using **push()** method.
  - Get the top element and print it.
  - Push another node n4.
  - Get the top element and print it.
  - Print the size using **size()** method.
  - Pop one element using **pop()** method.
  - Get the top element and print it.
  - Print the size using **size()** method.

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Declare the node of the stack or the queue in a separate file.
- Its good to have methods that returns the value of the top, front or rear.
- You should have method that tells you if the stack or the queue is empty.

6. Laboratory Exercises

You should write these classes by yourself as an exercise:

❖ Class **Student**:

- This class keeps four instance variables **name** of type String, **id** of type integer and two references to **Student** object; **next** and **prev**
- It has a constructor that initializes the instance variables.
- It has the **set** and **get** method for its attributes.

❖ Class **LQueue**:

- This class keeps references to **Student** object **front and rear** and an integer instance variables **noOfElements**.
- It has a constructor that initializes the instance variables.
- It has the following methods:
  - **size**() that returns the size of the queue.
  - **isEmpty**() that returns **true** if the queue is empty or **false** otherwise.

- **front**() that returns a reference to the **front** of the queue or **null** if the queue is empty.
- **enqueue**(**Student**) that add a **Student** object to the rear of the queue.
- **dequeue**() that remove the front object and returns it.

❖ Class **QueueTest**:

- This class includes **main** method to test your classes.
  - Create object of class **LQueue**.
  - Check if the queue is empty and print a message that says the queue is empty
  - Add three elements (1, n1), (2, n2), and (3, n3) to the queue using **enqueue**() method.
  - Get the front element and print its id.
  - Add another node (4, n4).
  - Get the front element and print its id.
  - Print the size using **size**() method.
  - Remove one element using **dequeue** () method.
  - Get the front element and print its id.
  - Print the size using **size**() method.

Laboratory #8 – Trees

1. Laboratory Objective

The objective of this laboratory is to train students on how to use the Tree data structure in the context of object oriented programming.

2. Laboratory Learning Outcomes: After conducting this laboratory students will be able to:

a. Apply basic operations on binary trees data structure, such as creating a tree, adding a node, deleting a node, counting the number of nodes, finding a node with certain key, and traversing a binary tree (pre-order, post-order, and in-order).
b. Apply Object Oriented basic concepts in dealing with binary trees.

3. Laboratory Introductory Concepts

- **Example** ***Binary Tree Class Declaration***

```
class BinaryTree
{
  private Node root;
  public void find(int key)
  {
  //
  }
  public void insert(int key, object otherinfo)
  {
  //
  }
  public void delete(int key)
  {
  //
  }
}
```

4. Laboratory Problem Description

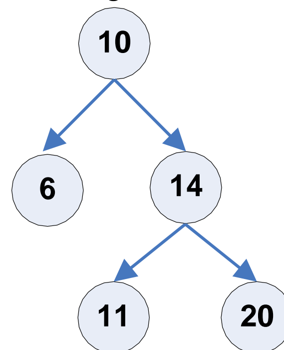
Suppose that you want to implement a binary tree. Create the following classes:

- Class **`TreeNode`**:
  - This class keeps integer value **`data`** and three references; **`left`**, **`right`** children and **`parent`**.
  - It has the **`set`** and **`get`** methods for its attributes.
  - It has method **`hasLeft()`** that return **`true`** if the node has a left child or **`false`** if not.

- It has method **hasRight()** that return **true** if the node has a right child or **false** if not.
- It has method **isExternal()** that return **true** if the node does not have a left and a right child or **false** if otherwise.

❖ Class **Tree**:

- This class keeps a reference to the tree **root**.
- It has an array to keep the location of the nodes in the tree, array size is 100.
- It has and integer value index initialized to 0 it keeps track of the array index.
- It has a constructor that creates the following tree:



- And it has the following methods:
  - **printPreorder()** is a method that calls **preorder(root)** to print the preorder traversal of the tree.
  - **preorder(TreeNode)** is a recursive method that prints the preorder traversal of the tree.
  - **printPostorder()** is a method that calls **postorder(root)** to print the postorder traversal of the tree.
  - **postorder(TreeNode)** is a recursive method that prints the postorder traversal of the tree.
  - **printInorder()** is a method that calls **inorder(root)** to print the inorder traversal of the tree.
  - **inorder(TreeNode)** is a recursive method that prints the inorder traversal of the tree.
  - **insertNode(int)** is a method that sets the **root** to point to a **TreeNode** object, if the Tree is empty or calls a method **insert(int, TreeNode)** with the integer value to be inserted and a reference to the **root**.
  - **insert(int, TreeNode)** is a recursive method that inserts the integer value in its proper location to maintain the property of the binary search tree.
  - **TreeNode search(int, TreeNode)** is a recursive method that searches for the integer value in the tree and returns a reference to the node storing the value.
  - **removeExternal(TreeNode)** is a method that deletes a node if it has one child or no children.
  - **findInorderArray(TreeNode)** is a recursive method that stores a reference to **TreeNode** in the array using inorder traversal of the tree.

- o **`delete(int)`** is a method that locates the node to be deleted node(i) by calling **`search(int, root)`**. If the node(i) has exactly two children, it will call **`findInorderArray(root)`** to locate the node(i+1) that follows node(i) in the inorder traversal. It will copy the value in node(i+1) into node(i) and remove node(i+1) by calling **`removeExternal(node(i+1))`**. If the node has one child or no children, it will call **`removeExternal(node(i))`**.

❖ Class **`Lab9`**:

- This class has a **`main`** method that creates an object of the class Tree and it does the following operations:
  - o It prints the preorder traversal of the tree.
  - o It prints the postorder traversal of the tree.
  - o It prints the inorder traversal of the tree.
  - o Delete a number from the binary search tree then print the resulted tree using inorder.
  - o Adds a number to the binary search tree then print the resulted tree using inorder.

5. Laboratory Instructions

You may want to make use of the following good programming practices

- o Declare the node of tree in a separate file.
- o Include a variable for the left, right and parent of the node.

You may want to avoid the following pitfalls

- o You should not have the recursive method without having another method that calls it, since you need to give the recursive method the base where it will start with.

6. Laboratory Exercises

Your exercise will be writing the following methods:

a. **`hasLeft() , hasRight(), isExternal()`** from class **`TreeNode.`**
b. **`insert(int, TreeNode)`** from class **`Tree.`**

Laboratory #9 – Binary Search Trees and Arithmetic evaluation

1. Laboratory Objectives

The objective of this laboratory is to train students on how to use Binary Tree data structure in the context of object oriented programming.

2. Laboratory Learning Outcomes

After conducting this laboratory students will be able to:

a. Apply basic operations on binary search trees data structure, such as creating a tree, adding a node, deleting a node, counting the number of nodes, finding a node with certain key, and traversing a binary tree (pre-order, post-order, and in-order).
b. To print an arithmetic expression stored in the binary tree using the pre, post and in order traversal.
c. To evaluate an arithmetic expression using post-order traversal of a binary tree.
d. Apply Object Oriented basic concepts in dealing with binary trees.

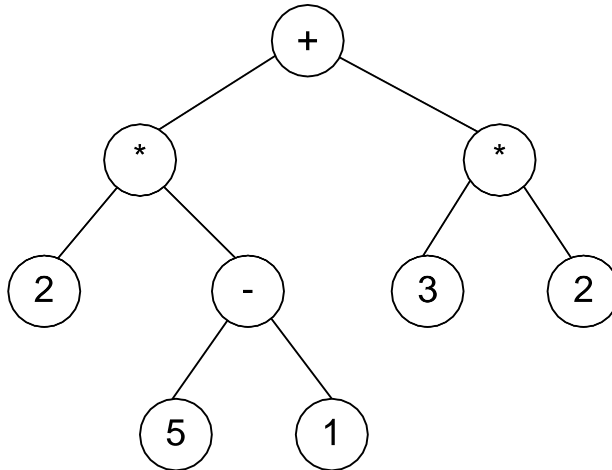
3. Laboratory Introductory Concepts

4. Laboratory Problem Description

Suppose that you want to implement a binary tree for evaluation of an arithmetic expressions. Create the following classes:

- Class **`BTNode`**:
  - This class keeps **`String`** variable **`element`** and three references; **`left`**, **`right`** children and **`parent`**.
  - It has a constructor that initializes the instance variables.
  - It has the **`set`** and **`get`** methods for its attributes.
  - It has method **`hasLeft()`** that return **`true`** if the node has a left child or **`false`** if not.
  - It has method **`hasRight()`** that return **`true`** if the node has a right child or **`false`** if not.
  - It has method **`isExternal()`** that return **`true`** if the node does not have a left and a right child or **`false`** if otherwise.

- Class **`Arithmetic`**:
  - This class keeps a reference to the tree **`root`**.
  - It has a constructor that creates the following tree:



- It has the following methods:
  - **`printPreorderTraversal()`** is a method that calls **`preorder(root)`** to print the preorder traversal of the tree.
  - **`preorder(BTNode)`** is a recursive method that prints the preorder traversal of the tree.
  - **`printPostorderTraversal()`** is a method that calls **`postorder(root)`** to print the postorder traversal of the tree.
  - **`postorder(BTNode)`** is a recursive method that prints the postorder traversal of the tree.
  - **`printInorderTraversal()`** is a method that calls **`inorder(root)`** to print the inorder traversal of the tree.
  - **`inorder(BTNode)`** is a recursive method that prints the inorder traversal of the tree.
  - **`print()`** is a method that calls **`printExpression(root)`** to print the arithmetic expression.
  - **`printExpression(BTNode)`** is a recursive method that prints the arithmetic expression using inorder traversal of the tree. If the node n has a left child, it prints "(" and if the node n has a right child, it prints ")".
  - **`printResult()`** is method that calls **`evaluate(root)`** to evaluate the arithmetic expression.
  - **`evaluate(BTNode)`** is a recursive method that evaluate the arithmetic expression using postorder traversal.

❖ Write a **`main`** method to test your classes.

a. You should create object of class arithmetic and call all the methods:

- **`printPreorderTraversal()`**
- **`printPostorderTraversal()`**
- **`printInorderTraversal()`**
- **`print()`**
- **`printResult()`**

Output

```
The preorder traversal is: + * 2 - 5 1 * 3 2

The postorder traversal is: 2 5 1 - * 3 2 * +

The inorder traversal is: 2 * 5 - 1 + 3 * 2

The expression is: ( ( 2 * ( 5 - 1 ) ) + ( 3 * 2 ) )

The result is: 14
```

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Declare the node of tree in a separate file.
- Include a variable for the left, right and parent of the node.

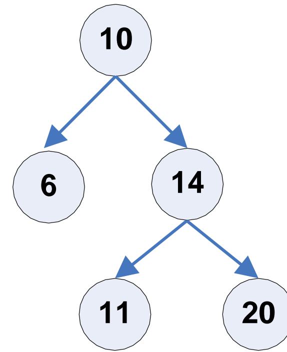
You may want to avoid the following pitfalls

- You should not have the recursive method without having another method that calls it, since you need to give the recursive method the base where it will start with.

6. Laboratory Exercises

Suppose that you want to implement a binary search tree. Create the following classes:

- Class **`TreeNode`**:
  - This class keeps integer value **`data`** and three references; **`left`**, **`right`** children and **`parent`**.
  - It has the **`set`** and **`get`** methods for its attributes.
  - It has method **`hasLeft()`** that return **`true`** if the node has a left child or **`false`** if not.
  - It has method **`hasRight()`** that return **`true`** if the node has a right child or **`false`** if not.
  - It has method **`isExternal()`** that return **`true`** if the node does not have a left and a right child or **`false`** if otherwise.
- Class **`Tree`**:
  - This class keeps a reference to the tree **`root`**.
  - It has an array to keep the location of the nodes in the tree, array size is 100.
  - It has an integer value index initialized to 0 it keeps track of the array index.
  - It has a constructor that creates the following tree:



- And it has the following methods:
  - **printPreorder()** is a method that calls **preorder(root)** to print the preorder traversal of the tree.
  - **preorder(TreeNode)** is a recursive method that prints the preorder traversal of the tree.
  - **printPostorder()** is a method that calls **postorder(root)** to print the postorder traversal of the tree.
  - **postorder(TreeNode)** is a recursive method that prints the postorder traversal of the tree.
  - **printInorder()** is a method that calls **inorder(root)** to print the inorder traversal of the tree.
  - **inorder(TreeNode)** is a recursive method that prints the inorder traversal of the tree.
  - **insertNode(int)** is a method that sets the **root** to point to a **TreeNode** object, if the Tree is empty or calls a method **insert(int, TreeNode)** with the integer value to be inserted and a reference to the **root**.
  - **insert(int, TreeNode)** is a recursive method that inserts the integer value in its proper location to maintain the property of the binary search tree.
  - **TreeNode search(int, TreeNode)** is a recursive method that searches for the integer value in the tree and returns a reference to the node storing the value.
  - **removeExternal(TreeNode)** is a method that deletes a node if it has one child or no children.
  - **findInorderArray(TreeNode)** is a recursive method that stores a reference to **TreeNode** in the array using inorder traversal of the tree.
  - **delete(int)** is a method that locates the node to be deleted node(i) by calling **search(int, root)**. If the node(i) has exactly two children, it will call **findInorderArray(root)** to locate the node(i+1) that follows node(i) in the inorder traversal. It will copy the value in node(i+1) into node(i) and remove node(i+1) by calling **removeExternal(node(i+1))**. If the node has one child or no children, it will call **removeExternal(node(i))**.

❖ Class **Lab9**:

- This class has a **main** method that creates an object of the class Tree and it does the following operations:
  - It prints the preorder traversal of the tree.
  - It prints the postorder traversal of the tree.

- It prints the inorder traversal of the tree.
- Delete a number from the binary search tree then print the resulted tree using inorder.
- Adds a number to the binary search tree then print the resulted tree using inorder.

Laboratory #10 – Graphs (1)

1. Laboratory Objectives

The objective of this laboratory is to train students on how to use Graphs as data structures in the context of object oriented programming.

2. Laboratory Learning Outcomes

After conducting this laboratory students will be able to:

a. Apply basic operations on graphs data structures, such as creating a graph, inserting a vertex, inserting an edge, removing a vertex, removing an edge, counting the number of vertexes, counting the number of edges, finding if two vertexes are adjacent, finding the incident edges on a given vertex, etc.
b. Apply Object Oriented basic concepts in dealing with graphs.

3. Laboratory Introductory Concepts

4. Laboratory Problem Description

Suppose that you want to implement a Graph. Create the following class:

- Class **`Node`**:
  a. This class keeps an integer variable **`id`**.
  b. And a double variable **`weight`**.
  c. A reference to **`Node`** object (**`next`**)
  d. It has the following methods:
     1. A constructor that sets the **`id`**, **`weight`** and **`next`**.
     2. Accessor and modifier methods for **`id`**, **`weight`** and **`next`**.
- Class **`Vertex`**:
  e. This class keeps an integer variable **`id`**.
  f. A reference to **`Node`** object (**`head`**)
  g. A reference to **`Vertex`** object (**`next`**)
  h. An integer variable **`noOfEdges`** initialized to 0
  i. It has the following methods:
     1. A constructor that sets the **`id`**, **`head`** and sets **`next`** to null
     2. Accessor and modifier methods for **`id`**, **`head, next`** and **`noOfEdges.`**
     3. It has method **`void addNode(double w, int i)`** that creates a node **`v`** with the passed information and insert it to the list. If the list is empty it makes the node **`v`** to point to the **`head`** otherwise it goes to the end of the list and adds the node, it also increments the **`noOfEdges`** every time it adds a node.
     4. It has method **`Node search(int id)`** that searches the list for the wanted node

and returns **null** if it did not find it or returns the **node** if it finds it.

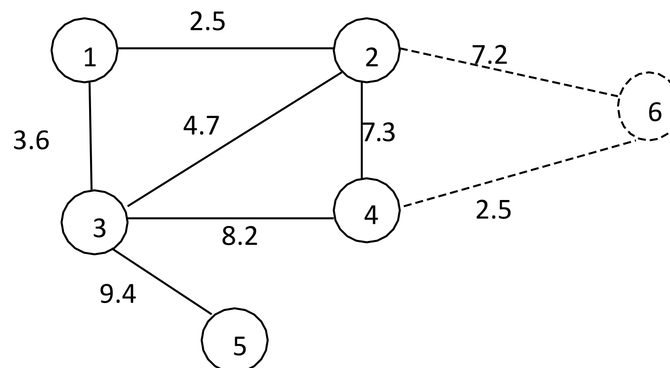
5. It has method **void deleteNode(int id)** that calls method search to find the node then deletes it in the same manner as the delete of the single linked list, it also decrements the **noOfEdges** every time it deletes a node.
6. It has method **void printNode()** that goes through all the nodes in the list and prints their information following format **(id , weight)**.

- Class **Graph**:
  a. A reference to **Vertex** object (**head**)
  b. An integer variable **noOfVertices** initialized to 0
  c. It has the following methods:
     1. Accessor and modifier methods for **head** and **noOfVertices.**
     2. It has method **Vertex addVertex(int id)** that creates a vertex **v** with the passed information and insert it to the list. If the list is empty it makes the vertex **v** to point to the **head** then returns the **head** otherwise it goes to the end of the list and adds the vertex then returns **v**, it also increments the **noOfVertices** every time it adds a vertex.
     3. It has method **Vertex search(int id)** that searches the list for the wanted vertex and returns **null** if it did not find it or returns the **vertex** if it finds it.
     4. It has method **void deleteVertex(int id)** that calls method search to find the vertex if the vertex is not null it goes through the list to locate the vertex to be deleted and its previous vertex. If the vertex to be deleted is the head of the list then it will go to each vertex using a while loop and calls method deleteNode(id) to delete all the occurrences of that vertex in the other vertices then it deletes the vertex, it also decrements the **noOfVertices** every time it deletes a vertex.
     5. It has method **void printVertex()** that goes through all the vertices in the list and prints their information following format **{id}**.
     6. It has method int degree(int id) that calls method search with the id and if the vertex not null it returns **noOfEdges** otherwise 0.

- Class **ListGraph**:
  a. It has the main method that does the following:
     1. Creates object from class **Graph** and **Vertex**.
     1. It prompts the user to enter the number of vertices in the graph.
     2. Then it loops from 1 to the given number of vetices and reads the id of the vertex and calls **addVertex(id)** to adds it to the graph.
     2. For every vertex that the user enters the systems prompts him to enter the number of edges for that vertex.
     3. Then it loops from 1 to the given number of edges to read the id and weight of that edge then calls method **addNode(weight, id)** to add the edge.

4. After constructing the graph it will call method **`printVertices()`** to print the graph.
5. Then the system will ask the user to enter a vertex to be added to the graph and calls method **`addVertex(id)`** also it will ask the user to enter the number of edges for that vertex.
6. Then it loops from 1 to the given number of edges to read the id and weight of that edge then it will call method **`graph.search(v).addNode(w,id)`** to add the edge to the other vetices that are connected to this new vertex then it will call method **`addNode(weight, id)`** to add the edge.
7. After adding the vertex to the graph it will call method **`printVertices()`** to print the graph.
8. Then the system will ask the user to enter a vertex to be deleted from the graph and calls method **`deleteVertex(id).`**
9. After deleting the vertex from the graph it will call method **`printVertices()`** to print the graph.
10. Finally the system will find and print the degree of the deleted vertex by calling **`degree(id),`** and print the total number of vertices in the graph by calling **`getNum()`**

Test your code with the following graph



Sample Output:

```
Enter the number of vertices in your graph:
5
Enter the id of vertex number 1:
1
Enter the number of edges of vertex 1:
2
Enter vertex number 1
2
Enter the weight of vertex number 1
2.1
 Enter vertex number 2
 3
 Enter the weight of vertex number 2
3.2
```

```
Enter the id of vertex number 2:
2
Enter the number of edges of vertex 2:
3
Enter vertex number 1
1
Enter the weight of vertex number 1
2.1
Enter vertex number 2
3
Enter the weight of vertex number 2
5.6
Enter vertex number 3
4
Enter the weight of vertex number 3
3.7
Enter the id of vertex number 3:
3
Enter the number of edges of vertex 3:
4
Enter vertex number 1
1
Enter the weight of vertex number 1
3.2
Enter vertex number 2
2
Enter the weight of vertex number 2
5.6
Enter vertex number 3
4
Enter the weight of vertex number 3
2.8
Enter vertex number 4
5
Enter the weight of vertex number 4
4.1
Enter the id of vertex number 4:
4
Enter the number of edges of vertex 4:
2
Enter vertex number 1
2
Enter the weight of vertex number 1
3.7
Enter vertex number 2
3
Enter the weight of vertex number 2
2.8er the id of vertex number 5:
5
```

```
Enter the number of edges of vertex 5:
1
Enter vertex number 1
3
Enter the weight of vertex number 1
4.1
Your Complete Graph
{ 1  } ( 2 - 2.1 )( 3 - 3.2 )
{ 2  } ( 1 - 2.1 )( 3 - 5.6 )( 4 - 3.7 )
{ 3  } ( 1 - 3.2 )( 2 - 5.6 )( 4 - 2.8 )( 5 - 4.1 )
{ 4  } ( 2 - 3.7 )( 3 - 2.8 )
{ 5  } ( 3 - 4.1 )
Enter a vertex to be added:
6
Enter the number of edges of vertex 6:
2
Enter vertex number 1
4
Enter the weight of vertex number 1
2.5
Enter vertex number 2
2
Enter the weight of vertex number 2
7.2
Your Graph after adding vertex 6
{ 1  } ( 2 - 2.1 )( 3 - 3.2 )
{ 2  } ( 1 - 2.1 )( 3 - 5.6 )( 4 - 3.7 )( 6 - 7.2 )
{ 3  } ( 1 - 3.2 )( 2 - 5.6 )( 4 - 2.8 )( 5 - 4.1 )
{ 4  } ( 2 - 3.7 )( 3 - 2.8 )( 6 - 2.5 )
{ 5  } ( 3 - 4.1 )
{ 6  } ( 4 - 2.5 )( 2 - 7.2 )
Enter a vertex to be deleted:
3
Your Graph after deleting vertex 3
{ 1  } ( 2 - 2.1 )
{ 2  } ( 1 - 2.1 )( 4 - 3.7 )( 6 - 7.2 )
{ 4  } ( 2 - 3.7 )( 6 - 2.5 )
{ 5  }
{ 6  } ( 4 - 2.5 )( 2 - 7.2 )
Degree of vertex 3 is 0
Number of vertices is: 5
```

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Declare the node of graph in a separate file.
- Include a variable for the weight of the node if it's a weighted graph.

You may want to avoid the following pitfalls

- When you delete a vertex make sure to delete all its corresponding edegs.

6. Laboratory Exercises

Write the code of class Node.

Laboratory #11 – Graphs (2)

1. Laboratory Objectives

The objective of this laboratory is to provide more advanced training for students on how to use the Graph data structure in the context of object oriented programming.

2. Laboratory Learning Outcomes

After conducting this laboratory students will be able to:

1. Apply a number of additional Graph operations, such as enumerating the Minimum Spanning Trees of a simple undirected Graph, Traversing a Graph using the Depth-First Technique, and Traversing a Graph using the Breadth-First technique.
2. Apply Object Oriented basic concepts in dealing with Graphs.

3. Laboratory Introductory Concepts

4. Laboratory Problem Description

Modify the previous lab classes (**`Node`**, **`Vertex`**, **`Graph`** and **`ListGraph`**) as follows:

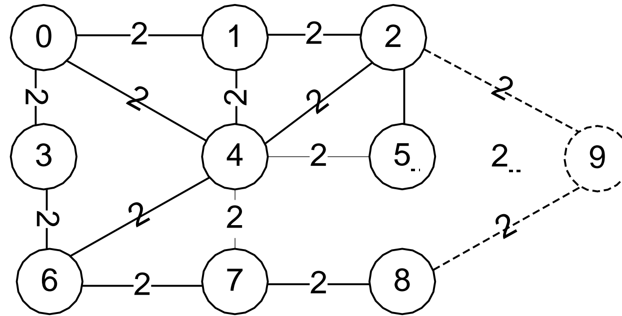
- ❖ Class **`Node`**:
  - a. Add a **`boolean`** variable **`visited`**.
  - b. In the constructor set **`visited`** to **`false`**.
  - c. Add set and get methods for **`visited`**.

- ❖ Class Vertex:
  - a. Add a boolean variable visited.
  - b. In the constructor set visited to false.
  - c. Add set and get methods for visited.

- ❖ Class Graph:

  Add the following methods:
  - a. void dfs(int vertex) that prints the depth first search traversal of the graph.
  - b. void bfs(int vertex) that prints the breadth first search traversal of the graph.
  - c. void clear() that makes the visited variable for all the vertices as falls.

- ❖ Class ListGraph:



It has the main method that does the following:

a. Creates object from class **Graph** and **Vertex**.
b. It prompts the user to enter the number of vertices in the graph.
c. Then it loops from 1 to the given number of vetices and reads the id of the vertex and calls **addVertex(id)** to adds it to the graph.
d. For every vertex that the user enters the systems prompts him to enter the number of edges for that vertex.
e. Then it loops from 1 to the number of edges to read the id and weight of that edge then calls method **addNode(weight, id)** to add the edge.
f. After constructing the graph it will call method **printVertices()** to print the graph.
g. Then the system will ask the user to enter a vertex to be added to the graph and calls method **addVertex(id)** also it will ask the user to enter the number of edges for that vertex.
h. Then it loops from 1 to the given number of edges to read the id and weight of that edge then it will call method **graph.search(v).addNode(w,id)** to add the edge to the other vetices that are connected to this new vertex then it will call method **addNode(weight, id)** to add the edge.
i. After adding the vertex to the graph it will call method **printVertices()** to print the graph.
j. Then the system will ask the user to enter a vertex to be deleted from the graph and calls method **deleteVertex(id).**
k. After deleting the vertex from the graph it will call method **printVertices()** to print the graph.
l. Then the system will ask the user to enter the id of the vertex to find its degree by calling **degree(id)**
m. Then the system will print the total number of vertices in the graph by calling **getNum()**
n. Then the system will print the depth first search by calling **dfs(0)**.
o. Then the system calls **clear()**.

p. Then the system will print the breadth first search by calling **`bfs(0)`**.

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Declare the node of graph in a separate file.
- Include a variable visited for the node.

You may want to avoid the following pitfalls

- When you call the search method again make sure to call method clear to set all the visited nodes to false again.

6. Laboratory Exercises

❖ Class Queue is a new class:

a. This class keeps array integer variable elements, integer variables f, r and N that are not private.

b. It has a constructor that takes one parameter int to set N then it creates the array elements with the size to be the passes integer; also it sets f and r to zero and N to the passed integer.

c. It has the following methods:

1. public int size() that returns this equation ((N-f+r)%N).
2. public Boolean isEmpty() that returns true if size() is zero or false otherwise.
3. public it front() that returns -1 if isEmpty() is true or it returns f if otherwise.
4. public void enqueue(int e) that prints a message "Queue is full" if the size() is equal to N-1 or adds e to elements[r] and makes r to be (r + 1) % N.
5. public int dequeue()that returns -1 if isEmpty() is true or it copies elements[f] to another integer variable and makes f to be (f + 1) % N then returns the copied element.

Laboratory #12 – Searching

1. Laboratory Objectives

The objective of this laboratory is to train students on how to use the searching algorithms in the context of object oriented programming.

2. Laboratory Learning Outcomes

After conducting this laboratory students will be able to:

1. Apply a number of searching algorithms, such as Sequential Search, Binary Tree Search, Depth-First Search, and Breadth-First Search.
2. Apply the hash based searching algorithm.
3. Apply Object Oriented basic concepts in dealing with search.

3. Laboratory Introductory Concepts

4. Laboratory Problem Description

Create class `searching` that has the following:

- Method `seqSearch(int[] list, int element)` that returns the index at which the number is in the list or -1 if the number is not in the list.
- Method `BinarySearch(int[] list, int start, int end, int x)` it's a recursive method that returns -1 if the number is not found or its index if it's found. This method searches the list based on the properties of the binary search tree.
- Method `main` that doeas the following:
  - Creates an array `list` using the array initialize.
  - Print the array using `Arrays.toString(list)`.
  - Calls `seqSearch(list, 77)`.
  - Prints the result of the search.
  - Calls `seqSearch(list, 3)`.
  - Prints the result of the search.
  - Calls `BinarySearch(list,0, 5, 77)`.
  - Prints the result of the search.
  - Calls `BinarySearch(list,0, 5, 44)`.
  - Prints the result of the search.

Problem Description:

Suppose you wish to implement the hashing function using java, create the following classes:

Class `HashEntry` that has the following:

- It has two instance variable of type integer `value` and `key`.
- It a constructor that takes two values to set the instance variables.
- It has get methods for both of its instance variables.

Class `Hashing` that has the following:

- It has a class variable of type integer `size` that is constant with the value 128.
- It has an instance variable of type array of `HashEntry` `table`.
- It a constructor that declares the array `table` then initializes it with `null` values.
- It has method `void put(int key, int value)` that finds the proper place for the new value then inserts it in the hash table.
- It has method `int get(int key)` that loops the hash table to find the corresponding key then it returns the value at which the key is found in the array or -1 if the key is not in the array.
- It has method `main` that doeas the following:
    - Creates an object `hash` of class `Hashing`.
    - Inserts 10 numbers into the hash table using the put method, where the key is equal to (i + 1) and the value is equal to (i * 2), where i = 0 – 9.
    - Searches for number 7 and 25 by calling `get(int)` method.
    - If the search result is equal to -1 the method prints a message that the key is not found other wish it prints the value of the corresponding key.

Sample Output:

```
Hash: 7 Key: 7  Value: 12
The key 7 is found
The key 25 is not found
```

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Import class arrays.
- Use Arrays.toString instead of having a loop to print the array contents.

You may want to avoid the following pitfalls

- You have to use Arrays.sort to sort the array before you call the BinarySearch.

6. Laboratory Exercises

- Write `seqSearch(int[] list, int element)` method.

Laboratory #13 – Sorting

1. Laboratory Objectives

The objective of this laboratory is to train students on how to perform sorting of a sequence of input data in the context of object oriented programming.

2. Laboratory Learning Outcomes

After conducting this laboratory students will be able to:

1. Develop and test a Java program that can perform the in-place Quick Sort Algorithm on a sequence of input data implemented as an array S.
2. Develop and test a Java program that can Recursively Merge two sorted sequences S1 and S2 implemented as linked lists into a third sequence S3 using the principles of the Merge Sorting Algorithm.
3. Develop and test a Java program that can perform the heap sort.
4. Apply Object Oriented basic concepts in sorting data sequences.

3. Laboratory Introductory Concepts

4. Laboratory Problem Description

- Create a class **`Sorting`** with the following methods:
  - **`void fillArray(int a[ ])`** that fills the array **a** with random integers between 1 and 100.
  - **`void printArray(int a[ ])`** that prints the array **a**.
  - **`int count(int a[ ])`** that counts and returns the number of nonzero elements in the array **`a[]`**.
  - **`void quickSort(int a[ ])`** that sorts the array **`a[]`** using quick sort.
  - **`void mergeSort(List in)`** that sorts the list **`in`** using merge sort.
  - **`void merge (List s1, List s2, List s)`** that merges two sorted lists **`s1`** and **`s2`** into a sorted list **`s`**.
  - **`main`** that tests the different methods of class sorting.

- Create a class **`Node`** with the following:
  - This class keeps `integer` value `element` and one reference; `next`.
  - It has a constructor that initializes the instance variables.
  - It has the set and get methods for its attributes.

- Create a class **`List`** with the following:
  - This class keeps one reference; `head`.
  - It has a constructor that sets `head` to `null`.
  - **`void createList()`** that creates a list with 10 nodes and fills it with random integers between 1 and 100.

- **`void printList()`** that prints the list.
- **`int countNodes()`** that counts and returns the number of nonzero nodes in the list.
- **`void addLast(Node p)`** that adds a node at the end of the list.
- **`Node remove(Node p)`** that deletes a node from the list and returns it.
- **`boolean isEmpty ()`** that returns true if the list is empty.
- **`Node first()`** that returns the first node in the list.

Sample Output:

```
The array before sorting:
67   36   50   33   72   90   84   0    72   58
The array after quick sort:
33   36   50   67   72   72   84   90   72   58
The first list before sorting:
59   97   64   38   57   63   30   77   73   5
The first list after merge sort:
5    30   38   57   59   63   64   73   77   97
The second list before sorting:
83   77   33   80   28   42   79   66   85   36
The second list after merge sort:
28   33   36   42   66   77   79   80   83   85
The two lists after merging them:
5    28   30   33   36   38   42   57   59   63   64   66   73   77   77
79   80   83   85   97
```

Problem Description:

Create a class **`HeapSorter`** with the following attributes and methods:

- Attributes: Integer variables n and an array of integers **`a[]`**.
- Methods:
  - **`void sort(int a0[ ])`** that initializes the array **`a[]`** to the value of array a0 passed as a parameter and assigns the length of array **`a[]`** to variable n then calls **`heapSort().`**

  - **`void heapSort()`** that sorts the array **`a[]`** using the heap sort.
    Algorithm heapSort:
    Input: binary tree
    Output: vertex labels in descending order
    buildHeap
    while n is greater than 1 do
    decrement n by 1
    call exchange(0, n)
    call downHeap(0)

  - **`void buildHeap()`** that builds a heap for a binary tree T of depth d(T), where d(T)=n/2.

Algorithm buildHeap:

Input: binary tree T of depth d(T)

Output: heap

for v = d(T) – 1 to 0 do

call downHeap(v)

- **`void downHeap(int v)`** that builds a heap from a binary tree by proceeding from bottom-up. It makes a heap from a semi-heap.

  Algorithm downHeap(v):

  Input: semi-heap with root

  Output: heap

  while v does not have the heap property do

  choose direct descendent w with maximum label a(w)

  call exchange a(v) and a(w)

  set v = w

- **`void exchange(int i, int j)`** that swaps the value of i and j.

Sample Output:

```
List before sorting:
54  55  20  9  61  53  42  54  100  57
List after sorting:
9  20  42  53  54  54  55  57  61  100
```

5. Laboratory Instructions

You may want to make use of the following good programming practices

- Include a method to count the number of non zero elements in the array or the list.
- To generate random numbers between 1 and 100 use Math.Random() * 100.

You may want to avoid the following pitfalls

- You have to cast the number from the random generator like this (int) Math.Random().

6. Laboratory Exercises

- Write method **`void fillArray(int a[ ])`** from class **`sorting`**.
- Write method **`void createList()`** from class **`List`**.

Appendix A: Rules to fallow by Computer Lab Users

- The loud conversations / discussion that disturbing the other users is prohibited.
- Audio CDs or applications with audio output may only be used with headphones with minimum volume that it should not be disturb other users.
- All cell phones are to be turned off or set to silent while in the lab. If you receive a phone call, you should exit the lab before answering your cell phone.
- Do not bring food or beverages inside the lab.
- Any file saved on the computer hard drive will be deleted without notice. Students should save their work onto an external storage device such as USB drive or CD.
- Changing hardware and software configurations in the computer labs is prohibited. This includes modifications of the settings, modification of system software, unplugging equipment, etc.
- Open labs are reserved for academic use only. Use of lab computers for other purposes, such as personal email, non-academic printing, instant messaging, playing games, and listening to music is not permitted.
- Please leave the computer bench ready for the next patron. Leave the monitor on the login screen, and do not forget to take goods related to you. While leaving computer bench please push the chair inside the computer bench.
- Users are responsible for their own personal belongings and equipment. Do not leave anything in the Computer Lab unattended for any length of time. The Computer Labs staffs are not responsible for lost or stolen items.
- Users are not allowed to clear paper jams in the printer by themselves.
- Operate the lab equipments with care.
- After using white-board the user must clean for other user.

Thanks for your cooperation.

Information Science Department

Appendix B: Endorsement

LABORARTORY MANUAL FOR

ISC 241 Data Structures

#	Instructor name	Remarks	Signature	Date
1	Professor Mostafa Abd-El-Barr			
2	Professor Jehad Al Dallal			
3	Eng. Aisha Al-Noori			
4	Eng. Aisha Al-Houti			