Experiment#	07	Student ID	2200032060
Date		Student Name	A.Brahma Naidu

Experiment Title: Clean Coding Techniques (CCT) & Test-Driven Development (TDD)

Aim/Objective: To analyse the implementation of Clean Coding Techniques (CCT) & Test-Driven Development (TDD) for the real-time scenario.

Description:

The student will understand the concept of Clean Coding Techniques (CCT) & Test-Driven Development (TDD).

Pre-Requisites:

Knowledge: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1. Discuss about the various clean coding Techniques.

* various clean coding techniques:-

1) Descriptive Naming
2) single Responsibility principle (SRP)
3) DRY (Dont Repeat yourself)
4) comments and Documentation.
5) consistent formatting
6) Avoiding magic numbers
7) Avoiding nested callbacks (callback Hell)
8) sepexation of concerns.
9) Meaningful variable and Function Names.
10) proper Error Handling
11) Limiting function/Method length
12) Avoiding global variables
13) KISS (Keep it simple, stupid)
14) YAGNI (you Ain't Gonna needit)

15) Avoiding over engineering
16) Unit Testing and Test Driven Development
17) code Reviews and pair programming
18) using version control system (eg., Git)
19) proper file and folder organisation.
20) Minimizing side effects.

Experiment#		Student ID	
Date		Student Name	

2. Discuss the necessity of Test-Driven Development (TDD)

Test Driven Development (TDD) is necessary as it ensures code correctness, improves design, and provides a safety net for refactoring by writing tests before writing the actual code.

In-Lab:

Write a JAVA Program to manage the Student Information System (SIS) with the incorporation of the concepts of Classes, Objects, Constructors, Interfaces and inheritance by following the principles of Clean Coding Techniques (CCT).

- **Procedure/Program:**

```
interface student{
void displayInfo();
}
class person{
string name;
person(string name){
this.name = name;
}
void displayInfo(){
system.out.println("Name: "+name);
}}
class studentInfo extends person implements
student{
int rollNumber;
double marks;
studentInfo(string name, int rollNumber,
                double marks){
super(name);
this.rollnumber = rollnumber;
this.marks = marks;
```

```
public void display Info(){
super.display Info();
system.out.println("Roll Number: "+ rollNumber);
system.out. println(" Marks: "+ Marks);
}
}
public class SIS App{
public static void main (String[]args){
student Info student = new student Info ("sai",
                                        3071,95.4);
student. display Info();
}
}
```

- **Data and Results:**

Name: sai
Roll Number: 3071
Marks: 95.4

- **Analysis and Inferences:**

The provided Java program demonstrates a student information system using classes, objects, inheritance and interfaces. It follows clean coding principles by utilizing clear naming, seperation of concerns, and proper encapsulation.

Sample VIVA-VOCE Questions (In-Lab):

1. **State about "Write Unit Tests" principle in Clean Coding Techniques.**

"write unit tests" is a clean coding principle that involves creating automated tests for individual parts (or units) of your code to ensure they work correctly.

2. **Discuss about Single Responsibility Principle (SRP).**

single Responsibility principle (SRP) means each class should do only one thing, making code easier to manage.

3. **Illustrate the difference between Assertions and Annotations.**

* Assertions

→ purpose: validate expectations in code

→ usage:- used within code to check outcomes.

* Annotations:-

→ purpose:- Add metadata (or) instructions.

→ usage:- Added above methods/classes.

4. **List the Assertions in JAVA Test Driven Development (TDD)**

* Annotations in TDD

→ assertEquals (expected, ~~output~~ actual)

→ assertTrue (condition)

→ assertFalse (condition)

→ assertNull (Object)

→ assertNotNull (Object)

Experiment#		Student ID	
Date		Student Name	

Post-Lab:

1. Write a Test-Driven Development program to accept the password when the length of it should be between 5 to 10 characters ("Password validator")

Input:

Abc123

Output:

Valid password: accepted

Procedure/Program:

```
import org.junit.Test;
import org.junit.Assert.*;
public class passwordvalidator{
 public static boolean isvalidpassword (String
                                    Password){
 int length= password.length();
 return length>=5 && length<=10;
 }
}
public class passwordvalidatorTest{
public void test valid password(){
assertTrue (passwordvalidator.isvalid Password
            ("Abc123"));
}
public void test invalid short Password(){
assert false (passwordvalidator.is valid Password
            ("123"));
}
public void test invalid long Password(){
assertfalse (passwordvalidator.is valid Password
```

("AbC123456-78"));

}

}

- **Data and Results:**

* Input :-

  AbC 123

* output :-

  valid Password

- **Analysis and Inferences:**

* Analysis involves examining and interpreting data to gain insights and draw conclusions.
* The code has a 'PasswordValidator' class with a method to check if a password's length is between 5 and 10 characters.
* The 'PasswordValidatorTest' class has tests for valid, too short, and too long passwords using Junit assertions.

Post-Lab:

2. Write a Java program to check whether the given number is odd or even and dounit testing in JUnit.

Procedure/Program:

```
import org.junit.Test;
import static org.junit.Assert.*;
public class Numbercheckers{
public static boolean is even (int number){
return number %2==0;
}
public static boolean is odd (int number){
return!is even (number);
}
}
public class NumberCheckerTest{
@Test
public void test EvenNumber (){
assert True (Number Checker.is even(4));
}
@Test
Public void test odd Number(){
assert True (NumberChecker.isodd(7));
}
@Test
Public void test zero is even (){
assert True (Numberchecker.is even (-6));
}
```

@Test
public void testNegativeOddNumber(){
assertTrue(NumberChecker.isOdd(-3));
}
}

Experiment#	TO BE FILLED BY STUDENT	Student ID	TO BE FILLED BY STUDENT
Date	TO BE FILLED BY STUDENT	Student Name	TO BE FILLED BY STUDENT

- **Data and Results:**

*output

All tests passed (5 tests)

- **Analysis and Inferences:**

* The code checks if the numbers or odd (or) even.
* Tests ensure the code works as expected.

Evaluator Remark (if Any):	
good.	**Marks Secured:** 48 **out of 50**
	30/8/2023 **Signature of the Evaluator with Date**

Experiment#	08	Student ID	2200032060
Date		Student Name	A.Brahma Naidu

Experiment Title: Generics with Interfaces

Aim/Objective: To analyse the implementation of the concept of Generics with Interfaces for the real-time scenario.

Description:

The student will understand the concept of Generics with Interfaces.

Pre-Requisites:

Knowledge: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1. Discuss about the various types of Generics in JAVA.

* Types of Generics in Java are:-

(i) class Generics

(ii) method Generics

(iii) wild cards

(iv) Bounded wild cards

(v) Generic Interfaces and classes.

(vi) Generic methods.

(vii) Type erasure.

2. Discuss the necessity of Interfaces in Generics.

* necessity of Interfaces in Generics:-

(i) Abstraction:-
→ Interfaces define a common contract for classes.

(ii) polymorphism:-
→ Allows different classes to be treated uniformly.

(iii) code Reusability
→ Implementing a single interface for multiple classes.

(iv) Flexibility:-
→ easily switch implementations while maintaining functionality.

In-Lab:

Write a Java Program to identify the Maximum Value and Minimum Value in the arrays of different datatypes like Integer, String, Character & float by incorporating the concept of Generics with interfaces.

- **Procedure/Program:**

```
interface comparable <T> {
T getMaxValue (T[] arr);
Tget Min value (T[] arr);
}
class ArrayHelper <T extends comparable<T>> {
T getMaxvalue (T[] arr){
Tmax = arr[0];
for (T item : arr)
if (item.compareTO (max) >0) max=item;
return max;
}
T get Min value (T[] arr){
T min = arr[0];
for (T item : arr)
if (item.compareTO (min) <0) min=item;
return min;
}
}
class main {
Public static void main (String [] args){
Integer [] int Arr = {10, 5, 20, 8};
String [] StrArr = {"apple", "banana", "orange", "grape"};
ArrayHelper <Integer comparable> int Helper =
                    new ArrayHelper <> ();
```

```
ArrayHelper <String comparable> StrHelper=
                    new ArrayHelper<>();

system.out.println("Max Integer:"+intHelper.
                    getMaxValue(intArr));

system.out.Println("Min Integer:"+intHelper.
                    getMinvalue (IntArr));

system.out.println("Max String:"+StrHelper.
                    getMaxvalue (StrArr));

system.out.println("Min String:"+StrHelper.
                    getMinvalue (StrArr));
}
}
```

- **Data and Results:**

* output:-

Max Integer: 20

Min Integer: 5

Max String: orange

Min string: apple.

- **Analysis and Inferences:**

→ The above code uses interfaces & generics to find max/min different arrays. Defines 'comparable' interface & 'ArrayHelper' class shows int with int & string arrays, successfully identifying max/min values.

Experiment#		Student ID	
Date		Student Name	

Sample VIVA-VOCE Questions (In-Lab):

1. List the benefits of Generics.

* Benefits of Generics :-

→ type safety

→ code reuse

→ eliminates casting in collections & classes.

2. Discuss about the various types of Generics implementation in Java.

* various types of Generics implementation in Java.

(i) class Generics

(ii) Method Generics

(iii) wild card types

(iv) Bounded wildcards

(v) Generic Interfaces/classes.

(vi) Generic methods.

(vii) type Erasure.

3. **Illustrate about "Type Parameter Naming Conventions" In Generics**

* Type parameter Naming conventions in Generics

→ single Letter:-
use upper case letters like "A", "B", "Y", "Z"

→ Meaningful Names
choose letters that reflect the type's purpose.

→ consistency:-
Maintain consistent naming across code.

→ clarity:-
select names that make the code understandable.

4. **State about Generic Classes with example**

* Generic classes:-

→ Generic classes allow defining a class with a type parameter that can be replaced with specific types.

eg:- List<T> can hold a list of any type ("List<Integer>", List<String>).

5. **State about Generic Interfaces with example.**

* Generic Interfaces:-

→ Generic Interfaces define contracts with type parameters.

Eg:- 'Comparable<T>' allows comparing objects of different types ('Integer', 'String').

Post-Lab:

1. Create a generic class that implements a binary search algorithm. Test the class with different data types such as integers, doubles, and strings.

Procedure/Program:

```
public class Binary search <T extends comparable<T>>{
public Integer search (T value, T[] array){
int left=0, right= array.length-1;
while(left <= right){
int middle= left+(right-left)/2;
if(array[middle].equals (value)){
return middle;
}else if (array [middle].compareTo (value) <0){
left= middle+1;
}else{
right= middle-1;
}}
return null;
}
```

```
public static void main (String[] args) {
Integer[] int Array = {1, 3, 5, 7, 9, 11, 13, 15};
BinarySearch <Integer> int search = new BinarySearch<>();
Integer int Index = int search.search (7, int Array);
if (int Index != null){
System.out.Println ("found at index" + int Index);
}else{
System.out.Println ("Not Found");
}
Double [] double Array = {1.5, 2.5, 3.5, 4.5, 5.5, 6.5};
Binary Search <Double> double search = new Binary Search<>();
Integer double Index = double search.search (4.5, double Array);
if (double Index != null){
System.out.Println ("found at index" + double Index);
}else{
System.out.Println ("Not Found");
}
String[] String Array = {"apple", "banana", "cherry", "grape",
                         "orange", "Pear"};
Binary search <String> String search = new Binary search <>();
Integer String Index = String search.search ("orange", String Array);
if (String Index != null){
System.out.Println ("found at index" + String Index);
}else{
System.out.Println ("Not found");
}}
}
```

- **Data and Results:**

output:-

found at index 3

found at index 3

found at index 4

- **Analysis and Inferences:**

This Java code creates a 'Binary search' class and demonstrates how to test it with different datatypes:

(i) integers

(ii) doubles and

(iii) strings.

Experiment#		Student ID	
Date		Student Name	

Post Lab:

2. Create a generic method that sorts an array of objects using a bubble sort algorithm. Test the method with different types of objects such as integers, doubles, and strings.

Procedure/Program:

```
public class Bubblesort{
public static <T extends comparable <T>> void bubblesort
                                        (T[]array){

int n = array.length;
for (int i = 0; i<n-1; i++){
for(int j = 0; j<n-i-1; j++){
if (array[j].compareTo(array[j+1]) > 0){
Temp = array[j];
array[j] = array[j+1];
array[j+1] = temp;
}}
}
public static void main (String[] args){
Integer[] int Array = {5, 2, 8, 1, 3};
bubble sort (int Array);
system.out.Println (java.util.Arrays.toString (int Array));
Double[] double Array = {3.5, 1.2, 2.8, 5.1, 4.3};
bubble sort (double Array);
system.out.Println (java.util.Arrays.toString (double Array));
String[] String Array = {"banana", "apple", "grape", "cherry",
                         "orange"};
bubble sort (String Array);
System.out.Println (java.util.Arrays.toString (String Array));
}
}
```

- **Data and Results:**

*output :-

[1, 2, 3, 5, 8]

[1.2, 2.8, 3.5, 4.3, 5.1]

[apple, banana, cherry, grape, orange]

- **Analysis and Inferences:**

The code defines a generic 'bubblesort' method that sorts arrays of various types using the bubblesort algorithm and the 'comparable' interface for comparison.

Evaluator Remark (if Any):	
Practice the Code	Marks Secured: 48 out of 50
	30/8/2023 Signature of the Evaluator with Date

Experiment#	09	Student ID	2200032060
Date		Student Name	A Brahma Naidu

Experiment Title: Stacks and Queues

Aim/Objective: To analyse the implementation of the concept of Stacks and Queues with Interfaces for the real-time scenario.

Description:

The student will understand the concept of Stacks and Queues.

Pre-Requisites:

Knowledge: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1. Write a JAVA program for Basic ADT Operations on Stack Data Structure.

```
import java.util.Stack;
public class StackADTExample{
public static void main (String[]args){
Stack <Integer> Stack = new Stack <>();
Stack.Push (5);
Stack.Push (10);
Stack.Push (15);
int poppedvalue = Stack.Pop();
System.out.Println("Popped value:"+poppedvalue);
int topvalue = Stack.peek();
System.out.Println("Topvalue:"+topvalue);
boolean isEmpty = Stack.isEmpty();
System.out.Println("Is Stack empty?"+isEmpty);
int size = Stack.size();
System.out.Println("Stack size:"+size);
}
}
```

*output :-

popped value : 15

Top value : 10

Is stack empty? False

stack size : 2

2. Write a JAVA program for Basic ADT Operations on Queue Data Structure.

```
import java.util.LinkedList;
import java.util.Queue;
public class QueueADTExample{
public static void main (String[] args){
Queue <Integer> queue = new LinkedList <> ();
queue.offer(5);
queue.offer(10);
queue.offer(15);
int dequeued value = queue.poll();
system.out.println("Dequeued value:"+dequeued value);
int front value = queue.peek();
system.out.println("Front value:"+front value);

boolean is empty = queue.isempty();
system.out.Println("Is queue empty?"+is empty);
int size = queue.size();
system.out.Println("Queue size:"+size);
}
}
```

* output:-

Dequeued value : 5

front value : 10

Is queue empty ? False

Queue size : 2

In-Lab:

Create a generic interface for a stack data structure with additional methods for peeking at the top element without removing it and checking if the stack is empty. Implement the interface using a linked list and an array. Test the implementations with different data types.

- **Procedure/Program:**

```
interface Stack<T> {
void push (T item);
T pop(), T peek();
boolean isEmpty();
}
class LinkedList Stack<T> implements Stack<T> {
private java.util.LinkedList<T> list=new java.util.
                                    LinkedList<>();

public void push (T item){
 list.addFirst (item);
}
public T pop(){
return list.pollFirst();
}
public T peek (){
return list.peekFirst();
}
public boolean isEmpty(){
return list.isEmpty();
}
}

class ArrayStack<T> implements Stack<T> {
Private object[] array = new object[10];
Private int size = 0;
Public void push (T item) {
if(size == array.length){
object[] newArray = new object[array.length *2];
```

```
System.arraycopy(array, 0, newArray, 0, size);
array = newArray;
}
array[size++] = item;
}
public T pop(){
if (isEmpty()){
throw new IllegalStateException("Stack is empty");
}
T item = (T) array[--size];
array[size] = null;
return item;
}
public T peek(){
if (isEmpty()){
throw new IllegalStateException("Stack is empty");
}
T item = (T) array[size-1];
return item;
}
public boolean isEmpty(){
return size == 0;
}
}
public class Main{
public static void main (String[] args){
Stack<Integer> LinkedListStack = new LinkedList<>();
linkedliststack.push(5);
linkedliststack.push(10);
linkedliststack.push(15);
System.out.println("peeked:" + linkedliststack.peek());
System.out.println("Is empty?" linkedliststack.isEmpty());
while (!linkedliststack.isEmpty())
System.out.println("popped:" + linkedliststack.pop());
```

```
Stack<String> arrayStack = new ArrayStack<>();
arrayStack.push("apple", "banana", "cherry");
System.out.println("Peeked: "+arrayStack.peek());
System.out.println("Is empty? "+arrayStack.isEmpty());
while(!arrayStack.isEmpty())
System.out.println("Popped: "+arrayStack.pop());
}
}
```

- **Data and Results:**

Output :-

Peeked: 15

Is empty? False

Popped: 15

Popped: 10

Popped: 5

Peeked: cherry

Is empty? False

Popped: cherry

Popped: banana

Popped: apple.

- **Analysis and Inferences:**

This code defines the generic 'stack' interface with methods for pushing, popping and peeking, and checking if the stack is empty. It then provides two implementations

(i) Linked list stack - using linked list

(ii) Array stack - using an array

→ Finally, it tests both implementations with different datatypes (integer and string) to demonstrate their functionality.

Sample VIVA-VOCE Questions (In-Lab):

1. **List the Differences between Stack and Queue ADT.**

* Stack:-

→ operates on principle of LIFO

→ Elements are added & removed from top.

→ used for tasks like managing function calls, undo functionality and parsing expressions.

* Queue:-

→ operates on principle of FIFO

→ Elements are added at the back and removed from front.

→ used for tasks like managing tasks in a printer queue, BFS traversal, and event handling.

2. Discuss about the Priority Queue with example

* Priority Queue:

→ Elements are stored based on priorities.

→ Higher priority items are dequeued first

Eg:- Emergency room treating critical patients before others in a Hospital

3. Illustrate about "PERFORMING QUEUE ADT USING STACK DATA STRUCTURE".

* Using the Queue ADT like a stack:-

→ Imagine stacking books on a table. You add and remove books from top, just like pushing and popping items onto/from a queue used as a stack.

4. Illustrate about "PERFORMING STACK ADT USING QUEUE DATA STRUCTURE".

* using the stack ADT like a queue:-

→ Think of a stack of plates. you can add and remove plates from the top. Similarly a stack can mimic a queue by pushing and popping items from the same end, creating FIFO behaviour.

5. Discuss the necessity of Priority Queue Data Structure.

* priority queue:-

→ vital for tasks with varying importance, it ensures critical tasks are handled first, like a hospital treating severe patients promptly.

Pre Lab:

1. Create a generic class that implements a priority queue data structure. Test the class with different data types such as integers, doubles, and strings.

Procedure/Program:

```
import java.util.PriorityQueue;
class PriorityQueueExample<T> {
private PriorityQueue<T> priorityQueue = new PriorityQueue<>
                                        ();
public void enqueue(T item) {
priorityQueue.add(item);

}
public T dequeue() {
return priorityQueue.poll();
}
public boolean isEmpty() {
return priorityQueue.isEmpty();
}
public int size() {
return priorityQueue.size();
}
}

public class Main {
public static void main (String[] args) {
PriorityQueueExample<Integer> intQueue = new
                                PriorityQueueExample<>();
intQueue.enqueue(5);
intQueue.enqueue(3);
intQueue.enqueue(8);
while(!intQueue.isEmpty())
System.out.println("Dequeued:"+intQueue.dequeue());
PriorityQueueExample<Double> doubleQueue =
                        new PriorityQueueExample<>();
```

```
double queue.enqueue(3.5);
double queue.enqueue(1.2);
double queue.enqueue(5.8);
while(!double queue.isEmpty())
system.out.println("Dequeued:"+ double queue.
                                dequeue());

priority Queue Example <String> String Queue=
            new Priority Queue Example <>();

String Queue.enqueue {"banana", "apple", "grape"};

while(!String Queue.is Empty())
system.out.println("Dequeued:"+ String Queue.dequeue());
}
}
```

- **Data and Results:** Output:-

Dequeued : 3
Dequeued : 5
Dequeued : 8
Dequeued : 1.2
Dequeued : 3.5
Dequeued : 5.8
Dequeued : apple
Dequeued : banana
Dequeued ; grape

- **Analysis and Inferences:**

→ The code is an example of a generic class that implements a priority queue data structure in Java along with tests using different datatypes (integers, doubles, strings).

Evaluator Remark (if Any):	Marks Secured: 48 out of 50
Work on Data structure concepts.	30/8/2023 Signature of the Evaluator with Date

Experiment#	10	Student ID	2200032060
Date		Student Name	A.Brahma Naidu

Experiment Title: Comparator and Comparable

Aim/Objective: To analyse the implementation of the concept of Comparator and Comparable interfaces for the real-time scenario.

Description:

The student will understand the concept of Comparator and Comparable Interfaces.

Pre-Requisites:

Knowledge: Classes and Objects in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1. Discuss the differences between Comparator and Comparable by filling the below mentioned table.

Sr. No	Comparable	Comparator
1	Purpose:- provides natural odering for objects.	Purpose:- Defines custom comparison logic for sorting.
2	Interface:- Implemented by the class being compared	Interface:- Implemented as a seperate class or anonymous class.
3	Single sorting:- only one natural sorting order.	Single sorting:- multiple comparators can be defined for varied sorting.
4	modification:- Requires modification of the object's class	modification:- no need to modify the object's classes

2. Write a JAVA program to sort the array in Linked List using Comparable.

```
import java.util.Arrays;
import java.util.LinkedList;
class Studentimplements Comparable <Student>{
string name;
int age;
student(string n, int a){
name=n;
age=a;
}
public int compareTo(student s){
return Integer.compare(age, s.age);
}
public string toString(){
return name+" ("+age+" years old)";
}
}
public class Main {
public static void main(String[] args){
LinkedList<Student> students= new LinkedList<>(Arrays.
                                                  asList(
new Student("Alice", 22),
newStudent("Bob", 20),
new student("Charlie", 25)
));
Students.sort(null);
for (student student: students){
system.out.println(student);
}
}
```

* output
Bob (20 years old)
Alice (22 years old)
Charlie (25 years old)

Aim:

- Consider a Movie class that has members like, rating, name, year. Suppose we wish to sort a list of Movies based on year of release. Implement the Comparable interface with the Movie class, and we override the method compareTo() of Comparable interface.

Procedure/Program:

```
import java.util.ArrayList;
import java.util.Collections;
class Movie implements comparable <Movie>{
private double rating;
private string name;
private int year;
public Movie (String name, double rating, int year){
 this.name = name;
 this.rating = rating;
 this.year = year;
}
public int compareTo (Movie other){
return Integer.compare (this.year, other.year);
}
public string to string ( ){
return name + "("+year+"), Rating:"+rating;
}
}

public class Main {
public static void main (string[] args){
ArrayList <Movie> movies = new ArrayList<>();
movies.add (new Movie ("movie A", 8.5, 2010));
movies.add (new Movie ("movie B", 7.8, 2005));
movies.add (new Movie ("movie C", 9.2, 2020));
```

```
system.out.println("Before sorting:");
for (Movie movie : movies){
system.out.println(movie);

}
collections.sort(movies);
system.out.println("\n After sorting:");
for (Movie movie : movies){
system.out.println(movie);

}
}
```

Experiment#
Date

- Data and Results:
- output :-

Before sorting:
Movie A (2010), Rating: 8.5
Movie B (2005), Rating: 7.8
Moviec (2020), Rating: 9.2
After sorting
Movie B (2005), Rating: 7.8
Movie A (2010), Rating: 8.5
Movie C (2020), Rating: 9.2

- Analysis and Inferences:

→ The code above implements the comparable interface in the 'movie' class to sort a list of movies based on their year of release.

Experiment#
Date

Sample VIVA-VOCE Questions (In-Lab):

1. List the usage of comparable Interface.

* uses of comparable interface.
→ simplifies sorting of objects in a class.
→ Establishes the default order for objects.
→ enables easy sorting using standard methods.

2. List the usage of comparator interface.

* uses of comparator interface:-
→ offers various ways to sort objects
→ Allows sorting of external or unmodifiable classes.
→ Enables dynamic and flexible sorting approaches.

Experiment #

Date

Lab:

1. Write a JAVA Program to compare movies by Rating by incorporating the below mentioned 3 points:
   a. Create a class that implements Comparator (and thus the compare () method that does the work previously done by compareTo()).
   b. Make an instance of the Comparator class.
   c. Call the overloaded sort() method, giving it both the list and the instance of the class that implements Comparator.

Sample Input:

8.4 Return of the Jedi 1983

8.8 Empire Strikes Back 1980

8.3 Force Awakens 2015

8.7 Star Wars 1977

Sample Output:

Sorted by rating

8.3 Force Awakens 2015

8.4 Return of the Jedi 1983

8.7 Star Wars 1977

8.8 Empire Strikes Back 1980

Sorted by name

Empire Strikes Back 8.8 1980

Force Awakens 8.3 2015

Return of the Jedi 8.4 1983

Star Wars 8.7 1977

Sorted by year

1977 8.7 Star Wars

1980 8.8 Empire Strikes Back

1983 8.4 Return of the Jedi

2015 8.3 Force Awakens

Procedure/Program:

```
import java.util.ArrayList;
import java.util.collections;
import java.util.comparator;
class Movie {
double rating;
string name;
int year;
public Movie (double r, string n, int y) {
 rating = r;
 name = n;
 year = y;
}
public string to string () {
return name + " " + rating + " " + year;
}
}

class Rating comparator implements comparator <Movie> {
public int compare (movie m1, movie m2) {
return Double.compare (m1.rating, m2.rating);
}
}

class Name comparator implements comparator <Movie> {
public int compare (Movie m1, Movie m2) {
return m1.name.compare To (m2.name);
}

class Year comparator implements comparator <Movie> {
public int compare (movie m1, Movie m2) {
return Integer.compare (m1.year, m2.year);
}
```

Student ID	
Student Name	

```
public class MovieComparatorExample{
public static void main (String[]args){
ArrayList <Movie> movies = new ArrayList <>();
movies.add (new movie (8.4, "Return of the Jedi", 1983));
movies.add (new movie (8.8, "Empire strikes Back", 1980));
movies.add (new movie (8.3, "Force Awakens", 2015));
movies.add (new movie (8.7, " Star wars", 1977));

System.out.Println ("Sorted by rating");
collections.sort (movies, new Rating comparator ());
for (Movie movie : movies){
 system.out.Println (movie);
}
system.out.Println ("\n sorted by name");
collections.sort (movies, new Name comparator ());
for (Movie movie : movies){
System.out.Println (movie);
}

System.out.Println("\nsorted by year");
collections.sort (movies, new year comparator ());
for (Movie movie : movies){
System.out.Println (movie.year+ " " + movie);
}
}
```

		Student ID	<TO BE FILLED BY STUDENT>
		Student Name	<TO BE FILLED BY STUDENT>

Experiment# Date:

- Data and Results: * output

Sorted by rating
Empire strikes Back 8.8, 1980
Star wars 8.7 1977
Return of Jedi 8.4 1983
Force Awakens 8.3 2015

sorted by name
Empire strikes Back 8.8.1980
Force Awakens 8.3 2015
Return of the Jedi 8.4 1983
Star wars 8.7 1977

sorted by year
1977 Star wars 8.7
1980 Empire strikes Back 8.8
1983 Return of the Jedi 8.4
2015 Force Awakens 8.3

- Analysis and Inferences:

→ The above Java program compares movies by rating name and year using the 'comparator' interface and demonstrates sorting the movies based on rating, name and year.

Evaluator Remark (if Any):	Marks Secured: 49 out of 50
good.	30/8/23. Signature of the Evaluator with Date

Experiment#	11	Student ID	2200032060
Date		Student Name	A.Brahma Naidu

Experiment Title: Implementation of Cloneable and Iterator Interfaces.

Aim/Objective:

To understand the concept and implementation of the Cloneable and Iterator interfaces.

Description:

The student will understand the Cloneable and Iterator interface.

Pre-Requisites:

Knowledge: Classes, Interfaces, Objects, inheritance and polymorphism in JAVA.

Tools: Eclipse IDE for Enterprise Java and Web Developers.

Pre-Lab:

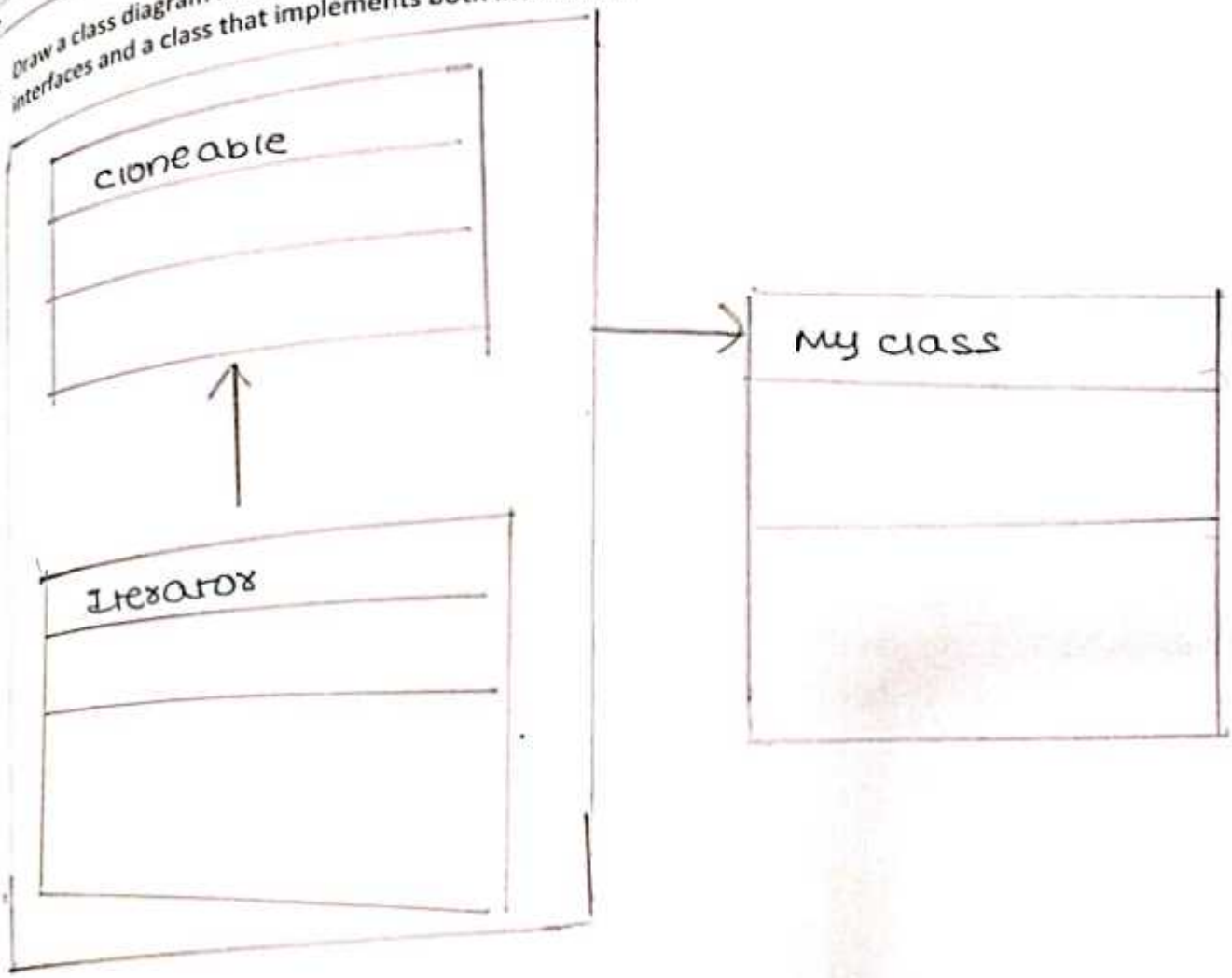
3. Explain the purpose and usage of the Cloneable interface in Java.

Cloneable

→ The cloneable interface in java indicates that an object's instance's can be cloned, or copied, using the 'clone()' method.

→ It doesn't have any methods itself, but serves as a marker for the Java runtime system to allow cloning.

4. Draw a class diagram illustrating the relationship between the Cloneable and Iterator interfaces and a class that implements both interfaces.



Lab: 8 **Write a Java Program Cloneable and Iterator Interfaces Scenario**

You are developing an Employee management system in Java. One of the requirements is to have a class called "EmployeeList" that stores a collection of employee objects. Each Employee has attributes such as Name, DOB, Mobile Number, and ID. Use the iterator interface to iterate over the list of employees and display their details.

Procedure/Program:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
class employee implements cloneable{
private string name, dob, mobileNumber;
private int id;
public employee (string name, string dob, string
                mobile Number, int id){
this.name = name;
this.dob = dob;
this.mobile Number = mobile Number;
this.id = id;
}
public object clone () throws cloneNotSupported
exception{
return super.clone();
}
Public string toString ()
return "employee{" + "name = '" + name + '\'' +
                ", dob + '\'' + ", mobile Number + '\'' +
                ", id = ", + id +
                '}';
}
```

```
class Employee List {
List <Employee> employees;
public Employee List () {
employees = new Array List <> ();

}
public void add employee (Employee employee){
employees.add (employee);

}
public void iterate employees () {
Iterator <Employee> iterator = employees.iterator();
while (iterator.hasNext()) {
Employee employee = (Employee) iterator.next().clone();
System.out.println (employee);

}
}
}

public class Main {
public static void main (String [] args){
EmployeeList.employeeList = new Employee List ();
employee, list.add Employee (new Employee ("John Doe",
        "1980-01-01", "1234567890", 1));

employee List.add Employee (new Employee ("Jane Doe",
        "1981-02-02", "9876543210", 2));

Employee List.iterate Employees();

}
}
```

Experiment No.
Date:

- **Data and Results:**

Output

Employee{name='John Doe', dob='1980-01-01',
mobileNumber='1234567890',id=1}

Employee{name='Jane Doe', dob='1981-02-02',
mobileNumber='9876543210',id=2}

		Student ID	
		Student Name	

VIVA-VOCE Questions (In-Lab):

6. **What is the purpose of the Cloneable interface?**

The cloneable interface is a marker in Java that indicates that an object can be cloned.
The purpose of cloneable interface is that it allows you to create a copy of an object

7. **How does the Cloneable interface enable object cloning in Java?**

The cloneable interface enables object cloning in Java by allowing classes to indicate that they can be cloned.
The clone() method is used to create a copy of an object

8. **Explain the role of the Iterator interface in Java collections.**

Java Iterator interface of java collections allows us to access elements of the collection and is used to iterate over the elements in the collection (Map, List, (or) Set etc..)

9. **What are the methods provided by the Iterator interface? Explain their significance.**

The iterator interface has the following methods.
(i) hasNext()
(ii) next()
(iii) remove()

* significance:-

(i) hasNext();- Returns true if the iterator has more elements to iterate over.

(ii) next(); Returns the next element in the iteration

(iii) remove(); Removes the element that was returned by the most recent call to the next() method.

Student ID	
Student Name	

Comments:

38. Can you explain the difference between the Cloneable interface and the Clone method in Java?

Cloneable interface:-

indicates that an object can be cloned

syntax:- interface cloneable

visibility:- public (access specifier)

Return value:- object

clone method:-

creates a copy of an object

syntax:- protected object clone() throws CloneNotSupported exception

visibility:- protected (access specifier)

Return value:- object

Experiment #
Date

Lab:

8. Create a class called "Student" that represents student information. This class has attributes such as name, roll number, and marks. Implement the Cloneable interface in the student class and write a code snippet to demonstrate the cloning of a Student object. Write code to demonstrate the cloning functionality.

- Procedure/Program:

```
class student implements cloneable{
private string name;
private int rollNumber;
private double marks;
public student (String name, int roll Number, doublemarks){
this.name = name;
this.roll Number = roll Number;
this.marks = marks;
}
public object clone() throws clone Not supported
Exception {
return super.clone();
}
public string to string ){
return "student{" + "name='" + name + '\'' +
                ", roll Number=" + roll Number +
                ", marks=" + marks +
                '}';
}
}
```

Student ID	
Student Name	

```
class Main {
    public static void main (String [] Args) {
        Student original student = new Student ("Alice", 101, 95.5);

        Student clone student = (Student) original student.clone();
        System.out.println ("original student: "+original student);
        System.out.println ("cloned student: "+ cloned student);
        cloned student.setName("Bob");
        cloned student.setRollNumber (102);
        cloned student.setMarks (88.0);
        System.out.println ("Modified cloned student: "+
                                cloned student);
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }
}
```

Experiment#		Student ID	
Date		Student Name	

- **Data and Results:**

Output:

Original student: student{name='Alice', rollNumber=101, marks=95.5}

Cloned student: student{name='Alice', rollNumber=101, marks=95.5}

Modified cloned student: student{name='Bob', rollNumber=102, marks=88.0}

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

		Student ID	
		Student Name	

Experiment Title: Implementation of Sets and Maps.

Learning Objective: To understand the concept and implementation of concepts of sets and maps.

Description:

The student will understand the concepts of sets and maps.

Prerequisites:

Knowledge: Classes and Objects, HashMap, TreeMap, HashSet and TreeSet in JAVA.

Tools: Eclipse IDE for Enterprise Java and Web Developers

Prelab:

3. Explain the concept of a set in Java. What is the main characteristic of a set? Provide an example code snippet demonstrating the usage of a set.

In Java, a set is a collection that represents a group of unique elements. The main characteristic of a set is that it does not allow duplicate values. Sets are part of the Java Collections Framework and are typically implemented by classes like 'HashSet', 'LinkedHashSet', and TreeSet.

Eg:-
```
import java.util.HashSet;
import java.util.Set;
public class SetExample {
public static void main (String[] args) {
set<String> stringSet = new HashSet<>();
stringSet.add ("apple");
stringSet.add ("Banana");
stringSet.add ("cherry");
stringSet.add ("Banana");
System.out.Println ("Elements in the set:"+
                    stringSet);
boolean containsBanana = stringSet.contains
                    ("Banana");
```

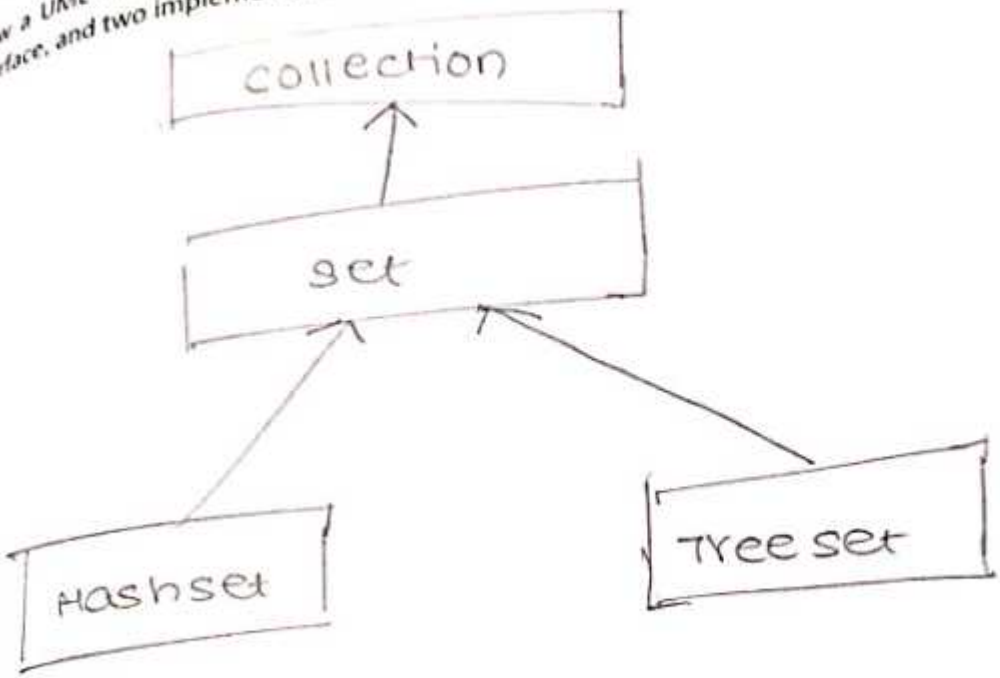
```
stringset.remove("cherry");
System.out.println("set after removing 'cherry':"+
                    stringset);
System.out.println("Iterating through the set:");
for (String fruit : stringset){
System.out.println(fruit);
    }
  }
}
```

* output:-

Elements in the set: [Banana, Apple, Cherry]
contains 'Banana': true
set after removing 'cherry': [Banana, Apple].
Iterating through the set:
Banana
Apple

Student ID	
Student Name	

Draw a UML diagram showing the relationship between the Set interface, the Collection interface, and two implementations of Set: HashSet and TreeSet.



Experiment#		Student ID	
Date		Student Name	

1. You are developing a contact management application. In this application, you need to store a collection of contacts, where each contact has a name and phone number. Implement a class called "ContactManager" that uses a map to store the contacts, with the name as the key and the phone number as the value. Write a code to demonstrate the implementation of the ContactManager class.

```
import java.util.HashMap;
import java.util.Map;
public class Contact Manager{
private Map<String,String> contacts;
public contact Manager(){
contacts= new HashMap<>();
}
public void add contact (String name, String phone
if (!contacts.contains key (name)){    Number){
contacts.put(name, phone Number);
system.out.println("Added contact: "+name+" " +
                        Phone Number);
}else{
system.out.Println("contact with name "+name+" "
                        already exists. use update contact
to change the phone number:");
}
}
public void update contact (String name, String new
                        Phone Number){
if (contacts.contains key (name)){
contacts.put (name, new phone Number);
system.out.Println("updated contact: "+name+" - "
                        +new Phone Number);
}else{
```

```
System.out.println("contact with name'"+name+"
                    does not exist. use addcontact to
                    create a new contact");
    }
}

Public void delete contact (String name){
if (contacts.containkey(name)){
contacts.remove(name);
System.out.println("Deleted contact:"+name);
}else{
system.out.println("contact with name'"+name+"
                    does not exist.");
    }
}

Public string get contact (String name){
if (contacts.containskey(name)){
return name +" - " + contacts.get(name);
}else{
return "contact with name'"+name+"'not found";
    }
}
}

Public class contactManager{
Public static void main (String [] args){
contactManager manager = new contactManager();
manager.addcontact("JohnDoe", "123-456-7890");
manager.addcontact("Alice smith", "908765-432");
system.out.println(manager.getContact("John Doe"));
manager.updatecontact("John Doe", "555-555-5555");
system.out.Println(manager.get contact("John Doe"));
```

```
manager.delete contact("Alice smith");
System.out.println(manager.get contact("Alice smith"));
```

* output:

Added contact: John Doe - 123-456-7890

Added contact: Alice smith - 908-765-4321

John Doe - 123-456-7890

updated contact: John Doe - 555-555-5555

Deleted contact: Alice smith

contact with name 'Alice smith' does not exist.

Experiment#
Date

: You are working on a program that stores student grades. Each student has a unique ID assigned to them. Design a data structure using sets and maps to efficiently store and retrieve student grades based on their ID. Write the code for adding a student's grade to the data structure given their ID and grade.

- **Procedure/Program:**

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
public class StudentGradeManager{
private Map<String,Double> gradeMap=new HashMap<>();
private Set<String>StudentIds= new HashSet<>();
public void addGrade(StudentId,double grade){
if(!StudentIds.contains(StudentId)){
gradeMap.put(StudentId, grade);
StudentIds.add(StudentId);
System.out.println("Added grade for student" + StudentId+":
                    + grade);
}else{
System.out.println("Student "+StudentId+"already exists:
  use updateGrade to change the grade");
}
}
Public void updateGrade(String StudentId,double new
                                    Grade){
if(StudentIds.contains(StudentId)){
gradeMap.put(StudentId,newGrade);
System.out.println("updated grade for student"+
                    StudentId+":"+newGrade);
}else{
System.out.println("Student"+ StudentId+"does not exist
                    cannot update grade");
}
```

```
public double getGrade(String studentId){
return gradeMap.getOrDefault(studentId,0.0);
}
public static void main(String[]args){
StudentGradeManager gradeManager= new StudentGradeManager();
gradeManager.addGrade("12345", 85.5);
gradeManager.addGrade("67890", 92.0);
gradeManager.updateGrade("12345", 90.0);
double studentGrade = gradeManager.getGrade("67890");
System.out.println("Grade for student 67890;"+
                studentGrade);
}
}
```

- **Data and Results:**

*output.

Added grade for student 12345 : 85.5
Added grade for student 67890 : 92.0
updated grade for student 12345 : 90.0
Grade for student 67890 : 92.0

- **Analysis and Inferences:**

The provided Java code defines a 'StudentGradeManager' class for managing student grades. The code initializes a map to store ('gradeMap') and a set to store student ID's ('StudentIds'). It provides methods for adding, updating and retreiving student grades as well as a 'main' method to demonstrate its usage.

Experiment#
Date

Sample VIVA-VOCE Questions (In-Lab):

6. Explain the difference between a set and a map in programming.

* set:-
→ A set is a collection that stores a group of unique elements with no associated values.

* Map:-
→ A Map is a collection that stores key-value pairs, where each key is associated with a specific value.

7. What is the difference between a Set and a HashSet?

* set:-
→ A set is an interface in Java that defines a collection of unique elements.

* Hashset:-
→ Hash set is a specific implementation of the set interface in Java. It uses a hash table for storage and does not guarantee the order of elements.

8. What is the difference between a Map and a HashMap?

* Map:-
→ Map is an interface in Java that defines a collection of key-value pairs.

* HashMap:-
→ HashMap is a specific implementation of the Map interface in Java. It uses a hash table for storage and allows fast retrieval of values by their keys.

Assignment#		Student ID	
Date		Student Name	

How does a map differ from an array or a list?

* Map
→ A map is a collection that stores key-value pairs, allowing data to be associated with specific keys
* Array/List:
→ An array/list is a collection that stores elements by their indices, without key value associations.

10. What is the purpose of the key-value pairs in a map? Can the same key have multiple values?

* Purpose of key-value pairs in a map:-
→ Key-value pairs in a map provide a way to associate data (the value) with a unique identifier (the key). Keys are used efficiently to retreive the data.
→ In many map implementations, the same key cannot have multiple values; each key is associated with a single value.

Lab:

1. Given a set of integers, write a code snippet to find the maximum value in the set.

```
import java.util.HashSet;
import java.util.Set;
import java.util.Collections;
public class MaxValueInSet {
public static void main (String[] args){
Set<Integer> integerSet = new HashSet<>();
integerSet.add (5)
integerSet.add (12)
integerSet.add (8)
integerSet.add (20)
integerSet.add (3);
int maxValue= Collections.max (integerSet);
System.out.println ("maximum value in the set:"
                    + maxValue);
}
}
```

* output:

Maximum value in the set: 20

	Student ID	
	Student Name	

Experiment#

2. Write a Java program that uses a Set to store a list of fruits. The program should then iterate over the Set and print out the fruits in alphabetical order.

Procedure/Program:

```
import java.util.set;
import java.util.Treeset;
public class fruitslist{
public static void main(string []args){
set <string> fruitsset= new Treeset<>();
fruitsset.add("Banana");
fruitsset.add("Apple");
fruitsset.add("orange");
fruitsset.add("Grapes");
fruitsset.add("cherry");
system.out.Println("fruits in alphabetical order:");
for (string fruit: fruitsset){
system.out.Println(fruit);
}
}
}
```

Experiment#

- **Data and Results:**

* output
Fruits in alphabetical order;
Apple
Banana
cherry
Grapes
orange

- **Analysis and Inferences:**

The fruits are printed in alphabetical order because we used a 'Tree set', which automatically sorts the elements as they are added to the set.

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Aim: Implement the...

Description:

The student will understand the concepts of Binary Search Trees (BST) and AVL Trees, which are binary tree data structures used for efficient searching and retrieval of data.

Pre-Requisites:

Knowledge: Classes, Objects, Understanding of Binary Search and AVL Tree.

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1. Write a Java program to insert the number k into a BST.

Sample Input and Output:

```
        100                                    100
       /   \          Insert 40               /   \
     20     500     --------->             20     500
    / \                                    / \
   10  30                                 10  30
                                                 \
                                                  40
```

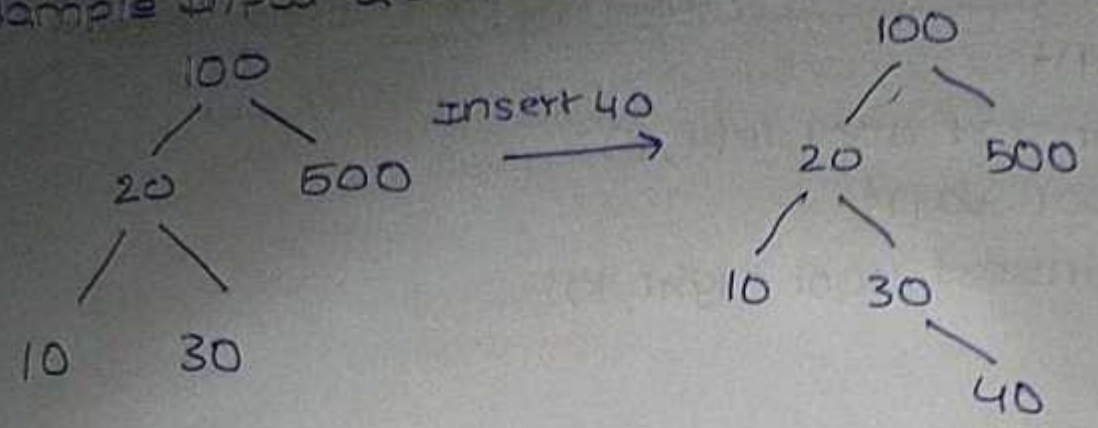
```
class TreeNode{
int val;
TreeNode left;
TreeNode right;
Public TreeNode (int val){
this.val = val;
this.left = null;
this.right = null;
}
}

Public class BSTInsertion{
Public static TreeNode insert (TreeNode root, int k){
if (root = = null){
return new TreeNode (k);
}
```


Scanned by Scanner Go

```
if (k<root.val){
root.left = insert (root.left, k);
}else if (k >root. val){
root.right = insert (root.right, k);

}
return root;

}
public static void inorder Traversal (TreeNode root){
if(root ! = null){
inorder Traversal (root.left);
system.out.Println (root.val+" ");
inorder Traversal (root.right);

}
}
public static void main (String [ ]args){
TreeNode root = null;
int [] numbers = {5,3,7,2,4,6, 8};
root = insert (root, number);
}
System.out.Println ("Inorder Traversal of the BST:")
inorderTraversal (root);
}
}
```


Scanned by Scanner Go

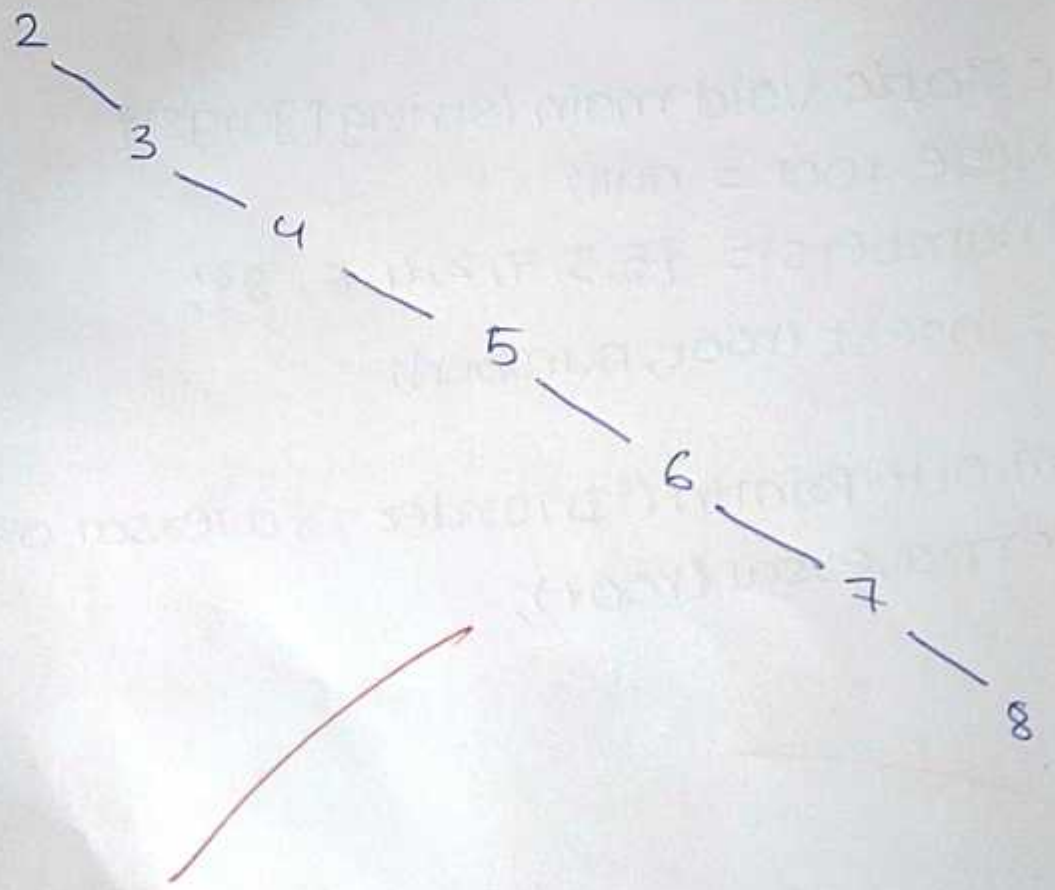
* sample input & output:



* output of above code:-

Inorder traversal of BST

2 3 4 5 6 7 8



Scanned by Scanner Go

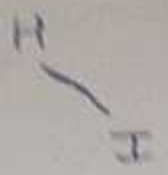
* construction of AVL tree having the elements
H, I, J, B, A, E, C, F, D, G, K, L

step 1 :- Insert H

H

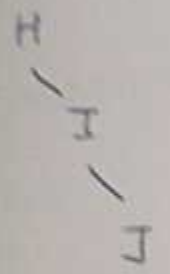
since it is the 1st element it becomes the root

step 2 :- Insert I



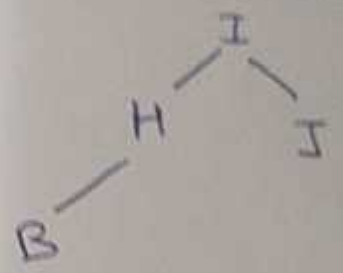
No rotation is needed because the balance factor of H is 0 and I is inserted to the right

step 3 :- Insert J



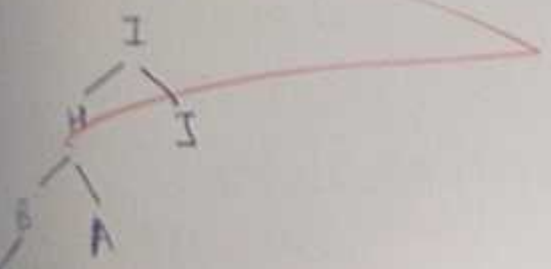
No rotation is needed because the balance factor of H & I is 0. J is inserted to right

step 4 :- Insert B



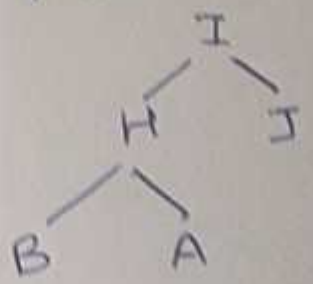
A left rotation is needed because the balance factor of I is 2. The tree is rebalanced by rotating I to left

step 5 :- Insert A



A right rotation is needed because the balance factor of H is 2. The tree is rebalanced by rotating H to right

step 6 :- Insert E



Scanned by Scanner Go

1. Write a program ... for the following operations:
   - Insert an element (no duplicates are allowed),
   - Delete an existing element,
   - Traverse the AVL (in-order, pre-order, and post-order)

INPUT:
Line 1 contains an integer NQ, the number of queries.
The next NQ lines contain queries and are of the form 'i xx' (insert xx into an AVL tree) or 'd xx' (Delete xx from an AVL tree).

OUTPUT:
For each query, print value of an unbalanced node (if any) at which rotation is being applied. The last three lines print 'Preorder traversal', 'Inorder traversal', and 'Postorder traversal' of an AVL tree that results after the execution of all NQ queries.

SAMPLE INPUT:

```
8
i1
i2
i3
i4
i5
i6
d4
d5
```

SAMPLE OUTPUT:

```
1
3
2
6
2163
1236
1362
```

```
class TreeNode{
int val, height;
TreeNode left;
TreeNode right;
Public TreeNode (int val){
this.val = val;
this.height = 1;
this.left = null;
this.right = null;
}
```


Scanned by Scanner Go

```
root = temp;
}
}else{
 root = temp; }
}else{
TreeNode temp = minvalueNode(root.right);
root.val = temp.val;
root.right = delete(root.right, temp.val);
}
}
if(root==null){
return root; }
root.height = 1 + Math.max(height(root.left), height(root.right));
int balanceFactor = getBalanceFactor(root);
if(balanceFactor > 1 && getBalanceFactor(root.left)>=0){
return rotateRight(root);
}
if(balanceFactor < -1 && getBalanceFactor(root.right)<=0){
return rotateLeft(root);
}
if(balanceFactor < -1 && getBalanceFactor(root.right)<=0){
return rotateLeft(root);
}
if(balanceFactor < -1 && getBalanceFactor(root.right)>0){
 root.right = rotateRight(root.right);
 return rotateLeft(root);
 } return root;
}
Public static void preorderTraversal(TreeNode root){
 if(root != null){
System.out.println(root.val + " ");
PreOrderTraversal(root.left);
PreOrderTraversal(root.right);
}}
Public static void inorderTraversal(TreeNode root){
 if(root != null){
 inorderTraversal(root.left);
```

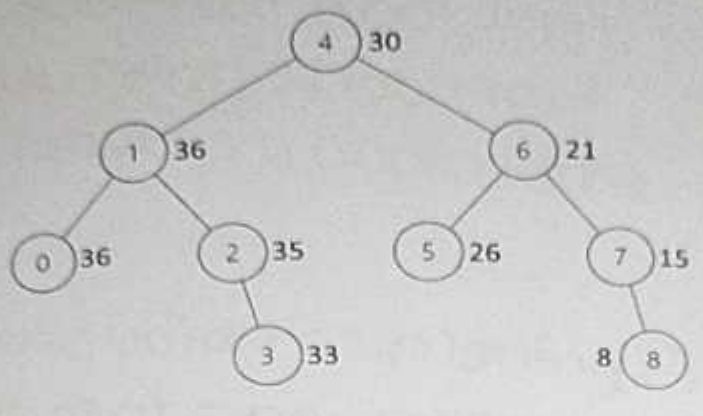
Scanned by Scanner Go

```
System.out.Println (root.val + " ");
inorderTraversal (root.right);

}
}
public static void postorderTraversal (TreeNode root){
if(root != null){
postorderTraversal (root.left);
postorderTraversal (root.right);
system.out.Println (root.val + " ");

}
}
public static void main (String[] args){
root = null;
String [] queries = {"i 1", "i 2", "i 3", "i 4", "i 5", "i 6",
                     "d 4", "d 5" };

for (String query : queries){
String[] parts = query.split (" ");
char operation = parts [0].charAt (0);
int value = Integer.parseInt (parts [1]);
if (operation == 'i'){
System.out.Println (insert (root, value).val);
}else if (operation == 'd'){
System.out.Println (delete (root, value).val);
}
}

System.out.Println ("preorder traversal:");
PreOrderTraversal (root);
System.out.Println ("\n inorder traversal:");
inorderTraversal (root);
system.out.Println ("\n post order traversal:");
post order traversal (root); } }
```


Scanned by Scanner Go

2. Given the root of a Binary Search Tree (BST), convert it to a Greater Tree such that every key of the original BST is changed to the original key plus the sum of all keys greater than the original key in BST.
As a reminder, a binary search tree is a tree that satisfies these constraints:
The left subtree of a node contains only nodes with keys less than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key.
Both the left and right subtrees must also be binary search trees.

Example :



Input: root = [4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
Output: [30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]

- **Procedure/Program:**

```
class TreeNode{
int val;
TreeNode left;
TreeNode right;
Public TreeNode (int val){
this.val = val;
this.left = left; null;
this.right = right; null;
}
}

Public class ConvertBST (TreeNode root){
if(root = = null){
return null;
}
```


Scanned by Scanner Go

```
convertBST(root.right);
sum += root.val;
root.val = sum;
convertBST(root.left);
return root;
}
public static void inorderTraversal(TreeNode root){
if(root != null){
inorderTraversal(root.left);
System.out.println(root.val + " ");
inorderTraversal(root.right);
}
}
public static void main(String[] args){
TreeNode root = new TreeNode(4);
root.left = new TreeNode(1);
root.right = new TreeNode(6);
root.left.left = new TreeNode(0);
root.left.right = new TreeNode(2);
root.right.left = new TreeNode(5);
root.right.right = new TreeNode(7);
root.left.left.right = new TreeNode(3);
root.right.right.right = new TreeNode(8);
ConvertBSTTOGreaterTree converter =
                    new ConvertBSTTOGreaterTree();
TreeNode convertedRoot = converter.convertBST(root);
System.out.println("Inorder traversal of converted BST");
inorderTraversal(convertedRoot);
}
```


Scanned by Scanner Go

- **Data and Results:**

*output:-
Inorder traversal of the converted BST:
36 36 33 32 30 26 21 15 8

- **Analysis and Inferences:**

The above program will convert the given BST into a Greater Tree and then perform an inorder traversal of the converted tree to display the result.

Scanned by Scanner Go

Student Name

VIVA-VOCE Questions (In-Lab):

6. **How does an AVL Tree differ from a regular Binary Search Tree (BST)?**

AVL Tree is a self balancing binary search tree whereas Regular BST is not self balancing.

7. **What is the time complexity of searching in a Binary Search Tree (BST)?**

worst case time complexity :- O(n).

Average case time complexity :- O(logn)

Scanned by Scanner Go

Experiment#
Date

8. What is the worst-case time complexity for searching an element in a balanced AVL Tree?

worst case time complexity for searching in a balanced AVL Tree is O(log n)

9. Compare the performance of BST and AVL Trees in terms of time complexity for various operations.

operation	BST	AVL Tree
search	O(log n)	O(log n)
Insertion	O(log n)	O(log n)
Deletion	O(log n)	O(log n)
worst case	O(n)	O(log n)

Scanned by Scanner Go

10. What is the main advantage of using an AVL Tree over a regular Binary Search Tree?

* main advantage of using an AVL Tree over a regular BST is :-

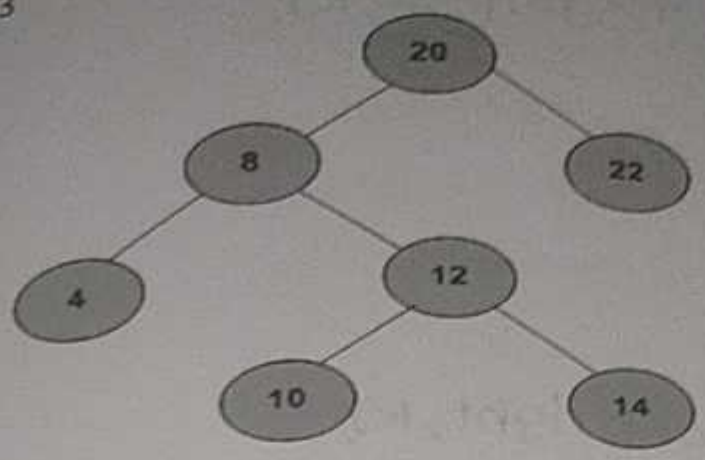
→ AVL tree's maintain balance, guaranteeing efficient logarithmic operations, whereas regular BST's can become unbalanced, leading to inefficient linear operations.

Scanned by Scanner Go

Post-Lab:

1. Write a Java program to find the K-th smallest element in BST.

Sample input: K=3



Sample Output:

10

- Programs

```
class TreeNode{
int val;
TreeNode left;
TreeNode right;
public TreeNode (int val){
this.val = val;
this.left = null;
this.right = null;
}
}
Public class KthSmallestInBST{
Private int count;
Private int result;
Public int KthSmallest (TreeNode root, int K){
count = 0, result = 0;
inorderTraversal (root, K);
return result;
}
```


Scanned by Scanner Go

```
private void inorderTraversal (node,
 if (node == null || count >= k){
  return;
 }
 inorderTraversal (node.left, k);
 count++;
 if (count == k){
 result = node.val;
 return;
 }
 inorderTraversal (node.right, k);
}
public static void main (String[] args){
TreeNode root = new TreeNode (5);
 root.left = new TreeNode (3);
 root.right = new TreeNode (8);
 root.left.left = new TreeNode (2);
  root.left.right = new TreeNode (4);
  root.right.left = new TreeNode (6);
  root.right.right = new TreeNode (10);
  int k=3;
  kth smallestInBST finder = new kth smallestinBST();
  int kth smallest element = finder.kth smallest(root, k);
  System.out.println("The "+k+"-th smallest element
                 in the BST is: "+kth smallest element);
 }
}
```

Scanned by Scanner Go

- **Data and Results:**

* Output :

The 3rd smallest element in the BST is : 4

Scanned by Scanner Go

Experiment
Date

- Analysis and Inferences:

In this program, the 'kthsmallest' method initiates the inordertraversal and updates the result when the k-th smallest element is found.

2. List out real-world applications of AVL trees.

* Real world applications of AVL trees:-

(i) Memory Management
(ii) Database query optimization
(iii) Network Routing algorithms
(iv) Text Editors.
(v) Compiler syntax trees etc....

Scanned by Scanner Go

Experiment#	14	Student ID	2200032060
Date		Student Name	A.Brahmanaidu

Experiment Title: Implementation of BFS and DFS Algorithms

Aim/Objective: To understand the concepts of Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms and implement them in a Java program.

Description:

The student will understand the concepts of Breadth-First Search (BFS) and Depth-First Search (DFS), which are graph traversal algorithms used to explore or search through a graph data structure.

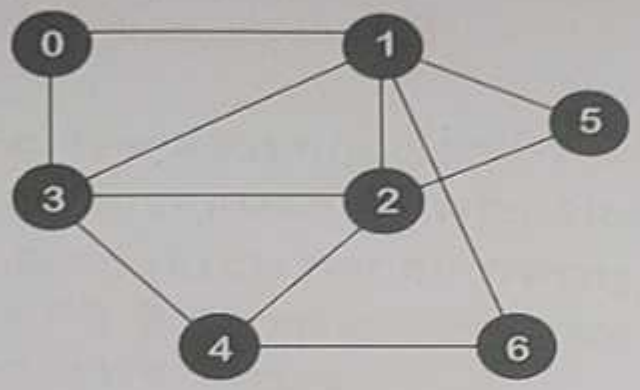
Pre-Requisites:

Knowledge: Classes, Objects, Understanding of graph data structure Breadth-First Search (BFS) and Depth-First Search (DFS) in Java.

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

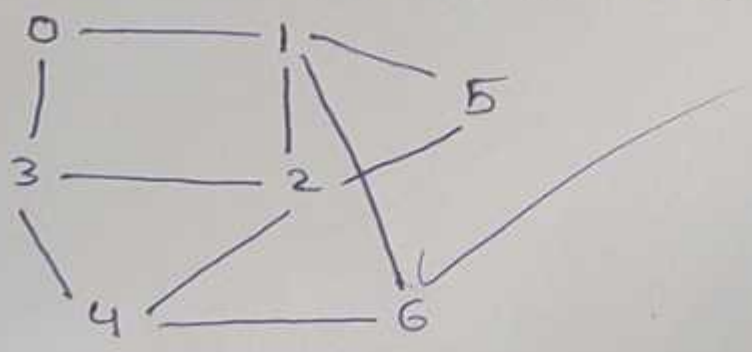
4. List down the list of traversed vertices starting from '0' by BFS and DFS methods. What data structures are used in BFS and DFS for traversal?



- **Program**

• BFS (Breadth First search):

→ BFS starts at the root node and explores all its neighbors at the current depth before moving on to the nodes at the next depth.



Scanned by Scanner Go

• Queue:-

0	1	3	2	5	6	4

∴ After exploring and moving elements in the queue. Pop the final elements in the queue as the result.

∴ The result is

0 - 1 - 3 - 2 - 5 - 6 - 4

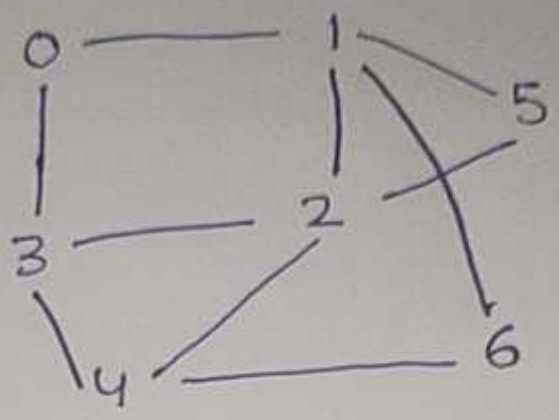
Scanned by Scanner Go

- DFS (Depth first search):-

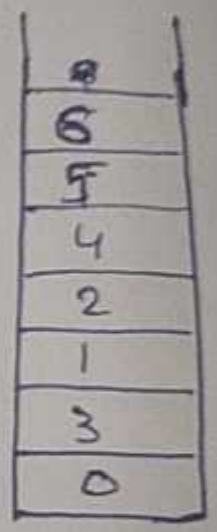
→ DFS has different traversal methods, such as Pre-order, inorder and post order.

→ It follows Back tracking

→ It uses stack to store the elements.



- stack:- Check the root node ('0' in this case) and push it into stack and also push the elements passing from the root node into the stack



Now do the back tracking for the elements in the stack and pop them out a result.

∴ The result is:-

6-5-1-4-2-1-3-0

Scanned by Scanner Go

Experiment#
Date

- Data and Results

* output for BFS :-

0-1-3-2-5-6-4

* output for DFS :-

6-5-4-2-1-3-0

- Analysis and Inferences

* BFS (Breadth first search) :-

→ BFS starts at the root node and explores all its neighbors at the current depth before moving onto the nodes at the next depth.

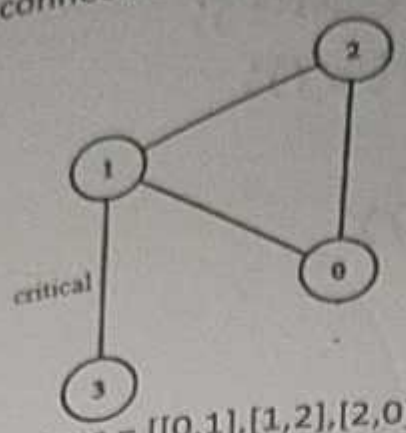
* DFS :-

→ DFS has different traversal methods such as pre-order in-order and post-order. It follows Backtracking.

Scanned by Scanner Go

5. There are n servers numbered from 0 to n - 1 connected by undirected server-to-server connections forming a network where connections[i] = [ai, bi] represents a connection between servers ai and bi. Any server can reach other servers directly or indirectly through the network.
A critical connection is a connection that, if removed, will make some servers unable to reach some other server.
Return all critical connections in the network in any order.

Example 1:



Input: n = 4, connections = [[0,1],[1,2],[2,0],[1,3]]
Output: [[1,3]]
Explanation: [[3,1]] is also accepted.

Example 2:
Input: n = 2, connections = [[0,1]]
Output: [[0,1]]

- Program

```
import java.util.*;
public class criticalconnections{
int time =0;
public List <List<Integer>> criticalconnections (int n, List<List
                                <Integer>> connections){

List<Integer>[] graph = new ArrayList[n];
for(int i=0; i<n; i++){
graph[i]= new ArrayList<>(); }
for(List<Integer> connection: connections){
int u= connection.get(0); int v = connection.get(1);
graph[u].add(v); graph[v].add(u); }
int[] discovery = new int[n]; int[] low= new int[n];
List<List<Integer>> result = new ArrayList<>();
dfs(0,-1,discovery, low, graph, result); return result;
}
```


Scanned by Scanner Go

Experiment#
Date

```
List<List<Integer>> result2 = solution.criticalConnections
                                (n2, connections2);

System.out.println("critical connections for n2:"
                    + result2);

}
}
```

- Data and Results

Course Title

Scanned by Scanner Go

Experiment
Date

* sample input and output
• for 'n1 = 4' and 'connections1':
critical connections for n1: [[1,3]]
• for 'n2 = 2' and 'connections 2':
critical connections for n2: [[0,1]]

• Analysis and Inferences

The above code calculates and prints the critical connections for two different network configurations.

Scanned by Scanner Go

- **Procedure/Program:**

```
import java.util.LinkedList;
import java.util.Queue;
public class shortestsafepath{
public static int shortestsafepath(int[][] matrix){
if(matrix==null||matrix.length==0||matrix[0].length==0)
return-1;
int m= matrix.length; int n= matrix[0].length;
boolean[][]visited = new boolean [m][n];
int[]dx={-1,0,1,0}; int dy[]={0,-1,0,1};
Queue<int[]>queue = new LinkedList<>();
for(int i=0; i<m; i++){
if(matrix[i][0]==1){
queue.offer(new int[]{i,0});
visited[i][0]= true; }}
int steps=0;
while(!queue.isempty()){
int size = queue.size();
steps++;
for(int i=0; i<size; i++){
int[]curr = queue.poll();
int x= curr[0]; int y= curr[1];
for(int j=0; j<4; j++){
int newx = x+dx[j]; int newy= y+dy[j];
if(newx>=0&&newx<m&&newy>=0&&newy<n&&
  !visited[newx][newy]&&matrix[newx][newy]==1){
if(newy==n-1){
return steps;
}
```


Scanned by Scanner Go

```
queue.offer(new int[]{newx, newy});
visited[newx][newy] = true;
}
}
}
}
return-1;
}
public static void main(String[]args){
int[][] matrix={
{1,1,1,1,0,1},
{1,0,1,1,1,1},
{1,1,0,1,1,1},
{1,1,1,1,1,1}
};
int shortestpathLength= shortestsafepath(matrix);
if(shortest pathLength!=-1){
system.out.println("shortest safe path length:"
+shortest pathLength);
}else{
system.out.println("NO safe path found");
}
}
}
```


Scanned by Scanner Go

- **Data and Results:**

* sample Input and Output:.

Input matrix :-

1	1	1	1	0	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	1	1	1	1

output :-

shortest safe path length : 11

- **Analysis and Inferences:**

This program uses BFS to traverse the matrix from the first column to the last column, keeping track of the shortest path while avoiding sensors (cells with value 0). It returns the length of the shortest safe path or -1 if no ~~path~~ safe path is found.

Scanned by Scanner Go

VIVA-VOCE Questions (In-Lab):

What is the difference between Breadth-First Search (BFS) and Depth-First Search (DFS)?

• Difference between BFS and DFS:

→ BFS explores all neighbours of a node before moving to the next level

→ DFS explores as far as possible along a branch before backtracking.

What is the time complexity of BFS and DFS algorithms? Explain.

• Time complexity of BFS and DFS:

→ BFS has O(V+E) time complexity, where V is the no: of vertices and E is the no: of edges.

→ DFS has O(V+E) time complexity as of BFS, where V is no: of vertices and E is no: of edges.

Scanned by Scanner Go

How does BFS differ from DFS in terms of traversal pattern?

- Difference in traversal pattern:-

→ BFS explores nodes level by level, moving horizontally through neighbours.

→ DFS explores nodes deeply along a branch before backtracking.

9. How can the depth-first search algorithm be modified to compute topological ordering of a directed acyclic graph?

- Modifying DFS for topological ordering:-

→ use DFS with a stack to push nodes after visiting all descendants.

→ The order of nodes in the stack is a topological ordering.

Scanned by Scanner Go

Department#

Student Name

10. How does BFS differ from DFS in terms of traversal pattern?

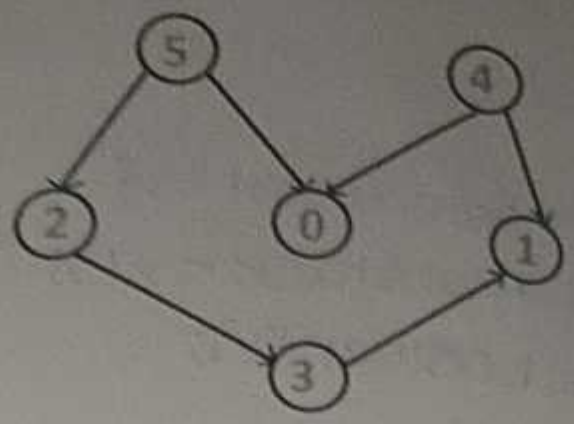
• Traversal pattern Difference:-

→ BFS explores nodes level by level.

→ DFS explores nodes deeply along a branch before backtracking.

Scanned by Scanner Go

1. Write ... using Topological sorting [Hint: apply DFS concept]
The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no incoming edges).



Output:
542310 or 452310

- Program

```
import java.util.*;
class Graph{
private int V;
private List <List<Integer>> adj;
public Graph (int vertices){
 v= vertices;
 adj= new ArrayList <>(vertices);
 for(int i=0; i<vertices; i++){
 adj.add (new LinkedList <>());
 }
}
public void add Edge (int u, int v){
adj.get(u).add (v);
}
private void topological sort util (int v, boolean[] visited,
                    Stack <Integer> stack){
```


Scanned by Scanner Go

```
visited [v] = true;
for (Integer neighbor : adj.get(v)){
if (!visited[neighbor]){
topologicalsortutil (neighbor, visited, stack);
 }
}
stack.push (v);
}
Public List <Integer> topological sort (){
stack <Integer> stack = new stack <>();
boolean [] visited = new boolean [V];
for (int i = 0; i<v; i++){
if (!visited [i]){
 topologicalsortutil (i, visited, stack);
 }
}
 List <Integer> result = new Array List <>();
 while (!stack.isempty ()){
 result.add (stack. pop());
 }
 return result;
 }
}
 Public class Main{
 Public static void main (string[] args){
 int numvertices = 0;
 Graph graph = new Graph (num vertices);
  graph.addEdge (5,2);
  graph. add Edge (5,0);
  graph.add Edge (4,0);
  graph.add edge (4,1);
```

Scanned by Scanner Go

```
graph.addEdge(2,3);
graph.addEdge(3,1);
List<Integer> topologicalOrder = graph.topologicalSort
for(Integer vertex : topologicalOrder){
System.out.println(vertex+ " ");

}
}
}
```

- Data and Results

* Output:-

5 4 2 3 1 0

Scanned by Scanner Go

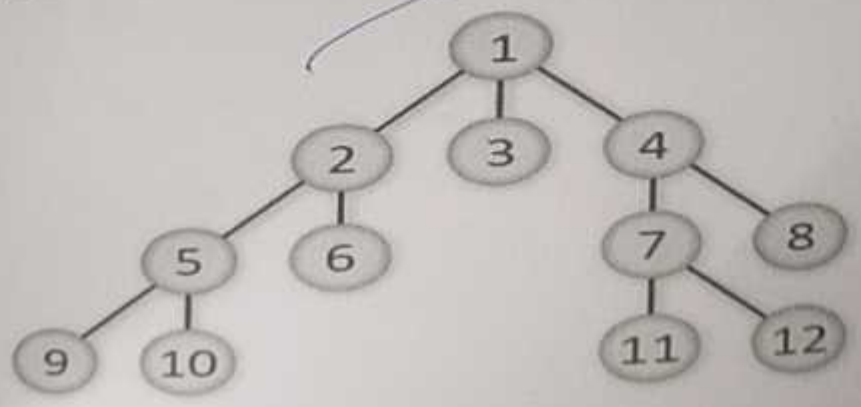
- Analysis and Inferences

→ This program creates a directed graph with six vertices and edges, and then it performs a topological sort.

→ The output will be one of the valid topological orders such as "5 4 2 3 1 0" or "4 5 2 3 1 0".

→ The specific order may vary depending on graph's structure.

2. Write the visited vertices list by BFS and DFS methods for the following tree.

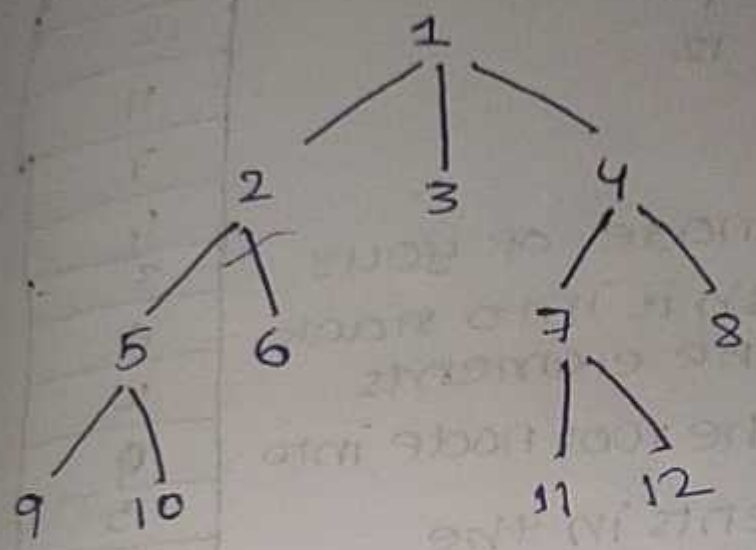


Scanned by Scanner Go

Experiment No.
Date

Procedure/Program:

- BFS (Breadth First Search):-
- BFS starts at the root node and explores all its neighbors at the current depth before moving on to the nodes at the next depth.
- Here in the above given tree let the root node be 1. Now let's explore all the nodes passing from 1.



- Queue:-

1	2	3	4	5	6	7	8	9	10	11	12

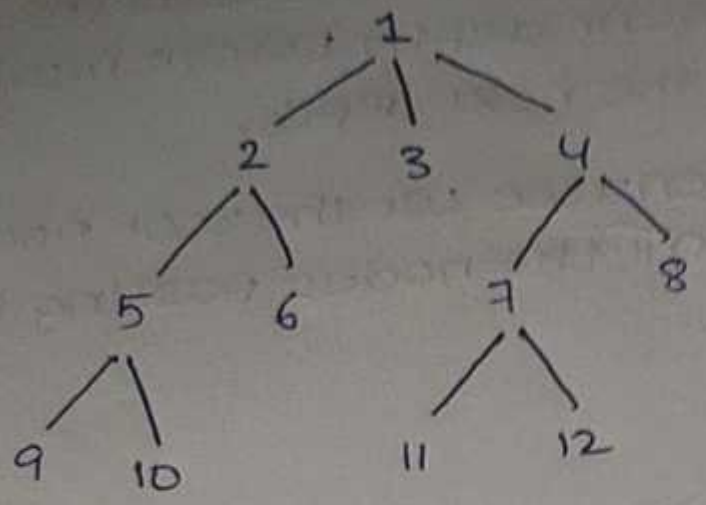
After performing BFS the elements in the queue must be popped out as a result.

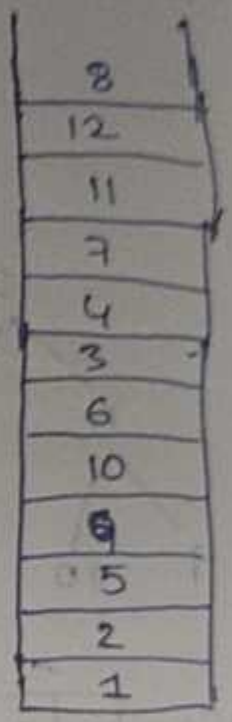
∴ the result is

1-2-3-4-5-6-7-8-9-10-11-12.

Scanned by Scanner Go

* DFS (Depth First Search)
→ DFS has different traversal methods such as Pre order, Inorder and Post order.
→ DFS follows back tracking.
→ The elements must be moved into stack.





* Stack:- select the root node of your choice and push it into stack and also push the elements passing from the root node into the stack.

Now check the elements in the stack. if ~~all~~ the elements related to the top element are present in stack means pop the element as result.

∴ The result is

8-12-11-7-4-3-6-10-9-5-2-1

Scanned by Scanner Go

- Data and Results:

* OUTPUT FOR BFS :-

1-2-3-4-5-6-7-8-9-10-11-12

* OUTPUT FOR DFS :-

8-12-11-7-4-3-6-10-9-5-2-1

Scanned by Scanner Go

Analysis and Inferences:

* BFS (Breadth First Search):-

→ BFS starts at the root node and explores all its neighbors at the current depth before moving on to nodes at the next depth.

→ It uses queue to store the elements.

* DFS (Depth First Search):-

→ DFS has different traversal methods such as pre order, inorder and post order.

→ DFS follows backtracking, and uses stack to store the elements.

Evaluator Remark (if Any):	Marks Secured: 49 out of 50
21/9/23	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.

Scanned by Scanner Go

...ng/Objective: To un... ...g in Java and explore the basics of concurrent programming.

...scription:

The student will understand the concept of threading in Java, which allows for concurrent execution of multiple tasks within a program. We will learn about the basics of threads, thread creation, thread synchronization, and thread communication.

...-Requisites:

Knowledge: Classes and Objects, Understanding of threads and their execution model in JAVA

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1. Explain the concept of threading in Java. What is a thread, and how does it enable concurrent execution? Describe the different ways to create a thread in Java, including extending the Thread class and implementing the Runnable interface.

Program

* Threading In Java.
* Thread in Java:-

→ A thread is the smallest unit of Java program's execution.

→ It enables concurrent execution, allowing multiple tasks to run simultaneously within the same program.

→ threads share the same memory space, enabling them to work together and share data.

* creating threads:-

(i) Extending thread class :-

→ Create a new class that extends the built in thread class.

→ Override the 'run()' method in your class to define the thread's behaviour.

→ Instantiate and start the thread using 'start()' method.

Scanned by Scanner Go

```
class MyThread extends Thread {
public void run() {

}
}
MyThread thread = new MyThread();
thread.start();
```

Implementing the 'Runnable' interface.

```
class MyRunnable implements Runnable {
public void run() {

}
}
MyRunnable runnable = new MyRunnable();
Thread thread = new Thread(runnable);
thread.start();
```


Scanned by Scanner Go

Data and Results

→ A thread in Java is the smallest unit of execution in Java.

→ It enables concurrent execution, running multiple tasks simultaneously.

Analysis and Inferences

4 ways to create the Thread are :-

(i) extending the thread class.

(ii) Implementing the runnable interface.

Scanned by Scanner Go

The first line of each test case contains a single integer N. The second line contains N space-separated integers A1,A2,...,ANA1,A2,...,AN.

Output:

For each test case, print a single line containing the string "RUN THREAD" if it is possible to choose required indices or "STOP THREAD" otherwise.

Sample Input:

```
4
4
1 1 2 3
4
2 1 3 3
5
5 4 4 3 1
5
3 2 1 1 4
```

Sample Output:

```
RUN THREAD
STOP THREAD
STOP THREAD
RUN THREAD
```

- **Program**

```
import java.util.*;
public class Main{
public static void main(string[]args){
scanner scanner = new scanner (System.in);
int t= scanner.nextInt();
while(t-->0){
int n = scanner.nextInt ();
int[]arr = new int [n];
for(int i=0; i<n; i++){
arr[i]= scanner.nextInt();
}
if(is possible (arr)){
```


Scanned by Scanner Go

```
system.out.p...
}else{
system.out.Println("STOPTHREAD");
}
}
}
public static boolean ispossible (int [] arr){
HashMap<Integer, Integer> hashMap = new HashMap<>();
for (int i=0; i < arr.length; i++){
int element= arr[i];
if (hashMap.containskey (element)){
int prevIndex = hashMap.get(element);
if (i- prevIndex >1){
return True;
}
}
hashMap.Put(element, i);
}
return False;
}
}
```

• Data and Results

Scanned by Scanner Go

Experiment#
Date
Student Name

* Sample Input

4
4
1 1 2 3
4
2 1 2 3
5
5 4 4 3 1
5
3 2 1 1 4

* Sample Output:

RUN THREAD
STOP THREAD
STOP THREAD
RUN THREAD.

- **Analysis and Inferences**

This program reads the input, processes each case and checks if it's possible to choose indices i and j such that Ai ≠ Aj but A[Ai

Scanned by Scanner Go

Experiment
Date

Lab:

You have the five functions:

- printTwo that prints the word " divisible by 2" to the console,
- printThree that prints the word " divisible by 3" to the console,
- printFour that prints the word " divisible by 4" to the console,
- printFive that prints the word "divisible by 5" and
- printNumber that prints a given integer in other conditions.

Implement all these functions by calling multiple threads to print the numbers from 1 to 15.

- Program

```
public class MultiThread Printing {
public static void main (String[] args){
for (int i= 1; i<=15; i++){
Thread thread = new Thread ( () -> {
if (i % 2 == 0){
printTwo();
}
if (i % 3 == 0){
Print Three();
}
if (i % 4 == 0){
Print Four ();
}
if (i % 5 == 0){
Print five();
}
if (i % 2 != 0 && i % 3 != 0 && i % 4 != 0 && i % 5 != 0){
Print Number (i);
}
});
thread.start ();
try{
Thread.join();
```


Scanned by Scanner Go

```
}catch(InterruptedEx
e.printStackTrace(); }}
}
Public static void printTwo(){
system.out.println("divisible by 2");
}public static void printThree(){
system.out.println("divisible by 3");
}public static void printFour(){
system.out.println("divisible by 4");
}public static void printFive(){
system.out.println("divisible by 5");
}publicstatic void printNumber(int num){
system.out.println(num);
}
}
```

Scanned by Scanner Go

- **Data and Results**

divisible by 2 — 1
divisible by 3 — 3
divisible by 2 — 7
divisible by 4 — 6
divisible by 5 — 9
divisible by 2 — 11
divisible by 3 — 13
divisible by 2 — 8
divisible by 4 — 10
divisible by 5 — 12
divisible by 2 — 14
divisible by 3 — 5
divisible by 2
divisible by 4
divisible by 5

- **Analysis and Inferences**

→ In this program we loop through the numbers from 1 to 15 and create a seperate thread for each Number.

→ Each thread checks if the number is divisible by 2, 3, 4 (or) 5 and calls the corresponding function to Print the message.

Scanned by Scanner Go

You have a fu... ...rameter and prints it to the console.

For example, calling printNumber(7) prints 7 to the console.

You are given an instance of the class ZeroEvenOdd that has three functions: zero, even, and odd. The same instance of ZeroEvenOdd will be passed to three different threads:

Thread A: calls zero() that should only output 0's.

Thread B: calls even() that should only output even numbers. Thread C: calls odd() that should only output odd numbers.

Modify the given class to output the series "010203040506..." where the length of the series must be 2n.

Implement the ZeroEvenOdd class:

ZeroEvenOdd(int n) Initializes the object with the number n that represents the numbers that should be printed.

void zero(printNumber) Calls printNumber to output one zero.

void even(printNumber) Calls printNumber to output one even number. void odd(printNumber) Calls printNumber to output one odd number.

Example 1:

Input: n = 2 Output: "0102"

Explanation: There are three threads being fired asynchronously. One of them calls zero(), the other calls even(), and the last one calls odd(). "0102" is the correct output.

Example 2: Input: n = 5

Output: "0102030405"

- **Procedure/Program:**

```
import java.util.*;
class zeroEvenOdd{
Private int n;
Private semaphore zerosemaphore, evensemaphore,
odd semaphore;
Public zeroEvenOdd (int n){
this.n = n; this.zerosemaphore = new semaphore(1);
this.odd semaphore = new semaphore(0);
this.even semaphore = new semaphore(0);
}
Public void zero (int consumer printNumber) throws
interrupted exception {
```


Scanned by Scanner Go

```
for (int i=1; i<=n; i++){
zero semaphore.acquire(); print Number.accept(0);
(i%2==0 ? evensemaphore : odd semaphore).release();
}
}
Public void even (Int consumer, print Number) throws
Interrupted exception{
print (2, print Number, evensemaphore);
}
Public void odd (Int consumer print Number) throws
Interrupted exception {
print (1, print Number, odd semaphore);
}
Private void print (int start, Int consumer PrintNumber,
  semaphore semaphore) throws Interrupted Exception{
 for (int i= start; i<=n; i+=2){
  semaphore.acquire();
  print Number.accept (i);
  zero semaphore.release();
 }
}
}

Public class Main{
Public static void main (String [] args){
 int n=5;
 zero Even odd zero Even odd = new zero Even odd(n);
 Start Thread (() → zero Even odd.zero (system.out.Println));
 Start Thread (() → zero Even odd.even (system.out.Println));
 start Thread (() → zero Even odd.odd (system.out.Println));
}
```

Scanned by Scanner Go

Experi...
Date

```
private static void startThread(Runnable runnable){
Thread thread = new Thread(() -> {
try{
runnable.run();
}catch (InterruptedException e){
e.printStackTrace();

}
});
thread.start();
try{
thread.join();
}catch (InterruptedException e){
e.printStackTrace();

}
}
}
```

Scanned by Scanner Go

- Data and Results:

* Sample Input and output

Input : n = 2

Output : "0102".

- Analysis and Inferences:

'IntConsumer' is a functional interface used for printing integers and the 'semaphore' objects are used to synchronize the threads.

The 'zero()' 'odd()' 'even()' methods control the printing of zero's, odd and even numbers.

Scanned by Scanner Go

Student Name

VIVA-VOCE Questions (In-Lab):

What is a thread? How does it enable concurrent execution in Java?

thread in Java:-

A thread is a smallest unit execution in Java.

It enables concurrent execution by allowing multiple tasks to run simultaneously.

Describe the different ways to create a thread in Java.

creating threads in Java:-

extending the 'thread' class

Implementing the runnable Interface.

Scanned by Scanner Go

What is thread safety? How do you achieve thread safety in Java?

Thread safety:-

→ ensuring that data accessed by multiple threads doesn't get corrupted.

→ Acheived using synchronization mechanisms.

9. Explain the terms "synchronization" and "thread safety" in the context of threading.

synchronization and Thread safety:-

→ controlling access to shared resources in a multithreaded environment is called synchronization.

→ ensuring that operations on shared data are ~~fro~~ safe from race condition is known as Thread safety.

Scanned by Scanner Go

Experiment#

Date

Student Name

10. How can you prevent race conditions in multithreaded programs?

* preventing Race conditions:.

→ using synchronization mechanisms.

→ Avoid shared mutable data when possible.

→ use thread safe data structures.

Post-Lab:

1. Write a JAVA program which will generate the threads:

-To display 10 terms of Fibonacci series.

-To display 1 to 10 in reverse order.

Sample output:

```
Enter the Limit for fabonacci: 10
==================================
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
==================================

Reverse is:
==================================
10  9  8  7  6  5  4  3  2  1
==================================
```

Scanned by Scanner Go

Experiment No.
Date

Program:

```
public class Thread example {
public static void main (String[] args) {
start Thread (() -> {
int a = 0, b = 1;
system.out.Println ("Fibonacci series:");
for (int i = 0; i<10; i++) {
system.out.Println (a+" ");
a=b;
b=temp+b;
}
system.out.Println ();
});
start Thread (() -> {
system.out.Println ("Numbers in Reverse order:");
for (int i = 10; i>=1; i--) {
system.out.Println (i+" ");
}
system.out.Println ();
});
}
Private static void start Thread (Runnable runnable) {
Thread thread = new Thread (runnable);
thread.start ();
}
}
```


Scanned by Scanner Go

Experiment#		Student Name	
Date			

- **Data and Results:**

* output :

enter the limit for fibonacci series :- 10

fibonacci series : 0 1 1 2 3 5 8 13 21 34

numbers in reverse order : 10 9 8 7 6 5 4 3 2 1 0

- **Analysis and Inferences:**

This program creates two threads one for generating the fibonacci series and other for printing the numbers in reverse order. Both threads are started and joined to ensure they complete before the program exits.

Evaluator Remark (if Any):	Marks Secured: 49 out of 50
27/9/25	Signature of the Evaluator with Date

Scanned by Scanner Go

Experiment

Aim/Objective: To understand the Producer-Consumer pattern and implement it in a multi-threaded Java program.

Description:

The student will understand the concepts of the Producer-Consumer pattern, which is used to solve the problem of synchronizing access to a shared resource between multiple threads.

Pre-Requisites:

Knowledge: Classes, Objects, Understanding of multi-threading and synchronization in Java.

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

4. Explain the concept of the Producer-Consumer pattern in thread synchronization. What problem does it solve, and how does it work? Provide an example code demonstrating the implementation of the Producer-Consumer pattern in Java.

```
import java.util.*;
class ProducerConsumer{
private Queue<Integer> buffer= new LinkedList<>();
private int capacity =5;
Public void produce (int item) throws Interrupted
                                        Exception{
 synchronized (this){
 while (buffer.size() == capacity){
  wait ();
 }
 buffer.add (item);
 System.out.Println ("Produced: "+item);
 notify ();
 }
}

Public void consume() throws InterruptedException{
synchronized (this){
while (buffer.is Empty()) {
 wait ();
}
```


Scanned by Scanner Go

```
int item = buffer.poll();
System.out.println("consumed: " + item);
notify();
}
}
}
public class Main {
public static void main (String[] args){
Final ProducerConsumer PC = new ProducerConsumer();
Thread producer Thread = new Thread (() -> {
try{
for (int i=0; i<10; i++){
PC.produce (i);
Thread.sleep (1000);
}
}catch (InterruptedException e){
e.printStackTrace();
}
});
Thread consumer Thread = new Thread (() -> {
try{
for (int i=0; i<10; i++){
PC.consume ();
Thread.sleep (1500);
}
}catch (InterruptedException e){
e.printStackTrace();
}
});
producerThread.start();
consumerThread.start();
}
}
```

Scanned by Scanner Go

Experiment#		Student ID	
Date		Student Name	

The producer-consumer pattern in thread synchronization solves the problem of coordinating multiple threads where some produce items and others consume them from a shared buffer. It ensures producers don't

5. Write a code demonstrating the usage of the synchronized keyword in Java. where one increments a shared counter variable and the other decrements it. Ensure proper synchronization to avoid data corruption.

Scanned by Scanner Go

Experiment#	16	Student ID	2200032060
Date		Student Name	A. Brahma Naidu

Experiment Title: Implementation of Producer-Consumer Pattern

Aim/Objective: To understand the Producer-Consumer pattern and implement it in a multi-threaded Java program.

Description:

The student will understand the concepts of the Producer-Consumer pattern, which is used to solve the problem of synchronizing access to a shared resource between multiple threads.

Pre-Requisites:

Knowledge: Classes, Objects, Understanding of multi-threading and synchronization in Java.

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

4. Explain the concept of the Producer-Consumer pattern in thread synchronization. What problem does it solve, and how does it work? Provide an example code demonstrating the implementation of the Producer-Consumer pattern in Java.

```
import java.util.*;
class ProducerConsumer{
Private Queue<Integer> buffer= new LinkedList<>();
Private int capacity = 5;
Public void produce (int item) throws Interrupted
                                        exception{
 synchronized (this){
 while (buffer.size() == capacity){
 wait ();
 }
 buffer.add (item);
 System.out.Println ("Produced: "+item);
 notify ();
 }
}

Public void consume()throws InterruptedException{
synchronized (this){
while (buffer.is empty()) {
 wait ();
 }
```

```
int item = buffer.poll();
system.out.println("consumed: " + item);
notify();
}
}
}

public class Main {
public static void main (String[] args){
final ProducerConsumer pc = new ProducerConsumer();
Thread producerThread = new Thread(() -> {
try{
for (int i=0; i<10; i++){
pc.produce(i);
Thread.sleep(1000);
}
}catch (InterruptedException e){
e.printStackTrace();
}
});
Thread consumerThread = new Thread(() -> {
try{
for (int i=0; i<10; i++){
pc.consume();
Thread.sleep(1500);
}
}catch (InterruptedException e){
e.printStackTrace();
}
});
producerThread.start();
consumerThread.start();
}
}
```

The producer-consumer pattern in thread synchronization solves the problem of coordinating multiple threads where some produce items and others consume them from a shared buffer. It ensures producers don't

5. Write a code demonstrating the usage of the synchronized keyword in Java. where one increments a shared counter variable and the other decrements it. Ensure proper synchronization to avoid data corruption.

```
class counter{
Private int count = 0;
Public synchronized void increment(){
count++;
}
Pubic synchronized void decrement(){
count--;
}
Public synchronized int getcount(){
return count;
}
}

Public class Main{
Public static void main(String[]args){
counter counter = new counter();
Thread increment Thread = new Thread(()->{
for(int i=0; i<10000; i++){
```

```
counter.increment();
}
});

Thread decrementThread = new Thread(() -> {
for (int i=0; i<10000; i++){
counter.decrement();
}
});

incrementThread.start();
decrementThread.start();
try {
incrementThread.join();
decrementThread.join();
}catch (InterruptedException e){
e.printStackTrace();
}
System.out.println("Final counter value:"
                    + counter.getCount());
}
}
```

*Output:-

Final counter value: 0

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

You are developing a messaging application that simulates the exchange of messages between users. Implement a program that includes two classes: "Producer" and "Consumer." The Producer class represents a user who generates messages, and the Consumer class represents a user who consumes messages. Write a code to demonstrate the implementation of the Producer-Consumer pattern in the messaging application.

- **Procedure/Program:**

```
import java.util.*;
class Message{
Private string content;
Public Message (string content){
 this.content = content;
}

Public string getcontent(){
 return content;
}
}

class Producer implements Runnable{
Private BlockingQueue<Message> queue;
Private string name;
Public Producer (BlockingQueue<Message> queue,
                 String name){
 this.queue = queue; this.name = name;
}

Public void run(){
For (int i= 1; i<=5; i++){
String message content = "Message"+ i + "From"+name;
Message message = new Message (message content);
try{
 queue.put (message);
```

```
system.out.println(name + "produced:" + messagecontent);
 Thread.sleep(1000);
} catch (InterruptedException e){
   e.printstackTrace();
}
}
}

class consumer implements Runnable {
 Private BlockingQueue <Message> queue;
 Private String name;
 Public consumer (BlockingQueue <Message> queue,
                  String name){
  this.queue = queue; this.name = name;
 }
 Public void run (){
  while (true){
  try{
   Message message = queue.take();
   system.out.Println(name + "consumed" + message.getcontext());
   Thread.sleep(2000);
  } catch (InterruptedExceptione){
   e.Print stackTrace();
}
}
}
}

Public class MessagingApp{
Public static void main(String [] args){
BlockingQueue <Message> queue = new ArrayBlockingQueue<>
                                (10);
```

```
Producer Producer1 = new producer(queue, "User A");
Producer Producer2 = new producer (queue, "User B");
consumer consumer1 = new consumer(queue, "User X");
consumer consumer2 = new consumer(queue, "User Y");

Thread producerThread1 = new Thread (producer1);
Thread producer Thread2 = new Thread (Producer 2);
Thread consumer Thread1 = new Thread (consumer1);
Thread consumer Thread2 = new Thread (consumer2);

ProducerThread1.start();
ProducerThread2.start();
consumerThread1.start();
consumerThread2.start();
}
}
```

- **Data and Results:**

* Output:-

User A Produced: Message 1 From user A
User B Produced: Message 1 From user B
User x consumed: Message 1 From user A
User y consumed: Message 1 From user B
User A Produced: Message 2 From user A
User B Produced: Message 2 From user B
User x consumed: Message 2 from user A
User y consumed: Message 2 From user B

- **Analysis and Inferences:**

To implement the producer-consumer pattern we will use Blocking queue interface.
The Blocking queue provides a thread-safe way to manage a queue of messages that the producer adds to, and the consumer removes and processes.

Sample VIVA-VOCE Questions (In-Lab):

6. **What is the main challenge in implementing the producer-consumer problem?**

The main challenge in implementing the producer-consumer problem is co-ordinating multiple threads to avoid issues.

7. **How does the wait() and notify() methods facilitate inter-thread communication in the Producer-Consumer pattern?**

* wait () method:-

→ waits for the conditions. until they met.

* a certain condition.

* notify () method:-

→ notify other waiting threads when the condition changes.

8. **How can you ensure mutual exclusion between the producers and consumers while accessing the shared buffer?**

* By using synchronization mechanisms like 'synchronized' keyword or locks we can ensure the mutual excusion between the producers and consumers while accessing the shared buffer.

9. **How does the Producer-Consumer pattern ensure thread safety and synchronization?**

The Producer-consumer patternensure thread safety and synchronization by using synchronization mechanisms to control access to the shared buffer.

10. How does the synchronized keyword ensure that only one thread can access a shared resource at a time?

The 'synchronized' keyword ensures that only one thread can access a shared resource at a time by acquiring and releasing locks.

Post-Lab:

Write a code to implement a bounded buffer using the concepts learned in the experiment. Ensure that the buffer has a maximum capacity of 10 items, and the producer and consumer threads operate correctly while avoiding race conditions.

Procedure/Program:

```
import java.util.*;
class BoundedBuffer<T>{
 Private Final Queue<T> buffer; Private int capacity;
 Public Bounded Buffer (int capacity){
  this.capacity = capacity; this.buffer = new LinkedList<>();
 }

 Public synchronized void produce (T item) throws
  Interrupted Exception{
  while (buffer.size() == capacity){
   wait();
  }
  buffer.add(item);
  System.out.Println("Produced:" + item);
```

```
notifyAll();
}
public synchronized T consume () throws InterruptedException{
 while (buffer.isEmpty ()){
  wait ();
 }
 T item = buffer.poll ();
  System.out.println ("consumed: "+ item);
  notifyAll();
  return item;
 }
}
class Producer implements Runnable{
 Private BoundedBuffer <Integer> buffer;
 Public Producer (BoundedBuffer <Integer> buffer){
 this.buffer= buffer;
 }
 Public void run(){
 try{
  for(int i= 1; i<=20; i++){
   buffer.produce (i);
   Thread.sleep(100);
  }
 }catch (InterruptedException e){
   e.printStackTrace();
  }
 }
}
class Consumer implements Runnable{
 Private BoundedBuffer <Integer>buffer;
 Public Consumer (BoundedBuffer <Integer> buffer){
  this.buffer= buffer;
 }
```

```
Pubiic void run(){
try{
for(int i=0;i<20;i++){
Integer item= buffer.consume();
Thread.sleep (200);
}
}catch (InterruptedException e){
e.PrintStackTrace();
}
}
}

Public class BoundedBufferExample{
Public static void main (String [] args){
BoundedBuffer <Integer> buffer= newBounded
                                BUffer<>(10);
Thread ProducerThread.start();
       ConsumerThread.start();
}
}
```

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:**

Output:-

Produced: 1
consumed: 1
Produced: 2
consumed: 2
- - - - - -
- - - - -
Produced: 20
consumed: 20

- **Analysis and Inferences:**

In this program, we will create a 'Bounded Buffer' class with methods for adding items (producing) and removing items (consuming) while ensuring thread safety.

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date

Experiment#	17	Student ID	2200032060
Date		Student Name	A.Brahma Naidu

Experiment Title: Implementation of Servlets.

Aim/Objective: To understand the concept of servlets in Java and implement them in a web application.

Description:

The student will understand the concepts of servlets, which are Java classes used to extend the functionality of web servers and handle client requests.

Pre-Requisites:

Knowledge: Classes, Objects, Java Servlet API, Understanding of HTTP protocol and web development concepts.

Tools: Eclipse IDE for Enterprise Java and Web Developers, Apache Tomcat - Web Server, Java Servlet APIs.

Pre-Lab:

4. Explanation of what a servlet is and how it differs from other Java classes. What are the main advantages of using servlets in web development?

A servelet in Java class for processing web requests and generating dynamic web content. It differs from other Java classes by its web-specific roles.

* Advantages of using servlets in web development:

→ platform independence

→ Robustness

→ Reusability etc...

5. Write a servlet application code to print the current date and time.

```
import java.io.*;
import javax.servlet.annotation.webservlet;
import javax.servlet.http.*;

Public class currentDateTimeservlet extends
                                    HttpServlet{

 Protected void doGet(HttpServlet Request request,
                                    HttpServletResponse response)

 throws servlet exception,IO exception{
 response.setcontentType("text/html");
 Print writer out= response.get writer();
 Out.Println("<html><body><h2>current Date
            and Time: </h2><p>"+new java.util.Date()
            + "</p></body></html>");
 }
}
```

* output :-

```
<html>
<body>
    <h2> current Date and time: </h2>
    <p>Thu Oct 20 16:23:15 UTC 2023 </P>
 </body>
</html>
```

In-Lab:

You are developing a voter eligibility checker web application. The application should accept the user's age and name through a form submission and display a message indicating whether the user is eligible for voting or not. In this scenario, implement a servlet called "VoterEligibilityServlet" that handles the form submission and displays the eligibility message based on the user's age. If the age is 18 or above, display a message like "Hello [name], you are eligible to vote!" Otherwise, display a message like "Hello [name], you are not eligible to vote yet." Ensure proper validation and error handling for invalid input.

- **Procedure/Program:**

```
* voter eligibility servlet
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class voter eligibility servlet extends
                                HttP servlet{

protected void dopost(HttP servlet Request req,
                HttP servlet Response resp) throws
                servlet exception, IoExcepHon {
```

```
string name = req.getParameter("name");
int age = Integer.parseInt(req.getParameter("age"));
resp.getWriter().write("Hello"+ name +", you are "+
        (age>=18?"eligible":"not eligible")+
        "to vote.");
}
}
```

* HTML Form :-

```
<!DOCTYPE html>
<html>
<head>
  <title>Voter Eligibility Checker</title>
</head>
<body>
<form action="/your-context-path/VoterEligibility
        Servlet" method="post">
  <label for="name">Name:</label><input type
        ="text" id="name" name="name"
        required><br>
  <label for="age">Age:</label><input type=
        "number" id="age" name="age"
        required><br>
  <input type="submit" value="Check Eligibility">
</form>
</body>
</html>
```

- **Data and Results:**

* sample output

Name :- John

Age :- 25

Output :- "Hello John, you are eligible to vote.".

- **Analysis and Inferences:**

* The above code is used to create a voter eligibility checker web application using a Java servlet is a multi step process.

(i) create a Java servlet

(ii) override 'do post' method

(iii) create a form in HTML

Sample VIVA-VOCE Questions (In-Lab):

5. **What is a servlet? How does it relate to web development?**

A servlet is a Java technology for building web applications.
It releates to web development by processing HTTP requests and generating responses.

6. **How does a servlet handle client requests and generate responses?**

servlet handle client requests by extending the Http servlet class, overriding methods like doPost or doGet, and sending responses via Http servlet Response.

7. **How can servlets handle user input data submitted through an HTML form?**

servlets can handle user input by using methods like
'request.getParameter("parameterName")
to retrieve data submitted through an HTML form.

8. **What is the role of a servlet in a web application?**

The role of a servlet in a web app is to handle dynamic content, process user requests and generate HTML or other responses.

9. What are the steps involved in configuring and deploying a servlet in a web application.

* steps in configuring and deploying a servlet:

→ create a Java class that extends HttpServlet

→ Configure servlet mapping in web.xml

→ Deploy the servlet to a servlet container, like Tomcat.

→ Access the servlet through a URL in a web browser.

Post-Lab:

1. **Write a code demonstrating the usage of request and response objects in a servlet. Use a scenario where you need to retrieve user input from a form and display a customized response based on that input the entered data on the web page.**

Procedure/Program:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class simpleservlet extends HttpServlet{
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
throws ServletException, IOException{
 response.setContentType("text/html");
 PrintWriter out = response.getWriter();
 String name = request.getParameter("name");
 out.println("<html><body>");
 out.println("<h1>Hello,"+name+"!</h1>");
 out.println("<p> Thank you for using
                 our form.</p>");
 out.println("</body></html>");
}
}
```

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:**

* sample output

```
<!DOCTYPE html>
<html>
<head>
    <title> Response </title>
</head>
<body>
    <h1> Hello, John! </h1>
    <p> Thank you for using our form. </p>
</body>
</html>
```

- **Analysis and Inferences:**

* In this code the servlet retrieve's the user's name from a form and displays a customized response with the user's name on the web page.

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date

Experiment#	18	Student ID	2200032060
Date		Student Name	A.Brahma Naidu

Experiment Title: Implementation of Servlet with JDBC.

Aim/Objective: To understand how to integrate a servlet with JDBC (Java Database Connectivity) in order to perform database operations within a web application.

Description:

The student will understand the concepts of servlets with JDBC, which allows us to connect to a database, execute SQL queries, and perform database operations from within a web application.

Pre-Requisites:

Knowledge: Classes, Objects, Understanding of servlets, JDBC API, database connectivity, SQL queries, and basic web development concepts.

Tools: Eclipse IDE for Enterprise Java and Web Developers, Apache Tomcat - Web Server, Java Servlet API.

Pre-Lab:

6. Explain the concept of JDBC and its role in Java database connectivity. What are the main steps involved in performing database operations using JDBC?

→ JDBC is a Java API that allows Java applications to interact with databases.

→ It provides a standard interface for connecting to relational databases and executing SQL queries.

→ The main steps in using JDBC for database operations are :-

(i) Load the JDBC driver
(ii) Establish a database connection
(iii) Create a statement
(iv) Execute SQL queries
(v) Process query results
(vi) Close resources.

7. Write a code demonstrating the steps to establish a database connection using JDBC. Write a java program to accept the details of Teacher (TId, Name, Address) from the user and insert it into the MySQL database, specifying the necessary connection parameters such as URL, username, and password.

```
import java.sql.*;
Public class TeacherDatabaseInsert{
 Public static void main(String[]args){
String url = "jdbc:mysql://localhost:3306/
                                your database";

String username= "your username";
Stringpassword = "your password";
try{
 class.ForName ("com.mysql.cj.jdbc.Driver");
 connection connection = DriverManager.get
        connection(url, username, password);
 Scanner scanner = newScanner (system.in);
 System.out.Println ("enter TeacherID, Name and
                                Address: ");
 int teacherId = scanner.nextInt();
 String teacherName= scanner.next();
 String teacherAddress = scanner.next();
 String insertQuery = "INSERT INTO teachers
                    (TId, Name, Address) VALUES(?,?,?)";
 Prepared statement prepared statement =
                connection.prepare statement(insertQuery);
 Prepared statement.setInt (1, teacherId);
```

```
prepared statement.setstring(2, teacherName);
prepared statement.setstring(3, teacher Address);
int rowsAffected = prepared statement.executeUpdate();
system.out.println((rowsAffected >0)?"Teacher details
        inserted successfully.":"Failed to insert teacher
                                details.");
prepared statement.close();
connection.close();
}catch (ClassNotFoundException | SQLException e){
  e.printstackTrace();
  }
}
}
```

* sample output:
  Teacher details inserted successfully.

In-Lab:

1. You are developing a student information management system. Implement a servlet called "StudentDetailsServlet" that accepts student details (student ID, student name, address, program) through a form submission. The servlet should store these details in a database using JDBC integration. CRUD operations on Student Database sequence of steps to follow:

(a) Create a Student Database using JDBC program

(b) Create a Registration Table Inside Student Database with the fields

((i)id (ii)name(iii)address(iv)program and make id as a primary key) using JDBC Program

(c) Insert 4 records in Registration Table of Student Database using JDBC Program

(d) Display the records inserted in Registration Table using JDBC Program

(e) Update the age of student as 30 whose id values are as 100 and 101 using JDBC Program.

(f) Delete a Student Record whose id=101 from Registration Table using JDBC Program.

- **Procedure/Program:**

```
import java.sql.*;
public class student management system{
 Public static void main (String[]args){
  String jdbcurl = "jdbc:mysql://localhost:3306/student_db";
  String username="your_username";
  String password = "your_Password";
  try(connection connection = Driver Manager.getconnection(
                                  jdbcurl, username, Password)){
 //a)
  statement createDbstatement = connection.Create statement
                                                    ();
 Create Db statement.executeUpdate("CREATE DATABASE
                 IF NOT EXISTS student_db");
  Create Db statement.close();
 //b)
 statement create Tablestatement = connection.create
                                        statement();
  create Table statement.execute update ("CREATE TABLE IF
    NOT EXISTS Registration ("+
    "id INT PRIMARY KEY, "+"program VARCHAR (255)
    "name VARCHAR (255), "+ "address VARCHAR (255))");
  create Table statement.close();
```

//c)

```
statement insert statement = connection.create statement();
insert statement.execute update("INSERT INTO Registration
    (id, name, address, Program)"+"VALUES(100, 'John',
    '123 Main st', 'computer science')");
```

//d)

```
statement display statement = connection.create statement();
Resultset resultset = display statement.execute Query("
SELECT * FROM Registration");
while (resultset.next()) {
 int id = resultset.getInt("id");
 String name = resultset.getstring("name");
 String address = resultset.getstring("address");
 String program = resultset.get string("program");
 system.out.Println("ID:"+id+", Name:"+name+",
    Address:"+address+", program:"+program); }
```

//e)

```
statement updatestatement = connection.create
                                    statement();
update statement.execute update("UPDATE Registration
    SET age = 30 WHERE id IN(100, 101)");
```

//f)

```
statement delete statement = connection.create
                                    statement();
delete statement.execute update("DELETE FROM
                Registration WHERE id = 101");
}catch (SQLexception e){
 e.PrintStackTrace();
 }
}
}
```

- **Data and Results:**

* sample output (Display Records):

ID: 100,

Name: John,

Address: 123 Main st,

Program: computerscience

- **Analysis and Inferences:**

The above code covers the basic operations you described, but in a real world application, you should handle exceptions, resource management, and input validation.

Sample VIVA-VOCE Questions (In-Lab):

6. **What is the role of JDBC in Java database connectivity?**

* Role of JDBC in Java database connectivity.

→ JDBC facilitates Java programs to connect and interact with databases.

7. **What are the different types of JDBC drivers? Explain their differences and advantages.**

* different types of JDBC drivers :

(i) Type-1 (JDBC-ODBC)

(ii) Type-2 (Native-API),

(iii) Type-3 (Network Protocol)

(iv) Type-4 (Thin Driver)

8. **How do you establish a database connection using JDBC?**

we can establish a JDBC connection with DriverManager.getconnection (Url, username, password).

9. **Explain the difference between Statement and PreparedStatement in JDBC.**

→ The basic difference between statement and Prepared statement in JDBC is statement is simple, & prepared statement is precompiled for efficiency.

10. How do you handle exceptions related to database operations in JDBC? What are some common JDBC-related exceptions?

* By using try-catch blocks we can handle exceptions related to database operations in JDBC.
* Some common JDBC related exceptions
  → Class Not Found Exception
  → Data Truncation etc...

Post-Lab:

3. . Write a JDBC program to Connect to a Database in MySqlServer.

Hint: Create a "test" database in PostgreSQL server and check whether it is connected or not through JDBC Program
(Display a message "Connected to the database" if username and password matched, otherwise display a message "Invalid username or password".)

Procedure/Program:

```
import java.sql.*;
Public class MysqlConnectionTest{
Public static void main (String [] args){
String url = "jdbc:mysql://localhost:3306/test";
String username = "your_username";
String Password = "your_Password";
try{
```

```
connection connection=DriverManager.get
                        connection(Url, username,
                        Password);

System.out.println("connected to the database");

connection.close();
}catch (SQLExceptione){
System.out.println("Invalid username orpassword");
e.printstackTrace();
}
}
}
```

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:**

* Sample output 1 :-

connected to the database

* Sample output 2 :-

Invalid username or password.

- **Analysis and Inferences:**

→ The above is a Java JDBC program used to connect to a MySQL database, specifically a "test" database, and check whether the connection is successful based on username and password provided.

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date

Experiment#	19	Student ID	2200038060
Date		Student Name	A.Brahma Naidu

Experiment Title: Handling Form Data in JSP with Servlet

Aim/Objective: To understand how to handle form data in JSP (Java Server Pages) using a servlet in a web application.

Description:

The student will understand the concepts process of handling form data submitted by a user in a JSP page using a servlet. We will learn how to create a form in JSP, collect user input, and send the form data to a servlet for processing.

Pre-Requisites:

Knowledge: Classes, Objects, Java Servlet API, Understanding of HTTP protocol and web development concepts.

Tools: Eclipse IDE for Enterprise Java and Web Developers, Apache Tomcat - Web Server, Java Servlet API.

Pre-Lab:

3. Explain the concept of JSP and its role in web development. How does JSP differ from servlets? Describe the lifecycle of a JSP page.

→ JSP is a technology used in web development to create dynamic web pages. It's primary role is to enable the embedding of Java code within HTML, allowing for the generation of dynamic content on web pages.

→ JSP differs from servlets by embedding Java in HTML, making it easier for web designers.

→ The lifecycle of a JSP page involves several stages:

(i) Translation:

JSP page → servlet (Translation)

(ii) compilation:

servlet compilation (to bytecode)

(iii) Class Loading

Loading servlet class.

(iv) Instantiation:-

Creating servlet Instance

(v) Service:

Handling client Requests
(Generating HTML)

→ The JSP page is translated into a servlet, compiled, loaded, instantiated and then serves client requests by generating HTML dynamically.

4. A management company is organizing a bootcamp for students. They have instructed the students to create a sign-up page using JSP (JavaServer Pages) with fields for first name, last name, email address, phone number, and address. The sign-up page will collect the necessary information from students who want to participate in the bootcamp.

The operations have to be done in maven dynamic web project as follows:

a. Design UI(VIEW) using html to get inputs from the user.

b. Use the post method to store the details.

c. Use the get method and display on the browser.

a)
```
<!DOCTYPE html>
<html>
<head>
    <title> Student Bootcamp sign-up </title>
</head>
<body>
    <h1> Student Bootcamp sign-up </h1>
    <form method = "post" action = "signupservlet">
    <label for = "firstName"> firstName: </label>
    <input type = "text" name = "firstName"
                id = "firstName" required> <br>
    <input type = "submit" value = "sign up">
</form>
</body>
</html>
```

b)
```
protected void doPost (HttpServlet Request request,
        HttP servlet Response response) throws
        servlet Exception, IO exception {

String firstName = request. get parameter("firstName");
String lastName = request. get parameter("lastName");
String email = request. getparameter("Phone Number");
String address = request. get parameter("address");
```

```
response.sendRedirect("confirmation.jsp");
}
c) protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws
        ServletException, IOException {

    String firstName = "John";
    String lastName = "Doe";
    String email = "Jd123@gmail.com";
    String phoneNumber = "9966640275";
    String address = "123 Main st, city, country";
    String details = String.format("First Name: %s <br>
            Last Name: %s <br> email: %s <br>
            PhoneNumber: %s <br> Address: %s <br>",
            firstName, lastName, email, phoneNumber, address);
    response.setContentType("text/html");
    response.getWriter().println("<h1> StudentDetails:
                                <|h1>" + details);
}
```

* Sample output

Student Details:

First Name : John
Last Name : Doe
email : Jd 123@gmail.com
PhoneNumber: 9966640275
Address : 123 Main st, city, country.

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

1. Dinesh, a programmer, has been tasked by his client to develop a simple calculator application using servlets. The application should allow users to perform basic calculations such as addition, subtraction, multiplication, division, and modular operations. Dinesh needs to implement the servlets that handle the user input, perform the calculations, and display the results to the user.

The operations have to be done in maven dynamic web project as follows:

a. Design UI (VIEW) using html to get inputs from the user.

b. Create a Calculator (MODEL) class and implement the above-mentioned operations.

c. Create a Servlet (CONTROLLER) and interact with model and view.

- **Procedure/Program:**

a) Design the UI

```
<!DOCTYPE html>
<html>
<head>
        <title> calculator </title>
</head>
<body>
<form action="calculator servlet" method="post">
<label for="num1"> Number 1: </label>
<input type="text" name="num1" required><br>
<label for="num2"> Number 2; </label>
<input type="text" name="num2" required><br>
<select name="operation">
<option value="add"> Addition </option>
<option value="subtract"> subtraction </option>
<option value="multiply"> multiplication </option>
<option value="divide"> Division </option>
<option value="modulus"> Modulus</option>
</select><br>
<input type="submit" value="calculate">
</form>
</body>
</html>
```

b) create a calculator (Model) class :

```
public class calculator{
public static double perform operation (double num1,
                double num2, string operation){
 double result = 0;
  switch (operation){
  case "add": result = num1 + num2; break;
  case "subtract": result = num1 - num2; break;
  case "multiply": result = num1 * num2; break;
  case "divide": result = num1 / num2; break;
  case "modulus": result = num1 % num2; break;
  }
 return result;
}
}
```

c) create a servlet (controller)

```
import java.io.*;
public class calculator servlet extends Http servlet{
protected void doPost (Http servlet Request request,
                Http servlet Response response)
  throws servlet Exception, IO exception{
double num1 = Double. parse Double(request.
                get parameter ("num1"));
double num2 = Double. parse Double (request. get
                Parameter ("num2"));
String operation = request. get parameter ("operation");
double result = calculator. perform operation
                (num1, num2, operation);
request. set Attribute ("result", result);
request. get Request Dispatcher ("/result.jsp").
                forward (request, response);
}
}
```

Experiment#	<TO BE
Date	<TO BE

* create a

<%@ P

c

<!DO

<htm

<he

<

<!

<

* create a Result JSP (view)

```
<%@ page language="java" contentType="text/html;
    charset=UTF-8" pageEncoding="UTF-8" %>

<!DOCTYPE html>
<html>
<head>
  <title> Calculator Result </title>
</head>
<body>
    <h1> Result: <% = request.getAttribute("result")
                                                %> </h1>

</body>
</html>
```

- **Data and Results:** * sample output:-

  Number 1: 10

  Number 2: 5

  operation: Addition

  * The user clicks "calculate"
  * The result 15 is displayed on the "result.jsp" page.

- **Analysis and Inferences:**

  The above code is used to create a simple calculator application using servlets in Maven dynamic web project, you need to follow the MVC (Model view controller) architectural pattern.

Sample VIVA-VOCE Questions (In-Lab):

1. **What is the role of a servlet in handling form data submitted by a user in a JSP page?**

* Role of a servlet in handling form data submitted by a user in a JSP page.

→ A servlet processes and handles form data submitted by a user from a JSP page.

2. **What is the difference between the request and response objects in servlets?**

* difference between the request and response objects in servlets.

→ Request is for incoming data from the client, while Response is for outgoing data to the client in servlets.

3. Explain the process of sending form data from a JSP page to a servlet for processing.

- To send form data from a JSP page to a servlet for processing

(i) create a form in the JSP.
(ii) set the forms 'action' to the servlet URL.
(iii) use 'name' attributes for form fields.
(iv) In the servlet, use 'request.getParameter("parameterName")' to access the data.

4. What is the purpose of the web.xml deployment descriptor file in a Java web application?

'web.xml' file in a Java web application configures servlets, URL patterns, and other settings for a Java web app.

5. **How can you access the request parameters sent from a JSP form in a servlet?**

you can access request parameters sent from a JSP form in a servlet using the 'request.getParameter("parameter Name")' method.

Post-Lab:

2. Write a code snippet demonstrating the use of JSP and servlet to handle a login form. Use a scenario where a user submits a login form, and the servlet validates the credentials. Your code should include the following:
   - Create a JSP page with a login form that collects username and password.
   - Implement a servlet that receives the form data, validates the credentials, and redirects the user to a success or error page based on the validation result.
   - Test your program by submitting different login credentials to ensure proper validation and redirection.

Procedure/Program:

- create a JSP page for the login form

```
<!DOCTYPE html>
<html>
<head>
    <title>Login page</title>
</head>
<body>
    <h1> Login </h1>
    <form action="loginservlet" method="post">
  <label for="username">username:</label>
  <input type="text" name="username" required><br>
```

```
<label for="password">password:</label>
<input type="password" name="password" required><br>
<input type="submit" value="login">
</form>
</body>
</html>
```

- create a servlet for validation

```
import java.io.*;
Public class loginservlet extends HttpServlet{
protected void doPost(HttpServletRequest request,
                      HttpServletResponse response)
  throws ServletException, IOException{

string username = request.getParameter("username");
string password = request.getParameter("password");
if("admin".equals(username) && "password123".
                  equals(password)){
  response.sendRedirect("success.jsp");
}else{
  response.sendRedirect("error.jsp");
}
}
}
```

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:**

* sample output

username : admin

Password : password 123

→ The user clicks "Login".

→ since the credentials are correct, the user is redirected to the success page. (success.jsp)

- **Analysis and Inferences:**

The above code demonstrates the validation and redirection functionality of the login form and servlet.

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date

Experiment#	20	Student ID	2200032060
Date		Student Name	A.Brahma Naidu

Experiment Title: JSP - Servlet - JDBC Interaction

Aim/Objective: To understand how to interact with a database using JSP, Servlet, and JDBC (Java Database Connectivity).

Description:

The student will understand the concepts of connecting a JSP page and a servlet to a database using JDBC. We will learn how to establish a database connection, execute SQL queries, retrieve data from the database, and display it in the JSP page.

Pre-Requisites:

Knowledge: Classes, Objects, Understanding of servlets, JDBC API, database connectivity, SQL queries, and basic web development concepts.

Tools: Eclipse IDE for Enterprise Java and Web Developers, Apache Tomcat - Web Server, Java Servlet API.

Pre-Lab:

1. Create a table into the database by using the **CREATE TABLE** command for the below mentioned fields.

Create a table "**PERSON**" in **"SOCIETAL"** database that contains five columns:

Fields	**Datatype**
personid	INT, PRIMARY KEY NOT NULL
firstname	VARCHAR(50)
lastname	VARCHAR(50
address	VARCHAR(100)
City	VARCHAR(50)

Create a Project and implement the below mentioned operations in Persontable using switch case for selecting the respective operations:

a. Insert 5 records into the PERSON table.
b. Display all the records in the PERSON table.
c. Insert the below mentioned records in the PERSON Table
   i. 1008, "Jhon", "Wick", "Stains", "NJ"
   ii. 1009, "Shree", "Madhu", "RajNgr", "Pondy"
d. Update the address of "Sree Madhu" as she is migrated to the address "kovai" in the city of Coimbatore.
e. View all the records in the PERSON table of SOCIETAL database.
f. Delete the details of "Jhon" as he is not willing to be a part of the societal database.
g. View all the records in the PERSON table of SOCIETAL database.

* creating the table

```
CREATE DATABASE SOCIETAL;
USE SOCIETAL;
CREATE TABLE PERSON (
 person id INT PRIMARY KEY NOT NULL,
 firstname VARCHAR (50), lastname VARCHAR (50),
 address VARCHAR (100), city VARCHAR (50)
);
```

* Inserting 5 records

```
INSERT INTO PERSON VALUES (1001, 'first1', 'Last1', 'Address1', 'city1');
INSERT INTO PERSON VALUES (1002, first2, 'last2', 'Address2', 'city2');
INSERT INTO PERSON VALUES (1003, 'first3', 'last3', 'Address3', 'city3');
INSERT INTO PERSON VALUES (1004, first4, 'last4', 'Address4', 'city4');
INSERT INTO PERSON VALUES (1005, 'first5', 'last5', 'Address5', 'city5');
```

* Displaying all records

```
SELECT * FROM PERSON;
```

* Inserting additional records

```
INSERT INTO PERSON VALUES (1008, 'Jhon', 'wick', 'stains', 'NJ');
INSERT INTO PERSON VALUES (1009, 'shree', 'Madhu', 'Rajngr', 'Pondy');
```

* updating address

```
UPDATE PERSON SET address = 'Kovai', city = 'coimbatore' WHERE
 firstname = 'shree' AND lastname = 'Madhu';
```

* viewing all records

```
SELECT * FROM PERSON;
```

* Deleting a record

```
DELETE FROM PERSON WHERE firstname = 'Jhon' AND
 lastname = 'wick';
```

* viewing all records

```
SELECT * FROM PERSON;
```

* Java servlet code implementing the above queries.

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
Public class Database servlet extends HttpServlet {
 Public void doGet (HttpServlet Request request, Http
                    servlet Response response) throws
                    servlet Exception, IO exception){
  response.setcontentType("text/html");
  Print writer out = response.getwriter ();
  try{
      Class.ForName ("com.mysql.jbbc.Driver");
connection con = DriverManager.getConnection
  ("jbbc:mysql://localhost:3306/SOCIETAL","username",
   "password");
 statement stmt = con.createstatement();
 //Inserting 5 records
 for(int i = 1; i<= 5; i++){
 stmt.executeupdate("INSERT INTO PERSON VALUES ("+(1000+i)+
   ", 'first"+i"', 'last"+i, "', 'Address"+i", 'city"+i"')");
 }
 //inserting additional records
 stmt.executeupdate("INSERT INTO PERSON VALUES (1008,'Jhon',
          'wick', 'stains', 'NJ')");
 stmt.executeupdate("INSERT INTO PERSON VALUES (
   1009, 'shree', 'Madhu', 'Rajngr', 'Pondy')");
 //updating address
  stmt.executeupdate("UPDATE PERSON SET address =
   'Kovai', city = 'coimbatore' WHERE firstname = 'shree'
   AND lastname = 'Madhu'");
```

```
//Deleting a record
stmt.executeUpdate("DELETE FROM PERSON WHERE
Firstname='Jhon' AND lastname='wick'");
//Displaying all records.
Resultset rs = stmt.executeQuery("SELECT* FROM PERSON");
while (rs.next()){
out.println (rs.getInt (1) + " " + rs.getstring (2) + " " +
" " rs.getstring(4) + " " + rs.getstring (5));
out.println ("<br>");
}
con.close();
}catch (exception e){
out.Println (e);
}
}
```

2. Discuss the importance of prepared statements and parameterized queries in JDBC. Explain how they help prevent SQL injection attacks and improve the security of database operations.

* prepared statements and parameterized queries in JDBC protect against SQL injection attacks by sanitizing input data and providing ~~see~~ secure ways to interact with databases. This enhances security by preventing malicious input from altering SQL queries.

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

1. JDBC – SERVELT – JSP – Connectivity related operations for *"User Registration JSP Form"* & *"Login Authentication JSP Form"* with form Validation.

Create a table into the database by using the **CREATE TABLE** command for the below mentioned fields and create the stored procedure as mentioned below.

a. Create a table "**EMPLOYEE**" in **"IT_INDUSTRY"** database that contains five columns and feed the 4 Employee records manually.

Fields	Datatype
empid	INT, PRIMARY KEY NOT NULL
firstname	VARCHAR(50)
lastname	VARCHAR(50)
dept	VARCHAR(50)
Assigned_project_no	INT
Contact_No	VARCHAR(50)

i. Create a Project and implement the below mentioned operations in EMPLOYEE table with the respective operations:

a. Create a JSP file (VIEW) with the text boxes for all the aforementioned fields along with "Submit" button in order to insert the 5 records in the EMPLOYEE table using Servlet(CONTROLLER) along with the logic writtenin the JAVA Class File (MODEL).

b. Display the form fed data into another JSP (VIEW) file through the Servelt (CONTROL).

- **Procedure/Program:**

*code to create an Employee Table in the database IT_industry using JDBC

```
import java.sql.*;
Public class create employee Table{
Public static void main (String[]args){throws Exception{
connection connection=DriverManager.getconnection("jdbc:my
sql://local host:3306/IT_INDUSTRY", "root", "");
statement.execute("CREATE TABLE employee
(empid INT PRIMARY KEY NOT NULL,
firstname, VARCHAR(50), lastname,
VARCHAR(50), dept VARCHAR(50),
Assigned_Project_no INT, ContactNo
VARCHAR(50))");
Statement statement=
connection.create
statement();
statement.close();
connection.close();
}
}
```

* code shows a simple JSP file for inserting records into the employee table.

```
<%@page content Type="text/html; charset=UTF-8" %>
<html>
<head>
    <title>Insert employee Record </title>
</head>
<body>
    <form action="/insert employee" method="post">
<input type> = "text" name="empid" placeholder="employeeID"/>
<input types="text" name="firstname" placeholder="First Name"/>
<input type="text" name="last name" placeholder="Last Name"/>
<input type="text" name="dept" placeholder="Department"/>
  - - -
  - - -
<input type> = 'submit' value = 'submit'/>
</form>
</body>
</html>
```

* code that shows a simple ~~servlet for~~ hon JSP file to display the form fed data:

```
<%@ page content type="text/html;charset=UTF-8"%>
<html>
<head>
    <title> Display form fed Data </title>
</head>
 <body>
    <h1> form fed data </h1>
    <table>
      <tr>
        <td> employee Id: </td>
        <td><% = request.get parameter("empid")%>
        </td>
      - - -
      - - -
      </tr>
    </table>
 </body>
</html>
```

- **Data and Results:**

* sample output

employee record inserted successfully!
employee ID: [empid]
First Name: [firstname]
last Name: [lastname]
Department: [dept]

- **Analysis and Inferences:**

* The above code is used to record the details of the employees in the database
* By using the concept of JDBC the above task can be implemented successfully.

Sample VIVA-VOCE Questions (In-Lab):

1. **Explain the concept of prepared statements in JDBC. How do they improve the security of database operations?**

Prepared statements in JDBC are pre compiled SQL statements with placeholders for parameters. They improve security by automatically sanitizing data, preventing SQL injection attacks, and seperating SQL from user-provided data.

2. **How can you retrieve data from a database using JDBC? Explain the steps involved?**

* To retrieve data from a database using JDBC, you can follow these steps:-

→ import JDBC packages
→ Load the JDBC driver
→ establish the databaseconnection.
→ create an SQL query.
→ use 'Prepared statement' to execute the query.
→ Retrieve data from the 'Resultset'
→ close resources and handle exceptions.

3. How can you prevent SQL injection attacks when using JDBC for database operations.

* To prevent SQL injection attacks when using JDBC for database operations.

→ use prepared statements.
→ parameterize queries with placeholders
→ sanitize and validate user input
→ escape special characters.
→ Limit permissions and use least privilege principles.

4. Explain the role of JDBC in connecting to and interacting with a database?

* Role of JDBC in connecting to and interacting with a database:-

→ JDBC connects Java applications to databases, handles SQL queries, results, transactions and errors, enabling interaction with databases.

Experiment#	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

5. **How do you establish a database connection using JDBC?**

* To establish a database connection using JDBC.
→ import JDBC packages.
→ Load the JDBC driver with 'class.forName ("driver class name")'
→ specify the database URL, username, and password.
→ use 'Driver Manager.get connection (url, username, password)' to create a connection.

Post-Lab:

JDBC – SERVELT – JSP – Connectivity related operations for *"User Registration JSP Form"* & *"Login Authentication JSP Form"* with form Validation.

Create a table into the database by using the **CREATE TABLE** command for the below mentioned fields and create the stored procedure as mentioned below.

a. Create a table "**LOGIN**" in **"CREDENTIALS"** database that contains five columns and feed the 4 credentials' records manually.

Fields	Datatype
UserName	VARCHAR(50) – UNIQUE & NOTNULL
Password	VARCHAR(50)

i. Create a Project and implement the below mentioned operations in EMPLOYEE table with the respective operations:

c. Create a JSP file (VIEW) with the text boxes for all a forementioned fields along with "Submit" button named as "LOGIN".

d. Validate the Credentials fed in the form by JAVA Class (MODEL) file throughthe SEVELET (CONTROLLER).

e. Display the acknowledgement of the validation in the Same JSP (VIEW) File.

Procedure/Program:

```
a) import java.sql.*;
   public class CreateLoginTable{
   Public static void main (String [] args) throws exception{
   connection connection = DriverManager.getconnection
      ("jdbc:mysql://localhost:3306/CREDENTIALS", "root", "");
   Statement statement = connection.Create statement();
   statement.execute("CREATE TABLE LOGIN (username VARCHAR
      (50) UNIQUE NOTNULL, Password VARCHAR (50)");
   statement.close(); connection.close(); }}
```

(i) <%@ page content type = "text/html; charset = UTF-8" %>

c) <html>
<head>
<title> Login </title>
</head>
<body>
<h1> Login </h1>
<form action = "/login" method = "post">
<input type = "text" name = "username" placeholder = "username"/>
<input type = "password" name = "password" placeholder = "password"/>
<input type = "submit" name = "login" value = "login"/>
</form>
</body>
</html>

d) To validate credentials the below code can be used

```
Public class credential validator{
Public static boolean validate credentials(string username,
                string password){
    return true;
  }
}
```

e) * To display the acknowledgement of the validation in the same JSP (VIEW) file, you can use the below code.

```
<div style = "color:green;" > ${message}</div>
```

- **Data and Results:**

* sample output :

login is successful !

- **Analysis and Inferences:**

→ The above code is used to create a JSP page for login with form validation and acknowledgement in green text.

Evaluator Remark (if Any):	Marks Secured: _____out of 50
	Signature of the Evaluator with Date