Experiment#		Student ID	
Date		Student Name	

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

Tools: Eclipse IDE for Enterprise Java and Web Developers

Pre-Lab:

1) Explain the purpose and usage of the Cloneable interface in Java.

→ Purpose:

The cloneable interface allows objects of a class to be cloned

→ Marker Interface:

It does not define any methods; it just signals that the clone() method can be safely called.

→ clone() method:

* Defined in the object class.
* Performs a shallow copy of the object.
* Must be overridden in the class that implements Cloneable to handle specific

→ cloning logic.

→ Checked at Runtime:

If the clone() method is called on an object whose class does not implement Cloneable, a CloneNotSupportedException is thrown.

→ Usage Steps:

* Implement the Cloneable interface in the class.
* Override the clone() method to customize the cloning process.
* Call super.clone() within the overriden method to perform the default shallow copy.

Experiment#		Student ID	
Date		Student Name	

In-Lab:

1) 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;
class Employee implements Cloneable {
    private String name;
    private string dob;
    private String mobileNumber;
    private int id;
    public Employee(String name, String dob, String mobileNumber, int id) {
        this.name = name;
        this.dob = dob;
        this.mobileNumber = mobileNumber;
        this.id = id;
} public String getName() {
        return name;
} public string getDob() {
        return dob;
} public string getMobileNumber() {
        return mobileNumber;
}
```

```
public int getId() {
    return id;
}protected Employee clone() throws CloneNotSupportedException{
    return (Employee) super.clone();
}public void displayEmployeeDetails() {
    System.out.println("ID: "+id+", Name: "+name+", DOB: "+dob+", Mobile: "
                       +mobileNumber);
}
}class EmployeeList implements Itevable<Employee>{
    private ArrayList<Employee> employees = new ArrayList<>();
    public void addEmployee(Employee employee) {
        employees.add(employee);
    }public Iterator<Employee> iterator(){
        return new EmployeeIterator();
}private class EmployeeIterator implements Iterator<Employee>{
    private int index = 0;
    public boolean hasNext() {
        return index < employees.size();
    }public Employee next() {
        return employees.get(index++);
    }
}
}
```

```
public class EmployeeManagementSystem {
    public static void main(String[] args) {
        EmployeeList employeeList = new EmployeeList();
        employeeList.addEmployee(new Employee("Alice", "1990-01-01", "9876543210", 1));
        employeeList.addEmployee(new Employee("Bob", "1985-05-15", "9876543211", 2));
        employeeList.addEmployee(new Employee("Charlie", "1992-10-20", "9876543212", 3));
        for (Employee emp : employeeList) {
            emp.displayEmployeeDetails();
        }
        try {
            Employee clonedEmployee = employeeList.iterator().next().clone();
            System.out.println("Cloned Employee:");
            clonedEmployee.displayEmployeeDetails();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
}
```

Experiment#		Student ID	
Date		Student Name	

✓ **Data and Results:**

O/P:

ID:1, Name: Alice, DOB:1990-01-01, Mobile: 9876543210

ID:2, Name: Bob, DOB: 1985-05-15, Mobile: 9876543211

ID:3, Name: Charlie, DOB: 1992-10-20, Mobile: 9876543212

Cloned Employee:

ID:1, Name: Alice, DOB:1990-01-01, Mobile: 9876543210

✓ **Analysis and Inferences:**

Iteration: The program successfully iterates through the EmployeeList using the Iterator interface, displaying each employee's details.

Cloning: The Cloneable interface allows for cloning, demonstrated by creating a copy of the first employee.

Conclusion: The program fulfills its purpose of managing employee data, iterating through it, and creating clones when necessary.

Experiment#		Student ID	
Date		Student Name	

VIVA-VOCE Questions (In-Lab):

1) What is the purpose of the Cloneable interface?

The cloneable interface allows objects to be cloned, indicating that the class supports field-by-field copying.

2) How does the Cloneable interface enable object cloning in Java?

Implementing Cloneable Signals to java that the clone() method can be called to create a shallow copy of the object. Without it, calling clone() will throw a CloneNotSupportedException.

3) Explain the role of the Iterator interface in Java collections

The Iterator interface provides a way to traverse elements in a collection sequentially without exposing the underlying structure.

4) What are the methods provided by the Iterator interface? Explain their significance.

hasNext(): checks if there are more elements to iterate.

next(): Returns the next element in the iteration.

remove(): Removes the current element

5) Can you explain the difference between the Cloneable interface and the Clone method in Java?

* Cloneable is a marker interface indicating that cloning is allowed.
* clone() is the method that performs the actual copying of an object.

Implementing Cloneable is necessary for the clone() method to work correctly.

Post-Lab:

1) 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 rollNumber, double marks) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.marks = marks;
}
    public String getName() {
        return name;
}   public int getRollNumber(){
        return rollNumber;
}
```

```
public double getMarks() {
    return marks;
} protected Object clone() throws CloneNotSupportedException {
    return super.clone();
} public String toString() {
    return "Student {" + "name='" + name + '\'' + ", rollNumber=" + rollNumber +
        ", marks=" + marks + '}';
  }
}
public class Main {
   public static void main(String[] args) {
       try{
          Student originalStudent = new Student("John Doe", 101, 85.5);
          System.out.println("Original Student: " + originalStudent);
          Student clonedStudent = (Student) originalStudent.clone();
          System.out.println("Cloned Student: " + clonedStudent);
       } catch(CloneNotSupportedException e) {
          e.printStackTrace();
       }
   }
}
```

Experiment#		Student ID	
Date		Student Name	

✓ **Data and Results:**

Original Student : Student {name = 'John Doe', rollNumber = 101, marks = 85.5}

Cloned Student : Student {name = 'John Doe', rollNumber = 101, marks = 85.5}

✓ **Analysis and Inferences:**

* The implementation of the Cloneable interface allows for creating duplicates of Student objects without manually copying their attributes.

* The overridden clone method correctly utilizes the default behavior of object to duplicate the object's state.

* The results indicate that cloning maintains the integrity of original object's data, making it useful for scenarios where independent copies of objects are needed.

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.