Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 5

Title of the Laboratory Exercise: Generic classes, Collections and their uses

1. Introduction and Purpose of Experiment

   Students apply object oriented programming concepts Generic classes and Collections to solve problems.

2. Aim and Objectives

   Aim

   To apply object oriented programming concepts Generic classes and Collections to solve problems

   Objectives

   At the end of this lab, the student will be able to

   - Apply Generic classes and Collections for solving simple problems
   - Express generic solutions in Java language

3. Experimental Procedure

For the problems listed below, design the data structures, algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Java with C programming languages. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Write a program to solve for the following scenario:

Extend the program created in laboratory 2 to include logic for input of multiple employees. These employees should be stored in a collection such as ArrayList. The program should display the salaries of each employee as output, after all the employee details are entered. Extend the program further to display the employee details in ascending order of their salaries.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	**Procedure:** 1. Generating a new Java Application Open Netbeans and create a new Project and select Java -> Java Application, give a proper package name and a proper Project Name, this will generate a CLI Interface Java Application, with a main method in a Class File with the Project's name. 2. Design the Class Diagram Analyze the given problem and identify the classes and objects, find the common state and action of the various objects and classes, basically perform an object decomposition and make a UML Diagram for the same, this will make the development easier while implementing the application. 3. Creating the Classes To separate the models that were identified in the UML Diagram, create a sub-package "models", add the identified classes here. Use the keyword `extends` to inherit a parent class into the base class. Add the classes as per the UML Diagram. Add the state for each of the classes, i.e. the variables inside each of the classes. Use the keyword abstract to declare an abstract class or an abstract method, any class containing an abstract method must be an abstract class. Objects of abstract classes cannot be instantiated. 4. Implementing the methods Now that all the classes have been added, the methods in those needs to be implemented as per the logic for the individual classes, refer the question for the same. Follow the following procedure when implementing the methods for the state inside the class. i. Right click -> insert code -> Constructor : to add the constructor to the class. This method is called whenever an object of this class or its child class is made.

ii. Right click -> insert code -> getters and setters : make sure to click on encapsulate fields, this will make the variables in the class to private, this is required for abstraction. Getters and Setters are methods that can change the state of the object and fetch the current state of the object.

Now the abstract method from the parent classes are to be implemented in the child classes. Write your business logic here in those methods wherever required.

Write the definition for the main method, and make sure to import the previously defined classes using the correct package name.

5. Creating a generic container for Employee Objects

Since we need to store many employees, we create an ArrayList, which is a generic type list, and can store any kind of object.

6. Overriding generic Methods

Java Classes by default inherit the Object Class which comes with some generic methods such as `toString`, `equals`, these can be overridden too. To indicate an overridden method, use the annotation `@Override` before the method definition. This needs to be done to display an Employee on the console. Since we also need to be able to compare two employees we would override `compareTo` method, this method is from an interface Comparable, and the syntax of implementing is `implements Comparable<ClassName>`, this is a generic type in Java that can take any class as argument, and any class that wants to override `compareTo` must implement the Comparable Class, doing this enables us to compare two object of our class and sort them accordingly. `Collections.sort` is a static method that can sort an ArrayList of Objects, provided the objects's class must have the definition of `compareTo`. Override `toString` to display the details of an Employee.

7. Execute and Debug

Execute the program by clicking on Clean and Build and then Run, and also perform proper tests on it. Verify that the program is as per the specifications required.

8. Documentation

Write documentation for the methods and Classes implemented in the program, with its usage and parameter, the developer's name and date.

Algorithms:

Algorithm compareTo for Employee

Params: Employee e to be compared with the current employee

```
Step 1: Start
Step 2: return getSalary of current employee – getSalary of employee e
Step 5: Stop
```

Algorithm toString for Employee

Params: None (although the object is passed internally)

```
Step 1: Start
Step 2: print firstName, lastName, aadharNumber, getSalary
Step 3: Stop
```

Algorithm for main method

Params: Command Line Arguments (None in this case)

```
Step 1: Start
Step 2: Display the Menu of Employees
Step 3: Instantiate a new ArrayList employees of Employee type.
Step 4: Instantiate a new Employee employee and assign null to it
Step 5: Take the choice from user and Instantiate the respective employee, and assign it to employee.
```

	Step 6: If employee != null, then take the input for the employee from user Step 7: add the employee to the ArrayList employees Step 8: Repeat step 5 to step 7 to add more employees to the ArrayList Step 9: Collections.sort(employees) Step 10: print employees Step 7: Stop
b. Conclusions :	The problem here required a Generic Solution, since the problem states the employees are to be stored, this needed an ArrayList, which is a Generic Class and can store objects of any Type, ArrayList is a subclass of AbstractList which implements AbstractCollection which is then implements Collection, A Collection can work on any kind of Data Type in Java. Here which is an Employee. Using Generic Classes in this way is another form of Polymorphism in Java. The same class can work with different data types. Some data specific methods such as comapareTo, toString are to be overridden to sort and print the data when required. It's easy for a developer to understand this kind of code since the code to sort is already defined in Collections, and needn't be written, the way the ArrayList stores the Data is also not relevant since that is also written by trusted developers and the ArrayList can be looked at, at a more abstract level. In this way we are reducing the amount of code that we have to write and hence making it easier to debug.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Results and Discussions:

Screenshot:

```
public abstract class Employee implements Comparable<Employee> {

    private String firstName;
    private String lastName;
    private String aadharNumber;

    public Employee() {
    }

    public Employee(String firstName, String lastName, String aadharNumber) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.aadharNumber = aadharNumber;
    }

    @Override
    public int compareTo(Employee o) {
        return (int) (this.getSalary() - o.getSalary());
    }

    public abstract Float getSalary();

    public void getInput() {
        Scanner input = new Scanner(System.in);
        input.useDelimiter("\n");
        System.out.print("Enter First Name : ");
        this.firstName = input.nextLine();
        System.out.print("Enter Last Name : ");
        this.lastName = input.nextLine();
        System.out.print("Enter Aadhar Number : ");
        this.aadharNumber = input.nextLine();
    }

    @Override
    public String toString() {
        return "\nEmployee :: " + "First Name : " + firstName + ", Last Name : " + lastName + ",
Aadhar Number : " + aadharNumber + ", Salary : " + getSalary();
    }
```

Discussion:

Employee is the super class of the different employees that inherit it, and we need to be able to compare two employees in order to sort them, since they are to be sorted using their salaries, this state property of employees will be used to sort them. Collections.sort is a static method from the Class Collections, that contains many generic methods such as that, it can sort an ArrayList of Objects, each object of the ArrayList is compared by calling the compareTo method of the class, and hence we need to have a compareTo method, Collections.sort will check if the Objects present in the ArrayList that was passed to it as an argument implements Comparable or not, Comparable is an Generic interface which contains a method compareTo, so if our Class implements Comparable then it has to override compareTo. compareTo method takes an

Employee as an argument and compares the current employee with the formal parameter employee, and returns an int.

Since we also need to print the list of employees, the ArrayList can be printed using System.our.println, the only problem with this is that ArrayList will call toString method for each of the objects that were stored in, every Class is a subclass of Object and hence the generic toString method will be called that prints the hashCode of the object, we need to override toString in our employee to give the right output, hence we display the firstName, lastName, aadharNumber and salary of employee, salary is the getSalary of employee, which gives the salary of the current employee, this way we have created our class such that generic methods can be used with it.

Screenshot:

```
public class ACMEEmployeeSalary {

    /**
     * @param args the command line arguments
     */
    static ArrayList<Employee> employees = new ArrayList<>();
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        Integer choice;
        String mainMenu
                = "------------------------- ACME Employee --------------------------------\n"
                + "1.\tAdd Employee to the List\n"
                + "2.\tCalculate Salary for Employees\n"
                + "3.\tExit"
                + "\n\nYour Choice : ";
        String menu
                = "Select the type of employee : \n"
                + "1.\tCommission Employee\n"
                + "2.\tBase Plus Employee\n"
                + "3.\tSalaried Employee\n"
                + "4.\tHourly Employee\n"
                + "5.\tPiece Work Emlpoyee\n"
                + "6.\tExit"
                + "\n\tYour Choice : ";
        System.out.print(mainMenu);
        choice = input.nextInt();
        switch (choice) {
            case 1: {
                System.out.print(menu);
                choice = input.nextInt();
                Employee employee = null;
                switch (choice) {
                    case 1:
                        employee = new CommissionEmployee();
                        break;
                    case 2:
                        employee = new BasePlusCommissionEmployee();
                        break;
                    case 3:
                        employee = new SalariedEmployee();
                        break;
                    case 4:
```

```
                        employee = new HourlyEmployee();
                        break;
                    case 5:
                        employee = new PieceWorkEmployee();
                        break;
                    case 6:
                        System.exit(0);
                    default:
                        System.out.println("Wrong Choice !");
                        break;
                }
                if (employee != null) {
                    employee.getInput();
                    employees.add(employee);
                    System.out.println(employee.toString());
                    System.out.println("Employee Added to the list");
                }
            }
            break;
            case 2: {
                Collections.sort(employees);
                System.out.println(employees);
                System.out.println();
            } break;
            case 3: {
                System.exit(0);
            }
            default:
                System.out.println("Wrong Choice !");
                break;
        }
        /* Infinite Recursion */
        main(args);
    }
}
```

Discussion:

The main method is the driver method for the ACMEEmployeesPayroll, this presents the user with a menu of the kinds of employees supported, the selected employee is created by using the `new` operator that creates an object of that class and returns a reference to it. The input from the user is taken using the getInput method of employee and the salary for that employee is obtained by calling the getSalary method. getInput method uses the Scanner class of java. The created Employee Object is then added to the ArrayList, these steps are repeated for any amount of employees the user wants, when the user wants to view the payroll of the employees, since we also want them to be sorted in the magnitude of their salary, they are sorted using Collections.sort, and then displayed using System.out.println, as we had overridden the toString method, the output is pretty printed.

Screenshot:

```
run:
------------------------- ACME Employee --------------------------------
1.     Add Employee to the List
2.     Calculate Salary for Employees
3.     Exit
```

```
Your Choice : 1
Select the type of employee :
1.     Commission Employee
2.     Base Plus Employee
3.     Salaried Employee
4.     Hourly Employee
5.     Piece Work Emlpoyee
6.     Exit
       Your Choice : 1
Enter First Name : Satyajit
Enter Last Name : Ghana
Enter Aadhar Number : 123412341234
Enter Commission Rate : 20
Enter Sales : 12000

Employee :: First Name : Satyajit, Last Name : Ghana, Aadhar Number : 123412341234, Salary
: 2400.0
Employee Added to the list
------------------------- ACME Employee --------------------------------
1.     Add Employee to the List
2.     Calculate Salary for Employees
3.     Exit

Your Choice : 1
Select the type of employee :
1.     Commission Employee
2.     Base Plus Employee
3.     Salaried Employee
4.     Hourly Employee
5.     Piece Work Emlpoyee
6.     Exit
       Your Choice : 2
Enter First Name : John
Enter Last Name : Doe
Enter Aadhar Number : 123412341235
Enter Commission Rate : 22
Enter Sales : 10000
Enter basic Salary : 10000

Employee :: First Name : John, Last Name : Doe, Aadhar Number : 123412341235, Salary :
12200.0
Employee Added to the list
------------------------- ACME Employee --------------------------------
1.     Add Employee to the List
2.     Calculate Salary for Employees
3.     Exit

Your Choice : 2
[
Employee :: First Name : Satyajit, Last Name : Ghana, Aadhar Number : 123412341234, Salary
: 2400.0,
Employee :: First Name : John, Last Name : Doe, Aadhar Number : 123412341235, Salary :
12200.0]

------------------------- ACME Employee --------------------------------
1.     Add Employee to the List
2.     Calculate Salary for Employees
3.     Exit
```

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

```
Your Choice : 3
BUILD SUCCESSFUL (total time: 1 minute 15 seconds)
```

Discussion:

This is a menu driven program, first the ArrayList of employees is created which is empty in the beginning then the payroll menu is presented to the user which presents the list of operations that the user can do, the user then select the option, here, to add a new employee, then the kind of employee menu is displayed, after selection of this a new Employee object is created and getInput method is called on it which takes the input for that kind of employee from the console. Then the Employee is added to the ArrayList using the add method of the array. Then main menu is displayed again, and the same process continues. When the display payroll option is selected, this ArrayList's sort method is called first to sort the list, which used the comparator that we have implemented and overridden in the Employee class, this comparator compares two employees based on their salary, then to display the info of the employee, the ArrayList's toString method is called, which in-turn calls the toString method of the Employee object, we have overridden toString for employee which displays the employee info with the salary, and since we have already sorted the ArrayList the employees are displayed in terms of their Salary in ascending order.