
# ASSIGNMENT # 01

---

## Student Name: Kabeer Abdul Karim

## Student ID:67517

## Department: Computer Science

## Program: BSCS

## Semester: Fall 2024

## Section Code:

## Course Title: Object Oriented programming

## Course Code:

## Course Instructor: Kamran khan

# Lab Session 7 (Open-Ended Lab assignment)

| Open Ended Lab | |
|---|---|
| **Blooms Taxonomy** | **GAs** |
| C3<br><br>A2 | GA-2<br>GA-6 |

**Title:**
Build Your Own Mini Application with OOP

**Objective:**
To design and implement a mini-application using Object-Oriented Programming concepts. Students will apply OOP principles such as encapsulation, inheritance, polymorphism, and abstraction to develop a real-world application of their choice.

**Methodology:**

1. **Proposal Submission (Initial Planning)**

   Each student or team will submit a brief proposal outlining the application they want to create. The proposal should include:

   - The problem they aim to solve.
   - The application's purpose and expected functionality.
   - A list of classes, objects, and methods they plan to implement.
   - Justification for how they will use key OOP principles (encapsulation, inheritance, polymorphism, abstraction).

2. **Design Phase (Class Diagram and Structure)**

   Develop a UML class diagram to represent the relationships between classes in their application.

   - Identify attributes and methods for each class.
   - Outline inheritance hierarchies and any abstract classes or interfaces.

3. **Coding Phase (Implementing Core Features)**

   Implement the application based on the design plan. Emphasize:

- **Encapsulation**: Use private/protected attributes and appropriate getters and setters.
- **Inheritance**: Implement a base class and at least one derived class.
- **Polymorphism**: Use method overriding and polymorphic behavior where possible.
- **Abstraction**: If applicable, use abstract classes or interfaces.

4. **Advanced Feature Implementation (Optional)**

   Add additional features or optimizations to enhance the application. Examples:

   - **Persistence**: Save and retrieve data from a file or database.
   - **Design Patterns**: Apply any relevant design patterns (e.g., Singleton, Factory).
   - **Error Handling**: Implement error handling to manage invalid inputs or system errors.

**Deliverables:**

1. A one-page document detailing the application idea and initial design.
2. A class diagram with a brief explanation of each class and its responsibilities within the application.
3. A working codebase with well-documented code and inline comments explaining each class and method.
4. An enhanced version of the application code with documentation.

***Suggested Application Ideas***

Here are some open-ended application ideas to inspire creativity:

1. **Library Management System**
   - Track books, patrons, and loans.
   - Use inheritance for different book genres or patron types.
2. **Task Management App**
   - Create tasks, set deadlines, and assign priorities.
   - Use polymorphism for different task categories (e.g., Work, Personal).
3. **Inventory System for a Retail Store**
   - Manage stock, sales, and customer details.
   - Implement inheritance for different product categories.
4. **Simple Game (e.g., Card Game, Tic-Tac-Toe)**
   - Design classes for players, game mechanics, and rules.
   - Use polymorphism to support multiple types of moves or actions.
5. **Student Grading System**
   - Track student grades, subjects, and calculate averages.
   - Implement inheritance for different student categories (e.g., Full-Time, Part-Time).

***Assessment Criteria***

- **Creativity**: Originality and thoughtfulness of the application idea.
- **OOP Principles**: Proper and effective use of encapsulation, inheritance, polymorphism, and abstraction.
- **Design Quality**: Clarity, completeness, and accuracy of the class diagram.
- **Functionality**: The application works as expected and meets the initial requirements.
- **Code Quality**: Code readability, structure, and use of comments.
- **Documentation**: Quality of documentation, explaining the classes, methods, and program structure.

***Reflection Report***

After completing the project, each student/team will write a short reflection report addressing:

- Challenges faced during the project.
- How they applied each OOP concept.
- Potential improvements if given more time.

***Learning Outcomes***

By the end of this lab, students will:

- Gain hands-on experience in designing and implementing an OOP-based application.
- Understand and apply OOP principles in a real-world context.
- Develop skills in software planning, problem-solving, and project management

**Code:**

```
import java.io.*;
import java.util.*;

class Person {
    private String name;
    private int age;
    private String contact;

    public Person(String name, int age, String contact) {
        this.name = name;
        this.age = age;
        this.contact = contact;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getContact() {
        return contact;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Age: " + age + ", Contact: " + contact;
    }
}

class Employee extends Person {
    private static int employeeCounter = 0;
    private final int employeeId;
    private String designation;
    private double salary;

    public Employee(String name, int age, String contact, String designation,
double salary) {
        super(name, age, contact);
        this.employeeId = ++employeeCounter;
```

```
        this.designation = designation;
        this.salary = salary;
    }

    public int getEmployeeId() {
        return employeeId;
    }

    public String getDesignation() {
        return designation;
    }

    public void setDesignation(String designation) {
        this.designation = designation;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    @Override
    public String toString() {
        return "ID: " + employeeId + ", " + super.toString() + ", Designation: "
+ designation + ", Salary: " + salary;
    }
}

class EmployeeManagementSystem {
    private List<Employee> employees = new ArrayList<>();

    public void addEmployee(String name, int age, String contact, String
designation, double salary) {
        Employee emp = new Employee(name, age, contact, designation, salary);
        employees.add(emp);
        System.out.println("Employee " + name + " added successfully!");
    }

    public void viewEmployees() {
        if (employees.isEmpty()) {
            System.out.println("No employees found!");
        } else {
```

```
            for (Employee emp : employees) {
                System.out.println(emp);
            }
        }
    }

    public void searchEmployee(String searchKey) {
        List<Employee> results = new ArrayList<>();
        for (Employee emp : employees) {
            if (String.valueOf(emp.getEmployeeId()).equalsIgnoreCase(searchKey)
                    || emp.getName().equalsIgnoreCase(searchKey)) {
                results.add(emp);
            }
        }
        if (results.isEmpty()) {
            System.out.println("No matching employees found!");
        } else {
            for (Employee emp : results) {
                System.out.println(emp);
            }
        }
    }

    public void updateEmployee(int empId, String designation, Double salary) {
        for (Employee emp : employees) {
            if (emp.getEmployeeId() == empId) {
                if (designation != null) {
                    emp.setDesignation(designation);
                }
                if (salary != null) {
                    emp.setSalary(salary);
                }
                System.out.println("Employee " + empId + " updated
successfully!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }

    public void deleteEmployee(int empId) {
        Iterator<Employee> iterator = employees.iterator();
        while (iterator.hasNext()) {
            Employee emp = iterator.next();
            if (emp.getEmployeeId() == empId) {
```

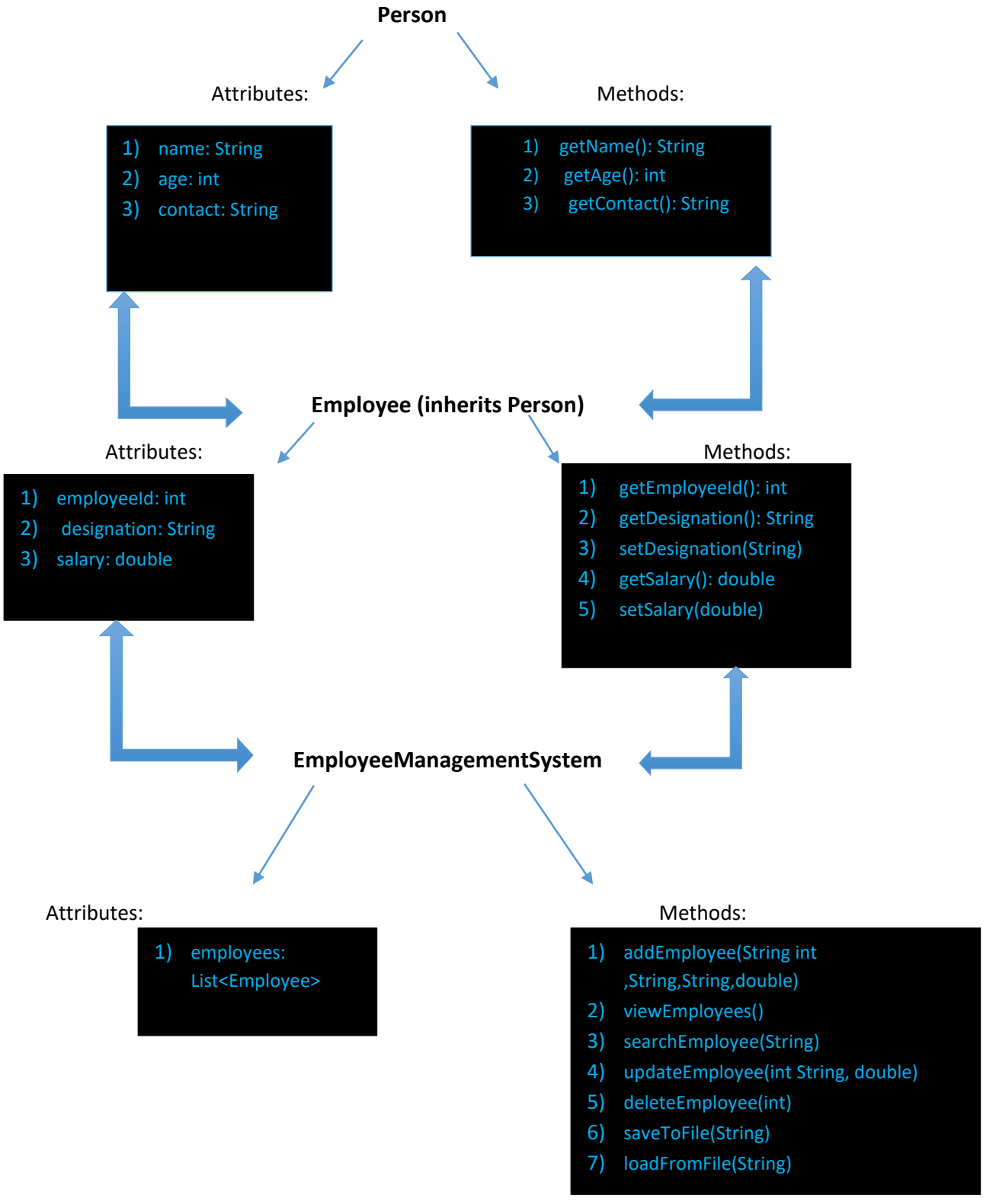
```
                iterator.remove();
                System.out.println("Employee " + empId + " deleted
successfully!");
                return;
            }
        }
        System.out.println("Employee not found!");
    }

    public void saveToFile(String filename) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(filename))) {
            oos.writeObject(employees);
            System.out.println("Data saved to file successfully!");
        } catch (IOException e) {
            System.out.println("Error saving to file: " + e.getMessage());
        }
    }

    @SuppressWarnings("unchecked")
    public void loadFromFile(String filename) {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(filename))) {
            employees = (List<Employee>) ois.readObject();
            System.out.println("Data loaded from file successfully!");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error loading from file: " + e.getMessage());
        }
    }
}

public class EmployeeManagement {
    public static void main(String[] args) {
        EmployeeManagementSystem system = new EmployeeManagementSystem();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("\n--- Employee Management System ---");
            System.out.println("1. Add Employee");
            System.out.println("2. View Employees");
            System.out.println("3. Search Employee");
            System.out.println("4. Update Employee");
            System.out.println("5. Delete Employee");
            System.out.println("6. Save to File");
            System.out.println("7. Load from File");
```

```
            System.out.println("8. Exit");
            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine();

            switch (choice) {
                case 1:
                    System.out.print("Enter name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter age: ");
                    int age = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter contact: ");
                    String contact = scanner.nextLine();
                    System.out.print("Enter designation: ");
                    String designation = scanner.nextLine();
                    System.out.print("Enter salary: ");
                    double salary = scanner.nextDouble();
                    system.addEmployee(name, age, contact, designation, salary);
                    break;

                case 2:
                    system.viewEmployees();
                    break;

                case 3:
                    System.out.print("Enter employee ID or name to search: ");
                    String searchKey = scanner.nextLine();
                    system.searchEmployee(searchKey);
                    break;

                case 4:
                    System.out.print("Enter employee ID to update: ");
                    int empId = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter new designation (leave blank to
skip): ");
                    String newDesignation = scanner.nextLine();
                    System.out.print("Enter new salary (leave blank to skip): ");
                    String salaryInput = scanner.nextLine();
                    Double newSalary = salaryInput.isEmpty() ? null :
Double.parseDouble(salaryInput);
                    system.updateEmployee(empId, newDesignation.isEmpty() ? null
: newDesignation, newSalary);
```

```
                    break;

                case 5:
                    System.out.print("Enter employee ID to delete: ");
                    empId = scanner.nextInt();
                    system.deleteEmployee(empId);
                    break;

                case 6:
                    System.out.print("Enter filename to save: ");
                    String saveFile = scanner.next();
                    system.saveToFile(saveFile);
                    break;

                case 7:
                    System.out.print("Enter filename to load: ");
                    String loadFile = scanner.next();
                    system.loadFromFile(loadFile);
                    break;

                case 8:
                    System.out.println("Exiting the system. Goodbye!");
                    scanner.close();
                    return;

                default:
                    System.out.println("Invalid choice! Please try again.");
            }
        }
    }
}
```

# UML Diagram: Employee Management System

This UML class diagram illustrates the Employee Management System, demonstrating the relationships and methods between classes. Shapes are scaled for visibility.



# OUTPUT:

## Adding an Employee:

```
--- Employee Management System ---
1. Add Employee
2. View Employees
3. Search Employee
4. Update Employee
5. Delete Employee
6. Save to File
7. Load from File
8. Exit
Enter your choice: 1
Enter name: abc
Enter age: 25
Enter contact: 090000000
Enter designation: sales
Enter salary: 50000
Employee abc added successfully!

--- Employee Management System ---
1. Add Employee
2. View Employees
3. Search Employee
4. Update Employee
5. Delete Employee
6. Save to File
7. Load from File
8. Exit
Enter your choice:
```

## Checking Employees:

```
--- Employee Management System ---
1. Add Employee
2. View Employees
3. Search Employee
4. Update Employee
5. Delete Employee
6. Save to File
7. Load from File
8. Exit
Enter your choice: 2
ID: 1, Name: abc, Age: 25, Contact: 090000000, Designation: sales, Salary: 50000.0
```

## Searching an Employee:

```
--- Employee Management System ---
1. Add Employee
2. View Employees
3. Search Employee
4. Update Employee
5. Delete Employee
6. Save to File
7. Load from File
8. Exit
Enter your choice: 3
Enter employee ID or name to search: abc
ID: 1, Name: abc, Age: 25, Contact: 090000000, Designation: sales, Salary: 50000.0
```

## Updating an Employee:

```
--- Employee Management System ---
1. Add Employee
2. View Employees
3. Search Employee
4. Update Employee
5. Delete Employee
6. Save to File
7. Load from File
8. Exit
Enter your choice: 4
Enter employee ID to update: 1
Enter new designation (leave blank to skip): Manager
Enter new salary (leave blank to skip): 70000
Employee 1 updated successfully!
```

## Deleting An Employee:

```
--- Employee Management System ---
1. Add Employee
2. View Employees
3. Search Employee
4. Update Employee
5. Delete Employee
6. Save to File
7. Load from File
8. Exit
Enter your choice: 5
Enter employee ID to delete: 1
Employee 1 deleted successfully!
```