# OODP workshop 10

# Student Class

1. Think about a class named Employee in payroll system of an organisation.

Ans:

An Employee class in a payroll system should include:

* **Employee ID** (unique identifier)
* **Name** (full name)
* **Department** (e.g., HR, Engineering)
* **Salary** (base compensation)
* **Hire Date** (date of employment)
* **Tax Rate** (percentage deducted for taxes)
* **Leave Balance** (remaining paid time off days).

1. What are the different attributes that an Employee class can have?

Ans:

Employee Class Attributes

A well-designed Employee class for a payroll system should include the following attributes:

1. **Employee ID** (employeeId): Unique identifier for each employee.
2. **Name** (name): Full name of the employee.
3. **Department** (department): Organizational unit (e.g., "HR", "Engineering").
4. **Salary** (salary): Base compensation amount.
5. **Hire Date** (hireDate): Date of employment start.
6. **Tax Rate** (taxRate): Percentage deducted for taxes.
7. **Leave Balance** (leaveBalance): Remaining paid time off days.
8. What are the methods that an Employee class can have?

Ans:

**Employee Class Methods**

1. **Constructors**
   * **Default constructor: Initializes empty fields.**
   * **Parameterized constructor: Sets all attributes**

**2. Getters/Setters**

* **One pair per attribute (7 getters, 7 setters)**

**3. Functional Methods**

* **calculateNetSalary(): Computes post-tax income.**
* **applyLeave(int days): Reduces leave balance**

1. How many constructors this class can have?

Ans: A class can have **multiple constructors**:

* Empty constructor (default values).
* Constructor with essential fields (ID, name, salary).
* Full-parameter constructor (all attributes).

1. How many getters and setters this class can have?

Ans:

* In general - One getter and one setter per attribute .

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Getter Method** | **Setter Method** |
| employeeId | getEmployeeId() | setEmployeeId(int id) |
| name | getName() | setName(String name) |
| department | getDepartment() | setDepartment(String dept) |

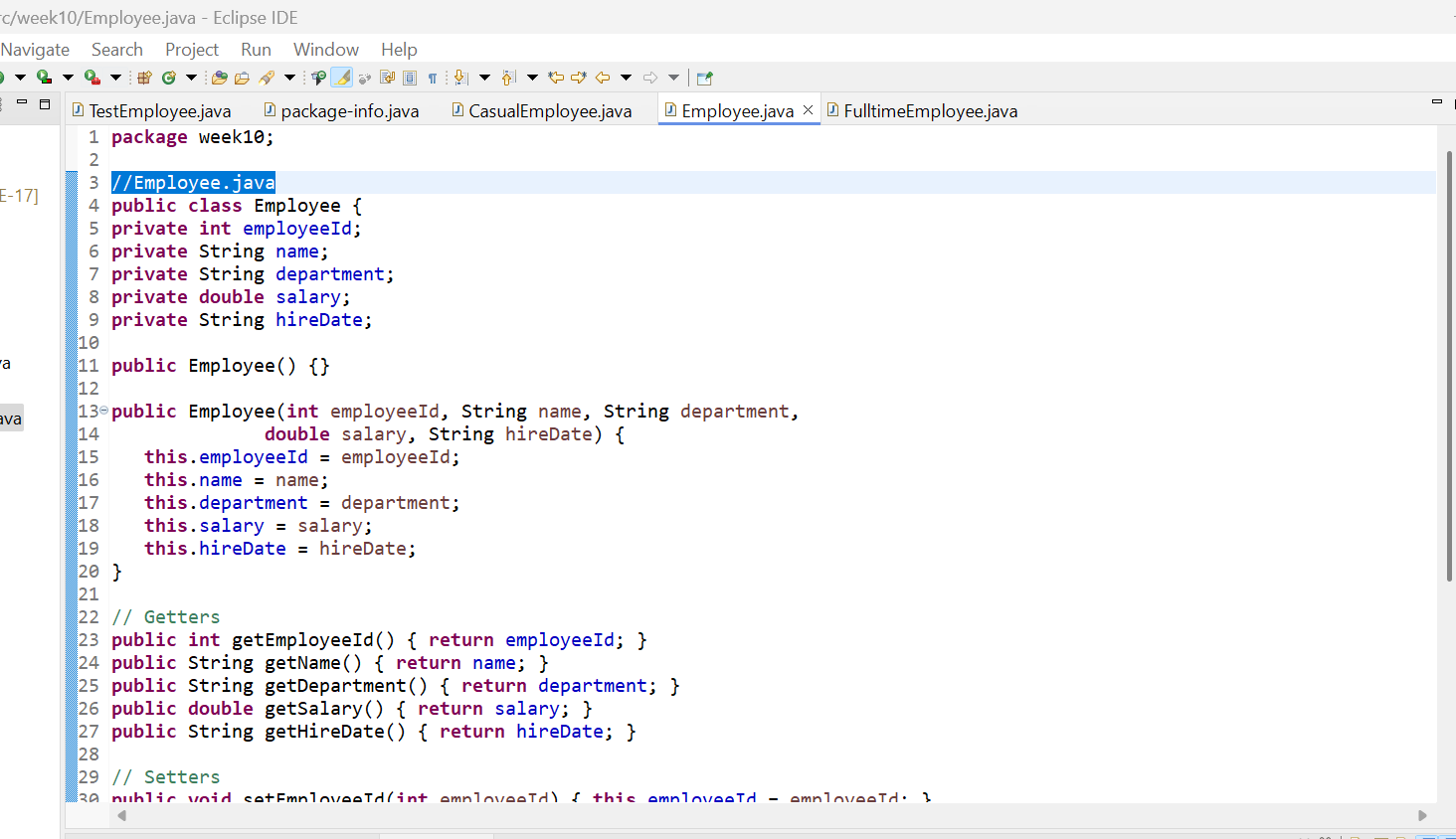
1. What a toString() method will display in this class?

Ans:

Returns a string summarizing all attributes, formatted as:

Employee[ID=101, Name=John Doe, Department=HR, Salary=50000.0, HireDate=2023-01-01, TaxRate=0.20, LeaveBalance=10]

Open Eclipse, and develop this class on the basis of information you have written in answers above.



# FulltimeEmployee Class

Consider FulltimeEmployee class as a sub class of Employee

1. What is the additional attribute that a FulltimeEmplloyee class can have?

Ans: **Additional Attribute**

annualBonus: Represents yearly performance-based compensation.

1. Discuss the different types of constructor that this class can have and how will you use “Super” keyword?

Ans:

**Constructors and Super** Usage ( super keyword Calls default constructor)

* + **Default Constructor**:

**public** FulltimeEmployee() {

**super**(); *// Calls Employee's default constructor*

}

* **Purpose**: Creates a FulltimeEmployee object with default values inherited from Employee and annualBonus=0.0.
* **Super Usage**: Explicitly invokes the parent class's default constructor using super(), ensuring proper initialization of inherited fields.
  + **Parameterized Constructor**:

**public** FulltimeEmployee(**int** employeeId, String name, String department,

**double** salary, String hireDate, **double** annualBonus) {

**super**(employeeId, name, department, salary, hireDate); *// Initializes Employee fields*

**this**.annualBonus = annualBonus;

}

* **Purpose**: Creates a FulltimeEmployee with specific values for both inherited and subclass-specific attributes.
* **Super Usage**:
  + Uses super(employeeId, name, department, salary, hireDate) to initialize the Employee portion of the object.
  + Must be the **first statement** in the constructor.
  + After initializing parent fields, sets annualBonus to the subclass-specific value.

1. Write getter and setter for additional attributes and override the toString() method of Employee class which will display the all details of that particular FulltimeEmployee.

Ans:

1. **Getter/Setter and toString()**

public double getAnnualBonus() { return annualBonus; }

public void setAnnualBonus(double annualBonus) { this.annualBonus = annualBonus; }

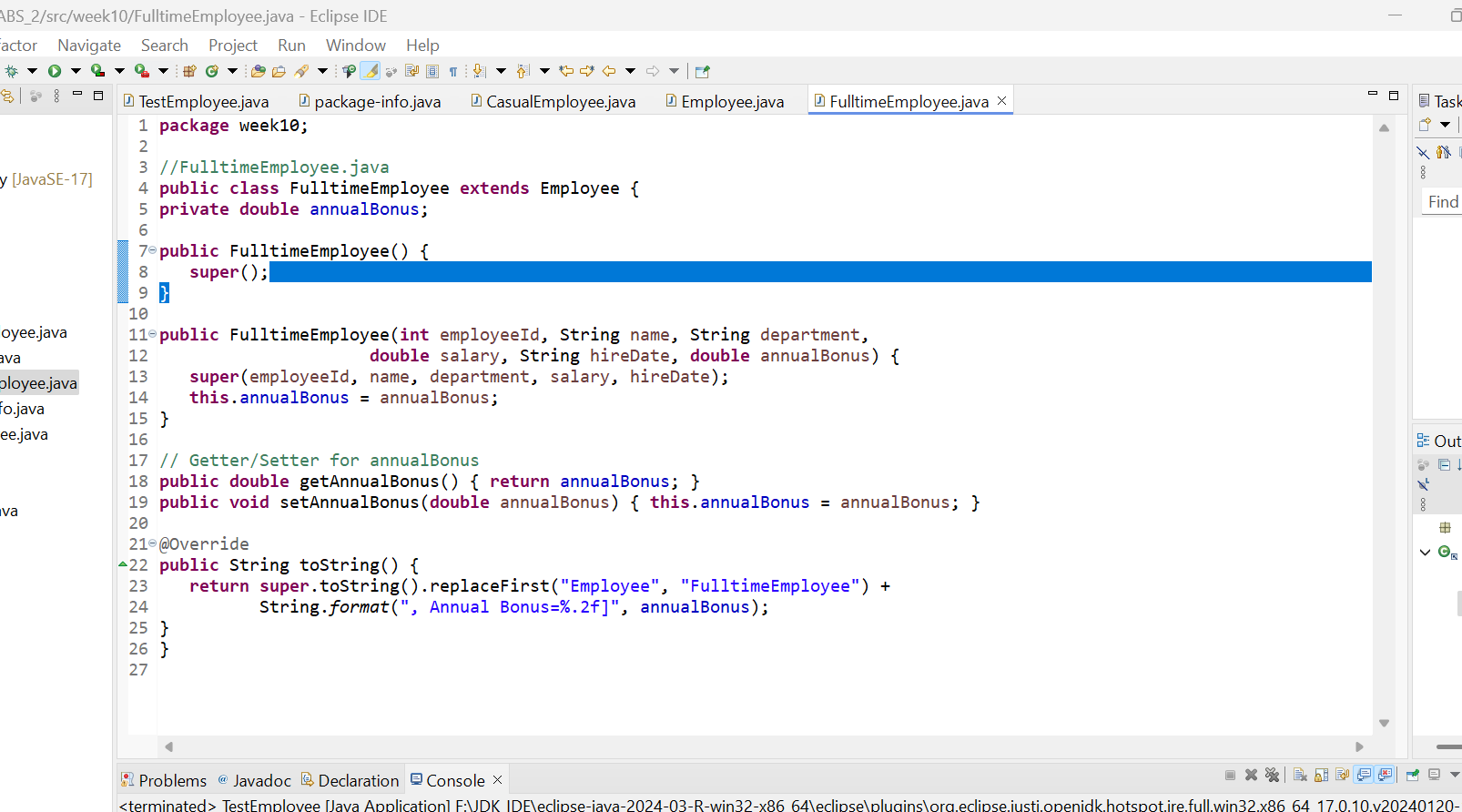
@Override

public String toString() {

return super.toString().replace("Employee", "FulltimeEmployee") +

", Annual Bonus: " + annualBonus + "]";

}



# CasualEmployee Class

Consider CasualEmployee class as a sub class of Employee

1. What is the additional attribute that a CasualEmployee class can have?

Ans:

**Additional Attributes**

* + hourlyRate: Payment per hour.
  + hoursPerWeek: Weekly hours worked.

1. Discuss the different types of constructor that this class can have and how will you use “Super” keyword?

Ans:

**Constructors and Super Usage**

* + **Default Constructor**:

public CasualEmployee() {

super();

}

* **Purpose**: Initializes a CasualEmployee with default values inherited from Employee and hourlyRate=0.0, hoursPerWeek=0.
* **Super Usage**: Calls Employee’s default constructor implicitly (even without explicit super(), but included for clarity).
  + **Parameterized Constructor**:

public CasualEmployee(int employeeId, String name, String department,

double hourlyRate, String hireDate, int hoursPerWeek) {

super(employeeId, name, department, hourlyRate \* hoursPerWeek, hireDate);

this.hourlyRate = hourlyRate;

this.hoursPerWeek = hoursPerWeek;

}

* **Purpose**: Creates a CasualEmployee with calculated salary based on hourly rate and hours worked.
* **Super Usage**:
  + Computes salary as hourlyRate \* hoursPerWeek and passes it to super() to initialize the Employee’s salary.
  + Sets subclass-specific attributes (hourlyRate, hoursPerWeek) after parent initialization.

1. Write getter and setter for additional attributes and override the toString() method of Employee class which will display the all details of that particular CasualEmployee.

Ans:

**Getters/Setters and toString()**

public double getHourlyRate() { return hourlyRate; }

public void setHourlyRate(double hourlyRate) {

this.hourlyRate = hourlyRate;

super.setSalary(hourlyRate \* hoursPerWeek); *// Updates salary*

}

public int getHoursPerWeek() { return hoursPerWeek; }

public void setHoursPerWeek(int hoursPerWeek) {

this.hoursPerWeek = hoursPerWeek;

super.setSalary(hourlyRate \* hoursPerWeek);

}

@Override

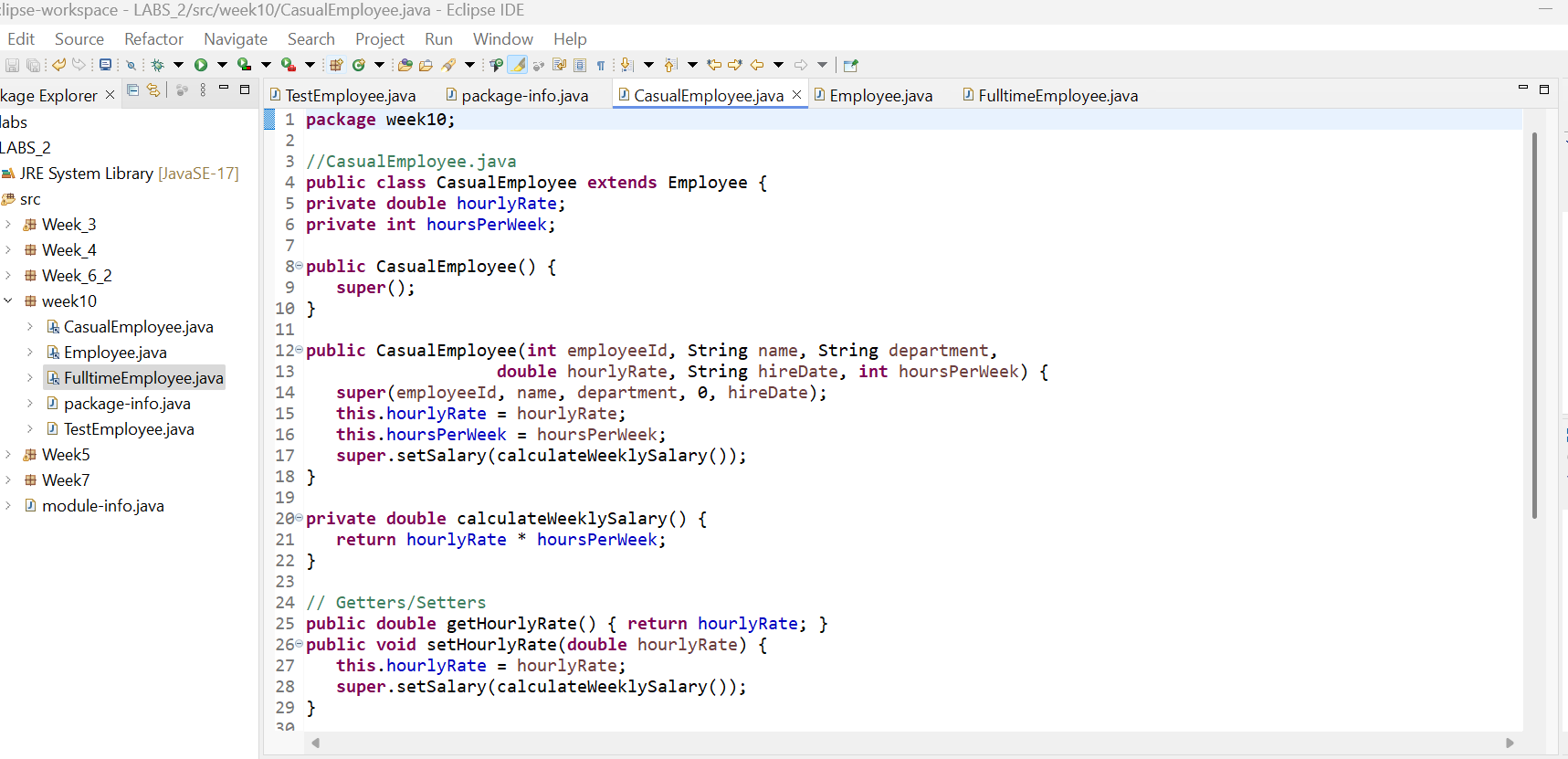
public String toString() {

return super.toString().replace("Employee", "CasualEmployee") +

", Hourly Rate: " + hourlyRate +

", Hours/Week: " + hoursPerWeek + "]";

}



# Create a test class and test all classes by creating their objects and by calling their toString() method.

public class TestEmployee {

public static void main(String[] args) {

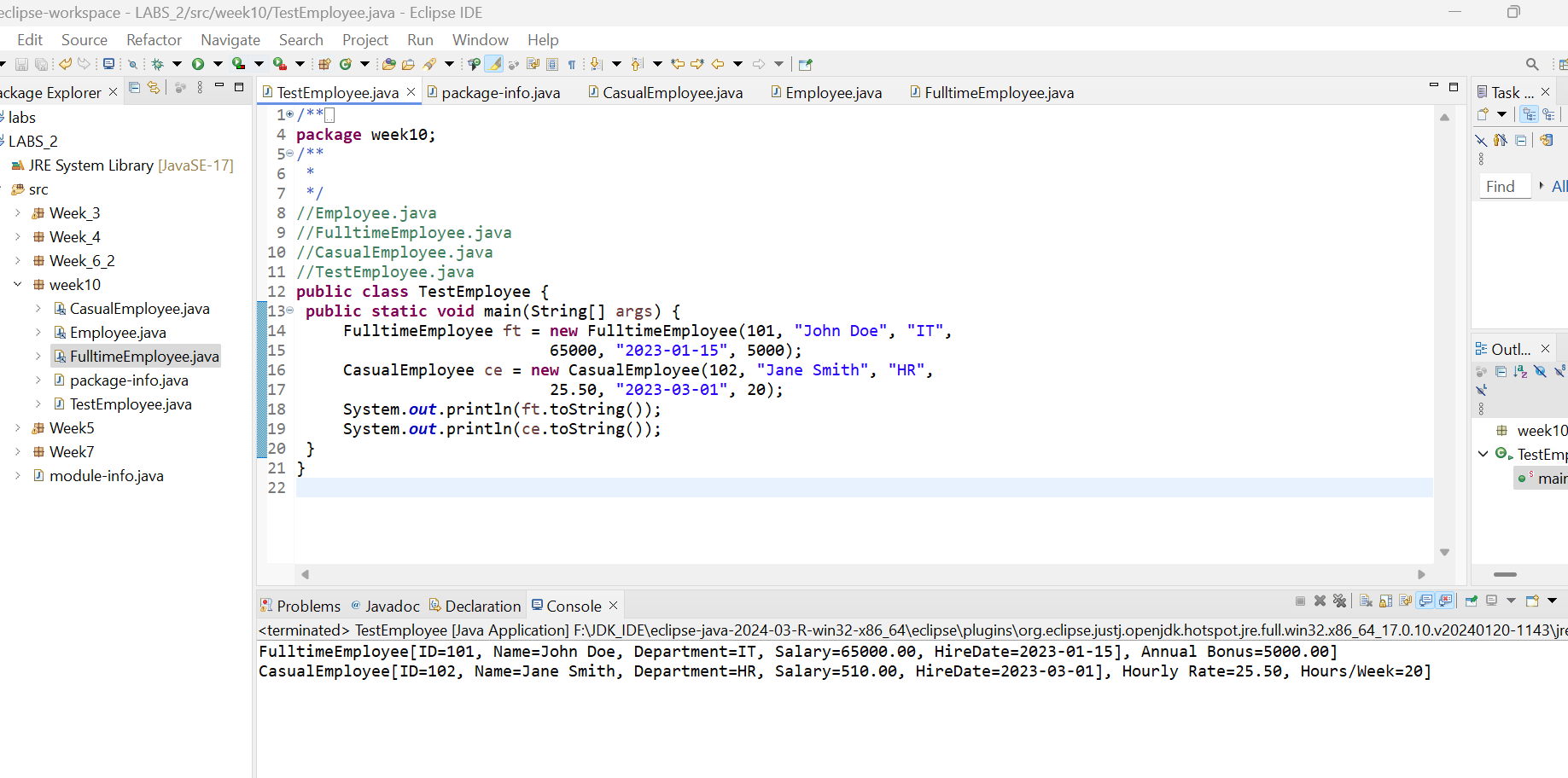
FulltimeEmployee ft = new FulltimeEmployee(101, "John Doe", "IT", 65000, "2023-01-15", 5000);

CasualEmployee ce = new CasualEmployee(102, "Jane Smith", "HR", 25.50, "2023-03-01", 20);

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

System.out.println(ce.toString()); }

}



# Create a rough class diagram for above scenario.

