

OODP workshop 10

Student Class

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

Ans: We will think of a payroll system which have an Employee class with Attribute member variables and Behaviour Methods. We will put this code in Employee.java file.

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

Ans: **Attributes of the Employee class:**

- a) name: String
- b) employeeId: int
- c) dateOfBirth: LocalDate
- d) address: String
- e) phoneNumber: String
- f) email: String
- g) department: String
- h) designation: String
- i) dateOfJoining: LocalDate
- j) salary: double
- k) paymentMethod: PaymentMethod
- l) bankDetails: BankDetails
- m) leaveBalance: int
- n) employeeType: EmployeeType
- o) taxRate: double

3. What are the methods that an Employee class can have?

Ans:

Methods of the Employee class:

- a) Constructors: Default, parameterized with essential attributes, parameterized with most attributes.
- b) Getters and Setters for each attribute.
- c) calculateSalary()
- d) applyLeave(int daysToApply)
- e) updatePersonalInfo(String address, String phoneNumber, String email)
- f) changeDepartment(String newDepartment)
- g) promoteEmployee(String newDesignation, double newSalary)
- h) terminateEmployee(LocalDate terminationDate)
- i) generatePayslip()
- j) calculateTaxDeduction()
- k) updateBankDetails(BankDetails newBankDetails)
- l) requestLeave(int daysToRequest)
- m) printEmployeeDetails()
- n) calculateNetPay()

4. How many constructors this class can have?

- Ans: There are three constructors: default, parameterized with essential attributes, parameterized with most attributes.

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

Ans: Each attribute has a getter and a setter. With 15 attributes, there will be 15 getters and 15 setters, making a total of 30 getter and setter methods.

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

Ans:

@Override

```
public String toString() {  
    return "Employee{ " +  
        "name=" + name + "\" +  
        ", employeeId=" + employeeId +  
        ", dateOfBirth=" + dateOfBirth +  
        ", address=" + address + "\" +  
        ", phoneNumber=" + phoneNumber + "\" +  
        ", email=" + email + "\" +  
        ", department=" + department + "\" +  
        ", designation=" + designation + "\" +  
        ", dateOfJoining=" + dateOfJoining +  
        ", salary=" + salary +  
        ", paymentMethod=" + paymentMethod +  
        ", bankDetails=" + bankDetails +  
        ", leaveBalance=" + leaveBalance +  
        ", employeeType=" + employeeType +  
        ", taxRate=" + taxRate +  
        '}'  
}
```

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


```

148     if (daysToRequest <= leaveBalance) {
149         applyLeave(daysToRequest);
150         System.out.println("Leave request approved. Remaining leave balance: " +
151     } else {
152         System.out.println("Insufficient leave balance. Request denied.");
153     }
154 }
155
156 public void printEmployeeDetails() {
157     System.out.println("Employee Details:");
158     System.out.println("Name: " + name);
159     System.out.println("Employee ID: " + employeeId);
160     System.out.println("Date of Birth: " + dateOfBirth);
161     System.out.println("Address: " + address);
162     System.out.println("Phone Number: " + phoneNumber);
163     System.out.println("Email: " + email);
164     System.out.println("Department: " + department);
165     System.out.println("Designation: " + designation);
166     System.out.println("Date of Joining: " + dateOfJoining);
167     System.out.println("Salary: " + salary);
168     System.out.println("Payment Method: " + paymentMethod);
169     System.out.println("Bank Details: " + bankDetails);
170     System.out.println("Leave Balance: " + leaveBalance);
171     System.out.println("Employee Type: " + employeeType);
172     System.out.println("Tax Rate: " + taxRate);
173 }
174
175 public double calculateNetPay() {
176     double grossPay = calculateSalary();
177     double taxDeduction = calculateTaxDeduction();
178     double netPay = grossPay - taxDeduction;
179     return netPay;
180 }

```

Full-Time Employee Details:

FulltimeEmployee Class

Consider FulltimeEmployee class as a sub class of Employee

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

Ans: benefits: String

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

Ans: **Constructors and usage of super:**

- a) Default constructor.
 - b) Constructor with essential attributes.
 - c) Constructor with all attributes including the additional one.
3. 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: // Getter and Setter for bonus

```

public double getBonus() {
    return bonus;
}

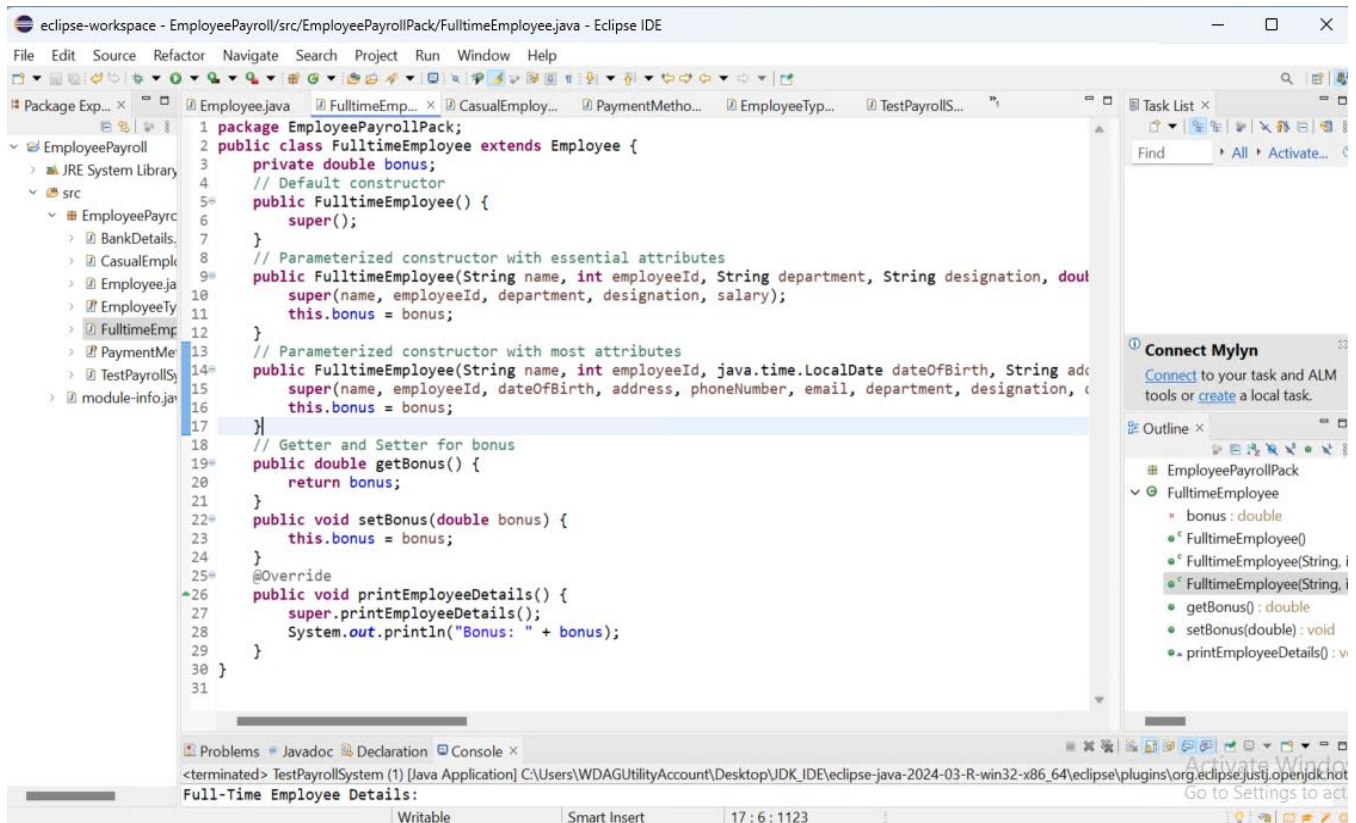
public void setBonus(double bonus) {
    this.bonus = bonus;
}

```

```

@Override
public void printEmployeeDetails() {
    super.printEmployeeDetails();
    System.out.println("Bonus: " + bonus);
}
}

```



CasualEmployee Class

Consider CasualEmployee class as a sub class of Employee

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

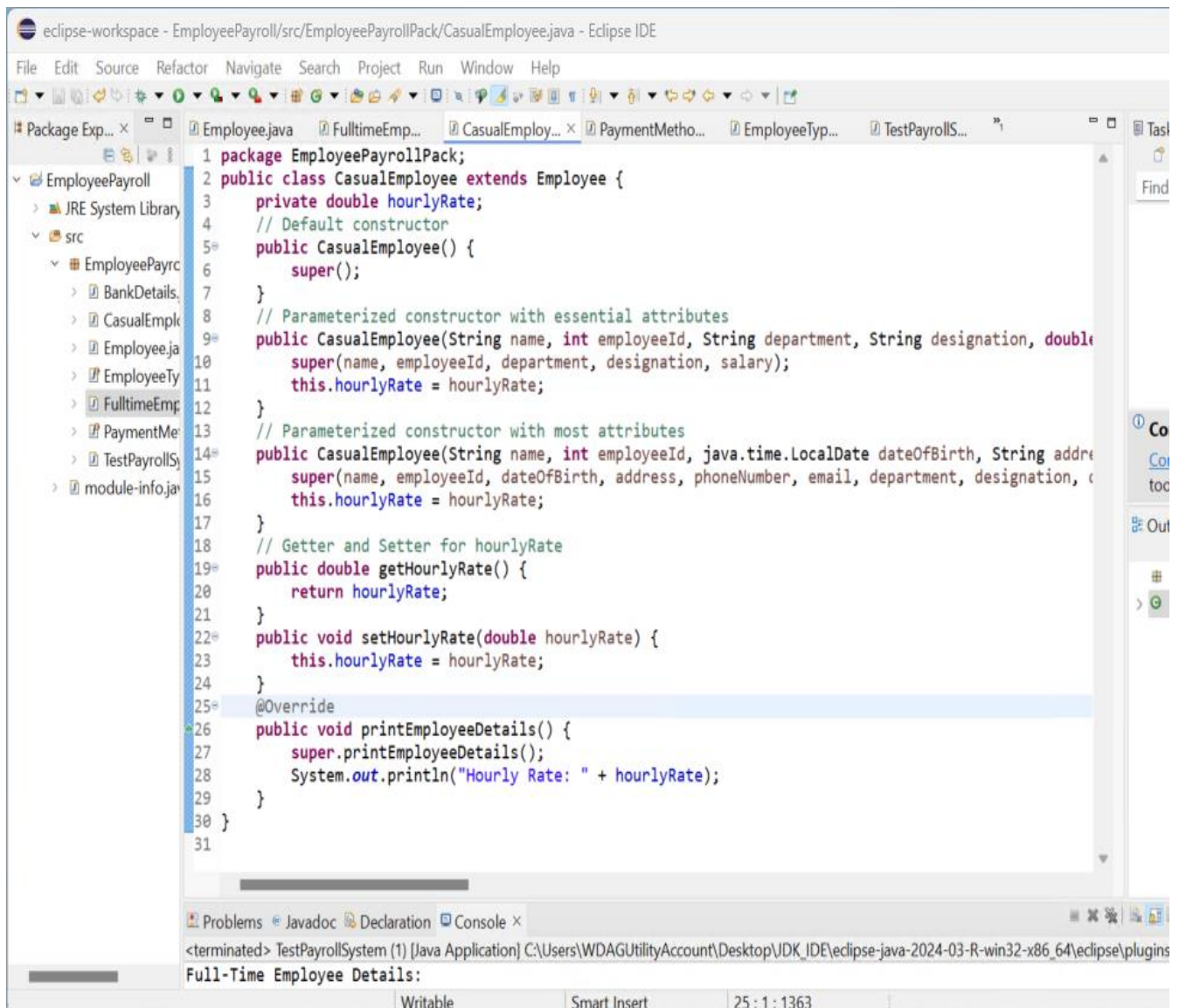
Ans: hourlyRate: double

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

Ans:

- a) Default constructor.
- b) Constructor with essential attributes.
- c) Constructor with all attributes including the additional one.

3. 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.

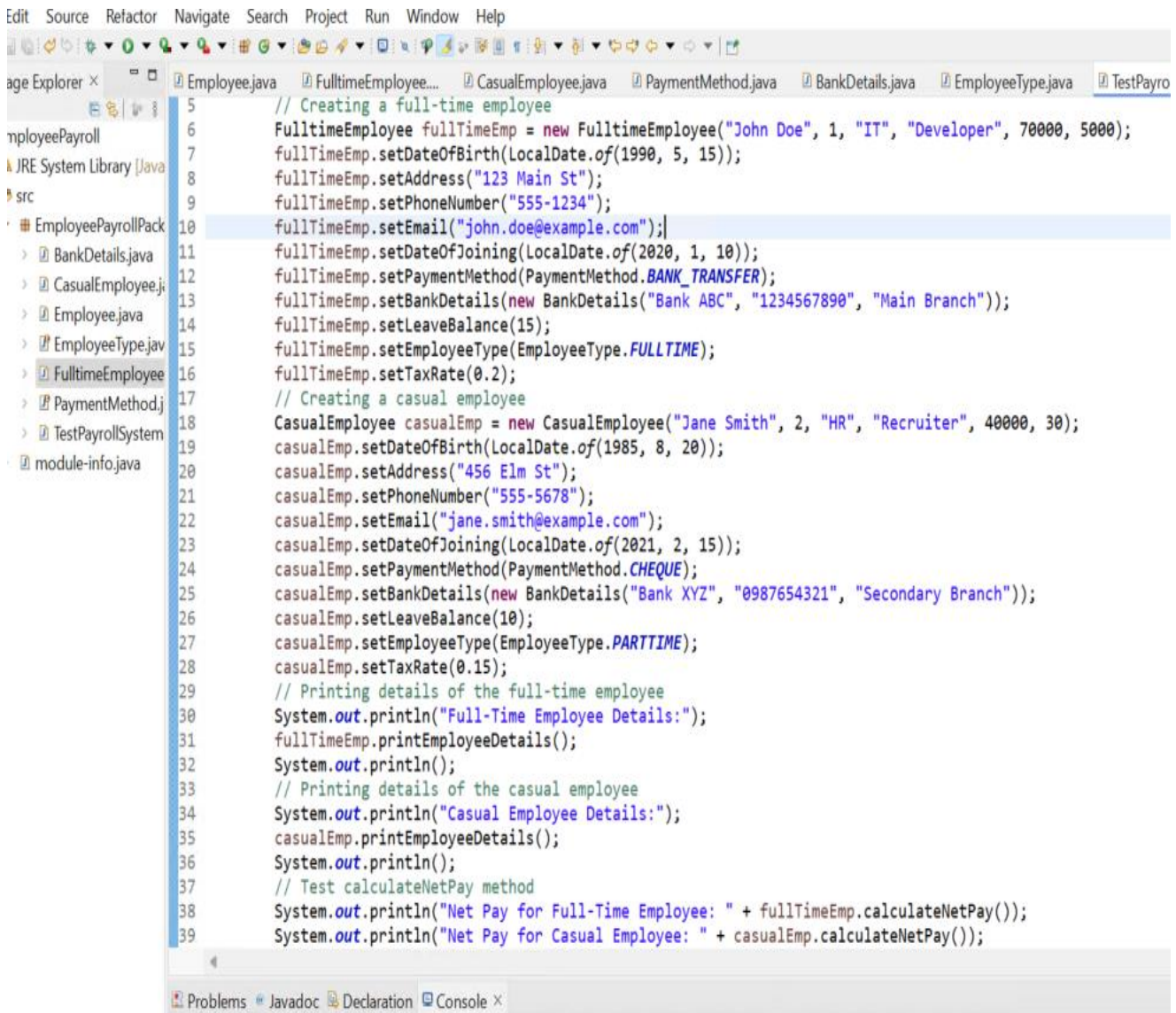


```
1 package EmployeePayrollPack;
2 public class CasualEmployee extends Employee {
3     private double hourlyRate;
4     // Default constructor
5     public CasualEmployee() {
6         super();
7     }
8     // Parameterized constructor with essential attributes
9     public CasualEmployee(String name, int employeeId, String department, String designation, double salary) {
10        super(name, employeeId, department, designation, salary);
11        this.hourlyRate = hourlyRate;
12    }
13    // Parameterized constructor with most attributes
14    public CasualEmployee(String name, int employeeId, java.time.LocalDate dateOfBirth, String address, String phone, String email, String department, String designation, double salary, double hourlyRate) {
15        super(name, employeeId, dateOfBirth, address, phone, email, department, designation, salary);
16        this.hourlyRate = hourlyRate;
17    }
18    // Getter and Setter for hourlyRate
19    public double getHourlyRate() {
20        return hourlyRate;
21    }
22    public void setHourlyRate(double hourlyRate) {
23        this.hourlyRate = hourlyRate;
24    }
25    @Override
26    public void printEmployeeDetails() {
27        super.printEmployeeDetails();
28        System.out.println("Hourly Rate: " + hourlyRate);
29    }
30 }
31
```

Full-Time Employee Details:

Writable	Smart Insert	25 : 1 : 1363
----------	--------------	---------------

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



```
5 // Creating a full-time employee
6 FulltimeEmployee fullTimeEmp = new FulltimeEmployee("John Doe", 1, "IT", "Developer", 70000, 5000);
7 fullTimeEmp.setDateOfBirth(LocalDate.of(1990, 5, 15));
8 fullTimeEmp.setAddress("123 Main St");
9 fullTimeEmp.setPhoneNumber("555-1234");
10 fullTimeEmp.setEmail("john.doe@example.com");
11 fullTimeEmp.setDateOfJoining(LocalDate.of(2020, 1, 10));
12 fullTimeEmp.setPaymentMethod(PaymentMethod.BANK_TRANSFER);
13 fullTimeEmp.setBankDetails(new BankDetails("Bank ABC", "1234567890", "Main Branch"));
14 fullTimeEmp.setLeaveBalance(15);
15 fullTimeEmp.setEmployeeType(EmployeeType.FULLTIME);
16 fullTimeEmp.setTaxRate(0.2);
17 // Creating a casual employee
18 CasualEmployee casualEmp = new CasualEmployee("Jane Smith", 2, "HR", "Recruiter", 40000, 30);
19 casualEmp.setDateOfBirth(LocalDate.of(1985, 8, 20));
20 casualEmp.setAddress("456 Elm St");
21 casualEmp.setPhoneNumber("555-5678");
22 casualEmp.setEmail("jane.smith@example.com");
23 casualEmp.setDateOfJoining(LocalDate.of(2021, 2, 15));
24 casualEmp.setPaymentMethod(PaymentMethod.CHEQUE);
25 casualEmp.setBankDetails(new BankDetails("Bank XYZ", "0987654321", "Secondary Branch"));
26 casualEmp.setLeaveBalance(10);
27 casualEmp.setEmployeeType(EmployeeType.PARTTIME);
28 casualEmp.setTaxRate(0.15);
29 // Printing details of the full-time employee
30 System.out.println("Full-Time Employee Details:");
31 fullTimeEmp.printEmployeeDetails();
32 System.out.println();
33 // Printing details of the casual employee
34 System.out.println("Casual Employee Details:");
35 casualEmp.printEmployeeDetails();
36 System.out.println();
37 // Test calculateNetPay method
38 System.out.println("Net Pay for Full-Time Employee: " + fullTimeEmp.calculateNetPay());
39 System.out.println("Net Pay for Casual Employee: " + casualEmp.calculateNetPay());
```

Create a rough class diagram for above scenario.

Ans:

Employee
<ul style="list-style-type: none">- name: String- employeeId: int- dateOfBirth: LocalDate- address: String- phoneNumber: String- email: String- department: String- designation: String- dateOfJoining: LocalDate- salary: double

- paymentMethod: PaymentMethod
- bankDetails: BankDetails
- leaveBalance: int
- employeeType: EmployeeType
- taxRate: double

+ Employee()
+ Employee(String, int, String, String, double)
+ Employee(String, int, LocalDate, String, String, String, String, String, LocalDate, double, PaymentMethod, BankDetails, int, EmployeeType, double)
+ String getName()
+ void setName(String)
+ int getEmployeeId()
+ void setEmployeeId(int)
+ LocalDate getDateOfBirth()
+ void setDateOfBirth(LocalDate)
+ String getAddress()
+ void setAddress(String)
+ String getPhoneNumber()
+ void setPhoneNumber(String)
+ String getEmail()
+ void setEmail(String)
+ String getDepartment()
+ void setDepartment(String)
+ String getDesignation()
+ void setDesignation(String)
+ LocalDate getDateOfJoining()
+ void setDateOfJoining(LocalDate)
+ double getSalary()
+ void setSalary(double)
+ PaymentMethod getPaymentMethod()
+ void setPaymentMethod(PaymentMethod)
+ BankDetails getBankDetails()
+ void setBankDetails(BankDetails)
+ int getLeaveBalance()
+ void setLeaveBalance(int)
+ EmployeeType getEmployeeType()
+ void setEmployeeType(EmployeeType)
+ double getTaxRate()

+ void setTaxRate(double)
+ double calculateSalary()
+ void applyLeave(int)
+ void updatePersonalInfo(String, String, String)
+ void changeDepartment(String)
+ void promoteEmployee(String, double)
+ void terminateEmployee(LocalDate)
+ void generatePayslip()
+ double calculateTaxDeduction()
+ void updateBankDetails(BankDetails)
+ void requestLeave(int)
+ void printEmployeeDetails()
+ double calculateNetPay()

FulltimeEmployee
- double bonus
+ FulltimeEmployee()
+ FulltimeEmployee(String, int, String, String, double, double)
+ FulltimeEmployee(String, int, LocalDate, String, String, String, String, String, String, LocalDate, double, PaymentMethod, BankDetails, int, EmployeeType, double, double)
+ double getBonus()
+ void setBonus(double)
+ void printEmployeeDetails()

CasualEmployee
- double hourlyRate
+ CasualEmployee()
+ CasualEmployee(String, int, String, String, double, double)
+ CasualEmployee(String, int, LocalDate, String, String, String, String, String, String, LocalDate, double, PaymentMethod, BankDetails, int, EmployeeType, double, double)
+ double getHourlyRate()
+ void setHourlyRate(double)
+ void printEmployeeDetails()

PaymentMethod
Constants
- BANK_TRANSFER
- CASH
- CHEQUE
- MOBILE_PAYMENT

BankDetails
- String bankName
- String accountNumber
- String branch

+ BankDetails(String, String, String)
+ String getBankName()
+ void setBankName(String)
+ String getAccountNumber()
+ void setAccountNumber(String)
+ String getBranch()
+ void setBranch(String)
+ String toString()

EmployeeType
Constants
- FULLTIME
- PARTTIME
- CONTRACT

TestPayrollSystem
+ main(String[] args)