CASE STUDY 3: Employee Payroll & Bonus System

Case Study Description

A company wants an **automated payroll system** to calculate employee salaries with optional bonuses.

The system must include:

- **Classes:** `Employee`, `Payroll`, `Manager` (inheritance)
- **Methods & validation** with error handling
- **Custom errors** for invalid salaries or missing info
- **Timer logic:** `setInterval` to simulate monthly salary disbursement
- **Closures** for bonus percentage calculation
- **Spread operator** for combining allowances
- **Arrow functions** for internal calculations
- **Type conversion** for salary validation

Step-by-Step Implementation: Employee Payroll & Bonus System

1. Create Classes

a) Employee Class

- Represents a single employee.
- **Properties needed:**
  - `name`: employee's name (string)
  - `baseSalary`: basic salary (number)
- **Responsibilities:**
  - Validate that `name` and `baseSalary` are provided

    - Ensure `baseSalary` is a positive number
- **Why:** Ensures that every employee has valid and complete information.

b) Payroll Class

- Represents the payroll system that calculates salary.
- **Responsibilities:**
    - Accept an employee object
    - Calculate total salary including bonuses (if applicable)
    - Return a salary summary (e.g., name, base salary, total salary)
- **Why:** Separates payroll logic from employee details.

c) Manager Class (Inheritance)

- A manager is a type of employee but may have **extra allowances**.
- **Extra Property:** `allowances` → array of additional payments (like housing, travel, performance incentives)
- **Responsibilities:**
    - Inherit `name` and `baseSalary` from Employee
    - Add allowances to base salary to calculate total salary
    - Optionally, override methods to include additional calculations
- **Why:** Demonstrates inheritance and method overriding.

2. Use Constructors, Methods, Static Properties, and Error Handling

Constructors

- Initialize objects with the required properties.
- Employee constructor initializes `name` and `baseSalary`.
- Manager constructor initializes `allowances` in addition to inherited properties.

Methods

- `calculateSalary` → sums base salary + allowances (for managers)
- Payroll may have methods like `generateSalarySummary` to prepare detailed statements.

Static Properties

- Can hold shared information (e.g., company name) accessible by all instances.

Error Handling

- Use `try/catch` blocks when creating employees or managers.
- Throw **custom errors** (like `ValidationError`) for:
  - Missing name
  - Invalid salary (negative or non-numeric)
  - Invalid allowances

3. Payroll Logic

a) Calculate Monthly Salary

- Base salary + allowances (if applicable)
- For regular employees, only base salary is considered
- For managers, allowances are added to base salary

b) Add Bonuses Using Closure

- Create a closure function that **stores bonus percentage**
- Apply this bonus to the calculated salary
- **Why:** Demonstrates how closures can store values and be reused.

c) Simulate Salary Disbursement Using setInterval

- Use `setInterval` to simulate **monthly salary payments**
- Each interval represents **one month**
- Print a salary summary each month showing:
  - Employee name
  - Base salary

- Allowances (if any)
- Bonus applied
- Total salary

d) Detailed Salary Statement

- Include all necessary details for clarity:
  - Name of employee
  - Base salary
  - Allowances
  - Bonus applied
  - Total salary
- Helps employees and HR track salary properly.

4. Execution Flow for Beginners

1. **Create employees** with name and base salary.
2. **Create managers** with name, base salary, and allowances.
3. **Create payroll system** (or directly use employee methods).
4. **Calculate total salary** including allowances and bonuses.
5. **Apply bonus** using a closure function.
6. **Simulate monthly disbursement** using timers.
7. **Print detailed salary statement** each month.