Java Code

```
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.*;
import java.util.function.*;
import java.io.*;

// Custom exceptions for different error scenarios
class InvalidInputException extends Exception {
    public InvalidInputException(String message) {
        super(message);
    }
}

class EmployeeNotFoundException extends Exception {
    public EmployeeNotFoundException(String message) {
        super(message);
    }
}

// Abstract Employee class with basic properties and abstract methods
abstract class Employee {
    protected String name;
    protected int id;
    protected double salary;
    protected String department;

    public Employee(String name, int id, double salary, String department) {
        this.name = name;
        this.id = id;
        this.salary = salary;
        this.department = department;
    }

    public String getName() {
        return name;
    }
```

```
    public int getId() {
        return id;
    }

    public double getSalary() {
        return salary;
    }

    public String getDepartment() {
        return department;
    }

    // Abstract method for employees to work
    public abstract void work();

    // Abstract method to display employee details
    public abstract void displayDetails();

    // Abstract method to calculate yearly bonus
    public abstract double calculateBonus();

    public void changeDepartment(String newDepartment) {
        this.department = newDepartment;
    }
}

// Manager class implementing the Employee class
class Manager extends Employee {
    private List<Employee> team;
    private static final double BONUS_PERCENTAGE = 0.10;

    public Manager(String name, int id, double salary, String department) {
        super(name, id, salary, department);
        this.team = new ArrayList<>();
    }

    // Add employee to team
    public void addEmployee(Employee employee) {
        team.add(employee);
    }
```

```
    @Override
    public void work() {
        System.out.println(name + " is managing the team and overseeing projects.");
    }

    @Override
    public void displayDetails() {
        System.out.println("Manager ID: " + id + ", Name: " + name + ", Department: " +
department + ", Salary: $" + salary);
        System.out.println("Managing team members:");
        for (Employee emp : team) {
            emp.displayDetails();
        }
    }

    @Override
    public double calculateBonus() {
        return salary * BONUS_PERCENTAGE;
    }
}

// Developer class implementing the Employee class
class Developer extends Employee {
    private String programmingLanguage;
    private static final double BONUS_PERCENTAGE = 0.15;

    public Developer(String name, int id, double salary, String department, String
programmingLanguage) {
        super(name, id, salary, department);
        this.programmingLanguage = programmingLanguage;
    }

    @Override
    public void work() {
        System.out.println(name + " is writing code in " + programmingLanguage + ".");
    }

    @Override
    public void displayDetails() {
```

```
        System.out.println("Developer ID: " + id + ", Name: " + name + ", Department: " +
department + ", Salary: $" + salary + ", Language: " + programmingLanguage);
    }

    @Override
    public double calculateBonus() {
        return salary * BONUS_PERCENTAGE;
    }

    public void switchProgrammingLanguage(String newLanguage) {
        this.programmingLanguage = newLanguage;
    }
}

// Analyst class implementing the Employee class
class Analyst extends Employee {
    private String tools;
    private static final double BONUS_PERCENTAGE = 0.12;

    public Analyst(String name, int id, double salary, String department, String tools) {
        super(name, id, salary, department);
        this.tools = tools;
    }

    @Override
    public void work() {
        System.out.println(name + " is analyzing data using tools like " + tools + ".");
    }

    @Override
    public void displayDetails() {
        System.out.println("Analyst ID: " + id + ", Name: " + name + ", Department: " +
department + ", Salary: $" + salary + ", Tools: " + tools);
    }

    @Override
    public double calculateBonus() {
        return salary * BONUS_PERCENTAGE;
    }
```

```
    public void updateTools(String newTools) {
        this.tools = newTools;
    }
}

// EmployeeManager to manage employees and perform operations like adding,
removing, and searching
class EmployeeManager {
    private List<Employee> employees;
    private Map<Integer, Employee> employeeById;

    public EmployeeManager() {
        this.employees = new ArrayList<>();
        this.employeeById = new HashMap<>();
    }

    // Add employee to the system
    public void addEmployee(Employee emp) {
        employees.add(emp);
        employeeById.put(emp.getId(), emp);
    }

    // Remove employee from the system
    public void removeEmployee(int id) throws EmployeeNotFoundException {
        Employee emp = employeeById.remove(id);
        if (emp == null) {
            throw new EmployeeNotFoundException("Employee with ID " + id + " not
found.");
        }
        employees.remove(emp);
    }

    // Search for employee by ID
    public Employee searchEmployeeById(int id) throws EmployeeNotFoundException {
        Employee emp = employeeById.get(id);
        if (emp == null) {
            throw new EmployeeNotFoundException("Employee with ID " + id + " not
found.");
        }
        return emp;
```

```
    }

    // Display all employees
    public void displayAllEmployees() {
        for (Employee emp : employees) {
            emp.displayDetails();
        }
    }

    // Sorting employees by salary in descending order using Comparator
    public void sortEmployeesBySalary() {
        employees.sort(Comparator.comparingDouble(Employee::getSalary).reversed());
    }

    // Filter employees based on department using Streams
    public List<Employee> filterEmployeesByDepartment(String department) {
        return employees.stream()
                .filter(emp -> emp.getDepartment().equals(department))
                .collect(Collectors.toList());
    }

    // Display employee bonuses
    public void displayEmployeeBonuses() {
        for (Employee emp : employees) {
            System.out.println(emp.getName() + "'s Bonus: $" + emp.calculateBonus());
        }
    }
}

// Main class to demonstrate various Java features and concepts
public class MAANGImpress {
    public static void main(String[] args) throws InterruptedException,
EmployeeNotFoundException {
        // Create EmployeeManager instance
        EmployeeManager manager = new EmployeeManager();

        // Create Employee objects (Manager, Developer, Analyst)
        Manager teamLead = new Manager("John Doe", 101, 120000, "Engineering");
        Developer dev1 = new Developer("Alice Smith", 102, 95000, "Engineering",
"Java");
```

```
        Developer dev2 = new Developer("Bob Johnson", 103, 100000, "Engineering",
"Python");
        Analyst dataAnalyst = new Analyst("Emma Wilson", 104, 85000, "Data Science",
"Excel");

        // Adding employees to EmployeeManager
        manager.addEmployee(teamLead);
        manager.addEmployee(dev1);
        manager.addEmployee(dev2);
        manager.addEmployee(dataAnalyst);

        // Add employees to Manager's team
        teamLead.addEmployee(dev1);
        teamLead.addEmployee(dev2);

        // Multi-threading simulation (e.g., employees working on tasks concurrently)
        ExecutorService executor = Executors.newFixedThreadPool(4);

        // Submitting employee tasks to the executor
        executor.submit(() -> teamLead.work());
        executor.submit(() -> dev1.work());
        executor.submit(() -> dev2.work());
        executor.submit(() -> dataAnalyst.work());

        // Waiting for all tasks to finish
        executor.shutdown();
        executor.awaitTermination(5, TimeUnit.SECONDS);

        // Display all employee details
        System.out.println("\n--- Employee Details ---");
        manager.displayAllEmployees();

        // Sort employees by salary
        manager.sortEmployeesBySalary();
        System.out.println("\n--- Employees Sorted by Salary ---");
        manager.displayAllEmployees();

        // Filter employees by department
        System.out.println("\n--- Employees in Engineering ---");
        List<Employee> engineeringEmployees =
```

```
manager.filterEmployeesByDepartment("Engineering");
        engineeringEmployees.forEach(emp -> emp.displayDetails());

        // Display employee bonuses
        manager.displayEmployeeBonuses();

        // Handle employee removal with exception handling
        try {
            manager.removeEmployee(105); // Employee not in system
        } catch (EmployeeNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
        }

        // Search employee by ID
        try {
            Employee searchedEmployee = manager.searchEmployeeById(102); // Searching
for Alice Smith
            System.out.println("\n--- Employee Found ---");
            searchedEmployee.displayDetails();
        } catch (EmployeeNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```