**Exercise 1: Lambda Expression**

**Problem:**

Create a lambda expression that multiplies two integers and returns the result. Use it to calculate the product of 5 and 6.

**Exercise 2: Functional Interface**

**Problem:**

Define a functional interface called Calculator with a method calculate(int a, int b) and use a lambda to implement addition and subtraction.

**Exercise 3: Streams API**

**Problem:**

Given a list of names, use Streams to:

* Convert them to uppercase
* Filter names starting with "A"
* Sort the list
* Print the result

**Exercise 4: Optional Class**

**Problem:**

Create a method that returns an Optional<String> with a name. Print the name if present, or print "Name not found".

**Exercise 5: Default Method in Interface**

**Problem:**

Create an interface Greeting with:

* Abstract method sayHello()
* Default method sayBye() which prints "Bye!"

Create a class that implements the interface.

**Case Study: Employee Management System (EMS)**

**Problem Statement:**

You are developing an **Employee Management System** for HR. The system should:

* Use a **lambda expression** to calculate bonus.
* Use a **functional interface** to apply a salary raise.
* Use the **Streams API** to filter, sort, and map employees.
* Use the **Optional class** to safely retrieve employee data.
* Use **default methods in interfaces** to provide reusable logic.

**Classes and Features Used**

| **Feature** | **Use Case** |
| --- | --- |
| Lambda Expression | Compute employee bonus |
| Functional Interface | Raise salary using SalaryUpdater interface |
| Streams API | Filter employees by department, sort by salary, list names, etc. |
| Optional | Search employee by ID and handle missing data |
| Default Methods | Shared utility in interface (e.g., print details or check salary range) |

**Output:**

Bonus Calculation:

Alice bonus: ₹8000.0

Bob bonus: ₹18000.0

Charlie bonus: ₹9500.0

Diana bonus: ₹15750.0

Eve bonus: ₹8700.0

Applying 10% Raise to All Employees:

Bob (IT) - ₹132000.0

Diana (IT) - ₹115500.0

Searching for Employee ID 3:

Charlie (Finance) - ₹104500.0

Searching for Employee ID 10:

Employee not found

**Case Study 2: Product Inventory System**

**Problem Statement:**

Create a product inventory system that:

* Calculates **discounts** using **lambda expressions**
* Updates prices using a **functional interface**
* Uses **Streams API** to filter products, sort by price, and get product names
* Uses **Optional** to search for a product safely
* Provides **default methods** in interface for common operations (e.g., display formatting)

**Java 8 Features Used:**

| **Feature** | **Use Case** |
| --- | --- |
| Lambda Expression | Calculate discount price |
| Functional Interface | Interface to update product price |
| Streams API | Filter products by category, sort by price, collect names |
| Optional | Search product by name safely |
| Default Methods | Interface provides default method to display product with currency |

**Sample Output:**

Discounted Prices:

Laptop discounted price: ₹58500.0

Chair discounted price: ₹3325.0

Phone discounted price: ₹27000.0

Desk discounted price: ₹7125.0

Headphones discounted price: ₹2250.0

Increasing all prices by 5%:

Electronics (sorted): [Headphones, Phone, Laptop]

Searching for 'Desk':

Desk (Furniture) - ₹7875.0

Searching for 'Tablet':

Product not found

**Case Study 3: Student Grading System**

**Problem Statement:**

Build a student grading system that:

* Uses **lambda expressions** to compute grades based on marks
* Uses a **functional interface** to update scores (curve/grace marks)
* Uses the **Streams API** to filter, sort, and list top students
* Uses the **Optional class** to search for a student
* Uses **default methods** in an interface for helper operations

**Features Breakdown:**

| **Java 8 Feature** | **Application in Grading System** |
| --- | --- |
| Lambda Expression | Calculate grades based on average marks |
| Functional Interface | Interface for updating scores with curves |
| Streams API | Filter pass/fail, sort by average, list top performers |
| Optional | Safely search for a student by ID |
| Default Methods | Common logic like isPass() and formatStudent() |

**Sample Output**

Initial Grades:

Alice got grade: B

Bob got grade: C

Charlie got grade: F

Diana got grade: A

Eve got grade: C

Applying 5 Marks Grace to Everyone:

Top Students After Grace:

Diana (Avg: 100.0)

Alice (Avg: 90.0)

Eve (Avg: 75.0)

Students Who Failed:

Charlie (Avg: 30.0)

Search for Student ID 3:

Charlie (Avg: 30.0)

Search for Student ID 10:

Student not found

**Case Study 4: Banking Account Management System**

**Problem Statement:**

Build a banking system that:

* Calculates **interest** using **lambda expressions**
* Updates **account balances** using a **functional interface**
* Uses the **Streams API** to filter accounts, sort by balance, and list customers
* Uses **Optional** to safely retrieve account details
* Uses **default methods** in an interface for reusable utility logic (e.g., isOverdraft())

**Java 8 Features Applied**

| **Java 8 Feature** | **Use Case** |
| --- | --- |
| Lambda Expression | Calculate interest |
| Functional Interface | Add balance update logic (e.g., deposit, withdrawal) |
| Streams API | Filter overdrawn accounts, sort by balance, collect account names |
| Optional | Search account by number |
| Default Methods | Provide isOverdraft() and currency format method in interface |

**Sample Output:**

Interest at 5%:

Alice earns interest: ₹250.00

Bob earns interest: ₹600.00

Charlie earns interest: ₹-75.00

Diana earns interest: ₹375.00

Eve earns interest: ₹15.00

Depositing ₹1000 to All Accounts:

Accounts sorted by balance:

Bob (AC102) - ₹13000.0

Diana (AC104) - ₹8500.0

Alice (AC101) - ₹6000.0

Eve (AC105) - ₹1300.0

Charlie (AC103) - ₹-500.0

Overdraft Accounts:

Charlie (AC103) - ₹-500.0

Search for Account 'AC103':

Charlie (AC103) - ₹-500.0

Search for Account 'AC999':

Account not found

**Case Study 5: Shopping Cart Checkout System**

**Requirements:**

* Calculate final prices using lambdas (discounts, tax).
* Use functional interface to apply cart-level pricing rules.
* Use streams to filter and sort cart items.
* Use Optional to safely retrieve an item by name.
* Provide default methods to check expensive items.

**Components:**

* CartItem (name, price, quantity)
* CartRule functional interface
* ShoppingCartService with stream and lambda logic

**Case Study 6: Invoice and Tax Calculation System**

**Requirements:**

* Use lambdas to apply tax and discount.
* Define a TaxCalculator functional interface.
* Use streams to list invoices above certain total.
* Use Optional to search invoice by ID.
* Use default method for tax formatting.

**Case Study 7: Library Book Reservation System Requirements:**

* Lambda to calculate late fee.
* Functional interface for book availability rule.
* Streams to list available books, filter by genre.
* Optional to find a book by ISBN.
* Default methods to check overdue.

**Case Study 8: Car Rental Management System**

**Requirements:**

* Lambda to calculate rental charges (per day \* days + insurance).
* Functional interface for dynamic price rule.
* Streams to filter cars by type, list top expensive rentals.
* Optional to search car by registration number.
* Default methods to check if the car is luxury.