**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.

**Solution:**

import java.util.function.BiFunction;

public class LambdaExercise {

public static void main(String[] args) {

BiFunction<Integer, Integer, Integer> multiply = (a, b) -> a \* b;

System.out.println("Product: " + multiply.apply(5, 6)); // Output: 30

}

}

**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.

**Solution:**

@FunctionalInterface

interface Calculator {

int calculate(int a, int b);

}

public class FunctionalInterfaceExercise {

public static void main(String[] args) {

Calculator add = (a, b) -> a + b;

Calculator subtract = (a, b) -> a - b;

System.out.println("Add: " + add.calculate(10, 4)); // Output: 14

System.out.println("Subtract: " + subtract.calculate(10, 4)); // Output: 6

}

}

**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

**Solution:**

import java.util.\*;

import java.util.stream.\*;

public class StreamsExercise {

public static void main(String[] args) {

List<String> names = Arrays.asList("Alice", "Bob", "Angela", "Steve", "adam");

names.stream()

.map(String::toUpperCase)

.filter(n -> n.startsWith("A"))

.sorted()

.forEach(System.out::println);

}

}

**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".

**Solution:**

import java.util.Optional;

public class OptionalExercise {

public static Optional<String> getName() {

return Optional.ofNullable("John");

}

public static void main(String[] args) {

Optional<String> name = getName();

System.out.println(name.orElse("Name not found")); // Output: John

}

}

**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.

**Solution:**

interface Greeting {

void sayHello();

default void sayBye() {

System.out.println("Bye!");

}

}

class Greeter implements Greeting {

public void sayHello() {

System.out.println("Hello!");

}

}

public class DefaultMethodExercise {

public static void main(String[] args) {

Greeter g = new Greeter();

g.sayHello(); // Output: Hello!

g.sayBye(); // Output: Bye!

}

}

**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) |

**Step-by-Step Implementation**

**1. Employee.java**

public class Employee {

private int id;

private String name;

private String department;

private double salary;

// Constructor, Getters, Setters

public Employee(int id, String name, String department, double salary) {

this.id = id;

this.name = name;

this.department = department;

this.salary = salary;

}

public int getId() { return id; }

public String getName() { return name; }

public String getDepartment() { return department; }

public double getSalary() { return salary; }

public void setSalary(double salary) {

this.salary = salary;

}

@Override

public String toString() {

return name + " (" + department + ") - ₹" + salary;

}

}

**2. SalaryUpdater.java (Functional Interface)**

@FunctionalInterface

interface SalaryUpdater {

double update(double currentSalary);

// Default method

default boolean isHighEarner(double salary) {

return salary > 1\_00\_000;

}

}

**3. EmployeeService.java**

import java.util.\*;

import java.util.function.\*;

import java.util.stream.\*;

public class EmployeeService {

private List<Employee> employees = new ArrayList<>();

public EmployeeService() {

employees.add(new Employee(1, "Alice", "HR", 80000));

employees.add(new Employee(2, "Bob", "IT", 120000));

employees.add(new Employee(3, "Charlie", "Finance", 95000));

employees.add(new Employee(4, "Diana", "IT", 105000));

employees.add(new Employee(5, "Eve", "HR", 87000));

}

// Lambda: Bonus calculator

public void printBonus() {

BiFunction<String, Double, Double> bonusCalculator = (dept, salary) -> {

if ("IT".equals(dept)) return salary \* 0.15;

else return salary \* 0.10;

};

employees.forEach(emp -> {

double bonus = bonusCalculator.apply(emp.getDepartment(), emp.getSalary());

System.out.println(emp.getName() + " bonus: ₹" + bonus);

});

}

// Functional Interface: Raise salaries

public void applyRaise(SalaryUpdater updater) {

employees.forEach(emp -> {

double newSalary = updater.update(emp.getSalary());

emp.setSalary(newSalary);

});

}

// Streams: Filter and sort

public void showITEmployeesSortedBySalary() {

employees.stream()

.filter(emp -> "IT".equals(emp.getDepartment()))

.sorted(Comparator.comparingDouble(Employee::getSalary).reversed())

.forEach(System.out::println);

}

// Optional: Find employee by ID

public void findEmployeeById(int id) {

Optional<Employee> result = employees.stream()

.filter(emp -> emp.getId() == id)

.findFirst();

String info = result.map(Employee::toString)

.orElse("Employee not found");

System.out.println(info);

}

}

**4. Main.java**

public class Main {

public static void main(String[] args) {

EmployeeService service = new EmployeeService();

System.out.println(" Bonus Calculation:");

service.printBonus();

System.out.println("\n Applying 10% Raise to All Employees:");

service.applyRaise(sal -> sal \* 1.10); // Lambda for raise

service.showITEmployeesSortedBySalary();

System.out.println("\n Searching for Employee ID 3:");

service.findEmployeeById(3);

System.out.println("\n Searching for Employee ID 10:");

service.findEmployeeById(10);

}

}

**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

**What You Learned**

| **Feature** | **How Used** |
| --- | --- |
| Lambda | Bonus calculator, salary raiser |
| Functional Interface | SalaryUpdater for raise logic |
| Streams API | Filtering, sorting, mapping employees |
| Optional | Safe retrieval of employee info |
| Default Methods | isHighEarner() in SalaryUpdater (can be used in extension) |

**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 |

**1. Product.java**

public class Product {

private String name;

private String category;

private double price;

// Constructor, Getters, Setters

public Product(String name, String category, double price) {

this.name = name;

this.category = category;

this.price = price;

}

public String getName() { return name; }

public String getCategory() { return category; }

public double getPrice() { return price; }

public void setPrice(double price) {

this.price = price;

}

@Override

public String toString() {

return name + " (" + category + ") - ₹" + price;

}

}

**2. PriceUpdater.java (Functional Interface)**

@FunctionalInterface

interface PriceUpdater {

double update(double currentPrice);

default String formatWithCurrency(double amount) {

return "₹" + String.format("%.2f", amount);

}

}

**3. InventoryService.java**

import java.util.\*;

import java.util.function.\*;

import java.util.stream.\*;

public class InventoryService {

private List<Product> products = new ArrayList<>();

public InventoryService() {

products.add(new Product("Laptop", "Electronics", 65000));

products.add(new Product("Chair", "Furniture", 3500));

products.add(new Product("Phone", "Electronics", 30000));

products.add(new Product("Desk", "Furniture", 7500));

products.add(new Product("Headphones", "Electronics", 2500));

}

// Lambda: Discount calculator

public void printDiscountedPrices() {

BiFunction<String, Double, Double> discountCalculator = (category, price) -> {

if ("Electronics".equalsIgnoreCase(category)) return price \* 0.9;

else return price \* 0.95;

};

products.forEach(p -> {

double discounted = discountCalculator.apply(p.getCategory(), p.getPrice());

System.out.println(p.getName() + " discounted price: ₹" + discounted);

});

}

// Functional Interface: Update prices

public void applyPriceUpdate(PriceUpdater updater) {

products.forEach(p -> {

double newPrice = updater.update(p.getPrice());

p.setPrice(newPrice);

});

}

// Streams: Filter + Sort + Map

public void listElectronicsSortedByPrice() {

List<String> sortedNames = products.stream()

.filter(p -> "Electronics".equalsIgnoreCase(p.getCategory()))

.sorted(Comparator.comparingDouble(Product::getPrice))

.map(Product::getName)

.collect(Collectors.toList());

System.out.println("Electronics (sorted): " + sortedNames);

}

// Optional: Safe search

public void findProductByName(String name) {

Optional<Product> result = products.stream()

.filter(p -> p.getName().equalsIgnoreCase(name))

.findFirst();

System.out.println(result.map(Product::toString)

.orElse("Product not found"));

}

}

**4. InventoryMain.java**

public class InventoryMain {

public static void main(String[] args) {

InventoryService service = new InventoryService();

System.out.println(" Discounted Prices:");

service.printDiscountedPrices();

System.out.println("\n Increasing all prices by 5%:");

service.applyPriceUpdate(price -> price \* 1.05);

service.listElectronicsSortedByPrice();

System.out.println("\n Searching for 'Desk':");

service.findProductByName("Desk");

System.out.println("\n Searching for 'Tablet':");

service.findProductByName("Tablet");

}

}

**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

**Recap of Java 8 Features Used:**

| **Feature** | **Where Used** |
| --- | --- |
| Lambda Expression | Discount calculator, price updater |
| Functional Interface | PriceUpdater for price increase |
| Streams API | Filter + sort + map product names |
| Optional | Safe product search by name |
| Default Methods | Format price using formatWithCurrency() (extendable) |

**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() |

**Step-by-Step Implementation**

**1. Student.java**

import java.util.List;

public class Student {

private int id;

private String name;

private List<Integer> marks;

public Student(int id, String name, List<Integer> marks) {

this.id = id;

this.name = name;

this.marks = marks;

}

public int getId() { return id; }

public String getName() { return name; }

public List<Integer> getMarks() { return marks; }

public double getAverage() {

return marks.stream().mapToInt(Integer::intValue).average().orElse(0);

}

public void updateMarks(List<Integer> newMarks) {

this.marks = newMarks;

}

@Override

public String toString() {

return name + " (Avg: " + getAverage() + ")";

}

}

**2. ScoreUpdater.java (Functional Interface)**

import java.util.List;

@FunctionalInterface

interface ScoreUpdater {

List<Integer> apply(List<Integer> marks);

default boolean isPass(double average) {

return average >= 40;

}

default String formatStudent(String name, double average) {

return name + " scored an average of " + String.format("%.2f", average);

}

}

**3. StudentService.java**

import java.util.\*;

import java.util.function.\*;

import java.util.stream.\*;

public class StudentService {

private List<Student> students = new ArrayList<>();

public StudentService() {

students.add(new Student(1, "Alice", Arrays.asList(75, 80, 90)));

students.add(new Student(2, "Bob", Arrays.asList(45, 50, 40)));

students.add(new Student(3, "Charlie", Arrays.asList(20, 30, 25)));

students.add(new Student(4, "Diana", Arrays.asList(85, 90, 95)));

students.add(new Student(5, "Eve", Arrays.asList(60, 70, 65)));

}

// Lambda: Grade calculator

public void printGrades() {

Function<Double, String> gradeCalculator = avg -> {

if (avg >= 85) return "A";

else if (avg >= 70) return "B";

else if (avg >= 50) return "C";

else if (avg >= 40) return "D";

else return "F";

};

students.forEach(s -> {

double avg = s.getAverage();

System.out.println(s.getName() + " got grade: " + gradeCalculator.apply(avg));

});

}

// Functional Interface: Update marks (e.g., add 5 grace marks)

public void applyGraceMarks(ScoreUpdater updater) {

students.forEach(s -> {

List<Integer> newMarks = updater.apply(s.getMarks());

s.updateMarks(newMarks);

});

}

// Streams: List top 3 students

public void showTopStudents() {

System.out.println(" Top Performers:");

students.stream()

.sorted(Comparator.comparingDouble(Student::getAverage).reversed())

.limit(3)

.forEach(System.out::println);

}

// Streams: List failed students

public void showFailedStudents() {

System.out.println(" Failed Students:");

students.stream()

.filter(s -> s.getAverage() < 40)

.forEach(System.out::println);

}

// Optional: Find student by ID

public void findStudentById(int id) {

Optional<Student> result = students.stream()

.filter(s -> s.getId() == id)

.findFirst();

System.out.println(result.map(Student::toString)

.orElse("Student not found"));

}

}

**4. Main.java**

import java.util.stream.Collectors;

public class Main {

public static void main(String[] args) {

StudentService service = new StudentService();

System.out.println(" Initial Grades:");

service.printGrades();

System.out.println("\n➕ Applying 5 Marks Grace to Everyone:");

service.applyGraceMarks(marks -> marks.stream()

.map(m -> Math.min(m + 5, 100)) // cap at 100

.collect(Collectors.toList()));

System.out.println("\n Top Students After Grace:");

service.showTopStudents();

System.out.println("\n Students Who Failed:");

service.showFailedStudents();

System.out.println("\n Search for Student ID 3:");

service.findStudentById(3);

System.out.println("\n Search for Student ID 10:");

service.findStudentById(10);

}

}

**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

**What You Practiced**

| **Feature** | **Implementation** |
| --- | --- |
| Lambda | Used to calculate grades |
| Functional Interface | Grace mark updater + helper methods like isPass() |
| Streams API | Used for filtering, sorting, mapping students |
| Optional | Used to handle safe search |
| Default Methods | Extended reusable logic in functional interface |

**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 |

**Implementation**

**1. Account.java**

public class Account {

private String accountNumber;

private String holderName;

private double balance;

public Account(String accountNumber, String holderName, double balance) {

this.accountNumber = accountNumber;

this.holderName = holderName;

this.balance = balance;

}

public String getAccountNumber() { return accountNumber; }

public String getHolderName() { return holderName; }

public double getBalance() { return balance; }

public void setBalance(double balance) {

this.balance = balance;

}

@Override

public String toString() {

return holderName + " (" + accountNumber + ") - ₹" + balance;

}

}

**2. BalanceUpdater.java (Functional Interface)**

@FunctionalInterface

interface BalanceUpdater {

double apply(double balance);

default boolean isOverdraft(double balance) {

return balance < 0;

}

default String format(double amount) {

return "₹" + String.format("%.2f", amount);

}

}

**3. BankService.java**

import java.util.\*;

import java.util.function.\*;

import java.util.stream.\*;

public class BankService {

private List<Account> accounts = new ArrayList<>();

public BankService() {

accounts.add(new Account("AC101", "Alice", 5000));

accounts.add(new Account("AC102", "Bob", 12000));

accounts.add(new Account("AC103", "Charlie", -1500)); // Overdraft

accounts.add(new Account("AC104", "Diana", 7500));

accounts.add(new Account("AC105", "Eve", 300));

}

// Lambda: Interest calculation

public void printInterest(double rate) {

BiFunction<Double, Double, Double> interestCalculator = (balance, r) -> balance \* (r / 100);

accounts.forEach(acc -> {

double interest = interestCalculator.apply(acc.getBalance(), rate);

System.out.println(acc.getHolderName() + " earns interest: ₹" + String.format("%.2f", interest));

});

}

// Functional Interface: Deposit/Withdraw

public void updateBalances(BalanceUpdater updater) {

accounts.forEach(acc -> acc.setBalance(updater.apply(acc.getBalance())));

}

// Streams: Sort and list accounts

public void showAccountsSortedByBalance() {

System.out.println(" Accounts sorted by balance:");

accounts.stream()

.sorted(Comparator.comparingDouble(Account::getBalance).reversed())

.forEach(System.out::println);

}

// Streams: Filter overdraft accounts

public void listOverdraftAccounts() {

System.out.println(" Overdraft Accounts:");

accounts.stream()

.filter(acc -> acc.getBalance() < 0)

.forEach(System.out::println);

}

// Optional: Find account by number

public void findAccountByNumber(String accNo) {

Optional<Account> result = accounts.stream()

.filter(acc -> acc.getAccountNumber().equalsIgnoreCase(accNo))

.findFirst();

System.out.println(result.map(Account::toString)

.orElse("Account not found"));

}

}

**4. BankMain.java**

public class BankMain {

public static void main(String[] args) {

BankService bank = new BankService();

System.out.println(" Interest at 5%:");

bank.printInterest(5);

System.out.println("\n Depositing ₹1000 to All Accounts:");

bank.updateBalances(bal -> bal + 1000);

bank.showAccountsSortedByBalance();

System.out.println("\n Overdraft Accounts:");

bank.listOverdraftAccounts();

System.out.println("\n Search for Account 'AC103':");

bank.findAccountByNumber("AC103");

System.out.println("\n Search for Account 'AC999':");

bank.findAccountByNumber("AC999");

}

}

**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

**Summary of Java 8 Feature Usage**

| **Feature** | **Where Used** |
| --- | --- |
| Lambda Expression | Interest calculation |
| Functional Interface | BalanceUpdater for deposit/withdrawal |
| Streams API | Sorted listing, filtering overdraft accounts |
| Optional | Account search with safe fallback |
| Default Methods | Helper for isOverdraft() and formatting |

**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

**Highlights:**

@FunctionalInterface

interface CartRule {

double apply(double total);

default boolean isExpensiveItem(double price) {

return price > 1000;

}

}

List<CartItem> cart = Arrays.asList(

new CartItem("Laptop", 50000, 1),

new CartItem("Mouse", 500, 2),

new CartItem("Keyboard", 1500, 1)

);

// Calculate subtotal using Streams

double subtotal = cart.stream()

.mapToDouble(item -> item.getPrice() \* item.getQuantity())

.sum();

// Apply discount rule

CartRule discount = total -> total > 10000 ? total \* 0.90 : total;

double finalAmount = discount.apply(subtotal);

// Search item

Optional<CartItem> found = cart.stream()

.filter(i -> i.getName().equalsIgnoreCase("Laptop"))

.findFirst();

**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.

**Highlights:**

@FunctionalInterface

interface TaxCalculator {

double calculate(double amount);

default String formatTax(double value) {

return "₹" + String.format("%.2f", value);

}

}

TaxCalculator gst18 = amount -> amount \* 0.18;

List<Invoice> invoices = ...;

invoices.stream()

.filter(inv -> inv.getTotal() > 1000)

.sorted(Comparator.comparingDouble(Invoice::getTotal).reversed())

.forEach(System.out::println);

Optional<Invoice> invoice = invoices.stream()

.filter(inv -> inv.getId() == 105)

.findFirst();

**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.

**Highlights:**

@FunctionalInterface

interface BookPolicy {

boolean isAvailable(Book b);

default boolean isOverdue(int daysBorrowed) {

return daysBorrowed > 14;

}

}

Function<Integer, Double> lateFee = days -> days > 14 ? (days - 14) \* 2.0 : 0.0;

books.stream()

.filter(Book::isAvailable)

.sorted(Comparator.comparing(Book::getTitle))

.map(Book::getTitle)

.forEach(System.out::println);

Optional<Book> found = books.stream()

.filter(b -> b.getIsbn().equals("978-123"))

.findFirst();

**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.

**Highlights:**

@FunctionalInterface

interface RentalCalculator {

double calculate(double perDay, int days);

default boolean isLuxuryCar(String model) {

return model.contains("BMW") || model.contains("Mercedes");

}

}

RentalCalculator calc = (perDay, days) -> (perDay \* days) + 1000; // insurance flat

cars.stream()

.filter(c -> c.getType().equalsIgnoreCase("SUV"))

.sorted(Comparator.comparingDouble(Car::getRentalCost).reversed())

.limit(3)

.forEach(System.out::println);

Optional<Car> result = cars.stream()

.filter(c -> c.getRegNo().equals("AP01XY1234"))

.findFirst();