

# Rajalakshmi Engineering College

Name: Shreenidhi T  
Email: 240701503@rajalakshmi.edu.in  
Roll no: 240701503  
Phone: 9150942326  
Branch: REC  
Department: CSE - Section 10  
Batch: 2028  
Degree: B.E - CSE

Scan to verify results



## 2024\_28\_III\_OOPS Using Java Lab

### REC\_Week 12\_Java\_Lamba Expressions\_PAH

Attempt : 1  
Total Mark : 40  
Marks Obtained : 37.5

#### Section 1 : COD

##### 1. Problem Statement

Rishi is working as an HR analyst in a software company. He wants to filter a list of employees based on their salary using modern Java techniques. He has a list of employee names and salaries and wants to use lambda expressions to filter those who earn more than a specific threshold.

Implement a program using lambda expressions and functional interfaces to print the names of employees whose salary is greater than or equal to 50,000.

##### ***Input Format***

The first line of input consists of an integer n, representing the number of employees.

The next n lines. Each line contains a String (employee name) and an int (salary).

### **Output Format**

The output prints the names of employees whose salary is greater than or equal to 50000, each on a new line.

If no employee found with salary greater than 50000, print: No employee found with salary  $\geq$  50000

Refer to the sample output for formatting specifications.

### **Sample Test Case**

Input: 4  
Amit 45000  
Sneha 50000  
Ravi 60000  
Priya 30000  
Output: Sneha  
Ravi

### **Answer**

```
// You are using Java
import java.util.*;
import java.util.stream.*;
class Employee {
    String name;
    int salary;
    Employee(String name, int salary) {
        this.name = name;
        this.salary = salary;
    }
}
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        sc.nextLine();
        List<Employee> employees = new ArrayList<>();
        for (int i = 0; i < n; i++) {
```

```
String name = sc.next();
int salary = sc.nextInt();
employees.add(new Employee(name, salary));
}

List<Employee> filtered = employees.stream()
    .filter(e -> e.salary >= 50000)
    .collect(Collectors.toList());
if (filtered.isEmpty()) {
    System.out.println("No employee found with salary >= 50000");
} else {
    filtered.forEach(e -> System.out.println(e.name));
}
}
```

**Status :** Correct

**Marks :** 10/10

## 2. Problem Statement

Aditya is developing a reading app that recommends books to users based on a predefined list.

Each time a user opens the app, it should supply the next book title in the list, one at a time, using a lambda expression and the Supplier functional interface.

When all books have been recommended, the list should start again from the beginning.

### **Input Format**

The first line contains an integer  $n$  — the total number of available book titles.

The next  $n$  lines each contain a book title (a string).

The next line contains an integer  $m$  — the number of times users open the app (i.e., the number of recommendations to be made).

### **Output Format**

Print the supplied book title for each recommendation, one per line.

If  $m > n$ , repeat the list from the start.

### **Sample Test Case**

Input: 3

The Alchemist

Atomic Habits

Ikigai

5

Output: The Alchemist

Atomic Habits

Ikigai

The Alchemist

Atomic Habits

### **Answer**

```
// You are using Java
import java.util.*;
import java.util.function.Supplier;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        sc.nextLine();
        List<String> books = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            books.add(sc.nextLine());
        }
        int m = sc.nextInt();

        final int[] index = {0};

        Supplier<String> nextBook = () -> {
            String book = books.get(index[0]);
            index[0] = (index[0] + 1) % n;
            return book;
        };

        for (int i = 0; i < m; i++) {
            System.out.println(nextBook.get());
        }
    }
}
```

}

**Status :** Correct

**Marks :** 10/10

### 3. Problem Statement

Emily, an analyst at a data processing firm, is tasked with cleaning up datasets to remove duplicate values from lists of integers.

Create a Java program that allows Emily to input a series of integers, with the program then utilizing a lambda expression to efficiently remove any duplicates.

#### **Input Format**

The first line of input consists of an integer N, representing the size of the array.

The second line consists of N space-separated integers, each denoting an array element.

#### **Output Format**

The output prints the array elements after removing the duplicates inside the square bracket separated by a comma and space.

Refer to the sample output for formatting specifications.

#### **Sample Test Case**

Input: 15

1 2 3 4 3 2 1 2 3 4 4 4 5 5 6

Output: [1, 2, 3, 4, 5, 6]

#### **Answer**

```
import java.util.*;
import java.util.stream.Collectors;

public class Main {
    public static void main(String[] args) {
```

```

Scanner sc = new Scanner(System.in);
int N = sc.nextInt();
int[] arr = new int[N];
for (int i = 0; i < N; i++) {
    arr[i] = sc.nextInt();
}

List<Integer> uniqueList = Arrays.stream(arr)
    .distinct()
    .boxed()
    .collect(Collectors.toList());
System.out.println(uniqueList);
sc.close();
}
}

```

**Status :** Correct

**Marks :** 10/10

#### 4. Problem Statement

Sneha is developing a feature for an e-commerce application that helps display product details after applying a seasonal discount.

She decides to use lambda expressions with the Consumer functional interface to print each product's name, original price, and discounted price neatly.

The program should:

Accept a list of product names and their prices. Apply a 15% discount on all products. Use a Consumer lambda expression to display the details in a formatted manner.

##### **Input Format**

The first line of input consists of an integer  $n$ , representing the number of products.

The next  $n$  lines each contain a String (product name) and a double (price) separated by a space.

### **Output Format**

For each product, print the details in the format:

Product: <name>, Original Price: <price>, Discounted Price: <discounted price>

If there are no products, print:

No products available

### **Sample Test Case**

Input: 1

Phone 60000

Output: Product: Phone, Original Price: 60000.0, Discounted Price: 51000.0

### **Answer**

```
// You are using Java
```

```
import java.util.*;
```

```
import java.util.function.Consumer;
```

```
class Product {
```

```
    String name;
```

```
    double price;
```

```
    Product(String name, double price) {
```

```
        this.name = name;
```

```
        this.price = price;
```

```
    }
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
```

```
        int n = Integer.parseInt(sc.nextLine().trim());
```

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

```
        for (int i = 0; i < n; i++) {
```

```
            String line = sc.nextLine().trim();
```

```
            String[] parts = line.split(" ");
```

```
            String pname = parts[0];
```

```
            double pprice = Double.parseDouble(parts[1]);
```

```
            products.add(new Product(pname, pprice));
```

```
}  
if (products.isEmpty()) {  
    System.out.println("No products available");  
} else {  
    Consumer<Product> display = p -> {  
        double discountPrice = p.price * 0.85;  
        System.out.println("Product: " + p.name +  
            ", Original Price: " + p.price +  
            ", Discounted Price: " + discountPrice);  
    };  
    products.forEach(display);  
}  
sc.close();  
}
```

**Status :** Partially correct

**Marks :** 7.5/10