# Day 7

1)

package day7;

import java.util.ArrayList;

import java.util.Scanner;

public class StudentList {

    public static void main(String[] args) {

        // Create an ArrayList to store student names

        ArrayList<String> studentNames = new ArrayList<>();

        // Add student names to the list

        studentNames.add("John");

        studentNames.add("Jane");

        studentNames.add("Alice");

        studentNames.add("Bob");

        studentNames.add("Michael");

        // Display the list of student names

        System.out.println("List of Students: " + studentNames);

        // Get the name to search from the user

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the name to search: ");

        String nameToSearch = scanner.nextLine();

        // Check if the name exists in the list

        if (studentNames.contains(nameToSearch)) {

            System.out.println(nameToSearch + " exists in the list.");

        } else {

            System.out.println(nameToSearch + " does not exist in the list.");

        }

        scanner.close();

    }

}

Enter the name to search: John

John exists in the list.

Enter the name to search: Mary

Mary does not exist in the list.

2)

package day7;

import java.util.HashSet;

import java.util.Scanner;

class Product {

    private String productId;

    private String productName;

    // Constructor

    public Product(String productId, String productName) {

        this.productId = productId;

        this.productName = productName;

    }

    // Getters

    public String getProductId() {

        return productId;

    }

    public String getProductName() {

        return productName;

    }

    // Override equals and hashCode for proper comparison in HashSet

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true;

        if (obj == null || getClass() != obj.getClass()) return false;

        Product product = (Product) obj;

        return productId.equals(product.productId);

    }

    @Override

    public int hashCode() {

        return productId.hashCode();

    }

    // Override toString for better display

    @Override

    public String toString() {

        return "ProductId: " + productId + ", ProductName: " + productName;

    }

}

public class ProductHashSet {

    public static void main(String[] args) {

        HashSet<Product> products = new HashSet<>();

        // Adding products to the HashSet

        products.add(new Product("P001", "Maruti 800"));

        products.add(new Product("P002", "Maruti Zen"));

        products.add(new Product("P003", "Maruti Dezire"));

        products.add(new Product("P004", "Maruti Alto"));

        Scanner scanner = new Scanner(System.in);

        while (true) {

            System.out.println("\nProduct Management System:");

            System.out.println("1. Display all products");

            System.out.println("2. Search for a product by ID");

            System.out.println("3. Remove a product by ID");

            System.out.println("4. Exit");

            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();

            scanner.nextLine(); // Consume newline

            switch (choice) {

                case 1:

                    System.out.println("\nProducts in the HashSet:");

                    for (Product product : products) {

                        System.out.println(product);

                    }

                    break;

                case 2:

                    System.out.print("Enter the Product ID to search: ");

                    String searchId = scanner.nextLine();

                    boolean found = false;

                    for (Product product : products) {

                        if (product.getProductId().equals(searchId)) {

                            System.out.println("Product found: " + product);

                            found = true;

                            break;

                        }

                    }

                    if (!found) {

                        System.out.println("Product not found.");

                    }

                    break;

                case 3:

                    System.out.print("Enter the Product ID to remove: ");

                    String removeId = scanner.nextLine();

                    Product toRemove = null;

                    for (Product product : products) {

                        if (product.getProductId().equals(removeId)) {

                            toRemove = product;

                            break;

                        }

                    }

                    if (toRemove != null) {

                        products.remove(toRemove);

                        System.out.println("Product removed successfully.");

                    } else {

                        System.out.println("Product not found.");

                    }

                    break;

                case 4:

                    System.out.println("Exiting the application. Goodbye!");

                    scanner.close();

                    return;

                default:

                    System.out.println("Invalid choice. Please try again.");

            }

        }

    }

}

Product Management System:

1. Display all products

2. Search for a product by ID

3. Remove a product by ID

4. Exit

Enter your choice: 1

Products in the HashSet:

ProductId: P004, ProductName: Maruti Alto

ProductId: P001, ProductName: Maruti 800

ProductId: P003, ProductName: Maruti Dezire

ProductId: P002, ProductName: Maruti Zen

Product Management System:

1. Display all products

2. Search for a product by ID

3. Remove a product by ID

4. Exit

Enter your choice: 2

Enter the Product ID to search: P004

Product found: ProductId: P004, ProductName: Maruti Alto

Product Management System:

1. Display all products

2. Search for a product by ID

3. Remove a product by ID

4. Exit

Enter your choice: 3

Enter the Product ID to remove: P004

Product removed successfully.

Product Management System:

1. Display all products

2. Search for a product by ID

3. Remove a product by ID

4. Exit

Enter your choice: 4

Exiting the application. Goodbye!

3)

package day7;

import java.util.LinkedList;

import java.util.Iterator;

import java.util.ListIterator;

// Employee class

class Employee {

    private int employeeNo;

    private String employeeName;

    private String address;

    // Constructor to initialize Employee details

    public Employee(int employeeNo, String employeeName, String address) {

        this.employeeNo = employeeNo;

        this.employeeName = employeeName;

        this.address = address;

    }

    @Override

    public String toString() {

        return "EmployeeNo: " + employeeNo + ", EmployeeName: " + employeeName + ", Address: " + address;

    }

}

public class EmployeeLinkedList {

    public static void main(String[] args) {

        LinkedList<Employee> employees = new LinkedList<>();

        // Adding employee details to LinkedList

        addInput(employees, new Employee(101, "John Doe", "123 Elm Street"));

        addInput(employees, new Employee(102, "Jane Smith", "456 Oak Avenue"));

        addInput(employees, new Employee(103, "Alice Johnson", "789 Pine Road"));

        addInput(employees, new Employee(104, "Bob Brown", "321 Maple Lane"));

        // Displaying employee details in forward and reverse order

        display(employees);

    }

    // Method to add employee details to LinkedList

    public static void addInput(LinkedList<Employee> employees, Employee employee) {

        employees.add(employee);

    }

    // Method to display employee details

    public static void display(LinkedList<Employee> employees) {

        System.out.println("\nDisplaying Employees in Forward Order:");

        Iterator<Employee> iterator = employees.iterator();

        while (iterator.hasNext()) {

            System.out.println(iterator.next());

        }

        System.out.println("\nDisplaying Employees in Reverse Order:");

        ListIterator<Employee> listIterator = employees.listIterator(employees.size());

        while (listIterator.hasPrevious()) {

            System.out.println(listIterator.previous());

        }

    }

}

Displaying Employees in Forward Order:

EmployeeNo: 101, EmployeeName: John Doe, Address: 123 Elm Street

EmployeeNo: 102, EmployeeName: Jane Smith, Address: 456 Oak Avenue

EmployeeNo: 103, EmployeeName: Alice Johnson, Address: 789 Pine Road

EmployeeNo: 104, EmployeeName: Bob Brown, Address: 321 Maple Lane

Displaying Employees in Reverse Order:

EmployeeNo: 104, EmployeeName: Bob Brown, Address: 321 Maple Lane

EmployeeNo: 103, EmployeeName: Alice Johnson, Address: 789 Pine Road

EmployeeNo: 102, EmployeeName: Jane Smith, Address: 456 Oak Avenue

EmployeeNo: 101, EmployeeName: John Doe, Address: 123 Elm Street

4)

package day7;

import java.util.HashMap;

import java.util.Scanner;

public class PhoneBook {

    public static void main(String[] args) {

        HashMap<String, String> phoneBook = new HashMap<>();

        Scanner scanner = new Scanner(System.in);

        while (true) {

            System.out.println("\nPhone Book Menu:");

            System.out.println("1. Add new phone book entry");

            System.out.println("2. Search Phone Number");

            System.out.println("3. Quit");

            System.out.print("Enter your choice: ");

            int choice = scanner.nextInt();

            scanner.nextLine(); // Consume newline

            switch (choice) {

                case 1:

                    System.out.print("Enter Name: ");

                    String name = scanner.nextLine();

                    System.out.print("Enter Phone Number: ");

                    String phoneNumber = scanner.nextLine();

                    phoneBook.put(name, phoneNumber);

                    System.out.println("Entry added successfully.");

                    break;

                case 2:

                    System.out.print("Enter Name to search: ");

                    String searchName = scanner.nextLine();

                    if (phoneBook.containsKey(searchName)) {

                        System.out.println("Phone Number: " + phoneBook.get(searchName));

                    } else {

                        System.out.println("Name not found in the phone book.");

                    }

                    break;

                case 3:

                    System.out.println("Exiting the program. Goodbye!");

                    scanner.close();

                    return;

                default:

                    System.out.println("Invalid choice. Please try again.");

            }

        }

    }

}

Phone Book Menu:

1. Add new phone book entry

2. Search Phone Number

3. Quit

Enter your choice: 2

Enter Name to search: basha

Name not found in the phone book.

Phone Book Menu:

1. Add new phone book entry

2. Search Phone Number

3. Quit

Enter your choice: 1

Enter Name: suhail

Enter Phone Number: 7094030232

Entry added successfully.

Phone Book Menu:

1. Add new phone book entry

2. Search Phone Number

3. Quit

Enter your choice: 2

Enter Name to search: suhail

Phone Number: 7094030232

Phone Book Menu:

1. Add new phone book entry

2. Search Phone Number

3. Quit

Enter your choice: 3

Exiting the program. Goodbye!

5)

package day7;

import java.time.LocalDate;

import java.util.\*;

class Book implements Comparable<Book> {

    private int bookId;

    private String title;

    private double price;

    private LocalDate publicationDate;

    private String author;

    public Book(int bookId, String title, double price, LocalDate publicationDate, String author) {

        this.bookId = bookId;

        this.title = title;

        this.price = price;

        this.publicationDate = publicationDate;

        this.author = author;

    }

    public int getBookId() {

        return bookId;

    }

    public String getTitle() {

        return title;

    }

    public double getPrice() {

        return price;

    }

    public LocalDate getPublicationDate() {

        return publicationDate;

    }

    public String getAuthor() {

        return author;

    }

    @Override

    public String toString() {

        return "BookId: " + bookId + ", Title: " + title + ", Price: " + price + ", Publication Date: " + publicationDate + ", Author: " + author;

    }

    @Override

    public int hashCode() {

        return Objects.hash(bookId, title, price, publicationDate, author);

    }

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true;

        if (obj == null || getClass() != obj.getClass()) return false;

        Book book = (Book) obj;

        return bookId == book.bookId && Double.compare(book.price, price) == 0 && Objects.equals(title, book.title) && Objects.equals(publicationDate, book.publicationDate) && Objects.equals(author, book.author);

    }

    @Override

    public int compareTo(Book other) {

        return this.author.compareTo(other.author);

    }

}

public class BookTreeSet {

    public static void main(String[] args) {

        TreeSet<Book> books = new TreeSet<>();

        books.add(new Book(1, "Java Programming", 599.99, LocalDate.of(2020, 5, 15), "John Doe"));

        books.add(new Book(2, "Python Basics", 499.99, LocalDate.of(2018, 8, 10), "Alice Smith"));

        books.add(new Book(3, "Data Structures", 699.99, LocalDate.of(2021, 3, 20), "Bob Brown"));

        books.add(new Book(4, "Algorithms", 799.99, LocalDate.of(2019, 11, 25), "Charlie Johnson"));

        books.add(new Book(5, "Web Development", 299.99, LocalDate.of(2022, 1, 5), "David White"));

        System.out.println("\nBooks sorted by Author (Natural Ordering):");

        for (Book book : books) {

            System.out.println(book);

        }

        System.out.println("\nBooks sorted by Publication Date (Descending Order):");

        books.stream()

                .sorted(Comparator.comparing(Book::getPublicationDate).reversed())

                .forEach(System.out::println);

        System.out.println("\nBooks sorted by Title (Ascending Order):");

        books.stream()

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

                .forEach(System.out::println);

        System.out.println("\nBooks sorted by Book ID (Descending) and Publication Date (Ascending):");

        books.stream()

                .sorted(Comparator.comparing(Book::getBookId).reversed()

                        .thenComparing(Book::getPublicationDate))

                .forEach(System.out::println);

    }

}

Books sorted by Author (Natural Ordering):

BookId: 2, Title: Python Basics, Price: 499.99, Publication Date: 2018-08-10, Author: Alice Smith

BookId: 3, Title: Data Structures, Price: 699.99, Publication Date: 2021-03-20, Author: Bob Brown

BookId: 4, Title: Algorithms, Price: 799.99, Publication Date: 2019-11-25, Author: Charlie Johnson

BookId: 5, Title: Web Development, Price: 299.99, Publication Date: 2022-01-05, Author: David White

BookId: 1, Title: Java Programming, Price: 599.99, Publication Date: 2020-05-15, Author: John Doe

Books sorted by Publication Date (Descending Order):

BookId: 5, Title: Web Development, Price: 299.99, Publication Date: 2022-01-05, Author: David White

BookId: 3, Title: Data Structures, Price: 699.99, Publication Date: 2021-03-20, Author: Bob Brown

BookId: 1, Title: Java Programming, Price: 599.99, Publication Date: 2020-05-15, Author: John Doe

BookId: 4, Title: Algorithms, Price: 799.99, Publication Date: 2019-11-25, Author: Charlie Johnson

BookId: 2, Title: Python Basics, Price: 499.99, Publication Date: 2018-08-10, Author: Alice Smith

Books sorted by Title (Ascending Order):

BookId: 4, Title: Algorithms, Price: 799.99, Publication Date: 2019-11-25, Author: Charlie Johnson

BookId: 3, Title: Data Structures, Price: 699.99, Publication Date: 2021-03-20, Author: Bob Brown

BookId: 1, Title: Java Programming, Price: 599.99, Publication Date: 2020-05-15, Author: John Doe

BookId: 2, Title: Python Basics, Price: 499.99, Publication Date: 2018-08-10, Author: Alice Smith

BookId: 5, Title: Web Development, Price: 299.99, Publication Date: 2022-01-05, Author: David White

Books sorted by Book ID (Descending) and Publication Date (Ascending):

BookId: 5, Title: Web Development, Price: 299.99, Publication Date: 2022-01-05, Author: David White

BookId: 4, Title: Algorithms, Price: 799.99, Publication Date: 2019-11-25, Author: Charlie Johnson

BookId: 3, Title: Data Structures, Price: 699.99, Publication Date: 2021-03-20, Author: Bob Brown

BookId: 2, Title: Python Basics, Price: 499.99, Publication Date: 2018-08-10, Author: Alice Smith

BookId: 1, Title: Java Programming, Price: 599.99, Publication Date: 2020-05-15, Author: John Doe

6)

package day7;

import java.util.\*;

import java.util.stream.\*;

class Person implements Comparable<Person> {

    private int id;

    private String name;

    private int age;

    private double salary;

    public Person(int id, String name, int age, double salary) {

        this.id = id;

        this.name = name;

        this.age = age;

        this.salary = salary;

    }

    // Getters

    public int getId() { return id; }

    public String getName() { return name; }

    public int getAge() { return age; }

    public double getSalary() { return salary; }

    @Override

    public String toString() {

        return "Person{id=" + id + ", name='" + name + "', age=" + age + ", salary=" + salary + '}';

    }

    @Override

    public boolean equals(Object obj) {

        if (this == obj) return true;

        if (obj == null || getClass() != obj.getClass()) return false;

        Person person = (Person) obj;

        return id == person.id;

    }

    @Override

    public int hashCode() {

        return Objects.hash(id);

    }

    @Override

    public int compareTo(Person other) {

        return Integer.compare(this.id, other.id);  // Sort by id in ascending order

    }

    public static void main(String[] args) {

        Set<Person> persons = new HashSet<>();

        // Accepting person details

        persons.add(new Person(1, "John", 25, 50000));

        persons.add(new Person(2, "Jane", 30, 60000));

        persons.add(new Person(3, "Jack", 20, 55000));

        persons.add(new Person(4, "Jerry", 35, 70000));

        persons.add(new Person(5, "Jessica", 28, 75000));

        persons.add(new Person(6, "Jill", 23, 65000));

        // 6. Print all the persons details using Streams and Method Reference

        System.out.println("All Persons:");

        persons.stream().forEach(System.out::println);

        // Print all the persons details by sorting the id in ascending order using Comparable and Streams

        System.out.println("\nPersons sorted by ID:");

        persons.stream()

               .sorted()

               .forEach(System.out::println);

        // Print all the persons details by sorting the name in ascending order using Comparator and Streams

        System.out.println("\nPersons sorted by Name (ascending):");

        persons.stream()

               .sorted(Comparator.comparing(Person::getName))

               .forEach(System.out::println);

        // Print all the persons details by sorting the names in descending order using Comparator and Streams

        System.out.println("\nPersons sorted by Name (descending):");

        persons.stream()

               .sorted(Comparator.comparing(Person::getName).reversed())

               .forEach(System.out::println);

        // Print all the persons details whose Name start with 'J' using Streams

        System.out.println("\nPersons whose name starts with J:");

        persons.stream()

               .filter(p -> p.getName().startsWith("J"))

               .forEach(System.out::println);

        // Print the count number of persons using Streams

        long count = persons.stream().count();

        System.out.println("\nNumber of persons: " + count);

        // Print the Max salary among all persons using Streams

        OptionalDouble maxSalary = persons.stream()

                                          .mapToDouble(Person::getSalary)

                                          .max();

        maxSalary.ifPresent(s -> System.out.println("\nMax salary: " + s));

        // Print the Min salary among all persons using Streams

        OptionalDouble minSalary = persons.stream()

                                          .mapToDouble(Person::getSalary)

                                          .min();

        minSalary.ifPresent(s -> System.out.println("\nMin salary: " + s));

        // Print the average of all salaries using Stream

        OptionalDouble avgSalary = persons.stream()

                                          .mapToDouble(Person::getSalary)

                                          .average();

        avgSalary.ifPresent(s -> System.out.println("\nAverage salary: " + s));

        // Print the sum of all salaries using Streams

        double totalSalary = persons.stream()

                                    .mapToDouble(Person::getSalary)

                                    .sum();

        System.out.println("\nSum of all salaries: " + totalSalary);

        // Print the First Person whose Name start with J using Streams - filter and findFirst method

        System.out.println("\nFirst person whose name starts with 'J':");

        persons.stream()

               .filter(p -> p.getName().startsWith("J"))

               .findFirst()

               .ifPresent(System.out::println);

        // Check whether all the persons age is greater than 10 using Streams – allMatch method

        boolean allAbove10 = persons.stream()

                                    .allMatch(p -> p.getAge() > 10);

        System.out.println("\nAll persons are older than 10: " + allAbove10);

        // Print the average of all salaries using Streams and Collector

        Double averageSalary = persons.stream()

                                      .collect(Collectors.averagingDouble(Person::getSalary));

        System.out.println("\nAverage salary (using Collector): " + averageSalary);

        // Print all the persons details group by salary using Streams and Collectors

        System.out.println("\nGroup by salary:");

        Map<Double, List<Person>> groupBySalary = persons.stream()

                                                          .collect(Collectors.groupingBy(Person::getSalary));

        groupBySalary.forEach((salary, people) -> {

            System.out.println("Salary: " + salary);

            people.forEach(System.out::println);

        });

        // Print all the names after joining whose age is greater than 18 using Streams and Collectors

        System.out.println("\nNames of persons older than 18:");

        String joinedNames = persons.stream()

                                     .filter(p -> p.getAge() > 18)

                                     .map(Person::getName)

                                     .collect(Collectors.joining(", "));

        System.out.println(joinedNames);

        // Check whether all the persons age is greater than 50 using Streams – noneMatch method

        boolean noneAbove50 = persons.stream()

                                     .noneMatch(p -> p.getAge() > 50);

        System.out.println("\nNo persons are older than 50: " + noneAbove50);

    }

}

All Persons:

Person{id=1, name='John', age=25, salary=50000.0}

Person{id=2, name='Jane', age=30, salary=60000.0}

Person{id=3, name='Jack', age=20, salary=55000.0}

Person{id=4, name='Jerry', age=35, salary=70000.0}

Person{id=5, name='Jessica', age=28, salary=75000.0}

Person{id=6, name='Jill', age=23, salary=65000.0}

Persons sorted by ID:

Person{id=1, name='John', age=25, salary=50000.0}

Person{id=2, name='Jane', age=30, salary=60000.0}

Person{id=3, name='Jack', age=20, salary=55000.0}

Person{id=4, name='Jerry', age=35, salary=70000.0}

Person{id=5, name='Jessica', age=28, salary=75000.0}

Person{id=6, name='Jill', age=23, salary=65000.0}

Persons sorted by Name (ascending):

Person{id=3, name='Jack', age=20, salary=55000.0}

Person{id=2, name='Jane', age=30, salary=60000.0}

Person{id=4, name='Jerry', age=35, salary=70000.0}

Person{id=5, name='Jessica', age=28, salary=75000.0}

Person{id=6, name='Jill', age=23, salary=65000.0}

Person{id=1, name='John', age=25, salary=50000.0}

Persons sorted by Name (descending):

Person{id=1, name='John', age=25, salary=50000.0}

Person{id=6, name='Jill', age=23, salary=65000.0}

Person{id=5, name='Jessica', age=28, salary=75000.0}

Person{id=4, name='Jerry', age=35, salary=70000.0}

Person{id=2, name='Jane', age=30, salary=60000.0}

Person{id=3, name='Jack', age=20, salary=55000.0}

Persons whose name starts with J:

Person{id=1, name='John', age=25, salary=50000.0}

Person{id=2, name='Jane', age=30, salary=60000.0}

Person{id=3, name='Jack', age=20, salary=55000.0}

Person{id=4, name='Jerry', age=35, salary=70000.0}

Person{id=5, name='Jessica', age=28, salary=75000.0}

Person{id=6, name='Jill', age=23, salary=65000.0}

Number of persons: 6

Max salary: 75000.0

Min salary: 50000.0

Average salary: 62500.0

Sum of all salaries: 375000.0

First person whose name starts with 'J':

Person{id=1, name='John', age=25, salary=50000.0}

All persons are older than 10: true

Average salary (using Collector): 62500.0

Group by salary:

Salary: 70000.0

Person{id=4, name='Jerry', age=35, salary=70000.0}

Salary: 75000.0

Person{id=5, name='Jessica', age=28, salary=75000.0}

Salary: 50000.0

Person{id=1, name='John', age=25, salary=50000.0}

Salary: 55000.0

Person{id=3, name='Jack', age=20, salary=55000.0}

Salary: 60000.0

Person{id=2, name='Jane', age=30, salary=60000.0}

Salary: 65000.0

Person{id=6, name='Jill', age=23, salary=65000.0}

Names of persons older than 18:

John, Jane, Jack, Jerry, Jessica, Jill

No persons are older than 50: true