**Problem Statement 1: Implement the flexible data structures using Collection.**

1. **Write a program to add list of student names to ArrayList and it should find a particular name whether it exists or not in the list.**

**package** day7assignment;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** Arraylist {

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

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

ArrayList <String> students =**new** ArrayList<>();

System.***out***.println("enter number of strings:");

**int** n = sc.nextInt();

sc.nextLine();

System.***out***.println("enter student names:");

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

String name = sc.nextLine();

students.add(name);

}

System.***out***.println("enter a name to search:");

String givenname = sc.nextLine();

**if**(students.contains(givenname)) {

System.***out***.println("the name "+givenname+" is present in list");

}**else** {

System.***out***.println("the name "+givenname+" is not present in list");

}

}

}

**Output:**

enter number of strings:

5

enter student names:

sai

gaurav

suhail

farooq

chandu

enter a name to search:

sai

the name sai is present in list

**2. Create a Product class with Product Id & Product Name. Write a program to accept information of 10 products and store that in HashSet. Do following operations,**

**a. Search a particular product in the HashSet.**

**b. Remove a particular product from the HashSet by using product id.**

**c. (Refer below table for the product list)**

**Product Id Product Name**

**P001 Maruti 800**

**P002 Maruti Zen**

**P003 Maruti Dezire**

**P004 Maruti Alto**

package day7assignment;

import java.util.HashSet;

import java.util.Scanner;

class Product{

String productid;

String productname;

public Product(String productid,String productname) {

this.productid=productid;

this.productname=productname;

}

@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

public String toString() {

return productid + ": " + productname;

}

}

public class Productmanager{

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

System.out.println("enter no products:");

int n = sc.nextInt();

sc.nextLine();

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

System.out.println("enter product "+(i+1)+" id:");

String id =sc.nextLine();

System.out.println("enter product name:");

String name = sc.nextLine();

products.add(new Product(id,name));

}

while (true) {

System.out.println("\n1. Search product");

System.out.println("2. Remove product by ID");

System.out.println("3. Display all products");

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

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

int choice = sc.nextInt();

sc.nextLine();

if (choice == 1) {

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

String name = sc.nextLine();

for (Product p : products) {

if (p.productname.equalsIgnoreCase(name)) {

System.out.println("Found: " + p);

break;

}

else {

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

}

}

}

else if (choice == 2) {

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

String id = sc.nextLine();

products.remove(new Product(id, ""));

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

}

else if (choice == 3) {

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

for (Product p : products) {

System.out.println(p);

}

}

else if (choice == 4) {

System.out.println("Exiting...");

break;

} else {

System.out.println("Invalid choice.");

}

}

}

}

**Output:**

enter no products:

**4**

**enter product 1 id:**

**P001**

**enter product name:**

**Maruti 800**

**enter product 2 id:**

**P002**

**enter product name:**

**Maruti Zen**

**enter product 3 id:**

**P003**

**enter product name:**

**Maruti Dezire**

**enter product 4 id:**

**P004**

**enter product name:**

**Maruti Alto**

1. Search product

2. Remove product by ID

3. Display all products

4. Exit

Enter choice: **1**

**Enter product name to search: Maruti Zen**

**Found: P002: Maruti Zen**

1. Search product

2. Remove product by ID

3. Display all products

4. Exit

Enter choice**: 2**

**Enter product ID to remove: P003**

**Product removed.**

1. Search product

2. Remove product by ID

3. Display all products

4. Exit

Enter choice: **3**

**Products in the list:**

**P004: Maruti Alto**

**P001: Maruti 800**

**P002: Maruti Zen**

1. Search product

2. Remove product by ID

3. Display all products

4. Exit

Enter choice**: 4**

**Exiting...**

**3. Implement LinkedList class for this problem**

**a. Create an Employee class which will have details like EmployeeNo, EmployeeName and Address. You should pass value for EmployeeNo, EmployeeName and Address through constructor.**

**b. Create a method addInput( ) which will add employee details to LinkedList.**

**c. Create method display( ) which should display all data from LinkedList using forward and reverse order using Iterator and ListIterator interfaces.**

**d. Note: addInput( ) and display( ) should not be member functions of Employee class.**

**package** day7assignment;

**import** java.util.LinkedList;

**import** java.util.ListIterator;

**class** Employee{

**private** **int** empno;

**private** String empname;

**private** String address;

**public** Employee(**int** empno,String empname,String address) {

**this**.empno=empno;

**this**.empname=empname;

**this**.address=address;

}

**public** **int** getempno() {

**return** empno;

}

**public** String getempname() {

**return** empname;

}

**public** String getaddress() {

**return** address;

}

@Override

**public** String toString() {

**return** "EmployeeNo: " + empno + ", Name: " + empname + ", Address: " + address;

}

}

**class** EmployeeLinkedList {

**private** LinkedList<Employee> employeeList;

**public** EmployeeLinkedList() {

employeeList = **new** LinkedList<>();

}

**public** **void** addInput(**int** employeeNo, String employeeName, String address) {

Employee employee = **new** Employee(employeeNo, employeeName, address);

employeeList.add(employee);

}

**public** **void** display() {

System.***out***.println("Displaying employee details (Forward Order):");

ListIterator<Employee> iterator = employeeList.listIterator();

**while** (iterator.hasNext()) {

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

}

System.***out***.println("\nDisplaying employee details (Reverse Order):");

iterator = employeeList.listIterator(employeeList.size());

**while** (iterator.hasPrevious()) {

System.***out***.println(iterator.previous());

}

}

}

**public** **class** Linkedlist {

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

EmployeeLinkedList employeeLinkedList = **new** EmployeeLinkedList();

employeeLinkedList.addInput(101, "sai", "1234 kadapa");

employeeLinkedList.addInput(102, "lakshmi", "5678 tirupati");

employeeLinkedList.addInput(103, "pranav", "91011 hydrabad");

employeeLinkedList.display();

}

}

**Output:**

Displaying employee details (Forward Order):

EmployeeNo: 101, Name: sai, Address: 1234 kadapa

EmployeeNo: 102, Name: lakshmi, Address: 5678 tirupati

EmployeeNo: 103, Name: pranav, Address: 91011 hydrabad

Displaying employee details (Reverse Order):

EmployeeNo: 103, Name: pranav, Address: 91011 hydrabad

EmployeeNo: 102, Name: lakshmi, Address: 5678 tirupati

EmployeeNo: 101, Name: sai, Address: 1234 kadapa

**4. Create a Phone Book having userinterface like,**

**a. Add new phone book entry**

**b. Search Phone Number**

**c. Quit**

**Option i :it allows add name and Phone no.**

**Option ii: it must take name as input from the user and based on that it should return phone No.**

**Option iii: will terminate the program.**

**Note: Use HashMap to store phone book entries.**

**package** day7assignment;

**import** java.util.HashMap;

**import** java.util.Scanner;

**public** **class** Phonebook {

**private** HashMap<String, String> phoneBook;

**public** Phonebook() {

phoneBook = **new** HashMap<>();

}

**public** **void** addEntry(String name, String phoneNumber) {

phoneBook.put(name, phoneNumber);

System.***out***.println("Phone book entry added successfully!");

}

**public** **void** searchPhoneNumber(String name) {

**if** (phoneBook.containsKey(name)) {

System.***out***.println("Phone Number for " + name + ": " + phoneBook.get(name));

} **else** {

System.***out***.println("No entry found for " + name);

}

}

**public** **void** displayMenu() {

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();

**switch** (choice) {

**case** 1:

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

String name = scanner.nextLine();

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

String phoneNumber = scanner.nextLine();

addEntry(name, phoneNumber);

**break**;

**case** 2:

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

String searchName = scanner.nextLine();

searchPhoneNumber(searchName);

**break**;

**case** 3:

System.***out***.println("Exiting the phone book.");

scanner.close();

System.*exit*(0);

**default**:

System.***out***.println("Invalid choice! Please choose again.");

}

}

}

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

Phonebook pb = **new** Phonebook();

pb.displayMenu();

}

}

**Output:**

Phone Book Menu:

1. Add new phone book entry

2. Search Phone Number

3. Quit

Enter your choice: **1**

**Enter Name: sai**

**Enter Phone Number: 1236547890**

**Phone book 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: sai**

**Phone Number for sai: 1236547890**

Phone Book Menu:

1. Add new phone book entry

2. Search Phone Number

3. Quit

Enter your choice: **3**

**Exiting the phone book.**

package day7assignment;

import java.text.SimpleDateFormat;

import java.util.Comparator;

import java.util.Date;

import java.util.TreeSet;

class Book implements Comparable<Book> {

private int bookId;

private String title;

private double price;

private Date dateOfPublication;

private String author;

private static SimpleDateFormat *sdf* = new SimpleDateFormat("dd/MM/yy");

public Book(int bookId, String title, double price, String author, Date dateOfPublication) {

this.bookId = bookId;

this.title = title;

this.price = price;

this.dateOfPublication = dateOfPublication;

this.author = author;

}

public int getBookId() {

return bookId;

}

public String getTitle() {

return title;

}

public double getPrice() {

return price;

}

public Date getDateOfPublication() {

return dateOfPublication;

}

public String getAuthor() {

return author;

}

@Override

public String toString() {

return "BookId: " + bookId + ", Title: " + title + ", Price: " + price + ", Author: " + author + ", DOP: " + *sdf*.format(dateOfPublication);

}

@Override

public int hashCode() {

return Integer.*hashCode*(bookId);

}

@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;

}

@Override

public int compareTo(Book otherBook) {

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

}

public static Comparator<Book> *dateOfPublicationDesc* = new Comparator<Book>() {

@Override

public int compare(Book b1, Book b2) {

return b2.dateOfPublication.compareTo(b1.dateOfPublication);

}

};

public static Comparator<Book> *titleAsc* = new Comparator<Book>() {

@Override

public int compare(Book b1, Book b2) {

return b1.title.compareTo(b2.title);

}

};

public static Comparator<Book> *bookIdDescAndDateAsc* = new Comparator<Book>() {

@Override

public int compare(Book b1, Book b2) {

int bookIdCompare = Integer.*compare*(b2.bookId, b1.bookId);

if (bookIdCompare != 0) {

return bookIdCompare;

} else {

return b1.dateOfPublication.compareTo(b2.dateOfPublication);

}

}

};

}

public class Main {

public static void main(String[] args) {

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

books.add(new Book(101, "Java Programming", 499.99, "James Gosling", new Date(120, 5, 12)));

books.add(new Book(102, "Data Structures", 599.99, "Donald Knuth", new Date(118, 10, 23)));

books.add(new Book(103, "Algorithms", 799.99, "Donald Knuth", new Date(121, 2, 5)));

books.add(new Book(104, "Clean Code", 349.99,"Robert C. Martin", new Date(119, 8, 30)));

books.add(new Book(105, "Design Patterns", 699.99,"Erich Gamma",new Date(117, 6, 19)));

System.*out*.println("--------------------------Books sorted by Author (Ascending)--------------------------------");

for (Book book : books) {

System.*out*.println(book);

}

TreeSet<Book> booksByDateDesc = new TreeSet<>(Book.*dateOfPublicationDesc*);

booksByDateDesc.addAll(books);

System.*out*.println("\n-------------------------Books sorted by Date of Publication (Descending)--------------------");

for (Book book : booksByDateDesc) {

System.*out*.println(book);

}

TreeSet<Book> booksByTitleAsc = new TreeSet<>(Book.*titleAsc*);

booksByTitleAsc.addAll(books);

System.*out*.println("\n--------------------------Books sorted by Title (Ascending)--------------------------------");

for (Book book : booksByTitleAsc) {

System.*out*.println(book);

}

TreeSet<Book> booksByIdDescAndDateAsc = new TreeSet<>(Book.*bookIdDescAndDateAsc*);

booksByIdDescAndDateAsc.addAll(books);

System.*out*.println("\n------------------Books sorted by BookId (Descending) and DOP (Ascending)-------------------");

for (Book book : booksByIdDescAndDateAsc) {

System.*out*.println(book);

}

}

}

--------------------------Books sorted by Author (Ascending)--------------------------------

BookId: 102, Title: Data Structures, Price: 599.99, Author: Donald Knuth, DOP: 23/11/18

BookId: 105, Title: Design Patterns, Price: 699.99, Author: Erich Gamma, DOP: 19/07/17

BookId: 101, Title: Java Programming, Price: 499.99, Author: James Gosling, DOP: 12/06/20

BookId: 104, Title: Clean Code, Price: 349.99, Author: Robert C. Martin, DOP: 30/09/19

-------------------------Books sorted by Date of Publication (Descending)--------------------

BookId: 101, Title: Java Programming, Price: 499.99, Author: James Gosling, DOP: 12/06/20

BookId: 104, Title: Clean Code, Price: 349.99, Author: Robert C. Martin, DOP: 30/09/19

BookId: 102, Title: Data Structures, Price: 599.99, Author: Donald Knuth, DOP: 23/11/18

BookId: 105, Title: Design Patterns, Price: 699.99, Author: Erich Gamma, DOP: 19/07/17

--------------------------Books sorted by Title (Ascending)--------------------------------

BookId: 104, Title: Clean Code, Price: 349.99, Author: Robert C. Martin, DOP: 30/09/19

BookId: 102, Title: Data Structures, Price: 599.99, Author: Donald Knuth, DOP: 23/11/18

BookId: 105, Title: Design Patterns, Price: 699.99, Author: Erich Gamma, DOP: 19/07/17

BookId: 101, Title: Java Programming, Price: 499.99, Author: James Gosling, DOP: 12/06/20

------------------Books sorted by BookId (Descending) and DOP (Ascending)-------------------

BookId: 105, Title: Design Patterns, Price: 699.99, Author: Erich Gamma, DOP: 19/07/17

BookId: 104, Title: Clean Code, Price: 349.99, Author: Robert C. Martin, DOP: 30/09/19

BookId: 102, Title: Data Structures, Price: 599.99, Author: Donald Knuth, DOP: 23/11/18

BookId: 101, Title: Java Programming, Price: 499.99, Author: James Gosling, DOP: 12/06/20

import java.util.\*;

import java.util.stream.Collectors;

class Person implements Comparable<Person> {

private int id;

private String name;

private int age;

private double salary;

// Constructor

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 toString method

@Override

public String toString() {

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

}

// Override hashCode method

@Override

public int hashCode() {

return Objects.hash(id, name, age, salary);

}

// Override equals method

@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 && age == person.age &&

Double.compare(person.salary, salary) == 0 && Objects.equals(name, person.name);

}

// Override compareTo method for sorting by id

@Override

public int compareTo(Person other) {

return Integer.compare(this.id, other.id);

}

}

public class PersonMain {

public static void main(String[] args) {

// Creating a HashSet to store persons

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

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

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

persons.add(new Person(3, "Bob", 22, 40000));

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

persons.add(new Person(5, "Jake", 40, 80000));

persons.add(new Person(6, "Jonah", 29, 55000));

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

System.out.println("All Person details:");

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

// 7. Print all persons details by sorting the id in ascending order using Comparable and Streams

System.out.println("\nSorted by ID (ascending):");

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

// 8. Print all persons details by sorting the name in ascending order using Comparator and Streams

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

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

.forEach(System.out::println);

// 9. Print all persons details by sorting the names in descending order using Comparator and Streams

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

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

.forEach(System.out::println);

// 10. Print all 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);

// 11. Print the count number of persons using Streams

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

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

// 12. Print the Max salary among all persons using Streams

persons.stream().mapToDouble(Person::getSalary).max().ifPresent(max ->

System.out.println("\nMax Salary: " + max));

// 13. Print the Min salary among all persons using Streams

persons.stream().mapToDouble(Person::getSalary).min().ifPresent(min ->

System.out.println("\nMin Salary: " + min));

// 14. Print the average of all salaries using Streams

persons.stream().mapToDouble(Person::getSalary).average().ifPresent(avg ->

System.out.println("\nAverage Salary: " + avg));

// 15. Print the sum of all salaries using Streams

double sum = persons.stream().mapToDouble(Person::getSalary).sum();

System.out.println("\nTotal Sum of Salaries: " + sum);

// 16. Print the First Person whose Name start with J using Streams - filter and findFirst method

persons.stream().filter(p -> p.getName().startsWith("J")).findFirst().ifPresent(person ->

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

// 17. Check whether all the persons age is greater than 10 using Streams – allMatch method

boolean allAgeAbove10 = persons.stream().allMatch(p -> p.getAge() > 10);

System.out.println("\nAre all persons' age greater than 10? " + allAgeAbove10);

// 18. Print the average of all salaries using Streams and Collectors

double averageSalary = persons.stream().collect(Collectors.averagingDouble(Person::getSalary));

System.out.println("\nAverage Salary (using Collectors): " + averageSalary);

// 19. Print all the persons details group by salary using Streams and Collectors

Map<Double, List<Person>> groupedBySalary = persons.stream().collect(Collectors.groupingBy(Person::getSalary));

System.out.println("\nPersons grouped by Salary:");

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

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

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

});

// 20. Print all the names after joining whose age is greater than 18 using Streams and Collectors

String namesAbove18 = persons.stream()

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

.map(Person::getName)

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

System.out.println("\nNames of persons whose age is greater than 18: " + namesAbove18);

// 21. Check whether all the persons age is greater than 50 using Streams – noneMatch method

boolean noneAgeAbove50 = persons.stream().noneMatch(p -> p.getAge() > 50);

System.out.println("\nAre none of the persons' age greater than 50? " + noneAgeAbove50);

}

}

All Person details:

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

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

Person{id=3, name='Bob', age=22, salary=40000.0}

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

Person{id=5, name='Jake', age=40, salary=80000.0}

Person{id=6, name='Jonah', age=29, salary=55000.0}

Sorted by ID (ascending):

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

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

Person{id=3, name='Bob', age=22, salary=40000.0}

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

Person{id=5, name='Jake', age=40, salary=80000.0}

Person{id=6, name='Jonah', age=29, salary=55000.0}

Sorted by Name (ascending):

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

Person{id=3, name='Bob', age=22, salary=40000.0}

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

Person{id=5, name='Jake', age=40, salary=80000.0}

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

Person{id=6, name='Jonah', age=29, salary=55000.0}

Sorted by Name (descending):

Person{id=6, name='Jonah', age=29, salary=55000.0}

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

Person{id=5, name='Jake', age=40, salary=80000.0}

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

Person{id=3, name='Bob', age=22, salary=40000.0}

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

Persons whose Name starts with 'J':

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

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

Person{id=5, name='Jake', age=40, salary=80000.0}

Person{id=6, name='Jonah', age=29, salary=55000.0}

Count of persons: 6

Max Salary: 80000.0

Min Salary: 40000.0

Average Salary: 61666.666666666664

Total Sum of Salaries: 370000.0

First Person whose Name starts with 'J': Person{id=1, name='John', age=25, salary=50000.0}

Are all persons' age greater than 10? true

Average Salary (using Collectors): 61666.666666666664

Persons grouped by Salary:

Salary: 40000.0

Person{id=3, name='Bob', age=22, salary=40000.0}

Salary: 50000.0

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

Salary: 55000.0

Person{id=6, name='Jonah', age=29, salary=55000.0}

Salary: 60000.0

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

Salary: 70000.0

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

Salary: 80000.0

Person{id=5, name='Jake', age=40, salary=80000.0}

Names of persons whose age is greater than 18: John, Alice, Bob, Jasmine, Jake, Jonah

Are none of the persons' age greater than 50? true