**DAY-6 PRACTICE EXERCISE**

**BASICS OF JAVA**

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

**1.Student Class**

**package** org.software.com;

**import** java.util.ArrayList;

**import** java.util.Scanner;

**public** **class** Student {

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

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

students.add("Sreekanth");

students.add("Akhila");

students.add("Karthik");

students.add("Chaitanya");

students.add("Sridevi");

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

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

String name = sc.nextLine();

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

**System**.***out***.println("Name is present in the list.");

} **else** {

**System**.***out***.println("Name is not in the list.");

}

}

}

**2.Product Class**

**package** org.software.com;

**import** java.util.HashSet;

**import** java.util.Iterator;

**import** java.util.Scanner;

**class** Product {

**private** String productId;

**private** String productName;

**public** Product(String productId, String productName) {

**this**.productId = productId;

**this**.productName = productName;

}

**public** String getProductId() {

**return** productId;

}

**public** String getProductName() {

**return** 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** "Product ID: " + productId + ", Product Name: " + productName;

}

}

**ProductDemo Class**

**public** class ProductDemo {

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

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

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

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

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

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

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

productSet.add(new Product("P005", "Maruti Sw**if**t"));

productSet.add(new Product("P006", "Tata Punch"));

productSet.add(new Product("P007", "Tata Safari"));

productSet.add(new Product("P008", "Tata Nexon"));

productSet.add(new Product("P009", "Tata EV"));

productSet.add(new Product("P010", "Mercedes Benz"));

**System**.out.println("Enter Product ID to Search: ");

String searchId = sc.nextLine();

boolean found = false;

for (Product product : productSet) {

**if** (product.getProductId().equalsIgnoreCase(searchId)) {

**System**.out.println("Product Found");

**System**.out.println(product);

found = true;

break;

}

}

**if** (!found) {

**System**.out.println("Product Not Found");

}

**System**.out.println("\nEnter Product ID to Delete: ");

String removeId = sc.nextLine().trim();

Iterator<Product> iterator = productSet.iterator();

found = false;

while (iterator.hasNext()) {

Product product = iterator.next();

**if** (product.getProductId().equalsIgnoreCase(removeId)) {

iterator.remove();

**System**.out.println("Product with ID " + removeId + " deleted");

found = true;

break;

}

}

**if** (!found) {

**System**.out.println("Product with ID " + removeId + " not found");

}

**System**.out.println("\n Products are: ");

for (Product product : productSet) {

**System**.out.println(product);

}

}

}

**3.Employee Class**

**package** org.software.com;

**import** java.util.Iterator;

**import** java.util.LinkedList;

**import** java.util.ListIterator;

**import** java.util.Scanner;

**class** Employee {

**private** **int** employeeNo;

**private** String employeeName;

**private** String address;

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

**this**.employeeNo = employeeNo;

**this**.employeeName = employeeName;

**this**.address = address;

}

**public** **int** getEmployeeNo() {

**return** employeeNo;

}

**public** String getEmployeeName() {

**return** employeeName;

}

**public** String getAddress() {

**return** address;

}

@Override

**public** String toString() {

**return** "Employee Number: " + employeeNo + ", Employee Name: " + employeeName + ", Address: " + address;

}

}

**public** **class** EmployeeTest {

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

**private** LinkedList<Employee> emps;

**public** EmployeeTest() {

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

}

**public** **void** addInput() {

**System**.***out***.println("Enter Employee Number ");

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

*sc*.nextLine(); // consume newline

**System**.***out***.println("Enter Employee Name : ");

String empName = *sc*.nextLine();

**System**.***out***.println("Enter Employee Address : ");

String empAddress = *sc*.nextLine();

Employee emp = **new** Employee(empNo, empName, empAddress);

emps.add(emp);

}

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

**System**.***out***.println("Employee Details in forward order:");

Iterator<Employee> itr = emps.iterator();

**while** (itr.hasNext()) {

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

}

**System**.***out***.println("\nEmployee Details in reverse order:");

ListIterator<Employee> listItr = emps.listIterator(emps.size());

**while** (listItr.hasPrevious()) {

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

}

}

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

EmployeeTest empTest = **new** EmployeeTest();

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

**while** (**true**) {

**System**.***out***.println("Enter Option ");

**System**.***out***.println("1.Add Employee ");

**System**.***out***.println("2.Display Employees ");

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

**switch** (key) {

**case** 1:

empTest.addInput();

**break**;

**case** 2:

empTest.display();

**break**;

**default**:

**System**.***out***.println("Invalid Option");

**break**;

}

}

}

}

**4.PhoneBook Class**

**package** org.software.com;

**import** java.util.HashMap;

**import** java.util.Scanner;

**public** **class** PhoneBook {

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

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

**public** PhoneBook() {

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

}

**public** **void** addEntry() {

**System**.***out***.println("Enter name : ");

String name = *sc*.next();

**System**.***out***.println("Enter phone number : ");

Long phoneNo = *sc*.nextLong();

phoneBook.put(name, phoneNo);

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

}

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

Long phoneNo = phoneBook.get(name);

**if** (phoneNo != **null**) {

**System**.***out***.println("Phone number for " + name + ": " + phoneNo);

} **else** {

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

}

}

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

PhoneBook phoneBook = **new** PhoneBook();

**while** (**true**) {

**System**.***out***.println("Phone 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***.println("Enter your choice: ");

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

**switch** (choice) {

**case** 1:

phoneBook.addEntry();

**break**;

**case** 2:

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

String searchName = *sc*.next();

phoneBook.searchPhoneNo(searchName);

**break**;

**case** 3:

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

**System**.*exit*(0);

**break**;

**default**:

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

**break**;

}

}

}

}

**Problem : 5**

**Book Class**

**package** org.software.com;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Comparator;

**import** java.util.Date;

**import** java.util.TreeSet;

**import** java.util.Objects;

class Book implements Comparable<Book> {

private String bookId;

private String title;

private double price;

private Date dop; // Date of **Public**ation

private String author;

**public** Book(String bookId, String title, double price, Date dop, String author) {

this.bookId = bookId;

this.title = title;

this.price = price;

this.dop = dop;

this.author = author;

}

**public** String getBookId() {

**return** bookId;

}

**public** void setBookId(String bookId) {

this.bookId = bookId;

}

**public** String getTitle() {

**return** title;

}

**public** void setTitle(String title) {

this.title = title;

}

**public** double getPrice() {

**return** price;

}

**public** void setPrice(double price) {

this.price = price;

}

**public** Date getDop() {

**return** dop;

}

**public** void setDop(Date dop) {

this.dop = dop;

}

**public** String getAuthor() {

**return** author;

}

**public** void setAuthor(String author) {

this.author = author;

}

@Override

**public** String toString() {

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

**return** "Book [bookId=" + bookId + ", title=" + title + ", price=" + price + ", author=" + author + ", dop="

+ dateFormat.format(dop) + "]";

}

@Override

**public** int hashCode() {

**return** Objects.hash(bookId, title, price, dop, 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** Double.compare(book.price, price) == 0 && Objects.equals(bookId, book.bookId)

&& Objects.equals(title, book.title) && Objects.equals(dop, book.dop)

&& Objects.equals(author, book.author);

}

@Override

**public** int compareTo(Book other) {

**return** this.author.compareTo(other.author); // Natural ordering by author name

}

}

**BookTest Class**

**public** class BookTest {

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

SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");

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

books.add(new Book("1003", "Java Programming", 523.0, dateFormat.parse("23/11/1984"), "Gilad Bracha"));

books.add(new Book("1004", "Read C++", 295.0, dateFormat.parse("19/11/1984"), "Henry Harvin"));

books.add(new Book("1005", ".Net Platform", 3497.0, dateFormat.parse("06/03/1984"), "Mark J Price"));

books.add(new Book("1001", "Python Learning", 715.0, dateFormat.parse("02/02/2020"), "Martin C Brown"));

books.add(new Book("1002", "Modern Mainframe", 295.0, dateFormat.parse("19/05/1997"), "Sharad"));

**System**.out.println("Books sorted by author names Ascending :");

for (Book book : books) {

**System**.out.println(book);

}

**System**.out.println("\nBooks sorted by date of **public**ation Descending :");

TreeSet<Book> booksByDate = new TreeSet<>(Comparator.comparing(Book::getDop).reversed());

booksByDate.addAll(books);

for (Book book : booksByDate) {

**System**.out.println(book);

}

**System**.out.println("\nBooks sorted by title Ascending :");

TreeSet<Book> booksByTitle = new TreeSet<>(Comparator.comparing(Book::getTitle));

booksByTitle.addAll(books);

for (Book book : booksByTitle) {

**System**.out.println(book);

}

**System**.out.println("\nBooks sorted by book ID Descending and date of **public**ation Ascending:");

TreeSet<Book> booksByIdAndDate = new TreeSet<>( Comparator.comparing(Book::getBookId).reversed().thenComparing(Book::getDop));

booksByIdAndDate.addAll(books);

for (Book book : booksByIdAndDate) {

**System**.out.println(book);

}

}

}

**Problem Statement 2: Processing Data with Java SE 8 Streams**

**Problem : 6 to 21**

**Person Class**

**package** org.software.com;

**import** java.util.Objects;

**public** **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;

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

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

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** **double** getSalary() {

**return** salary;

}

**public** **void** setSalary(**double** salary) {

**this**.salary = salary;

}

@Override

**public** **int** compareTo(Person other) {

**return** Integer.*compare*(**this**.id, other.id); // Default sorting by ID

}

@Override

**public** **int** hashCode() {

**return** Objects.*hash*(id, name, age, 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 &&

age == person.age &&

Double.*compare*(person.salary, salary) == 0 &&

Objects.*equals*(name, person.name);

}

@Override

**public** String toString() {

**return** String.*format*("Person [ID=%d, Name=%s, Age=%d, Salary=%.2f]", id, name, age, salary);

}

}

**PersonTest Class**

**package** org.software.com;

**import** java.util.\*;

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

**public** class PersonTest {

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

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

persons.add(new Person(4, "Jones", 22, 6999.0));

persons.add(new Person(6, "Tom", 42, 3999.0));

persons.add(new Person(1, "Jerry", 12, 999.0));

persons.add(new Person(5, "John", 32, 1999.0));

persons.add(new Person(2, "Smith", 22, 2999.0));

persons.add(new Person(3, "Popeye", 21, 5999.0));

**System**.out.println("----------Print all The Persons records-----------");

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

**System**.out.println("\n----------Sorted by Id Ascending Order--------------");

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

**System**.out.println("\n----------Sory by Name Ascending Order--------------");

persons.stream().sorted(Comparator.comparing(Person::getName)).forEach(**System**.out::println);

**System**.out.println("\n----------Sorted by Name Descending Order--------------");

persons.stream().sorted(Comparator.comparing(Person::getName).reversed()).forEach(**System**.out::println);

**System**.out.println("\n----------Names Starting with 'J'--------------");

persons.stream().filter(p -> p.getName().startsWith("J")).forEach(**System**.out::println);

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

**System**.out.println("\n-----------Count of Persons----------");

**System**.out.println(count);

**System**.out.println("\n----------Max Salary Among Persons------");

persons.stream().map(Person::getSalary).max(Double::compareTo)

.**if**Present(maxSalary -> **System**.out.println(maxSalary));

**System**.out.println("\n----------Min Salary Among Persons-------");

persons.stream().map(Person::getSalary).min(Double::compareTo)

.**if**Present(minSalary -> **System**.out.println(minSalary));

**System**.out.println("\n---------Average Salary---------");

double avgSalary = persons.stream().mapToDouble(Person::getSalary).average().or**Else**(0);

**System**.out.println(avgSalary);

**System**.out.println("\n--------Sum of All Salaries----------");

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

**System**.out.println(totalSal);

**System**.out.println("\n-------First Person with Name Starting with 'J'------");

persons.stream().filter(p -> p.getName().startsWith("J")).findFirst().**if**Present(**System**.out::println);

**System**.out.println("\n------Check **if** All Ages Greater Than 10------");

boolean above10 = persons.stream().allMatch(p -> p.getAge() > 10);

**System**.out.println(above10);

**System**.out.println("\n--------Average Salary Using Collectors----------");

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

**System**.out.println(avgAllSalarys);

**System**.out.println("\n---------Group By Salary----------");

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

groupBySal.forEach((salary, personList) -> {

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

personList.forEach(**System**.out::println);

});

**System**.out.println("\n-------Names of Persons with Age Greater Than 18------");

String names = persons.stream().filter(p -> p.getAge() > 18).map(Person::getName)

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

**System**.out.println(names);

**System**.out.println("\n----Check **if** No Person's Age is Greater Than 50--------");

boolean above50 = persons.stream().noneMatch(p -> p.getAge() > 50);

**System**.out.println(above50);

}

}