**Asutosh Dash**

**Daily Assignments**

**Sprint 1 Day 2**

1. Shopping Cart with ArrayList (10 points)

Problem:

Create a simple shopping cart application that allows users to:

Add items to the cart

Remove items from the cart

View all items in the cart

Calculate the total price of all items

Requirements:

Use ArrayList to store cart items

Each item should have a name, price, and quantity

Implement methods for all the operations mentioned above

#### Explanation:

In this question we have to create an arraylist that store the data of the cart items and also add features like add items, remove, update items. We will create a Class that have the name, price and quantity of the store item and write methods for manipulation of data.

### Solution:

import java.util.List;  
import java.util.ArrayList;  
class Item{  
 String name;  
 double price;  
 int quantity;  
 Item(String name,double price,int quantity){  
 this.name=name;  
 this.price=price;  
 this.quantity=quantity;;  
 }  
 }  
  
 class shopping{  
 List<Item>cart=new ArrayList<>();  
 void additem(String name,double price,int quantity){  
 cart.add(new Item(name,price,quantity));  
 }  
 void remove(String name){  
 cart.removeIf(i-> {  
 return i.name.equalsIgnoreCase(name);  
 });  
  
 }  
 void viewItem(){  
 for(Item i:cart){  
 System.*out*.println(i.name+"-"+i.price+"-"+i.quantity);  
 }  
 }  
 double totalprice(){  
 double total=0;  
 for(Item i :cart){  
 total +=i.price\*i.quantity;  
 }  
 return total;  
 }  
 }  
public class Main {  
 public static void main(String[] args) {  
 shopping cart = new shopping();  
  
 cart.additem("Banana",0.99,8);  
 cart.additem("Apple",0.98,10);  
  
  
 cart.additem("orange",1.00,2);  
  
 cart.viewItem();  
 cart.remove("Banana");  
 cart.viewItem();  
 System.*out*.println("Total Price"+cart.totalprice());  
  
  
 }  
}

1. Student Management with ArrayList (10 points)

Problem:

Create a student management system where you can:

Add new students to a list

Remove students from the list

Search for a student by ID

Display all students

Requirements:

Use ArrayList to store Student objects

Each Student should have ID, name, and grade attributes

#### Explanation:

In this question we have to create an arraylist that store the data of the students and also add features like add new students , remove students , search student by ID and display all student . We will create a Class that have the name, ID and Grade of the student and write methods for manipulation of data.

### Solution:

import java.util.List;  
import java.util.ArrayList;  
class Student{  
 String name;  
 int ID;  
 int Grade;  
 Student(String name,int ID,int Grade){  
 this.name=name;  
 this.ID=ID;  
 this.Grade=Grade;;  
 }  
 }  
  
 class StudentDetails{  
 List<Student>details=new ArrayList<>();  
 void additem(String name,int ID,int Grade){  
 details.add(new Student(name,ID,Grade));  
 }  
 void remove(String name){  
 details.removeIf(i-> {  
 return i.name.equalsIgnoreCase(name);  
 });  
  
 }  
 void viewItem(){  
 for(Student i:details){  
 System.*out*.println(i.name+"-"+i.ID+"-"+i.Grade);  
 }  
 }  
 void serach(int ID){  
 for(Student i:details){  
 if(i.ID==ID){  
 System.*out*.println("Name: "+i.name+"ID: "+i.ID+"Grade: "+i.Grade);  
 }  
 }  
 }  
  
 }  
public class Main {  
 public static void main(String[] args) {  
 StudentDetails stud = new StudentDetails();  
  
 stud.additem("Asutosh",420,10);  
 stud.additem("Anaya",120,11);  
  
  
 stud.additem("Aman",200,8);  
  
 stud.viewItem();  
 stud.remove("anaya");  
 stud.viewItem();  
 stud.serach(200);  
  
  
 }  
}

1. Email Duplicates Remover with HashSet (10 points)

Problem:

Create a program that removes duplicate email addresses from a list of contacts.

Requirements:

Use HashSet to efficiently remove duplicates

Allow adding multiple email addresses

Display the unique email addresses

#### Explanation:

In this question we have to create a hash list in order to avoid duplicate emails and only return unique emails .

### Solution:

import java.util.HashSet;  
import java.util.Set;  
  
class emails{  
 Set<String> eml=new HashSet<>();  
 void addemls(String emls){  
 eml.add(emls.toLowerCase());  
 }  
 void displayeml(){  
 for(String emls:eml){  
 System.*out*.println(emls);  
 }  
 }  
  
}  
public class Main {  
 public static void main(String[] args) {  
 emails eml=new emails();  
 eml.addemls("123@gmail.com");  
 eml.addemls("133@gmail.com");  
 eml.addemls("123@gmail.com");  
 eml.addemls("113@gmail.com");  
 eml.displayeml();  
 }  
}

1. Product Inventory with HashMap (15 points)

Problem:

Create a product inventory system using HashMap that allows:

Adding products with their quantities

Updating product quantities

Removing products

Checking if a product is in stock

Requirements:

Use HashMap to store products (name as key, quantity as value)

Implement methods for all operations mentioned above

#### Explanation:

Use a HashMap with product name as key and quantity as value. Perform add, update, remove, and check operations.

### Solution:

import java.util.HashMap;  
import java.util.Map;  
  
  
class Inventory{  
 Map<String,Integer> products=new HashMap<>();  
 void addproducts(String name,int quantity){  
 products.put(name,quantity);  
 }  
 void updateproducts(String name,int quatity){  
 products.put(name,quatity);  
 }  
 void removeProduct(String name){  
 products.remove(name);  
 }  
 boolean check(String name){  
 return products.containsKey(name) && products.get(name)>0;  
 }  
  
}  
public class Main {  
 public static void main(String[] args) {  
 Inventory in=new Inventory();  
 in.addproducts("Laptop1",10);  
 in.addproducts("Laptop2",20);  
 in.addproducts("Laptop3",30);  
 in.updateproducts("Laptop1",40);  
 System.*out*.println(in.check("Laptop1"));  
 in.removeProduct("Laptop2");  
  
  
 }  
}

1. Browser History with Stack (10 points)

Problem:

Create a simple browser history feature using Stack that allows:

Visiting new websites (pushing URLs to stack)

Going back to previous sites (popping from stack)

Viewing current page

Viewing entire history

Requirements:

Use java.util.Stack to implement browser history

Implement methods for all the operations mentioned above

#### Explanation:

Use Stack to simulate browser history. Push for visiting, pop for back, and peek for current page.

### Solution:

import java.util.Stack;  
  
  
class Urls{  
 Stack<String> hist=new Stack<>();  
 void addURL(String url){  
 hist.push(url);  
 }  
 void popURL(){  
 if(!hist.isEmpty())hist.pop();  
 }  
 void currentURL(){  
 if(!hist.isEmpty())System.*out*.println("Current URLS: "+hist.peek());  
 }  
 void viewhist(){  
 for(String urls:hist){  
 System.*out*.println(urls);  
 }  
 }  
  
  
}  
public class Main {  
 public static void main(String[] args) {  
 Urls history=new Urls();  
 history.addURL("www.example1.com");  
 history.addURL("www.example2.com");  
 history.addURL("www.example3.com");  
 history.addURL("www.example4.com");  
 history.popURL();  
 history.currentURL();  
 history.popURL();  
 history.currentURL();  
 history.viewhist();  
 history.addURL("www.example3.com");  
 history.addURL("www.example4.com");  
 history.viewhist();  
  
  
  
 }  
}

1. Employee Records with TreeSet (15 points)

Problem:

Create a program that stores and displays employee records in alphabetical order by name.

Requirements:

Use TreeSet to automatically sort employees by name

Employee class should include ID, name, department, and salary

Should allow adding employees and viewing all employees in sorted order

#### Explanation:

Use TreeSet with a custom comparator to sort by name automatically.  
  
Solution:

import java.util.\*;  
  
  
class Employee {  
 int id;  
 String name;  
 String dept;  
 double salary;  
  
 Employee(int id, String name, String dept, double salary) {  
 this.id = id;  
 this.name = name;  
 this.dept = dept;  
 this.salary = salary;  
 }  
}  
  
  
class EmployeeManager {  
 Set<Employee> employees = new TreeSet<>(Comparator.*comparing*((Employee e) -> e.name));  
  
 void addEmployee(Employee e) {  
 employees.add(e);  
 }  
  
 void displayAll() {  
 for (Employee e : employees) {  
 System.*out*.println(e.id + " " + e.name + " " + e.dept + " " + e.salary);  
 }  
 }  
}  
  
  
public class Main {  
 public static void main(String[] args) {  
 EmployeeManager manager = new EmployeeManager();  
  
  
 manager.addEmployee(new Employee(101, "Alice", "HR", 50000));  
 manager.addEmployee(new Employee(102, "Bob", "IT", 60000));  
 manager.addEmployee(new Employee(103, "Charlie", "Finance", 55000));  
 manager.addEmployee(new Employee(104, "Alice", "Marketing", 58000)); // Duplicate name  
  
  
 System.*out*.println("Employees sorted by name:");  
 manager.displayAll();  
 }  
}

1. Word Frequency Counter with HashMap (15 points)

Problem:

Create a program that counts the frequency of each word in a text input.

Requirements:

Use HashMap to store words and their counts

Ignore case and punctuation

Display words sorted by frequency (highest first)

Explanation:

Use HashMap to store word counts. Normalize case and remove punctuation.  
  
Solution:

import java.util.\*;  
  
class WordFrequency {  
  
 void countWords(String text) {  
 Map<String, Integer> freq = new HashMap<>();  
  
   
 String[] words = text.toLowerCase().replaceAll("[^a-z ]", "").split("\\s+");  
  
  
 for (String word : words) {  
 freq.put(word, freq.getOrDefault(word, 0) + 1);  
 }  
  
  
 freq.entrySet().stream()  
 .sorted(Map.Entry.<String, Integer>*comparingByValue*(Comparator.*reverseOrder*()))  
 .forEach(entry -> System.*out*.println(entry.getKey() + ": " + entry.getValue()));  
 }  
  
  
 public static void main(String[] args) {  
 WordFrequency wf = new WordFrequency();  
 String sampleText = "Java is great! Java is powerful. Java is easy to learn.";  
 wf.countWords(sampleText);  
 }  
}

1. Restaurant Menu with LinkedHashMap (10 points)

Problem:

Create a restaurant menu system that:

Stores menu items with their prices

Groups menu items by category (appetizer, main course, dessert)

Allows adding, removing, and updating menu items

Displays the menu by category

Requirements:

Use LinkedHashMap to maintain insertion order of categories

Use HashMap for storing menu items in each category

Each menu item should have a name, description, and price

Explanation:

Used LinkedHashMap to maintain order of categories, each mapped to a HashMap of menu items.

Solution:

import java.util.\*;  
  
  
class MenuItem {  
 String name;  
 String description;  
 double price;  
  
 MenuItem(String name, String description, double price) {  
 this.name = name;  
 this.description = description;  
 this.price = price;  
 }  
}  
  
  
class RestaurantMenu {  
 Map<String, Map<String, MenuItem>> menu = new LinkedHashMap<>();  
  
   
 void addItem(String category, MenuItem item) {  
 menu.putIfAbsent(category, new HashMap<>());  
 menu.get(category).put(item.name, item);  
 }  
  
   
 void removeItem(String category, String name) {  
 if (menu.containsKey(category)) {  
 menu.get(category).remove(name);  
 }  
 }  
  
   
 void displayMenu() {  
 for (String category : menu.keySet()) {  
 System.*out*.println("\n" + category + ":");  
 for (MenuItem item : menu.get(category).values()) {  
 System.*out*.println(" - " + item.name + " | " + item.description + " | $" + item.price);  
 }  
 }  
 }  
}  
  
  
public class Main {  
 public static void main(String[] args) {  
 RestaurantMenu menu = new RestaurantMenu();  
  
   
 menu.addItem("Appetizers", new MenuItem("Spring Rolls", "Crispy rolls with veggies", 5.99));  
 menu.addItem("Appetizers", new MenuItem("Garlic Bread", "Toasted bread with garlic butter", 4.49));  
 menu.addItem("Main Course", new MenuItem("Grilled Chicken", "Served with veggies and rice", 12.99));  
 menu.addItem("Desserts", new MenuItem("Cheesecake", "Classic creamy cheesecake", 6.49));  
 menu.addItem("Drinks", new MenuItem("Lemonade", "Freshly squeezed lemon drink", 2.99));  
  
   
 menu.removeItem("Appetizers", "Garlic Bread");  
  
   
 menu.displayMenu();  
 }  
}

1. Event Calendar with TreeMap (10 points)

Problem:

Create an event calendar application that:

Stores events by date

Allows adding and removing events

Displays events for a specific date

Lists all upcoming events

Requirements:

Use TreeMap with date strings as keys for automatic chronological sorting

Each date can have multiple events

Each event should have title, time, and description

Explanation:

In this question we will use TreeMap in order to maintain the order of the data and sorting data string.

Solution:

import java.util.\*;  
  
class Event {  
 String title;  
 String time;  
 String description;  
  
 Event(String title, String time, String description) {  
 this.title = title;  
 this.time = time;  
 this.description = description;  
 }  
}  
  
class EventCalendar {  
 Map<String, List<Event>> calendar = new TreeMap<>();  
  
 void addEvent(String date, Event event) {  
 calendar.putIfAbsent(date, new ArrayList<>());  
 calendar.get(date).add(event);  
 }  
  
 void removeEvent(String date, String title) {  
 if (calendar.containsKey(date)) {  
 calendar.get(date).removeIf(e -> e.title.equalsIgnoreCase(title));  
 }  
 }  
  
 void displayEvents(String date) {  
 if (calendar.containsKey(date)) {  
 System.*out*.println("Events on " + date + ":");  
 for (Event e : calendar.get(date)) {  
 System.*out*.println(" - " + e.title + " | " + e.time + " | " + e.description);  
 }  
 } else {  
 System.*out*.println("No events on " + date);  
 }  
 }  
  
 void listAllEvents() {  
 for (String date : calendar.keySet()) {  
 System.*out*.println("\n" + date + ":");  
 for (Event e : calendar.get(date)) {  
 System.*out*.println(" - " + e.title + " | " + e.time + " | " + e.description);  
 }  
 }  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 EventCalendar myCalendar = new EventCalendar();  
 myCalendar.addEvent("2025-05-01", new Event("Project Meeting", "10:00 AM", "Team sync-up for project X"));  
 myCalendar.addEvent("2025-05-01", new Event("Dentist Appointment", "3:00 PM", "Routine dental check-up"));  
 myCalendar.addEvent("2025-05-02", new Event("Yoga Class", "7:00 AM", "Morning wellness session"));  
 myCalendar.removeEvent("2025-05-01", "Dentist Appointment");  
 myCalendar.displayEvents("2025-05-01");  
 System.*out*.println("\nAll Events:");  
 myCalendar.listAllEvents();  
 }  
}

1. Movie Collection Manager with ArrayList (10 points)

Problem:

Create a movie collection manager that:

Stores movie information (title, director, genre, release year, rating)

Allows adding and removing movies

Supports filtering by genre, director, or release year

Sorts movies by different criteria (title, year, rating)

Requirements:

Use ArrayList to store Movie objects

Implement filtering and sorting with comparators

Display movie information in a formatted table

Explanation:

Use ArrayList to store movie objects and comparators for filtering and sorting.  
  
Solution:

import java.util.\*;  
  
class Movie {  
 String title;  
 String director;  
 String genre;  
 int year;  
 double rating;  
  
 Movie(String title, String director, String genre, int year, double rating) {  
 this.title = title;  
 this.director = director;  
 this.genre = genre;  
 this.year = year;  
 this.rating = rating;  
 }  
}  
  
class MovieManager {  
 List<Movie> movies = new ArrayList<>();  
  
 void addMovie(Movie m) {  
 movies.add(m);  
 }  
  
 void removeMovie(String title) {  
 movies.removeIf(m -> m.title.equalsIgnoreCase(title));  
 }  
  
 List<Movie> filterByGenre(String genre) {  
 return movies.stream().filter(m -> m.genre.equalsIgnoreCase(genre)).toList();  
 }  
  
 void sortByTitle() {  
 movies.sort(Comparator.*comparing*(m -> m.title));  
 }  
  
 void sortByYear() {  
 movies.sort(Comparator.*comparingInt*(m -> m.year));  
 }  
  
 void sortByRating() {  
 movies.sort((a, b) -> Double.*compare*(b.rating, a.rating));  
 }  
  
 void displayMovies() {  
 for (Movie m : movies) {  
 System.*out*.println(m.title + " " + m.director + " " + m.genre + " " + m.year + " " + m.rating);  
 }  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 MovieManager manager = new MovieManager();  
 manager.addMovie(new Movie("Inception", "Christopher Nolan", "Sci-Fi", 2010, 8.8));  
 manager.addMovie(new Movie("Interstellar", "Christopher Nolan", "Sci-Fi", 2014, 8.6));  
 manager.addMovie(new Movie("The Godfather", "Francis Ford Coppola", "Crime", 1972, 9.2));  
 manager.addMovie(new Movie("The Dark Knight", "Christopher Nolan", "Action", 2008, 9.0));  
  
 manager.removeMovie("Interstellar");  
  
 manager.sortByRating();  
 manager.displayMovies();  
  
 System.*out*.println("\nFiltered by genre: Sci-Fi");  
 List<Movie> filtered = manager.filterByGenre("Sci-Fi");  
 for (Movie m : filtered) {  
 System.*out*.println(m.title + " " + m.director + " " + m.genre + " " + m.year + " " + m.rating);  
 }  
 }  
}

1. Implement a simple stack using ArrayDeque that has push, pop, and peek operations.

Explanation: We implement push, pop, and peek using ArrayDeque.

Solution:

import java.util.ArrayDeque;  
import java.util.Deque;  
  
class SimpleStack {  
 Deque<Integer> stack = new ArrayDeque<>();  
  
 void push(int value) {  
 stack.push(value);  
 }  
  
 int pop() {  
 return stack.pop();  
 }  
  
 int peek() {  
 return stack.peek();  
 }  
}  
public class Main {  
 public static void main(String[] args) {  
 SimpleStack s = new SimpleStack();  
 s.push(10);  
 s.push(20);  
 s.push(30);  
  
 System.*out*.println("Top element: " + s.peek());  
 System.*out*.println("Popped: " + s.pop());  
 System.*out*.println("Top element after pop: " + s.peek());  
 }  
}

1. Write a method that returns the sorted unique elements from an array of integers using TreeSet.

Explanation: Use TreeSet to automatically sort and remove duplicates.

Solution:

import java.util.\*;  
  
public class Main {  
 static List<Integer> sortedUnique(int[] arr) {  
 return new TreeSet<>(Arrays.*stream*(arr).boxed().toList()).stream().toList();  
 }  
  
 public static void main(String[] args) {  
 int[] arr = {5, 3, 9, 1, 5, 3, 7};  
 List<Integer> result = *sortedUnique*(arr);  
 System.*out*.println(result);  
 }  
}

1. Write a method that finds and returns common elements between two lists.

Explanation: Use retainAll on two ArrayLists to find common elements.

Solution:

import java.util.\*;  
  
public class Main {  
 static List<Integer> commonElements(List<Integer> a, List<Integer> b) {  
 List<Integer> result = new ArrayList<>(a);  
 result.retainAll(b);  
 return result;  
 }  
  
 public static void main(String[] args) {  
 List<Integer> listA = Arrays.*asList*(1, 2, 3, 4, 5);  
 List<Integer> listB = Arrays.*asList*(4, 5, 6, 7, 8);  
 List<Integer> result = *commonElements*(listA, listB);  
 System.*out*.println(result);  
 }  
}

1. Write code that sorts a list of Product objects (name, category, price) first by category in alphabetical order, then by price from lowest to highest.

Explanation: Use Comparator to sort first by category, then by price.

Solution:

import java.util.\*;  
  
class Product {  
 String name;  
 String category;  
 double price;  
  
 Product(String name, String category, double price) {  
 this.name = name;  
 this.category = category;  
 this.price = price;  
 }  
}  
  
public class Main {  
 static void sortProducts(List<Product> products) {  
 products.sort(Comparator.*comparing*((Product p) -> p.category).thenComparing(p -> p.price));  
 }  
  
 public static void main(String[] args) {  
 List<Product> products = new ArrayList<>();  
 products.add(new Product("Laptop", "Electronics", 800));  
 products.add(new Product("Shirt", "Clothing", 30));  
 products.add(new Product("Smartphone", "Electronics", 600));  
 products.add(new Product("Shoes", "Clothing", 50));  
  
 *sortProducts*(products);  
 for (Product p : products) {  
 System.*out*.println(p.name + " " + p.category + " " + p.price);  
 }  
 }  
}