**Exercise 1: Shuffle a List**

Shuffle the elements of an **ArrayList** using the **shuffle** method from the **Collections** class.

**Solution 1:**

import java.util.ArrayList;

import java.util.Collections;

public class ShuffleList {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<>();

for (int i = 1; i <= 10; i++) {

numbers.add(i);

}

System.out.println("Original List: " + numbers);

// Shuffle the list

Collections.shuffle(numbers);

System.out.println("Shuffled List: " + numbers);

}

}

**Exercise 2: Find the Maximum Element in a Collection**

Find the maximum element in an **ArrayList** using the **max** method from the **Collections** class.

**Solution 2:**

import java.util.ArrayList;

import java.util.Collections;

public class MaxElementInList {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(42);

numbers.add(15);

numbers.add(78);

numbers.add(23);

numbers.add(56);

// Find the maximum element

Integer maxElement = Collections.max(numbers);

System.out.println("Maximum Element: " + maxElement);

}

}

**Exercise 3: Reverse a List**

Reverse the elements of a **LinkedList** using the **reverse** method from the **Collections** class.

**Solution 3:**

import java.util.LinkedList;

import java.util.Collections;

public class ReverseLinkedList {

public static void main(String[] args) {

LinkedList<String> colors = new LinkedList<>();

colors.add("Red");

colors.add("Green");

colors.add("Blue");

colors.add("Yellow");

System.out.println("Original LinkedList: " + colors);

// Reverse the list

Collections.reverse(colors);

System.out.println("Reversed LinkedList: " + colors);

}

}

**Exercise 4: Sort a Set in Natural Order**

Sort a **TreeSet** of strings in natural order using the **sort** method from the **Collections** class.

**Solution 4:**

import java.util.TreeSet;

import java.util.Collections;

public class SortTreeSet {

public static void main(String[] args) {

TreeSet<String> names = new TreeSet<>();

names.add("Alice");

names.add("Charlie");

names.add("Bob");

System.out.println("Original TreeSet: " + names);

// Sort the set in natural order

Collections.sort(new ArrayList<>(names));

System.out.println("Sorted TreeSet: " + names);

}

}

**Exercise 5: Sort a List of User-Defined Objects**

Sort a **List** of user-defined objects (e.g., **Person** class) based on a specific attribute (e.g., age).

**Solution 5:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public int getAge() {

return age;

}

@Override

public String toString() {

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

}

}

public class SortListOfObjects {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Alice", 25));

people.add(new Person("Bob", 30));

people.add(new Person("Charlie", 22));

// Sort the list based on age

Collections.sort(people, Comparator.comparingInt(Person::getAge));

System.out.println("Sorted List of People (by Age): " + people);

}

}

**Exercise 6: Find the Minimum Object in a Collection**

Find the minimum object in a **Set** of user-defined objects (e.g., **Product** class) based on a specific attribute (e.g., price).

**Solution 6:**

import java.util.HashSet;

import java.util.Collections;

import java.util.Set;

class Product {

private String name;

private double price;

public Product(String name, double price) {

this.name = name;

this.price = price;

}

public double getPrice() {

return price;

}

@Override

public String toString() {

return "Product{name='" + name + "', price=" + price + '}';

}

}

public class MinObjectInCollection {

public static void main(String[] args) {

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

products.add(new Product("Laptop", 999.99));

products.add(new Product("Smartphone", 599.99));

products.add(new Product("Tablet", 349.99));

// Find the minimum product based on price

Product minProduct = Collections.min(products, Comparator.comparingDouble(Product::getPrice));

System.out.println("Minimum Product (by Price): " + minProduct);

}

}

**Exercise 7: Shuffle a List of Custom Objects**

Shuffle the elements of a **List** of user-defined objects (e.g., **Card** class representing a deck of cards).

**Solution 7:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Card {

private String suit;

private String rank;

public Card(String suit, String rank) {

this.suit = suit;

this.rank = rank;

}

@Override

public String toString() {

return rank + " of " + suit;

}

}

public class ShuffleListOfObjects {

public static void main(String[] args) {

List<Card> deck = new ArrayList<>();

// Populate the deck with cards...

System.out.println("Original Deck:");

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

// Shuffle the deck

Collections.shuffle(deck);

System.out.println("\nShuffled Deck:");

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

}

}

**Exercise 8: Reverse a List of User-Defined Objects**

Reverse the order of elements in a **List** of user-defined objects (e.g., **Book** class).

**Solution 8:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Book {

private String title;

private String author;

public Book(String title, String author) {

this.title = title;

this.author = author;

}

@Override

public String toString() {

return "Book{title='" + title + "', author='" + author + "'}";

}

}

public class ReverseListOfObjects {

public static void main(String[] args) {

List<Book> books = new ArrayList<>();

books.add(new Book("The Great Gatsby", "F. Scott Fitzgerald"));

books.add(new Book("To Kill a Mockingbird", "Harper Lee"));

books.add(new Book("1984", "George Orwell"));

System.out.println("Original List of Books:");

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

// Reverse the list

Collections.reverse(books);

System.out.println("\nReversed List of Books:");

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

}

}

**Exercise 9: Find the Maximum Object in a Collection**

Find the maximum object in a **List** of user-defined objects (e.g., **Car** class) based on a specific attribute (e.g., price).

**Solution 9:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Car {

private String model;

private double price;

public Car(String model, double price) {

this.model = model;

this.price = price;

}

public double getPrice() {

return price;

}

@Override

public String toString() {

return "Car{model='" + model + "', price=" + price + '}';

}

}

public class MaxObjectInCollection {

public static void main(String[] args) {

List<Car> cars = new ArrayList<>();

cars.add(new Car("Toyota Camry", 25000.0));

cars.add(new Car("Honda Accord", 27000.0));

cars.add(new Car("Ford Mustang", 35000.0));

// Find the maximum car based on price

Car maxCar = Collections.max(cars, Comparator.comparingDouble(Car::getPrice));

System.out.println("Maximum Car (by Price): " + maxCar);

}

}

**Exercise 10: Rotate Elements in a List of User-Defined Objects**

Rotate the elements in a **List** of user-defined objects (e.g., **Person** class) by a specified distance.

**Solution 10:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

@Override

public String toString() {

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

}

}

public class RotateListOfObjects {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Alice", 25));

people.add(new Person("Bob", 30));

people.add(new Person("Charlie", 22));

System.out.println("Original List of People:");

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

// Rotate the list by a specified distance

int distance = 2;

Collections.rotate(people, distance);

System.out.println("\nRotated List of People (by " + distance + " positions):");

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

}

}

**Exercise 11: Custom Sorting of a List of User-Defined Objects**

Create a **List** of user-defined objects (e.g., **Employee** class) and implement a custom comparator to sort the list based on multiple attributes (e.g., first by department, then by salary).

**Solution 11:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Employee {

private String name;

private String department;

private double salary;

public Employee(String name, String department, double salary) {

this.name = name;

this.department = department;

this.salary = salary;

}

@Override

public String toString() {

return "Employee{name='" + name + "', department='" + department + "', salary=" + salary + '}';

}

}

public class CustomSortListOfObjects {

public static void main(String[] args) {

List<Employee> employees = new ArrayList<>();

employees.add(new Employee("Alice", "HR", 50000.0));

employees.add(new Employee("Bob", "Engineering", 60000.0));

employees.add(new Employee("Charlie", "Sales", 55000.0));

System.out.println("Original List of Employees:");

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

// Custom comparator for sorting by department and then by salary

Comparator<Employee> customComparator = Comparator

.comparing(Employee::getDepartment)

.thenComparingDouble(Employee::getSalary);

// Sort the list using the custom comparator

Collections.sort(employees, customComparator);

System.out.println("\nSorted List of Employees (by Department and Salary):");

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

}

}

**Exercise 12: Finding Median in a List of User-Defined Objects**

Create a **List** of user-defined objects (e.g., **Student** class) and find the median based on a specific attribute (e.g., exam scores).

**Solution 12:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Student {

private String name;

private int examScore;

public Student(String name, int examScore) {

this.name = name;

this.examScore = examScore;

}

public int getExamScore() {

return examScore;

}

@Override

public String toString() {

return "Student{name='" + name + "', examScore=" + examScore + '}';

}

}

public class FindMedianListOfObjects {

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student("Alice", 85));

students.add(new Student("Bob", 92));

students.add(new Student("Charlie", 78));

students.add(new Student("David", 90));

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

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

// Sort the list by exam scores

Collections.sort(students, Comparator.comparingInt(Student::getExamScore));

// Find the median (middle element) in the sorted list

int size = students.size();

Student medianStudent = size % 2 == 0 ? students.get(size / 2 - 1) : students.get(size / 2);

System.out.println("\nMedian Student (by Exam Score): " + medianStudent);

}

}

**Exercise 13: Implement a Priority Queue with Custom Comparator**

Create a priority queue using the **PriorityQueue** class and a custom comparator to manage a list of tasks (e.g., **Task** class) based on priority.

**Solution 13:**

import java.util.PriorityQueue;

import java.util.Queue;

class Task {

private String name;

private int priority;

public Task(String name, int priority) {

this.name = name;

this.priority = priority;

}

public int getPriority() {

return priority;

}

@Override

public String toString() {

return "Task{name='" + name + "', priority=" + priority + '}';

}

}

public class PriorityQueueWithCustomComparator {

public static void main(String[] args) {

Queue<Task> taskQueue = new PriorityQueue<>((task1, task2) -> task2.getPriority() - task1.getPriority());

taskQueue.add(new Task("Write code", 2));

taskQueue.add(new Task("Review code", 1));

taskQueue.add(new Task("Test application", 3));

System.out.println("Tasks in Priority Queue (Ordered by Priority):");

while (!taskQueue.isEmpty()) {

System.out.println(taskQueue.poll());

}

}

}

**Exercise 14: Implement a Custom LRU Cache**

Implement a custom LRU (Least Recently Used) cache using a combination of a **LinkedHashMap** and a custom **removeEldestEntry** method in the **Cache** class.

**Solution 14:**

import java.util.LinkedHashMap;

import java.util.Map;

class Cache<K, V> extends LinkedHashMap<K, V> {

private final int maxSize;

public Cache(int maxSize) {

super(maxSize, 0.75f, true);

this.maxSize = maxSize;

}

@Override

protected boolean removeEldestEntry(Map.Entry eldest) {

return size() > maxSize;

}

}

public class LRUCacheImplementation {

public static void main(String[] args) {

Cache<String, String> lruCache = new Cache<>(3);

lruCache.put("1", "One");

lruCache.put("2", "Two");

lruCache.put("3", "Three");

System.out.println("LRU Cache Contents after Initial Insertions:");

System.out.println(lruCache);

lruCache.put("4", "Four"); // This will remove the least recently used entry

System.out.println("\nLRU Cache Contents after Additional Insertion (1 is removed):");

System.out.println(lruCache);

}

}

**Exercise 15: Implement a Thread-Safe Singleton List**

Create a thread-safe singleton list using the **Collections.synchronizedList** method.

**Solution 15:**

import java.util.Collections;

import java.util.List;

public class ThreadSafeSingletonList {

private static List<String> singletonList;

private ThreadSafeSingletonList() {

// Private constructor to prevent instantiation

}

public static List<String> getInstance() {

if (singletonList == null) {

synchronized (ThreadSafeSingletonList.class) {

if (singletonList == null) {

singletonList = Collections.synchronizedList(new java.util.ArrayList<>());

}

}

}

return singletonList;

}

public static void main(String[] args) {

List<String> threadSafeList = ThreadSafeSingletonList.getInstance();

threadSafeList.add("Item 1");

threadSafeList.add("Item 2");

System.out.println("Thread-Safe Singleton List Contents:");

System.out.println(threadSafeList);

}

}

**Exercise 16: Implement a Custom Binary Search on a List of Objects**

Create a utility class (**BinarySearchUtility**) that provides a method to perform binary search on a sorted list of user-defined objects (e.g., **Person** class) based on a specified attribute (e.g., age).

**Solution 16:**

import java.util.Collections;

import java.util.List;

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public int getAge() {

return age;

}

@Override

public String toString() {

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

}

}

class BinarySearchUtility {

public static int binarySearch(List<Person> people, int targetAge) {

int left = 0;

int right = people.size() - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

int midAge = people.get(mid).getAge();

if (midAge == targetAge) {

return mid; // Found the target age at index mid

} else if (midAge < targetAge) {

left = mid + 1; // Search in the right half

} else {

right = mid - 1; // Search in the left half

}

}

return -1; // Target age not found

}

}

public class CustomBinarySearchOnObjects {

public static void main(String[] args) {

List<Person> people = List.of(

new Person("Alice", 25),

new Person("Bob", 30),

new Person("Charlie", 35),

new Person("David", 40)

);

int targetAge = 30;

Collections.sort(people, Comparator.comparingInt(Person::getAge));

int resultIndex = BinarySearchUtility.binarySearch(people, targetAge);

if (resultIndex != -1) {

System.out.println("Person with age " + targetAge + " found at index " + resultIndex);

} else {

System.out.println("Person with age " + targetAge + " not found");

}

}

}

**Exercise 17: Implement a Multi-Key Map using a Composite Key**

Create a multi-key map (**MultiKeyMap**) to store information about students and their courses. Each entry in the map should be identified by a composite key (e.g., a combination of student ID and course code).

**Solution 17:**

import java.util.HashMap;

import java.util.Map;

class MultiKey {

private String studentId;

private String courseCode;

public MultiKey(String studentId, String courseCode) {

this.studentId = studentId;

this.courseCode = courseCode;

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

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

MultiKey multiKey = (MultiKey) obj;

return studentId.equals(multiKey.studentId) && courseCode.equals(multiKey.courseCode);

}

@Override

public int hashCode() {

return studentId.hashCode() + courseCode.hashCode();

}

}

class MultiKeyMap {

private Map<MultiKey, String> map = new HashMap<>();

public void put(String studentId, String courseCode, String courseName) {

map.put(new MultiKey(studentId, courseCode), courseName);

}

public String get(String studentId, String courseCode) {

return map.get(new MultiKey(studentId, courseCode));

}

}

public class MultiKeyMapExample {

public static void main(String[] args) {

MultiKeyMap studentCourseMap = new MultiKeyMap();

// Adding entries to the map

studentCourseMap.put("101", "MATH101", "Mathematics 101");

studentCourseMap.put("101", "CS101", "Computer Science 101");

studentCourseMap.put("102", "PHYS101", "Physics 101");

// Retrieving information from the map

System.out.println("Course for student 101, MATH101: " + studentCourseMap.get("101", "MATH101"));

System.out.println("Course for student 102, PHYS101: " + studentCourseMap.get("102", "PHYS101"));

}

}

**Exercise 18: Implement a Voting System with User-Defined Classes**

Create a voting system using a combination of user-defined classes (**Voter**, **Candidate**, and **Election**) and the **Collections** class to manage votes and determine the winner.

**Solution 18:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Voter {

private String voterId;

public Voter(String voterId) {

this.voterId = voterId;

}

public String getVoterId() {

return voterId;

}

}

class Candidate {

private String candidateId;

private String name;

public Candidate(String candidateId, String name) {

this.candidateId = candidateId;

this.name = name;

}

public String getCandidateId() {

return candidateId;

}

public String getName() {

return name;

}

}

class Election {

private List<Candidate> candidates;

private List<Voter> votes;

public Election(List<Candidate> candidates) {

this.candidates = candidates;

this.votes = new ArrayList<>();

}

public void vote(Voter voter, Candidate candidate) {

votes.add(voter);

System.out.println("Voter " + voter.getVoterId() + " voted for " + candidate.getName());

}

public Candidate determineWinner() {

// Shuffle candidates to simulate random voting order

Collections.shuffle(candidates);

// Simulate voting (in a real scenario, you'd count actual votes)

int[] voteCount = new int[candidates.size()];

for (Voter vote : votes) {

int candidateIndex = (int) (Math.random() \* candidates.size());

voteCount[candidateIndex]++;

}

// Find the candidate with the maximum votes

int maxVotes = 0;

int winningCandidateIndex = -1;

for (int i = 0; i < candidates.size(); i++) {

if (voteCount[i] > maxVotes) {

maxVotes = voteCount[i];

winningCandidateIndex = i;

}

}

return (winningCandidateIndex != -1) ? candidates.get(winningCandidateIndex) : null;

}

}

public class VotingSystem {

public static void main(String[] args) {

Candidate candidate1 = new Candidate("C001", "Alice");

Candidate candidate2 = new Candidate("C002", "Bob");

Candidate candidate3 = new Candidate("C003", "Charlie");

List<Candidate> candidates = new ArrayList<>(List.of(candidate1, candidate2, candidate3));

Election election = new Election(candidates);

Voter voter1 = new Voter("V001");

Voter voter2 = new Voter("V002");

Voter voter3 = new Voter("V003");

election.vote(voter1, candidate1);

election.vote(voter2, candidate2);

election.vote(voter3, candidate3);

Candidate winner = election.determineWinner();

System.out.println("\nWinner: " + winner.getName());

}

}

**Exercise 19: Implement a Library System with User-Defined Classes**

Create a library system using user-defined classes (**Book**, **Library**, and **Member**) and the **Collections class for sorting books:**

**Solution 19:**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Book {

private String bookId;

private String title;

private boolean available;

public Book(String bookId, String title) {

this.bookId = bookId;

this.title = title;

this.available = true;

}

public String getBookId() {

return bookId;

}

public String getTitle() {

return title;

}

public boolean isAvailable() {

return available;

}

public void setAvailable(boolean available) {

this.available = available;

}

}

class Member {

private String memberId;

private String name;

public Member(String memberId, String name) {

this.memberId = memberId;

this.name = name;

}

public String getMemberId() {

return memberId;

}

public String getName() {

return name;

}

}

class Library {

private List<Book> books;

private List<Member> members;

public Library() {

this.books = new ArrayList<>();

this.members = new ArrayList<>();

}

public void addBook(Book book) {

books.add(book);

}

public void addMember(Member member) {

members.add(member);

}

public void borrowBook(Member member, Book book) {

if (book.isAvailable()) {

System.out.println(member.getName() + " borrowed " + book.getTitle());

book.setAvailable(false);

} else {

System.out.println("Sorry, " + book.getTitle() + " is not available for borrowing.");

}

}

public void displayAvailableBooks() {

// Sort the books by title using Collections.sort

Collections.sort(books, Comparator.comparing(Book::getTitle));

System.out.println("\nAvailable Books:");

for (Book book : books) {

if (book.isAvailable()) {

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

}

}

}

}

public class LibrarySystem {

public static void main(String[] args) {

Book book1 = new Book("B001", "The Great Gatsby");

Book book2 = new Book("B002", "To Kill a Mockingbird");

Book book3 = new Book("B003", "1984");

Member member1 = new Member("M001", "Alice");

Member member2 = new Member("M002", "Bob");

Library library = new Library();

library.addBook(book1);

library.addBook(book2);

library.addBook(book3);

library.addMember(member1);

library.addMember(member2);

library.borrowBook(member1, book1);

library.borrowBook(member2, book2);

library.borrowBook(member1, book3);

library.displayAvailableBooks();

}

}

**Exercise 20: University Course Schedule with User-Defined Classes**

Create a university course schedule system with classes representing **Course**, **Student**, and **Schedule**. Use the **Collections** class to sort the schedule based on different criteria such as course name, student name, and class time.

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Course {

private String courseCode;

private String courseName;

public Course(String courseCode, String courseName) {

this.courseCode = courseCode;

this.courseName = courseName;

}

public String getCourseCode() {

return courseCode;

}

public String getCourseName() {

return courseName;

}

}

class Student {

private String studentId;

private String name;

public Student(String studentId, String name) {

this.studentId = studentId;

this.name = name;

}

public String getStudentId() {

return studentId;

}

public String getName() {

return name;

}

}

class Schedule {

private List<Course> courses;

private List<Student> students;

public Schedule() {

this.courses = new ArrayList<>();

this.students = new ArrayList<>();

}

public void addCourse(Course course) {

courses.add(course);

}

public void addStudent(Student student) {

students.add(student);

}

public void displaySchedule() {

// Sort the schedule by course name

Collections.sort(courses, Comparator.comparing(Course::getCourseName));

System.out.println("University Course Schedule:");

for (Course course : courses) {

System.out.println("Course: " + course.getCourseName() + " (" + course.getCourseCode() + ")");

}

// Sort the schedule by student name

Collections.sort(students, Comparator.comparing(Student::getName));

System.out.println("\nStudents:");

for (Student student : students) {

System.out.println("Student: " + student.getName() + " (ID: " + student.getStudentId() + ")");

}

}

}

public class UniversityScheduleSystem {

public static void main(String[] args) {

Course course1 = new Course("C101", "Introduction to Computer Science");

Course course2 = new Course("MATH201", "Calculus II");

Course course3 = new Course("ENG101", "English Composition");

Student student1 = new Student("S001", "Alice");

Student student2 = new Student("S002", "Bob");

Student student3 = new Student("S003", "Charlie");

Schedule schedule = new Schedule();

schedule.addCourse(course1);

schedule.addCourse(course2);

schedule.addCourse(course3);

schedule.addStudent(student1);

schedule.addStudent(student2);

schedule.addStudent(student3);

schedule.displaySchedule();

}

}

**Exercise 21: Messaging System with User-Defined Classes**

Consider a messaging system with classes representing **User**, **Message**, and **MessagingSystem**. Use the **Collections** class and Java streams to implement the following operations:

1. **Sorting Messages:**
   * Implement a method in the **MessagingSystem** class to display messages sorted by sender names.
   * Implement another method to display messages sorted by receiver names.
2. **Most Recent Messages:**
   * Implement a method to display the most recent messages. Allow the user to specify the number of most recent messages to display.
3. **User-Specific Messages:**
   * Implement a method to display messages for a specific user. The method should take a **User** object as a parameter and display all messages where the user is either the sender or the receiver.

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.stream.Collectors;

class User {

private String userId;

private String name;

public User(String userId, String name) {

this.userId = userId;

this.name = name;

}

public String getUserId() {

return userId;

}

public String getName() {

return name;

}

}

class Message {

private String messageId;

private User sender;

private User receiver;

private String content;

private long timestamp;

public Message(String messageId, User sender, User receiver, String content, long timestamp) {

this.messageId = messageId;

this.sender = sender;

this.receiver = receiver;

this.content = content;

this.timestamp = timestamp;

}

public String getMessageId() {

return messageId;

}

public User getSender() {

return sender;

}

public User getReceiver() {

return receiver;

}

public String getContent() {

return content;

}

public long getTimestamp() {

return timestamp;

}

}

class MessagingSystem {

private List<User> users;

private List<Message> messages;

public MessagingSystem() {

this.users = new ArrayList<>();

this.messages = new ArrayList<>();

}

public void addUser(User user) {

users.add(user);

}

public void sendMessage(Message message) {

messages.add(message);

}

public void displayMessagesSortedBySender() {

// Sort messages by sender name

Collections.sort(messages, Comparator.comparing(message -> message.getSender().getName()));

System.out.println("Messages Sorted by Sender:");

for (Message message : messages) {

System.out.println("Sender: " + message.getSender().getName() +

" | Receiver: " + message.getReceiver().getName() +

" | Content: " + message.getContent() +

" | Timestamp: " + message.getTimestamp());

}

}

public void displayMessagesSortedByReceiver() {

// Sort messages by receiver name

Collections.sort(messages, Comparator.comparing(message -> message.getReceiver().getName()));

System.out.println("\nMessages Sorted by Receiver:");

for (Message message : messages) {

System.out.println("Sender: " + message.getSender().getName() +

" | Receiver: " + message.getReceiver().getName() +

" | Content: " + message.getContent() +

" | Timestamp: " + message.getTimestamp());

}

}

public void displayMostRecentMessages(int count) {

// Display the most recent messages

List<Message> recentMessages = messages.stream()

.sorted(Comparator.comparingLong(Message::getTimestamp).reversed())

.limit(count)

.collect(Collectors.toList());

System.out.println("\nMost Recent Messages:");

for (Message message : recentMessages) {

System.out.println("Sender: " + message.getSender().getName() +

" | Receiver: " + message.getReceiver().getName() +

" | Content: " + message.getContent() +

" | Timestamp: " + message.getTimestamp());

}

}

public void displayMessagesForUser(User user) {

// Display messages for a specific user

List<Message> userMessages = messages.stream()

.filter(message -> message.getSender().equals(user) || message.getReceiver().equals(user))

.collect(Collectors.toList());

System.out.println("\nMessages for User " + user.getName() + ":");

for (Message message : userMessages) {

System.out.println("Sender: " + message.getSender().getName() +

" | Receiver: " + message.getReceiver().getName() +

" | Content: " + message.getContent() +

" | Timestamp: " + message.getTimestamp());

}

}

}

public class EnhancedMessagingSystemExample {

public static void main(String[] args) {

User user1 = new User("U001", "Alice");

User user2 = new User("U002", "Bob");

User user3 = new User("U003", "Charlie");

Message message1 = new Message("M001", user1, user2, "Hello Bob!", System.currentTimeMillis());

Message message2 = new Message("M002", user2, user1, "Hi Alice!", System.currentTimeMillis() + 1000);

Message message3 = new Message("M003", user3, user1, "Greetings Alice!", System.currentTimeMillis() + 2000);

MessagingSystem messagingSystem = new MessagingSystem();

messagingSystem.addUser(user1);

messagingSystem.addUser(user2);

messagingSystem.addUser(user3);

messagingSystem.sendMessage(message1);

messagingSystem.sendMessage(message2);

messagingSystem.sendMessage(message3);

messagingSystem.displayMessagesSortedBySender();

messagingSystem.displayMessagesSortedByReceiver();

messagingSystem.displayMostRecentMessages(2);

messagingSystem.displayMessagesForUser(user1);

}

}

**Exercise 22: Inventory Management System with User-Defined Classes**

Consider a simplified inventory management system with classes representing **Product**, **Inventory**, and **InventoryManager**. Implement the following operations:

1. **Adding Products:**
   * Create a method in the **Inventory** class to add a product to the inventory.
2. **Updating Quantity:**
   * Create a method in the **Inventory** class to update the quantity of a specific product based on its ID.
3. **Displaying Inventory:**
   * Implement a method in the **Inventory** class to display the current inventory information, including product ID, name, price, and quantity.
4. **Sorting Products:**
   * Create a method in the **Inventory** class to sort products by their names in ascending order using the **Collections** class.
   * Implement another method to sort products by their quantities in descending order using the **Collections** class.
5. **Searching for a Product:**
   * Implement a method in the **Inventory** class to search for a product by its name. Return the product information if found; otherwise, indicate that the product is not in the inventory.
6. **Calculating Total Inventory Value:**
   * Create a method in the **Inventory** class to calculate the total value of the inventory (sum of the value of each product, where value = price \* quantity).
7. **Removing a Product:**
   * Implement a method in the **Inventory** class to remove a product from the inventory based on its ID.

Use the provided classes to structure your solution. Create sample products and test the implemented methods in the **InventoryManager** class.

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Product {

private String productId;

private String name;

private double price;

private int quantity;

public Product(String productId, String name, double price, int quantity) {

this.productId = productId;

this.name = name;

this.price = price;

this.quantity = quantity;

}

public String getProductId() {

return productId;

}

public String getName() {

return name;

}

public double getPrice() {

return price;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getValue() {

return price \* quantity;

}

}

class Inventory {

private List<Product> products;

public Inventory() {

this.products = new ArrayList<>();

}

public void addProduct(Product product) {

products.add(product);

}

public void updateQuantity(String productId, int newQuantity) {

for (Product product : products) {

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

product.setQuantity(newQuantity);

break;

}

}

}

public void displayInventory() {

System.out.println("Inventory Information:");

for (Product product : products) {

System.out.println("Product ID: " + product.getProductId() +

" | Name: " + product.getName() +

" | Price: $" + product.getPrice() +

" | Quantity: " + product.getQuantity() +

" | Value: $" + product.getValue());

}

}

public void sortProductsByName() {

Collections.sort(products, Comparator.comparing(Product::getName));

}

public void sortProductsByQuantity() {

Collections.sort(products, Comparator.comparingInt(Product::getQuantity).reversed());

}

public Product searchProductByName(String productName) {

for (Product product : products) {

if (product.getName().equalsIgnoreCase(productName)) {

return product;

}

}

return null;

}

public double calculateTotalInventoryValue() {

double totalValue = 0;

for (Product product : products) {

totalValue += product.getValue();

}

return totalValue;

}

public void removeProduct(String productId) {

products.removeIf(product -> product.getProductId().equals(productId));

}

}

public class InventoryManager {

public static void main(String[] args) {

Product product1 = new Product("P001", "Laptop", 899.99, 10);

Product product2 = new Product("P002", "Smartphone", 499.99, 20);

Product product3 = new Product("P003", "Headphones", 79.99, 30);

Inventory inventory = new Inventory();

inventory.addProduct(product1);

inventory.addProduct(product2);

inventory.addProduct(product3);

// Display initial inventory

System.out.println("Initial Inventory:");

inventory.displayInventory();

// Sort products by name and display

inventory.sortProductsByName();

System.out.println("\nInventory after sorting by name:");

inventory.displayInventory();

// Update quantity for a product

inventory.updateQuantity("P002", 15);

// Display updated inventory

System.out.println("\nInventory after updating quantity:");

inventory.displayInventory();

// Sort products by quantity and display

inventory.sortProductsByQuantity();

System.out.println("\nInventory after sorting by quantity:");

inventory.displayInventory();

// Search for a product by name

String searchProductName = "Laptop";

Product searchedProduct = inventory.searchProductByName(searchProductName);

if (searchedProduct != null) {

System.out.println("\nProduct Found: " + searchedProduct.getName());

} else {

System.out.println("\nProduct Not Found: " + searchProductName);

}

// Calculate total inventory value

double totalValue = inventory.calculateTotalInventoryValue();

System.out.println("\nTotal Inventory Value: $" + totalValue);

// Remove a product and display the updated inventory

String productIdToRemove = "P002";

inventory.removeProduct(productIdToRemove);

System.out.println("\nInventory after removing product (ID: " + productIdToRemove + "):");

inventory.displayInventory();

}

}

**Exercise 23: Library Management System with User-Defined Classes and Collections**

Consider a library management system with classes representing **Book**, **Library**, and **LibraryManager**. Implement the following operations:

1. **Adding Books:**
   * Create a method in the **Library** class to add a book to the library.
2. **Displaying Books:**
   * Implement a method in the **Library** class to display the current library book information, including book ID, title, author, and availability status.
3. **Borrowing and Returning Books:**
   * Create methods in the **Book** class to borrow and return books. Update the availability status accordingly.
4. **Sorting Books:**
   * Implement methods in the **Library** class to sort books by title and by author using the **Collections** class.

Use the provided classes to structure your solution. Create sample books and test the implemented methods in the **LibraryManager** class.

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

class Book {

private String bookId;

private String title;

private String author;

private boolean available;

public Book(String bookId, String title, String author) {

this.bookId = bookId;

this.title = title;

this.author = author;

this.available = true;

}

public String getBookId() {

return bookId;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public boolean isAvailable() {

return available;

}

public void borrowBook() {

if (available) {

available = false;

System.out.println("Book borrowed successfully: " + title);

} else {

System.out.println("Book is not available for borrowing: " + title);

}

}

public void returnBook() {

if (!available) {

available = true;

System.out.println("Book returned successfully: " + title);

} else {

System.out.println("Book is already available: " + title);

}

}

}

class Library {

private List<Book> books;

public Library() {

this.books = new ArrayList<>();

}

public void addBook(Book book) {

books.add(book);

}

public void displayBooks() {

System.out.println("Library Books:");

for (Book book : books) {

System.out.println("Book ID: " + book.getBookId() +

" | Title: " + book.getTitle() +

" | Author: " + book.getAuthor() +

" | Available: " + (book.isAvailable() ? "Yes" : "No"));

}

}

public void sortBooksByTitle() {

Collections.sort(books, (b1, b2) -> b1.getTitle().compareToIgnoreCase(b2.getTitle()));

}

public void sortBooksByAuthor() {

Collections.sort(books, (b1, b2) -> b1.getAuthor().compareToIgnoreCase(b2.getAuthor()));

}

}

public class LibraryManager {

public static void main(String[] args) {

Book book1 = new Book("B001", "The Great Gatsby", "F. Scott Fitzgerald");

Book book2 = new Book("B002", "To Kill a Mockingbird", "Harper Lee");

Book book3 = new Book("B003", "1984", "George Orwell");

Library library = new Library();

library.addBook(book1);

library.addBook(book2);

library.addBook(book3);

// Display initial library books

System.out.println("Initial Library Books:");

library.displayBooks();

// Borrow a book and display updated status

book2.borrowBook();

System.out.println("\nLibrary Books after borrowing:");

library.displayBooks();

// Return a book and display updated status

book1.returnBook();

System.out.println("\nLibrary Books after returning:");

library.displayBooks();

// Sort books by title and display

library.sortBooksByTitle();

System.out.println("\nLibrary Books after sorting by title:");

library.displayBooks();

// Sort books by author and display

library.sortBooksByAuthor();

System.out.println("\nLibrary Books after sorting by author:");

library.displayBooks();

}

}

**Exercise 24: Movie Database System with User-Defined Classes, Collections, and with Add, Show, Search, Sort, Update and Remove Options**

Consider a movie database system with classes representing **Movie**, **MovieDatabase**, and **MovieManager**. Implement the following operations:

1. **Adding Movies:**
   * Create a method in the **MovieDatabase** class to add a movie to the database.
2. **Displaying Movies:**
   * Implement a method in the **MovieDatabase** class to display the current movie information.
3. **Searching for Movies:**
   * Create methods in the **MovieDatabase** class to search for movies by director and by rating.
4. **Sorting Movies:**
   * Implement methods in the **MovieDatabase** class to sort movies by title, release year, and rating using the **Collections** class.
5. **Updating Movie Rating:**
   * Create a method in the **MovieDatabase** class to update the rating of a movie by its title.
6. **Removing Movie:**
   * Implement a method in the **MovieDatabase** class to remove a movie by its title.

Use the provided classes to structure your solution. Create sample movies and test the implemented methods in the **MovieManager** class.

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Movie {

private String title;

private String director;

private int releaseYear;

private double rating;

public Movie(String title, String director, int releaseYear, double rating) {

this.title = title;

this.director = director;

this.releaseYear = releaseYear;

this.rating = rating;

}

public String getTitle() {

return title;

}

public String getDirector() {

return director;

}

public int getReleaseYear() {

return releaseYear;

}

public double getRating() {

return rating;

}

public void setRating(double rating) {

this.rating = rating;

}

@Override

public String toString() {

return "Movie{" +

"title='" + title + '\'' +

", director='" + director + '\'' +

", releaseYear=" + releaseYear +

", rating=" + rating +

'}';

}

}

class MovieDatabase {

private List<Movie> movies;

public MovieDatabase() {

this.movies = new ArrayList<>();

}

public void addMovie(Movie movie) {

movies.add(movie);

}

public List<Movie> getMovies() {

return movies;

}

public List<Movie> searchMoviesByDirector(String director) {

List<Movie> result = new ArrayList<>();

for (Movie movie : movies) {

if (movie.getDirector().equalsIgnoreCase(director)) {

result.add(movie);

}

}

return result;

}

public List<Movie> searchMoviesByRating(double minRating) {

List<Movie> result = new ArrayList<>();

for (Movie movie : movies) {

if (movie.getRating() >= minRating) {

result.add(movie);

}

}

return result;

}

public void displayMovies() {

System.out.println("Movie Database:");

for (Movie movie : movies) {

System.out.println(movie);

}

}

public void sortMoviesByTitle() {

Collections.sort(movies, Comparator.comparing(Movie::getTitle));

}

public void sortMoviesByReleaseYear() {

Collections.sort(movies, Comparator.comparingInt(Movie::getReleaseYear));

}

public void sortMoviesByRating() {

Collections.sort(movies, Comparator.comparingDouble(Movie::getRating).reversed());

}

public void updateMovieRating(String title, double newRating) {

for (Movie movie : movies) {

if (movie.getTitle().equalsIgnoreCase(title)) {

movie.setRating(newRating);

System.out.println("Rating updated successfully for " + title);

return;

}

}

System.out.println("Movie not found: " + title);

}

public void removeMovie(String title) {

movies.removeIf(movie -> movie.getTitle().equalsIgnoreCase(title));

System.out.println("Movie removed successfully: " + title);

}

}

public class MovieManager {

public static void main(String[] args) {

Movie movie1 = new Movie("Inception", "Christopher Nolan", 2010, 8.8);

Movie movie2 = new Movie("The Shawshank Redemption", "Frank Darabont", 1994, 9.3);

Movie movie3 = new Movie("The Dark Knight", "Christopher Nolan", 2008, 9.0);

MovieDatabase movieDatabase = new MovieDatabase();

movieDatabase.addMovie(movie1);

movieDatabase.addMovie(movie2);

movieDatabase.addMovie(movie3);

// Display initial movie database

System.out.println("Initial Movie Database:");

movieDatabase.displayMovies();

// Search for movies by director

String searchDirector = "Christopher Nolan";

List<Movie> moviesByDirector = movieDatabase.searchMoviesByDirector(searchDirector);

System.out.println("\nMovies directed by " + searchDirector + ":");

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

// Search for movies by rating

double minRating = 9.0;

List<Movie> highlyRatedMovies = movieDatabase.searchMoviesByRating(minRating);

System.out.println("\nHighly rated movies (rating >= " + minRating + "):");

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

// Sort movies by title and display

movieDatabase.sortMoviesByTitle();

System.out.println("\nMovie Database after sorting by title:");

movieDatabase.displayMovies();

// Sort movies by release year and display

movieDatabase.sortMoviesByReleaseYear();

System.out.println("\nMovie Database after sorting by release year:");

movieDatabase.displayMovies();

// Sort movies by rating and display

movieDatabase.sortMoviesByRating();

System.out.println("\nMovie Database after sorting by rating:");

movieDatabase.displayMovies();

// Update movie rating and display updated database

String movieToUpdate = "The Dark Knight";

double newRating = 9.5;

movieDatabase.updateMovieRating(movieToUpdate, newRating);

System.out.println("\nMovie Database after updating rating:");

movieDatabase.displayMovies();

// Remove a movie and display updated database

String movieToRemove = "The Shawshank Redemption";

movieDatabase.removeMovie(movieToRemove);

System.out.println("\nMovie Database after removing movie:");

movieDatabase.displayMovies();

}

}