LAB 7: JAVA COLLECTION

Part 1 - HashSet in Java:

Exercise 1:

Write a program that prompts the user to enter a series of integers and stores them in a HashSet. Print the elements of the HashSet in the order they were entered.

Exercise 2:

Write a program that prompts the user to enter a sentence and stores the unique words in a HashSet. Print the set of unique words.

Exercise 3:

Write a program that creates a HashSet of custom objects and checks whether a specific object exists in the HashSet. Prompt the user to enter the attributes of the object and check its existence. Print an appropriate message indicating the result.

Exercise 4:

Write a program that creates a HashSet of Integer objects and finds the smallest and largest numbers in the HashSet. Print the smallest and largest numbers.

Exercise 5:

Write a program that prompts the user to enter a series of names and stores them in a HashSet. Print the number of unique names entered.

Exercise 6:

Write a program that creates a HashSet of strings and removes a specific string from the HashSet. Prompt the user to enter a string and remove it from the HashSet. Print the HashSet after removing the string.

Exercise 7:

Write a program that creates a HashSet of Double objects and calculates the sum of all the elements in the HashSet. Print the sum.

Exercise 8:

Write a program that creates a HashSet of strings and checks whether it is empty or not. Print an appropriate message indicating the result.

Exercise 9:

Write a program that prompts the user to enter a series of numbers and stores them in a HashSet. Print the distinct even numbers from the HashSet.

Exercise 10:

Write a program that creates a HashSet of strings and finds the longest string in the HashSet. Print the longest string.

Part 2 - HashSet with object-oriented programming

Exercise 1:

Create a class called Student with attributes like name, roll number, and age. Implement the equals() and hashCode() methods in the Student class to ensure proper comparison and hashing. Create a HashSet of Student objects and add multiple students to it. Verify if the HashSet contains a specific student based on roll number.

Exercise 2:

Create a class called Employee with attributes like id, name, and department. Implement the Comparable interface in the Employee class to define the natural ordering based on employee id. Create a HashSet of Employee objects and add multiple employees to it. Print the employees in ascending order based on their ids using the HashSet.

Exercise 3:

Create a class called Product with attributes like name, price, and quantity. Override the equals() and hashCode() methods in the Product class to compare products based on their names. Create a HashSet of Product objects and add multiple products to it. Verify if a specific product exists in the HashSet based on its name.

Exercise 4:

Create a class called Book with attributes like title, author, and ISBN. Implement the Comparable interface in the Book class to define the natural ordering based on the book title. Create a HashSet of Book objects and add multiple books to it. Print the books in alphabetical order based on their titles using the HashSet.

Exercise 5:

Create a class called Song with attributes like title, artist, and duration. Override the equals() and hashCode() methods in the Song class to compare songs based on their titles and artists. Create a HashSet of Song objects and add multiple songs to it. Remove duplicate songs from the HashSet and print the unique set of songs.

Exercise 6:

Create a class called Customer with attributes like id, name, and email. Override the equals() and hashCode() methods in the Customer class to compare customers based on their email addresses. Create a HashSet of Customer objects and add multiple customers to it. Verify if a specific customer exists in the HashSet based on their email address.

Exercise 7:

Create a class called Product with attributes like name, price, and category. Implement the Comparable interface in the Product class to define the natural ordering based on the product price. Create a HashSet of Product objects and add multiple products to it. Find the cheapest and most expensive products in the HashSet using the HashSet.

Exercise 8:

Create a class called Animal with attributes like name and age. Implement the Comparable interface in the Animal class to define the natural ordering based on the animal age. Create a HashSet of Animal objects and add multiple animals to it. Print the animals in ascending order based on their ages using the HashSet.

Exercise 9:

Create a class called Student with attributes like roll number, name, and marks. Implement the Comparable interface in the Student class to define the natural ordering based on student marks. Create a HashSet of Student objects and add multiple students to it. Calculate the average marks of all the students in the HashSet using the HashSet.

Exercise 10:

Create a class called Product with attributes like name, price, and quantity. Override the equals() and hashCode() methods in the Product class to compare products based on their names. Create a HashSet of Product objects and add multiple products to it. Remove a specific product from the HashSet based on its name and print the updated HashSet.

Part 3 - HashMap in Java

Exercise 1:

Write a program that creates a HashMap to store the names of students as keys and their corresponding grades as values. Add several key-value pairs to the HashMap and print the elements.

Exercise 2:

Write a program that creates a HashMap to store the names of countries as keys and their capitals as values. Prompt the user to enter a country name and retrieve the corresponding capital from the HashMap. Print an appropriate message indicating the capital.

Exercise 3:

Write a program that creates a HashMap to store the names of books as keys and their authors as values. Prompt the user to enter a book name and retrieve the corresponding author from the HashMap. Print an appropriate message indicating the author.

Exercise 4:

Write a program that creates a HashMap to store the names of fruits as keys and their quantities as values. Prompt the user to enter a fruit name and update its quantity in the HashMap. Print the updated HashMap.

Exercise 5:

Write a program that creates a HashMap to store the names of cities as keys and their populations as values. Prompt the user to enter a city name and retrieve the corresponding population from the HashMap. Print an appropriate message indicating the population.

Exercise 6:

Write a program that creates a HashMap to store the names of students as keys and their favorite subjects as values. Prompt the user to enter a student name and retrieve the corresponding favorite subject from the HashMap. Print an appropriate message indicating the subject.

Exercise 7:

Write a program that creates a HashMap to store the names of countries as keys and their populations as values. Find and print the country with the highest population from the HashMap.

Exercise 8:

Write a program that creates a HashMap to store the names of countries as keys and their official languages as values. Prompt the user to enter a country name and retrieve the corresponding official language from the HashMap. Print an appropriate message indicating the language.

Exercise 9:

Write a program that creates a HashMap to store the names of cities as keys and their average temperatures as values. Calculate and print the average temperature of all the cities in the HashMap.

Exercise 10:

Write a program that creates a HashMap to store the names of fruits as keys and their prices as values. Calculate and print the total cost of all the fruits in the HashMap.

Part 4 - HashMap with object-oriented programming:

Exercise 1:

Create a class called Student with attributes like roll number, name, and age. Implement the equals() and hashCode() methods in the Student class to ensure proper comparison and hashing. Create a HashMap with roll number as the key and Student object as the value. Add multiple students to the HashMap and retrieve a student object based on a specific roll number.

Exercise 2:

Create a class called Employee with attributes like id, name, and department. Implement the equals() and hashCode() methods in the Employee class to ensure proper comparison and hashing. Create a HashMap with id as the key and Employee object as the value. Add multiple employees to the HashMap and retrieve an employee object based on a specific id.

Exercise 3:

Create a class called Product with attributes like name, price, and quantity. Implement the equals() and hashCode() methods in the Product class to ensure proper comparison and hashing. Create a

HashMap with name as the key and Product object as the value. Add multiple products to the HashMap and retrieve a product object based on a specific name.

Exercise 4:

Create a class called Book with attributes like title, author, and ISBN. Implement the equals() and hashCode() methods in the Book class to ensure proper comparison and hashing. Create a HashMap with ISBN as the key and Book object as the value. Add multiple books to the HashMap and retrieve a book object based on a specific ISBN.

Exercise 5:

Create a class called Customer with attributes like id, name, and email. Implement the equals() and hashCode() methods in the Customer class to ensure proper comparison and hashing. Create a HashMap with email as the key and Customer object as the value. Add multiple customers to the HashMap and retrieve a customer object based on a specific email.

Exercise 6:

Create a class called Animal with attributes like name and age. Implement the equals() and hashCode() methods in the Animal class to ensure proper comparison and hashing. Create a HashMap with name as the key and Animal object as the value. Add multiple animals to the HashMap and retrieve an animal object based on a specific name.

Exercise 7:

Create a class called Order with attributes like orderNumber, customer, and totalAmount. Implement the equals() and hashCode() methods in the Order class to ensure proper comparison and hashing. Create a HashMap with orderNumber as the key and Order object as the value. Add multiple orders to the HashMap and retrieve an order object based on a specific orderNumber.

Exercise 8:

Create a class called Product with attributes like name, category, and price. Implement the equals() and hashCode() methods in the Product class to ensure proper comparison and hashing. Create a HashMap with category as the key and a List of Product objects as the value. Add multiple products to the HashMap based on their categories and retrieve the list of products for a specific category.

Exercise 9:

Create a class called Employee with attributes like id, name, and department. Implement the equals() and hashCode() methods in the Employee class to ensure proper comparison and hashing. Create a HashMap with department as the key and a Set of Employee objects as the value. Add multiple employees to the HashMap based on their departments and retrieve the set of employees for a specific department.

Exercise 10:

Create a class called Movie with attributes like title, genre, and releaseYear. Implement the equals() and hashCode() methods in the Movie class to ensure proper comparison and hashing. Create a HashMap with genre as the key and a PriorityQueue of Movie objects as the value. Add multiple movies to the HashMap based on their genres and retrieve the priority queue of movies for a specific genre.

Part 5 - TreeMap

Exercise 1:

Write a program that creates a TreeMap of String keys and Integer values. Add some key-value pairs to the TreeMap and print the elements in ascending order of the keys.

Exercise 2:

Write a program that prompts the user to enter a sentence and counts the occurrences of each word using a TreeMap. Print the word frequencies in alphabetical order.

Exercise 3:

Write a program that creates a TreeMap of custom objects that implement the Comparable interface. Add several objects to the TreeMap and print them in sorted order based on the custom sorting criteria.

Exercise 4:

Write a program that creates a TreeMap of Integer keys and String values. Add some key-value pairs to the TreeMap and print the values in ascending order of the keys.

Exercise 5:

Write a program that creates a TreeMap of String keys and stores the names of students as keys and their scores as values. Prompt the user to enter a student name and retrieve the corresponding score from the TreeMap. Print an appropriate message indicating the score.

Exercise 6:

Write a program that creates a TreeMap of Integer keys and calculates the sum of all the values in the TreeMap. Print the sum.

Exercise 7:

Write a program that creates a TreeMap of String keys and checks whether a specific key exists in the TreeMap. Prompt the user to enter a key and print an appropriate message indicating the result.

Exercise 8:

Write a program that creates a TreeMap of String keys and removes a specific key-value pair from the TreeMap. Prompt the user to enter a key and remove the corresponding entry from the TreeMap. Print the TreeMap after removing the entry.

Exercise 9:

Write a program that creates a TreeMap of Integer keys and finds the smallest and largest keys in the TreeMap. Print the smallest and largest keys.

Exercise 10:

Write a program that creates a TreeMap of custom objects as keys and counts the occurrences

Part 6 - TreeMap with OOP

Exercise 1:

Create a class called Student with attributes like roll number, name, and age. Implement the Comparable interface in the Student class to define the natural ordering based on the student's roll number. Create a TreeMap with roll number as the key and Student object as the value. Add multiple students to the TreeMap and print the students in ascending order of their roll numbers.

Exercise 2:

Create a class called Employee with attributes like id, name, and department. Implement the Comparable interface in the Employee class to define the natural ordering based on the employee's id. Create a TreeMap with id as the key and Employee object as the value. Add multiple employees to the TreeMap and retrieve an employee object based on a specific id.

Exercise 3:

Create a class called Product with attributes like name, price, and quantity. Implement the Comparable interface in the Product class to define the natural ordering based on the product's name. Create a TreeMap with name as the key and Product object as the value. Add multiple products to the TreeMap and retrieve a product object based on a specific name.

Exercise 4:

Create a class called Book with attributes like title, author, and publication year. Implement the Comparable interface in the Book class to define the natural ordering based on the book's publication year. Create a TreeMap with publication year as the key and Book object as the value. Add multiple books to the TreeMap and retrieve a book object based on a specific publication year.

Exercise 5:

Create a class called Customer with attributes like id, name, and email. Implement the Comparable interface in the Customer class to define the natural ordering based on the customer's id. Create a TreeMap with id as the key and Customer object as the value. Add multiple customers to the TreeMap and retrieve a customer object based on a specific id.

Exercise 6:

Create a class called Animal with attributes like name and age. Implement the Comparable interface in the Animal class to define the natural ordering based on the animal's age. Create a TreeMap with age as the key and Animal object as the value. Add multiple animals to the TreeMap and print the animals in ascending order of their ages.

Exercise 7:

Create a class called Order with attributes like orderNumber, customer, and totalAmount. Implement the Comparable interface in the Order class to define the natural ordering based on the order's orderNumber. Create a TreeMap with orderNumber as the key and Order object as the value. Add multiple orders to the TreeMap and retrieve an order object based on a specific orderNumber.

Exercise 8:

Create a class called Product with attributes like name, category, and price. Implement the Comparable interface in the Product class to define the natural ordering based on the product's price. Create a TreeMap with price as the key and a List of Product objects as the value. Add multiple products to the TreeMap and retrieve the list of products in ascending order of their prices.

Exercise 9:

Create a class called Employee with attributes like id, name, and department. Implement the Comparable interface in the Employee class to define the natural ordering based on the employee's department. Create a TreeMap with department as the key and a Set of Employee objects as the value. Add multiple employees to the TreeMap and retrieve the set of employees for a specific department.

Exercise 10:

Create a class called Movie with attributes like title, genre, and releaseYear. Implement the Comparable interface in the Movie class to define the natural ordering based on the movie's release year. Create a TreeMap with releaseYear as the key and a PriorityQueue of Movie objects as the value. Add multiple movies to the TreeMap and retrieve the priority queue of movies for a specific release year.

Part 7 - TreeSet

Exercise 1:

Write a program that prompts the user to enter a series of integers and stores them in a TreeSet. Print the elements of the TreeSet in ascending order.

Exercise 2:

Write a program that prompts the user to enter a sentence and stores the unique words in a TreeSet. Print the sorted set of unique words.

Exercise 3:

Write a program that creates a TreeSet of Strings and adds some names to it. Prompt the user to enter a name and check whether it exists in the TreeSet. Print an appropriate message indicating the result.

Exercise 4:

Write a program that creates a TreeSet of Integer objects and adds some numbers to it. Find and print the smallest and largest numbers in the TreeSet.

Exercise 5:

Write a program that creates a TreeSet of custom objects that implement the Comparable interface. Add several objects to the TreeSet and print them in sorted order based on the custom sorting criteria.

Exercise 6:

Write a program that creates a TreeSet of Strings and removes all the elements that start with a specific letter. Print the TreeSet after removing the elements.

Exercise 7:

Write a program that creates a TreeSet of Double objects and calculates the sum of all the elements in the TreeSet. Print the sum.

Exercise 8:

Write a program that prompts the user to enter a series of words and stores them in a TreeSet. Print the words in reverse order, starting from the last element.

Exercise 9:

Write a program that creates a TreeSet of Strings and checks whether it is empty or not. Print an appropriate message indicating the result.

Exercise 10:

Write a program that creates a TreeSet of Integer objects and removes all the even numbers from the TreeSet. Print the TreeSet after removing the even numbers.

Part 8 - TreeSet with OOP

Exercise 1:

Create a class called Employee with attributes like id, name, and salary. Implement the Comparable interface in the Employee class to define the natural ordering based on employee salary. Create a TreeSet of Employee objects and add multiple employees to it. Print the employees in ascending order based on their salaries using the TreeSet.

Exercise 2:

Create a class called Student with attributes like roll number, name, and grade. Implement the Comparable interface in the Student class to define the natural ordering based on student grades. Create a TreeSet of Student objects and add multiple students to it. Print the students in ascending order based on their grades using the TreeSet.

Exercise 3:

Create a class called Book with attributes like title, author, and publication year. Implement the Comparable interface in the Book class to define the natural ordering based on the book publication years. Create a TreeSet of Book objects and add multiple books to it. Print the books in ascending order based on their publication years using the TreeSet.

Exercise 4:

Create a class called Customer with attributes like id, name, and email. Override the equals() and hashCode() methods in the Customer class to compare customers based on their email addresses.

Create a TreeSet of Customer objects and add multiple customers to it. Verify if a specific customer exists in the TreeSet based on their email address.

Exercise 5:

Create a class called Product with attributes like name, price, and quantity. Implement the Comparable interface in the Product class to define the natural ordering based on the product prices. Create a TreeSet of Product objects and add multiple products to it. Find the cheapest and most expensive products in the TreeSet.

Exercise 6:

Create a class called Song with attributes like title, artist, and duration. Override the equals() and hashCode() methods in the Song class to compare songs based on their titles and artists. Create a TreeSet of Song objects and add multiple songs to it. Remove duplicate songs from the TreeSet and print the unique set of songs.

Exercise 7:

Create a class called Animal with attributes like name and age. Implement the Comparable interface in the Animal class to define the natural ordering based on animal ages. Create a TreeSet of Animal objects and add multiple animals to it. Print the animals in ascending order based on their ages using the TreeSet.

Exercise 8:

Create a class called Student with attributes like roll number, name, and marks. Implement the Comparable interface in the Student class to define the natural ordering based on student marks. Create a TreeSet of Student objects and add multiple students to it. Calculate the average marks of all the students in the TreeSet.

Exercise 9:

Create a class called Employee with attributes like id, name, and hire date. Implement the Comparable interface in the Employee class to define the natural ordering based on employee hire dates. Create a TreeSet of Employee objects and add multiple employees to it. Print the employees in ascending order based on their hire dates using the TreeSet.

Exercise 10:

Create a class called Product with attributes like name, price, and category. Implement the Comparable interface in the Product class to define the natural ordering based on the product names. Create a TreeSet of Product objects and add multiple products to it. Remove a specific product from the TreeSet based on its name and print the updated TreeSet.

Part 9 - Comparator in Java:

Exercise 1:

Write a program that creates a list of strings and sorts them in ascending order using a Comparator. Print the sorted list.

Exercise 2:

Write a program that creates a list of custom objects and sorts them in descending order based on a specific attribute using a Comparator. Print the sorted list.

Exercise 3:

Write a program that creates a list of integers and sorts them based on their absolute values using a Comparator. Print the sorted list.

Exercise 4:

Write a program that creates a list of strings and sorts them in alphabetical order ignoring case sensitivity using a Comparator. Print the sorted list.

Exercise 5:

Write a program that creates a list of custom objects and sorts them based on multiple attributes using a Comparator. Print the sorted list.

Exercise 6:

Write a program that creates a list of dates and sorts them in chronological order using a Comparator. Print the sorted list.

Exercise 7:

Write a program that creates a list of strings and sorts them based on their lengths using a Comparator. Print the sorted list.

Exercise 8:

Write a program that creates a list of custom objects and sorts them based on a specific attribute in reverse order using a Comparator. Print the sorted list.

Exercise 9:

Write a program that creates a list of integers and sorts them in ascending order, with even numbers appearing before odd numbers, using a Comparator. Print the sorted list.

Exercise 10:

Write a program that creates a list of strings and sorts them based on their number of vowels using a Comparator. Print the sorted list.