Object Oriented Programming (Java)

Generics with Bubble sort and Merge sort

In this program, the usage of generics allows for flexible sorting of different types of objects, such as employees in this case. By using generics, the code can handle sorting based on either salaries or departments without needing to modify the sorting algorithm itself. This flexibility is achieved by defining methods like bubbleSort and mergeSort with the generic type T which extends the Comparable interface. This means that any class that implements Comparable can be used with these sorting methods.  
  
  
  
Code:

import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

public class SortAssign<T extends Comparable<T>> {

private static class Employee implements Comparable<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;

}

public String getName() {

return name;

}

public String getDepartment() {

return department;

}

public double getSalary() {

return salary;

}

public String toString() {

return "Name: " + name + ", Department: " + department + ", Salary: " + salary;

}

@Override

public int compareTo(Employee other) {

return Double.compare(this.salary, other.salary);

}

}

public void bubbleSort(T[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j].compareTo(arr[j + 1]) > 0) {

T temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

public void mergeSort(T[] arr, Comparator<T> comparator) {

if (arr.length > 1) {

int mid = arr.length / 2;

T[] left = Arrays.copyOfRange(arr, 0, mid);

T[] right = Arrays.copyOfRange(arr, mid, arr.length);

mergeSort(left, comparator);

mergeSort(right, comparator);

merge(arr, left, right, comparator);

}

}

private void merge(T[] arr, T[] left, T[] right, Comparator<T> comparator) {

int i = 0, j = 0, k = 0;

while (i < left.length && j < right.length) {

if (comparator.compare(left[i], right[j]) <= 0) {

arr[k++] = left[i++];

} else {

arr[k++] = right[j++];

}

}

while (i < left.length) {

arr[k++] = left[i++];

}

while (j < right.length) {

arr[k++] = right[j++];

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of employees: ");

int numEmployees = scanner.nextInt();

scanner.nextLine();

Employee[] employees = new Employee[numEmployees];

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

System.out.println("Enter details for employee " + (i + 1) + ":");

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

String name = scanner.nextLine();

System.out.print("Department: ");

String department = scanner.nextLine();

System.out.print("Salary: ");

double salary = scanner.nextDouble();

scanner.nextLine();

employees[i] = new Employee(name, department, salary);

}

scanner.close();

SortAssign<Employee> empSystem = new SortAssign<>();

Employee[] employeesBySalaryBubbleSort = Arrays.copyOf(employees, employees.length);

empSystem.bubbleSort(employeesBySalaryBubbleSort);

System.out.println("\nEmployees sorted by salary (Bubble Sort):");

for (Employee emp : employeesBySalaryBubbleSort) {

System.out.println(emp);

}

Employee[] employeesByDepartmentMergeSort = Arrays.copyOf(employees, employees.length);

empSystem.mergeSort(employeesByDepartmentMergeSort, Comparator.comparing(Employee::getDepartment));

System.out.println("\nEmployees sorted by department (Merge Sort):");

for (Employee emp : employeesByDepartmentMergeSort) {

System.out.println(emp);

}

}

}