Q1: ToDoList.java

import java.util.InputMismatchException;  
import java.util.PriorityQueue;  
import java.util.Scanner;  
  
public class ToDoList  
{  
 // Instance variable(s) [1 point]  
 private final PriorityQueue<Task> taskQueue;  
  
 public ToDoList() // [2 point]  
 {  
 taskQueue = new PriorityQueue<>();  
 }  
  
 public static void main(String[] args) // [10 points]  
 {  
 // Display a user instruction menu for various options  
 System.*out*.println("Select an option: \n1. Add a task\n2. Show next task\n3. Quit");  
  
 /\* Continuously ask for the user's input,  
 such as adding new tasks, removing/printing the task,  
 (the following methods 'addTask' and 'printTask' are called accordingly)  
 until a 'quit' option is given.  
 \*/  
 Scanner in = new Scanner(System.*in*);  
 ToDoList toDoList = new ToDoList();  
  
 while (true) {  
 String option = in.nextLine();  
 switch (option) {  
 case "1":  
 toDoList.addTask();  
 break;  
 case "2":  
 toDoList.printTask();  
 break;  
 case "3":  
 System.*out*.println("Exiting program.");  
 return;  
 default:  
 System.*out*.println("Invalid option. Please try again.");  
 break;  
 }  
 }  
 }  
   
 // prompt user to enter tasks and add to the queue [10 points]  
 public void addTask() {  
 // prompt user to continue entering a task (priority and description)  
 // and add the task into the queue till 'Done' is entered by user  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.println("Enter a task in 'priority description' format (or 'Done' to finish):");  
  
 while (in.hasNextLine()) {  
 String input = in.nextLine();  
 if (input.equalsIgnoreCase("Done")) {  
 break;  
 }  
  
 String[] parts = input.split(" ", 2);  
 if (parts.length < 2) {  
 System.*out*.println("Invalid format. Please use 'priority description' format.");  
 continue;  
 }  
  
 try {  
 int priority = Integer.*parseInt*(parts[0]);  
 if (priority<1 || priority>9) throw new InputMismatchException("Priority out of range");  
 String description = parts[1];  
 Task newTask = new Task(priority, description);  
 taskQueue.offer(newTask);  
  
 //*TODO: implement the hind for task recorded to improve user interaction*// System.out.println("Task recorded enter another or 'Done' to finish):");  
  
 } catch (NumberFormatException | InputMismatchException e) {  
 System.*out*.println("Invalid priority. Please enter a valid number.");  
 }  
 }  
 }  
  
 // Get the next highest priority task and print it [5 point]  
 // If no more task left, print 'No more tasks'.  
 public void printTask() {  
 if (taskQueue.isEmpty()) {  
 System.*out*.println("No more tasks.");  
 } else {  
 Task task = taskQueue.poll();  
 System.*out*.println("Next task: " +task.getPriority()+" "+ task.getDescription());  
 }  
 }  
}

Scanner in = new Scanner(System.*in*);  
 System.*out*.println("Enter a task in 'priority description' format (or 'Done' to finish):");  
  
 while (in.hasNextLine()) {  
 String input = in.nextLine();  
 if (input.equalsIgnoreCase("Done")) {  
 break;  
 }  
  
 String[] parts = input.split(" ", 2);  
 if (parts.length < 2) {  
 System.*out*.println("Invalid format. Please use 'priority description' format.");  
 continue;  
 }  
  
 try {  
 int priority = Integer.*parseInt*(parts[0]);  
 if (priority<1 || priority>9) throw new InputMismatchException("Priority out of range");  
 String description = parts[1];  
 Task newTask = new Task(priority, description);  
 taskQueue.offer(newTask);  
  
 //*TODO: implement the hind for task recorded to improve user interaction*// System.out.println("Task recorded enter another or 'Done' to finish):");  
  
 } catch (NumberFormatException | InputMismatchException e) {  
 System.*out*.println("Invalid priority. Please enter a valid number.");  
 }  
 }  
 }  
  
 // Get the next highest priority task and print it [5 point]  
 // If no more task left, print 'No more tasks'.  
 public void printTask() {  
 if (taskQueue.isEmpty()) {  
 System.*out*.println("No more tasks.");  
 } else {  
 Task task = taskQueue.poll();  
 System.*out*.println("Next task: " +task.getPriority()+" "+ task.getDescription());  
 }  
 }  
}

Task.java

// Define a class whose instances may be added into a priority queue  
public class Task implements Comparable<Task>  
{  
 // add attributes [1 point]  
 private int priority;  
 private String description;  
  
 public Task(int priority, String description) {  
 this.priority = priority;  
 this.description = description;  
 }  
  
 // add getters and setters [5 point]  
 public int getPriority() {  
 return priority;  
 }  
  
 public void setPriority(int priority) {  
 this.priority = priority;  
 }  
  
 public String getDescription() {  
 return description;  
 }  
  
 public void setDescription(String description) {  
 this.description = description;  
 }  
  
 // override a method defined in the interface this class implements  
 // for being used for priority queue [5 point]  
 @Override  
 public int compareTo(Task other) {  
 // Lower priority value means higher priority task  
 return Integer.*compare*(this.priority, other.priority);  
 }  
}

Q2. Polynomial.java

import java.util.LinkedList;  
import java.util.ListIterator;  
  
public class Polynomial  
{  
 // attributes [1 point]  
 private LinkedList<Term> terms;  
   
 // Constructs an empty polynomial [1 point]  
 public Polynomial() {  
 terms = new LinkedList<>();  
 }  
  
 // Constructs a new polynomial with the given term [1 point]  
 public Polynomial(Term t) {  
 this();  
 terms.add(t);  
 }  
  
 // Adds the polynomial such that the terms are in sorted order [10 point]  
 public void add(Polynomial p) {  
 for (Term term : p.terms) {  
 boolean added = false;  
 ListIterator<Term> iterator = terms.listIterator();  
  
 while (iterator.hasNext()) {  
 Term current = iterator.next();  
  
 if (term.getPower() > current.getPower()) {  
 iterator.previous();  
 iterator.add(term);  
 added = true;  
 break;  
 } else if (term.getPower() == current.getPower()) {  
 current.addIfSamePower(term);  
 added = true;  
 break;  
 }  
 }  
  
 if (!added) {  
 terms.add(term);  
 }  
 }  
 }  
  
 // Multiplies the given polynomial with this one and returns the result [3 point]  
 public Polynomial multiply(Polynomial p){  
 Polynomial result = new Polynomial();  
  
 for (Term t1 : this.terms) {  
 for (Term t2 : p.terms) {  
 result.add(new Polynomial(t1.multiply(t2)));  
 }  
 }  
  
 return result;  
 }  
  
 // Prints the polynomial "nicely" so that it reads from highest term to lowest  
 // and doesn't have a leading "+" if the first term is positive. [4 point]  
 public void print(){  
 StringBuilder sb = new StringBuilder();  
  
 for (Term t : terms) {  
 if (!sb.isEmpty() && t.getCoefficient() >= 0) {  
 sb.append(" + ");  
 } else if (!sb.isEmpty() && t.getCoefficient() < 0) {  
 sb.append(" ");  
 }  
 sb.append(t.toString());  
 }  
  
 System.*out*.print(sb);  
 }  
  
 // Tests your class  
 public static void main(String[] args) {  
 Polynomial p = new Polynomial(new Term(-10, 0));  
 p.print();  
 System.*out*.println("\nExpected: - 10.0");  
 p.add(new Polynomial(new Term(-1, 1)));  
 p.print();  
 System.*out*.println("\nExpected: - 1.0x - 10.0");  
 p.add(new Polynomial(new Term(9, 7)));  
 p.print();  
 System.*out*.println("\nExpected: 9.0x^7 - 1.0x - 10.0");  
 p.add(new Polynomial(new Term(5, 10)));  
 p.print();  
 System.*out*.println("\nExpected: 5.0x^10 + 9.0x^7 - 1.0x - 10.0");  
  
 Polynomial q = p.multiply(p);  
 q.print();  
 System.*out*.println("\nExpected: 25.0x^20 + 90.0x^17 + 81.0x^14 - 10.0x^11 - 100.0x^10 - 18.0x^8 - 180.0x^7 + 1.0x^2 + 20.0x + 100.0");  
 }  
}

result.add(new Polynomial(t1.multiply(t2)));  
 }  
 }  
  
 return result;  
 }  
  
 // Prints the polynomial "nicely" so that it reads from highest term to lowest  
 // and doesn't have a leading "+" if the first term is positive. [4 point]  
 public void print(){  
 StringBuilder sb = new StringBuilder();  
  
 for (Term t : terms) {  
 if (!sb.isEmpty() && t.getCoefficient() >= 0) {  
 sb.append(" + ");  
 } else if (!sb.isEmpty() && t.getCoefficient() < 0) {  
 sb.append(" ");  
 }  
 sb.append(t.toString());  
 }  
  
 System.*out*.print(sb);  
 }  
  
 // Tests your class  
 public static void main(String[] args) {  
 Polynomial p = new Polynomial(new Term(-10, 0));  
 p.print();  
 System.*out*.println("\nExpected: - 10.0");  
 p.add(new Polynomial(new Term(-1, 1)));  
 p.print();  
 System.*out*.println("\nExpected: - 1.0x - 10.0");  
 p.add(new Polynomial(new Term(9, 7)));  
 p.print();  
 System.*out*.println("\nExpected: 9.0x^7 - 1.0x - 10.0");  
 p.add(new Polynomial(new Term(5, 10)));  
 p.print();  
 System.*out*.println("\nExpected: 5.0x^10 + 9.0x^7 - 1.0x - 10.0");  
  
 Polynomial q = p.multiply(p);  
 q.print();  
 System.*out*.println("\nExpected: 25.0x^20 + 90.0x^17 + 81.0x^14 - 10.0x^11 - 100.0x^10 - 18.0x^8 - 180.0x^7 + 1.0x^2 + 20.0x + 100.0");  
 }  
}

Term.java

*/\*\*  
 A class to represent an algebraic term.  
\*/*class Term  
{  
 private double coefficient;  
 private int power;  
  
 // constructor with parameters [1 point]  
 public Term(double coefficient, int power){  
 this.coefficient = coefficient;  
 this.power = power;  
 }  
  
 // getters and setters [2 point]  
 public double getCoefficient() {  
 return coefficient;  
 }  
  
 public int getPower() {  
 return power;  
 }  
  
 public void setCoefficient(double coefficient) {  
 this.coefficient = coefficient;  
 }  
  
 public void setPower(int power) {  
 this.power = power;  
 }  
  
 // Multiplies two coefficient together and returns the result [2 point]  
 public Term multiply(Term t)  
 {  
 return new Term(this.coefficient \* t.coefficient, this.power + t.power);  
 }  
  
 // Adds the term to this term if the powers are the same [2 point]  
 public void addIfSamePower(Term t)  
 {  
 if (this.power == t.power) {  
 this.coefficient += t.coefficient;  
 }  
 }  
  
 // Returns a string representation of the term [3 point]  
 // with a ^ representing the exponent  
 public String toString()  
 {  
 String coefficientToString = coefficient>=0? String.*valueOf*(coefficient) : "- " + Math.*abs*(coefficient);  
 if (power == 0) return coefficientToString;  
 if (power == 1) return coefficientToString + "x";  
 return coefficientToString + "x^" + power;  
 }  
}

if (power == 0) return coefficientToString;  
 if (power == 1) return coefficientToString + "x";  
 return coefficientToString + "x^" + power;  
 }  
}

Q3.

public class MyPriorityQueue<E>  
{  
 Node<E> front; // reference to the first node in the queue  
  
 public MyPriorityQueue() // [1 point]  
 {  
 front = null;  
 }  
  
 // add an item E into the queue (according to its priority)  
 public void push (Node<E> node) // [10 point]  
 {  
 if (front == null || node.priority > front.priority) {  
 node.next = front;  
 front = node;  
 } else {  
 Node<E> current = front;  
 while (current.next != null && current.next.priority >= node.priority) {  
 current = current.next;  
 }  
 node.next = current.next;  
 current.next = node;  
 }  
 }  
  
 // pop up the time (with the highest priority from the queue  
 // throw an exception if the queue is empty  
 public Node<E> pop() // [2 point]  
 {  
 if (front == null) throw new RuntimeException("Queue is empty");  
 Node<E> tempNode = front;  
 front = front.next;  
 return tempNode;  
 }  
  
 // display/print all the items in the queue  
 // print a message if the queue is empty  
 public void display() // [2 point]  
 {  
 if (front == null) {  
 System.*out*.println("Queue is empty");  
 return;  
 }  
 Node<E> temp = front;  
 while (temp != null) {  
 System.*out*.println(temp.item + " (Priority: " + temp.priority + ")");  
 temp = temp.next;  
 }  
 }  
  
 public static void main(String[] args) // [5 points]  
 {  
 // create an instance of MyPriorityQueue<E>  
 MyPriorityQueue<String> stringQueue = new MyPriorityQueue<>();  
  
 // add no less than 5 String (or int, float) to the queue  
 stringQueue.push(new Node<>("Apple", 5));  
 stringQueue.push(new Node<>("Banana", 2));  
 stringQueue.push(new Node<>("Cherry", 7));  
 stringQueue.push(new Node<>("Date", 4));  
 stringQueue.push(new Node<>("Elderberry", 6));  
  
 // display the contents of queue to test it  
 System.*out*.println("Queue after adding strings:");  
 stringQueue.display();  
  
 // pop up one element from the queue and display again  
 stringQueue.pop();  
 System.*out*.println("\nQueue after popping one string:");  
 stringQueue.display();  
  
 // Create an instance of MyPriorityQueue for MyItem objects  
 MyPriorityQueue<MyItem> itemQueue = new MyPriorityQueue<>();  
 // add no less than 5 instances of your own defined class  
 itemQueue.push(new Node<>(new MyItem("Item1", 100), 3));  
 itemQueue.push(new Node<>(new MyItem("Item2", 200), 1));  
 itemQueue.push(new Node<>(new MyItem("Item3", 300), 5));  
 itemQueue.push(new Node<>(new MyItem("Item4", 400), 4));  
 itemQueue.push(new Node<>(new MyItem("Item5", 500), 2));  
 // display the contents of queue to test it  
 System.*out*.println("\nQueue after adding MyItem objects:");  
 itemQueue.display();  
 // pop up one element from the queue and display again  
 itemQueue.pop();  
 System.*out*.println("\nQueue after popping one MyItem object:");  
 itemQueue.display();  
 }  
  
 // complete the node class [5 points]  
 static class Node<E>  
 {  
 // add attributes so that it's used for a linked list  
 // add a constructor (and other methods if needed)  
 E item;  
 int priority;  
 Node<E> next;  
  
 Node(E item, int priority) {  
 this.item = item;  
 this.priority = priority;  
 this.next = null;  
 }  
  
 }  
  
 // define your own class [3 points]  
 // whose instances will be added into the queue  
 static class MyItem  
 {  
 // add attributes or methods needed  
 String name;  
 int value; // Example attributes  
  
 MyItem(String name, int value) {  
 this.name = name;  
 this.value = value;  
 }  
  
 @Override  
 public String toString() {  
 return "MyItem{" + "name='" + name + '\'' + ", value=" + value + '}';  
 }  
 }  
}

{  
 // create an instance of MyPriorityQueue<E>  
 MyPriorityQueue<String> stringQueue = new MyPriorityQueue<>();  
  
 // add no less than 5 String (or int, float) to the queue  
 stringQueue.push(new Node<>("Apple", 5));  
 stringQueue.push(new Node<>("Banana", 2));  
 stringQueue.push(new Node<>("Cherry", 7));  
 stringQueue.push(new Node<>("Date", 4));  
 stringQueue.push(new Node<>("Elderberry", 6));  
  
 // display the contents of queue to test it  
 System.*out*.println("Queue after adding strings:");  
 stringQueue.display();  
  
 // pop up one element from the queue and display again  
 stringQueue.pop();  
 System.*out*.println("\nQueue after popping one string:");  
 stringQueue.display();  
  
 // Create an instance of MyPriorityQueue for MyItem objects  
 MyPriorityQueue<MyItem> itemQueue = new MyPriorityQueue<>();  
 // add no less than 5 instances of your own defined class  
 itemQueue.push(new Node<>(new MyItem("Item1", 100), 3));  
 itemQueue.push(new Node<>(new MyItem("Item2", 200), 1));  
 itemQueue.push(new Node<>(new MyItem("Item3", 300), 5));  
 itemQueue.push(new Node<>(new MyItem("Item4", 400), 4));  
 itemQueue.push(new Node<>(new MyItem("Item5", 500), 2));  
 // display the contents of queue to test it  
 System.*out*.println("\nQueue after adding MyItem objects:");  
 itemQueue.display();  
 // pop up one element from the queue and display again  
 itemQueue.pop();  
 System.*out*.println("\nQueue after popping one MyItem object:");  
 itemQueue.display();  
 }  
  
 // complete the node class [5 points]  
 static class Node<E>  
 {  
 // add attributes so that it's used for a linked list  
 // add a constructor (and other methods if needed)  
 E item;  
 int priority;  
 Node<E> next;  
  
 Node(E item, int priority) {  
 this.item = item;  
 this.priority = priority;  
 this.next = null;  
 }  
  
 }  
  
 // define your own class [3 points]  
 // whose instances will be added into the queue  
 static class MyItem  
 {  
 // add attributes or methods needed  
 String name;  
 int value; // Example attributes  
  
 MyItem(String name, int value) {  
 this.name = name;  
 this.value = value;  
 }  
  
 @Override  
 public String toString() {  
 return "MyItem{" + "name='" + name + '\'' + ", value=" + value + '}';  
 }  
 }  
}

static class MyItem  
 {  
 // add attributes or methods needed  
 String name;  
 int value; // Example attributes  
  
 MyItem(String name, int value) {  
 this.name = name;  
 this.value = value;  
 }  
  
 @Override  
 public String toString() {  
 return "MyItem{" + "name='" + name + '\'' + ", value=" + value + '}';  
 }  
 }  
}