```java
import java.util.Arrays;
import java.util.Scanner;

// Define a SortingAlgorithm interface
interface SortingAlgorithm {
    int[] sort(int[] data);
}

// Implement different sorting algorithms using the SortingAlgorithm interface
class BubbleSort implements SortingAlgorithm {
    @Override
    public int[] sort(int[] data) {
        int n = data.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (data[j] > data[j + 1]) {
                    int temp = data[j];
                    data[j] = data[j + 1];
                    data[j + 1] = temp;
                }
            }
        }
        return data;
    }
}


class QuickSort implements SortingAlgorithm {
    @Override
    public int[] sort(int[] data) {
        quickSort(data, 0, data.length - 1);
        return data;
    }

    private void quickSort(int[] data, int low, int high) {
        if (low < high) {
            int pivotIndex = partition(data, low, high);
            quickSort(data, low, pivotIndex - 1);
            quickSort(data, pivotIndex + 1, high);
        }
    }

    private int partition(int[] data, int low, int high) {
        int pivot = data[high];
        int i = low - 1;

        for (int j = low; j < high; j++) {
            if (data[j] < pivot) {
                i++;
                int temp = data[i];
                data[i] = data[j];
                data[j] = temp;
            }
        }
        int temp = data[i + 1];
        data[i + 1] = data[high];
        data[high] = temp;

        return i + 1;
    }
}


class MergeSort implements SortingAlgorithm {
    @Override
    public int[] sort(int[] data) {
        // Implement Merge Sort
        mergeSort(data, 0, data.length - 1);
        return data;
    }

    private void mergeSort(int[] data, int low, int high) {
        if (low < high) {
            int mid = (low + high) / 2;
            mergeSort(data, low, mid);
            mergeSort(data, mid + 1, high);
            merge(data, low, mid, high);
        }
    }

    private void merge(int[] data, int low, int mid, int high) {
        int n1 = mid - low + 1;
        int n2 = high - mid;

        int[] left = new int[n1];
        int[] right = new int[n2];

        for (int i = 0; i < n1; i++) {
            left[i] = data[low + i];
        }
        for (int j = 0; j < n2; j++) {
            right[j] = data[mid + 1 + j];
        }

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

        while (i < n1 && j < n2) {
            if (left[i] <= right[j]) {
                data[k] = left[i];
                i++;
            } else {
                data[k] = right[j];
                j++;
            }
            k++;
        }

        while (i < n1) {
            data[k] = left[i];
            i++;
            k++;
        }

        while (j < n2) {
            data[k] = right[j];
            j++;
            k++;
        }
    }
}

// Create a context class that uses the selected sorting algorithm
class SortContext {
    private SortingAlgorithm sortingAlgorithm;

    public void setSortingAlgorithm(SortingAlgorithm sortingAlgorithm) {
        this.sortingAlgorithm = sortingAlgorithm;
    }

    public int[] performSort(int[] data) {
        if (sortingAlgorithm != null) {
            return sortingAlgorithm.sort(data);
        }
        return data; // Return the original data if no sorting algorithm is set.
    }
}

public class Client {
    public static void main(String[] args) {
        int[] data = {5, 2, 9, 1, 5, 6};
        Scanner scanner = new Scanner(System.in);

        // Create a SortContext
        SortContext sortContext = new SortContext();

        // Allow the client to select the sorting algorithm
        System.out.println("Select a sorting algorithm: (1) Bubble Sort, (2) Quick Sort, (3)
Merge Sort");
        int choice = scanner.nextInt();
        SortingAlgorithm selectedAlgorithm;

        switch (choice) {
            case 1:
                selectedAlgorithm = new BubbleSort();
                break;
            case 2:
                selectedAlgorithm = new QuickSort();
                break;
            case 3:
                selectedAlgorithm = new MergeSort();
                break;
            default:
                selectedAlgorithm = null;
                break;
        }

        sortContext.setSortingAlgorithm(selectedAlgorithm);

        // Perform the sort
        int[] sortedData = sortContext.performSort(data);

        // Print the sorted data
        System.out.println("Sorted Data: " + Arrays.toString(sortedData));
    }
}
```