Write a Java program to create an interface Sortable with a method sort (int[] array) that sorts an array of integers in descending order. Create two classes QuickSort and MergeSort that implement the Sortable interface and provide their own implementations of the sort() method.

Sample Solution:

Java Code:

```java
// Sortable.java

// Declare the Sortable interface
interface Sortable {
    // Declare the abstract method "sort" that classes implementing this ir
    void sort(int[] array);
}
```

```java
// QuickSort.java

// Declare the QuickSort class, which implements the Sortable interface
class QuickSort implements Sortable {
    // Implement the "sort" method required by the Sortable interface
    public void sort(int[] array) {
        quickSort(array, 0, array.length - 1);
    }

    // Helper method for the QuickSort algorithm
    private void quickSort(int[] array, int low, int high) {
        if (low < high) {
            // Find the partition index using the "partition" method
            int partitionIndex = partition(array, low, high);

            // Recursively sort the sub-arrays before and after the partiti
            quickSort(array, low, partitionIndex - 1);
            quickSort(array, partitionIndex + 1, high);
        }
    }

    // Helper method to partition the array
    private int partition(int[] array, int low, int high) {
        // Choose the pivot element, which is the element at the "high" inc
```

```
            int pivot = array[high];
            int i = low - 1;

            // Iterate through the elements in the array
            for (int j = low; j < high; j++) {
                // If the current element is greater than or equal to the pivot
                if (array[j] >= pivot) {
                    i++;
                    int temp = array[i];
                    array[i] = array[j];
                    array[j] = temp;
                }
            }

            // Swap the pivot element with the element at the (i + 1) index
            int temp = array[i + 1];
            array[i + 1] = array[high];
            array[high] = temp;

            // Return the partition index
            return i + 1;
        }
    }
```

```
    // MergeSort.java

    // Declare the MergeSort class, which implements the Sortable interface
    class MergeSort implements Sortable {
        // Implement the "sort" method required by the Sortable interface
        public void sort(int[] array) {
            mergeSort(array, 0, array.length - 1);
        }

        // Helper method for the MergeSort algorithm
        private void mergeSort(int[] array, int low, int high) {
            if (low < high) {
                // Calculate the middle index
                int mid = (low + high) / 2;

                // Recursively sort the left and right sub-arrays
            mergeSort(array, low, mid);
            mergeSort(array, mid + 1, high);

            // Merge the sorted sub-arrays
            merge(array, low, mid, high);
        }
    }

    // Helper method to merge two sub-arrays
    private void merge(int[] array, int low, int mid, int high) {
        // Calculate the sizes of the left and right sub-arrays
        int leftSize = mid - low + 1;
        int rightSize = high - mid;

        // Create temporary arrays to hold the left and right sub-arrays
        int[] leftArray = new int[leftSize];
        int[] rightArray = new int[rightSize];

        // Copy elements from the original array to the left and right sub
        for (int i = 0; i < leftSize; i++) {
            leftArray[i] = array[low + i];
        }

        for (int i = 0; i < rightSize; i++) {
            rightArray[i] = array[mid + 1 + i];
        }

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

        // Merge the two sub-arrays back into the original array
        while (i < leftSize && j < rightSize) {
            if (leftArray[i] >= rightArray[j]) {
                array[k] = leftArray[i];
                i++;
            } else {
                array[k] = rightArray[j];
                j++;
            }
            k++;
        }
```

```java
            // Copy any remaining elements from the left and right sub-arrays
            while (i < leftSize) {
                array[k] = leftArray[i];
                i++;
                k++;
            }

            while (j < rightSize) {
                array[k] = rightArray[j];
                j++;
                k++;
            }
        }
    }
```

```java
    // Main.java

    // Declare the Main class
    public class Main {
      // Main method, the entry point of the program
      public static void main(String[] args) {
        // Create an array of integers
        int[] arr = {
          9,
          5,
          2,
          8,
          0,
          3,
          1,
          6
        };

        // Create an instance of QuickSort and use it to sort the array in desc
        Sortable quickSort = new QuickSort();
        quickSort.sort(arr);

        // Print the sorted array using Quick Sort
        System.out.print("Quick Sort (Descending Order): ");
      printArray(arr);

      // Create another array of integers
      int[] arr2 = {
        9,
        5,
        2,
        8,
        0,
        3,
        1,
        6
      };

      // Create an instance of MergeSort and use it to sort the second array
      Sortable mergeSort = new MergeSort();
      mergeSort.sort(arr2);

      // Print the sorted array using Merge Sort
      System.out.print("Merge Sort (Descending Order): ");
      printArray(arr2);
    }

    // Helper method to print an array of integers
    private static void printArray(int[] arr) {
      for (int num: arr) {
        System.out.print(num + " ");
      }
      System.out.println();
    }
  }
```

Sample Output:

```
Quick Sort (Descending Order): 9 8 6 5 3 2 1 0
Merge Sort (Descending Order): 9 8 6 5 3 2 1 0
```

Explanation: