```
class Quicksort {


    /* quicksort soll das Array a zwischen left (inklusive) und
       right (exklusive) sortieren. */
    static void quicksort(int[] a, int left, int right) {

        /* Das Sortieren ist nur dann sinnvoll, wenn zwischen der linken und
           der rechten Grenze mehr als ein Eintrag vorhanden ist. */
        if (left < right-1) {

            /* partition sortiert das Array in Abhängigkeit von dem pivot-
               Element und liefert dessen Position im Array zurück. */
            int pivotPos = partition(a, left, right);

            /* Der vordere Teil (mit allen Elementen, die kleiner als das
              pivot-Element sind) ... */
            quicksort(a, left, pivotPos);

            /* ... und der hintere Teil (mit allen Elementen, die größer
              sind als das pivot-Element) müssen noch sortiert werden. */
            quicksort(a, pivotPos+1, right);
        }
    }


    /* partition teilt das Array a in einen Teil, der alle Elemente enthält,
       die kleiner (oder gleich) dem pivot-Element sind, und in einen Teil,
       der alle Elemente enthält, die größer sind als das pivot-Element;
       zwischen diesen beiden Teilen steht das pivot-Element. */
    static int partition(int[] a, int left, int right) {

      /* Als pivot-Element wird der letzte Wert im zu sortierenden
         Array-Teil ausgewählt. */
        int pivot = a[right-1];

        /* Zur Einordnen der Elemente werden zwei Pointer benötigt:
          i zeigt auf die Array-Position, dessen Eintrag gegen den
            aktuell zu überprüfenden Eintrag getausch werden kann.
          j zeigt auf die (aktuelle) Array-Position, dessen Eintrag gegen
            das pivot-Element überprüft wird.
          Das Array a wird mittels j durchiteriert und jeder Eintrag mit
          dem pivot-Element verglichen. */
        int i = left;
        for (int j = left; j < right-1; j++) {

            /* Wenn der aktuelle Eintrag kleiner oder gleich dem pivot-
               Element ist, wird dieser gegen den Eintrag von Position i
               vertauscht, d.h. i muß einen Array-Eintrag weiter gesetzt
               werden. */
            if (a[j] <= pivot) {
                swap(a, i, j);
                i++;
            }
        }

        /* Am Ende zeigt i auf den ersten Wert, der größer ist als das
          pivot-Element, also wird dieser Wert gegen das pivot-Element
          getauscht und die nun aktuelle Position des pivot-Elements
          zurückgegeben */
        swap(a, i, right-1);
        return i;
    }
```

```
    /* swap vertauscht die beiden Werte im Array a, dessen Position durch
       i und j angegeben wird. */
    static void swap(int[] a, int i, int j) {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }



    // toString-Methode zur Ausgabe der Array-Einträge
    static void toString(int[] a) {
        for(int i = 0; i < a.length; i++)
            System.out.print(a[i] + "  ");
    }



    // Testlauf
    public static void main(String[] args) {

        System.out.println("\nQUICKSORT\n\n");

        int size = 0;
        int[] array = new int[size];

        for(int i = 0; i < 5; i++) {

            size = (int) (Math.random() * 20);
            array = new int[size];
            for(int j = 0; j < array.length; j++)
                array[j] = (int) (Math.random() * 100);

            System.out.println("\n\nunsortiertes Array: ");
            toString(array);

            quicksort(array, 0, array.length);

            System.out.println("\nsortiertes Array:");
            toString(array);
        }
    }
}
```