# QuickUnionUF.java

Below is the syntax highlighted version of QuickUnionUF.java from §1.5 Case Study: Union-Find.

```
/******************************************************************************
 *  Compilation:  javac QuickUnionUF.java
 *  Execution:  java QuickUnionUF < input.txt
 *  Dependencies: StdIn.java StdOut.java
 *  Data files:   http://algs4.cs.princeton.edu/15uf/tinyUF.txt
 *                http://algs4.cs.princeton.edu/15uf/mediumUF.txt
 *                http://algs4.cs.princeton.edu/15uf/largeUF.txt
 *
 *  Quick-union algorithm.
 *
 ******************************************************************************/

/**
 *  The {@code QuickUnionUF} class represents a <em>union-find data type</em>
 *  (also known as the <em>disjoint-sets data type</em>).
 *  It supports the <em>union</em> and <em>find</em> operations,
 *  along with a <em>connected</em> operation for determining whether
 *  two sites are in the same component and a <em>count</em> operation that
 *  returns the total number of components.
 *  <p>
 *  The union-find data type models connectivity among a set of <em>n</em>
 *  sites, named 0 through <em>n</em>-1.
 *  The <em>is-connected-to</em> relation must be an
 *  <em>equivalence relation</em>:
 *  <ul>
 *  <li> <em>Reflexive</em>: <em>p</em> is connected to <em>p</em>.
 *  <li> <em>Symmetric</em>: If <em>p</em> is connected to <em>q</em>,
 *       then <em>q</em> is connected to <em>p</em>.
 *  <li> <em>Transitive</em>: If <em>p</em> is connected to <em>q</em>
 *       and <em>q</em> is connected to <em>r</em>, then
 *       <em>p</em> is connected to <em>r</em>.
 *  </ul>
 *  <p>
 *  An equivalence relation partitions the sites into
 *  <em>equivalence classes</em> (or <em>components</em>). In this case,
 *  two sites are in the same component if and only if they are connected.
 *  Both sites and components are identified with integers between 0 and
 *  <em>n</em>-1.
 *  Initially, there are <em>n</em> components, with each site in its
 *  own component.  The <em>component identifier</em> of a component
 *  (also known as the <em>root</em>, <em>canonical element</em>, <em>leader</em>,
 *  or <em>set representative</em>) is one of the sites in the component:
 *  two sites have the same component identifier if and only if they are
 *  in the same component.
 *  <ul>
 *  <li><em>union</em>(<em>p</em>, <em>q</em>) adds a
 *      connection between the two sites <em>p</em> and <em>q</em>.
 *      If <em>p</em> and <em>q</em> are in different components,
 *      then it replaces
 *      these two components with a new component that is the union of
 *      the two.
 *  <li><em>find</em>(<em>p</em>) returns the component
 *      identifier of the component containing <em>p</em>.
 *  <li><em>connected</em>(<em>p</em>, <em>q</em>)
 *      returns true if both <em>p</em> and <em>q</em>
```

```
 *      are in the same component, and false otherwise.
 *  <li><em>count</em>() returns the number of components.
 *  </ul>
 *  <p>
 *  The component identifier of a component can change
 *  only when the component itself changes during a call to
 *  <em>union</em>—it cannot change during a call
 *  to <em>find</em>, <em>connected</em>, or <em>count</em>.
 *  <p>
 *  This implementation uses quick union.
 *  Initializing a data structure with <em>n</em> sites takes linear time.
 *  Afterwards, the <em>union</em>, <em>find</em>, and <em>connected</em>
 *  operations  take linear time (in the worst case) and the
 *  <em>count</em> operation takes constant time.
 *  For alternate implementations of the same API, see
 *  {@link UF}, {@link QuickFindUF}, and {@link WeightedQuickUnionUF}.
 *
 *  <p>
 *  For additional documentation, see <a href="http://algs4.cs.princeton.edu/15uf">Section 1.5</a> of
 *  <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin Wayne.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */
public class QuickUnionUF {
    private int[] parent;  // parent[i] = parent of i
    private int count;     // number of components

    /**
     * Initializes an empty union-find data structure with {@code n} sites
     * {@code 0} through {@code n-1}. Each site is initially in its own
     * component.
     *
     * @param  n the number of sites
     * @throws IllegalArgumentException if {@code n < 0}
     */
    public QuickUnionUF(int n) {
        parent = new int[n];
        count = n;
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }

    /**
     * Returns the number of components.
     *
     * @return the number of components (between {@code 1} and {@code n})
     */
    public int count() {
        return count;
    }

    /**
     * Returns the component identifier for the component containing site {@code p}.
     *
     * @param  p the integer representing one object
     * @return the component identifier for the component containing site {@code p}
     * @throws IndexOutOfBoundsException unless {@code 0 <= p < n}
     */
    public int find(int p) {
        validate(p);
        while (p != parent[p])
            p = parent[p];
        return p;
```

```
    }

    // validate that p is a valid index
    private void validate(int p) {
        int n = parent.length;
        if (p < 0 || p >= n) {
            throw new IndexOutOfBoundsException("index " + p + " is not between 0 and " + (n-1));
        }
    }

    /**
     * Returns true if the the two sites are in the same component.
     *
     * @param  p the integer representing one site
     * @param  q the integer representing the other site
     * @return {@code true} if the two sites {@code p} and {@code q} are in the same component;
     *         {@code false} otherwise
     * @throws IndexOutOfBoundsException unless
     *         both {@code 0 <= p < n} and {@code 0 <= q < n}
     */
    public boolean connected(int p, int q) {
        return find(p) == find(q);
    }


    /**
     * Merges the component containing site {@code p} with the
     * the component containing site {@code q}.
     *
     * @param  p the integer representing one site
     * @param  q the integer representing the other site
     * @throws IndexOutOfBoundsException unless
     *         both {@code 0 <= p < n} and {@code 0 <= q < n}
     */
    public void union(int p, int q) {
        int rootP = find(p);
        int rootQ = find(q);
        if (rootP == rootQ) return;
        parent[rootP] = rootQ;
        count--;
    }

    /**
     * Reads in a sequence of pairs of integers (between 0 and n-1) from standard input,
     * where each integer represents some object;
     * if the sites are in different components, merge the two components
     * and print the pair to standard output.
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        int n = StdIn.readInt();
        QuickUnionUF uf = new QuickUnionUF(n);
        while (!StdIn.isEmpty()) {
            int p = StdIn.readInt();
            int q = StdIn.readInt();
            if (uf.connected(p, q)) continue;
            uf.union(p, q);
            StdOut.println(p + " " + q);
        }
        StdOut.println(uf.count() + " components");
    }


}
```

*Last updated: Sat Dec 31 07:32:27 EST 2016.*