```java

import components.stack.Stack;

/**
 * {@code Queue} represented as a {@code Sequence} of entries, with
 * implementations of primary methods.
 *
 * @param <T>
 *            type of {@code Queue} entries
 * @correspondence this = $this.entries
 */
public class HelloWorld {

    /**
     * Shifts entries between {@code leftStack} and {@code rightStack}, keeping
     * reverse of the former concatenated with the latter fixed, and resulting
     * in length of the former equal to {@code newLeftLength}.
     *
     * @param <T>
     *            type of {@code Stack} entries
     * @param leftStack
     *            the left {@code Stack}
     * @param rightStack
     *            the right {@code Stack}
     * @param newLeftLength
     *            desired new length of {@code leftStack}
     * @updates leftStack, rightStack
     * @requires <pre>
     * 0 <= newLeftLength  and
     * newLeftLength <= |leftStack| + |rightStack|
     * </pre>
     * @ensures <pre>
     * rev(leftStack) * rightStack = rev(#leftStack) * #rightStack and
     * |leftStack| = newLeftLength}
     * </pre>
     */
    private static <T> void setLengthOfLeftStack(Stack<T> leftStack,
            Stack<T> rightStack, int newLeftLength) {
        int numToTransfer = Math.abs(newLeftLength - leftStack.length());
        if (newLeftLength < leftStack.length()) {
            for (int i = 0; i < numToTransfer; i++) {
                T x = rightStack.pop();
                leftStack.push(x);
            }
        } else if (newLeftLength > leftStack.length()) {
            for (int i = 0; i < numToTransfer; i++) {
                T x = leftStack.pop();
                rightStack.push(x);
            }
        }

    }
}
```