```java
/**
 * Utility class with implementations of methods aFactor and aNonTrivialFactor
 * to be used in exploring JUnit features.
 *
 * @author Paolo Bucci
 *
 */
public final class FactoringUtility {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private FactoringUtility() {
    }

    /**
     * Reports some factor of a number.
     *
     * @param n
     *            the given number
     * @return a factor of the given number
     * @requires n > 0
     * @ensures aFactor > 0 and n mod aFactor = 0
     */
    public static int aFactor(int n) {
        assert n > 0 : "Violation of: n > 0";
        return 1;
    }

    /**
     * Reports some non-trivial factor of a composite number.
     *
     * @param n
     *            the given number
     * @return a non-trivial factor of the given number
     * @requires n > 2 and [n is not a prime number]
     * @ensures 1 < aNonTrivialFactorV1 < n and n mod aNonTrivialFactorV1 = 0
     */
    public static int aNonTrivialFactorV1(int n) {
        assert n > 2 : "Violation of: n > 2";
        int factor = 3;
        boolean found = false;
        while (!found) {
            if (n % factor == 0) {
                found = true;
            } else {
                factor = factor + 1;
            }
        }
        return factor;
    }

    /**
     * Reports some non-trivial factor of a composite number.
     *
     * @param n
     *            the given number
     * @return a non-trivial factor of the given number
     * @requires n > 2 and [n is not a prime number]
     * @ensures 1 < aNonTrivialFactorV2 < n and n mod aNonTrivialFactorV2 = 0
     */
    public static int aNonTrivialFactorV2(int n) {
        assert n > 2 : "Violation of: n > 2";
        int factor = 2;
        boolean found = false;
        while (!found) {
            if (n % factor == 0) {
                found = true;
            } else {
                factor = factor + 1;
            }
        }
        return factor;
    }

    /**
     * Reports some non-trivial factor of a composite number.
     *
     * @param n
     *            the given number
     * @return a non-trivial factor of the given number
     * @requires n > 2 and [n is not a prime number]
     * @ensures 1 < aNonTrivialFactorV3 < n and n mod aNonTrivialFactorV3 = 0
     */
    public static int aNonTrivialFactorV3(int n) {
        assert n > 2 : "Violation of: n > 2";
        int factor = 4;
        boolean found = false;
        while (!found) {
            if (n % factor == 0) {
                found = true;
            } else {
                factor = factor + 1;
            }
        }
        return factor;
    }

}
```