CSE 331 Method Specifications Reference

A *method specification* is a precise English description of the expected behavior of a method, which consists of:

- a *precondition*, which is an assertion that must hold each time the method is called
- a *postcondition*, which is an assertion that must hold each time the method returns

It is the caller's job to establish the precondition, and the method implementor's job to establish the postcondition under the assumption that the precondition is true.

A method is *correct* if it satisfies its specification. This means that:

> For any input that satisfies the precondition, running the method will lead to a behavior that satisfies the preconditions

We write method specifications using Javadoc-like syntax, which is based on various `@tags`.

- `@param`: Describes each parameter. Short preconditions on the parameter may be included here as well.
- `@requires`: (331 specific) Describes longer preconditions or preconditions that relate multiple parameters to each other.
- `@return`: Describes the return value. In 331, we use it to write the part of the postcondition that has to do with the return value.
- `@throws`: Describes what exceptions are guaranteed to be thrown and under what conditions. (This is also part of the postcondition.)
- `@modifies`: (331 specific) Describes which parameters *might* be mutated (changed) by the method. If a parameter is not mentioned in the `@modifies` clause, then it is guaranteed *not* to be mutated.
- `@effects`: (331 specific) Describes *how* the parameters listed in the `@modifies` tag are changed by the method. (This is also part of the postcondition.)

Example of `@param`/`@requires`/`@return`:

```
/**
 * Returns the index of the first occurrence of a value in a range of an array.
 *
 * @param a the array to search; must be non-null
 * @param lo the (inclusive) low end of the range to search
 * @param hi the (exclusive) high end of the range to search
 * @param x the value to search for
 * @requires 0 <= lo <= hi <= a.length
 * @return the smallest index i with lo <= i < hi and a[i] == x, or -1 if no
 *     such index exists
 */
public static int indexOf(int[] a, int lo, int hi, int x)
```

Example of `@modifies`/`@effects`/`@throws`:
uses `a[i..j]` to mean the elements of `a` at indices from `i` (inclusive) up to `j` (exclusive)

```
/**
 * Inserts a[n-1] into the correct position in a[0..n] so that it is sorted.
 *
 * @param a the non-null array
 * @param n the length of the prefix to sort
 * @requires 1 <= n <= a.length and a[0..n-1] is sorted
 * @modifies a
 * @effects a[0..n] is sorted and a[n..] is unchanged
 */
public static void insert(int[] a, int n)
```