CMPT 280

Self-Guided Tutorial: Javadoc Review and Usage Expectations.

Mark G. Eramian

University of Saskatchewan

Review: Javadoc

- Javadoc is a tool to automatically produce HTML documentation from code. The Java API (https://docs.oracle.com/en/java/javase/19/) is documented with javadoc.
- Comments for classes, fields, constructors are included in the doucmentation if they use this form:

```
/**
 * This is a comment.
 */
```

- Javadoc tags can be used in these comments:

```
/**
 * This is a comment.
 * @param   varname   description of parameter ``varname'' (Always include)
 * @return  what is returned from this method              (Always include)
 *
 * @author   (optional)
 * @version  (optional)
 */
```

Custom Javadoc Tags for This Course

- We introduce three custom javadoc tags for specifying preconditions, postconditions, and time complexity of methods:

```
/**
 * This is a comment.
 * @param   var  description of parameter/variable       (Always include)
 * @return  what is returned from this method         (Always include)
 *
 * @author   (optional)
 * @version  (optional)
 *
 * @precond  the precondition for this constructor/method      (custom)
 * @postcond the postcondition for this constructor/method     (custom)
 * @timing   the time requirements for this constructor/method (custom)
 */
```

Using Custom Javadoc Tags with IntelliJ

- Custom Javadoc tags can be defined in IntelliJ.
- If IntelliJ complains about a one of our three custom javadoc tags not being defined simply:
    1. Position the cursor on the tag and press "Alt-Enter".
    2. Choose the pop-up option to add it as a custom tag.

Using Custom Javadoc Tags

- Now your javadoc pages will include information from `@precond`, `@postcond`, and `@timing` tags.
- The custom tag definitions only need to be entered once per Java Project. They will be remembered for future (re)-generation of the javadoc documentation.

Javadoc Expectations for this Course

For every class or interface you write, provide a description of its purpose in a javadoc comment block. Include @param tags for generic type parameters.

```java
/**
 * This arrayed list is implemented as a circular list to allow for
 * constant-time insertions and deletions at the beginning and the end.
 *
 * @param <I> - type of elements stored in the list.
 */
public class ArrayedList280<I extends Comparable<? super I>>
     implements SimpleList280<I> {
...
}
```

Javadoc Expectations for this Course

For every instance variable, provide a short description in a javadoc comment block.

```java
/**
 * This arrayed list is implemented as a circular list to allow for
 * constant-time insertions and deletions at the beginning and the end.

 * @param <I>
 */
public class ArrayedList280<I extends Comparable<? super I>>
     implements SimpleList280<I> {

     /**
      * Array where the elements are stored.
      */
     protected I[] listElements;

     /**
      * Indices of the beginning and end of the list.
      * List is empty when head = tail.  List is full when
      * ((this.tail - 1) mod capacity) == this.head
      *
      */
     protected int head, tail;
     ...
}
```

Javadoc Expectations for this Course

For every method, in a javadoc comment give a brief description of the method's semantics (what it does). In addition:

- If the method has any parameters, include a `@param` tag for each parameter.
- If the method returns a value, describe the returned value with a `@return` tag.
- If the method throws any exceptions, include a `@throws` tag for each exception.

Javadoc Expectations for this Course

- If the method has any important preconditions or postconditions that are not obvious from the brief method description, describe them using `@precond` or `@postcond` tags.
- `@timing` tags are not normally expected, unless they are required explicitly by an assignment's description.

```java
/**
 * Obtain the last node in the list.
 * @precond !isEmpty()
 * @return the last node in  he list.
 * @throws ContainerEmpty280Exception
 */
public LinkedNode280<I> lastNode() throws ContainerEmpty280Exception {
    if( this.isEmpty() ) throw new ContainerEmpty280Exception(
        "Tried to get last node of an empty list.");
    return tail;
}
```

Javadoc Expectations for this Course

One last example, with a parameter.

```
    /**   Delete the item x.
     * @precond has(x)
     * @param x item to be deleted from the dictionary
     * @throws ItemNotFound280Exception when the dictionary does not contain x.
     */
    public void delete(I x) throws ItemNotFound280Exception;
```

General Commenting Expectations for this Course

For methods that are more than a few lines long, inline (non-javadoc) comments describing the function of small groups of statements are expected.

```
    public boolean has(I y) {
        // save the cursor's current position
        CursorPosition280 savePos = this.currentPosition();

        // Search for the element y (changes the cursor position)
        this.search(y);
        boolean result = itemExists();

        // Restore the original cursor position.
        this.goPosition(savePos);

        return result;
    }
```