CSE 331, Autumn 2011
Section 2 cheat sheet

JUnit, Javadoc, Eclipse

JUnit 4

Method annotations:

tag	**description**

`@Test`
`@Test (timeout =` **time**`)`
`@Test (expected =` **exception**`.class)`
	Turns a public method into a JUnit test case. Adding a timeout will cause the test case to fail after **time** milliseconds. Adding an expected exception will cause the test case to fail if **exception** is not thrown.

`@Before`	Method to run before every test case
`@After`	Method to run after every test case
`@BeforeClass`	Method to run once, before any test cases have run
`@AfterClass`	Method to run once, after all test cases have run

Assertion methods:

method	**description**
`assertTrue`(**test**)	fails if the Boolean test is `false`
`assertFalse`(**test**)	fails if the Boolean test is `true`
`assertEquals`(**expected, actual**)	fails if the values are not equal
`assertSame`(**expected, actual**)	fails if the values are not the same (by ==)
`assertNotSame`(**expected, actual**)	fails if the values are the same (by ==)
`assertNull`(**value**)	fails if the given value is not `null`
`assertNotNull`(**value**)	fails if the given value is `null`
`fail`()	causes the current test to immediately fail

Each method can also be passed a string to display if it fails, e.g.
`assertEquals`(**“message”, expected, actual**)

Unit testing tips:

- The entire goal is **FAILURE ATOMICITY**- the ability to know exactly what failed when a test case did not pass
- Tests should be self-contained and not care about each other
- you cannot test everything! Instead think about:
  - boundary cases,
  - empty cases,
  - behavior in combination (but not to excess)
- Each test case should test ONE THING
  - 10 small tests are better than 1 test 10x as large
  - Rule of thumb: 1 assert statement per test case
  - Try to avoud complicated logic
- Torture tests are ok, but only *in addition* to simple tests

JUnit best practices:

- Use descriptive test names
- Add a default timeout to every test
- Use private methods to get rid of redundant test code
- Create test suites using `@RunWith` and `@Suite.SuiteClasses` to run tests for several classes at once
- Build quick arrays and collections using array literals
  - `int[] quick = new int[] {1, 2, 3, 4};`
  - `List<Integer> list = Arrays.asList(7, 4, -3, 18);`
  - `Set<Integer> set = new HashSet<Integer>(Arrays.asList(5, 6, 10) );`

Javadoc

- Whenever you write a class to be used by clients, you should write full Javadoc comments for all of its public behavior (private methods should have comments, but they shouldn't be Javadoc).
- Don't repeat yourself or write vacuous comments.
- Each class constant or enumeration value can be commented.
- **precondition**: Something assumed to be true at the start of a call.
- **postcondition**: Something your method promises will be true at the end of its execution, if all preconditions were true at the start.
- **Assertions**: used to check preconditions

On a method or constructor:

tag	**description**
`@param` ***name description***	describes a parameter
`@return` ***description***	describes what value will be returned
`@throws` ***ExceptionType reason***	describes an exception that may be thrown (and what would cause it to be thrown)
{`@code` ***sourcecode*** }	for showing Java code in the comments
{`@inheritDoc`}	allows a subclass method to copy Javadoc comments from the superclass version

On a class header:

tag	**description**
`@author` ***name***	author of a class
`@version` ***number***	class's version number, in any format