Use of JSDoc

@Author: Yogesh Gaikwad

@Reference: http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html

All files, classes, methods and properties should be documented with JSDoc comments with the appropriate tags and types. Textual descriptions for methods, method parameters and method return values should be included unless obvious from the method or parameter name.

Inline comments should be of the `//` variety.

Avoid sentence fragments. Start sentences with a properly capitalized word, and end them with punctuation.

Comment Syntax

The JSDoc syntax is based on JavaDoc . Many tools extract metadata from JSDoc comments to perform code validation and optimizations. These comments must be well-formed.

```
/**
 * A JSDoc comment should begin with a slash and 2 asterisks.
 * Inline tags should be enclosed in braces like {@code this}.
 * @desc Block tags should always start on their own line.
 */
```

JSDoc Indentation

If you have to line break a block tag, you should treat this as breaking a code statement and indent it four spaces.

```
/**
 * Illustrates line wrapping for long param/return descriptions.
 * @param {string} foo This is a param with a description too long to fit in
 *     one line.
 * @return {number} This returns something that has a description too long to
 *     fit in one line.
 */
project.MyClass.prototype.method = function(foo) {
  return 5;
};
```

You should not indent the `@fileoverview` command.

Even though it is not preferred, it is also acceptable to line up the description.

```
/**
 * This is NOT the preferred indentation method.
 * @param {string} foo This is a param with a description too long to fit in
 *                     one line.
 * @return {number} This returns something that has a description too long to
 *                  fit in one line.
 */
project.MyClass.prototype.method = function(foo) {
  return 5;
};
```

HTML in JSDoc

Like JavaDoc, JSDoc supports many HTML tags, like <code>, <pre>, <tt>, <strong>, <ul>, <ol>, <li>, <a>, and others.

This means that plaintext formatting is not respected. So, don't rely on whitespace to format JSDoc:

```
/**
 * Computes weight based on three factors:
 *   items sent
 *   items received
 *   last timestamp
 */
```

It'll come out like this:

```
Computes weight based on three factors: items sent items received items
received
```

Instead, do this:

```
/**
 * Computes weight based on three factors:
 * <ul>
 * <li>items sent
 * <li>items received
 * <li>last timestamp
 * </ul>
 */
```

The JavaDoc style guide is a useful resource on how to write well-formed doc comments.

Top/File-Level Comments

A copyright notice and author information are optional. The top level comment is designed to orient readers unfamiliar with the code to what is in this file. It should provide a description of the file's contents and any dependencies or compatibility information. As an example:

```
/**
 * @fileoverview Description of file, its uses and information
 * about its dependencies.
 */
```

Class Comments

Classes must be documented with a description and a type tag that identifies the constructor.

```
/**
 * Class making something fun and easy.
 * @param {string} arg1 An argument that makes this more interesting.
 * @param {Array.<number>} arg2 List of numbers to be processed.
 * @constructor
 * @extends {goog.Disposable}
 */
project.MyClass = function(arg1, arg2) {
  // ...
};
goog.inherits(project.MyClass, goog.Disposable);
```

Method and Function Comments

Parameter and return types should be documented. The method description may be omitted if it is obvious from the parameter or return type descriptions. Method descriptions should start with a sentence written in the third person declarative voice.

```
/**
 * Operates on an instance of MyClass and returns something.
 * @param {project.MyClass} obj Instance of MyClass which leads to a long
 *     comment that needs to be wrapped to two lines.
 * @return {boolean} Whether something occured.
 */
function PR_someMethod(obj) {
  // ...
}
```

Property Comments

```
/** @constructor */
project.MyClass = function() {
  /**
   * Maximum number of things per pane.
   * @type {number}
   */
  this.someProperty = 4;
}
```

JSDoc Tag Reference

Tag	Template & Examples	Description

@author	`@author username@google.com (first last)`
	
	For example:
	
	`/**`
	` * @fileoverview Utilities`
		Document the author of a file or the owner of a test, generally only used in the`@fileoverview` comment.

	`for handling textareas.`
	` * @author kuth@google.com`
	`(Uthur Pendragon)`
	` */`	

@code	`{@code ...}`
	
	For example:
	
	`/**`
	` * Moves to the next`
	`position in the selection.`
	` * Throws {@code`
	`goog.iter.StopIteration}`
	`when it`
	` * passes the end of the`
	`range.`
	` * @return {Node} The node`
	`at the next position.`
	` */`
	`goog.dom.RangeIterator.prot`
	`otype.next = function() {`
	`  // ...`
	`};`
		Indicates that a term in a JSDoc description is code so it may be correctly formatted in generated documentation.

@const	`@const`
	`@const {type}`
	
	For example:
	
	`/** @const */ var MY_BEER =`
	`'stout';`
	
	`/**`
	` * My namespace's favorite`
	`kind of beer.`
	` * @const {string}`
	` */`
	`mynamespace.MY_BEER =`
	`'stout';`
	
	`/** @const */`
	`MyClass.MY_BEER = 'stout';`
	
	`/**`
	` * Initializes the request.`
	` * @const`
	` */`
	`mynamespace.Request.prototy`
	`pe.initialize = function()`
	`{`
	`  // This method cannot be`
	`overriden in a subclass.`
	`}`
		Marks a variable (or property) as read-only and suitable for inlining.
		
		A `@const` variable is a immutable pointer to a value. If a variable or property marked as`@const` is overwritten, JSCompiler will give warnings.
		
		The type declaration of a constant value can be omitted if it can be clearly inferred. An additional comment about the variable is optional.
		
		When `@const` is applied to a method, it implies the method is not only not overwritable, but also that the method is *finalized* — not overridable in subclasses.
		
		For more on `@const`, see the Constants section.

@constructo	`@constructor`	Used in a class's documentation to indicate the

r	*For example:*
	`/**`
	` * A rectangle.`
	` * @constructor`
	` */`
	`function GM_Rect() {`
	`  ...`
	`}`
		constructor.

@define	`@define {Type} description`
	For example:
	`/** @define {boolean} */`
	`var TR_FLAGS_ENABLE_DEBUG = true;`
	
	`/** @define {boolean} */`
	`goog.userAgent.ASSUME_IE = false;`
		Indicates a constant that can be overridden by the compiler at compile-time. In the example, the compiler flag `--define='goog.userAgent.ASSUME_IE=true'` could be specified in the BUILD file to indicate that the constant `goog.userAgent.ASSUME_IE` should be replaced with `true`.

@deprecated	`@deprecated Description`
	For example:
	`/**`
	` * Determines whether a node is a field.`
	` * @return {boolean} True if the contents of`
	` *     the element are editable, but the element`
	` *     itself is not.`
	` * @deprecated Use isField().`
	` */`
	`BN_EditUtil.isTopEditableField = function(node) {`
	`  // ...`
	`};`
		Used to tell that a function, method or property should not be used any more. Always provide instructions on what callers should use instead.

@dict	`@dict Description`
	For example:
	`/**`
	` * @constructor`
	` * @dict`
	` */`
	`function Foo(x) {`
	`  this['x'] = x;`
	`}`
		When a constructor (`Foo` in the example) is annotated with `@dict`, you can only use the bracket notation to access the properties of `Foo` objects. The annotation can also be used directly on object literals.

	`var obj = new Foo(123);`
	`var num = obj.x;  // warning`
	
	`(/** @dict */ { x: 1 }).x = 123;  // warning`	

@enum	`@enum {Type}`
	
	For example:
	
	`/**`
	` * Enum for tri-state values.`
	` * @enum {number}`
	` */`
	`project.TriState = {`
	`  TRUE: 1,`
	`  FALSE: -1,`
	`  MAYBE: 0`
	`};`	

@export	`@export`
	
	For example:
	
	`/** @export */`
	`foo.MyPublicClass.prototype.myPublicMethod = function() {`
	`  // ...`
	`};`
		Given the code on the left, when the compiler is run with the `--generate_exports` flag, it will generate the code:
		
		`goog.exportSymbol('foo.MyPublicClass.prototype.myPublicMethod',`
		
		`foo.MyPublicClass.prototype.myPublicMethod);`
		
		which will export the symbols to uncompiled code. Code that uses the `@export` annotation must either
		
		1. include `//javascript/closure/base.js`, or
		2. define both `goog.exportSymbol` and `goog.exportProperty` with the same method signature in their own codebase.

@expose	`@expose`
	
	For example:
	
	`/** @expose */`
	`MyClass.prototype.exposedProperty = 3;`
		Declares an exposed property. Exposed properties will not be removed, or renamed, or collapsed, or optimized in any way by the compiler. No properties with the same name will be able to be optimized either.
		
		`@expose` should never be used in library code, because it will prevent that property from ever getting removed.

@extends	`@extends Type`
	`@extends {Type}`
	
	For example:
	
	`/**`
	` * Immutable empty node list.`
	` * @constructor`
	` * @extends goog.ds.BasicNodeList`
	` */`
	`goog.ds.EmptyNodeList = function() {`
	`  ...`
	`};`
		Used with `@constructor` to indicate that a class inherits from another class. Curly braces around the type are optional.

@externs	`@externs`
	
	For example:
	
	`/**`
	` * @fileoverview This is an externs file.`
	` * @externs`
	` */`
	
	`var document;`
		Declares an externs file.

@fileoverview	`@fileoverview Description`
	
	For example:
	
	`/**`
	` * @fileoverview Utilities for doing things that require this very long`
	` * but not indented comment.`
	` * @author kuth@google.com (Uthur Pendragon)`
	` */`
		Makes the comment block provide file level information.

@implements	`@implements Type`
	`@implements {Type}`
	
	For example:
	
	`/**`
	` * A shape.`
	` * @interface`
	` */`
	`function Shape() {};`
	`Shape.prototype.draw =`
		Used with `@constructor` to indicate that a class implements an interface. Curly braces around the type are optional.

	`function() {};`
	
	`/**`
	` * @constructor`
	` * @implements {Shape}`
	` */`
	`function Square() {};`
	`Square.prototype.draw =`
	`function() {`
	`  ...`
	`};`	

@inheritDoc	`@inheritDoc`
	
	For example:
	
	`/** @inheritDoc */`
	`project.SubClass.prototype.`
	`toString() {`
	`  // ...`
	`};`
		Deprecated. Use `@override` instead.
		
		Indicates that a method or property of a subclass intentionally hides a method or property of the superclass, and has exactly the same documentation. Notice that `@inheritDoc`implies `@override`

@interface	`@interface`
	
	For example:
	
	`/**`
	` * A shape.`
	` * @interface`
	` */`
	`function Shape() {};`
	`Shape.prototype.draw =`
	`function() {};`
	
	`/**`
	` * A polygon.`
	` * @interface`
	` * @extends {Shape}`
	` */`
	`function Polygon() {};`
	`Polygon.prototype.getSides`
	`= function() {};`
		Used to indicate that the function defines an inteface.

@lends	`@lends objectName`
	`@lends {objectName}`
	
	For example:
	
	`goog.object.extend(`
	`    Button.prototype,`
	`    /** @lends`
	`{Button.prototype} */ {`
	`      isButton: function()`
	`{ return true; }`
	`    });`
		Indicates that the keys of an object literal should be treated as properties of some other object. This annotation should only appear on object literals.
		Notice that the name in braces is not a type name like in other annotations. It's an object name. It names the object on which the properties are "lent". For example, `@type {Foo}`means "an instance of Foo", but `@lends {Foo}` means "the constructor Foo".
		The JSDoc Toolkit docs have more information

		on this annotation.

@license or @preserve	`@license Description`
	
	For example:
	
	`/**`
	` * @preserve Copyright 2009 SomeThirdParty.`
	` * Here is the full license text and copyright`
	` * notice for this file. Note that the notice can span several`
	` * lines and is only terminated by the closing star and slash:`
	` */`
		Anything marked by `@license` or `@preserve` will be retained by the compiler and output at the top of the compiled code for that file. This annotation allows important notices (such as legal licenses or copyright text) to survive compilation unchanged. Line breaks are preserved.

@noalias	`@noalias`
	
	For example:
	
	`/** @noalias */`
	`function Range() {}`
		Used in an externs file to indicate to the compiler that the variable or function should not be aliased as part of the alias externals pass of the compiler.

@nosideeffects	`@nosideeffects`
	
	For example:
	
	`/** @nosideeffects */`
	`function noSideEffectsFn1() {`
	`  // ...`
	`};`
	
	`/** @nosideeffects */`
	`var noSideEffectsFn2 = function() {`
	`  // ...`
	`};`
	
	`/** @nosideeffects */`
	`a.prototype.noSideEffectsFn3 = function() {`
	`  // ...`
	`};`
		This annotation can be used as part of function and constructor declarations to indicate that calls to the declared function have no side-effects. This annotation allows the compiler to remove calls to these functions if the return value is not used.

@override	`@override`
	
	For example:
	
	`/**`
	` * @return {string} Human-readable representation of`
		Indicates that a method or property of a subclass intentionally hides a method or property of the superclass. If no other documentation is included, the method or property also inherits documentation from its superclass.

	`project.SubClass.`
	`  * @override`
	`  */`
	`project.SubClass.prototype.toString() {`
	`   // ...`
	`};`	

@param	`@param {Type} varname Description`
	
	For example:
	
	`/**`
	`  * Queries a Baz for items.`
	`  * @param {number} groupNum Subgroup id to query.`
	`  * @param {string|number|null} term An itemName,`
	`  *     or itemId, or null to search everything.`
	`  */`
	`goog.Baz.prototype.query = function(groupNum, term) {`
	`   // ...`
	`};`
		Used with method, function and constructor calls to document the arguments of a function. Type names must be enclosed in curly braces. If the type is omitted, the compiler will not type-check the parameter.

@private	`@private`
	`@private {type}`
	
	For example:
	
	`/**`
	`  * Handlers that are listening to this logger.`
	`  * @private {!Array.<Function>}`
	`  */`
	`this.handlers_ = [];`
		Used in conjunction with a trailing underscore on the method or property name to indicate that the member is private. Trailing underscores may eventually be deprecated as tools are updated to enforce `@private`.

@protected	`@protected`
	`@protected {type}`
	
	For example:
	
	`/**`
	`  * Sets the component's root element to the given element.  Considered`
	`  * protected and final.`
	`  * @param {Element} element Root element for the component.`
	`  * @protected`
		Used to indicate that the member or property is protected. Should be used in conjunction with names with no trailing underscore.

	` */`
	`goog.ui.Component.prototype.setElementInternal = function(element) {`
	`  // ...`
	`};`	

@return	`@return {Type} Description`
	
	For example:
	
	`/**`
	` * @return {string} The hex ID of the last item.`
	` */`
	`goog.Baz.prototype.getLastId = function() {`
	`  // ...`
	`  return id;`
	`};`
		Used with method and function calls to document the return type. When writing descriptions for boolean parameters, prefer "Whether the component is visible" to "True if the component is visible, false otherwise". If there is no return value, do not use an `@return` tag.
		Type names must be enclosed in curly braces. If the type is omitted, the compiler will not type-check the return value.

@see	`@see Link`
	
	For example:
	
	`/**`
	` * Adds a single item, recklessly.`
	` * @see #addSafely`
	` * @see goog.Collect`
	` * @see goog.RecklessAdder#add`
	` ...`
		Reference a lookup to another class function or method.

@struct	`@struct Description`
	
	For example:
	
	`/**`
	` * @constructor`
	` * @struct`
	` */`
	`function Foo(x) {`
	`  this.x = x;`
	`}`
	`var obj = new Foo(123);`
	`var num = obj['x'];  // warning`
	`obj.y = "asdf";  // warning`
	
	`Foo.prototype = /** @struct */ {`
	`  method1: function() {}`
	`};`
	`Foo.prototype.method2 =`
		When a constructor (`Foo` in the example) is annotated with `@struct`, you can only use the dot notation to access the properties of `Foo` objects. Also, you cannot add new properties to`Foo` objects after they have been created. The annotation can also be used directly on object literals.

	`function() {};  // warning`	

@supported	`@supported Description`
	
	For example:
	
	`/**`
	` * @fileoverview Event Manager`
	` * Provides an abstracted interface to the`
	` * browsers' event systems.`
	` * @supported So far tested in IE6 and FF1.5`
	` */`
		Used in a fileoverview to indicate what browsers are supported by the file.

@suppress	`@suppress {warning1|warning2}`
	
	For example:
	
	`/**`
	` * @suppress {deprecated}`
	` */`
	`function f() {`
	`  deprecatedVersionOfF();`
	`}`
		Suppresses warnings from tools. Warning categories are separated by `|`.

@template	`@template`
	
	For example:
	
	`/**`
	` * @param {function(this:T, ...)} fn`
	` * @param {T} thisObj`
	` * @param {...*} var_args`
	` * @template T`
	` */`
	`goog.bind = function(fn, thisObj, var_args) {`
	`...`
	`};`
		This annotation can be used to declare a template typename.

@this	`@this Type`
	`@this {Type}`
	
	For example:
	
	`pinto.chat.RosterWidget.extern('getRosterElement',`
	`/**`
	` * Returns the roster widget element.`
		The type of the object in whose context a particular method is called. Required when the`this` keyword is referenced from a function that is not a prototype method.

	` * @this`
	`pinto.chat.RosterWidget`
	` * @return {Element}`
	` */`
	`function() {`
	`  return`
	`this.getWrappedComponent_()`
	`.getElement();`
	`});`	

@type	`@type Type`
	`@type {Type}`
	
	For example:
	
	`/**`
	` * The message hex ID.`
	` * @type {string}`
	` */`
	`var hexId = hexId;`
		Identifies the type of a variable, property, or expression. Curly braces are not required around most types, but some projects mandate them for all types, for consistency.

@typedef	`@typedef`
	
	For example:
	
	`/** @typedef`
	`{(string|number)} */`
	`goog.NumberLike;`
	
	`/** @param`
	`{goog.NumberLike} x A`
	`number or a string. */`
	`goog.readNumber =`
	`function(x) {`
	`  ...`
	`}`
		This annotation can be used to declare an alias of a more complex type.