**Comments**

/\* \*/ - Use to comment a whole block. For example, at the top of a class to describe a class.

/// - Use at the top of methods to describe them.

// - Use to comment lines within the code whenever necessary.

**Naming**

Pascal Case - the first letter of each word should be capitalized.  
Camel Case – the first letter is in lower case, while all other words start with an uppercase.

* Descriptive variable and method names. E.g orderTotal not oTot.
* Public Variable/Local Parameter Names – Names should be in Camel case, e.g numberOfItems
* Private Variable Names – Names should be in Camel case with an underscore prefix, e.g \_stateTax
* Variable property – Names should be in Pascal case, e.g TotalCost.
* Method Names – Names should be in Pascal case, e.g CalculateCostOfPurchase.
* Class Names – Names should be in Pascal case with I Prefix, e.g IOrders.
* Interfaces – Names should be in Pascal case, e.g PurchaseRecords.
* Constants – Constants should be all-uppercase, with underscores to separate words. E.g MAX\_LENGTH.
* Enum – Names should be in Pascal case e.g OrderPending.

**Indentations**

* Tabs or spaces can be used for indentations.
* Indention size can be from 2 to 4 spaces.

**Nesting**

* Nesting depth can be maximum of three.
* Use methods to avoid deeper nests.

**Braces**

* Always use curly braces under statements. e.g
  + forach(....)  
    {  
    }
  + if(...){  
    }

**Fancy Coding**

* Avoid fancy coding as much as possible.

**Line length**

* Each line of code should not exceed 80 characters.

**Layout**

* One declaration per line e.g
  + int min;
  + int max;

**Cyclomatic Complexity**

* Complexity from 1 to 10: Normal
* Complexity from 11 to 20: Moderate
* Complexity from 21 to 50: Risky
* Complexity over 50: Unstable/Unmaintainable

**Test Coverage**

* Tests should be written to target all areas of the code at least once
* Every requirement should be tested at least once
* Steps to ensure full code coverage:
  + Develop a Test Plan based on Design Specifications
  + Understand the Software Requirements and develop Functional Test Specifications
  + Develop several sets of test cases, such as:
    - Smoke (Build Verification Testing) Test cases – Ensure the most important functions work
    - Sanity Test cases – Ensure bugs have been fixed and no issues have been introduced due to code changes
    - Regression Test Cases – Ensure code changes have no adverse effects on existing features
    - Negative Test Cases – To test that the system does not do what it shouldn’t
  + Static code analysis to search for things such as security flaws and to enforce coding standards

**Documentation**

* To ensure that code is properly documented, a good approach is to measure the undocumented public API and report it as a ratio (how many public methods are undocumented vs the total number of public methods