Coding conventions and style

Malhar GitHub repository contains operator library and demos built on top of the DataTorrent platform. The code is open source, viewable and downloadable by all. Anyone can make code submissions to the repository to add new features or fix bugs. The process to do so is to first make a personal fork of the repository, make changes in the fork and then generate a pull request with the changes against the Malhar repository.

Malhar administrators look at pull requests regularly and merge them into the repository. The pull requests have to follow certain guidelines in order to minimize the possibility of issues and problems arising from the merge, to keep the code maintainable going forward and to keep the licensing. The guidelines are as follows

- The submitted code should compile without any errors with JDK 1.6.
- The code should not have any warnings. All warnings should be resolved prior to submission. If there is a scenario where legitimate code has warnings SuppressWarnings annotation should be used appropriately.
- All classes, methods and constants should be well java documented. The code will not be accepted without documentation and the reviewer will comment on the pull request accordingly.
- All code should have unit tests and the tests should pass successfully.
- New files should not have any license. A license will be added by the reviewer. License should not be modified for existing files.
- There should be no author, date or organization references in the comments or java documentation

It is highly recommended that the code follow a certain coding style so that it is consistent with the existing code that is already in the repository. It helps when someone new who has not worked on that part of the code is looking at it. Below is a visual example that expresses the coding style. The salient points are noted after the code listing.

```
package com.datatorrent.lib.util;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Creates empty files for a list of specified file paths. If a file exists it is
truncated.
 */
public class ClassA
{
```

```
  /**
   * The list of file paths.
   */
  private List<String> filePaths = new ArrayList<String>();
  /**
   * A flag that controls if the path elements in a file path should be created if they
don't exist
   */
  private boolean createPath = false;

  /**
   * Create the empty files. Empty files are created for the specified file paths. If
{@link #createPath}
   * is set to true, for each file path if any element in the path doesn't exist it is
created. If a file exists it
   * is truncated.
   * @throws IOException if an I/O error occurs
   */
  public void createEmptyFiles() throws IOException
  {
    for (String filePath : filePaths) {
      File f = new File(filePath);
      if (createPath) {
        f.mkdirs();
      }
      if (f.exists()) {
        truncateFile(f);
      }
      else {
        f.createNewFile();
      }
    }
  }

  /**
   * Truncate the given file.
   * @param f the file
   * @throws IOException if an I/O error occurs
   */
  private void truncateFile(File f) throws IOException
  {
    try {
      FileOutputStream fout = new FileOutputStream(f);
      fout.close();
    }
    catch (FileNotFoundException fne) {
      // Not possible
    }
  }

  /**
   * Add a file path to the list of file paths.
   * @param filePath the file path
   */
  public void addFilePath(String filePath)
  {
    filePaths.add(filePath);
```

```
  }

  /**
   * Clear the list of file paths.
   */
  public void clearFilePaths()
  {
    filePaths.clear();
  }

  /**
   * Return whether create path flag is set.
   * @return the create path flag
   */
  public boolean isCreatePath()
  {
    return createPath;
  }

  /**
   * Set the create path flag. If it is specified as true then path elements in a file
path are created if they
   * don't exist when creating the files otherwise they are not created.
   * @param createPath the create path flag
   */
  public void setCreatePath(boolean createPath)
  {
    this.createPath = createPath;
  }
}
```

- Notice that imports are mentioned individually instead of lumping them into a single import with the wildcard ‘*’
- The indentation is 2 spaces and not tab
- The opening braces for classes and methods start on a new lines whereas the braces for loops and conditions on the same line. For a try/catch block the catch is in a new line after the ending brace of the try.
- All public classes and methods are java documented