The Change Counter
System Design Document

Gangof4 Team

Abhishek Minde
Joao Carlos Almeida
Krishnan Ganesan
Nisha Narayan

Version : 1.0

Date : April 14th, 2009

Table of Contents

GangOf4	**Version**: 1.0
System Design Document	**Date**: April 14th, 2009

Document Revision History

Version	Date	Author	Reviewer	Updates
1.0	April 14th, 2009	Team GO4	Team	Initial Version

1. Purpose:

This document describes the design of the Code Change counter system. The audience of this document involves the development team and the instructor. This document describes use cases, class diagrams and it also discusses some key operations in terms of sequence diagrams.

2. Use Cases:

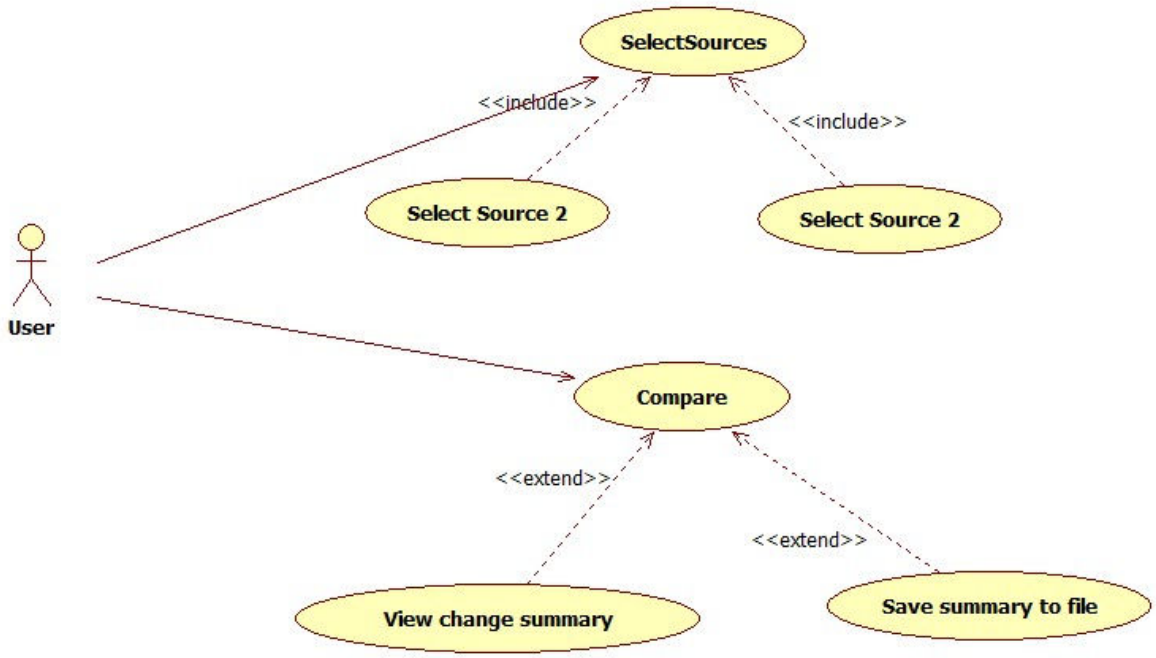


Fig 2.1: Use case diagram

Diagram 2.1 depicts the use case diagram for the system.

2.1 Actors

User:

User is responsible for selecting source files and viewing the comparison results. User can select two source files from the given graphical user interface. Then user can compare these two files. After doing the comparison, user can view the change summary and or write the change summary to a file.

2.2 Use cases

Select Sources:

The user actor can select two files. The user mentions filenames (absolute path) for the two files to be compared.

Compare:

A user can invoke comparison of the selected two files. User can also request to view the difference between two files. Second file is considered to be the later version of the first file. Based on the given two versions of a file, a user can view the total number of lines added, deleted and modified.

3. Class diagram:

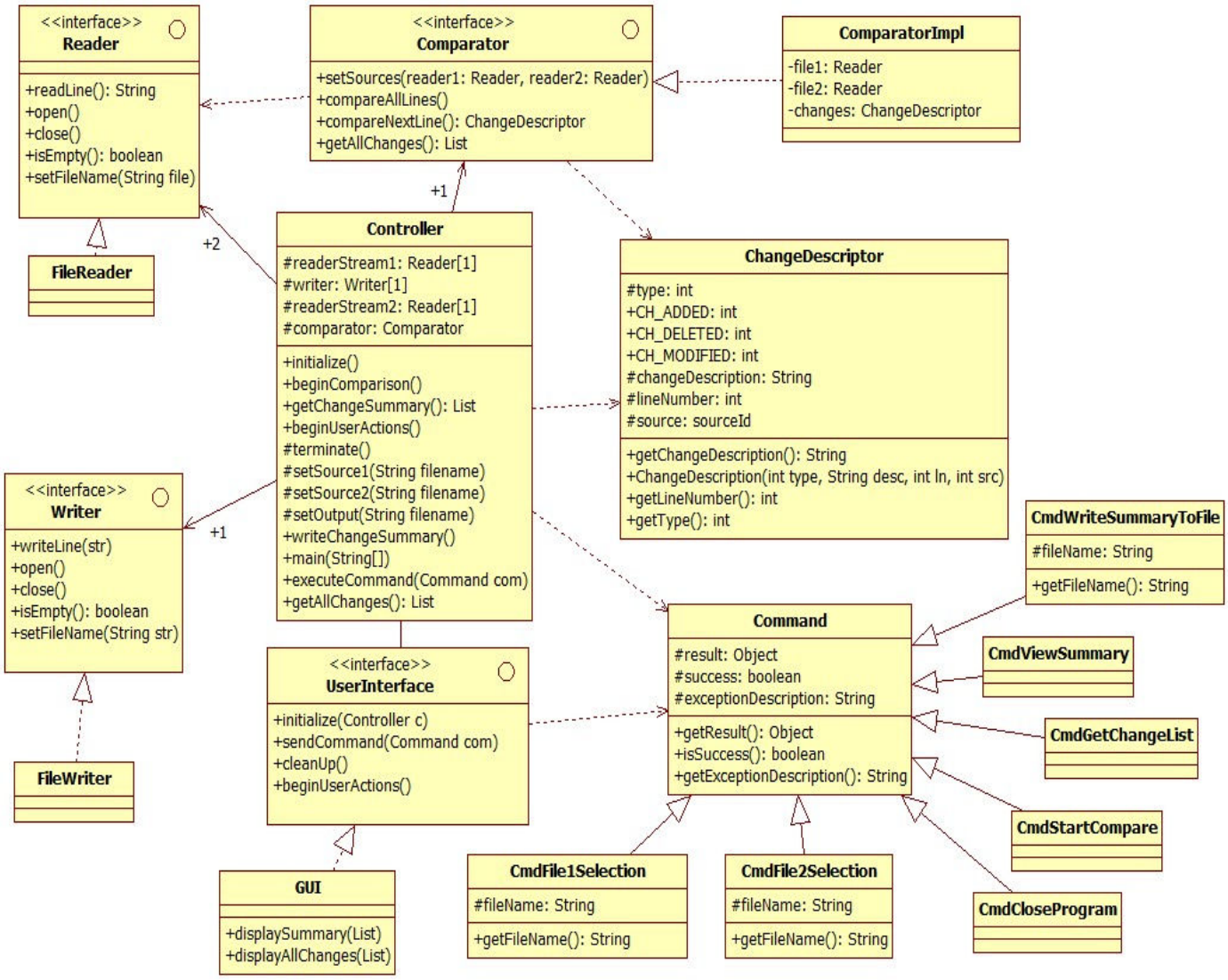


Fig 3.1 : Class diagram

3.1 Interfaces

Reader:

This interface has a responsibility to read a source file. It has following methods

- String ReadLine():
  This function causes the reader object to read a line from the open file. The text line is returned as a string object to the caller.
- Open():

This function opens the file and keeps it ready for being read.

- Close:
  This function closes the file. No further reading from that file is possible after calling this function.
- setFileName(String):
  This sets the name of the file to be read.
- Boolean IsEmpty():
  This function returns true if the end of file is reached, i.e. the complete file has been read. Otherwise, it returns false.

Writer

This interface has a responsibility to read a source file. It has following methods

- writeLine(String):
  This function causes the writer object to write a string to the opened file. The text line is passed as a string parameter to this method.
- Open():
  This function opens the file and keeps it ready for being written.
- Close:
  This function closes the file. No further writing to that file is possible after calling this function.
- setFileName(String):
  This sets the name of the file to be written.

UserInterface

This interface defines the functionality that is needed for the user interface. This interface can be implemented by a text console or a GUI.

- displaySummary(list): This contains a list of strings. Each line is a message. The summary of changes is represented in a list of strings.

- displayAllchanges(list): This contains a list of strings. Each line is a change description. The list of changes is represented in a list of strings.

Comparator

This interface has a responsibility to read a source file. It has following methods

- setSources(Reader reader1, Reader reader2):
  This method accepts two reader objects. Reader1 represents the version1 of a file, reader2 represents the version2 of the file. Comparator reads from these two files and analyzes the differences between them.
- compareAllLines():
  Calling this function compares two files that have been set using setSources().

- CompareNextLine():
  This function compares two lines from two reads. It compares only one line.

- List getAllChanges():
  This function returns a list of differences. It returns a list that contains ChangeDescriptor objects.

3.2 Classes

Controller

The controller is the heart of the change counter. The program boots up with this class.

- Initialize():
  This method should be called first. This initializes the user interface.

- beginComparison():
  This method should be called after selection two source files. This initiates the comparison between two files.

- List getChangeSummary():
  This method returns the list of summary. This returns summary as a list of strings. This method internally uses Comparator's getchangesummary() method.

- setSource1(String file1)
  This method accepts a filename of the input file1. It creates a reader1 for that filename.

- setSource2(String file2)
  This method accepts a filename of the input file2. It creates a reader2 for that filename.

- setOutput(String file)
  This method accepts a filename for an output file where change summary will be written.

- writeChangeSummary()
  This method causes the controller to write the change summary to the set output file. Precondition for this method is, two input files and one output file should have been selected and comparison must have been done.

- executeCommand(Command c):
  This method accepts one of the command objects. Based on the command subclass, it invokes certain operations in the controller.

ChangeDescriptor

ChangeDescriptor class is responsible to store change information for a single difference instance in two files. This means, for each line added, deleted or modified there will be one corresponding ChangeDescriptor object.

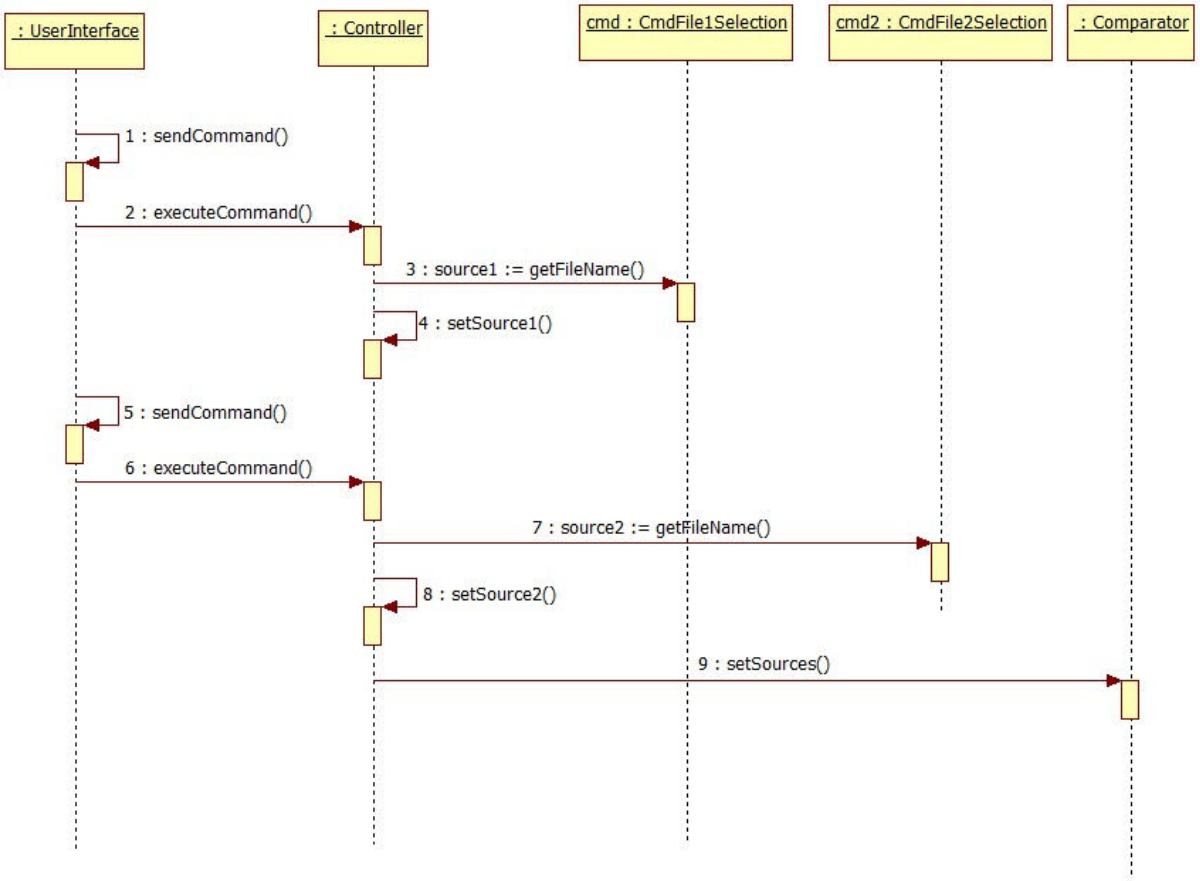
- String getChangeDescription():
  This method returns information about the change. This information will be set by the comparator.

- int getLineNumber():
  This method returns line number of the changed line.

- int getType():
  This method returns the type of the change. E.g. modification it returns CH_MODIFIED.

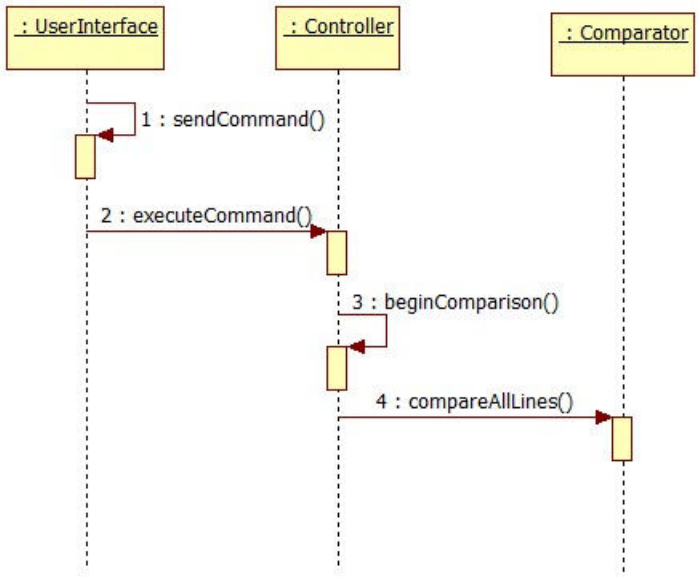
4. Sequence diagrams

4.1 Initialization scenario:



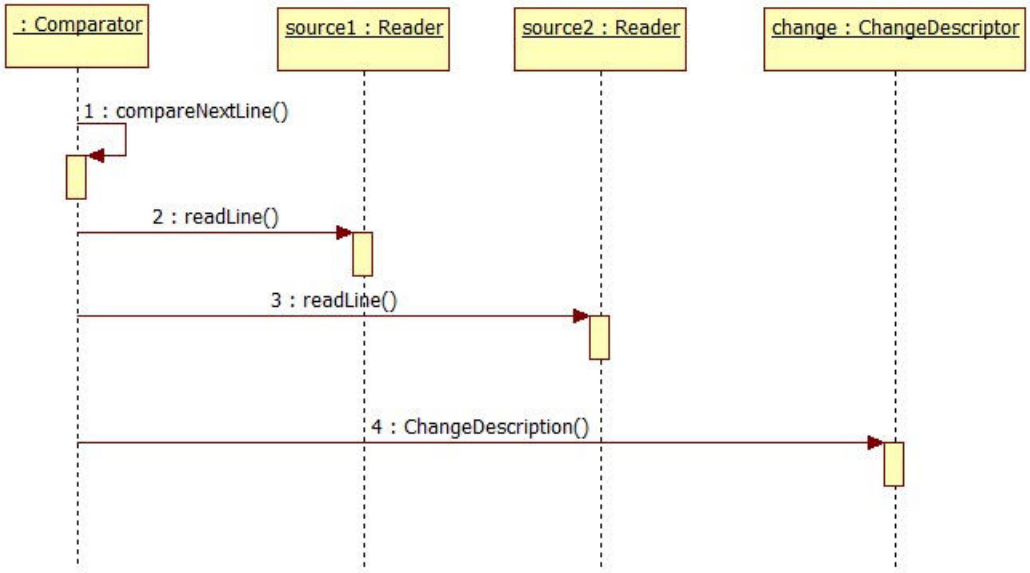
Above figure depicts how initialization takes place. When user selects a filename for source1, userinterface sends a command object to the controller. Then controller sets the source1 by instantiating a Reader object for the given filename. Similarly, source2 and output files are also initialized.

4.2 Compare operation scenario:



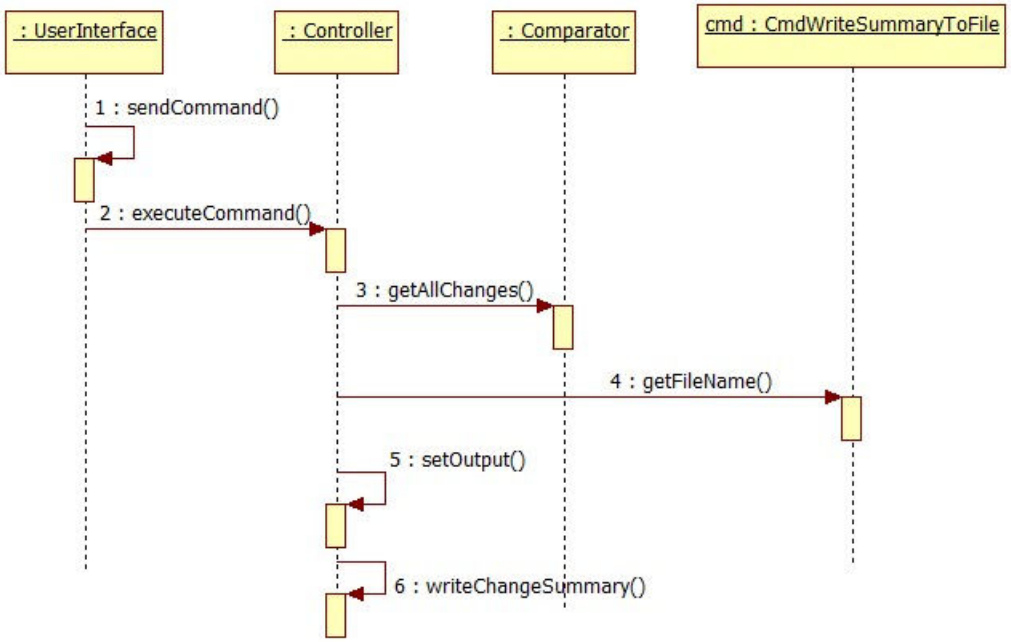
Above sequence diagram depicts how comparison operation is executed. When the user sends a command to begin comparison, user interface forwards the command object to the controller. The controller then calls the beginComparison() method which internally calls compareAllLines() method.

4.3 One line comparison:



Above sequence diagram depicts how comparison of two lines is executed. Controller calls the compareNextLine() method which internally reads two lines from two sources and then if it finds the difference it creates a corresponding ChangeDescription object.

4.4 Writing change summary to a file:



Above sequence diagram depicts how a user calls for summary of changes. When the user sends a command to save summary to an output file, user interface forwards the command object to the controller. The controller then calls the retrieves the changes and then it invokes the method writeChangeSummary(), which writes all the change summary to the set output file.