```
This file was updated on Monday, 2015-11-23 at 12:44 PM


======================================================================
main.cpp
======================================================================
```

```
/*
 * File:   main.cpp
 * Author: Jesse Heines, heines@cs.uml.edu
 *
 * Created on November 23, 2015, 10:23 AM
 */

#include <iostream>  // for cout and friends
#include <sstream>   // for string streams
#include <string>    // for the STL string class
#include <regex>     // for regular expressions

#include "jmhUtilities.h"

using namespace std;

// forward references
void ParseCommandString( string strUserEntry ) ;
void ProcessAddCommand( string strUserEntry ) ;
void ProcessAddElementCommand( string strUserEntry ) ;
void ProcessAddAttributeCommand( string strUserEntry ) ;
void ProcessPrintCommand( string strUserEntry ) ;


/**
 * The standard C++ main function.
 * @param argc number of command line arguments
 * @param argv array of command line arguments
 * @return
 */
int main(int argc, char** argv) {

  // display program title
  cout << "=============================" << endl ;
  cout << "Assignment No. 7 Starter Code" << endl ;
  cout << "=============================" << endl ;

  // string entered by the user in response to the command prompt
  string strUserEntry = "" ;

//  // prompt the user to enter a command string
//  // version 1 without using a regular expression
//  while ( ! jmhUtilities::caseInsCompare( jmhUtilities::trim( strUserEntry ), "quit" ) ) {
//    cout << "\nYour command: " ;
//    getline( cin, strUserEntry ) ;
//  }

  // prompt the user to enter a command string
  // version 2 using a regular expression
  regex reQuit( "\\s*quit\\s*", regex::icase ) ;
  while( ! regex_match( strUserEntry, reQuit ) ) {
    cout << "\nYour command: " ;
    getline( cin, strUserEntry ) ;

    // if the user didn't enter 'quit', go parse the command string
    if ( ! regex_match( strUserEntry, reQuit ) ) {
      ParseCommandString( strUserEntry ) ;
    }
  }

  return EXIT_SUCCESS ;
}


```

```
/**
 * Check for a valid basic command.
 * @param strUserEntry command string entered by the user
 */
void ParseCommandString( string strUserEntry ) {
  // regular expressions for basic commands
  regex reBasicAddCommand( "\\s*add.*", regex::icase ) ;
  regex reBasicPrintCommand( "\\s*print.*", regex::icase ) ;
  regex reBasicHelpCommand( "\\s*help.*", regex::icase ) ;

  // test for each basic command in turn
  if ( regex_match( strUserEntry, reBasicAddCommand ) ) {
    ProcessAddCommand( strUserEntry ) ;
  } else if ( regex_match( strUserEntry, reBasicPrintCommand ) ) {
    ProcessPrintCommand( strUserEntry ) ;
  } else if ( regex_match( strUserEntry, reBasicHelpCommand ) ) {
    cout << "  Commands are:" << endl ;
    cout << "    add element parent_name element_name" << endl ;
    cout << "    add attribute parent_name attribute_name attribute_value" << endl ;
    cout << "    print" << endl ;
    cout << "    help (this command)" << endl ;
    cout << "    quit" << endl ;
  } else {
    cout << "  Invalid command.  Acceptable commands are 'add', 'print', 'help', and 'quit'." << endl ;
  }
}


/**
 * Handle an add command entered by the user
 * @param strUserEntry command string entered by the user
 */
void ProcessAddCommand( string strUserEntry ) {
  // regular expressions for the second parameter in the add command
  regex reAddElementCommand( "\\s*add\\s+element.*", regex::icase ) ;
  regex reAddAttributeCommand( "\\s*add\\s+attribute.*", regex::icase ) ;

  // test for each possible second parameter in turn
  if ( regex_match( strUserEntry, reAddElementCommand ) ) {
    ProcessAddElementCommand( strUserEntry ) ;
  } else if ( regex_match( strUserEntry, reAddAttributeCommand ) ) {
    ProcessAddAttributeCommand( strUserEntry ) ;
  } else {
    cout << "  Invalid add command: 2nd parameter must be 'element' or 'attribute'." << endl ;
  }
}


/**
 * Handle an add element command entered by the user
 * @param strUserEntry command string entered by the user
 */
void ProcessAddElementCommand( string strUserEntry ) {
  // the what variable is actually an array that will be populated by the regex_match function
  //    when matched groups are found
  cmatch what;
  // what[0] contains the entire matched string
  // what[1] contains the first matched group
  // what[2] contains the second matched group
  // what[3] etc.

  // regular expression to pick out the name of the parent to which the new element is to be added
  //    and the name of the new element itself
  regex reAddElementCmd( "^\\s*add\\s*element\\s(\\w+)\\s(\\w+)(.*)$", regex::icase ) ;

  // note that the following variant of the regex_match command requires a C string, not an STL string
  if ( regex_match( strUserEntry.c_str(), what, reAddElementCmd ) ) {
    cout << "  You have specified that you want to add a new element named '" << what[2]
         << "' to parent element '" << what[1] << "'." << endl ;
  } else {
    cout << "  Invalid 'add element' command." << endl ;
    cout << "    'add element' must be followed by two more parameters:" << endl ;
    cout << "      (1) the name of the parent to which the new element is to be added, and" << endl ;
    cout << "      (2) the name of the new element itself." << endl ;
  }
}


/**
 * Handle an add attribute command entered by the user
 * @param strUserEntry command string entered by the user
 */
void ProcessAddAttributeCommand( string strUserEntry ) {
  // the what variable is actually an array that will be populated by the regex_match function
  //    when matched groups are found
  cmatch what;
  // what[0] contains the entire matched string
  // what[1] contains the first matched group
  // what[2] contains the second matched group
  // what[3] etc.

  // regular expression to pick out the name of the element to which the new attribute is to be added,
  //    the name of the new attribute, and the value of that attribute
  regex reAddAttributeCmd( "^\\s*add\\s*attribute\\s(\\w+)\\s(\\w+)\\s(\\w+)(.*)$", regex::icase ) ;

  // note that the following variant of the regex_match command requires a C string, not an STL string
  if ( regex_match( strUserEntry.c_str(), what, reAddAttributeCmd ) ) {
    cout << "  You have specified that you want to add a new attribute named '" << what[2]
         << "' with a value of '" << what[3] << "' to element '"  << what[1] << "'." << endl ;
  } else {
    cout << "  Invalid 'add attribute' command." << endl ;
    cout << "    'add attribute' must be followed by three more parameters:" << endl ;
    cout << "      (1) the name of the element to which the new attribute to be added," << endl ;
    cout << "      (2) the name of the new attribute to be added, and " << endl ;
    cout << "      (3) the value of the new attribute to be added." << endl ;
  }
}


/**
 * Handle a print command entered by the user
 * @param strUserEntry command string entered by the user
 */
void ProcessPrintCommand( string strUserEntry ) {
  cout << "  ... add your code to handle a print command here ..." << endl ;
}
```

```
======================================================================
jmhUtilities.h
======================================================================
```

```
/*
 * File:   jmhUtilities.h
 * Author: Jesse Heines, heines@cs.uml.edu
 *
 * Created on November 23, 2015, 10:46 AM
 */

#ifndef JMHUTILITIES_H
#define JMHUTILITIES_H

#include <string>

using namespace std ;

class jmhUtilities {
public:
  /**
   * NetBeans-supplied default constructor.
   */
  jmhUtilities();

  /**
   * NetBeans-supplied copy constructor.
   */
  jmhUtilities(const jmhUtilities& orig);

  /**
   * NetBeans-supplied destructor.
   */
  virtual ~jmhUtilities();

  /**
   * Trim leading and trailing white space (spaces, tabs, and newlines) from the
   *    string passed as an argument and return the trimmed string.
   * This version is more sophisticated than the one above.  It uses iterators
   *    and an improved search technique.
   * @param str string to trim (a copy, so that the original is not destroyed)
   * @return a copy of the original string with leading and trailing white space removed
   */
  static string trim( string str ) ;

  /**
   * Compare two characters in a case-insensitive manner by converting them both to
   *    uppercase and then testing if they are equal.
   * @param a the first character to compare
   * @param b the second character to compare
   * @return true if the two uppercase characters are equal, false otherwise
   * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
   */
  static inline bool caseInsCharCompareN(char a, char b) ;
  /**
   * Compare two strings in a case-insensitive manner using the companion case-insensitive
   *    character comparison helper function.
   * @param s1 the first string to compare
   * @param s2 the second string to compare
   * @return true if the two uppercase strings are equal, false otherwise
   * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
   */
  static bool caseInsCompare(const string& s1, const string& s2) ;

  private:

};

#endif /* JMHUTILITIES_H */
```

```
======================================================================
jmhUtilities.cpp
======================================================================
```

```
/*
 * File:   jmhUtilities.cpp
 * Author: Jesse Heines, heines@cs.uml.edu
 *
 * Created on November 23, 2015, 10:46 AM
 */

#include "jmhUtilities.h"
#include <string>
using namespace std ;

/**
 * NetBeans-supplied default constructor.
 */
jmhUtilities::jmhUtilities() {
}

```

```
/**
 * NetBeans-supplied copy constructor.
 */
jmhUtilities::jmhUtilities(const jmhUtilities& orig) {
}

/**
 * NetBeans-supplied destructor.
 */
jmhUtilities::~jmhUtilities() {
}


/**
 * Trim leading and trailing white space (spaces, tabs, and newlines) from the
 *    string passed as an argument and return the trimmed string.
 * This version is more sophisticated than the one above.  It uses iterators
 *    and an improved search technique.
 * @param str string to trim (a copy, so that the original is not destroyed)
 * @return a copy of the original string with leading and trailing white space removed
 */
string jmhUtilities::trim( string str ) {
  // define the iterator to be used in both loops
  string::iterator it ;

  // search for leading white space characters
  it = str.begin() ;
  while ( *it == ' ' || *it == '\t' || *it == '\n' ) {
    str.erase( it ) ;   // erase the found whitespace character
    it = str.begin() ;  // reposition the iterator to the first character in the string
  }

  // search for trailing white space characters
  it = str.end() - 1 ;
  while ( *it == ' ' || *it == '\t' || *it == '\n' ) {
    str.erase( it ) ;     // erase the found whitespace character
    it = str.end() - 1 ;  // reposition the iterator to the last character in the string
  }

  // return the result
  return str ;
}


/**
 * Compare two characters in a case-insensitive manner by converting them both to
 *    uppercase and then testing if they are equal.
 * @param a the first character to compare
 * @param b the second character to compare
 * @return true if the two uppercase characters are equal, false otherwise
   * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
 */
inline bool jmhUtilities::caseInsCharCompareN(char a, char b) {
  return ( toupper( a ) == toupper( b ) ) ;
}

/**
 * Compare two strings in a case-insensitive manner using the companion case-insensitive
 *    character comparison helper function.
 * @param s1 the first string to compare
 * @param s2 the second string to compare
 * @return true if the two uppercase strings are equal, false otherwise
   * @see https://www.safaribooksonline.com/library/view/c-cookbook/0596007612/ch04s14.html
 */
bool jmhUtilities::caseInsCompare( const string& s1, const string& s2 ) {
   return ( ( s1.size() == s2.size() ) &&
             equal( s1.begin(), s1.end(), s2.begin(), caseInsCharCompareN ) ) ;
}


======================================================================
```