Hom01

TokenCheck:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom01;

import java.io.Serializable;

import java.util.Scanner;

/\*\*

\* Takes a String made of tokens and determines their datatype.

\* @author Jacob Huesman

\*/

public class TokenCheck implements Serializable {

/\*\*

\* Declare instance variables.

\*/

private String data;

private String delimiter;

private String type;

/\*\*

\* Default constructor creates an object with an empty String, no specified delimiter, and unknown for the token type.

\*/

public TokenCheck(){

data = new String();

delimiter = new String();

type = "unknown";

}

/\*\*

\* Overloaded constructor that creates an object with the String and delimiter set to the parameter values and token type set to its appropriate value.

\* @param data String to be checked for tokens.

\* @param delimiter The delimiter used to separate the token from the String.

\*/

public TokenCheck(String data, String delimiter) {

this.data = data;

this.delimiter = delimiter;

determineTokenType();

}

/\*\*

\* Determines the token type for the String.

\*/

private void determineTokenType(){

if(checkTokens("int")){

type = "int";

} else if (checkTokens("double")){

type = "double";

} else if (checkTokens("boolean")){

type = "boolean";

} else if (checkTokens("char")){

type = "char";

} else {

type = "mixed";

}

}

/\*\*

\* Checks Tokens to see if they belong to a specified datatype.

\* @param dataType Datatype to check for.

\* @return true - if the tokens belong to that datatype; false - otherwise

\* @throws IllegalDataTypeException Throws if the method can't check for that datatype.

\*/

private boolean checkTokens(String dataType) throws IllegalDataTypeException {

//Initiate Scanner object and set it to use the specified delimiter. Also removes all whitespace to make scanning data accidental whitespace more accurate.

String operatingData = data.replaceAll(" ", "");

Scanner scan = new Scanner(operatingData);

scan.useDelimiter(delimiter.trim());

//Check what datatype the method is looking for.

switch (dataType.toLowerCase()){

case "int":

while(scan.hasNextInt()){

scan.nextInt();

}

if(scan.hasNext()){

return false;

}

return true;

case "double":

while(scan.hasNextDouble()){

scan.nextDouble();

}

if(scan.hasNext()){

return false;

}

return true;

case "boolean":

case "bool":

while(scan.hasNextBoolean()){

scan.nextBoolean();

}

if(scan.hasNext()){

return false;

}

case "character":

case "char":

while(scan.hasNext()){

if(scan.next().trim().length() != 1){

return false;

};

}

return true;

}

//If the datatype specified isn't handled by this method throw an IllegalDataTypeException.

throw new IllegalDataTypeException( "Datatype not supported." );

}

/\*\*

\* Sets the data and delimiter. Also recalculates the token type and stores the result.

\* @param data String to be checked for tokens.

\* @param delimiter Delimiter to separate Strings by.

\*/

public void setDataAndDelimiter( String data, String delimiter ){

this.data = data;

this.delimiter = delimiter;

determineTokenType();

}

/\*\*

\* Gets the String of data.

\* @return data : String

\*/

public String getData(){

return data;

}

/\*\*

\* Returns the delimiter set for this TokenCheck object.

\* @return delimiter : char

\*/

public String getDelimiter(){

return delimiter;

}

/\*\*

\* Returns the type of data passed to the constructor.

\* @return type : String

\*/

public String getType(){

return type;

}

/\*\*

\* Returns a String representation of the data and delimiter.

\* @return String

\*/

public String toString(){

return String.format("| %1$-60s | %2$-15s | %3$-20s\n", "Data: " + data, "Delimiter: " + delimiter, "Type: " + type);

}

/\*\*

\* Checks to see if this TokenCheck object equals another TokenCheck object.

\* @param token TokenCheck object to check against.

\* @return true - if they are equal; false - otherwise

\*/

public boolean equals(TokenCheck token){

if(this.data.equals(token.getData()) && this.delimiter.equals(token.getDelimiter()) && this.type.equals(token.getType())){

return true;

}

return false;

}

}

Hom01Client:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom01;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

import java.io.PrintWriter;

import java.util.ArrayList;

import java.util.Scanner;

import java.util.regex.PatternSyntaxException;

/\*\*

\* Client class to test functionality of TokenCheck.

\* @author Jacob Huesman

\*/

public class Hom01Client {

/\*\*

\* @param args the command line arguments

\*/

public static void main(String[] args) throws InterruptedException{

FilePath filePath = new FilePath();

Scanner scan;

ArrayList<TokenCheck> tokenArray = new ArrayList<>();

File file;

/\*\*

\* Step 1:

\* Read in a list of String-delimiter pairs from a text file.

\*/

//Makes an empty thread that causes the the main client to wait until the thread finishes executing. The thread finishes executing when the user either selects a file or closes the window.

Thread thread = new Thread(new Runnable(){

public void run(){try {while(true){Thread.sleep(1000);}} catch (InterruptedException e){}}});

thread.start();

//Runs an instance of GetFile. GetFile is a GUI that prompts the user for a file to import data from.

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

new GetFile(filePath, thread).setVisible(true);

}

});

//The client will wait for the user to select a file before continuing.

thread.join();

//Checks if the user provided a file. Triggered if user exits the application without importing a file.

if(!filePath.getIsFilePath()){

System.out.println("User failed to provide a valid file.");

return;

}

//Creates a new File object based on the path collected earlier and passes it to the Scanner constructor.

try {

file = new File(filePath.getFilePath());

scan = new Scanner(file);

} catch (FileNotFoundException e){

System.out.println("Couldn't find the file!");

return;

} catch (NullPointerException e){

System.out.println("Couldn't find the file!");

return;

}

//Scan in the String-delimiter pairs, construct a TokenCheck object from the pair, and store the object in an array.

try {

while(scan.hasNextLine()){

String first = scan.nextLine();

if(scan.hasNextLine()){

tokenArray.add(new TokenCheck(first, scan.nextLine()));

} else {

System.out.println("Invalid input file. Please check formatting.");

return;

}

}

}

catch (PatternSyntaxException e){

System.out.println("Unsupported character used as a delimiter");

}

/\*\*

\* Step 2:

\* Manually creates two additional instances of the TokenCheck class. One instance demonstrates the default constructor and the set methods. The other demonstrates the overloaded constructor.

\*/

//Default constructor and set method

tokenArray.add((new TokenCheck()));

tokenArray.get(tokenArray.size()-1).setDataAndDelimiter("6.7:8.3:9.4:4", ":");

//Overloaded Constructor

tokenArray.add(new TokenCheck("a, 6, 5, f5, 6, 5", ","));

/\*\*

\* Step 3:

\* This step is implemented in the SortTokenCheck class. The class is static and sorts the array passed to it.

\* Note: For ease of implementation the class sorts the actual object passed to it and does not return anything.

\*/

SortTokenCheck.sort(tokenArray);

/\*\*

\* Step 4:

\* Print out the contents of the sorted ArrayList using the toString method.

\*/

System.out.format("\nSorted ArrayList: \n");

for(TokenCheck token: tokenArray){

System.out.print(token.toString());

}

/\*\*

\* Step 5:

\* Write the contents of the sorted ArrayList as objects to a file named home01objects.

\* The file will be stored in the same folder the original file of data and delimiters was pulled from.

\*/

try {

//First file.

FileOutputStream fos = new FileOutputStream(file.toPath().getParent().resolve("home01objects").toFile(), false);

ObjectOutputStream oos = new ObjectOutputStream(fos);

for(TokenCheck token: tokenArray){

oos.writeObject(token);

}

oos.close();

//Second file.

fos = new FileOutputStream(file.toPath().getParent().resolve("home01.txt").toFile(), false);

PrintWriter pw = new PrintWriter(fos);

for(TokenCheck token: tokenArray){

pw.println(token.getData());

pw.println(token.getDelimiter());

pw.println(token.getType());

}

pw.close();

} catch (FileNotFoundException ex) {

System.out.println("Couldn't find file.");

} catch (IOException ex) {

System.out.println("Couldn't write to file.");

}

}

}

FilePath:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom01;

import java.io.File;

/\*\*

\* Simple class designed to hold the path to a file for storage across multiple threads.

\* @author Jacob Huesman

\*/

public class FilePath {

private String filePath;

private boolean isFilePath;

/\*\*

\* Default Constructor

\*/

public FilePath(){

filePath = "";

isFilePath = false;

}

/\*\*

\* Sets the file path and checks to make sure that the path specified points to a file.

\* @param filePath String that contains the file path to be set.

\*/

public void setFilePath(String filePath){

this.filePath = filePath;

isFilePath = new File(filePath).isFile();

}

/\*\*

\* Returns the file path.

\* @return String representing the file path.

\*/

public String getFilePath(){

return filePath;

}

/\*\*

\* Returns if the current file path is valid.

\* @return true - if file path is valid; false - otherwise

\*/

public boolean getIsFilePath(){

return isFilePath;

}

}

GetFile:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom01;

import java.awt.Color;

import java.awt.event.WindowEvent;

import java.io.File;

import javax.swing.JFileChooser;

import javax.swing.UIManager;

/\*\*

\* Simple GUI for selecting a file.

\* @author Jacob Huesman

\*/

public class GetFile extends javax.swing.JFrame {

/\*\*

\* Declare instance variables.

\*/

private final JFileChooser fc;

private FilePath file;

private Thread thread;

/\*\*

\* Creates new JFrame GetFile.

\*/

public GetFile(FilePath filePath, Thread thread) {

/\* Set the System look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(GetFile.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(GetFile.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(GetFile.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(GetFile.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

initComponents();

/\*\*

\* Instantiate new JFileChooser. Add references of filePath and thread passed from the client.

\*/

fc = new JFileChooser();

this.file = filePath;

this.thread = thread;

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

filePath = new javax.swing.JTextField();

findFile = new javax.swing.JButton();

returnFile = new javax.swing.JButton();

label = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.DO\_NOTHING\_ON\_CLOSE);

addWindowListener(new java.awt.event.WindowAdapter() {

public void windowClosing(java.awt.event.WindowEvent evt) {

formWindowClosing(evt);

}

});

filePath.setToolTipText("");

findFile.setText("File");

findFile.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

findFileActionPerformed(evt);

}

});

returnFile.setText("Import");

returnFile.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

returnFileActionPerformed(evt);

}

});

label.setText("Enter file path or choose a file to import data from.");

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addComponent(filePath)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(findFile, javax.swing.GroupLayout.PREFERRED\_SIZE, 65, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addComponent(returnFile, javax.swing.GroupLayout.PREFERRED\_SIZE, 65, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addGroup(layout.createSequentialGroup()

.addComponent(label, javax.swing.GroupLayout.PREFERRED\_SIZE, 508, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addGap(0, 0, Short.MAX\_VALUE)))

.addContainerGap())

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addContainerGap()

.addComponent(label)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(filePath, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(findFile)

.addComponent(returnFile))

.addContainerGap())

);

pack();

}// </editor-fold>

/\*\*

\* Opens JFileChooser dialog and then displays and stores the path.

\* @param evt

\*/

private void findFileActionPerformed(java.awt.event.ActionEvent evt) {

if(fc.showOpenDialog(this) == JFileChooser.APPROVE\_OPTION){

file.setFilePath(fc.getSelectedFile().getAbsolutePath());

filePath.setText(file.getFilePath());

}

}

/\*\*

\* Checks to that the path is actually pointing to a file. If it is the dialog closes. Otherwise the dialog continues prompting the user for a valid file.

\* @param evt

\*/

private void returnFileActionPerformed(java.awt.event.ActionEvent evt) {

if(file.getIsFilePath()){

thread.interrupt();

this.dispose();

} else {

label.setText("Please enter a valid file path.");

label.setForeground(Color.red);

}

}

/\*\*

\* On window close notify thread to stop executing. Then dispose of this Frame's resources.

\* @param evt

\*/

private void formWindowClosing(java.awt.event.WindowEvent evt) {

thread.interrupt();

this.dispose();

}

// Variables declaration - do not modify

private javax.swing.JTextField filePath;

private javax.swing.JButton findFile;

private javax.swing.JLabel label;

private javax.swing.JButton returnFile;

// End of variables declaration

}

IllegalDataTypeException:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom01;

/\*\*

\* Exception to handle bad data types being passed as a parameter.

\* @author Jacob Huesman

\*/

public class IllegalDataTypeException extends IllegalArgumentException {

/\*\*

\* Exception to be thrown

\* @param message String containing details regarding the exception.

\*/

public IllegalDataTypeException ( String message ){

super(message);

}

}

SortTokenCheck:

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package hom01;

import java.util.ArrayList;

/\*\*

\* Static class that takes an ArrayList of TokenCheck objects and sorts it.

\* @author Jacob Huesman

\*/

public class SortTokenCheck {

//Static class doesn't need a constructor.

private SortTokenCheck(){}

/\*\*

\* Sorts an ArrayList of TokenCheck objects alphabetically by the token type and further sorts them by the String.

\* @param array ArrayList of TokenCheck objects to be sorted.

\*/

public static void sort(ArrayList<TokenCheck> array){

//No point in sorting an array of one object or less.

if(!(array.size()>1)){

return;

}

//Insertion Sort Token Type aphabetically.

for(int i=1; i<array.size(); i++){

TokenCheck cur = array.get(i);

for(int a=i-1; a>=0; a--){

if(cur.getType().compareTo(array.get(a).getType()) < 0){

array.set(a+1, array.get(a));

array.set(a, cur);

} else {

break;

}

}

}

//Insertion Sort String alphabetically while retaining order of token type.

for(int i=1; i<array.size(); i++){

TokenCheck cur = array.get(i);

for(int a=i-1; a>=0; a--){

if(cur.getType().compareTo(array.get(a).getType()) == 0){

if(cur.getData().compareTo(array.get(a).getData()) < 0){

array.set(a+1, array.get(a));

array.set(a, cur);

}

} else {

break;

}

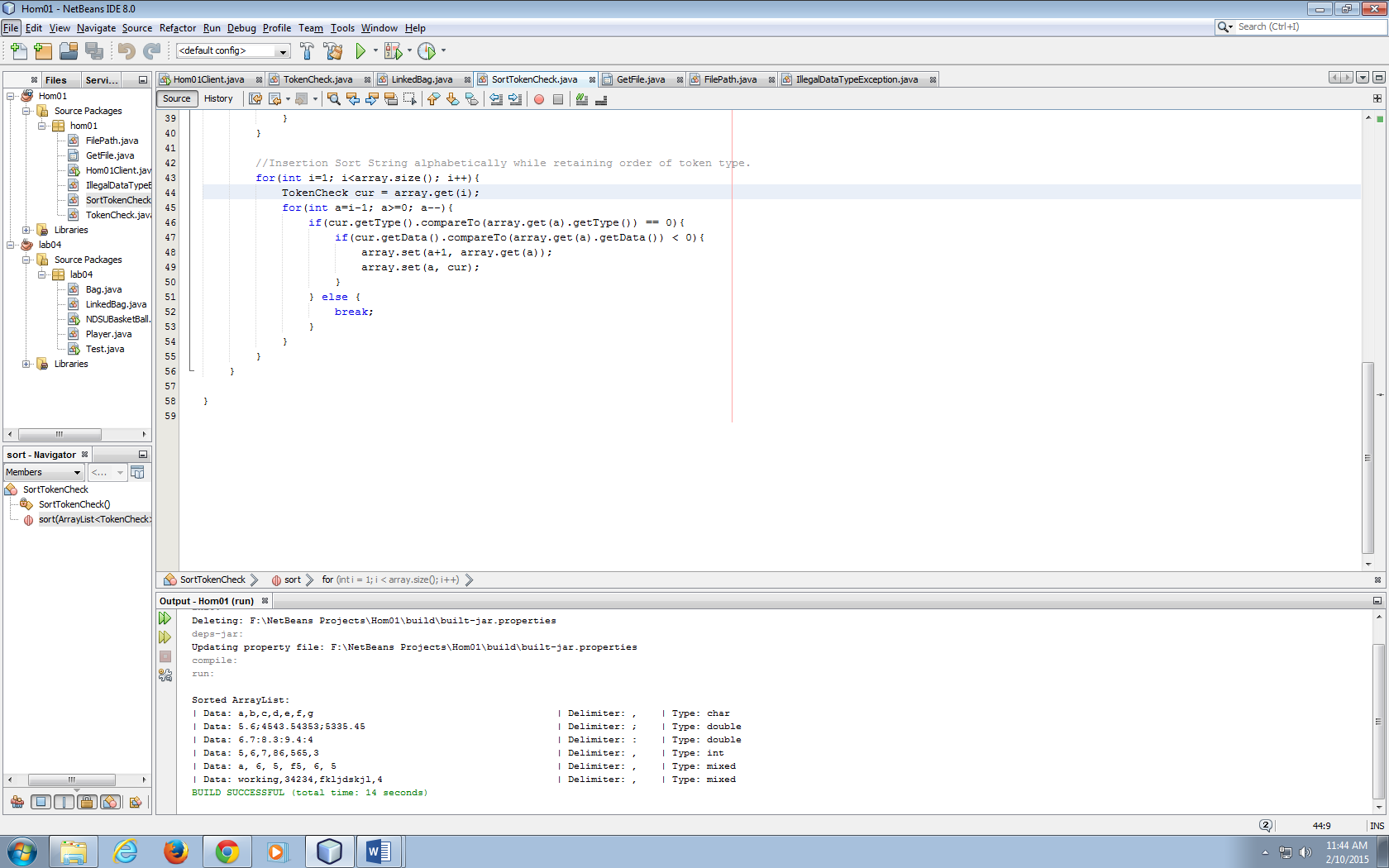
}

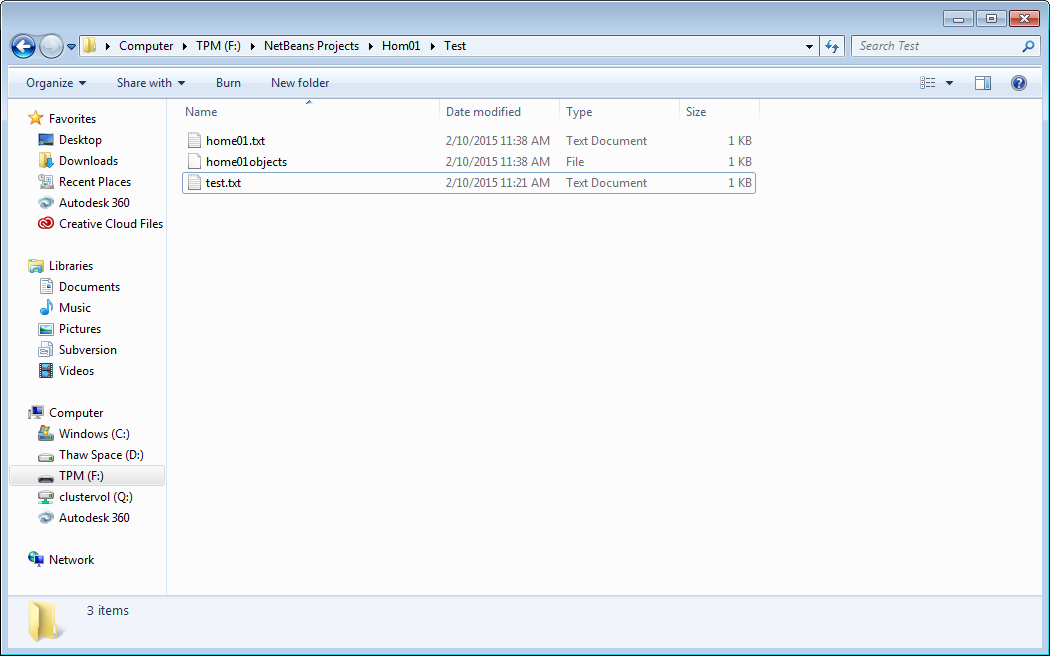
}

}

}

Output:





UML Diagram:

