/\*\*

\* Hangman.java

\*

\* Assignment 2 for COMP285.

\*

\* Play the game of Hangman.

\*

\* Show your Surname, Name and Student ID.

\*

\* Andrey Terakopov ID: 201434719

\* sgaterak@liverpool.ac.uk

\*

\* Some methods of this Java file are either wrongly implemented or not implemented.

\* A correct implementation is required to pass all test methods in the test case

\* {@link HangmanTest}.

\* So, currently the play does not function as required until fully implemented and tested.

\*

\*/

package hangman;

import java.util.Enumeration;

import java.util.Random;

import java.util.Scanner;

import java.util.Vector;

/\*\*

\*

\* Play the game of Hangman. The computer chooses a word (the

\* "word-to-be-guessed"), and displays a star sign \* for each letter in

\* the word (the \*-form of the word).

\*

\* The user guesses a letter: if that letter occurs in the word, then

\* every occurrence of that latter in the word is shown (i.e., the

\* letter is written instead of the \*).

\*

\* If the letter does not occur in the word or is chosen repeatedly, the

\* user loses a life. Play continues in this way until either

\*

\* - all the letters in the word-to-be-guessed have been guessed; the

\* user has won, or

\*

\* - the user has lost all their lives (10); the computer has won.

\*

\* After each round, the user has the option of quitting or playing

\* another round.

\*

\* If quitting, the number of rounds played is shown and the round

\* numbers are enumerated in which the user has won.

\*

\*/

public class Hangman

{

/\*\*

\* Boolean test (default value false) is used in {@link #input\_next()}

\* method to choose between two alternative versions of inputting a string:

\*

\* - either from the keyboard,

\*

\* - or from a given {@link #INPUT} (which is a Vector<String>).

\*

\* Take {@link test} = true to use {@link #INPUT}; useful in the case of

\* testing.

\*

\* Equivalently, add {@link test} = true in setUp method of the test case.

\* In this case INPUT should be somehow initialised there.

\*/

private boolean test = false; // default value false.

/\*\*

\* The ordinary user input from the keyboard.

\*

\* We use input.next() for getting the next input from the keyboard.

\*/

private Scanner input = new Scanner(System.in);

/\*\*

\* Vector<String> INPUT to accumulate all inputs by the user in a round.

\* Used in {@link #input\_next()}.

\*

\* Useful to aid testing and to be initialised in the test case.

\*/

private Vector<String> INPUT = new Vector<String>();

/\*\*

\* INPUT\_ENUMERATION is the Vector<String> {@link #INPUT} treated as

\* {@link java.util.Enumeration} to allow using the methods

\* hasMoreElements() and nextElement().

\*/

private Enumeration<String> INPUT\_ENUMERATION = getINPUT().elements();

/\*\*

\* The list of words to choose from. This list is used in

\* {@link #initialiseRound()}.

\*/

private String[] WORDS = new String[]

{ "syzygy", "erythropoeia", "quicksilver" };

/\*\*

\* Random number generator used in {@link #chooseWord(String[] words)} for

\* choosing the word-to-be-guessed from the available array of

\* {@link #WORDS}.

\*

\*/

private final Random rng = new Random();

/\*\*

\* String to hold the current word to be guessed; will be chosen by

\* {@link #chooseWord(String[] words)} which uses the random number

\* generator {@link #rng}.

\*/

private String word;

/\*\*

\* Current word with each letter L shown either as "\* " or as "L "; will be

\* obtained by {@link #starForm(word) starForm(word)}. During the game

\* letters will be guessed and substituted for corresponding stars.

\*/

private String word\_form;

/\*\*

\* The number of lives (for unluckily or repeatedly chosen letters) given to

\* the user.

\*/

private static final int NUM\_LIVES = 10;

/\*\*

\* The number of rounds in a play until user quits; initially 0.

\*/

private int numRounds = 0;

/\*\*

\* The play round numbers where user has won.

\*/

private Vector<Integer> PlayResults = new Vector<Integer>();

/\*\*

\* The letters that the user has unluckily or repeatedly guessed in a round.

\*

\* (Each such an unlucky guess costs a life. Typing mistakenly a string of

\* several letters costs nothing and will be asked to re-type.)

\*/

private Vector<Character> unluckyGuesses = new Vector<Character>();

/\*\*

\* Play the game. Pick a word, let the user guess it, and repeat until the

\* user chooses to quit the game.

\*

\* Calls methods: {@link #initialiseRound() initialiseRound()},

\* {@link #letUserGuessWord(word, word\_form, NUM\_LIVES)

\* letUserGuessWord(word, word\_form, NUM\_LIVES)}, {@link #input\_y\_n\_Char()

\* input\_y\_n\_Char()}.

\*

\* (This is the only method that should be public; the others are

\* package-visible for testing purposes.)

\*

\*/

public void playHangman()

{

// TODO Auto-generated method stub

/\*

\* boolean to indicate if user has won the round

\*/

boolean hasUserWon;

/\*

\* boolean to indicate if user has opted to quit

\*/

boolean userNotDone = true;

/\*

\* Now, repeatedly choose a word and let the user guess it, until the

\* user opts to quit.

\*/

while (userNotDone)

{

// set up a new round (in particularly a word to guess, its

// word\_form, etc.)

initialiseRound();

// let the user guess word step-by-step by observing word\_form;

// a method call letUserGuessWord(word, word\_form, NUM\_LIVES) should

// be used here and appropriately implemented in this class.

System.out.println("Guess the word: ");

hasUserWon = letUserGuessWord(getWord(), getWord\_form(), NUM\_LIVES);

// print result of game depending on the value of hasUserWon

System.out.println(hasUserWon ? getWord() + "\nWell done!"

: "I win, ha ha ha.\nThe word was " + getWord());

// play again?

System.out.println("Do you want to play again? (y/n)");

// amend numRounds and PlayResults

setNumRounds(getNumRounds() + 1);

if (hasUserWon)

{

getPlayResults().addElement(getNumRounds());

}

// get user response (y/n) and treat it as boolean value of

// userNotDone

char c = input\_y\_n\_Char();

if (c == 'y')

{

userNotDone = true;

} else

{

userNotDone = false;

}

}

// Show (i) total number of rounds played and (ii) a list of round

// numbers in which the user has won.

System.out.println("Total number of rounds played was: " + getNumRounds());

System.out.println("Round numbers which the user has won: "

+ getPlayResults());

}

/\*\*

\* Set up a round of Hangman. Pick a {@link #word word} in {@link #WORDS

\* WORDS}; set up the {@link #word\_form word\_form}; clear Vector<Character>

\* {@link #unluckyGuesses unluckyGuesses}.

\*

\* @return the word to be guessed

\*/

public String initialiseRound()

{

// TODO Auto-generated method stub

// return null;

// pick a word for the user to guess

setWord(chooseWord(getWORDS()));

// Transform it to the \*-form \*\*...\* to be shown to the user.

setWord\_form(starForm(getWord()));

// clear the bad guesses

getUnluckyGuesses().clear();

return getWord();

}

/\*\*

\* Play a round of Hangman.

\*

\* The method could have no parameters, but they are added to aid testing:

\*

\* Calls: {@link #inputChar() inputChar()},

\* {@link #occursAsNew(userGuess, wrd, w\_form) occursAsNew(userGuess, wrd,

\* w\_form)}, {@link #amend(wrd, w\_form, userGuess) amend(wrd, w\_form,

\* userGuess)} \*

\*

\* @param wrd

\* the word to be guessed

\* @param wrd\_form

\* the visible form of the word

\* @param num\_lives

\* user lives remaining

\* @return true, if the user wins (AllGuessed); false, otherwise

\* (!AllGuessed).

\*/

public boolean letUserGuessWord(String wrd, String wrd\_form, int num\_lives)

{

// Word \*-form; will repeatedly be amended in this method

String w\_form = wrd\_form;

// value to return.

// Initially false since wrd\_form should initially consist of stars.

boolean AllGuessed = false; // occurs('\*', w\_form);

// how many unlucky guesses the user has left; originally = num\_lives.

int numLivesLeft = num\_lives;

// user's guess

char userGuess;

// was it lucky?

boolean isUserLucky;

/\*

\* repeatedly:

\*

\* 1) show (updated) word \*-form to user 2) prompt user to guess a

\* letter 3) read the letter 4) update the word \*-form, list of

\* unluckily or repeatedly guessed letters, etc.

\*

\* until either all letters are guessed, or user has no more lives left

\* (which is actually the end of a round).

\*/

while (!AllGuessed && numLivesLeft > 0)

{

// 1) show the user the current word \*-form

System.out.println(w\_form);

// 2) prompt user (show all unluckily or repeatedly guessed letters,

// if any; how many lives are left; suggest to pick a letter)

if (numLivesLeft < num\_lives) // if some lives had been lost.

// (initially false)

{

System.out

.println("Leters you have unluckily or repeatedly guessed: "

+ getUnluckyGuesses());

}

System.out.println("You have " + numLivesLeft

+ " life(s) remaining. Pick a letter.");

// 3) get user's guess (and show it)

userGuess = inputChar();

// 4) update isUserLucky, w-form, notAllGuessed, numLivesLeft and

// unluckyGuesses.

isUserLucky = occursAsNew(userGuess, wrd, w\_form);

w\_form = amend(wrd, w\_form, userGuess);

AllGuessed = !occurs('\*', w\_form);

if (!isUserLucky) // if user is unlucky

{

numLivesLeft--;

getUnluckyGuesses().addElement(userGuess);

}

}

// End the round (i.e. the method letUserGuessWord()) by showing

// how many lives the user has remaining.

System.out.println("You have " + numLivesLeft + " life(s) remaining.");

// State the Boolean verdict.

return AllGuessed; // This could be true or false

// Note that appropriate verdict messages to the user (based on this

// Boolean value) will be done by playHangman() after calling the

// current method letUserGuessWord().

}

/\*\*

\* The question "Will user play again?" expects inputs "y" or "n". The

\* method repeats requesting to input one letter y/n by {@link #inputChar()}

\* until y/n is indeed obtained from the user.

\*

\* See more comments to the test method

\* {@link HangmanTest#testInput\_y\_n\_Char()}.

\*

\* @return the input letter 'y' or 'n'.

\*/

public char input\_y\_n\_Char()

{

char c = 0;

boolean flag = true;

while (flag == true) {

System.out.print("Will user play again?");

c = inputChar();

if (c == 'y' || c == 'n') { flag = false;}

}

// TODO Auto-generated method stub

return c; // Not yet implemented.

// NEXT LINES - EXPECTED SOLUTION BY STUDENTS

// Costs 7 marks

}

/\*\*

\* Pick a random word from an array of words for the user to guess.

\*

\* Uses {@link #rng rng}.

\*

\* @param String

\* [] words

\* @return the word the user has to guess in this game

\*/

public String chooseWord(String[] words)

{

// TODO Auto-generated method stub

// return null;

return words[rng.nextInt(words.length)];

}

/\*\*

\* @return the given word w with each letter L shown as '\*'

\*

\* @param String

\*/

public String starForm(String w)

{

for (int i = 0; i < w.length(); i++)

{

char c1 = w.charAt(i);

w = w.replace(c1, '\*');

}

return w;

}

/\*\*

\* Occurrence of a character c in a word.

\*/

public boolean occurs(char c, String word)

{

boolean b = false;

for (int i = 1; i < word.length(); i++)

{

if (word.charAt(i) == c)

{

b = true;

}

}

return b;

}

/\*\*

\* Whether a character c occurs in a {@link #word word} but does not occur

\* in {@link #word\_form word\_form}. Calls {@link #occurs(c, word) occurs(c,

\* word)} and {@link #occurs(c, word\_form) occurs(c, word\_form)}.

\*/

public boolean occursAsNew(char c, String word, String word\_form)

{

String s = Character.toString(c);

if (word.contains(s) && !word\_form.contains(s)) {

return occurs(c, word);

} else {

return occurs(c, word\_form);

}

// TODO Auto-generated method stub

//return false; // Not yet implemented!

// EXPECTED SOLUTION BY STUDENTS.

// Costs 5 marks

}

/\*\*

\* Amending a {@link #word\_form word\_form} by substituting all occurrences

\* of a given character c in a {@link #word word} into {@link #word\_form

\* word\_form} in same positions.

\*/

public String amend(String word, String word\_form, char c)

{

char[] word\_form\_char\_array = word\_form.toCharArray();

for (int i = 0; i < word\_form.length(); i++)

{

if (word.charAt(i) == c)

{

word\_form\_char\_array[i] = c;

}

}

word\_form = String.valueOf(word\_form\_char\_array);

return word\_form;

}

/\*\*

\* Inputting a character/string by user.

\*

\* If the user's input {@link #input\_next() input\_next()} (in general - a

\* string) is of length > 1, it is requested to input just one letter until

\* the user really does that.

\*

\* @return the first character so obtained. It should be a character, not a

\* one letter string!

\*/

public char inputChar()

{

char c = 0;

boolean flag = true;

while (flag == true) {

String s = input\_next();

if (s.length() > 1) {System.out.print("Please enter only single letter");}

else {c = s.charAt(0); flag = false;}

}

// TODO Auto-generated method stub

return c; // Not yet implemented!

// EXPECTED SOLUTION BY STUDENTS.

// Costs 10 marks

}

/\*\*

\* In the case of {@link test} == false, {@link #input\_next()} has the same

\* effect as the usual {@link input#next()} reading an input string from the

\* console, plus showing the input on the console.

\*

\* In the case of {@link test} == true, it behaves as a "mock" of

\* {@link input#next()} actually taking inputs from the Vector

\* {@link #INPUT INPUT} and can be used for testing purposes. (Actually, it

\* is more convenient to use {@link java.util.Enumeration}

\* {@link #INPUT\_ENUMERATION INPUT\_ENUMERATION} obtained from Vector

\* {@link #INPUT INPUT}). In this case, when {@link #INPUT INPUT} (or

\* {@link #INPUT\_ENUMERATION INPUT\_ENUMERATION}) is finished, the default

\* input "n" is assumed. This will eventually halt the game since "n" is the

\* answer to question whether the user opts to quit. Again, after getting a

\* next input it should be shown on the console.

\*

\* @return the next input string

\*/

public String input\_next()

{

if (!isTest())

{

String YOUR\_INPUT = input.next();

System.out.println("Your input: " + YOUR\_INPUT);

return YOUR\_INPUT;

} else

{

/\*\*

\* Vector INPUT considered as Enumeration to allow using

\* hasMoreElements() and nextElement().

\*

\* Enumeration<String> INPUT\_ENUMERATION = INPUT.elements(); SEE

\* ABOVE

\*/

if (getINPUT\_ENUMERATION().hasMoreElements())

{

String YOUR\_INPUT = getINPUT\_ENUMERATION().nextElement();

System.out.println("Your input: " + YOUR\_INPUT);

return YOUR\_INPUT;

} else

{

System.out.println("Your input: " + "n");

return "n"; // "n" is used here to eventually answer negatively

// the question "Do you want to play again?"

// and so quit the game avoiding infinite loop.

}

}

}

/\*\*

\* The main method which consists just in calling {@link #playHangman()

\* playHangman()}

\*

\* @param args

\* not used

\*/

public static void main(String[] args)

{

Hangman game=new Hangman();

game.playHangman();

}

public Vector<String> getINPUT() {

return INPUT;

}

public void setINPUT(Vector<String> iNPUT) {

INPUT = iNPUT;

}

public boolean isTest() {

return test;

}

public void setTest(boolean test) {

this.test = test;

}

public String[] getWORDS() {

return WORDS;

}

public void setWORDS(String[] wORDS) {

WORDS = wORDS;

}

public Enumeration<String> getINPUT\_ENUMERATION() {

return INPUT\_ENUMERATION;

}

public void setINPUT\_ENUMERATION(Enumeration<String> iNPUT\_ENUMERATION) {

INPUT\_ENUMERATION = iNPUT\_ENUMERATION;

}

public int getNumRounds() {

return numRounds;

}

public void setNumRounds(int numRounds) {

this.numRounds = numRounds;

}

public Vector<Integer> getPlayResults() {

return PlayResults;

}

public void setPlayResults(Vector<Integer> playResults) {

PlayResults = playResults;

}

public Vector<Character> getUnluckyGuesses() {

return unluckyGuesses;

}

public void setUnluckyGuesses(Vector<Character> unluckyGuesses) {

this.unluckyGuesses = unluckyGuesses;

}

public String getWord() {

return word;

}

public void setWord(String word) {

this.word = word;

}

public String getWord\_form() {

return word\_form;

}

public void setWord\_form(String word\_form) {

this.word\_form = word\_form;

}

}