Patel200921323MiniL7

December 8, 2020

1 Mini-project

2 Level 7

2.1 Jatinkumar Patel

2.2 3/12/2020

2.3 Version 4

2.4 Summary of the Question

Create a help bot that can help a user by answering text questions about a specific subject. In this case, the bot provides information about the game Minecraft, and can help teach the user different things about the game.

2.5 The literate program development

2.5.1 inputString

What it does *inputString is a method which allows us to get input from any method without having to initialise a scanner for each method we have in the program.*

Implementation (how it works) *inputString works like a normal input sequence; a scanner is initialised, the user is prompted to answer a question and their answer is saved to a variable and returned to the program.*

[1]:
```
public static String inputString(String message) { // inputString method to␣
↪allow for easier input scanning.
    Scanner scanner = new Scanner(System.in);
    String textinput;

    System.out.println(message);
    textinput = scanner.nextLine();
    return textinput;
}
```

Testing

[2]:
```
inputString("What is your name? ");
```

```
What is your name?
Jatin
```

[2]:
```
Jatin
```

2.5.2 starterChoice

What it does *This method gives the user a choice to load old answers into the program. If they don't want to, then it will proceed to prompt them to ask a question as normal.*

Implementation (how it works) *The type trigsAndResps is taken as an argument so that we can save an array to the record if we need to. An IOException thrower is enabled to allow for file reading and to deal with any errors that occur in the reading. A string variable (choice) is used to prompt the user to enter if they want to load old answers or not. We then check choice, and if it equals yes then we can proceed to load a file using readFromFile.*

[28]:
```
// Giving the user a choice to load answers in or not
public static void starterChoice(trigsAndResps trigsAndResps) throws␣
→IOException {
    String choice = inputString("Would you like to load answers? (please input␣
→yes or no) ");
    if (choice.equals("Yes") || choice.equals("yes")) {
        readFromFile("responses.txt", trigsAndResps);
    }
}
```

Testing

[55]:
```
starterChoice(tri);
```

```
Would you like to load answers? (please input yes or no)
no
```

2.5.3 writeToFile

What it does *This method saves the sorted answers from the current session when it ends.*

Implementation (how it works) *The type trigsAndResps is taken as an argument so that we know which data to save to a file. An IOException thrower is enabled to allow for file reading and to deal with any errors that occur in the writing. We also use a PrintWriter to write to the file, and when creating this printWriter we set the append argument to false. This will overwrite old information in the file that is no longer needed. A string array is made to equal the responses stored in our record. The program then loops through that array and writes each element to a new line in the file. We make sure to close the PrintWriter so that no errors are caused.*

[27]:
```
// Always saves the most recent sorted answers into a file
public static void writeToFile(trigsAndResps trigsResps) throws IOException{
    PrintWriter outputStream = new PrintWriter(new FileWriter("responses.txt",␣
↪false));
    String[] responses = trigsResps.responses;

    for (int i = 0; i < responses.length; i++) { // Storing the responses
        outputStream.println(responses[i]);
    }
    outputStream.close();
}
```

Testing

[54]:
```
writeToFile(tri);
```

2.5.4 readFromFile

What it does *This method reads answers stored from a text file and outputs them to the user.*

Implementation (how it works) *The type trigsAndResps is taken as an argument alongside a string filename, which tells us what file we are going to read data from. An IOException thrower is enabled to allow for file reading and to deal with any errors that occur in the reading. We also use a BufferedReader wrapped around a FileReader to improve the efficiency of reading from the file.*

A new string array entries is declared and initialised to hold the information that we read from the file. A string variable is also declared to hold the current piece of data being read. An integer position is initialised to 0 to control how many lines are read from the file.

We create a while loop that is set to terminate when position == 6. Until then, we read a line from the file, store it in current and save current to the next field in the array entries. We also output the line we have read to the user. Position is incremented by 1 to read the next line of the file. After the loop is complete, the responses array in our record is set to equal entries.

[26]:
```
// Will load old answers into an array
public static void readFromFile(String fileName, trigsAndResps trigsAndResps)␣
↪throws IOException {
    BufferedReader inputStream = new BufferedReader(new FileReader(fileName));
    String[] entries = new String[6];
    String current;
    int position = 0;
    while(position != 6) {
        current = inputStream.readLine();
        entries[position] = current;
        System.out.println(current);
        position++;
    }
    trigsAndResps.responses = entries;
}
```

Testing

[53]:
```
readFromFile("responses.txt", tri);
```

```
chop
mine
slash
block
snipe
protect
```

2.5.5 helpful

What it does *This method takes feedback from the user about the usefulness of the answer that the program just gave to the user's question.*

Implementation (how it works) *The user's input, the type questionAnswerDatabase and the position of the result that was given to the user are all taken as arguments to this method. These will be used later when we are logging usefulness and checking for more answers.*

We first prompt the user to tell us if the answer was helpful, and store their reply in the string helpful. We then check helpful to see if it equals yes or no. If it equals yes, then we output a message, take the position of the result we took as an argument, and add 1 to the usefulness array in our database at that position.

If helpful equals no, we apologise and check a different array for possible answers using the check-Answers method. For any other input, we output that nothing was found, and the method ends.

[25]:
```
public static void helpful(String input, questionAnswerDatabase qAndA, int␣
↪position) { // Method to ask if the answer was helpful
    String helpful = inputString("Did the answer help? Please enter yes or no:␣
↪");

    if (helpful.equals("yes") || helpful.equals("Yes")) {
        System.out.println("Great news!");
        setUsePosition(qAndA, position, 1);
    } else if (helpful.equals("no") || helpful.equals("No")) {
        System.out.println("Sorry for that! Let me try again.");
        checkAnswers(input, qAndA);
    } else {
        System.out.println("Nothing found. Sorry.");
    }
}
```

Testing

[52]:
```
helpful("input", qADB, 3);
```

```
Did the answer help? Please enter yes or no:
yes
```

```
Great news!
```

2.5.6 checkAnswers

What it does *This method will be called when we cannot find answers in our first database. It will go through another database to find possible answers that might help the user.*

Implementation (how it works) *The user's input and the type questionAnswerDatabase are taken as arguments to this method. We create a boolean variable found, and loop through the answers field of the record we took as an argument. If we find an answer, we print it to the user and set found to true, otherwise nothing happens. We then check if found is true or not. If found = false, we tell the user that nothing was found.*

[24]:
```
public static void checkAnswers(String input, questionAnswerDatabase qAndA) { //
↪ Going through potential answers
    boolean found = false;

    for (int i = 0; i < qAndA.questions.length; i++) {
        if (input.contains(qAndA.questions[i])) {
            System.out.println(qAndA.answers[i]);
            found = true;
        }
    }

    if (!found) {
        System.out.println("Nothing found. Sorry.");
    }
}
```

Testing

[51]:
```
checkAnswers("input", qADB);
```

```
Nothing found. Sorry.
```

2.5.7 bubbleSortUse

What it does *This method is used at the end of the program to sort out the answers in terms of usefulness ratings that the user gave. After bubble sorting, they are outputted to the user in order.*

Implementation (how it works) *In this method we take an integer array of usefulness ratings and a string array of answers. We first set a boolean sorted to false, which indicates if the array is sorted or not. A while loop is initialised, inside which we first set sorted to true. Inside the while loop, we create a for loop that loops through the usefulness array.*

We will be sorting both arrays that we took as arguments simultaenously. This will make sure that the usefulness rating in one array matches up to the answer in the other array.

Inside the for loop, we check if the usefulness rating of the next element is more than the rating of the current element of the array. If it is, then we perform a synchronised swap, where elements *in both arrays are swapped at the same time to make sure they still match up to each other after sorting. After the swap is complete we set sorted to false since the arrays were not sorted. The loops will execute until both arrays are sorted.*

[23]:
```java
public static void bubbleSortUse (int[] usefulness, String[] answers) { //␣
 ↪Sorts answers by usefulness
    boolean sorted = false;

    while (!sorted) {
        sorted = true;
        for (int i = 0; i < usefulness.length - 1; i++) { // Go through the␣
 ↪array
            if (usefulness[i] < usefulness[i+1]) { // Swap synchronised
                int temp = usefulness[i + 1];
                usefulness[i + 1] = usefulness[i];
                usefulness[i] = temp;

                String tempStr = answers[i + 1];
                answers[i + 1] = answers[i];
                answers[i] = tempStr;
                sorted = false;
            }
        }
    }
    System.out.println("Most useful answers: ");
    for (int i = 0; i < answers.length; i++) {
        System.out.println((i + 1) + ". " + answers[i]);
    }
}
```

Testing

[50]:
```java
bubbleSortUse(usefulness, answers);
```

```
Most useful answers:
1. axe
2. ore
3. block
4. pickaxe
5. sword
6. armor
```

2.5.8 Record - questionAnswerDatabase

What it does *This is a record with three fields. There are two String arrays to store question triggers and answers respectively. An integer array is also used to store ratings of how helpful answers were from the user.*

Implementation (how it works) *A new type is created questionAnswerDatabase. Inside the definition, we declare two string arrays and one integer array.*

[3]:
```
class questionAnswerDatabase { // Extra database to check if the first answers␣
↪don't help
    String[] questions;
    String[] answers;
    int[] usefulness;
}
```

Testing

[30]:
```
questionAnswerDatabase qADB = new questionAnswerDatabase();
```

2.5.9 setQues

What it does *Accessor method to set question triggers of a certain question-answer data type.*

Implementation (how it works) *The string array given in arguments is used to set the value of the responses field in the record.*

[5]:
```
// Accessor method to set questions
public static void setQues(questionAnswerDatabase questionAnswerDatabase,␣
↪String[] questions) {
    questionAnswerDatabase.questions = questions;
}
```

Testing

[49]:
```
setQues(qADB, questions);
```

2.5.10 setAns

What it does *Accessor method to set the answers of a certain question-answer data type.*

Implementation (how it works) *The string array given in arguments is used to set the value of the answers field in the record.*

[6]:
```
//Accessor method to set answers
public static void setAns(questionAnswerDatabase questionAnswerDatabase,␣
↪String[] answers) {
    questionAnswerDatabase.answers = answers;
}
```

Testing

[48]:
```
setAns(qADB, answers);
```

2.5.11 setUse

What it does *Accessor method to set the usefulness ratings of a certain question-answer data type.*

Implementation (how it works) *The integer array given in arguments is used to set the value of the usefulness field in the record.*

[7]:
```
// Accessor method to set usefulness ratings
public static void setUse(questionAnswerDatabase questionAnswerDatabase, int[]␣
↪usefulness) {
    questionAnswerDatabase.usefulness = usefulness;
}
```

Testing

[47]:
```
setUse(qADB, usefulness);
```

2.5.12 setUsePosition

What it does *Accessor method to set the usefulness rating of a certain question-answer data type.*

Implementation (how it works) *The integer position is used to increment an element of the questionAnswerDatabase.usefulness array by the integer newUse, which is the new usefulness rating.*

[8]:
```
// Accessor method to set usefulness ratings
public static void setUsePosition(questionAnswerDatabase␣
↪questionAnswerDatabase, int position, int newUse) {
    questionAnswerDatabase.usefulness[position] += newUse;
}
```

Testing

[46]:
```
setUsePosition(qADB, 3, 1);
```

2.5.13 getQuestions

What it does *Accessor method to get the question triggers of a certain question-answer data type.*

Implementation (how it works) *The question triggers of a certain question-answer data type are returned to the program.*

[9]:
```
//Accessor method to get question triggers
public static String[] getQuestions(questionAnswerDatabase␣
↪questionAnswerDatabase) {
    return questionAnswerDatabase.questions;
}
```

Testing

[45]:
```
getQuestions(qADB);
```

[45]:
```
[Ljava.lang.String;@3b6c2fb5
```

2.5.14 getAnswers

What it does *Accessor method to get the answers of a certain question-answer data type.*

Implementation (how it works) *The answers of a certain question-answer data type are returned to the program.*

[10]:
```
// Accessor method to get answers.
public static String[] getAnswers(questionAnswerDatabase␣
↪questionAnswerDatabase) {
    return questionAnswerDatabase.answers;
}
```

Testing

[44]:
```
getAnswers(qADB);
```

[44]:
```
[Ljava.lang.String;@5d05e44
```

2.5.15 getUse

What it does *Accessor method to get the usefulness ratings of a certain question-answer data type.*

Implementation (how it works) *The usefulness ratings of a certain question-answer data type are returned to the program.*

[11]:
```
// Accessor method to get usefulness.
public static int[] getUse(questionAnswerDatabase questionAnswerDatabase) {
    return questionAnswerDatabase.usefulness;
}
```

Testing

[43]:
```
getUse(qADB);
```

[43]:
```
[I@63be77b6
```

2.5.16 questions

What it does *Method to initialise the question triggers of the question-answer data type.*

Implementation (how it works) *Returns a String array that contains question triggers to be used when the user asks questions.*

[12]:
```
// Creating an array to scan when a second check is done for answers.
public static String[] questions() {
    return new String[] {"netherite", "diamond", "gold", "iron", "stone",␣
↪"wood"};
}
```

Testing

[31]:
```
String[] questions = questions();
```

2.5.17 answers

What it does *Method to initialise the answers of the question-answer data type.*

Implementation (how it works) *Returns a String array that contains answers to be used when the user asks questions.*

[13]:
```
// Creating an array to scan when a question word is recognised in userinput
public static String[] answers() {
    return new String[] {"ore", "block", "pickaxe", "axe", "sword", "armor"};
}
```

Testing

[32]:
```
String[] answers = answers();
```

2.5.18 usefulness

What it does *Method to initialise the usefulness ratings of the question-answer data type.*

Implementation (how it works) *Returns an integer array of all zeros to be used when the user rates answers.*

[14]:
```
// Creating an array to store ratings of answers from the user.
public static int[] usefulness() {
    return new int[] {0, 0, 0, 0, 0, 0};
}
```

Testing

[33]:
```
int[] usefulness = usefulness();
```

2.5.19 Record - trigsAndResps

What it does *This is a record with two fields. Both are string arrays, and one is used for triggers while the other is used for responses.*

Implementation (how it works) *A new type is created trigsAndResps. Inside the definition, we create two String array fields for triggers and responses.*

[4]:
```
class trigsAndResps {
    String[] triggers;
    String[] responses;
}
```

Testing

[34]:
```
trigsAndResps tri = new trigsAndResps();
```

2.5.20 setTrig

What it does *Accessor method to set the triggers of a certain trigger-response data type.*

Implementation (how it works) *The string array given in arguments is used to set the value of the triggers field in the record.*

[15]:
```
// Accessor method to set triggers and responses.
public static void setTrig(trigsAndResps trigsAndResps, String[] triggers){
    trigsAndResps.triggers = triggers;
}
```

Testing

[35]:
```
String[] triggers = {"a", "m" ,"b", "o", "w", "s"};
setTrig(tri, triggers);
```

2.5.21 setResp

What it does *Accessor method to set the responses of a certain trigger-response data type.*

Implementation (how it works) *The string array given in arguments is used to set the value of the responses field in the record.*

[16]:
```
// Accessor method to set responses.
public static void setResp(trigsAndResps trigsAndResps, String[] responses){
    trigsAndResps.responses = responses;
}
```

Testing

[36]:
```
String[] responses = {"at", "mw" ,"bq", "or", "wa", "so"};
setResp(tri, responses);
```

2.5.22 getTrig

What it does *Accessor method to get the triggers of a certain trigger-response data type.*

Implementation (how it works) *The triggers of a certain trigger-response data type are returned to the program.*

[17]:
```
// Accessor method to get triggers.
public static String[] getTrig(trigsAndResps trigsAndResps){ // Accessor method␣
 ↪to get triggers.
    return trigsAndResps.triggers;
}
```

Testing

[37]:
```
getTrig(tri);
```

[37]:
```
[Ljava.lang.String;@2603a43f
```

2.5.23 getResp

What it does *Accessor method to get the responses of a certain trigger-response data type.*

Implementation (how it works) *The responses of a certain trigger-response data type are returned to the program.*

[18]:
```
// Accessor method to get responses.
public static String[] getResp(trigsAndResps trigsAndResps){ // Accessor method␣
 ↪to get responses
    return trigsAndResps.responses;
}
```

Testing

[38]:
```
getResp(tri);
```

[38]:
```
[Ljava.lang.String;@314115b3
```

2.5.24 triggers

What it does *Method to initialise the triggers of the trigger-response data type.*

Implementation (how it works) *returns a String array that contains triggers to be used when the user asks questions.*

[19]:
```
public static String[] triggers() { // Creating an array with trigger words to␣
↪be used in records.
    return new String[] {"pickaxe", "axe", "sword", "shield", "bow", "armor"};
}
```

Testing

[39]:
```
triggers = triggers();
```

2.5.25 responses

What it does *Method to initialise the responses of the trigger-response data type.*

Implementation (how it works) *returns a String array that contains responses to be used when the user asks questions.*

[20]:
```
public static String[] responses() { // Creating an array with response words␣
↪to be used in records.
    return new String[] {"mine", "chop", "slash", "block", "snipe", "protect"};
}
```

Testing

[40]:
```
responses = responses();
```

2.5.26 createTrigResp

What it does *A method used to initialise a record of type trigsAndResps with given field arguments.*

Implementation (how it works) *We take 3 arguments, the name of the trigsAndResps type, and the two String arrays for triggers and responses respectively. We then set the fields in the record to equal the arrays given in the accessor method call.*

[21]:
```
// Method to initialise the record with given String arrays.
public static void createTrigResp(trigsAndResps trigsAndResps, String[]␣
↪triggers, String[] responses){
    setTrig(trigsAndResps, triggers);
    setResp(trigsAndResps, responses);
}
```

Testing

[41]:
```
createTrigResp(tri, triggers, responses);
```

2.5.27 createQuesAns

What it does *A method used to initialise a record of type questionAnswerDatabase with given field arguments.*

Implementation (how it works) *We take 4 arguments, the name of the questionAnswerDatabase type, two String arrays for triggers and responses respectively, and an integer array for usefulness. We then set the fields in the record to equal the arrays given in the accessor method call.*

[22]:
```
// Method to initialise questions and answers to recognise
public static void createQuesAns (questionAnswerDatabase␣
↪questionAnswerDatabase, String[] questions, String[] answers, int[]␣
↪usefulness) {
    setQues(questionAnswerDatabase, questions);
    setAns(questionAnswerDatabase, answers);
    setUse(questionAnswerDatabase, usefulness);
}
```

Testing

[42]:
```
createQuesAns(qADB, questions, answers, usefulness);
```

2.5.28 main

What it does *Calls the most important processes, initialises records, and makes sure that the program exits properly when it is meant to end.*

Implementation (how it works) *An IOException thrower is enabled to make sure that file input and output methods can throw errors if there are any. We use our previous methods triggers, responses, questions, answers and usefulness to initialise respective arrays that will be used when creating records. We also call the methods to initialise the records using the arrays for arguments.*

The starterChoice method is called to give the user a choice to load answers from a file or start fresh. After this method is executed, the main question loop method is called. When the user enters the message to exit, the bubbleSortUse method is called to sort ratings and answers.

After sorting, the writeToFile method is called to save the answers to a file. The user is then told that the answers have been saved from this session and a goodbye message is printed. The system exits to make sure no processes run forever.

Note: I will not run the method I have pasted below as it will not work properly in JupyterHub and will throw errors. Instead please see "Running the program" which contains and runs the contents of the main method.

[ ]:
```
public static void main(String[] args) throws IOException {
        String[] triggers = triggers();
        String[] responses = responses();
        trigsAndResps trigsResps = new trigsAndResps();
        createTrigResp(trigsResps, triggers, responses); // Creating a new␣
↪trigger/response record.

        String[] questions = questions();
        String[] answers = answers();
        int[] usefulness = usefulness();
        questionAnswerDatabase qAndADatabase = new questionAnswerDatabase();
        createQuesAns(qAndADatabase, questions, answers, usefulness); //␣
↪Creating a new question/answer database

        starterChoice(trigsResps);
        questionLoop(trigsResps, qAndADatabase);
        bubbleSortUse(qAndADatabase.usefulness, trigsResps.responses);
        writeToFile(trigsResps);
        System.out.println("Answers saved.");
        System.out.println("See you next time!");

        System.exit(0); // Exits the program to make sure it doesn't run forever
    }
```

2.5.29 questionLoop

What it does *The question loop of the help bot. This will prompt the user to ask questions repetedly until the user says goodbye.*

Implementation (how it works) *Outputs a statement which prompts the user to ask a question. Creates a while loop which runs while the user's input is not "goodbye". If the user gives an empty input, the program will prompt them to ask a question again. If the input is not empty, then the system will go through the stored triggers and see if any of them match the question from the user. If they do, then an appropriate response will be outputted to the user that should answer their question. The user then once again gets prompted to ask questions.*

[57]:
```
public static void questionLoop(trigsAndResps trigsResps,␣
↪questionAnswerDatabase qAndA){ // The main loop that will prompt the user to␣
↪ask questions.
    String userInput = inputString("Ask a question about minecraft: ");
    int position = 0;

    while(!userInput.equals("Goodbye") && !userInput.equals("goodbye")) {
        if (userInput.isEmpty()) {
            System.out.println("I can't answer that. Please try again.");
        }

        for (int i = 0; i < trigsResps.triggers.length; i++) { // Checking if␣
↪the program recognises any triggers in the input
            if (userInput.equals(trigsResps.triggers[i])) {
                System.out.println(trigsResps.responses[i]);
                position = i;
                break;
            }
        }
        helpful(userInput, qAndA, position);
        userInput = inputString("Ask a question about minecraft: ");
    }
}
```

Testing

[58]:
```
questionLoop(tri, qADB);
```

```
Ask a question about minecraft:
pickaxe
chop
Did the answer help? Please enter yes or no:
yes
Great news!
Ask a question about minecraft:
axe
mine
Did the answer help? Please enter yes or no:
yes
Great news!
Ask a question about minecraft:
shield
block
Did the answer help? Please enter yes or no:
no
Sorry for that! Let me try again.
Nothing found. Sorry.
Ask a question about minecraft:
Goodbye
```

2.5.30 Running the program

Run the following call to simulate running the complete program.

[59]:
```
String[] triggers = triggers();
String[] responses = responses();
trigsAndResps trigsResps = new trigsAndResps();
createTrigResp(trigsResps, triggers, responses); // Creating a new trigger/
 ↪response record.

String[] questions = questions();
String[] answers = answers();
int[] usefulness = usefulness();
```

```java
questionAnswerDatabase qAndADatabase = new questionAnswerDatabase();
createQuesAns(qAndADatabase, questions, answers, usefulness); // Creating a new␣
 ↪question/answer database

starterChoice(trigsResps);
questionLoop(trigsResps, qAndADatabase);
bubbleSortUse(qAndADatabase.usefulness, trigsResps.responses);
writeToFile(trigsResps);
System.out.println("Answers saved.");
System.out.println("See you next time!");
```

```
Would you like to load answers? (please input yes or no)
no
Ask a question about minecraft:
pickaxe
mine
Did the answer help? Please enter yes or no:
yes
Great news!
Ask a question about minecraft:
axe
chop
Did the answer help? Please enter yes or no:
yes
Great news!
Ask a question about minecraft:
pickaxe
mine
Did the answer help? Please enter yes or no:
yes
Great news!
Ask a question about minecraft:
bow
snipe
Did the answer help? Please enter yes or no:
no
Sorry for that! Let me try again.
Nothing found. Sorry.
Ask a question about minecraft:
axe
chop
Did the answer help? Please enter yes or no:
yes
Great news!
Ask a question about minecraft:
shield
block
```

```
Did the answer help? Please enter yes or no:
yes
Great news!
Ask a question about minecraft:
Goodbye
Most useful answers:
1. mine
2. chop
3. block
4. slash
5. snipe
6. protect
Answers saved.
See you next time!
```

2.6 The complete program

This version will only compile here. To run it copy it into a file called initials.java on your local computer and compile and run it there.

[ ]:
```java
/* Jatinkumar Patel
 * 14/11/2020
 * Version 1
 * Minecraft help bot to teach a user how to play minecraft
 */
import java.io.*; // Needed for IOExceptions, and FileIO.
import java.util.Scanner; // Needed to make scanner available.

public class MCHelpBot {
    public static void main(String[] args) throws IOException {
        String[] triggers = triggers();
        String[] responses = responses();
        trigsAndResps trigsResps = new trigsAndResps();
        createTrigResp(trigsResps, triggers, responses); // Creating a new␣
 ↪trigger/response record.

        String[] questions = questions();
        String[] answers = answers();
        int[] usefulness = usefulness();
        questionAnswerDatabase qAndADatabase = new questionAnswerDatabase();
        createQuesAns(qAndADatabase, questions, answers, usefulness); //␣
 ↪Creating a new question/answer database

        starterChoice(trigsResps);
        questionLoop(trigsResps, qAndADatabase);
        bubbleSortUse(qAndADatabase.usefulness, trigsResps.responses);
        writeToFile(trigsResps);
        System.out.println("Answers saved.");
        System.out.println("See you next time!");

        System.exit(0); // Exits the program to make sure it doesn't run forever
    }

    public static String inputString(String message){ // Input method to allow␣
↪to scan for user input.
        Scanner scanner = new Scanner(System.in);
        String textInput;

        System.out.println(message);
        textInput = scanner.nextLine();

        return textInput;
    }

    // Giving the user a choice to load answers in or not
    public static void starterChoice(trigsAndResps trigsAndResps) throws␣
↪IOException {
        String choice = inputString("Would you like to load answers? (please␣
↪input yes or no) ");
        if (choice.equals("Yes") || choice.equals("yes")) {
            readFromFile("responses.txt", trigsAndResps);
        }
    }

    // Always saves the most recent sorted answers into a file
    public static void writeToFile(trigsAndResps trigsResps) throws IOException{
        PrintWriter outputStream = new PrintWriter(new FileWriter("responses.
↪txt", false));
        String[] responses = trigsResps.responses;

        for (int i = 0; i < responses.length; i++) { // Storing the responses
            outputStream.println(responses[i]);
        }
        outputStream.close();
    }

    // Will load old answers into an array
    public static void readFromFile(String fileName, trigsAndResps␣
↪trigsAndResps) throws IOException {
        BufferedReader inputStream = new BufferedReader(new␣
↪FileReader(fileName));
        String[] entries = new String[6];
        String current;
        int position = 0;
        while(position != 6) {
            current = inputStream.readLine();
            entries[position] = current;
            System.out.println(current);
            position++;
        }
        trigsAndResps.responses = entries;
    }

    public static void questionLoop(trigsAndResps trigsResps,␣
↪questionAnswerDatabase qAndA){ // The main loop that will prompt the user to␣
↪ask questions.
        String userInput = inputString("Ask a question about minecraft: ");
        int position = 0;

        while(!userInput.equals("Goodbye") && !userInput.equals("goodbye")) {
            if (userInput.isEmpty()) {
                System.out.println("I can't answer that. Please try again.");
            }

            for (int i = 0; i < trigsResps.triggers.length; i++) { // Checking␣
↪if the program recognises any triggers in the input
                if (userInput.equals(trigsResps.triggers[i])) {
                    System.out.println(trigsResps.responses[i]);
                    position = i;
                    break;
                }
            }

            helpful(userInput, qAndA, position);

            userInput = inputString("Ask a question about minecraft: ");
        }
    }

    public static void helpful(String input, questionAnswerDatabase qAndA, int␣
↪position) { // Method to ask if the answer was helpful
        String helpful = inputString("Did the answer help? Please enter yes or␣
↪no: ");

        if (helpful.equals("yes") || helpful.equals("Yes")) {
            System.out.println("Great news!");
            setUsePosition(qAndA, position, 1);
        } else if (helpful.equals("no") || helpful.equals("No")) {
            System.out.println("Sorry for that! Let me try again.");
            checkAnswers(input, qAndA);
        } else {
            System.out.println("Nothing found. Sorry.");
        }
    }

    public static void checkAnswers(String input, questionAnswerDatabase qAndA)␣
↪{ // Going through potential answers
        boolean found = false;

        for (int i = 0; i < qAndA.questions.length; i++) {
            if (input.contains(qAndA.questions[i])) {
                System.out.println(qAndA.answers[i]);
                found = true;
            }
        }

        if (!found) {
            System.out.println("Nothing found. Sorry.");
        }
    }

    public static void bubbleSortUse (int[] usefulness, String[] answers) { //␣
↪Sorts answers by usefulness
        boolean sorted = false;

        while (!sorted) {
            sorted = true;
            for (int i = 0; i < usefulness.length - 1; i++) { // Go through the␣
↪array
                if (usefulness[i] < usefulness[i+1]) { // Swap synchronised
                    int temp = usefulness[i + 1];
                    usefulness[i + 1] = usefulness[i];
                    usefulness[i] = temp;

                    String tempStr = answers[i + 1];
                    answers[i + 1] = answers[i];
                    answers[i] = tempStr;
                    sorted = false;
                }
            }
        }
        System.out.println("Most useful answers: ");
        for (int i = 0; i < answers.length; i++) {
            System.out.println((i + 1) + ". " + answers[i]);
        }
    }
    public static String[] triggers() { // Creating an array with trigger words␣
↪to be used in records.
        return new String[] {"pickaxe", "axe", "sword", "shield", "bow",␣
↪"armor"};
    }

    public static String[] responses() { // Creating an array with response␣
↪words to be used in records.
        return new String[] {"mine", "chop", "slash", "block", "snipe",␣
↪"protect"};
    }

    public static String[] questions() {
        return new String[] {"netherite", "diamond", "gold", "iron", "stone",␣
↪"wood"};
    }

    public static String[] answers() { // Creating an array to scan when a␣
↪question word is recognised in userinput
        return new String[] {"ore", "block", "pickaxe", "axe", "sword",␣
↪"armor"};
    }

    public static int[] usefulness() {
        return new int[] {0, 0, 0, 0, 0, 0};
    }

    // Accessor methods to get and set triggers and responses.
    public static void setTrig(trigsAndResps trigsAndResps, String[] triggers){
        trigsAndResps.triggers = triggers;
    }

    public static void setResp(trigsAndResps trigsAndResps, String[] responses){
        trigsAndResps.responses = responses;
    }

    public static String[] getTrig(trigsAndResps trigsAndResps){ // Accessor␣
↪method to get triggers.
        return trigsAndResps.triggers;
    }

    public static String[] getResp(trigsAndResps trigsAndResps){ // Accessor␣
↪method to get responses
        return trigsAndResps.responses;
    }
    // Method to initialise the record with given String arrays.
    public static void createTrigResp(trigsAndResps trigsAndResps, String[]␣
↪triggers, String[] responses){
        setTrig(trigsAndResps, triggers);
        setResp(trigsAndResps, responses);
    }

    // Method to initialise questions and answers to recognise
    public static void createQuesAns (questionAnswerDatabase␣
↪questionAnswerDatabase, String[] questions, String[] answers, int[]␣
↪usefulness) {
        setQues(questionAnswerDatabase, questions);
        setAns(questionAnswerDatabase, answers);
        setUse(questionAnswerDatabase, usefulness);
    }

    // Accessor methods to get and set questions, answers, and usefulness
    public static void setQues(questionAnswerDatabase questionAnswerDatabase,␣
↪String[] questions) {
        questionAnswerDatabase.questions = questions;
    }

    public static void setAns(questionAnswerDatabase questionAnswerDatabase,␣
↪String[] answers) {
        questionAnswerDatabase.answers = answers;
    }

    public static void setUse(questionAnswerDatabase questionAnswerDatabase,␣
↪int[] usefulness) {
        questionAnswerDatabase.usefulness = usefulness;
    }

    public static void setUsePosition(questionAnswerDatabase␣
↪questionAnswerDatabase, int position, int newUse) {
        questionAnswerDatabase.usefulness[position] += newUse;
    }

    public static String[] getQuestions(questionAnswerDatabase␣
↪questionAnswerDatabase) {
        return questionAnswerDatabase.questions;
    }

    public static String[] getAnswers(questionAnswerDatabase␣
↪questionAnswerDatabase) {
        return questionAnswerDatabase.answers;
    }
    public static int[] getUse(questionAnswerDatabase questionAnswerDatabase) {
        return questionAnswerDatabase.usefulness;
    }
}

class trigsAndResps { // Main triggers and responses
    String[] triggers;
    String[] responses;
}

class questionAnswerDatabase { // Extra database to check if the first answers␣
 ↪don't help
    String[] questions;
    String[] answers;
    int[] usefulness;
}
```

2.7 Text file example:

Below is an example of the text file that I am writing to, "responses.txt". The format is very simple, each response is written to a new line from an array in the program. When entering new data, previous data is overwritten since it is no longer needed. When reading, each line is put into an array element in the program. mine

chop

block

snipe

slash

protect

END OF LITERATE DOCUMENT