Nikhil Sathisha

Criterion A: Planning

The Problem:

My clients for this product are amaeteur mushroom foragers like myself. I interviewed a potential client to gain more insight.

The method by which he identified mushrooms currently is by first searching up the color of the mushroom, and searching through various images, sorting by cap shape, potentially stem texture, and size. By scouring websites like Wikipedia and foraging books, as well as potentially using AI tools like google lens, it takes an average of 30-40 minutes of internet searching for him to identify a single mushroom.

However, these AI tools are often inaccurate, often mistaking mushrooms for those of relative species and can be fooled by differences in environment lighting, especially if a mushroom is turned upside down, as I personally have observed in many of my identification attempts.

Having a reliable and fast tool for mushroom identification is imperative for foragers, especially as consuming a misidentified edible mushroom could lead to serious risk of death. Differences in species can often come down to minute details that cannot be captured on camera or would not be evident to an amaeteur.

As he gets better at identifying mushrooms, the internet tools he uses have been revealed to be clunky, slow, and potentially dangerous. The product I am designing has been created to hasten and safen the identification process for my client(s).

Starting Success Criteria:

- Input where user can create a mushroom they would like to identify
- User can easily see the options for the traits, such as different letters for different color options.
- User can gradually add traits in a menu-like system.
- User can cross-check mushroom efficiently with database, which gets gradually narrowed down with incompatible mushrooms
- User can restart the process easily, creating a new mushroom object
- Database can be reset easily after being narrowed down
- User can enter multiple inputs for a trait,, if they found for example: a brown and white mushroom
- User interface is intuitive

Nikhil Sathisha

Criterion B: Record Of Tasks

Task Number	Planned Action	Planned Outcome	Time Estimate	Target Completion	Criterion
1	Gauge Interest for product	Talk to other mycologists in NC to look at current methods (books etc and gauge interest for new software to assis)	1 Hour	Nov 1	A
2	Create Problem Statement and Decide on Project Idea	Assess interest and organize ideas	1.5 Hours	Nov 6	A
3	Gauge resources available for the project.	Find books/libraries that contain datasets suitable for the task and write them down	1 Hour	Nov 13	B
4	Look at possible methods to sort out datasets like this.	Look at decision trees, potentially AI that creates them	30 min	Nov 14	B
5	Look at how professional applications handle this kind of data	SQL tables for handling tag databases, custom implementations using hashmaps	2 hours	Nov 27	B
6	Talk to clients again	To refine my idea of inputs/outputs for the search/ elimination	1 hour	Dec 4	A
7	Reimagine product idea	Look again at different implementations possible for the application	45 min	Dec 15	A

8	Update Software	Re-install VSCode as it had been giving me issues previously	30 min	Jan 1	C
9	Interpret Data	Take the .csv file containing the dataset and interpret it into objects in code	20 min	Jan 3	C
10	Create basic search	Begin to take user inputs and create methods to cross off mushrooms	30 min	Jan 6	C
11	Debug	Had issues with wrapper classes and their interaction with my methods.	30 min	Jan 9	C
12	Consult College Student	Brought idea to college student, asked about different implementations possible for the product	30 min	Jan 9	A
13	Implement Suggested Changes	Change the data structure to possibly utilize hashing to make the search inefficient	45 min	Jan 10	B
14	Finish writing for Criterion A	Finish last sentences, update descriptions of product.	25 min	Jan 11	A
15	Create Code to implement operations	Make my code allow for Union and Intersection, allowing me to easily exclude and include objects in a search	40 min	Jan 12	C
16	Make interface more	Create a menu so	1 hr	Jan 13	C

	user friendly	that gives users the names of the mushrooms that it might be once the search has narrowed possible species down to less than 10			
17	Research ways to efficiently store and assign images in Java (VSCode)	Look at ways to store .png or .jpg files that can be referenced by the user in the menu	1 hr	Jan 13	B
18	Debug	Fix issues with the ArrayList implementation	2 hrs	Feb 15	C
19	Edit	Change menu so that user can find compatible mushrooms at any time	1 hr	Feb 16	C
20	Finalize	Make sure code works with given dataset, Film video	1hr	Feb 20	C

Nikhil Sathisha

Criterion C: Development

Libraries Used:

import java.util.*
import java.io.*

util.ArrayList, util.Scanner and io.File were extensively used to organize and read data and user inputs.

ArrayList methods used:
- ArrayList.get()
- ArrayList.remove()
- ArrayList.set()
- ArrayList.add()

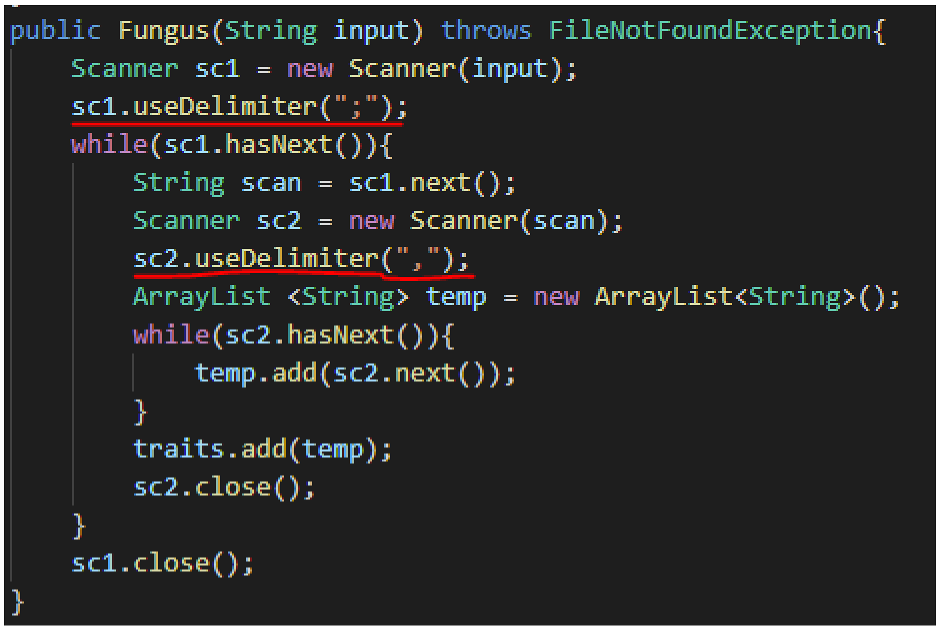
File methods used:
- File.File() (constructor)

Scanner methods used:
- Scanner.Scanner(File x) (constructor)
- Scanner.Scanner(System.in) (constructor)
- Scanner.next()
- Scanner.nextLine()
- Scanner.close()
- Scanner.nextInt()

Handling Data:

Through copying and pasting the data in a .csv file into a .txt file, and removing extraneous symbols using the find and replace tool build into VSCode, I could clean up the data to be used easily using two scanner objects, each with 2 different delimiters.

Figure 1: Scanner Objects With Delimiters



```
public Fungus(String input) throws FileNotFoundException{
    Scanner sc1 = new Scanner(input);
    sc1.useDelimiter(";");
    while(sc1.hasNext()){
        String scan = sc1.next();
        Scanner sc2 = new Scanner(scan);
        sc2.useDelimiter(",");
        ArrayList <String> temp = new ArrayList<String>();
        while(sc2.hasNext()){
            temp.add(sc2.next());
        }
        traits.add(temp);
        sc2.close();
    }
    sc1.close();
}
```

This corresponds to the divisions in the CSV, where the traits were separated by semicolons and the letters/numbers for each trait were separated by commas as shown below:

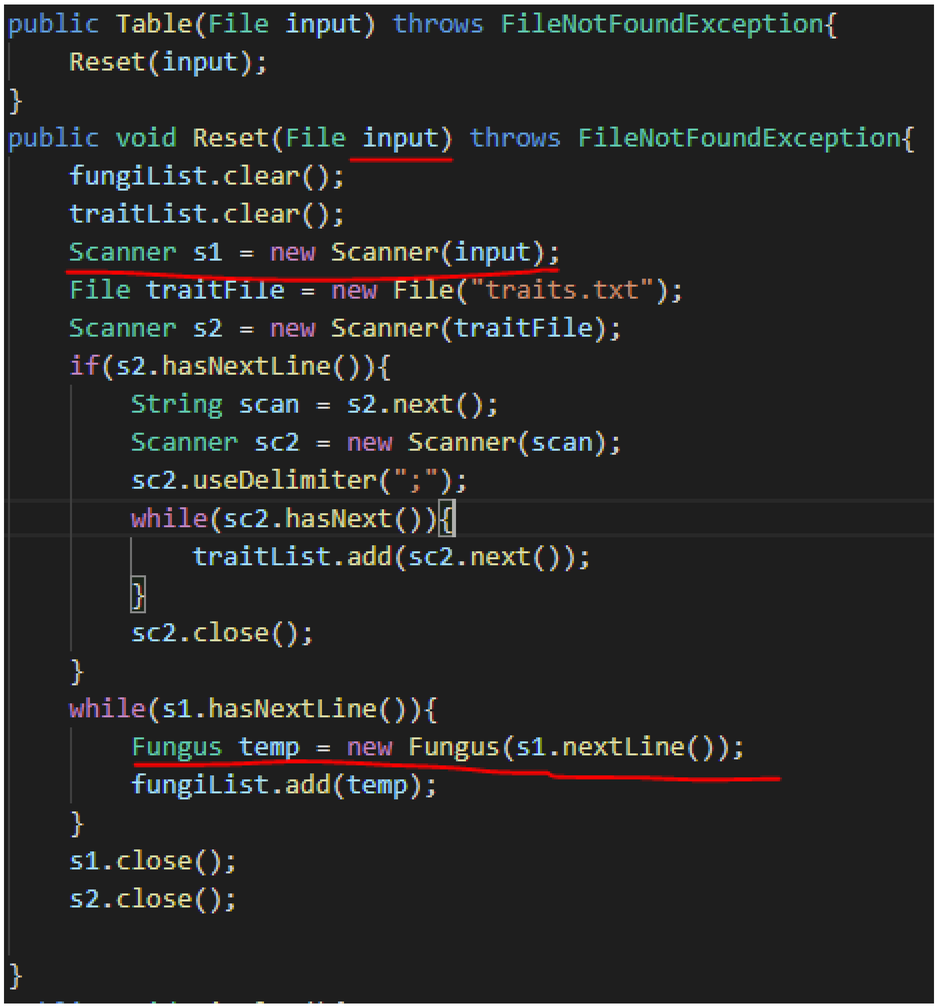
Fly Agaric;p**;10,20;**x,f,s;g,h;e,o,w;f;e;c;w,y;15,20;15,20;b;y;w;u;w;t;g,p;w;w;d;u,a,w;t

The bolded section represents a range of [10,20] for a given trait, as it is separated on left and right by semicolons and separated within by commas.

Organization

The table class would take the .txt file containing the cleaned database of mushrooms, and create mushroom objects stored in an ArrayList. Each mushroom object's constructor takes in a string containing the line from the .txt file as "divvied" by the table class.

Figure 2: Initializing all Fungus objects.



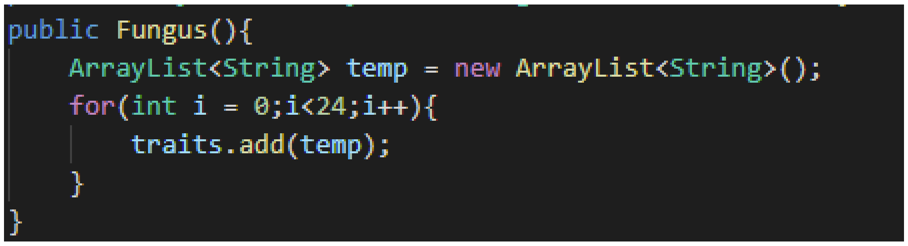
```
public Table(File input) throws FileNotFoundException{
    Reset(input);
}
public void Reset(File input) throws FileNotFoundException{
    fungiList.clear();
    traitList.clear();
    Scanner s1 = new Scanner(input);
    File traitFile = new File("traits.txt");
    Scanner s2 = new Scanner(traitFile);
    if(s2.hasNextLine()){
        String scan = s2.next();
        Scanner sc2 = new Scanner(scan);
        sc2.useDelimiter(";");
        while(sc2.hasNext()){
            traitList.add(sc2.next());
        }
        sc2.close();
    }
    while(s1.hasNextLine()){
        Fungus temp = new Fungus(s1.nextLine());
        fungiList.add(temp);
    }
    s1.close();
    s2.close();

}
```

The data from each line is put into an ArrayList of ArrayLists that contain strings in each Fungus object, to accommodate for the multiple entries for each trait in the database.

Creating Independent Fungus Objects

The user has to create a Fungus object to compare to the other Fungus objects stored in the table. In order to do this, I overloaded the constructor on the Fungus object, so that if an object is constructed with no argument, it will initialize the ArrayList of traits with 23 empty ArrayLists of Strings. This is so that the user can insert a trait for indices 0-23 without an indexoutofboundsexception.

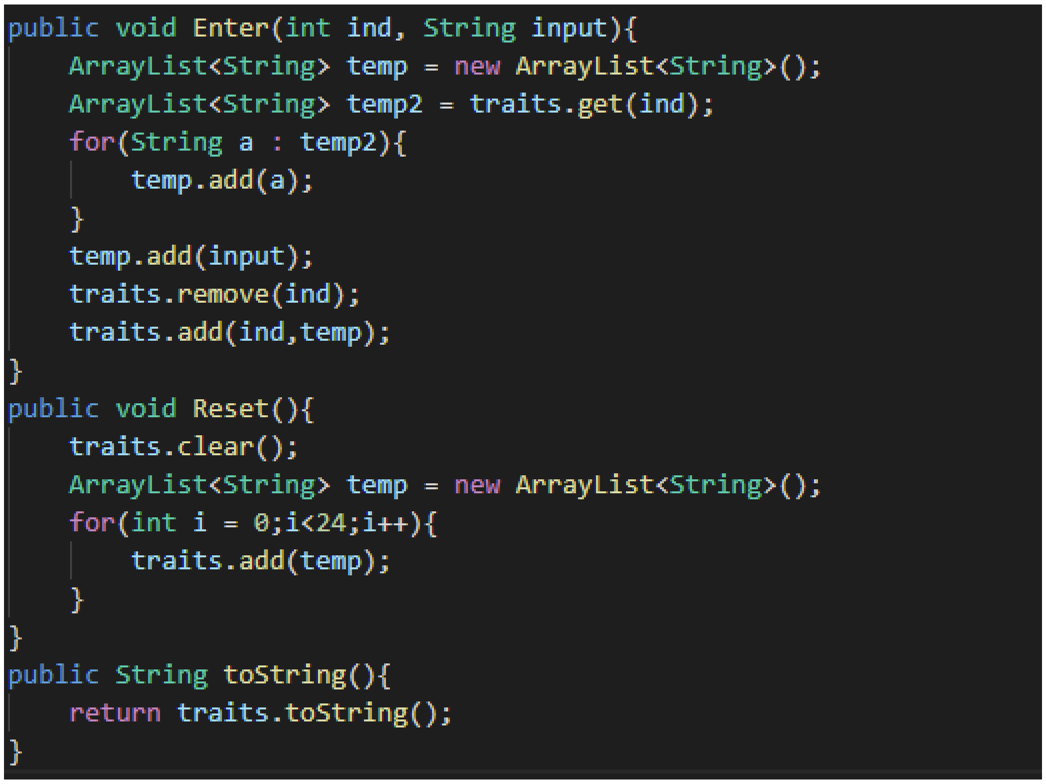
Figure 3: Fungus Constructor Overloading



```
public Fungus(){
    ArrayList<String> temp = new ArrayList<String>();
    for(int i = 0;i<24;i++){
        traits.add(temp);
    }
}
```

Creating the Fungus Enter, Reset and toString methods

Figure 4: Enter, Reset, and toString methods



```
public void Enter(int ind, String input){
    ArrayList<String> temp = new ArrayList<String>();
    ArrayList<String> temp2 = traits.get(ind);
    for(String a : temp2){
        temp.add(a);
    }
    temp.add(input);
    traits.remove(ind);
    traits.add(ind,temp);
}
public void Reset(){
    traits.clear();
    ArrayList<String> temp = new ArrayList<String>();
    for(int i = 0;i<24;i++){
        traits.add(temp);
    }
}
public String toString(){
    return traits.toString();
}
```

The Fungus Enter method allows via the menu for the user to enter an index and a string corresponding to that index to add to that trait.

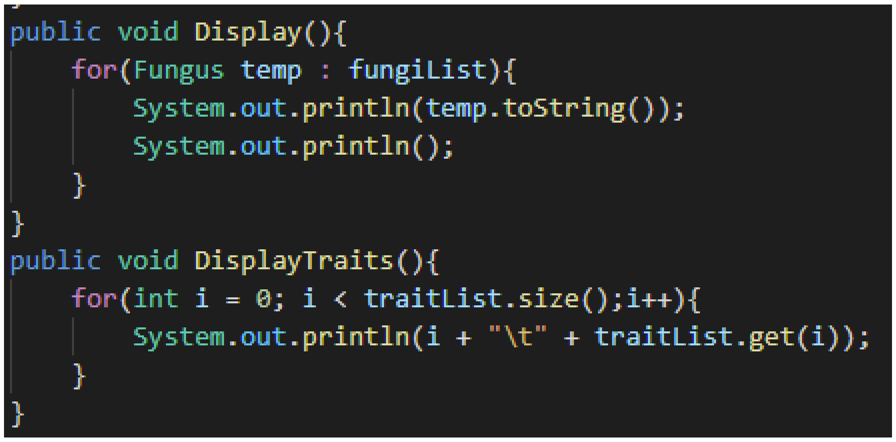
The enter method creates a new ArrayList (temp) and adds the Strings from the existing ArrayList at that index to the new temporary one. It then removes the old ArrayList from that index and replaces it with the temporary one.

The Reset method clears all ArrayLists and reinitializes them just like the constructor.

The toString method returns the same string as ArrayList.toString, returning all the traits (including the name). I found this format worked well visually and didn't need any changes.

Creating the display methods for Table.

Figure 5: Display methods



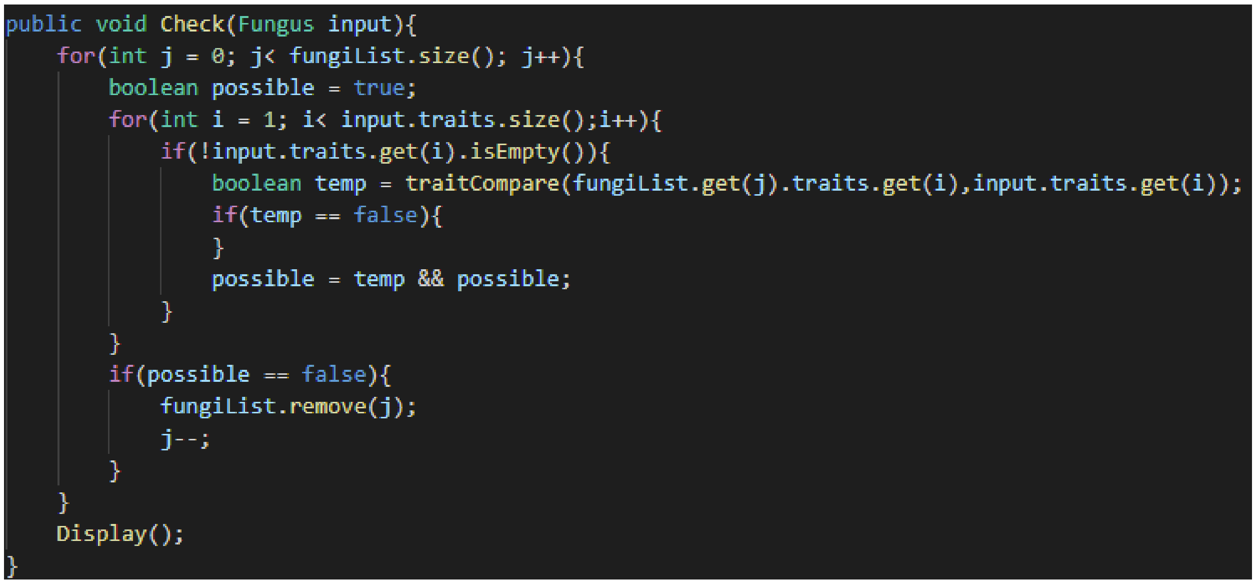
```java
public void Display(){
    for(Fungus temp : fungiList){
        System.out.println(temp.toString());
        System.out.println();
    }
}
public void DisplayTraits(){
    for(int i = 0; i < traitList.size();i++){
        System.out.println(i + "\t" + traitList.get(i));
    }
}
```

The Display() method prints out every Fungus object using its toString method and creating a new line for organization using a for-each loop.

The DisplayTraits() method prints out the traits with their indices from the traitList ArrayList with a tab to make it more visually appealing

Creating Check Method for Table

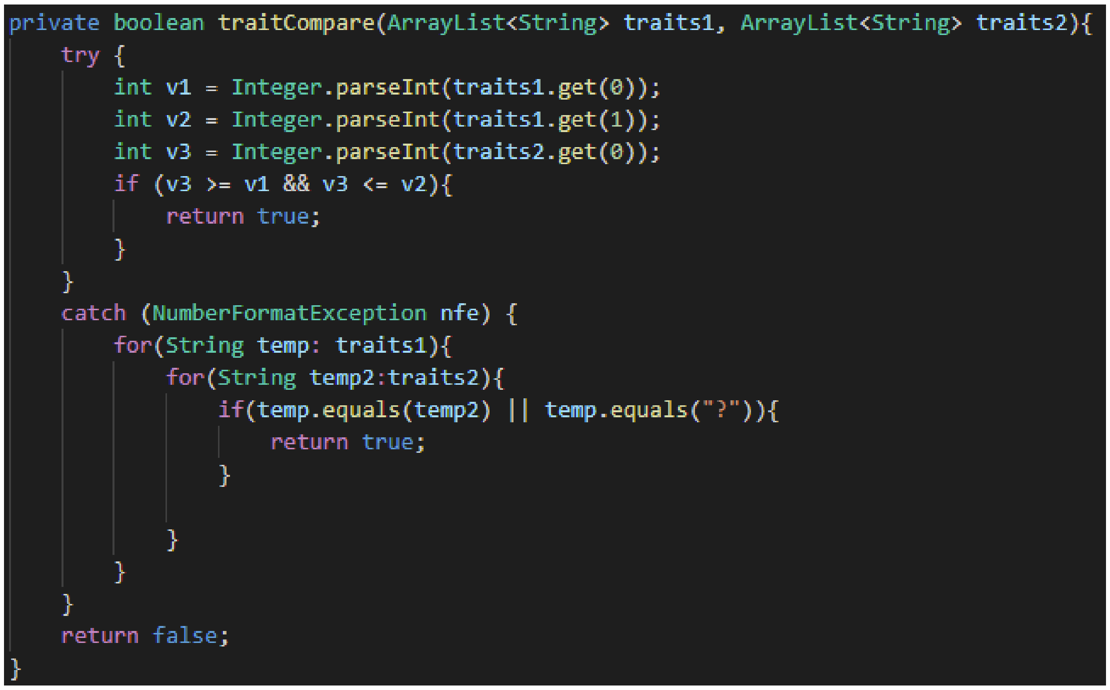
Figure 6: Check method



```java
public void Check(Fungus input){
    for(int j = 0; j< fungiList.size(); j++){
        boolean possible = true;
        for(int i = 1; i< input.traits.size();i++){
            if(!input.traits.get(i).isEmpty()){
                boolean temp = traitCompare(fungiList.get(j).traits.get(i),input.traits.get(i));
                if(temp == false){
                }
                possible = temp && possible;
            }
        }
        if(possible == false){
            fungiList.remove(j);
            j--;
        }
    }
    Display();
}
```

The check method utilizes a for loop to traverse the fungiList ArrayList, individually considering each fungus. Then it uses a nested for-loop with a condition to see whether the user input's mushroom object has empty traits to then call a nested method called traitCompare, inputting the ArrayList<String> from the Trait ArrayList from one Fungus pulled from the ArrayList contained in table, and the other from the user-created Fungus from the runner class.

Figure 7: traitCompare method



```java
private boolean traitCompare(ArrayList<String> traits1, ArrayList<String> traits2){
    try {
        int v1 = Integer.parseInt(traits1.get(0));
        int v2 = Integer.parseInt(traits1.get(1));
        int v3 = Integer.parseInt(traits2.get(0));
        if (v3 >= v1 && v3 <= v2){
            return true;
        }
    }
    catch (NumberFormatException nfe) {
        for(String temp: traits1){
            for(String temp2:traits2){
                if(temp.equals(temp2) || temp.equals("?")){
                    return true;
                }

            }
        }
    }
    return false;
}
```

Since the mushroom objects can either have numerical ranges, given as two numbers or can have letters representing categorical data, I used a try and catch statement, where it tries the Integer.parseInt() method, and if it throws NumberFormatException, I treat it as a letter and use String.equals().

This method only returns false if the number from the User input does not fall in the range, or there are no matches between the letters representing categorical data. Note: the database is somewhat incomplete and has “?”s representing unknowns, so the method also returns true if there is a “?” for the database values.

Also, due to the condition inside the nested for loops in the Check() method, there is no need to worry about nullPointerExceptions when trying to access the inputted ArrayLists’ values as we know the user input is not empty. This also makes the code much more efficient to run as it is only checking the traits which the user has created an input for.

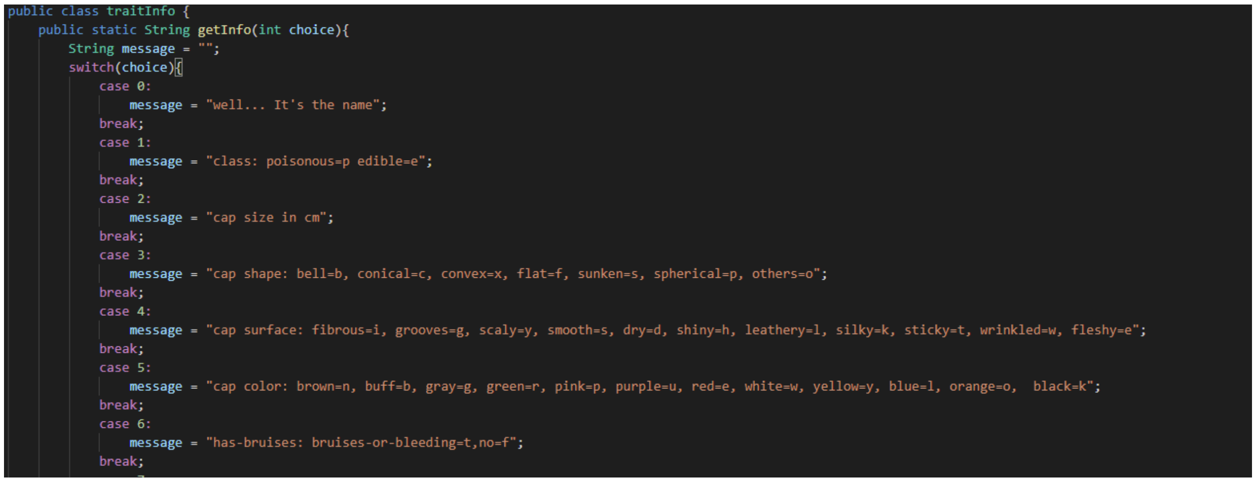
Finally, the check method utilized a boolean AND operation, such that if any of the traitCompare() calls return false, that mushroom is not considered and is removed from the fungiList.

This also makes the code more efficient, as after initial eliminations the code runs on fewer and fewer objects.

Nikhil Sathisha

traitInfo Class

Figure 8: traitInfo class

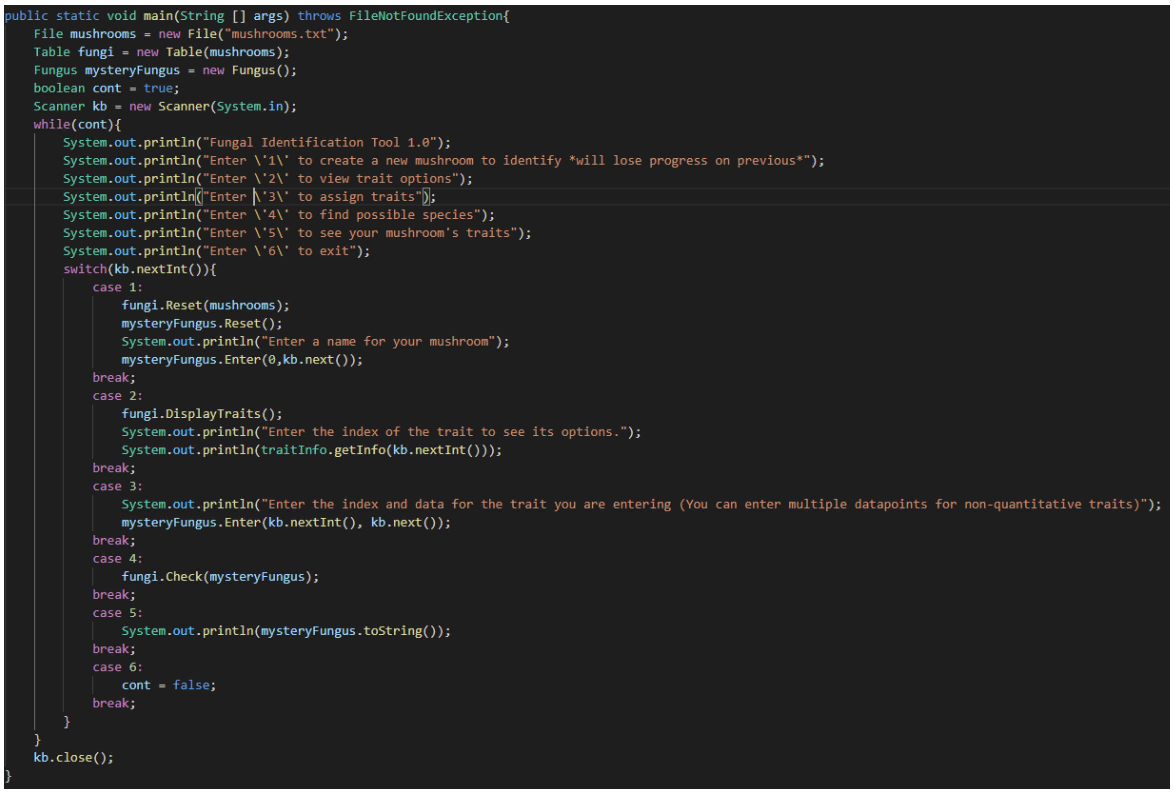


```
public class traitInfo {
    public static String getInfo(int choice){
        String message = "";
        switch(choice){
            case 0:
                message = "well... It's the name";
            break;
            case 1:
                message = "class: poisonous=p edible=e";
            break;
            case 2:
                message = "cap size in cm";
            break;
            case 3:
                message = "cap shape: bell=b, conical=c, convex=x, flat=f, sunken=s, spherical=p, others=o";
            break;
            case 4:
                message = "cap surface: fibrous=i, grooves=g, scaly=y, smooth=s, dry=d, shiny=h, leathery=l, silky=k, sticky=t, wrinkled=w, fleshy=e";
            break;
            case 5:
                message = "cap color: brown=n, buff=b, gray=g, green=r, pink=p, purple=u, red=e, white=w, yellow=y, blue=l, orange=o,  black=k";
            break;
            case 6:
                message = "has-bruises: bruises-or-bleeding=t,no=f";
            break;
```

Utilizing the data from the UC Irvine database, the README contained a key. I manually copied and pasted the key into this long switch statement, so the user can clarify on the key for each trait at any time in the menu.

Runner and main method

Figure 9: Runner class



```
public static void main(String [] args) throws FileNotFoundException{
    File mushrooms = new File("mushrooms.txt");
    Table fungi = new Table(mushrooms);
    Fungus mysteryFungus = new Fungus();
    boolean cont = true;
    Scanner kb = new Scanner(System.in);
    while(cont){
        System.out.println("Fungal Identification Tool 1.0");
        System.out.println("Enter \'1\' to create a new mushroom to identify *will lose progress on previous*");
        System.out.println("Enter \'2\' to view trait options");
        System.out.println("Enter \'3\' to assign traits");
        System.out.println("Enter \'4\' to find possible species");
        System.out.println("Enter \'5\' to see your mushroom's traits");
        System.out.println("Enter \'6\' to exit");
        switch(kb.nextInt()){
            case 1:
                fungi.Reset(mushrooms);
                mysteryFungus.Reset();
                System.out.println("Enter a name for your mushroom");
                mysteryFungus.Enter(0,kb.next());
            break;
            case 2:
                fungi.DisplayTraits();
                System.out.println("Enter the index of the trait to see its options.");
                System.out.println(traitInfo.getInfo(kb.nextInt()));
            break;
            case 3:
                System.out.println("Enter the index and data for the trait you are entering (You can enter multiple datapoints for non-quantitative traits)");
                mysteryFungus.Enter(kb.nextInt(), kb.next());
            break;
            case 4:
                fungi.Check(mysteryFungus);
            break;
            case 5:
                System.out.println(mysteryFungus.toString());
            break;
            case 6:
                cont = false;
            break;
        }
    }
    kb.close();
}
```

Nikhil Sathisha

The runner class contains initialization statements for the Table and for the user- edited Fungus.

It then enters a menu where the user can utilize the aforementioned classes and methods to narrow down and identify from the Fungus database.

Nikhil Sathisha

Criterion E: Evaluation

Success Criteria

Input where user can create a mushroom they would like to identify	Met, the program does this with a menu option
User can easily see the options for the traits, such as different letters for different color options.	Met, there is a menu option for this
User can gradually add traits in a menu-like system.	Met, there is a menu
User can cross-check mushroom efficiently with database, which gets gradually narrowed down with incompatible mushrooms	Almost Met, the list narrows down as traits are added but there are some bugs that eliminate Fungi objects that should not have been eliminated.
User can restart the process easily, creating a new mushroom object	Met, there is a menu option for this
Database can be reset easily after being narrowed down	Met, there is a menu option for this
User can enter multiple inputs for a trait, if they found for example: a brown and white mushroom	Met, the tool allows for this
User interface is intuitive	Not completely met, text based interface can be hard to follow, and user cannot input multiple traits before returning to the menu.

Recommendations for Further Development

1. Check the boolean operation code and cross-check with the database to check whether the program is eliminating Fungus objects that should not be
2. Re-evaluate the menu so that it can take multiple inputs
3. Potentially create a GUI so that instead of using numbers to refer to a trait's index, a user can click a trait to learn or enter values for their user Fungus. +