Handling Errors

|  |  |  |
| --- | --- | --- |
| # | Sample Potential Errors | Error Handling |
| 1 | System.out.println("Enter your choice");  int choice = input.nextInt();  If the user inputted a wrong number as option in the main menu (either wrong type or wrong number) | If statement identifies if the user inputs a wrong number  if (choice < 1 || choice > 2) {  System.out.println("Invalid choice. Please choose again");  displayMainMenu();}  If the user inputs a string instead of a number, there is a try and catch block. |
| 2 | At the ingredients submenu, if the user inputs a number that does not correspond to the listed options, or not a number.  1) Add an ingredient  2) Delete an ingredient  3) Search/Edit an ingredient  4) Sort ingredients  5) Return to main menu | This is handled using and if statement if the user entered a number, and a try and catch block if the user did not enter an int value, similar to #1. |
| 3. | When choosing a food group when adding an ingredient, if the user does not enter a valid option.  1) Diary  2) Fats  3) Fruit  4) Grains  5) Meats  6) Sweets  7) Vegetables | See #1 and #2 |
| 4 | When adding a food item, if the user enters a negative value when prompting for the stock value and mass. The input is the brackets.  Enter the stock of the food: (-50) | An if statement is used to determine if the inputted number is above 0.  int sampleStock = input.nextInt();  input.nextLine();  if (sampleStock < 0)  {  System.out.println("Cannot have negative stock");  displayIngredientsMenu(arrayOfFoodItems);  } |
| 5 | When adding a food, and the user must enter all the features of the food, if the user does not enter any feature (inputs x), then the features field of that ingredient is empty, and thus an empty line would be printed in the text file. When printing to the text file after an empty line, the empty line is replaced, as well as all the information about that ingredient. | String lineOfFeatures = "";  boolean exit = false;  while (!exit)  {  String currentFeature = input.nextLine();  if(currentFeature.equalsIgnoreCase("x"))  {  exit = true;  if (lineOfFeatures == "")  {  lineOfFeatures = "None";  }  }  else  {  lineOfFeatures = lineOfFeatures + "," + currentFeature;  }  The combined use of the while loop and the for loop can help handle this potential error. If the user does not enter a feature, that is inputted x immediately, then the String lineOfText will contain empty space. The if statement searches for this condition, and assigns the lineOfText a value of “None”. |
| 6 | When deleting a food, and the user enters a food that is not in the database. | The food is searched by the searchByName(String sampleName, Ingredient[] ingredients) method which returns an int. If this method returns -1, then the ingredient with name field of sampleName is not in the array ingredients. |
| 7 | Discrepancy between array of ingredients saved in the data file and the updated array of foods used during runtime. | saveArrayOfFoodItems(loadArrayOfFoodItems(arrayOfFoodItems))  The saving of the arrayOfFoodItems to the data file occurs immediately as it is loaded. |
| 8 | Information from text files are read into datatype String, which cannot be processed in some cases (such as assigning this String to the mass field of an ingredient) | Use of Integer.parseInt() and Double.parseDouble().  sample.setMass(Integer.parseInt(lineOfText)) |
| 9 | The constant use of input.nextInt() causes the program to skip input during runtime.  System.out.println("Enter the stock of the food: ");  int sampleStock = input.nextInt(); | The insertion of input.nextLine() at lines after input.nextInt() so that the buffer can proceed to the next line, instead of saving the empty space from input.nextInt() from before.  System.out.println("Enter the stock of the food: ");  int sampleStock = input.nextInt();  input.nextLine(); |
| 10 | If the file has been deleted by the user. | catch (FileNotFoundException e)  {  System.out.println("File could not be found");  }  Use of try and catch block so that the user can identify the problem of the program. |
| 11 | When printing to the database, print function is used for the instance of the PrintWriter class in a for loop.  databasePrinter.print("\n" + arrayOfFoodItems[a].getName())  databasePrinter.print("\n" + arrayOfFoodItems[a].getFoodGroup());  databasePrinter.print("\n" + arrayOfFoodItems[a].getColor());  databasePrinter.print("\n" + arrayOfFoodItems[a].getStock());  databasePrinter.print("\n" + arrayOfFoodItems[a].getMass());  databasePrinter.print("\n" + arrayOfFoodItems[a].getFeaturesInStringTabbed());  However for the first ingredient that is saved to the data file, there will be an empty line at the beginning of the text file. Thus, the number of lines is not divisible by 6, which is crucial error since the number of foods calculated in the array is the number of lines divided by 6. | An if statement is used within the for loop to identify the first object that is being saved.  for (int a = 0; a < arrayOfFoodItems.length; a++) // changes number of food to arrayOfFoodItems.length  {  if (a == 0)  {  databasePrinter.print(arrayOfFoodItems[a].getName());  }  else  {  databasePrinter.print("\n" + arrayOfFoodItems[a].getName());  }… |
| 12 | If there is some error in the text file, or if the end of the file is reached | catch (IOException e)  {  System.out.println("There is an error in the input file");  }  Use of try and catch block so that the user can identify the problem of the program. |
| 13 | When converting one instance of a food group class to a type of another food group class, there is an error since that instance cannot be assigned to null. It is still in the array, and thus still considered existing. | Since all food groups objects are sub classes to the Ingredients class, and all food groups are declared with the datatype Ingredient, then a food group object can be instantiated to another object type of food group.  sample = new Diary(sample.getName(), sampleFoodGroup, ….)  sample.setFoodGroup(sampleFoodGroup);  sample is of datatype Ingredient, and thus its previous datatype, which is a subclass to Ingredient, can be changed to another object type of the subclass of the Ingredient class. |
| 14 | Other unexpected errors | catch (Exception e)  {  System.out.println("Error");  }  Although this code will handle other potential errors, it is not useful since it is a general error and thus cannot immediately know why the error occured |

Success of Program

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Function | Purpose | Input | Output | Test Result |
| Selecting option from Main Menu | To test if the main menu can properly lead the user into the submenu, and if the main menu can quit the user. | Valid inputs (1) | Enters sub menu | Success |
| Invalid input (234323423) | “Invalid Choice Please choose again.” and displays the main menu again. | Success |
| “12???” is entered | “Invalid Input Type” and displays the main menu again. | Success |
| Valid input(2) | Quits and displays a friendly message | Success |
| Adding a Food Item | To test if a student can be added to the database. | (In order)  Apple Pie  Dessert  Brown  20  13  Crispy  Baked  Warm  x | “Apple Pie has been added” | Success |
| (In order)  Apple Pie  Dessert  Brown  -2 | “Cannot have negative stock | Success |
| Search for an ingredient (By name) | To test if the ingredients saved can be identified and information about that food can be accessed. | Input :  Apple Pie | “What would you like to do ” Then the edit menu is displayed | Success. |
| Guitar | “Guitar is not found in the database”. Then is main menu is displated | Success |
| Search (by food group) | To test if the list generated is accurate | Fruit | Displays ingredients with this food group and all their info | Success |
| Coffee | “No foods found of food group Coffee | Success |
| Search (by Key Feature) |  | Healthy | Displays all the information of the foods that have this feature | success |
| 8899jhjkhjkh7 | Prints an empty line, indicating that no results have been found. | Success |
| Modify Food information from edit menu | To see if changes can be saved to text file | Searched for pineapple, chose 2 to change the food group, and entered “vegetable” | “Successfully updated | Success. |
| Searched for pineapple, chose 2 to change the stock, and entered “tyt” | “Cannot have negative stock” | Success |
| Delete a food item | To test if a user defined ingredient can be deleted from the database | Chose 2 (for delete) from the ingredients submenu. Entered “Strawberry cake” which is not in the database. | “Strawberry cake is not found in the database” | Success |
| Chose 2 (for delete) from the ingredients submenu. Entered “Peanut” which is in the database. Put y for sure. | “Peanut is successfully deleted” | Success |
| Sorting Ingredients | To tests if sorts are accurate | Input to sort ingredients by stock | outputs the list of the name of the ingredients in the order of fewest stock count to greatest stock count. | Success |

|  |  |
| --- | --- |
| **Mastery Factor** | **Where it would be used** |
| 1. Arrays | an array of features for each ingredient |
| 1. User-defined objects | When adding an ingredient object, the user must define its name, food group, color, mass, stock count, and key features. |
| 1. Objects as data records | a record for each ingredient (eg. Inventory count) |
| 1. Simple selection (if-else) | If the results of the searchByName method is less than zero, then the target ingredient is not found in the existing arrayOfFoodItems. An if statement was used to identify this value, and output that the searched ingredient does not exist in the datafile. |
| 1. Complex selection (nested if, if with multiple conditions or switch) | The food submenu has multiple conditions since it has multiple options. Also, there are nested if statements to see if the stock number that the user inputted is above 0. |
| 1. Loops | A while loop for counting the number of lines in the database text file lineCounter(); |
| 1. Nested loops |  |
| 1. User-defined methods | Method for searching if a specific ingredient is in the existing arrayOfFoodItems. searchByName(String sampleName, Ingredients[] ingrediets) |
| 1. User-defined methods with parameters | The method for finding a recipe given a name |
| 1. User-defined methods with appropriate return values | Method for checking if the food group is going to expire in next 3 days. Returns true it if going to expire, return false if not going to expire. |
| 1. Sorting | Method for sorting ingredients by stock number |
| 1. Searching | A method for searching ingredients by their name |
| 1. File i/o | File input is a file detailing all the pre-existing ingredients.  File output is output to the same file, of all the ingredients that may have been modified. |
| 1. Use of additional library |  |
| 1. Use of sentinels or fla gs |  |

|  |  |
| --- | --- |
| **Mastery Factor** | **Where it would be used** |
| 1. Adding data to an instance of the **RandomAccessFile** class by direct manipulation of the file pointer using the **seek** method |  |
| 1. Deleting data from an instance of the **RandomAccessFile** class by direct manipulation of the file pointer using the **seek** method. (Data primitives or objects may be shuffled or marked as deleted by use of a flag field. Therefore files may be ordered or unordered). |  |
| 1. Searching for specified data in an instance of the **RandomAccessFile** class. |  |
| 1. Recursion | When adding the stock of each ingredient in the specific food group.  public int sumOfStock(IngredientNode n) |
| 1. Merging two or more sorted data structures | The arrayOfFoodItems, which is a String array, is used simultaneously with the array of food items in a linked list. |
| 1. Polymorphism | Overriding the toString() from the Ingredient class to the food groups sub classes. |
| 1. Inheritance | The Ingredients class is the super class to all the food groups class, which are the Diary, Fats, Fruits, Grains, Meats, Sweets, and Vegetables sub classes. |
| 1. Encapsulation | The Ingrredient class, which hides important data from other classes. |
| 1. Parsing a text file or other data stream | Reading from data file, which each 6 lines represent a single ingredient. In one of those lines contain a list of String that is separated by a single tab. |
| 1. Implementing a hierarchical composite data structure. A composite data structure in this definition is a class implementing a record style data structure. A hierarchical composite data structure is one that contains more than one element and at least one of the elements is a composite data structure. Examples are, an array or linked list of records, a record that has one field that is another record, or an array. | An array of Ingredients created in the FoodItemDatabase each have an array of features as their instance field. |
| 1. The use of any five standard level mastery factors—this can be applied only once | (see above) |
| 1. – 15. Up to four aspects can be awarded for the implementation of abstract data types (ADTs) according to the table entitled “Implementation of ADTs”. (Please refer to the IB curriculum document) | The use of linked list to search, sort and delete ingredients. |
| 16. Use of additional libraries (such as utilities and graphical libraries not included in appendix 2 Java Examination Tool Subsets) |  |
| 17. Inserting data into an ordered sequential file without reading the entire file into RAM. |  |
| 18. Deleting data from a sequential file without reading the entire file into RAM. |  |
| 19. Arrays of two or more dimensions. | 2D array for reading all ingredient info. Every six lines describe a new ingredient. |