Assignment 2

Team number: 75
Team members

Name	Student Nr.	Email
Lucas Kim	2798807	lucassukeunkim@gmail.com
Sanskar Shrestha	2797647	20216661@life.hkbu.edu.hk
Bora Tarlan	2797649	b.tarlan@student.vu.nl
Sean Hermes	2796616	s.h.hermes@student.vu.nl

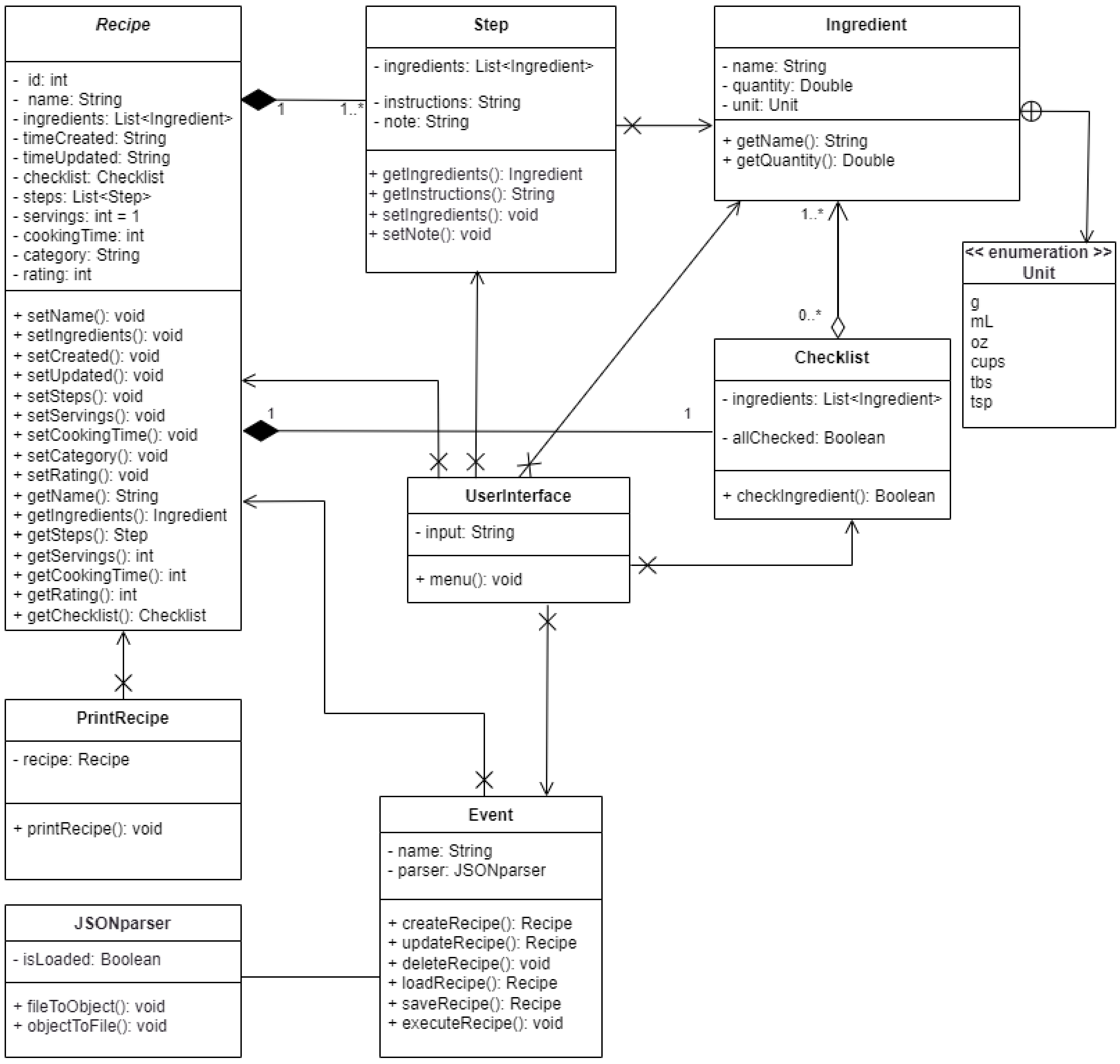
Summary of changes from Assignment 1

Author(s): all members

- **Functional Feature #5:** Instead of creating the recipe rating function for all possible users, keep the recipe rating locally for individual users.
- **Quality Requirement #3:** instead guaranteeing a quantity of time when referring to quick runtime, use non-quantitative language since every machine will run the program at a different speed

Class diagram

Author(s): all members



Recipe

The **Recipe** class is the most important class of the whole system - all other classes' functionalities will be directly or indirectly related to **Recipe**. Objects of this class will be akin to a recipe in a cookbook; they will hold information about every ingredient needed, every step of the recipe, and every instruction. The user will be able to save the data from a **Recipe** object in the form of a JSON file, and consequently will be able to load recipe data from a JSON file to create a **Recipe** object.

Attributes

- *id: int* → the ID of the **Recipe**
- *name: String* → the name of the **Recipe**

- *ingredients: List<Ingredient>* → the list of required **Ingredient**s for the **Recipe**
- *timeCreated: String* → time & date at which the **Recipe** was created
- *timeUpdated: String* → time & date at which the Recipe was updated
- *checklist: Checklist* → the **Ingredient**s checklist belonging to the **Recipe**, used at execution
- *steps: List<Step>* → the ordered list of **Step**s that contain all instructions for the user on how to cook the recipe. **Step**s will be executed one by one, in order.
- *servings: int* → the number of servings the **Recipe** is set to create (default 1). This affects ingredient quantities.
- *cookingTime: int* → the time it will take to cook the **Recipe**
- *category: String* → the category to which the **Recipe** belongs on the user's local machine. Categories will be local folders that hold recipes in the form of JSON files.
- *rating: int* → the user will be able to assign a rating to a **Recipe** after they are finished executing it based on how much they enjoyed the **Recipe**. This rating is stored along with all the other information of a **Recipe** in a JSON file.

Associations

- ***Step*** → Composition: every **Recipe** requires at least 1 and likely more **Step**s to execute, and individual **Step**s cannot exist outside of a **Recipe**. Also, each **Step** is unique to a single **Recipe**.
- ***Event*** → Binary Association: An **Event** needs access to the attributes of a **Recipe** in order to carry out certain actions properly, such as updating or executing a **Recipe**. However, **Recipe** does not need access to the **Event** class.
- ***Checklist*** → Composition: Each **Recipe** should have one **Checklist**, and every **Checklist** will belong to a specific **Recipe. Checklist**s cannot exist outside of a **Recipe**.
- ***UserInterface*** → Binary Association
- ***PrintRecipe*** → Binary Association

Ingredient

The **Ingredient** class is essential for the system to function as a cookbook would. Every **Recipe** requires certain **Ingredient**s, which can be found in the **Recipe**'s *ingredients* and *steps* attributes - every **Step** contains information about relevant **Ingredient**s and instructions. When a user decides to execute a **Recipe**, they will first have to check if they have all the necessary **Ingredient**s. Also, when a user creates a **Recipe**, they will have to specify what **Ingredient**s are necessary and what quantities.

Attributes

- *name: String* → the name of the **Ingredient**
- *quantity: Double* → the quantity of the **Ingredient** needed
- *unit: Unit* → the unit in which the quantity of the **Ingredient** is measured

Associations

- ***UserInterface*** → Binary Association
- ***Step*** → Binary Association
- ***Checklist*** → Shared Aggregation

Event

The **Event** class is the backbone of all system functionality. **Event**s will trigger important actions such as creating, deleting, updating, executing, loading, and saving **Recipe**s. **Event**s will be created through the user's decision making within the **UserInterface** class.

Attributes

- *name: String* → the name of the **Event**
- *parser: JSONparser* → instance of the **JSONparser** class to be used in order to load a JSON file of a recipe or save a recipe object into a JSON file.

Operations

- *createRecipe():* **Recipe** → creates an instance of the **Recipe** object with appropriate details as instructed by the user
- *updateRecipe()*: **Recipe** → changes the details of the Recipe after the JSON file of the particular recipe is loaded.
- *deleteRecipe()*: void → removes the JSON file of the targeted recipe with the help of an instance of the **JSONparser** class.
- *loadRecipe()*: **Recipe** → loads the details of the particular recipe using a **JSONparser** object.
- *saveRecipe()*: **Recipe** → saves the information of the recipe and triggers the **JSONparser** class in order to save the recipe as a JSON file.
- *executeRecipe()*: void → executes the details of the recipe.

Associations

- ***JSONparser*** → Binary Association
- ***UserInterface*** → Binary Association

UserInterface

The **UserInterface** class will contain the *main()* method of the program and within that, the program loop which allows the user to use the program for as long as they want and decide when they want to quit. The **UI** will display all necessary information to the user for them to be able to use the program properly and efficiently, such as through the *menu()* method.

Attributes

- *input: String* → Initially, the user will be prompted to input commands so as to access the desired functionality of the system.

Operations

- *menu()*: *void* → displays a menu of options available for the user to choose. Each option will lead to a different process in the program.

Associations

- ***Step*** → Binary Association
- ***Checklist*** → Binary Association

JSONparser

The **JSONparser** class will, as the name suggests, handle the parsing of the JSON files which store all **Recipe** data. The **Event** class will call upon **JSONparser** when its *loadRecipe()* method is called, as this is how recipe data from a JSON file will be stored in a newly created **Recipe** object, which can then be used in other ways by the system.

Attributes

- *isLoaded: Boolean* → To provide the indication of whether the data from the JSON file of the targeted recipe is stored in a Recipe object.

Operations

- *fileToObject(): void* → This function converts the JSON file of the particular recipe into a Recipe object which can be used by the user.
- *objectToFile(): void* → This function converts the data of the targeted Recipe object into JSON file in order to save the Recipe for future use.

Checklist

A **Checklist** object is created for a specific **Recipe** and will be used to ensure that the user has all required **Ingredient**s to make the food. The **Checklist** class is only ever used when the user first decides to execute a **Recipe**.

Attributes

- *ingredients: List<Ingredient>* → It stores the list of all the required ingredients for a recipe.
- *allChecked: Boolean*→ It indicates whether the user has all the ingredients needed for the recipe.

Operations

- *checkIngredient(): Boolean* → It is a function that is called when checking each of the ingredients with the required ingredients for a recipe.

PrintRecipe

The **PrintRecipe** class will be used to create a printable document containing all relevant information of a **Recipe** so that the user would be able to execute the **Recipe** without the need of running the program at the same time. A generated document would contain a description of a **Recipe**, its required **Ingredient**s, and all the data from each **Step**.

Attributes

- *recipe: Recipe* → the **Recipe** containing each **Ingredient** and **Step**. This information is required in the **PrintRecipe** class so that it can successfully create a printable document containing all of the **Recipe** information.

Operations

- *printRecipe(): void* → generates a printable document containing the relevant information of a Recipe.

Step

The **Step** class exists for **Recipe**s to have structured procedures when being executed by a user. Each **Step** of a **Recipe** contains a list of **Ingredient**s relevant to that **Step** as well as instructions for the user. When a **Recipe** is executed, the user will be taken through each **Step** in order, one at a time, until the **Recipe** is finished executing. When a user creates a new **Recipe**, they will have to specify all of the information relevant to each **Step**.

Attributes

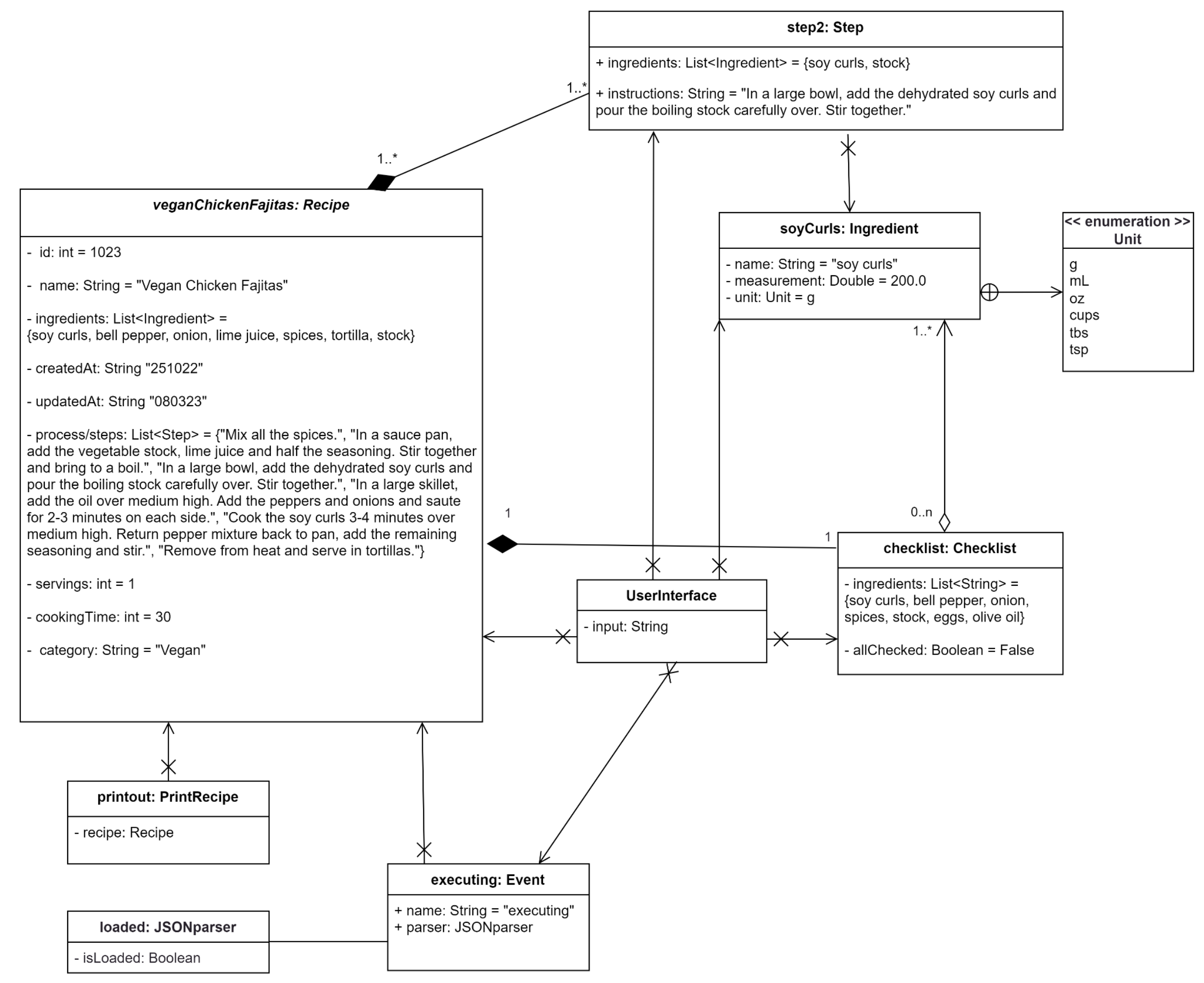
- *ingredients: List<Ingredient>* → the required **Ingredient**s for this specific **Step**
- *instructions: String* → the instructions for this specific **Step** that tell the user what to do with the **Ingredient**s
- *note: String* → a user-added note for this specific **Step** of a **Recipe**. Users can add notes to a **Step** during the execution of a **Recipe**.

Unit

The Unit class exists to specify in what unit of measurement the *quantity* of each **Ingredient** is measured. Every **Ingredient** will have an attribute called *unit* of the type **Unit**.

Object diagram

Author(s): Bora Tarlan



step2: Step
+ ingredients: List<Ingredient> = {soy curls, stock}
+ instructions: String = "In a large bowl, add the dehydrated soy curls and pour the boiling stock carefully over. Stir together."
1..*
1..*
veganChickenFajitas: Recipe
- id: int = 1023
- name: String = "Vegan Chicken Fajitas"
- ingredients: List<Ingredient> = {soy curls, bell pepper, onion, lime juice, spices, tortilla, stock}
- createdAt: String "251022"
- updatedAt: String "080323"
- process/steps: List<Step> = {"Mix all the spices.", "In a sauce pan, add the vegetable stock, lime juice and half the seasoning. Stir together and bring to a boil.", "In a large bowl, add the dehydrated soy curls and pour the boiling stock carefully over. Stir together.", "In a large skillet, add the oil over medium high. Add the peppers and onions and saute for 2-3 minutes on each side.", "Cook the soy curls 3-4 minutes over medium high. Return pepper mixture back to pan, add the remaining seasoning and stir.", "Remove from heat and serve in tortillas."}
- servings: int = 1
- cookingTime: int = 30
- category: String = "Vegan"
soyCurls: Ingredient
- name: String = "soy curls"
- measurement: Double = 200.0
- unit: Unit = g
<< enumeration >> Unit
g
mL
oz
cups
tbs
tsp
1..*
0..n
1
1
checklist: Checklist
- ingredients: List<String> = {soy curls, bell pepper, onion, spices, stock, eggs, olive oil}
- allChecked: Boolean = False
UserInterface
- input: String
printout: PrintRecipe
- recipe: Recipe
executing: Event
+ name: String = "executing"
+ parser: JSONparser
loaded: JSONparser
- isLoaded: Boolean

veganChickenFajitas: This is an instance of the **Recipe** class which as mentioned before is an integral part of the system that acts both as an information container and a central point of the system that communicates with the rest of the classes. This instance of the **Recipe** class holds a sample recipe object with its corresponding attributes, such as list of steps and ingredients.

checklist: The checklist object in the diagram is an instance of the class with the same name that tracks the ingredients that the user has in this snapshot of the system with its *ingredients* attribute. The object also has a boolean *allChecked* which conveys the information of whether the user has the necessary ingredients for the corresponding recipe by comparing the list of ingredients of the recipe and the user in its instantiation, which in this case evaluates to “False”.

printout: This object facilitates the process of having a printable document by storing the necessary information required to follow the recipe by having a recipe attribute. Through the binary association the **printRecipe** class has with the **Recipe** class, in the initialization of **printout**, a copy of **veganChickenFajitas** could be made.

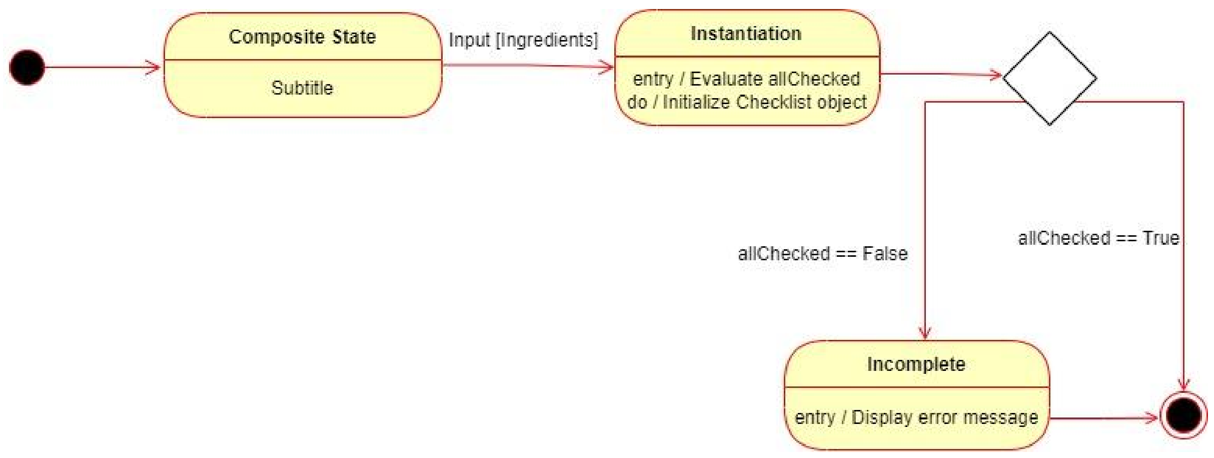
loaded: At the start of each execution of the system, the pre-saved recipes and relevant data should be loaded into the system via the JSON save files. At the moment of the snapshot the object diagram depicts, the loading of the JSON files must have been completed already, which is why *isLoaded* evaluates to “True”.

executing: The instance of the **event** class named executing is a key element of the system in this moment of the system since the diagram illustrates the condition of the system after the call executeRecipe() was made by the **event** class object, setting in motion the process which will be further explored by a sequence diagram.

State machine diagrams

Author(s): Sanskar Shrestha and Bora Tarlan

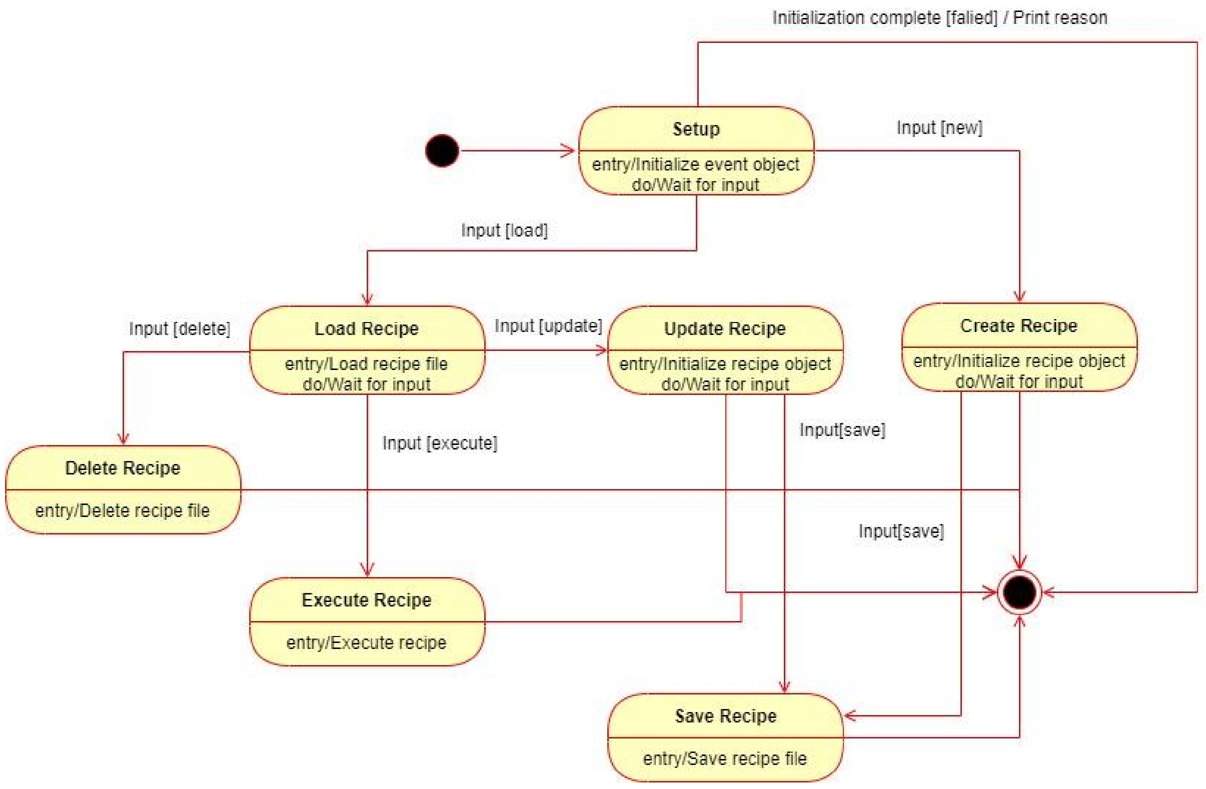
Checklist



The state machine diagram that describes the various stages an instance of the Checklist class goes through to provide the functionality of confirming whether the user can actually follow through with the recipe given the ingredients that they have.

This state machine diagram consists of three states and a decision node. The initial state transitions to the **Setup** state when the call to execute a recipe is made by the event class. When this occurs, the system, via its user interface, shall prompt the user for an input of the ingredients that they have. The user input transitions the state machine to the next state, **Instantiation** upon entry in which the system evaluates the value of allChecked attribute of the Checklist object to be created by comparing the user input with the ingredients attribute of the corresponding recipe. Then the checklist object will be constructed accordingly. Once this is done the state machine directly transitions to the decision node where it splits into two paths, one connected to the final state if allChecked is true, and the other transitioning to the **Incomplete** state if it is false. Upon entry to this state, the user is given an error message informing them that the recipe they are attempting to execute requires ingredients they do not have.

Event



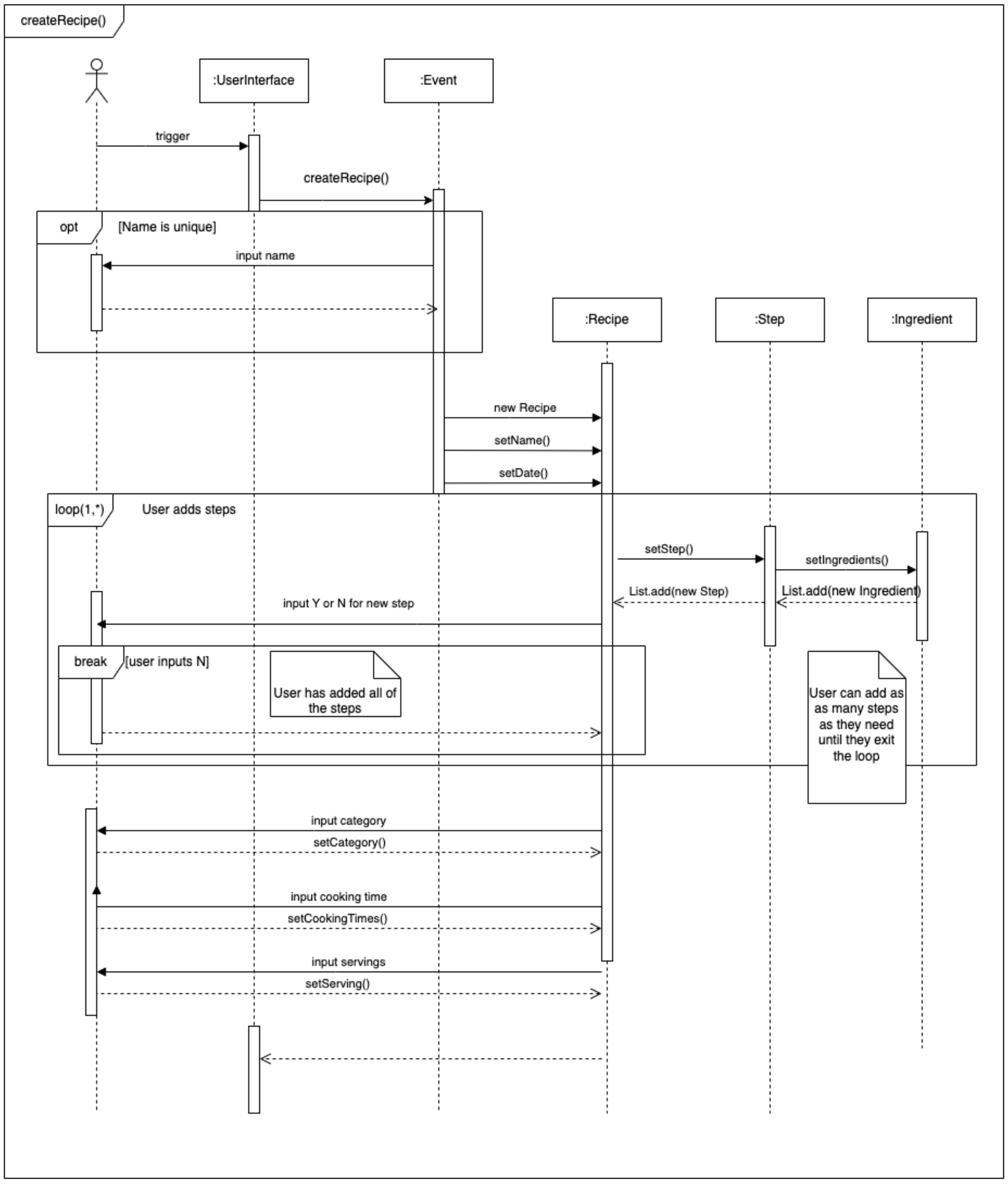
The Event class will act as a control point for the system. All the main functions that the system provides goes through an instance of this class. The state machine diagram of the Event class shows the different stages that an instance of the Event class goes through in order to fulfill the functionalities of the system.

The Event class first goes through the **Setup** state where an instance of the Event class is initialized after the user triggers an input through the Userinterface class. The state transition to **Create Recipe** will be straightforward if the user wants to create a new recipe where an instance of the Recipe class will be initialised. However, if the user wants to opt for other functionalities such as updating, deleting, and executing the recipe, then the Event object will first transition to **Load Recipe** state in which the JSON file of the particular recipe is loaded from an instance of the JSONparser class. After that, depending on the user's input, the Event object will transition to **Update Recipe**, **Delete Recipe**, and **Execute Recipe** states. If the Event object can not be initialized while in the Setup state, then the reason for happening so will be displayed. State transition to **Save Recipe** state will happen once the user commands save input after Update Recipe, or Create Recipe State. In the Save Recipe state, the JSON file of the recipe will be saved using JSONparser object.

Sequence diagrams

Author(s): Sean Hermes & Lucas Kim

Diagram 1: createRecipe()



The sequence diagram above follows the process that occurs when the user is creating a new recipe to add to their recipe book. To start this process the user must give the system the command in the userInterface class. The UI will then trigger the events class which will invoke the createRecipe() method.

The system will ask the user to input the name of the recipe in the form of a string. The name must be unique since the recipes are stored in JSON files with the names of the recipes, so there must not be two files of the same name.

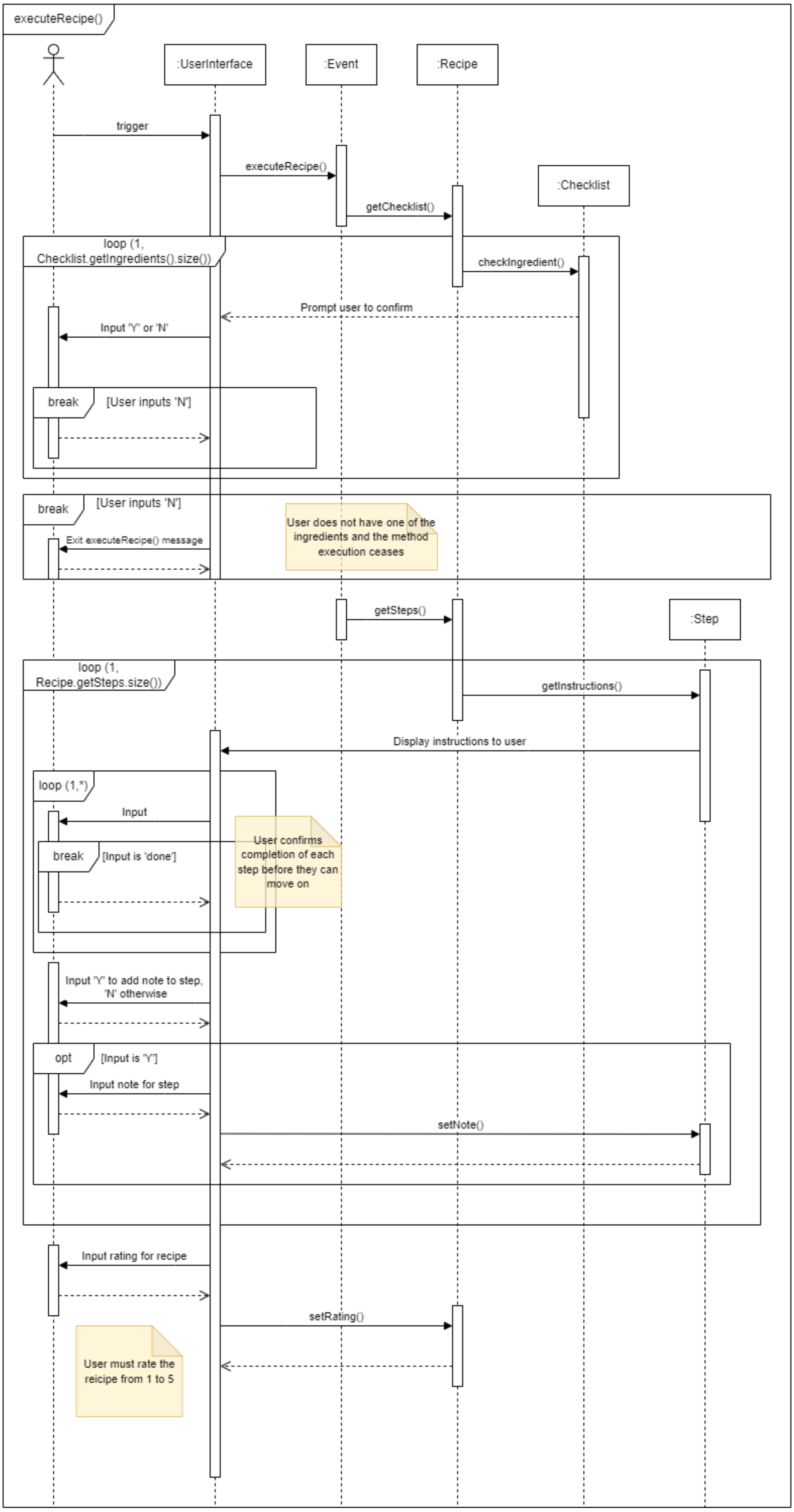
If the name is valid, the system creates a new Recipe object with the given name, and adds the date added to the given attribute.

From the new Recipe class, the setSteps() method will be called in the Steps and will call the setIngredients() method in the ingredients class too. These methods will allow the user to input all of the information and instructions that they need for a successful recipe. This process is inside of a loop to allow the user to add as many steps as they feel is needed. After each step is created and added to the Recipes List<steps> the system asks the user a yes or no to add a new step, if the user says no the loop is broken and the process continues.

After the steps are created, the system needs more information from the user so the recipe calls setCategory() which will ask the user if they want to attach a category to the recipe. Next the user will input the amount of time the recipe should take in the setCookingTime(). Lastly the user will input the amount of servings the recipe is designed for in the setServings() method.

All of the needed information has been applied to the newly created recipe, so the sequence is returned back to the user interface to await further instruction from the user.

Diagram 2: executeRecipe()


executeRecipe()
:UserInterface
:Event
:Recipe
:Checklist
:Step
trigger
executeRecipe()
getChecklist()
loop (1, Checklist.getIngredients().size())
checkIngredient()
Prompt user to confirm
Input 'Y' or 'N'
break [User inputs 'N']
break [User inputs 'N']
Exit executeRecipe() message
User does not have one of the ingredients and the method execution ceases
getSteps()
loop (1, Recipe.getSteps.size())
getInstructions()
Display instructions to user
loop (1,*)
Input
break [Input is 'done']
User confirms completion of each step before they can move on
Input 'Y' to add note to step, 'N' otherwise
opt [Input is 'Y']
Input note for step
setNote()
Input rating for recipe
setRating()
User must rate the reicipe from 1 to 5

This sequence diagram describes the process that occurs when the user chooses to execute a **Recipe**. There are 2 main loops in this process: the **Ingredient**s **Checklist** loop and the **Step** execution loop.

To begin, the user will choose to execute a **Recipe** by giving a command through the **UserInterface**. Then, the **UI** will tell the **Event** class to call the method *executeRecipe()*. Following this, the **Event** class will tell the **Recipe** to get its **Checklist** attribute which will then be told to call its *checkIngredient()* method. This is the beginning of the first loop.

The first loop is for the user to check if they have each **Ingredient** in the **Recipe**. The **Checklist** will tell the **UI** to ask the user to confirm if they have each **Ingredient** by entering either 'Y' or 'N'. If the user enters 'Y' for all the **Ingredient**s, the sequence then moves on from the loop. If the user enters 'N' for an **Ingredient**, then the loop will break and the method execution will cease as well.

Next, the **Recipe** will access its list of **Step**s which leads into the second main loop. The **Recipe** will get the instructions String from the **Step** using the *getInstructions()* method and will then return the String to the **UI**, which will then display it to the user. The user will then be prompted to confirm whether or not they are finished with the **Step** by entering the String 'done'. If anything else is entered, they will be prompted once again to confirm if they are done. Following this, the user will then be asked if they would like to write a note for the **Step**. If they enter 'Y', then they will be asked to enter a *note*, which will be saved as a String in the **Step** using the method *setNote()*. If the user chooses to not write a *note* by entering 'N', then the iteration of the loop is complete and the loop moves on to the next **Step**.

Once all **Step**s are complete, the user will finally be asked by the **UI** to enter a *rating* for the **Recipe**. The *rating* will be saved in the **Recipe** using the *setRating()* method.

This concludes the sequence of the *executeRecipe()* method, and so the system returns back to the main program loop to wait for the user's next decision.

Maximum number of pages for this section: 4

Time logs

Sanskar Shrestha	Slideshow presentation to the mentor and feedback	2	1
Bora Tarlan	Slideshow presentation to the mentor and feedback	2	1
Sean Hermes	Slideshow presentation to the mentor and feedback	2	1
Lucas Kim	Slideshow presentation to the mentor and feedback	2	1
Sanskar Shrestha	Self study on UML diagrams	3	1
Bora Tarlan	Self study on UML diagrams	3	1
Sean Hermes	Self study on UML diagrams	3	1
Lucas Kim	Self study on UML diagrams	3	1
Sanskar Shrestha	Writing up and and creating diagrams for Assignment 2	5	4
Bora Tarlan	Writing up and and creating diagrams for Assignment 2	5	4
Sean Hermes	Writing up and and creating diagrams for Assignment 2	5	4
Lucas Kim	Writing up and and creating diagrams for Assignment 2	5	4