REPORT 1 DESIGN DOCUMENT

Jingmian Zhang 213568993

Yichun Dai 213965488

2016 October

Table of Content

1 Introduction

This is report 1 of the course EECS 3311 which is created according to the specification in the report 1 specification as well as the program text given. In report 1, we implemented a program from a JSP description of its input and output (the mildred_mordred merge), designed a food company system which is comprised of multiple classes, and drew a BON diagram for the food company system. Through the process of this report, we have learned about the features and properties of Eiffel language and object-oriented concept considerably.

2 Mildred-Mordred design document

2.1 input JSP

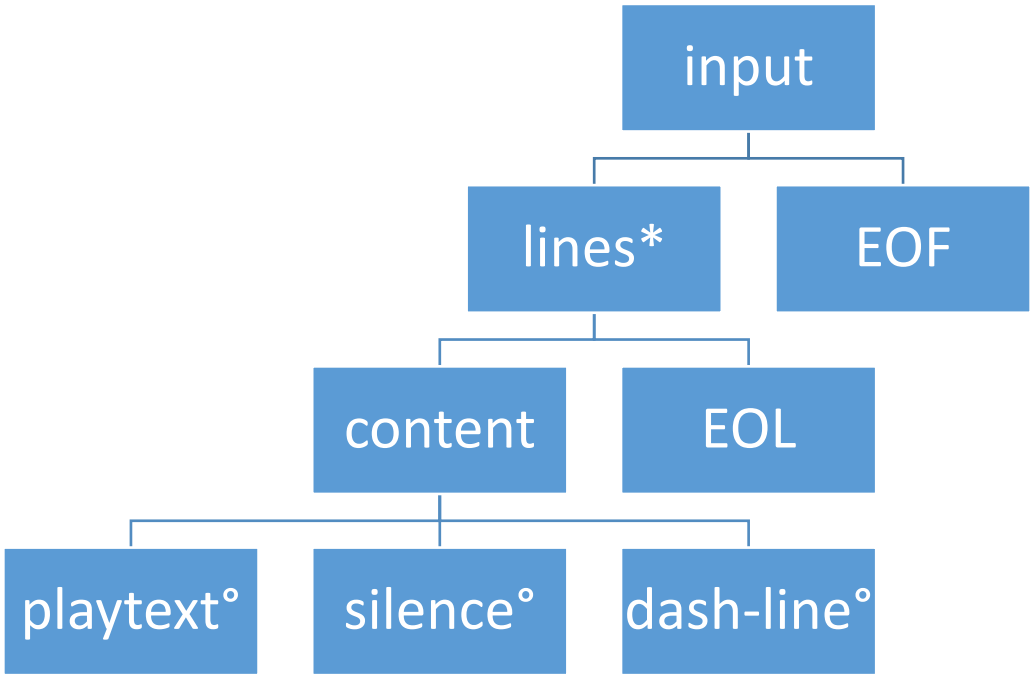


Figure1:input JSP

Explanation:

Figure1 shows the structures of the input files mildred or mordred, which are the same. The file contains zero or multiple lines, then ending with EOF. Each line has content, which is comprised of either playtext(words), silence(empty line), or dash-line(a line that only contains the dash,"-" character with at least MinDashCount dashes). Then an EOL symbol is followed by the content. This is basically the structure of the input files.

2.2 ouput JSP

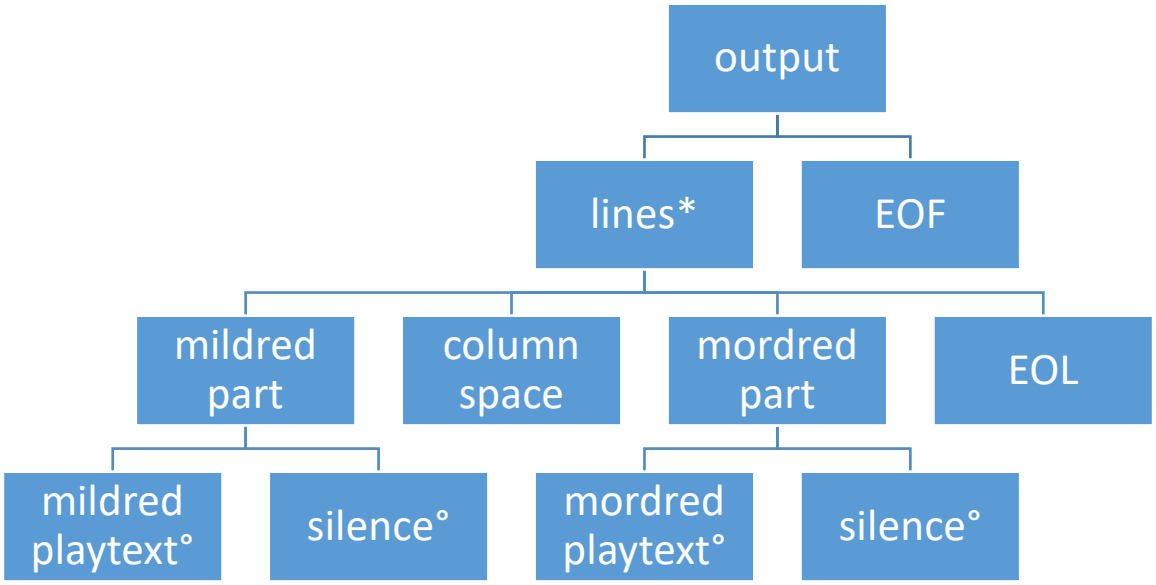


Figure2: output JSP

Explanation:

Figure2 shows the structures of the output file playtext, which contains zero or multiple lines, then ending with EOF. Each line consists of four parts, which are mildred part (input from mildred text), column space (space between columns), mordred part (input from mordred text) and EOL symbol. For mildred part and mordred part, the content of each of them can either be playtext or silence (empty line). The dash-lines in both mildred part and mordred part are first recognized(is_min_dash routine in program) and then aligned (print_mordred_line_until_dash and print_mildred_line_until_dash routtines) in the merge method of the program. The dash-lines are not to be printed in the output. In conclusion, the above JSP diagram depicts the output of the program.

3 Food Company design document

Proof of concept

1. Client requirements: This system is mainly designed to keep track of the food company's inventory and its contacts. More specifically, this system should be able to find the food which will expire at a certain date, find a specific food's producer, check customers' credit status and find the customer producers' contacts.

2. Client's input:

a. Client will try to add customer into system or remove customer from system

b. Client will try to add customer into system or remove customer from system

c. Client will input a certain date to find the food which will expire at this date

d. Client will input the food to find the producer of this food

e. Client will use a command to get all the customers who have good credit status

3. To deal with the clients' requirements, first of all, we created 4 classes to store information: Class PRODUCER – which stores producers' names and the food they produce, Class FOOD – which stores foods' names and expire dates, Class CUSTOMER --- which stores customers' names and credit points and Class CONTACTS --- which stores a specific customer's or producer's address. In root class START we have designed adder and remover for customer list and producer list – which satisfies Client's input a and b, method "list_customer_with_good_credit" to get all customers who have good credit status – which satisfies Client's input e, method expire_at_certain_date() to find the food which will expire at the date client inputs --- which satisfies Client's input c. And method producer_offers_specific() to find the food that client input's producer – which satisfies Client's input d.

Food company system ADT

1. Class START

a. OBJECT:

Hidden: credit_good

Exported Oen: customer:LINKED_LIST[CUSTOMER]

producer:SET[PRODUCER]

customer_with_good_credit:LINKED_LIST[CUSTOMER]

credit_good:INTEGER = 50

specific_producer:LINKED_LIST[PRODUCER]

food_expire_at_date:LINKED_LIST[FOOD]

inventory:LINKED_LIST[FOOD]

contacts:LINKED_LIST[CONTACT]

b. Operation:

list_customer_with_good_credit

expire_at_certain_date(d:DATE)

producer_offers_specific(f:FOOD)

add_customer(c:CUSTOMER)

remove_customer(c:CUSTOMER)

add_producer(p:PRODUCER)

remover_producer(p:PRODUCER)

2. Class FOOD

a. Object:

Exported:

food_name: STRING

expire_date: DATE

b. Opertion:

make(name:STRING; expire: DATE)

3. Class Producer

a. Object:

Exported:

company_name: STRING

food_produce:LINKED_LIST[FOOD]

b. Operation:

make(name:STRING)

add(food:FOOD) --- food_produce adder

remove(food:FOOD) --- food_produce remover

search(food:FOOD):BOOLEAN --- search for specific food

4. Class CUSTOMER

a. Object:

Exported:

customer_name:STRING

customer_credit:INTEGER

b. Operation:

make(name:STRING)

set_credit(c:INTEGER) --- credit settet

get_credit:INTEGER --- credit getter

5. Class CONTACT

a. Object:

Exported:

customer_contact: detachable CUSTOMER

producer_contact:detachable PRODUCER

address:STRING

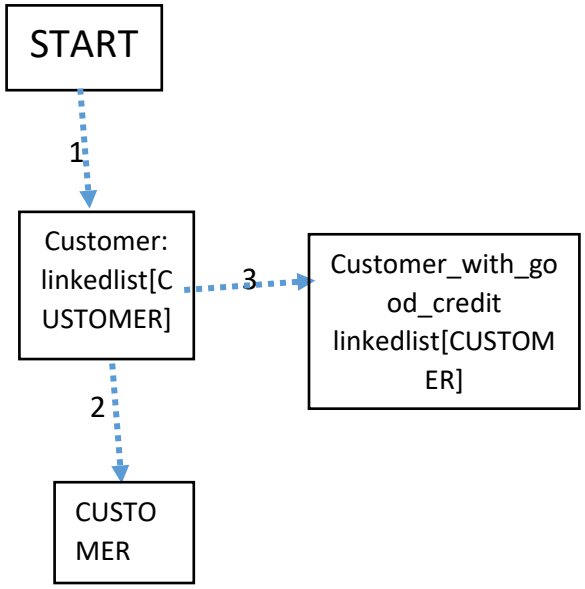
Hidden: emptystring: STRING=""

b. Operation:

set_customer(c:CUSTOMER) --- customer's contact setter

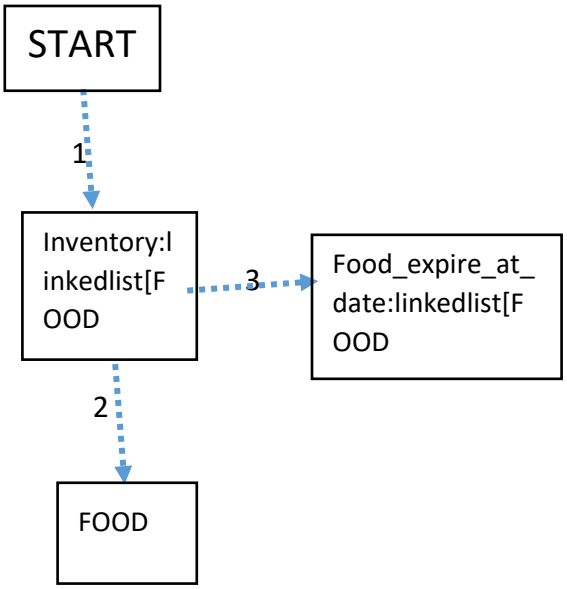
set_producer(p:PRODUCER) --- producer's contact setter

set_address(ad:STRING) --- customer or producer's address setter



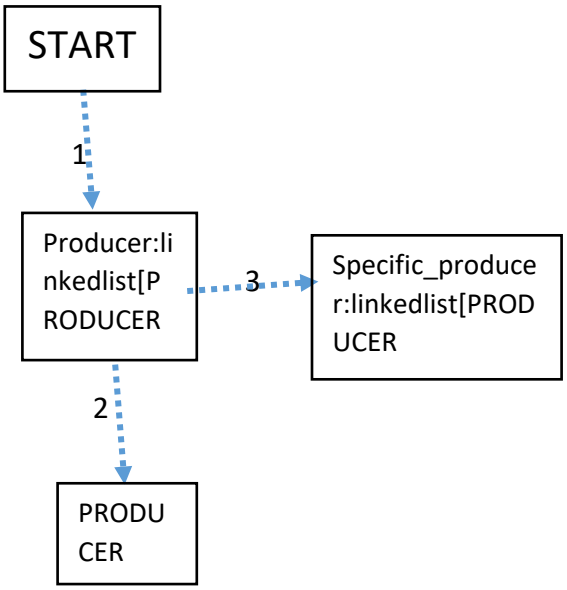
Scenario 1: get list of customers with good credit

1. Create customer linkedlist
2. Traverse and get customer_credit in CUSTOMER.
3. compare customer_credit in CUSTOMER for each instance to see if they are greater than good_credit, if true, than put them in customer_with_good_credit linkedlist.



Scenario 2: get food that expires at a certain date

1. Create Inventory:linkedlist.
2. Traverse and get food objects in FOOD with a particular expiry date.
3. Put result in 2 in food_expire_at_date linkedlist.



Scenario 3: get list of producers that offer specific food items.

1. Create Producer:linkedlist.
2. Traverse and search for produce that produces specific food.
3. Put the producer in the specific_producer linkedlist.

4 Other Items

As the report being the first report of implementing in Eiffel language, lots of problems and dilemmas occur in specifically in coding section. For example, while doing mildred-mordred merge, we were not able to find and utilize methods such as put_string, count and item in STRING_8 class until one of us perused through the library carefully. As a result, some usages of routines in the library were still unclear and tentative in our attempt to solve the problem, especially the methods called from linked_list class in the library. But we still managed to produce clear and functional codes after making effort to try out everything. In conclusion, this lab has acquainted us with some basic knowledge of Eiffel and we still need to work harder to have a better grasp of this programming language.

5 Appendix

Overall layout and summary of the design document—Jingmian Zhang

Mildred-Mordred part of the design document—Jingmian Zhang

Mildred-Mordred implementation, including appropriate comments—Jingmian Zhang,Yichun Dai

Food company part of the design document—Jingmian Zhang, Yichun Dai

Programming and comments the food company program text—Yichun Dai