Date: 10/26/21
Author: Freeman Smith
Reviewers: Kevin King

Introduction:
The individual devices in the Store need a controlled to handle the data that comes in from sensors and appliances as well as outputting commands to the appliances. Various events will be recognized by the devices and handled by the controller service and then an action will be taken based off the stimulus of the event.

Overview:

This design specifies the implementation of the Controller Service. The Controller Service manages sensors and appliances within the store. When an event happens in the store, for example a fire, spilled milk, or a customer question, that event needs to be recognized by the controller. Events are recognized from the input of sensors or appliances that recognize certain stimuli. When a known stimulus is recognized and event is created from one of the known options. An event is will have an action associated with it and sometimes that action will involve a command to one of the appliances. For example, spilling milk will trigger and event which invokes a command to a robot to clean up the milk. Both the Command Pattern and the Observer Pattern will be utilized for interaction between our controller and our model services.

Requirements:

The Controller Service will handle Sensors and Appliances. For both Sensors and Appliances, the Controller service will utilize the Observer Pattern to handle input from the devices via the Store Model Service. The Command Pattern will handle output commands and actions triggered by the events recognized by the Controller Service. There is a limited number of events that can be recognized by the Controller Service and they have specific actions associated with them. These are the events:

Emergency
This is recognized by a camera and can be either:

- Fire
- Flood
- Earthquake
- Armed Intruder

The stimulus for this will be “Emergency <emergency> in <aisle>.” The action output will be:

1. Open all turnstiles
2. Announce “There is a <emergency> in <aisle>, please leave <store> immediately”
3. Robot 1 will receive a stimulus to “address <emergency> in <aisle>”
4. The other robots will receive stimulus to “assist customers leaving the <store>”

Basket Event
Recognized by a camera.

The stimulus for this will be “Customer <customer>(adds|removes)<product> from<aisle:shelf>”. The action output will be:

1. Add/remove the product to/from <customer> basket
2. Remove/Add product from<aisle:shelf>
3. robot: perform task restock for <aisle:shelf> and <product>

Cleaning Event
Recognized by a camera.

The stimulus for this will be “<product> on floor<store:aisle>”. The action output will be Robot: "clean up <product> in <aisle>".

Broken Glass
Recognized by a microphone.

The stimulus for this will be “sound of breaking glass in <aisle>”. The action output will be Robot: "clean up broken glass in <aisle>".

Missing person
Recognized by a microphone.

The stimulus for this will be “can you help me find<customer name>”. The action output will be “locate customer <name> speaker: <name> is in aisle <aisle>”.

Customer Seen
Recognized by a camera.

The stimulus for this will be “Customer enter <aisle>”. The action output will be "Update customer location <aisle>".

Fetch product
Recognized by a microphone.

The stimulus for this will be “<customer> says: Please get me <number> of <product>”. The action output will be “robot command: fetch <number> of <product> from <aisle> and <shelf> and bring to customer <customer> in aisle <customer_location>”.

Check account Balance
Recognized by a microphone.

The stimulus for this will be “Customer <customer> says "What is the total basket value?"”. The action output will be:

1. compute the value of items in the basket
2. check account balance
3. speaker: "total value of basket items is <value> which is (more | less) than you account balance of <balance>

Assist customer to car
Recognized by a turnstile.

The stimulus for this will be “checkout: total weight of products in basket exceeds 10 lbs”. The action output will be “request robot to assist customer <customer> to car”.

Checkout
Recognized by a turnstile.

The stimulus for this will be “customer <customer> approaches turnstile”. The action output will be:

1. Identify customer<customer>
2. compute the total cost of items in the basket
3. create transaction
4. submit the transaction to the blockchain
5. open turnstile
6. goodbye message :"goodbye <customer_name>, thanks for shopping at <store_name>!"

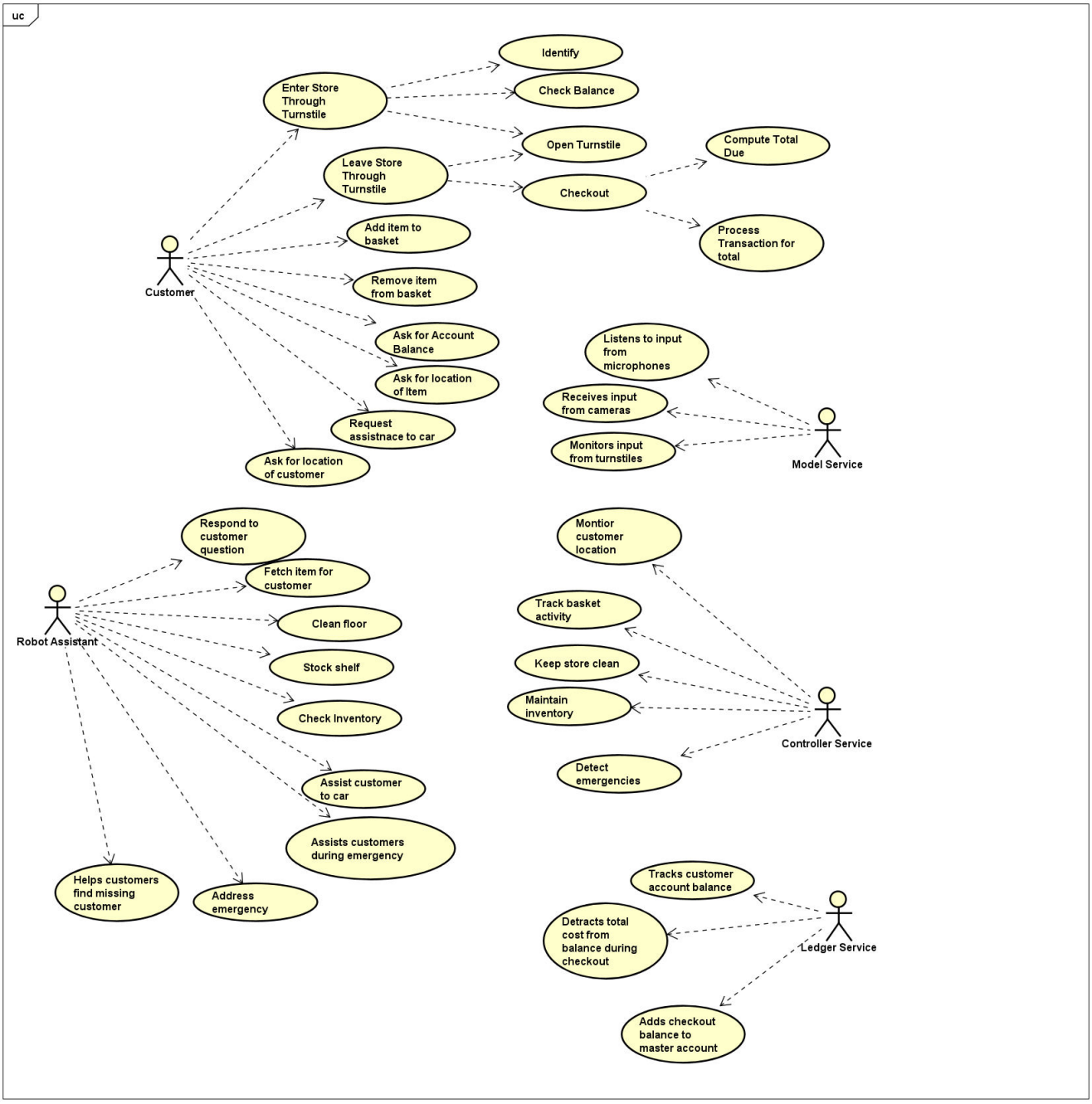
Enter store
Recognized by a turnstile.

The stimulus for this will be “<customer> waiting to enter at the turnstile <turnstile>”. The action output will be:

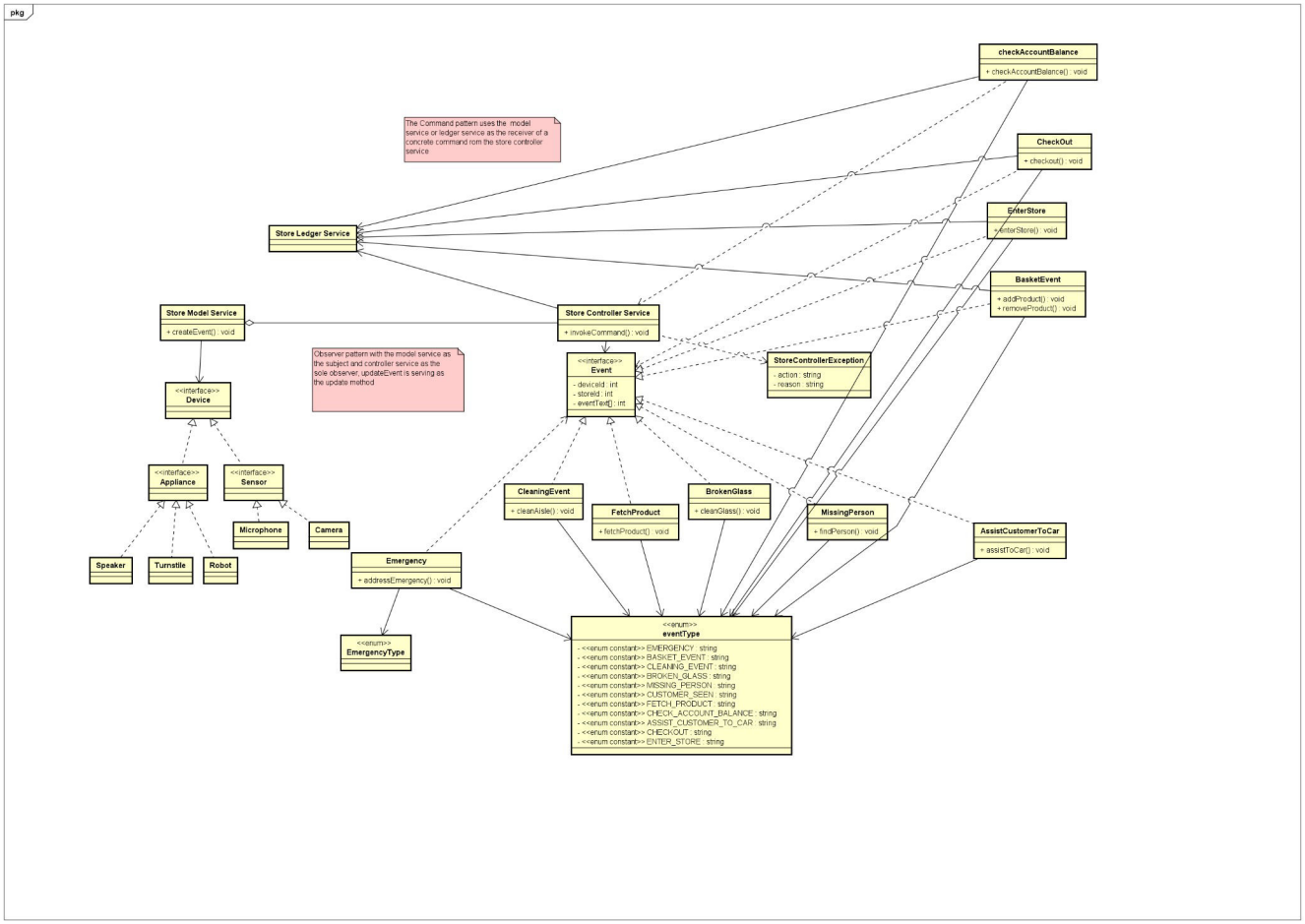
1. Lookup customer<customer>
2. Check for positive account balance
3. Assign Customer a basket
4. open turnstile <turnstile>
5. welcome message "Hello <customer_name>, welcome to <store name>!"

Command API
The Controller Service supports a Command Line Interface (CLI) for configuring stores from the overall Store 247 model. The commands can be listed in a file to provide a configuration script. The CLI should use the service interface to implement the commands. The commands that denote events recognized by the devices will emitted to the Model Service, which will be the subject of an observer pattern with the Command Service as the

Use Case Diagram:



Class Diagram:



The class diagram shows the simple use of the Observer and Command pattern. The store model service is the subject, it receives input from the devices and the notifies the observer, the store controller service. The controller service then has an Invoker command that creates a a custom command based on the specific event. Directions for responding to the stimulus of the event are known by the controller service, the model service only passes along the stimulus.

Class Dictionary

Store Model Service

Only one method has been added to the store model service. This is the subject of the observer pattern and the createEvent method is acting to notify the controller service via the observer pattern.

Methods

Methods Name	Signature	Description
Methods from last assignment…	…	...
createEvent	(event: Event) : void	Creates an event and updates the observer, the controller pattern.

Store Controller Service
Observes the Store Model Service for events and then invokes commands responding to those events

Methods

Methods Name	Signature	Description
invokeCommand	(event: Event) : void	Give an event being observed from the Store Model Service, this will invoke a command

Assist Customer to Car
Invokes a command to a robot to assist <customer id> to car

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
assistCustomer	(customerId: String) : void	Commands a robot to assist that customer to their car

Fetch Product
Fetch a product for a customer. Check to make sure there is enough inventory and if there isn’t restock that inventory.

Properties

Property Name	Type	Description
deviceId	String	The device that observed the

		event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
fetchProduct	(String[] : eventText): void	Fetches product, checks inventory, restocks if necessary, brings the product to the customer

Check Account Balance

When requested by a customer, speaker announces the balance of a customers account and whether that is more or less than what is in their cart

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description

checkAccountBalance	(String[] : eventText): void	Checks account balance, announces it and whether it's more or less than the sum of the basket

Checkout
When a customer exits through a turnstile, identify the customer, compute the value of the basket, create a transaction, submit it to the block chain, open the turnstile and say goodbye to the customer

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
checkout	(String[] : eventText): void	Identify the customer, compute the value of the basket, create a transaction, submit it to the block chain, open the turnstile and say goodbye to the customer

Basket Event
Add or remove a product to a customer basket. If necessary, command a robot to restock.

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace

eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
removeBasketItem	(String[] : eventText, int count): void	Remove item from customer basket
addBasketItem	(String[] : eventText, int count): void	Adds item to customer baskett, checks inventory, restocks if necessary,

Broken Glass

Invokes a command to a robot clean glass in specified aisle

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
cleanGlass	(aisle: String) : void	Commands a robot to clean broken glass in that aisle

Cleaning Event
Invokes a command clean a spill in specified aisle

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
cleanSpill	(product: String, aisle: String) : void	Commands a robot to clean spilled product in that aisle

Emergency
Announces an emergency to the customers, has a robot address it directly while other robots assist customers out of the store.

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
addressEmergency	(emergencyType: String, aisle: String, storeId: String) : void	Announce an emergency, address it and assist customers.

Missing Person
Locate a person and announce their location to the customer

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
locatePerson	(customerId: String) : void	Locate a person and announce their location to the customer

Customer Seen
Updates the location of the customer in the store model service

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to

		each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
updateCustomerLocation	(customerId: String, storeId: String, aisle: String) : void	Updates the location

Enter Store

When a customer enters through a turnstile, identify the customer, compute the value of the basket, create a transaction, submit it to the block chain, open the turnstile and say goodbye to the customer

Properties

Property Name	Type	Description
deviceId	String	The device that observed the event
storeId	String	The store where the event took polace
eventText	String[]	The string array describing details of that event, specific to each event

Associations

Association Name	Type	Description
type	EventType	An enum of different types of events observed by the observer pattern

Methods

Methods Name	Signature	Description
enterStore	(String[] : eventText): void	Identify the customer, check that they have a positive blaance, assign a basket, welcome the customer

Controller Service Exception
Handles exception from the store model service and stores the action and reason for the exception.

Properties

Property Name	Type	Description
action	String	What threw the exception
reason	String	Why the action caused the exception

Command Line Processor

Command Syntax:
Store Commands

Store Model Service Sample Script
Store Commands
definestore <identifier> name <name> address <address>
define-store store_123 name Eataly address "800 Boylston St, Boston, MA 02199"
show store <identifier>
show-store store_123
#Aisle Commands
Define an aisle within the store
define aisle <store_id>:<aisle_number> name <name> description <description> location (floor | store_room)
define-aisle store_123:aisle_A1 name AISLE_A1 description AISLE_A1_desc location store_room
define-aisle store_123:aisle_A2 name AISLE_A2 description AISLE_A2_desc location floor
define-aisle store_123:aisle_A3 name AISLE_A3 description AISLE_A3_desc location floor
define-aisle store_123:aisle_B1 name AISLE_B1 description AISLE_B1_desc location store_room
define-aisle store_123:aisle_B2 name AISLE_B2 description AISLE_B2_desc location floor
define-aisle store_123:aisle_B3 name AISLE_B3 description AISLE_B3_desc location floor
#Show the details of the aisle, including the name, description and list of shelves.
show aisle <store_id>[:<aisle_number>]
show-aisle store_123:aisle_A1
show-aisle store_123:aisle_A2
show-aisle store_123:aisle_B2
#Shelf Commands
Define a new shelf within the store
define shelf <store_id>:<aisle_number>:<shelf_id> name <name> level (high | medium | low) description <description> [temperature (frozen | refrigerated | ambient | warm | hot)]
define-shelf store_123:aisle_A1:shelf_q1 name Shelf_Q1 level high description Shelf_Q1_Desc temperature frozen
define-shelf store_123:aisle_A1:shelf_q2 name Shelf_Q2 level medium description Shelf_Q2_Desc temperature ambient
define-shelf store_123:aisle_A1:shelf_q3 name Shelf_Q3 level low description Shelf_Q3_Desc temperature refrigerated
define-shelf store_123:aisle_A1:shelf_q4 name Shelf_Q4 level low description Shelf_Q4_Desc temperature warm

define-shelf store_123:aisle_A1:shelf_q5 name Shelf_Q5 level medium description Shelf_Q5_Desc temperature hot
define-shelf store_123:aisle_A2:shelf_q1 name Shelf_Q1 level high description Shelf_Q1_Desc temperature frozen
define-shelf store_123:aisle_A2:shelf_q2 name Shelf_Q2 level medium description Shelf_Q2_Desc temperature ambient
define-shelf store_123:aisle_A2:shelf_q3 name Shelf_Q3 level low description Shelf_Q3_Desc temperature refrigerated
define-shelf store_123:aisle_A2:shelf_q4 name Shelf_Q4 level low description Shelf_Q4_Desc temperature warm
define-shelf store_123:aisle_A2:shelf_q5 name Shelf_Q5 level medium description Shelf_Q5_Desc temperature hot
define-shelf store_123:aisle_A3:shelf_q1 name Shelf_Q1 level high description Shelf_Q1_Desc temperature frozen
define-shelf store_123:aisle_A3:shelf_q2 name Shelf_Q2 level medium description Shelf_Q2_Desc temperature ambient
define-shelf store_123:aisle_A3:shelf_q3 name Shelf_Q3 level low description Shelf_Q3_Desc temperature refrigerated
define-shelf store_123:aisle_A3:shelf_q4 name Shelf_Q4 level low description Shelf_Q4_Desc temperature warm
define-shelf store_123:aisle_A3:shelf_q5 name Shelf_Q5 level medium description Shelf_Q5_Desc temperature hot
define-shelf store_123:aisle_B1:shelf_q1 name Shelf_Q1 level high description Shelf_Q1_Desc temperature frozen
define-shelf store_123:aisle_B1:shelf_q2 name Shelf_Q2 level medium description Shelf_Q2_Desc temperature ambient
define-shelf store_123:aisle_B1:shelf_q3 name Shelf_Q3 level low description Shelf_Q3_Desc temperature refrigerated
define-shelf store_123:aisle_B1:shelf_q4 name Shelf_Q4 level low description Shelf_Q4_Desc temperature warm
define-shelf store_123:aisle_B1:shelf_q5 name Shelf_Q5 level medium description Shelf_Q5_Desc temperature hot
define-shelf store_123:aisle_B2:shelf_q1 name Shelf_Q1 level high description Shelf_Q1_Desc temperature frozen
define-shelf store_123:aisle_B2:shelf_q2 name Shelf_Q2 level medium description Shelf_Q2_Desc temperature ambient
define-shelf store_123:aisle_B2:shelf_q3 name Shelf_Q3 level low description Shelf_Q3_Desc temperature refrigerated
define-shelf store_123:aisle_B2:shelf_q4 name Shelf_Q4 level low description Shelf_Q4_aislDesc temperature warm
define-shelf store_123:aisle_B2:shelf_q5 name Shelf_Q5 level medium description Shelf_Q5_Desc temperature hot
define-shelf store_123:aisle_B3:shelf_q1 name Shelf_Q1 level high description Shelf_Q1_Desc temperature frozen
define-shelf store_123:aisle_B3:shelf_q2 name Shelf_Q2 level medium description Shelf_Q2_Desc temperature ambient
define-shelf store_123:aisle_B3:shelf_q3 name Shelf_Q3 level low description Shelf_Q3_Desc temperature refrigerated

define-shelf store_123:aisle_B3:shelf_q4 name Shelf_Q4 level low description Shelf_Q4_Desc temperature warm
define-shelf store_123:aisle_B3:shelf_q5 name Shelf_Q5 level medium description Shelf_Q5_Desc temperature hot
Show the details of the shelf including id, name, level, description and temperature
show shelf <store_id>[:<aisle_number>[:<shelf_id>]]
show-shelf store_123:aisle_A2:shelf_q1
show-shelf store_123:aisle_B1:shelf_q5
Product Commands
Define a new product
define product <product_id> name <name> description <description> size <size> category <category> unit_price <unit_price> [temperature (frozen | refrigerated | ambient | warm | hot)]
define-product prod10 name bournvita description bournvita size 250g category Food unit_price 2 temperature ambient
define-product prod11 name tea description "green tea" size 500g category Food unit_price 1 temperature ambient
define-product prod12 name coffee description "premium coffee" size 100g category Food unit_price 3 temperature refrigerated
define-product prod13 name goggles description "eye wear" size 3 category Men_accessories unit_price 5 temperature ambient
define-product prod14 name sun_glass description "eye wear" size 3 category Fashion unit_price 7 temperature ambient
Show the details of the product
show product <product_id>
show-product prod10
#Inventory Commands
Define a new inventory item within the store
define inventory <inventory_id> location <store_id>:<aisle_number>:<shelf_id> capacity <capacity> count <count> product <product_id>
define-inventory inv_u21 location store_123:aisle_A1:shelf_q1 capacity 1500 count 1000 product prod10
define-inventory inv_u22 location store_123:aisle_A1:shelf_q2 capacity 1500 count 1000 product prod11
define-inventory inv_u23 location store_123:aisle_B2:shelf_q1 capacity 500 count 200 product prod11
define-inventory inv_u24 location store_123:aisle_B2:shelf_q2 capacity 500 count 200 product prod10
define-inventory inv_u25 location store_123:aisle_A2:shelf_q1 capacity 200 count 100 product prod10
define-inventory inv_u26 location store_123:aisle_A2:shelf_q3 capacity 300 count 100 product prod12
Show the details of the inventory
show inventory <inventory_id>
show-inventory inv_u24
show-inventory inv_u21
Update the inventory count, count must >= 0 and <= capacity
update inventory <inventory_id> update_count <increment or decrement>
update-inventory inv_u24 update_count 4

update-inventory inv_u21 update_count 7
#Customer Commands
Define a new customer
define customer <customer_id> first_name <first_name> last_name <last_name> type (registered|guest) email_address <email> account <account_address>
define-customer cust_AB first_name JSON last_name WALLACE type guest email_address json.wallace@ymail.com account json
define-customer cust_21 first_name BILL last_name ROSE type registered email_address bill.rose@gmail.com account bill
define-customer cust_22 first_name MARY last_name KELVIN type registered email_address mary.kevin@yahoomail.com account mary
define-customer cust_E2 first_name JANE last_name HAWK type guest email_address silver.hawk@rocketmail.com account jane
define-customer cust_23 first_name MEGON last_name FOX type guest email_address megonfox@testmail.com account megon
define-customer cust_24 first_name MARIA last_name WILIAMSON type registered email_address maria4567@ymail.com account maria
define-customer cust_S2 first_name SALINA last_name GOMEZ type registered email_address salina@gmail.com account salina
Update the location of a customer
update customer <customer_id> location <store:aisle>
update-customer cust_S2 location store_123:aisle_B2
update-customer cust_21 location store_123:aisle_A2
update-customer cust_22 location store_123:aisle_A2
Show the details of the customer
show customer <customer_id>
show-customer cust_21
show-customer cust_S2
Basket Commands
#define basket <basket_id>
define-basket b1
define-basket b2
define-basket b3
define-basket b4
define-basket b5
Associate basket with Customer
#assign basket <basket_id> customer <customer_id>
assign-basket b1 customer cust_21
assign-basket b2 customer cust_S2
assign-basket b3 customer cust_22
Get basket_id associated with the customer, create new basket if the customer does not already have a basket associated.
get_customer_basket <customer_id>
get-customer-basket cust_21
get-customer-basket cust_S2
get-customer-basket cust_22
Add a product item to a basket
add_basket_item <basket_id> product <product_id> item_count <count>
add-basket-item b1 product prod10 item_count 4

```
add-basket-item b3 product prod11 item_count 2
add-basket-item b3 product prod12 item_count 7
# Remove a product item from a basket
# remove_basket_item <basket_id> product <product_id> item_count <count>
remove-basket-item b1 product prod10 item_count 1
remove-basket-item b3 product prod12 item_count 5
# Clear the contents of the basket and remove the customer association
# clear_basket <basket_id>
# Get the list of product items in the basket, include the product_id and count
# Show basket_items <basket_id>
Show-basket-items b3 # it is empty
Show-basket-items b1
Show-basket-items b5 #it is not assigned yet
#Sensor Commands
# Define device of type sensor
# define-device <device_id> name <name> type (microphone|camera) location <store>:<aisle>
define-device mic_A1 name MicrophoneA1 type microphone location store_123:aisle_A2
define-device cam_A1 name CameraA1 type camera location store_123:aisle_A2
define-device cam_A2 name CameraA2 type camera location store_123:aisle_A1
define-device cam_B2 name CameraB1 type camera location store_123:aisle_B2
# Show device details
# show-device <device_id>
show-device cam_A1
show-device mic_A1
show-device cam_A2
# Create a sensor event, this simulates a sensor event
# create_event <device_id> event <event>
#create-event cam_A1 event item_added_to_basket b1 prod10
#create-event cam_A1 event item_added_to_basket b1 prod11
#create-event mic_A1 event custmer_asked_question cust_S2 "where can I find the milk?"
Appliance Commands
# Define device of type appliance
# define device <device_id> name <name> type (speaker | robot | turnstile) location
<store>:<aisle>
define-device rob_1 name ROBOT_1 type robot location store_123:aisle_A1
define-device rob_2 name ROBOT_2 type robot location store_123:aisle_A2
define-device spk_1 name SPEAKER_1 type speaker location store_123:aisle_A1
define-device spk_2 name SPEAKER_2 type speaker location store_123:aisle_A2
define-device turn_a1 name TURNSTILE_A1 type turnstile location store_123:aisle_A2
define-device turn_a2 name TURNSTILE_A2 type turnstile location store_123:aisle_A2
define-device turn_a3 name TURNSTILE_A3 type turnstile location store_123:aisle_A2
# Show appliance device details
show-device turn_a1
show-device rob_2


# Create enter-store events for 3 customers
create-event turn_a1 store-location store_123 event "enter-store cust_21 turn_a2 store_123:aisle_A2"
create-event turn_a2 store-location store_123 event "enter-store cust_22 turn_a2 store_123:aisle_A2"
```

#########
#this demonstrates controller exception handling
create-event turn_a2 store-location store_123 event "enter-store cust_E2 turn_a2 store_123:aisle_A2"
#########

Create a basket-event event (leaving 9 out of the 11 milk products on the shelf)
create-event cam_A1 store-location store_123 event "basket-event cust_22 prod10 inv_u24 store_123:aisle_B2:shelf_q2 2"

Create a fetch-product event - expect a command to 1 robot to fetch the product
Created this event to demonstrate command for fetch product with restocking
create-event mic_A1 store-location store_123 event "fetch-product cust_21 prod10 inv_u24 store_123:aisle_B2:shelf_q2 5"

Created this event to demonstrate command for fetch product with restocking
create-event mic_A1 store-location store_123 event "fetch-product cust_21 prod10 inv_u24 store_123:aisle_B2:shelf_q2 300"

Create an emergency event - all turnstiles open, 1 attends to emergency and the rest help
customers, while speakers announce for customers to leave store
create-event cam_A1 store-location store_123 event "emergency ARMED_INTRUDER store_123:aisle_A2"

Show the details of the customers before updating location
show-customer cust_22

Create a customer-seen event for CUSTOMER00003 - location should change to {store_123 aisle A2}
create-event cam_A1 store-location store_123 event "customer-seen cust_22 store_123:aisle_A2"
show-customer cust_22

Create a broken-glass event - expect a robot to be assigned a new command to clean up glass
create-event cam_A1 store-location store_123 event "broken-glass store_123:aisle_A2"

Create a product-spill event - expect a robot to be assigned a command to clean the milk
create-event cam_A1 store-location store_123 event "product-spill store_123:aisle_A2 prod11"

Create a missing-person event using a microphone - expect that the missing customer is in aisle 1
create-event mic_A1 store-location store_123 event "missing-person store_123 cust_22"

Create a check-acc-bal event using a microphone - expect only one speaker to give response - total cost = 47
create-event mic_A1 store-location store_123 event "check-acc-bal cust_22 store_123:aisle_A2"

Create an assist-customer event using a turnstile location
create-event turn_a2 store-location store_123 event "assist-customer cust_22"

Create a checkout event
create-event turn_a2 store-location store_123 event "checkout cust_22 store_123:aisle_A2"

Command Line Processor
Handles exception from the store model service and stores the action and reason for the exception.

Methods

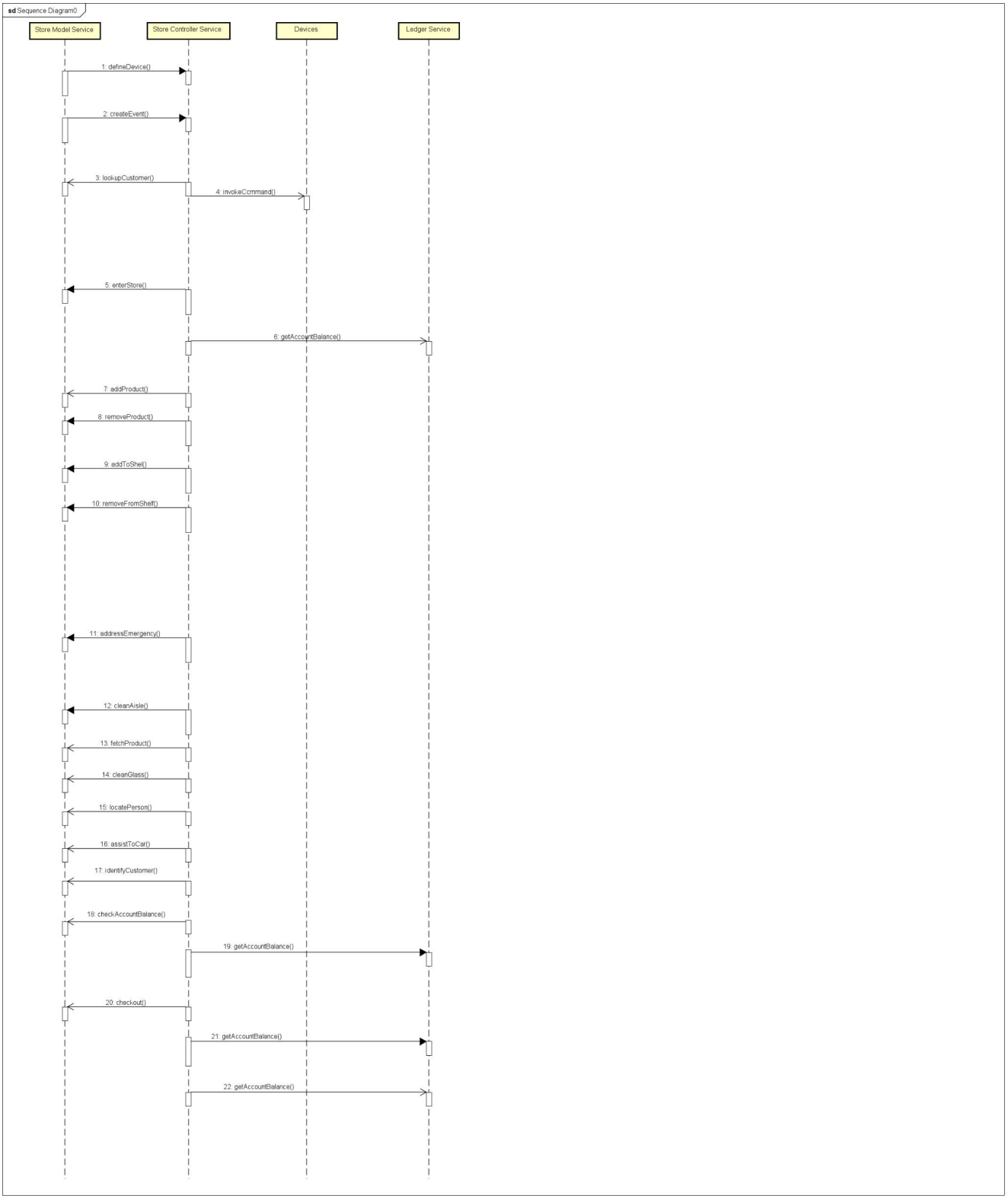
Method Name	Signature	Description
processCommand	(command:string):void	Process a single command. The output of the command is formatted and displayed to stdout. Throw CommandProcessorException on error
processCommandFile	(commandFile:string):void	Process est of commands from a file. Throw CommandProcessorException on error

Command Processor Exception
Handles exception from the command line processor and stores the command, line number and reason for the exception.

Properties

Property Name	Type	Description
command	String	What command threw the exception
reason	String	Why the action caused the exception
LineNumber	Int	The line number of the command

Sequence Diagram



Design

The design is a relatively simple combination of the Observer Patter and the Command Pattern. The model service is the subject with all the information on the store, customers, and devices. The controller service in this pattern is the observer and waits for an event to occur in the model service. When it does, the controller updates the devices by creating a concrete event that knows how to handle the event details. Then given the event it will trigger commands. In this case the devices are the invokers that trigger an command from the Controller. The controller will then create a concrete command that will either command the Model or Ledger service as the receiver depending on the details of the concrete command. Everything else will be handled within those services.

Testing

Implement a test driver class called TestDriver that implements a static main() method. The main() method should accept a single parameter, which is a command file. The main method will call the CommandProcessor.processCommandFile(file:string) method, passing in the name of the provided command file. The TestDriver class should be defined within the package "com.cscie97.store.test".
A test command file will be provided, but you should create your own test file and process that. Include the test input and output with your submission.

Risks

There are some pretty important responsibilities being handled by full automated devices. For example, in the event of a fire or an armed intruder, it is very important that everything works perfectly. In a realistic situation, this would definitely not be handled by the same system as cleaning up spilled milk. The risk of the turnstiles not opening in the event of a fire would be far too much liability to entrust with a fully automated system. Even in the more understandable automated tasks, there definitely should be redundancy built in. For example, broken glass is very dangerous, a microphone listening for the sound of broken glass would not be sufficient to keep the store safe. Also, the baskets and turnstiles access the customers account ledger with no security. This will probably be addressed in the authorization assignment next, but even with authorization, there is risk in the information being so easily exposed. There would need to be layers of encryption in addition to the authorization service.