Store Model Service Design Document

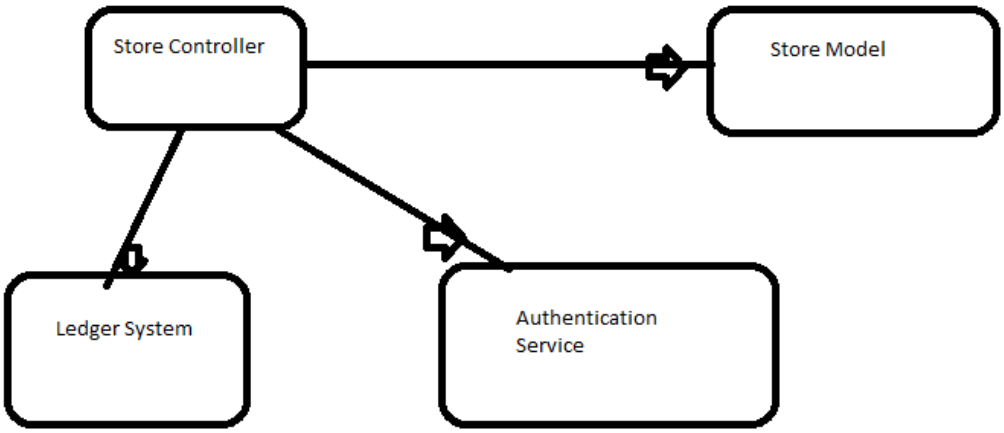
Date: 10/09/2019
Author: Tofik Mussa
Reviewer(s): Antony Gavidia, Xin Yu

Introduction

The store model service is an automated store which, in this iteration of the project, consists mostly of stateful objects. It is a store where a customer can walk by a turnstile, pick items to add to basket and leave the store without the overhead of manual checkout. The store has autonomous equipment like robots, smart turnstiles, microphones and speakers that orchestrate the store.

Overview

In this era of automation, repetitive tasks are better of solved by robots and such. With the rise of the Internet of Things, automation that would have otherwise been beyond imagination has become possible. This project mimics a store like Amazon Go: It adds business value by reducing labor costs and performing tasks that maybe error prone when done manually.



Requirements

Requirements for the Store Model Service System

1) The ability to define virtual objects like stores, aisles, shelves, appliances and sensors
2) The ability to register customers and recognizing them as they walk into the store using facial/voice recognition technology
3) The ability to maintain inventory count and perform janitorial tasks.
4) Simulating events and listening to them
5) The ability to keep track of the location of customers within the store
6) The ability to monitor the products that a customer may purchase by putting into his basket as well as updating inventory count accordingly when a customer makes up their mind not to purchase
7) Not allowing guests buy items without established accounts

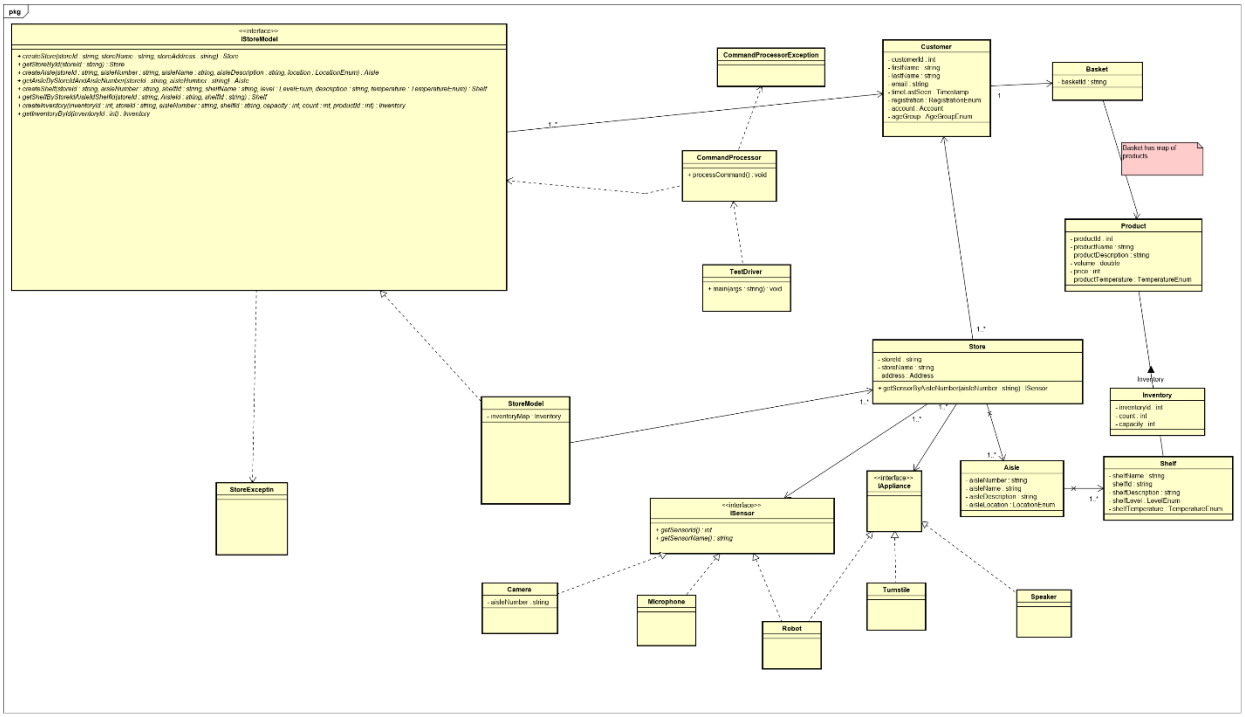
COMMAND TO RUN THE SCRIPTS

```
javac com/cscie97/store/model/*.java com/cscie97/store/model/test/*.java
java -cp . com.cscie97.store.model.test.TestDriver store.script
```

Use Cases


uc
Admin
Send command to appliances
Listen to sensor events
Register new customers
Update customer location
Create Stores, aisles, shelves
Restock shelf, update inventory
Simulate events
create sensors & apliances
Sensors & Appliances
CustomerObj

Class Diagram



Class Dictionary

This section describes the class dictionary for the Store Model Service System

Store

Properties

Property Name	Type	Description
storeId	String	Global unique identifier
storeName	String	Name of the store
address	Address	The address object contains street, city and state

Methods

Method Name	Signature	Description
addAIsleToShelf	(Aisle aisle): void	Adds an aisle to the list of stored aisles

Associations

Association Name	Type	Description
aisles	List<Aisle>	Every store contains certain number of aisles

Aisle

Properties

Property Name	Type	Description
aisleNumber	String	Identifier for an aisle. It is unique within a store
aisleDescription	String	Description of the Aisle
location	LocationType	LocationType is an enum with values of Store_Room or Floor. An aisle can be in store room or floor

Methods

Method Name	Signature	Description
addShlelf	(Shelf shelf): void	Adds a shelf to aisle
addSensor	(Isensor sensor): void	Adds a sensor to aisle
addAppliance	(Iappliance appliance): void	Adds appliance to aisle

Associations

Association Name	Type	Description
shelves	List<Shelf>	An aisle has one to many shelves
sensors	List<Isensor>	An aisle may have sensors
appliances	List<Iappliance>	An aisles may have appliances

Shelf

Property Name	Type	Description
shelfId	String	Identifier unique in any given aisle
shelfName	String	Name of the shelf
level	Level	Level is an enum with values: high, medium and low
shelfDescription	String	Description of what the shelf contains
temperature	Temperature	Corresponds to the temperature of the shelf. It can be - FROZEN, REFRIGERATED, AMBIENT, WARM or HOT

Methods

Method Name	Signature	Description
addInventory	(Inventory inventory) : void	Adds inventory to the list in a shelf
getInventoryInTheShelfByInventoryId	(String inventoryId) : Inventory	Finds inventory for a specific id

Associations

Association	Type	Description

Name		
inventoryList	List<Inventory>	Contains list of inventories which associated with a product, has certain count and capacity

Inventory

Properties

Property Name	Type	Description
invetoryId	String	Globally unique identifier of inventory. All of the stores in the store model service label a suit of product with unique id
Count	int	How many of a certain product there are in a shelf
capacity	int	The total capacity of inventory that a shelf can
inventoryLocation	InventoryLocation	This is the exact storeId, aisleNumber and shelfId of an inventory

Associations

Association Name	Type	Description
product	Product	Each inventory has a one to one mapping with product

Product

Properties

Property Name	Type	Description
productId	String	We don’t have any power over the products the store gets from vendors so this id may not be unique

productName	String	Name of the product
productDescription	String	Description of the product
price	int	Unit price of the product
category	String	Product category – can be used to assemble suit of products
volume	int	Volume is calculated using size to the power of 3
temperature	Temperature	Corresponds to the temperature of the product. It can be - FROZEN, REFRIGERATED, AMBIENT, WARM or HOT

ISensor – An interface

Methods

Method Name	**Signature**	**Description**
getSensorId	(): String	Returns sensor id
getSensorName	(): String	Returns sensor name
getSensorLocation	(): InventoryLocation	Returns storeId and aisleNumber of a sensor. An empty string is set to shelfId field
getSensorType	():String	Returns type of a sensor. This is set to the class name in the implementation
generateSensorEvent	(Event event): String	Simulates sensor event

Iappliance – An interface

Methods

Method Name	Signature	Description
getApplianceId	():String	Returns appliance id
getApplianceName	():String	Returns appliance name
getApplianceLocation	(): InventoryLocation	Returns storeId and aisleNumber of appliance. An empty string is set to shelfId field
getApplianceType	():String	Returns type of an appliance. This is set to the class name in the implementation
generateApplianceEvent	(Event event): String	Simulates appliance event
listenTocommand	(Command command): String	Listens to commands and acts upon them

Camera
Implements all of the methods in ISensor
Microphone
Provides implementation for all of the methods in ISensor
Robot
Provides implementation for ISensor and IAppliance
Turnstile
Provides implementation for all of the methods in Iappliance
Speaker
Provides implementation for IAppliance

Basket

Properties

Property Name	Type	Description
basketId	String	This is provided by the system when a customer picks a basket. The basket itself is temporal and gets garbage

		collected as a customer leaves the store

Methods

Method Name	Signature	Description
addProductToBasket	(Product product, int count):void	Adds items to product Map in the basket and increments count
removeProductFromBasket	(Product product, int count):void	Removes items from product Map in the basket and decrements count

Associations

Association Name	Type	Description
productsMap	Map<Product, Integer>	Keeps track of products and their count

Customer

Properties

Property Name	Type	Description
customerId	String	Customer's unique identifier among all stores
firstName	String	Customer's first name
lastName	String	Customer's last name
customerType	CustomerType	A customer can be either registered or guest
accountAddress	String	This is the blockchain account of every customer
customerLocation	InventoryLocation	This is the storeid and aisleNumber location of a customer. ShelfId is set to empty string since it is assumed that a customer may move fast between aisles

emailAddress	String	Email of the customer
timeLastSeen	LocalDateTime	This gets updated every time the customer's location changes

Associations

Association Name	**Type**	**Description**
basket	Basket	Each customer is assoicated with a basket while in the store

Command

Property Name	**Type**	**Description**
message	String	This is a message that gets listened to by appliances

Event

Property Name	**Type**	**Description**
message	String	This is a message that gets generated by sensors and appliances

StoreException

Property Name	**Type**	**Description**
message	String	This is a standard exception class being thrown when validation fails

CommandProcessorException

Property Name	Type	Description
command	String	The command that failed
lineNumber	int	The line number that failed to execute
reason	String	Reason of failure

StoreModelService – implements IstoreModelService

Method Name	signature	Description
getInstance	():StoreModelService	Returns a single instance of StoreModelService

Associations

Association Name	Type	Description
inventoryMap	map<String, Inventory>	Provides inventory lookup by id
customers	List<Customer>	Maintains all of the customers associated with the system
productMap	Map<String, Product>	Provides product lookup by id
stores	List<Store>	Keeps track of all of the stores in the system

IstoreModelService – interface

This is the brain of the system but since there are around 30 methods, I chose not to include it for brevity. I have instead below a screen shot of the class.

```
Store createAStore(String storeId, String storeName, Address storeAddress)
Store getStoreById(String storeId)
Aisle createAisle(String storeId, String aisleNumber, String aisleDescription, String location)
Aisle getAisleByStoreIdAndAisleNumber(String storeId, String aisleNumber)
Shelf createAShelf(String storeId, String aisleNumber, String shelfId, String shelfName, String level, String shelfDescription, String temperature)
Shelf getShelfByStoreIdAisleNumShelfId(String storeId, String aisleNumber, String shelfId)
Inventory createInventory(String inventoryId, String storeId, String aisleNumber, String shelfId, int capacity, int count, String productId)
Inventory getInventoryById(String inventoryId)
int updateInventoryCount(String inventoryId, int difference)
Product createAProduct(String productId, String productName, String productDescription, int size, String category, int price, String temperature);
Product getProductById(String productId)
Customer createCustomer(String customerId, String firstName, String lastName, String type, String emailAddress, String accountAddress)
Customer getCustomerById(String customerId)
InventoryLocation updateCustomerLocation(String customerId, String storeId, String aisleNumber)
Basket getBasketOfACustomer(String customerId)
Basket createBasketForACustomer(String customerId, String basketId)
Basket addItemToBasket(String basketId, String productId, int count)
Inventory getInventoryByProductId(String productId)
Basket removeItemFromBasket(String basketId, String productId, int countReturned)
Customer clearBasketAndRemoveAssociationWithACustomer(String basketId)
Map<Product, Integer> getBasketItems(String basketId)
ISensor createASensor(String sensorId, String sensorName, String sensorType, String storeId, String aisleNumber)
ISensor getSensorByLocationAndSensorId(String storeId, String aisleNumber, String sensorId)
IAppliance getApplianceByLocationAndSensorId(String storeId, String aisleNumber, String applianceId)
String createSensorEvent(String storeId, String aisleNumber, String sensorId, Event event)
IAppliance createAnAppliance(String applianceId, String applianceName, String applianceType, String storeId, String aisleNumber)
String createApplianceEvent(String storeId, String aisleNumber, String applianceId, Event event)
String createApplianceCommand(String storeId, String aisleNumber, String applianceId, Command command)
```

Implementation Details

I ended up having plenty utility classes that I did not think about when doing the design initially. I would consider them as opportunities to make the structure of the code more maintainable. I found myself repeating logic in few places which led me to think perhaps I need to refactor, extract the logic and reuse.

Exception Handling

There are two types of exceptions thrown: CommandProcessorException and StoreException. StoreException is thrown by classes in the model mainly to enforce invariants. CommandProcessorException may result from error reading file or a command not being able to exceute because of value not found

Testing

In my opinion unit testing will make the system more robust. Reverse engineering with Test Driven Development may also provide some insights.

Risks

Making data in sync has to be thought about. For example, I have a map in the StoreModelService class for inventory to track the global inventory among all of the stores but when there is a request for updating count both the global count and the count in a specific shelf need to be updated.