Store Model Service Design Document

Date: 9/28/19 (revised 10/8/19)

Author: Matthew Thomas

Reviewer(s): Steven Hines, Stephen Thompson

# Introduction

This design specifies the implementation of a Store Model Service, one component of the Store24X7 Software System. The Store Model Service is responsible for defining the configuration of a store and all its components, as well as maintaining the state of the individual components as objects are created and updated. It receives input from Customers and various Devices (Sensors and Appliances) located throughout individual Stores.

The design document provides an overview of what problem the Store Model Service aims to solve, a list of requirements the Store Model Service will need to address, and use case scenarios about how actors will interact with the service. This document will also include a class diagram and dictionary to assist a program with implementation, and notes on exception handling, testing, and known risks with the Store Model Service as designed.

Overview

As online shopping grows in popularity, owners of brick-and-mortar stores are increasingly looking for ways to bring customers into their shops and to cut costs. With the advance of technology and the advent of the Internet of Things (IoT), it has become possible to create an automated store, run entirely by sensors (microphones and cameras) and appliances (speakers, robots, and turnstiles) that monitors how customers shop.

An automated store is more appealing for Customers because it reduces the number of headaches they encounter. For example, if a customer cannot remember whether the olive was in the “Pantry” aisle or the “Baking” aisle, they can instead ask one of many microphones located throughout the store “Where is the olive oil?” Likewise, self-checkout machines often raise errors when scanning certain products and complain when it detects an unknown object in the bagging area. By having cameras detect what products customer’s take off the shelf, it vastly simplifies the process when it comes time to check out; all the customer needs to do is walk out of the store — the running total is already calculated as they shop.

Likewise, for the business owner, an automated store drastically cuts down on costs by reducing the need for positions like cashiers (replaced by the turnstile), shelf stockers (replaced by robots), and customer service representatives (replaced by the ubiquitous microphones and speakers). Defining and maintaining stores using a Store Model Service also allows for great flexibility in terms of the size and layout as well as enables the owner to have a centralized place to check in on the state of their operations.

# Requirements

This section defines the requirements for the Store Model Service.

## Store Configuration

1. Define all store entities including Store, Aisle, Shelf, and Inventory.
2. Access the details of all entities including identifying information.
3. Update the state of the Inventory (must always be >=0 and <= capacity as specified on page 4 of the requirements).

## Customers

1. Register their Ledger account with the Store Model Service.
2. Query Store Sensors and Appliances with questions or tasks.
3. Move around the Store and add/remove Products to their Basket.
4. Registered customers can leave a Store provided they have a sufficient account balance to pay for any items in their basket.
5. Customer’s not registered with the Store Model Service (guest customers) are not allowed to remove items from the store (requirements, page 5).

## Store Model Service

1. The Store Model Service will perform create, read, and update the store configuration (see requirements above; system architecture, page 5).
2. Receive events from physical sensors.
3. Receive commands from physical appliances.
4. Control sensors and appliances based on received inputs.
5. Provide a public API to manage the state of the store (requirements, pages 6-7).

## Sensors/Appliances

1. Sensors will capture data about the condition of objects located within a Store.
2. Appliances will capture data about the condition of itself and objects within a Store.

## Not required

### Authentication

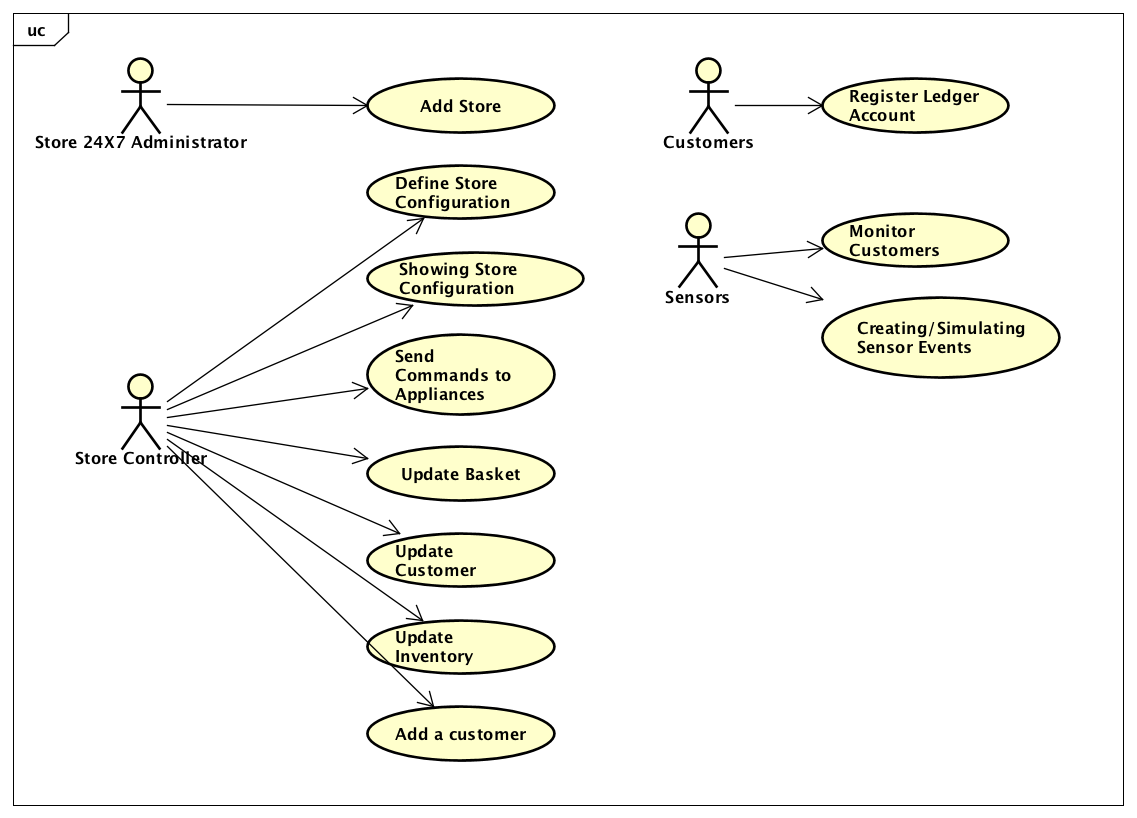
To verify Customers and restrict access to certain parts of the store, an Authentication Service is required. This will be implemented as part of Assignment 4. For now, the Store Model Service will accept any Customer that tries to register. As specified by the requirements on page 7, all API methods include an “authToken” parameter, but to limit this implementation, it is treated as an opaque string that does not affect the operations.

### Persistence

As in Assignment 1, the Store Model Service and all defined objects will be maintained in memory. Saving the state of the Store Model Service is not required beyond keeping the objects in memory.

# Use Cases

The following Use Case diagram describes the use cases supported by the Store Model Service.



## Actors:

The actors of the Store Model Service include Store 24X7 Admin, Store Controller, Sensors, and Customers.

### Store 24X7 Admin

The store Admin is responsible for provisioning stores and all the objects within. For this assignment, this is performed via a script file which uses the Store Model Service API to run commands.

### Store Controller

The Store Controller is responsible for monitoring the sensors and controlling appliances within the store. The Controller uses the sensors and appliances to monitor the location of customers and respond to commands or events (system architecture, page 3). For example, when a customer adds an item to their basket; leaves the store through turnstile.

### Customers

The Customer is responsible for registering themselves and their associated Ledger Account. Customer actions and requests are monitored by Store Sensors and Appliances through the Store Model Service; a customer’s direct input is not required.

### Sensors/Appliances

The Sensors and Appliances are responsible for providing input and events to the Store Controller to assist with monitoring the state of the store. Appliances have the added responsibility of receiving commands to perform an action.

## Use Cases:

### Add Store

A store is added by setting the globally unique identifier, name, and address (requirements, page 4). Stores are created by an Administrator (or for assignment 2, a test script), but stored and managed by the Store Model Service. A Store consists of several store entities, including Aisles, Shelves, and Inventories.

### Setup Store Entities

As specified in the requirements (pages 4-5), each store contains Aisle, Shelf, and Inventory objects. An Aisle is unique within a Store, a Shelf is unique within an Aisle, and an Inventory is unique within a Shelf. Each object stores similar basic information (id, name) as well as more specific information (location of the Aisle, the level of the Shelf, capacity of the Inventory).

### Show Store Configuration

Displays the details of an object within the store. Options include: Aisle, Basket, Customer, Inventory, Product, Shelf, Store, Sensor, and Appliance. Each object provides details specific to itself and the details are described further in the class dictionary below.

### Register Ledger Account

Customers register their existing Ledger Account with the Store Model Service’s system.

### Monitor Customers

Sensors will monitor the current location of a Customer as they move throughout the Store. Sensors will monitor when products are added/removed from an inventory or customer basket as the customer moves throughout the store. When the customer exits through a turnstile, an event is triggered to checkout the customer (system architecture, page 3).

### Creating Sensor Events

Physical sensors and appliances generate events which are processed by the Store Model Service and handled by the Store Controller.

### Update Store State

The state of Store objects are maintained by the Store Model Service, but managed by the Controller (system architecture, page 3). The Store Model Service validates any change before applying the update to ensure store objects remain in a clean state.

### Listen for Events

The Controller listens for events that are triggered by sensors and appliances. Upon receiving an event, the Controller may trigger an update to store objects. For example, as a customer moves throughout the store, a camera sensor detects that movement and sends an event which updates the customer’s location in the store (and the time of the last change — requirements, page 5).

### Add a customer

Customers register their accounts with the Ledger Service, but the Store Controller is responsible for adding a customer to the Store Model Service. Guest customers are allowed into the store, but guests are not allowed to remove products from the store without registration (requirements, page 5).

# Implementation

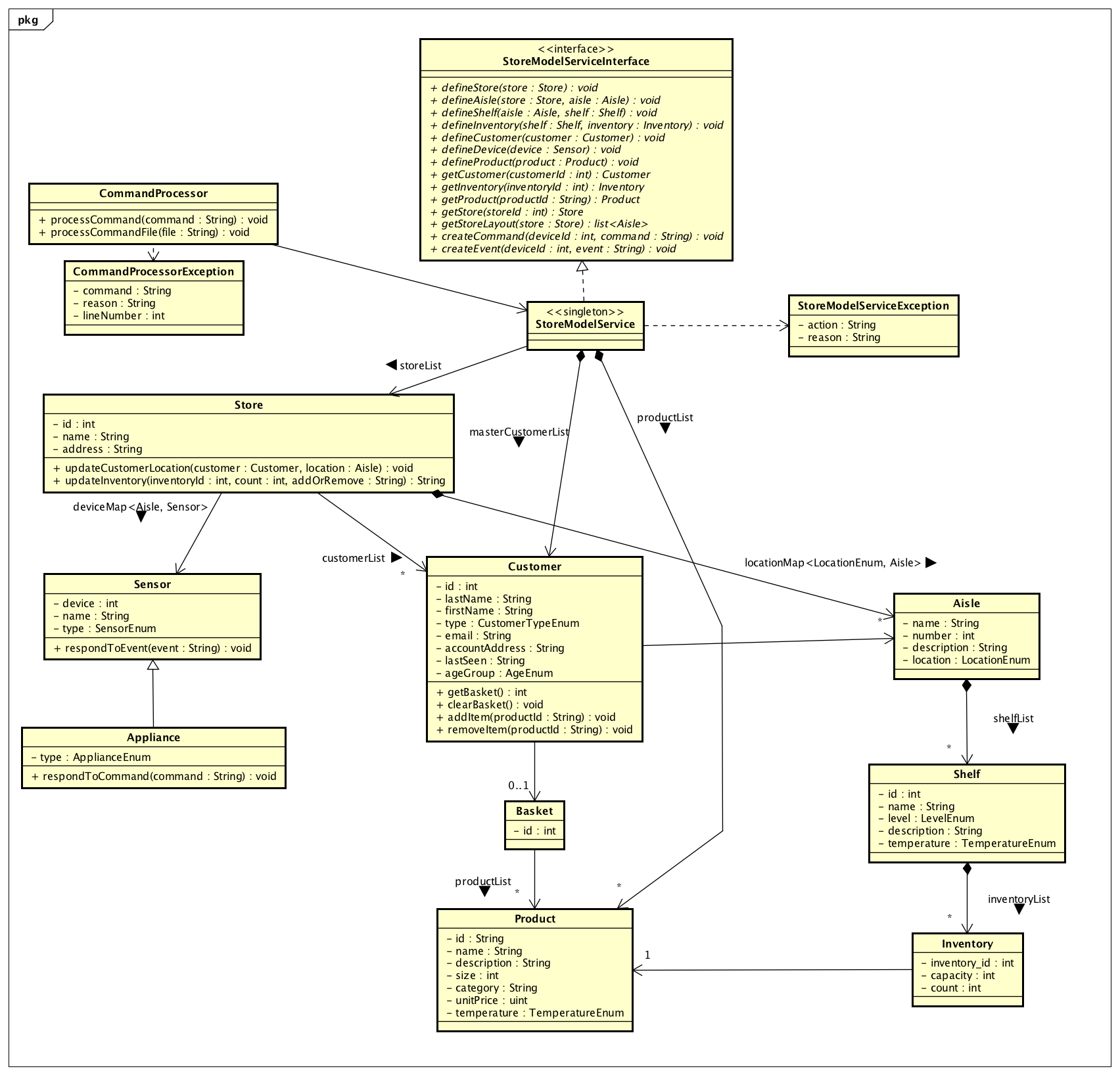
*This section of the document will describe the implementation details for ...*

*The implementation section should cover the following topics:*

* *What are the important interfaces and how they will be implemented?*
* *How are the requirements addressed?*

# Class Diagram

The following class diagram defines the Store Model Service implementation classes contained within the package “com.cscie97.store.model”.



# Class Dictionary

This section specifies the class dictionary for the Store Model Service. The classes are defined within the package “com.cscie97.store.model”.

## Store Model Service

The Store Model Service is an interface which defines basic functionality. When implemented, a singleton is instantiated to provide access to the Store Model Service through a Public API.

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| masterCustomerList | Customer | A master list of Customers known to the Store 24X7 system. Customers can move from Store to Store, and if registered, should be detected upon entering. |
| storeList | Store | A list of all Stores created by the Store 24X7 Administrator. |
| productList | Product | A list of all Products known about by the Store 24X7 system. Products can exist and be known to multiple Stores. To reduce duplicating information and to keep a centralized list of all Products that are known, the list is managed by the StoreModelService. Products can be associated with Inventory or a Basket. |

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| defineStore | (store : Store) : void | Validate and add a Store object to the storeList. |
| defineAisle | (store : Store, aisle : Aisle) : void | Validate and add an Aisle object |
| defineShelf | (aisle : Aisle, shelf : Shelf) : void | Validate and add a Shelf object to a specified Aisle. (Note, Store object not needed since Aisle are specific to a given Store). |
| defineInventory | (shelf : Shelf, inventory : Inventory) : void | Validate and add an Inventory object to a Shelf’s inventoryList. |
| defineProduct | (product : Product) : void | Validate and create a Product object. |
| defineCustomer | (customer : Customer) : void | Validate and add a Customer |
| defineDevice | (device : Sensor) : void | Validate and add a new Device (Sensor/Appliance). |
| getCustomer | (customerId : int) : Customer | Show Customer details. |
| getInventory | (inventoryId : int) : Inventory | Show Inventory details. |
| getProduct | (productId : String) : Product | Show Product details. |
| getStore | (storedId : int) : Store | Show Store details. |
| getStoreLayout | (store : Store) : list <Aisle> | Retrieve the list of Aisles for a specified Store. |
| getDevice | (deviceId : int) : Sensor | Show device (Sensor/Appliance) details. |
| createCommand | (deviceId : int, command : String) : void | Send the Appliance specified by deviceId, the command. (Commands will be specified in Assignment 3). |
| createEvent | (deviceId : int, event : String) | Send the Device (Sensor/Appliance) a simulated event. (Events will be specified in Assignment 3). |

## Store

The Store class represents an individual store that is created by the Store 24X7 Administrator. It contains several domain objects, created by the Store Model Service. Multiple Stores must be allowed to co-exist and each much have a globally unique identifier.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | String | Globally unique identifier for the Store (assigned by the Store 24X7 Admin — for assignment 2, it is done via store.script). |
| name | String | Name of store. |
| address | String | Location of the store with City, Street, State, Zip Code (e.g. 1400 Massachusetts Avenue, Cambridge, MA 02138). |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| customerList | Customer | A list of customers that are currently located in the Store. |
| locationMap | map <LocationEnum, Aisle> | A map of Aisles within a store, and what part they are located in. Options for LocationEnum include Floor and Stock Room. |
| deviceMap | map <Aisle, Sensor> | A map of Sensors/Appliances and their location in the Store (based on Aisle). |

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| updateCustomerLocation | (customer : Customer, location : Aisle) : void | Update a Customer’s locatation within a Store, based on Sensor’s feedback. |
| updateInventory | (inventoryId: int, count: int, addOrRemove : String) : String | Updates the inventory specified by inventoryId. It increments or decrements the product count by ‘count’. addOrRemove is a String specifying either “Increment” or “Decrement”. |

## Aisle

The Aisle is a location within the store where shelves can be placed.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| number | int | The assigned Aisle number. |
| name | String | The name of the Aisle (e.g. dairy). |
| description | String | Aisle description (e.g. milk, cheese, eggs). |
| location | LocationEnum | Options include: Floor (accessible by all) or Stock Room (not accessible by customers). |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| shelfList | Shelf | List of all shelves located within the aisle. |

## Shelf

The Shelf is a platform within an aisle within a store for inventory to be placed.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | int | The Shelf identifier. |
| name | String | The name of the Shelf (e.g. milk). |
| level | LevelEnum | The heigh of the shelf. Options for LevelEnum include High, Medium, or Low. |
| description | String | The description of the contents on the shelf. (e.g. Skim and whole milk, soy and almond milk) |
| temperature | TemperatureEnum | TemperatureEnum options include Frozen, Refrigerated, Ambient, Warm, Hot. Default value: Ambient. |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| inventoryList | Inventory | List of all Inventory objects specifying what Products are located on a Shelf. |

## Inventory

The Inventory class defines the products available for sale within the store and where they are located. The class specifies the amount of a given Product that can be stored (the capacity) and the amount still left on the shelf (the count).

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | int | The Inventory identifier. |
| capacity | int | The maximum number of product items that can fit on the shelf. |
| count | int | The current number of product items on the shelf (must remain >= 0 and <= capacity). |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| product | Product | The Product that is tracked by the Inventory class. |

## Product

The Product class represents a generic item available for purchase in the Store. Products are associated with an Inventory for a given shelf/shelves and can be associated with a Customer’s Basket when the Product is removed from the shelf.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | String | The Product identifier (usually provided by the manufacturer). |
| name | String | The name of the Product (e.g. Teddy’s Peanut Butter) |
| description | String | The description of the Product (e.g. “All natural, smooth peanut butter. No preservatives, non-homogenized.“). |
| size | int | The weight and/or volume of the Product. |
| category | String | The type of Product (e.g. produce, dairy, deli, etc.) |
| unitPrice | uint | The price of an individual Product, in the block chain currency (unsigned integers). |
| temperature | TemperatureEnum | Options include Frozen, Refridgerated, Ambient, Warm, Hot. Default value: Ambient. |

## Customer

The Store is an interface to model a Store instance from.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | int | The Customer identifier. |
| firstName | String | The Customer’s first name. |
| lastName | String | The Customer’s last name. |
| type | CustomerTypeEnum | Options include: Registered, Guest. |
| email | String | The Customer’s email address. |
| accountAddress | String | The Customer’s blockchain address (assigned by the Ledger Service). |
| lastSeen | String | Time last seen (updated when location is updated). |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| location | Aisle | The current Aisle where the Customer is located. Location updated in response to Sensor detections. |
| basket | Basket | The Customer’s Basket. Only allowed to have one Basket at a time. |

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| getBasket | () : int | Get the basket\_id currently associated with the Customer, if one exists. If not, create a new Basket and return the id. |
| clearBasket | () : void | Dissociates all Products currently in a Customer’s Basket and removes the Basket from the Customer. |
| addItem | (productId : String) : void | Add the specified Product to the Basket. |
| removeItem | (productId: String) : void | Remove the specified Product from the Basket. |

## Basket

The Store is an interface to model a Store instance from.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | int | The Basket identifier. |

**Associations**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| productList | Product | A list of Products currently in the Basket. |

## Sensor

The Sensor is a parent class representing both Sensors and Appliances. The Sensor class defines all basic identifying properties and a method to process events.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| id | int | The device identifier. |
| name | String | The name of the device. |
| type | SensorEnum | The type of Sensor. SensorEnum options include: Microphone and Camera. |

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| respondToEvent | (event : String) : void | Process an event detected by the Sensor. |

## Appliance

The Appliance class is a child to the Sensor class, and inherits both the ‘id’ and ‘name’ properties. The Appliance class also includes a ‘type’ property, but this is overridden with Appliance-specific options defined in ApplianceEnum.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Property Name** | **Type** | **Description** |
| type | ApplianceEnum | The type of Appliance. ApplianceEnum options include: Speaker, Robot, and Turnstile. |

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| respondToCommand | (command : String) : void | Process a command received by the Appliance. |

## CommandProcessor

The CommandProcessor is a utility class for feeding the Ledger a set of operations, using a command syntax. The command syntax to be used for the Store Model Service can be found in the Store Model Service requirements document (Note: this was implemented in Assignment 1 and is copied, with modifications, for Assignment 2.)

**Methods**

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Signature** | **Description** |
| processCommand | (command : String) : void | Process a single command. Output of the command is formatted and displayed to stdout. Throw a CommandProcessorException on error. |
| processCommandFile | (commandFile : String) : void | Process a set of commands provided within the given commandFile. Throw a CommandProcessorException on error. |

# Implementation Details

*Explain details of the implementation.*

*How do the various parts fit together or interact?*

*How does the design address the requirements? Justify your design decisions and how they address the requirements.*

*Some implementation details may be addressed in the class dictionary, but for things that are not, describe them here.*

*Remember to reference the requirements from the body of the design document to show how your design is addressing the requirements.*

# Exception Handling

*Provide details on your exception handling. What types of exceptions are expected and how are they handled by the design? Describe your exception classes and their properties.*

## StoreModelServiceException

The Store Model Service is an interface which defines basic functionality. When implemented, a singleton is instantiated to provide access to the Store Model Service through a Public API.

**Properties**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| action | String | A list of all Stores created by the Store 24X7 Administrator. |
| reason | String |  |

## CommandProcessorException

The CommandProcessorException is returned from the CommandProcessor methods in response to an error condition. The CommandProcessorException captures the command that was attempted and the reason for the failure. In the case where commands are read from a file, the line number of the command should be included in the exception. (Note: this was implemented in Assignment 1 and is copied verbatim for Assignment 2.)

**Properties**

|  |  |  |
| --- | --- | --- |
| **Association Name** | **Type** | **Description** |
| Command | String | Command that was performed (e.g., “define store”). |
| reason | String | Reason or exception (e.g. “A Store already exists with the specified ID”). |
| lineNumber | int | Line number of the command in the input file. |

# Testing

*Provide a testing strategy for testing the component.*

* *Functional*
* *Performance*
* *Regression*
* *Exception Handling*

A TestDriver class will be defined within the package “cscie97.store.model.test” and will handle reading in a test script (store.script), and handing off instructions to the CommandProcessor. The test script will handle negative and boundary testing (e.g. creating a Store with id of -1, creating a second store with a duplicate ID, etc.) as well as functional testing (e.g. if a Customer adds two items costing 5 units each, is the Customer charged 10 units when leaving the store through a turnstile).

Exception handling is described in more detail above.

# Risks

*Document any risks identified during the design process.*

*Are there parts of the design that may not work or need to be implemented with special care or additional testing?*