Store Model Service Design Document

Date: 9/28/19 (revised 10/9/19)
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 programmer with implementation, plus 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 monitor how customers shop.

An automated store is more appealing for Customers because it reduces the number of obstacles they encounter. For example, if a customer cannot remember whether the olive oil was in the “Pantry” aisle or the “Baking” aisle, they can ask one of many microphones located throughout the store “Where is the olive oil?” Even existing in-store technology can cause difficulties. Self-checkout machines, for example, often raise errors when scanning certain products and complain when they detect 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.

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. Customers 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

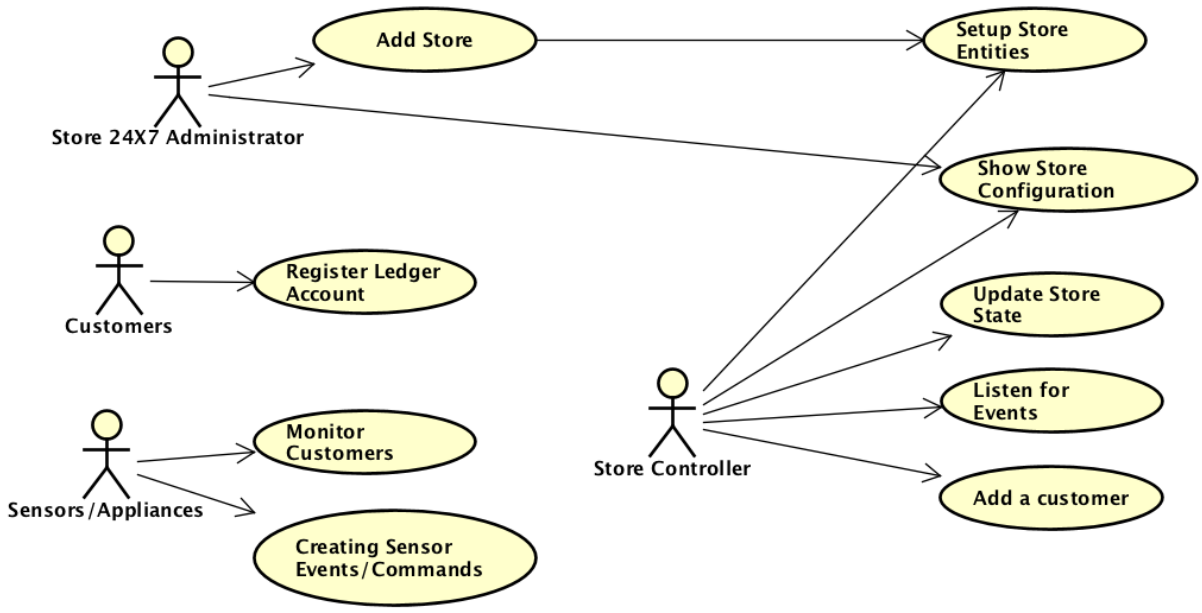
To verify Customers and restrict access to certain parts of the store, an Authentication Service is required. This will be implemented as part of Assignment 4. For now, the Store Model Service will accept any Customer that tries to register. As specified by the requirements on page 7, all API methods 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 or leaves the store through aturnstile.

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).

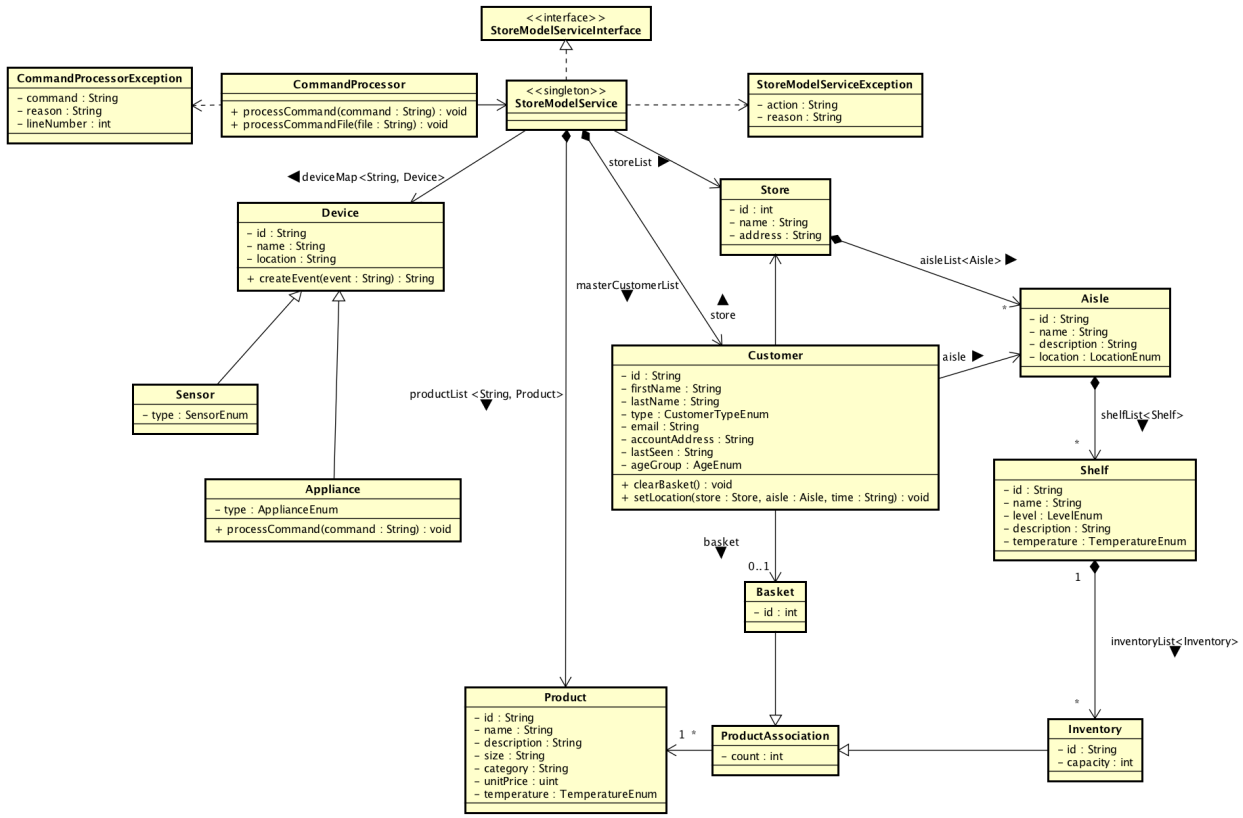
Add a Customer

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

Class Diagram

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

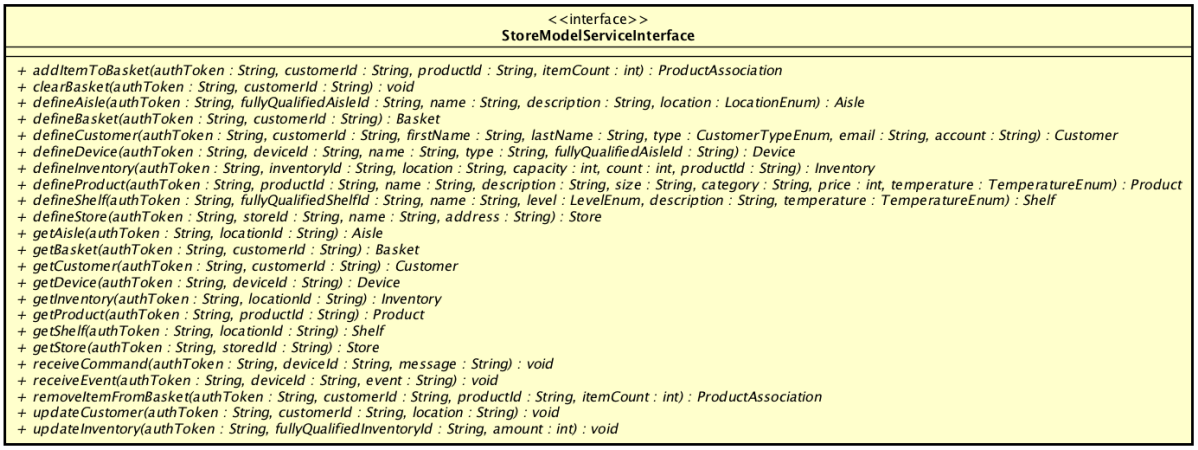


Class Dictionary

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 its basic functionality which includes creating, updating, and accessing the state of store entities, including Store, Aisle, Shelf, Inventory, Product, Device, Customer, and Basket objects. When implemented, a singleton is instantiated to provide access to the Store Model Service through a public API. All methods accept an authToken parameter, the functionality of which will be defined in Assignment 3. For now, it is treated as an opaque string and performs no function.



Associations

Association Name	Type	Description
customerMap	map<String, Customer>	A master list of Customers known to the Store 24X7 system. Customers who have registered their Ledger accounts with the system should be detected upon entering a Store. Guest Customers are added to the list when detected by a store Device.
storeMap	map<String, Store>	A list of all Stores created by the Store 24X7 Administrator. Stores are globally unique within the system.
productMap	map<String, 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, via a ProductAssociation.
deviceMap	map<String, Device>	A list of all Devices (Sensors, Appliances) that are provisioned for use in the Store

		24X7 system. Devices are located in one store at a time, but since they are globally unique within the system, can be moved between stores by the Store 24X7 Administrator.

Methods

Method Name	Signature	Description
addItemToBasket	(authToken:String, customerId:String, productid:String, itemCount:int):ProductAssociation	Add the given item count of a Product specified by the product id to the Customer's basket. If the basket already contains the product, increase the count. Otherwise, add the count of the Product. Returns the association that specifies how much of the given Product is in the basket.
clearBasket	(authToken:String, customerId:String):void	Removes all ProductAssociation objects currently in the Basket specified by customerId and deletes the Basket.
defineAisle	(authToken:String, fullyQualifiedAisleId:String, name:String, description:String, location:LocationEnum):Aisle	Validate and add an Aisle object. Checks that an Aisle with the same id does not currently exist in the Store referenced in the fully qualified id and adds to a Store's aisle list if successful.
defineBasket	(authToken:String, customerId:String):Basket	Creates a new Basket object and associates it with the customer given by customer id.
defineCustomer	(authToken:String, customerId:String, firstName:String, lastName:String, type:CustomerTypeEnum, email:String, account:String):Customer	Validate and add a Customer. Checks that a Customer with the given customer id does not already exist in the Store Model Service.
defineDevice	(authToken:String, deviceId:String, name:String, type:String, fullyQualifiedAisleId:String):Device	Validate and add a new Device. Checks that a device with the given deviceid does not already exist. Checks the given type against a list of known Sensor and Appliance types.
defineInventory	(authToken:String, inventoryId:String, location:String, capacity:int, count:int,	Validate and add an Inventory object. Checks that count is >= 0 and <= capacity. Checks that the Inventory does not already exist within the Shelf-scope specified by the

	productid:String):Inventory	location. Inventory is added to the inventory list of the Shelf specified in the location.
defineProduct	(authToken:String, productid:String, name:String, description:String, size:String, category:String, price:int, temperature:TemperatureEnum):Product	Validate and create a Product object. Checks that a product with the given product id does not already exist in the Store Model Service.
defineShelf	(authToken:String, fullyQualifiedShelfId:String, name:String, level:LevelEnum, description:String, temperature:TemperatureEnum):Shelf	Validate and add a Shelf object. Checks that a shelf with the same id does not currently exist in the Aisle referenced in the fully qualified id and adds to an Aisle's list if successful.
defineStore	(authToken:String, storedId:String, name:String, address:String):Store	Validate and add a Store object to the Store Model Service storeMap.
getAisle	(authToken:String, locationId:String):Aisle	Return the Aisle for the given aisle id within given in the location id. Location id is assumed to follow the syntax <store_id>:<aisle_id>.
getBasket	(authToken:String, cutomerId:String):Basket	Return the Basket associated with the Customer for the customer id.
getCustomer	(authToken:String, cutomerId:String):Customer	Return the Customer for the given customer id.
getDevice	(authToken:String, deviceId:String):Device	Return the Device for the given device id.
getInventory	(authToken:String, locationId:String):Inventory	Return the Inventory for the given inventory id within the given location id. Location id is assumed to follow the syntax <store_id>:<aisle_id>:<shelf_id>:<inventory_id>.
getProduct	(authToken:String, productId : String) : Product	Return the Product for the given product id.
getShelf	(authToken:String, locationId:String):Shelf	Return the Shelf for the given shelf id within the given location id. Location id is assumed to follow the syntax

		<store_id>:<aisle_id>:<shelf_id>.
getStore	(authToken:String, storeId:String):Store	Return the Store for the given store id.
receiveCommand	(authToken:String, deviceId:String, message:String):void	Send the Appliance specified by deviceId, the command. (Commands will be specified in Assignment 3).
receiveEvent	(authToken:String, deviceId:String, event:String):void	Send the Device (Sensor/Appliance) a simulated event. (Events will be specified in Assignment 3).
removeItemFromBasket	(authToken:String, customerId:String, productid:String, itemCount:int):ProductAssociation	Remove a given 'itemCount' amount of a Product from the specified Customer's Basket. If removing 'itemCount' results in 0 or fewer items remaining, the Product is removed entirely.
updateCustomer	(authToken:String, customerId:String, location:String):Void	Updates the Customer's location with references to the current Store and Aisle, parsed from the location String. The time at which this change is made is also stored, via the setter in Customer.
updateInventory	(authToken:String, fullyQualifiedInventoryId:String, amount:int):void	Update the count in the specified Inventory object by 'amount'. The count must remain >=0 and <= capacity.

Store

The Store represents a physical store that is created by the Store 24X7 Administrator. It contains several store entity objects, created by the Store Model Service. Stores must be globally unique which is guaranteed by the Store Model Service. A Store contains Customers and Devices, which are tracked and update by the Store Model Service. A Store also contains and tracks a list of Aisles.

Properties

Property Name	Type	Description
id	String	Globally unique identifier for the Store — assigned by the Store 24X7 Admin and checked by the Store Model Service. For assignment 2, this is done via a provided test script.
name	String	Name of store.
address	String	Location of the store with City, Street, State, Zip Code (e.g. 123 Anywhere Lane, Beverly

		Hills, CA 90210).

Associations

Association Name	Type	Description
aisleList	list<Aisle>	A list of Aisles located within a store.

Aisle

An Aisle stores a list of Shelf objects which break down further to ultimately track Products within a Store. The Aisle is also where Customers can walk and Devices (Sensors and Appliances) can monitor activity and also respond to commands (Appliances only). An Aisle can be located either on the 'floor' (which is accessible by all) or the 'stock_room' (which is not accessible by customers).

Properties

Property Name	Type	Description
id	String	The identifier for the Aisle. The requirements (page 4), specify an aisle ‘number’, but to maintain consistency between store entities, an ‘id’ is used.
name	String	The name of the Aisle (e.g. produce).
description	String	Aisle description (e.g. fruits and vegetables).
location	LocationEnum	Options include: Floor (accessible by all) or Stock_Room (not accessible by customers). The requirements (page 4, 7) specify ‘store_room’, but to reduce ambiguity, this design calls for ‘stock_room’ to emphasize the point that it is inaccessible 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. The Shelf can be at a height specified by the LevelEnum, or high, medium, or low. Each shelf must be unique within a given Aisle and

can contain inventory that tracks the capacity and count of products located on the Shelf. A Shelf is specified to maintain a temperature that its products should match.

Properties

Property Name	Type	Description
id	String	The Shelf identifier.
name	String	The name of the Shelf (e.g. apples).
level	LevelEnum	The height of the shelf. Options for LevelEnum include High, Medium, or Low.
description	String	The description of the contents on the shelf. (e.g. Red Delicious, Gala, McIntosh, Granny Smith)
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.

ProductAssociation

The ProductAssociation is an association class that references a Product and stores a given count of that Product. The count can be increased or decreased based on events detected in the Store.

The count is updated by the class which extends the ProductAssociation. Behavior for what happens when a count equals or goes below 0, or goes above a certain threshold, varies and is dependent on the child class.

Properties

Property Name	Type	Description
count	int	The current number of product items.

Associations

Association Name	Type	Description

product	Product	The Product that is tracked.

Inventory

The Inventory class extends the ProductAssociation class which stores a product id and the count of the product. The Inventory adds a capacity (how much of the Product can be stored) and an id (where is the inventory located). The requirements (pages 4-5) specify that the inventory contains a location attribute (of the format store:aisle:shelf), but this would lead to an inventory being maintained by the Store Model Service. Instead, an Inventory is treated as on level deeper, located within a shelf and accessed using the syntax store:aisle:shelf:inventory.

An inventory must maintain the count, such that it is >= 0 and <= capacity. This management is performed by the Store Model Service in response to events detected by Devices.

Properties

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

Product

The Product class represents a generic item available for purchase in the Store. Information about a Product is usually obtained through the third-party vendor who makes the Product and supplies it to the Store 24X7 System. Products are associated with an Inventory or Basket through the ProductAssociation class. Products are tracked by the Store Model Service to provide a centralized place where details can be updated and propagated to the various places a Product is stored.

Properties

Property Name	Type	Description
id	String	The Product identifier (usually provided by the manufacturer, also known as a SKU).
name	String	The name of the Product (e.g. Teddie Peanut Butter)

description	String	The description of the Product (e.g. “All natural, smooth peanut butter. No preservatives, non-homogenized.“).
size	String	The weight and/or volume of the Product (e.g. 16oz, 500g, 2L, etc.).
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	The temperature at which a product needs to be stored at. Options include Frozen, Refridgerated, Ambient, Warm, Hot. Default value: Ambient.

Customer

The Customer contains an identifier that is unique to the Store 24X7 System as well as various contact information including name, email, and customer type. A Customer is responsible for registering for an account with the Ledger Service, and to report that account address to the Store Model Service. Other properties and associations are maintained by the Store Model Service in response to Device events as they move around a Store located in the system. Registered customers are allowed to “checkout” from the Store by having the cost of products in their Basket deducted from their Ledger account. If the Customer is a Guest, they are not allowed to remove items from the Store.

Properties

Property Name	Type	Description
id	String	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
store	Store	The current Store where the Customer is located. Location updated in response to Sensor events.
Aisle	Aisle	The current Aisle where the Customer is located. Location updated in response to Sensor events.
basket	Basket	The Customer's Basket. Only allowed to have one Basket at a time.

Methods

Method Name	Signature	Description
clearBasket	() : void	Removes all Products currently in a Customer's Basket and removes the Basket from the Customer.
setLocation	(store:Store, aisle:Aisle, time:String):void	Updates the current location references to a Store and Aisle as well as the 'lastSeen' property.

Basket

The Basket is an object that tracks Products and amounts (through ProductAssociation objects) that are associated with a given Customer. Each Customer is allowed only one Basket, and therefore share ids. The requirements (page 6) specify that the Basket includes an id but doesn't specify how a unique id would be referenced. Using the Customer id for the Basket identifier more closely links the two and allows for easier management and lookup.

Properties

Property Name	Type	Description
id	int	The Basket identifier (same as the Customer ID to which it is associated).

Associations

Association Name	Type	Description
productList	list<ProductAssociation>	A list of Products, and their counts, currently in the Basket.

Device

The Device is globally unique within the System and can be classified as one of two sub-types: Sensor or Appliance. A Device is responsible for capturing data about conditions in the store; the kind and type of data can vary depending on the type of Device. All Devices emit 'events' based on activity within a store.

Properties

Property Name	Type	Description
id	String	Globally unique device identifier.
name	String	The name of the device.
location	String	The location of the device, specified as an Aisle within a Store. The location is stored as a fully-qualified aisle id, of the format <store_id>:<aisle_id>.

Methods

Method Name	Signature	Description
createEvent	(event : String) : String	Create an event detected by the Device that is processed by the Store Model Service. For assignment 2, the event is an opaque string (requirements, page 9).

Sensor

A Sensor is a Device that monitors and captures data about conditions in the store. The Sensor can be a microphone (which listens for voice commands from customers), or a camera (which updates the location of customers).

Properties

Property Name	Type	Description
type	SensorTypeEnum	The type of Sensor. SensorTypeEnum options include: Microphone and Camera.

Appliance

An Appliance is a Device that monitors and captures data about conditions in the store and itself. The Appliance can be a speaker (to communicate with customers), a robot (which communicates with customers as well as performs tasks), or a turnstile (which communicates with customers as well as opens/closes the turnstile).

An Appliance differs from a Sensor, or general Device, in that it can respond to commands and

be controlled.

Properties

Property Name	Type	Description
type	ApplianceTypeEnum	The type of Appliance. ApplianceTypeEnum options include: Speaker, Robot, and Turnstile.

Methods

Method Name	Signature	Description
processCommand	(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

The core component for the system is the Store Model Service. This class provides the public API for creating, reading, and updating store objects. Before performing any actions, the Store Model Service validates that all required parameters are included and fall within requirements. When creating store objects (Store, Aisle, Shelf, Inventory), ids are checked for uniqueness within the location hierarchy and that “holding containers” exist to store each new object. For example, an Aisle cannot be created in a Store that does not exist and a Shelf cannot be created in a Store and/or Aisle that does not exist. Likewise, two Stores cannot share the same id, nor can two Shelf objects share the same id if they are located within the same Aisle.

Several objects that might make sense to be associated with a Store, like a Customer (customers shop in a Store), a Device (devices are placed and operate within a Store), or Products (which are located within a Store) actually make more sense to be managed by the Store Model Service itself. This centralized control fulfills the requirements by allowing a "Customer to be recognized by all stores" (requirements, page 5), and for Devices and Products to move between stores should the retailer choose to adjust the layout or deployment. It provides flexibility as described in the system architecture document.

Exception Handling

StoreModelServiceException

The StoreModelServiceException is returned from the Store Model Service API in response to an error condition. The StoreModelServiceException captures the action that was attempted and the reason for the failure.

The main operations of the Store Model Service are the CRU of the CRUD operations: creating, reading, and updating objects. StoreModelServiceExceptions will occur when an object does not exist, when creating a new object would create a duplicate object, or when the update parameters fail to meet requirements.

Properties

Association Name	**Type**	**Description**
action	String	Action that was performed (e.g., "define store").
reason	String	Reason for exception (e.g. "A Store already exists with that ID").

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

A TestDriver class will be defined within the package “com.cscie97.store.test” and will handle reading in a test script (store.script, and others are provided), 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).

Performance testing is not handled for this assignment as we do not have all the tools to handle doing so. Performance testing could be achieved by writing a long test script or adjusting the CommandProcessor to accept multiple script files in succession to achieve a kind of scale we might anticipate in production. For the purposes of this assignment, handling the creation of tens of store objects satisfies the requirements to ensure the Store Model Service works as intended.

Likewise, regression testing is also not handled for this assignment. As the Store Model Service is a standalone module begin developed in isolation for assignment 2, regression testing does not really come into play. As the components get incorporated in later assignments, regression testing might play a larger role to ensure that, for instance, the Store Controller does not change the performance or functionality of the Store Model Service or the Ledger Service from assignment 1.

Exception handling is described in its own section above.

Risks

As with the Ledger Service in assignment 1, security is a big risk that should be taken into account. Hackers may try to infiltrate the system to alter prices, disable Devices, or register fake Customer accounts. The Store Model Service also has to deal with the fact that, althoughs a software abstraction, it is dealing with physical store locations that might be robbed or tampered with to interfere with operations. For example, humans may try to obscure cameras to prevent detection of removal of items.

Also as in assignment 1, the Store Model Service is maintained in memory so the risk of losing store layout information, Customer baskets, and inventory counts, among other pieces of information can be lost should the computer or server malfunction for any reason.