Store Model Service Design Document

Date: 10/3/2021
Author: Burak Ufuktepe
Reviewers: Jiajia Chen, Ellen F Siminoff

Introduction

This document defines the design for the Store Model Service which is responsible for managing domain entities such as the store, inventory, customers, sensors etc. of the Store 24x7 System. Details about Requirements, Class Diagram, Class Dictionary, Design Details, Exception Handling, Testing, and Risks can be found in the following sections.

Overview

The Store 24x7 System fully automates the retail store shopping experience by utilizing sensors and appliances which collect and share data. The Store Model Service maintains the state of these sensors and appliances. Furthermore, appliances such as Turnstiles, Robots, Speakers etc. can be controlled by the Store Model Service.

The Store Model Service provides a public API through which the state of a store can be managed. The API supports commands such as defining the store configuration, creating sensor events, sending command messages to appliances, accessing state of sensors and appliances, monitoring and supporting customers.

Figure 1 shows how the Store Model Service fits into the Store 24x7 System. The Store Model Service provides a service for provisioning stores and controlling appliances. Apart from the Store Model Service, there are three other components of the Store 24x7 System. The Store Controller responds to the events received from the sensors by controlling the appliances. The Ledger Service processes transactions, maintains account balances, and manages blocks that make up the Blockchain. The Authentication Service is responsible for authenticating users and controlling access to the store.

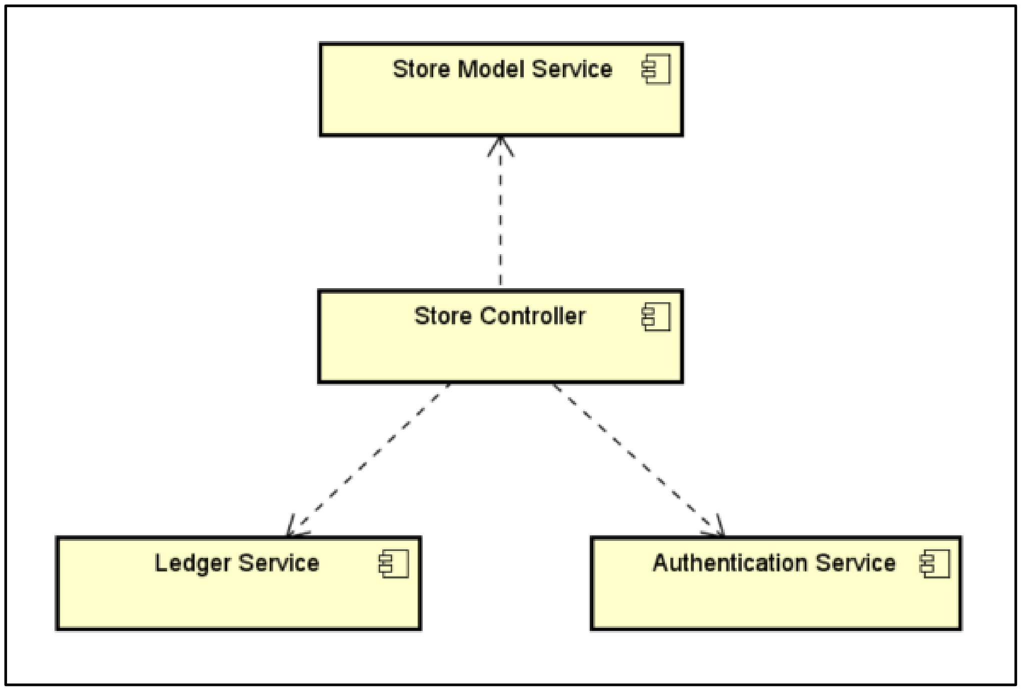


Figure 1: UML Component Diagram for the Store 24x7 System

Requirements

This section provides a summary of the requirements for the Store Model Service. The following store domain objects should be managed by the Store Model Service:

Store: is used to model a store instance which includes inventory, one or more aisles and shelves and zero or more customers, sensors, and appliances. Each store has a globally unique identifier, name and address.

Aisle: represents a location within the store where shelves are placed. Each aisle has a name, number, and description. Aisles can be located either in the storeroom or floor.

Shelf: represents a platform within an aisle where inventory is placed. Each shelf has an identifier, name, level, description, and temperature.

Inventory: defines the products in the store and maintains the count of the product. The location of an inventory is specified with a store aisle and shelf. Each inventory has an id, location, capacity, count, and a product id.

Product: defines the types of products which are placed on shelves as inventory. Products are used to calculate the bill for customers. Each product has an id, name, description, size, category, unit price, and temperature.

Customer: represents a person who may be a registered customer or a guest. Registered customers can be identified by all stores. The location of customers inside the store are determined by cameras and microphones. Each customer has an id, first name, last name, type (registered or guest), email address, blockchain account address, current location, and time last seen property.

Basket: represents a customer’s shopping basket that carries product items. As registered customers enter the store they are assigned a basket. Each basket has an id and a list of products with a product-specific count.

Sensor: represents an IoT device such as a microphone or camera that captures/shares data. Each sensor records a specific type of data and sends it to the Store 24x7 System. Each sensor has a unique id, name, type, and is located within an aisle of the store.

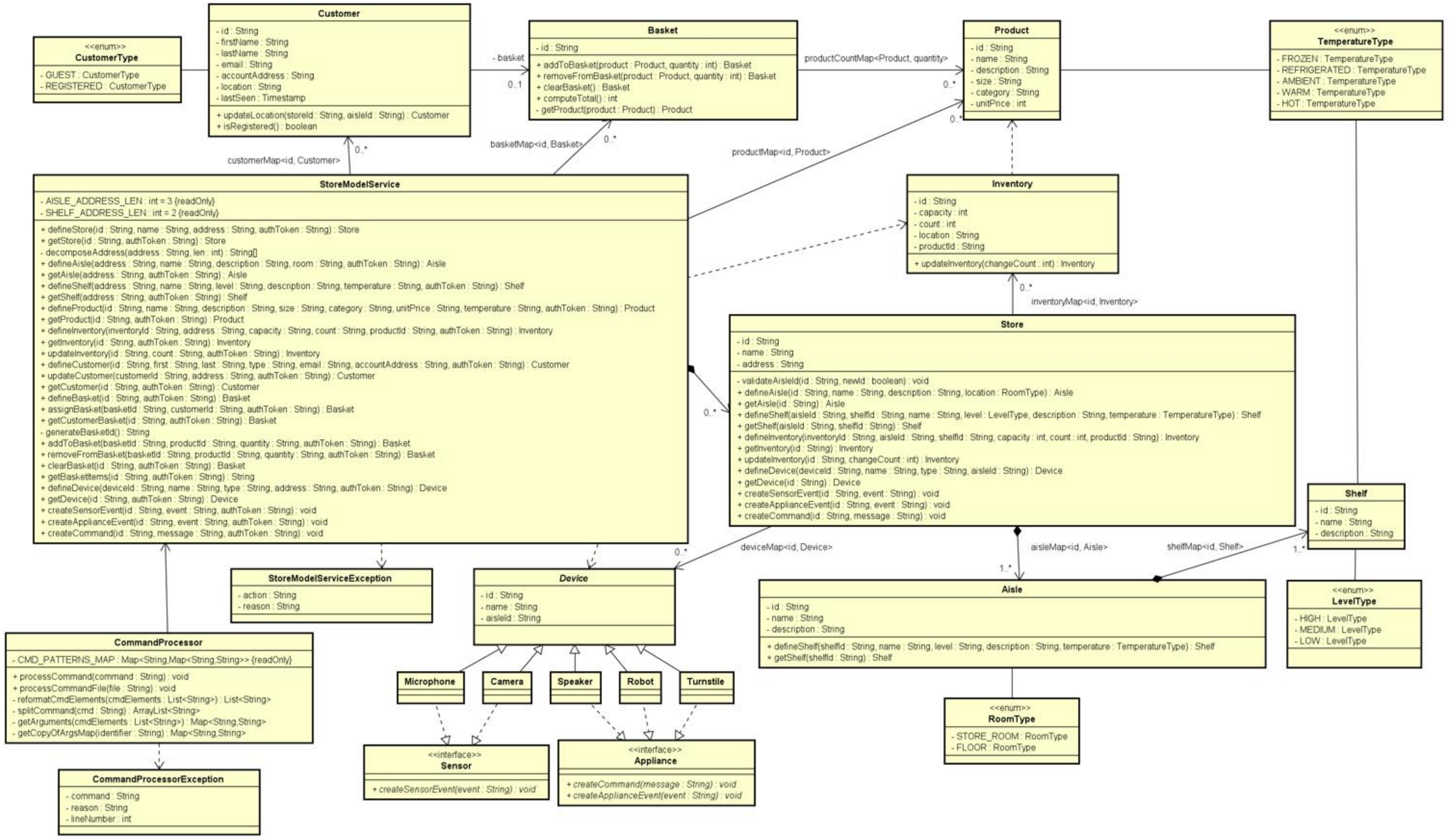
Appliance: represents a device such as a speaker, robot or turnstile that captures/shares data and can also be controlled, i.e. speakers, robots, turnstiles.

Store Model Service: provides a public API for external entities. The following commands should be supported by the Store Model Service:

1. Define a new store configuration, aisle, shelf, inventory, product, customer, sensor or appliance.
2. Show details of a store, aisle, shelf, inventory, product, customer, sensor or appliance.
3. Update inventory count.
4. Update location of a customer.
5. Associate a basket with a customer.
6. Get a customer’s basket and create a new basket for a customer if the customer does not already have a basket.
7. Add/remove a product to/from a basket.
8. Empty a basket and remove the customer association.
9. Get the list of products from a basket including the product id and count.
10. Create a sensor or appliance event.
11. Send a command to an appliance.

Class Diagram

The following class diagram defines the classes defined in this design.



<<enum>>
CustomerType
- GUEST : CustomerType
- REGISTERED : CustomerType
Customer
- id : String
- firstName : String
- lastName : String
- email : String
- accountAddress : String
- location : String
- lastSeen : Timestamp
+ updateLocation(storeId : String, aisleId : String) : Customer
+ isRegistered() : boolean
- basket
0..1
Basket
- id : String
+ addToBasket(product : Product, quantity : int) : Basket
+ removeFromBasket(product : Product, quantity : int) : Basket
+ clearBasket() : Basket
+ computeTotal() : int
- getProduct(product : Product) : Product
productCountMap<Product, quantity>
0..*
Product
- id : String
- name : String
- description : String
- size : String
- category : String
- unitPrice : int
0..*
<<enum>>
TemperatureType
- FROZEN : TemperatureType
- REFRIGERATED : TemperatureType
- AMBIENT : TemperatureType
- WARM : TemperatureType
- HOT : TemperatureType
customerMap<id, Customer>
0..*
basketMap<id, Basket>
0..*
productMap<id, Product>
StoreModelService
- AISLE_ADDRESS_LEN : int = 3 {readOnly}
- SHELF_ADDRESS_LEN : int = 2 {readOnly}
+ defineStore(id : String, name : String, address : String, authToken : String) : Store
+ getStore(id : String, authToken : String) : Store
- decomposeAddress(address : String, len : int) : String[]
+ defineAisle(address : String, name : String, description : String, room : String, authToken : String) : Aisle
+ getAisle(address : String, authToken : String) : Aisle
+ defineShelf(address : String, name : String, level : String, description : String, temperature : String, authToken : String) : Shelf
+ getShelf(address : String, authToken : String) : Shelf
+ defineProduct(id : String, name : String, description : String, size : String, category : String, unitPrice : String, temperature : String, authToken : String) : Product
+ getProduct(id : String, authToken : String) : Product
+ defineInventory(inventoryId : String, address : String, capacity : String, count : String, productId : String, authToken : String) : Inventory
+ getInventory(id : String, authToken : String) : Inventory
+ updateInventory(id : String, count : String, authToken : String) : Inventory
+ defineCustomer(id : String, first : String, last : String, type : String, email : String, accountAddress : String, authToken : String) : Customer
+ updateCustomer(customerId : String, address : String, authToken : String) : Customer
+ getCustomer(id : String, authToken : String) : Customer
+ defineBasket(id : String, authToken : String) : Basket
+ assignBasket(basketId : String, customerId : String, authToken : String) : Basket
+ getCustomerBasket(id : String, authToken : String) : Basket
- generateBasketId() : String
+ addToBasket(basketId : String, productId : String, quantity : String, authToken : String) : Basket
+ removeFromBasket(basketId : String, productId : String, quantity : String, authToken : String) : Basket
+ clearBasket(id : String, authToken : String) : Basket
+ getBasketItems(id : String, authToken : String) : String
+ defineDevice(deviceId : String, name : String, type : String, address : String, authToken : String) : Device
+ getDevice(id : String, authToken : String) : Device
+ createSensorEvent(id : String, event : String, authToken : String) : void
+ createApplianceEvent(id : String, event : String, authToken : String) : void
+ createCommand(id : String, message : String, authToken : String) : void
Inventory
- id : String
- capacity : int
- count : int
- location : String
- productId : String
+ updateInventory(changeCount : int) : Inventory
0..*
inventoryMap<id, Inventory>
0..*
Store
- id : String
- name : String
- address : String
- validateAisleId(id : String, newId : boolean) : void
+ defineAisle(id : String, name : String, description : String, location : RoomType) : Aisle
+ getAisle(id : String) : Aisle
+ defineShelf(aisleId : String, shelfId : String, name : String, level : LevelType, description : String, temperature : TemperatureType) : Shelf
+ getShelf(aisleId : String, shelfId : String) : Shelf
+ defineInventory(inventoryId : String, aisleId : String, shelfId : String, capacity : int, count : int, productId : String) : Inventory
+ getInventory(id : String) : Inventory
+ updateInventory(id : String, changeCount : int) : Inventory
+ defineDevice(deviceId : String, name : String, type : String, aisleId : String) : Device
+ getDevice(id : String) : Device
+ createSensorEvent(id : String, event : String) : void
+ createApplianceEvent(id : String, event : String) : void
+ createCommand(id : String, message : String) : void
Shelf
- id : String
- name : String
- description : String
0..*
deviceMap<id, Device>
aisleMap<id, Aisle>
shelfMap<id, Shelf>
1..*
1..*
StoreModelServiceException
- action : String
- reason : String
Device
- id : String
- name : String
- aisleId : String
Aisle
- id : String
- name : String
- description : String
+ defineShelf(shelfId : String, name : String, level : String, description : String, temperature : TemperatureType) : Shelf
+ getShelf(shelfId : String) : Shelf
<<enum>>
LevelType
- HIGH : LevelType
- MEDIUM : LevelType
- LOW : LevelType
CommandProcessor
- CMD_PATTERNS_MAP : Map<String,Map<String,String>> {readOnly}
+ processCommand(command : String) : void
+ processCommandFile(file : String) : void
- reformatCmdElements(cmdElements : List<String>) : List<String>
- splitCommand(cmd : String) : ArrayList<String>
- getArguments(cmdElements : List<String>) : Map<String,String>
- getCopyOfArgsMap(identifier : String) : Map<String,String>
Microphone
Camera
Speaker
Robot
Turnstile
<<enum>>
RoomType
- STORE_ROOM : RoomType
- FLOOR : RoomType
<<interface>>
Sensor
+ createSensorEvent(event : String) : void
<<interface>>
Appliance
+ createCommand(message : String) : void
+ createApplianceEvent(event : String) : void
CommandProcessorException
- command : String
- reason : String
- lineNumber : int

Class Dictionary

This section specifies the class dictionary for the Store Model Service which is defined within the package cscie97.store.model.

StoreModelService

The Store Model Service provides the API used by clients of the Store Model Service. It contains Stores, Products, Customers, and Baskets. Responsibilities related to Aisle, Shelf, Inventory, and Device operations are passed onto the Store class. It has an inventoryStoreMap and a deviceStoreMap to locate the associated Store for a given inventory Id or device Id. Additionally, it includes a basketCustomerMap to retrieve a customer given a basket Id. All StoreModelService methods include an authToken parameter that will be later used to support access control.

Methods

Method Name	**Signature**	**Description**
defineStore	(id:String, name:String, address:String, authToken:String):Store	Validate the given store ID and then create a new Store for the given store Id, name, and store address. Add the store to storeMap.
getStore	(id:String, authToken:String):Store	Validate the given store ID and then return the Store for the given store Id.
decomposeAddress	(address:String, int:len):String[]	Decompose the given address string which is in the form of storeId:aisleId or storeId:aisleId:shelfId into a String array. The len parameter indicates the number of components in the address string that are separated by colons.

defineAisle	(address:String, name:String, description:String, room: String, authToken:String):Aisle	Create a new Aisle in a store given an address (storeId:aisleId), name, description, and room.
getAisle	(address:String, authToken:String):Aisle	Validate the given address and then return the Aisle for the given address (storeId:aisleId).
defineShelf	(address:String, name:String, level:String, description:String, temperature: String, authToken:String):Shelf	Validate the given address, level, and temperature and then create a new Shelf in an aisle in a store for the given address (storeId:aisleId:shelfId), name, level, description, and temperature.
getShelf	(address:String, authToken:String):Shelf	Validate the given address and then return the Shelf for the given address (storeId:aisleId:shelfId).
defineProduct	(id:String, name:String, description:String, size:String, category:String, unitPrice:String, temperature: String, authToken:String):Product	Validate the given product ID, unit price, and temperature and then create a new Product for the given product Id, name, description, size, category, unit price, and temperature. Add the Product to productMap.
getProduct	(id:String, authToken:String):Product	Validate the given product

		ID and then return the Product.
defineInventory	(inventoryId:String, address:String, capacity:String, count:String, productId:String, authToken:String):Inventory	Validate the given address, inventory ID, product ID, capacity, and count, Then create a new Inventory. Add the new inventory to inventoryStoreMap.
getInventory	(id:String, authToken:String):Inventory	Validate the given inventory ID and then return the Inventory.
updateInventory	(id:String, count:String, authToken:String):Inventory	Validate the given inventory ID and count. Then increment or decrement the inventory count for the given inventory Id.
defineCustomer	(id:String, first:String, last:String, type:String, email:String, accountAddress:String, authToken:String):Customer	Validate the customer ID and type. Then create a new Customer for the given customer ID, first name, last name, email, and blockchain account address.
updateCustomer	(customerId:String, address:String, authToken:String):Customer	Validate the given address, customer ID, and aisle ID. Then update the location of the customer for the given address (storeId:aisleId).
getCustomer	(id:String, authToken:String):Customer	Validate the customer ID and then return the

		Customer object for the given customer ID.
defineBasket	(id:String, authToken:String):Basket	Validate the given basket ID. Then create a new basket and add it to basketMap.
assignBasket	(basketId:String, customerId:String, authToken:String):Basket	Associate the given basket with the given customer. Throw a StoreModelServiceException if any of the following conditions are satisfied: - The given customer ID is invalid. - The given basket ID is invalid. - The given customer is not a registered customer. - The given customer already has a basket. - The given basket is already assigned to a customer.
getCustomerBasket	(id:String, authToken:String):Basket	Validate the given customer ID, check if registered and return the customer's basket. If the customer does not already have a basket, assign a new basket to the customer.
generateBasketId	():String	Generate a unique basket

		ID.
addToBasket	(basketId:String, productId:String, quantity:String, authToken:String):Basket	Validate the given basket ID, product ID, and quantity. Then add the product item to the basket.
removeFromBasket	(basketId:String, productId:String, quantity:String, authToken:String):Basket	Validate the given basket ID, product ID, and quantity. Then remove the product item from the basket.
clearBasket	(id:String, authToken:String):Basket	Validate the given basket ID. Then set the customer's basket to null, remove the basket-customer association and clear the contents of the basket.
getBasketItems	(id:String, authToken:String):String	Validate the given basket ID and check if it is assigned to a customer. Return the contents of the basket as text in a list format.
defineDevice	(deviceId:String, name:String, type:String, address:String, authToken:String):Device	Validate the given device ID and address. Then create a new Device for the given device Id, name, type, and address (storeId:aisleId).
getDevice	(id:String, authToken:String):Device	Validate the given device ID and then return the Device object for the given device

		ID.
createSensorEvent	(id:String, event:String, authToken:String):void	Create a Sensor event given a device ID and an event description. (Placeholder for next assignment)
createApplianceEvent	(id:String, event:String, authToken:String):void	Create an Appliance event given a device ID and an event description. (Placeholder for next assignment)
createCommand	(id:String, message:String, authToken:String):void	Send a command to an Appliance given a device ID and a message. (Placeholder for next assignment)

Properties

Property Name	**Type**	**Description**
AISLE_ADDRESS_LEN	int	Number of components in an aisle address (storeId:aisleId).
SHELF_ADDRESS_LEN	int	Number of components in a shelf address (storeId:aisleId:shelfId).

Associations

Association Name	**Type**	**Description**
storeMap	Map<String, Store>	Map of Store Ids and Store objects.

customerMap	Map<String, Customer>	Map of Customer Ids and Customer objects.
productMap	Map<String, Product>	Map of Product Ids and Product objects.
basketMap	Map<String, Basket>	Map of Basket Ids and Basket objects.
inventoryStoreMap	Map<String, Store>	Map of Inventory Ids and Store objects. Used to locate the Store given an Inventory Id.
deviceStoreMap	Map<String, Store>	Map of Device Ids and Store objects. Used to locate the Store given a Device Id.
basketCustomerMap	Map<String, Customer>	Map of Basket Ids and associated Customer objects. Used to locate the Customer given a Basket Id.

StoreModelServiceException

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

Properties

Method Name	**Signature**	**Description**
action	String	Action that was performed.
reason	String	Reason for the exception.

Store

The Store class contains Aisle, Inventory, and Device objects and performs all operations related to Aisles, Inventories, and Devices. The responsibility of Shelf operations is passed on to the

Aisle class. The Store class has properties for id, name, and address.

Methods

Method Name	Signature	Description
validateAisleId	(id:String, newId:boolean):void	Validates a given aisle ID for uniqueness or existence. Throws an IllegalArgumentException for the following conditions: - If newId is true and the given ID is not unique. - If newId is false and the given ID doesn't exist.
defineAisle	(id:String, name:String, description:String, location: RoomType):Aisle	Validate the given aisle ID and then create a new Aisle for the given aisle Id, name, description, and location. Add the Aisle to aisleMap.
getAisle	(id:String):Aisle	Validate the given aisle ID and then return the Aisle object for the given aisle Id.
defineShelf	(aisleId:String, shelfId:String, name:String, level:LevelType, description:String, temperature:TemperatureType):Shelf	Validate the given aisle ID and then create a new Shelf in an aisle for the given aisle Id, shelf Id, name, level, description, and temperature.
getShelf	(aisleId:String, shelfId:String):Shelf	Validate the given aisle ID and then return the Shelf object for the given aisle Id, and shelf Id.

defineInventory	(inventoryId:String, aisleId:String, shelfId:String, capacity:int, count:int, productId:String):Inventory	Validate the given aisle ID and shelf ID and then create a new Inventory for the given inventory Id, aisle Id, shelf Id, capacity, count, and product Id. Add the Inventory to inventoryMap.
getInventory	(id:String):Inventory	Return the Inventory for the given inventory Id.
updateInventory	(id:String, changeCount:int):Inventory	Increment or decrement the inventory count for the given inventory Id. A positive changeCount indicates an increment whereas a negative changeCount indicates a decrement.
defineDevice	(deviceId:String, name:String, type:String, aisleId:String):Device	Create a new Device (camera, microphone, robot, speaker, or turnstile) given a device Id, name, type, and aisle Id.
getDevice	(id:String):Device	Return the Device for the given device Id.
createSensorEvent	(id:String, event:String):void	Create a Sensor event given a device ID and an event description. (Placeholder for next assignment)
createApplianceEvent	(id:String, event:String):void	Create an Appliance event given a device Id and an event description. (Placeholder for next assignment)

createCommand	(id:String, message:String):void	Send a command to an Appliance given a device Id and a message. (Placeholder for next assignment)

Properties

Property Name	Type	Description
id	String	Unique identifier for the store.
name	String	Name of the store.
address	String	Address of the store.

Associations

Association Name	Type	Description
aisleMap	Map<String, Aisle>	Map of Aisle numbers and Aisle objects.
inventoryMap	Map<String, Inventory>	Map of Inventory Ids and Inventory objects.
deviceMap	Map<String, Device>	Map of Device Ids and Device objects.

Aisle

The Aisle class represents aisles within the store where shelves are placed. It contains Shelf objects and has properties for id, name, description, and location. The location can take values STORE_ROOM or FLOOR. Aisles are maintained by the Store class.

Methods

Method Name	Signature	Description

defineShelf	(shelfId:String, name:String, level:LevelType, description:String, temperature: TemperatureType):Shelf	Create a new Shelf given a shelf Id, name, level, description, and temperature. Add the Shelf to shelfMap. Throw an IllegalArgumentException if shelf Id is not unique. Level is of type LevelType which is an enum that can take values HIGH, MEDIUM, LOW. Temperature is of type TemperatureType which is also an enum and can take values FROZEN, REFRIGERATED, AMBIENT, WARM, HOT.
getShelf	(shelfId:String):Shelf	Return the Shelf for the given shelf Id. Throw an IllegalArgumentException if the shelf Id does not exist.

Properties

Property Name	Type	Description
id	String	Unique identifier for the aisle that is unique within a store.
name	String	Name of the aisle.
description	String	Aisle description.
location	RoomType (enum)	Location of the aisle. RoomType is an enum that has values STORE_ROOM or FLOOR.

Associations

Association Name	Type	Description
shelfMap	Map<String, Shelf>	Map of shelf IDs and Shelf objects. Each aisle includes

		at least one Shelf.

Shelf

The Shelf class represents a platform within an aisle where inventories are placed. Shelf objects are maintained by the Aisle class.

Properties

Property Name	Type	Description
id	String	Unique identifier for the shelf that is unique within an aisle.
name	String	Name of the shelf.
level	LevelType (enum)	Height of the shelf. LevelType is an enum that has values HIGH, MEDIUM, LOW.
description	String	Description of shelf contents.
temperature	TemperatureType (enum)	Temperature of the shelf. TemperatureType is an enum that has values FROZEN, REFRIGERATED, AMBIENT, WARM, HOT.

Inventory

The Inventory class helps identify the location of products within the Store. An Inventory contains a single product type and has a location in the form of storeId:aisleId:shelfId. Each inventory has a capacity which identifies the maximum number of products that can fit on the shelf. It also maintains the count of products which must remain >= 0 and <= capacity. Inventories are maintained by the Store class.

Methods

Method Name	Signature	Description

updateInventory	(changeCount:int):Inventory	Increment or decrement the inventory count. changeCount can be either positive or negative. Throw an IllegalArgumentException if the number of product items exceeds shelf capacity or if the new quantity becomes negative. Return the Inventory.

Properties

Property Name	Type	Description
id	String	Globally unique identifier for the inventory.
capacity	int	Total capacity of the inventory on the shelf. Must be a positive integer.
count	int	Product count that is less than or equal to the capacity and non-negative.
location	String	Location of the inventory in the form of storeId:aisleId:shelfId.
productId	String	Product ID of the product in the inventory.

Product

The Product class represents the products available for sale within the store. Products sit on shelves as inventory and they can be placed in shopping baskets by Customers. Products are managed by the Store Model Service.

Properties

Property Name	Type	Description
id	String	Globally unique identifier for the product.

name	String	Name of the product.
description	String	Description of the product.
size	String	Weight or volume of the product.
category	String	Type of product.
unitPrice	int	Unit price in blockchain currency.
temperature	TemperatureType	Storage temperature of the product. TemperatureType is an enum that has values FROZEN, REFRIGERATED, AMBIENT, WARM, HOT.

Customer

The Customer class represents a registered or guest customer. Guest customers cannot remove items from the store. Each customer has a current location in the form of storeId:aisleId and a time last seen property based on the time of last location update. A registered customer also has a basket which is assigned as the customer enters the store. Customers are maintained by the StoreModelService class.

Methods

Method Name	**Signature**	**Description**
isRegistered	():boolean	Returns true if the customer is registered. Otherwise, it returns false.
updateLocation	(storeId:String, aisleId:String):Customer	Update the location given a store Id, and aisle Id. Updates time last seen as well.

Properties

Property Name	Type	Description
id	String	Globally unique identifier for the customer.
firstName	String	First name of the customer.
lastName	String	Last Name of the customer.
type	CustomerType	Type of the customer. CustomerType is an enum which can take values GUEST or REGISTERED.
email	String	Email of the customer.
accountAddress	String	Blockchain account of the customer.
currentLocation	String	Location of the Customer in the form of storeId:aisleId.
lastSeen	Timestamp	Time last seen based on the time of last location update.

Associations

Association Name	Type	Description
basket	Basket	Basket for registered customers. Each customer can have at most one basket.

Basket

The Basket class represents a shopping basket which is assigned to registered customers. Each basket has a globally unique identifier and contains products. The products are stored in productCountMap which maps Product objects to their count in the basket. Baskets are maintained by the StoreModelService class.

Methods

Method Name	Signature	Description
addToBasket	(product:Product, quantity:int):Basket	Add a product item to a basket. Quantity must be a positive integer which represents the number of products to be added to the basket. Throw an IllegalArgumentException if quantity is not positive.
removeFromBasket	(product:Product, quantity:int):Basket	Remove a product item from the basket. The product must already exist in the basket. Quantity must be a positive integer which represents the number of products to be removed from the basket. Throw an IllegalArgumentException if input quantity is negative, if the new quantity after removing the product becomes negative or if the product does not exist in the basket.
clearBasket	():Basket	Clear the contents of the basket.
computeTotal	():int	Compute the total bill for the basket.
getProduct	(product:Product):Product	Return the Product object in productCountMap given a Product object by matching their product Ids. Return null if no match is found.

Properties

Property Name	Type	Description

id	String	Globally unique identifier for the basket.

Associations

Association Name	Type	Description
productCountMap	Map<Product, Integer>	Map of Product objects and their count in the basket.

Device

Device is an abstract class. Sensors (i.e. microphones, cameras) and appliances (i.e. speakers, robots, turnstiles) inherit from the Device class. A device has an id, name, and a location. Devices are maintained by the Store class.

Properties

Property Name	Type	Description
id	String	Globally unique identifier for the device.
name	String	Name of the device.
aisleId	String	Id of the Aisle where the device is located.

Sensor (Interface)

Sensor is an interface which is implemented by sensors such as microphones and cameras. It defines a method signature for simulating sensor events. The event is treated as a placeholder for an actual event.

Methods

Method Name	Signature	Description
createSensorEvent	(event:String):void	Simulate a Sensor event.

Appliance (Interface)

Appliance is an interface which is implemented by appliances such as speakers, robots, turnstiles. It defines method signatures for simulating appliance events and sending commands to appliances. The event and message are treated as placeholders for actual events and messages.

Methods

Method Name	Signature	Description
createApplianceEvent	(event:String):void	Simulate an Appliance event.
createCommand	(message:String):void	Send the appliance a command.

CustomerType (enum)

Identifies the type of customer as guest or registered.

Properties

Property Name	Type	Description
GUEST	CustomerType	Guest customer.
REGISTERED	CustomerType	Registered customer.

LevelType (enum)

Identifies the height of a shelf as high, medium, or low.

Properties

Property Name	Type	Description
HIGH	LevelType	High shelf.

MEDIUM	LevelType	Medium shelf.
LOW	LevelType	Low shelf.

RoomType (enum)

Identifies the room of a store as store room or floor.

Properties

Property Name	Type	Description
STORE_ROOM	RoomType	Room of a store where inventory is stored.
FLOOR	RoomType	Room of a store where products are displayed for sale.

SensorType (enum)

Identifies the type of sensor as microphone or camera.

Properties

Property Name	Type	Description
MICROPHONE	SensorType	Microphone device.
CAMERA	SensorType	Camera device.

TemperatureType (enum)

Identifies the temperature of a shelf or storage temperature of a product.

Properties

Property Name	Type	Description

FROZEN	TemperatureType	Temperature below 32°F.
REFRIGERATED	TemperatureType	Temperature at 40°F.
AMBIENT	TemperatureType	Temperature at 73°F.
WARM	TemperatureType	Temperature at 140°F.
HOT	TemperatureType	Temperature above 140°F.

CommandProcessor

The CommandProcessor is a utility class for feeding the Store Model Service a set of operations, using command syntax.

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 a CommandProcessorException on error.
processCommandFile	(commandFile:String):void	Process a set of commands provided within the given command file. Throw a CommandProcessorException on error.
reformatCmdElements	(List<String>):List<String>	Reformat the commands list to ensure that the first element in the list corresponds to a valid command.
splitCommand	(cmd:String):ArrayList<String>	Split the command by whitespace. Keeps the strings between quotation marks as is.

getArguments	(cmdElements:List<String>):Map<String, String>	Extract arguments from a list of commands.
getCopyOfArgsMap	(identifier:String):Map<String, String>	Deep copy a Map in CMD_PATTERNS_MAP. The identifier argument is used to identify the command in CMD_PATTERNS_MAP.

Properties

Property Name	Type	Description
CMD_PATTERNS_MAP	Map<String, Map<String, String>>	Map of command patterns which are used for parsing commands.

Associations

Association Name	Type	Description
storeModelService	StoreModelService	Store Model Service instance that is used to call Store Model Service API methods.

CommandProcessorException

The CommandProcessorException is returned from the CommandProcessor methods in response to an error condition.

Properties

Method Name	Signature	Description
command	String	Command that was performed.
reason	String	Reason for the exception.
lineNumber	int	The line number of the command in the input file.

Design Details

The StoreModelService class is the core of the Store Model Service. It provides the API for interacting with the service and implements all API methods. The StoreModelService class manages Stores, Customers, Products, and Baskets. It also includes two maps called inventoryStoreMap and deviceStoreMap to locate the associated Store when it receives a request to show details of an Inventory or Device.

The Store class manages Inventories, Aisles, and Devices. Since there is a composition relationship between the Aisle class and the Shelf class, the Aisle class manages Shelves. When the StoreModelService receives a request to show details of a Shelf, given a storeId:aisleId:shelfId, the StoreModelService calls the associated Store. Then the Store calls the associated Aisle and the Aisle returns the Shelf object. This way the client is decoupled from the implementation. A similar approach is used to retrieve Inventory and Device objects.

Each device has an id, location, and name attribute. Therefore, devices such as microphones, cameras, speakers, robots and turnstiles inherit from the Device abstract class. Microphones and cameras will likely have similar functionality. Hence, they implement the Sensor interface. Similarly, appliances such as speakers, robots, turnstiles will have similar functionality. Based on their functionality, they may implement the Appliance interface and possibly the Sensor interface as well.

Exception Handling

The error conditions in the CommandProcessor methods result in a CommandProcessorException. The CommandProcessorException contains the failed command and the reason for the failure. Also, the line number of the command is included if the commands are read from a file.

For validating method arguments, IllegalArgumentExceptions are used and these exceptions are caught by the StoreModelService class. The StoreModelService class gets the message from these exceptions and throws StoreModelServiceExceptions.

StoreModelServiceExceptions can also result from error conditions in the Store Model Service methods. It captures the attempted action and the reason for failure.

Testing

A test driver class is implemented with a static main() method which accepts a command file. The main() method calls the processCommandFile(file:string) method of the CommandProcessor. The test command file includes a sample store configuration and sample queries.

Risks

The system does not have any knowledge of Shelf capacity. This might lead to situations where a Shelf may contain more inventory than it can hold. To address this issue, a property for shelf capacity should be included in the Shelf class and the current available space should be checked before adding inventory to any Shelf.