Milestone Report for Grocery Store

Group 1

Jaeden Carpenter, Malcolm Hayes, Sam Kerns

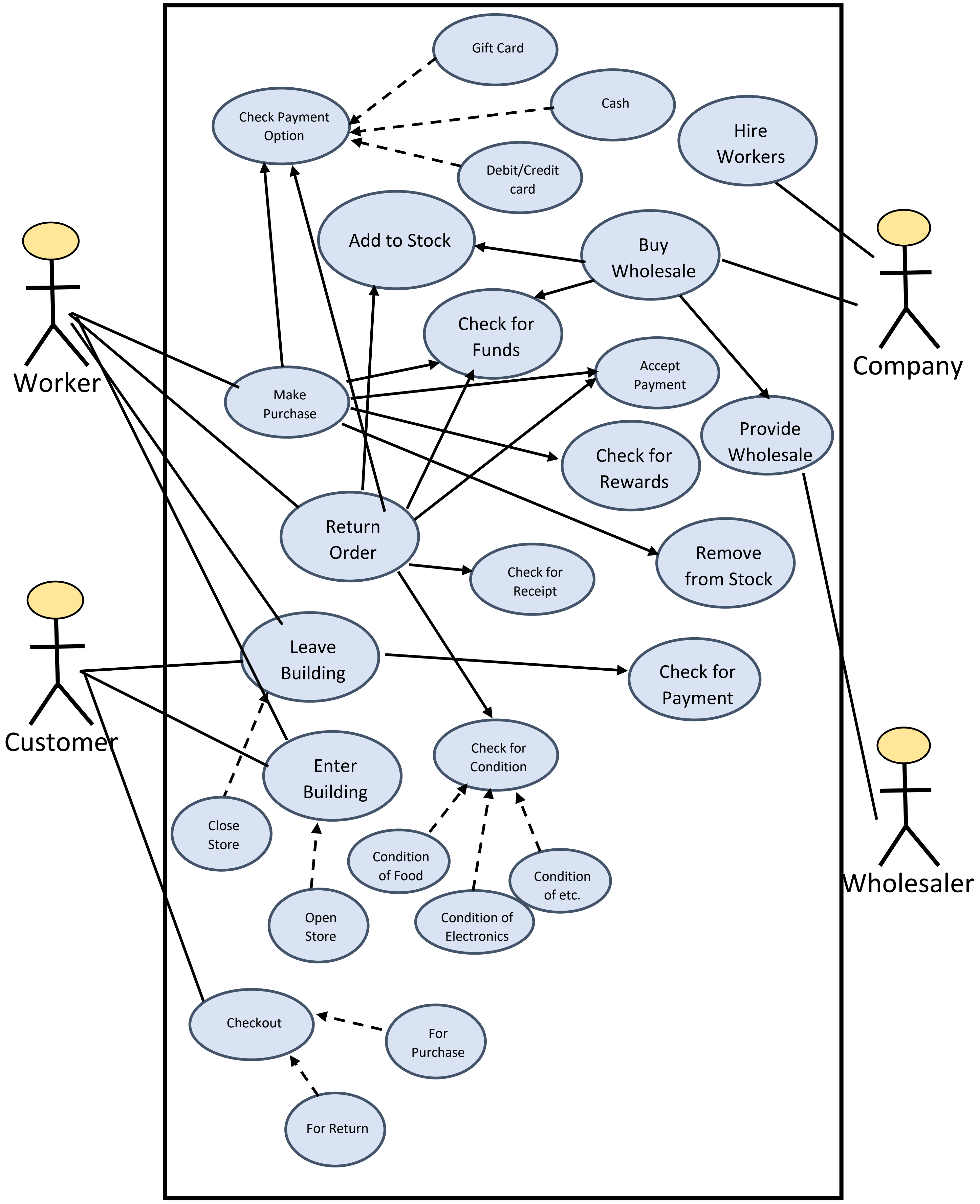
November 17, 2022

YouTube Link:

https://youtu.be/l7Os3_S5mQU

Table of Contents

Solid arrows are includes. Dotted arrows are extends.



Use Case Name: Check for Funds

Actors: Customer, Worker, Company, Wholesaler

Preconditions: A purchase is being made. A payment option has been chosen if it is a customer purchase.

Description: When a purchase is being made, the funds available are checked to see if they are sufficient or not. If they are sufficient, the payment is accepted, if not, there is an error.

Exceptions: *System is Not On:* If the system is not turned on then the check for payment fails and the purchase is not accepted.

Post conditions: An error message is displayed if the funds are not sufficient. The purchase is accepted if the funds are sufficient. System waits for next input from make purchase.

Use Case Name: Checkout

Actors: Customer, worker

Preconditions: All items the customer needs have been taken from the shelves or the customer has the necessary items for a return. The customer is in the store.

Description: The customer may checkout when they want to return items or purchase items. After they checkout, the system waits for the worker to make purchase or to make return.

Exceptions: *Customer Has No Items:* An error appears of the customer does not have any items, and the system waits for customer action.

Post conditions: The customer has provided their items for checkout. The system is waiting for worker action.

Use Case Name: Check Payment Option

Actors: Customer, Worker

Preconditions: A purchase or return is being made and a method of payment is required.

Description: Whenever a return or purchase is being done, the customer must provide their method of payment for providing or receiving payment.

Exceptions: *No Payment Option Available:* An error appears when the customer does not have a method of payment. System waits for worker action.

Post conditions: The method of payment has been chosen. System waits for next worker action.

Use Case Name: Add to Stock

Actors: Worker, Company

Preconditions: Items have been selected and are prepared to be added to stock.

Description: When a customer returns an item or a company purchases wholesale, the items get added to the stock of the store.

Exceptions: *Item not Listed in Stock:* An error appears when trying to add an item to the stock when there is no place in stock available for the listed item. System waits for worker or company action.

Post conditions: The stock shows the items as added. System waits for next worker or customer action.

Use Case Name: Return Order

Actors: Customer, Worker, Company

Preconditions: All items have been scanned. The customer has method of accepting payment. System is waiting for the next worker action.

Description: The customer provides the receipt and the items needing to be returned. If the items are still in acceptable condition and within the return date. The worker completes the return, funds are taken from the company and given to the customer, and the merchandise is added to the stock if it is viable to be resold.

Exceptions: *Nonviable Condition:* An error appears when the condition of the item is not suitable for return. This condition depends on the type of item being returned. System waits for worker action.

Insufficient Funds: An error message appears when the company does not have enough funds to make the purchase. System waits for worker action.

No Receipt: If items are attempted to be returned without a receipt, an error message appears. System waits for worker action.

Post conditions: The stock shows the items as added. The store loses money equal to the items' value. Customer gains money equal to the items' value. The system waits for next worker action.

Use Case Name: Make Purchase

Actors: Customer, Worker, Company

Preconditions: All items have been scanned. The customer has a method of payment. System is waiting for the next worker action. The Customer checks out.

Description: The customers funds are checked to see if they have enough for the purchase. If the customer is a rewards member, they get a discount. The worker completes the purchase, the funds are taken from the customer and added to the company, and the merchandise is removed from stock.

Exceptions: *Insufficient Funds:* An error message appears when the customer does not have enough funds to make the purchase. System waits for worker action.

Post conditions: The stock shows the items as removed. The store gains money equal to the money spent. Customer loses money equal to money spent. The system waits for next worker action.

Use Case Name: Purchase Wholesale

Actors: Company, Wholesaler

Preconditions: Wholesaler has provided the items that the company has asked for. The company has sufficient funds. System is waiting for the next company action.

Description: The wholesaler provides wholesale items, and the company provides the proper funds in order to purchase the items. The items are then added to the company's stock.

Exceptions: *Insufficient Funds:* An error message appears when the company does not have enough funds to make the purchase. System waits for sufficient funds. System waits for company action.

Incorrect Wholesale: An error message appears when the provided wholesale is not the same as the requested wholesale. The company may choose to abort the purchase. System waits for company action.

Post conditions: The stock shows the items as added. The company loses money equal to the money spent. The system waits for next company action.

Use Case Name: Leave Building

Actors: Customer, Worker

Preconditions: Customer or worker is at the exit door. All items carried have been purchased. The building has been opened. System is waiting for the next worker or customer action.

Description: The customer or worker may choose to leave the building and will exit once it is made sure that all carried items have been purchased and they are at the exit door. The worker may choose to close the store.

Exceptions: *Item not Purchased:* An error appears and the person leaving is unable to leave until they remove the items or purchase the items. System waits for next worker or customer action.

Worker does not lock the door: If the worker attempts to close, but the door is not locked, an error appears reminding the worker to lock the door. System waits for worker action.

Post conditions: The worker or customer has left the building. If the worker closes the building, then the lights are off, the doors are locked, and the security alarm is set. The system waits for next worker or customer action.

Use Case Name: Enter Building

Actors: Customer, Worker

Preconditions: Customer or worker is at the entrance door. The building has been opened. The customer is following the proper dress code. System is waiting for the next customer or worker action.

Description: The customer or worker enters the building through the entrance door and is now able to access the internal systems within the building. The worker must open the store if it is closed.

Exceptions: *Store is closed:* The customer is unable to enter the store. The worker may choose to open the store. The system is waiting for worker action.

Improper Dress Code: An error appears when the customer does not follow the proper dress code and they are unable to enter the building.

Post conditions: The customer or worker is inside the building. If the worker opens the building, then the lights are on, the security system is deactivated, and the doors are unlocked. The system waits for next worker or customer action.

Use Case Name: Hire Worker

Actors: Company

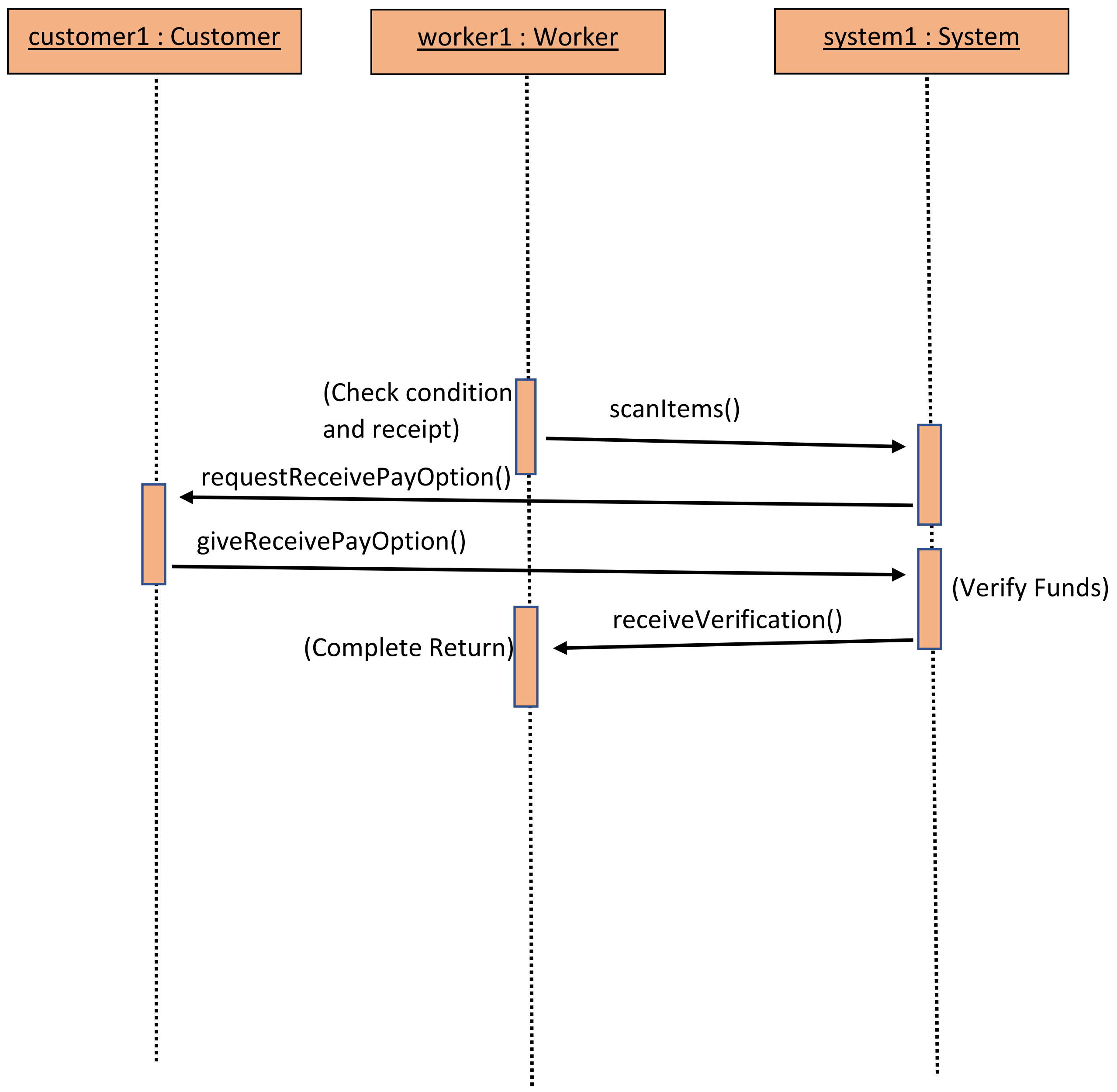
Preconditions: A worker is available.

Description: The company hires a worker, and the worker is added to the company's worker list.

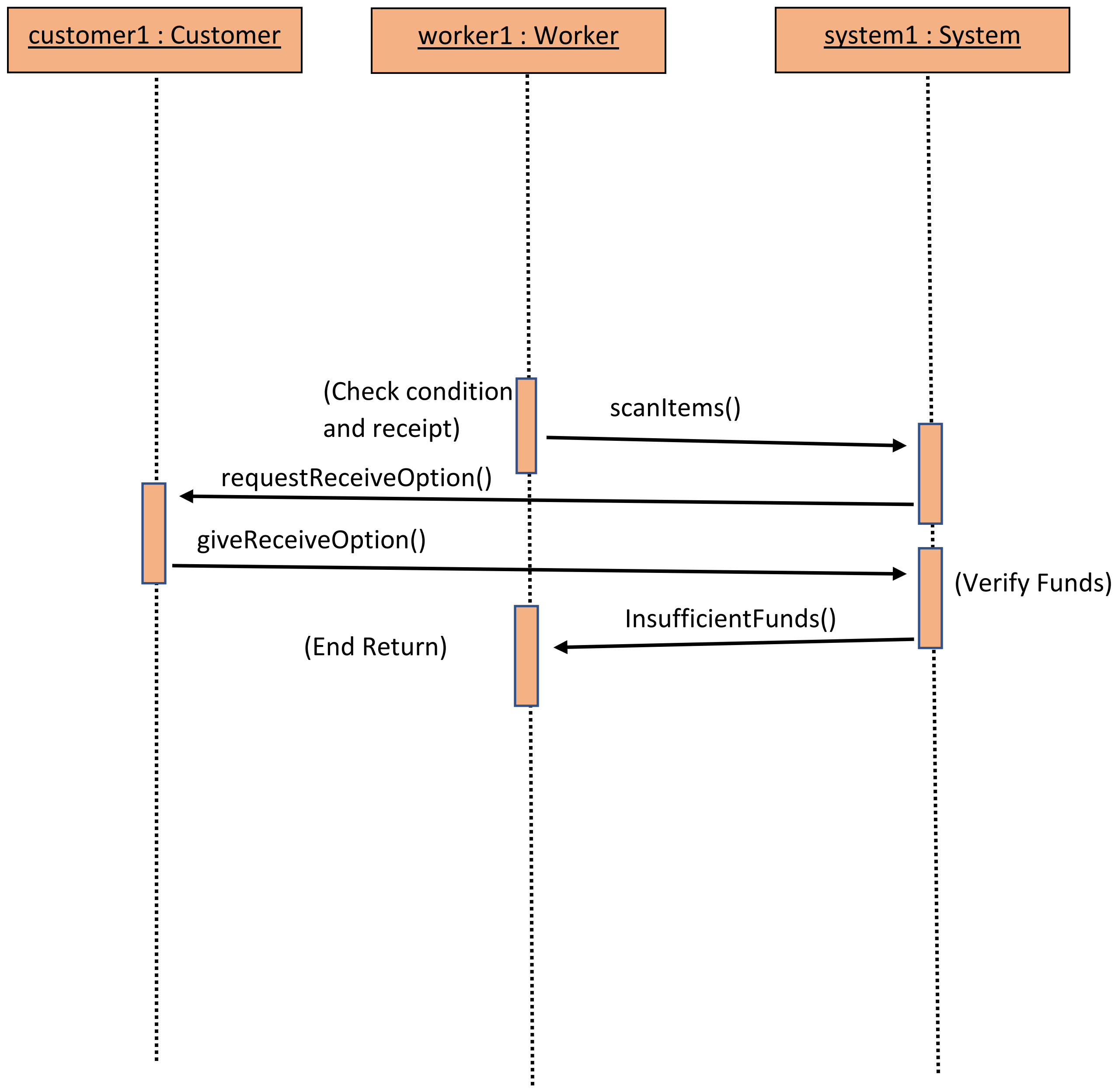
Exceptions: *No Worker Available*: The worker is not added, and the company is returned to waiting for action.

Post conditions: The worker is added to the company's worker list.

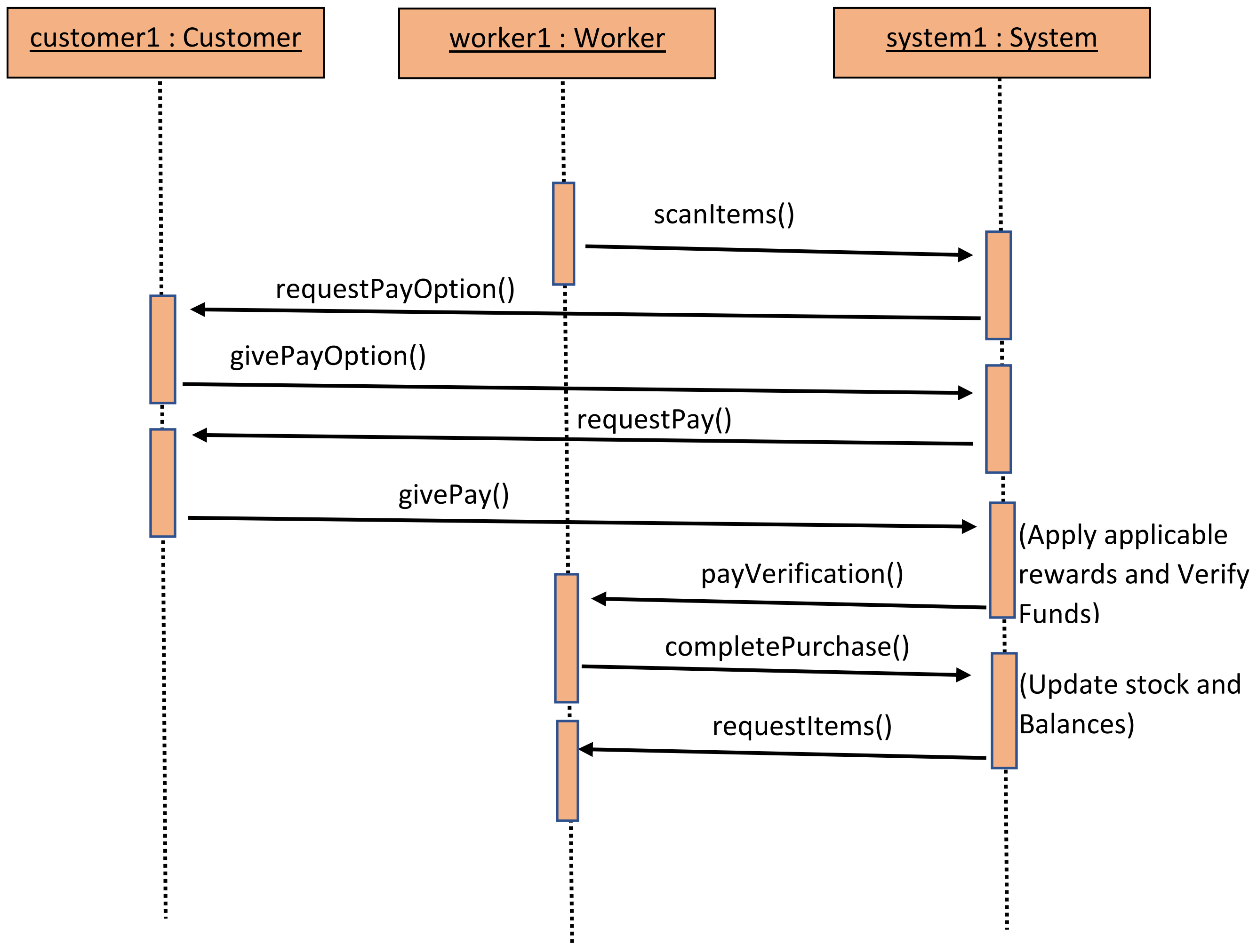
Sequence diagram for Return Order



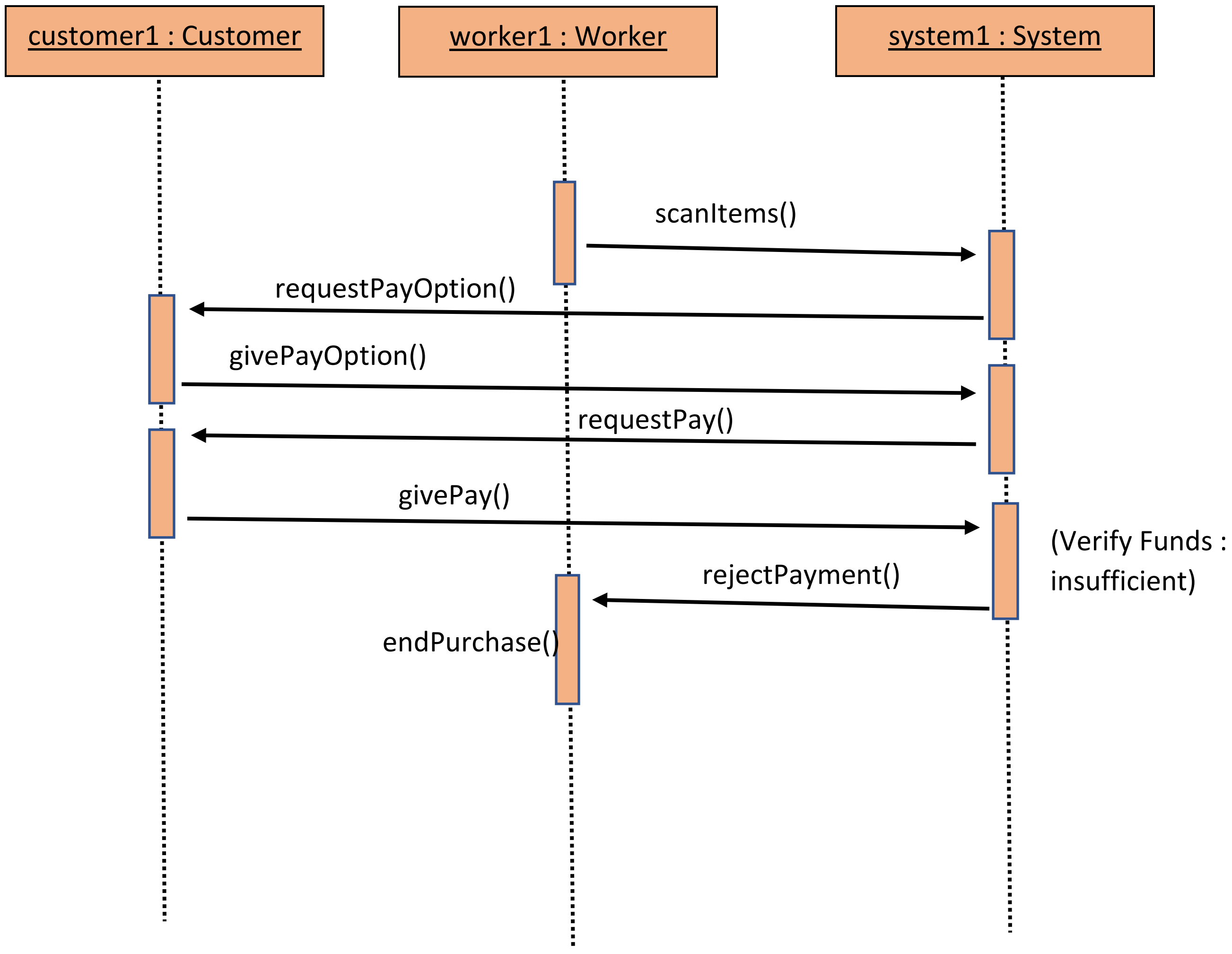
Sequence diagram for Return Order with insufficient funds



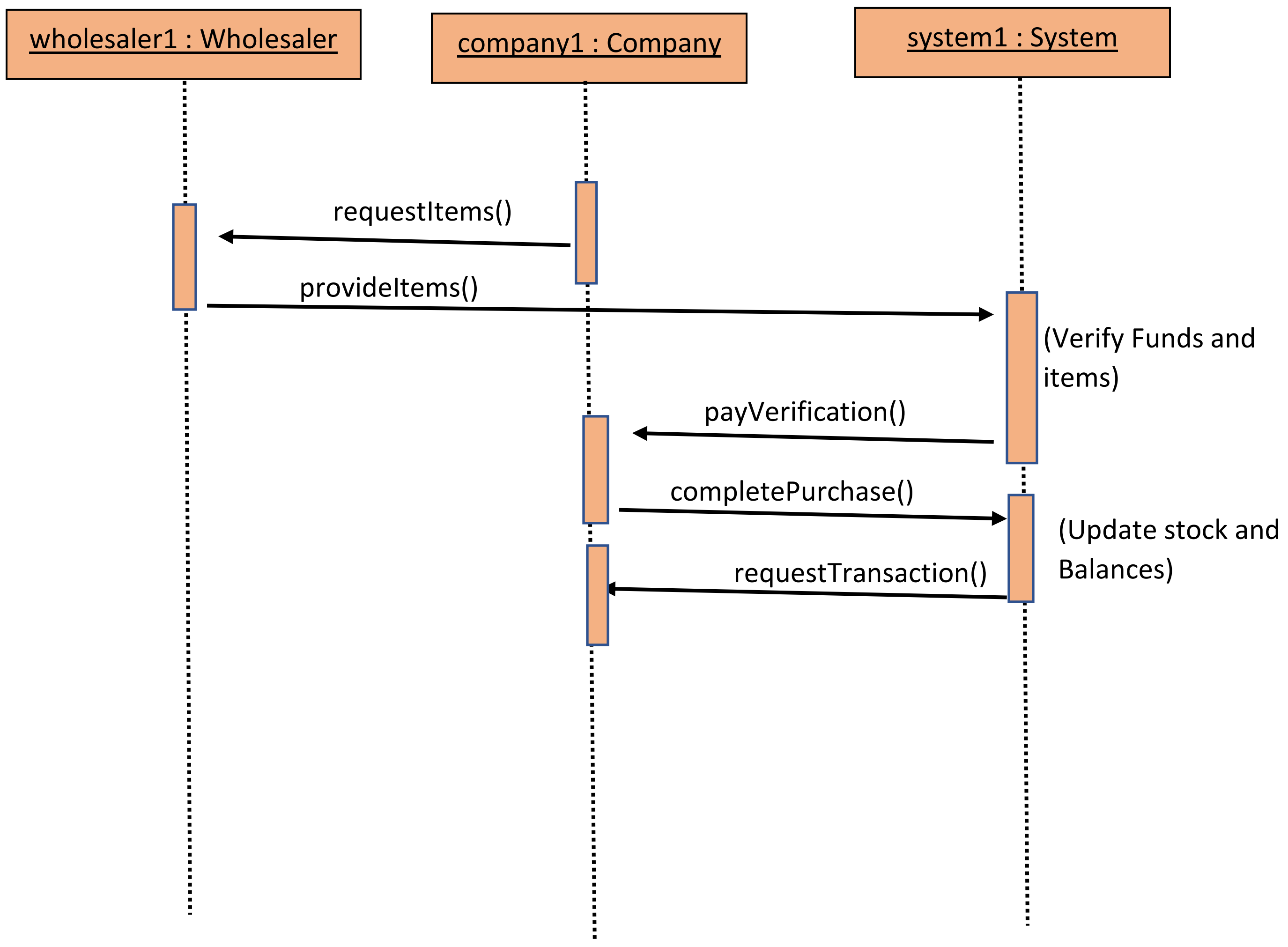
Sequence diagram for Make Purchase



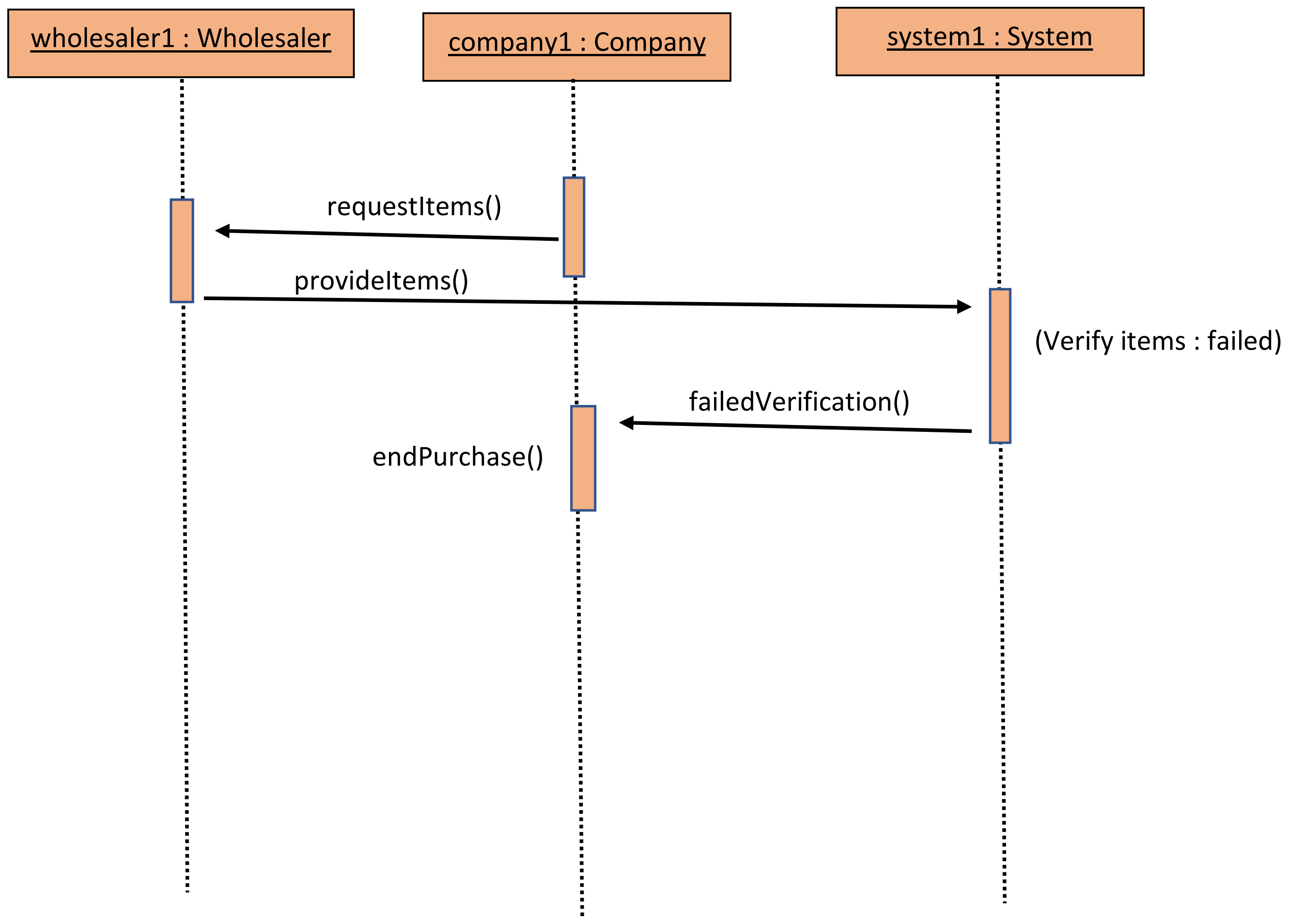
Sequence diagram for Make Purchase with insufficient funds



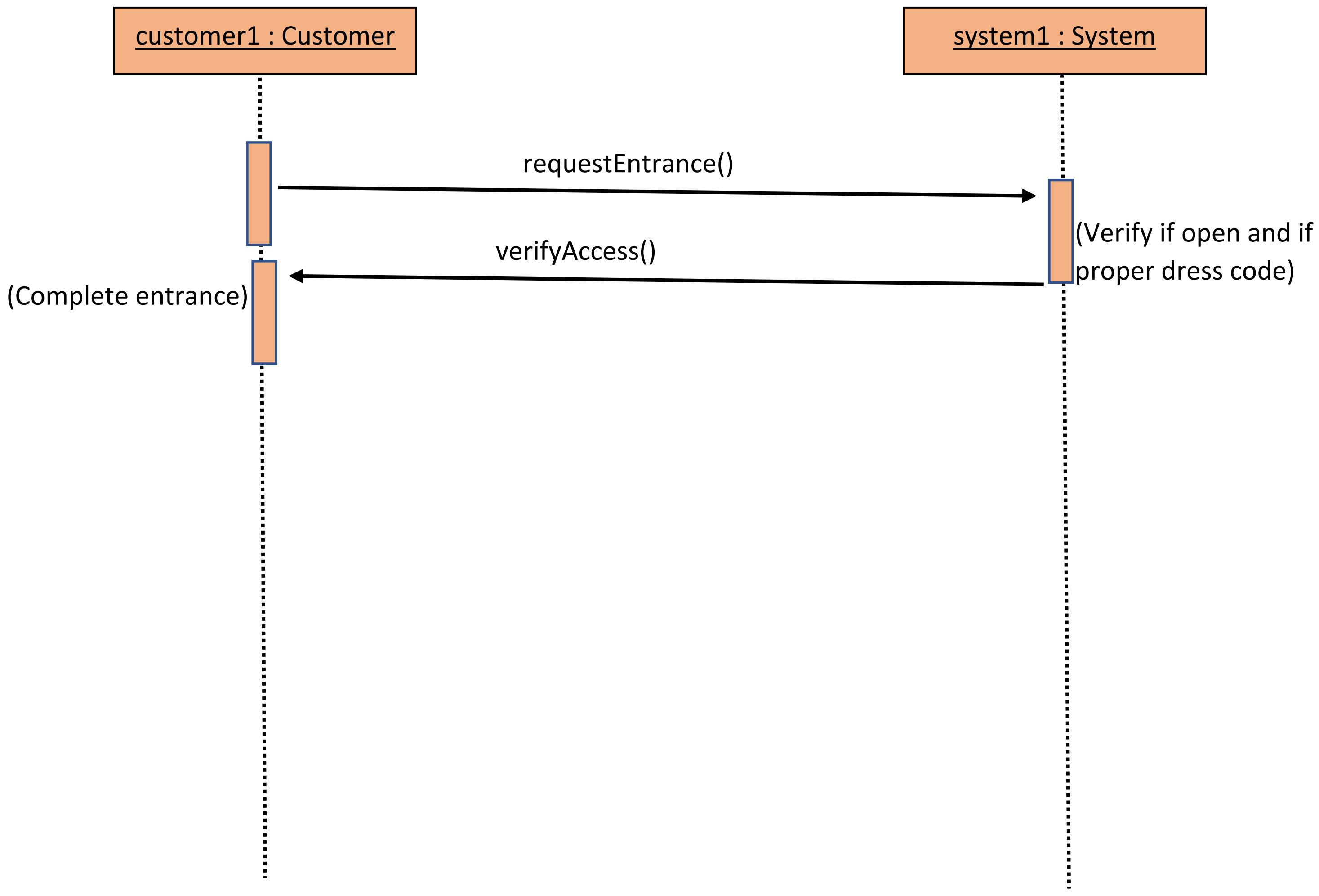
Sequence diagram for Purchase Wholesale



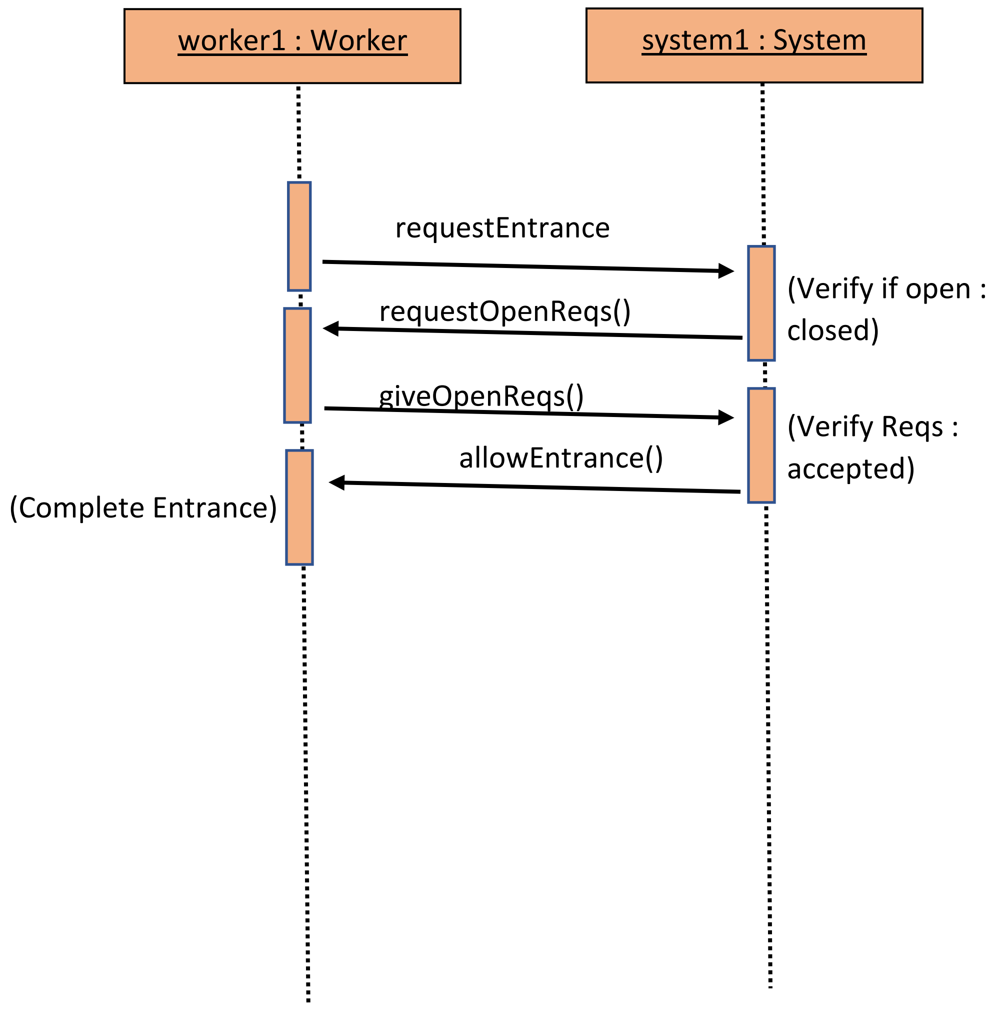
Sequence diagram for Purchase Wholesale with Incorrect Items



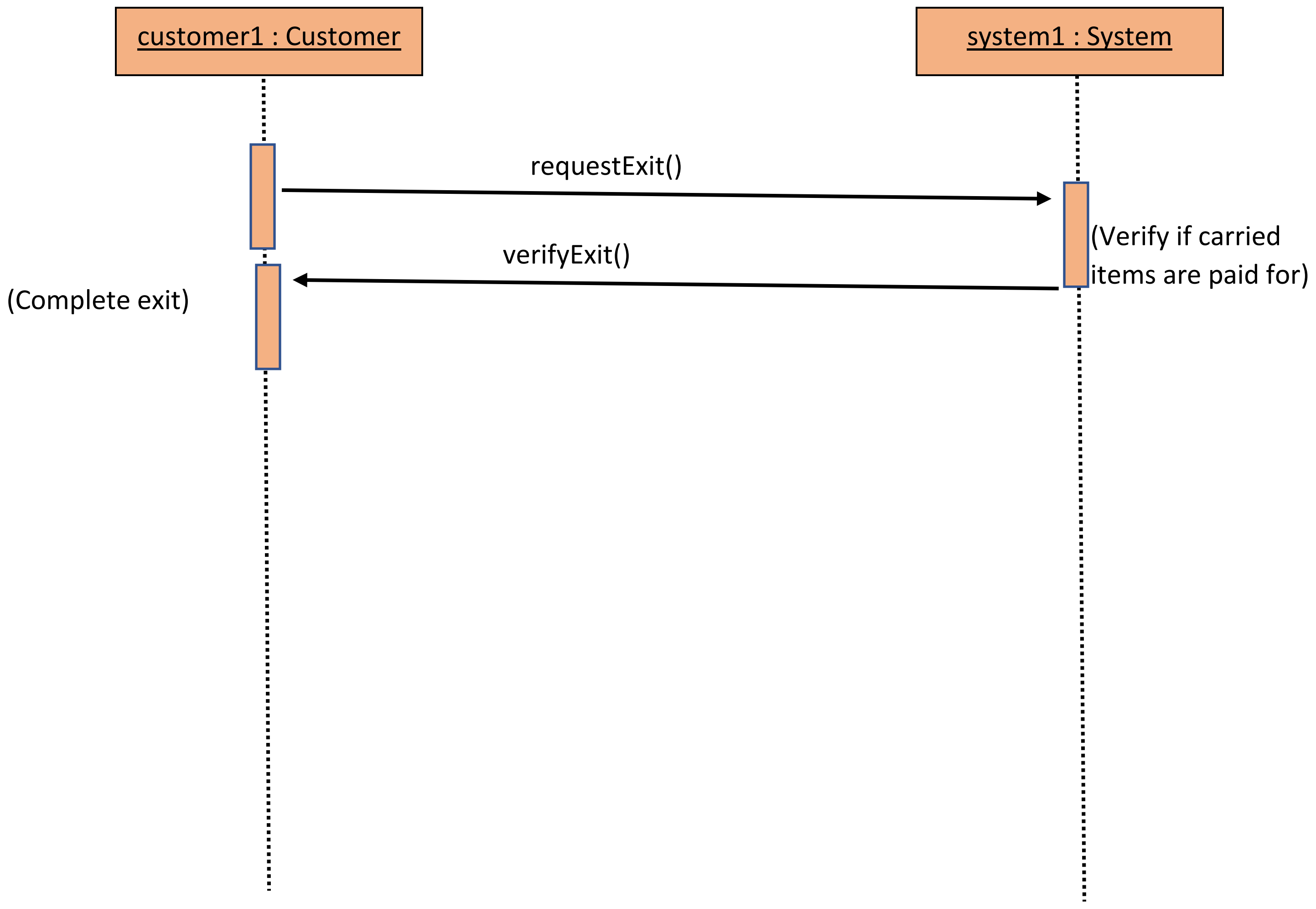
Sequence diagram for customer Enter Building



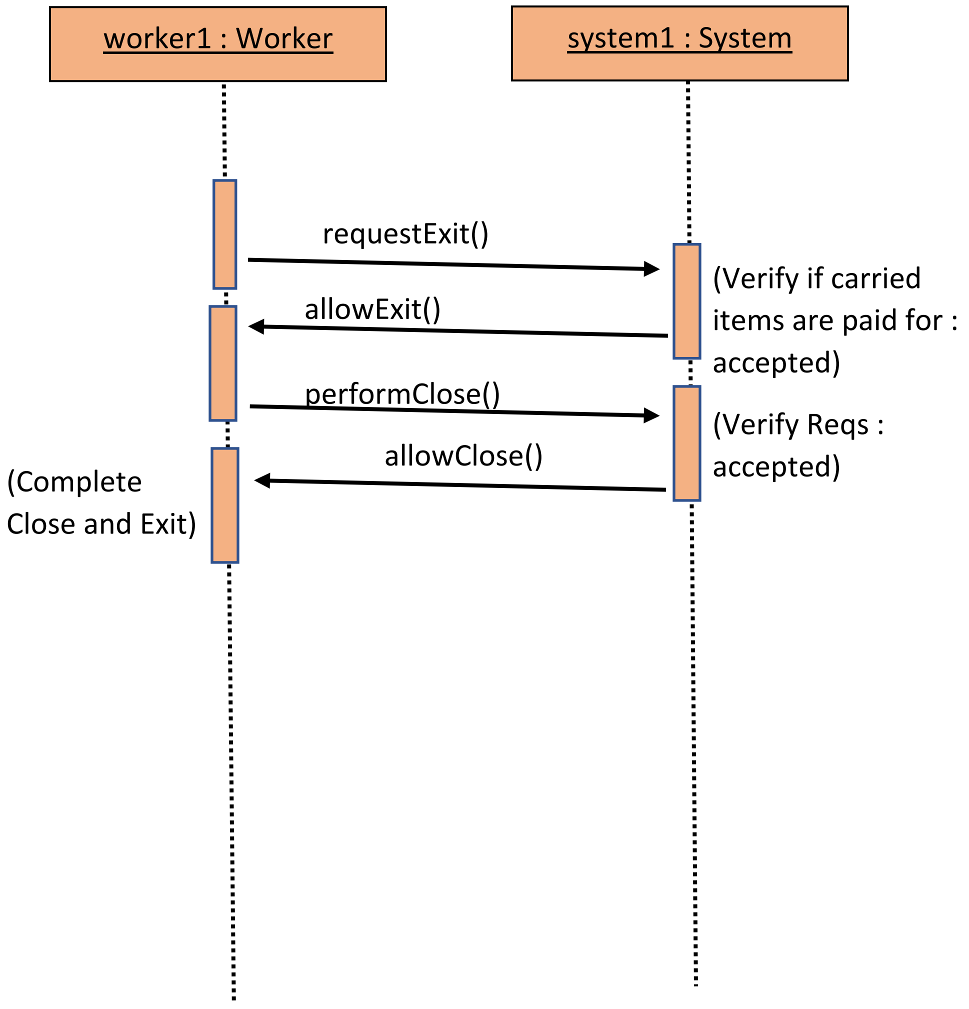
Sequence diagram for worker Enter Building with opening building



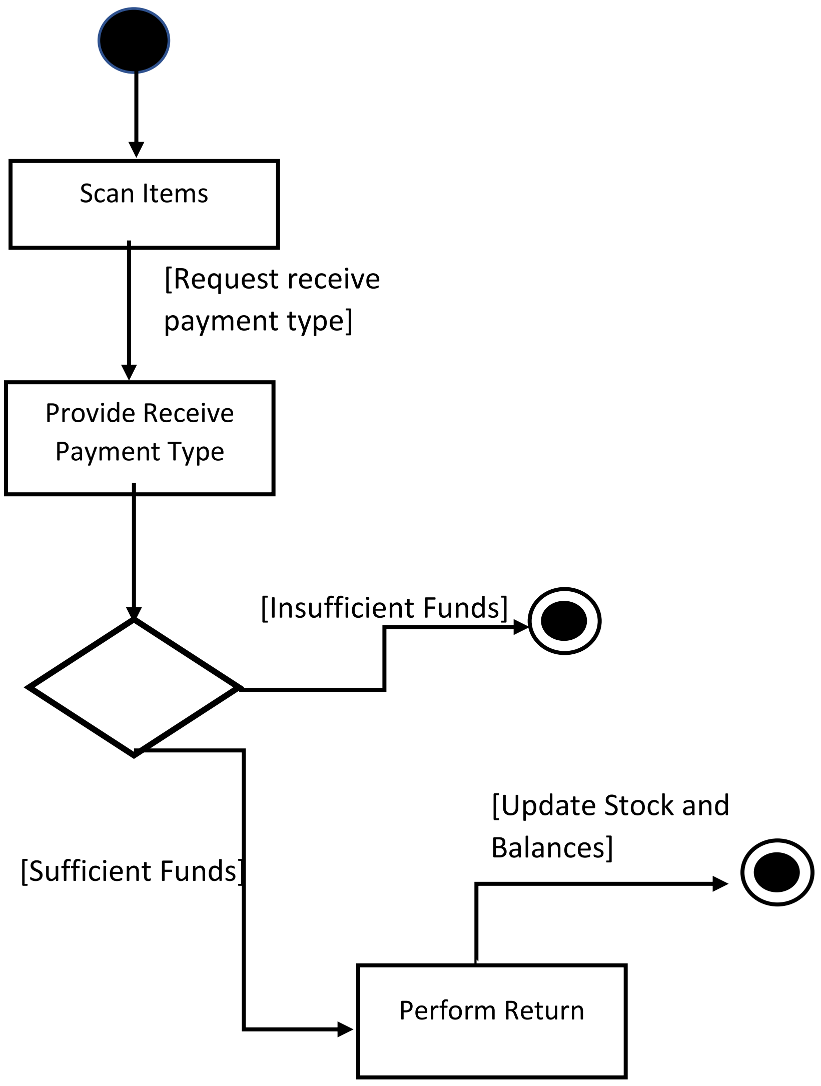
Sequence diagram for customer Leave Building



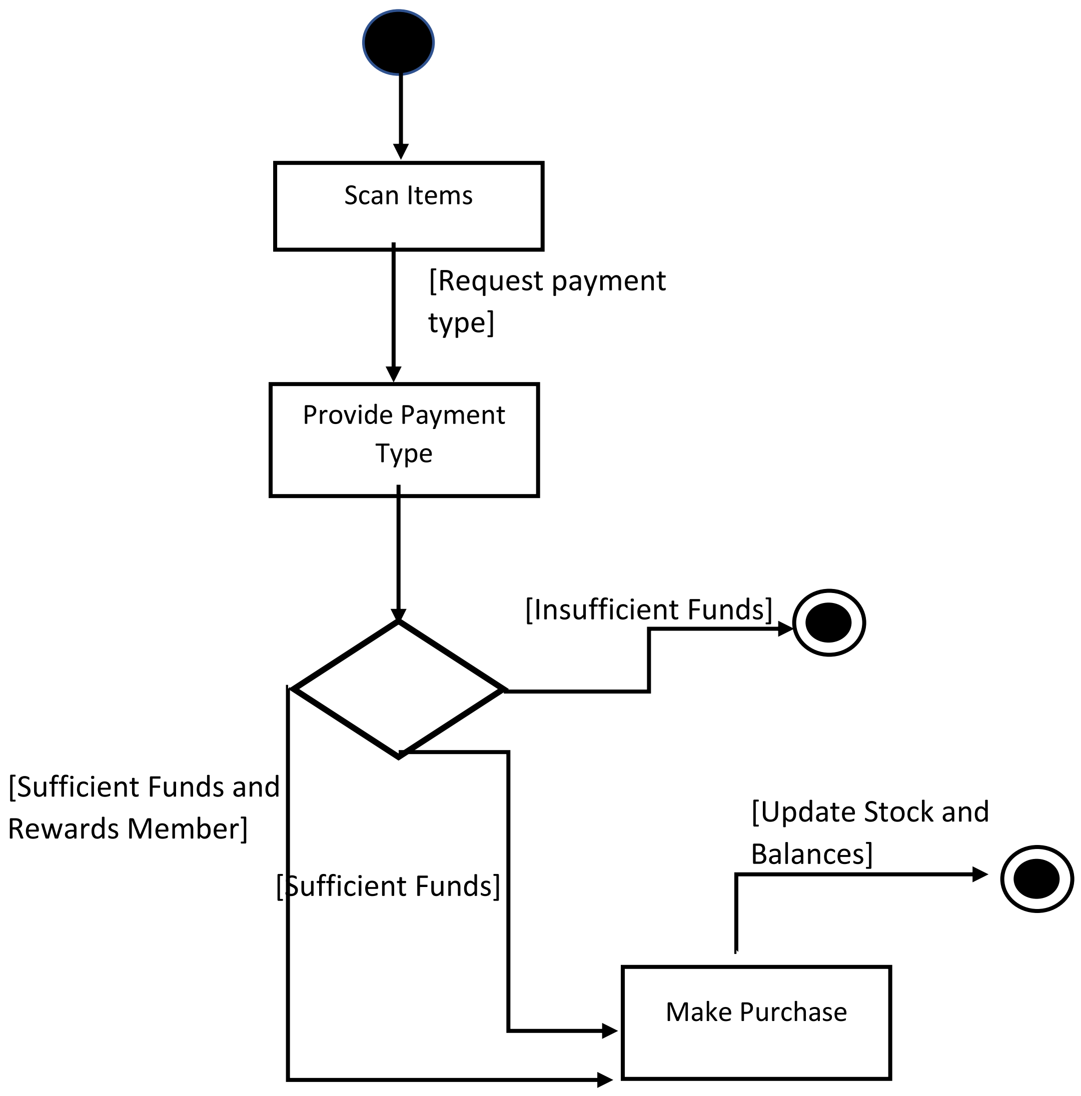
Sequence diagram for worker Leave Building with closing building



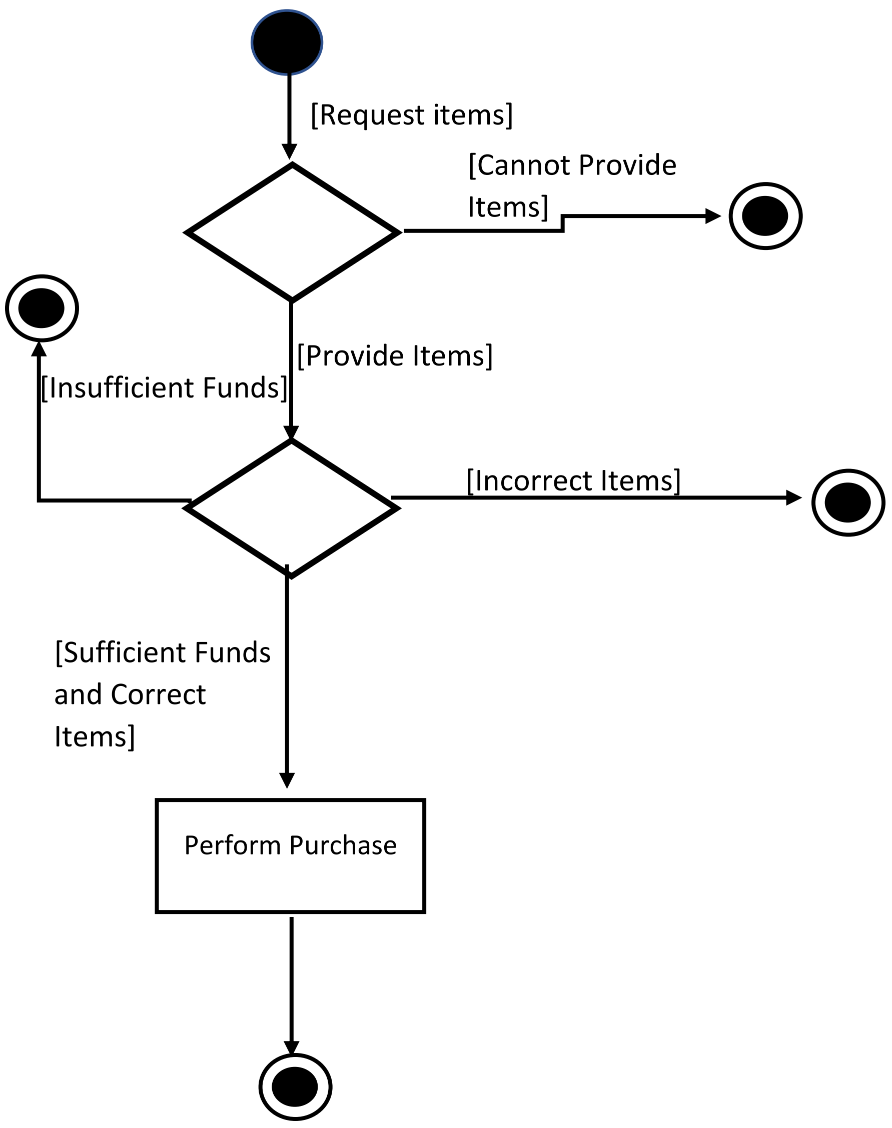
Activity Diagram for Return Order


Scan Items
[Request receive payment type]
Provide Receive Payment Type
[Insufficient Funds]
[Sufficient Funds]
[Update Stock and Balances]
Perform Return

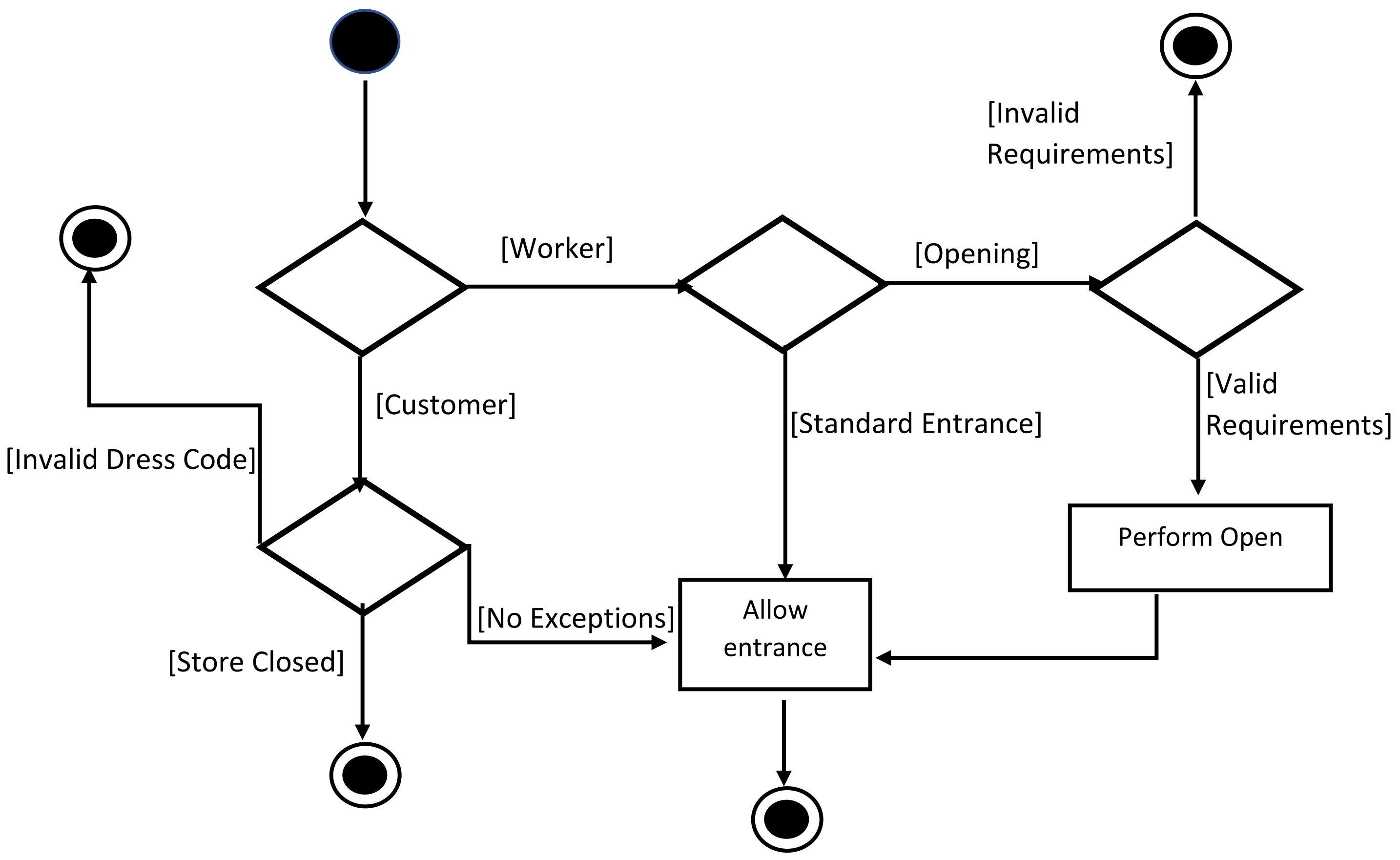
Activity Diagram for Make Purchase



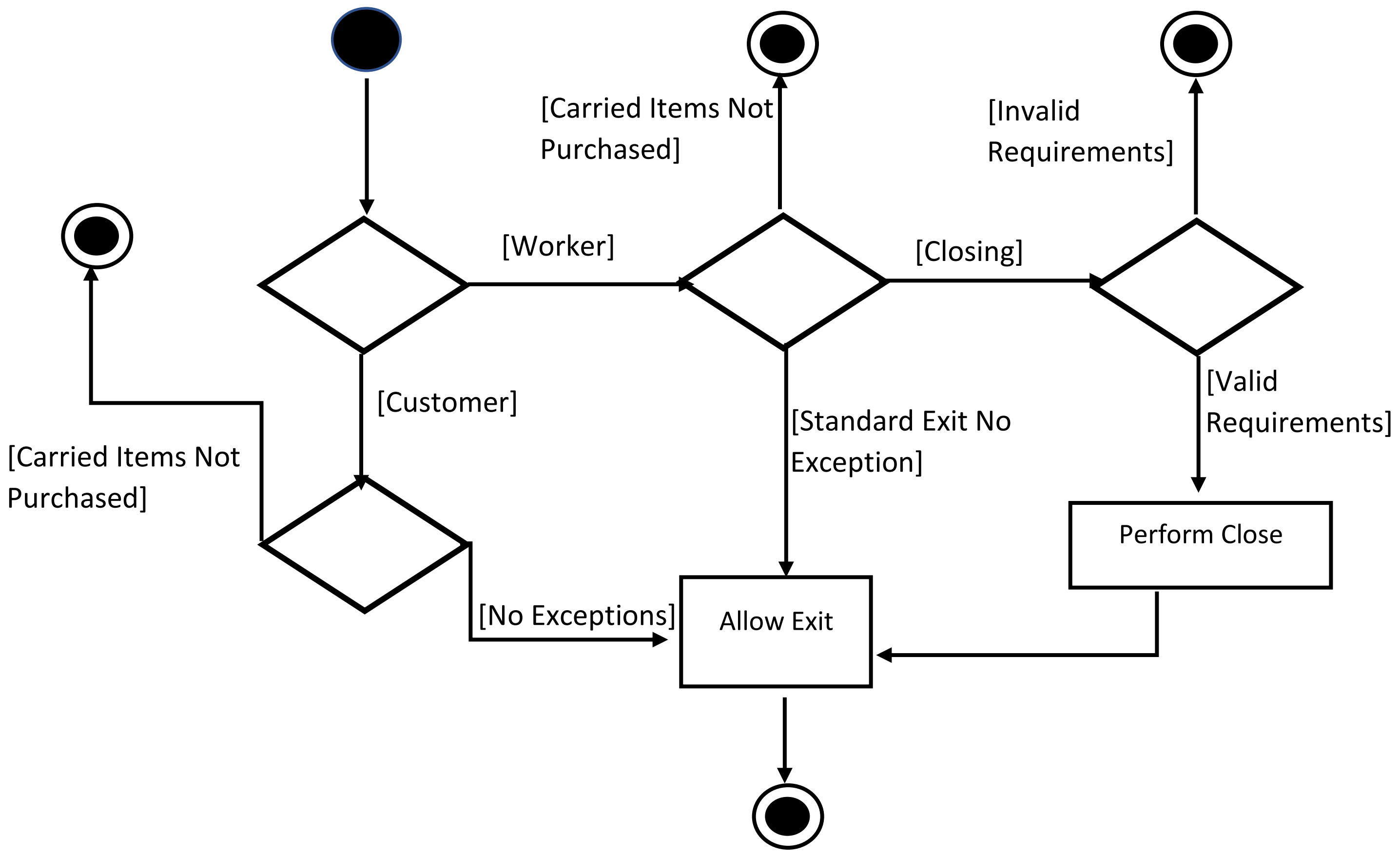
Activity Diagram for Purchase Wholesale



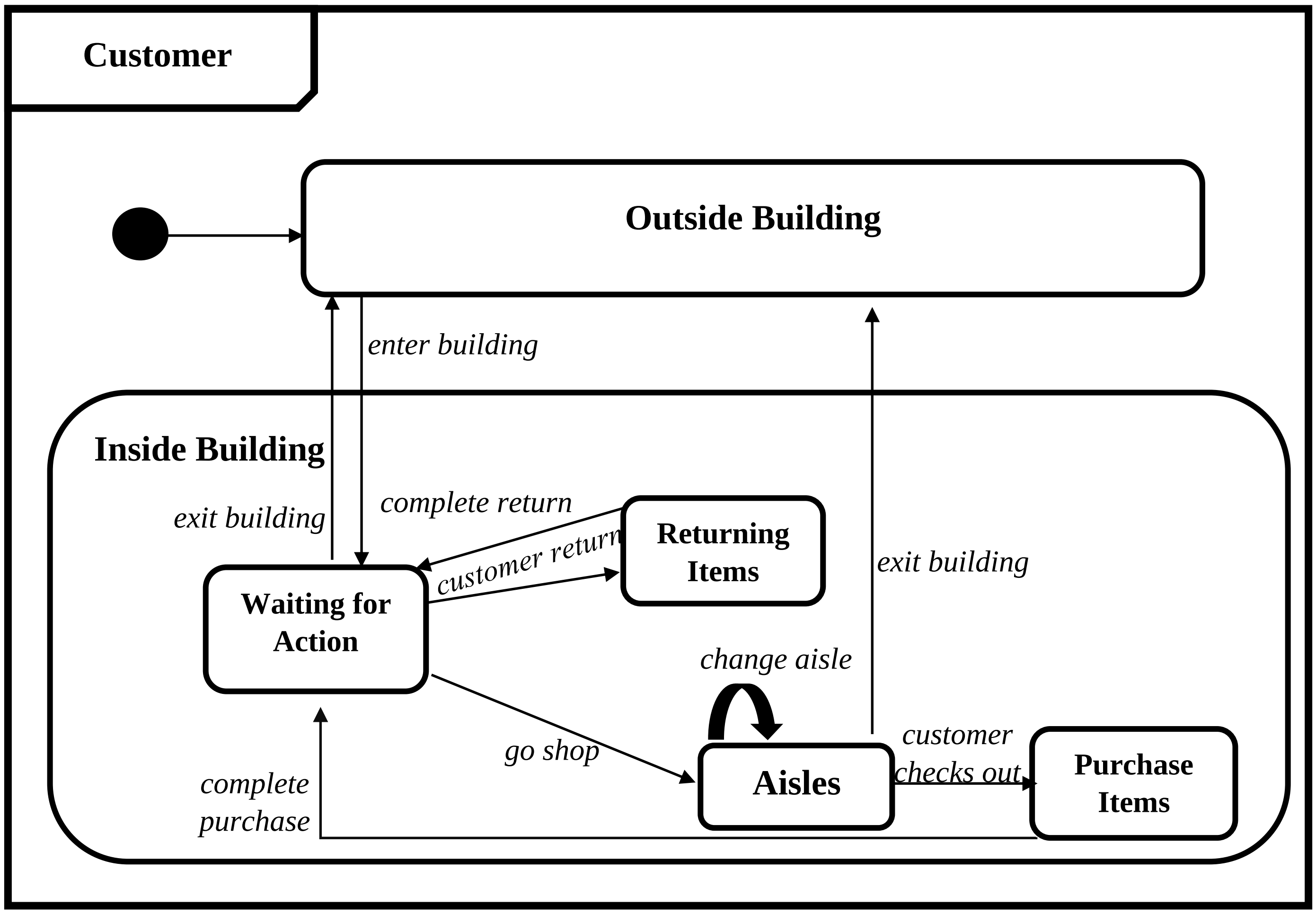
Activity Diagram for Enter Building

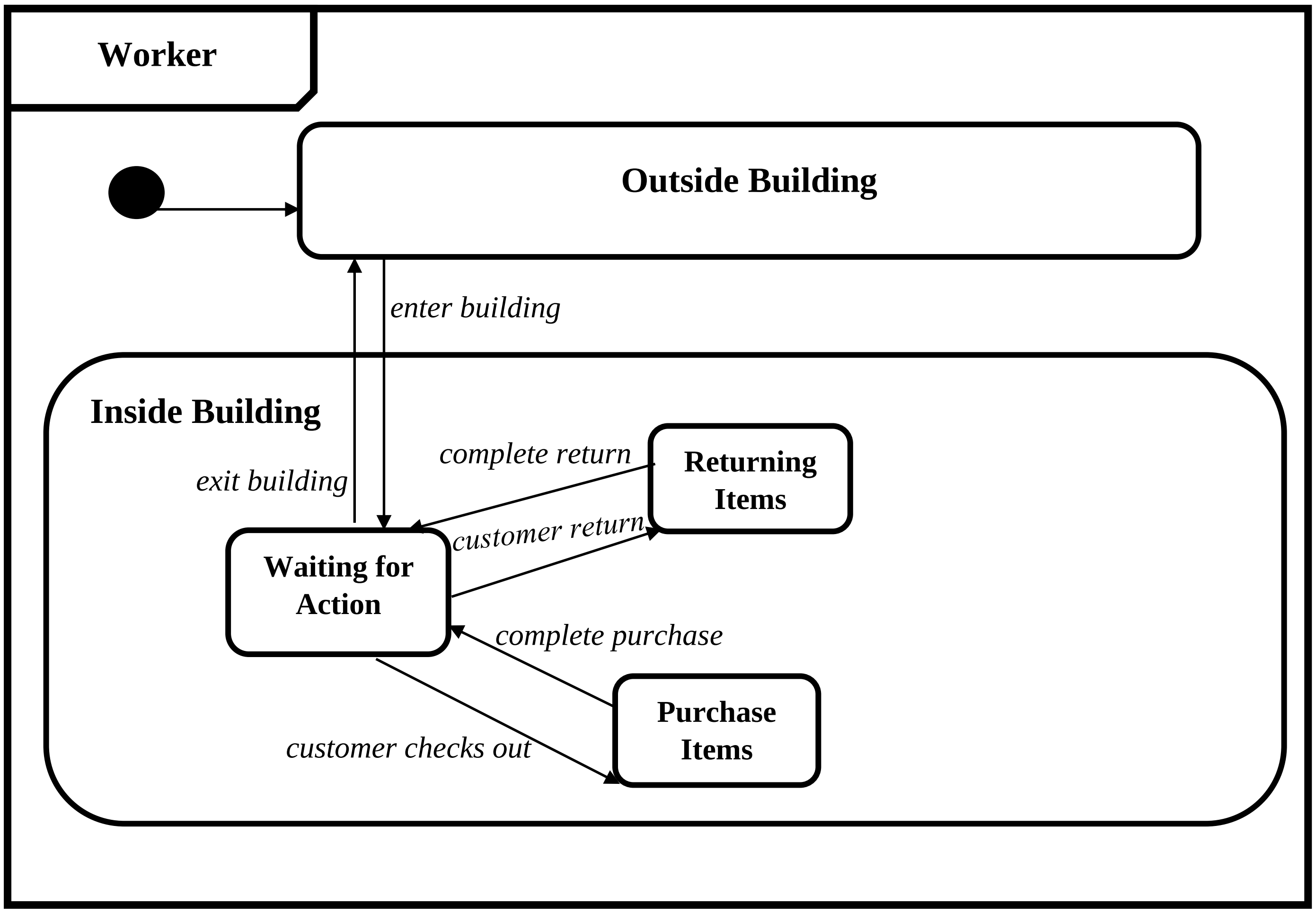


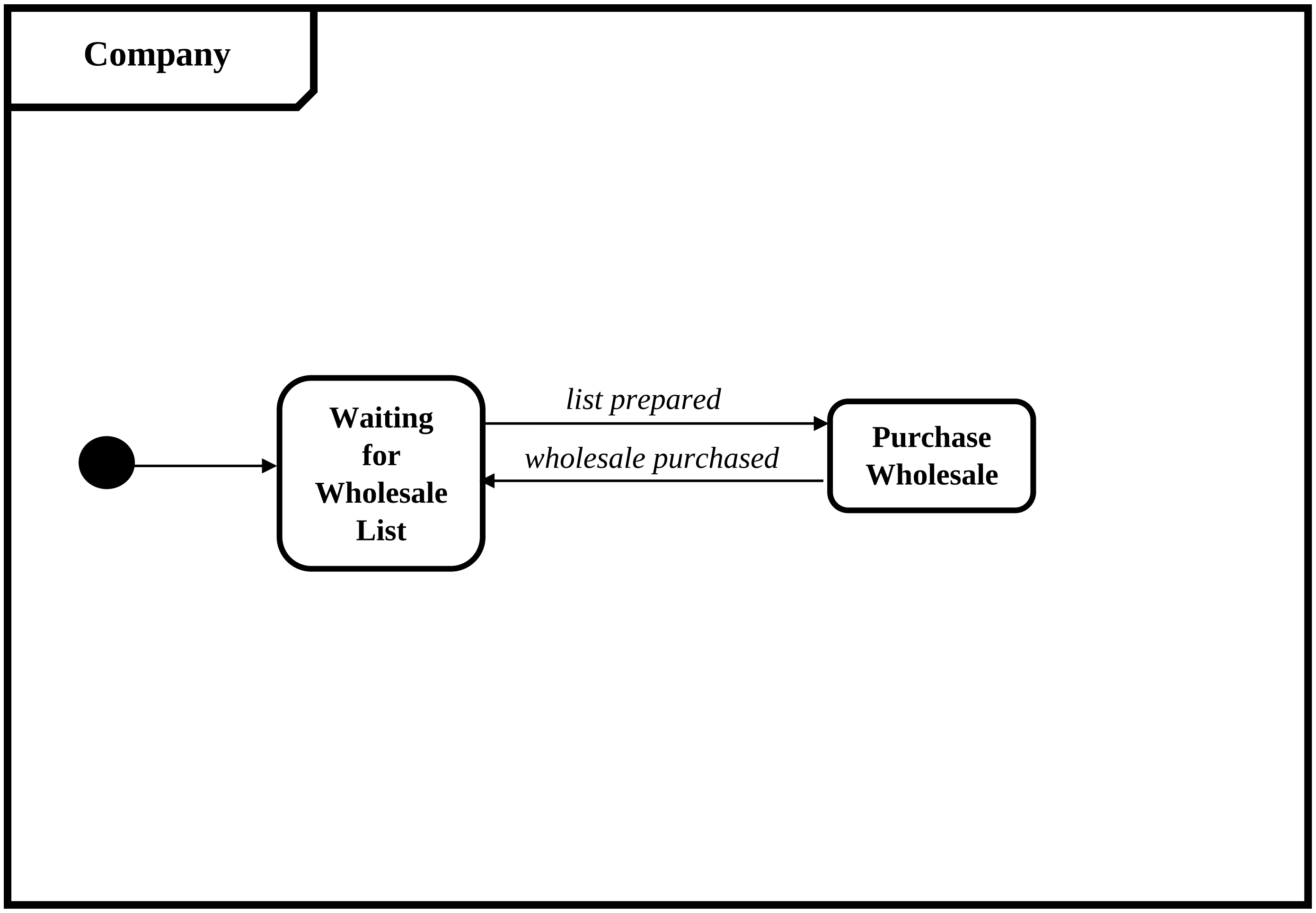
Activity Diagram for Leave Building

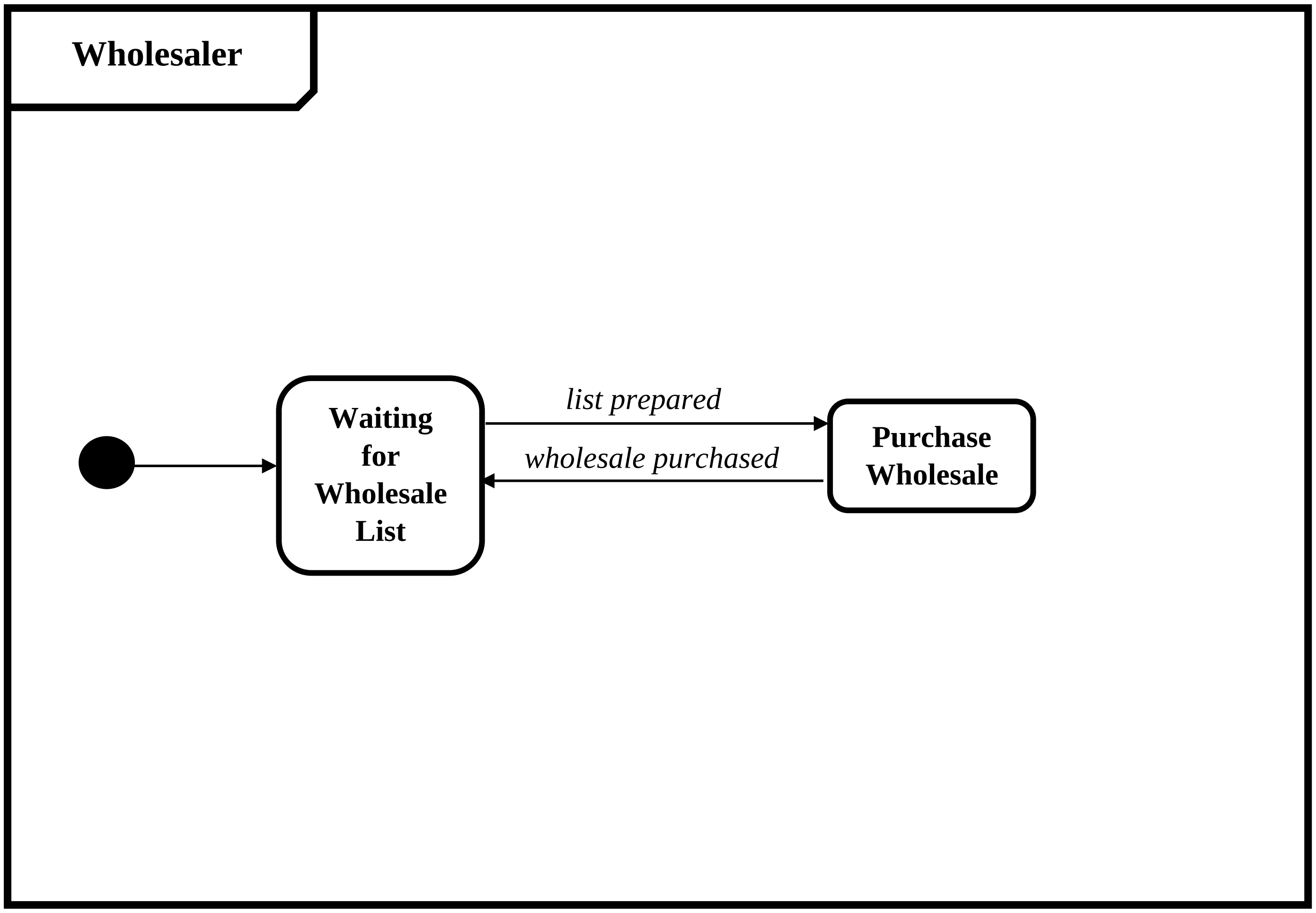


State Diagrams:




Worker
Outside Building
enter building
Inside Building
exit building
complete return
Returning Items
customer return
Waiting for Action
complete purchase
Purchase Items
customer checks out


Company
Waiting for Wholesale List
list prepared
wholesale purchased
Purchase Wholesale


Wholesaler
Waiting for Wholesale List
list prepared
wholesale purchased
Purchase Wholesale

State Descriptions:

State Name: *Outside Building*

State Description: Person is outside of the building and may enter the building.

Event Sequence that produces the state: *removePeopleInBuilding(Person p1)*

Condition that characterizes this state: Person is not found in the ArrayList<Person> *peopleInBuilding*

Events accepted in this state:

Event: *enter building*

Response: *addPeopleInBuilding(Person p1)*

Next State: *inside building and inside waiting for action*

State Name: *Waiting for Action*

State Description: Customer is inside building and is waiting to go to some other state.

Event Sequence that produces the state: *addPeopleInBuilding(Person p1)*

completeReturn()

completePurchase()

Condition that characterizes this state: Customer is found in the ArrayList<Person> *peopleInBuilding*

boolean *shopping* = 0

Events accepted in this state:

Event: *customer return*

Response: *makeReturn(Item I1, Payment P1)*

Next State: *waiting for action*

Event: *customer checks out*

Response: *makePurchase(Item I1, Payment P1)*

Next State: *purchase items*

Event: *go shop*

Response: *enterAisle(int A1)*

Next State: *Aisles*

Event: *leave building*

Response: *removePeopleInBuilding(Person p1)*

Next State: *outside building*

State Name: *purchasing Items*

State Description: Items are being purchased from the company.

Event Sequence that produces the state: *requestPurchase() -> makePurchase(Item I1, Payment P1)*

Condition that characterizes this state: The items the customer is purchasing are in the ArrayList<Item> *stock* of the company. *makePurchase(Item I1, Payment P1)* has not completed without errors. boolean isWorkerAvailable is true.

Events accepted in this state:

Event:	*complete purchase*
Response:	*makePurchase(Item I1, Payment P1)* has completed without errors. If isWorkerAvailable is false, it is made to be true.
Next State:	*waiting for action*

State Name: *returning Items*

State Description: Items are being returned to the company.

Event Sequence that produces the state: *requestReturn() -> makeReturn(Item I1, Payment P1)*

Condition that characterizes this state: The items that the customer is returning are not in the ArrayList<Item> *stock* of the company. *makeReturn(Item I1, Payment P1)* has not completed without errors. boolean isWorkerAvailable is true.

Events accepted in this state:

Event:	*complete return*
Response:	*makeReturn(Item I1, Payment P1)* has completed without errors.
Next State:	*waiting for action*

State Name: *Aisles*

State Description: Customer is in the aisles and able to add items to their cart.

Even Sequence that produces the state: *goShop()*

changeAisle(int A1)

Condition that characterizes this state: int *Aisles* is a non-zero integer

Events accepted in this state:

Event:	*customer checks out*
Response:	*requestPurchase()*
Next State:	*purchase items*

Event:	*outside building*
Response:	*removePeopleInBuilding(Person p1)*
Next State:	*outside building*

Event:	*change aisle*
Response:	*changeAisle(int A1)*
Next State:	*Aisles*

State Name: *Purchase Wholesale*

State Description: The company is buying wholesale from the wholesaler

Even Sequence that produces the state: *purchaseWholesale(ArrayList<Item> list)*

Condition that characterizes this state: The company's ArrayList<Item> *wholesaleList* is not empty. *purchaseWholesale(ArrayList<Item> list)* has not completed without errors.

Events accepted in this state:

Event:	*wholesale purchased*
Response:	*purchaseWholesale(ArrayList<Item> list)* has completed without errors.
Next State:	*waiting for wholesale list*

State Name: *waiting for wholesale list*

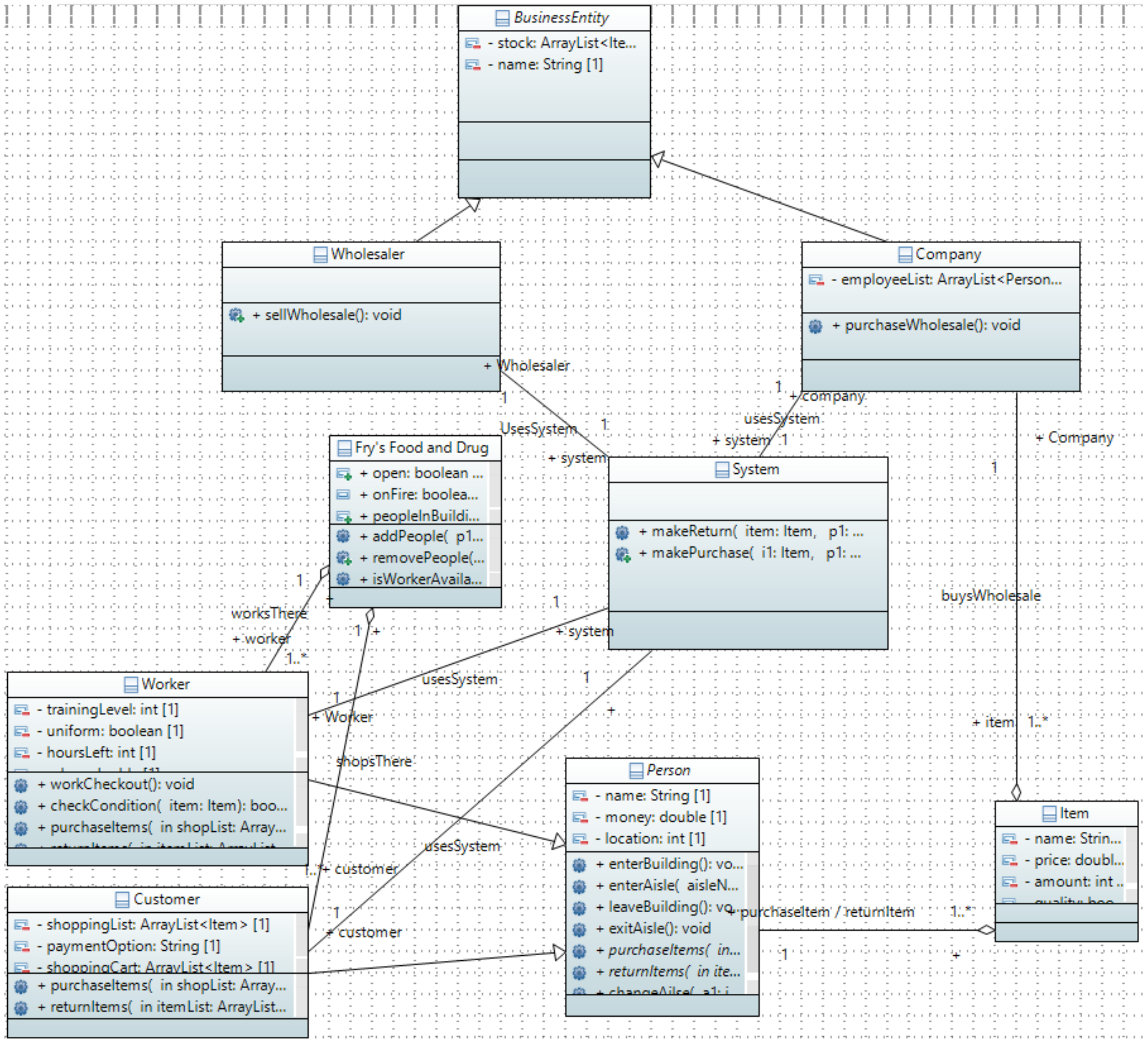
State Description: The company is preparing a wholesale list, and the wholesaler is waiting to provide the wholesale.

Even Sequence that produces the state: *purchaseWholesale(ArrayList<Item> list)* has completed without errors.

Condition that characterizes this state: The company's ArrayList<Item> *wholesaleList* is not finalized.

Events accepted in this state:

Event:	*list prepared*
Response:	*purchaseWholesale(ArrayList<Item> list)* is ran.
Next State:	*purchase wholesale*


BusinessEntity
- stock: ArrayList<Ite...
- name: String [1]
Wholesaler
+ sellWholesale(): void
Company
- employeeList: ArrayList<Person...
+ purchaseWholesale(): void
+ Wholesaler
1
UsesSystem
1
+ system
1
+ company
usesSystem
+ system 1
+ Company
1
Fry's Food and Drug
+ open: boolean ...
+ onFire: boolea...
+ peopleInBuildi...
+ addPeople(p1...
+ removePeople(...
+ isWorkerAvaila...
System
+ makeReturn(item: Item, p1: ...
+ makePurchase(i1: Item, p1: ...
buysWholesale
1
worksThere
+ worker
1..*
1
+
1
+ system
usesSystem
1
+ Worker
+ item 1..*
Worker
- trainingLevel: int [1]
- uniform: boolean [1]
- hoursLeft: int [1]
+ workCheckout(): void
+ checkCondition(item: Item): boo...
+ purchaseItems(in shopList: Array...
shopsThere
Person
- name: String [1]
- money: double [1]
- location: int [1]
+ enterBuilding(): vo...
+ enterAisle(aisleN...
+ leaveBuilding(): vo...
+ exitAisle(): void
+ purchaseItems(in...
+ returnItems(in ite...
Item
- name: Strin...
- price: doubl...
- amount: int ...
usesSystem
1..* + customer
Customer
- shoppingList: ArrayList<Item> [1]
- paymentOption: String [1]
+ purchaseItems(in shopList: Array...
+ returnItems(in itemList: ArrayList...
1
+ customer
+ purchaseItem / returnItem
1..*
1
+

Notes on how to Set Up and Run the System

There are four different possible users of the system. These are the customer, the worker, the company, and the wholesaler.

For a customer, there are six different states. These are “Outside the Building”, “Inside the Building”, “Aisles”, “Returning items”, “Waiting for Action”, and “Purchasing Items”. The customer begins in the “Outside the Building” state. When the customer is in the “Outside the Building” state, they may enter the building by calling the *enterBuilding(Frys_Food_and_Drug f)* function. This puts the customer into the “Inside the Building” state as well as the “Waiting for Action” state. From the “Waiting for Action” state, the customer can go to the “Outside the Building”, “Returning Items”, or “Aisles” states by calling the *leaveBuilding(Frys_Food_and_Drug f), returnItems(Frys_Food_and_Drug f),* or *enterAisle()* functions. At the “Returning Items” state, the customer will enter the “Waiting for Actions” state when the *returnItems(Customer c1)* function has been executed without error. At the “Purchasing Items” state, the customer will enter the “Waiting for Actions” state when the *purchaseItems(Customer c1)* function has been executed without error. At the “Aisles” state, the customer may change aisles by calling the *changeAisle(int a1)* function, the customer may checkout by calling the *goCheckOut()* function and then purchase their items by calling the *purchaseItems(Frys_Food_and_Drug f)* function, the customer may add items to their shopping cart by calling the *addItem(int aisle, Item i, int num)* function, the customer may see their shopping list or their shopping cart by calling the *PrintList(ArrayList<Item> list)* function, or the customer may leave the building by calling the *leaveBuilding(Frys_Food_and_Drug f)* function.

For a worker, there are five different states. These are “Outside the Building”, “Inside the Building”, “Aisles”, “Returning items”, “Waiting for Action”, and “Purchasing Items”. The worker begins in the “Outside the Building” state. When the worker is in the “Outside the Building” state, they may enter the building by calling the *enterBuilding(Frys_Food_and_Drug f)* function. This puts the worker into the “Inside the Building” state as well as the “Waiting for Action” state. From the “Waiting for Action” state, the worker can go to the “Outside the Building” by calling the *leaveBuilding(Frys_Food_and_Drug f)* function. The worker may enter “Returning Items” or “Purchasing Items” states as well. This happens when a customer calls the *returnItems(Frys_Food_and_Drug f)* or *purchaseItems(Frys_Food_and_Drug f)* functions. At the “Returning Items” state, the worker may make the return by calling the *returnItems(Customer c1)* function, and when that function executes without error, the worker goes to the “Waiting for Action” state. At the “Purchasing Items” state, the worker may make the purchase by calling the *purchaseItems(Frys_Food_and_Drug f)* function, and when that function executes without error, the worker goes to the “Waiting for Action” state.

For a company, there are two different states. These are “Waiting for Wholesale List” and “Purchase Wholesale”. The company starts in the “Waiting for Wholesale List” where they can add an item to the wholesale list by calling the *addToWholesaleList(Item i)* function, or they could enter the “Purchase Wholesale” state by calling the *purchaseWholesaleList(Wholesaler w)* or *purchaseWholesale(Item i, Wholesaler w)* function. At the “Purchase Wholesale” state, the company will enter the “Waiting for Wholesale List” state when the *purchaseWholesaleList(Wholesaler w)* or *purchaseWholesale(Item i, Wholesaler w)* function has completed without errors.

For a wholesaler, there are two different states. These are “Waiting for Wholesale List” and “Purchase Wholesale”. The wholesaler starts in the “Waiting for Wholesale List” where they enter the “Purchase Wholesale” state when a company calls the *purchaseWholesaleList(Wholesaler w)* or *purchaseWholesale(Item i, Wholesaler w)* function. At the “Purchase Wholesale” state, the wholesaler can call the *sellWholesale(Item i)* function in order to sell wholesale, and they will enter the “Waiting for Wholesale List” state when the *sellWholesale(Item i)* function has completed without errors.

Note: It may be possible in the current implementation to call a function out of order by using the function call, but with GUI implementation, that will not be an option.

Use Case Testing

The Driver.java file under the ProjectSoftware.pkg package tests all of the use cases and follows the state diagrams to test that each state and the transitions between states are working properly. Since this driver file runs without errors, it shows that the accessibility of our system is acceptable.

Unit Testing

In the UnitTests source folder under the ProjectSoftware.pkg.test package, there are four JUnit test files for each of the four users of the system (Customer, Worker, Company, Wholesaler). In these unit tests, every individual function that is necessary for the system is used to complete the tests. Since the tests all passed without error, this shows that the functionality of the system is acceptable.

Revisions

The main change that was added from task three to task four is the addition of a GUI that is capable of running the system. This GUI allows the person running the system to go through the use cases of each user, and some methods were updated to make the experience more streamlined and enjoyable. The code was also better organized in order for someone to better understand what is meant to happen.

Challenges

Many of the challenges came with the implementation of a GUI. Our GUI is very dynamic and gives the user many more options than any other type of project that we have worked on in class. It also implements images that make the interface more appealing and figuring out how to insert the images as well as the proper layouts to use for the dynamic button placements was a challenge. This is especially the case for the implementation of random aisle generation every time a customer picks an item. Each aisle had to be made separately and tested separately, and to do that process for four separate aisles was a large task that requires many lines of code.

Our system also uses a clever network of classes in order to implement the system as it would be implemented in real life. One example is how for a customer to checkout it requires methods from the customer, worker, and the system class. Figuring out how to properly connect these methods so that the system works smoothly was a challenge, but it ultimately paid off.

Also, since we added four different ways of using the system, it was essentially as if we created four different systems. The customer user is the most intensive user, but it was a lot of work to not only make sure the customer works, but to implement three other functionalities of the system.

Altogether, there were times when our group was worried that we chose a scope that is too large, but we are glad that we were capable of completing the system and having it run as it does currently.