Home Appliance / AGD Store

s26816 David Condratiuc
s31766 Maksymilian Kozub
s27559 Kacper Tkacz
s30165 Mert Altinbas

Table of Contents

User Requirements

Home Appliances Store System supports the daily operations of a retail business selling both new and used products. It integrates all essential business processes such as managing product information, handling customer orders, monitoring employee work and leave records and maintaining store operations including opening hours and deliveries.

Customer can browse and purchase or sell products, create orders and pay or get paid though multiple methods.

Products in the system contain detailed data such as name, description, model number, brand, warranty period, related products, weight and available stock for both new and used items and products currently in repair. Each product can have multiple custom properties such as color or material limited by the category it is in, which may be part of a hierarchical structure of parent and subcategories. A product can be a freestanding product that have a movement cost and/or an integrated product that require assembly and have an integration cost. Some integrated products must be installed by the store.

Customers can place buy/sell orders containing one or more products. The system stores the order date and automatically calculates the total cost based on product prices and quantities and for sell orders amount paid can be reduced based on its state. Each order is linked to a specific customer. Products included in an buy order must be available in stock before the order can be confirmed. Orders can include a delivery which specifies the sending and receiving dates, delivery company, cost, tracking number and whether the order has been received.

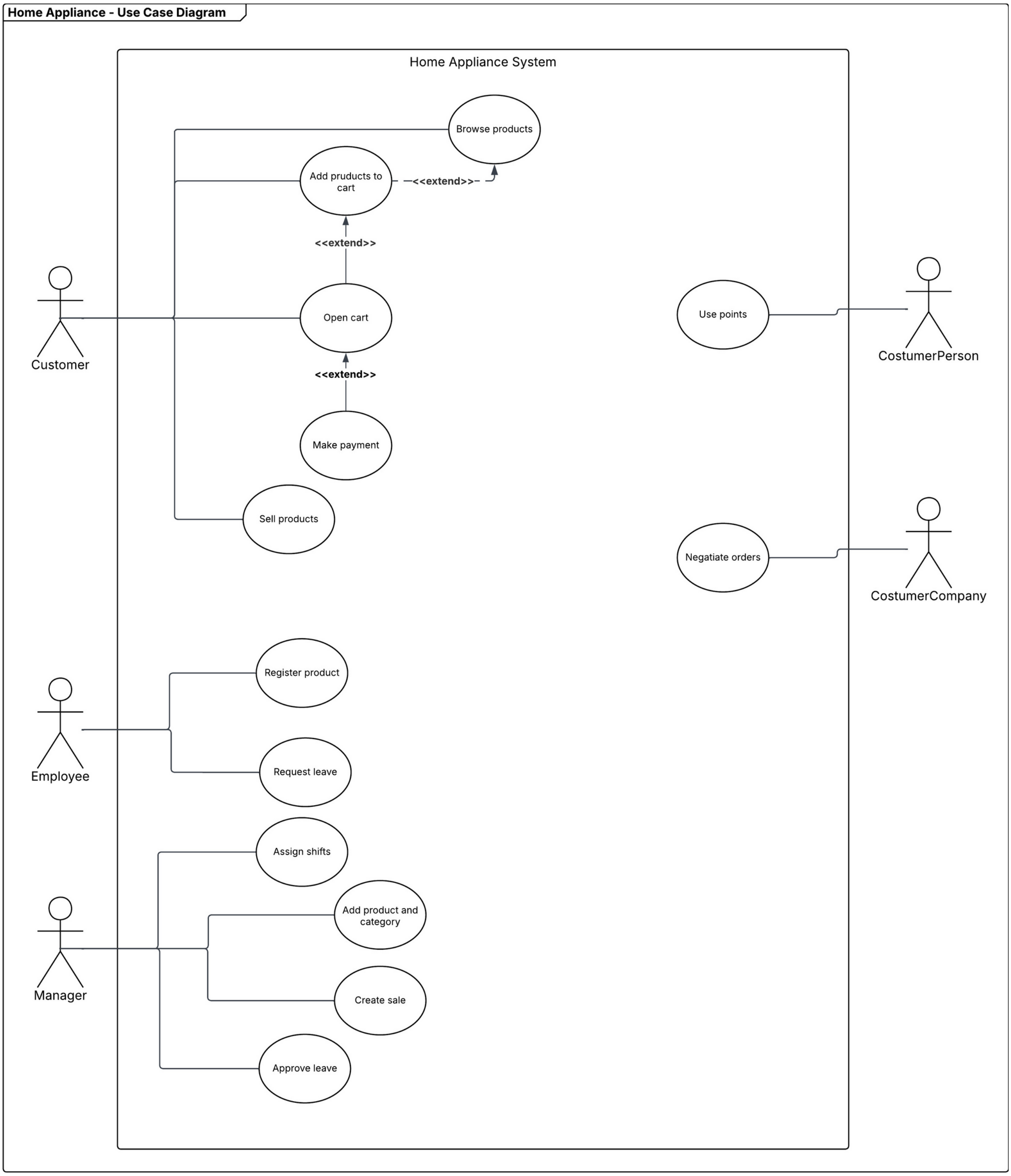
The payment module supports multiple payment methods: card, BLIK, PayPal or cash. Each payment type includes specific information which must be included for it, such as card number and CVV for Card payments, a transaction ID for Blik and a paypal account ID for PayPal.

Customer data includes name, middle name, surname, date of birth and address. Customers accumulate loyalty points that can be used in purchases. The customer's age is automatically calculated based on the date of birth.

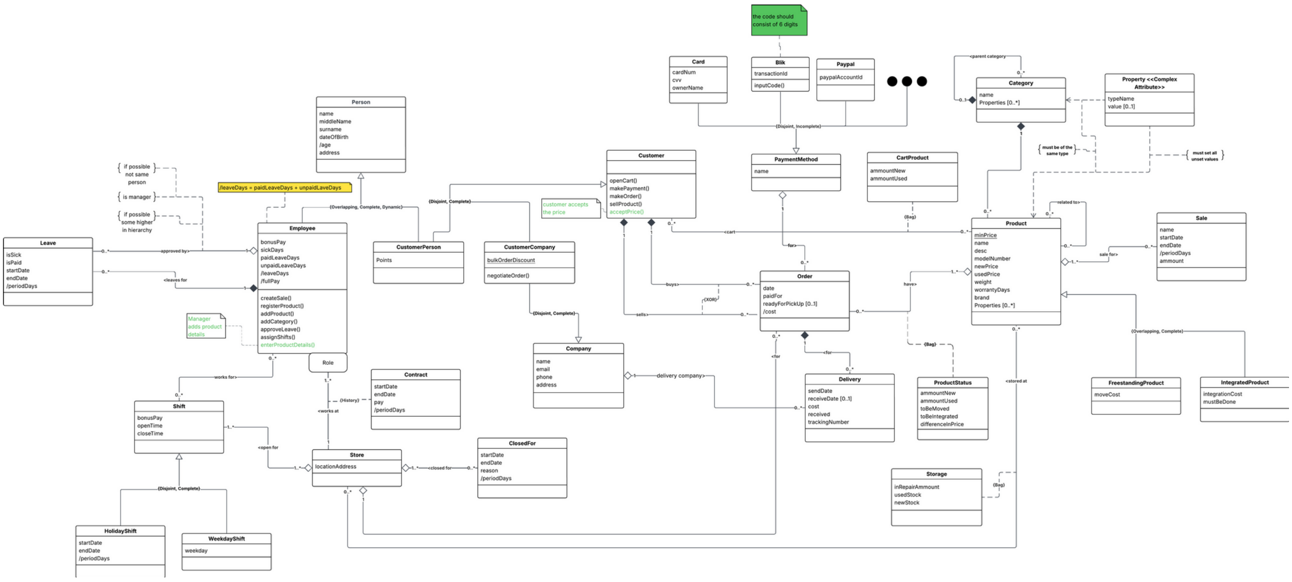
Employees are connected to roles through employment contracts which define start and end dates and store the contract duration as a derived value. Each employee can have multiple leave entries categorized as sick, paid, unpaid. They are also given shifts for specific days of the week or holidays, they may be given a bonus for working during a specific shift.

Store record includes its name, E-mail, phone number and address. System also records working hours of the store. Store can also be marked as closed for specific periods such as renovations or holidays.

Use Case Diagram



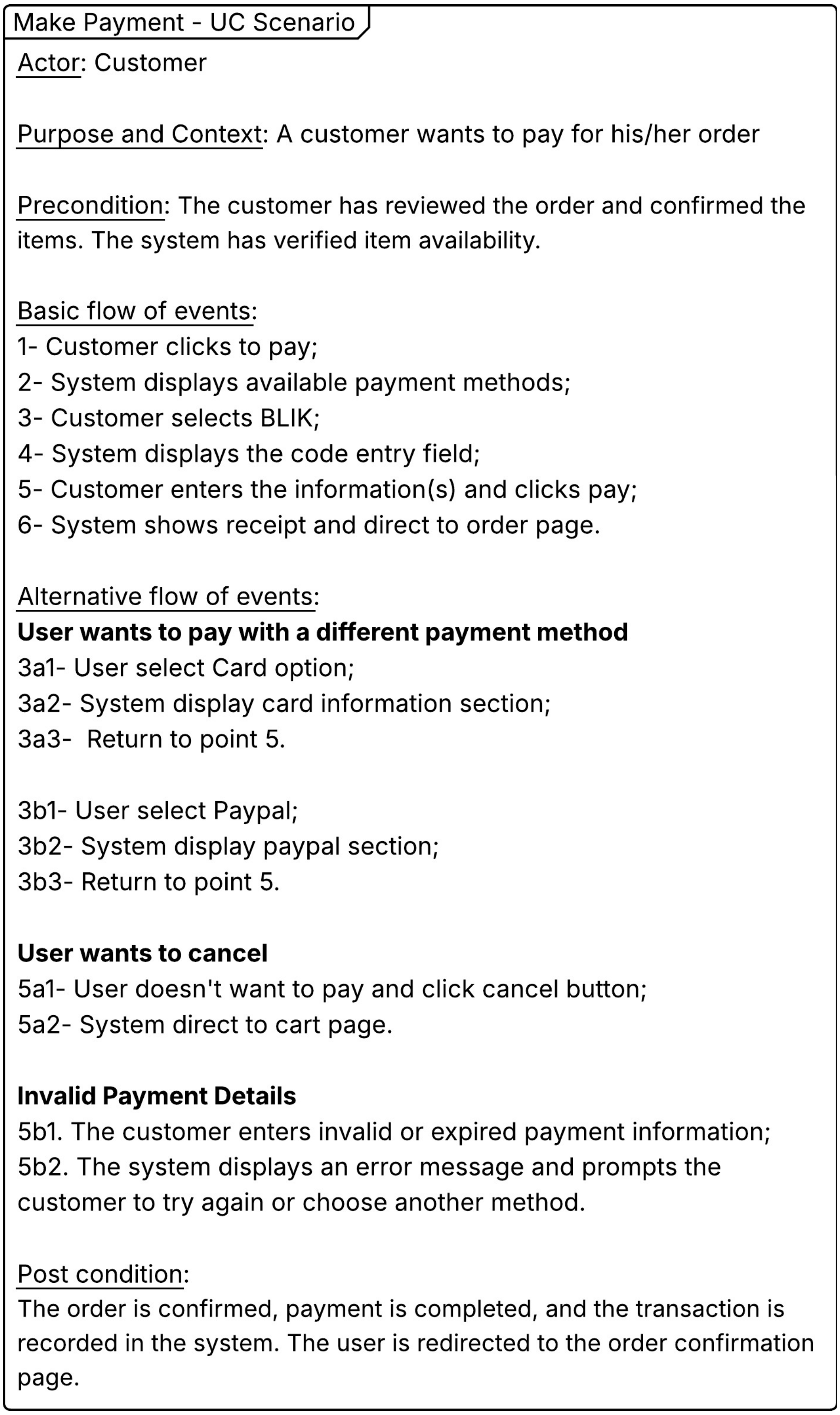
Class Diagram



Class Diagram – Description

The class diagram is of a Home Appliances Store System which includes customers, employees, products, orders, deliveries, and payments. It defines how employees manage stores, contracts, shifts, and leaves, and how customers place orders and earn loyalty points. Products are categorized, can be new or used, and include freestanding and built-in products with their extended attributes. Different payment methods (Card, Blik, PayPal) and deliveries are accepted. Some details, like the Role class and ProductStatus function, are not clearly defined in the diagram.

Use Case Scenario: “Make Payment”



Make Payment - UC Scenario

Actor: Customer

Purpose and Context: A customer wants to pay for his/her order

Precondition: The customer has reviewed the order and confirmed the items. The system has verified item availability.

Basic flow of events:
1- Customer clicks to pay;
2- System displays available payment methods;
3- Customer selects BLIK;
4- System displays the code entry field;
5- Customer enters the information(s) and clicks pay;
6- System shows receipt and direct to order page.

Alternative flow of events:
User wants to pay with a different payment method
3a1- User select Card option;
3a2- System display card information section;
3a3- Return to point 5.

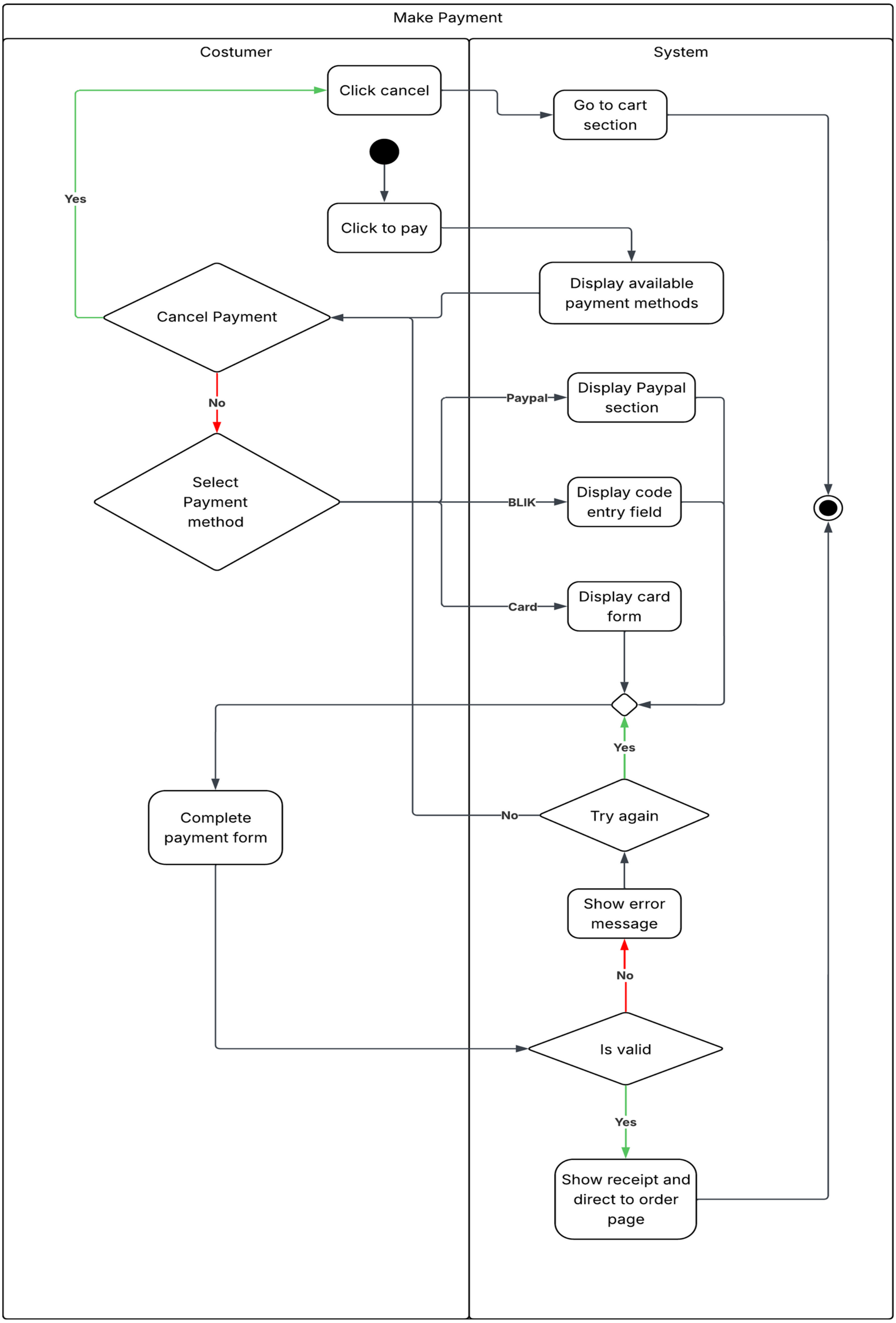
3b1- User select Paypal;
3b2- System display paypal section;
3b3- Return to point 5.

User wants to cancel
5a1- User doesn't want to pay and click cancel button;
5a2- System direct to cart page.

Invalid Payment Details
5b1. The customer enters invalid or expired payment information;
5b2. The system displays an error message and prompts the customer to try again or choose another method.

Post condition:
The order is confirmed, payment is completed, and the transaction is recorded in the system. The user is redirected to the order confirmation page.

Activity Diagram: “Make Payment”



Activity diagram "Make Payment" – Description

This activity diagram represents the process of making a payment by a customer. The customer begins by clicking "**Click to pay**" after which the system displays available payment methods. The customer may choose to cancel the payment or proceed by selecting a payment method such as PayPal, BLIK or card. Depending on the chosen method the system displays the corresponding payment form. The customer then completes the payment form and the system verifies its validity. If the information is invalid an error message is shown and the customer can try again. Once the payment is validated the system shows the receipt and redirects the customer to the order page completing the transaction.

Use Case Scenario: “Request Leave”

Request Leave - UC Scenario

Actor: Employee

Purpose and Context: A employee requests leave

Assumpition: Employee still works at store

Precondition: Employee is logged in

Basic flow of events:
1- Employee clicks to request leave
2- System displays leave form;
3- Employee fills it;
4- Employee clicks to next;
5- System checks if all required fields are filled;
6- System checks if employee has sufficient leave days;
7- Employee notified by system.

Alternative flow of events:

Form is incorect

5a1- Return to point 3;

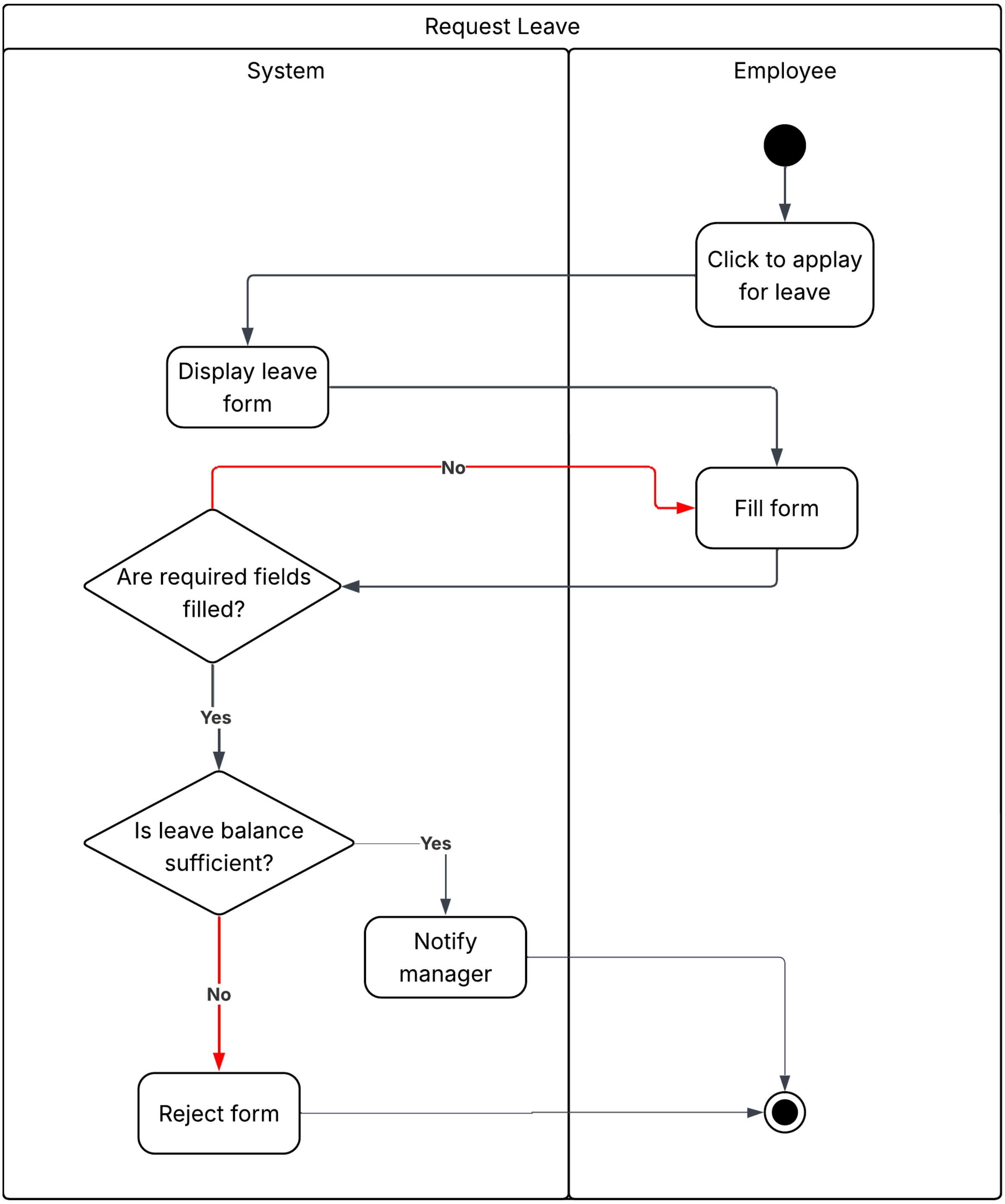
Employee does not have sufficient leave days

6a1- Form is rejected;

Post condition:

Basic: The leave is recorded and the timetable updated
Alternative: No changes occur

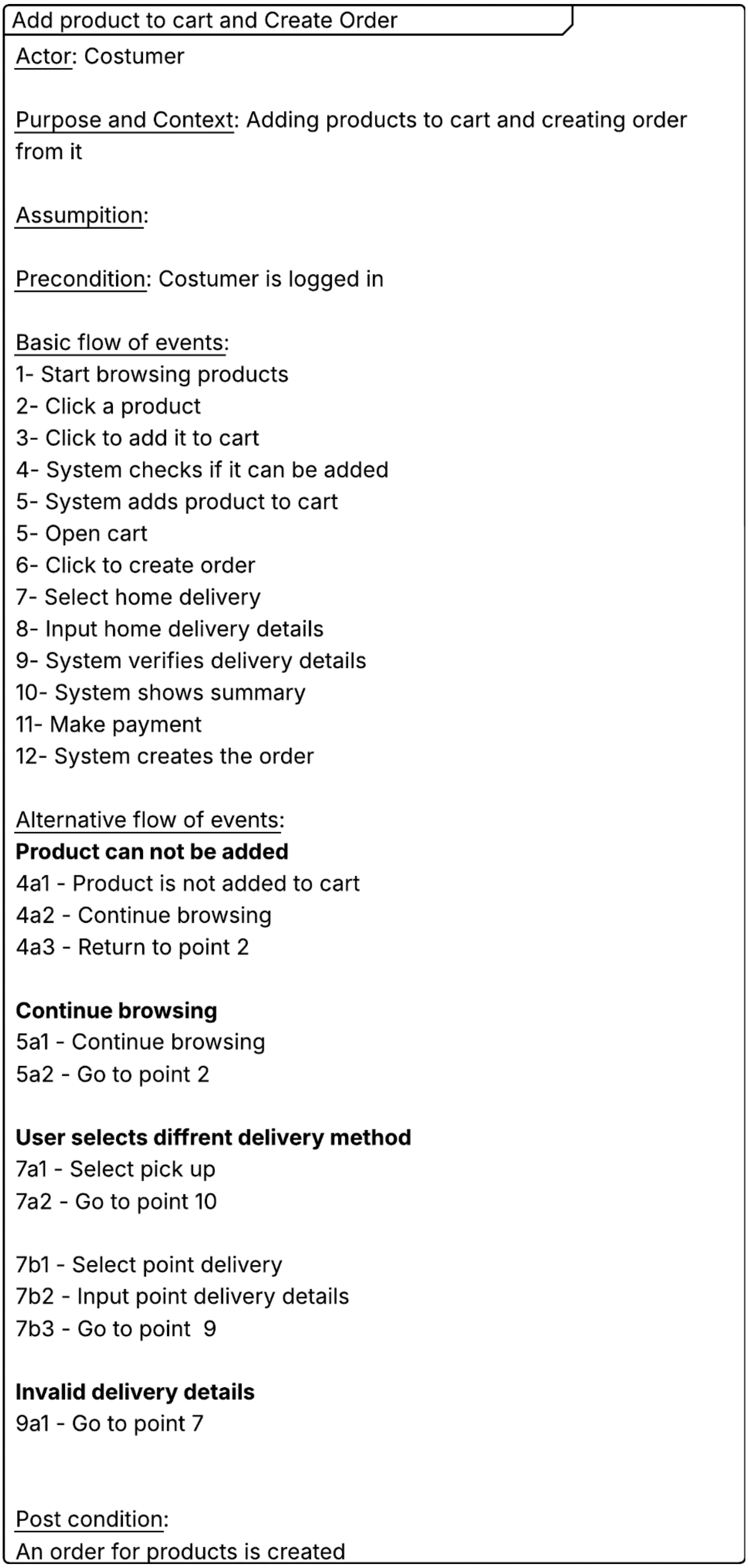
Activity Diagram: “Request Leave”



Activity diagram "Request Leave" – Description

This activity diagram illustrates the leave request process between an employee, the system and a manager. The employee initiates the process by clicking to apply for leave and filling out the form. The system then checks whether all required fields are filled and if the employee's leave balance is sufficient. If either check fails the form is rejected and the employee is notified. If both checks pass the request is sent to the manager for approval. The manager can either approve or reject the request. If approved the system updates the shifts table and notifies the employee. If rejected the employee receives a rejection notification.

Use Case Scenario: “Add product to cart and Create Order”



Add product to cart and Create Order

Actor: Costumer

Purpose and Context: Adding products to cart and creating order from it

Assumpition:

Precondition: Costumer is logged in

Basic flow of events:
1- Start browsing products
2- Click a product
3- Click to add it to cart
4- System checks if it can be added
5- System adds product to cart
5- Open cart
6- Click to create order
7- Select home delivery
8- Input home delivery details
9- System verifies delivery details
10- System shows summary
11- Make payment
12- System creates the order

Alternative flow of events:

Product can not be added
4a1 - Product is not added to cart
4a2 - Continue browsing
4a3 - Return to point 2

Continue browsing
5a1 - Continue browsing
5a2 - Go to point 2

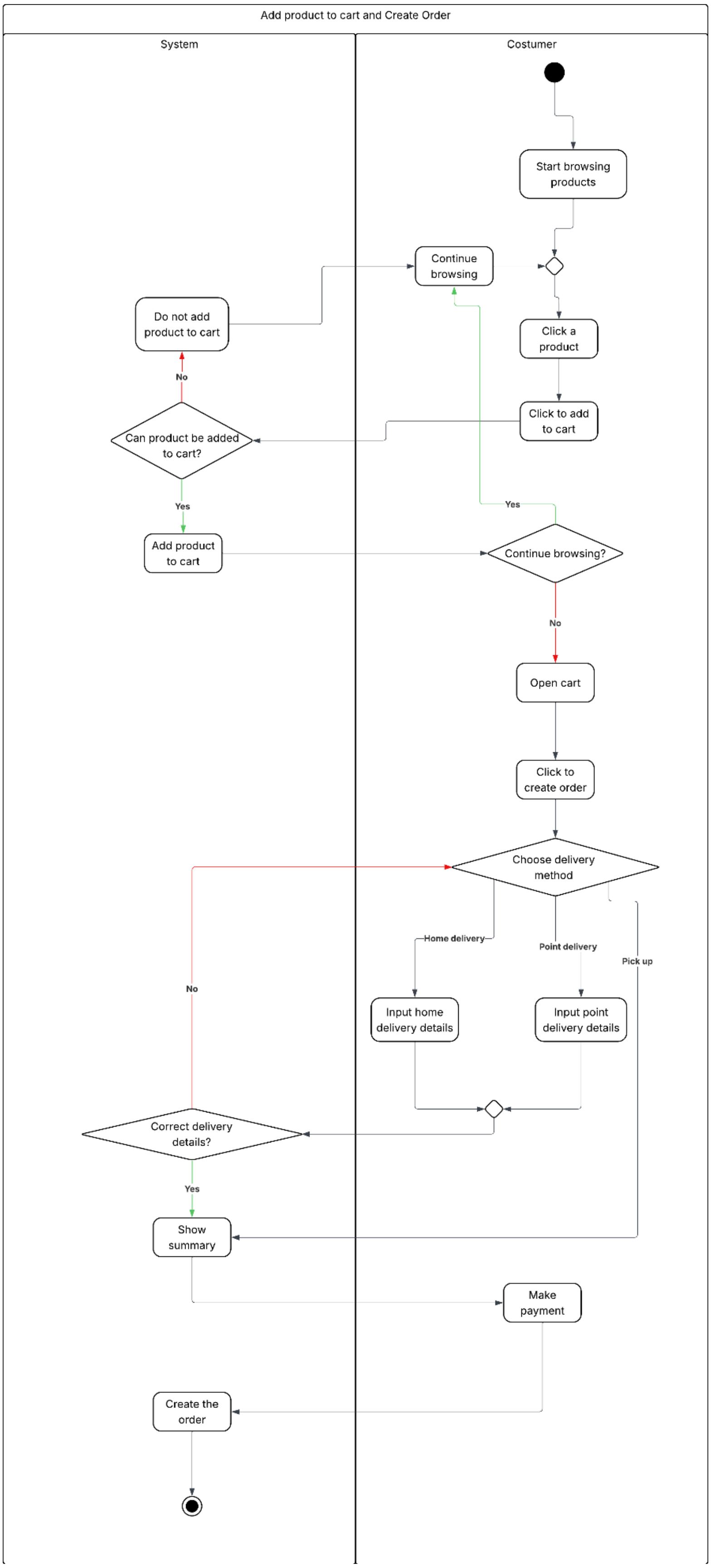
User selects diffrent delivery method
7a1 - Select pick up
7a2 - Go to point 10

7b1 - Select point delivery
7b2 - Input point delivery details
7b3 - Go to point 9

Invalid delivery details
9a1 - Go to point 7

Post condition:
An order for products is created

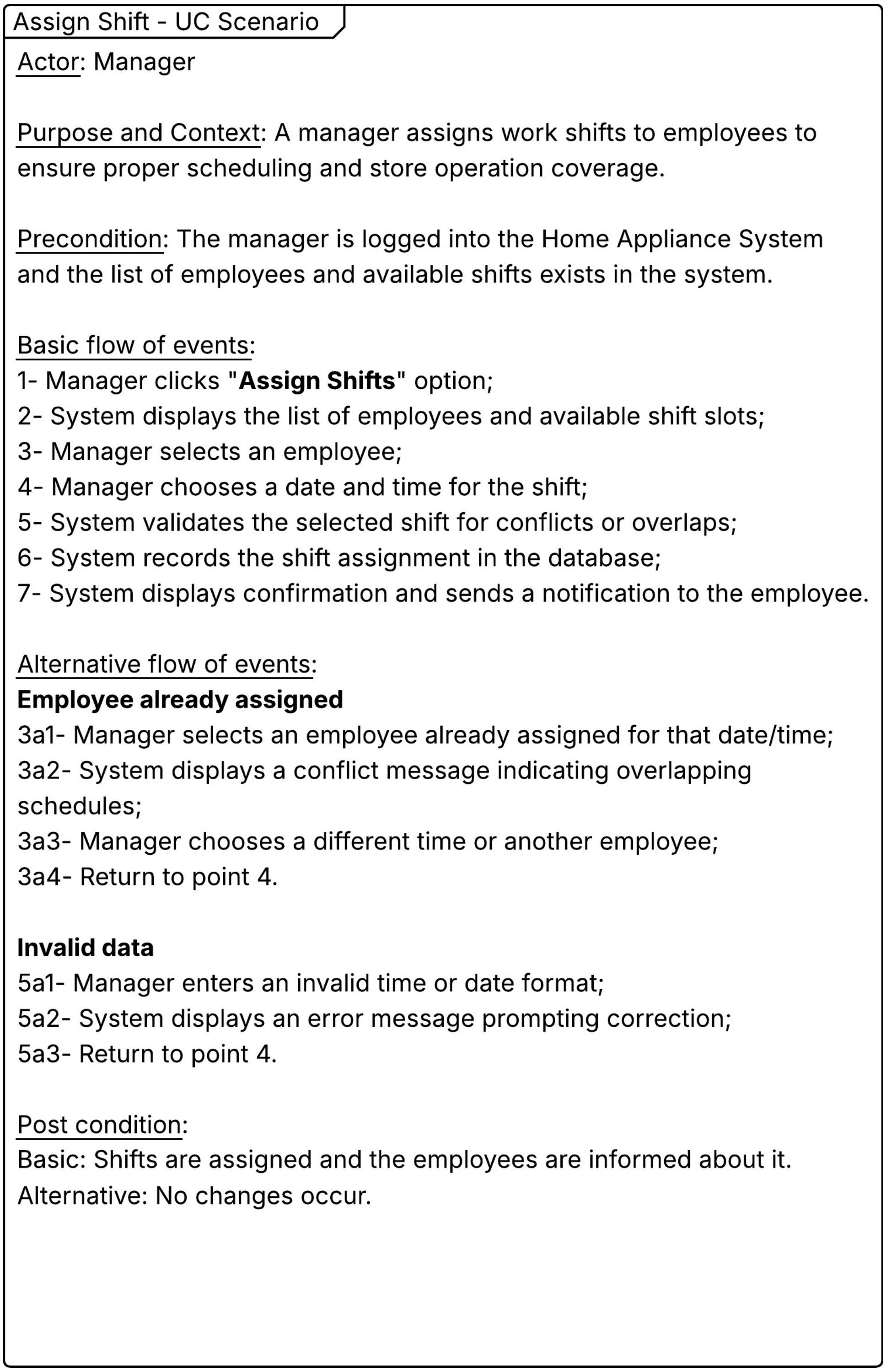
Activity Diagram: “Add product to cart and Create Order”



Activity diagram " Add product to cart and Create Order" – Description

This activity diagram illustrates the process of adding products to the cart and creating an order. The customer begins by browsing products and selecting items to add to the cart. The system checks whether the selected product can be added. If valid it is added to the cart. Once the customer finishes browsing they open the cart and proceed to create an order. The customer then chooses a delivery method either home delivery or point pickup and provides the necessary delivery details. The system verifies the entered information, displays an order summary and the customer makes the payment. After successful payment the system finalizes and creates the order.

Use Case Scenario: “Assign Shift”



Assign Shift - UC Scenario

Actor: Manager

Purpose and Context: A manager assigns work shifts to employees to ensure proper scheduling and store operation coverage.

Precondition: The manager is logged into the Home Appliance System and the list of employees and available shifts exists in the system.

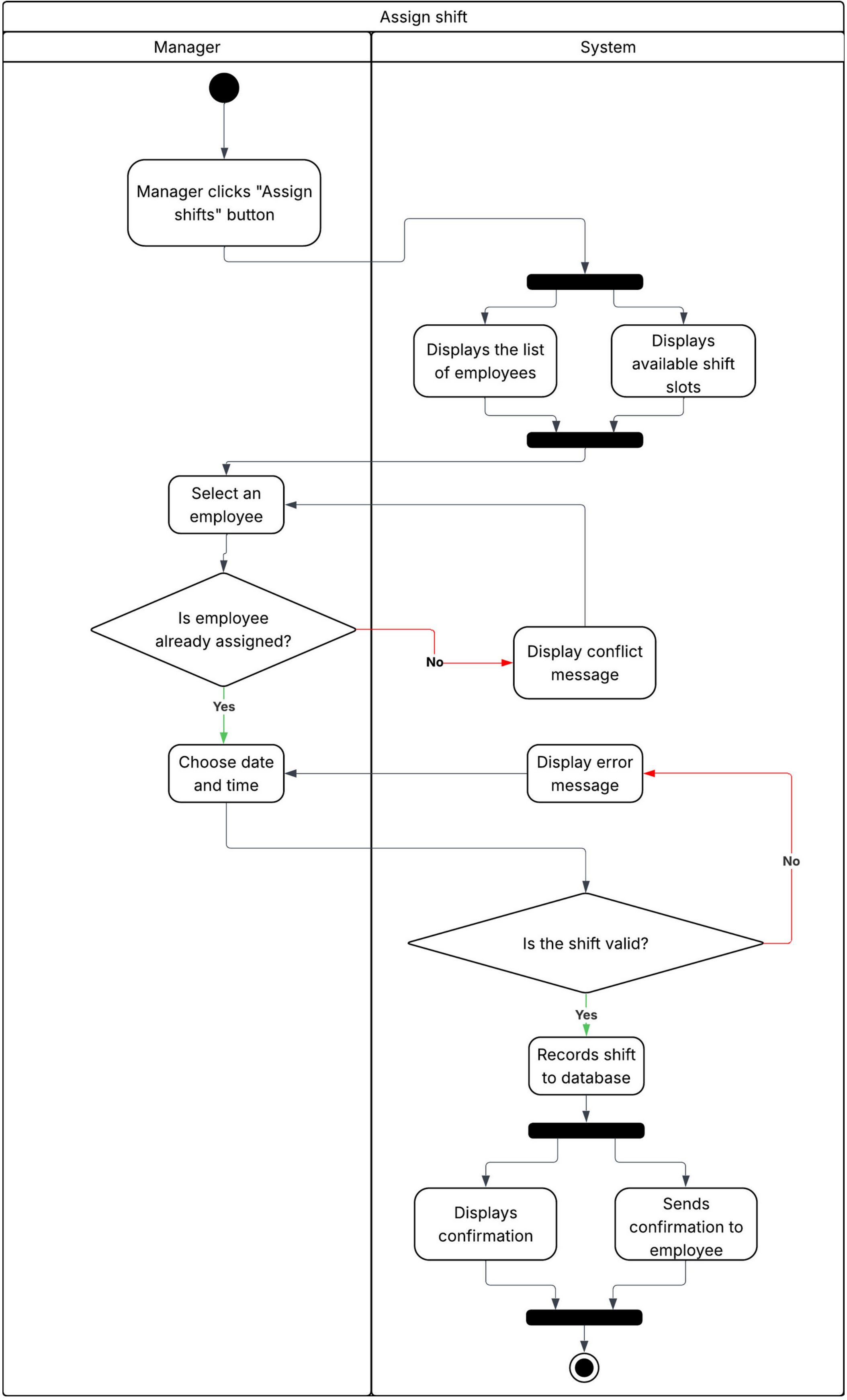
Basic flow of events:
1- Manager clicks "**Assign Shifts**" option;
2- System displays the list of employees and available shift slots;
3- Manager selects an employee;
4- Manager chooses a date and time for the shift;
5- System validates the selected shift for conflicts or overlaps;
6- System records the shift assignment in the database;
7- System displays confirmation and sends a notification to the employee.

Alternative flow of events:
Employee already assigned
3a1- Manager selects an employee already assigned for that date/time;
3a2- System displays a conflict message indicating overlapping schedules;
3a3- Manager chooses a different time or another employee;
3a4- Return to point 4.

Invalid data
5a1- Manager enters an invalid time or date format;
5a2- System displays an error message prompting correction;
5a3- Return to point 4.

Post condition:
Basic: Shifts are assigned and the employees are informed about it.
Alternative: No changes occur.

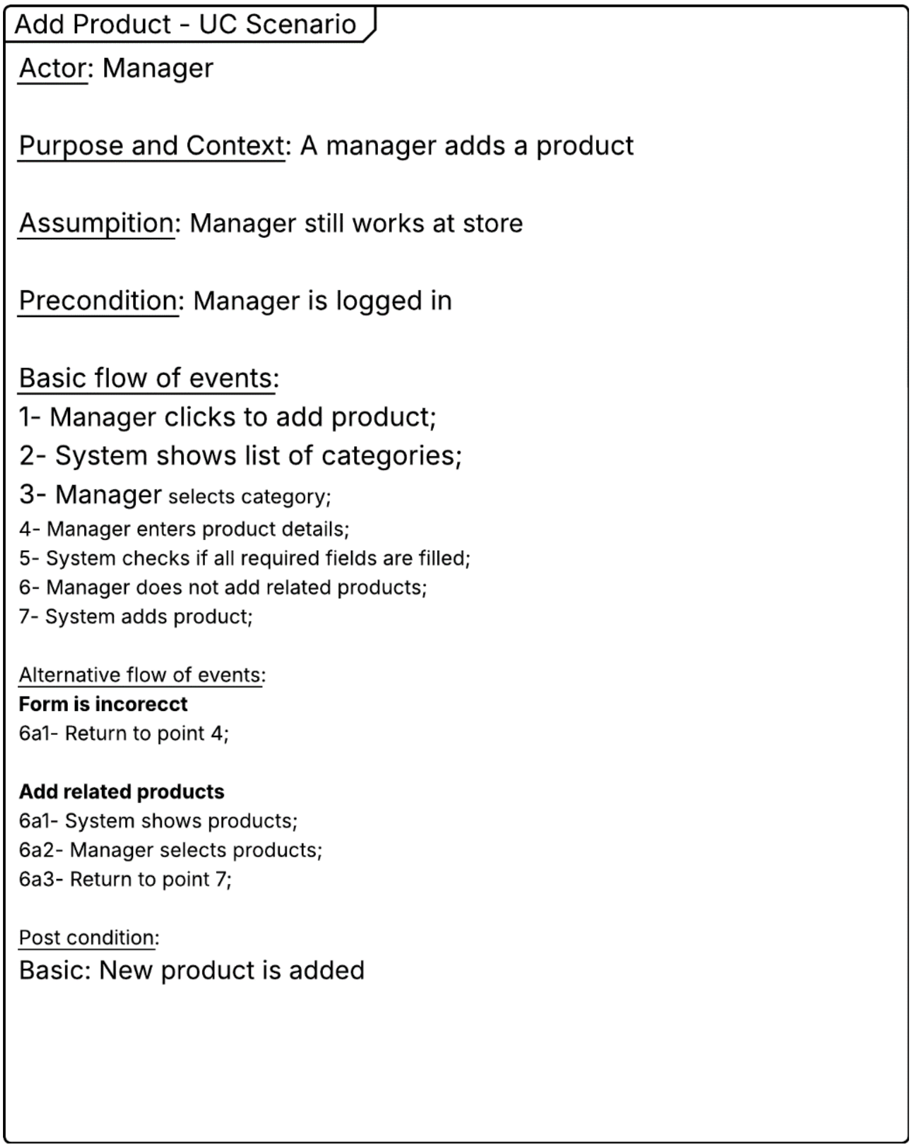
Activity Diagram: “Assign Shift”


Assign shift
Manager
System
Manager clicks "Assign shifts" button
Displays the list of employees
Displays available shift slots
Select an employee
Is employee already assigned?
No
Display conflict message
Yes
Choose date and time
Display error message
No
Is the shift valid?
Yes
Records shift to database
Displays confirmation
Sends confirmation to employee

Activity diagram "Assign Shift " - Description

This activity diagram illustrates how a manager assigns work shifts to employees. The process starts when the manager clicks the "Assign Shifts" button after which the system displays available employees and shift slots. The manager selects an employee, chooses a date and time and the system checks for scheduling conflicts or invalid inputs. If no issues are found the shift is recorded in the database and both confirmation and notification are sent to the manager and employee respectively.

Use Case Scenario: “Add Product”



Add Product - UC Scenario

Actor: Manager

Purpose and Context: A manager adds a product

Assumpition: Manager still works at store

Precondition: Manager is logged in

Basic flow of events:

1- Manager clicks to add product;
2- System shows list of categories;
3- Manager selects category;
4- Manager enters product details;
5- System checks if all required fields are filled;
6- Manager does not add related products;
7- System adds product;

Alternative flow of events:

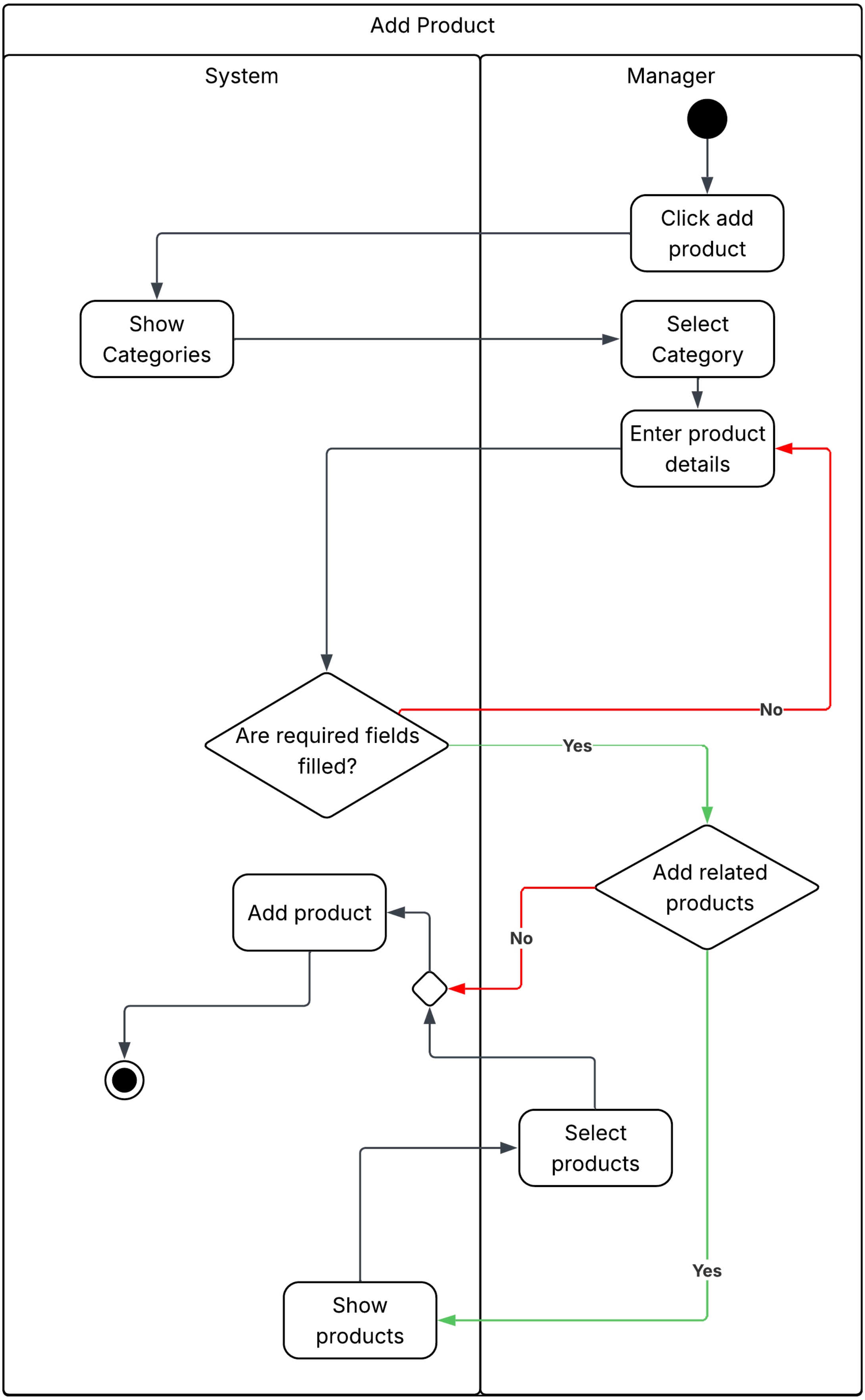
Form is incorecct
6a1- Return to point 4;

Add related products
6a1- System shows products;
6a2- Manager selects products;
6a3- Return to point 7;

Post condition:

Basic: New product is added

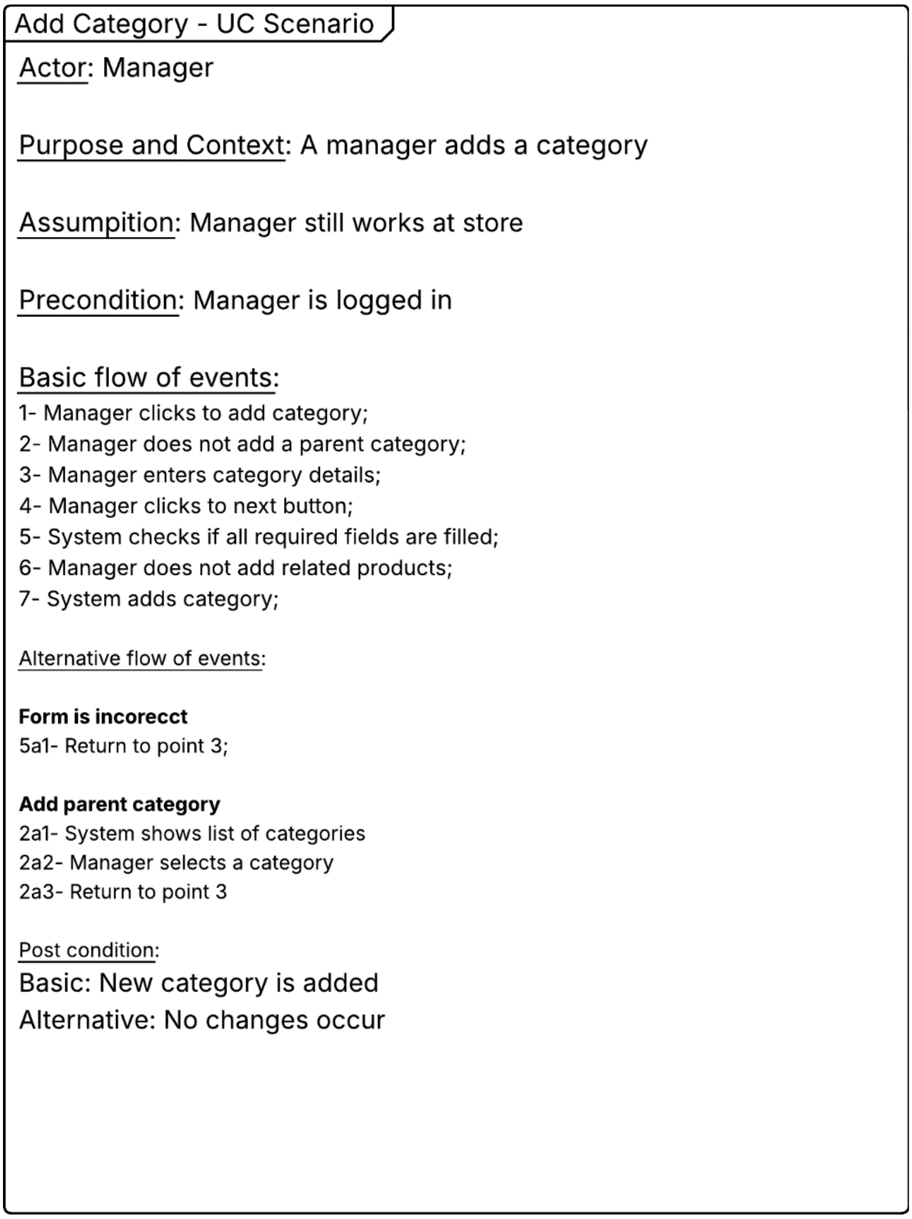
Activity Diagram: “Add Product”



Activity diagram " Add Product" - Description

This activity diagram shows the process of adding a new product by a manager. The manager starts by clicking “**Add product**” then selects a category and enters product details. The system checks whether all required fields are filled. If not the manager must complete the missing information. Once the form is correctly filled the manager can choose to add related products. If there are no related products the system proceeds to add the product directly. After the product is successfully added the manager can select products to view and the system displays them in the product list.

Use Case Scenario: “Add Category”



Add Category - UC Scenario

Actor: Manager

Purpose and Context: A manager adds a category

Assumpition: Manager still works at store

Precondition: Manager is logged in

Basic flow of events:

1- Manager clicks to add category;
2- Manager does not add a parent category;
3- Manager enters category details;
4- Manager clicks to next button;
5- System checks if all required fields are filled;
6- Manager does not add related products;
7- System adds category;

Alternative flow of events:

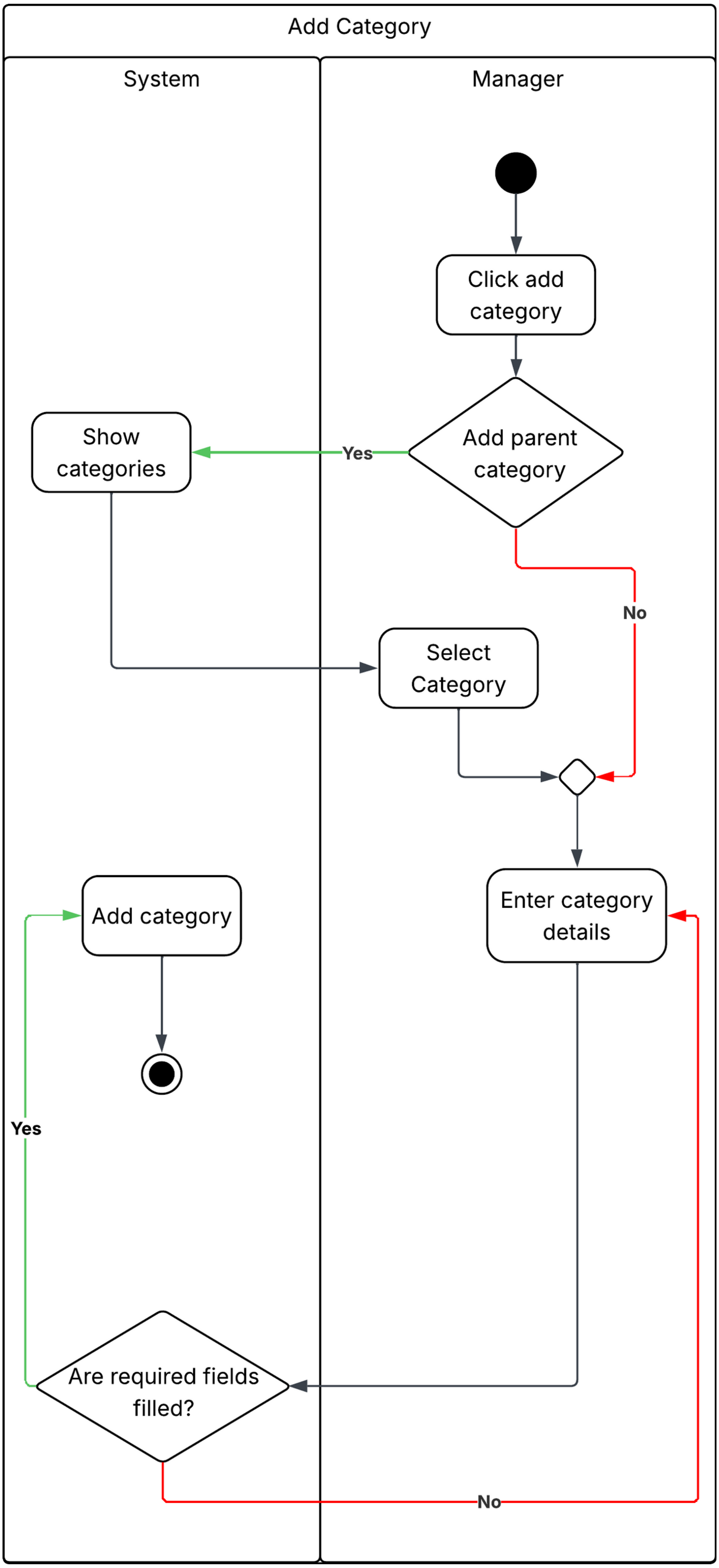
Form is incorecct
5a1- Return to point 3;

Add parent category
2a1- System shows list of categories
2a2- Manager selects a category
2a3- Return to point 3

Post condition:

Basic: New category is added
Alternative: No changes occur

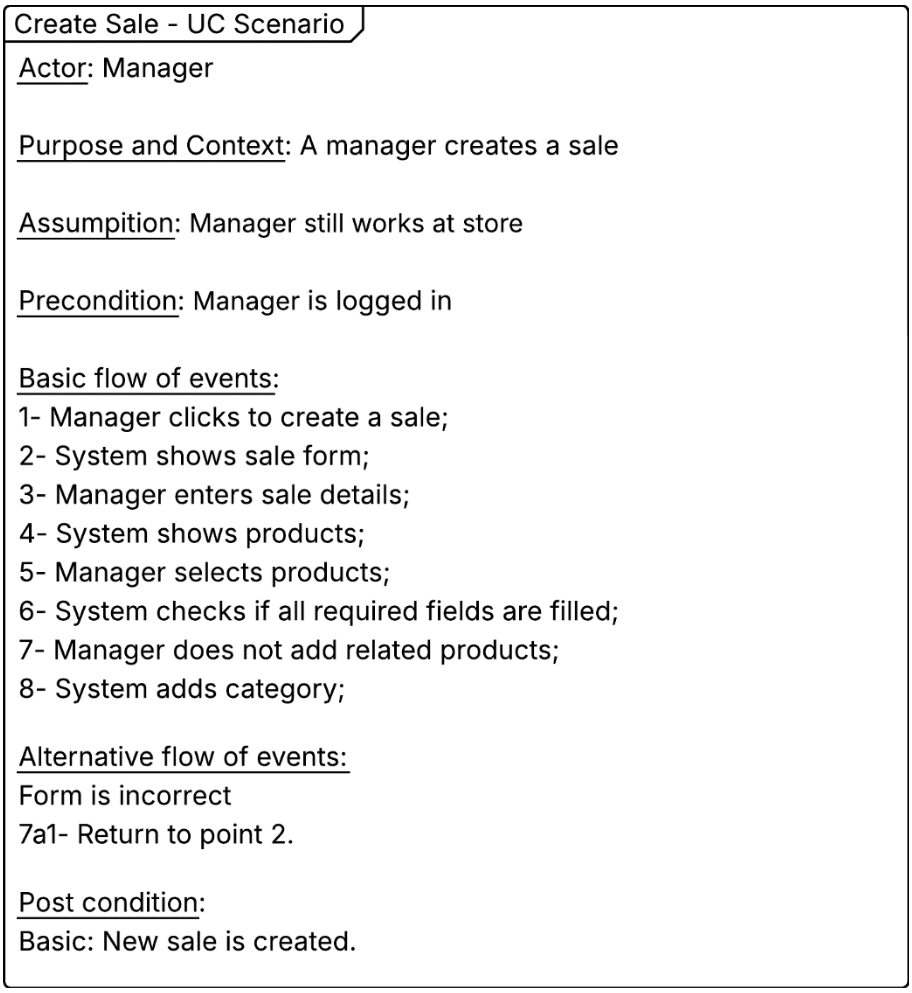
Activity Diagram: “Add Category”



Activity diagram "Add Category" - Description

This activity diagram represents the process of adding a new category by a manager. The process begins when the manager clicks "**Add Category**". The manager then decides whether to add a parent category. If yes the system displays existing categories to choose from. If no parent category is needed the manager proceeds to enter category details. The system verifies that all required fields are filled. If any field is missing the manager is prompted to complete the information. Once all details are correctly provided the system adds the new category and completes the process.

Use Case Scenario: “Create Sale”



Create Sale - UC Scenario

Actor: Manager

Purpose and Context: A manager creates a sale

Assumpition: Manager still works at store

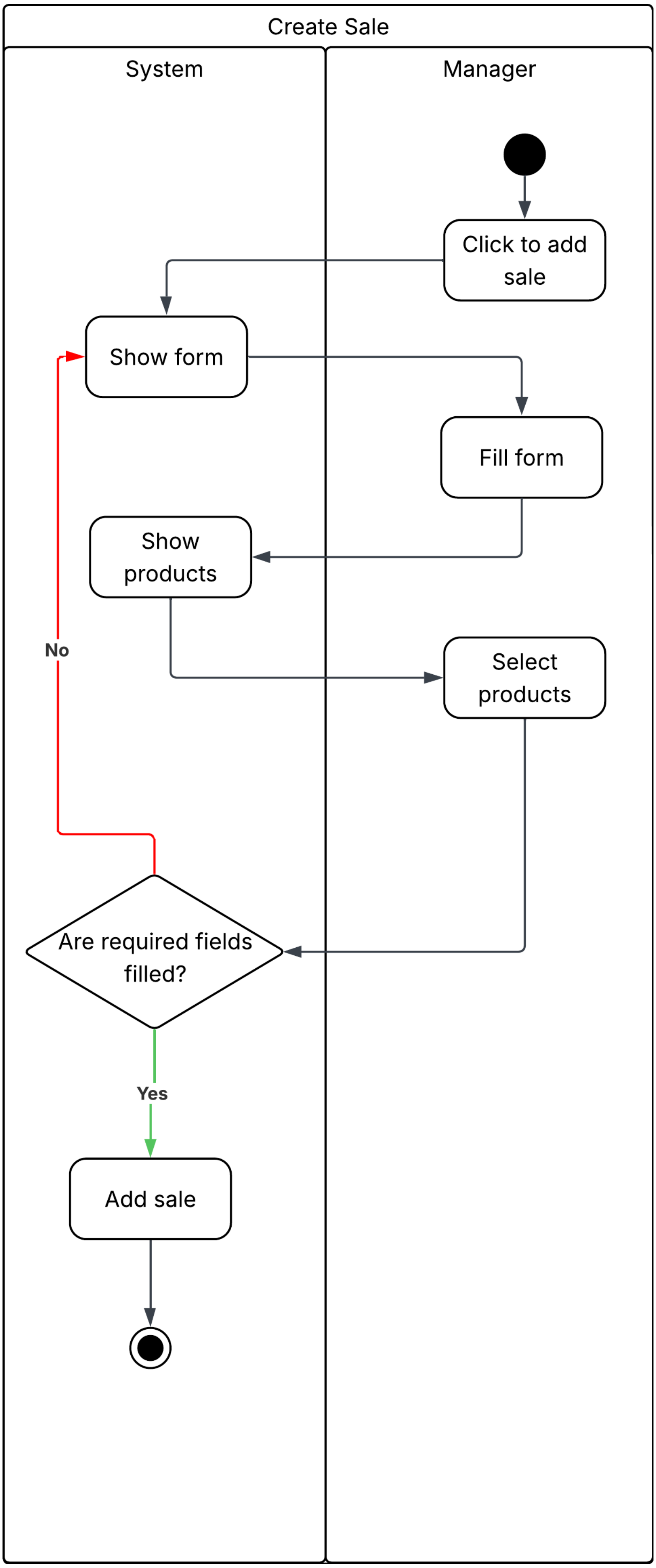
Precondition: Manager is logged in

Basic flow of events:
1- Manager clicks to create a sale;
2- System shows sale form;
3- Manager enters sale details;
4- System shows products;
5- Manager selects products;
6- System checks if all required fields are filled;
7- Manager does not add related products;
8- System adds category;

Alternative flow of events:
Form is incorrect
7a1- Return to point 2.

Post condition:
Basic: New sale is created.

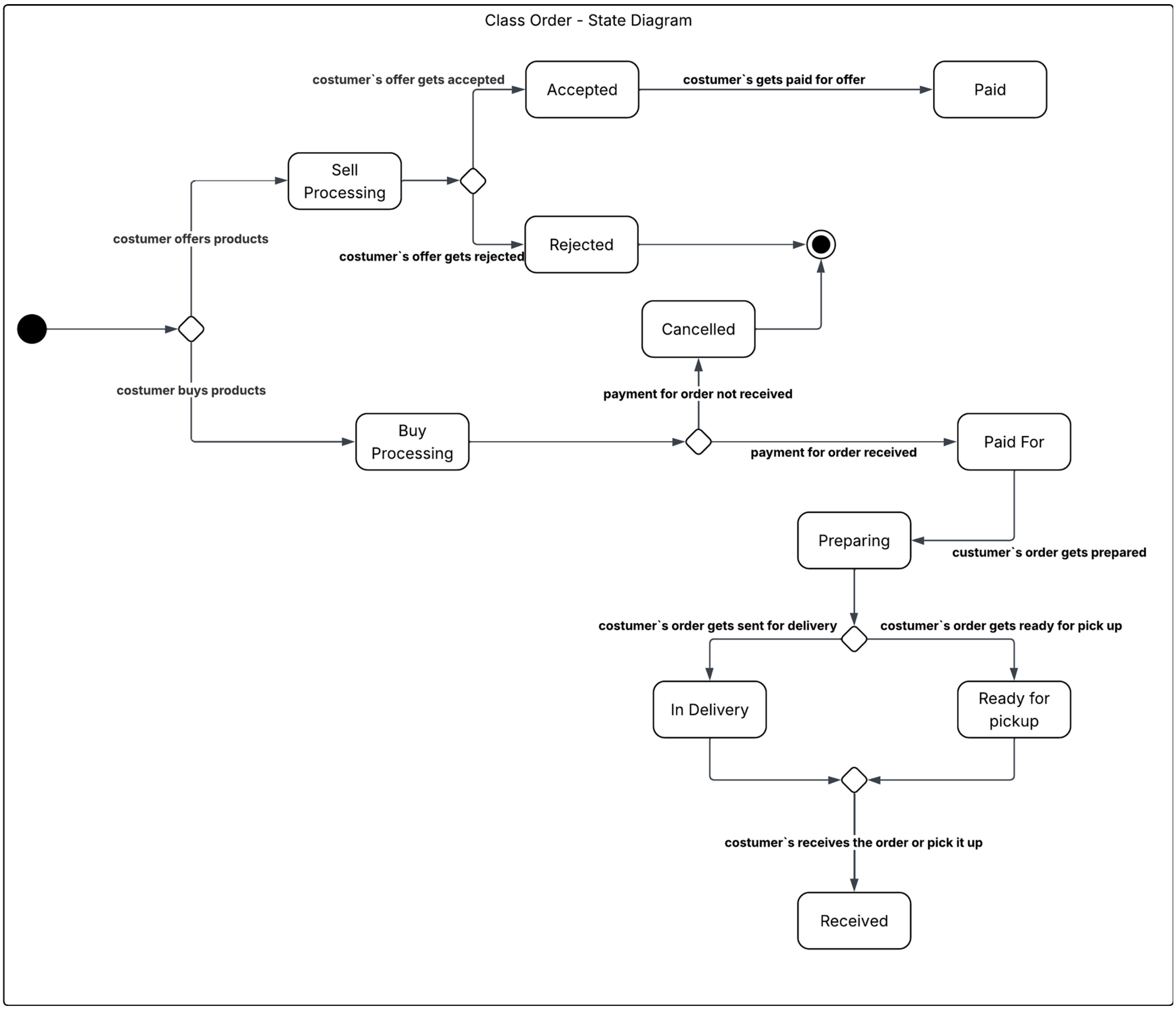
Activity Diagram: “Create Sale”



Activity diagram "Create Sale" - Description

This activity diagram illustrates the process of creating a sale by a manager. The process starts when the manager clicks "**Add Sale**". The system then displays the sale form and the manager proceeds to fill in the required information and select products for the sale. Once the details are entered the system checks whether all required fields are filled. If any information is missing the manager is prompted to complete the form. When the form is correctly filled out the system adds the sale and the process is successfully completed.

State Diagram – Order Class


Class Order - State Diagram
costumer`s offer gets accepted
Accepted
costumer`s gets paid for offer
Paid
Sell Processing
costumer offers products
Rejected
costumer`s offer gets rejected
Cancelled
costumer buys products
payment for order not received
Buy Processing
payment for order received
Paid For
Preparing
custumer`s order gets prepared
costumer`s order gets sent for delivery
costumer`s order gets ready for pick up
In Delivery
Ready for pickup
costumer`s receives the order or pick it up
Received

State Diagram – Description

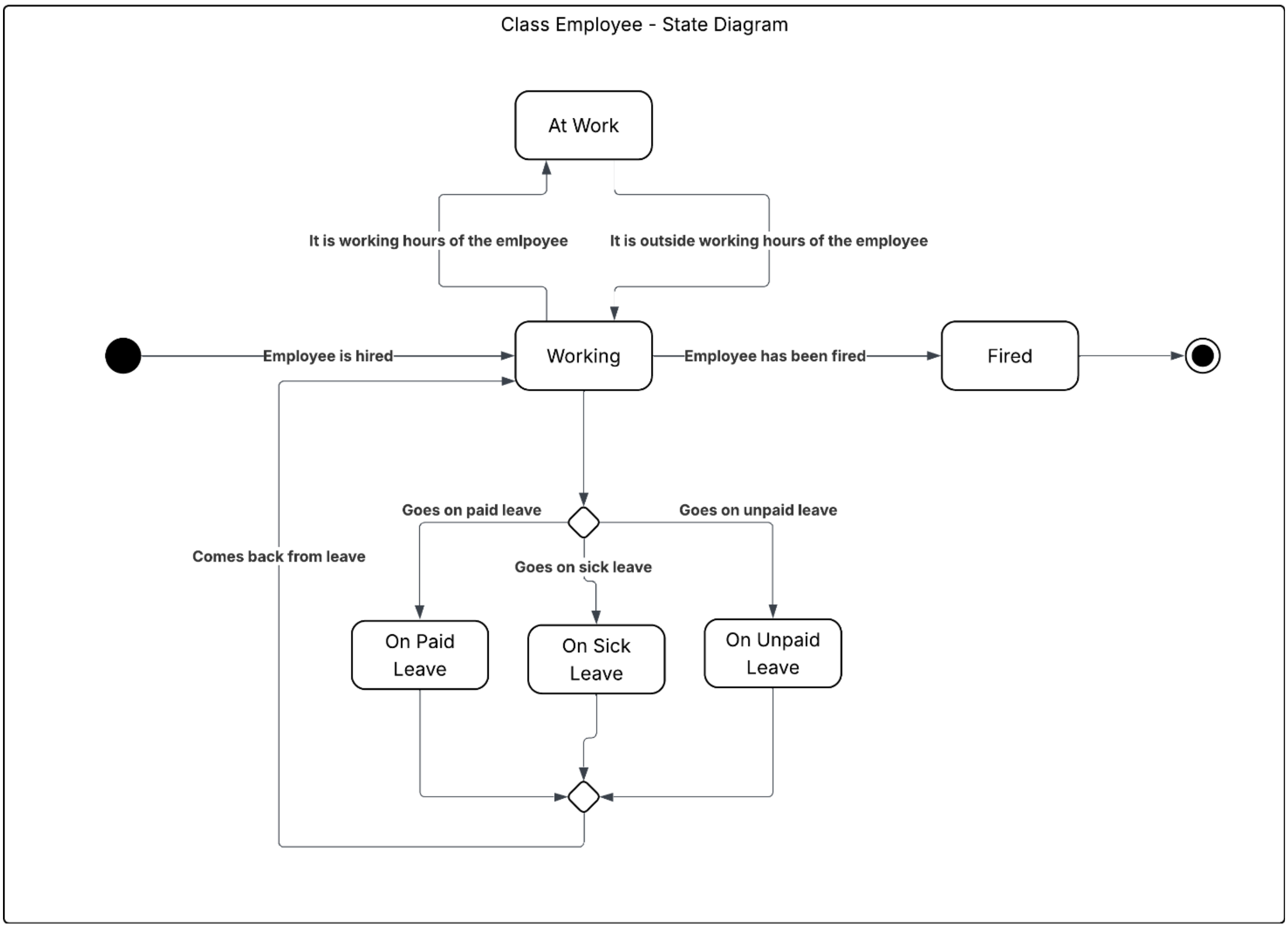
The state diagram represents the different stages of an Order. An order can be created when a customer either offer products for sale or buys products.

Depending on the outcome of the actions and payment processing the order can go through several states.

"**Sell processing**" - this state occurs when the customer offers products. The order moves to:
"**Accepted**" if the offer is approved.
"**Rejected**" if the offer is declined.
2. "**Accepted**" - when the customer’s offer is accepted the order proceeds to the "**Paid**" state after successful payment.
3. "**Paid**" - indicates that the customer has received payment for their offer.
4. "**Buy Processing**" - this state occurs when the customer buys products. After processing:
If payment is received the order moves to "**Paid For**".
If payment fails the order transitions to "**Cancelled**".
5. "**Paid For"** - once payment is confirmed the order enters "**Preparing**" where the store prepares the products.
6. "**Preparing**" - during preparation, the order can move to:
"**In Delivery"** if the order is shipped to the customer.
"**Ready for Pickup**" if the order is available for collection.
7. "**In Delivery"** and "**Ready for Pickup**" - these states represent the order being on its way or waiting for the customer.
Once the customer receives or picks up the order, it transitions to "**Received**".
8. "**Received**" - the final state meaning the customer has obtained the order successfully.
9. "**Canceled**" - the order is terminated due to a failed payment or rejection. This is also a possible end state.

The object lifecycle can thus end in two ways:
"**Received**" – when the order is successfully completed.
"**Cancelled**" – when the process fails due to rejection or payment issues.

State Diagram – Employee Class

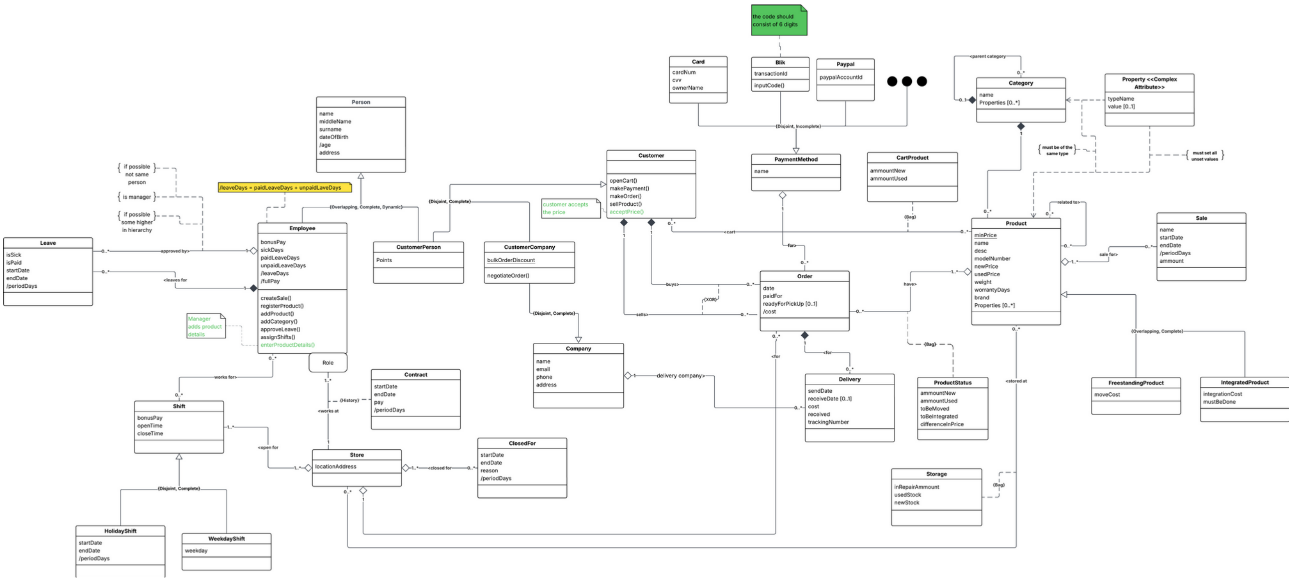


Dynamic Analysis

In the Use Case Scenario when a Customer is selling a product to the store he has to accept the price that the employee proposes to him in order to make the sell final. We could implement this function in our class diagram to know that the employee has the final decision in this process.

In the Use Case Scenario when a Manager is adding a product he has to input all the information about it. That's why we could introduce a method for the Manager called enterProductDetails(). This method would be used when his adding the product but maybe in the future it could have some more use cases like maybe automating the inventory management.

Changes to the Class Diagram after dynamic analysis



Class diagram changes - Description

We added methods for the Manager like: createSale(), addProduct(), addCategory(), approveLeave() and assignShifts().
For the Employee: checkProductStatus() and also enterProductDetails().
For the Customer class we introduced: openCart(), makePayment(), makeOrder(), sellProduct() and acceptPrice().

Design Decisions

Data persistance

Each class which needs to be saved implements Extent, in which the logic what handels saving and loading of master lists and also getting a immutable master list, and they store a master list of all objects of that class in memory. That list is saved each time a new object is removed and/or created also it is saved on each change to an object of that class. It is also loaded once on class initialization.

Data validation

Data is validated during setters and constructors. When data is invalid an appropriate Exception is thrown.

Multi-Value Attributes

They are represented by a ArrayList of a appropriate type.

Derived Attributes

They are represented by making a getter which calculates that attribute.

Optional Attributes

The getters of those attributes return the value wrapped in Optional class.

Types Used For Numbers

For real numbers we use BigDecimal to avoid floating point errors, because we work with money. For natural numbers we use long, because they can represent bigger numbers.

Data Encapsulation

Every attribute is private, but they have public getters and setters. This for data validation and saving.

Any Type

For any type we use generic classes and specify that a type must be serializable.

Assosiations

Basic Assosiations

They are represneted by attributes in their resprective classes. Most classes store each side of the association, aka there is a lot of doubling of data (If they are not present they are just null or empty). Groups of objects are represeented by ArrayList or Sets. ArrayLists are supposed to be checked and add data to the collection only once.

AddMethods either add an object to a collection or create a new object

RemoveMethods either delete the object remove it from a collection and/or sett attributes to null

ChangeMethods are done using setters

Present reverse connections store data on one side of the connection and one side has less assosiation methods.

Deletion happens in the delete() method.

Errors and Validating Assosiations

They are supposed to throw propper exceptions, while setting them.

Agreggate Assosiations

Are those present in the cunstructor but not in the delete method.

Composition Assosiations

Are those present in the constructor and there are present in delete() methods of the assosiation.

Qualified Assosiations

The qualified assosiation got absorbed into a class assosiation of Conrtact.

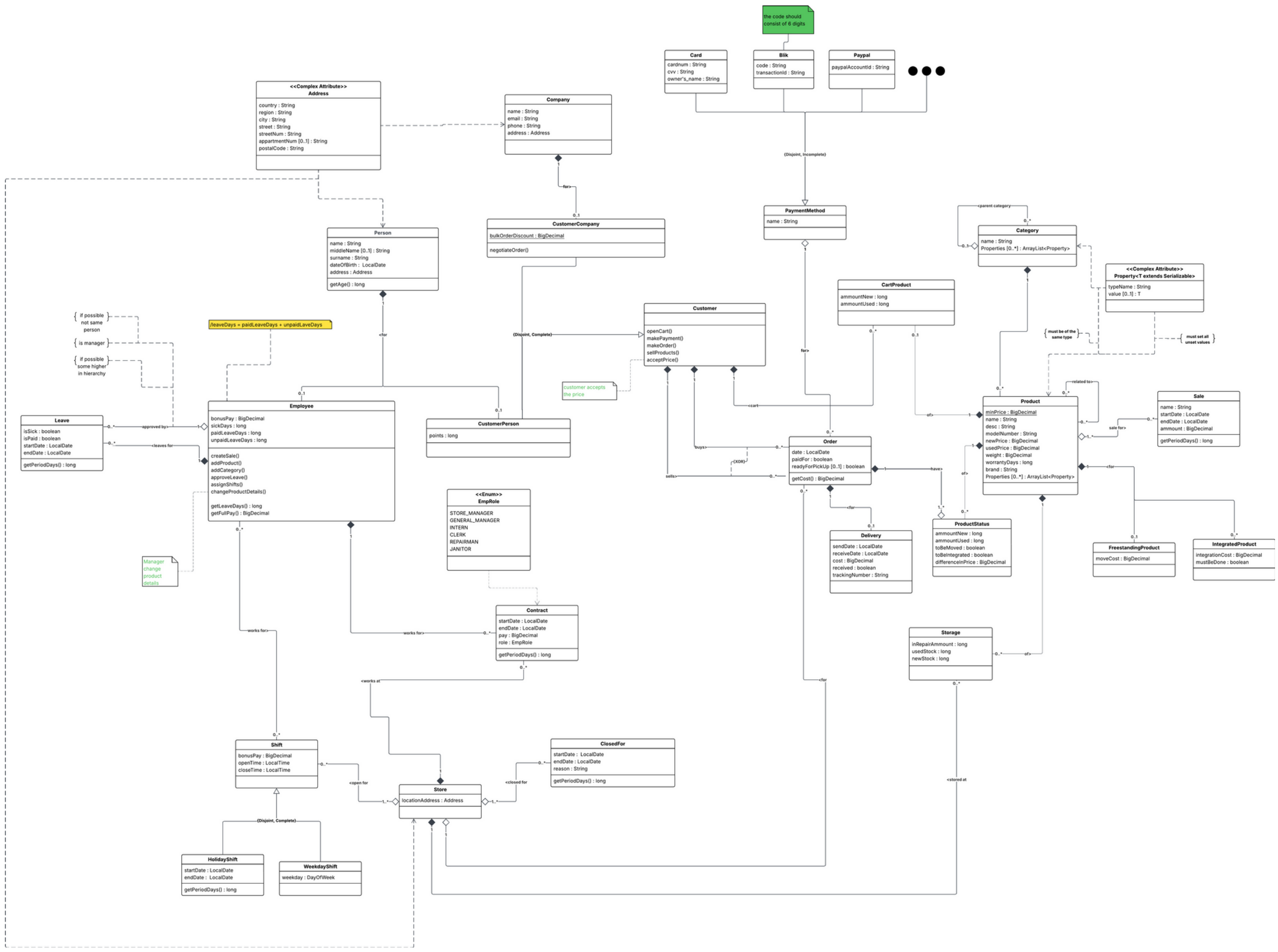
Class Assosiations

They link one instance of a class with another. They either are stored on one side and have the other as attribute or have both as an attribute.

Reflex Assosiations

They sorted in the instance of object of the class and are mostly and are mostly treated like normal assosiatioins.

Design Diagram


the code should consist of 6 digits
Card
cardnum : String
cvv : String
owner's_name : String
Blik
code : String
transactionId : String
Paypal
paypalAccountId : String
{Disjoint, Incomplete}
PaymentMethod
name : String
<<Complex Attribute>>
Address
country : String
region : String
city : String
street : String
streetNum : String
appartmentNum [0..1] : String
postalCode : String
Company
name : String
email : String
phone : String
address : Address
for>
CustomerCompany
bulkOrderDiscount : BigDecimal
negotiateOrder()
Person
name : String
middleName [0..1] : String
surname : String
dateOfBirth : LocalDate
address : Address
getAge() : long
<for
{Disjoint, Complete}
Customer
openCart()
makePayment()
makeOrder()
sellProducts()
acceptPrice()
customer accepts the price
CustomerPerson
points : long
if possible not same person
is manager
if possible some higher in hierarchy
/leaveDays = paidLeaveDays + unpaidLaveDays
Leave
isSick : boolean
isPaid : boolean
startDate : LocalDate
endDate : LocalDate
getPeriodDays() : long
approved by>
<leaves for
Employee
bonusPay : BigDecimal
sickDays : long
paidLeaveDays : long
unpaidLeaveDays : long
createSale()
addProduct()
addCategory()
approveLeave()
assignShifts()
changeProductDetails()
getLeaveDays() : long
getFullPay() : BigDecimal
Manager change product details
<<Enum>>
EmpRole
STORE_MANAGER
GENERAL_MANAGER
INTERN
CLERK
REPAIRMAN
JANITOR
Contract
startDate : LocalDate
endDate : LocalDate
pay : BigDecimal
role : EmpRole
getPeriodDays() : long
works for>
works for>
<works at
Shift
bonusPay : BigDecimal
openTime : LocalTime
closeTime : LocalTime
{Disjoint, Complete}
HolidayShift
startDate : LocalDate
endDate : LocalDate
getPeriodDays() : long
WeekdayShift
weekday : DayOfWeek
<open for
Store
locationAddress : Address
<closed for
ClosedFor
startDate : LocalDate
endDate : LocalDate
reason : String
getPeriodDays() : long
CartProduct
ammountNew : long
ammountUsed : long
for>
<cart
buys>
{XOR}
sells>
Order
date : LocalDate
paidFor : boolean
readyForPickUp [0..1] : boolean
getCost() : BigDecimal
<for
Delivery
sendDate : LocalDate
receiveDate : LocalDate
cost : BigDecimal
received : boolean
trackingNumber : String
have>
ProductStatus
ammountNew : long
ammountUsed : long
toBeMoved : boolean
toBeIntegrated : boolean
differenceInPrice : BigDecimal
<parent category
Category
name : String
Properties [0..*] : ArrayList<Property>
<<Complex Attribute>>
Property<T extends Serializable>
typeName : String
value [0..1] : T
must be of the same type
must set all unset values
of>
of>
related to>
Product
minPrice : BigDecimal
name : String
desc : String
modelNumber : String
newPrice : BigDecimal
usedPrice : BigDecimal
weight : BigDecimal
worrantyDays : long
brand : String
Properties [0..*] : ArrayList<Property>
sale for>
Sale
name : String
startDate : LocalDate
endDate : LocalDate
ammount : BigDecimal
getPeriodDays() : long
<for
FreestandingProduct
moveCost : BigDecimal
IntegratedProduct
integrationCost : BigDecimal
mustBeDone : boolean
Storage
inRepairAmmount : long
usedStock : long
newStock : long
of>
<stored at
<for
0..1
0..*
1
1..*