Tab 1

Notes before we start : Any new additions since the last assignment will be highlighted in red in the **tables of contents and in the respective sections**

After the end of updating this document the following should be included in your solution :

- **User requirement**
- **Use case diagram**
- **Analytical class diagram**
- **Design class diagram**
- **Your chosen use case scenarios**
- **Your chosen activity diagram**
- **Your chosen state diagram**
- **Class diagram after dynamic analysis (do note that after adding the first design diagram, the final diagram after dynamic analysis should be converted to a design diagram.)**

To clarify your documentation must include 3 class diagrams :

- Analytical
- Design
- **Design diagram after dynamic analysis (more information can be found at the final page in regards to the conversion)**

Hobby Store Management System - Software Requirements & Implementation Specifications

Table of contents

User Requirements

The Hobby Store is looking to set up an electronic website system. The system will be used by the customers to browse the selection available in the store, with the possibility to order any merchandise available. The system is also used by employees of the store to help organize orders.

Customers are able to view the items available in the store and add them to their cart. They can search for the items using the website search tab or by directly clicking on the item they wish to add. Upon clicking on the Item, the system displays the content of the item and gives the user the option to add it with the specific desired quantity. Once the item is added, the website presents the option for the user to continue browsing the website or immediately go to finalize the order. The system should keep track of the current available quantity of a given item in the store and should inform the users in the case that the item is not available (showcased with a message when displaying the item page). In the case that an item is available, but the chosen quantity is higher than the available stock. The system would notify the user about the lack of quantity and request that they choose a lower quantity value.

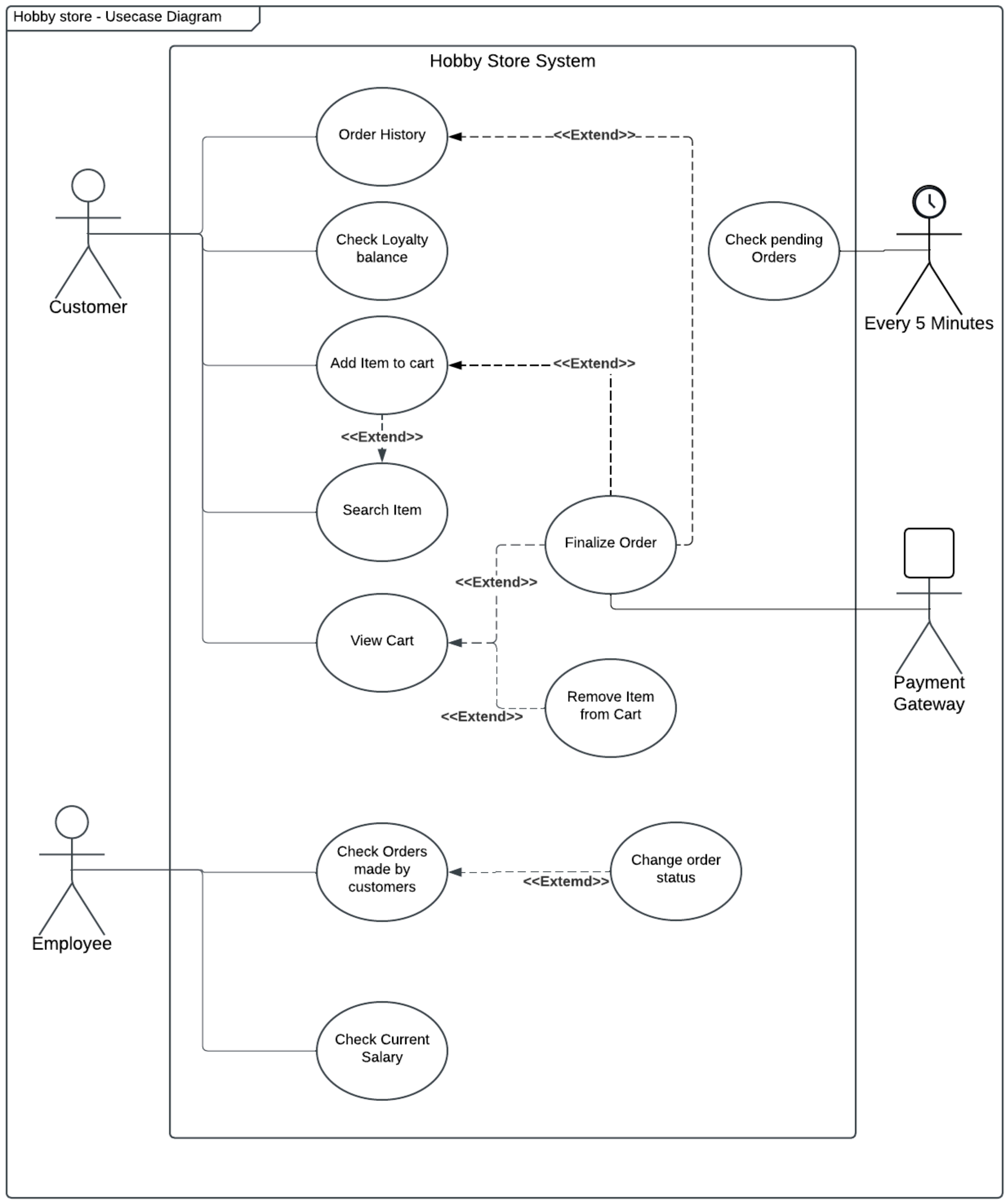
Customers can also view the content of their cart, where they can remove certain items or finalize the order (which starts the order finalization process). Payments for a given order can either be done using the payment gateway or the in-currency loyalty points that a customer obtains after the completion of the order finalization process. In the case of issues with payment, the customers are given a 2-hour pending period where the order can be paid for again before canceling. To attempt to finalize an order after failing the first time, the customers must access the order from the order history.

The store does not deliver any items. All items must be picked up from the hobby store directly, and the information regarding the process of a given order should be modified by the employees currently working in the store in the system. Employees of the store mainly use the system to keep track of all the orders made by the customer and change the status of a given order to the following three options:

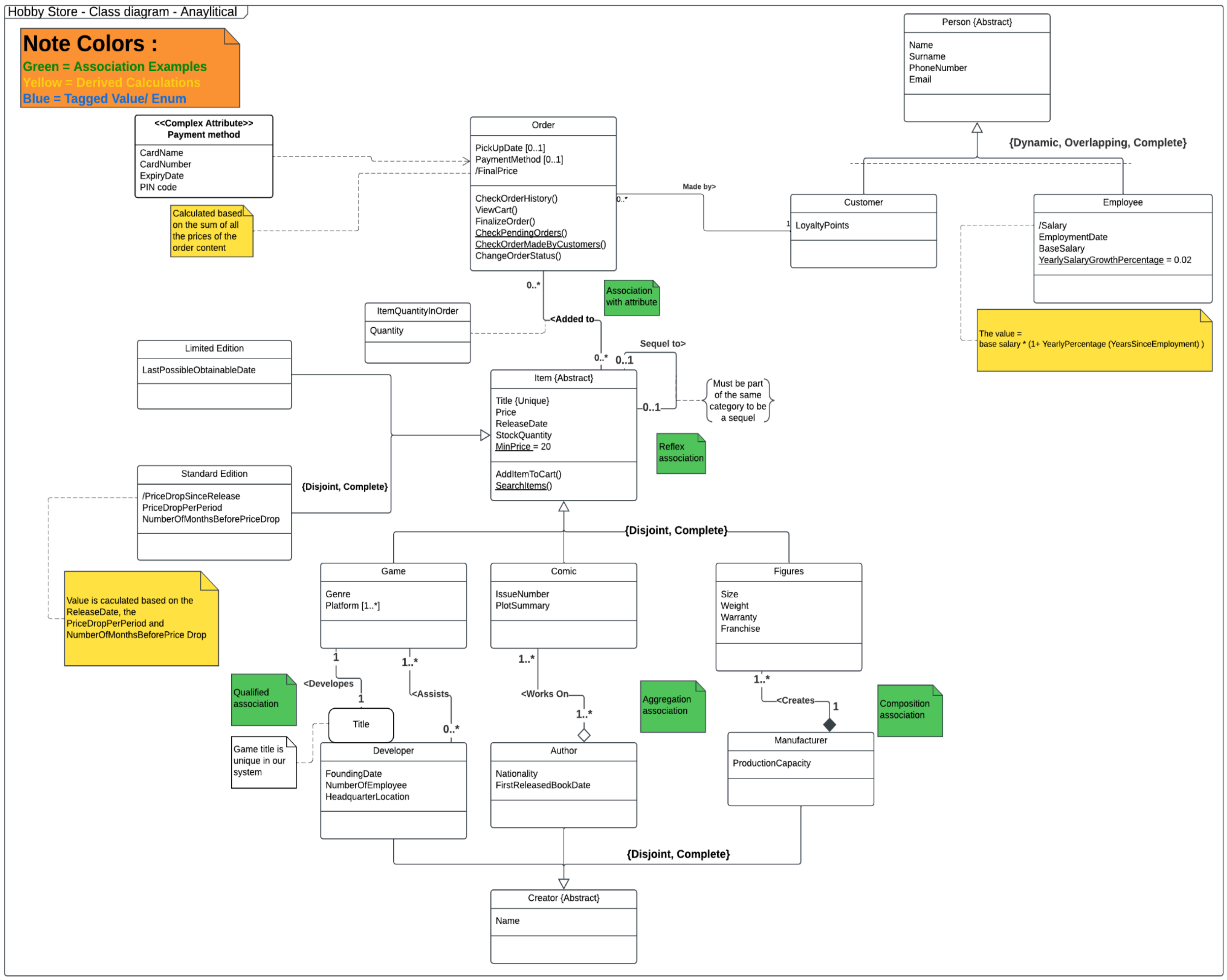
1. Preparing the order: After an order is accepted
2-Ready to pick up: Once the order is completed
3-Completed: Once the customer has picked up the order
The employees have the option to display their current salary info along with their work history (how many days they have been working in the store). The base salary is increased for all employees by 2% after every additional year that an employee has worked for, and should be displayed when checking for their current salary through the system.

The store sells three types of items (comics, games, and figures). Each of these items has specific information about them that needs to be also stored (e.g., the platforms a game is available on, and the developers working on a given game or the comic book issue number). Additionally, all items should have information about their title, price, and release date. The minimum price for any item sold in the store is 20 PLN.

Use case diagram



Class diagram - Analytical



New note for the students : Now that the use case diagram has been added, we want to add all the use cases showcased on the diagram as methods in our system.

Analytical Class diagram - Description

Note for student: Briefly describe the class diagram showcased in this section.

Mention things that are not clearly stated in the requirement description

Class diagram - Design

Note for students : now that we have begun working on the implementation of the class diagram, we are required to create the design version of the class diagram

A design diagram should denote two key things :

1-The language in which you will be coding in

2-The exact implementation information for the elements on the diagram. For now will only create the design diagram with the attribute typing and enumeration notations (as we are only working on implementing that part in the current assignment).

After assignment 6 (association implementation) we will update the diagram once again to reflect the changes of association implementation.

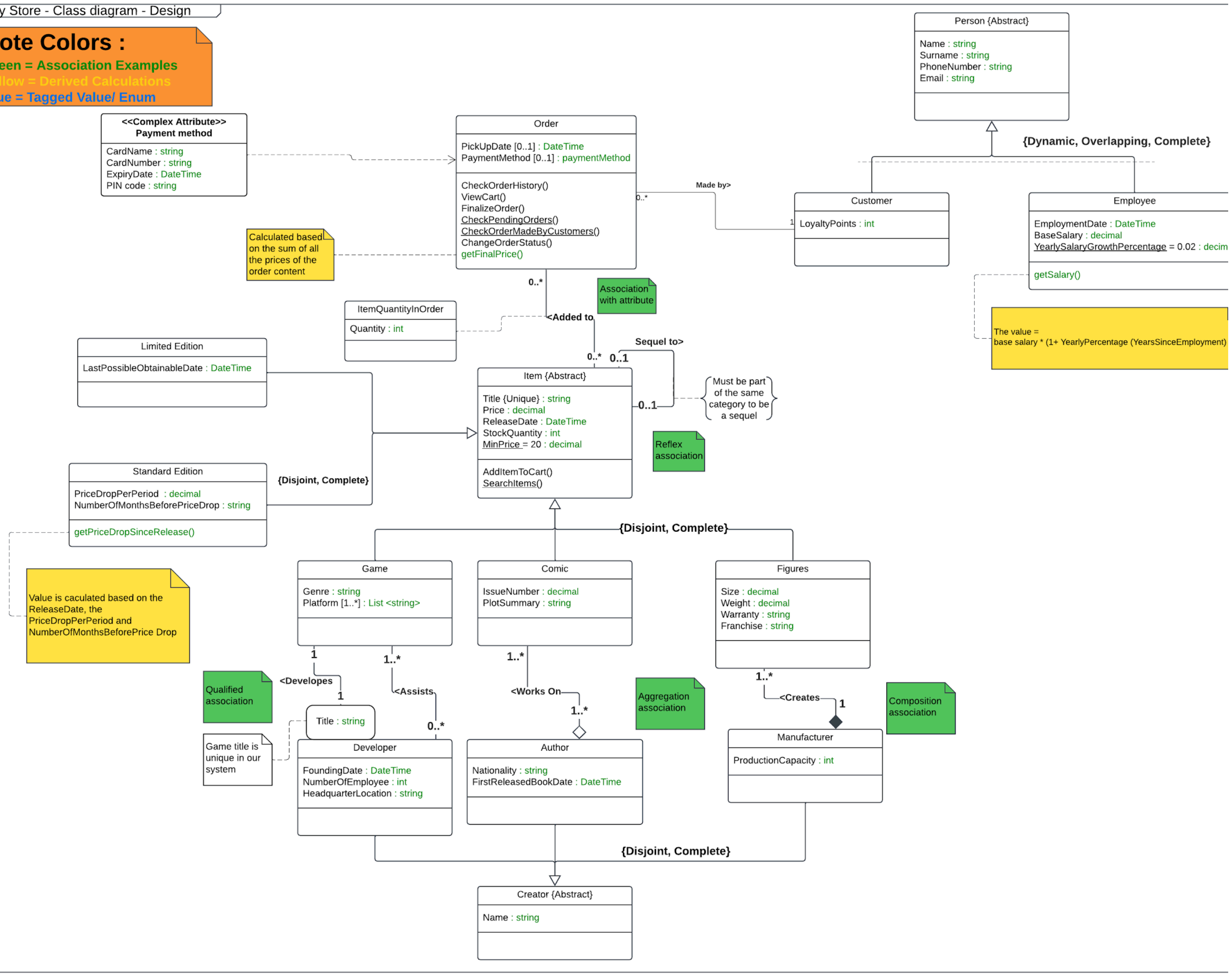
After assignment 7 (inheritance implementation) we will update the diagram one last time to reflect the implementation choices for our inheritance.

For now we do not concern ourselves with associations and inheritance.

What has changed from the analytical class diagram (so far) :

1. All the attribute types are added (in relation to the language of choice)
2. Tagged values are changed to enumerations to reflect their implementation choices (**example can be found on the class diagram after dynamic analysis at the final pages)**
3. Derived attributes have been converted to get methods (the note on how to implement them must be kept)

All the changes from analytical to design have been highlighted in green (it should also be changed highlighted in your solution.


y Store - Class diagram - Design
ote Colors :
een = Association Examples
llow = Derived Calculations
ue = Tagged Value/ Enum
<<Complex Attribute>>
Payment method
CardName : string
CardNumber : string
ExpiryDate : DateTime
PIN code : string
Order
PickUpDate [0..1] : DateTime
PaymentMethod [0..1] : paymentMethod
CheckOrderHistory()
ViewCart()
FinalizeOrder()
CheckPendingOrders()
CheckOrderMadeByCustomers()
ChangeOrderStatus()
getFinalPrice()
Calculated based on the sum of all the prices of the order content
Person {Abstract}
Name : string
Surname : string
PhoneNumber : string
Email : string
{Dynamic, Overlapping, Complete}
Made by>
0..*
1
Customer
LoyaltyPoints : int
Employee
EmploymentDate : DateTime
BaseSalary : decimal
YearlySalaryGrowthPercentage = 0.02 : decim
getSalary()
The value =
base salary * (1+ YearlyPercentage (YearsSinceEmployment)
0..*
Association with attribute
ItemQuantityInOrder
Quantity : int
<Added to
0..*
0..1
Sequel to>
Limited Edition
LastPossibleObtainableDate : DateTime
Item {Abstract}
Title {Unique} : string
Price : decimal
ReleaseDate : DateTime
StockQuantity : int
MinPrice = 20 : decimal
AddItemToCart()
SearchItems()
0..1
Must be part of the same category to be a sequel
Reflex association
Standard Edition
PriceDropPerPeriod : decimal
NumberOfMonthsBeforePriceDrop : string
getPriceDropSinceRelease()
{Disjoint, Complete}
{Disjoint, Complete}
Value is caculated based on the ReleaseDate, the PriceDropPerPeriod and NumberOfMonthsBeforePrice Drop
Game
Genre : string
Platform [1..*] : List <string>
Comic
IssueNumber : decimal
PlotSummary : string
Figures
Size : decimal
Weight : decimal
Warranty : string
Franchise : string
1
1..*
1..*
1..*
Qualified association
<Developes
1
<Assists
<Works On
1..*
Aggregation association
<Creates
1
Composition association
Title : string
0..*
Game title is unique in our system
Developer
FoundingDate : DateTime
NumberOfEmployee : int
HeadquarterLocation : string
Author
Nationality : string
FirstReleasedBookDate : DateTime
Manufacturer
ProductionCapacity : int
{Disjoint, Complete}
Creator {Abstract}
Name : string

Design decision

Note for students :

Creating the design diagram is not the only important step when beginning to document your implementation. The next thing you are tasked with doing is describing your implementation choices for all the components on your diagram.

For now you only need to describe your implementation choices for the attributes and class extent persistence.

After assignment 6 (association implementation) you will write down your implementation choices for association

After assignment 7 (inheritance implementation) you will write down your implementation choices for inheritance

Extremely important note for students (ignore it and you will fail this assignment) :

Based on the requirements of the assignment, storing the information about your class diagram should be done through class extent presentency (stored locally) and not through the usage of a database.

THEREFORE, IT IS IMPORTANT TO NOTE : that your design decision should not reflect the usage of any sort of database solution (for example like entity frame work which is mentioned in the example).

Furthermore, it also is why the example below was purposely written with a database implementation in mind. The reason is that the below design description is only :

- **An example to give you an idea on how to structure your own design decision descriptions**
- **Does not cover all the elements that you should speak about in your solution (for assignment 4)**

So copying it 1-1 without a clear direction of changes based on your own implementation OR Mentioning implementation that are not reflected in your code (for example talking about how you implemented class extent presentency instead of using entity frame work) will be considered as an **automatic failure** for the entire final submission as it clearly states that your team lacks the understanding of what needs to be done here or on how to describe your implementation.

Your own design decision should denote the following :

- How you were able to implement all the required fields (implantation point 2A in the assignment requirements) in relation to the language you chose to program the project in.
- Your solution choice for class extent presentency (implementation point 3 in the assignment requirement) NOT your solution using a database of any kind (as it goes against the requirements of the assignments)
- How did you handle attribute validation (implementation point 2C in the assignment requirement)
- **Future requirements will be added here after assignment 6 and 7 on what to add to your design decision.**

Example on how to structure your design decision

The above design class diagram illustrates the changes necessary to model UML constructs that do not exist in the C# language, The final implementation will use entity framework core for persistence reasons, and the design choices related are also discussed in the upcoming section. The changes introduced are highlighted in green. The following section will describe the transformation process of certain UML constructs such as attributes types (based on the uml notations).

Attribute Validations

Where possible, the project has implemented attribute level validation using C# System.ComponentModel.DataAnnotations annotations. These annotations decrease the time taken to write and ensure an extra level of security for the user rather than implementing them manually. Furthermore, Entity Framework Core Uses these attributes, along with the fluent API entity configuration builders to determine what data types should be used on the database level.

Optional Attributes

Optional attributes are attributes that may contain no value. Optional attributes are simply C# properties that are marked as being nullable, which is to say they may contain null as a value. In Entity Framework Core these are attributes with a nullable column type. In the diagram they are showcase with the multiplicity range of [0..1]

Multi-value attributes

Whilst C# natively supports multi-value attributes as classes inheriting from ICollection, it is worth mentioning how this project aims to manage multi-value attributes in Entity Framework Core. Game's platforms attribute is a multi-value attribute that will be used to describe the possible platforms a game can be sold in. Instead of mapping each platform to a separate entity, this project has chosen to implement them using the compression and flattening of strings. As such each entity will have a column and attribute which is simply the compressed version of each platform in the platforms list.

Derived Attributes

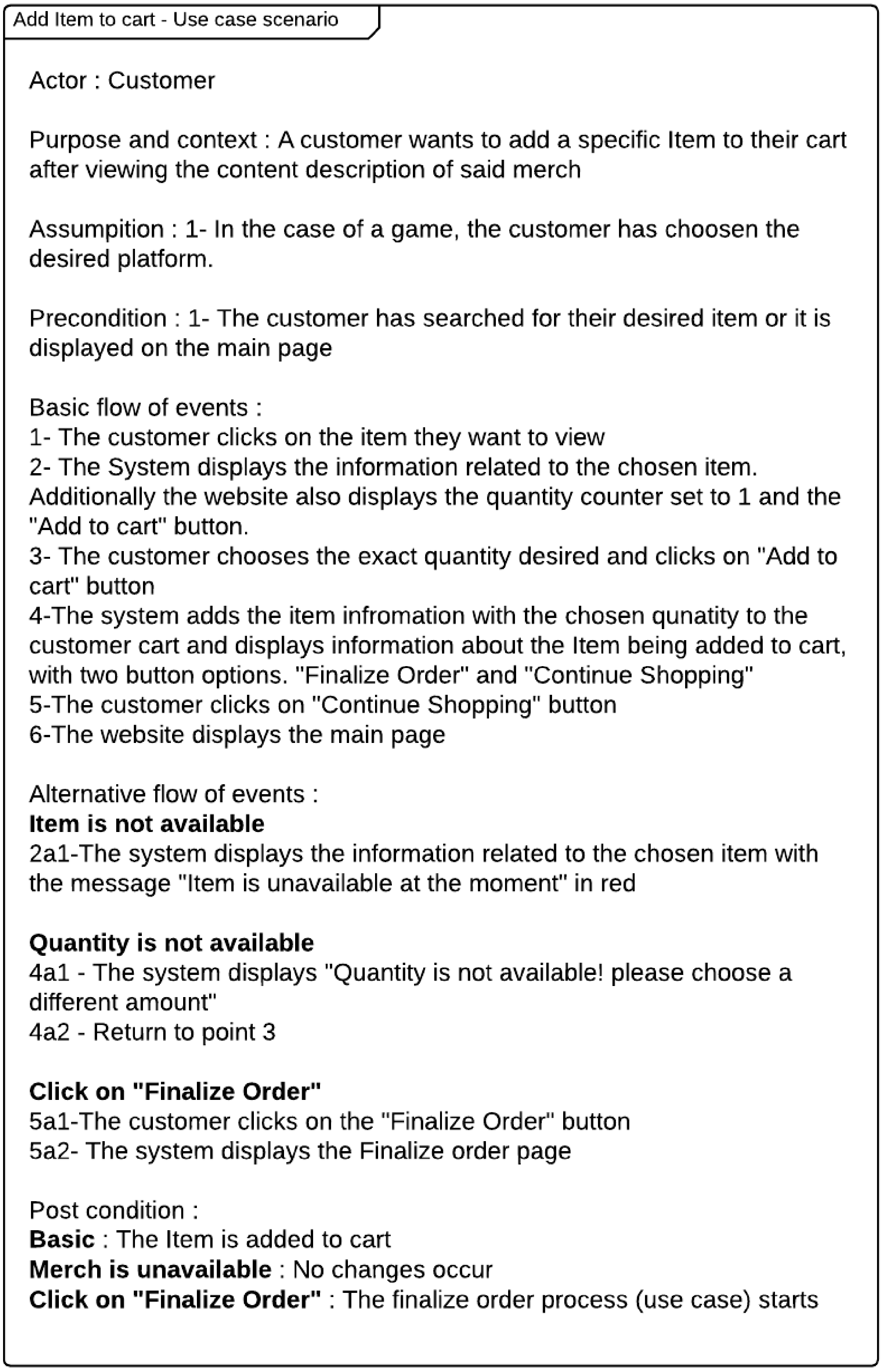
Derived attributes utilize that get feature when declaring attributes in c#. In most instances showcased, the derived attributes would have the logics directly created on the getter and would return the final total. Ensuring an automatic creation of the derived attributes based on the other values that create the base of a derived attribute.

For example the Salary attribute would be created as

Public decimal _salary { get => baseSalary * (1 + yearlyPercentage (YearsSinceEmploymeent) }

DO NOTE: that this doesn't cover everything, for example I don't mention how I implemented class attributes or tagged values. Make sure your design description covers everything required and implemented in your solution for assignment 4

Use case scenario: “Add item to cart”



Add Item to cart - Use case scenario

Actor : Customer

Purpose and context : A customer wants to add a specific Item to their cart after viewing the content description of said merch

Assumpition : 1- In the case of a game, the customer has choosen the desired platform.

Precondition : 1- The customer has searched for their desired item or it is displayed on the main page

Basic flow of events :
1- The customer clicks on the item they want to view
2- The System displays the information related to the chosen item. Additionally the website also displays the quantity counter set to 1 and the "Add to cart" button.
3- The customer chooses the exact quantity desired and clicks on "Add to cart" button
4-The system adds the item infromation with the chosen qunatity to the customer cart and displays information about the Item being added to cart, with two button options. "Finalize Order" and "Continue Shopping"
5-The customer clicks on "Continue Shopping" button
6-The website displays the main page

Alternative flow of events :
Item is not available
2a1-The system displays the information related to the chosen item with the message "Item is unavailable at the moment" in red

Quantity is not available
4a1 - The system displays "Quantity is not available! please choose a different amount"
4a2 - Return to point 3

Click on "Finalize Order"
5a1-The customer clicks on the "Finalize Order" button
5a2- The system displays the Finalize order page

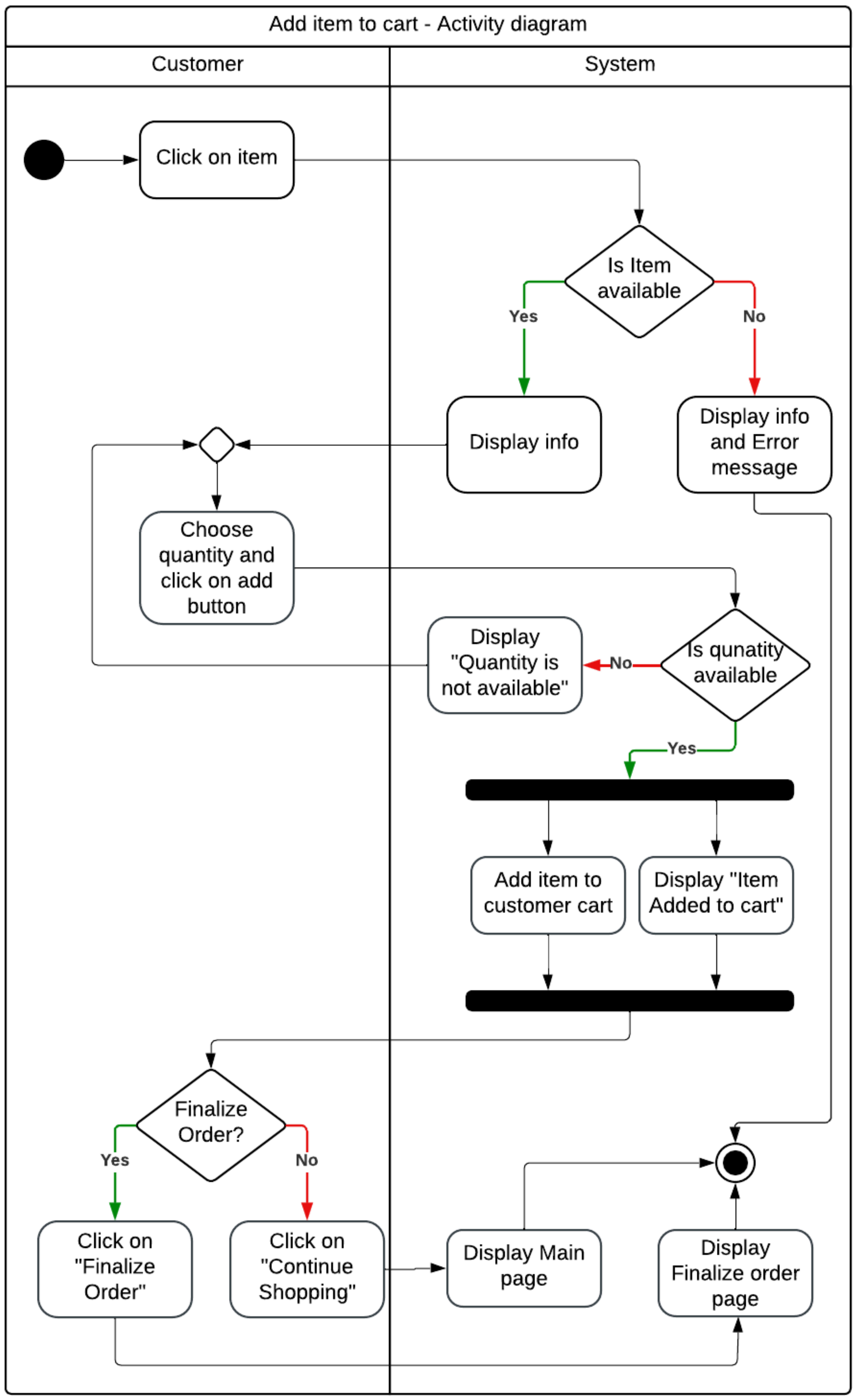
Post condition :
Basic : The Item is added to cart
Merch is unavailable : No changes occur
Click on "Finalize Order" : The finalize order process (use case) starts

Notes on what is a non trivial use case:

What constitutes a non-trivial use case :

- A use case that has a justified complexity (more than 4 steps and the possibility of alternatives)
- A use case that requires the utilization of more than one class (connected by an association)
- A use case that does one of the following:
    - Creates a new object reference between two or more classes (e.g. adding new items to an order, booking a new appointment for a customer, creating a new menu item for a menu)
    - Updates information of certain objects references between two classes or more classes (e.g Update booking for customer, modify schedule for subject, change subscription plan for user)
    - Deletes object references between classes (e.g Cancel booking, delete assignment for a given group, remove a certain train route from the schedule

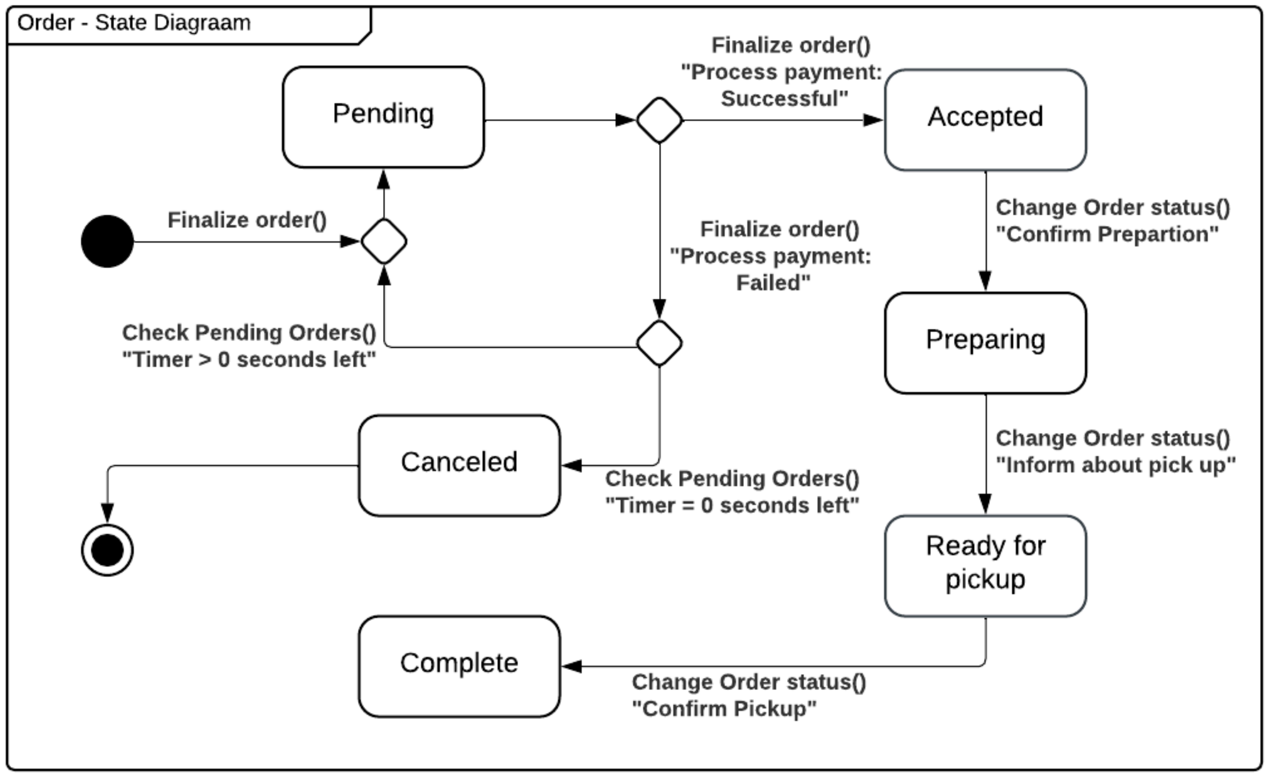
Activity Diagram: “Add item to cart”


Add item to cart - Activity diagram
Customer
System
Click on item
Is Item available
Yes
No
Display info
Display info and Error message
Choose quantity and click on add button
Display "Quantity is not available"
No
Is qunatity available
Yes
Add item to customer cart
Display "Item Added to cart"
Finalize Order?
Yes
No
Click on "Finalize Order"
Click on "Continue Shopping"
Display Main page
Display Finalize order page

Activity diagram - Description

Note for students: Briefly describe the activity diagram showcased above.

State Diagram - Order class


Order - State Diagraam
Pending
Finalize order()
Finalize order() "Process payment: Successful"
Accepted
Change Order status() "Confirm Prepartion"
Finalize order() "Process payment: Failed"
Check Pending Orders() "Timer > 0 seconds left"
Preparing
Canceled
Check Pending Orders() "Timer = 0 seconds left"
Change Order status() "Inform about pick up"
Ready for pickup
Complete
Change Order status() "Confirm Pickup"

State diagram - Description

Note for students: Briefly describe the State diagram showcased above.

Example :
The state diagram describes the state of Order Objects. When using the Finalized Order() method, a new Finalized order object is created with the status of “**Pending”**. Depending on how the payment is processed (Successful Or Failed) there are three possible states that the Object can be assigned to.

1. “**Accepted**” - In the case of a successful payment.
2. “**Pending”** - If the payment was unsuccessful and the timer that gets created (only in the case of a failed payment) is not equal to 0
3 “**Canceled**” If the payment was unsuccessful and the timer that gets created (only in the case of a failed payment) is equal to 0

The timer status is checked through the check pending orders () method

Once the object enters the accepted state, further modifications are done to it using the ChangeOrderStatus() method, which is used by the employees of the store. The order of the states is as follows :
1.” **Preparing**” Upon the selection of the “Confirm preparation” option
2.” **Ready for Pickup**” Upon the selection ofthe “Inform to pickup” option
3. “**Complete**” Upon the selection of the “Confirm pickup” option

The state of the object can end in two ways. Either as “**Complete**” or “**Canceled**”
If canceled, the object should be removed and the stock quantity should be modified to accommodate the previously ordered items

Dynamic analysis notes

Dynamic analysis describes any possible changes to your class diagram that are worth taking into consideration after creating the use case scenarios and state diagrams for your system (known as behavioral diagrams)

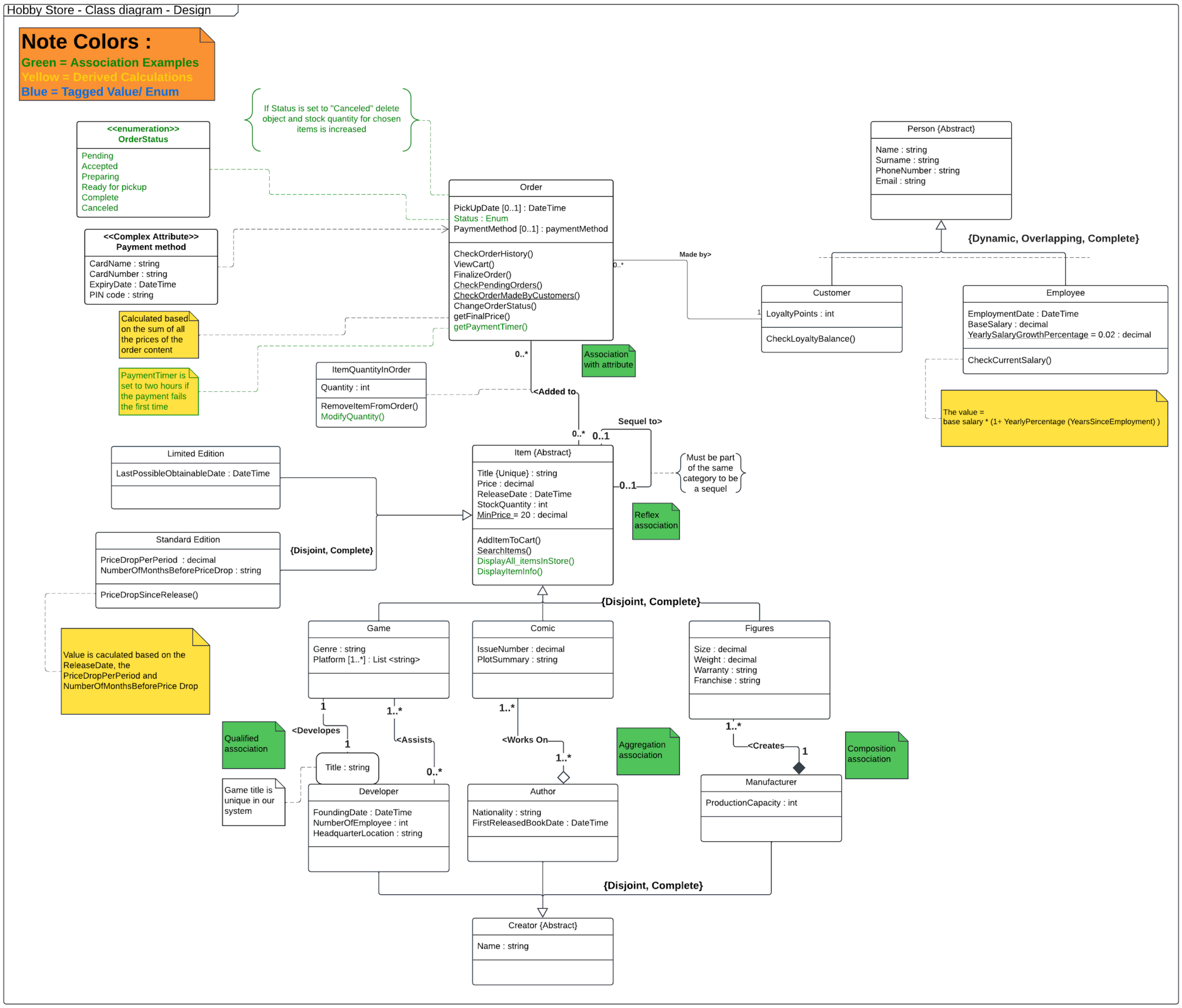
The assumption is that there can be new potential attributes, methods, constraints, notes, or even classes added after creating and analyzing the scenarios and state diagrams (A good dynamic analysis would require an introduction of some of if not all the elements mentioned above - if applicable)

Dynamic Analysis on “Add item to cart” use case & “Order class” state diagram

Example of an introduction : Following the above-presented diagrams, a dynamic analysis was performed. The result of this analysis indicated a need for further expansion on the design class diagram to encompass newly identified requirements and behaviors of the system. All introduced changes can be seen in the below presented final design class diagram.

//Here you should talk about all the new changes that should be introduced after dynamic analysis, with a clear justification behind why we would need to add that new element.

Changes to the Class diagram after dynamic analysis



Note for students :

1. All the elements added after the dynamic analysis have been highlighted in green.
2. The diagram above has been converted to a design class diagram. It must also be converted into your solution. Furthermore, here you have an example of how to showcase enumeration on your design diagram

Your enumeration examples should be shown on both design diagrams. The reason why it's only in my diagram after dynamic analysis is because I did not have any examples of tagged value (enumeration) before conducting a dynamic analysis (but that does not mean it will be the case for your solution)

Class diagram changes - Description

Briefly describe the new elements added to the diagram (the ones highlighted in green)