



APPLIED SOFTWARE ENGINEER

Coursework 1

Academic Year: 2023/24

Course: BSc Computer Science

Module: Applied Software Engineer

Assignment: Coursework 1

Module Leader:

Student: Nunzio Emanuele Sgroi

Student ID:

Case Study

Hill & Knowlton Looks for a New Knowledge Management System

Table of Contents

Table of Figures	1
Task 1: Use Case Model	2
Use Case Description	2
Use Case Diagram	3
Task 2: Class Diagram.....	4
Engineering Analysis Findings	5
Engineering Task Issues	5
Software Design Issues	6
Principles of Object-Orientation	6
Task 3: Sequence diagram	7
Task 4: Component Models	8
Component Model Description.....	8
Task 5: Object Constraint Language (OCL).....	9
Preconditions and Postconditions	9
Constraints	10
References	11

Table of Figures

Figure 1 - Use Case Diagram	3
Figure 2 - Class Diagram.....	4
Figure 3 - Sequence Diagram	7
Figure 4 - Component Diagram.....	8

Task 1: Use Case Model

The selected use case for this assessment is titled “**Purchasing Items in the hk.net Store Using Beenz Credits**”. In this scenario, the primary actor is an employee who engages in the process of acquiring items from the integrated store within the hk.net platform by utilizing Beenz Credits as the chosen method of payment.

Use Case Description

Use Case Name: Purchasing items in hk.net store using Beenz credits.

Actors: H&K Employee

Goal: The goal of this use case is to enable an H&K Employee to successfully purchase an item from the hk.net store using their Beenz credits.

Preconditions:

- The H&K Employee has Beenz Credits available in their account.

Postconditions:

- The H&K Employee successfully completes a purchase of an item from the hk.net store
- The payment is made using the Beenz Credits from their account.

Main Flow:

1. Employee browses items.
2. Employee adds an item to the cart.
3. System updates the cart.
4. Employee opens the cart.
5. Employee proceeds to check out.
6. System displays the provisional order details
7. User inputs shipping address.
8. User selects “Beenz” as payment method (specific amount of Beenz)
9. System informs employee that “Beenz” is being checked
10. <<include>> Check Beenz
11. System deducts the Beenz amount from the total.
12. Employee selects “Finish and Pay”
13. System updates Beenz Credits in employee account
14. System updates cart to “empty”
15. System confirms the transaction was successful and provide Employee with confirmation number.

Alternative:

4.1 Employee empties the cart.

5.1 Employee cancels transaction.

11.1 If the employee inputs a Beenz amount greater than the one available in their account, system informs user that Beenz credits available are not enough.

Use Case Diagram

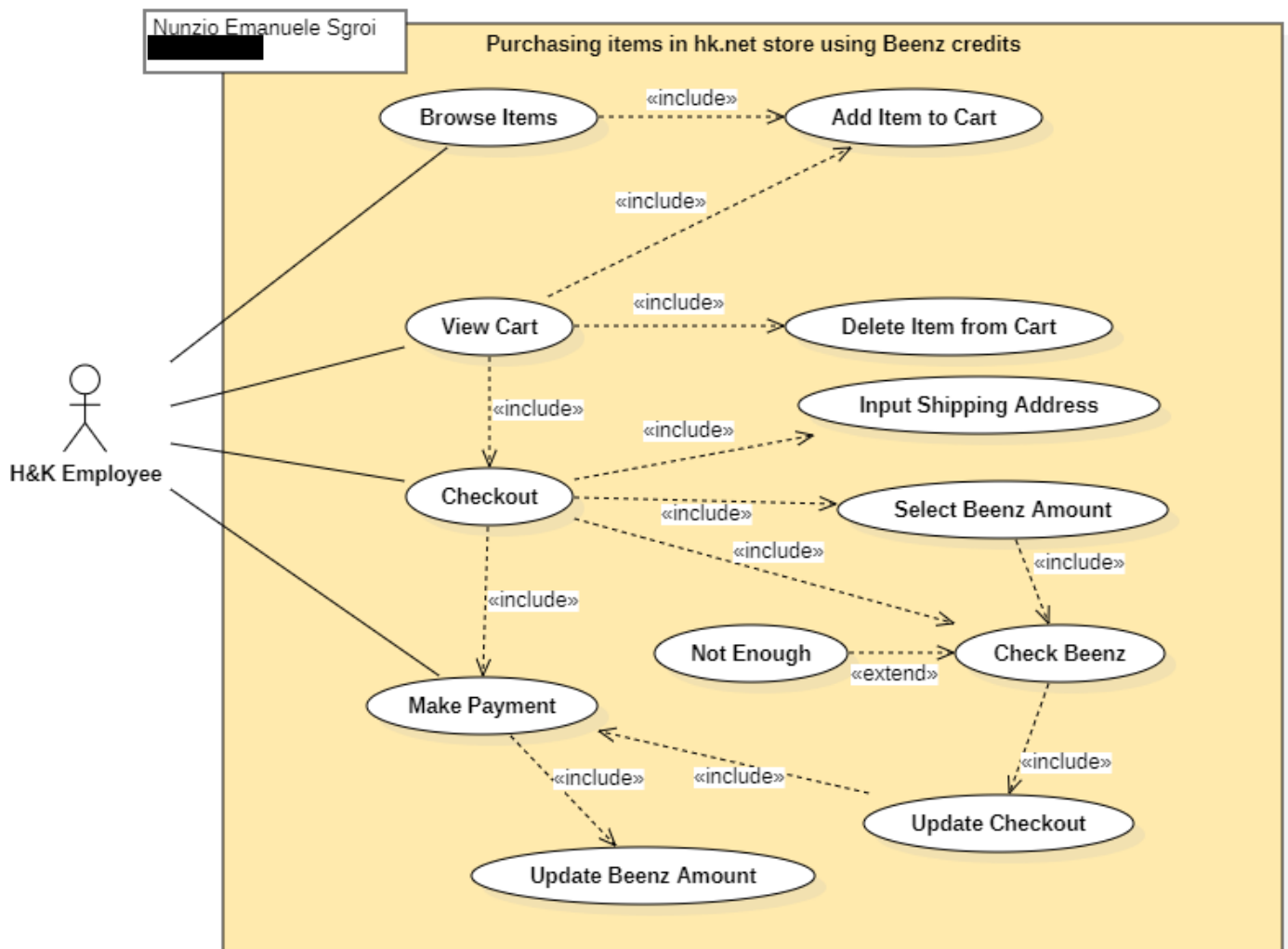
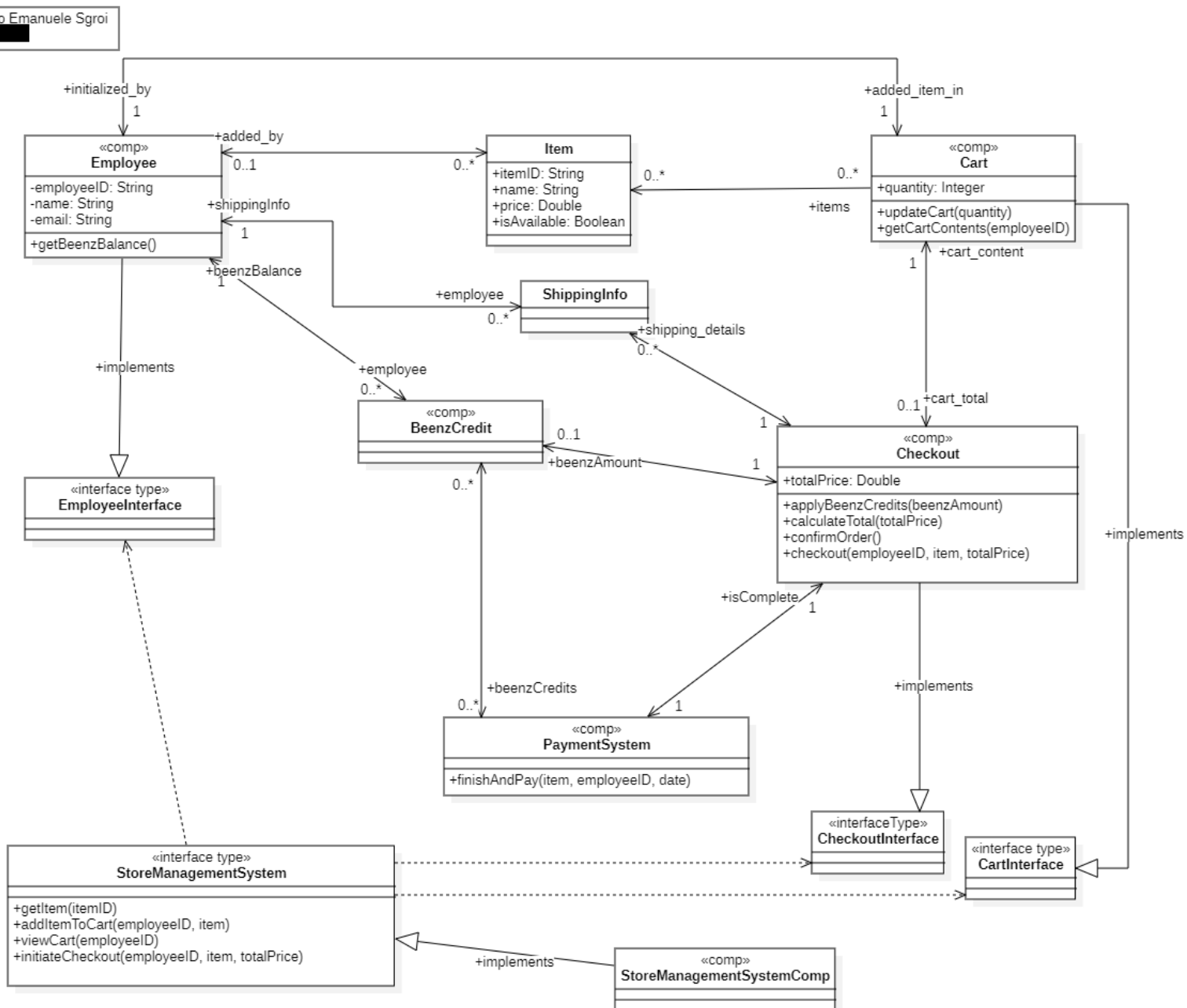


Figure 1 - Use Case Diagram

Task 2: Class Diagram



To encourage employees to use the Extranet to find out information, H&K has embedded a form of micro-payment called "beenz" throughout the site. Every time an individual opens a document or contributes information, that person stands a chance of collecting beenz, which can be redeemed for books, CDs, and even vacations.

Figure 2 - Class Diagram

Engineering Analysis Findings

Design Decision:

Classes and relationships in the class diagram have been chosen to reflect the domain model of the hk.net store closely:

- "StoreManagementSystem" acts as the central system class, managing key operations like checkout and cart item management, ensuring that the system oversees these functions rather than individual classes like "Cart" and "Checkout".
- Connections between "Employee" and "BeenzCredit" illustrate individual credit balances, while its link to "Cart" reflects an employee's personal shopping actions.
- "Item" is linked to both "Employee" and "Cart", indicating that employees are responsible for adding items to their carts.
- "Checkout" interacts with "BeenzCredit", "PaymentSystem", "Cart", and "ShippingInfo" for a complete checkout process, such as validating Beenz credit availability, processing payment details, and handling shipping information. Associating "Checkout" with "PaymentSystem" enhances design and facilitates potential third-party payment processing. Furthermore, "BeenzCredit" and "ShippingInfo" link to "Employee", indicating direct interaction with the checkout process.
- Interfaces are employed to provide multiple interaction points for users within the system, although actual operations like adding to cart and checkout are centralized in the "StoreManagementSystem".
- Components such as "Checkout", "PaymentSystem" and "BeenzCredit" share the same interface type: "CheckoutInterface"

Assumptions:

- To simplify the use case scenario, it is assumed that the employee can purchase any item, rather than just CDs, books and vacations as mentioned in the case study.
- It is assumed that the system is using a third-party payment process
- It is assumed that every employee can have Beenz in their account, but this can be equal to 0, which can be translated as "unavailable".

Engineering Task Issues

- **Concurrency:** To handle multiple employees purchasing items simultaneously, the system should implement transaction management and locking mechanisms to ensure data integrity. Optimistic or pessimistic locking can be used depending on the expected system load and transaction volume (Chandrakant, 2022).
- **Security:** Beenz Credits available in employee accounts should be protected with encryption
- **Scalability:** The purchasing system should be designed to accommodate an increasing number of transactions, potentially through scalable cloud services (Filip Dimitrievski et al., 2015).

Software Design Issues

- **Data Integrity:** Ensuring that each transaction is processed in a way that accurately reflects the Beenz credit deduction and inventory status is essential. The use of transaction management systems that support atomic operations can help maintain data integrity (Chandrakant, 2022).
- **Session Management:** The portal must reliably manage user sessions, keeping track of individual shopping carts and Beenz balances throughout the shopping experience. This involves implementing state management strategies that are resilient to concurrency (Raj, 2023).
- **Payment System Integration:** Despite being part of hk.net, it's imperative to follow recognized e-commerce security standards, especially for the payment processing component. This includes employing Secure Socket Layer (SSL) for data encryption, enforcing Strong Customer Authentication (SCA) measures, and utilizing reliable third-party vendors for enhanced security (Francom, 2023).

Principles of Object-Orientation

- **Inheritance:** The design of the class diagram does not implement inheritance. The classes are distinct and fulfil unique roles within the system through associations, which define how they interact with each other.
- **Encapsulation:** In the class diagram, encapsulation is evident in the "Employee" class, by making attributes like id, name, and email private. Access to an Employee's data is accessed through interactions with methods in classes such as "StoreManagementSystem", which utilizes "employeeID" for operations like checkout.
- **Polymorphism:** The system design demonstrates polymorphism through the "CheckoutInterface", which is implemented by the "Checkout", "PaymentSystem", and "BeenzCredit" components. This enables different components to be utilized interchangeably when performing checkout operations, allowing the system to invoke the appropriate methods regardless of the specific component being used.
- **Abstraction:** The design employs abstraction by using the CheckoutInterface to define key actions for the checkout process. Components like "Checkout", "PaymentSystem", and "BeenzCredit" implement this interface, abstracting away the complexities of the checkout operation. This approach allows each component to handle specific aspects of the checkout process, such as item selection, payment processing, and Beenz credit management, in a consistent and interchangeable manner across the system.

Task 3: Sequence diagram

The following Sequence Diagram was produced for the use case “**Purchasing Items in the hk.net Store Using Beenz Credits**”. It includes guards, denoted between square brackets (i.e. [Item.isAvailable = true] : selects item), and iterations, represented with a loop frame accompanied by a condition. All message operations are also included.

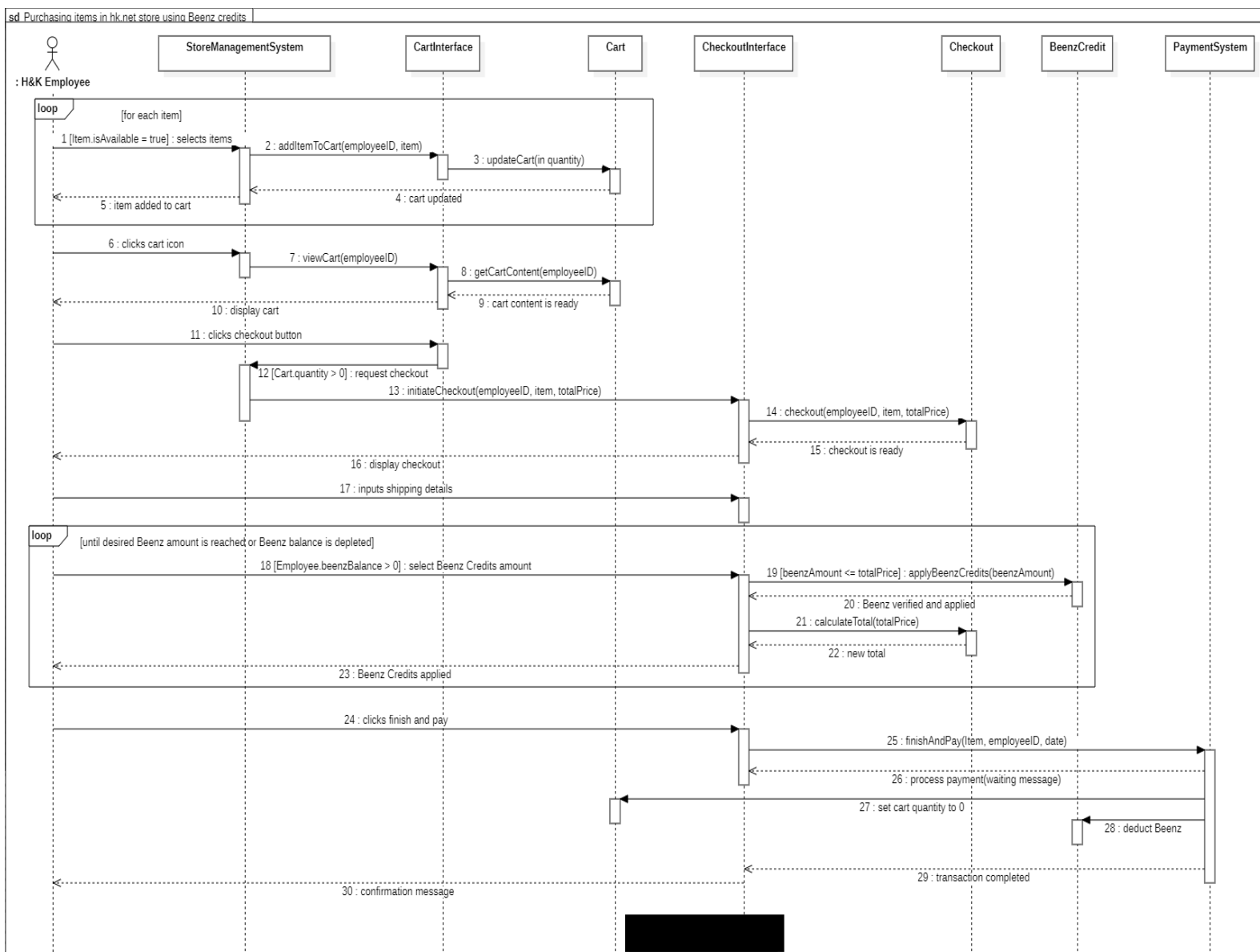


Figure 3 - Sequence Diagram

Task 4: Component Models

The Component Diagram created for this task translates the Class Diagram from Task 2 into a broader view of the system's structure. It organizes the system into clear, manageable sections, each with its own set of tasks. This overview shows how the different sections work together to make item selection, cart management, Beenz credit application, and payment possible. Each section of the system, marked by interfaces, lays out a clear set of actions that link the sections, promoting an organised and flexible design.

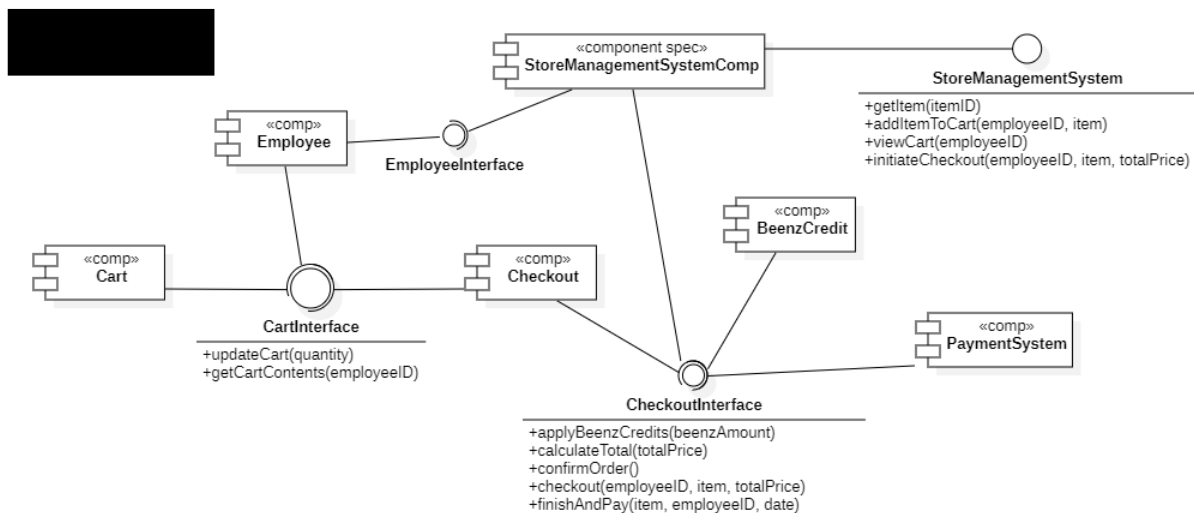


Figure 4 - Component Diagram

Component Model Description

The design chosen closely reflects the logic presented in the Class Diagram, where the "StoreManagementSystem" component is pivotal, managing essential tasks such as item addition and checkout initialization. The layout of interfaces highlights how the user, an H&K Employee, engages with the system via different touchpoints, each linked to various components that handle specific functions within the system. Additionally, the Component Diagram demonstrates the multipurpose nature of interfaces, connecting to more than one component, indicating that these components are governed by a common set of interface rules.

The "CheckoutInterface," for example, is connected to the primary "StoreManagementSystem" because it plays a crucial role in starting the checkout process. Linking the "Checkout" component to both the "CartInterfaces" and "CheckoutInterface" aims to indicate that the "Checkout" component's reliance on the "Cart" component, similar to how a user would move from viewing their cart to proceeding to checkout.

The design shown in the Component Diagram makes sure the system works well, is strong, and can easily handle changes. Its modular nature means parts of the system can be changed or updated without affecting the whole system. The design also allows for the system to grow over time. Moreover, the use of common interfaces across multiple components simplifies integration and maintenance processes, ensuring long-term system reliability.

Task 5: Object Constraint Language (OCL)

This task employs the Object Constraint Language (OCL) to define and apply the rules and constraints critical to the “**Purchasing Items in the hk.net Store Using Beenz Credits**” use case. OCL is used here to ensure the system operates correctly and maintains data integrity throughout the purchasing process.

Preconditions and Postconditions

Operation: Employee make a purchase using Beenz

- **Precondition:** The employee must have a Beenz balance greater than or equal to the amount of Beenz they wish to spend, and the amount being spent should be greater than zero.
- **Postcondition:** After the purchase, the employee's Beenz balance is reduced by the amount of Beenz spent.

context Checkout::applyBeenzCredits(beenzAmount: Double)

pre: self.cart.employee.beenzBalance >= beenzAmount and beenzAmount > 0

post: self.cart.employee.beenzBalance = self.cart.employee.beenzBalance@pre - beenzAmount

Description: This OCL statement ensures that employees can only use the Beenz Credits if the Beenz Balance is greater than 0. After using the Beenz Credits to make a purchase, the amount Beenz balance is recalculated.

Operation: Add item to cart

- **Precondition:** The item must be available.
- **Postcondition:** The item is added to the cart.

context StoreManagementSystem::addItemToCart(employeeID: String, item: Item)

pre: item.isAvailable

post: employee.cart.items->includes(item)

Description: In this OCL statement, the precondition checks that the Item instance passed to the “addItemToCart” operation is available, while the postcondition ensures that after the operation, the Item is included in the items collection of the Cart instance.

Operation: Apply Beenz credits and adjust total price

- **Precondition:** The amount of Beenz Credits to be applied must not exceed the employee's current Beenz balance and must be a positive value.
- **Postcondition:** The checkout's total price is updated to reflect the deduction of the applied Beenz Credits from the initial total price.

context Checkout::applyBeenzCredits(beenzAmount: Double)

pre: `beenzAmount <= self.cart.employee.beenzBalance and beenzAmount > 0`

post: `self.totalPrice = self.totalPrice@pre - beenzAmount`

Description: This OCL statement ensures that the amount of Beenz Credits an employee wishes to use is available in their account and is a positive number. Upon applying these credits, the total price in the checkout is reduced accordingly to reflect the new payable amount.

Operation: Complete Payment and Empty Cart.

- **Precondition:** Not applicable
- **Postcondition:** The payment is marked as complete, and the cart associated with the checkout is emptied of all items.

context `PaymentSystem::finishAndPay(item: Item, employeeID: String, date: Date)`

post: `self.checkout.isComplete = true`

post: `self.checkout.cart.items->forAll(i | i.quantity = 0)`

Description: The “finishAndPay” operation marks the checkout associated with the payment system as complete and sets the quantity of items in the cart to zero, effectively emptying the cart.

Constraints

Employee Beenz balance invariant

- Every employee's account balance of Beenz Credits must always be non-negative.

context `Employee`

invariant: `self.beenzBalance >= 0`

Cart items integrity invariant

- A cart should always reference the items it contains, ensuring there are no orphaned items not linked to a cart.

context `Cart`

invariant: `self.items->forAll(i | i.cart = self)`

Checkout completion invariant

- If a checkout's payment process is marked as complete, then certain conditions must be satisfied, such as payment confirmation received, and the cart must be empty

context `Checkout`

invariant: `self.isComplete implies (self.paymentConfirmed and self.cart.isEmpty())`

References

- Chandrakant, K. (2022). *Design Principles and Patterns for Highly Concurrent Applications*. Retrieved from baeldung: <https://www.baeldung.com/concurrency-principles-patterns>
- Filip Dimitrievski et al. (2015). Scalable architecture of e-ordering system in cloud. *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1523-1528. Retrieved from <https://api.semanticscholar.org/CorpusID:16971769>
- Francom, S. R. (2023). *Online Payment Security: Top 6 Safety Practices*. Retrieved from business.org: <https://www.business.org/finance/payment-processing/online-payment-security-5-steps-to-ensure-safe-transactions/#:~:text=1,standards%20for%20electronic%20payment%20processing>
- Raj, P. (2023). *Concurrent Model in Software Engineering: A Powerful Model for Enhancing Efficiency and Performance*. Retrieved from Heavy Coding: <https://heavycoding.com/concurrent-model-in-software-engineering/#:~:text=The%20concurrent%20model%20in%20software,the%20use%20of%20parallel%20processing>