Team 26 - Project 2

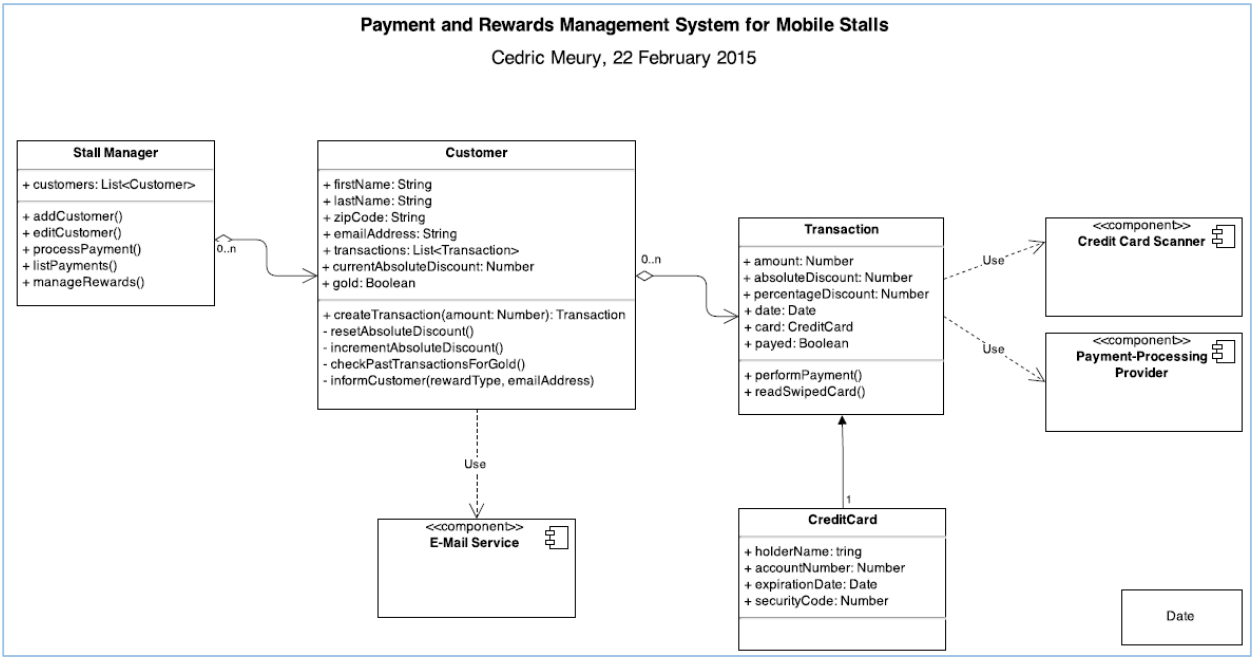
Design Discussions

Individual Designs

1. Design-1

1.1 Cedric Meury (cmeury3)

1.2 Class Diagram



1.3 Pros

- Hides complexities regarding the E-Mail Service, Credit Card Scanner and Payment-Processing Provider correctly as per the requirements. It's a simple design
- Captures entities (nouns), attributes (adjectives) and relations (verbs) correctly

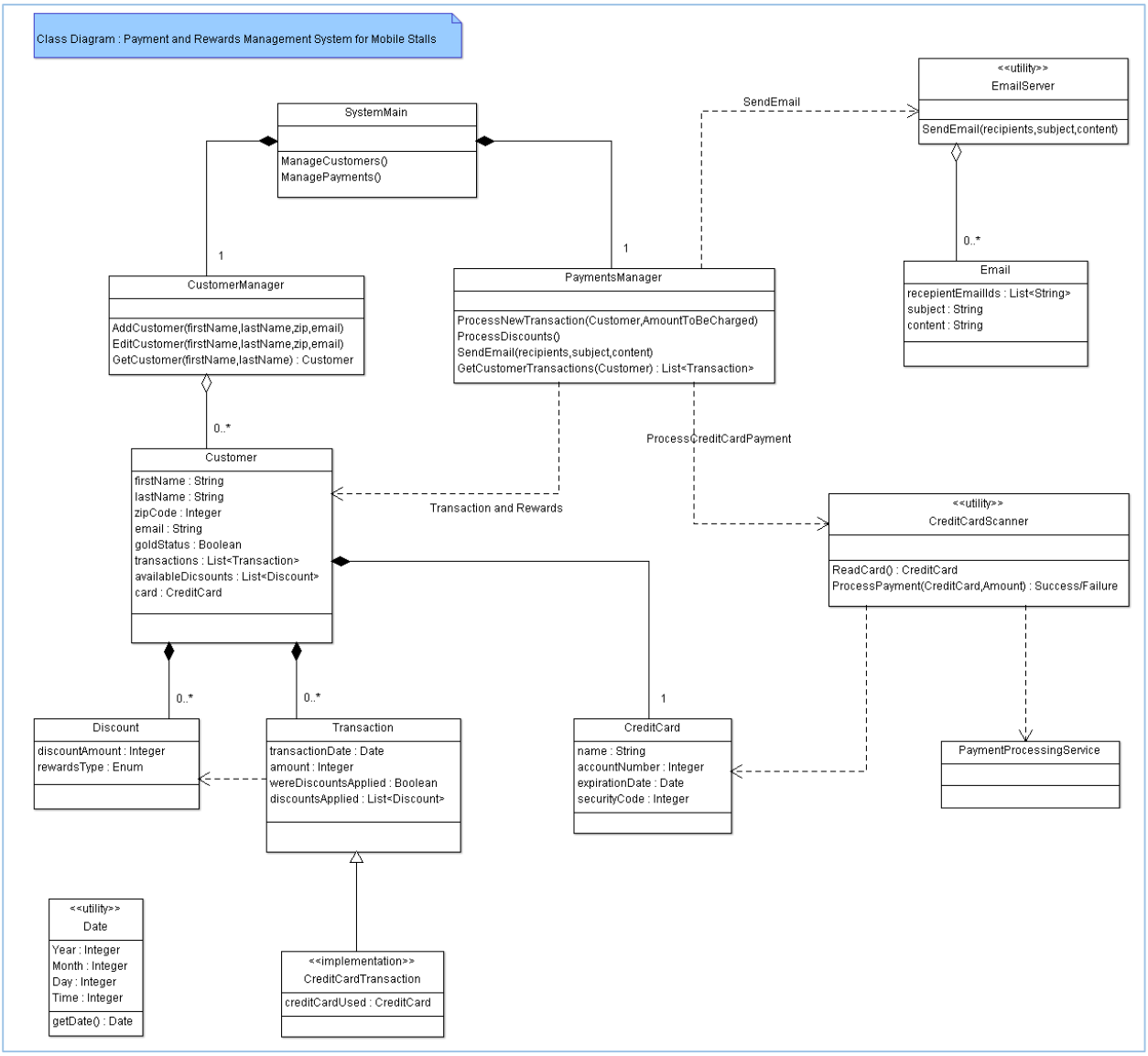
1.4 Cons

- `CreditCard` should be aggregated within `Transaction` (show aggregation line)
- Separate classes for Discounts and Rewards would support future expansions and reduce complexity
- The data type of the `holderName` attribute in the CreditCard has a typo, it should be `String`

2. Design-2

2.1 Ganesh Shivashankaraiah (ganesh30)

2.2 Class Diagram



2.3 Pros

- Having CustomerManager and PaymentsManager results in separation of concern
- Having a separate EmailServer utility class

2.4 Cons

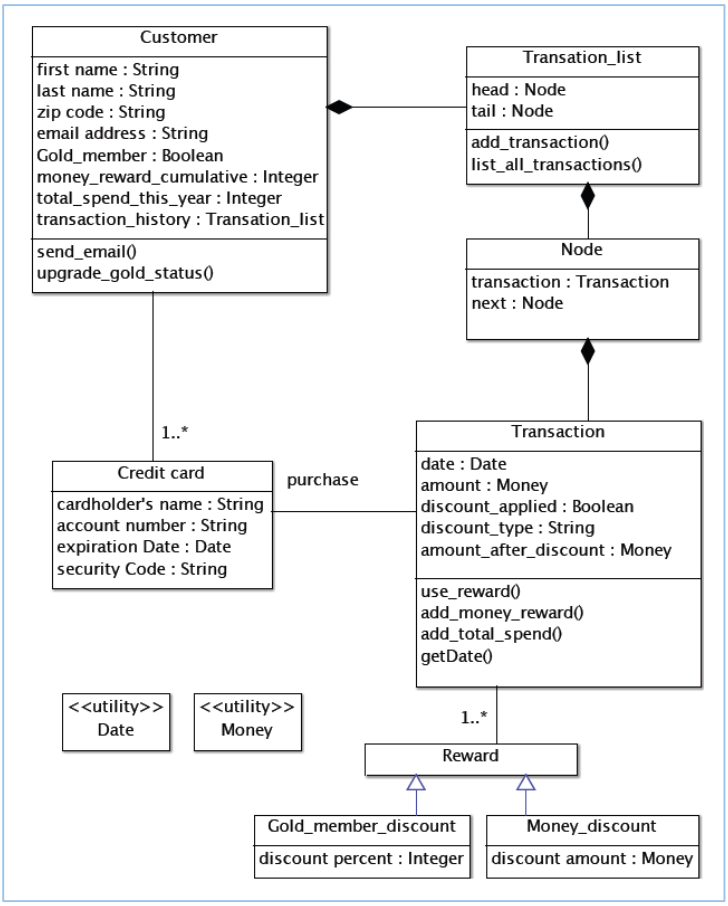
- Discount can be split to have a base class with 2 child classes for each type of discount

- (Transaction – Amount) and (Discount – discountAmount) attributes can be double/Number
- CreditCardScanner and PaymentProcessingService should be components
- Remove `Email` class as it encapsulates email functionality
- (Customer - Discount) multiplicity should be 0...2 (only 2 types of discounts - money, 5%)

3. Design-3

3.1 Yue Li (yli801)

3.2 Class Diagram



3.3 Pros

- Having a separate utility class for Money. Can handle all complexities within this class
- Having a base class for `Reward`. This helps in future expansion to add more type of rewards
- Having Rewards within the transaction

3.4 Cons

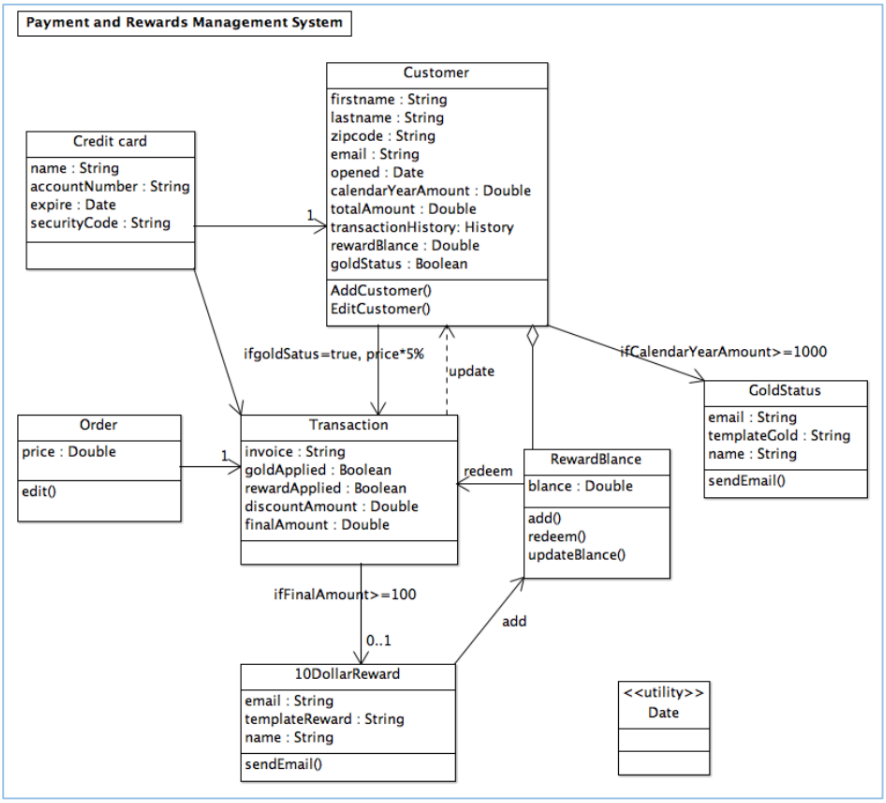
- TransactionList, Node and Transaction have been implemented like a linked list. This can be a simple relation using just a `list` object and TransactionList, Node can be removed

- (Customer – Credit Card) relation should be a composition. Can be a temporary object for simplicity
- Does not capture `Email` related attributes
- Also need a main application class that ties all the entities

4. Design-4

4.1 Yuchun Qin (yqin47)

4.2 Class Diagram



4.3 Pros

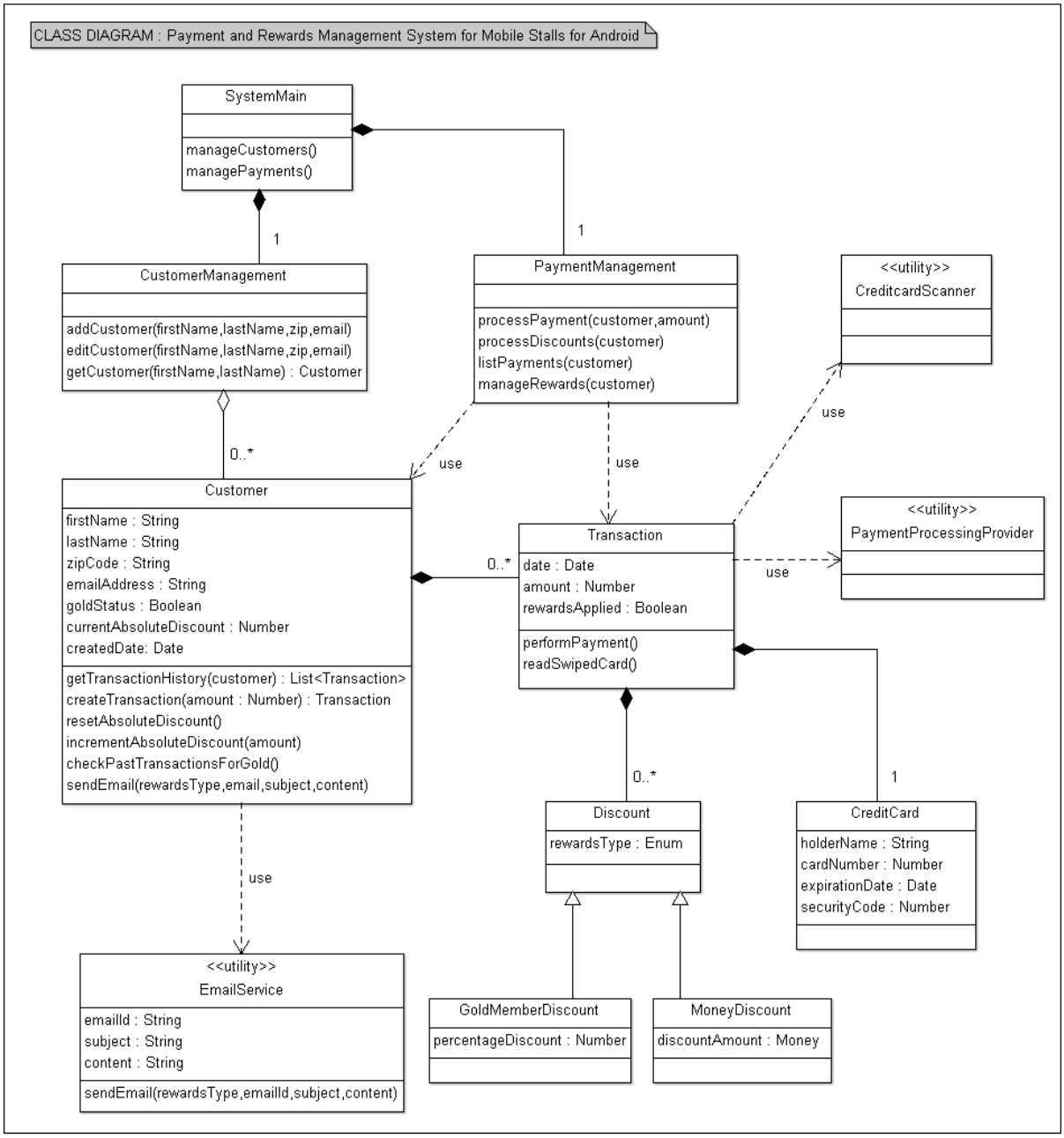
- The entities `Credit Card`, `Customer` and their attributes are captured correctly
- The (Customer – RewardBalance) relation is correct

4.4 Cons

- Implementation details such as the `if` condition on connections should not be shown in the diagram.
- There should not be two relations between `Customer` and `Transaction`
- `Redeem` and `Add` can be operations in the entities rather than UML relation. And `10DollarReward` can be a subclass of `RewardBalance`
- `Edit()` operation in `Order` may not be required. The `Order` class can be removed
- Can show the components – CreditCardScanner and PaymentProcessingService

Team Design

4.5 Class Diagram



4.6 Main Commonalities and Differences

This section discusses the main commonalities and differences between this design and the individual ones, and justifies the main design decisions

- Do we need classes such as `Stall Manager` and `System Main` as in the individual class diagrams?
    - We will definitely need a main class that ties all the child classes and has the overarching functionality. Maybe we can add Customer and Payment processor helper classes for having separation of concerns and to reduce complexity.
- It was decided to not show the Utility classes which are part of the standard Java/Android libraries for clarity. (Ex: Money, Date, Time etc…)
- Email functionality is embedded in a utility class.
- All team members created `Customer` and `Transaction` classes, and used similar utility classes.
- Ganesh used a Main interface and two Manager classes that tie everything together. The team design uses his idea and changed their name to "Customer Management" and "Payment Management".
- The "Credit Card" class is used in all member's designs, but shows different relations. The team design connected "Credit Card" with "Transaction", because there was no specific requirement to store credit cards per customer and we assumed we just store the card separately for every single transaction.
- Cedric used a component "Email Service" in his design. The team design used this part and specified the email sending conditions.
- The team design employed Yue's design to organize two type of discounts ("Gold status" and "Rewards")

4.7 Summary

This section summarizes the lessons learnt in the process of discussing the designs, in terms of design and team work.

- Understood the differences between aggregation and composition and this is depicted in the team design.
- Elimination of classes available in the standard library.
- The implementation level attributes are not shown in the design diagrams. Ex – Lists etc.
- The design is simplified to adhere to the actual requirements. We tried to be cautious about reading too much between the lines to not add completely new features that are not desired.
- We need focus on the structure of the system, rather than the implement details
- For teamwork, it was difficult to find a common design; but we initially set a design and the rest of the team members gave feedback on the common design