Hotel-project
Week 4: Design 3

This weeks topics are: *Component diagrams*, *Class diagrams* and *Sequence diagrams*.

Assignment

This is your last assignment where you create new models. From next week on, you will implement your models using Java and will only update the existing models. Before that, you will do three more types of diagrams: component diagrams, class diagrams, and sequence diagrams. Besides these, you shall also review the work of another group. The instructions for the halftime review can be found in another PDF in the project part of the course homepage.

Class Diagram

Draw a class diagram that supports your complete use cases from last week's assignment. Use the domain model as a starting point for the class diagram and add classes with operations and attributes which you need in order to realise your use cases. Remember that the domain model reflects a conceptual perspective of the domain while class diagrams represent a software development perspective. Draw the class diagram using Papyrus and hand it in as an image export from Papyrus.

Iterate between the class diagram, the component diagram and the sequence diagrams.

Component Diagram

Draw a component diagram of your system and its surroundings.

Your booking system shall be placed in at least one component. You are free to divide it into further components if you think this is reasonable! Use your actors (assignment 2) and your system operations (assignment 3) in order to define interfaces for you component(s). Visualise your actors in the component diagram by adding one component per actor and connecting them to your system's interfaces.

Banking Component

Your customer has decided that you shall use a commercial banking service during the every booking. For a booking to be successful, a valid credit card has to be specified. In order to verify the validity of the credit card, you are required to use a banking service offered by a third party. Additionally, you shall handle the payment of the booking through this service. Whether you do this during the booking or at later stages, such as check-in or check-out, is up to you. Update your system description (assignments 1 through 3) to accomodate this requirement! In the component diagram, include the banking component. Your booking system shall not use the administration interface of the banking interface. However, the components you are adding for your actors can use this interface. The component is described in more detail in appendix A.

The papyrus model of the banking component can be downloaded on the course homepage. So you can simply copy the contents from the provided component diagram into your component diagram!

Sequence diagrams

Construct a sequence diagram for each complete use case that you wrote last week. Use the system operations which you have defined in assignment 3. The sequence diagrams should depict a interaction between the use case actor and your system component(s). This means that they have to follow the component interface! Also, do not forget to include the interaction with the banking component, where necessary!

The implementation of the search for available rooms does not need to be sophisticated. For passing the course it is enough to return a room with the right amount of beds or deny the booking if there are no available rooms. Implementations returning a sufficient amount of rooms to accomodate the number of people in the booking is of course more satisfying.

Checklist

- A Component Diagram, modelled in Papyrus, containing your system's component(s), components visualising your actors, and the banking component.
- A Class Diagram, modelled in Papyrus.
- A Sequence Diagram, modelled in Papyrus, for each complete Use Case from assignment 3.
- Halftime review, see separate document.

Literature

- Craig Larman, *"Applying UML and Patterns"*, chapters 15 and 16.

A Banking Component

The banking service which we will use in this course has two interfaces - an interface for customer access and an interface for administration access. The administration interface will only be used for testing purposes and should not be accessed by your hotel system. You will however use this interface in your testing code during the implementation phase. The administration interface and implementation details for the component will be provided together with next week's assignment!

Figure 1 depicts the component diagram of the banking component. The

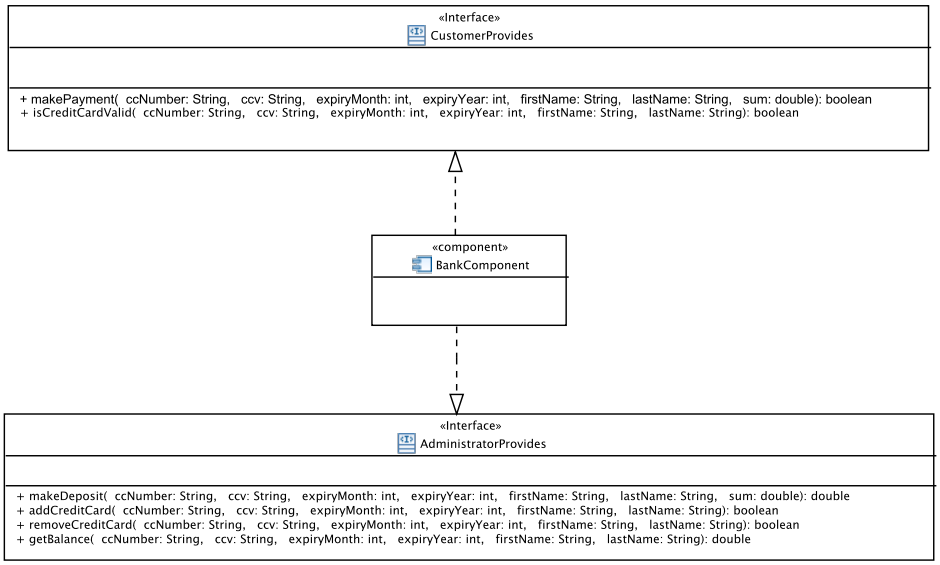

Figure 1: Component Diagram: Banking Component

customer interface *CustomerProvides* provides a method to check whether a credit card is valid (*isCreditCardValid*) and a method to actually process a payment (*makePayment*). They are defined as described in the following subsections.

A.1 isCreditCardValid

Signature: *boolean: isCreditCardValid(ccNumber:String, ccv:String, expiryMonth:int, expiryYear:int, firstName:String, lastName:String)*
Description: Given a credit card number, a ccv (checksum) number, the

expiry date (month and year) of the credit card and the full name of the credit card owner, this operation returns true if the information corresponds to a valid credit card and false otherwise. Valid means that the credit card is known to the bank and not expired. The operation also returns false if any erroneous information is provided (e.g. null parameters, invalid month/year values).
Preconditions: None
Postcondition: Returns *true* if (*ccNumber* is not null and *ccv* is not null and *expiryMonth* is between 1 and 12 and *expiryYear* is between 14 and 20 and *firstName* is not null and *lastName* is not null and a credit card with the specified parameters exists in the system). Else, returns false.
Sequence Diagram: The sequence diagram for this operation is depicted in Figure 2.

A.2 makePayment

Signature: *boolean: makePayment(ccNumber:String, ccv:String, expiryMonth:int, expiryYear:int, firstName:String, lastName:String, sum:double)*
Description: Given a credit card number, a ccv (checksum) number, the expiry date (month and year) of the credit card, the full name of the credit card owner, and a non-negative amount, this operation subtracts the amount from the given credit card account given that the credit card is valid (see *isCreditCardValid* operation) and the balance is larger or equal than the specified amount. It returns true if the amount was successfully subtracted, false otherwise.
Preconditions: None
Postcondition:(Returns *true* and subtracts *sum* from the credit card's balance) if (credit card is valid and the balance for this credit card is larger or equal than *sum* and *sum* is larger than 0.0). Else, returns false.
Sequence Diagram: The sequence diagram for this operation is depicted in Figure 2.

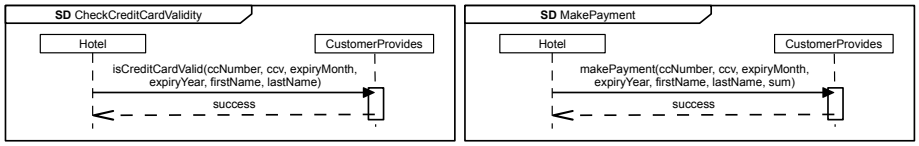

Figure 2: Sequence Diagrams for isCreditCardValid (left) and makePayment (right).