UNIVERSITÉ Concordia UNIVERSITY

AUTOMATIC TICKETING MACHINE

DELIVERABLE 3

by

AKINRINADE, Oyeyemi Akintoyese (ID: 27017766)
MAKKAR, Gagandeep Kaur (ID: 26613748)
Gagandeep Kaur (ID: 27683731)
BANIK, Biswajit (ID: 26831419)
LIN, Xuefei (ID: 27312377)

A
submission in partial fulfillment
of the requirements of
COEN 6312

February 29, 2016

Table of Contents

1. Class Diagram

Class diagram is a static structure diagram that describes the structure of a system by showing the classes of the system, their interrelationships (including inheritance, aggregation, and association) and the operations (or methods) and attributes of the classes.

It is the fundamental (or foundational) building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application and for detailed modelling translating the models into programming code. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

A class is represented with a box which has three (3) parts:

- The top part contains the name of the class. It is printed in Bold, centered and the first letter capitalized.
- The middle part contains the attributes of the class. They are left aligned and the first letter is lower case.
- The bottom part gives the methods or operations the class can take or undertake. They are also left aligned and the first letter is lower case.

UML Class Diagram for an Automatic Ticketing Machine is given in Figure 1 on the next page.

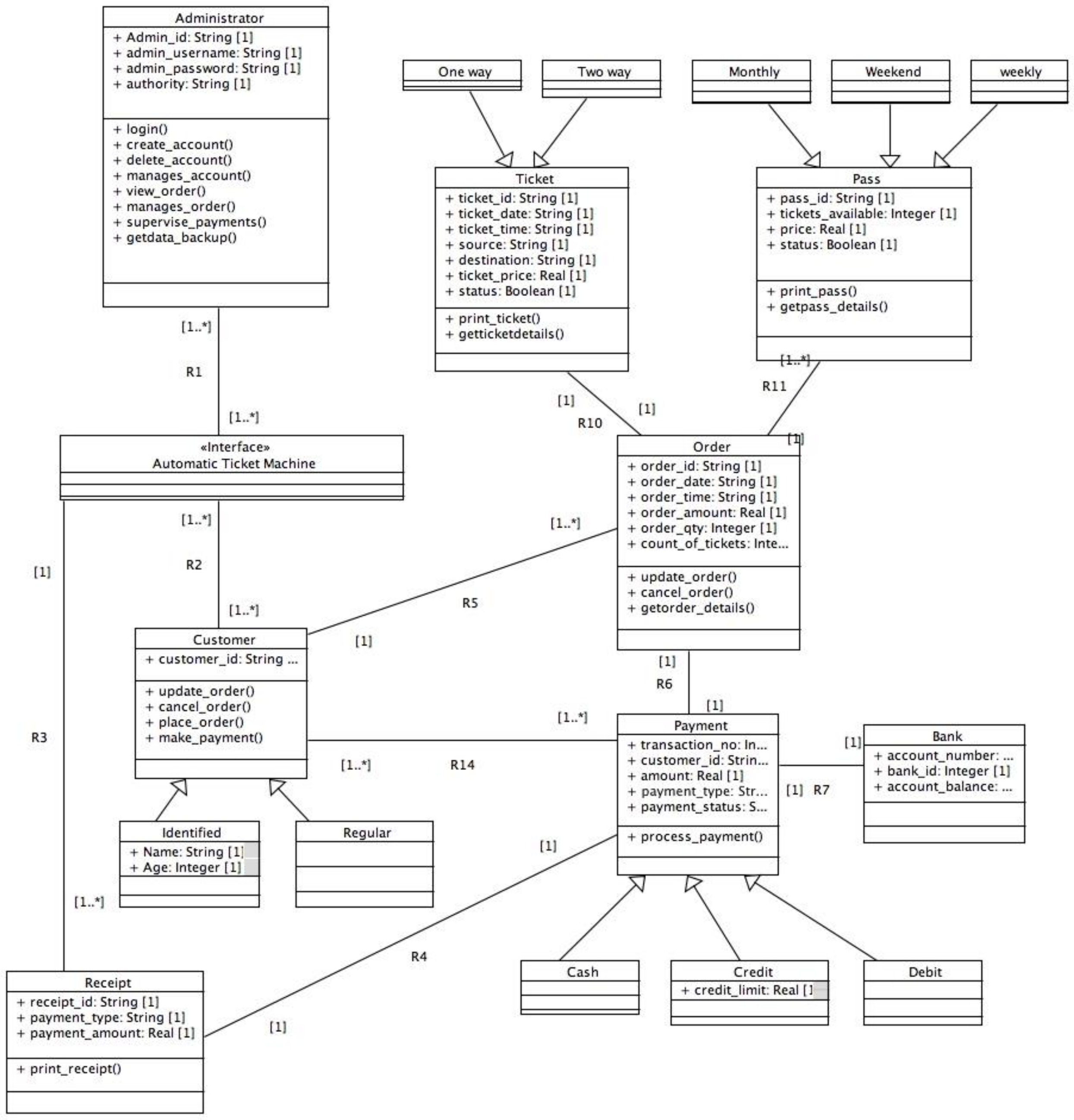

Figure 1: Class Diagram for Automatic Ticketing Machine.

1. Order

The customer will place an order of ticket(s) and the order contains all the information regarding the order. This information pertains to count of tickets or the tickets in the pass, order date and time and order_qty. The attributes and methods involved in this class are as follows:

Attributes:

- **order_id:** The order id is required for the customer and the administrator to track the status of the order.
- **order_date:** The order date is useful to keep track of the date on which order was placed.
- **order_time:** The order time is useful to keep track of the time of order placed for future reference.
- **order_amount:** This attribute contains the amount to be paid by the customer for the number and type of tickets ordered by him/her.
- **order_qty:** The customer is allowed to order only the permissible number of tickets which can be of type, one way or two way that are allowed for that particular type of customer. It also refers to the pass recharge, if it is identified or regular.
- **count_of_tickets:** This attribute can have unlimited to limited count of tickets as per the order of customer.

Methods:

- **update_order:** The customer has the flexibility to update the order before placing it.
- **cancel_order:** The customer can discard the order or cancel the order, if not placed.
- **getorder_details:** The customer can view the details of order by inputting the order id.

2. Payment

The customer needs to pay the amount associated with the order, to get the tickets printed either on paper or on pass. The customer can pay the amount using a Credit or Debit card or Cash.

The attributes and methods involved in this class are as follows:

Attributes:

- **customer_id:** The customer id is required to identify the customer who is paying the account.
- **amount:** The amount is associated with the order placed by customer.
- **payment_ type:** The card can be either a debit card or a credit card.
- **transaction_no:** The transaction id is required to keep track of payment associated with the account of customer.
- **Payment_status:** This attribute keeps track of payments whether it is successful or not.

Methods:

- **process_payment:** The customer can make the payment by cash or credit or debit card. It also keeps the status of payment (successful or not).

3. Bank

The bank class imports the data of customer required to make the payment for the order placed.

The attributes involved in this class are:

Attributes:

- **account_number:** The account number associated with the payment method chosen by customer is stored in this attribute.
- **bank_id:** This attribute gives the details of bank with which customer’s account is associated.
- **account_balance:** This attribute is required to compare if the amount of order placed is less than the available balance in customer’s account. If yes, and no other exceptions, payment get successfully done.

4. Pass

This is one of the type of order which decides the amount of tickets to be applied to the pass and the price to be charged to the customer. The pass may be recharged for a *month* or a *week* or just for *weekend.* The attributes and methods involved in this class are as follows:

Attributes:

- **pass_id:** The pass_id is a unique attribute used to identify the pass.
- **tickets_available:** This attribute is important to keep the record of available tickets in the pass for use and recharge.
- **price:** Different type of recharges have different prices and different count of tickets.
- **status:** This attribute is used to compute the validity of the pass for its expiry.

Methods:

- **print_pass:** This attribute print the tickets onto the pass if it is valid and according to the order placed by customer.
- **getpassdetails:** If at any point of time, customer wants to access the details of pass, he or she can view the number of tickets available and status. This is allowed for both the identified and the regular customer.

5. Ticket

This is one of the type of order which decides the amount of tickets ordered by the customer and the price to be charged to the customer. The tickets printed can be of type *one way* or *two way.* The attributes and methods involved in this class are as follows:

Attributes:

- **ticket_id:** The ticket_id is a unique attribute used to identify the ticket.
- **ticket_date:** This attribute is important to keep the record of status of ticket validity for its use.

- **ticket_time:** This attribute is important to keep record of status of ticket validity for its use.
- **source:** The customer have to specify the source for ticket or the travel.
- **destination:** The customer have to specify the destination for ticket or the travel.
- **ticket_price:** Different type of tickets have different status and prices.
- **status:** This attribute is used to compute the validity of the ticket for its expiry.

Methods:

- **print_ticket:** This attribute print the tickets onto the paper according to the order placed by customer.
- **getticketdetails:** If at any point of time, customer wants to access the details of pass, he or she can view the number of tickets available and status. This is allowed for both the identified and the regular customer.

6. Administrator

This is one of the users of Automatic Ticketing Machine (ATM) who is responsible for taking data backups, repairs and maintenance. The attributes and methods involved in this class are as follows:

Attributes:

- **admin_id:** This attribute is used to keep record of access and the changes made by which administrator.
- **admin_username:** This attribute is used to allow access to only the identified administrators and for their login process.
- **admin_password:** This attribute is used to allow login to administrator and for authentication purposes.
- **authority:** This attribute keep record of various grants and revoke rights to the administrator registered with the system.

Methods:

- **create_account:** The administrator can create account for customer only who qualify the identified user criteria.
- **delete_account:** The administrator can delete account of customer once they get non-qualified for identified user criteria.
- **manages_account:** The administrator can view the details of customer account whenever required and can make changes into it.
- **view_order:** The administrator can view the details of order made by customer for tracking purposes.
- **manages_order:** The administrator can not only cancel any order before a time limit but can also modify it.
- **supervise_payments:** The payments made by customer are kept under record for future reference by the administrator.
- **getdata_backup:** The administrator is responsible for getting data backup from time to time.

7. Customer

The customer is the user of the system who initiates the order and can be either the identified or the regular. The identified user are the one with age from 11 to 25 or above 65. The others automatically lies in category of regular customers. For the identified customer, it keep record of age and date of birth of customer. The attributes and methods involved in this class are as follows:

Attributes:

- **customer_id:** The customer id is required for identification purposes of the customer.

Methods:

- **update_order:** The customer has the flexibility to update the order before placing it.

- **cancel_order:** The customer can discard the order or cancel the order, if not placed.
- **place_order:** The customer can finalize the order after selecting the tickets or the pass plan.
- **make_payment:** The customer can make the payment by inserting the card or cash into the machine.

8. Receipt

The system generates the receipt once the order get successfully paid. The attributes and methods involved in this class are as follows:

Attributes:

- **receipt_id:** Every receipt has a unique id for future reference and to get the order details.
- **payment_type:** This receipt also mentions the mode of payment used by the customer.
- **payment_amount:** The receipt generated also print the amount of payment made by the customer.

Methods:

- **print_receipt:** The receipt is printed either on customer request or machine generated. The print receipt bears all the information regarding the order and the payment status.

2. Constraints

Constraints are the imposed conditions that must be satisfied while performing certain function to get the desired output. It is not always possible to express all the constraints in graphical language form so we use Object Constraint Language (OCL) that provides a formal language for specifying constraints which supplement the models created in terms of UML diagrams.

The following are the constraints, based on our class diagram:

1. **Automatic Ticketing Machine serves two types of customers**

- ➢ ***Identified:*** **This type of customer can have account with Automatic Ticketing Machine and must be age between 11 and 25 or above 65. They have special prices for monthly and weekly pass tickets. They get usually a discounted price by 30% of original price.**
- ➢ ***Regular:*** **This type of customer cannot have an account with Automatic Ticketing Machine and does not belong to age group of identified customers.**

Context Customer

inv:

```
if (self.customer_type=identified)

        self.R5 ->order_amount = (order_amount - (order_amount * 0.30))

end if
```

Explanation

According to this constraint, there are two types of customers: regular and identified. If the customer is of type regular, he/she pays the normal price of the order. The customers are registered only for pass recharge. But the identified customers get discounted price - 30% less than the original price of subscription plan.

2. Order is of two types

- ➢ ***Pass:*** **This type of order is specialized and has special prices; by default, it has only one way tickets and those tickets are unlimited in count but have only validity.**
- ➢ ***Ticket:*** **This is in the paper form with unique id and count can be decided by the customer.**

Context Order

inv:

```
if (self.order_type=weekly pass or self.order_type=monthly pass or
self.order_type=weekend pass)
        self.count of tickets=unlimited

end if
```

Explanation

According to this constraint, if the customer needs to get their tickets printed according to weekend, weekly or monthly plan, the count of tickets is unlimited.

3. Customer must have an order

Context Customer

inv:

    self.R5 -> notEmpty()

Explanation

According to this constraint a customer must have an order.

4. Selection of a payment mode

Context Customer

inv:

    self.R14 -> notEmpty()

Explanation

According to this constraint a customer must select a payment mode to make payment.

5. Order is placed only after payment is made

Context Customer

inv:

    self.R14.R6 -> includesAll (self.R5)

Explanation

According to this constraint, customer must make the payment corresponding to the order placed.

6. Each order must have at least one item which can either be a pass or a ticket

Context Order

inv:

    self.R11-> notEmpty()

Explanation

According to this constraint, each order must have something in the cart.

7. Customer age for having account should be greater than 11 and less than 25 or greater than 65 years.

Context Customer

inv:

(self.customer_age>=11 and self.user_age <= 25) or (self.customer_age>65)

Explanation

According to this constraint, user can have an account if his/her age lies in 11 and 25 years or greater than 65 years.

8. System must print the receipt if the payment has been made

Context Payment

inv:

self. R6-> notempty() implies self.R14-> notempty()

Explanation

According to this constraint, the receipt should be printed in the case if payment has been made by the customer.

9. System must check for account balance of the customer before making the payment of order

Context Payment :: process_payment (d : order_id)

inv:

@pre : self.account_balance (self.R7.account_balance > self.R6.order_price)

@post : self.account_balance @pre – self.R6.order_price

Explanation

According to this constraint, the system checks for the account balance of the customer in case of card payment from the bank account. If the account balance is more than order price, only then payment is processed.

10. Customer can change their order if order is under process

Context Customer

inv:
@pre :   self.R5.order_id -> self.order_underprocess AND
          self.R5.order_changedate > = currentDate
@post :   self.R5.order_id = self.R5.neworder
          self.R5.order_id.previousorder = self.R5.order_id @pre
          self.R5.order_id.previousorder.order_expirydate = self.R5.order_changedate

Explanation

According to this constraint, user can change his order if his order is still in process.

11. No two orders can have same id.

Context Customer

inv:

self.R5 -> forAll (O1, O2 | O1<>O2 and O1.order_id<> O2.order_id)

Explanation

According to this constraint, each order must have a unique identity.

12. No two tickets can have same id.

Context Customer

inv:

self.R5.R11 -> forAll (T1, T2 | T1<>T2 and T1.ticket_id<> T2.ticket_id)

Explanation

According to this constraint, each ticket must have a unique identity.

References

[1] Dr Abdel-Wahab Hamou Lhadj lecture slides and notes on the study of OCL.
[2] https://en.wikipedia.org/wiki/Class_diagram.
[3] http://www.uml-diagrams.org/class-diagrams-examples.html.
[4] http://www.agilemodeling.com/artifacts/classDiagram.htm.
[5] http://www.omg.org/spec/OCL/2.3.1.