COMP201: Software Engineering I

Object Oriented Design

Coursework Assignment 2

Modelling with UML

Ziheng Zhang – 201220030 – x6zz
13 December 2016

TASK 1.

Given the following informal specification, **identify good candidates** for classes and attributes, and **identify things that are outside** of the problem domain. Also identify **all potential inheritance relationships**. You should ensure that data is **NOT** duplicated across classes even if a user places multiple bookings. Use the noun identification method of class elicitation for the first pass.

> *Your customer is a travel agency that wants a reservation system that will run on the Internet. This reservation system will allow clients to keep track of all their travel reservations for airlines, hotel and rental cars. The client must enter the names of all his/her traveling companions, but all reservations will be under the name of the primary client. The system needs to make it easy for a client to have multiple reservations. All reservations will include a booking number as well as their names, passport numbers and dates of birth of all the travelers involved in the reservation. The system should also have an address for the primary client. Airline reservations will include the airline, flight number, class of seat and travel dates and times. Hotel reservations will include the type (twin, single, double) and of rooms and the dates staying, and name and address of the hotel. Car rental reservations will include the type of car requested, dates and the drivers' license number of the primary client.*

SOLUTION

Firstly, all the nouns in the specification are underlined with the noun identification method.

> *Your customer is a travel agency that wants a reservation system that will run on the Internet. This reservation system will allow clients to keep track of all their travel reservations for airlines, hotel and rental cars. The client must enter the names of all his/her traveling companions, but all reservations will be under the name of the primary client. The system needs to make it easy for a client to have multiple reservations. All reservations will include a booking number as well as their names, passport numbers and dates of birth of all the travelers involved in the reservation. The system should also have an address for the primary client. Airline reservations will include the airline, flight number, class of seat and travel dates and times. Hotel reservations will include the type (twin, single, double) of rooms and the dates staying, and name and address of the hotel. Car rental reservations will include the type of car requested, dates and the drivers' license number of the primary client.*

An initial class and attribute list for the system is given as below.

- TravelAgency: superfluous candidate
- ReservationSystem: superfluous candidate
- Client: candidate class
- name: attribute of class Client
- passportNumber: attribute of class Client
- dateOfBirth: attribute of class Client
- Reservation: candidate class
- bookingNumber: attribute of class Reservation
- nameOfPrimary: attribute of class Reservation
- Airline: candidate class
- airline: attribute of class Airline
- flightNumber: attribute of class Airline
- classOfSeat: attribute of class Airline
- travelDates: attribute of class Airline
- travelTimes: attribute of class Airline
- Hotel: candidate class
- typeOfRooms: attribute of class Hotel
- stayingDates: attribute of class Hotel
- hotelName: attribute of class Hotel
- hotelAddress: attribute of class Hotel
- RentalCar: candidate class
- typeOfCar: attribute of class RentalCar
- dates: attribute of class RentalCar
- licenseNumberOfPrimary: attribute of class RentalCar
- TravelingCompanion: superfluous candidate

- PrimaryClient: candidate class
- drivingLicense: attribute of class PrimaryClient
- address: attribute of class PrimaryClient
- MultipleReservation: superfluous candidate

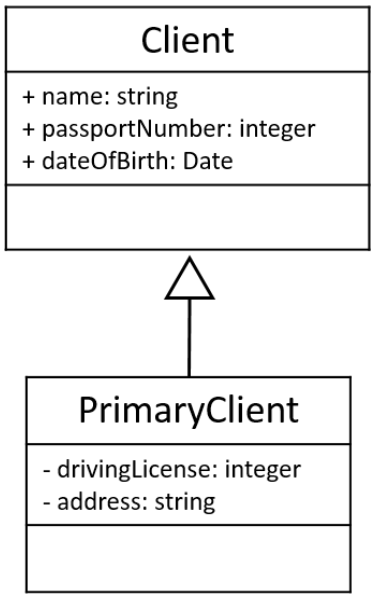
The superfluous candidate classes are listed as below.

- TravelAgency: removing, because it is outside the problem domain
- ReservationSystem: removing, because it is not a part of the domain but part of the meta-language of requirements description
- TravelingCompanion: removing, because redundant covered by Client
- MultipleReservation: removing, because redundant covered by Reservation

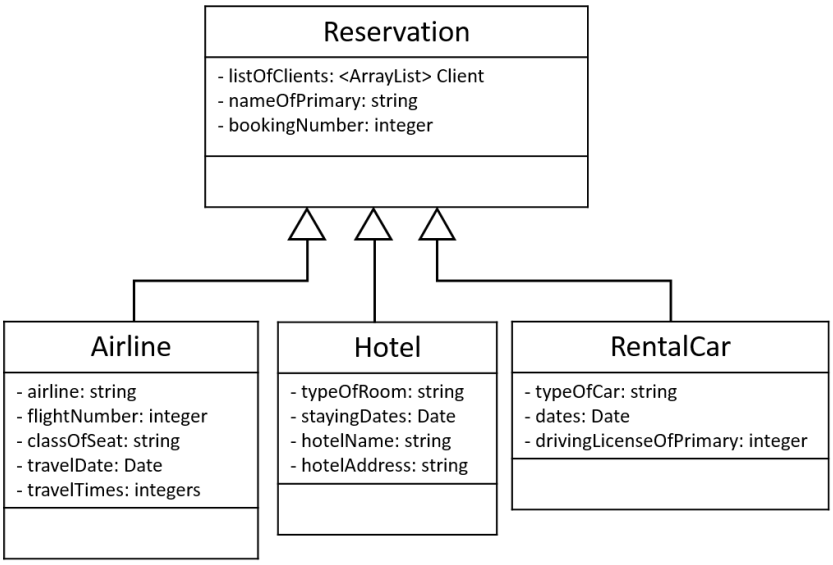
After removing superfluous candidates, there are six classes remaining (What are inside bracket are their own attributes).

- **Client** (name, passportNumber, dateOfBirth)
- **Reservation** (bookingNumber, nameOfPrimary)
- **Airline** (airline, flightNumber, classOfSeat, travelDates, travelTimes)
- **Hotel** (typeOfRooms, stayingDates, hotelName, hotelAddress)
- **RentalCar** (typeOfCar, dates, licenseNumberOfPrimary)
- **PrimaryClient** (drivingLicense, address)

Inheritance is the sharing of attributes and operations among classes based upon a hierarchical relationship. A class can be defined broadly and then refined into successively finer subclasses. The potential inheritance relationships are displayed as below.



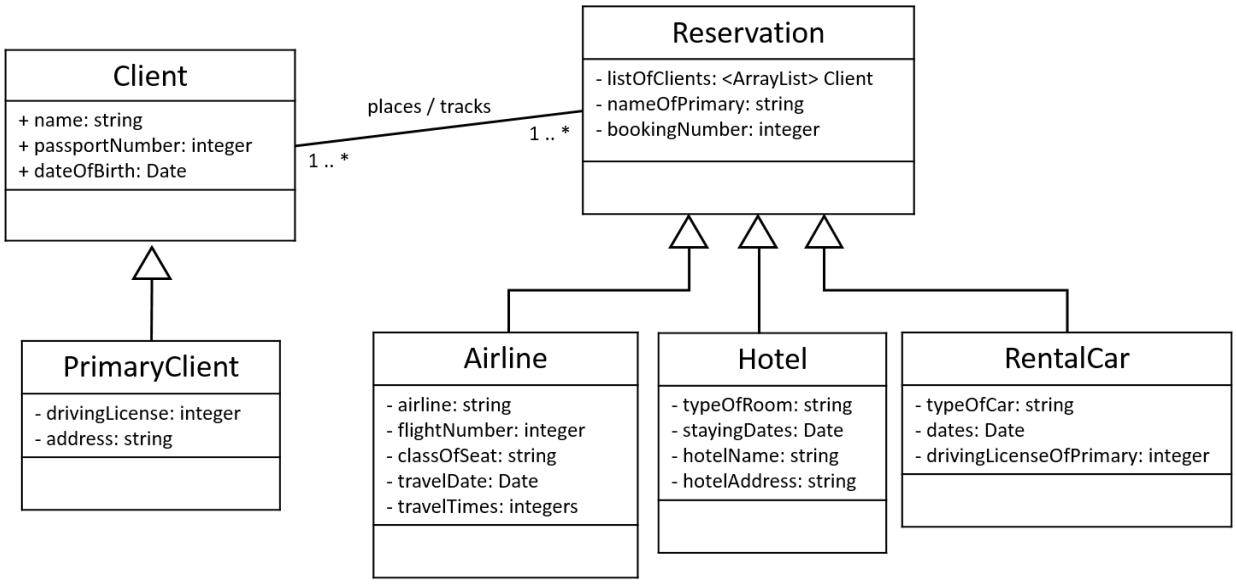
The class **Primary client** is the extended subclass of class **Client**. The explanation is that primary client is also a client but he or she is special in some aspects compared with other clients. A primary client should also have the same attributes as a client, but with more special attributes, driving license and address.



The class **Airlines**, **Hotel**, **RentalCars** are the extended subclass of class **Reservation**. A reservation may involve several clients so there is an array list to store all involved clients. But it should have an attribute of nameOfPrimary to put this **Reservation** object under the name of primary client. This will not duplicate data as the information of clients from class **Client** is accessible to class **Reservation** to store. Therefore, the attributes in class **Client** are public for class **Reservation** to access.

In this system, whether a traveler is a client or a primary client is depending on the reservation. For example, one traveler could be the client of one reservation but also be the primary client of the other reservation. So every client should be able to place or track the reservation that he is involved in. Meanwhile, one client, whether primary or not, can have more than one reservation. Therefore, the diagram records that (association with no navigability):

- For each object of class **Client**, there are some **Reservation** objects (the number is unspecified) which are associated with **Client**.
- For each object of class **Reservation**, there are some **Client** objects (the number is unspecified) which are associated with **Reservation**.



TASK 2.

You are required to draw a UML **activity diagram** to represent the following scenario of a hairdresser's salon.

> *Customers enter the salon and wait until the next hairdresser is free. They then indicate whether they would like their hair washed first or a "dry-cut" without having their hair washed. The hairdresser washes the hair (if asked for) and then cuts it. After finishing the customer's hair the hairdresser moves onto the next waiting customer, or waits for another one to enter the salon. The customer goes to the till and waits for a cashier to be free to take their payment. They can pay by either cash or by credit card (where they need to type their pin into the machine) and they then leave the salon.*

SOLUTION

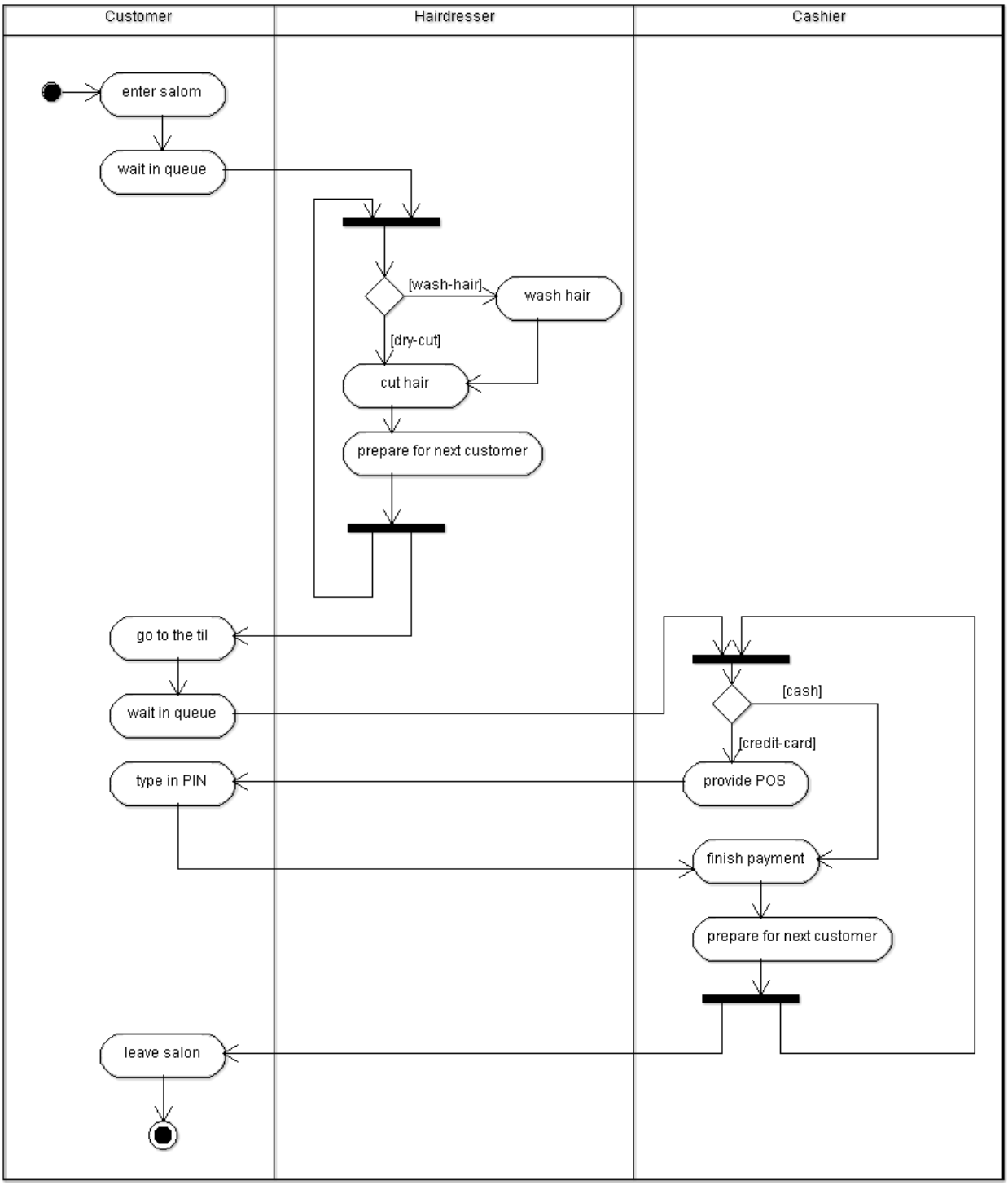
Firstly, all the activities in the scenario are listed as below.

1. Customer enters a salon
2. Customer waits until one hairdresser is free
3. Customer indicates whether washing hair **(after synchronisation bar)**
   3.1. Hairdresser washes hair
4. Hairdresser cuts hair
5. Hairdresser prepares for the next customer **(before synchronisation bar)**
6. Customer goes to the till
7. Customer waits until a cashier is free
8. Customer indicates the payment method **(after synchronisation bar)**
   8.1. Cashier provides POS and Customer types in PIN to pay
   8.2. Customer pays by cash
9. Cashier prepares for the next customer **(before synchronisation bar)**
10. Customer leaves the salon

It is clear that this scenario has three involved actors, Customer, Hairdresser and Cashier. The activity diagram starts at Customer entering a salon and stops at Customer leaving the salon. There are two decision to make during this activity diagram, Customer indicating whether to wash hair and Customer indicates the way to pay.

As the synchronisation bar describes the co-ordination of activities which must all be completed before the activity edges leading from the bar are fired. In other words, for example, only when one hairdresser is preparing for next customer and one customer is waiting in queue, the next action will occur that customer indicates whether to wash hair or not. After haircutting, the customer moves on to the till and the hairdresser movers on to preparing for next customer, where two activities happen synchronously. Only when one customer is waiting in queue and one cashier is preparing for next customer, the next action will occur that customer choose the way to pay. After payment, the customer moves on to leave salon and the cashier moves on to preparing for next customer.

Based on the previous analysis of ordered activities, the UML activity diagram is displayed as below.


Customer
Hairdresser
Cashier
enter salom
wait in queue
[wash-hair]
wash hair
[dry-cut]
cut hair
prepare for next customer
go to the til
wait in queue
type in PIN
[cash]
[credit-card]
provide POS
finish payment
prepare for next customer
leave salon

TASK 3.

Read the following passage **carefully**.

> *An employee has a name, address, phone number, date of birth and job title. Employees can be appointed and can leave, and are either monthly paid employees or weekly paid employees.*
>
> *Monthly paid employees have a bank sort code, bank account number and number of holidays while weekly paid employees are paid in cash on a specified day of the week - their payday. Weekly paid employees may apply to be promoted to a monthly paid employee. Monthly paid employees can take a holiday if they have sufficient number of holidays remaining.*
>
> *All employees are entitled to use the Sports Centre if they register to do so. The Sports Centre is made up of two gyms (with a maximum capacity), three tennis courts and a bar.*
>
> *The bar can be booked for special events, and has three rates of hire - a working hours' rate, an evening rate and a weekend rate. The Sports Centre holds a list of employees who have registered.*
>
> *An employee's age can be calculated from their date of birth, in order to prevent under-age drinking at the bar.*

You are required to draw a UML **class diagram** for the above system. All the key words you need to include are underlined – do *not* invent any details additional to those given above:

1. Illustrate the various classes that exist, with their attributes and operations (including any derived ones, represented in the usual way)
2. Mark on the relationships that exist between the classes using the standard UML symbols to represent the *type* of each relationship
3. Add multiplicities
4. for any relationships of **association:**
    a. mark on the navigability
    b. appropriately name the two roles

SOLUTION

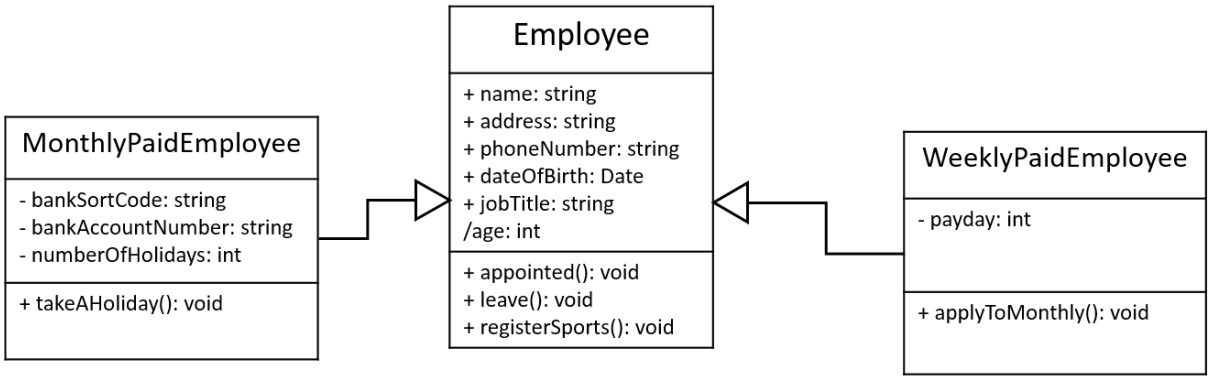
List all the underlined noun and identify classes among them (What are inside the basket are the attributes and operations).

- Employee (name, address, phoneNumber, dateOfBirth, jobTitle, appointed(), leave(), registerSports(), getAge())
- MonthlyPaidEmployee (bankSortCode, bankAccountNumber, numberOfHolidays, takeAHoliday())
- WeeklyPaidEmployee (payday, applyToMonthly())
- SportsCentre (listOfEmployees)
- Gym (maxCapacity)
- TennisCourt
- Bar (workingRate, eveningRate, weekendRate, beBooked())

The visibility of attributes in class **Employee** is public as class **SportsCentre** should get access to all the registered employee's information.

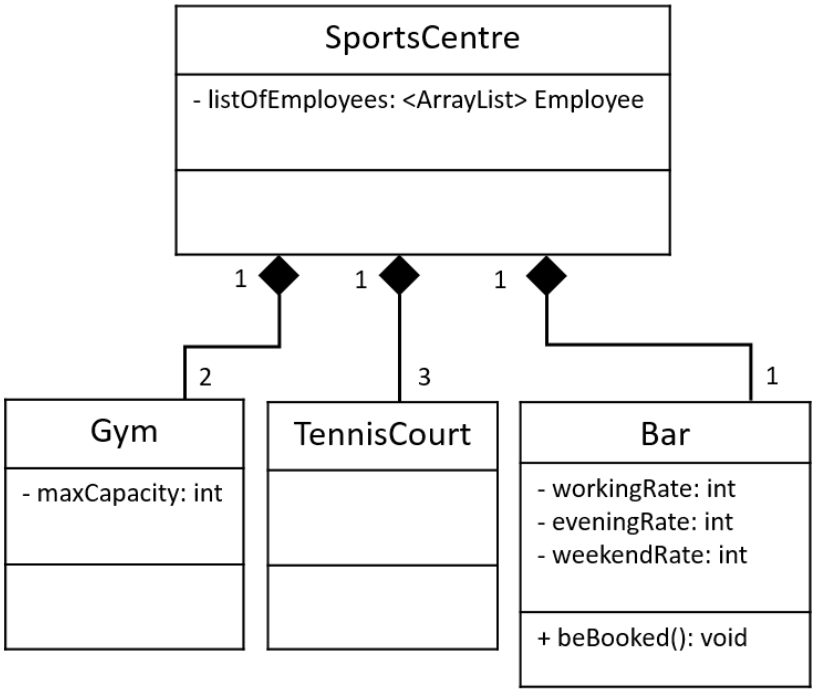
Since classes are interrelated to each other with their different type of logical connections. Both type of employees, monthly paid and weekly paid, are the employees who is registered with all information and has the same right to share the Sport Centre. The relationship, therefore, between **Employee**,

MonthlyPaidEmployee and **WeeklyPaidEmployee** is inheritance. The attribute age is a derived attribute whose value is calculated (derived) from the attribute dateOfBirth. The inheritance relationship is displayed as below.



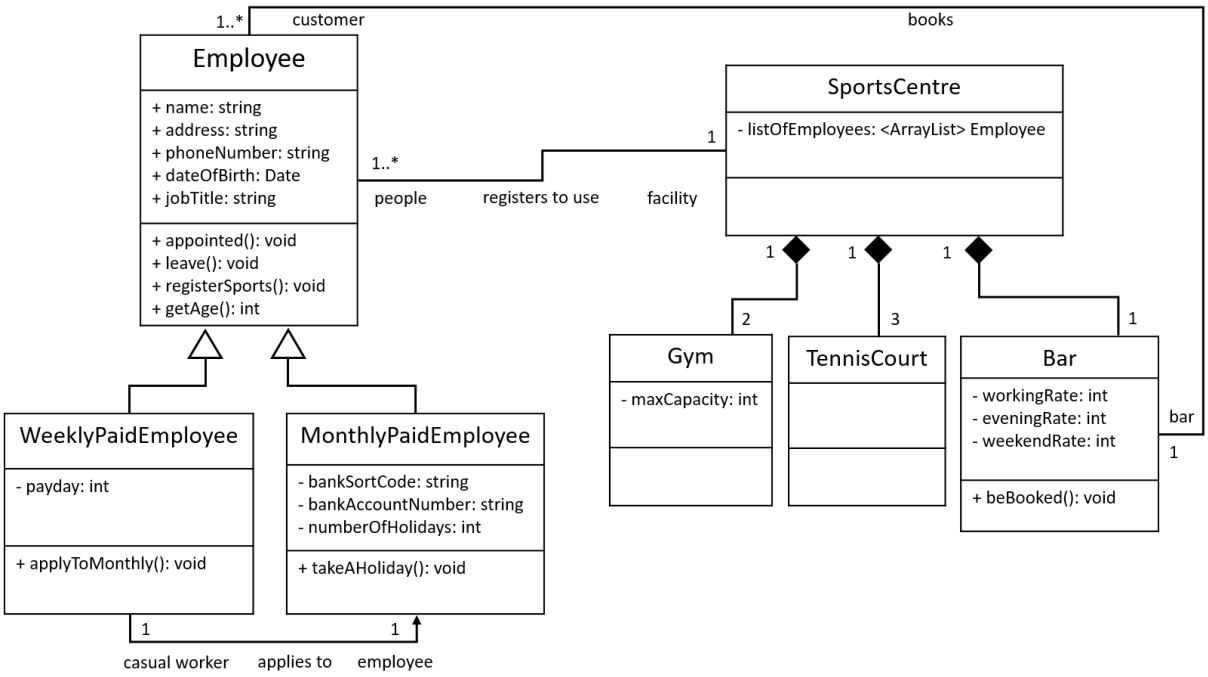
Relationship between classes is **inheritance**

Since it is indicated that "*The Sports Centre is made up of two gyms, three tennis courts and a bar*", the whole Sports Centre strongly owns its parts gym, tennis court and bar, which means, in other words, if the Sports Centre does not exist, the parts will no longer exist. The relationships, therefore, between **SportsCentre** and **Gym**, **SportsCentre** and **TennisCourt**, **SportsCentre** and **Bar** are composition. One Sports Centre has two gyms, three tennis courts and a bar. The multiplicities should be added to the relationship that is displayed as below.



Relationship between classes is **composition**

The UML class diagram is displayed as below.



The association between **MonthlyPaidEmployee** and **WeeklyPaidEmployee** should be navigable that only weekly paid employee has opportunity to apply for monthly paid employee. In other words, the messages can only be sent in the direction of arrow. The roles are clarified that casual worker applies to employee.

Every employee can register to use the Sports Centre, so the association between **Employee** and **SportsCentre** records that: For each object of class **Employee** there are only one object of class **SportsCentre** which are associated with the **Employee**. For each object of class **SportsCentre** there are some **Employee** objects associated with the **SportsCentre**. The roles are clarified that people registers to use facility.

The association between **Employee** and **Bar** is quite similar. There are some **Employee** objects associated with the **Bar** and there is one object of class **Bar** associated with the **Employee**. The roles are clarified that customer books bar.

TASK 4.

Draw a UML **sequence diagram** that specifies the following protocol of initiating a two-party phone call. NOTE: ArgoUML does not fully support Sequence Diagrams, it may be better to use a different program (such as OpenOffice Draw/ Microsoft PowerPoint) or (neatly) draw the diagram by hand. Let us assume that there are four objects involved:

- two Callers (s and r),
- an unnamed telephone Switch, and
- Conversation (c) between the two parties.

The sequence begins with one Caller (s) sending a message (liftReceiver) to the Switch object. In turn, the Switch calls setDialTone on the Caller, and the Caller iterates (7 times) on the message dialDigit to itself. The Switch object then calls itself with the message routeCall. It then creates a Conversation object (c), to which it delegates the rest of the work. The Conversation object (c) rings the Caller (r), who asynchronously sends the message liftReceiver. The Conversation object then tells both Caller objects to connect, after which they talk. Once Caller (r) sends a disconnect message to Conversation then Conversation tells both Caller objects to disconnect and also it tells the Switch to disconnect. After that Switch deletes the object Conversation.

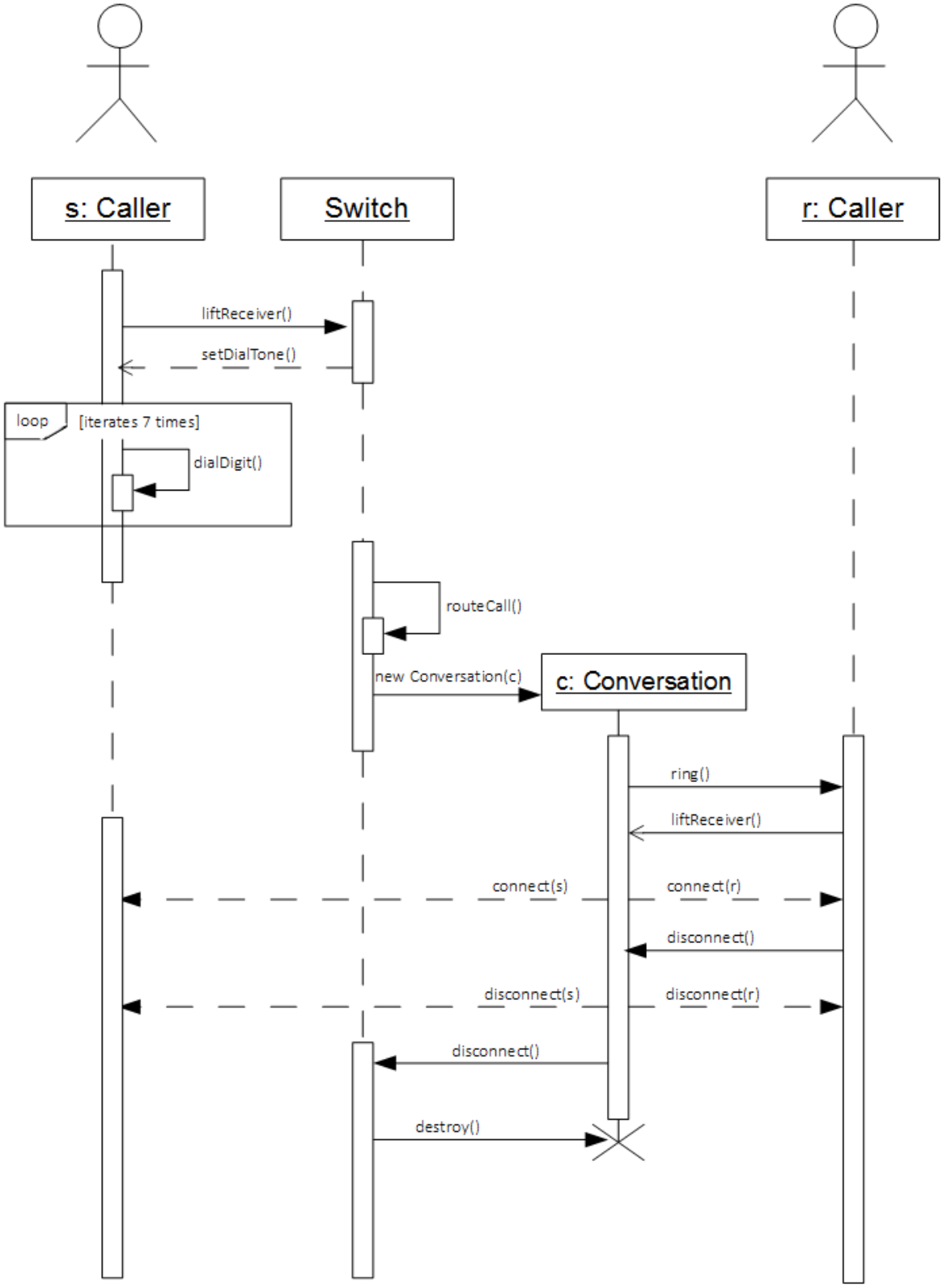
All the key words you need to include are underlined – do *not* invent any details additional to those given above.

SOLUTION

Sequence diagrams shows the objects and actor which take part in a collaboration at the top of dashed lines. Therefore, the actors are clarified as Caller (s) and Caller (r) and the objects are clarified as Switch and Conversation (c). The whole sequence is listed as below.

1. Caller (s) sends message liftReceiver() to Switch
   1.1. Switch sends message setDialTone() back to Caller (s)
2. Caller (s) sends message dialDigit() to itself for a loop of seven times.
3. Switch sends message routeCall() to itself
4. Switch creates an object Conversation (c)
5. Conversation (c) sends message ring() to Caller (r)
   5.1. Caller(r) sends message liftReceiver() back to Conversation (c) asynchronously
   5.2. Conversation (c) sends message connect(s) to Caller (s) and message connect(r) to Caller (r)
6. Caller (r) sends message disconnect() to Conversation (c)
   6.1. Conversation (c) sends message disconnect(s) to Caller (s) and message disconnect(r) to Caller (r)
7. Conversation sends message disconnect() to Switch
   7.1. Switch destroys the object Conversation (c)

Based on previous analysis, the exact sequence diagram is displayed as below.


s: Caller
Switch
r: Caller
liftReceiver()
setDialTone()
loop
[iterates 7 times]
dialDigit()
routeCall()
new Conversation(c)
c: Conversation
ring()
liftReceiver()
connect(s)
connect(r)
disconnect()
disconnect(s)
disconnect(r)
disconnect()
destroy()