OO Analysis: Domain Model
Case Study

Here we discuss information system of “Hire-n-Ride-Bike” shop which we have described in lecture notes “OO Analysis - Use-Case-Modeling-A Case Study”

In Islamabad for environment protection, a bike(bicycle) hire shop (Hire-n-Ride-Bike) is started Super-market. Main aim is to encourage residents and tourists to explore Islamabad and Margalla hills tracks on bikes. For that purpose, shop maintains a range of bikes (ladies, sports, children, mountain etc.).
At present all the activities of the shop are maintained through registers and paper receipts. All the information about the bikes (bike number, type, size, make, model, daily charge rate and deposit) are recorded on a register. The shop also has a small workshop which is looked after by a mechanic who is responsible to keep the bikes in running conditions and inspects the bike when it is returned to access any damages to a bike.

Shop owner is eager to have a computer-based system to manage all the activities for bike hiring and to maintain bikes inventory. To help the owner we offer to create a software-based solution for his hire-n-ride-bike shop.

We have already created use case model in lecture notes “OO Analysis - Use-Case-Modeling-A Case Study”
Here we will create Domain Model (Basic Class Diagram).
A class diagram that sets out to model all of the classes in the problem domain. We perform Objects Identification by identifying Nouns in requirements description of “Hire-n-Ride-Bike” requirements and highlighted in blue as listed below:

- R1: The system must keep a complete list of all bikes and their details including bike number, type, size, make, model, daily charge rate, deposit

- R2: keep a record of all customers and their past hire transactions
- R3: work out automatically how much it will cost to hire a given bike for a given number of days

- R4: record the details of a hire transaction including the start date, estimated duration, customer and bike, in such a way that it is easy to find the relevant transaction details when a bike is returned

- R5: keep track of how many bikes a customer is hiring so that the customer gets one unified receipt not a separate one for each bike

- R6: cope with a customer who hires more than one bike, each for different amounts of time

- R7: work out automatically, on the return of a bike, how long it was hired for, how many days were originally paid for, how much extra is due

- R8: record the total amount due and how much has been paid
- R9: print a receipt for each customer
- R10: keep track of the state of each bike, e.g. whether it is in stock, hired out or being repaired
- R11: provide the means to record extra details about specialist bikes

We list identified objects by removing duplicates as in the following:

list of bikes charge rate, deposit
details of bikes: bike number, type, size, make, model, daily
record of customers past hire transactions
bike number of days
details of a hire transaction: start date, estimated duration
customer different amounts of time
return of a bike total amount due
state of each bike
extra details about specialist bikes

Now from the above list of candidate objects, we reject those that are unsuitable by considering following factors:

Attributes: Sometimes it is clear that a noun is an attribute of an object rather than an object itself. Bike number, type, size, make, model, daily charge rate and deposit are clearly attributes of a bike object. Similarly, hire transaction sounds like a possible object, with start date and estimated duration as its attributes.

Redundant: Sometimes the same concept appears in the text in different guises. Here past hire transactions and hire transaction are probably the same thing. Different amounts of time, number of days and estimated duration are probably the same thing – all three refer to the length of a hire (in any case they are attributes, not objects).

Too vague: If we don't know exactly what is meant by a term it is unlikely to make a good object/class. For this reason, we reject return of a bike as an object.

Too tied up with physical inputs and outputs: This refers to something that exists in the real world but is a product of the system or data input to the system and not an object in its own right. Receipt qualifies for rejection under this heading. Receipt is an output.

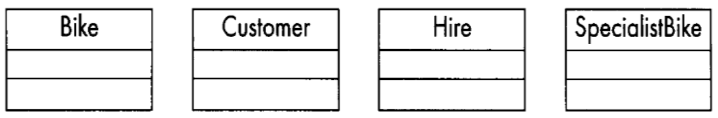
Associations: If there is data associated with the relationship, then we probably want to model it as an object.

Really an operation: If a candidate object seems to have no data associated with it, then it might be better modelled as an operation on another class.

By performing noun analysis and considering above stated factors we derive following four candidate classes:

Bike Customer
Hire transaction Specialist bike

These classes are listed below in a visual form:

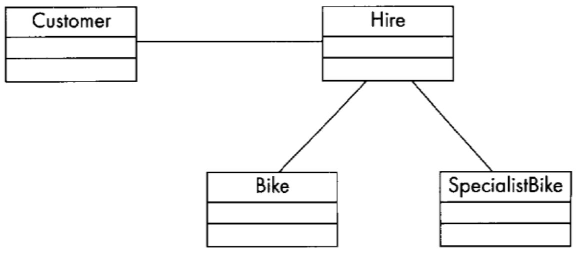


We can use these classes to form the basis of our class diagram, i.e. initial class diagram for "Hire-n-Ride-Bike" system.

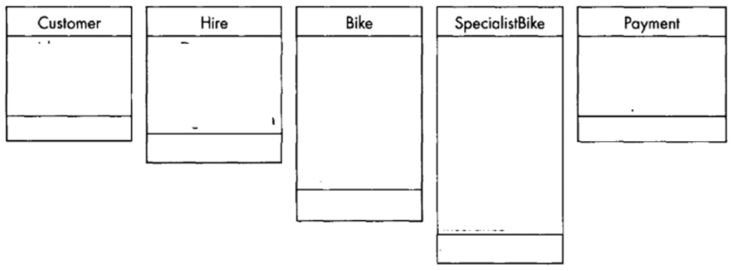
Next, we determine relationships between classes. As we know that relationships between classes can be linked with one another in three ways:

- a sub-/superclass relationship (Generalization)
- by means of an Aggregation/Composition
- via Associations

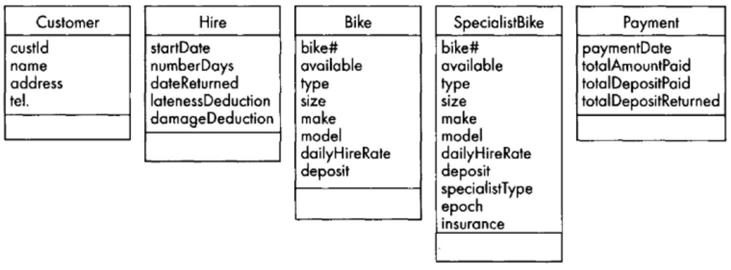
By guessing about relationships between classes, we could have a first draft of domain model as shown below:



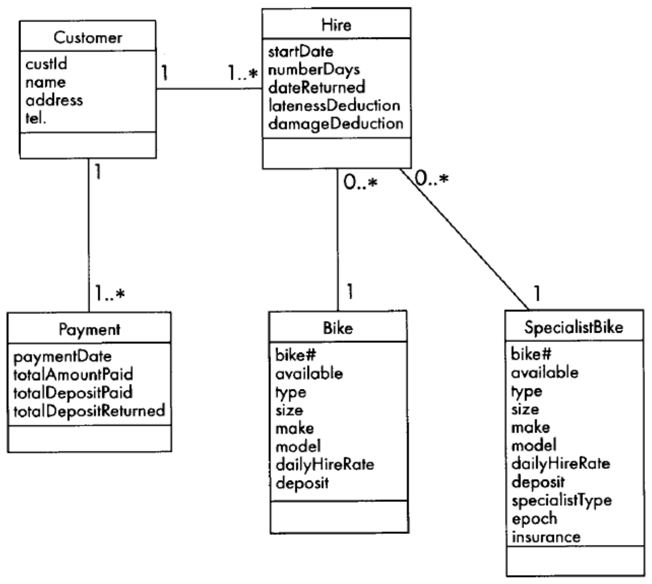
It is much simpler and less messy to store the dates in a Hire class associated with both the Customer class and the Bike class. Each hire object will contain the attributes relating to only one bike and one customer.
To cater payment issue, in requirement R5 customer will expect to pay for multiple bikes at once and have a single receipt. With the classes we have so far we have to see where to put data about payments and an operation to work out totals. The customer will expect to pay for all bikes at once and have a single receipt. With the classes we have so far it's hard to see where to put data about payments and an operation to work out totals. We introduce a separate Payment class and store details about financial transactions.
Now we have five classes as shown below:



Next, we identify the attributes, and we can describe our classes in more detail using attributes. One can define meaningful data types for attributes even though these may not be included in the specification. Use cases descriptions and scenarios are also helpful in defining attributes. We write attributes of respective classes in attributes section of visual classes as shown below:

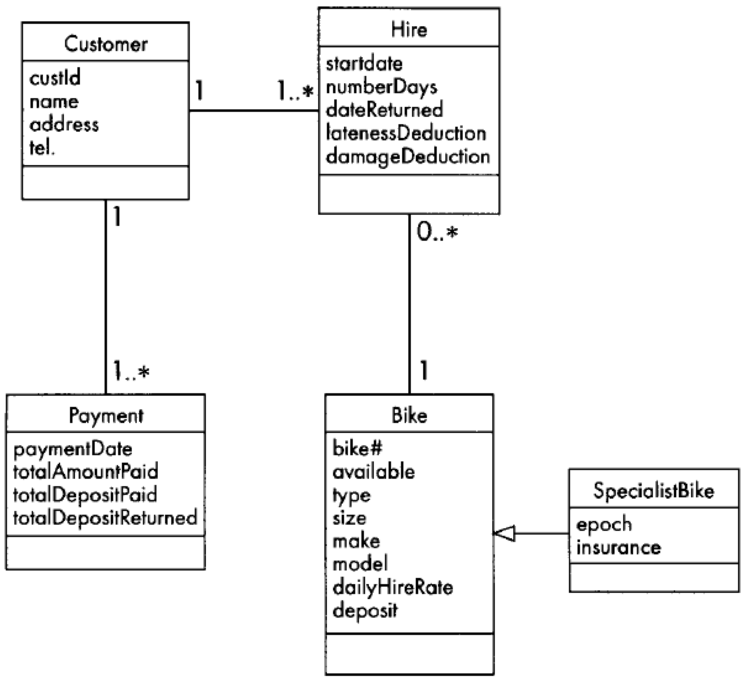


Along with attributes of classes, and relationships between classes identified we can refine our first of draft of domain model as depicted below:



A :Customer can make many :Hires, but a :Hire is a specific transaction relating to just one :Customer. A :Payment is made by just one :Customer but a :Customer can make many :Payments. Similarly a :Hire is for one :Bike only. A :Bike, on the other hand, can be hired many times or may not be hired at all. These situations are depicted in in association multiplicities of the model.

These seems to be a generalization association (Class-Subclass Relationship) also referred Inheritance between *Bike* and SpecialistBike class. With this association being depicted in the final form of the domain model as shown below:


Customer
custId
name
address
tel.
1
1..*
Hire
startdate
numberDays
dateReturned
latenessDeduction
damageDeduction
1
0..*
1..*
1
Payment
paymentDate
totalAmountPaid
totalDepositPaid
totalDepositReturned
Bike
bike#
available
type
size
make
model
dailyHireRate
deposit
SpecialistBike
epoch
insurance