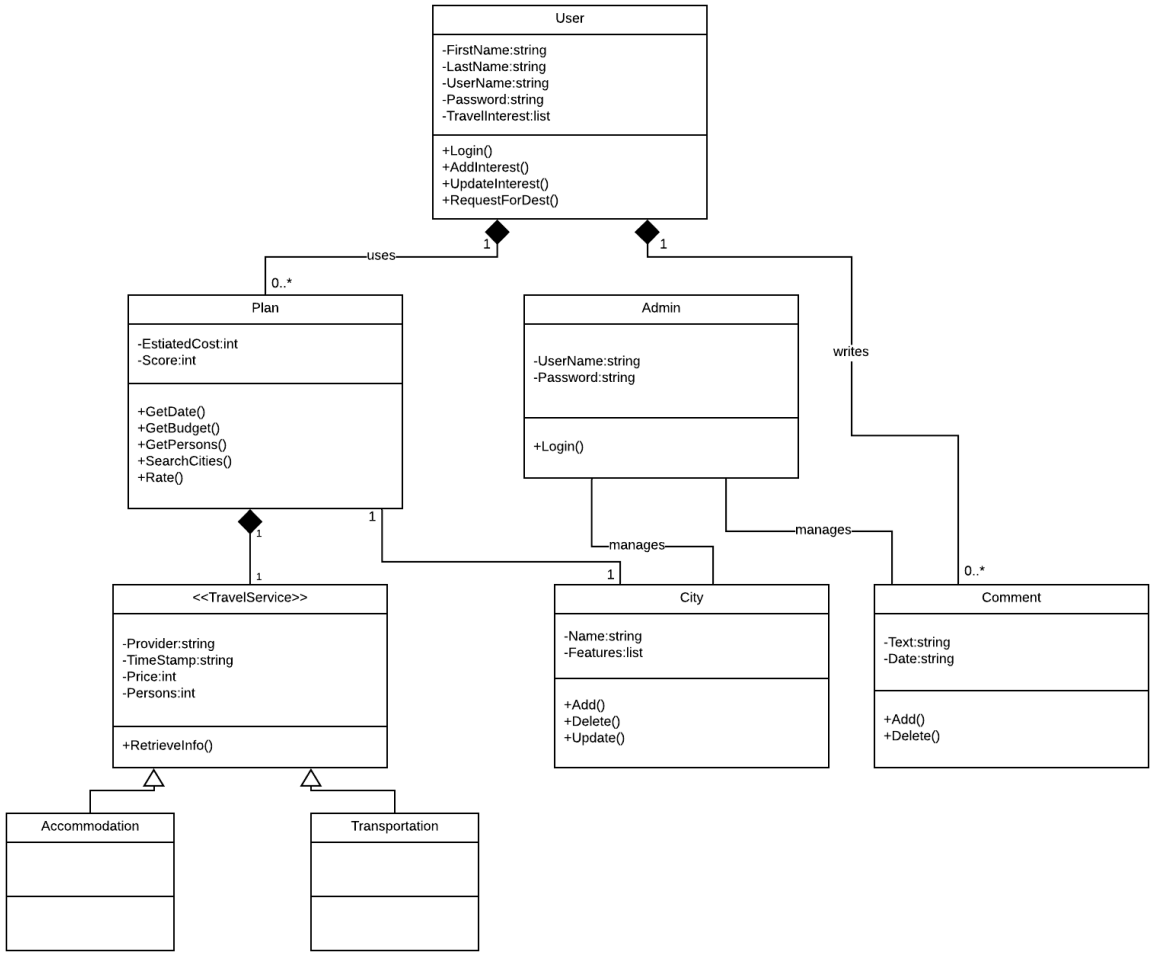
Avoiding Superfluity:

In order to avoid superfluity, we can:

1. Combine classes with same features and methods into a super-class. In our diagram the accommodation and transportation classes can inherit from a larger abstract class called travel service, which makes it easier for the later extension, for example for adding new travel services like food to our plan, and makes the design meet the open/close principle of solid principles.



2. Remove superfluous attributes of the class which are in association with each other. In our diagram such attributes do not exist.

Detecting Cycles of Associations:

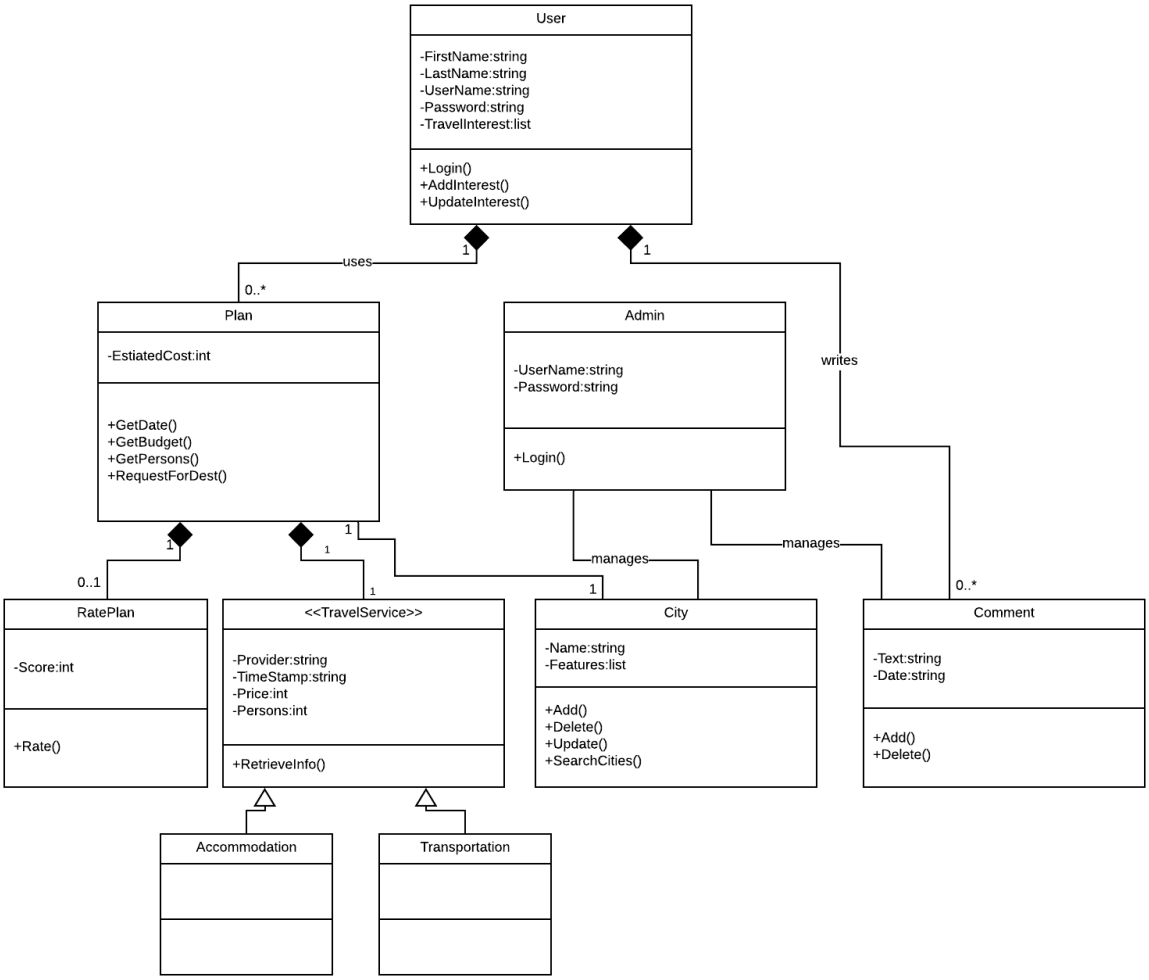
The only cycle in our diagram is the one including the classes User, Plan, City, Admin and Comment, fortunately this is not a cycle of associations since two of its relationships are composition (Plan-User, Comment-User), hence we must not modify the diagram according to this feature.

Trying to apply SOLID principles on the diagram:

Single purpose: for this property to hold, we should check-out every single class in our diagram to see if it is responsible for too much:

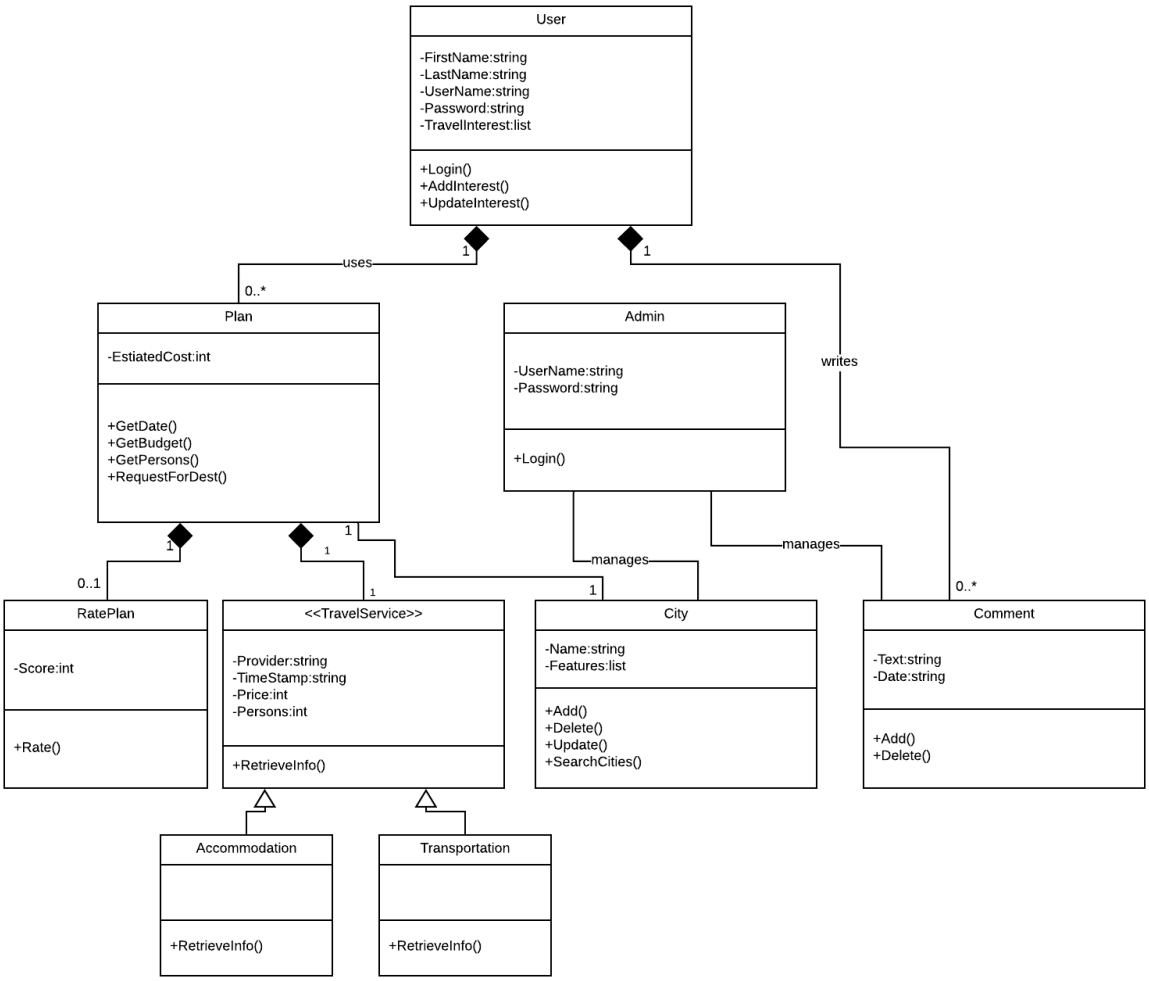
Class User: all of the methods belonging to this class are related to the stuff associated only with the user except the method Request Destination which tends more towards the Plan.

Therefore, if we just remove this method from the class User and add it to the class Plan the User will become a single purpose class.
Class Plan: The above situation also holds for the Class Plan and its search cities method, which is handled by adding this method to the Class City.
Also, all of the methods of this class are related to the act of information gathering for proposing a plan except the method Rate which is used to evaluate the plan. We can fix the problem of the class Plan being too responsible by using composition. In this solution, we can make a class named Rate Plan which has an instance of the class Plan.



open/close: Apparently, our class diagram is open to extension and close to modification

Liskov Substitution Principal: the classes Transportation and Accommodation extend the class Travel Service but we cannot remove and replace this super-class with its subclasses. Thus, we should modify the class diagram as follows:



Interface Segregation: Apparently, Clients are not forced to depend on methods they do not use.

Dependency Inversion: It seems that Higher level classes do not depend on lower level classes.

Resources:

1. Optimizing UML Class Diagrams, Maxim Sergievskiy and Ksenia Kirpichnikova https://www.researchgate.net/publication/324345033_Optimizing_UML_Class_Diagrams
2. SOLID principles