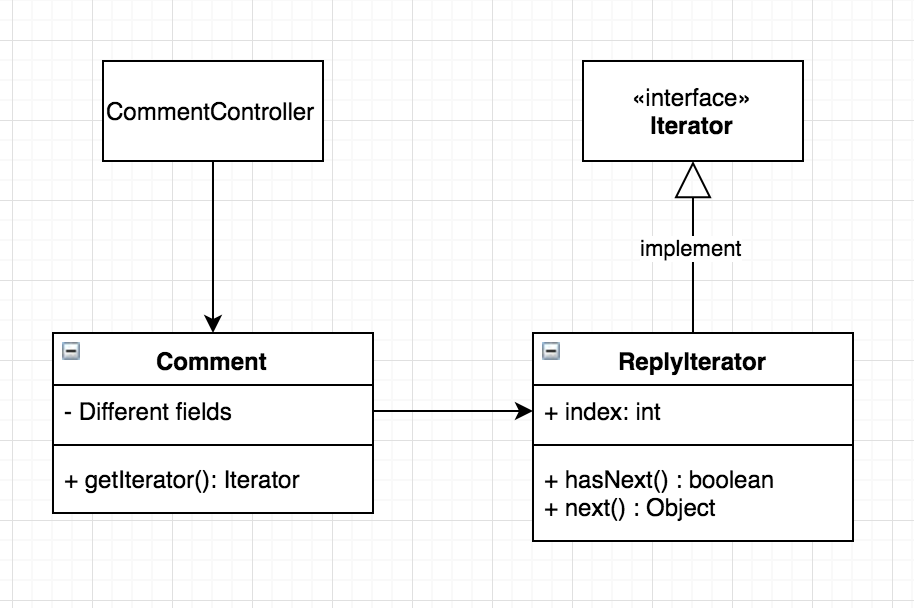
# Architectural Design

**Iterator Pattern**

Iterator pattern is a design pattern in which an iterator is used to traverse a container and access the container's elements. The iterator pattern decouples algorithms from containers.

Here in our back-end code, we used Iterator pattern to help traverse all the replies in Comment class. First of all, I defined an ReplyIterator class in Comment class which implements Iterator interface. I rewrited the hasNext() and next() method to help us go through all replies in reply list. In comment controller, when we need to get all the replies in reply list, we just use ReplyIterator to do that thing.

The UML graph is shown as below:

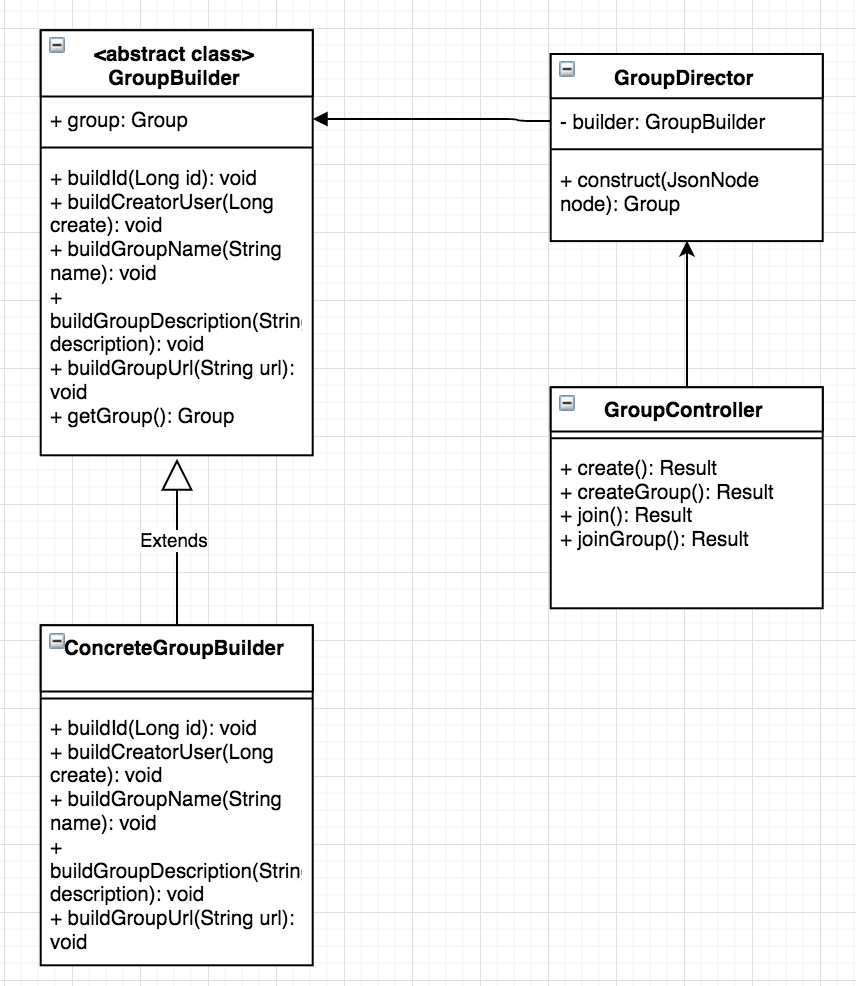


**Builder Pattern**

Builder Pattern is a design pattern used to build complex pattern by using simple objects and using step by step approach. A builder class is needed in this pattern. This builder class builds the final object step by step. And this builder is independent of other objects.

In out implementation, we used builder pattern to help create new group. Group is divided into 5 different parts - id, creatoruser, groupname, groupdescription, groupurl. Abstract class GroupBuilder contains methods to create those fields step by step. Concrete is the class to extend GroupBuilder and implement abstract methods in detail. We also made a GroupDirector to manage the use of GroupBuilder. Whenever a new Group is needed to be created, I just call GroupDirector to make a new Group.

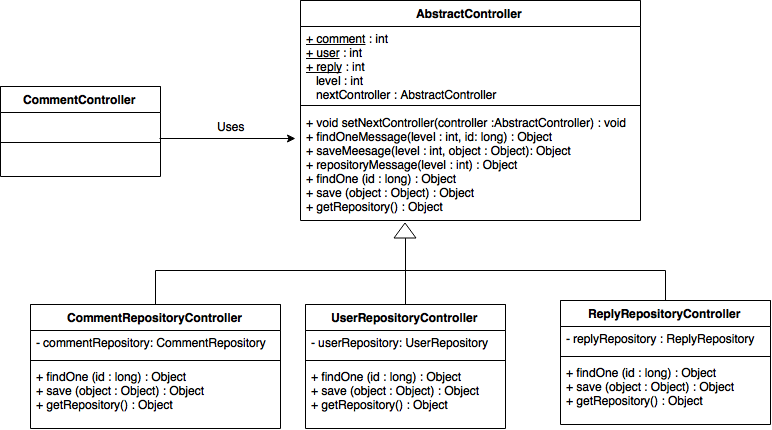
The UML graph is shown as below:



**Chains of Responsibility Pattern**

Chains Of Responsibility is used to avoid coupling together between request sender and receiver, enable multiple objects have possibility to receive request. These object will be connect into a chain and pass the request along the chain until an object handles the request.

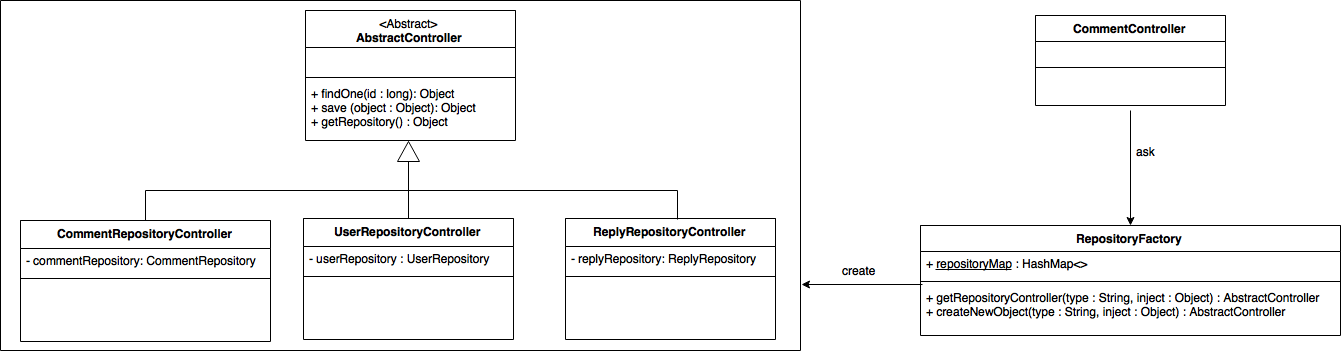
In our implementation, different repositories are different object. Thus I connected each repositories us into a chain, and handle multiple requests.



**Flight Weight Pattern**

Flyweight Pattern is mainly used to reduce the number of object created so as to reduce memory usage and improve performance.

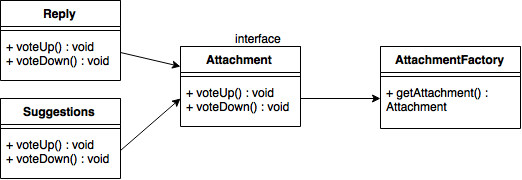
According to our design, we need to used the repository objects in different controller classes frequently. Therefore, we can use Flyweight pattern with static hashmap to store different repository object.



**Factory Pattern**

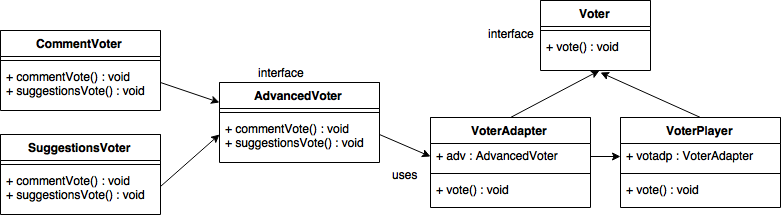
Factory pattern is one of the creational patterns that can be used to this project. It’s a design pattern for creating object using a common interface and without exposing the creation logic to the client.

We create Attachment interface containing voting functions, and making it implemented by Reply and Suggestions concrete classes. Then we create AttachmentFactory class to generate objects of the concrete classes based on parameters we send. Then when we want to create Reply or Suggestions object, we can directly use the factory and send different parameters.



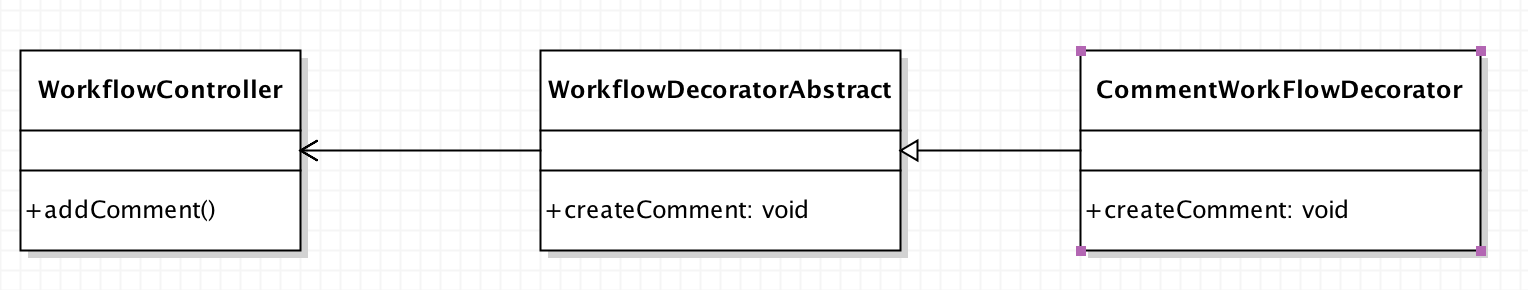
**Adapter Pattern**

Adapter pattern is one of the structural patterns used as a bridge between two incompatible interfaces. So we create Voter and AdvancedVoter interface to join functionalities, making them implemented by CommentVoter and SuggestionsVoter concrete classes. Then we create VoterAdapter class to implement Voter interface. And at last we create VoterPlayer class to implement AdvancedVoter and receiving different parameters.



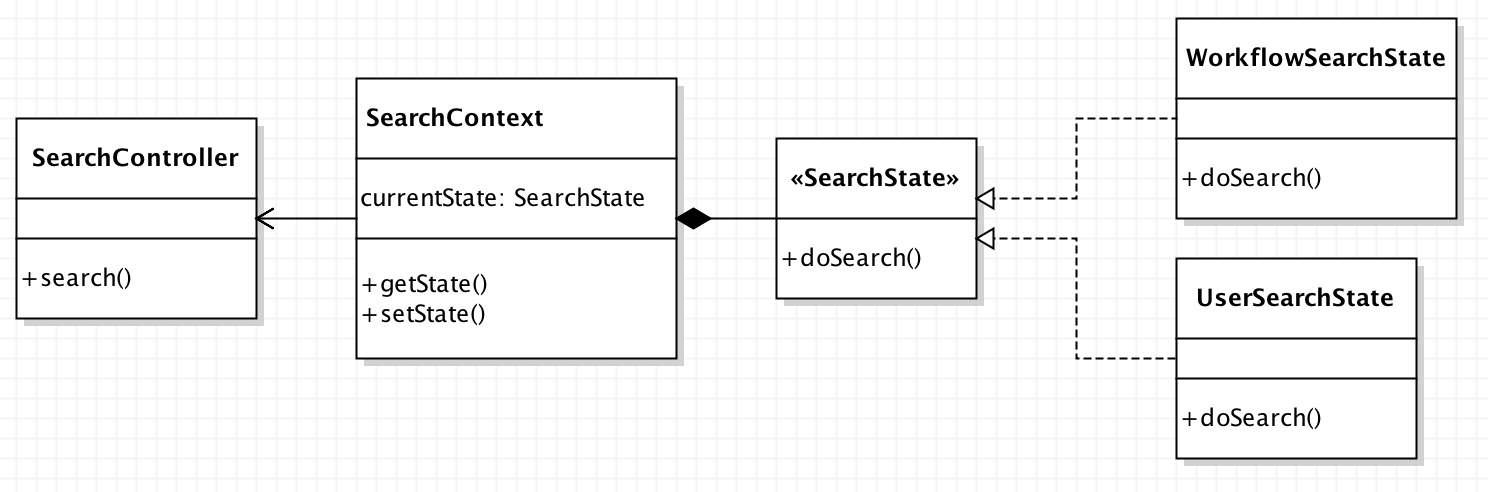
**Decorator Pattern**

The decorator pattern can be used to enhance the functionality of an existing object. Here, we use the decorator pattern to enhance the Workflow object when a Comment is added to it. The WorkflowDecoractor Abstract Class defines the functionality of a a workflow decorator, with the specific implementation in the concrete class CommentWorkflowDecorator. This, when a comment is added in the WorkflowController class, the decorator pattern takes care of it.



**State Pattern**

The state design pattern can be used to control the flow of an application based on its current state. In this application, we use it to control the Search functionality based on different states: User search or Workflow Search. The SearchContext maintains the current state, with the State and operations that can be performed in a state described by the SearchState interface and WorkflowSearchState and UserSearchState concrete classes. The search context is set and specific search performed in the SearchController.



**Criteria Pattern**

The criteria design pattern can be used to filter a set of objects based on a particular criteria. Here, we use it to filter the search and only display GMail users. This is done by sepcifiying a Criteria interface to define the function that filters a list of users, and the UserGmailEmailCriteria concrete class to implement the function that filters GMail users.

