Name: Shane Lockwood
Github link: https://github.com/slockwoo/Programming-Project-Registry

Title: Programming Project Registry

Description: A registry where individuals could post a software application description they wish they had, or could use, but are unwilling to pay for it. Other users can review these descriptions and develop the application if they desire. Features include searching for projects given a keyword, filtering/sorting projects (date, estimated difficulty, completed, votes), links to code or websites developers have implemented, voting for heavily demanded applications, and individual pages for users with data related to them such as projects posted, and applications developed. Related works include Reddit's "Programming Requests" and Upwork.

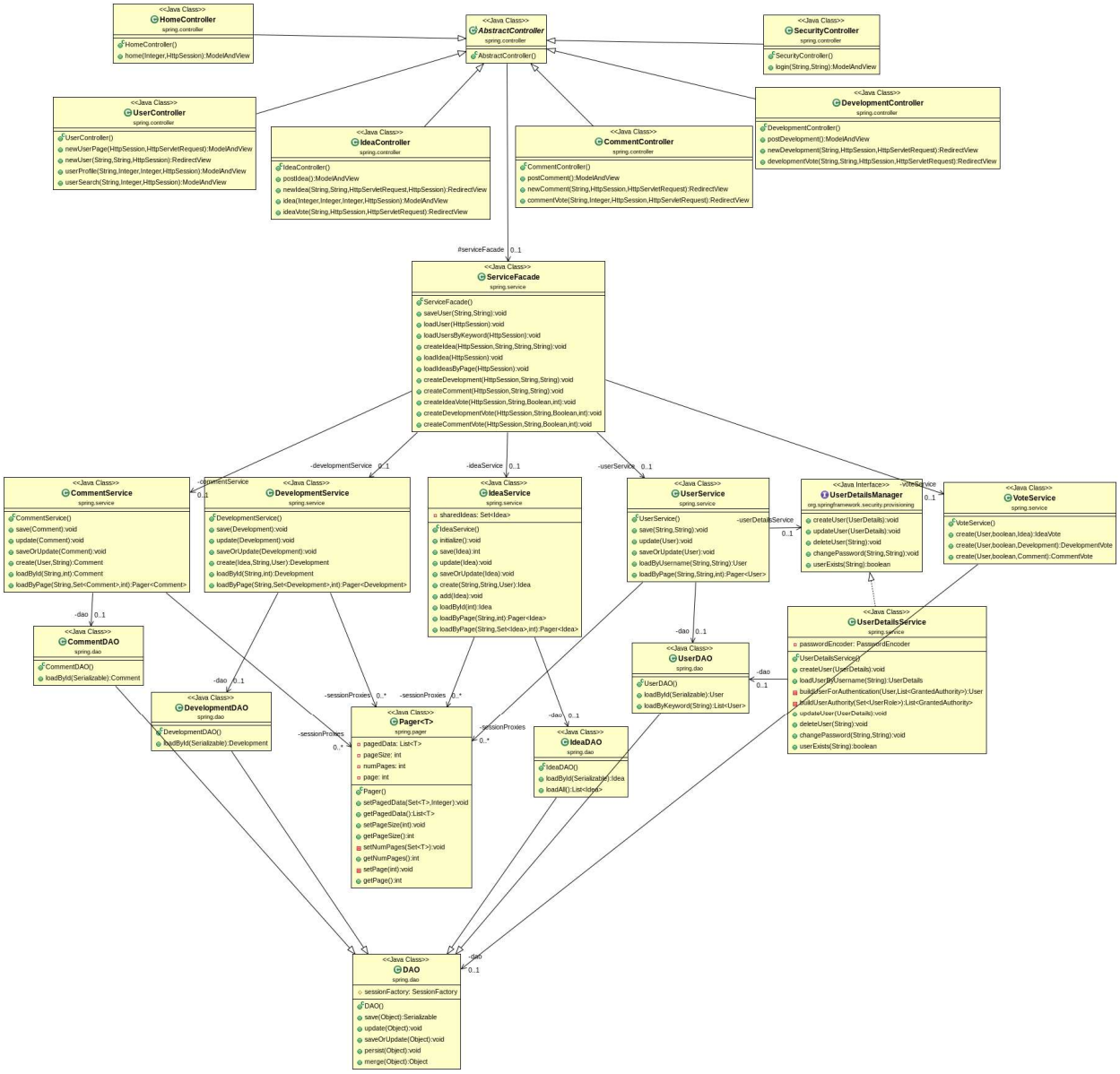
Implemented features:

ID	Title
UC-001.01	Post new ideas.
UC-002.01	View the details of an idea.
UC-002.02.b	Sort ideas by vote count.
UC-003	Vote on ideas.
UC-004	View user's profiles.
UC-005.01	Add link to relevant source code or end product.
UC-007	Vote on links.
UC-008	Add comments to ideas.
UC-009	Vote on comments.
UC-012	Create user.
UC-013	Search for users given a keyword.
UC-014	Control number of ideas displayed on home page.
UC-014.02	Transition between ideas displayed.
UC-015	Control number of links displayed on idea page.
UC-015.02	Transition between links displayed.
UC-016	Control number of comments displayed on idea page.
UC-016.02	Transition between comments displayed.
UC-017	Control number of ideas displayed on user profile page.
UC-017.02	Transition between ideas displayed.
UC-018	Control number of links displayed on user profile page.
UC-018.02	Transition between links displayed.
UC-019	Control number of users displayed on user search page.
UC-019.02	Transition between users displayed.

Not-implemented features:

ID	Description
UC-001.02	Modify ideas
UC-001.03	Delete ideas
UC-001.04	Hide ideas
UC-002.02.a	Sort ideas by date.
UC-002.03.a	Filter ideas by date.
UC-002.-3.b	Filter ideas by number of votes.
UC-006	Delete links to own ideas.
UC-010	Delete comments to own ideas.
UC-011	Delete ideas.
UC-014.01	Modify number of ideas displayed on home page.
UC-015.01	Modify number of links displayed on idea page.
UC-016.01	Modify number of comments displayed on idea page.
UC-017.01	Modify number of ideas displayed on user profile page.
UC-018.01	Modify number of links displayed on user profile page.
UC-019.01	Modify number of users displayed on user search page.

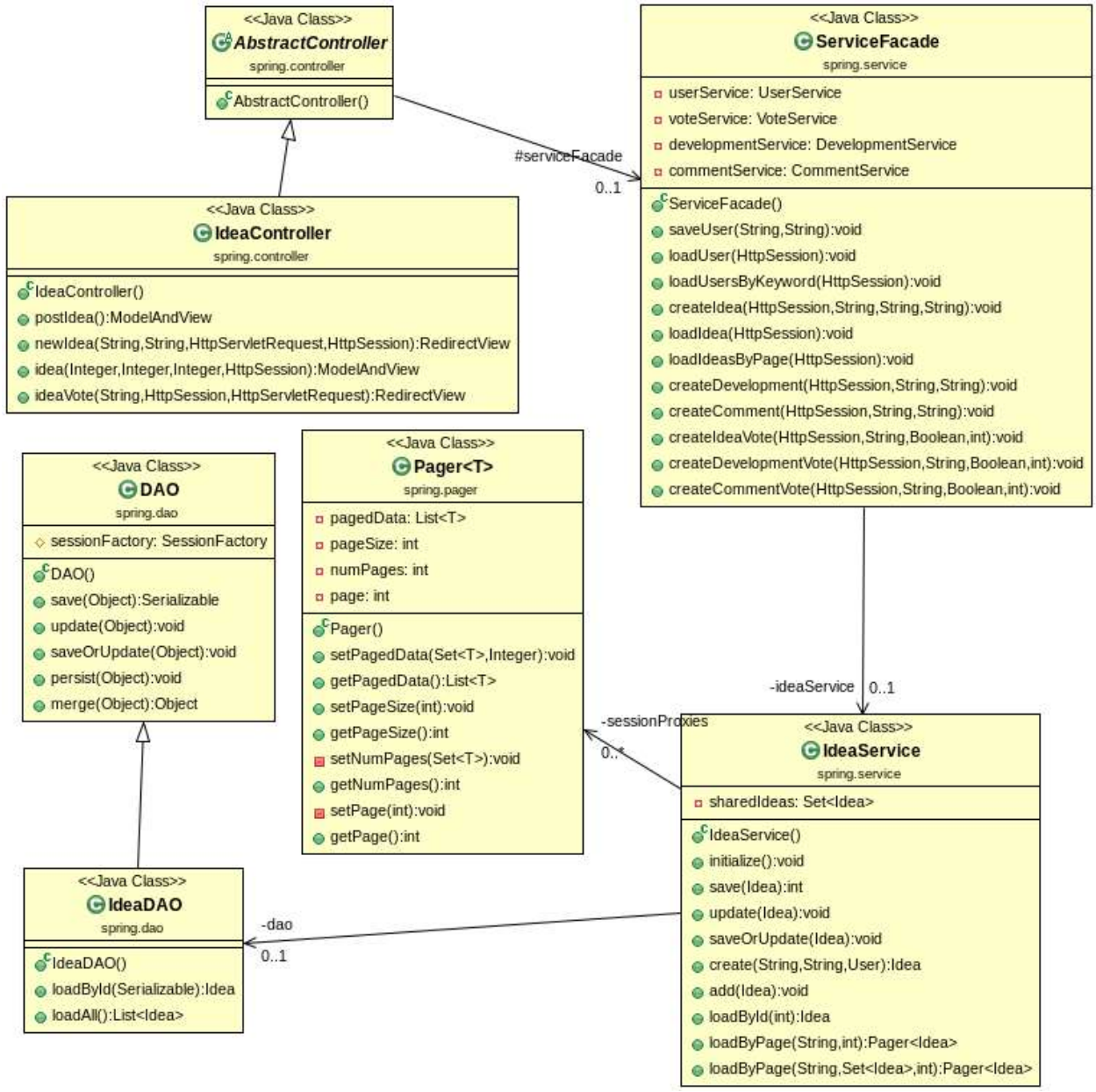
Full class diagram:



The top of the diagram shows the 6 Controllers inheriting their parent class (AbstractController). Each Controller is dedicated to a separate aspect of the application. AbstractController is composed of the Façade for the Services. The Services handle separate models and are composed of separate Data Access Objects (DAO). 4 of the Services are composed of Pager to aid with controlling the number of objects returned.

No interfaces are displayed in this diagram for simplicity. ServiceFacade is composed of 2 interfaces instead of composing IdeaService and DAO. The interfaces are appropriately displayed in the design pattern diagrams below.

Partial class diagram:

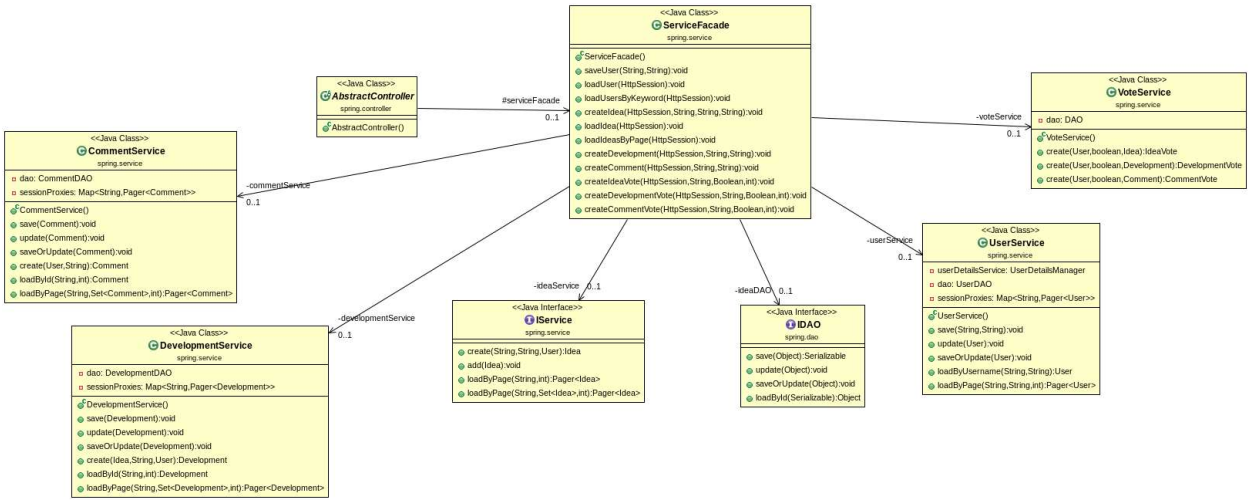


This class diagram shows a single Controller, a single Service, a single DAO, along with the shared parts of the application to tie them together (AbstractController, ServiceFacade, Pager, DAO).

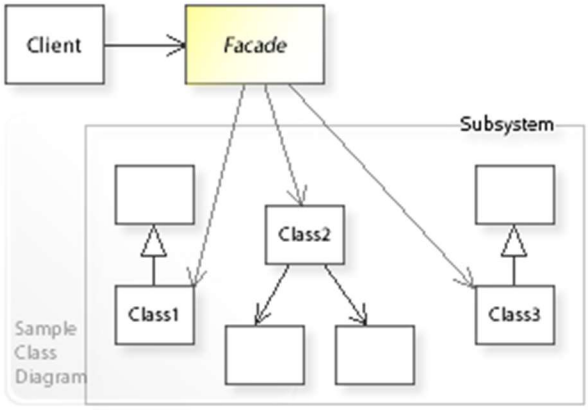
Changes from the original class diagram include separating the 1 Controller into 6 separate Controllers. An abstract Controller was integrated to handle the similarities in the Controllers. A Façade was inserted between the Controllers and Services to handle some business logic combining the Services before sending the results to the Controllers. The Builders were transitioned into Factory Methods inside the Services. A DAO class was implemented to handle all the similarities in the model specific DAOs. The model specific Proxies were combined into a generic Pager. 2 interfaces were implemented above IdeaService and DAO to accommodate the create the design patterns described below.

Design patterns implemented:

Façade:



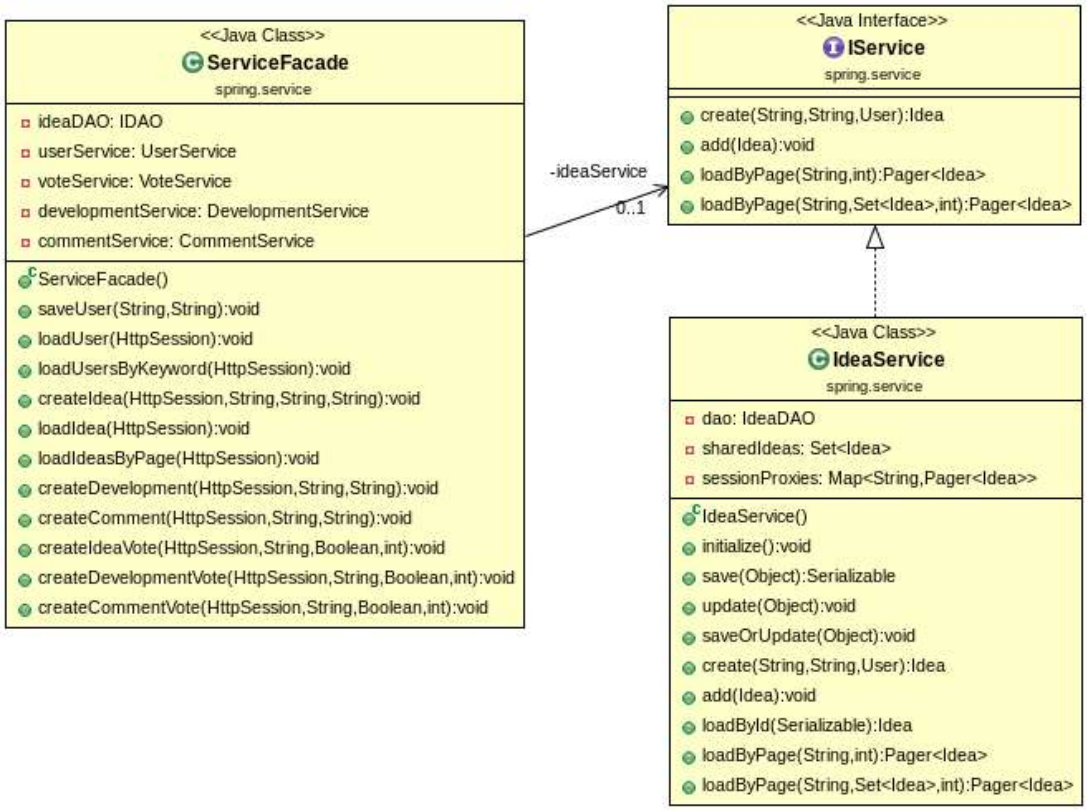
Implemented class diagram.



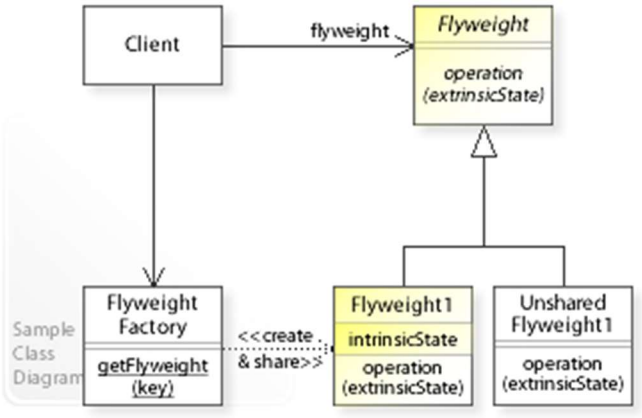
GoF class diagram.

This design pattern was chosen to unify the number of Services the Controllers needed to compose and to eliminate the need for the Services to compose each other which decreased coupling. AbstractController is Client, ServiceFacade is Façade while the other classes and interfaces are the subsystem.

Flyweight:



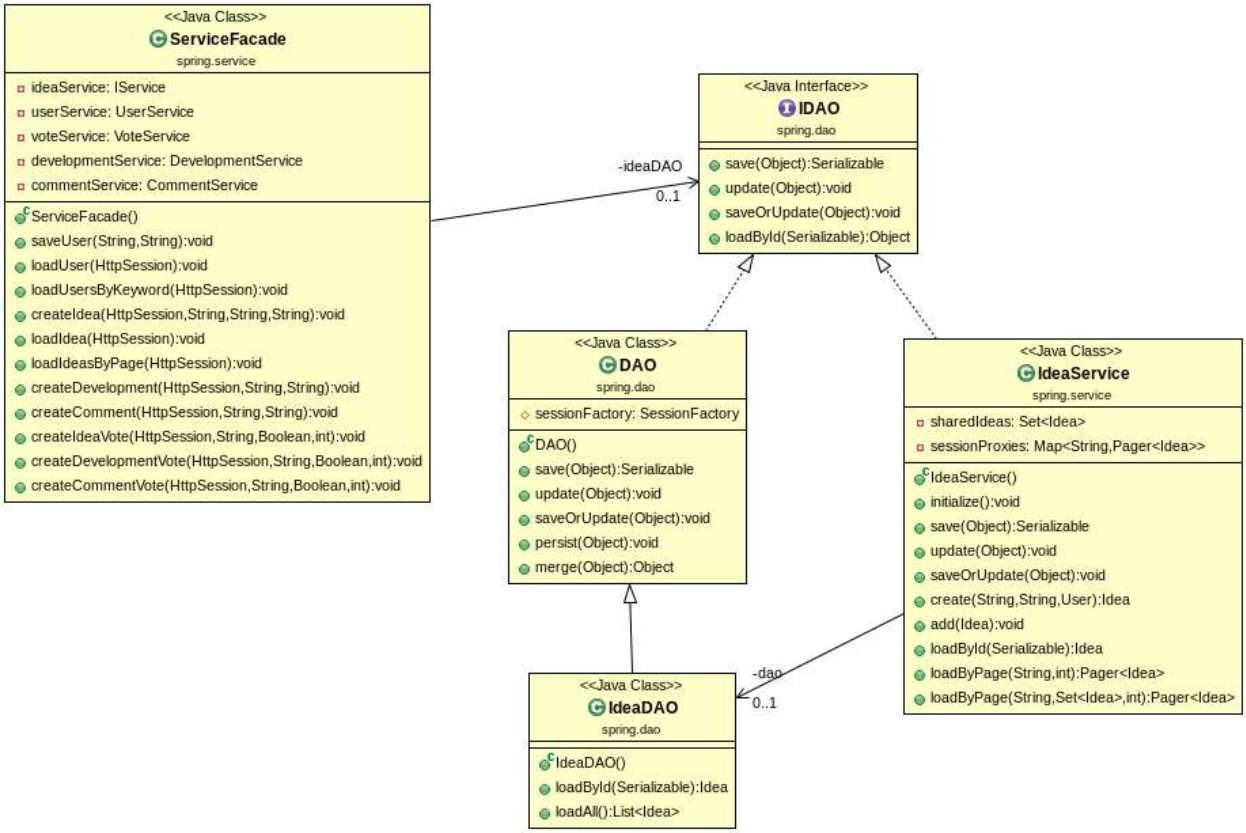
Implemented class diagram.



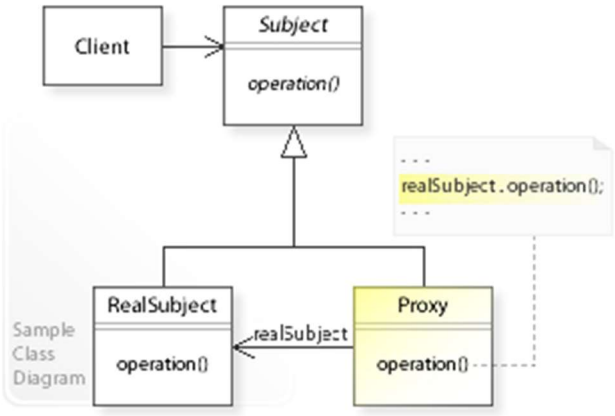
GoF class diagram.

This design pattern was chosen to minimize the number of Ideas loaded into memory since each user will load all Ideas when they first load the home page. There are likely concurrency issues but that is beyond the scope of this project. ServiceFacade is Client, IService is FlyweightFactory and sharedIdeas inside IdeaService is Flyweight.

Proxy:



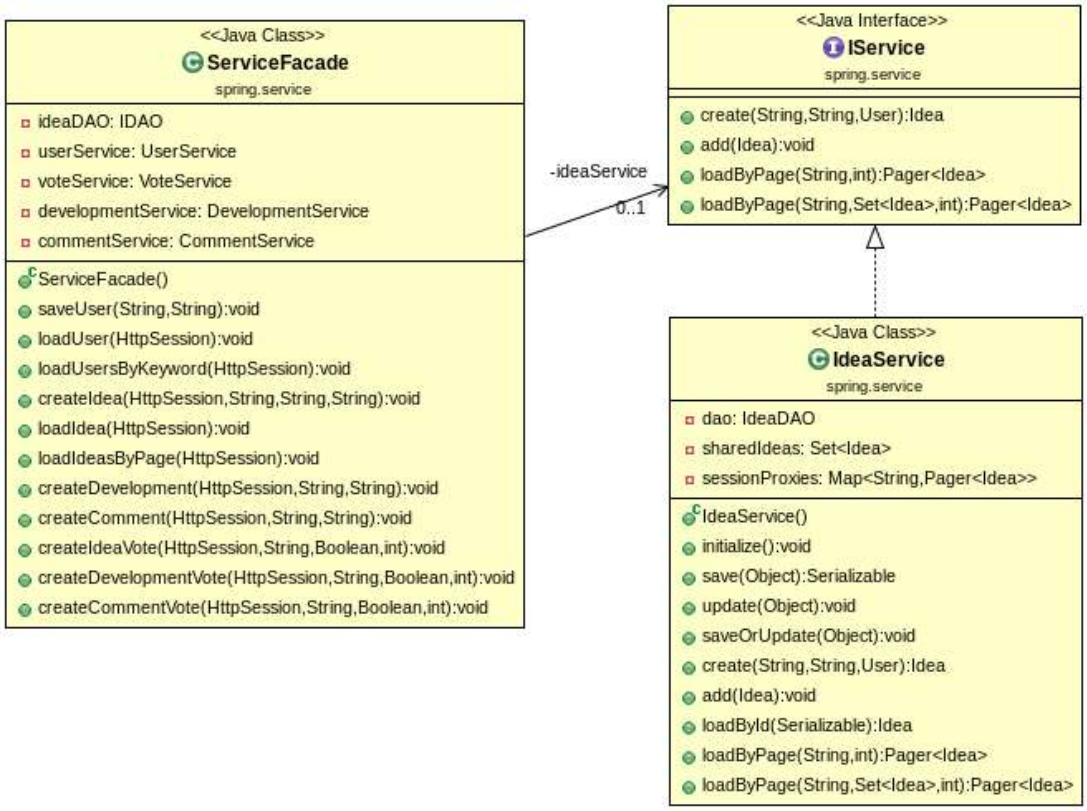
Implemented class diagram.



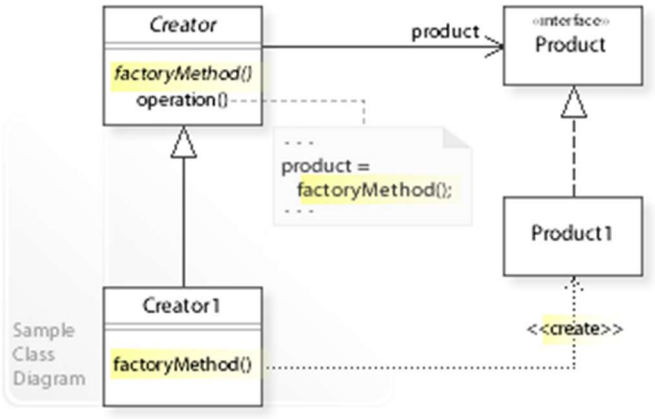
GoF class diagram.

This design pattern was chosen because every user will load all the ideas when they load the home page, and the ideas rarely change, so sharing the ideas with all the users rather than create a new set of copies for each user seemed efficient. ServiceFacade is Client, IDAO is Subject, IdeaService is Proxy, and IdeaDAO is RealSubject.

Factory Method:



Implemented class diagram.



GoF class diagram.

This design pattern was chosen to instantiate new objects in a consistent location rather than ad-hoc when needed. ServiceFacade is Client, IService is Creator, IdeaService is Creator1, and Idea is Product.

Lessons learned:
I'm not sure if planning the architecture before coding improved productivity at all, but I do know that I had a clear picture of what I was going to implement before starting. I'm confident that with more experience, architecture design and converting UML and test cases into working code will become second nature. I struggled with other obstacles such as stopping myself from optimizing too soon and knowing that code can always be refactored later, such that I never went back to plan my next steps.

Auto-generated documentation:
HTML pages were generated using Javadoc and are stored in the doc folder on Github.