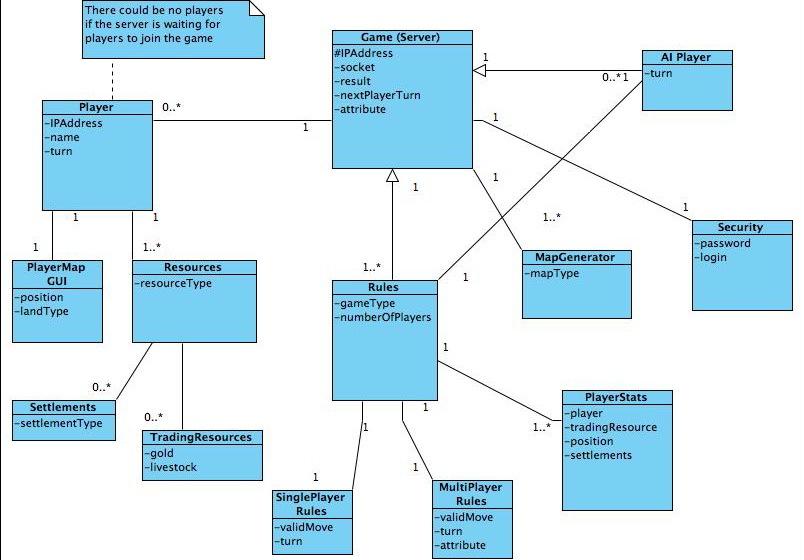
**Project Name**

*Student Name and Number (one per line)*

*Date (Due March 25th @ 5:30pm)*

You must submit this milestone through **GitHub** and have uploaded the project's source code (-2 marks if you don’t use GitHub)

# **Class Diagram of Actual System (5 points)**



1.1 Conceptual Domain Diagram - FreeCol

The Domain Diagram and conceptual architecture for the initial design of FreeCol was based on the client-server model. The reason for this was specifically to support multiplayer games that were to be played over the internet. The communication of the Player and the server was limited to a single control flow between the player and the server (the game), and knowledge was either contained on the server side or the client side. The designers of FreeCol however chose a different and perhaps unique approach of creating an additional layer called Common.

The Common layer contains aspects of the game that are shared by both the client and the server. It doesn’t contain details like GUI but the model of the game common to both client and server. The model could be described as the conceptual or imaginary aspects of the game. for example the concept of “building”. A building has a type and a builder, these aspects award the player different scores depending on different aspects of the building. Both Server and Client need this information in order to play the game. On the other hand the Server needs no information about the GUI since it is the player who needs a visual representation of the game. Creating the Common model for both Client and Server will obviously lead to much less code repetition, since both Client and Server utilize the same classes for common features of the game, this in turn reduces the size of the Server and Client side Classes. FreeCol is however a large program and the rest of this document will focus on the Server SIde aspects of the game. It is important however to get an overview of the whole game to see how the game runs.

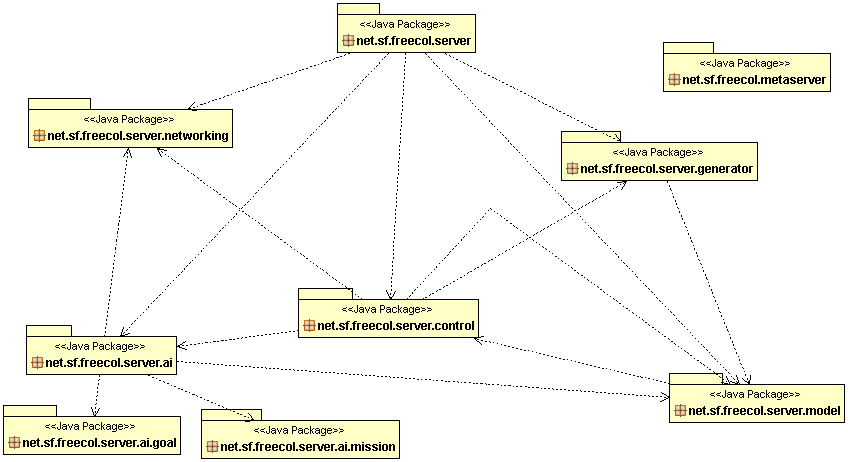
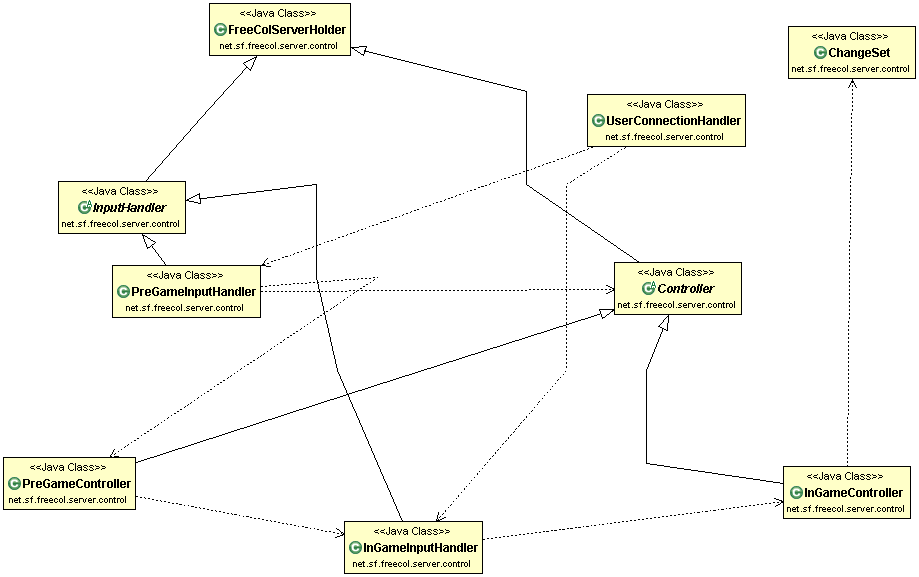


Figure 1.2 Package Diagram for the server aspects of FreeCol

The Server packages of FreeCol that this project focuses on are shown in figure 1.2 Package Diagram, though we will not include the AI packages, but focus on the model, control, networking and generator packages. Each package as we can see from the diagram above has a relationship with the other packages but performing a specific task related to game play.

Control Package (net.sf.freecol.server.control):-



Description:-

In the Server. Control package Contains the classes responsible for the control of the game and have nine classes i.e. ChangeSet, Controller, FreeColServerHolder, InGameController, InGameInputHandler, InputHandler, PreGameController, PreGameInputHandler, UserConnectionHandler) they are connected with some associations. These associations are described as follows

* InputHandler and Controller are subclasses of FreeColServerHolder
* PreGameInputHandler and InGameInputHandler are subclasses of InputHandler
* PreGameController and InGameController are subclasses of Controller
* UserConnectionHandler Is depends on the PreGameInputHandler and InGameInputHandler
* PreGameInputHandler is depends on the Controller and PreGameController
* PreGameController Depends on InGameInputHandler which depends on InGameController which further depends on the ChangeSet.

Description of classes:-

InputHandler: - Handles the network messages.

Controller: - The control object that is responsible for making changes to the internal model and for communicating with the clients.

FreeColServerHolder: - This base class provides thread-safe access to a [FreeColServer](http://www.freecol.org/javadoc/net/sf/freecol/server/FreeColServer.html) for several subclasses.

PreGameInputHandler: - Handles the network messages that arrive before the game starts.

InGameInputHandler: - Handles the network messages that arrive while [in game](http://www.freecol.org/javadoc/net/sf/freecol/server/FreeColServer.GameState.html#IN_GAME).

PreGameController: - The control object that is responsible for setting parameters and starting a new game.

InGameController: - The main server controller.

UserConnectionHandler: - Handles a new client connection.

ChangeSet: - Changes to be sent to the client.

In the Conceptual Domain Model We think there is some kind of controller that control whole server side activity these include making connection, checking rules of game and also check all networking activities which was described by the rule class in the conceptual model. Whereas in the actual class diagram these entire work is done by the Controller class in the server. Control Package. This concludes that whatever we thought in conceptual Domain model is somehow is similar to our actual class diagram.

We all should know what we are working on for Sunday but get some stuff done for Friday/Saturday so Sunday we can review what has been done

Hardev upload the package diagram for the server..

Ede - client common server - Overall Description of the Game.Done!!

Hardev Mishal - Control Package

Navrang - networking and Generator packages

Ronak - Model

1 to 2 pages of text -> Describe the diagram and the relationships between the classes. Which conceptual classes map to which actual classes? Is there a discrepancy between the concepts and the actual classes? What does it mean to the architecture of the system? Describe any reverse engineering tools used (e.g., Diver, ObjectAid UML Explorer, Enterprise Architect - Sparx Systems, ArgoUML).

In maximum of one page, for two classes and the relationship between them: Copy-and-paste the class, method, and attributes **declarations** (and anything else that is necessary) directly from the source code. Do not include code that is unnecessary (be selective, you will lose marks for large dumps of source code.)

# **Code Smell and Possible Refactoring (5 marks)**

Be ambitious! You will get no marks if all you suggest is a rename of a class. (Note: in the M4 you will implement a small part of the refactoring you suggest here)

Describe the code smell(s) and how you will combine together a series of refactorings to fix the smells. Describe how the refactorings are interrelated and how they correct the problems you identified in the above. For example, “First, I moved the methods X and fields Y and Z, to increase cohesion and to reduce feature envy. Second I renamed the class to reflect its new limited responsibilities. Third I removed the coupling to …”

1 to 2 pages of text + 1 to 2 UML diagrams (diagrams should be at most one additional page).

In maximum of one page, for one of the refactorings you suggest: Copy-and-paste the class, method, and attributes **declarations** (and anything else that is necessary) directly from the source code. Do not include code that is unnecessary (be selective, you will lose marks for large dumps of source code.)