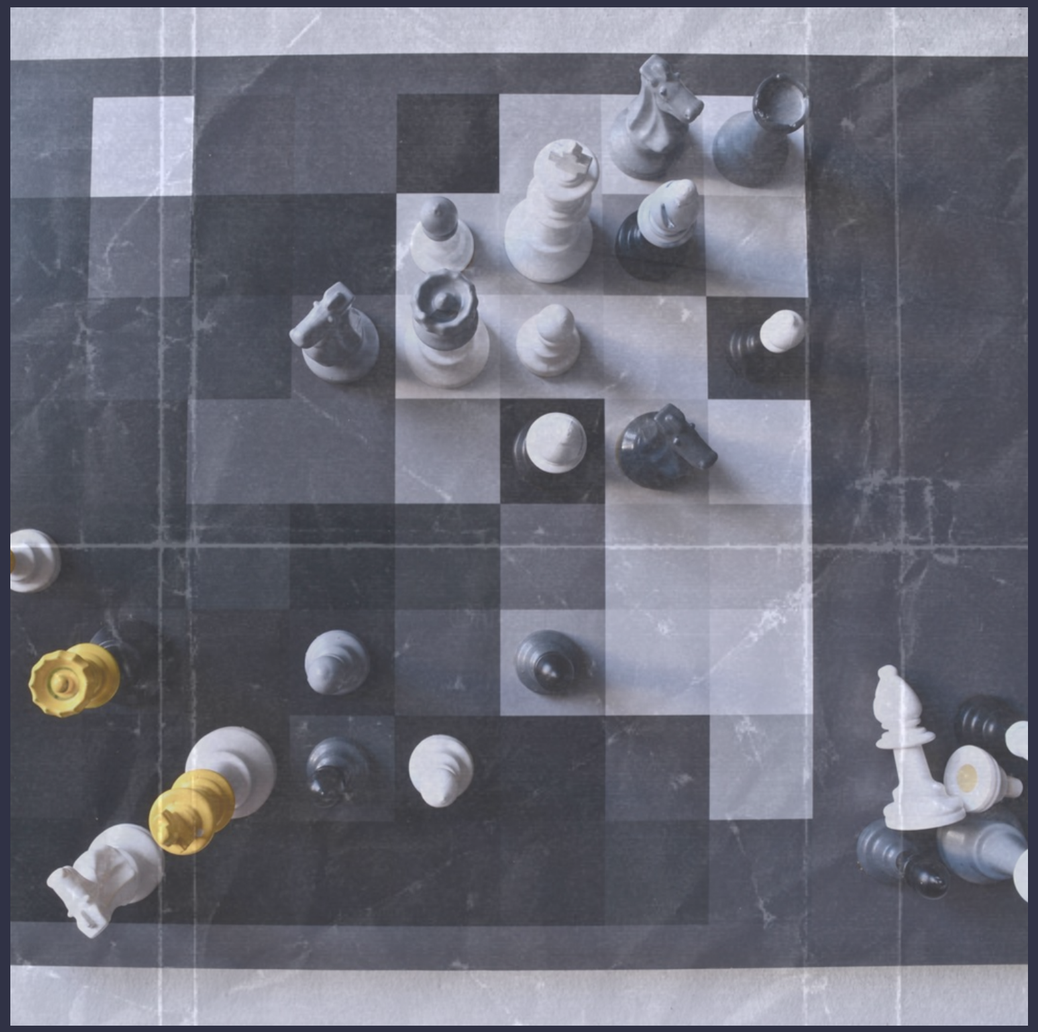


Chess Project

by Adilkhan Yerbolat,
Dimash Yeskendir,
Nurassyl Raimbek

Purpose

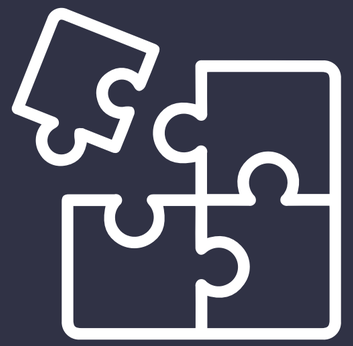
The main purpose of the project is to discover the usage of design patterns in solving problems in developing network based games and programs.

Design Patterns

We had a lot of problems during the development of the project. And using patterns helped us to quickly find and solve them so that it did not interfere with the working part of the code.

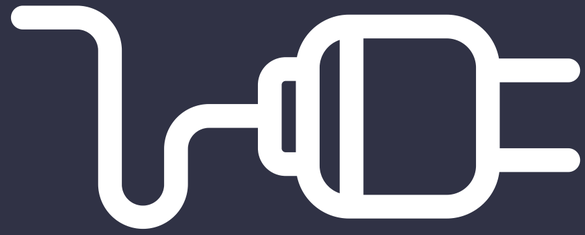
We used these patterns:

Strategy



Factory Method

Adapter



Strategy pattern

Chess is a game where set of moves depends directly on the type of the piece. And in some circumstances this behavior can change. This suggested us the structure of game pieces' hierarchy.

The main idea was to define a common class that should contain all piece related data and some useful methods. This class called "Piece" was also the container for a concrete strategy class which implemented the interface for evalutaing moves of the piece. For example Pawn had PawnStrategy class and so on.

Strategy pattern

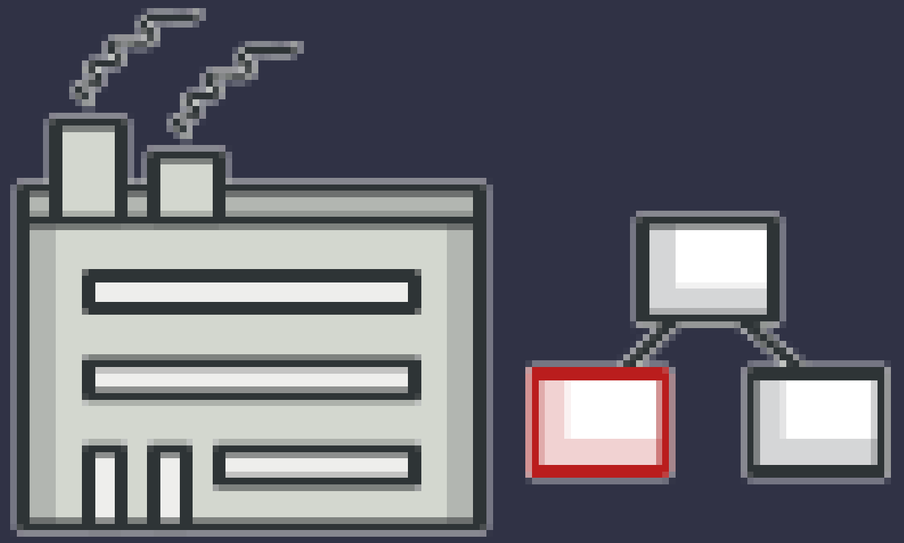
By using this approch we could sufficintly simplify the process of evaluating moves for pieces.
With our method when we were needed to estimate possible moves for each side we just used a 8x8 array of Pieces representing the board and by traversing through it invoked evaluateMoves() method for each piece

Thus we defined a convinient interface for and incapsulated all the details behind a single class.

Factory method

Defining suitable interfarce for calculating moves it’s crucial in such kind of projects.

But in order to create instance of pieces the client should know about inner structure of the class. To hide this unnecessary information from potential users and set up a convenient way to access pieces objects without difficulties some additional measures should be taken. Thus we have decided to use factory method pattern.

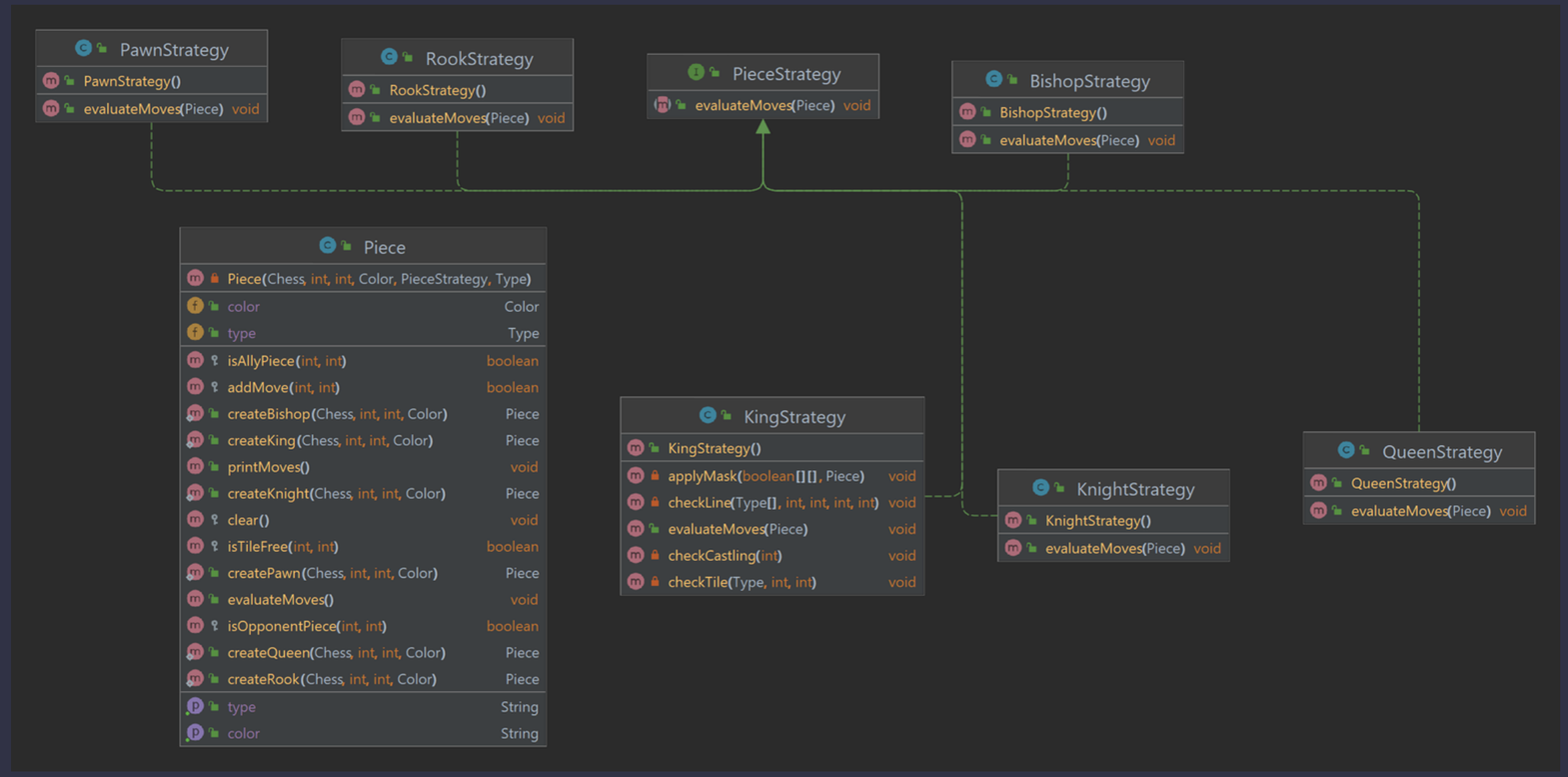


Factory method

To address this issue we have defined several factory methods in “Piece” class. Each of them creates an instance of a specified type of chess piece.

For example to get a knight piece the client just need to invoke the corresponding static method from the Piece class. Thus we can hide information about inner structure of a piece behind its factory method which will construct and return a ready one to the user class without discovering details about its structure.

As result we solved the problems of getting instances and estimating moves of different types of pieces by using just two suitiable patterns



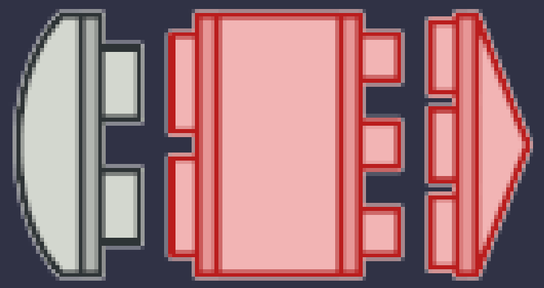
Adapter

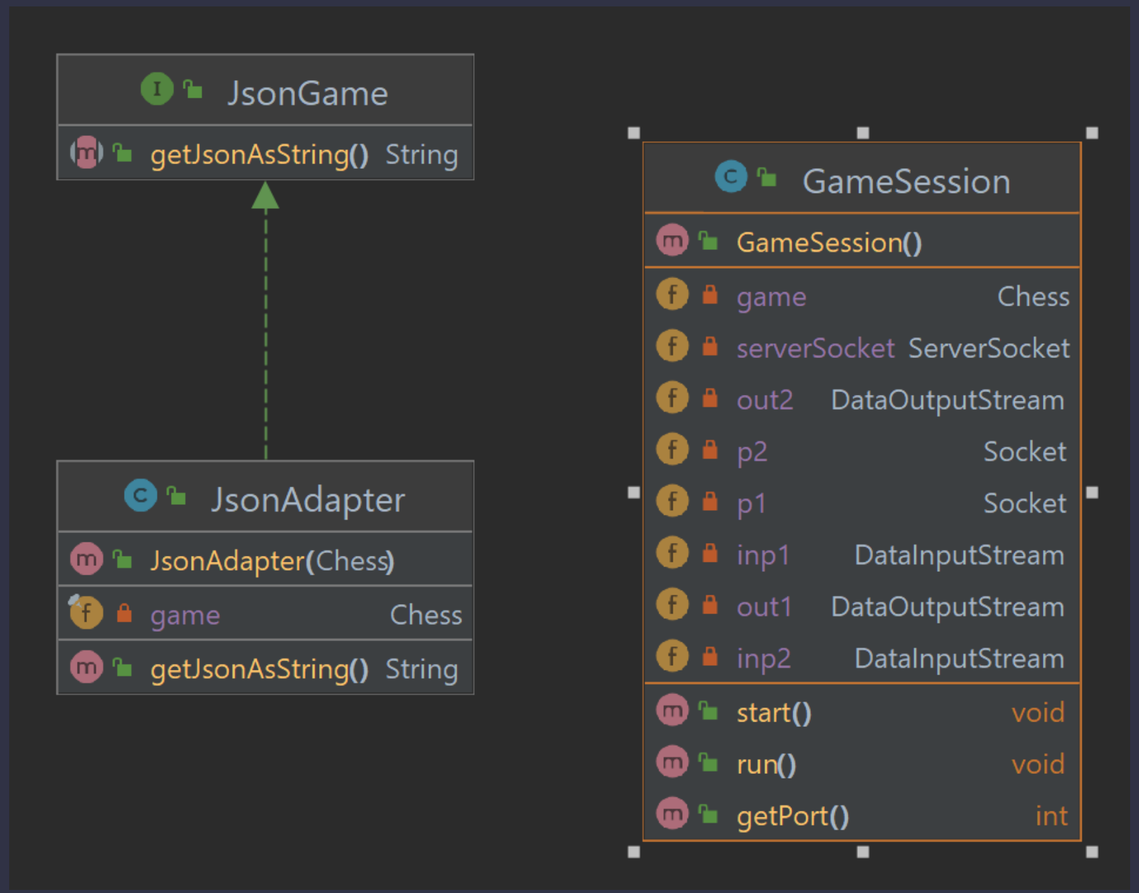
Since we are developing game where two players can interact through network, a simple and versatile way of transmitting information about the state of the game should be developed.

After a few considerations it has been decided that transmitting raw objects through the network is a very clumsy and limiting way. Thus we have decided to transform data and transmit it as the text.
This decision has been influenced by two factors.
First its quite easy to send text data by using common tools of java language.
Second this way is platform independent. Receiving side could be written on the other platform and programming language and hence making our project much more versatile and friendly.

Adapter

It has been decided to transform the game state into the json format as it’s quite popular way of communication between programs in text format. It’s has a good performance and readable by human.
So the Adapter pattern was used in order to transmit the game state information through the network.
And thus we have solved a problem of interaction between participants of the session.




JsonGame
getJsonAsString() String
JsonAdapter
JsonAdapter(Chess)
game Chess
getJsonAsString() String
GameSession
GameSession()
game Chess
serverSocket ServerSocket
out2 DataOutputStream
p2 Socket
p1 Socket
inp1 DataInputStream
out1 DataOutputStream
inp2 DataInputStream
start() void
run() void
getPort() int