GRASP PATTERNS AND ITS TYPES

PRESENTED BY:
SAIRA BANO
RAMSHA GHAFFAR
SYED HASSAN ALI HASHMI

DEFINITION

- GRASP or General Responsibility Assignment Software Principles help guide object-oriented design by clearly outlining WHO does WHAT. object or class is responsible for what action or role? GRASP also helps us define how classes work with one another. The key point of GRASP is to have efficient, clean, understandable code

PATTERNS

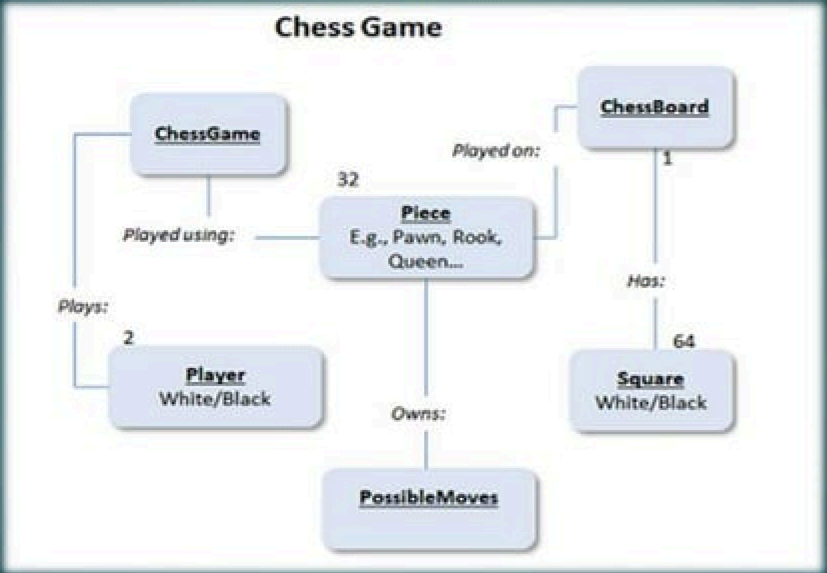
- In OO design, a pattern is a named description of a problem and solution that can be applied to new contexts; ideally, a pattern advises us on how to apply its solution in varying circumstances and considers the forces and trade-offs. Many patterns, given a specific category of problem, guide the assignment of responsibilities to objects.

PRINCIPLES

Within GRASP there are nine principles that we want to cover. They are:

- Creator
- Controller
- Information Expert
- Low Coupling
- High Cohesion
- Indirection
- Polymorphism
- Protected Variations
- Pure Fabrication.

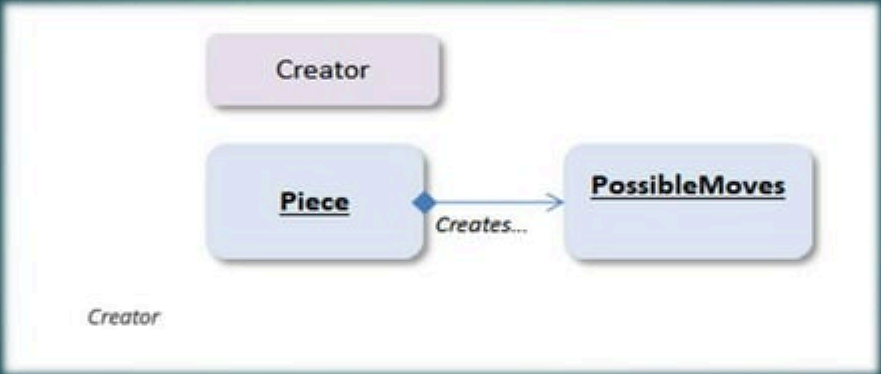
We'll be talking about a chess game and the various responsibilities and relationships between the objects and classes within the game.



CREATOR

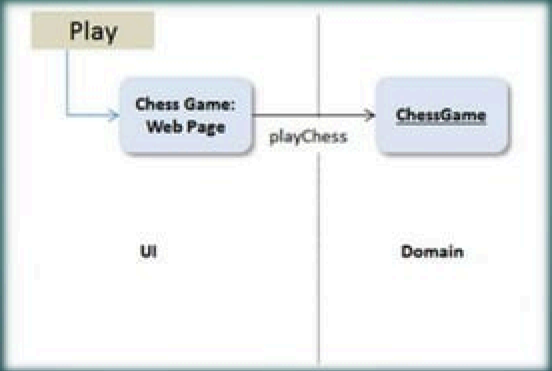
- The Creator defines WHO instantiates WHAT object. In object-oriented design lingo, we need to ask the question of who creates an object A. The solution is that we give class B the role of instantiating (creating an instance of) a class A if:
- B contains A
- B uses most of A's features
- B can initialize A

So far this doesn't really help us understand how this works. Let's use a real-world example of a chess game. A chess game includes 2 players, 32 pieces (16 per player) and a game board with 64 squares.



CONTROLLER

- In our chess example, the end user is going to interact with our program through a user interface (UI). The **Controller** is the FIRST object to receive a command from the UI. In our case, when the user presses Play, the first object that should be triggered is the Chess Game.



INFORMATION EXPERT

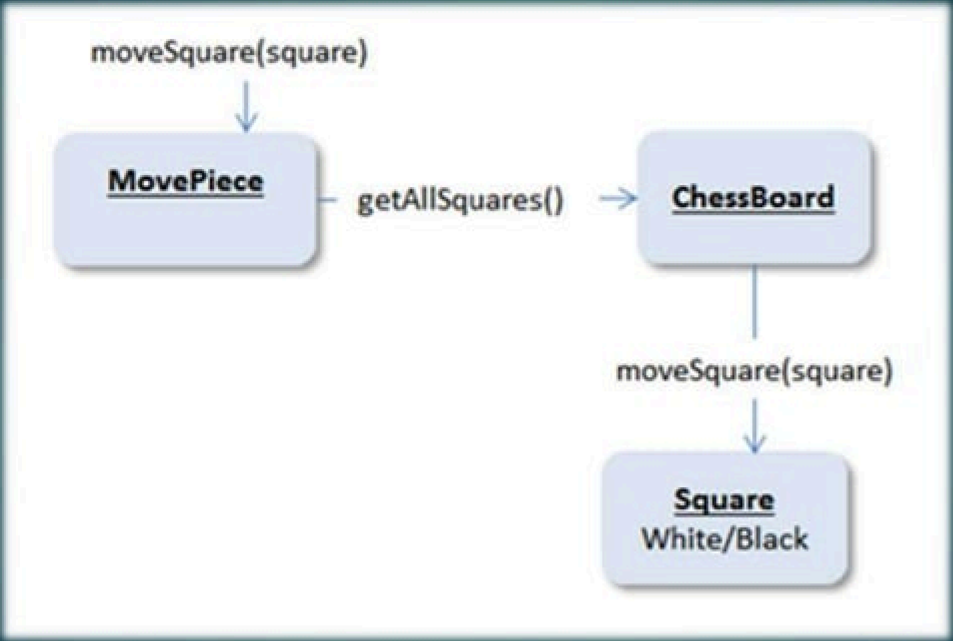
- Information expert is a principle used to determine where to delegate responsibilities. These responsibilities include methods, computed fields, and so on.
- Using the principle of information expert, a general approach to assigning responsibilities is to look at a given responsibility, determine the information needed to fulfill it, and then determine where that information is stored.
- Information expert will lead to placing the responsibility on the class with the most information required to fulfill it

INFORMATION EXPERT

- The **Information Expert** pattern states that we need to assign responsibilities to the right expert. Is the game board itself the expert on how pieces can move or are the pieces themselves the experts at their moves? In the case of the chess board, the piece is the expert on the possible move options for that piece.

LOW COUPLING

- Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. Low coupling is an evaluative pattern that dictates how to assign responsibilities to support
- lower dependency between the classes,
- change in one class having lower impact on other classes,
- higher reuse potential.


moveSquare(square)
MovePiece
getAllSquares()
ChessBoard
moveSquare(square)
Square
White/Black

LOW COUPLING

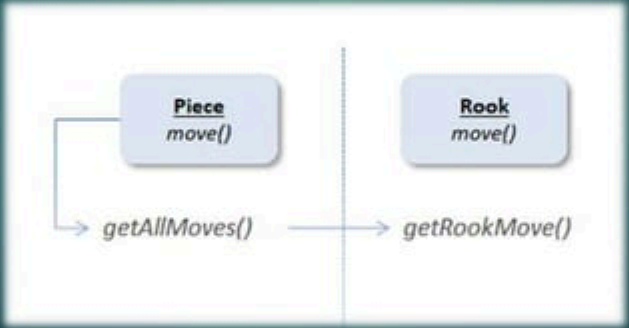
- **Low Coupling** can be described as following the path of least resistance. Coupling is a measure of how much objects are tied to one another. We can follow the information expert for the lowest level of coupling. So, to get the moves available to a piece, we start with the information expert, and not some other class. In the chess game, the MovePiece class needs to get information from the board and the place it intends to move to. We can couple all of this together in one flow:

HIGH COHESION

- It is important to have code that is clean. Objects need to be manageable, easy to maintain and have clearly-stated properties and objectives. This is High Cohesion which includes defined purposes of classes, ability to reuse code, and keeping responsibility to one unit. High Cohesion, Low Coupling, and clearly defined responsibilities go together. To achieve High Cohesion, a class should have ONE job. A game piece should move across the board. It should not need to setup the board or define moves for other players.

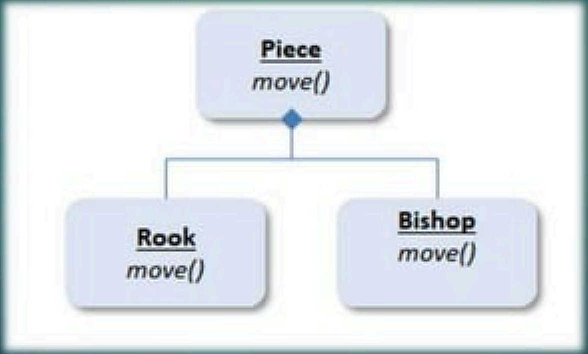
INDIRECTION

- In order to support lower coupling between objects, we look for Indirection, that is creating an intersection object between two or more objects so they aren't connected to each other. Indirection and Polymorphism go hand in hand.



POLYMORPHISM

- This sounds like a science fiction term, but **Polymorphism** really means that one thing can be performed in different ways. All chess pieces can move, but each has a special way of moving.



PROTECTED VARIATION

- The protected variations pattern protects elements from the variations on other elements (objects, systems, subsystems) by wrapping the focus of instability with an interface and using polymorphism to create various implementations of this interface.

PURE FABRICATION

- A pure fabrication is a class that does not represent a concept in the problem domain, specially made up to achieve low coupling, high cohesion, and the reuse potential thereof derived (when a solution presented by the *information expert* pattern does not). This kind of class is called a "service" in domain-driven design.

OVERVIEW

Informational Expert	Assign a responsibility to the class that has the information needed to fulfill it..
Creator	Assign class B the responsibility to create an instance class A if one of these is true (the more the better): • B "contains" or compositely aggregates A. • B records A. • B closely uses A. • B has the initializing data for A that will be passed A when it is created. Thus B is an Expert with to creating A.
Controller	Assign the responsibility to a class representing one of the following choices: • Major subsystem classes • A use case scenario classes within which the system event occurs.
Low Coupling	Assign a responsibility so that coupling remains low.
High Cohesion	Assign a responsibility so that cohesion remains high.

OVERVIEW

Polymorphism	The same name operations (methods) in the difference classes is defined. And assign a responsibility to the class the class that the behavior is changed.
Pure Fabrication	Define a class for convenience sake that doesn't express the concept of the problem area at all.
Indirection	Assign the responsibility to an intermediate object to mediate between other components or services, so that they are not directly coupled.
Protected Variations	Assign responsibility to create a stable interface around unstable or predictably variable subsystem or

Thank you...