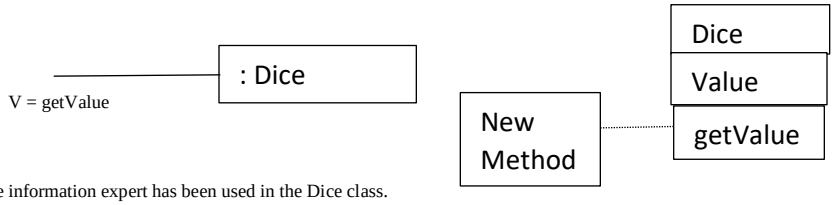
1. **Information Expert**

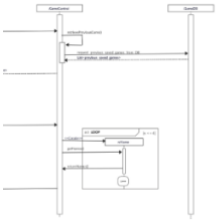
By recognizing Dice as the Information Expert for the value, we can identify a new method.



The information expert has been used in the Dice class.

2. **Creator**

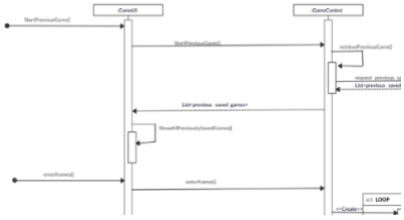
GameControl is a good candidate for having the responsibility to create Name.



This pattern is found in the enterName class in the Choose Players for the game use case.

3. **Controller**

The candidates for control of enterNames are: GameUI (facade), or GameControl(session).



Separates application logic from interface layer. [so can re-use or plug-in interface] [can separate from entity (domain) objects too]. This pattern is found in the Choose Players for the game use case.

4. **Polymorphism**

Use polymorphic operations to assign the responsibility for the different behaviors to the types themselves.

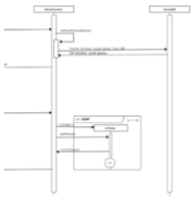
Example of the code:

```
Random rand = new Random();

  graph board = new graph();
```

This code is found in the main_game file.

5. **Low Coupling**



The pattern is used in multiple places. But one of it would in the Choose Players for the Game use case.

easier re-use of code. easier maintenance.

6. **Pure Fabrication**

Pure Fabrication could have been used instead of Information Expert used in the Dice Class. solutions offered by Information Expert are not appropriate.

An artificial class which does not represent a problem domain concept but does support low cohesion and high coupling hence a ‘pure fabrication’