Refactoring operations - Build 3

Potential Targets

1. Moved phase execution from gameplay to game
2. Refactored views
3. Refactored player to include strategy
4. Moved views to new package

1. Moved phase execution from gameplay to game
Refactoring technique - Move Method

Moved the methods - *executeStartupPhase, executeReinforcePhase, executeAttackPhase , executeFortificationPhase* from GamePlay to the *Game* class. This helps the concrete class implementation to be hidden.
Abstraction is achieved which allows us to modify the implementation under the hood without any side-effects.

2. Refactored views
Refactoring technique - Extract method

Separated and moved code for registering views to new methods (registerGame(), registerPlayers(), registerCountries()) based on the the entity such entity such as Game, Players and Countries.

This will enhance code readability and to add new view under any entity in future without affecting other entity views.

3. Refactored Player to include strategy
Refactoring technique - Replace type code with strategy

Phases(Reinforce, Attack and Fortify) are created as different strategies, which achieves polymorphism. It allows a method to be swapped out at runtime by any other method (*strategy*) without the client realizing it.

Another advantage is that if we need to add a new phase , all we need to do is to add a new strategy without altering the existing code.

4. Moved views to new package
Refactoring technique - Move class

According to the architecture , it was more relevant to move the views into a self-defined Views package from the interactor package.
This helps to remove complex package level dependencies and make it easier to find and re-use classes and also to add new views.