Updated justification for the building action, considering the scenario where the active player has the Demeter card and the opposing player has no card:

Implementation of Valid Build Determination with Demeter God Card: Objects and Methods Involved:

- GameController: Receives input and invokes build actions on the Player.
- Player: Responsible for selecting a worker and executing build actions through the worker.
- Worker: Performs the build action on a target Cell.
- Cell: Contains the isValidBuild(Cell workerCell, Cell targetCell) method to determine if the build action on the target cell is valid.
- Grid: Provides the getAdjacentCells(Cell cell) method to retrieve adjacent cells.
- Tower: Responsible for managing the tower structure on a cell, with methods addLevel() and addDome() to modify the tower.
- GodCard (Interface): Defines the common behavior and methods for all god cards.
- DemeterGodCard: Implements the GodCard interface, providing the specific rules and behavior for the Demeter god card.

Process:

- GameController receives a build command, including coordinates for the build location.
- GameController delegates the action to the current Player, identifying the selected Worker based on the game's state.
- The Player instructs the selected Worker to perform the build action on the target Cell.
- The Worker checks with the Cell if the build is valid through Cell.isValidBuild().
- Inside isValidBuild(), the following checks are performed:
  o Check if the workerCell and targetCell are not null. o Check if the targetCell is not occupied by a worker using targetCell.hasWorker().
  o Check if the targetCell does not already have a dome using targetCell.getTower().hasDome().
  o Retrieve the adjacent cells of the workerCell using grid.getAdjacentCells(workerCell).
  o Check if the targetCell is in the list of adjacent cells.
- If all checks pass, the build is considered valid.
- Upon validation, the Cell updates its Tower, either by adding a level using addLevel() or a dome using addDome().
- If the active player has the Demeter god card (currentPlayer.getGodCard()), additional steps are performed:
  o After the first build, the DemeterGodCard's firstBuildCell is set to the target cell. o If the DemeterGodCard's firstBuildCell is not null (indicating a previous build), the canBuild() method of the DemeterGodCard is called to determine if a second build is allowed on the target cell.
  o If the second build is allowed (target cell is different from the first build cell), the build action is performed, and the DemeterGodCard's build state is reset.
  o If the second build is not allowed (target cell is the same as the first build cell), an appropriate error message is provided, and the build action is not performed.

Justification and Design Principles: Single Responsibility Principle:

- Each class has a clear, distinct responsibility. Cell manages cell-related rules, Tower manages structural changes, Worker initiates the action, and GodCard defines the specific behavior for each god card.
- The DemeterGodCard class encapsulates the specific rules and behavior for the Demeter god card, adhering to the Single Responsibility Principle.

Encapsulation and Information Hiding:

- The internal state of each tower is hidden and managed through public methods like addLevel() and addDome().
- The DemeterGodCard's internal state (firstBuildCell) is encapsulated and managed through its methods, ensuring the integrity of the god card's rules.

Low Coupling:

- Classes are designed to minimize dependencies. Worker doesn't directly manipulate the Tower; it goes through Cell, adhering to the game's rules of interaction.
- The GodCard interface allows for loose coupling between the Player and the specific god card implementation, enabling easy switching and addition of new god cards.

Open-Closed Principle:

- The GodCard interface and the use of composition allow for the extension of god card functionality without modifying the existing code.
- New god cards can be added by implementing the GodCard interface, without affecting the existing classes and game logic.

Law of Demeter:

- By having GameController initiate actions and pass them to the closest neighbor (Player), the system adheres to the Law of Demeter, with each part interacting with only closely related objects.
- The Player interacts with the Worker, and the Worker interacts with the Cell, maintaining a clear chain of responsibility and minimizing unnecessary dependencies.

Alternatives and Trade-offs: God Card Validation in Cell:

- Considered moving the god card validation logic inside the Cell class, specifically in the isValidBuild() method.
- Rejected because it would violate the Single Responsibility Principle, as the Cell class should focus on cell-related rules and not be concerned with god card-specific logic.
- Keeping the god card validation in the Player and GodCard classes maintains a clear separation of concerns and promotes modularity.

Separate Build Method for God Cards:

- Considered creating a separate build method specifically for god cards, such as performGodCardBuild().
- Decided against it to maintain consistency and simplicity in the build action flow, as the existing playerBuild() method can handle both regular and god card-specific builds.
- The playerBuild() method checks if the active player has a god card and invokes the necessary god card methods accordingly, keeping the build logic centralized and easier to understand.

Conclusion: The chosen design for the building action in Santorini, considering the Demeter god card, adheres to essential object-oriented principles such as Single Responsibility, Encapsulation, Low Coupling, Open-Closed Principle, and the Law of Demeter. It effectively distributes responsibilities among the GameController, Player, Worker, Cell, Grid, Tower, and GodCard classes.

The introduction of the GodCard interface and the DemeterGodCard class allows for the seamless integration of god card-specific behavior into the build action flow. The DemeterGodCard encapsulates the logic for allowing a second build on a different cell, maintaining the integrity of the god card's rules.

The GameController and Player classes orchestrate the build action, delegating responsibilities to the appropriate objects and ensuring that the game's rules and god card-specific behavior are enforced. The use of composition and interfaces promotes flexibility, extensibility, and modularity in the design, enabling the easy addition of new god cards without modifying the existing codebase.

Deliverable 6 : object-level interaction diagram