Assignment 3 Design Rationale

Requirement 2: The Static Factory

Design Goal:

The design goal for this assignment is to enable the player (intern) to interact with a humanoid figure within a factory's spaceship parking lot to sell collected items.

Design Decision:

We implemented the Humanoid Figure as a class that extends Actor, providing it a dynamic presence capable of engaging with the player through transactions. This entity possesses the ability to purchase items from the player, which are defined through the Sellable interface, ensuring a flexible and scalable system for handling various types of transactions. Each transaction potentially affects the game's outcome by influencing the player's resources and strategic options.

Alternative Design:

An alternative would have been to treat the Humanoid Figure as a static part of the environment, similar to a piece of Ground. This could simplify some interactions but would limit the dynamic interactions possible between the figure and the player, such as moving around or responding to game events dynamically.

Analysis of Alternative Design:

The alternative design, treating the Humanoid Figure as a static part of the environment, such as a piece of Ground, raises several concerns about SOLID design principles:

1. **Single Responsibility Principle (SRP):**
    - By integrating the transaction functionality directly into a static environment class like Ground, this design conflates the responsibilities of representing physical space with the business logic of item transactions. This could lead to a class that has multiple reasons to change, which complicates maintenance and scalability.
2. **Dependency Inversion Principle (DIP):**
    - Implementing transactions directly within the Ground or any concrete class ties the system tightly to specific implementation details rather than abstractions. This can make the system harder to refactor or extend, as changes in the transaction logic might affect many parts of the codebase that should ideally not be affected.
3. **Interface Segregation Principle (ISP):**
    - If transaction capabilities are embedded directly in the Ground class, any entity that extends Ground would have to support transactional interfaces, even if it

does not require these capabilities. This could lead to too many interfaces where many methods are irrelevant to most of the subclasses, reducing the system's overall cleanliness and understandability.

Final Design:
In our design, we closely adhere to several key object-oriented principles:

1. **(S)ingle Responsibility Principle (SRP):**
   - **Application:** We defined the HumanoidFigure as an Actor, separate from other game environment elements, focusing solely on interactions related to transactions. This clear separation allows the HumanoidFigure to manage trading interactions without being cluttered with unrelated functionalities. Additionally, we introduced the Status.PASSIVE to certain items and actors to signify that these entities should not initiate or be involved in combat scenarios, defining their roles within the game's mechanics.
   - **Why:** Adhering to SRP simplifies each class's responsibilities, making the system easier to understand, test, and maintain.
   - **Pros/Cons:** The primary advantage is easier maintenance and testing due to reduced class complexity. However, this might require more classes to be created, potentially increasing the overall system size.
2. **(O)pen/Closed Principle (OCP):**
   - **Application:** The design uses interfaces like Sellable to define behaviors for selling items. This allows new types of items to be introduced and sold without altering the existing code that handles the selling logic.
   - **Why:** Following OCP enhances the system’s flexibility by enabling the addition of new features without disturbing existing functionality.
   - **Pros/Cons:** The major benefit is the ease of adding new features without modifying existing code, reducing the risk of introducing bugs. The downside could be the complexity of managing multiple interfaces and implementations if not properly organized.
3. **(I)nterface Segregation Principle (ISP):**
   - **Application:** Our Sellable interface is designed to be specific and minimalistic, only including methods necessary for selling items. This prevents classes from being forced to implement methods they do not use.
   - **Why:** Complying with ISP reduces the overhead of managing unnecessary interfaces, making our classes lighter and more focused on their intended purposes.
   - **Pros/Cons:** The primary benefit is that it avoids loading classes with unnecessary functionality, keeping them streamlined. However, it may lead to

a higher number of interfaces, which could increase complexity if not managed well.

Conclusion:

Our chosen design, which extends Actor class uses interfaces for item transactions, adhering closely to the established object-oriented principles. By separating responsibilities effectively and using abstractions, we have constructed a system that not only meets the current requirements but is also well-prepared for future enhancements and scalability.