**Features**

**Key Features:**

* Used 7 Design Patterns
* Used multithreading
* Implemented above and beyond

**Design Patterns Used**

1. **Singleton Design Pattern**

**Used in:**

* Game class

**How:**

* Game class has a private static instance (private static Game instance) and a getInstance() method that ensures only one instance of the game exists across the application.

1. **Observer Pattern:**

**Used in:**

* Observer Interface, ObserverManager class, PirateShip

**Purpose:**  
Notifies pirate ships when Columbus moves.

**How:**

* ObserverManager maintains a list of Observers (e.g., PirateShip)
* notifyObservers(position) is called with Columbus' new position
* Pirates update their state accordingly

1. **Strategy Design Pattern**

**Used in:**

* MovementStrategy interface and its implementations like SlowChaseStrategy and FastChaseStrategy

**Purpose:**  
Allows slow and fast pirate ships to toggle chase strategies at runtime using “Toggle” button on the game screen

1. **Factory Method Design Pattern**

**Used in:**

* PirateShipFactory

**Purpose:**  
Encapsulates the object creation logic for different types of PirateShip (slow, fast, patrol) and makes code more modular.

1. **Decorator Design Pattern**

**Used in:**

* InvisibleColumbusDecorator

**Purpose:**

* Temporarily enhances Columbus ship’s abilities (invisibility) without modifying the base class columbus ship class.
* Player can click on “CC’s invisibility cloak” button and CC’s power to become invisible for 5 consecutive turns gets activated.
* After 5 turns, CC becomes visible again (decorator gets unwrapped).

**How:**

ColumbusShip cloaked = new InvisibleColumbusDecorator(stateManager.getColumbus(), 5);

* The decorator overrides methods like isInvisible() and decrementCloak() to provide enhanced behavior.

1. **Composite Design Pattern**

**Used in:**

* EntityGroup for grouping multiple entities like slowpirate and fastpirate

**How:**

* When CC gets invisibility power, slowpirate and fast pirate behave uniformly by ignoring CC for 5 turns.

1. **Facade Design Pattern**

**Used in:**

* Game class

**How:**

Game class is simplifying access to multiple subsystems.

The Game class provides a single entry point (façade) for:

* Moving Columbus
* Getting game state
* Resetting or starting the game
* Activating invisibility cloak
* Toggling pirate strategies
* Accessing the underlying subsystem components (GameStateManager, EntityManager, MovementController)
* ColumbusGameServer simply does:

Game game = Game.getInstance();

game.move("up");

game.togglePirateStrategies();

game.getState();

**Use of Multithreading**

**Used in:**

* MovementController

**How:**

* Multithreading is implemented in the MovementController class via ScheduledExecutorService, which manages concurrent tasks in separate threads.
* These threads periodically run monster and patrol pirate movement (every 3 seconds), independent of the main game loop or HTTP requests.
* Monsters move concurrently: A background thread is created that calls move() on all monsters every 3 seconds.
* Patrol pirate moves concurrently: Another independent thread moves the patrol pirate on a schedule and checks for adjacent collisions.
* Multiple executors (threads) run in parallel, updating game entities independently from user actions or HTTP routes.
* These tasks run in parallel with the main NanoHTTPD server thread (which handles web requests).