**Design Patterns:**

1. **Singleton**

Class: OrbitalControl as Singleton Class (one instance)

Usage: Control all operations with any task in orbit, including sending messages (User1->StarlinkX->User2), launching repair/resupply tasks etc.

Note: Text

Description automatically generated

1. **Mediator**

Classes:

* (Concrete)Colleague: OrbitalControl
* (Concrete)Mediator: StarlinkSatelite

Usage: When a user connects, it will have to connect to a vaid satellite in orbit, thus either one contained by the OrbitalControl which is valid, similarly for the disconnect and message. Used in conjunction with the Command and Chain patterns followed below.

1. **Command**

Classes:

* Invoker: User
* Command: Signal
* ConcreteCommand: ConnectSignal, DisconnectSignal, MessageSignal
* Receiver: OrbitalControl

Usage: When a User invokes a command, the OrbitalControl will receive it and relay it to the correct starlink satellite.

1. **Chain**

Classes:

* (Concrete)Handler: StarlinkSatelite
* Client: OrbitalControl

Usage: For any signal made to the StarlinkSatelite, the correct satellite needs to handle the request, thus the chain will ensure that only the correct satellite can handle specific requests.

1. **Template Method**

Classes:

* Abstract Class: Spacecraft
* Concrete Class: CrewDragon, Dragon

Usage: Return appropriate totalWeight amount according to subclasses.

1. **Strategy**

Classes:

* Strategy: LaunchController
* ConcreteStrategy: TestPhase, LaunchPhase, DockPhase
* Context: Main.cpp

Usage: Each \*Phase of the LaunchController will run different operations for that specific Phase. After completion of the phase, the LaunchController will move on to the next Phase.

TestPhase:

* Ensure that a Rocket and a Spacecraft is present
* Ensure all Rocket Compisitions are operational
* Ensure all Spacecraft Cargo and Crew are secure

LaunchPhase:

* Fire the engines!
* Timed events (just for show)

DockPhase:

* Update all starlink satelites to be in orbit
* Unload cargo/crew if need be

CompletionPhase: Display message

1. **Abstract Factory**

Classes:

* Abstract Factory: Configuration Controller
* Concrete Factory: Falcon9Configuration, FalconHeavyConfiguration
* Concrete Products: Falcon9, FalconHeavy

Usage: Refer the instantiation of the specific phases of the rockets to the Configuration Controller. Will return a Rocket object according to the correct phase.

1. **Observer**

Classes:

* Subject: Rocket
* Concrete Subject: Falcon9, FalconHeavy
* Observer: Composition
* Concrete Observer: Falcon9Core, MerlinEngine, VaccuumMerlinEngine

Usage: Appropriately start and check all Composition parts to make sure they are operational and also running.

1. **State**

Classes:

* State: Spacecraft
* Concrete State: CrewDragon, Dragon
* Context: LaunchController

Usage: Change the behaviour/operation of the Spacecraft according to its derived class. The CrewDragon and Dragon have different ways of calculating their weights and the State design pattern will help in implementing this.

1. **Prototype**

Classes;

* Prototype: Composition
* ConcretePrototype: Falcon9Core, MerlinEngine, VaccuumMerlinEngine
* Client: ConfigurationController

Usage: clone the Composition parts, since their constructions are identical and require multiple instances of themselves most of the time.