

COS214

Project

Team Hogwarts



Team members

Nasiphi Mjobo(team leader)	u18074074
Yané van der Westhuizen	u19021209
Morgan Else	u18025685
Lisa Fellingham(Rotter)	u19072067
Devon Hood	u19123460
Tawanda Jimu	u20494166
Risenga Sono	u19228882

Link to Google Doc:

https://docs.google.com/document/d/1tN-i67_5TNC_vISpLULe978v_v71J5AZBQebymdpUmA/edit#heading=h.la5jp5tnimjw

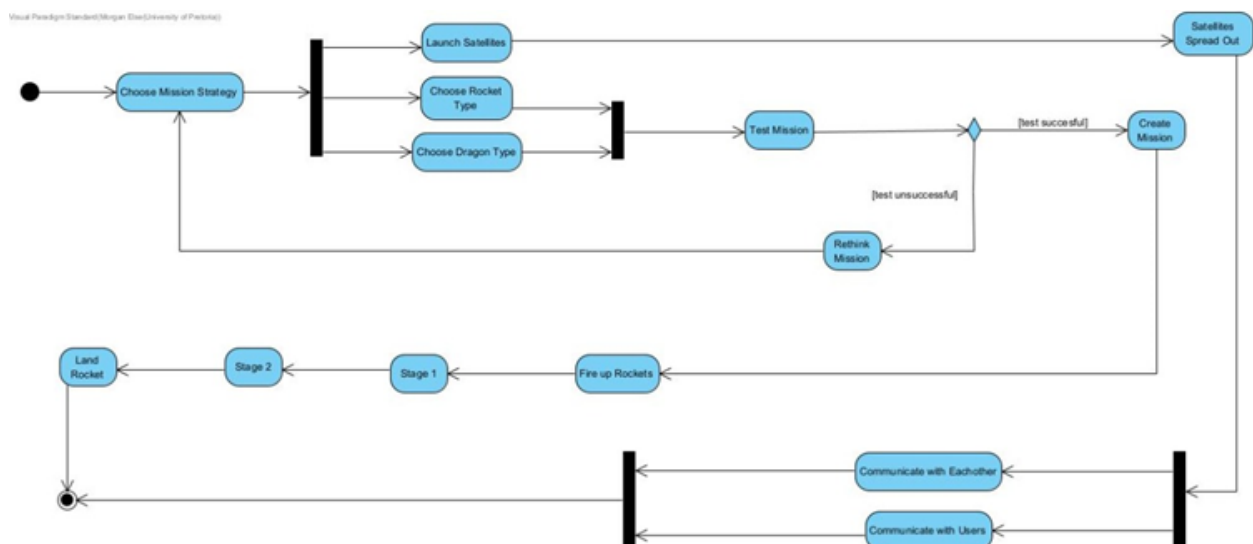
Task 1: Design

Functional requirements

Create various missions that have various types of rockets, engines and dragons within these missions:

- Observe and implement different phases of the respective rockets
- Iterate through the components that make the mission and be able to make changes during run-time(make changes to components)
- To create interface to simulate missions
- Create satellites that interact and communicate with each other and ground-people

Activity Diagram



Patterns used

1. Factory method
2. Mediator
3. Prototype
4. Strategy
5. State
6. Template
7. Composite
8. Builder
9. Decorator
10. Command

Rough design for patterns

Factory method:

Creator:	RocketFactory
ConcreteCreator:	Falcon9Factory, FalconHeavyFactory
Product:	Rocket
ConcreteProduct:	Falcon9, FalconHeavy

Mediator:

Mediator:	GroundReceiver
Colleague:	Satellite
ConcreteColleague:	Starlink

Prototype:

Prototype:	Rocket
ConcretePrototypeA:	Satelite
ConcretePrototypeB:	Dragon
ConcretePrototypeC:	Engine

Strategy:

Strategy:	MissionStrategy
-----------	-----------------

ConcreteStrategy:	RocketStrategy, EngineStrategy, DragonStrategy
Context:	Mission

Template:

AbstractClass:	Dragon
ConcreteClass:	CargoDragon & CrewDragon

FalconRockets

AbstractClass:	FalconRockets
ConcreteClasses:	FalconHeavy & Falcon-9

F9stages

AbstractClass:	F9Stages
ConcreteClass:	Stage1 and Stage2

FalconHeavystages

AbstractClass:	FalconHeavyStages
ConcreteClass:	Stage1 and Stage2

Composite:

Falcon9

Component:	Falcon9
Leaf:	Stage1 & Stage2
Composite:	F9Stages

FalconHeavy

Component:	FalconHeavy
Leaf:	Stage1 & Stage2
Composite:	FalconHeavyStages

SpaceCommand

Component:	SpaceCommand
Leaf:	DroneShip, Falcon9, FalconRockets
Composite:	FalconRockets

Builder:

Builder:	Dragon
ConcreteBuilder:	CargoDragonMaker , CrewDragonMaker
Product:	CargoDragon , CrewDragon

Decorator:

Component:	Engine
ConcreteComponent:	Core

```
Decorator:
ConcreteDecorator:
```

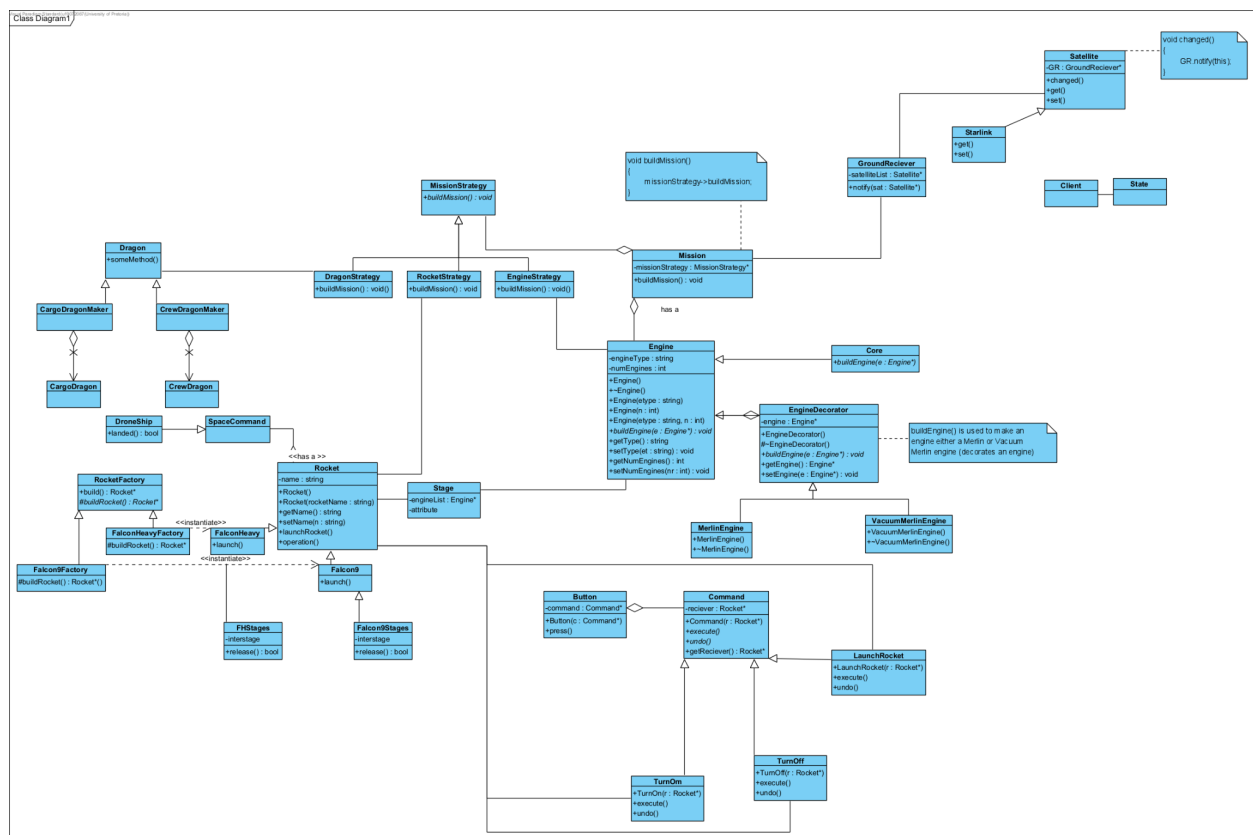
EngineDecorator
MerlinEngine, VacuumMerlinEngine

Command:

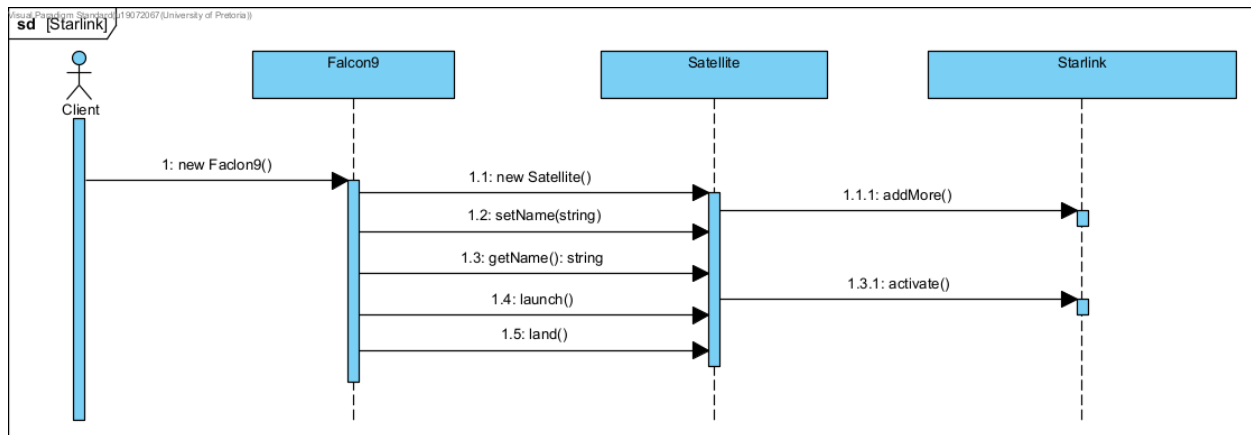
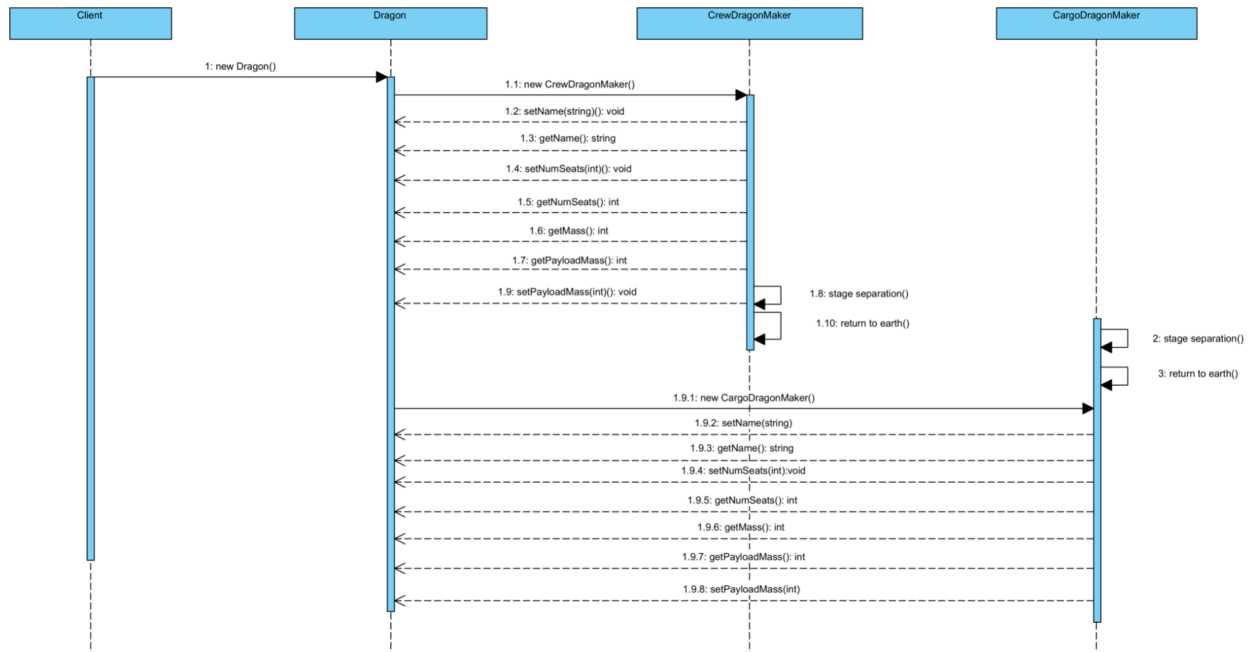
Command:
ConcreteCommands:
Invoker:
Receiver:

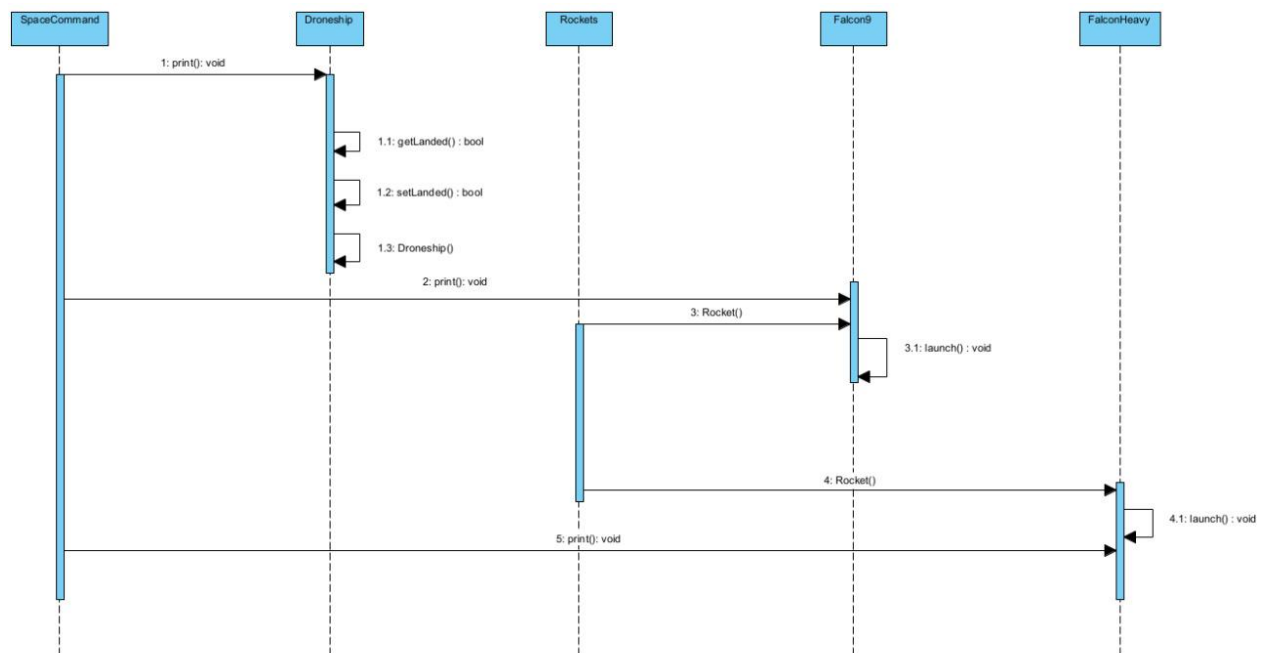
Command
TurnOn, TurnOff
Button
Rocket, Engine

Class Diagram

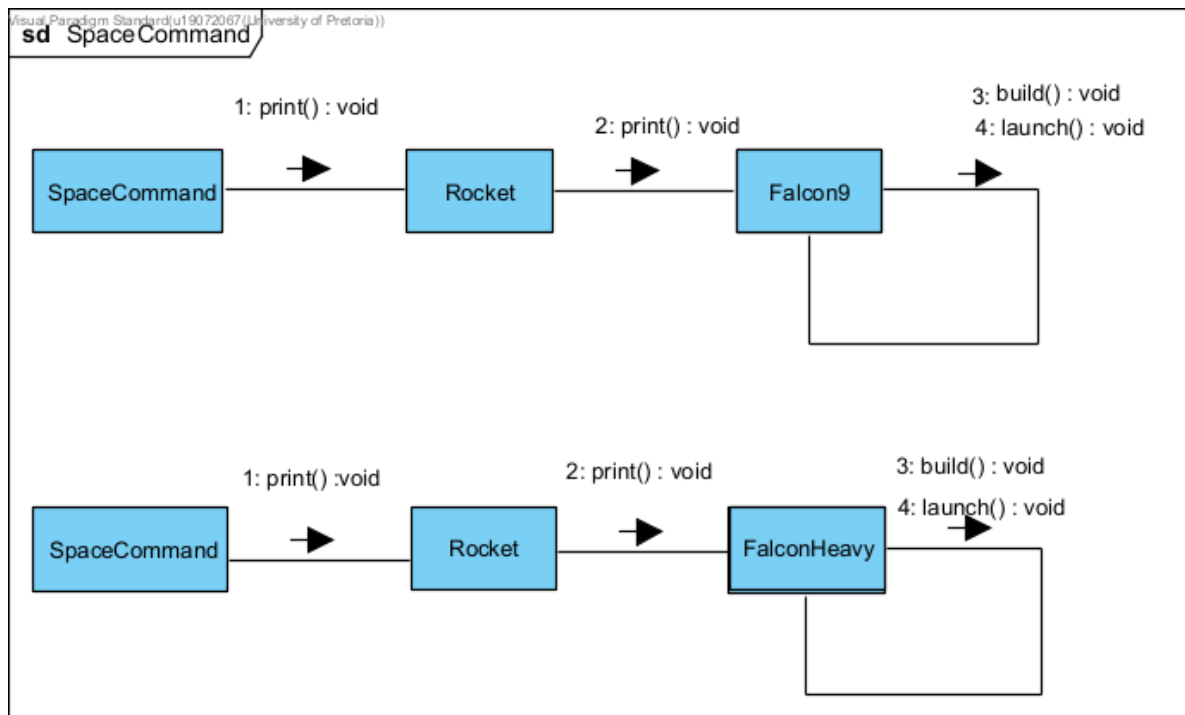


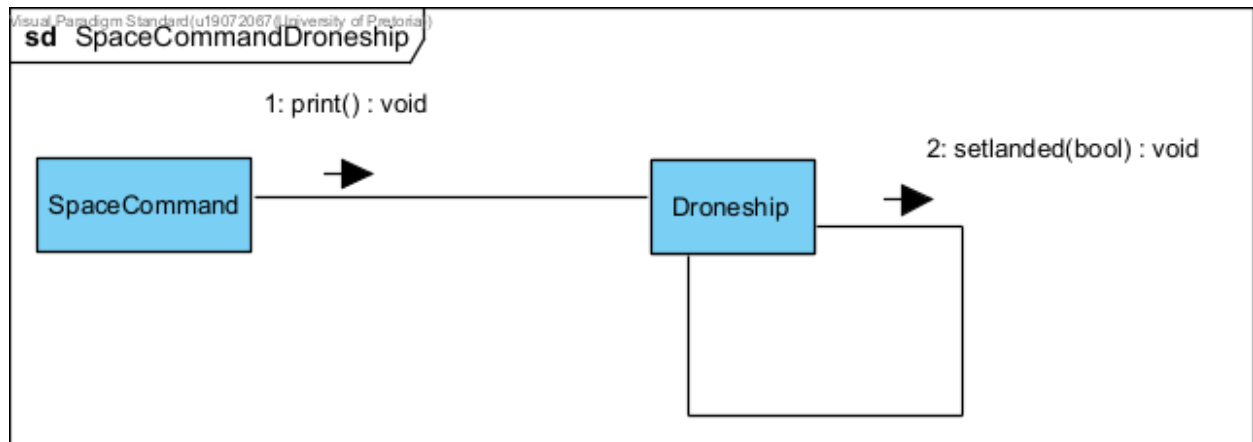
Sequence Diagrams



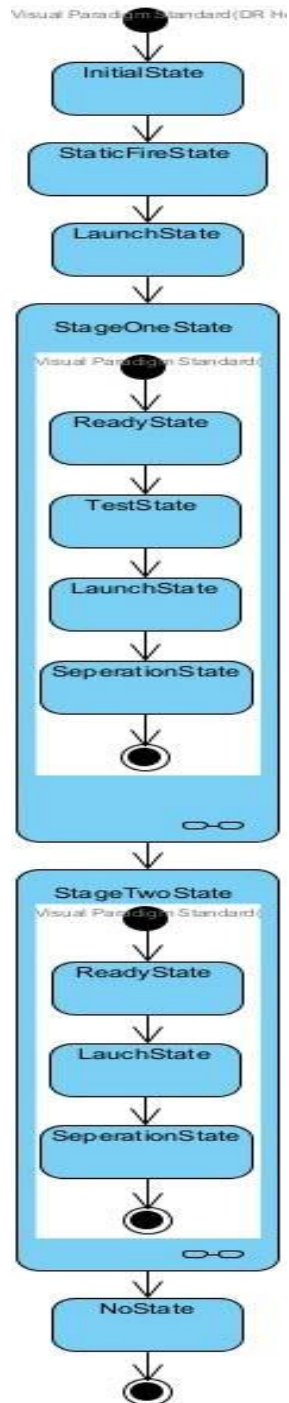


Communication Diagram

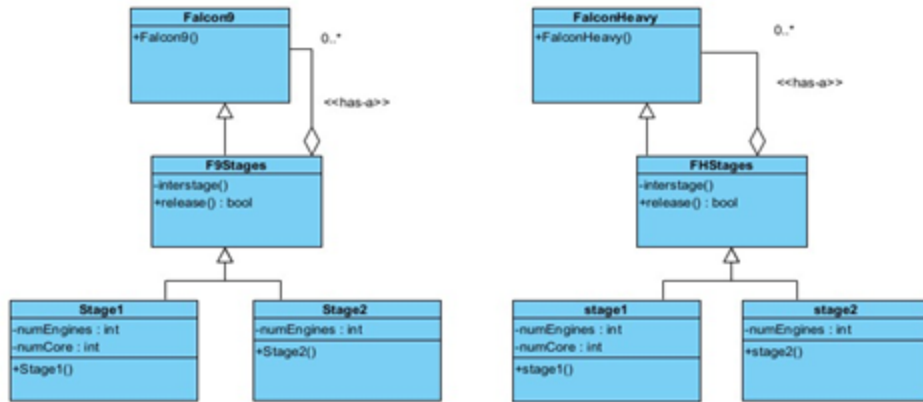




State Diagram



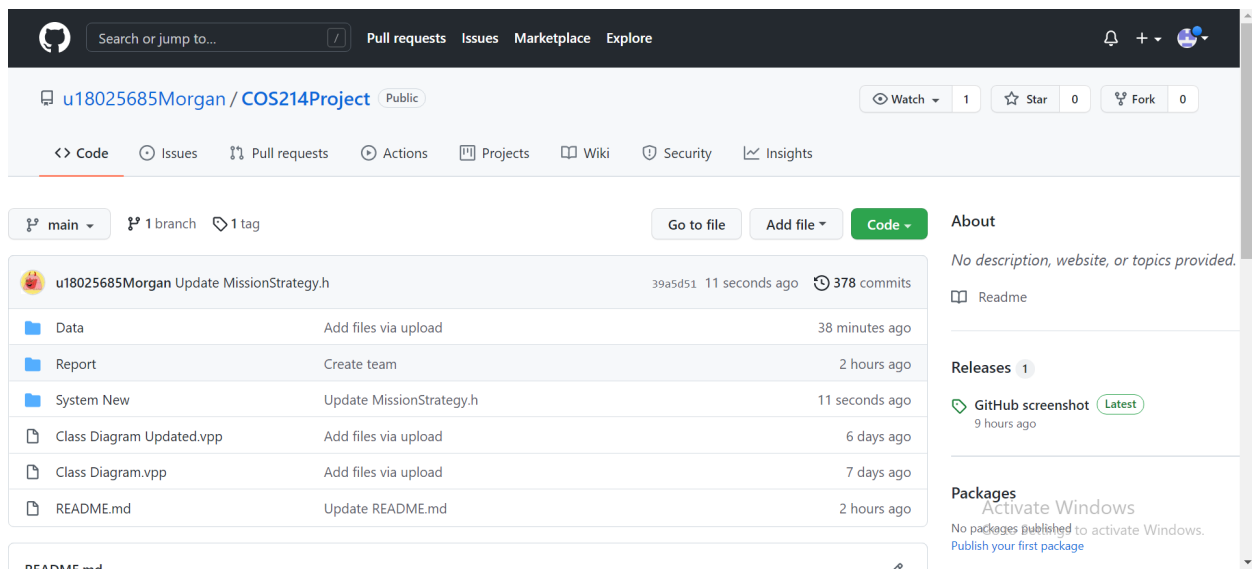
Object Diagram



Task 2: Implementation

Link: <https://github.com/u18025685Morgan/COS214Project.git>

ScreenShot:

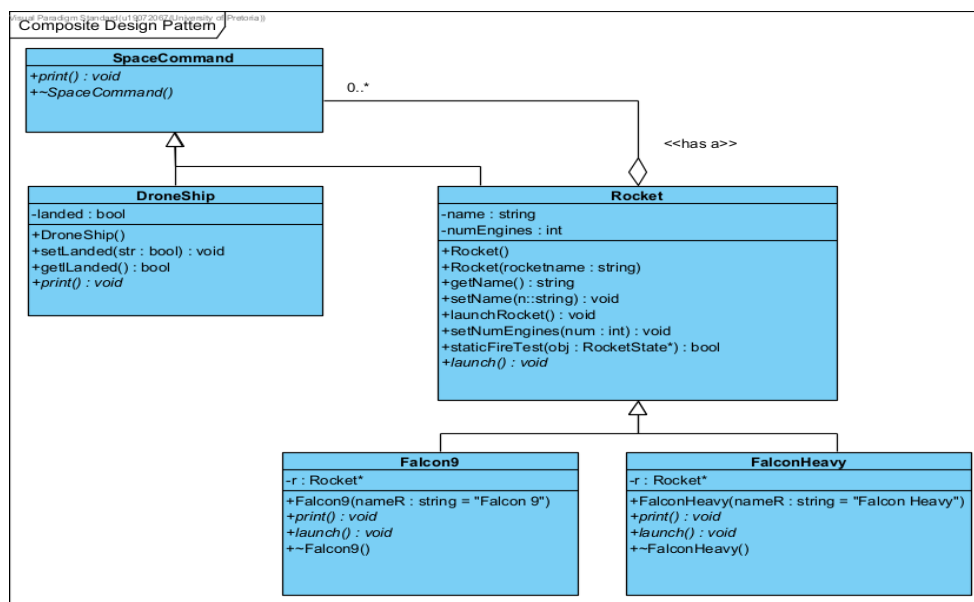


Task 3: Report

Composite Design Pattern

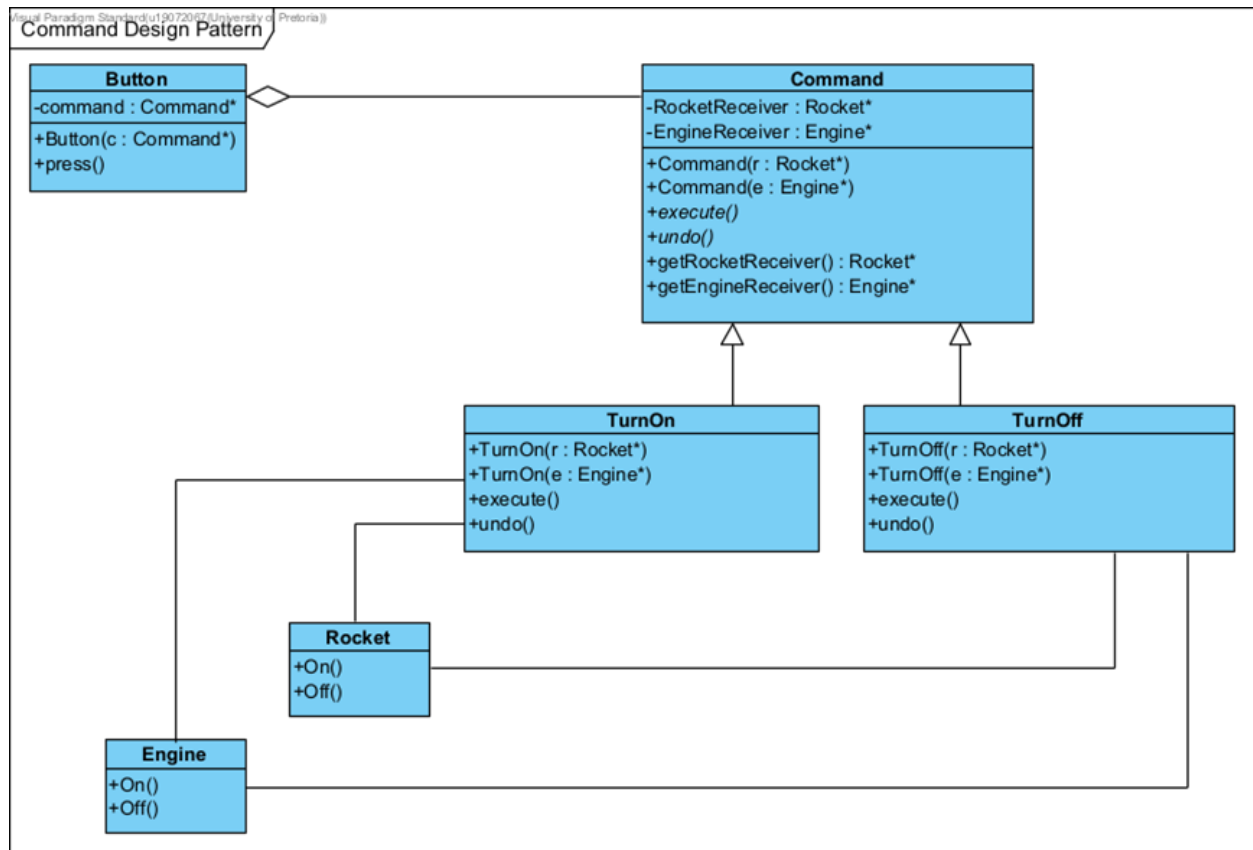
- We've applied the composite design pattern in the implementation of the rockets, where the participants are as follows:
 - Component : SpaceCommand , this class serves as the command center for the rockets.
 - Composite: Rockets, the abstract class Rockets will be the composite as the SpaceCommand "has-a" Rocket(s) - Rockets will have children classes Falcon9 & FalconHeavy.
 - Leaf: DroneShip, this class represents the drone ship where spaceX lands the first stage, thus it inherits from space command centre

We found that the composite design pattern was the most fitting to represent this relationship.



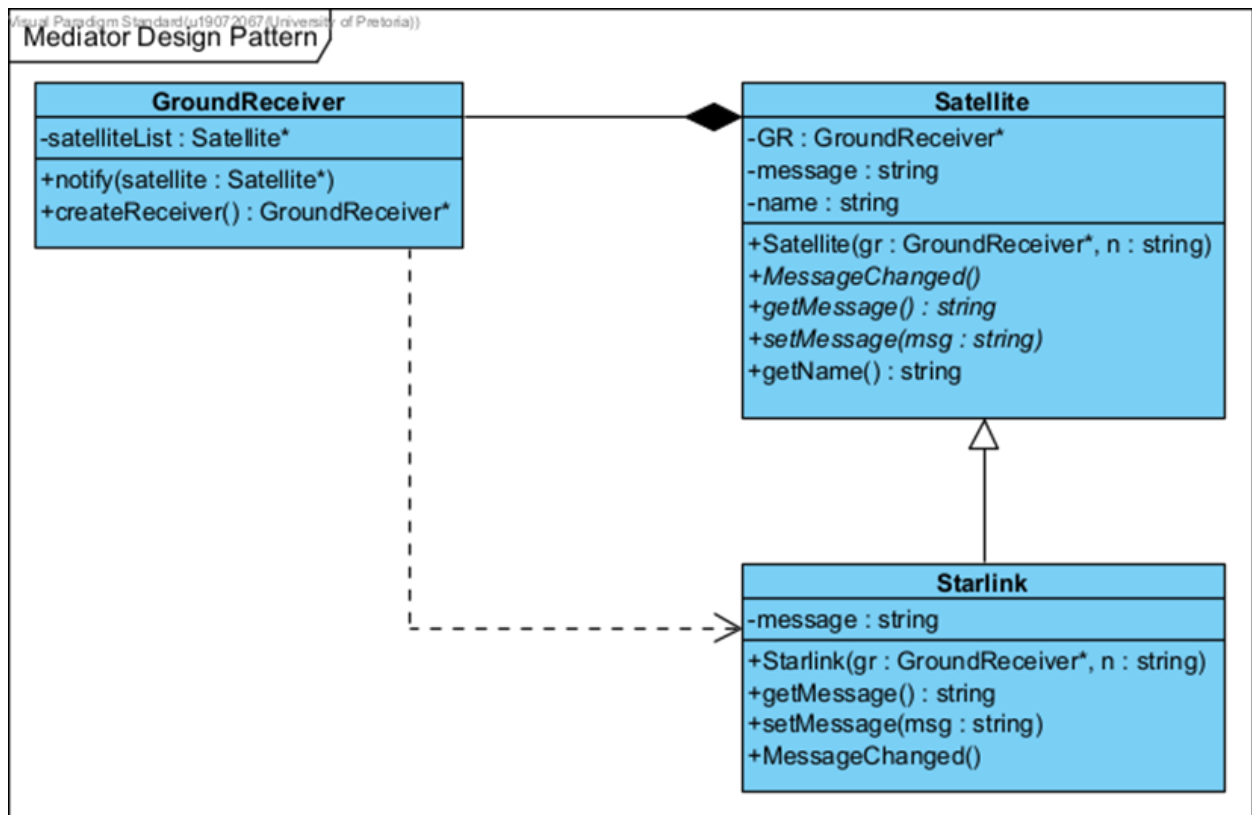
Command Design Pattern

The command design pattern allows us to turn on and off the Rockets and the engines by pushing a button.



Mediator Design Pattern

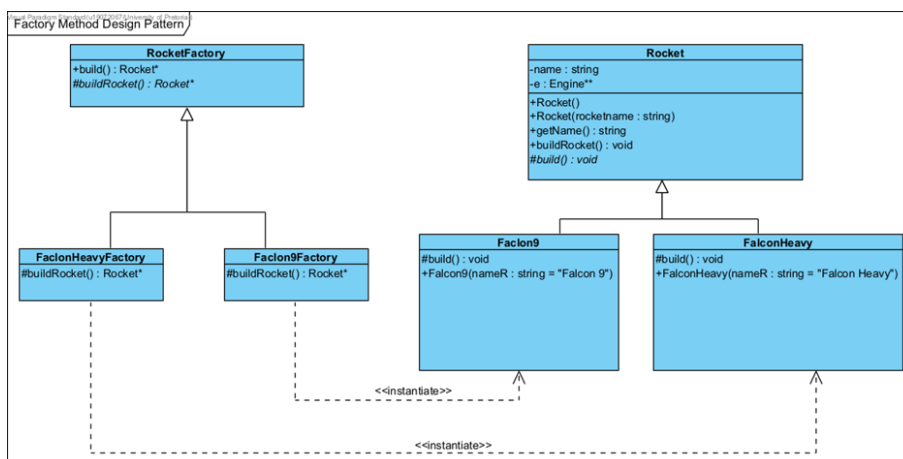
This design Pattern allows the satellites (Starlink Satellites) to send messages to each other without directly connecting. They send their messages to GroundReceiver who sends the messages to the other Satellites.



Factory Method Design Pattern

This design pattern is used to build the rockets (Falcon 9 and Falcon Heavy). When the respective rockets are built, the engines needed for each specific rocket is also added to them.

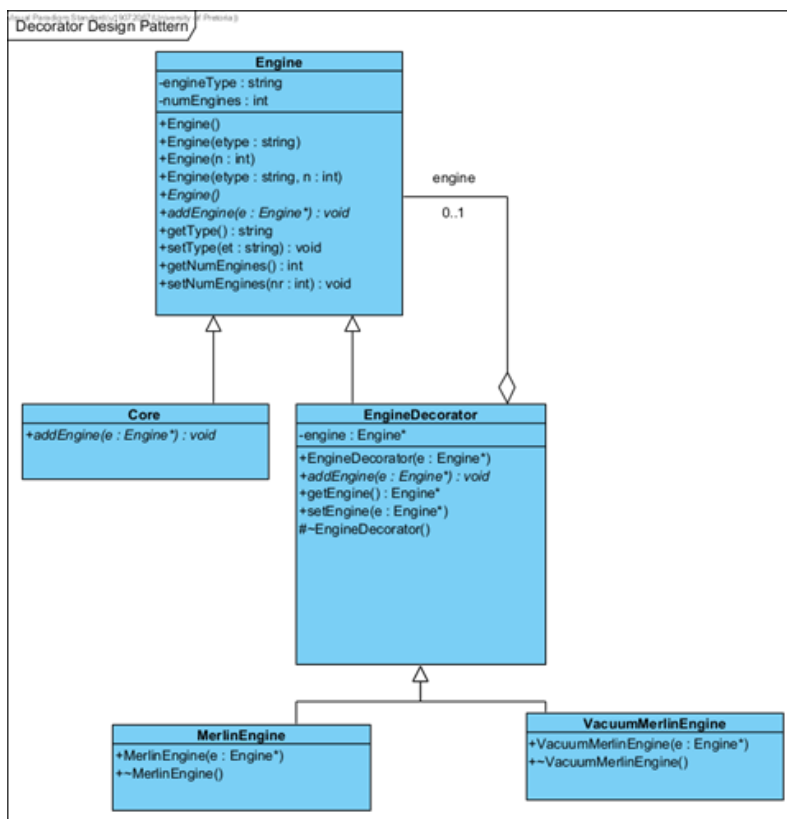
- **Creator** : This participant is represented by the RocketFactory class. It declares the factory method, buildRocket(), as a pure virtual function which returns a pointer to the Rocket (the product participant) class.
- **ConcreteCreator**: For this scenario, there will be exactly two ConcreteCreators known Facon9Factory and FalconHeavyFactory, which implements the factory method, buildRocket() and instantiates the rockets that corresponds to the specific concretecreeator. These two factories inherit publicly from the RocketFatory class.
- **Product**: The Rocket class represents this participant of the Factory Method Design Pattern. This class also includes a buildRocket method, which returns a void and therefore is not the factory method, that adds the necessary amount of engines to the rockets depending on the rocket's name. ○ the rocket's name.
- **ConcreteProduct**: The Falcon9 and FalconHeavy classes represent this participant. Both these classes inherit publicly from the Rocket class and has constructors that sets the name attribute of the rocket by default.



Decorator Design Pattern

This design pattern was chosen to represent the engines, where a “regular” engine can be decorated to become a Merlin Engine or a Vacuum Merlin Engine.

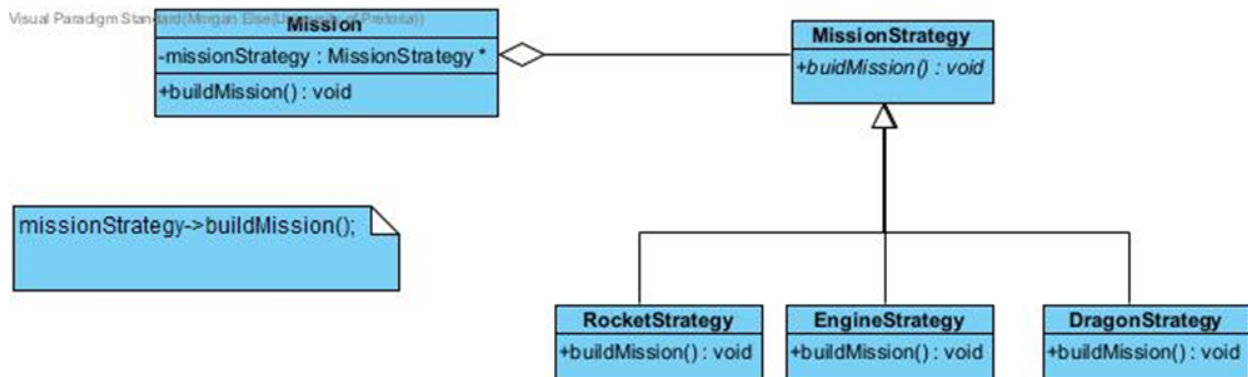
- **Component:** The Engine class represents this participant, and defines the operation(), addEngine(Engine* e), as a pure virtual function.
- **ConcreteComponent:** The Core inherits publicly from the component, Engine, and includes the addEngine(Engine* e) function.
- **Decorator:** The EngineDecorator implements the addEngine(Engine* e) function and defines a reference to an object of the component.
- **ConcreteDecorator:** For the purposes of this scenario, there are two ConcreteDecorators, MerlinEngine and VacuumMerlineEngine. They both inherit publicly from the decorator participant.



Strategy Design Pattern

The Strategy design pattern is used as the main interface of our simulation, allowing the client to choose the different aspects of the mission in one place. The components in this pattern are as follows:

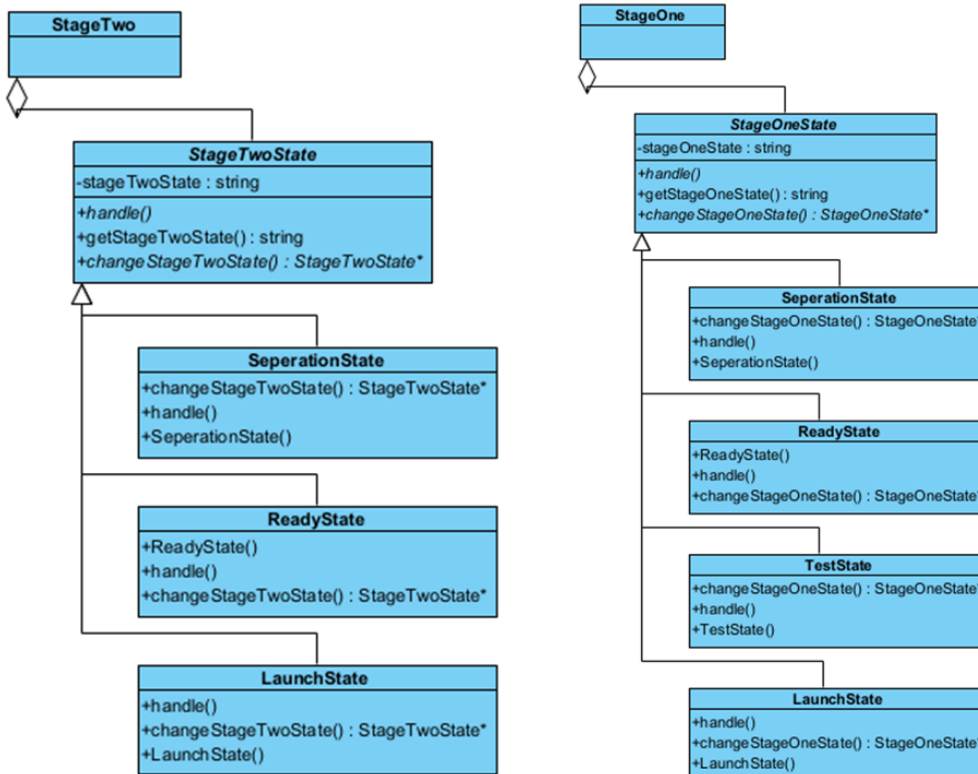
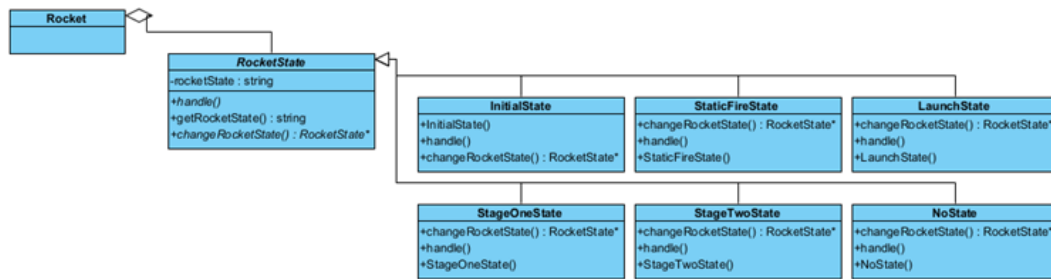
- **Strategy class:** MissionStrategy – the Mission uses this to call any algorithms defined by the ConcreteStrategies (RocketStrategy, EngineStrategy, DragonStrategy)
- **ConcreteStrategy class:** RocketStrategy, EngineStrategy, DragonStrategy – these classes implement the functionality that is defined in the MissionStrategy class. These classes create and test the missions that have various combinations of hardware.
- **Context class:** Mission – is the interface that lets the MissionStrategy access its methods and attributes. This class contains a reference to a MissionStrategy object.



State Design Pattern

We use the state design pattern as it allows us to change the behavior of the class depending on what state the object is in. In our mission, we have 3 different sets of states, RocketState, StageOneState and StateTwoStage, all representing the state of their respective objects.

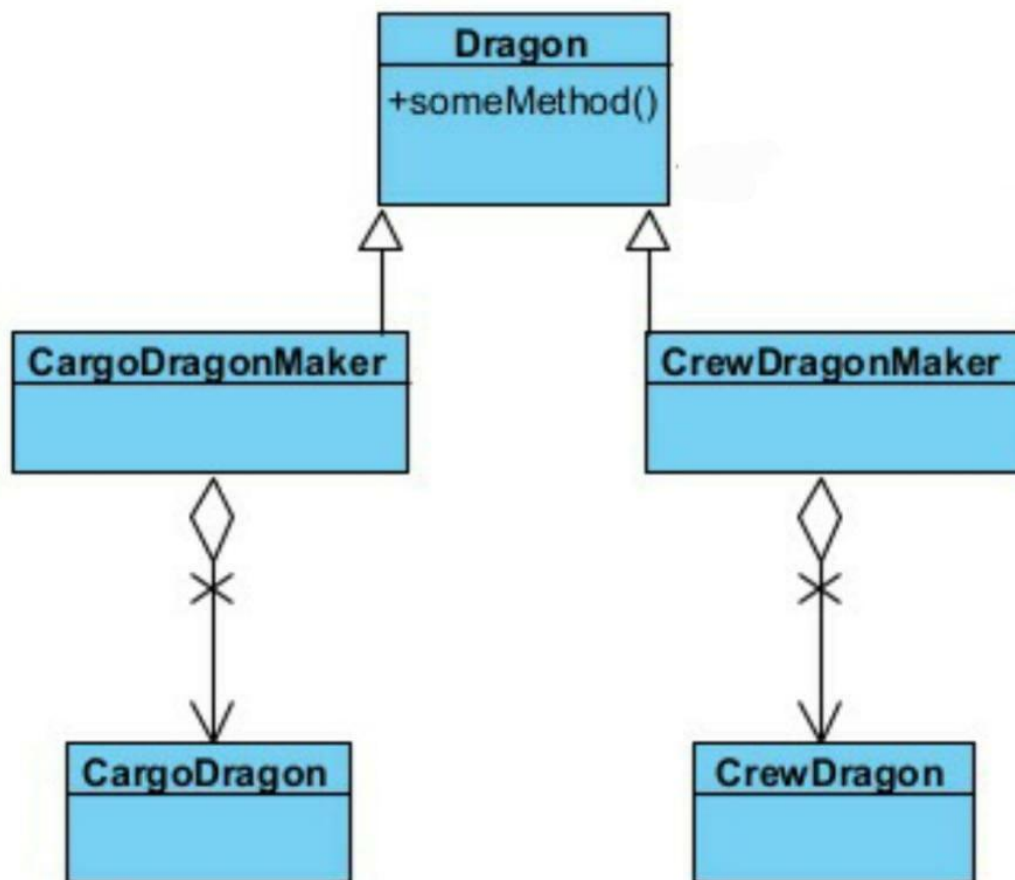
- **State Classes:** RocketState, StageOneState, and StageTwoState.
- **Context Classes:** Rocket, StageOne, and StageTwo.
- **Concrete States**
 - RocketState
 - InitialState
 - StaticFireState
 - LaunchState
 - StageOneState
 - StageTwoState
 - NoState
 - StageOneState
 - ReadyState
 - TestState
 - LaunchState
 - SeperationState
 - StageTwoState
 - ReadyState
 - LaunchState
 - SeperationState



Builder Design Pattern

The builder design pattern is to separate the construction of the Dragon object from its representation so that the same construction process can create different representations.

- Builder : Dragon
- ConcreteBuilder: CargoDragonMaker, CrewDragonMaker
- Product: CargoDragon, CrewDragon



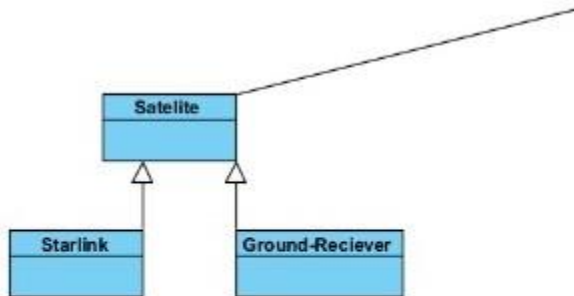
Prototype Design Pattern

The Prototype design pattern is to specify the kind of objects using a prototypical instance of the Satellites class and making copies using it

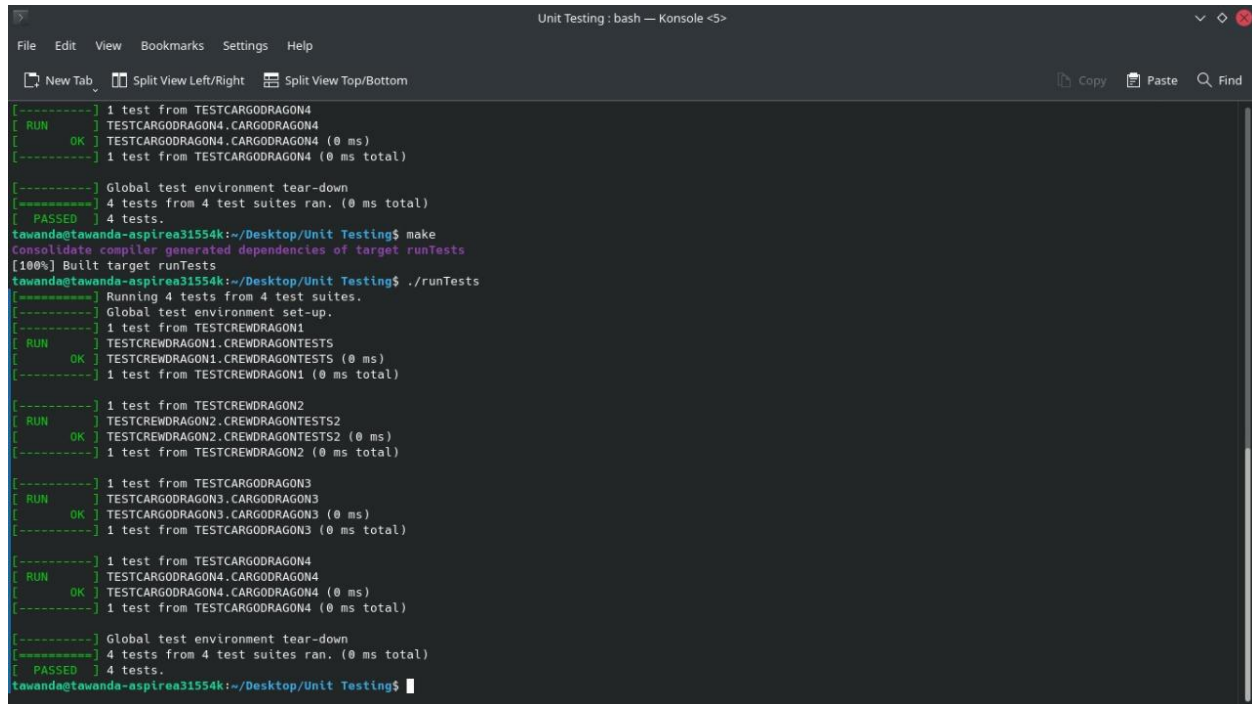
Prototype = Satellite

ConcretePrototype = Starlink and Ground-Receiver

Client = Falcon9



Unit Testing outcome screenshot



The screenshot shows a terminal window titled "Unit Testing : bash — Konsole <5>". The terminal displays the output of a test run. It starts with a global test environment tear-down, followed by 4 tests from 4 test suites. The tests are: TESTCARGODRAGON4, TESTCREWDAGON1, TESTCREWDAGON2, and TESTCARGODRAGON3. Each test suite has one test case, and all tests pass. The output is as follows:

```
Unit Testing : bash — Konsole <5>
File Edit View Bookmarks Settings Help
New Tab Split View Left/Right Split View Top/Bottom Copy Paste Find
[-----] 1 test from TESTCARGODRAGON4
[ RUN      ] TESTCARGODRAGON4.CARGODRAGON4
[ OK       ] TESTCARGODRAGON4.CARGODRAGON4 (0 ms)
[-----] 1 test from TESTCARGODRAGON4 (0 ms total)

[-----] Global test environment tear-down
[-----] 4 tests from 4 test suites ran. (0 ms total)
[ PASSED   ] 4 tests.
tawanda@tawanda-aspirea31554k:~/Desktop/Unit Testing$ make
Consolidate compiler generated dependencies of target runTests
[100%] Built target runTests
tawanda@tawanda-aspirea31554k:~/Desktop/Unit Testing$ ./runTests
[-----] Running 4 tests from 4 test suites.
[-----] Global test environment set-up.
[-----] 1 test from TESTCREWDAGON1
[ RUN      ] TESTCREWDAGON1.CREWDAGONTESTS
[ OK       ] TESTCREWDAGON1.CREWDAGONTESTS (0 ms)
[-----] 1 test from TESTCREWDAGON1 (0 ms total)

[-----] 1 test from TESTCREWDAGON2
[ RUN      ] TESTCREWDAGON2.CREWDAGONTESTS2
[ OK       ] TESTCREWDAGON2.CREWDAGONTESTS2 (0 ms)
[-----] 1 test from TESTCREWDAGON2 (0 ms total)

[-----] 1 test from TESTCARGODRAGON3
[ RUN      ] TESTCARGODRAGON3.CARGODRAGON3
[ OK       ] TESTCARGODRAGON3.CARGODRAGON3 (0 ms)
[-----] 1 test from TESTCARGODRAGON3 (0 ms total)

[-----] 1 test from TESTCARGODRAGON4
[ RUN      ] TESTCARGODRAGON4.CARGODRAGON4
[ OK       ] TESTCARGODRAGON4.CARGODRAGON4 (0 ms)
[-----] 1 test from TESTCARGODRAGON4 (0 ms total)

[-----] Global test environment tear-down
[-----] 4 tests from 4 test suites ran. (0 ms total)
[ PASSED   ] 4 tests.
tawanda@tawanda-aspirea31554k:~/Desktop/Unit Testing$
```