GeekHub Project Documentation

Orifha Mbedzi, Lindo Mlambo, Pako Diale

October 28, 2019

Contents

1 Overview 1
2 Factory Method 1
3 AbstractFactory 2
4 Iterator 3
5 Observer 5
6 Mediator 6
7 Chain of Responsibility 7
8 Decorator 8
9 State 9
10 Memento 10
11 Template Method 11
12 Singleton 12
13 Interpreter 13
14 Command 14

1 Overview

This document details all design patterns used in the project and highlights any notable functions.

2 Factory Method

Factory Method Pattern was used to build Spaceships.

Participants are listed below.

- Creator - SpaceshipFactory
- ConcreteCreator - BattleshipsFactory, ExplorationFactory, SpaceStationFactory, SPFighterFactory, SPTransporterFactory, FrigatesFactory
- Product - Spaceship
- ConcreteProduct - Battleship, Exploration, Frigates, SpaceStation, SPFighter, SPTransporter

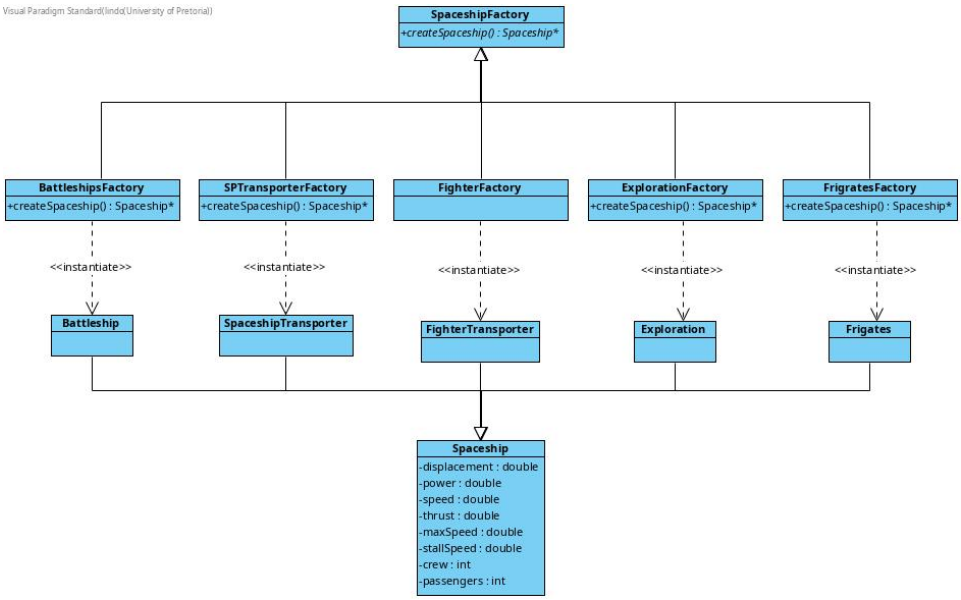

Figure 1: Factory Pattern class diagram.

3 AbstractFactory

Abstract Factory Pattern was used to create objects for all types of occupants on the ship. All occupants are of type Person, but each is first classified as either a Passenger or a CrewMember. CrewMembers further narrow down to the specific type of job they do.

- AbstractFactory - PersonFactory
- ConcreteFactories - PassengerFactory, CrewFactory
- AbstractProduct - Person
- ConcreteProduct - Passenger, Doctor, Engineers, Fighter, Comms, Captain, Commander, Navigator

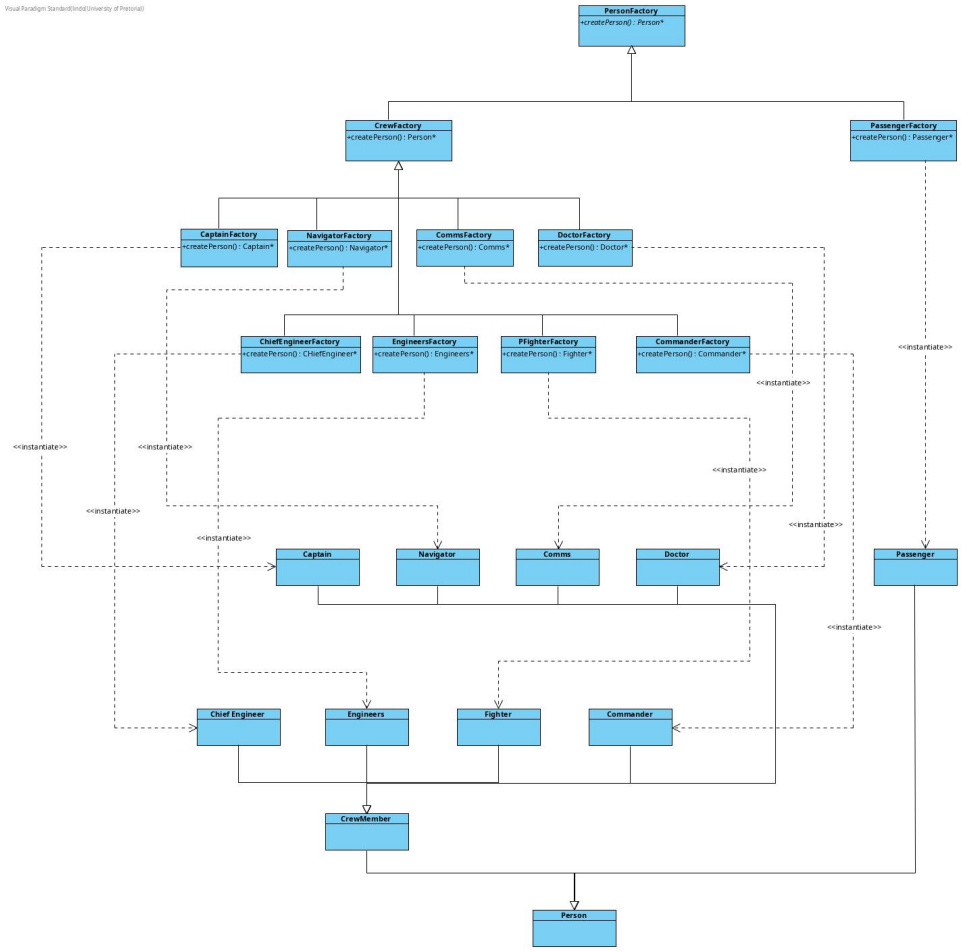

Figure 2: Abstract Factory class diagram.

4 Iterator

Iterator Pattern was used to model the Captain's log, which uses a vector as its data structure. Each log entry is represented as a string, the Captain has the ability to add log entries to the log and remove as seen fit. The iterator pattern was used to traverse the log.

The Captains the following functions to traverse his log.

```
next()  - fetches the next log entry.

current()  - fetched the current log entry.

hasNext()  - tells the Captain if he still has any more log entries to see.

isEmpty()  - tells the Captain if the log is empty, that is, has no entries.
```

Participants are listed below.

- Aggregate - CrewMember
- ConcreteAggregate - Captain
- Iterator - Iterator
- ConcreteIterator - LogIterator

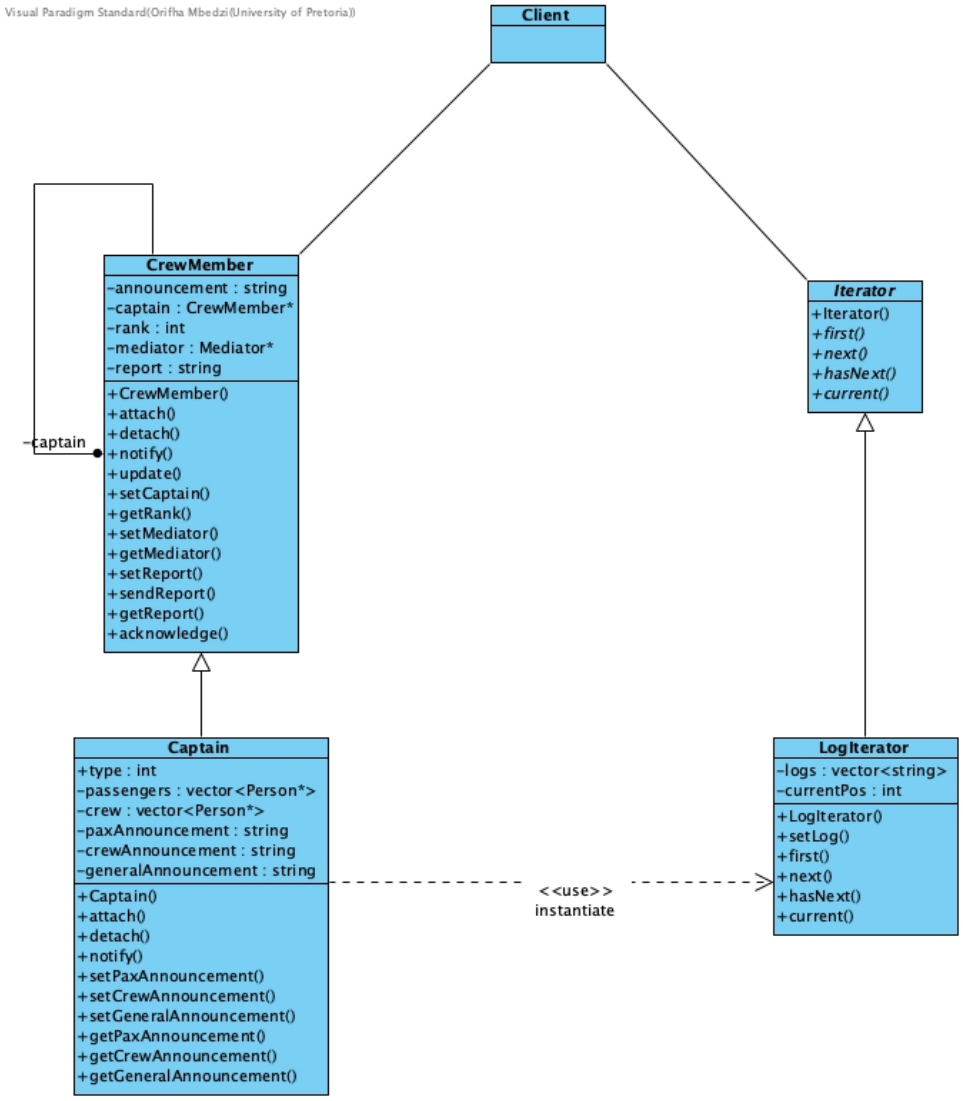

Figure 3: Iterator Pattern class diagram.

5 Observer

The ship's Captain needs the ability to send out announcements to his crew and passengers, the Observer Pattern was used to notify all passengers and crew of announcements. All passengers and crew subscribe to the Captain and can hear his announcement. However, the captain has a choice to alert either passengers only, crew only or both parties. He can do this via:

`setPassengerAnnouncement()  - notifies passengers of an exclusive annoucement.`

`setCrewAnnouncement()  - notifies crew of an exclusive announcement.`

`setGeneralAnnouncement()  - notifies all occupants announcement.`

Participants are listed below.

- Subject - CrewMember
- ConcreteSubject - Captain
- Observer - Person
- ConcreteObserver - Passenger, Doctor, Engineers, Fighter, Comms, Captain, Commander, Navigator

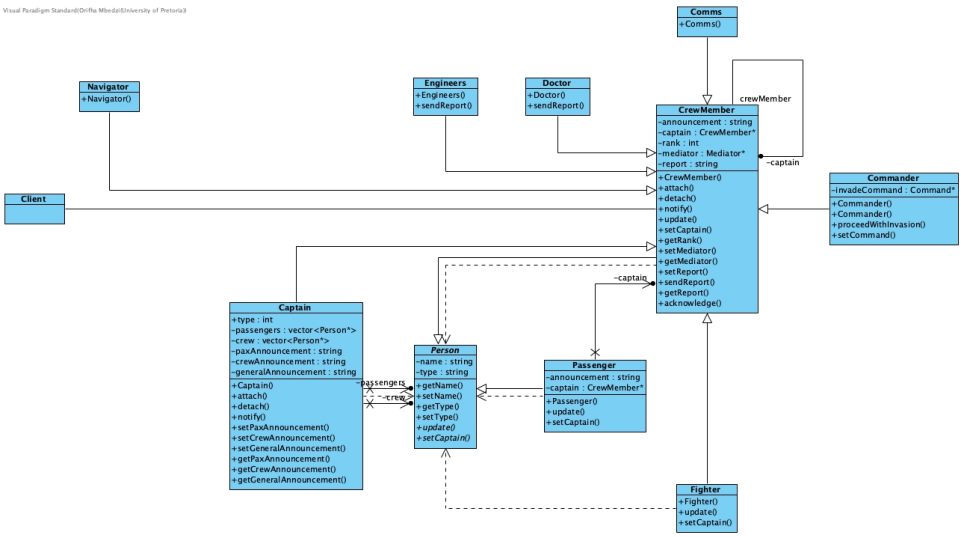

Figure 4: Observer Pattern class diagram.

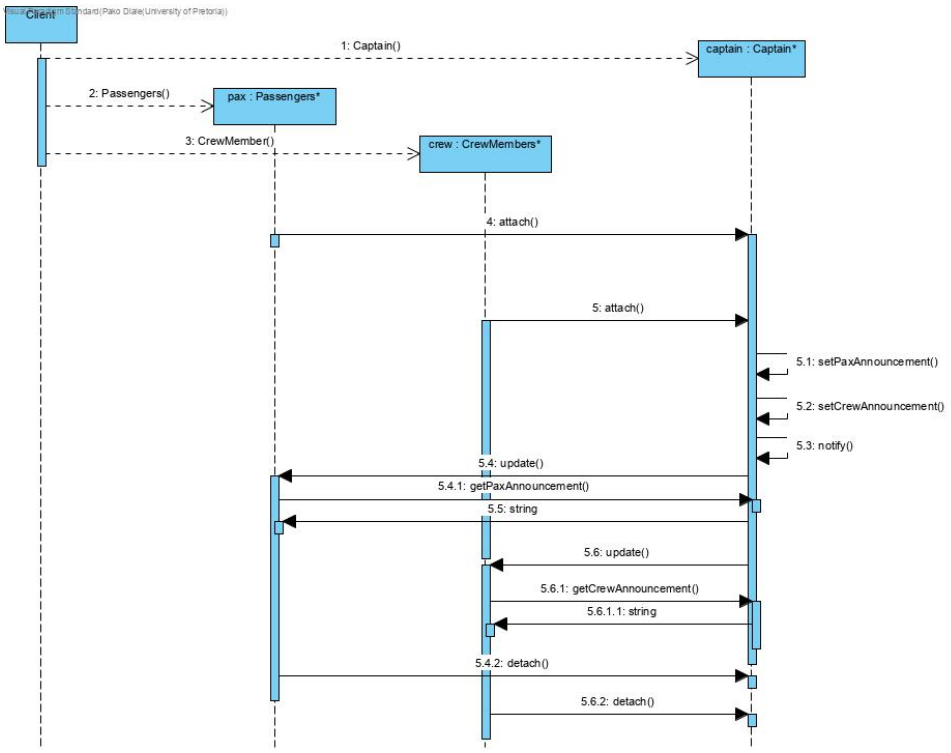

Figure 5: Observer Pattern sequence diagram.

6 Mediator

Every crew member has to report crew-wide reports, daily, and about important issues that he feels every crew member needs to know about. The following functions make this possible.

```
notify()  - This function belonging to the mediator send the message to all colleag

sendReport()  - The crew member calls this to send his report, initiating the proce

acknowledge()  - All notified colleagues acknowledge and read back the message fro
```

- Mediator - Mediator
- Colleague - CrewMember
- ConcreteMediator - ConcreteMediator
- ConcreteColleague - Doctor, Engineers, Fighter, Comms, Captain, Commander, Navigator

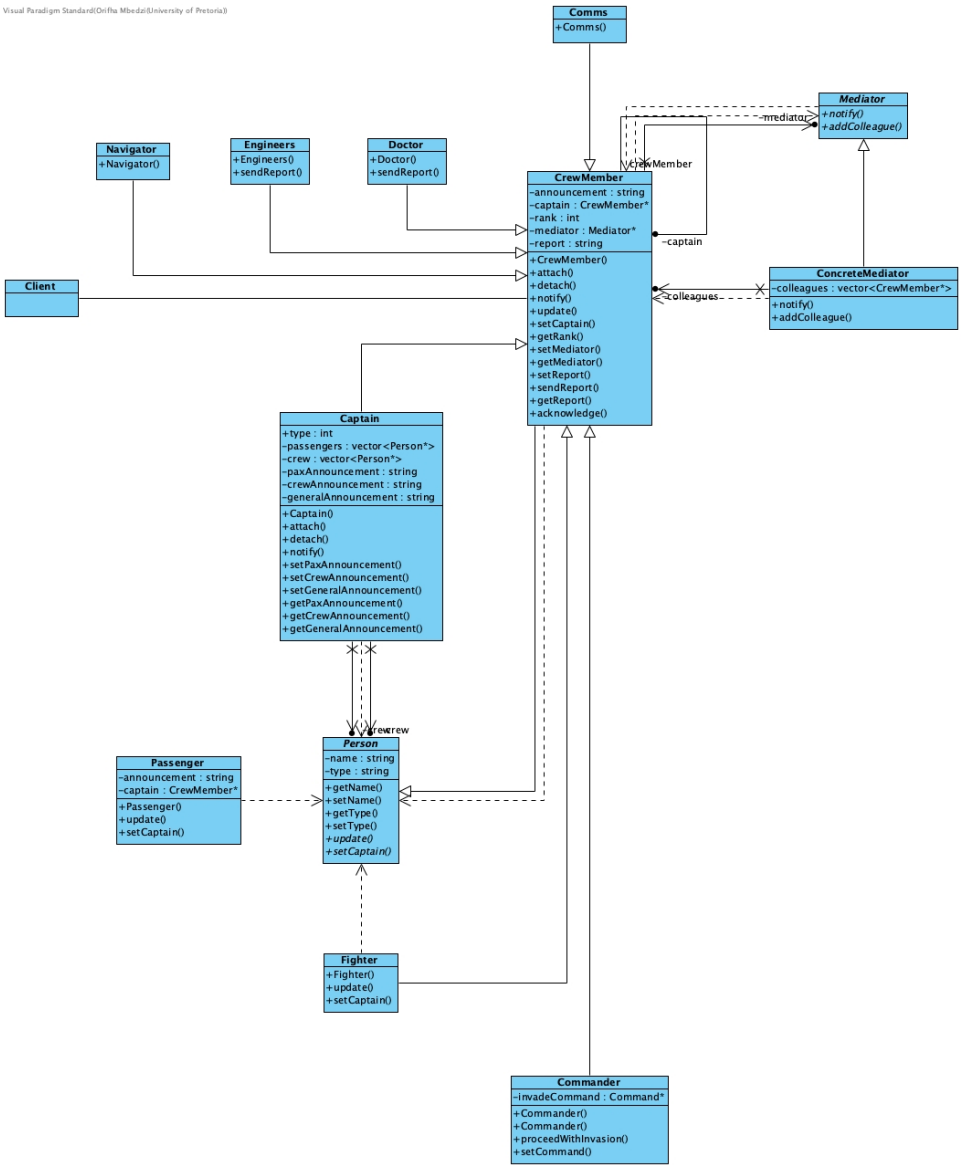

Figure 6: Mediator Pattern class diagram.

7 Chain of Responsibility

Crew members should be able to hand off problems they can't fix over to more capable, higher ranked crew. Each crew member has a rank. Each Handler entity has the ability to set a possible handler to hand off work to in case it cannot handle it. Only one function is of major importance to the client.

`handle() - Checks if problem is fixable by current handler, else it passes off to t`

Participants:

- Handler - ProblemHandler
- ConcreteHandler - DoctorHandler, EngineerHandler, FighterHandler

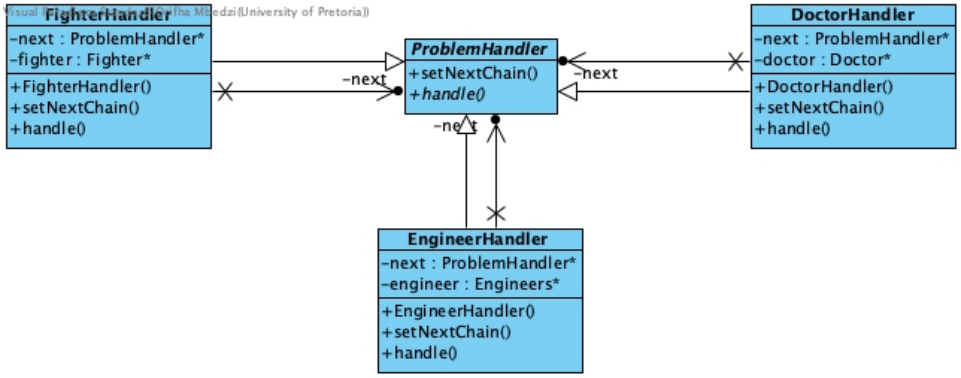

Figure 7: Chain of Responsibility Pattern class diagram.

8 Decorator

Sadly, Not all Spaceship are made equal, that's how life goes. Some are nicer than others. Some have TVs, Airconditioning and some super fancy premiums software.

Participants:

- Component - Component
- ConcreteComponent - SickBay, SleepingQuarters, Bridge
- Decorator - Decorator
- ConcreteDecorator - TVDecorator, PremiumSoftwareDecorator, AirconditioningDecorator

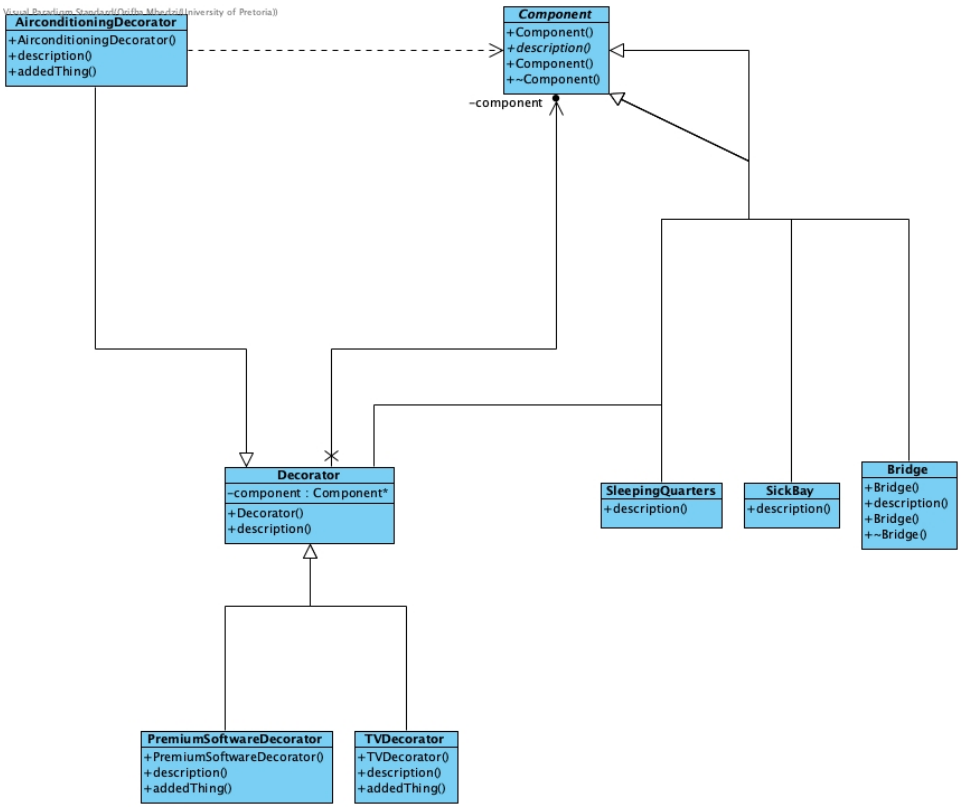

Figure 8: Decorator Pattern class diagram.

9 State

Each ship's state is encapsulated in a state object. Each ship's state can either be empty, half-empty or full. If Empty, it get refilled to half-empty and if half-empty, to full. No fuel gets more than half refill, to save on cost. Crunch space time.

Participants:

- Context - Spaceship
- State - State
- ConcreteState - EmptyFuelState, MidFuelState, FullFuelState

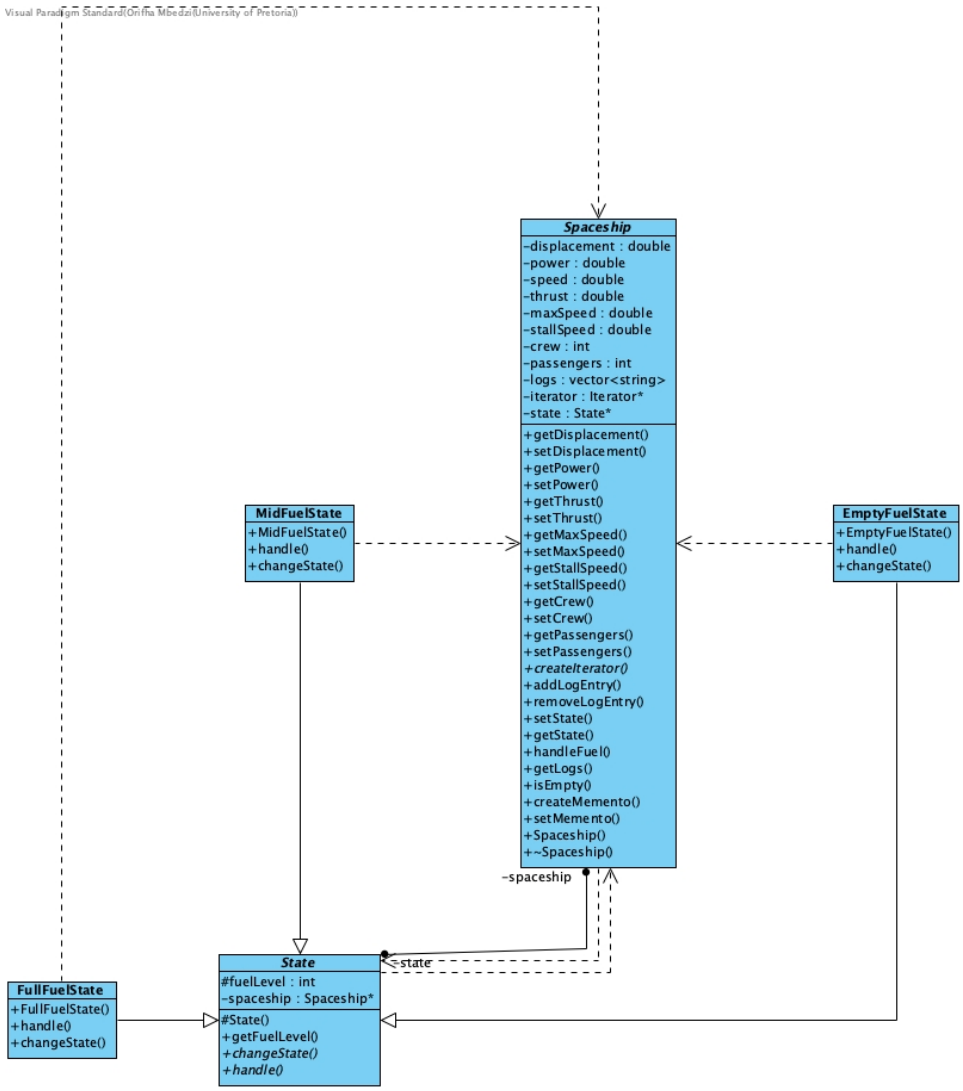

Figure 9: State Pattern class diagram.

10 Memento

For whatever reason, a spaceship can have the spacestation revert its fuel levels to a previous state. The Memento Pattern is used to revert the fuel state to a previously stored state. Notable functions:

`createMemento() - Spaceship creates a memento, which a Caretaker will take care of`

`getMemento() -  When its time to revert, Spaceship calls this function to set its s`

Participants:

- Originator - Spaceship
- Memento - Memento
- Caretaker - Caretaker

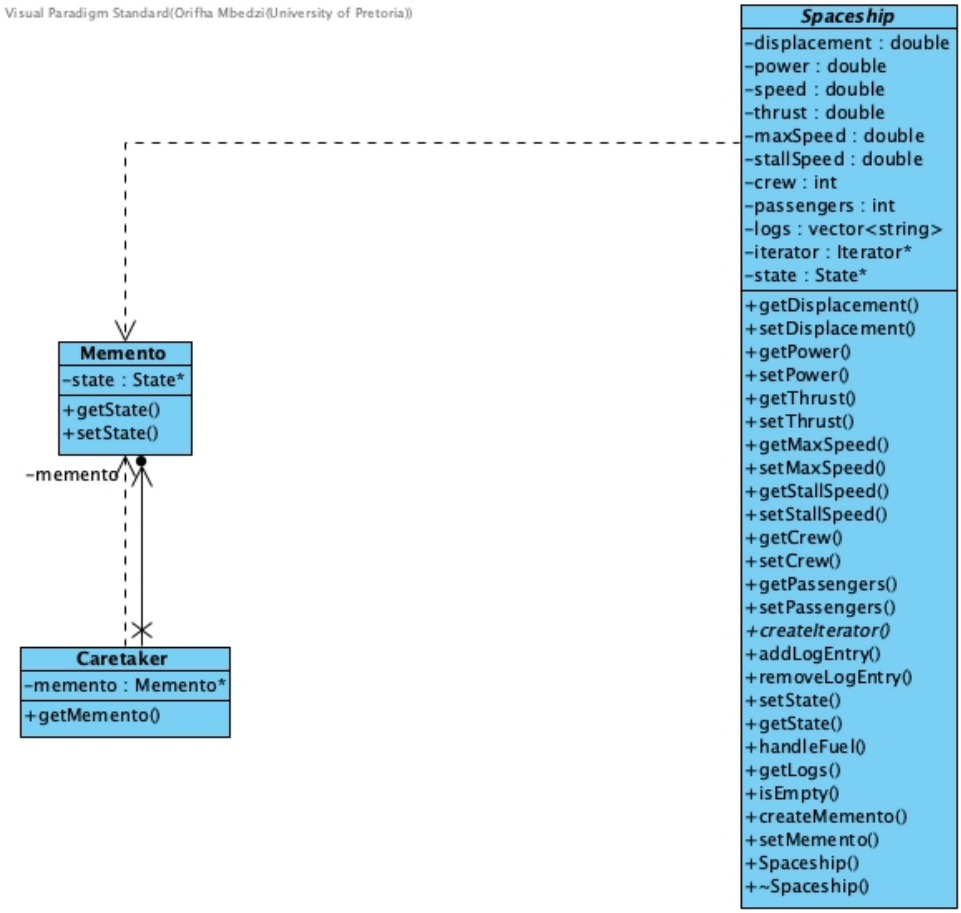

Figure 10: Memento Pattern class diagram.

11 Template Method

In the State Design Pattern, the function

```
handle()
```

is implemented differently as per subclass. It also makes uses of the changeState() primitive function. The function is a template method.

Participants:

- AbstractClass - State

- ConcreteClass - EmptyFuelState, MidFuelState, FullFuelState

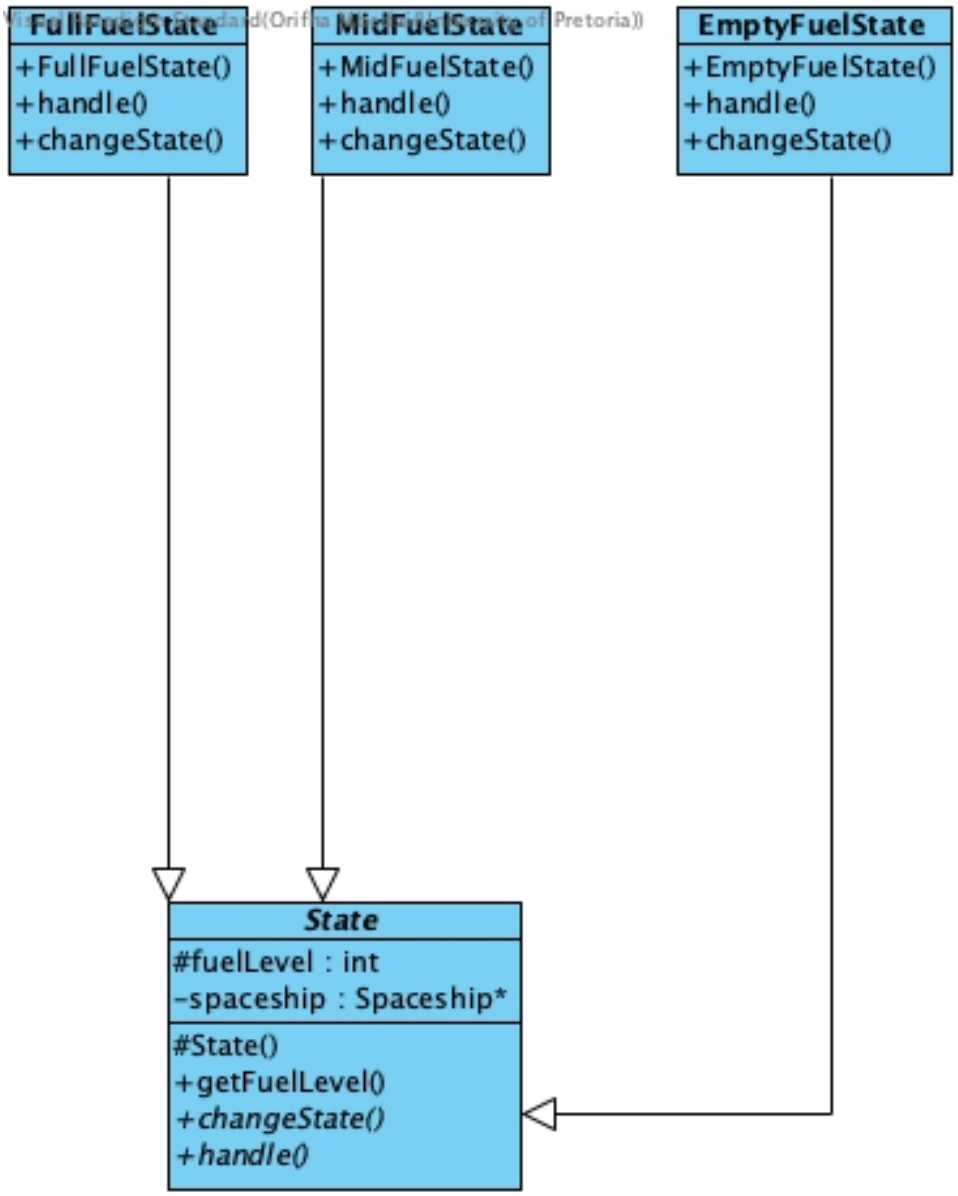

Figure 11: Template Pattern class diagram.

12 Singleton

The Singleton Design Pattern is used to ensure only one instance of the SpaceStation is created. The function responsible for creating the object is

`instance() - Checks if an instance has been created, if not create one, else retur`

The instance is a static object.

Participants:

- Singleton - Spacestation

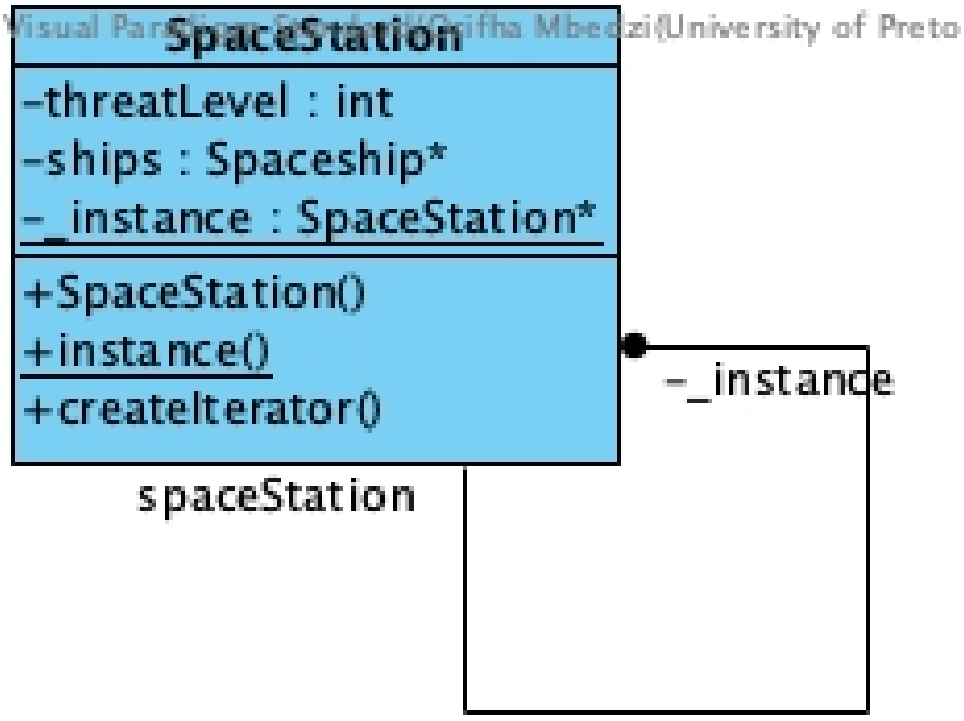

Figure 12: Singleton Pattern class diagram.

13 Interpreter

Communication between Critters and People would be difficult without this pattern.

Participants:

- Context - Context
- AbstractExpression - AbstractExpression
- TerminalExpression - ConstantExpression
- NonTerminalExpression - VariableExpression

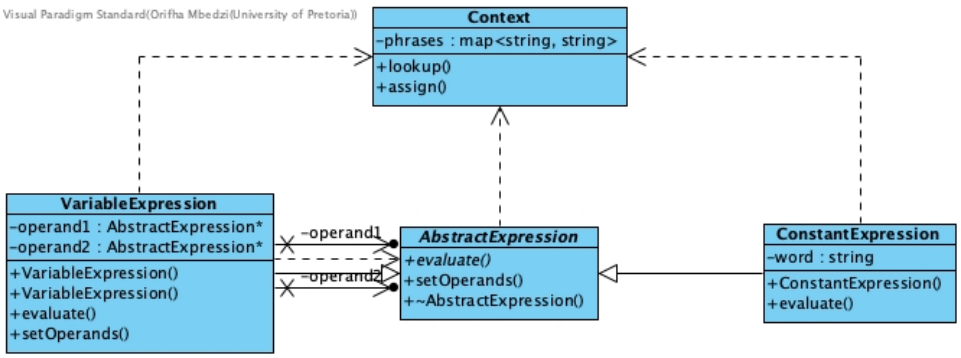

Figure 13: Interpreter Pattern class diagram.

14 Command

The commander needs the ability to give his troops orders to proceed based on findings they present to him.

Participants:

- Invoker - Commander
- Receiver - Spaceship
- Command - Command
- ConcreteCommand - InvadeCommand

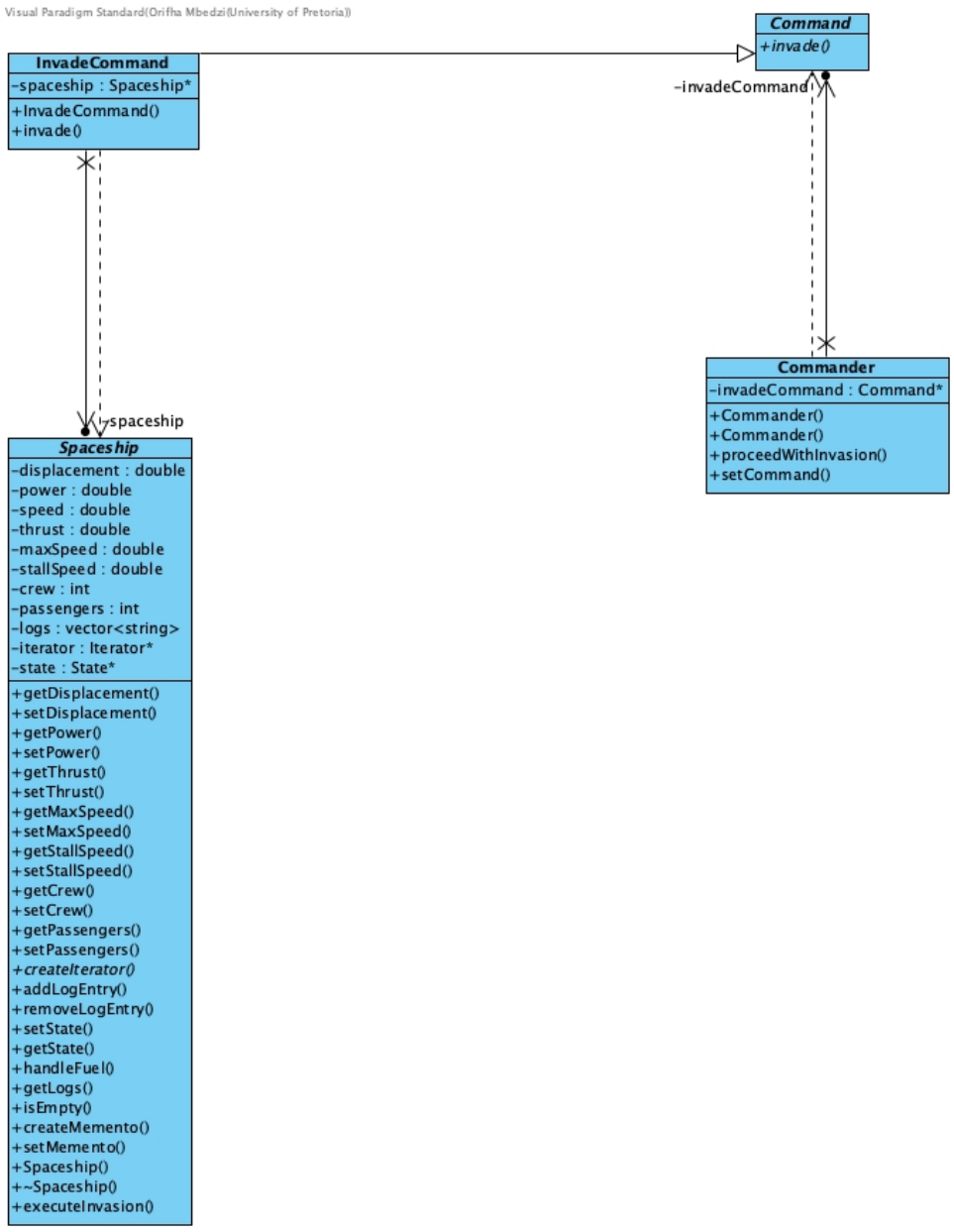

Figure 14: Command Pattern class diagram.

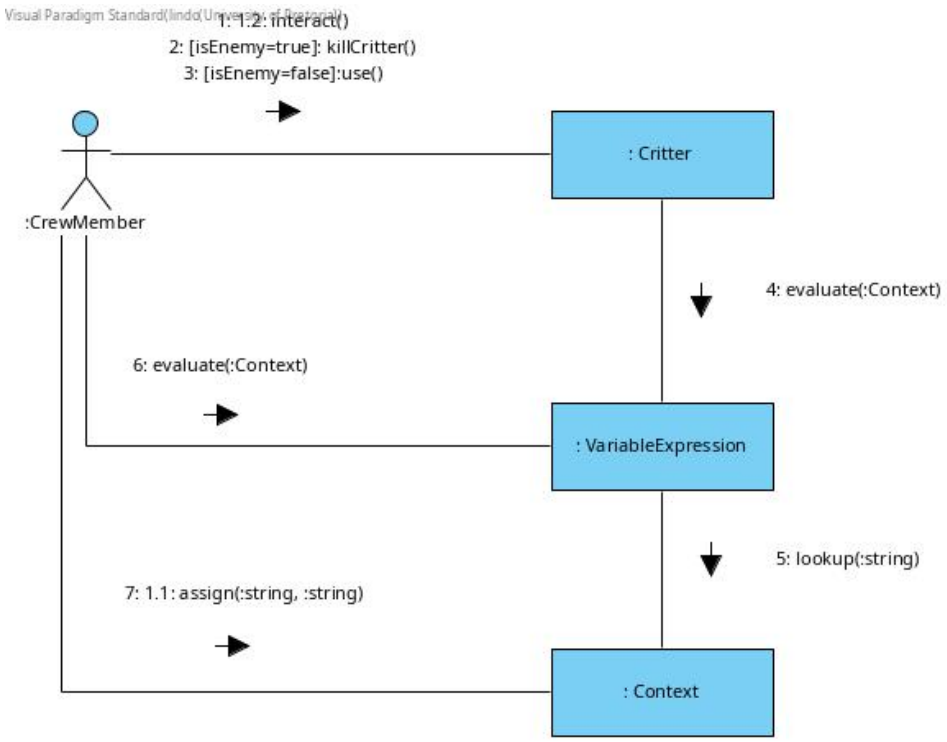

Figure 15: Command Pattern communication diagram.