INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR
DE TECNOLOGIA

First Phase Report

Object-Oriented Programming

Rodrigo Ferreira Moura-27995

\

INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE
ESCOLA SUPERIOR DE TECNOLOGIA

Introduction

This project is a simulation of a space agency, developed in C# and organized into various classes that represent different aspects of space missions. The structure of the project includes entities such as astronauts, missions, launches, rockets, and the space agency itself. Each of these classes is designed to model a crucial component in the planning and execution of space missions.

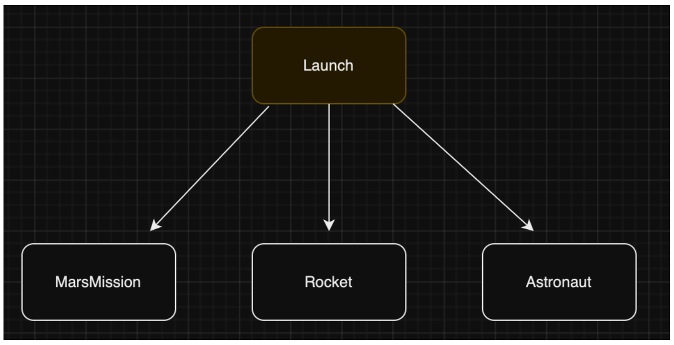
The **`SpaceAgency`** class centralizes information about the agency, storing the name, country of origin, and a collection of rockets available for missions. The **`Rocket`** class represents a specific rocket, with details such as fuel type, capacity, and maximum payload. These rockets are associated with specific missions, such as a mission to Mars, represented by the **`MarsMission`** class, which includes details such as the mission's objective, duration, and whether it is manned or unmanned.

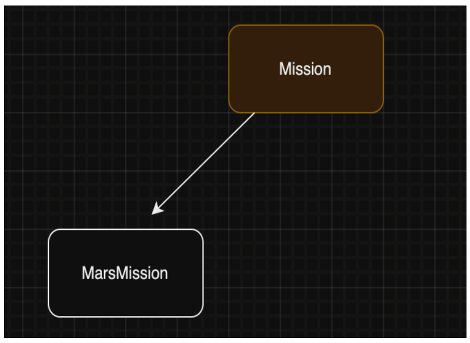
The project also features the **`Astronaut`** class, which represents astronauts with individual characteristics such as name, role, and years of experience, and the **`Launch`** class, which represents the launch of a mission, including information about the launch date and the rocket used.

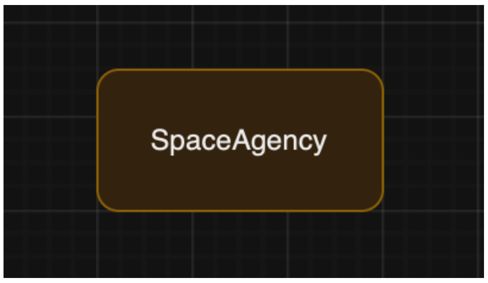
Each class in the project has methods and properties carefully validated to ensure that the data and actions performed in the simulation are consistent with the real-world constraints of a space mission. This modular structure allows the system to be easily expanded and adapted, simulating the operations and management of a space agency.

INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE
ESCOLA SUPERIOR
DE TECNOLOGIA

Diagrama de Estruturas de Classes







Implementations and Key Concepts Used in the Space Agency Project

The Space Agency project uses several important concepts to model the workings of a space agency. Here's a simplified overview of the key features:

1. Object-Oriented Programming (OOP)

The project is designed with OOP principles, focusing on creating classes that represent real-world objects, such as rockets and missions:

- **Encapsulation**: The classes have private properties and public methods to manage data safely.
- **Inheritance**: The `MarsMission` class inherits from the base class `CMission`, which allows for code reuse and easy expansion.
- **Polymorphism**: Methods like `ToString()` are customized to provide different outputs based on the object type.

2. Data Validation

Input validation is used throughout the project to ensure data is correct:

- **Constructor Validation**: When creating objects like `Rocket` or `Mission`, the constructors check that the input is valid (e.g., a rocket's capacity must be greater than zero).
- **Error Handling**: If invalid data is entered, the system throws exceptions, preventing issues in the program.

3. Using Enums

An enum called **`RoleType`** is used to define astronaut roles (e.g., Commander, Pilot). This ensures consistency and improves code readability.

4. Collections (List<T>)

The **`SpaceAgency`** class uses a **`List<Rocket>`** to store all rockets owned by the agency. The **`List`** makes it easy to add, remove, and access rockets.

- **LINQ** is used in **`GetRocketNames()`** to quickly generate a list of rocket names.

5. DateTime Handling

Dates are managed using the **`DateTime`** type. For example:

- The **`Launch`** class calculates how many days are left until a mission launches.
- The **`MarsMission`** class uses the start date and mission duration to calculate the mission's end date.

6. String Manipulation

Strings are used to display object information clearly:

- Methods like `ToString()` in classes such as `Rocket` and `Mission` return readable descriptions of objects.
- The `ShortDescription()` method in the `CMission` class gives a brief overview of a mission.

7. Exception Handling

The project handles errors gracefully by using exceptions. For example, if invalid data is provided (e.g., a negative mission duration), an exception is thrown to prevent problems in the program.

8. Custom Methods

Each class has custom methods to perform specific tasks:

- `CanCarryPayload()` in the `Rocket` class checks if the rocket can carry a certain weight.
- `UpdateRole()` in the `Astronaut` class allows the astronaut's role to be changed if needed.

9. Overriding Methods

Some methods, like `ToString()`, are customized in each class to return detailed and meaningful descriptions based on the object's data.

Future Projects

In future projects, we envision some adjustments in this first phase. The implementation in the second phase will include Windows Forms, as well as the integration of a database, along with minor adjustments in the classes and their respective subclasses. Improvements will also be made to the documentation, and a universal language will be adopted to provide better understanding for future developers.

Conclusion

In this project, the design and implementation of a space system using object-oriented programming and classes allowed for a modular and organized approach to simulating and managing various components of space missions.

By organizing the system into classes, we could easily manage and simulate how these parts work together. This approach made the system easier to expand and maintain. Overall, the project showed how using classes can help build a flexible and simple way to model complex space systems.