Report

Research Brief

Urban development is a process by which cities and towns grow in order to improve living conditions, infrastructure, economic activities, and overall quality of life for residents. Continuous development is essential in order to accommodate increasing populations and evolving citizen needs. City management principles are put in place to ensure that resources are used sustainably to promote improvement of the city. Key components - such as buildings, utilities, transportation, citizens, government, resources, and taxation - play interconnected roles in city development. These components require careful planning and strategic investments must be made to ensure a successful development of an urban area.

Different types of buildings provide various benefits to a city. Residential buildings provide homes for citizens, commercial buildings provide jobs and stimulate the economy, industrial buildings such as warehouses act as storage facilities for resources, and landmarks foster cultural connection and enhance urban character (Zhang et al., 2024). For this reason, it was decided that landmarks would contribute towards citizen satisfaction. Investments for the creation of such buildings are pricey and require a great quantity of resources, however, they bring vital functionality which is needed to accommodate development of cities, and they should thus be chosen carefully.

Utilities such as power, water, and waste management ensure that the city operates smoothly. Utilities must be managed effectively to minimise resource shortages, maintain health and safety standards, and promote quality of life. If these resources run low, the city is at risk of a catastrophe, so one must ensure they are all safely managed.

Transportation includes development of roads, public transport, and airports. Poor transportation systems have proven to stunt productivity and economic growth in cities (Wilcox & Nohrová , 2014). Maintenance of roads and public transport is crucial to keep a city operating at maximal efficiency. Airports act as hubs of economic activity, and allow immigration of foreigners into a city, and cargo transport exchange. This promotes foreign income and facilitates profit for the city's economy (Andrews, 2024).

The citizens of a city are the central component. They create a demand for housing, employment, services and a good quality of life. In return they generate income for the city in the form of tax, and provide services which ensure a smooth running for the city. Therefore, their satisfaction should be made a priority, and the city will benefit as a result, being rewarded with resources at the end of each year.

The government is responsible for tax collection, budget allocation and the making of policies. Such policies serve as laws in which citizens must adhere to in order to keep the city from failing. South Africas are very familiar with water and electricity restrictions, therefore, such policies were adopted into the implementation of the program.

Resources include materials, energy and water. Materials are required to build infrastructure. Energy is required to keep buildings and services up and running, and water is a basic need for the citizens of the city. Power plants and water supplies maintain safe levels of energy and water respectively, and materials are rewarded as a result of quality satisfaction.

Taxes usually generate the majority of a city's income (gfoa, 2017). Unfortunately, they are also a source of displeasure for the citizens of a city. The tax system must balance revenue

generation with citizen satisfaction, as high tax rates may deter business activity, while low rates may strain resources.

All of these components contribute to the chances of success a city will have with its development. Intense planning and strategizing must therefore be put into the allocation of resources and expenditure of the city's budget. This challenge is simulated in our city builder video game with the goal to achieve as much city growth before succumbing to the city's demise.

Git Branching Strategy:

For our project, we used Git as our Version Control System to manage and track changes efficiently. Each team member was responsible for implementing different classes, allowing us to work concurrently and avoid conflicts.

Our branching strategy involved creating individual feature branches for each class. Each team member developed their assigned class in their respective branch and committed changes regularly to document progress. Once the classes were fully implemented and tested, we merged these feature branches into the main branch.

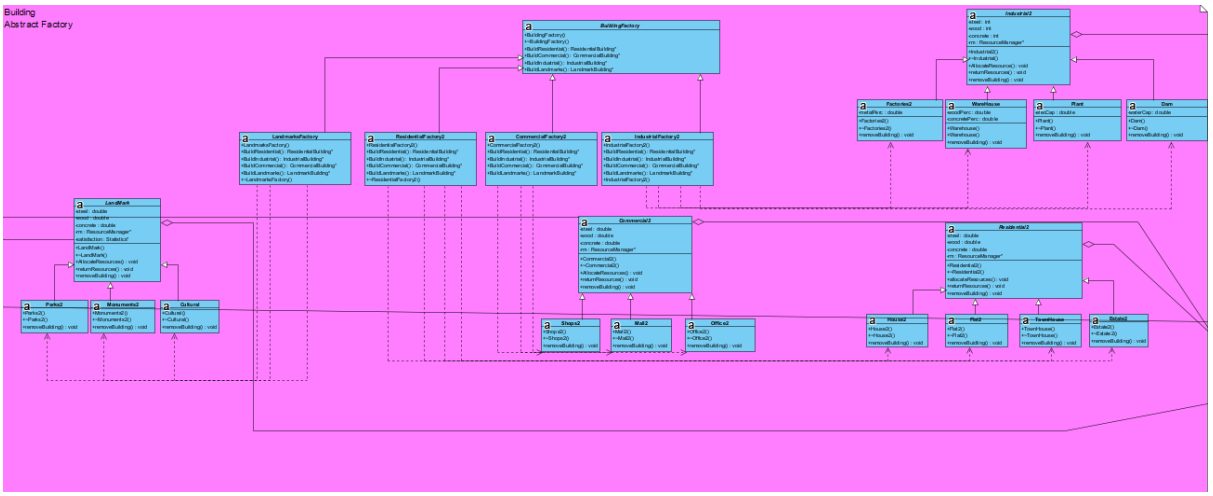
To prepare for our demo, we combined all the classes in the main branch, where we integrated and tested them together to ensure smooth functionality and collaboration among components. This allowed us to showcase the completed project as a unified system.

Additionally, we created a diagrams folder in the repository to store our design diagrams. Each team member uploaded their respective diagrams to this folder, ensuring that our UML and other design documentation were well-organised and accessible for reference. This structured approach helped us maintain a clean and organised codebase while facilitating efficient collaboration throughout the project.

Design Pattern Application Report

Abstract Factory Design Pattern:

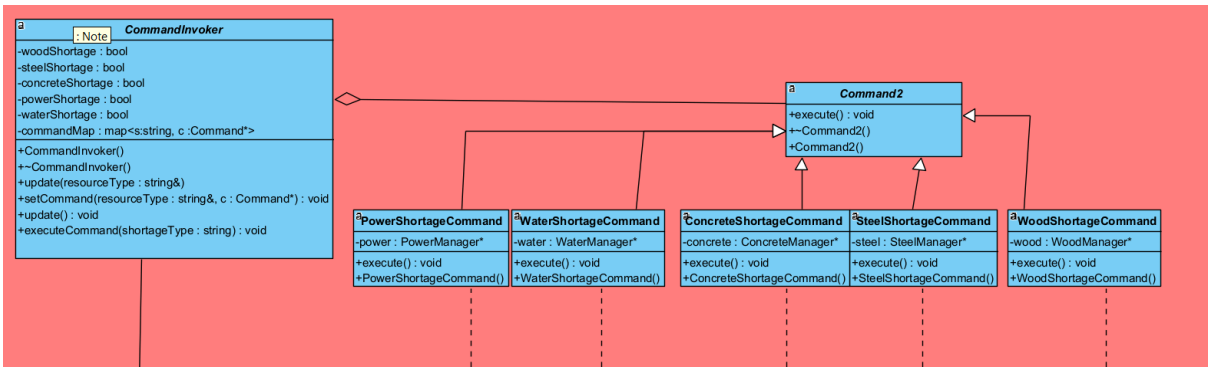
The abstract design pattern was specifically chosen for this project for the sole purpose of creating various building types. Each of these types of buildings fall under a distinct family (these families being Commercial buildings, Residential buildings, Landmarks , and Industrial buildings). This pattern encapsulates the object creation logic which greatly reduces code duplication logic and makes it easier to manage the buildings no matter the amount! By centralising the construction of these building types, adding new structures or extending existing families becomes seamless, allowing for maximum flexibility. Another important thing this pattern achieves is that it decouples the client code from the concrete classes of the buildings which simplifies the maintenance process on the client's side.



The diagram above depicts how the abstract design pattern is being used and how it conveniently allows for the user to create multiple buildings of the same type without any hassle. This pattern utilises the creation process and seamlessly integrates with various other design patterns, without requiring the client to be aware of the underlying complexities.

Command Design Pattern:

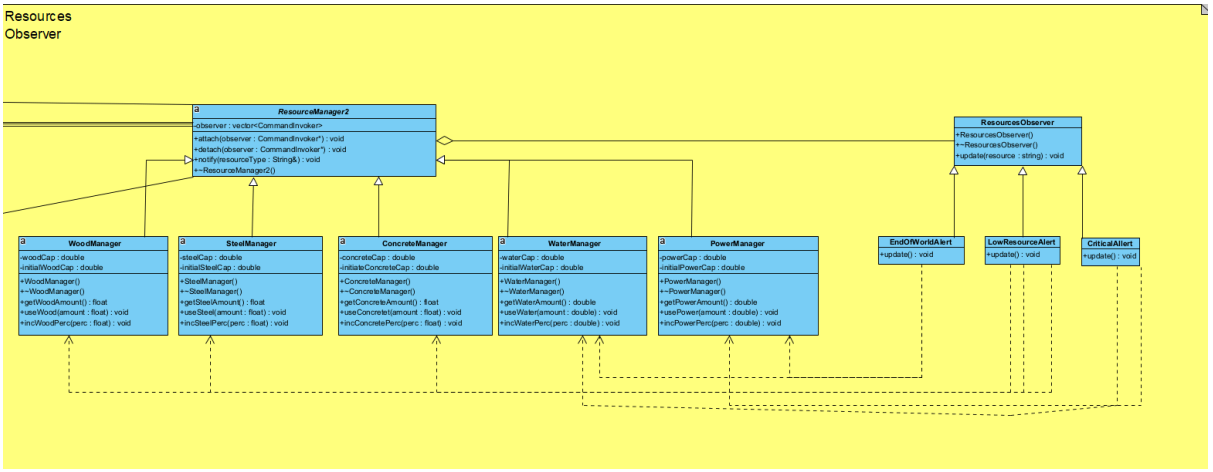
This was one of the design patterns that did not succeed to the final implementation of the program. The primary reason for this decision was that the underlying approach led to circular dependencies, therefore leading to unnecessary complexity.



In this idea, the Command Invoker was not only an invoker, but also the observer for the observer design pattern. The commands would then act as both the commands and concrete observers with the resources as their receiver. This design proved overly complicated and impractical, providing no tangible benefits to go with the added complexity.

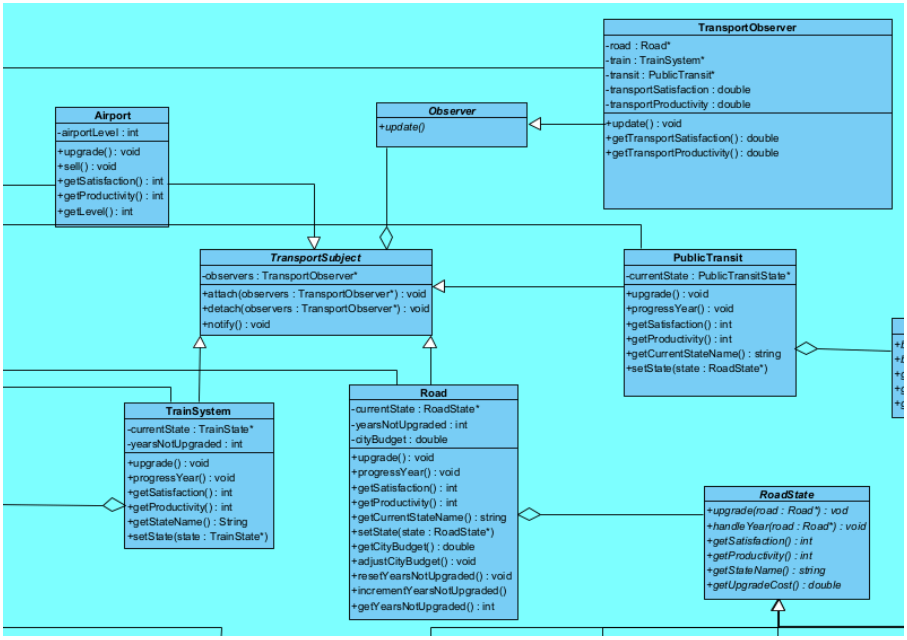
Observer Design Pattern:

The Observer design pattern is implemented in this project to monitor resource levels and trigger notifications when resources fall below critical thresholds. When the state of any resource transitions to a critical level, all subscribed classes (observers) are immediately notified. This notification mechanism ensures that any dependent actions are promptly executed, such as initiating resource conservation measures or triggering automated deliveries to replenish supplies.



The observer design pattern is efficient over here as the resource manager acts as a subject that manages the available resources used by the buildings. These include wood, concrete, steel,water, and power. This same resource manager contains a list of observers which will be notified once resources go below a certain threshold or once they deplete. Depending on what the issue is, the observer class will then implement the update method accordingly.

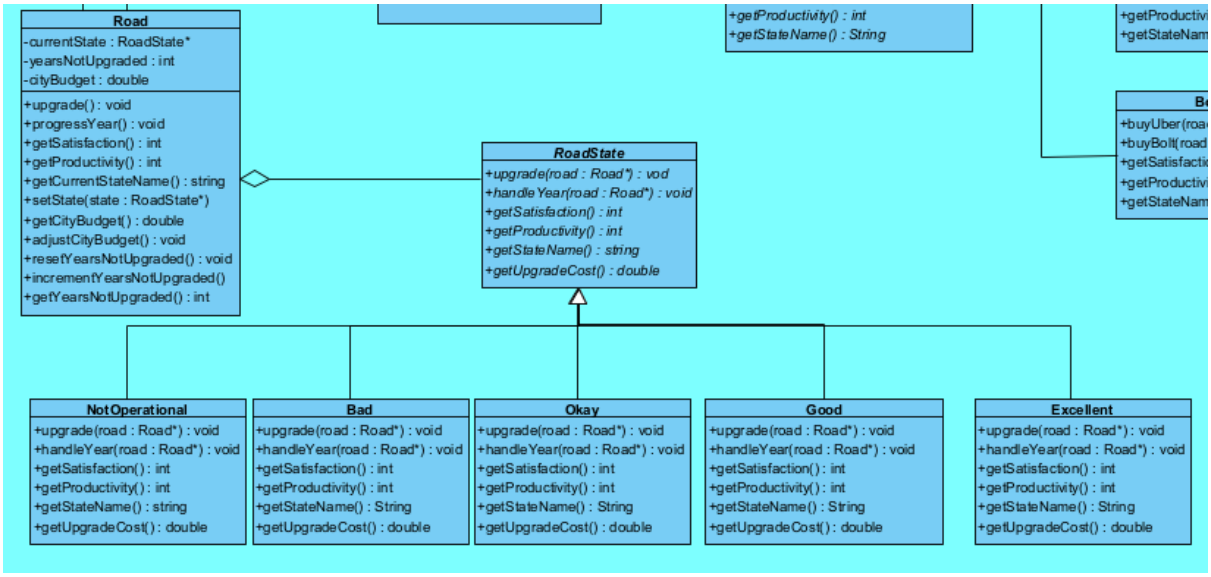
This pattern is also used for the Transportation of the program to detect when there is a state change in any of the transport systems. When a change is detected in the state, a call is made to update the satisfaction and productivity which would have been affected by the change. As can be seen in the snippet of the UML class diagram below, the context of all the State patterns are used as Concrete Subjects for the observers to monitor, and all the updates to the satisfaction and productivity values are done in the Concrete Observer.



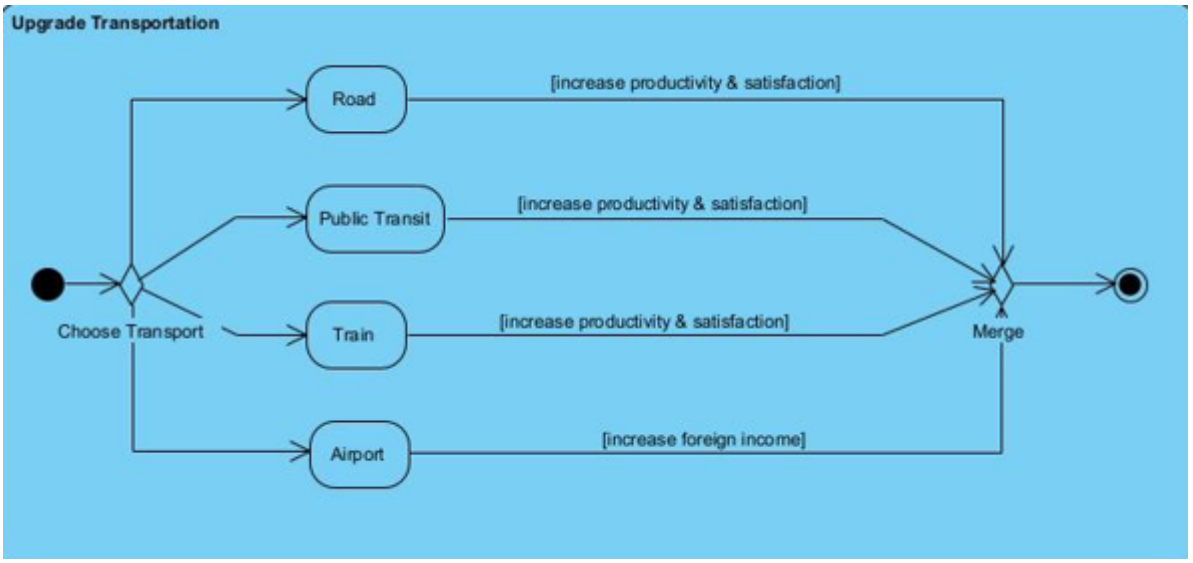
The use of the Observer pattern allows for a well-structured and efficient way to manage dependencies between the resource management system and the rest of the building infrastructure. By decoupling the resource monitoring logic from the specific actions taken in response, the system becomes more flexible and easier to maintain or extend. For example, adding new observers or modifying existing ones can be done without changing the core monitoring logic.

State Design Pattern:

The State design pattern is used extensively in the Transportation system of the project. Roads, trains, public transit and the airport all make use of the state pattern to effectively encapsulate state-specific behaviour. Each of these systems have different “ratings” which are represented as states, and each of these states define the behaviour of the system regarding the productivity, satisfaction, and upgrade cost. The state pattern also simplifies conditional logic to control transitions for upgrades and downgrades. This keeps the system organised, allowing the “ratings” to be adjusted as needed, all while keeping the code flexible and modular. Below is a snippet of how the UML class diagram looks for implementing the state design pattern for the road system.



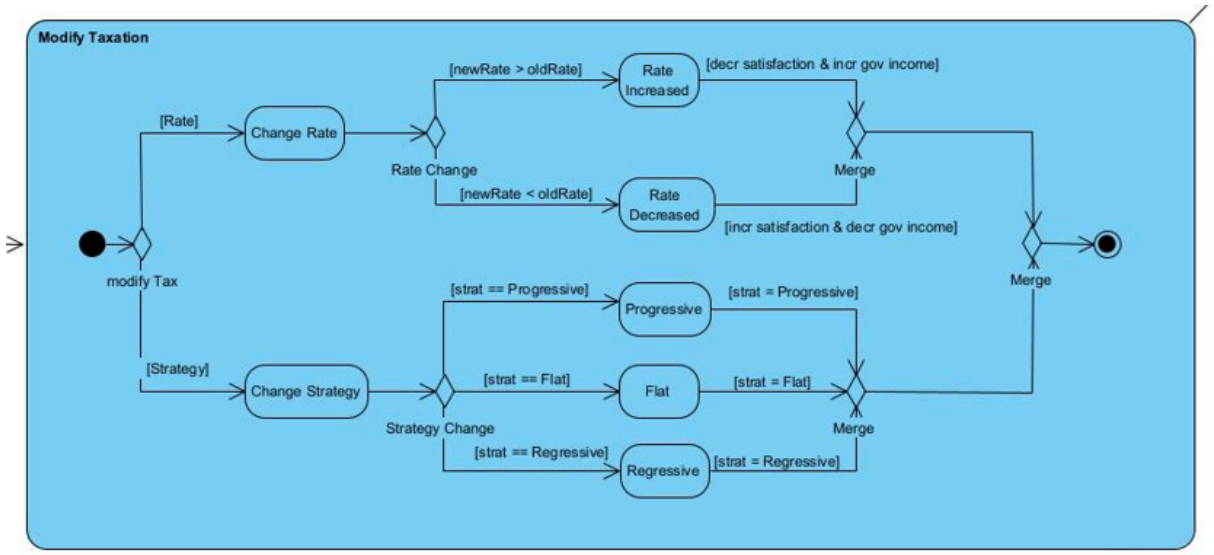
Users have the option to upgrade transportation by spending the city budget and using resources. As a result, the state of the system becomes one better than the current, and the productivity multiplier and citizen satisfaction is increased, or foreign income is increased in the case of the airport. This is depicted by the snippet of the UML activity diagram given below.



Strategy Design Pattern:

Users have the option to choose between different Taxation strategies, therefore the Strategy design pattern was an obvious pick for this taxation system. Three types of taxes make up the Concrete Strategies, including progressive, flat, and regressive tax. Each of these algorithms affect the yearly tax income made by the government depending on the number of citizens there are of each income bracket. This design pattern encapsulates each algorithm used to calculate the tax in its own class, which makes it easy to add, remove or modify algorithms without affecting other parts of the code.

The process which the user follows is represented in the UML activity diagram given below.

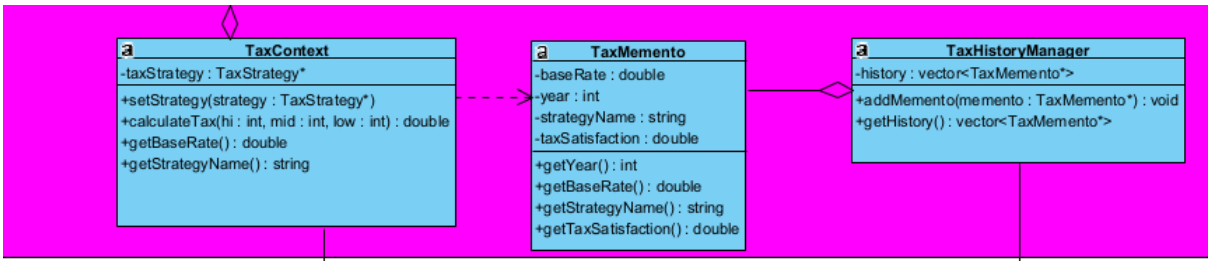


The Strategy pattern allows the user to effortlessly switch between different taxation algorithms so that they can experience which works best for their dictation style.

Memento Design Pattern:

The progression of the city is captured using the memento pattern. This allows the user to analyse their city's growth and compare previous years to their current year. The population, housing units, employment rate, infrastructure, satisfaction, hygiene, and productivity are all recorded and can be restored by the user to see if they are progressing or not.

The Memento pattern is also used in the taxation system which allows the user to restore the different taxation strategies they used over the years. This includes their taxation rate, strategy, and citizen satisfaction for each year. This functionality allows the user to review their past decisions and decide which one benefits the city the most.



The Memento pattern preserves encapsulation by keeping the internal details of an object's state to remain private. The memento object stores the state without exposing or modifying the originator's internal representation. These advantages made the Memento design pattern a suitable choice to implement within these two systems of the project.

Mediator design pattern:

was chosen for the government class in this project to manage communication and coordination between various departments (e.g., Public Services, Infrastructure, Economy, Security) through a central entity, the CityManager. By using the Mediator pattern, we encapsulate interactions between departments within the CityManager, reducing direct dependencies between departments and promoting a more modular structure.

This approach allows each department to communicate through the mediator (CityManager), simplifying the flow of information and the implementation of complex policies. For example, when the Public Services Department needs additional funding or when the Infrastructure Department requires permits, they can notify the CityManager, which then coordinates actions such as fund allocation, policy enforcement, or notifications to other departments.

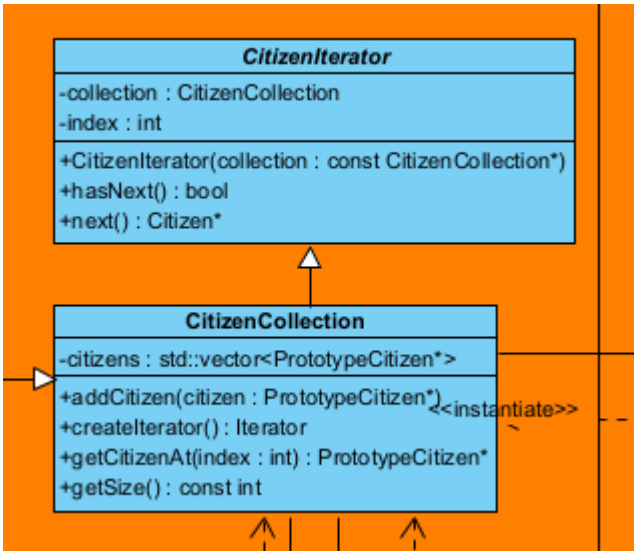
The Mediator pattern also ensures that changes in one department do not directly impact others, keeping the system organized, flexible, and easy to maintain. In case new departments are added or policy adjustments are made, only the mediator needs modification rather than each department, thereby enhancing extensibility.

Iterator Design Pattern:

We chose the Iterator Design Pattern to carefully iterate through each instance of a citizen in a collection, it can be used to change or update a specific citizen by its index and make it much easier to find and delete citizens in the collection where all citizens are stored.

This approach allows the user to easily manage their Citizens in the city if they made any miss inputs or would like to deal with overpopulation. An example is, a user wants to delete 20 Citizens, they can input how many citizens they want deleted and the type they want to remove, it will then iterate through each instance of hasNext in the collection of citizen and remove as many citizens of that type, if there is too little to delete, it will delete all those citizens and end.

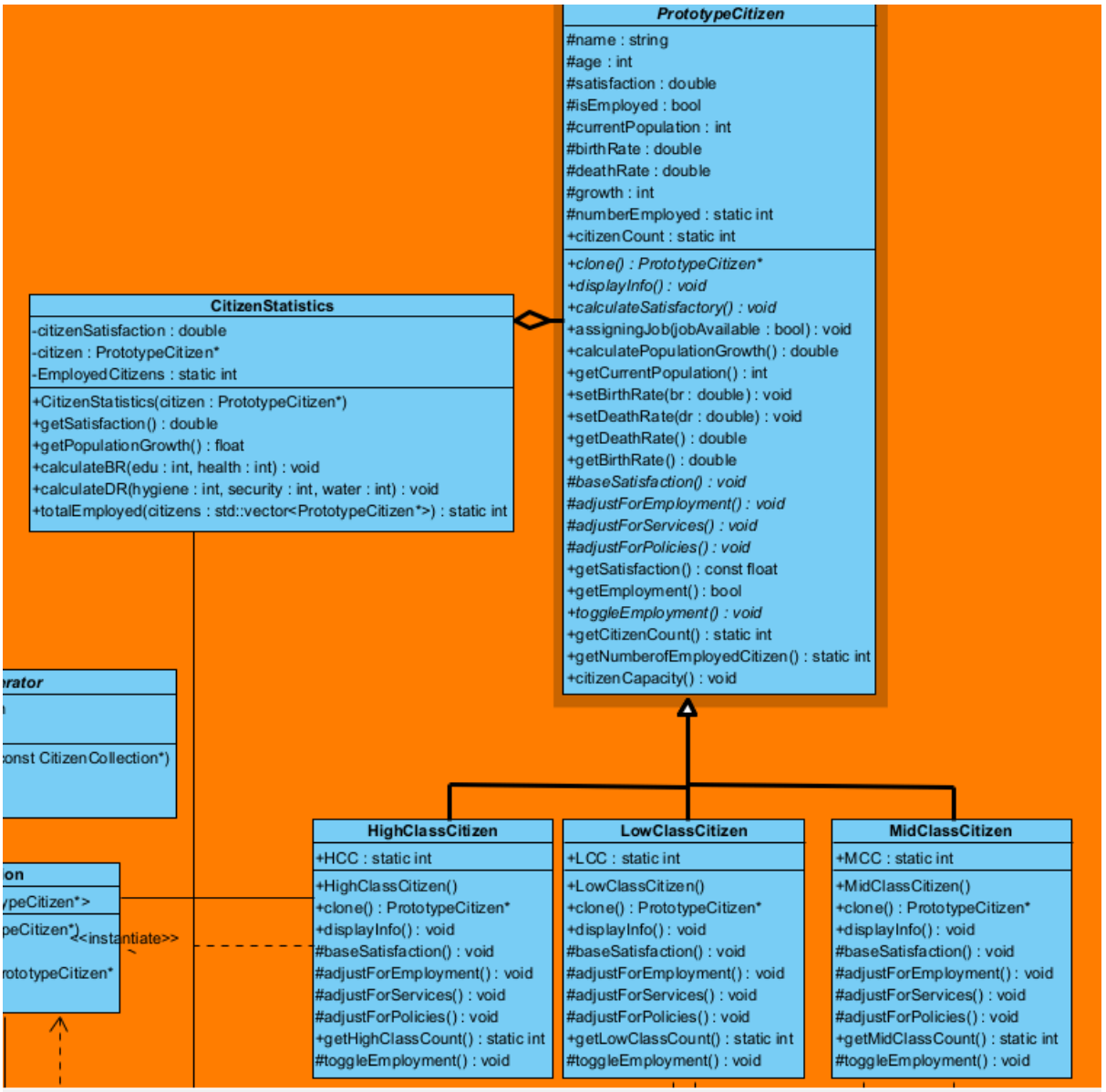
This implementation allows safety with management of components and resources for the user.



Template Design Pattern:

Template Design Pattern is used in the creation of Citizens as it is best used here,in our design we have 3 different types of Citizen with same implementation, but their outcomes are different to the other, this is where Template was chosen, to calculate the satisfaction of each citizen of different type, they had to have their own function based for them.

This improves scalability as if there were to be added a new type of Citizen, it can be easily incorporated. It also allows different design patterns to be incorporated and improve that patterns performance by removing their issues through template use.

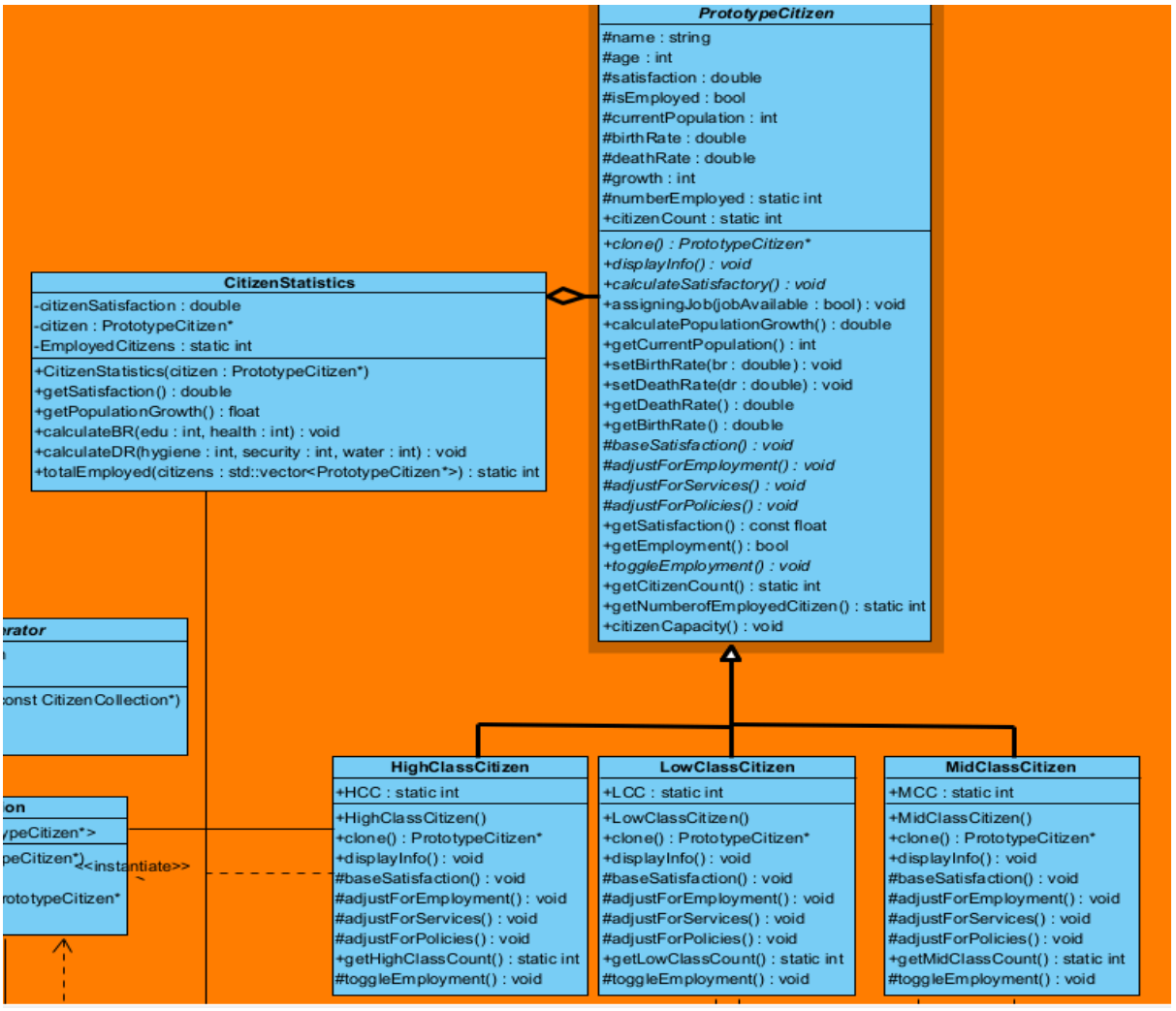


Prototype Design Pattern:

Prototype Design Pattern was used to create Citizens in an infinite way for the user, we create a base Citizen and expand it to its classes through cloning, this is where an iterator is used to create multiple Citizens for the user, as many as the user wants. We chose Prototype over Factory because there were special implementations that were needed for citizens to fully be utilised in the system, with template methods included, it guaranteed that goal we were looking for.

The benefit of Prototype is its simplicity, as the user can easily create or remove citizens as they please, with no issues on any other parts of the simulation. It's also a lot easier to manage as Citizens do not need names of any kind, their only useful use is their satisfaction and age for birth rate and death rate. Which is much more achievable for Prototype. This makes the User have less effort to be made and makes the simulation more user friendly.

We also store the data of Citizen into another class which knows that Prototype exists. Makes data a lot easier to be accessed in runtime.



Facade Design Pattern:

The Facade pattern provides a simplified interface to a complex subsystem, allowing clients (in this case the `main.cpp`) to interact with the subsystem without needing to understand its complexities. Manager classes, the subsystems, each manage a particular section, effectively breaking down the complexity of the project into distinct areas of responsibility. A single interface is used to interact with these managers and hide the intricate details of each subsystem.

This simplifies the main interface by having only interact with a few high-level manager classes, instead of directly managing every single component and design pattern. This keeps the main file cleaner, reducing its responsibilities and making it easier to maintain. Each manager class handles specific functionality, and encapsulates the behaviours and patterns of its respective area, enhancing the code's modularity.

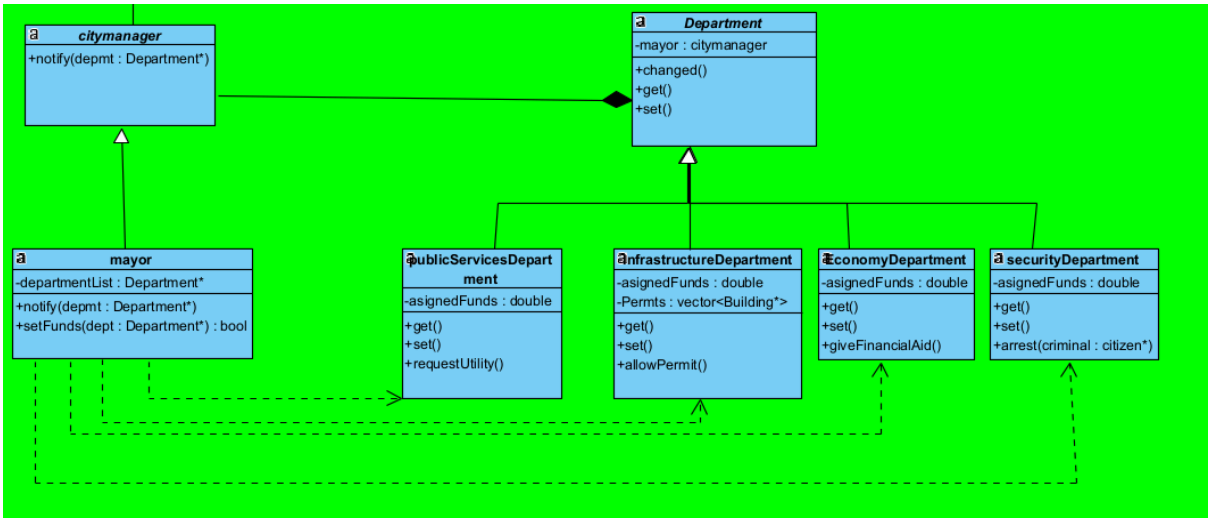
This approach creates a layered structure where the main files only sees a high-level, simplified interface, while each manager (subsystem) class hides the underlying complexities.

Singleton Design Pattern:

The Singleton Design Pattern was chosen because it offers an optimal solution for implementing a government administration system, especially through mayor where we create instances that exist throughout the systems life cycle. It allows those instances to do different types of classes from the Department and all inherited classes have their own functionality.

This guarantees centralised control over policies and service decisions and resource allocation, preventing potential conflicts that can arrive through multiple instances and resource management issues.

The singleton Design Pattern allows global use, allowing other classes to use, which is what our goal is for Government.



References

Andrews, E. (2024) *The economic impact of airports on local communities*, *LinkedIn*. Available at: https://www.linkedin.com/pulse/economic-impact-airports-local-communities-elias-double-a-andrews-rwhxc#:~:text=The%20significance%20of%20airports%20extends,of%20airports%20has%20evolved%20dramatically. (Accessed: 01 November 2024).

gfoa (2017) *Local Government revenue sources - cities*, *Government Finance Officers Association*. Available at: https://www.gfoa.org/revenue-dashboard-cities (Accessed: 01 November 2024).

Wilcox, Z. and Nohrová , N. (2014) *Transport essential for growth in cities*, *Centre for Cities*. Available at: https://www.centreforcities.org/reader/delivering-change-making-transport-work-for-cities/transport-essential-growth-cities/ (Accessed: 01 November 2024).

Zhang, F. *et al.* (2024) *Effects of urban landmark landscapes on residents' place identity: The moderating role of Residence duration*, *MDPI*. Available at: https://www.mdpi.com/2071-1050/16/2/761#:~:text=Landmark%20landscapes%2C%20as%20visual%20representations,urban%20character%2C%20and%20promotes%20tourism. (Accessed: 01 November 2024).