# REPORT

Urban development | City management principles | City Components

# Research Brief

Urban development focuses on improving cities through infrastructure, housing, and utilities to meet the needs of growing populations. Economic growth and political decisions influence city planning. Infrastructure, including transportation systems, public services, and utilities, is key to supporting the functionality and growth of urban areas.

## Key Principles of City Management

- **Infrastructure development** that is well coordinated to accommodate population growth without overloading resources
- **Acquire sufficient funding** to finance running costs and new infrastructure
- **Develop accountable governance and craft dedicated policies** in important areas of the city

## Sustainable Economic Growth

- **Citizen Well-Being and Public Services** which involves enhancing quality of life for citizens by providing services like healthcare. Which impact citizen satisfaction which is essential for successful urban management
- **Metropolitan management** which involves managing growth at a large scale.

## Role of various components in a city

- The quality of a city’s infrastructure directly affects its state of economic and social development
- Terminals such as: ports, airports and rail yards are highly localized entities with access points supporting high traffic levels.
- Logistic zones including warehouses are associated with distribution and light manufacturing activities.

- Commercial districts are a core component of urban centrality. They concern consumer-related freight movements such as retail activities through delivery vans and trucks. The clustering of offices and large institutions such as the government is also a large generator in freight demand. (Rodrigue, 2024)

## Assumptions

- The map has a predefined fixed dimensions of [25 x 25].
- The map represents the plot of land occupied by the buildings.
- Each building on the map is a unique component, no two buildings can have the same location coordinates.
- The map is divided into predefined sections such that buildings of different types cannot occupy the same section.
- There is a limit to the number of buildings a player can create
- Each citizen has a satisfaction score and this score is out of 100
- We assume the user will want to play our game!
- We assume only one device plays the game locally
- It is up to the player to decide what to do like create or demolish etc.

# Design Influence

In creating the city builder simulator in C++, the components outlined above shaped my design choices by emphasizing the importance of infrastructure and land use planning. Each component's role guided the simulation's logic, ensuring realistic interactions between different types of buildings and their impact on urban dynamics. This structure enabled me to model effective city management strategies and promote sustainable development within the simulation.

**Buildings and Infrastructure**: Each building type (residential, commercial, industrial, landmarks) in the simulation is represented as a unique object. These objects have properties like size, resource consumption, and impact on citizen satisfaction, which allows for dynamic interactions and scalability. The use of inheritance and polymorphism in C++ enables easy addition of new building types as the simulation grows.

- **Utilities**: The utility systems (water, power, sewage, waste management) are designed as separate modules. Each utility impacts various buildings and citizens in distinct ways. For example, a power plant object provides electricity to nearby residential and industrial buildings, which can be modeled as interactions between classes. This modularity ensures that any changes in one utility (e.g., increasing power generation capacity) can be easily reflected in the simulation without affecting the overall structure.
- **Transportation Systems**: Roads, public transit, and rail systems are implemented as interconnected objects. Their efficiency depends on the strategy used, affecting traffic flow and citizen satisfaction. For example, a poorly planned road network might lead to traffic congestion, which directly influences citizen happiness and economic performance. This interdependency is modeled by updating the

transportation system in the simulation loop and analyzing its impact on traffic patterns.

- **Citizen Needs & Satisfaction**: Citizens respond to changes in the city's infrastructure, policies, and available services (e.g., healthcare, education). A key feature of the design is a **Citizen Satisfaction Module**, which evaluates the overall happiness of the population based on factors like job availability, public services, taxes, and environmental conditions. If a building's proximity to essential services (such as hospitals or schools) decreases, citizen satisfaction drops, influencing the population's growth and migration patterns.

**DESIGN DECISIONS**

**FACTORY**

### 4.3.2 Structure

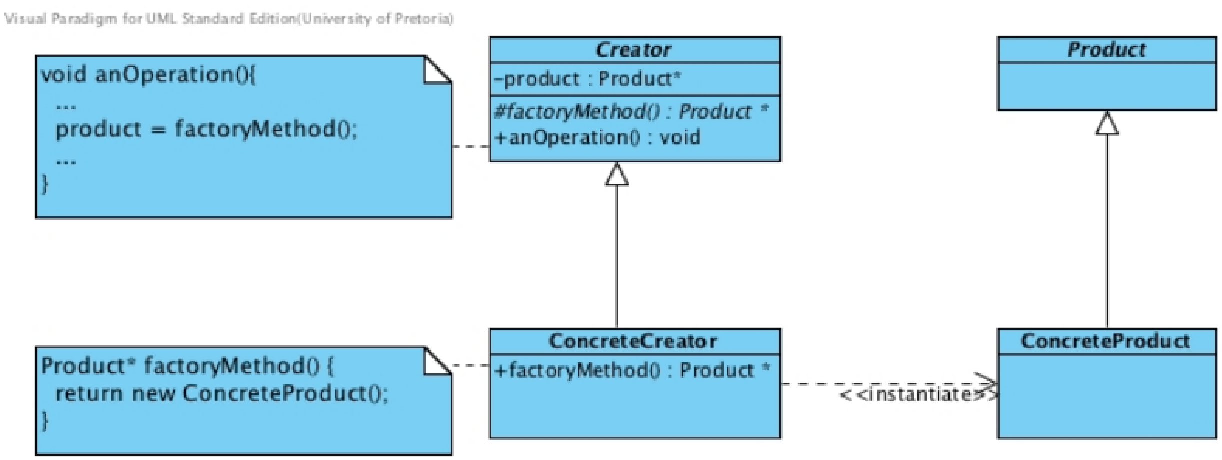
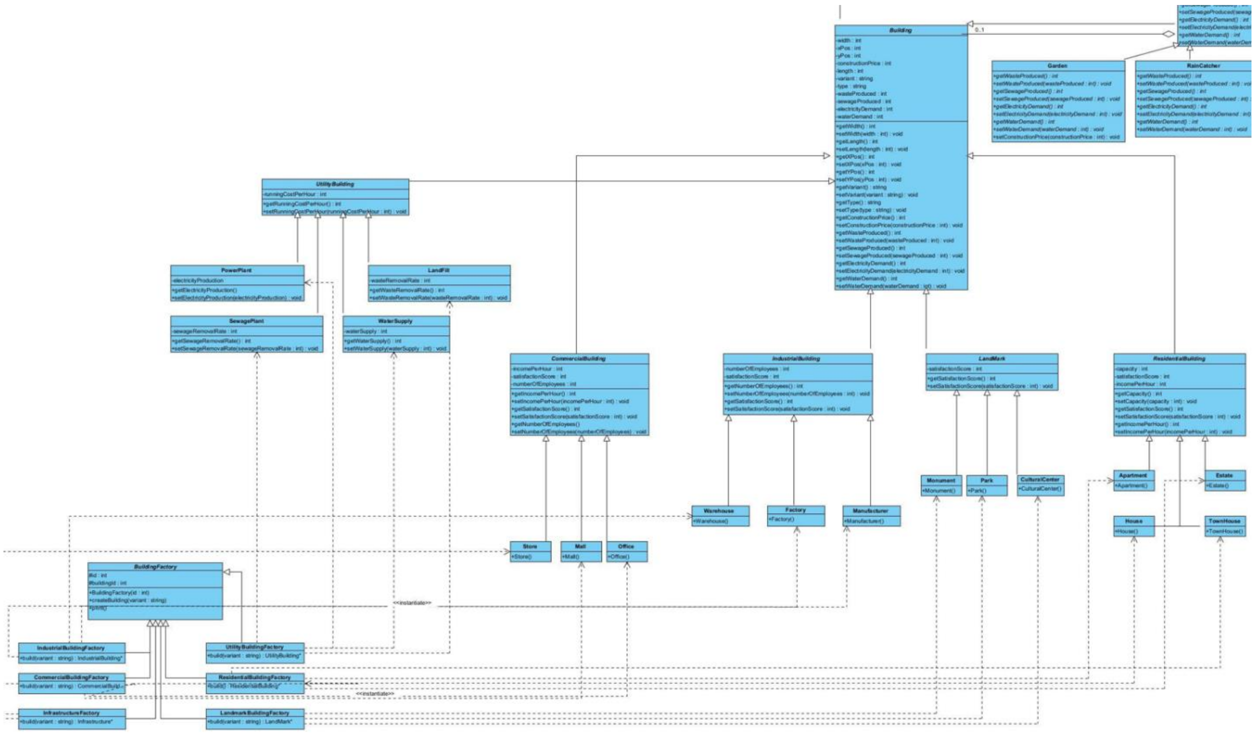



Figure 2: The structure of the Factory Method Pattern




**Reason for use:**

Used in Buildings to efficiently create different types. Pattern allows BuildingFactory to generate building instances based on specific needs and provides flexibility in creating diverse building structures with unique attributes and behaviours, supporting the simulation's scalability and variety.

- **Creator:** BuildingFactory.
- **ConcreteCreators:** IndustrialBuildingFactory, UtilityBuildingFactory, CommercialBuildingFactory, ResidentialBuildingFactory, InfrastructureFactory, LandMarkBuildingFactory**.**
- **Product:** Building.
- **ConcreteProducts:** IndustrialBuilding, UtilityBuilding, CommercialBuilding, ResidentialBuilding,CommercialBuilding,ResidentialBuilding,Infrastructure,Landmark.

**DECORATOR**

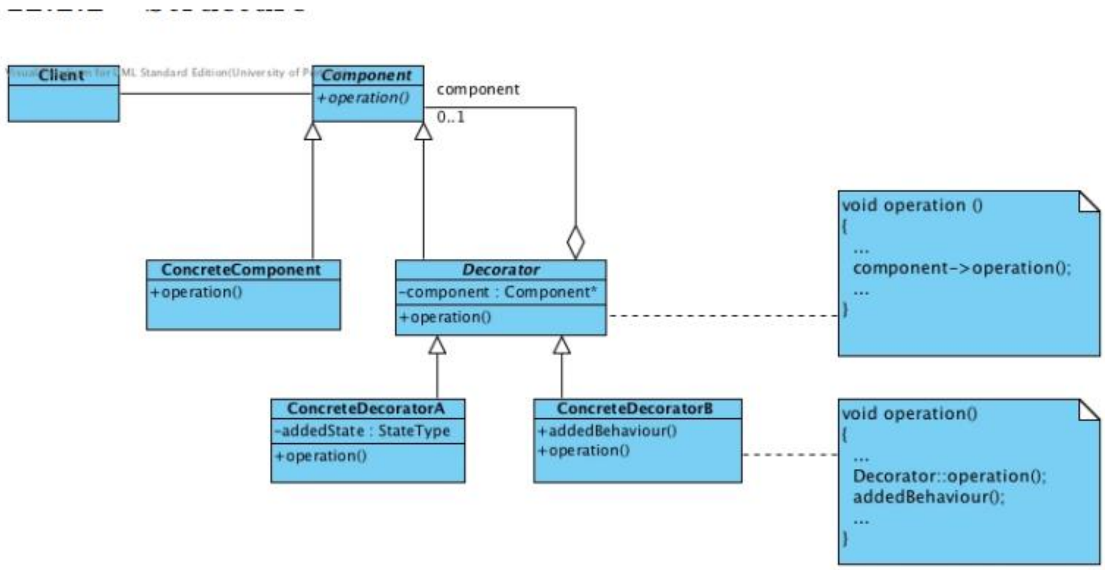
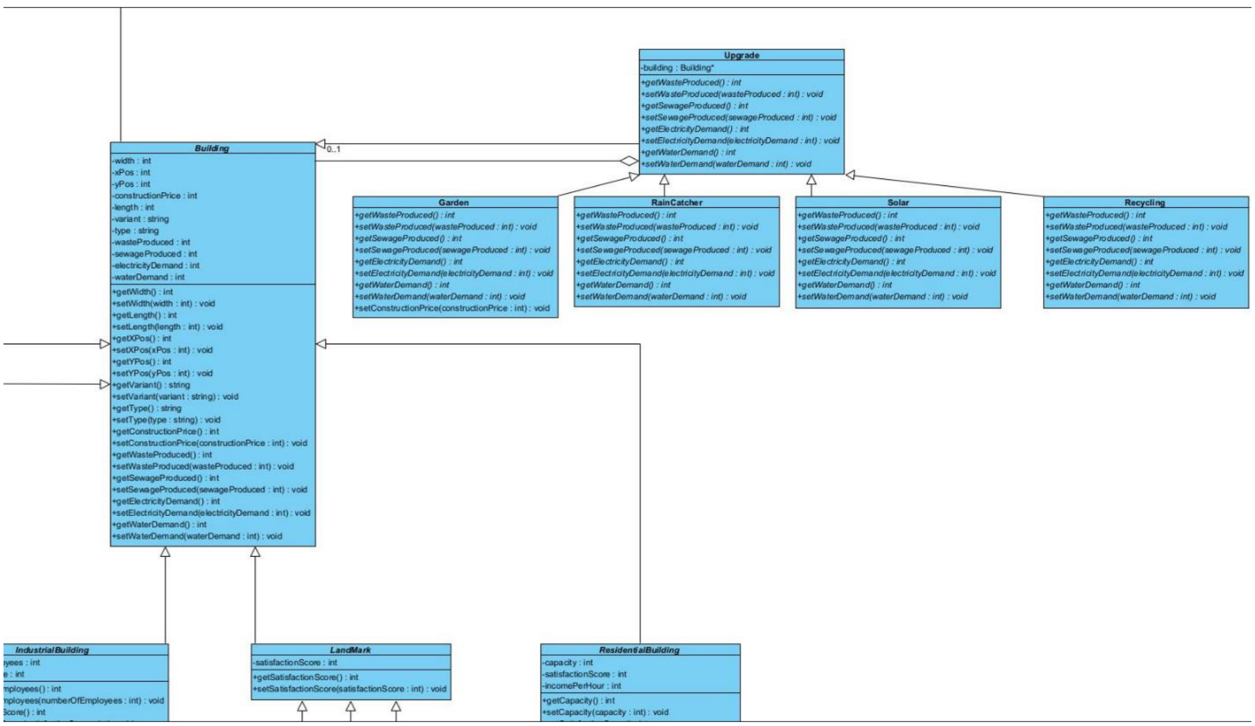



Figure 1: The structure of the Decorator Pattern




**Reason for use:**

Used to add optional features to a Building without altering its core structure. By applying decorators like Garden, RainCatcher, Solar, and Recycling, buildings can be "upgraded" with new functionalities in a flexible way. This allows for dynamic customization of buildings in the city simulation, enhancing reusability and maintainability.

- **Component:** Building
- **Decorator:** Upgrade
- **Concrete Decorators:** Garden, RainCatcher, Solar, Recycling
- **Concrete Components:** Commercial Building, Residential Building, LandMark, Residential Building

**BUILDER**

## 22.2 Builder Design Pattern

### 22.2.1 Identification

| Name | Classification | Strategy |
|---|---|---|
| Builder | Creational | Delegation |
| **Intent** | | |
| *Separate the construction of a complex object from its representation so that the same construction process can create different representations.* ([1]:97) | | |

### 22.2.2 Structure

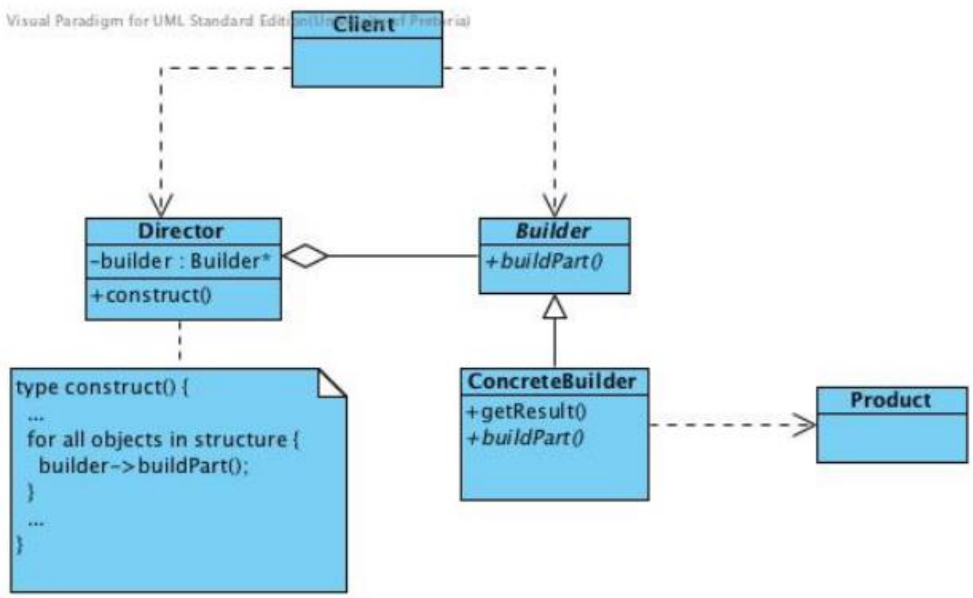
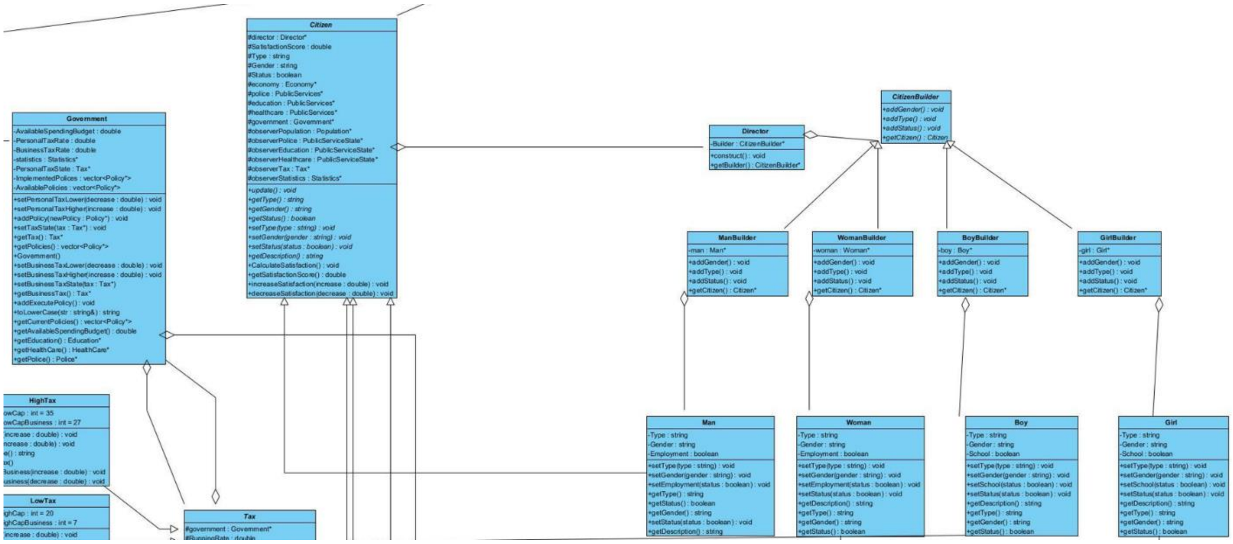



Figure 1: The structure of the Builder Design Pattern



**Reason for use:**

Used in **Government** to create complex citizen objects with varying attributes efficiently. Builder enables flexible and controlled construction of diverse citizen types, simplifying the customization process and supporting scalability in managing different demographics within the city.

Participants:

- **Client:** Government.
- **Director:** Director
- **Builder:** CitizenBuilder
- **ConcreteBuilders:** ManBuilder, WomanBuilder, BoyBuilder, GirlBuilderProduct: Man, Boy, Woman, Girl

**STATE**

**<u>The benefits of State over Strategy</u>**

The Strategy pattern differs the implementation to achieve the same result whereas the State pattern allows us to accomplish different actions based on the current state of the object..

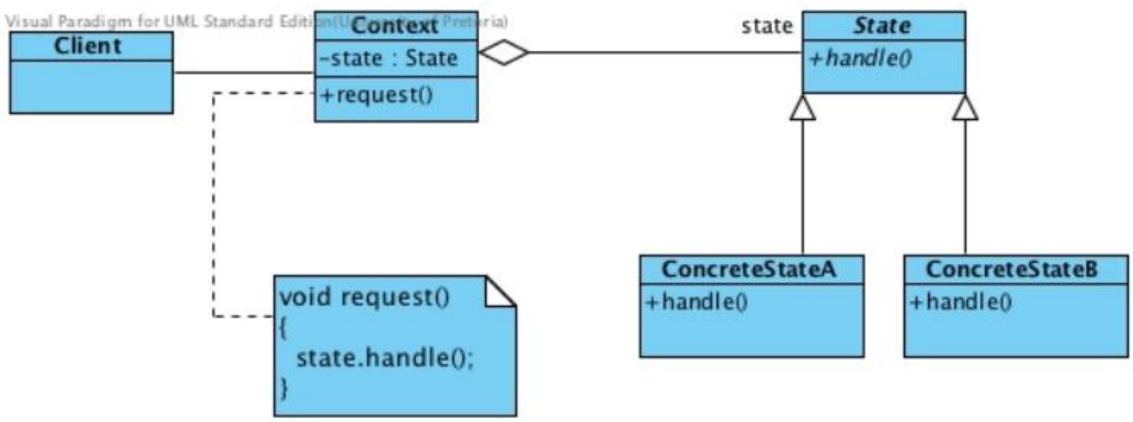
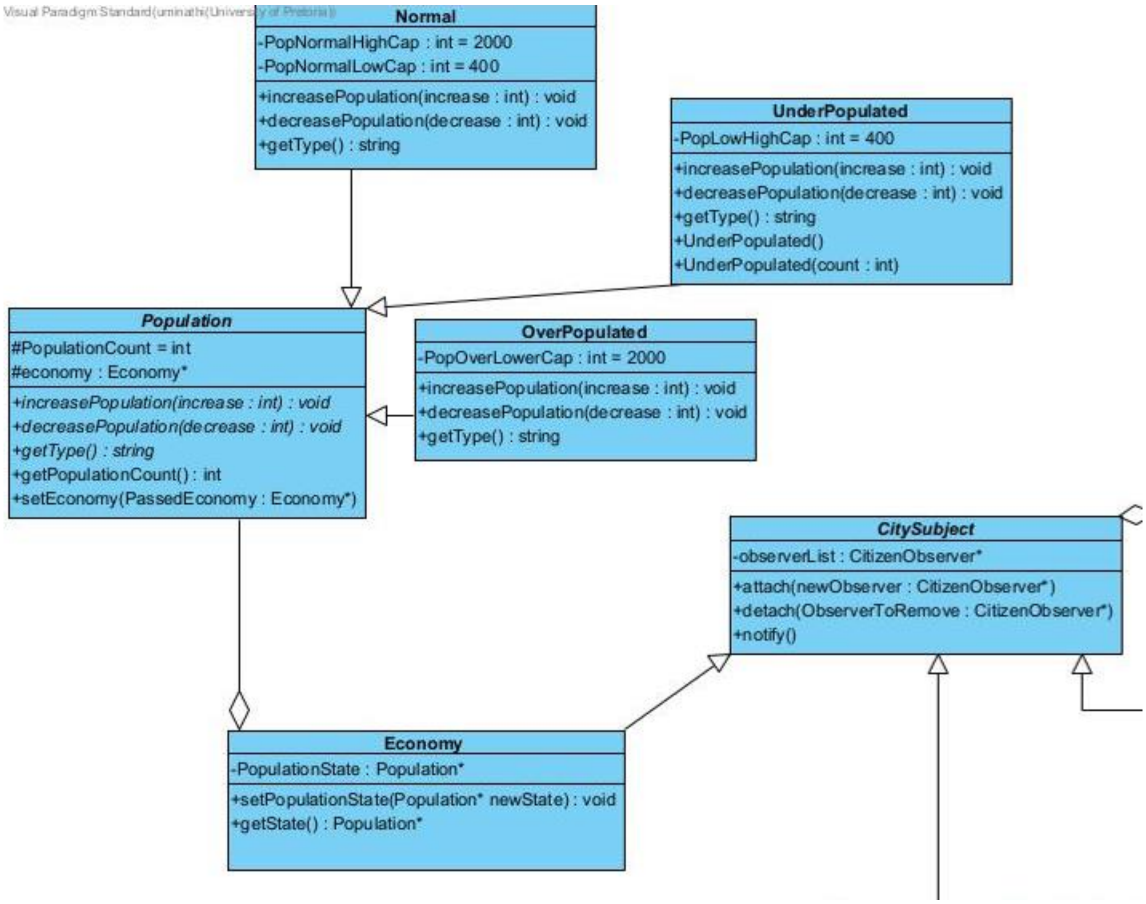
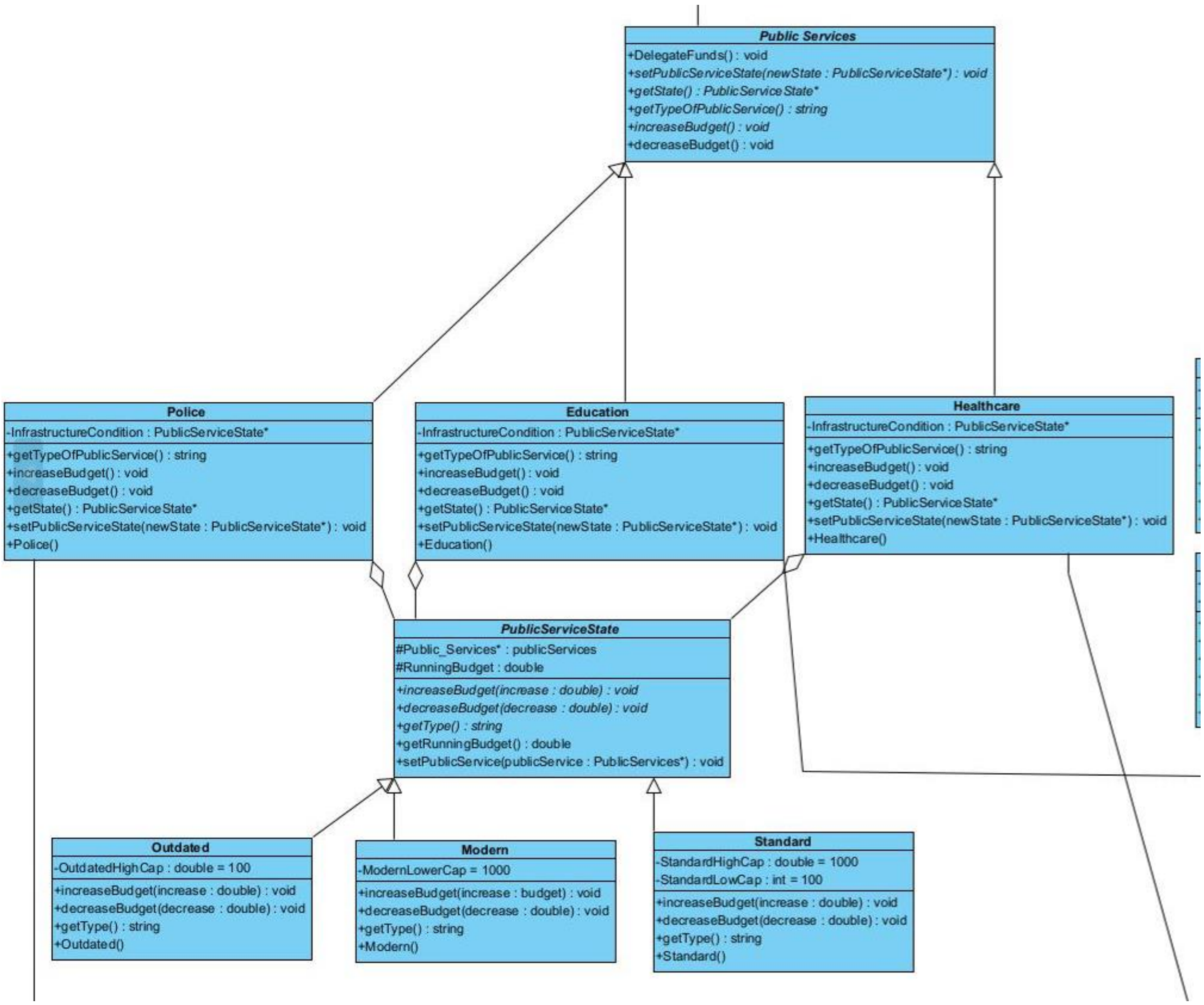



Figure 1: The structure of the State Design Pattern


Visual Paradigm Standard(uminathi(University of Pretoria))
Normal
-PopNormalHighCap : int = 2000
-PopNormalLowCap : int = 400
+increasePopulation(increase : int) : void
+decreasePopulation(decrease : int) : void
+getType() : string
UnderPopulated
-PopLowHighCap : int = 400
+increasePopulation(increase : int) : void
+decreasePopulation(decrease : int) : void
+getType() : string
+UnderPopulated()
+UnderPopulated(count : int)
Population
#PopulationCount = int
#economy : Economy*
+increasePopulation(increase : int) : void
+decreasePopulation(decrease : int) : void
+getType() : string
+getPopulationCount() : int
+setEconomy(PassedEconomy : Economy*)
OverPopulated
-PopOverLowerCap : int = 2000
+increasePopulation(increase : int) : void
+decreasePopulation(decrease : int) : void
+getType() : string
CitySubject
-observerList : CitizenObserver*
+attach(newObserver : CitizenObserver*)
+detach(ObserverToRemove : CitizenObserver*)
+notify()
Economy
-PopulationState : Population*
+setPopulationState(Population* newState) : void
+getState() : Population*




**Reason for use:**

Used in **Citizens** and **Taxes** to manage population dynamics and different taxation levels. Pattern allows the simulation to adjust behaviour and policies based on population levels, enabling adaptive responses to changes which impact citizen satisfaction and city growth. Also allows the system to dynamically adjust tax behaviour based on current state,

- **Context:** Economy | Government
- **State:** Population | Tax
- **ConcreteStates:** Normal, UnderPopulated, OverPopulated | HighTax, LowTax, StandardTax

## OBSERVER

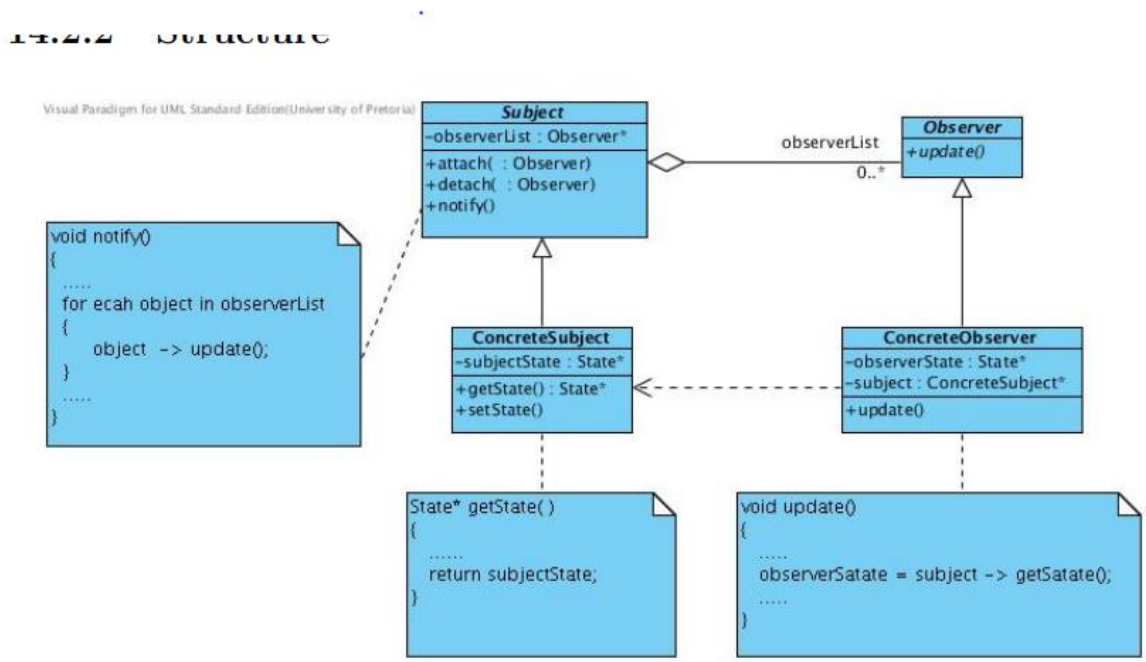
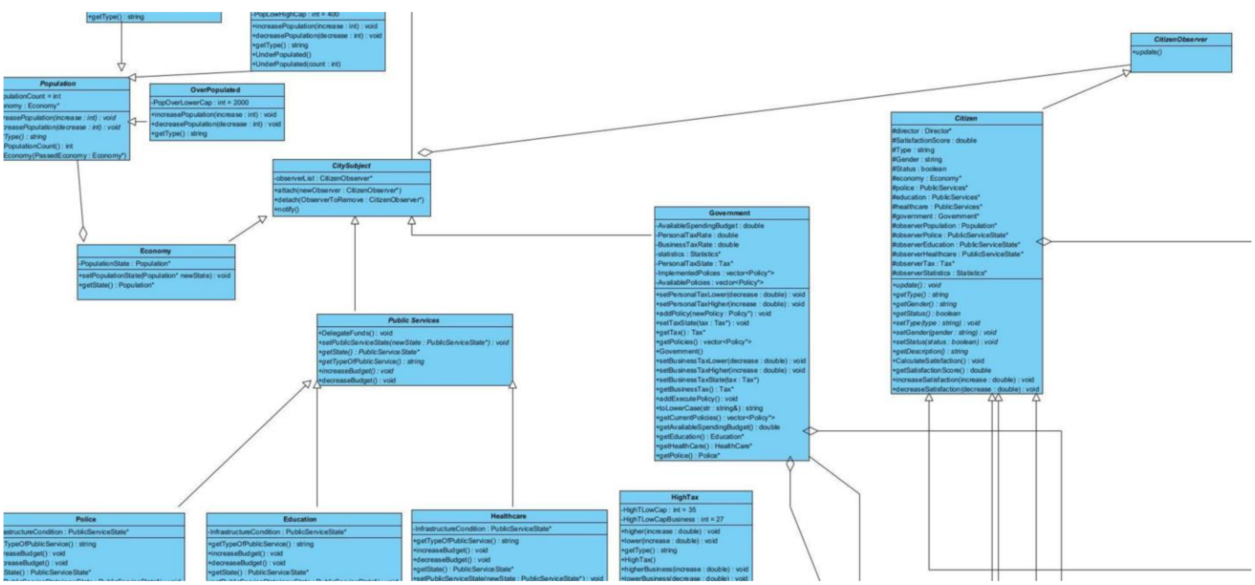



Figure 1: The structure of the Observer Pattern



**Reason for use:**

Used in the **Citizens** and **Utilities** to keep citizens informed of changes in city aspects. Pattern allows CitySubject to notify CitizenObserver whenever relevant changes occur, enabling citizens to react accordingly. It provides a decoupled way for citizens to dynamically respond to government policies and economic or infrastructural changes, enhancing the simulation’s realism.

- **Subject:** CitySubject
- **ConcreteSubject:** Economy, Police, Education, Healthcare, Government
- **Observer:** CitizenObserver

**ConcreteObservers:** Citizen

**COMMAND**

### 18.2.3 Structure

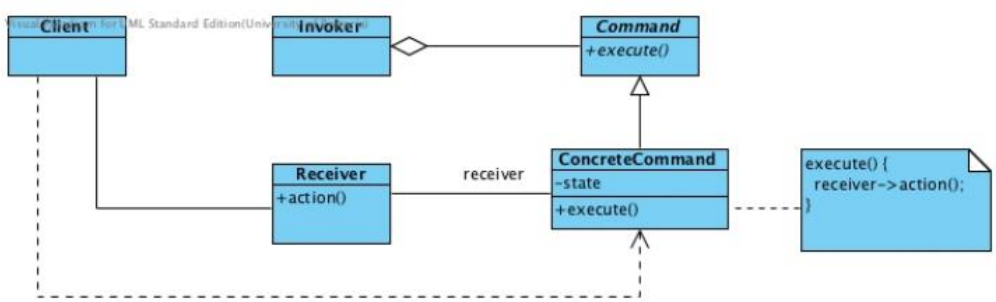
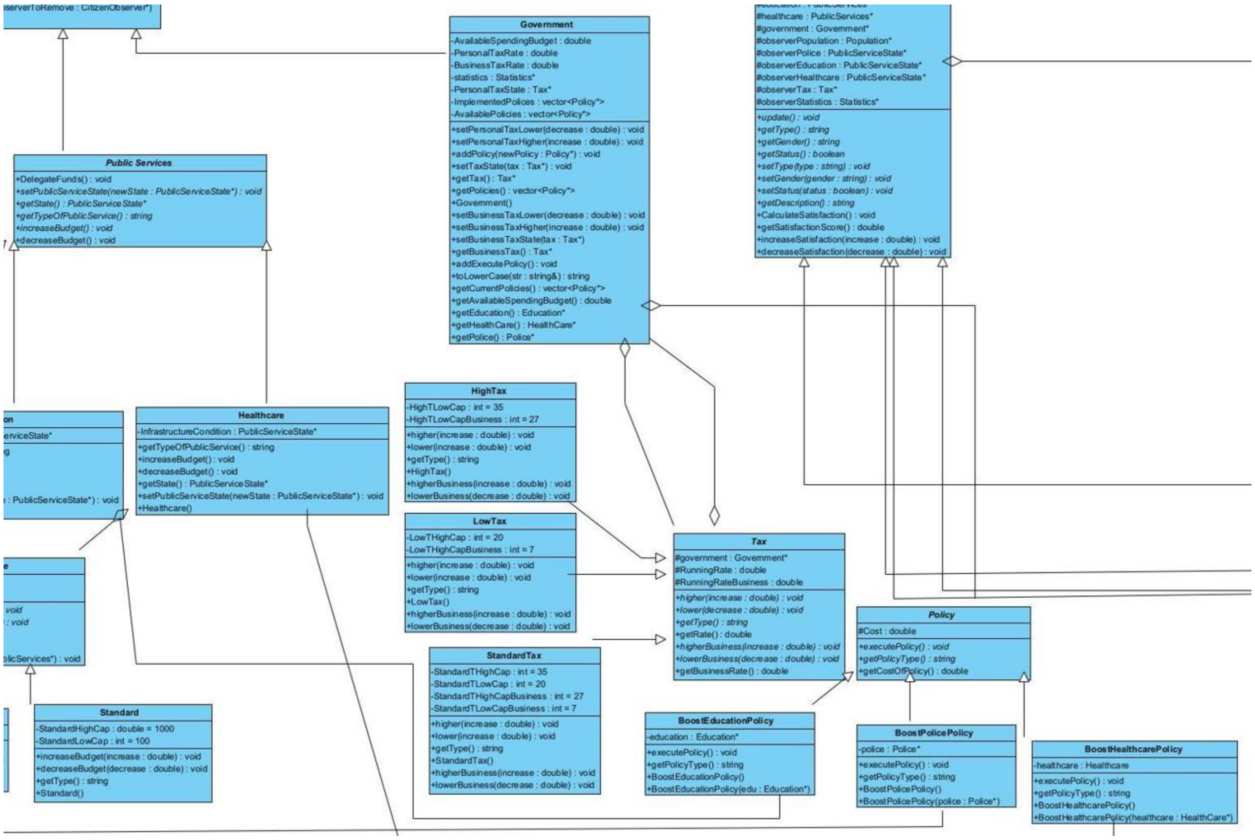



Figure 1: The structure of the Command Design Pattern




**Reason for use:**

Used in **Government** to handle policy actions like BoostEducationPolicy etc. Pattern allows the government to issue commands that execute specific policies on different sectors independently. Enables flexible and decoupled policy management, making it easy to add, remove, or change policies without altering the government or sector implementations.

- **Command:** Policy.
- **ConcreteCommands:** BoostEducationPolicy, BoostPolicePolicy, BoostHealthcarePolicy.
- **Invoker:** Government.

**Receiver:** Police, Education, Healthcare.

## SINGLETON

### Structure

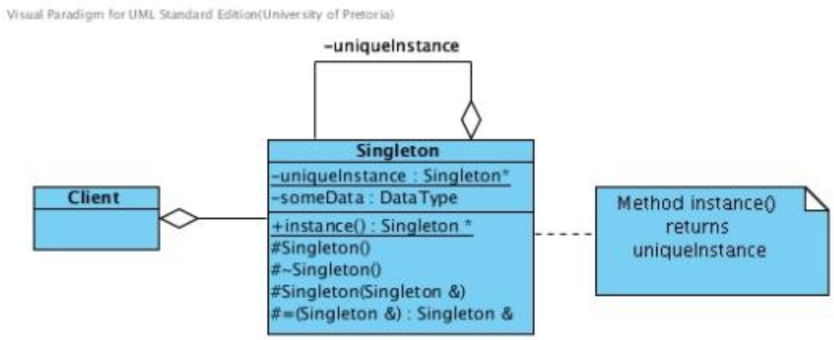
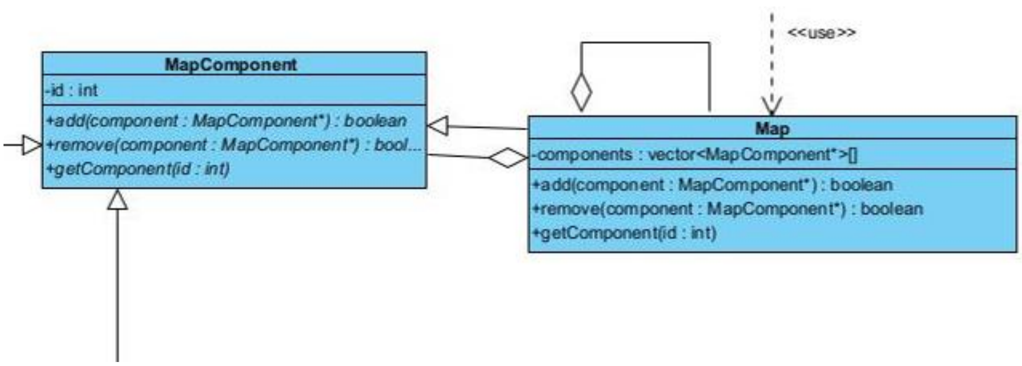



Figure 2: The structure of the Singleton Design Pattern




**Reason for use:**

The Singleton Pattern was used in the **Map** component to ensure there is only one instance of the Map throughout the simulation. This allows centralized access to the city's layout and coordinates, preventing conflicts or duplication, and making it easier to manage and reference the city's state consistently across different components.

**Participants:**

- **Singleton:** Map

- **Client:** MapComponent

**FAÇADE**

### 25.2.3 Structure

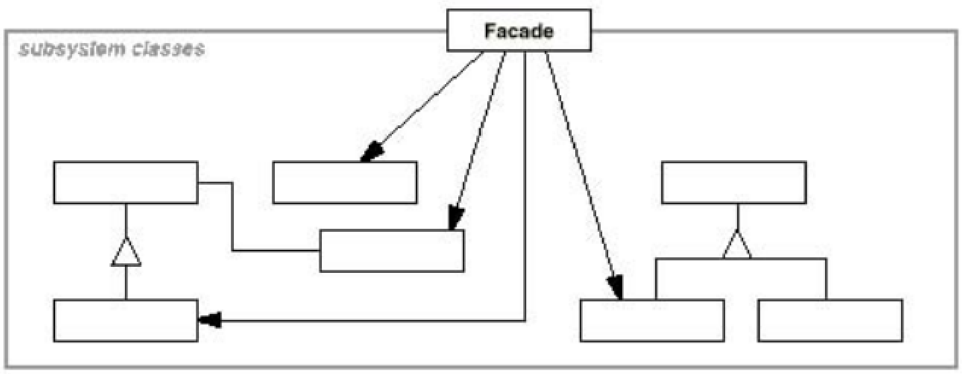
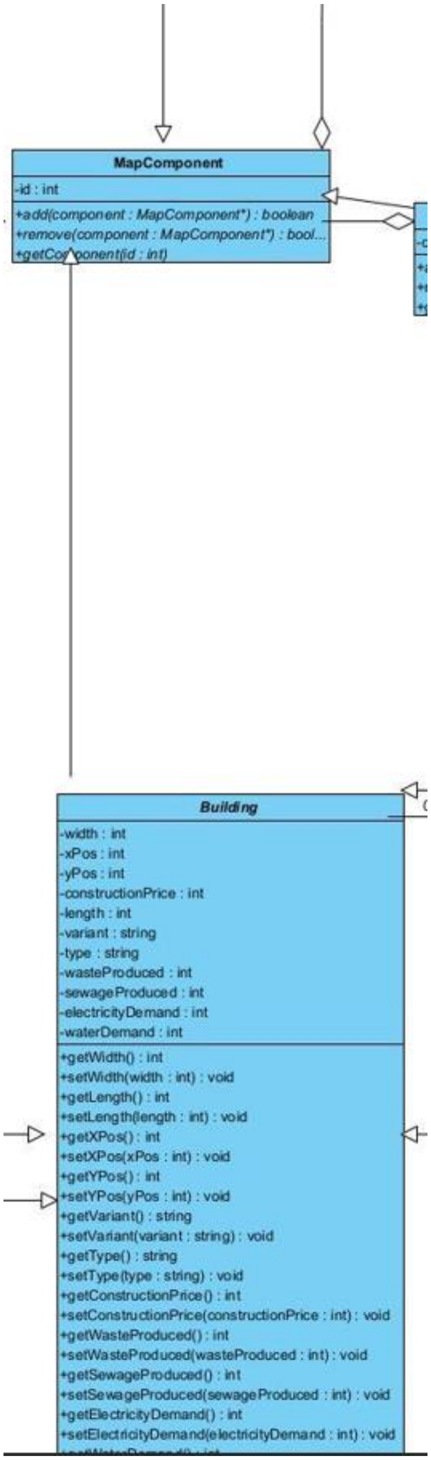



Figure 1: The structure of the Façade Design Pattern




**Reason for use:**

Used in the **MapComponent** to provide a simplified interface for interacting with complex subsystems like Buildings, Infrastructure, CityBuilder, and TransportationServices. Pattern allows other components to interact with the city's infrastructure easily without needing to understand the underlying complexities of each subsystem, improving modularity and ease of use.

**Participants:**

- **Facade:** MapComponent
- **Subsystem Classes:** Buildings, Infrastructure, CityBuilder, TransportationServices

**COMPOSITE**

## 11.3.2 Structure

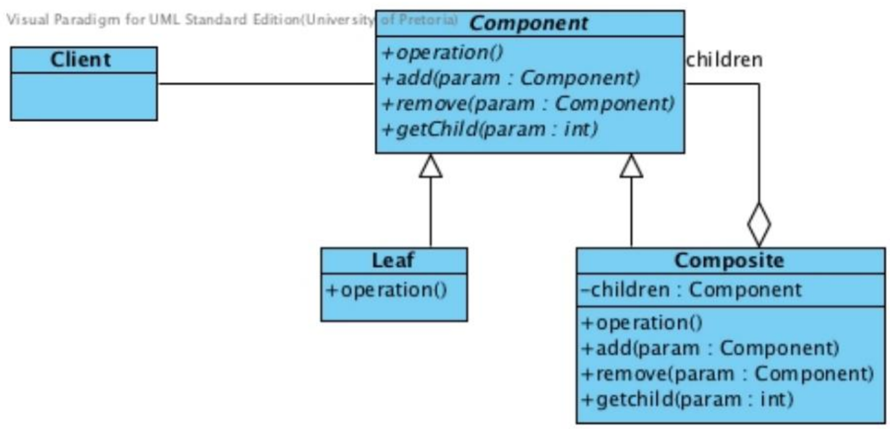
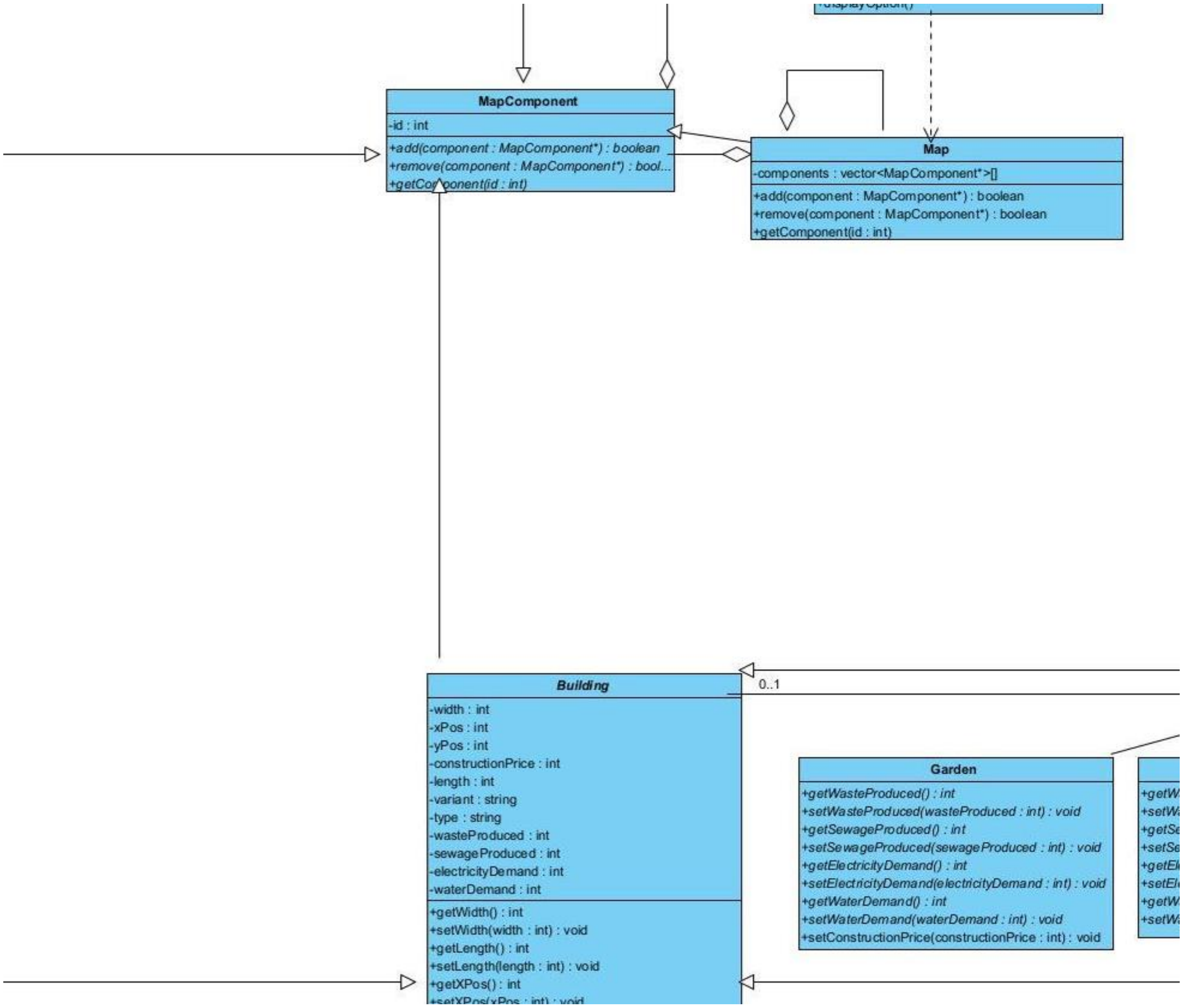



Figure 1: The structure of the Composite Pattern




**Reason for use:**

Used in the Map to organize MapComponent objects, allowing Map to treat individual elements and groups of elements uniformly. Pattern enables hierarchical structures where Map can manage both single components and collections, making it easier to handle complex interactions and support scalable city layouts within the simulation.

**Participants:**

- **Component**: MapComponent.
- **Leaf:** Building, Citizen.

**Composite**: Map.

## STRATEGY

### 8.3.2 Structure

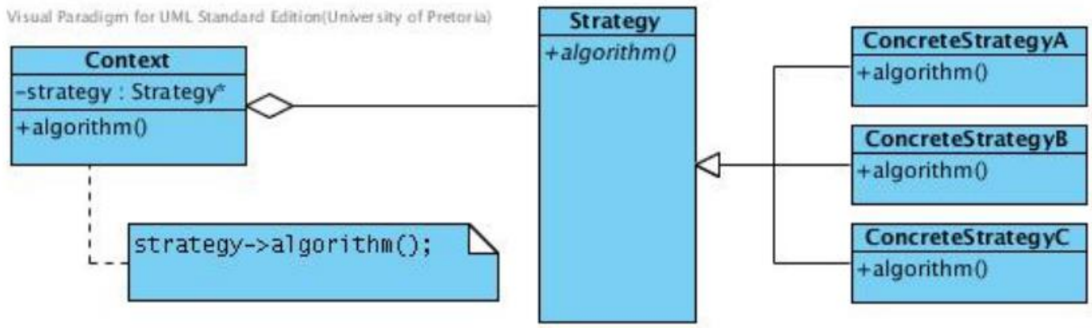
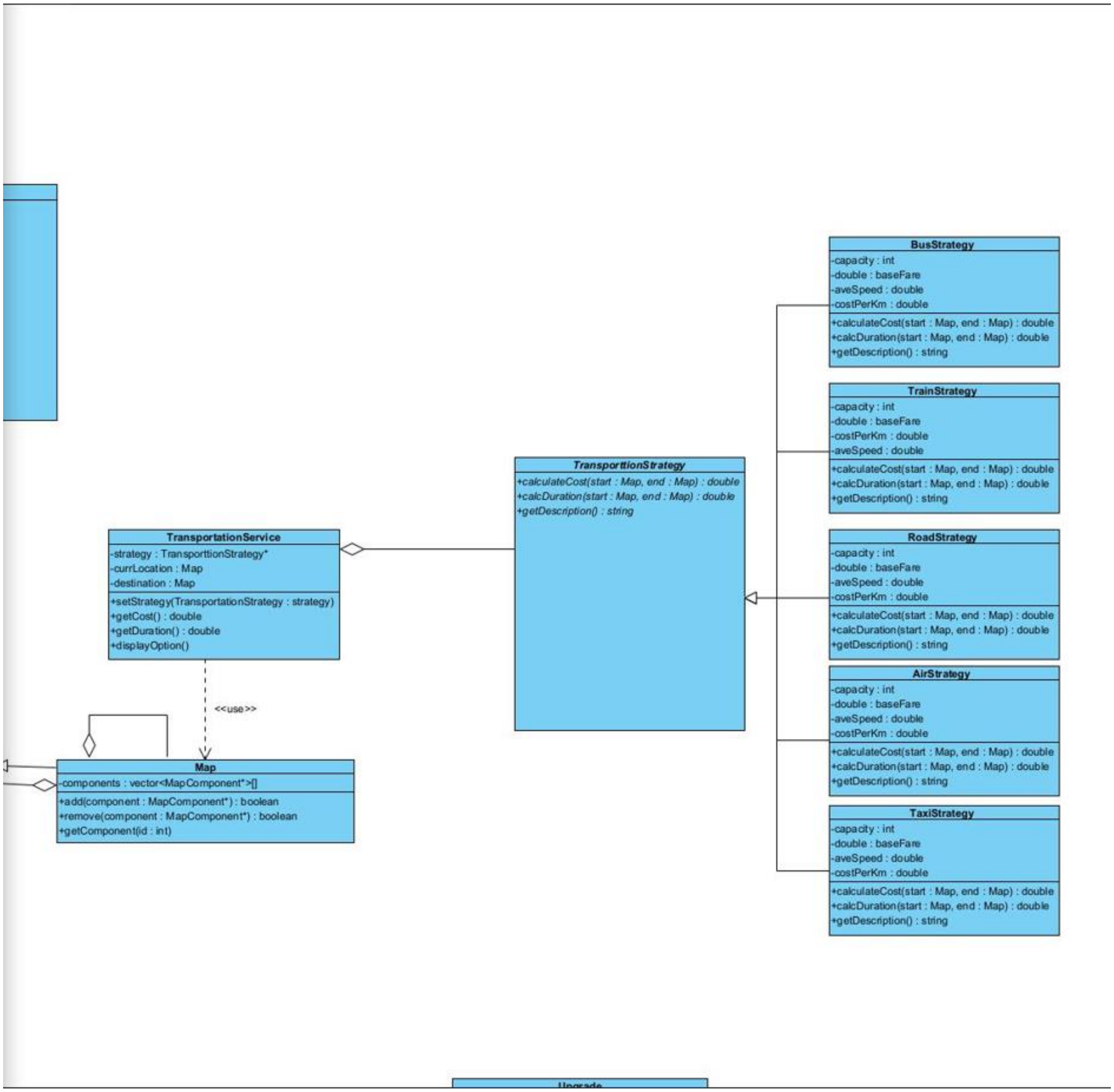



Figure 3: The structure of the Strategy Design Pattern

3




**Reason for use:**

Applied in Transportation; allows interchangeable cost calculation methods for different transportation modes. By encapsulating each mode's cost and duration logic in separate strategies, the pattern enables dynamic selection based on distance and capacity, enhancing scalability and easy maintenance within the city simulation.

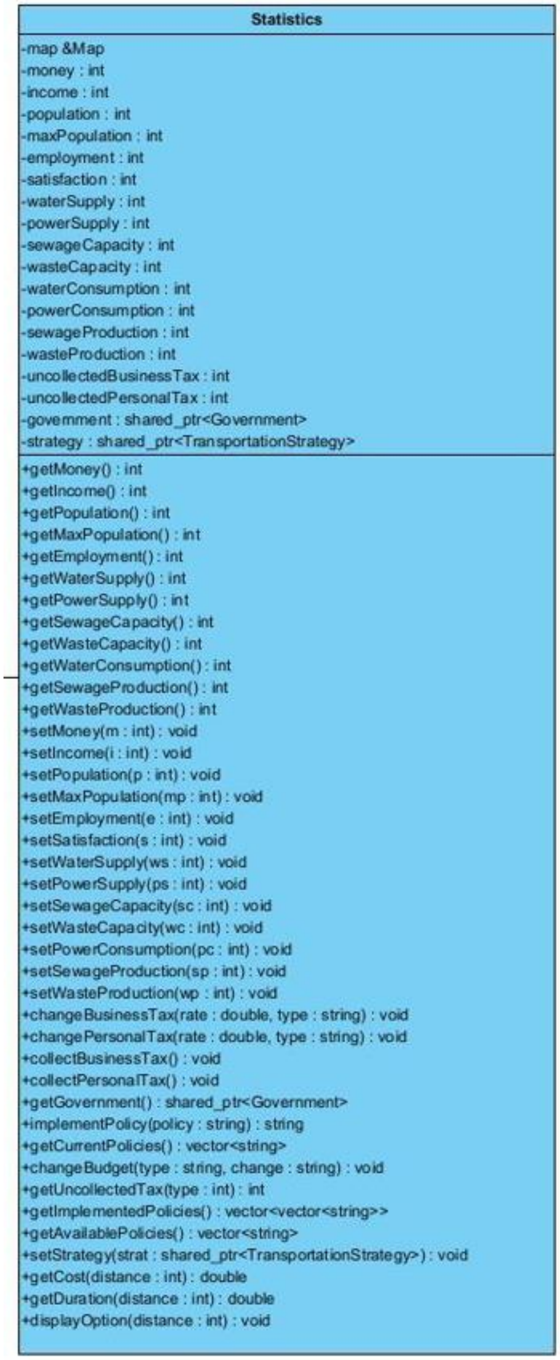
**Participants:**

- **Context:** Statisctics.

- **Strategy:** TransportationStrategy.
- **ConcreteStrategies:** RoadStrategy, AirStrategy, PublicStrategy, TrainStrategy**.**

**BONUS**

Statistics class that displays in-game statistics


Statistics
-map &Map
-money : int
-income : int
-population : int
-maxPopulation : int
-employment : int
-satisfaction : int
-waterSupply : int
-powerSupply : int
-sewageCapacity : int
-wasteCapacity : int
-waterConsumption : int
-powerConsumption : int
-sewageProduction : int
-wasteProduction : int
-uncollectedBusinessTax : int
-uncollectedPersonalTax : int
-government : shared_ptr<Government>
-strategy : shared_ptr<TransportationStrategy>
+getMoney() : int
+getIncome() : int
+getPopulation() : int
+getMaxPopulation() : int
+getEmployment() : int
+getWaterSupply() : int
+getPowerSupply() : int
+getSewageCapacity() : int
+getWasteCapacity() : int
+getWaterConsumption() : int
+getSewageProduction() : int
+getWasteProduction() : int
+setMoney(m : int) : void
+setIncome(i : int) : void
+setPopulation(p : int) : void
+setMaxPopulation(mp : int) : void
+setEmployment(e : int) : void
+setSatisfaction(s : int) : void
+setWaterSupply(ws : int) : void
+setPowerSupply(ps : int) : void
+setSewageCapacity(sc : int) : void
+setWasteCapacity(wc : int) : void
+setPowerConsumption(pc : int) : void
+setSewageProduction(sp : int) : void
+setWasteProduction(wp : int) : void
+changeBusinessTax(rate : double, type : string) : void
+changePersonalTax(rate : double, type : string) : void
+collectBusinessTax() : void
+collectPersonalTax() : void
+getGovernment() : shared_ptr<Government>
+implementPolicy(policy : string) : string
+getCurrentPolicies() : vector<string>
+changeBudget(type : string, change : string) : void
+getUncollectedTax(type : int) : int
+getImplementedPolicies() : vector<vector<string>>
+getAvailablePolicies() : vector<string>
+setStrategy(strat : shared_ptr<TransportationStrategy>) : void
+getCost(distance : int) : double
+getDuration(distance : int) : double
+displayOption(distance : int) : void

# Bibliography

Forum, W. E., 2021. *7 principles of good governance for the post-pandemic city.* [Online]
Available at: https://www.weforum.org/stories/2021/01/7-principles-of-good-city-governance-real-estate/#:~:text=The%20seven%20principles%201%201.%20Political%20consensus%20and,agility%20...%207%207.%20Brand%20and%20story%20
[Accessed 4 November 2024].

Nusca, A., 2011. *Four principles of effective city management.* [Online]
Available at: https://www.zdnet.com/article/four-principles-of-effective-city-management/
[Accessed 4 November 2024].

Riya Patel, H. B., 2024. *6.7 Infrastructure in Urban Development.* [Online]
Available at: https://library.fiveable.me/ap-hug/unit-6/infrastructure-urban-development/study-guide/nez2tBKJMblwbts0qHJm
[Accessed 4 November 2024].

Rodrigue, J. P., 2024. *The Geography of transport systems.* [Online]
Available at: https://transportgeography.org/contents/geography-city-logistics/what-is-city-logistics/city-functions-urban-distribution/
[Accessed 4 November 2024].