UML Class Diagram Notes

Buildings Component - Composite

Component - city

Composite - compositeBuildings

leafs - Residential,Commercial,Industrial, Landmarks (has derived classes)

Utilities Component - Decorator

Component - city

ConcreteComponent - Building

Decorator - Utilities

ConcreteDecorator - PowerPlants, WaterSupply, WasteManagement, SewageSystems

Transportation Component

Strategy

context - Citizen

Strategy -TransportStrategy

ConcreteStrategy – ModeOfTransport, derived classes - PublicTransport, PrivateTransport (e.g. Car, Bus, Bike, Train)

State

context - ModeofTransport

State -TransportationState

ConcreteState - PublicTransportState , PrivateTransportState (both have derived classes e.g. Available, Unavailable)

Citizens Component

Observer

subject - Government

concreteSubject - ...

Observer - Citizen

Chain of responsibility

client - citizen

handler - PublicServicesDepartment

concreteHandler - education, lawEnforcement, healthCare

Resources Component - Flyweight

Flyweight - resources

ConcreteFlyweight - Materials, Energy, Water, Budget

FlyweightFactory -resourcesFactory

Client - city

City Growth Component - Builder

Director - director

Builder - builder

ConcreteBuilder - cityBuilding

Product - compositeBuilder

client –

Other patterns used

Adapter

Adaptee - compositeBuilding

Adapter - adapter

Target - cityTarget

Government Component (Command Pattern)

Participants

- Government (InterfaceClass, Invoker), has-a (aggregation) City
- TaxationDepartment, Budget Department, PoliciesDepartment, PublicServicesDepartment (ConcreteClass)
- Command (Command)
- TaxCommand, AllocateBudgetCommand, PolicyCommand, PublicServicesCommand (ConcreteCommand)
- Citizen, City (Receiver)

Taxes (Strategy Pattern)

Participants

- TaxCommand (Context)
- TaxStrategy (Strategy)
- CitizenTax, CityTax (Concrete Strategy)

Class Members

-Every function should be assumed to have getter, even if it's not listed under its Class's "Functions"

City

Variables

- string cityName – Name of the city.
- int population – Total population of the city.
- float cityBudget – The city's current financial resources.
- Building* buildings – List of all buildings in the city.
- Utilities* utilities – City's utility system (Power, Water, etc.).
- Government* government – Government entity managing the city.
- Citizen* citizens – List of all citizens in the city.
- ResourcesFlyweight* resources – City's resources (materials, energy, etc.).

Functions

- void addBuilding(Building* building) – Adds a new building to the city.
- void allocateResources(ResourceFlyweight* resource) – Allocates resources for city development.
- void updateCitizenSatisfaction() – Updates the satisfaction level of citizens based on city conditions.
- void simulateGrowth() – Simulates the city's growth over time, affecting population and infrastructure.
- void displayCityInformation() – Displays an overview of the city's current state (population, budget, infrastructure, etc.).

Building

Variables

- bool hasElectricity: Indicates electricity availability.
- bool hasWaterSupply: Indicates whether water is supplied to the building.
- bool wasteCollected: Indicates whether waste is being properly collected.
- bool sewageManaged: Indicates whether the sewage system is functioning properly.
- float value: The value of the building,fgbnv which is used to calculate property tax amount.

Residential

Variables

- string type – Type of residential building (Houses, Flats, Townhouses, Estates).
- int population – Number of residents in this building.
- int availableUnits – Number of housing units available for new residents.

Functions

- float calculateOccupancyRate() – Calculates how full the building is based on current population and available units. This can be used to decide whether to expand the

Residential objects of the city (e.g. if available units is 0, expand Residential by creating/cloning new Residentials objects.

Commercial

Variables

- string type – Type of commercial building (Shops, Offices, Malls).
- int entertainmentRating (affects Citizen, the more a Citizen interacts with this class, the higher this value will be (0-10), the higher this value, the higher the Citizen's entertainment variable).
- int employementRating (affects Citizen, from 0 (no employed citizens) to 10(fully employed, no vacancies)).
- float revenue – The revenue generated by this commercial building.

Functions

- void updateEmploymentRating(int affect) – Adjusts the employment rating based on whether a citizen quits(affect= -1) or gets a new job(affect=1), this function can therefore be called by a Citizen object
- void updateEntertainmentRating(int affect) – Adjusts the entertainment rating based on citizen interaction. If entertained by interaction with object, (affect= 1) else (affect = -1)

Industrial

Variables

- string type – Type of industrial building (Factories, Warehouses, Plants).
- int employmentRating – Employment rating from 0-10, affects citizen employment levels.
- float pollutionLevel – Amount of pollution generated by the industrial activity.

Functions

- void updateEmploymentRating(int employedCitizens) – Adjusts the employment rating based on the number of citizens employed.
- void updatePollutionLevel(float pollution) – Modifies the pollution level based on production activity.

Landmarks

Variables

- string type – Type of landmark (Parks, Monuments, Cultural Centers).
- int entertainmentRating – Entertainment rating from 0-10, affecting citizen satisfaction.
- float maintenanceCost – Cost of maintaining the landmark.

Functions

- void updateEntertainmentRating(int affect) – Updates the entertainment rating based on Citizen interactions

Utilities(Decorator)

Variables

- Building* building – The building being decorated with utilities.
- float utilityCost – The cost associated with providing this utility service to the building.
- Resource* resource: This will point to the shared resource that the utility uses (e.g., Energy, Water), acquired from the ResourceFactory.
- ResourceFactory* resourceFactory: This will be used to retrieve the required resources for each utility (e.g., energy for PowerPlants, water for WaterSupply).

Functions

- virtual void applyUtility(Building* building) – Apply the utility service to the building (will be overridden by concrete decorators) by retrieving the necessary resource (e.g., energy, water) from the ResourceFactory and applying it to the building.
- float getUtilityCost() – Returns the cost of this utility.

PowerPlants (Concrete Decorator)

Variables

- int powerGenerated – The amount of electricity generated in kilowatts.
- int powerConsumed – The amount of power consumed by the building.

Functions

- void applyUtility(Building* building) – Supplies power to the building and ensures it meets power needs.
- bool checkPowerAvailability() – Checks if the generated power meets the consumption demands of the building.
- void updatePowerGeneration(int additionalPower) – Adjusts the power generated based on city growth.

WaterSupply (Concrete Decorator)

Variables

- int waterCapacity – Total water supply capacity for the city.
- int waterConsumed – Amount of water consumed by the building.

Functions

- void applyUtility(Building* building) – Ensures water supply is delivered to the building.
- bool checkWaterSupply() – Verifies if the water capacity is enough for the current city needs.
- void updateWaterConsumption(int newUsage) – Updates water consumption based on changes in population or building type.

WasteManagement (Concrete Decorator)

Variables

- int wasteCapacity – Maximum waste that can be processed by the city's waste management system.
- int wasteGenerated – Amount of waste generated by the building.

Functions

- void applyUtility(Building* building) – Handles waste collection and recycling for the building.
- void increaseRecyclingRate(float rate) – Adjusts the recycling rate to reduce waste impact.
- bool checkWasteCapacity() – Ensures that waste capacity isn't exceeded by building activities.

SewageSystems (Concrete Decorator)

Variables

- int sewageCapacity – Maximum sewage that can be processed.
- int sewageGenerated – Amount of sewage currently produced by the building.

Functions

- void applyUtility(Building* building) – Ensures that sewage from the building is handled properly.
- bool checkSewageCapacity() – Checks if the sewage system can handle the waste load.
- void updateSewageCapacity(int newCapacity) – Increases the sewage system's capacity based on population or building expansions.

Transportation Component

Citizen (Context for Strategy)

Variables

- TransportStrategy* preferredModes[3] – An array to store the citizen's three preferred modes of transport (public or private).
- TransportationState* currentState – Represents the current state of the citizen's ability to travel.
- string name – Name of the citizen.
- float satisfaction – Citizen's satisfaction, affected by commute efficiency.
- float commuteTime – The time it takes for the citizen to travel.

Functions

- void setTransportStrategy(TransportStrategy* strategy) – Allows setting a new transport strategy.
- bool canTravel() – Checks if at least one preferred mode of transport is available.
- void travel() – Initiates travel using the first available preferred mode.

TransportStrategy (Strategy Interface)

Functions

- float calculateCommuteTime() – Calculates the time required for travel using this transport mode.
- bool isAvailable() – Returns the availability status of this transport mode.

ModeOfTransport (Concrete Strategy)

Functions

- void displayModeDetails() – Displays specific details about the transport mode.

PublicTransport (Derived from ModeOfTransport)

Variables

- int capacity – Maximum number of passengers it can accommodate.
- int routesAvailable – Number of routes operated by this public transport mode.

Functions

- void stopAtStation() – Simulates stopping at a station for passengers to board and alight.

Bus (Concrete Class)

Variables

- float fare – Fare for using this bus service.

Train (Concrete Class)

Variables

- float ticketPrice – Price of a ticket for this train service.

Taxi (Concrete Class)

Variables

- float taxiFare – Taxi fare for a taxi ride.

Plane (Concrete Class)

Variables

- float ticketCost – Cost of a plane ticket.

PrivateTransport (Derived from ModeOfTransport)

Variables

- float maintanceCost – Cost to keep the vehicle (e.g. fuel, service etc.)

Car (Concrete Class)

Bike (Concrete Class)

TransportationState (State Interface)

Functions

- void changeState() – Changes the state of the mode of transport.

PublicTransportState (Concrete State)

Variables

- bool isAvailable – Availability status of the public transport mode.

Available (Concrete State)

Functions

- void changeState() – Changes the state to unavailable when capacity is reached.

Unavailable (Concrete State)

Functions

- void changeState() – Changes the state to available when capacity is freed.

PrivateTransportState (Concrete State)

Variables

- bool isAvailable – Availability status of the private transport mode.

Available (Concrete State)

Functions

- void changeState() – Changes the state to unavailable when the vehicle is in use.

Unavailable (Concrete State)

Functions

- void changeState() – Changes the state to available when the vehicle is returned

Citizens, Government, Tax Component

Government

Variables

- Citizen* observers: List of citizens observing the government for updates (policies, taxes, services).
- float taxRate: The current tax rate that citizens are subjected to.
- string currentPolicy: The latest policy implemented by the government.
- vector<string> availableServices: List of public services available (healthcare, education, law enforcement, etc.).
- Command* commandQueue: Queue of commands waiting to be executed (e.g., budget allocation, policy changes).
- City* city: Aggregation of the city that the government manages.

Functions

- void addObserver(Citizen* observer): Adds a citizen to the list of observers.
- void removeObserver(Citizen* observer): Removes a citizen from the list of observers.
- void notifyObservers(): Notifies all citizens of changes (taxes, policies, etc.).
- void updateTaxes(float newTaxRate): Updates the tax rate and notifies citizens.
- void implementPolicy(string& policy): Implements a new policy and notifies citizens.
- void updatePublicServices(vector<string>& services): Updates the available public services and notifies citizens.
- void addCommand(Command* command): Adds a command to the queue.
- void executeCommands(): Executes all commands in the queue.

TaxationDepartment

-Handles tax-related changes and notifications.

BudgetDepartment

-Manages city budget allocations and informs citizens of changes.

PoliciesDepartment

-Implements city-wide policies and communicates them to citizens.

PublicServicesDepartment

-Manages services like healthcare, education, and law enforcement, and keeps citizens informed.

Variables

- PublicServiceHandler* nextHandler: The next handler in the chain.

Functions

- virtual void handleIssue(const std::string& issue): Handles the issue raised by the citizen. Passes it to the next handler if not resolved.

Citizen

Variables

- string employmentStatus: Whether the citizen is employed or unemployed.
- string currentEducation: The current education level of the citizen (affects satisfaction).
- string healthcareAccess: Whether the citizen has access to healthcare.
- float satisfaction: Overall satisfaction level, influenced by taxes, policies, and services.
- float taxRate: The citizen's awareness of the current tax rate.
- string currentPolicy: The policy the citizen is currently subjected to.
- string issueType: The type of issue (health, education, security) that the citizen faces.
- PublicServiceHandler* publicServiceHandler: Reference to the chain of responsibility for resolving citizen issues.

Functions

- void update(float taxRate, string policy, vector<string> services): Updates the citizen's state based on changes in tax, policies, and services.
- void respondToGovernmentChanges(): Alters satisfaction and life conditions based on the government's decisions (e.g., tax hikes reduce satisfaction).
- void raiseIssue(string& issue): Sends an issue to the public services chain for resolution.

Command (Interface)

Functions:

- virtual void execute() = 0: Abstract method for executing a command.

TaxCommand

Variables

- float newTaxRate: The new tax rate to be set.

Functions

- void execute(): Changes the city's tax rate.

AllocateBudgetCommand:

Variables

- float budgetAmount: The amount of budget to be allocated.

Functions

- void execute(): Allocates the budget to various city departments.

PolicyCommand:

Variables

- string newPolicy: The policy to be implemented.

Functions

- void execute(): Implements the policy in the city.

PublicServicesCommand:

Variables

- string serviceName: The public service to be improved or expanded.

Functions:

- void execute(): Expands or improves a specific public service (e.g., healthcare, education).

Taxation Component (Strategy Pattern)

TaxCommand

-The context that uses the tax collection strategy to execute different tax commands for citizens and buildings.

Variables

- TaxStrategy* taxStrategy: The strategy for collecting taxes.
- Citizen* citizen: The citizen on whom the tax command will be applied (for citizen-specific taxes like income tax).
- Building* building: The building on which the tax command will be applied (for property taxes, etc.).

Functions

- void setTaxStrategy(TaxStrategy* strategy): Sets the strategy for tax collection (CitizenTax or BuildingTax).
- void executeTaxCommand(): Executes the tax collection strategy based on the provided context (citizen or building).

TaxStrategy

-An abstract interface for tax collection strategies. Defines how tax collection should be applied.

Functions

- virtual void collectTax(Citizen* citizen) = 0: Abstract method to be overridden for citizen-specific taxes (e.g., income tax).
- virtual void collectTax(Building* building) = 0: Abstract method to be overridden for building-specific taxes (e.g., property tax).

CitizenTax (Concrete Strategy for taxing citizens)

-Responsible for collecting taxes from citizens, such as income tax or sales tax.

Variables

- float incomeTaxRate: Rate at which income tax is collected.

Functions

- void collectTax(Citizen* citizen) override: Applies the tax collection to the citizen based on their employment status and income. Also adjusts citizen satisfaction based on the tax burden.

BuildingTax (Concrete Strategy for taxing buildings)

-Responsible for collecting property taxes and other building-related taxes.

Variables

- float propertyTaxRate: The rate at which property taxes are collected.

Functions

- void collectTax(Building* building) override: Applies tax collection to the building (e.g., property tax). Adjusts the financial resources of the building's owner accordingly.

Citizen (Receiver of TaxCommand)

Variables

- string employmentStatus: Determines if the citizen is employed or unemployed (affects income tax collection).
- string currentEducation: The current education level of the citizen (affects overall satisfaction and can modify tax behavior based on education policy).
- string healthcareAccess: Whether the citizen has access to healthcare (affects satisfaction).
- float satisfaction: Overall satisfaction level, influenced by taxes, policies, and services.
- float taxRate: The citizen's awareness of the current tax rate, which impacts their satisfaction.
- string currentPolicy: The policy the citizen is currently subjected to.
- string issueType: The type of issue (health, education, security) that the citizen faces.
- PublicServiceHandler* publicServiceHandler: Reference to the chain of responsibility for resolving citizen issues.

Functions

- void payIncomeTax(float amount): Calculates and deducts income tax from the citizen based on employment status and salary.
- void adjustSatisfaction(float taxImpact): Reduces or increases satisfaction based on the tax burden.

Building (Receiver of TaxCommand)

Variables

- float maintenanceCost: The cost to maintain the building (affects additional tax calculations).
- float taxRate: The property tax rate that the building is subject to.

Functions

- void payPropertyTax(float amount): Calculates and deducts property tax based on the building's value.

- void updateBuildingTaxRate(float newRate): Updates the building's tax rate based on government policy.

How the components work

2. Utilities

In the City Builder Simulation project, the **Utilities Decorators** (e.g., **PowerPlant**, **WaterSupply**, **WasteManagement**, **SewageSystem**) would primarily affect the state or behaviours of the **ConcreteComponents** (i.e., **Residential**, **Commercial**, **Industrial**, **Landmarks**) by interacting with specific variables related to citizens' needs, building operations, and city management. Below is a breakdown of how each of the **Decorators** could affect key variables in the **ConcreteComponents**.

PowerPlant Decorator

Effect on ConcreteComponents:

- Provides electricity to buildings. The presence of electricity improves functionality and citizen satisfaction.

Affected Variables:

- **Residential**:
    - int population: Electricity makes living conditions better, possibly attracting more citizens.
    - **New Variable**: bool hasElectricity: Indicates whether the building has electricity supplied by a PowerPlant.
- **Commercial**:
    - int employmentRating: Commercial buildings need electricity to operate. A lack of electricity could reduce employment opportunities.
    - **New Variable**: bool hasElectricity: Indicates electricity availability.
- **Industrial**:
    - int employmentRating: Industrial production requires electricity. Without it, the employment rating could drop.
    - float pollutionLevel: The power plant itself may increase the overall pollution level, affecting nearby industrial buildings.
    - **New Variable**: bool hasElectricity: Indicates electricity availability.
- **Landmarks**:
    - int entertainmentRating: Some landmarks, like parks or monuments, might rely on lighting or other power-dependent systems to enhance the citizen experience.
    - **New Variable**: bool hasElectricity: Indicates whether the landmark is powered.

WaterSupply Decorator

Effect on ConcreteComponents:

- Provides a steady water supply to buildings, which affects functionality, population growth, and citizen health.

Affected Variables:

- **Residential**:
    - int population: Proper water supply could increase the building's attractiveness to residents.
    - **New Variable**: bool hasWaterSupply: Indicates whether water is supplied to the building.
- **Commercial**:
    - int employmentRating: Businesses like restaurants or malls depend on water. A lack of water could impact operations and employment.
    - **New Variable**: bool hasWaterSupply: Indicates whether water is available.
- **Industrial**:
    - int employmentRating: Many industrial processes need water. A disruption in supply could reduce employment levels.
    - **New Variable**: bool hasWaterSupply: Indicates water availability.
- **Landmarks**:
    - int entertainmentRating: For landmarks with fountains, pools, or other water features, the water supply affects the entertainment rating.
    - **New Variable**: bool hasWaterSupply: Indicates if water is supplied.

WasteManagement Decorator

Effect on ConcreteComponents:

- Manages waste and recycling, improving the overall hygiene and reducing pollution in and around buildings.

Affected Variables:

- **Residential**:
    - int population: Efficient waste management could increase the attractiveness of residential areas, attracting more people.
    - **New Variable**: bool wasteCollected: Indicates whether waste is being properly collected.
- **Commercial**:
    - int employmentRating: Businesses, especially those generating significant waste, rely on waste management services. Poor waste handling may negatively affect employment.

    - **New Variable**: bool wasteCollected: Indicates whether waste is managed.
- **Industrial**:
    - float pollutionLevel: Waste management directly impacts the pollution generated by industrial buildings. Efficient waste disposal reduces pollution levels.
    - **New Variable**: bool wasteCollected: Indicates whether industrial waste is being managed.
- **Landmarks**:
    - int entertainmentRating: Parks and cultural centers benefit from proper waste management to maintain cleanliness and citizen satisfaction.
    - **New Variable**: bool wasteCollected: Indicates whether waste is properly disposed of in the landmark area.

SewageSystem Decorator

Effect on ConcreteComponents:

- Manages sewage disposal, impacting citizens' health, hygiene, and overall building performance.

Affected Variables:

- **Residential**:
    - int population: Proper sewage systems can improve living conditions, potentially increasing population.
    - **New Variable**: bool sewageManaged: Indicates whether the sewage system is functioning properly.
- **Commercial**:
    - int employmentRating: Businesses, especially those with high water usage, require sewage systems to function properly. A lack of sewage systems may affect operations and employment.
    - **New Variable**: bool sewageManaged: Indicates sewage management status.
- **Industrial**:
    - int employmentRating: Some industrial activities rely on sewage systems for waste disposal. Lack of proper sewage management may reduce employment.
    - float pollutionLevel: Effective sewage treatment can reduce environmental pollution caused by industrial waste.
    - **New Variable**: bool sewageManaged: Indicates if sewage from the industrial building is properly managed.
- **Landmarks**:

- int entertainmentRating: Poor sewage management around landmarks (e.g., parks) can negatively affect their attractiveness and the entertainment rating.
- **New Variable**: bool sewageManaged: Indicates whether the sewage system is functioning around the landmark.

Summary

- **PowerPlant** affects: hasElectricity (for all building types).
- **WaterSupply** affects: hasWaterSupply (for all building types).
- **WasteManagement** affects: wasteCollected (for all building types).
- **SewageSystem** affects: sewageManaged (for all building types).

3. Transportation

In the context of the **Transportation Component** where the **Strategy** and **State** patterns are implemented, here's how these implementations would affect the variables in their respective **Contexts**

Citizen Context

Variables

- TransportStrategy* preferredModes[3]:
    - **Affected by Strategy Implementation:** This array will be populated with different modes of transport (e.g., Bus, Car, etc.). The **TransportStrategy** will determine how the citizen interacts with their preferred modes of transport.
    - **Impact:** If all preferred modes are unavailable, the canTravel() function will return false, indicating the citizen cannot travel.
- TransportationState* currentState:
    - **Affected by State Implementation:** This variable will hold the current state of transportation availability for the citizen (e.g., Available, Unavailable).
    - **Impact:** The citizen's ability to travel will depend on the state of the preferred transport modes, as indicated by this variable.

ModeOfTransport Context

Variables

- **TransportationState** (not a variable but the type of the current transport mode):
    - **Affected by State Implementation:** The state of a transport mode (e.g., Available, Unavailable) will dictate whether the transport mode can be utilized at any given time.
    - **Impact:** The operational status of the transport mode will directly influence whether it can be used for travel and how citizens interact with it.

In the **State Pattern** implementation:

TransportationState

Variables

- bool isAvailable:
    - **Affected by State Implementation:** Represents whether the transport mode is currently available for use.
    - **Impact:** This will change based on conditions (e.g., when a vehicle is in use or at full capacity) and directly affects the citizen's ability to use this transport mode.

PublicTransportState and PrivateTransportState

Variables

- These states will maintain their own isAvailable status, which will change based on actions taken by the citizen or the transport mode itself.
    - **Impact:** Each concrete state will define behaviors for changing availability (e.g., when a bus leaves a station, its state may change from Available to Unavailable), affecting the overall transport availability for citizens.

Summary

The **Strategy** implementation primarily affects the preferred modes of transport for citizens, dictating how they can plan their travel. The **State** implementation influences the availability of those transport modes, affecting whether citizens can actually utilize their preferred choices. Together, these patterns enhance the simulation by allowing dynamic interactions between citizens and their transportation options.

6. Resources

Utilities --> Resource (Flyweight) Association

This association represents how each concrete utility (e.g., PowerPlants, WaterSupply) directly interacts with a shared **Resource** object. Since the **Resource** is implemented using the **Flyweight** design pattern, it means that all instances of utility classes share these resource objects to ensure memory efficiency and avoid duplication.

How It Works

- **Purpose**: The concrete utility classes, such as PowerPlants and WaterSupply, need access to shared resources like **Energy** or **Water** to operate.
- **Interaction**:
    - When a building requests power, the PowerPlants utility decorator will interact with the **Energy** flyweight to check how much energy is available.
    - Similarly, when a building requires water, the WaterSupply decorator will interact with the **Water** flyweight to verify the water capacity and ensure enough supply is available for the building.
- **Example for PowerPlants**:
    - **PowerPlants** decorator has variables powerGenerated and powerConsumed.
    - It needs to ensure that the power consumed by the building doesn't exceed the total power available.
    - The PowerPlants decorator will interact with the **Energy** flyweight (shared by all buildings using power) to:
        - **Check availability**: It calls methods from the **Energy** resource to check if enough power can be allocated.
        - **Consume energy**: It adjusts the available energy for the building by interacting with the Energy flyweight to decrement the available power.
- **Efficiency**: Since **Energy** is a flyweight, this shared resource means that rather than creating individual energy instances for each building, the PowerPlants decorator references a central **Energy** object, updating or checking the availability in one place, shared across all instances of the utility system.

UML Representation

- **Utilities** (abstract class) --> **Resource** (Flyweight) is an **Association**, indicating that every Utilities object can access shared resources.
- The **Concrete Utility Classes** (PowerPlants, WaterSupply, WasteManagement, etc.) will each have an association to their specific resource type (e.g., **Energy**, **Water**), leveraging the flyweight for efficient resource management.

Utilities --> ResourceFactory (FlyweightFactory) Association

This association represents how the concrete utility decorators (like PowerPlants, WaterSupply, etc.) interact with the **ResourceFactory** to retrieve the specific resource flyweights (i.e., Energy,

Water, Materials, Budget). The **ResourceFactory** is responsible for managing and creating instances of the resource flyweights, ensuring that the same shared resource is used wherever necessary.

How It Works:

- **Purpose**: The **ResourceFactory** acts as a **FlyweightFactory** to provide instances of shared resources to the utility decorators. Rather than each utility creating its own instance of Energy, Water, etc., they request the required resources from the factory.
- **Interaction**:
    - When a utility needs access to a particular resource, it doesn't create the resource directly. Instead, it queries the **ResourceFactory**.
    - The factory checks if the resource has already been created:
        - If the resource already exists (e.g., **Energy** flyweight), it returns the existing instance.
        - If the resource doesn’t exist (e.g., a new type of material is needed), the factory creates a new flyweight instance and stores it for future use.
- **Example for PowerPlants**:
    - When the PowerPlants utility decorator is applied to a building, it needs to access the available **Energy** resource.
    - Instead of creating an Energy object, PowerPlants will ask the **ResourceFactory** to provide it with the current **Energy** flyweight.
    - The **ResourceFactory** checks if the **Energy** flyweight has already been created:
        - If yes, it returns the existing flyweight.
        - If no, it creates a new **Energy** flyweight, stores it, and returns it to PowerPlants.
- **Efficiency**: This system reduces memory usage by ensuring that there is only one instance of each resource type, shared across all relevant components. The factory centralizes control over resource creation, ensuring no duplication occurs.

UML Representation:

- **Utilities** (abstract class) --> **ResourceFactory** is also an **Association**, representing that every utility component interacts with the factory to retrieve the appropriate shared resources (flyweights).
- The **Concrete Utility Classes** will call the **ResourceFactory** to obtain the relevant resources, without needing to know how these resources are created or managed.

Summary

Utilities --> Resource (Flyweight) Association

- **Role**: Allows concrete utility decorators to interact with shared resource objects like **Energy** or **Water**.
- **Interaction**: Utility decorators check availability, allocate, or consume resources through the shared flyweights.
- **Efficiency**: Ensures that a single instance of a resource (e.g., energy) is used across multiple buildings.

Utilities --> ResourceFactory (FlyweightFactory) Association

- **Role**: Utilities request resources from the factory rather than creating them directly.
- **Interaction**: The factory manages the lifecycle of the flyweights, returning either new or existing resource objects as needed by the utilities.
- **Efficiency**: Centralizes resource management, ensuring that only necessary flyweights are created and reused across the simulation.

These associations in the design ensure efficient resource management and allow the city to grow dynamically without redundant resource allocation, aligning with the flyweight pattern's goal of minimizing memory usage.

8. TaxCommand

Setting the Strategy

The government, through its taxation department, uses TaxCommand as the invoker. It can dynamically switch between strategies (either CitizenTax or BuildingTax) based on whether it's taxing a citizen or a building.

Executing the Tax Command:

- If the strategy is **CitizenTax**, the collectTax function will calculate the income tax or sales tax based on the citizen's employmentStatus and other relevant factors, adjusting their satisfaction accordingly.
- If the strategy is **BuildingTax**, the collectTax function will calculate property taxes and maintenance taxes based on the building's buildingValue and other attributes, adjusting the financial state of the building or its owner.

Impact on Satisfaction and City Economy

Taxes collected from citizens and buildings are used to fund city projects and services. Citizens' satisfaction will be influenced by their tax burden, while buildings' financial resources will adjust based on their tax rates.

Example Scenario

Income Tax Scenario (Citizen):

- The TaxCommand invokes the **CitizenTax** strategy.
- The collectTax(Citizen* citizen) method calculates income tax based on the citizen's employment status and income, deducting the amount from their resources, then adding that amount to the City's cityBudget.
- The citizen's satisfaction decreases slightly due to the tax burden.

Property Tax Scenario (Building):

- The TaxCommand invokes the **BuildingTax** strategy.
- The collectTax(Building* building) method calculates property tax based on the building's value and tax rate, adding that amount to the City's cityBudget.

This way, the tax system dynamically adjusts to collect revenue from both citizens and buildings, funding city services while impacting overall satisfaction and economic growth.