COS 214 Project Team 12

Generated by Doxygen 1.9.1

Chapter 1

README

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 MapGrid::Cell Struct Reference

Collaboration diagram for MapGrid::Cell:

Public Attributes

- std::shared_ptr< CityComponent > component
- std::vector< std::shared_ptr< UtilityFlyweight > > affectingUtilities
- std::shared_ptr< ZoneComposite > zone
- char symbol = ' '

5.1.1 Member Data Documentation

5.1.1.1 affectingUtilities

```
std::vector<std::shared_ptr<UtilityFlyweight> > MapGrid::Cell::affectingUtilities
```

5.1.1.2 component

```
std::shared_ptr<CityComponent> MapGrid::Cell::component
```

5.1.1.3 symbol

```
char MapGrid::Cell::symbol = ' '
```

5.1.1.4 zone

```
std::shared_ptr<ZoneComposite> MapGrid::Cell::zone
```

The documentation for this struct was generated from the following file:

- COS214-Poject/src/MapGrid.h

5.2 ChemicalFactory Class Reference

A factory class for creating chemical resources used in construction and income generation.

```
#include <ChemicalFactory.h>
```

Inheritance diagram for ChemicalFactory:

Collaboration diagram for ChemicalFactory:

Public Member Functions

- ChemicalFactory ()
  Constructs a new ChemicalFactory object.
- ~ChemicalFactory ()
  Destroys the ChemicalFactory object.
- std::unique_ptr< IncomeResourceProduct > createIncomeR (int quantity) override
  Creates an income-generating resource.
- std::unique_ptr< ConstructionResourceProduct > createConstructionR (int quantity) override
  Creates a construction resource.

5.2.1 Detailed Description

A factory class for creating chemical resources used in construction and income generation.

The `ChemicalFactory` class specializes in producing income-generating and construction resources. It inherits from `ResourceFactory` and implements the factory methods to create specific types of resource products.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 ChemicalFactory()

```
ChemicalFactory::ChemicalFactory ( )
```

Constructs a new ChemicalFactory object.

Initializes any required state for the chemical resource factory.

Outputs a message indicating the creation of a ChemicalFactory instance.

5.2.2.2 ∼ChemicalFactory()

```
ChemicalFactory::∼ChemicalFactory ( )
```

Destroys the ChemicalFactory object.

Cleans up resources allocated by the factory.

Outputs a message indicating the deletion of a ChemicalFactory instance.

5.2.3 Member Function Documentation

5.2.3.1 createConstructionR()

```
std::unique_ptr< ConstructionResourceProduct > ChemicalFactory::createConstructionR (
             int quantity )  [override], [virtual]
```

Creates a construction resource.

Parameters

quantity	The amount of construction resource to create.

Returns

A unique pointer to a `ConstructionResourceProduct` instance.

This method creates a construction resource of type `Concrete` with a specified quantity.

Parameters

quantity	The quantity of the construction resource to create.

Returns

A unique pointer to a `ConstructionResourceProduct` instance representing the construction resource.

Implements ResourceFactory.

5.2.3.2 createIncomeR()

```
std::unique_ptr< IncomeResourceProduct > ChemicalFactory::createIncomeR (
             int quantity )  [override], [virtual]
```

Creates an income-generating resource.

Parameters

quantity	The amount of income resource to create.

Returns

A unique pointer to an `IncomeResourceProduct` instance.

This method creates an income resource of type `Oil` with a specified quantity.

Parameters

quantity	The quantity of the income resource to create.

Returns

A unique pointer to an `IncomeResourceProduct` instance representing the income resource.

Implements ResourceFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/ChemicalFactory.h
- COS214-Poject/src/ChemicalFactory.cpp

5.3 CityComponent Class Reference

Abstract base class representing a component of a city.

```
#include <CityComponent.h>
```

Inheritance diagram for CityComponent:

Collaboration diagram for CityComponent:

Public Member Functions

- virtual void add (std::shared_ptr< CityComponent > component)

  Adds a subcomponent to the city component.
- virtual void remove (std::shared_ptr< CityComponent > component)

  Removes a subcomponent from the city component.
- virtual void displayStatus ()=0

  Displays the status of the city component.
- virtual ~CityComponent ()

  Destructor for CityComponent.
- void addNpc ()

  Adds an NPC observer to the component.
- void removeNpc ()

Removes an NPC observer from the component.

- void notify ()

  Notifies all observers of changes in the city component.
- virtual std::string getBuildingType () const =0

  Gets the type of building as a string.
- virtual void accept (taxCollector ∗collector)=0

  Accepts a tax collector visitor.
- void setLocation (int x, int y)

  Sets the location of the component using x and y coordinates.
- void setLocation (const Location &loc)

  Sets the location of the component using a Location object.
- Location getLocation () const

  Gets the current location of the component.

Private Attributes

- std::vector< NPCObserver ∗ > observers
- Location location

5.3.1 Detailed Description

Abstract base class representing a component of a city.

This class defines an interface for city components, allowing for adding/removing subcomponents, managing NPC observers, setting locations, and displaying status. Derived classes represent specific city components, such as buildings or utilities.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 ∼CityComponent()

```
CityComponent::∼CityComponent ( )  [virtual]
```

Destructor for CityComponent.

Destructor for the CityComponent class.

Cleans up resources and observers when a CityComponent is destroyed.

Cleans up all dynamically allocated NPC observers to prevent memory leaks.

5.3.3 Member Function Documentation

5.3.3.1 accept()

```
virtual void CityComponent::accept (
                taxCollector * collector )  [pure virtual]
```

Accepts a tax collector visitor.

Part of the Visitor pattern, allowing tax collectors to visit city components and perform operations based on the component's type.

Parameters

collector	A pointer to a taxCollector object.

Implemented in ZoneComposite, ResidentialBuilding, PublicService, CommercialBuilding, LandMark, Industry, IncomeResourceProduct, ConstructionResourceProduct, CityComposite, and UtilityFlyweight.

5.3.3.2 add()

```
virtual void CityComponent::add (
             std::shared_ptr< CityComponent > component )  [inline], [virtual]
```

Adds a subcomponent to the city component.

Allows derived classes to implement adding subcomponents, supporting composite structures.

Parameters

component	The city component to add as a subcomponent.

Reimplemented in CityComposite, and ZoneComposite.

5.3.3.3 addNpc()

```
void CityComponent::addNpc ( )
```

Adds an NPC observer to the component.

Adds NPCs to the city component and adjusts the city's population growth and happiness level.

This method calculates the population growth based on the city's residential and utility infrastructure, creates new NPCs based on this growth, and adjusts the overall happiness level depending on available housing and job opportunities. Here is the call graph for this function:

5.3.3.4 displayStatus()

```
virtual void CityComponent::displayStatus ( )  [pure virtual]
```

Displays the status of the city component.

A pure virtual function that must be implemented by derived classes to show component-specific information.

Implemented in PublicService, Wood, Stone, Steel, School, ResidentialBuilding, PoliceStation, Park, Oil, LandMark, Hospital, Gold, FireStation, Diamonds, Concrete, CommercialBuilding, Coal, CityComposite, ZoneComposite, UtilityFlyweight, Monument, Industry, CulturalCenter, and ConstructionResourceProduct.

5.3.3.5 getBuildingType()

```
virtual std::string CityComponent::getBuildingType ( ) const  [pure virtual]
```

Gets the type of building as a string.

A pure virtual function for retrieving the type of building represented by the component.

Returns

The type of building as a string.

Implemented in UtilityFlyweight, Townhouse, Stone, Shops, ResidentialBuilding, PublicService, Office, Malls, LandMark, Industry, House, Flat, Estate, CommercialBuilding, CityComposite, ZoneComposite, Wood, Steel, Park, Oil, Monument, Hospital, Gold, Diamonds, CulturalCenter, Concrete, and Coal.

5.3.3.6 getLocation()

```
Location CityComponent::getLocation ( ) const
```

Gets the current location of the component.

Gets the current location of the city component.

Returns

The current location as a Location object.

5.3.3.7 notify()

```
void CityComponent::notify ( )
```

Notifies all observers of changes in the city component.

Notifies all NPC observers of changes in the city component.

This function iterates through the list of observers and calls their `update` method to respond to changes. Here is the caller graph for this function:

5.3.3.8 remove()

```
virtual void CityComponent::remove (
            std::shared_ptr< CityComponent > component )  [inline], [virtual]
```

Removes a subcomponent from the city component.

Allows derived classes to implement removal of subcomponents.

Parameters

component	The city component to remove.

Reimplemented in CityComposite, and ZoneComposite.

5.3.3.9 removeNpc()

```
void CityComponent::removeNpc ( )
```

Removes an NPC observer from the component.

Removes NPCs from the city component based on certain conditions.

This method decreases the population based on the city's highest state (Crime or Revolt), simulating citizen departures or deaths. It removes NPC observers up to the calculated decrease amount. Here is the call graph for this function:

5.3.3.10 setLocation() [1/2]

```
void CityComponent::setLocation (
             const Location & loc )
```

Sets the location of the component using a Location object.

Sets the location of the city component using a Location object.

Parameters

loc	A Location object representing the new location.

5.3.3.11 setLocation() [2/2]

```
void CityComponent::setLocation (
             int x,
             int y )
```

Sets the location of the component using x and y coordinates.

Sets the location of the city component using x and y coordinates.

Parameters

x	The x-coordinate of the location.
y	The y-coordinate of the location.

5.3.4 Member Data Documentation

5.3.4.1 location

```
Location CityComponent::location  [private]
```

Location of the city component

5.3.4.2 observers

```
std::vector<NPCObserver*> CityComponent::observers  [private]
```

List of NPC observers observing this component

The documentation for this class was generated from the following files:

- COS214-Poject/src/CityComponent.h
- COS214-Poject/src/CityComponent.cpp

5.4 CityComposite Class Reference

A composite class representing the entire city and containing multiple zones.

```
#include <CityComposite.h>
```

Inheritance diagram for CityComposite:

Collaboration diagram for CityComposite:

Public Member Functions

- CityComposite (const std::string &name)
  Constructs a new CityComposite with the given name.
- ∼CityComposite ()
  Destructor for CityComposite.
- void add (std::shared_ptr< CityComponent > component) override
  Adds a city component (zone) to the composite.
- void remove (std::shared_ptr< CityComponent > component) override
  Removes a city component (zone) from the composite.
- void displayStatus () override
  Displays the status of the city composite and all its zones.
- bool checkCityConditions ()
  Checks if the city meets certain conditions.
- std::string getBuildingType () const override
  Returns the type of building as a string.
- void accept (taxCollector ∗tax) override
  Accepts a tax collector visitor.
- void connectZones (CityComponent ∗zoneA, CityComponent ∗zoneB, std::unique_ptr< Transport > transport)
  Connects two zones with a specified transport method.
- void updateHappinessForNewComponent (const std::shared_ptr< CityComponent > &component)
  Updates the city's happiness level based on the addition of a new component.
- void adjustHappinessBasedOnTaxRate ()
  Adjusts the city's happiness level based on the current tax rate.

Private Attributes

- std::vector< std::shared_ptr< CityComponent > > zones
- std::string cityName

5.4.1 Detailed Description

A composite class representing the entire city and containing multiple zones.

This class allows the addition and removal of zones (other CityComponents) and manages high-level attributes like city name, connections between zones, and the happiness level.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 CityComposite()

```
CityComposite::CityComposite (
              const std::string & name )
```

Constructs a new CityComposite with the given name.

Constructs a CityComposite object with a given name.

Parameters

name	The name of the city.

5.4.2.2 ∼CityComposite()

```
CityComposite::∼CityComposite ( )
```

Destructor for CityComposite.

Cleans up resources allocated for the city composite, including its zones.

Clears all zones managed by the composite.

5.4.3 Member Function Documentation

5.4.3.1 accept()

```
void CityComposite::accept (
            taxCollector * tax )  [inline], [override], [virtual]
```

Accepts a tax collector visitor.

Currently, this function does not perform any actions. The function is part of the Visitor pattern to allow tax collectors to visit city zones and collect taxes.

Parameters

tax	A pointer to a taxCollector object.

Implements CityComponent.

5.4.3.2 add()

```
void CityComposite::add (
            std::shared_ptr< CityComponent > component )  [override], [virtual]
```

Adds a city component (zone) to the composite.

Adds a city component (zone) to the city.

Parameters

component	A shared pointer to the CityComponent to add.

This function also updates the happiness for the new component if the city already has a population.

Parameters

component	A shared pointer to the CityComponent to be added.

Reimplemented from CityComponent.

Here is the call graph for this function:

5.4.3.3 adjustHappinessBasedOnTaxRate()

```
void CityComposite::adjustHappinessBasedOnTaxRate ( )
```

Adjusts the city's happiness level based on the current tax rate.

This function recalculates happiness based on tax policies affecting citizens and zones.

This function applies a happiness adjustment to NPCs in response to the income tax rate. Here is the call graph for this function:

5.4.3.4 checkCityConditions()

```
bool CityComposite::checkCityConditions ( )
```

Checks if the city meets certain conditions.

Checks if the city meets specific conditions based on building counts.

This function evaluates specific conditions for city management. The conditions and their checks are implemented within the function.

Returns

true if conditions are met; otherwise, false.

This function evaluates conditions based on the number of public service, residential, and utility buildings, and sets population growth if conditions are met.

Returns

true if the city conditions are met; false otherwise.

Here is the call graph for this function:

5.4.3.5 connectZones()

```
void CityComposite::connectZones (
            CityComponent * zoneA,
            CityComponent * zoneB,
            std::unique_ptr< Transport > transport )
```

Connects two zones with a specified transport method.

Establishes a connection between two city zones using the given transport object, representing roads, railways, etc.

Parameters

zoneA	A pointer to the first zone.
zoneB	A pointer to the second zone.
transport	A unique pointer to the Transport object representing the connection.

5.4.3.6 displayStatus()

```
void CityComposite::displayStatus ( ) [override], [virtual]
```

Displays the status of the city composite and all its zones.

Displays the status of the city and its zones.

This function iterates over all zones and displays their status information.

This function prints the city name, iterates through each zone to display their status, and displays government statistics.

Implements CityComponent.

Here is the call graph for this function:

5.4.3.7 getBuildingType()

```
std::string CityComposite::getBuildingType ( ) const  [inline], [override], [virtual]
```

Returns the type of building as a string.

Returns

A string indicating this component is a "City".

Implements CityComponent.

5.4.3.8 remove()

```
void CityComposite::remove (
             std::shared_ptr< CityComponent > component )  [override], [virtual]
```

Removes a city component (zone) from the composite.

Removes a city component (zone) from the city.

Parameters

component	A shared pointer to the CityComponent to remove.
component	A shared pointer to the CityComponent to be removed.

Reimplemented from CityComponent.

5.4.3.9 updateHappinessForNewComponent()

```
void CityComposite::updateHappinessForNewComponent (
             const std::shared_ptr< CityComponent > & component )
```

Updates the city's happiness level based on the addition of a new component.

This function modifies happiness levels to reflect the influence of newly added components, such as new residential or utility zones.

Parameters

component	A shared pointer to the new CityComponent being added.

This function calculates the impact of a new component on happiness based on its type. It may also adjust employment rates and pollution levels for specific building types.

Parameters

component	A shared pointer to the CityComponent being added.

Here is the call graph for this function: Here is the caller graph for this function:

5.4.4 Member Data Documentation

5.4.4.1 cityName

`std::string CityComposite::cityName [private]`

Name of the city

5.4.4.2 zones

`std::vector<std::shared_ptr<CityComponent> > CityComposite::zones [private]`

Collection of city zones

The documentation for this class was generated from the following files:

- COS214-Poject/src/CityComposite.h
- COS214-Poject/src/CityComposite.cpp

5.5 CityGame Class Reference

`#include <CityGame.h>`

Collaboration diagram for CityGame:

Public Member Functions

- CityGame (int width, int height)
- void run ()

Private Member Functions

- void initializeResources ()
- void initializeResourceSystem ()
- void spawnResources ()
- template<typename T >
  void spawnIncomeResource (int count, int quantity, double marketValue)
- template<typename T >
  void spawnConstructionResource (int count, int quantity, int unitCost)
- void displayMenu ()
- void displayCollectedResources () const
- void handleSaveGame ()
- void handleLoadGame ()
- void handlePlaceTransport ()
- void handleCreateZone ()
- void showZoneInfo ()
- void handleBuildStructure ()
- void handlePlaceUtility ()
- void collectTaxes ()
- void showStatistics ()
- void handleProcessResources ()
- bool isResourceInRange (const Location &industryLoc, const Location &resourceLoc, int collectionRange) const
- template<typename T >
  std::pair< std::shared_ptr< T >, Location > findClosestResource (const Location &industryLoc)
- void addCollectedResource (const std::string &resourceName, int amount)
- bool hasRequiredResources (const std::string &buildingType)
- void consumeResources (const std::string &buildingType)
- void displayResourceRequirements (const std::string &buildingType)

Private Attributes

- MapGrid grid
- GameState state
- bool running = true
- std::shared_ptr< CityComponent > city = std::make_shared<CityComposite>("Adventure City")
- std::map< Location, std::shared_ptr< IncomeResourceProduct > > incomeResourceMap
- std::map< Location, std::shared_ptr< ConstructionResourceProduct > > constructionResourceMap
- std::map< std::string, int > collectedResources
- std::vector< std::shared_ptr< IncomeResourceProduct > > incomeResources
- std::vector< std::shared_ptr< ConstructionResourceProduct > > constructionResources
- std::vector< std::unique_ptr< GameStateMemento > > savedStates
- size_t currentSaveIndex = 0
- Government & government = Government::getInstance()
- NPCManager & npcManager = NPCManager::getInstance()

5.5.1 Constructor & Destructor Documentation

5.5.1.1 CityGame()

```
CityGame::CityGame (
             int width,
             int height )  [inline]
```

Here is the call graph for this function:

5.5.2 Member Function Documentation

5.5.2.1 addCollectedResource()

```
void CityGame::addCollectedResource (
             const std::string & resourceName,
             int amount )  [inline], [private]
```

5.5.2.2 collectTaxes()

```
void CityGame::collectTaxes ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.3 consumeResources()

```
void CityGame::consumeResources (
             const std::string & buildingType )  [inline], [private]
```

Here is the caller graph for this function:

5.5.2.4 displayCollectedResources()

```
void CityGame::displayCollectedResources ( ) const  [inline], [private]
```

Here is the caller graph for this function:

5.5.2.5 displayMenu()

```
void CityGame::displayMenu ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.6 displayResourceRequirements()

```
void CityGame::displayResourceRequirements (
             const std::string & buildingType )  [inline], [private]
```

5.5.2.7 findClosestResource()

```
template<typename T >
std::pair<std::shared_ptr<T>, Location> CityGame::findClosestResource (
             const Location & industryLoc )  [inline], [private]
```

5.5.2.8 handleBuildStructure()

```
void CityGame::handleBuildStructure ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.9 handleCreateZone()

```
void CityGame::handleCreateZone ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.10 handleLoadGame()

```
void CityGame::handleLoadGame ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.11 handlePlaceTransport()

```
void CityGame::handlePlaceTransport ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.12 handlePlaceUtility()

```
void CityGame::handlePlaceUtility ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.13 handleProcessResources()

```
void CityGame::handleProcessResources ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.14 handleSaveGame()

```
void CityGame::handleSaveGame ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.15 hasRequiredResources()

```
bool CityGame::hasRequiredResources (
             const std::string & buildingType )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.16 initializeResources()

```
void CityGame::initializeResources ( )  [inline], [private]
```

Here is the caller graph for this function:

5.5.2.17 initializeResourceSystem()

```
void CityGame::initializeResourceSystem ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.18 isResourceInRange()

```
bool CityGame::isResourceInRange (
             const Location & industryLoc,
             const Location & resourceLoc,
             int collectionRange ) const  [inline], [private]
```

5.5.2.19 run()

```
void CityGame::run ( )  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.20 showStatistics()

```
void CityGame::showStatistics ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.21 showZoneInfo()

```
void CityGame::showZoneInfo ( )  [inline], [private]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.5.2.22 spawnConstructionResource()

```
template<typename T >
void CityGame::spawnConstructionResource (
             int count,
             int quantity,
             int unitCost )  [inline], [private]
```

Here is the call graph for this function:

5.5.2.23 spawnIncomeResource()

```
template<typename T >
void CityGame::spawnIncomeResource (
             int count,
             int quantity,
             double marketValue )  [inline], [private]
```

Here is the call graph for this function:

5.5.2.24 spawnResources()

```
void CityGame::spawnResources ( )  [inline], [private]
```

Here is the caller graph for this function:

5.5.3 Member Data Documentation

5.5.3.1 city

```
std::shared_ptr<CityComponent> CityGame::city = std::make_shared<CityComposite>("Adventure
City")  [private]
```

5.5.3.2 collectedResources

```
std::map<std::string, int> CityGame::collectedResources  [private]
```

5.5.3.3 constructionResourceMap

```
std::map<Location, std::shared_ptr<ConstructionResourceProduct> > CityGame::construction←
ResourceMap  [private]
```

5.5.3.4 constructionResources

```
std::vector<std::shared_ptr<ConstructionResourceProduct> > CityGame::constructionResources
[private]
```

5.5.3.5 currentSaveIndex

```
size_t CityGame::currentSaveIndex = 0  [private]
```

5.5.3.6 government

```
Government& CityGame::government = Government::getInstance()  [private]
```

5.5.3.7 grid

```
MapGrid CityGame::grid  [private]
```

5.5.3.8 incomeResourceMap

```
std::map<Location, std::shared_ptr<IncomeResourceProduct> > CityGame::incomeResourceMap  [private]
```

5.5.3.9 incomeResources

```
std::vector<std::shared_ptr<IncomeResourceProduct> > CityGame::incomeResources  [private]
```

5.5.3.10 npcManager

```
NPCManager& CityGame::npcManager = NPCManager::getInstance()  [private]
```

5.5.3.11 running

```
bool CityGame::running = true  [private]
```

5.5.3.12 savedStates

```
std::vector<std::unique_ptr<GameStateMemento> > CityGame::savedStates  [private]
```

5.5.3.13 state

```
GameState CityGame::state  [private]
```

The documentation for this class was generated from the following file:

- COS214-Poject/src/CityGame.h

5.6 Coal Class Reference

Represents a coal resource in the game, derived from IncomeResourceProduct.

```
#include <Coal.h>
```

Inheritance diagram for Coal:

Collaboration diagram for Coal:

Public Member Functions

- Coal (int quantity, double marketValue)

  Constructs a Coal object with a specified quantity and market value.
- ∼Coal ()

  Destructor for the Coal class.
- void displayStatus () override

  Displays the status of the coal resource.
- std::string getBuildingType () const

  Retrieves the type of building/resource, which is "Coal" for this class.

5.6.1 Detailed Description

Represents a coal resource in the game, derived from IncomeResourceProduct.

This class provides functionality specific to coal resources, including initialization, status display, and identifying its type as "Coal".

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Coal()

```
Coal::Coal (
            int quantity,
            double marketValue )
```

Constructs a Coal object with a specified quantity and market value.

Parameters

quantity	The initial quantity of coal.
marketValue	The market value per unit of coal.

Initializes the coal resource with a name, quantity, and market value.

Parameters

quantity	The initial quantity of coal.
marketValue	The market value per unit of coal.

5.6.2.2 ∼Coal()

```
Coal::∼Coal ( )
```

Destructor for the Coal class.

Cleans up resources for the coal object if needed.

5.6.3 Member Function Documentation

5.6.3.1 displayStatus()

```
void Coal::displayStatus ( )  [override], [virtual]
```

Displays the status of the coal resource.

Displays the current status of the coal resource.

Overrides the displayStatus method to show details specific to coal.

Outputs details specific to coal, including quantity, market value per unit, and total calculated value.

Implements CityComponent.

Here is the call graph for this function:

5.6.3.2 getBuildingType()

```
std::string Coal::getBuildingType ( ) const  [inline], [virtual]
```

Retrieves the type of building/resource, which is "Coal" for this class.

Returns

A string representing the building type as "Coal".

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Coal.h
- COS214-Poject/src/Coal.cpp

5.7 CollectionStrategy Class Reference

Base class for collection strategies in the NPC system.

```
#include <CollectionStrategy.h>
```

Inheritance diagram for CollectionStrategy:

Collaboration diagram for CollectionStrategy:

Public Member Functions

- virtual ~CollectionStrategy ()=default

  Virtual destructor for the CollectionStrategy class.
- virtual int collect (int baseAmount) const =0

  Pure virtual function to collect resources based on a strategy.
- virtual ~CollectionStrategy ()=default
- virtual int collect (int amount) const =0

  Pure virtual function to determine collection efficiency.

5.7.1 Detailed Description

Base class for collection strategies in the NPC system.

Abstract base class for resource collection strategies in the game.

Provides an interface for defining different collection efficiency levels for Worker NPCs.

Defines the interface for various strategies that determine how resources are collected. Concrete strategies derived from this class will implement specific collection behaviors.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 ~CollectionStrategy() [1/2]

```
virtual CollectionStrategy::~CollectionStrategy ( )  [virtual], [default]
```

Virtual destructor for the CollectionStrategy class.

Ensures proper cleanup of resources for derived classes.

5.7.2.2 ~CollectionStrategy() [2/2]

```
virtual CollectionStrategy::~CollectionStrategy ( )  [virtual], [default]
```

5.7.3 Member Function Documentation

5.7.3.1 collect() [1/2]

```
virtual int CollectionStrategy::collect (
              int amount ) const  [pure virtual]
```

Pure virtual function to determine collection efficiency.

Parameters

amount	The base amount to be collected.

Returns

int The adjusted amount based on the strategy's efficiency.

Implemented in FastCollectionStrategy, ModerateCollectionStrategy, and SlowCollectionStrategy.

5.7.3.2 collect() [2/2]

```
virtual int CollectionStrategy::collect (
            int baseAmount ) const  [pure virtual]
```

Pure virtual function to collect resources based on a strategy.

Derived classes implement this method to adjust the amount collected based on a specific collection strategy.

Parameters

baseAmount	The base amount to be collected.

Returns

The adjusted amount collected based on the strategy.

Implemented in FastCollectionStrategy, ModerateCollectionStrategy, and SlowCollectionStrategy.

The documentation for this class was generated from the following files:

- COS214-Poject/src/CollectionStrategy.h
- COS214-Poject/src/NPCSystem.h

5.8 Command Class Reference

Abstract base class representing a command in the command pattern.

```
#include <Command.h>
```

Inheritance diagram for Command:

Collaboration diagram for Command:

Public Member Functions

- virtual ∼Command ()=default

  Virtual destructor for the Command base class.
- virtual void execute ()=0

  Executes the command.
- virtual void undo ()=0

  Undoes the command.
- virtual std::unique_ptr< Command > clone () const =0

  Clones the command, creating a new unique pointer to a Command.

5.8.1 Detailed Description

Abstract base class representing a command in the command pattern.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 ∼Command()

```
virtual Command::∼Command ( )  [virtual], [default]
```

Virtual destructor for the Command base class.

5.8.3 Member Function Documentation

5.8.3.1 clone()

```
virtual std::unique_ptr<Command> Command::clone ( ) const  [pure virtual]
```

Clones the command, creating a new unique pointer to a Command.

Returns

A unique pointer to a copy of the command.

Implemented in PlaceComponentCommand.

5.8.3.2 execute()

```
virtual void Command::execute ( ) [pure virtual]
```

Executes the command.

Implemented in PlaceComponentCommand.

5.8.3.3 undo()

```
virtual void Command::undo ( ) [pure virtual]
```

Undoes the command.

Implemented in PlaceComponentCommand.

The documentation for this class was generated from the following file:

- COS214-Poject/src/Command.h

5.9 CommercialBuilding Class Reference

Represents a commercial building in the city, capable of interacting with utilities and tax collection.

```
#include <CommercialBuilding.h>
```

Inheritance diagram for CommercialBuilding:

Collaboration diagram for CommercialBuilding:

Public Member Functions

- CommercialBuilding (double cost, std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

    Constructs a CommercialBuilding with specified utilities and cost.
- void accept (taxCollector ∗TC) override

    Accepts a tax collector to process tax collection for the building.
- void setTaxPaid (bool paid)

    Sets the tax-paid status for the building.
- double getPrice () const

    Gets the price of the building.
- bool isTaxPaid () const

    Checks if the tax for the building has been paid.
- std::string getBuildingType () const override

    Gets the type of the building as a string.
- void displayStatus () override

    Displays the status of the building, including utility and tax information.
- virtual char getDisplaySymbol () const =0

Gets the display symbol for the building on the city grid.
- int getUtilityCoverage () const

  Calculates the building's utility coverage.
- void setWaterSupply (std::shared_ptr< UtilityFlyweight > utility)

  Sets the water supply utility for the building.
- void setPowerSupply (std::shared_ptr< UtilityFlyweight > utility)

  Sets the power supply utility for the building.
- void setWasteManagement (std::shared_ptr< UtilityFlyweight > utility)

  Sets the waste management utility for the building.
- void setSewageManagement (std::shared_ptr< UtilityFlyweight > utility)

  Sets the sewage management utility for the building.
- virtual std::string getDisplayColor () const

  Gets the display color based on utility coverage.

Protected Attributes

- double price
- bool taxPaid
- std::shared_ptr< UtilityFlyweight > waterSupply
- std::shared_ptr< UtilityFlyweight > powerSupply
- std::shared_ptr< UtilityFlyweight > wasteManagement
- std::shared_ptr< UtilityFlyweight > sewageManagement

5.9.1 Detailed Description

Represents a commercial building in the city, capable of interacting with utilities and tax collection.

This class provides functionalities for tax management, utility connections, and displaying the building status on the city grid.

5.9.2 Constructor & Destructor Documentation

5.9.2.1 CommercialBuilding()

```
CommercialBuilding::CommercialBuilding (
            double price,
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a CommercialBuilding with specified utilities and cost.

Constructs a CommercialBuilding with specified price and utility connections.

Parameters

cost	The price of the building.
water	Pointer to the water supply utility.
power	Pointer to the power supply utility.
waste	Pointer to waste management utility.
sewage	Pointer to sewage management utility.

Initializes the building with a price and establishes connections to water, power, waste, and sewage utilities if provided.

Parameters

price	The cost of the commercial building.
water	Shared pointer to the water supply utility.
power	Shared pointer to the power supply utility.
waste	Shared pointer to waste management utility.
sewage	Shared pointer to sewage management utility.

5.9.3 Member Function Documentation

5.9.3.1 accept()

```
void CommercialBuilding::accept (
            taxCollector * TC )  [override], [virtual]
```

Accepts a tax collector to process tax collection for the building.

Accepts a tax collector to process tax collection for this building.

Parameters

TC	Pointer to the tax collector.

Invokes the tax collector's visit method, enabling tax processing for the building.

Parameters

TC	Pointer to the tax collector.

Implements CityComponent.

Here is the call graph for this function:

5.9.3.2 displayStatus()

```
void CommercialBuilding::displayStatus ( )  [override], [virtual]
```

Displays the status of the building, including utility and tax information.

Displays the status of the commercial building.

Outputs the building's type, price, tax status, and utility coverage to the console.

Implements CityComponent.

Here is the call graph for this function:

5.9.3.3 getBuildingType()

```
std::string CommercialBuilding::getBuildingType ( ) const  [override], [virtual]
```

Gets the type of the building as a string.

Retrieves the building type as a string.

Returns

The type of the building.

A string representing the building type ("Commercial Building").

Implements CityComponent.

Reimplemented in Shops, Office, and Malls.

Here is the caller graph for this function:

5.9.3.4 getDisplayColor()

```
virtual std::string CommercialBuilding::getDisplayColor ( ) const  [inline], [virtual]
```

Gets the display color based on utility coverage.

Uses ANSI color codes to return a string representing the color: yellow for full coverage, white for partial coverage, and red for insufficient coverage.

Returns

A string containing the ANSI color code for the display color.

Here is the call graph for this function:

5.9.3.5 getDisplaySymbol()

```
virtual char CommercialBuilding::getDisplaySymbol ( ) const  [pure virtual]
```

Gets the display symbol for the building on the city grid.

Returns

A character representing the building symbol.

Implemented in Shops, Office, and Malls.

5.9.3.6 getPrice()

```
double CommercialBuilding::getPrice ( ) const  [inline]
```

Gets the price of the building.

Returns

The price of the building.

Here is the caller graph for this function:

5.9.3.7 getUtilityCoverage()

```
int CommercialBuilding::getUtilityCoverage ( ) const
```

Calculates the building's utility coverage.

Calculates the utility coverage for the building.

Returns

An integer representing the utility coverage level.

Counts the number of connected utilities (water, power, waste, sewage) and returns the total.

Returns

An integer representing the number of connected utilities (0 to 4).

Here is the caller graph for this function:

5.9.3.8 isTaxPaid()

```
bool CommercialBuilding::isTaxPaid ( ) const  [inline]
```

Checks if the tax for the building has been paid.

Returns

True if tax is paid, false otherwise.

5.9.3.9 setPowerSupply()

```
void CommercialBuilding::setPowerSupply (
             std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

Sets the power supply utility for the building.

Parameters

utility	Shared pointer to the power supply utility.

5.9.3.10 setSewageManagement()

```
void CommercialBuilding::setSewageManagement (
             std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

Sets the sewage management utility for the building.

Parameters

utility	Shared pointer to the sewage management utility.

5.9.3.11 setTaxPaid()

```
void CommercialBuilding::setTaxPaid (
             bool paid )
```

Sets the tax-paid status for the building.

Sets the tax-paid status for the commercial building.

Parameters

paid	Boolean indicating whether tax has been paid.

Marks the tax as paid and, if paid, increases the employment rate to reflect the positive economic impact.

Parameters

paid	Boolean indicating whether tax has been paid.

Here is the call graph for this function: Here is the caller graph for this function:

5.9.3.12 setWasteManagement()

```
void CommercialBuilding::setWasteManagement (
             std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

Sets the waste management utility for the building.

Parameters

utility	Shared pointer to the waste management utility.

5.9.3.13 setWaterSupply()

```
void CommercialBuilding::setWaterSupply (
             std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

Sets the water supply utility for the building.

Parameters

utility	Shared pointer to the water supply utility.

5.9.4 Member Data Documentation

5.9.4.1 powerSupply

```
std::shared_ptr<UtilityFlyweight> CommercialBuilding::powerSupply  [protected]
```

Pointer to the power supply utility

5.9.4.2 price

```
double CommercialBuilding::price  [protected]
```

The cost of the commercial building

5.9.4.3 sewageManagement

```
std::shared_ptr<UtilityFlyweight> CommercialBuilding::sewageManagement  [protected]
```

Pointer to sewage management utility

5.9.4.4 taxPaid

```
bool CommercialBuilding::taxPaid  [protected]
```

Status indicating whether taxes have been paid

5.9.4.5 wasteManagement

```
std::shared_ptr<UtilityFlyweight> CommercialBuilding::wasteManagement  [protected]
```

Pointer to waste management utility

5.9.4.6 waterSupply

```
std::shared_ptr<UtilityFlyweight> CommercialBuilding::waterSupply  [protected]
```

Pointer to the water supply utility

The documentation for this class was generated from the following files:

- COS214-Poject/src/CommercialBuilding.h
- COS214-Poject/src/CommercialBuilding.cpp

5.10 CommercialFactory Class Reference

Abstract factory class for creating commercial buildings.

```
#include <CommercialFactory.h>
```

Inheritance diagram for CommercialFactory:

Collaboration diagram for CommercialFactory:

Public Member Functions

- virtual std::unique_ptr< CommercialBuilding > createBuilding ()=0

  Creates a commercial building.
- virtual ∼CommercialFactory ()

  Virtual destructor for the CommercialFactory class.

5.10.1 Detailed Description

Abstract factory class for creating commercial buildings.

Provides an interface for creating different types of commercial buildings, allowing subclasses to define the specific type of commercial building created.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 ∼CommercialFactory()

```
virtual CommercialFactory::∼CommercialFactory ( )  [inline], [virtual]
```

Virtual destructor for the CommercialFactory class.

5.10.3 Member Function Documentation

5.10.3.1 createBuilding()

```
virtual std::unique_ptr<CommercialBuilding> CommercialFactory::createBuilding ( )  [pure virtual]
```

Creates a commercial building.

This pure virtual function must be implemented by derived classes to create a specific type of commercial building.

Returns

A unique pointer to a newly created CommercialBuilding.

Implemented in OfficeFactory, ShopsFactory, and MallsFactory.

The documentation for this class was generated from the following file:

- COS214-Poject/src/CommercialFactory.h

5.11 Concrete Class Reference

Represents a concrete resource used for construction within the game.

```
#include <Concrete.h>
```

Inheritance diagram for Concrete:

Collaboration diagram for Concrete:

Public Member Functions

- Concrete (int quantity, int unitCost)

  Constructs a Concrete resource with specified quantity and unit cost.
- ∼Concrete ()

  Destructor for the Concrete class.
- void displayStatus () override

  Displays the current status of the concrete resource.
- std::string getBuildingType () const

  Gets the building type for this resource.

5.11.1 Detailed Description

Represents a concrete resource used for construction within the game.

This class provides specific functionality and characteristics for concrete as a construction resource, including cost and quantity management.

5.11.2 Constructor & Destructor Documentation

5.11.2.1 Concrete()

```
Concrete::Concrete (
            int quantity,
            int unitCost )
```

Constructs a Concrete resource with specified quantity and unit cost.

Constructs a Concrete resource with a specified quantity and unit cost.

Parameters

quantity	The amount of concrete units available.
unitCost	The cost per unit of concrete.

Initializes the concrete resource, setting its name to "Concrete" and establishing the initial quantity and cost per unit.

Parameters

quantity	The amount of concrete available.
unitCost	The cost per unit of concrete.

5.11.2.2 ∼Concrete()

```
Concrete::∼Concrete ( )
```

Destructor for the Concrete class.

Cleans up resources associated with the Concrete object.

5.11.3 Member Function Documentation

5.11.3.1 displayStatus()

`void Concrete::displayStatus ( )  [override], [virtual]`

Displays the current status of the concrete resource.

Displays the status of the concrete resource.

Outputs details specific to concrete, including quantity and cost, to the console.

Outputs details specific to the concrete resource, including its quantity, unit cost, and total cost, to the console.

Implements CityComponent.

Here is the call graph for this function:

5.11.3.2 getBuildingType()

`std::string Concrete::getBuildingType ( ) const  [inline], [virtual]`

Gets the building type for this resource.

Returns

A string representing the building type ("Concrete").

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Concrete.h
- COS214-Poject/src/Concrete.cpp

5.12 ConcreteTaxCollector Class Reference

A concrete implementation of the taxCollector interface, responsible for collecting taxes from various types of buildings and zones.

`#include <concreteTaxCollector.h>`

Inheritance diagram for ConcreteTaxCollector:

Collaboration diagram for ConcreteTaxCollector:

Public Member Functions

- void visit (ResidentialBuilding ∗RB)

  Collects tax from a residential building.
- void visit (CommercialBuilding ∗CB)

  Collects tax from a commercial building.
- void visit (ZoneComposite ∗Rzone)

  Collects tax from a composite zone.

Additional Inherited Members

5.12.1 Detailed Description

A concrete implementation of the taxCollector interface, responsible for collecting taxes from various types of buildings and zones.

The ConcreteTaxCollector class defines specific tax collection behaviors for residential buildings, commercial buildings, and zones within the city.

5.12.2 Member Function Documentation

5.12.2.1 visit() [1/3]

```
void ConcreteTaxCollector::visit (
            CommercialBuilding * CB )  [virtual]
```

Collects tax from a commercial building.

Defines the tax collection behavior for commercial buildings, taking into account specific commercial tax rules.

Parameters

CB	Pointer to the CommercialBuilding from which to collect tax.

Calculates an 8% tax based on the building's price and adds it to the government's funds.

Parameters

CB	Pointer to the CommercialBuilding from which to collect tax.

Implements taxCollector.

Here is the call graph for this function:

5.12.2.2 visit() [2/3]

```
void ConcreteTaxCollector::visit (
            ResidentialBuilding * RB )  [virtual]
```

Collects tax from a residential building.

Defines the tax collection behavior for residential buildings by accessing the necessary resources and applying tax rates specific to residential properties.

Parameters

RB	Pointer to the ResidentialBuilding from which to collect tax.

Calculates a 5% tax based on the building's price and adds it to the government's funds. If no NPCs are present in the building, no tax is collected.

Parameters

RB	Pointer to the ResidentialBuilding from which to collect tax.

Implements taxCollector.

Here is the call graph for this function: Here is the caller graph for this function:

5.12.2.3 visit() [3/3]

```
void ConcreteTaxCollector::visit (
            ZoneComposite * zone )  [virtual]
```

Collects tax from a composite zone.

Collects tax from all buildings within a zone.

Defines the tax collection behavior for a zone, which may encompass multiple buildings of various types.

Parameters

Rzone	Pointer to the ZoneComposite from which to collect tax.

Iterates through the buildings in the given ZoneComposite and collects tax from each by invoking the appropriate tax collection method.

Parameters

zone	Pointer to the ZoneComposite containing buildings to collect tax from.

Implements taxCollector.

Here is the call graph for this function:

The documentation for this class was generated from the following files:

- COS214-Poject/src/concreteTaxCollector.h
- COS214-Poject/src/concreteTaxCollector.cpp

5.13 ConstructionResourceProcessor Class Reference

Manages the processing and storage of construction resources within the game.

```
#include <ConstructionResourceProcessor.h>
```

Inheritance diagram for ConstructionResourceProcessor:

Collaboration diagram for ConstructionResourceProcessor:

Public Member Functions

- ConstructionResourceProcessor (std::shared_ptr< ConstructionResourceProduct > res, int maxStorage)
  Constructs a ConstructionResourceProcessor with specified resource and storage capacity.
- void process (int amount)
  Processes a specified amount of the resource.
- void store (int amount)
  Stores a specified amount of the resource, if capacity allows.
- void display () const
  Displays the current status of the resource processor.
- int getCurrentStorage () const
  Gets the current storage amount.

Private Attributes

- std::shared_ptr< ConstructionResourceProduct > resource
- int currentStorage
- const int maxStorage

5.13.1 Detailed Description

Manages the processing and storage of construction resources within the game.

This class handles construction resources by allowing processing and storage up to a defined maximum capacity, and it provides methods to display the resource status.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 ConstructionResourceProcessor()

```
ConstructionResourceProcessor::ConstructionResourceProcessor (
              std::shared_ptr< ConstructionResourceProduct > res,
              int maxStorage )
```

Constructs a ConstructionResourceProcessor with specified resource and storage capacity.

Constructs a ConstructionResourceProcessor with the given resource and maximum storage capacity.

Parameters

res	A shared pointer to the construction resource product to be processed.
maxStorage	The maximum storage capacity for this processor.

Initializes the processor with a specific construction resource and sets the maximum storage capacity.

Parameters

res	A shared pointer to the construction resource to be processed.
maxStorage	The maximum capacity for storing this resource.

5.13.3 Member Function Documentation

5.13.3.1 display()

```
void ConstructionResourceProcessor::display ( ) const  [virtual]
```

Displays the current status of the resource processor.

Displays the current storage status of the resource processor.

Outputs details about the current storage, maximum storage, and the type of resource being managed.

Outputs the current and maximum storage levels, as well as the resource's details.

Implements ResourceProcessor.

5.13.3.2 getCurrentStorage()

```
int ConstructionResourceProcessor::getCurrentStorage ( ) const  [virtual]
```

Gets the current storage amount.

Gets the current storage amount of the resource.

Returns

The current amount of resource stored.

The amount of resource currently stored.

Implements ResourceProcessor.

5.13.3.3 process()

```
void ConstructionResourceProcessor::process (
             int amount )  [virtual]
```

Processes a specified amount of the resource.

Handles resource processing logic, consuming a certain amount of the stored resource.

Parameters

amount	The amount of resource to process.

Decreases the storage by the specified amount if enough resource is available, and adds the processed resource to the government's resources. Outputs a message if there isn't enough in storage to process the requested amount.

Parameters

amount	The quantity of the resource to process.

Implements ResourceProcessor.

Here is the call graph for this function:

5.13.3.4 store()

```
void ConstructionResourceProcessor::store (
             int amount )  [virtual]
```

Stores a specified amount of the resource, if capacity allows.

Stores a specified amount of the resource.

Adds the given amount of resource to storage, up to the maximum storage limit.

Parameters

amount	The amount of resource to store.

Increases the storage by the specified amount if it doesn't exceed the max capacity. Outputs a message if storage is full.

Parameters

amount	The quantity of the resource to store.

Implements ResourceProcessor.

5.13.4 Member Data Documentation

5.13.4.1 currentStorage

```
int ConstructionResourceProcessor::currentStorage  [private]
```

The current amount of resource in storage.

5.13.4.2 maxStorage

```
const int ConstructionResourceProcessor::maxStorage  [private]
```

The maximum storage capacity for the resource.

5.13.4.3 resource

```
std::shared_ptr<ConstructionResourceProduct> ConstructionResourceProcessor::resource  [private]
```

The construction resource being processed.

The documentation for this class was generated from the following files:

- COS214-Poject/src/ConstructionResourceProcessor.h
- COS214-Poject/src/ConstructionResourceProcessor.cpp

5.14 ConstructionResourceProduct Class Reference

Represents a construction resource product in the city-building simulation.

`#include <ConstructionResourceProduct.h>`

Inheritance diagram for ConstructionResourceProduct:

Collaboration diagram for ConstructionResourceProduct:

Public Member Functions

- ConstructionResourceProduct (string name, int quantity, int unitCost)
  Constructs a ConstructionResourceProduct with specified name, quantity, and unit cost.
- ∼ConstructionResourceProduct ()
  Destructor for ConstructionResourceProduct.
- void ConsumeResource (int amount)
  Consumes a specified amount of the resource.
- void displayStatus ()
  Displays the current status of the resource, including name, quantity, and total cost.
- int getTotalCost () const
  Calculates and returns the total cost of the available resource.
- int getQuantity () const
  Returns the current quantity of the resource.
- void replenish (int amount)
  Replenishes the resource by adding a specified amount.
- std::string getBuildingType ()
  Retrieves the building type for this resource product.
- int getUnitCost () const
  Retrieves the unit cost of the resource.
- bool isReadyForCollection ()
  Checks if the resource is ready for collection.
- virtual void accept (taxCollector ∗TC)
  Allows tax collection on the resource by accepting a tax collector.
- std::string getName () const
  Retrieves the name of the resource.
- void setLocation (const Location &loc)
  Sets the location of the resource.
- Location getLocation () const
  Retrieves the location of the resource.

Private Attributes

- int quantity
- int unitCost
- string name
- bool readyForCollection
- Location location

5.14.1 Detailed Description

Represents a construction resource product in the city-building simulation.

This class models a resource used for construction, managing attributes like quantity, unit cost, and location, and providing methods for consumption, replenishment, and status display.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 ConstructionResourceProduct()

```
ConstructionResourceProduct::ConstructionResourceProduct (
            string name,
            int quantity,
            int unitCost )
```

Constructs a ConstructionResourceProduct with specified name, quantity, and unit cost.

Parameters

name	The name of the resource.
quantity	The initial quantity of the resource.
unitCost	The cost per unit of the resource.

5.14.2.2 ∼ConstructionResourceProduct()

```
ConstructionResourceProduct::∼ConstructionResourceProduct ( )
```

Destructor for ConstructionResourceProduct.

5.14.3 Member Function Documentation

5.14.3.1 accept()

```
virtual void ConstructionResourceProduct::accept (
            taxCollector * TC ) [inline], [virtual]
```

Allows tax collection on the resource by accepting a tax collector.

Parameters

TC	The tax collector.

Implements CityComponent.

5.14.3.2 ConsumeResource()

```
void ConstructionResourceProduct::ConsumeResource (
            int amount )
```

Consumes a specified amount of the resource.

Consumes a specified amount of the resource, decreasing the quantity.

Parameters

amount	The amount to consume.

5.14.3.3 displayStatus()

```
void ConstructionResourceProduct::displayStatus ( ) [virtual]
```

Displays the current status of the resource, including name, quantity, and total cost.

Displays the current status of the resource, including its name, quantity, unit cost, and total value.

Implements CityComponent.

Reimplemented in Wood, Stone, and Steel.

Here is the call graph for this function:

5.14.3.4 getBuildingType()

```
std::string ConstructionResourceProduct::getBuildingType ( ) [inline]
```

Retrieves the building type for this resource product.

Returns

A string representing the building type.

5.14.3.5 getLocation()

```
Location ConstructionResourceProduct::getLocation ( ) const  [inline]
```

Retrieves the location of the resource.

Returns

The location of the resource.

5.14.3.6 getName()

```
std::string ConstructionResourceProduct::getName ( ) const  [inline]
```

Retrieves the name of the resource.

Returns

The name of the resource.

5.14.3.7 getQuantity()

```
int ConstructionResourceProduct::getQuantity ( ) const
```

Returns the current quantity of the resource.

Gets the current quantity of the resource.

Returns

The quantity of the resource.

The current quantity of the resource.

Here is the caller graph for this function:

5.14.3.8 getTotalCost()

```
int ConstructionResourceProduct::getTotalCost ( ) const
```

Calculates and returns the total cost of the available resource.

Returns

The total cost of the resource.

The total cost, calculated as quantity $*$ unit cost.

Here is the caller graph for this function:

5.14.3.9 getUnitCost()

```
int ConstructionResourceProduct::getUnitCost ( ) const
```

Retrieves the unit cost of the resource.

Returns

The unit cost of the resource.

Here is the caller graph for this function:

5.14.3.10 isReadyForCollection()

```
bool ConstructionResourceProduct::isReadyForCollection ( )
```

Checks if the resource is ready for collection.

Checks if the resource is ready for collection based on a minimum threshold.

Returns

True if the resource is ready for collection, false otherwise.

True if the resource quantity is above or equal to the threshold, otherwise false.

5.14.3.11 replenish()

```
void ConstructionResourceProduct::replenish (
             int amount )
```

Replenishes the resource by adding a specified amount.

Replenishes the resource by adding a specified amount to the quantity.

Parameters

amount	The amount to add to the resource quantity.
amount	The amount to add to the current quantity.

5.14.3.12 setLocation()

```
void ConstructionResourceProduct::setLocation (
             const Location & loc )  [inline]
```

Sets the location of the resource.

Parameters

loc	The location to set for the resource.

5.14.4 Member Data Documentation

5.14.4.1 location

```
Location ConstructionResourceProduct::location  [private]
```

The location of the resource

5.14.4.2 name

```
string ConstructionResourceProduct::name  [private]
```

Name of the resource

5.14.4.3 quantity

```
int ConstructionResourceProduct::quantity  [private]
```

The quantity of the resource available

5.14.4.4 readyForCollection

```
bool ConstructionResourceProduct::readyForCollection  [private]
```

Indicates if the resource is ready for collection

5.14.4.5 unitCost

```
int ConstructionResourceProduct::unitCost  [private]
```

The cost per unit of the resource

The documentation for this class was generated from the following files:

- COS214-Poject/src/ConstructionResourceProduct.h
- COS214-Poject/src/ConstructionResourceProduct.cpp

5.15 CrimeState Class Reference

Represents the crime state for NPCs, impacting their behavior and the city's overall conditions.

```
#include <CrimeState.h>
```

Inheritance diagram for CrimeState:

Collaboration diagram for CrimeState:

Public Member Functions

- void handle () override
  Handles actions specific to the crime state.
- std::string getStateName () override
  Retrieves the name of the current state.
- NPCState ∗ clone () const override
  Creates a copy of the current CrimeState instance.
- ∼CrimeState () override=default
  Default destructor for CrimeState.

5.15.1 Detailed Description

Represents the crime state for NPCs, impacting their behavior and the city's overall conditions.

In this state, NPCs behave in a manner that reflects an increase in crime rates, potentially affecting city statistics.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 ∼CrimeState()

```
CrimeState::∼CrimeState ( )  [override], [default]
```

Default destructor for CrimeState.

5.15.3 Member Function Documentation

5.15.3.1 clone()

```
NPCState * CrimeState::clone ( ) const  [override], [virtual]
```

Creates a copy of the current CrimeState instance.

Returns

A pointer to a newly cloned CrimeState object.

A pointer to a new CrimeState object, cloned from this instance.

Implements NPCState.

5.15.3.2 getStateName()

```
std::string CrimeState::getStateName ( )  [override], [virtual]
```

Retrieves the name of the current state.

Returns

A string representing the name of this state, "Crime".

The name of this state, "CrimeState".

Implements NPCState.

5.15.3.3 handle()

```
void CrimeState::handle ( )  [override], [virtual]
```

Handles actions specific to the crime state.

Handles the effects of the CrimeState on the city's mortality rate.

Executes the behavior associated with NPCs in the crime state, impacting relevant NPC or city statistics.

Calculates the likelihood and impact of crime affecting the mortality rate based on the number of NPCs in the crime state. If a randomly generated value falls within the calculated chance, the mortality rate is increased by a factor proportional to the crime NPC count. < Base probability of crime impact.

< Mortality rate increase factor per crime NPC.

Implements NPCState.

Here is the call graph for this function:

The documentation for this class was generated from the following files:

- COS214-Poject/src/CrimeState.h
- COS214-Poject/src/CrimeState.cpp

5.16 CrystalCraftIndustry Class Reference

Represents an industry focused on processing diamonds and stone for various uses.

```
#include <CrystalCraftIndustry.h>
```

Inheritance diagram for CrystalCraftIndustry:

Collaboration diagram for CrystalCraftIndustry:

Public Member Functions

- CrystalCraftIndustry (std::shared_ptr< IncomeResourceProduct > diamond, std::shared_ptr< ConstructionResourceProduct > stone, MapGrid *grid, std::map< std::string, int > &collectedResources)

  Constructs a CrystalCraftIndustry with specific resources and map grid location.
- void processDiamond (int amount)

  Processes a specified amount of diamonds.
- void processStone (int amount)

  Processes a specified amount of stone.

Static Public Attributes

- static const int CRYSTAL_CRAFT_RANGE = 5

Additional Inherited Members

5.16.1 Detailed Description

Represents an industry focused on processing diamonds and stone for various uses.

This class defines specialized methods for processing diamonds and stone, leveraging specific resource processors.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 CrystalCraftIndustry()

```
CrystalCraftIndustry::CrystalCraftIndustry (
             std::shared_ptr< IncomeResourceProduct > diamond,
             std::shared_ptr< ConstructionResourceProduct > stone,
             MapGrid * grid,
             std::map< std::string, int > & collectedResources )
```

Constructs a CrystalCraftIndustry with specific resources and map grid location.

Constructs a CrystalCraftIndustry with specified diamond and stone resources, along with map grid and collected resources.

Parameters

diamond	Shared pointer to an IncomeResourceProduct representing diamond.
stone	Shared pointer to a ConstructionResourceProduct representing stone.
grid	Pointer to the MapGrid where the industry is located.
collectedResources	Reference to a map of collected resources.

Initializes the industry with processors for income and construction resources, setting up the necessary infrastructure for resource processing.

Parameters

diamond	Shared pointer to an IncomeResourceProduct representing diamond.
stone	Shared pointer to a ConstructionResourceProduct representing stone.
grid	Pointer to the MapGrid object, which manages the industry's location and layout.
collectedResources	Reference to a map of collected resources for the city.

5.16.3 Member Function Documentation

5.16.3.1 processDiamond()

```
void CrystalCraftIndustry::processDiamond (
             int amount )
```

Processes a specified amount of diamonds.

Processes a specified amount of diamond, contributing to the city's income resources.

Parameters

amount	The amount of diamonds to process.

The method uses the income resource processor to convert diamond quantities into processed output.

Parameters

amount	The amount of diamond to process.

Here is the call graph for this function:

5.16.3.2 processStone()

```
void CrystalCraftIndustry::processStone (
             int amount )
```

Processes a specified amount of stone.

Processes a specified amount of stone, contributing to the city's construction resources.

Parameters

amount	The amount of stone to process.

The method uses the construction resource processor to convert stone quantities into processed output.

Parameters

amount	The amount of stone to process.

Here is the call graph for this function:

5.16.4 Member Data Documentation

5.16.4.1 CRYSTAL_CRAFT_RANGE

```
const int CrystalCraftIndustry::CRYSTAL_CRAFT_RANGE = 5  [static]
```

The range within which this industry can gather resources.

The documentation for this class was generated from the following files:

- COS214-Poject/src/CrystalCraftIndustry.h
- COS214-Poject/src/CrystalCraftIndustry.cpp

5.17 CulturalCenter Class Reference

Represents a cultural center landmark in the city, providing cultural activities and boosting nearby happiness.

```
#include <CulturalCenter.h>
```

Inheritance diagram for CulturalCenter:

Collaboration diagram for CulturalCenter:

Public Member Functions

- CulturalCenter (const std::string &type, int capacity, double price, std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Constructs a CulturalCenter with specified attributes.
- ∼CulturalCenter ()

  Destructor for CulturalCenter.
- void displayStatus ()

  Displays the current status of the cultural center, including capacity and utility connections.
- std::string getBuildingType () const

  Gets the building type as a string.
- std::unique_ptr< LandMark > clone () const

  Creates a copy of the current CulturalCenter instance.

Additional Inherited Members

5.17.1 Detailed Description

Represents a cultural center landmark in the city, providing cultural activities and boosting nearby happiness.

5.17.2 Constructor & Destructor Documentation

5.17.2.1 CulturalCenter()

```
CulturalCenter::CulturalCenter (
            const std::string & type,
            int capacity,
            double price,
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a CulturalCenter with specified attributes.

Constructs a CulturalCenter with specified type, capacity, price, and utility connections.

Parameters

type	The type of cultural center.
capacity	The seating or attendance capacity of the cultural center.
price	The cost of constructing the cultural center.
water	Shared pointer to a UtilityFlyweight object representing the water supply.
power	Shared pointer to a UtilityFlyweight object representing the power supply.
waste	Shared pointer to a UtilityFlyweight object representing the waste management system.
sewage	Shared pointer to a UtilityFlyweight object representing the sewage management system.

5.17.2.2 ∼CulturalCenter()

```
CulturalCenter::∼CulturalCenter ( )
```

Destructor for CulturalCenter.

5.17.3 Member Function Documentation

5.17.3.1 clone()

```
std::unique_ptr< LandMark > CulturalCenter::clone ( ) const
```

Creates a copy of the current CulturalCenter instance.

Returns

A unique pointer to a cloned CulturalCenter object.

5.17.3.2 displayStatus()

```
void CulturalCenter::displayStatus ( ) [virtual]
```

Displays the current status of the cultural center, including capacity and utility connections.

Displays the current status of the Cultural Center, including its type and utility connections.

Reimplemented from LandMark.

Here is the call graph for this function:

5.17.3.3 getBuildingType()

```
std::string CulturalCenter::getBuildingType ( ) const [virtual]
```

Gets the building type as a string.

Returns

A string indicating the building type.

A string indicating the building type ("Cultural Center").

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/CulturalCenter.h
- COS214-Poject/src/CulturalCenter.cpp

5.18 CulturalCenterFactory Class Reference

Factory class responsible for creating CulturalCenter landmarks.

```
#include <CulturalCenterFactory.h>
```

Inheritance diagram for CulturalCenterFactory:

Collaboration diagram for CulturalCenterFactory:

Public Member Functions

- std::unique_ptr< LandMark > createLandMark ()

  Creates a new CulturalCenter landmark.

Additional Inherited Members

5.18.1 Detailed Description

Factory class responsible for creating CulturalCenter landmarks.

The CulturalCenterFactory class inherits from LandMark and provides a method to create instances of CulturalCenter, which represent cultural landmarks in the city simulation.

5.18.2 Member Function Documentation

5.18.2.1 createLandMark()

```
std::unique_ptr< LandMark > CulturalCenterFactory::createLandMark ( )
```

Creates a new CulturalCenter landmark.

Creates a new instance of CulturalCenter with default parameters.

Returns

A unique pointer to a newly created CulturalCenter instance.

This function constructs a CulturalCenter landmark with default values for type, capacity, price, and utility connections.

Returns

A unique pointer to a new CulturalCenter instance.

The documentation for this class was generated from the following files:

- COS214-Poject/src/CulturalCenterFactory.h
- COS214-Poject/src/CulturalCenterFactory.cpp

5.19 Diamonds Class Reference

Represents diamonds as an income-generating resource in the city simulation.

```
#include <Diamonds.h>
```

Inheritance diagram for Diamonds:

Collaboration diagram for Diamonds:

Public Member Functions

- Diamonds (int quantity, double marketValue)
  Constructs a Diamonds resource with specified quantity and market value.
- ∼Diamonds ()
  Destructor for the Diamonds resource.
- void displayStatus () override
  Displays the current status of the diamond resource.
- std::string getBuildingType () const
  Retrieves the building type of this resource.

5.19.1 Detailed Description

Represents diamonds as an income-generating resource in the city simulation.

The Diamonds class inherits from IncomeResourceProduct and provides functionalities specific to diamond resources, including displaying their status and fetching their type.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 Diamonds()

```
Diamonds::Diamonds (
             int quantity,
             double marketValue )
```

Constructs a Diamonds resource with specified quantity and market value.

Parameters

quantity	The amount of diamonds available.
marketValue	The market value per unit of diamonds.

Initializes the Diamonds resource with a given quantity and market value per unit.

Parameters

quantity	The amount of diamonds available.
marketValue	The market value per unit of diamonds.

5.19.2.2 ∼Diamonds()

```
Diamonds::∼Diamonds ( )
```

Destructor for the Diamonds resource.

Cleans up resources associated with the Diamonds object upon destruction.

5.19.3 Member Function Documentation

5.19.3.1 displayStatus()

```
void Diamonds::displayStatus ( )  [override], [virtual]
```

Displays the current status of the diamond resource.

Displays the current status of the Diamonds resource.

Outputs the resource's name, quantity, market value per unit, and total calculated income to the console.

Implements CityComponent.

Here is the call graph for this function:

5.19.3.2 getBuildingType()

```
std::string Diamonds::getBuildingType ( ) const  [inline], [virtual]
```

Retrieves the building type of this resource.

Returns

A string indicating the resource type, "Diamond".

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Diamonds.h
- COS214-Poject/src/Diamonds.cpp

5.20 DonationState Class Reference

Represents the state where NPCs are in a donation mood.

```
#include <DonationState.h>
```

Inheritance diagram for DonationState:

Collaboration diagram for DonationState:

Public Member Functions

- void handle () override
  Handles the donation logic for NPCs in this state.
- std::string getStateName () override
  Gets the name of the current state.
- NPCState ∗ clone () const override
  Clones the current DonationState instance.
- ∼DonationState () override
  Destructor for the DonationState class.

5.20.1 Detailed Description

Represents the state where NPCs are in a donation mood.

This class handles the logic for NPCs when they are willing to donate, affecting the city's economy and happiness levels.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 ∼DonationState()

```
DonationState::∼DonationState ( )  [override], [default]
```

Destructor for the DonationState class.

Cleans up any resources associated with the DonationState.

This destructor cleans up any resources associated with the DonationState.

5.20.3 Member Function Documentation

5.20.3.1 clone()

```
NPCState * DonationState::clone ( ) const  [override], [virtual]
```

Clones the current DonationState instance.

Returns

A pointer to a new DonationState instance that is a copy of this state.

Implements NPCState.

5.20.3.2 getStateName()

```
std::string DonationState::getStateName ( )  [override], [virtual]
```

Gets the name of the current state.

Returns

A string representing the name of the donation state.

Implements NPCState.

5.20.3.3 handle()

```
void DonationState::handle ( )  [override], [virtual]
```

Handles the donation logic for NPCs in this state.

Handles the logic for NPCs in the donation state.

This method is called to process the actions related to NPC donations, including any impacts on the game state or economy.

This method calculates whether a donation occurs based on the number of NPCs in the donation state and updates the city's funds accordingly.

The chance of receiving an investment increases with the number of NPCs in the DonationState. If an investment occurs, the specified amount is added to the city's funds.

Implements NPCState.

Here is the call graph for this function:

The documentation for this class was generated from the following files:

- COS214-Poject/src/DonationState.h
- COS214-Poject/src/DonationState.cpp

5.21 Estate Class Reference

Represents an estate building in the city simulation.

`#include <Estate.h>`

Inheritance diagram for Estate:

Collaboration diagram for Estate:

Public Member Functions

- Estate ()

  Default constructor for the Estate class.
- Estate (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_↩ ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Parameterized constructor for the Estate class.
- char getDisplaySymbol () const override

  Gets the display symbol for the estate.
- std::string getBuildingType () const override

  Gets the type of the building.

Static Public Attributes

- static constexpr int BASE_COST = 300

Additional Inherited Members

5.21.1 Detailed Description

Represents an estate building in the city simulation.

Inherits from the ResidentialBuilding class and provides functionality specific to estate buildings, including their cost and utility connections.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 Estate() [1/2]

```
Estate::Estate ( )
```

Default constructor for the Estate class.

Initializes an estate with default settings.

Initializes an estate with a base capacity of 6 and a base cost of $300. All utility connections are set to nullptr.

5.21.2.2 Estate() [2/2]

```
Estate::Estate (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Parameterized constructor for the Estate class.

Initializes an estate with the specified utility connections.

Parameters

water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage management utility.

Initializes an estate with specified utility connections and a base capacity of 6 and a base cost of $300.

Parameters

water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage management utility.

5.21.3 Member Function Documentation

5.21.3.1 getBuildingType()

```
std::string Estate::getBuildingType ( ) const  [inline], [override], [virtual]
```

Gets the type of the building.

Returns

A string indicating the building type ("Estate").

Implements CityComponent.

5.21.3.2 getDisplaySymbol()

```
char Estate::getDisplaySymbol ( ) const  [inline], [override], [virtual]
```

Gets the display symbol for the estate.

Returns

A character representing the estate on the city grid ('E').

Implements ResidentialBuilding.

5.21.4 Member Data Documentation

5.21.4.1 BASE_COST

```
constexpr int Estate::BASE_COST = 300  [static], [constexpr]
```

Base cost for constructing an estate.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Estate.h
- COS214-Poject/src/Estate.cpp

5.22 EstateFactory Class Reference

The EstateFactory class is responsible for creating Estate buildings.

```
#include <EstateFactory.h>
```

Inheritance diagram for EstateFactory:

Collaboration diagram for EstateFactory:

Public Member Functions

- std::unique_ptr< ResidentialBuilding > createBuilding ()

  Creates a new instance of an Estate building.

5.22.1 Detailed Description

The EstateFactory class is responsible for creating Estate buildings.

This class inherits from the ResidentialBuildingFactory and provides a concrete implementation for creating instances of ResidentialBuilding specifically for the Estate type.

5.22.2 Member Function Documentation

5.22.2.1 createBuilding()

```
std::unique_ptr< ResidentialBuilding > EstateFactory::createBuilding ( )  [virtual]
```

Creates a new instance of an Estate building.

This method returns a unique pointer to a ResidentialBuilding that is specifically an Estate.

Returns

std::unique_ptr<ResidentialBuilding> A unique pointer to the newly created Estate building.

This method overrides the createBuilding method from the ResidentialBuildingFactory class to specifically create an instance of an Estate. It returns a unique pointer to the newly created Estate object.

Returns

std::unique_ptr<ResidentialBuilding> A unique pointer to the newly created Estate building.

Implements ResidentialBuildingFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/EstateFactory.h
- COS214-Poject/src/EstateFactory.cpp

5.23 FastCollectionStrategy Class Reference

Collection strategy for fast collection rate.

`#include <FastCollectionStrategy.h>`

Inheritance diagram for FastCollectionStrategy:

Collaboration diagram for FastCollectionStrategy:

Public Member Functions

- int collect (int baseAmount)
- int collect (int amount) const override

    Pure virtual function to collect resources based on a strategy.

5.23.1 Detailed Description

Collection strategy for fast collection rate.

Implements a 200% collection efficiency.

5.23.2 Member Function Documentation

5.23.2.1 collect() [1/2]

```
int FastCollectionStrategy::collect (
            int baseAmount ) const  [inline], [override], [virtual]
```

Pure virtual function to collect resources based on a strategy.

Derived classes implement this method to adjust the amount collected based on a specific collection strategy.

Parameters

baseAmount	The base amount to be collected.

Returns

The adjusted amount collected based on the strategy.

< Returns double the input amount.

Implements CollectionStrategy.

5.23.2.2 collect() [2/2]

```
int FastCollectionStrategy::collect (
            int baseAmount )
```

The documentation for this class was generated from the following files:

- COS214-Poject/src/FastCollectionStrategy.h
- COS214-Poject/src/NPCSystem.h
- COS214-Poject/src/FastCollectionStrategy.cpp

5.24 FireStation Class Reference

Represents a fire station in the city.

`#include <FireStation.h>`

Inheritance diagram for FireStation:

Collaboration diagram for FireStation:

Public Member Functions

- FireStation ()=default

  Default constructor for FireStation.
- FireStation (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > electricity, std↩ ::shared_ptr< UtilityFlyweight > wasteManagement, std::shared_ptr< UtilityFlyweight > sewage, std::string buildingStatus)

  Parameterized constructor for FireStation.
- ∼FireStation ()=default

  Default destructor for FireStation.
- void provideService () override

  Provides fire protection services.
- std::unique_ptr< PublicService > clone () const

  Clones the current FireStation object.
- char getDisplaySymbol () const override

  Gets the display symbol for the fire station.
- void displayStatus () override

  Displays the current status of the fire station.

Private Attributes

- std::string status

  Current status of the fire station (e.g., Operational, Out of Service).

Additional Inherited Members

5.24.1 Detailed Description

Represents a fire station in the city.

The FireStation class is a concrete implementation of the PublicService interface. It provides services related to fire safety and emergency response within the city.

5.24.2 Constructor & Destructor Documentation

5.24.2.1 FireStation() [1/2]

```
FireStation::FireStation ( )  [default]
```

Default constructor for FireStation.

5.24.2.2 FireStation() [2/2]

```
FireStation::FireStation (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > electricity,
            std::shared_ptr< UtilityFlyweight > wasteManagement,
            std::shared_ptr< UtilityFlyweight > sewage,
            std::string buildingStatus )
```

Parameterized constructor for FireStation.

Parameters

water	Shared pointer to the water utility.
electricity	Shared pointer to the electricity utility.
wasteManagement	Shared pointer to the waste management utility.
sewage	Shared pointer to the sewage utility.
buildingStatus	Status of the fire station (e.g., Operational).

Initializes a FireStation object with the specified utilities and building status.

Parameters

water	Shared pointer to the water utility.
electricity	Shared pointer to the electricity utility.
wasteManagement	Shared pointer to the waste management utility.
sewage	Shared pointer to the sewage utility.
buildingStatus	The operational status of the fire station.

5.24.2.3 ∼FireStation()

```
FireStation::∼FireStation ( )  [default]
```

Default destructor for FireStation.

5.24.3 Member Function Documentation

5.24.3.1 clone()

```
std::unique_ptr< PublicService > FireStation::clone ( ) const
```

Clones the current FireStation object.

Returns

A unique pointer to a new FireStation object that is a clone of this one.

A unique pointer to a new FireStation object that is a copy of this one.

5.24.3.2 displayStatus()

```
void FireStation::displayStatus ( )  [override], [virtual]
```

Displays the current status of the fire station.

This method outputs relevant information about the fire station's status and operational capacity.

Outputs information regarding the connectivity of utilities and the overall operational status of the fire station.

Implements PublicService.

5.24.3.3 getDisplaySymbol()

```
char FireStation::getDisplaySymbol ( ) const  [override], [virtual]
```

Gets the display symbol for the fire station.

Returns

The character symbol representing the fire station.

The character symbol representing the fire station, which is 'F'.

Implements PublicService.

5.24.3.4 provideService()

```
void FireStation::provideService ( )  [override], [virtual]
```

Provides fire protection services.

Provides fire extinguishing services.

This method implements the service provision functionality specific to the fire station.

This method simulates the fire station providing services to citizens in need of fire assistance.

Implements PublicService.

5.24.4 Member Data Documentation

5.24.4.1 status

```
std::string FireStation::status  [private]
```

Current status of the fire station (e.g., Operational, Out of Service).

The documentation for this class was generated from the following files:

- COS214-Poject/src/FireStation.h
- COS214-Poject/src/FireStation.cpp

5.25 FireStationFactory Class Reference

Factory class for creating FireStation objects.

`#include <FireStationFactory.h>`

Inheritance diagram for FireStationFactory:

Collaboration diagram for FireStationFactory:

Public Member Functions

- std::unique_ptr< PublicService > createPublicService ()
  Creates a new FireStation object.

5.25.1 Detailed Description

Factory class for creating FireStation objects.

Inherits from the PublicServiceFactory and provides the functionality to create instances of FireStation.

5.25.2 Member Function Documentation

5.25.2.1 createPublicService()

```
std::unique_ptr< PublicService > FireStationFactory::createPublicService ( )  [virtual]
```

Creates a new FireStation object.

This method returns a unique pointer to a newly created FireStation instance.

Returns

A unique pointer to a PublicService (FireStation).

This function utilizes the FireStation class to instantiate a new FireStation object and returns it as a unique pointer to the PublicService base class.

Returns

A unique pointer to a PublicService representing the newly created FireStation.

Implements PublicServiceFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/FireStationFactory.h
- COS214-Poject/src/FireStationFactory.cpp

5.26 Flat Class Reference

Represents a residential building of type Flat.

`#include <Flat.h>`

Inheritance diagram for Flat:

Collaboration diagram for Flat:

Public Member Functions

- Flat ()

    Default constructor for Flat.
- Flat (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

    Parameterized constructor for Flat.
- char getDisplaySymbol () const override

    Get the display symbol for Flat.
- std::string getBuildingType () const override

    Get the building type for Flat.

Static Public Attributes

- static constexpr int BASE_COST = 150

    Base cost of constructing a Flat.

Additional Inherited Members

5.26.1 Detailed Description

Represents a residential building of type Flat.

The Flat class inherits from ResidentialBuilding and includes specific properties and methods related to a flat residential structure, such as base cost and utility connections.

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Flat() [1/2]

```
Flat::Flat ( )
```

Default constructor for Flat.

Default constructor for the Flat class.

This constructor initializes a Flat object with default parameters, setting the number of floors to 2 and the base cost to $150.00.

5.26.2.2 Flat() [2/2]

```
Flat::Flat (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Parameterized constructor for Flat.

Parameterized constructor for the Flat class.

Parameters

water	A shared pointer to the UtilityFlyweight for water supply.
power	A shared pointer to the UtilityFlyweight for power supply.
waste	A shared pointer to the UtilityFlyweight for waste management.
sewage	A shared pointer to the UtilityFlyweight for sewage management.

This constructor initializes a Flat object with the specified utility connections, the number of floors set to 2, and the base cost set to $150.00.

Parameters

water	A shared pointer to the UtilityFlyweight for water supply.
power	A shared pointer to the UtilityFlyweight for power supply.
waste	A shared pointer to the UtilityFlyweight for waste management.
sewage	A shared pointer to the UtilityFlyweight for sewage management.

5.26.3 Member Function Documentation

5.26.3.1 getBuildingType()

```
std::string Flat::getBuildingType ( ) const  [inline], [override], [virtual]
```

Get the building type for Flat.

Returns

A string representing the type of building, which is "Flat".

Implements CityComponent.

5.26.3.2 getDisplaySymbol()

```
char Flat::getDisplaySymbol ( ) const  [inline], [override], [virtual]
```

Get the display symbol for Flat.

Returns

A character symbol representing the Flat (usually 'F').

Implements ResidentialBuilding.

5.26.4 Member Data Documentation

5.26.4.1 BASE_COST

```
constexpr int Flat::BASE_COST = 150  [static], [constexpr]
```

Base cost of constructing a Flat.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Flat.h
- COS214-Poject/src/Flat.cpp

5.27 FlatFactory Class Reference

Factory class for creating Flat residential buildings.

```
#include <FlatFactory.h>
```

Inheritance diagram for FlatFactory:

Collaboration diagram for FlatFactory:

Public Member Functions

- std::unique_ptr< ResidentialBuilding > createBuilding ()
  Creates a unique pointer to a ResidentialBuilding of type Flat.

5.27.1 Detailed Description

Factory class for creating Flat residential buildings.

The FlatFactory class inherits from ResidentialBuildingFactory and provides an implementation for creating instances of Flat buildings.

5.27.2 Member Function Documentation

5.27.2.1 createBuilding()

```
std::unique_ptr< ResidentialBuilding > FlatFactory::createBuilding ( )  [virtual]
```

Creates a unique pointer to a ResidentialBuilding of type Flat.

Creates a unique pointer to a Flat residential building.

This method instantiates a Flat object and returns it as a unique pointer.

Returns

std::unique_ptr<ResidentialBuilding> A unique pointer to the created Flat object.

This function instantiates a Flat object and returns it as a unique pointer. The created Flat building has default attributes as specified in its constructor.

Returns

std::unique_ptr<ResidentialBuilding> A unique pointer to the created Flat object.

Implements ResidentialBuildingFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/FlatFactory.h
- COS214-Poject/src/FlatFactory.cpp

5.28 GameState Class Reference

Manages the game state and command history.

```
#include <GameState.h>
```

Collaboration diagram for GameState:

Public Member Functions

- std::unique_ptr< GameStateMemento > createMemento () const

  Creates a memento that captures the current game state.
- void restoreFromMemento (const GameStateMemento &memento)

  Restores the game state from a given memento.
- void executeCommand (std::unique_ptr< Command > command)

  Executes a command and adds it to the command history.
- void undo ()

  Undoes the last executed command.
- void redo ()

  Redoes the last undone command.

Private Attributes

- std::vector< std::unique_ptr< Command > > commandHistory
- size_t currentCommandIndex = 0

5.28.1 Detailed Description

Manages the game state and command history.

The GameState class maintains a history of executed commands and allows undoing and redoing of commands. It also supports saving and restoring the game state using mementos.

5.28.2 Member Function Documentation

5.28.2.1 createMemento()

```
std::unique_ptr<GameStateMemento> GameState::createMemento ( ) const  [inline]
```

Creates a memento that captures the current game state.

Returns

std::unique_ptr<GameStateMemento> A unique pointer to the created memento, containing the current command history and index.

Here is the caller graph for this function:

5.28.2.2 executeCommand()

```
void GameState::executeCommand (
            std::unique_ptr< Command > command )  [inline]
```

Executes a command and adds it to the command history.

This method clears any redo history when a new command is executed, then executes the command and adds it to the history.

Parameters

command	A unique pointer to the command to execute.

Here is the caller graph for this function:

5.28.2.3 redo()

```
void GameState::redo ( )  [inline]
```

Redoes the last undone command.

If there is a command to redo, it executes the command at the current command index and increments the index. Here is the caller graph for this function:

5.28.2.4 restoreFromMemento()

```
void GameState::restoreFromMemento (
             const GameStateMemento & memento )  [inline]
```

Restores the game state from a given memento.

This function clears the existing command history and restores it from the specified memento, including the current command index.

Parameters

memento	The memento to restore the game state from.

Here is the caller graph for this function:

5.28.2.5 undo()

```
void GameState::undo ( )  [inline]
```

Undoes the last executed command.

If there is a command to undo, it decrements the current command index and invokes the undo operation on the last command. Here is the caller graph for this function:

5.28.3 Member Data Documentation

5.28.3.1 commandHistory

```
std::vector<std::unique_ptr<Command> > GameState::commandHistory  [private]
```

A history of executed commands.

5.28.3.2 currentCommandIndex

```
size_t GameState::currentCommandIndex = 0 [private]
```

The index of the current command for undo/redo operations.

The documentation for this class was generated from the following file:

- COS214-Poject/src/GameState.h

5.29 GameStateMemento Class Reference

Represents a memento for the GameState, capturing command history and the current command index.

```
#include <GameStateMemento.h>
```

Collaboration diagram for GameStateMemento:

Public Member Functions

- GameStateMemento (const std::vector< std::unique_ptr< Command >> &history, size_t index)

  Constructs a GameStateMemento with a command history and current command index.

Static Public Member Functions

- static std::unique_ptr< GameStateMemento > create (const std::vector< std::unique_ptr< Command >> &history, size_t index)

  Static factory method to create a new GameStateMemento instance.

Private Attributes

- std::vector< std::unique_ptr< Command > > commandHistory

  The history of commands executed.
- size_t currentCommandIndex

  The index of the current command in the history.

Friends

- class GameState

  Allows GameState to access private members.

5.29.1 Detailed Description

Represents a memento for the GameState, capturing command history and the current command index.

The GameStateMemento class allows for saving and restoring the state of the game, specifically the history of commands executed and the current position in that history. It provides a way to implement undo and redo functionality.

5.29.2 Constructor & Destructor Documentation

5.29.2.1 GameStateMemento()

```
GameStateMemento::GameStateMemento (
            const std::vector< std::unique_ptr< Command >> & history,
            size_t index )  [inline]
```

Constructs a GameStateMemento with a command history and current command index.

Parameters

history	A vector of unique pointers to the command history.
index	The current command index.

Here is the caller graph for this function:

5.29.3 Member Function Documentation

5.29.3.1 create()

```
static std::unique_ptr<GameStateMemento> GameStateMemento::create (
            const std::vector< std::unique_ptr< Command >> & history,
            size_t index )  [inline], [static]
```

Static factory method to create a new GameStateMemento instance.

Parameters

history	A vector of unique pointers to the command history.
index	The current command index.

Returns

A unique pointer to the newly created GameStateMemento.

Here is the call graph for this function:

5.29.4 Friends And Related Function Documentation

5.29.4.1 GameState

```
friend class GameState [friend]
```

Allows GameState to access private members.

5.29.5 Member Data Documentation

5.29.5.1 commandHistory

```
std::vector<std::unique_ptr<Command> > GameStateMemento::commandHistory [private]
```

The history of commands executed.

5.29.5.2 currentCommandIndex

```
size_t GameStateMemento::currentCommandIndex [private]
```

The index of the current command in the history.

The documentation for this class was generated from the following file:

- COS214-Poject/src/GameStateMemento.h

5.30 Gold Class Reference

Class representing the Gold resource.

```
#include <Gold.h>
```

Inheritance diagram for Gold:

Collaboration diagram for Gold:

Public Member Functions

- Gold (int quantity, double marketValue)
  Constructs a Gold object with specified quantity and market value.
- ~Gold ()
  Destructor for the Gold object.
- void displayStatus () override
  Displays the status of the gold resource.
- std::string getBuildingType () const
  Returns the type of building represented by this resource.

5.30.1 Detailed Description

Class representing the Gold resource.

Inherits from IncomeResourceProduct and provides functionalities specific to the Gold resource, including its quantity and market value.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 Gold()

```
Gold::Gold (
            int quantity,
            double marketValue )
```

Constructs a Gold object with specified quantity and market value.

Parameters

quantity	The amount of gold.
marketValue	The market value of a single unit of gold.

Initializes the Gold resource with its name, quantity, and market value.

Parameters

quantity	The amount of gold.
marketValue	The market value of a single unit of gold.

5.30.2.2 ∼Gold()

```
Gold::∼Gold ( )
```

Destructor for the Gold object.

Cleans up resources associated with the Gold object.

5.30.3 Member Function Documentation

5.30.3.1 displayStatus()

```
void Gold::displayStatus ( )  [override], [virtual]
```

Displays the status of the gold resource.

Outputs the quantity, market value per unit, and total value of the gold resource to the console.

Implements CityComponent.

Here is the call graph for this function:

5.30.3.2 getBuildingType()

```
std::string Gold::getBuildingType ( ) const  [inline], [virtual]
```

Returns the type of building represented by this resource.

Returns

A string indicating the resource type.

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Gold.h
- COS214-Poject/src/Gold.cpp

5.31 Government Class Reference

The Government class manages the state and functionality of the government in the simulation.

```
#include <Government.h>
```

Collaboration diagram for Government:

Public Member Functions

- Government (const Government &)=delete
- Government & operator= (const Government &)=delete
- void displayGovernmentStats ()

    Displays the current statistics of the government.
- void addMoney (double amount)

    Adds a specified amount of money to the government funds.
- void reduceMoney (double amount)

    Reduces the government funds by a specified amount.
- void reduceProduction (double factor)

    Reduces the production rate by a specified factor.
- void increaseProduction (double factor)

    Increases the production rate by a specified factor.
- void increaseCrimeRate (double factor)

    Increases the crime rate by a specified factor.
- double getMoney () const

    Gets the current amount of money the government has.
- double getProductionRate () const

    Gets the current production rate of the government.
- void setProductionRate (double rate)

    Sets the production rate of the government.
- double getMortalityRate () const

    Gets the current mortality rate.
- double getCrimeRate () const

    Gets the current crime rate.
- void setCrimeRate (int rate)

    Sets the crime rate.
- int getPopulationGrowth () const

    Gets the current population growth rate.
- void setPopulationGrowth (int growth)

    Sets the population growth rate.
- int getPopulation () const

    Gets the current population of the city.
- void updatePopulation ()

    Updates the population based on various factors.
- void setTax (double rate)

    Sets the tax rate.
- void increaseEmploymentRate (double rate)

    Increases the employment rate by a specified amount.
- void setBuildingAmount (std::string type, int amount)

    Sets the amount of a specified building type.
- int getBuildingAmount (std::string type)

    Gets the amount of a specified building type.
- void decreasePopulation (int amount)

    Decreases the population by a specified amount.
- void addProcessedResource (const std::string &resourceName, int amount, double value)

    Adds a processed resource to the government's records.
- void increasePopulation (int amount)

    Increases the population by a specified amount.
- void incrementPollutionLevel (int amount)

Increments the pollution level by a specified amount.
- void decrementPollutionLevel (int amount)

  Decrements the pollution level by a specified amount.
- int getPollutionLevel ()

  Gets the current pollution level.
- void calculateTax ()

  Calculates the tax revenue based on current settings.
- double getIncomeTaxRate ()

  Gets the current income tax rate.
- Government ()

  Constructs a Government object with default values.

Static Public Member Functions

- static Government & getInstance ()

  Retrieves the singleton instance of the Government class.

Private Attributes

- double money

  Current money of the government.
- double productionRate

  Current production rate.
- double mortalityRate

  Current mortality rate.
- double crimeRate

  Current crime rate.
- int population

  Current population of the city.
- int populationGrowth

  Current population growth rate.
- double incomeTax

  Current income tax.
- double incomeTaxRate

  Current income tax rate.
- double EMPLOYMENT_RATE

  Current employment rate.
- int publicServiceAmount

  Number of public service buildings.
- int utilityAmount

  Number of utility buildings.
- int residentialAmount

  Number of residential buildings.
- std::map< std::string, int > processedResources

  Store processed resources.
- std::map< std::string, double > resourceValues

  Store resource market values.
- int pollutionLevel

  Current pollution level.

5.31.1 Detailed Description

The Government class manages the state and functionality of the government in the simulation.

This class follows the Singleton design pattern, ensuring only one instance exists throughout the application.

5.31.2 Constructor & Destructor Documentation

5.31.2.1 Government() [1/2]

```
Government::Government (
             const Government & ) [delete]
```

5.31.2.2 Government() [2/2]

```
Government::Government ( ) [inline]
```

Constructs a Government object with default values.

This constructor is private to ensure the singleton pattern is maintained.

5.31.3 Member Function Documentation

5.31.3.1 addMoney()

```
void Government::addMoney (
             double amount )
```

Adds a specified amount of money to the government funds.

Parameters

amount	The amount of money to add.

Here is the caller graph for this function:

5.31.3.2 addProcessedResource()

```
void Government::addProcessedResource (
             const std::string & resourceName,
             int amount,
             double value )
```

Adds a processed resource to the government's records.

Parameters

resourceName	The name of the resource.
amount	The amount of the resource.
value	The market value of the resource.
resourceName	The name of the resource.
amount	The amount of the resource.
value	The market value of the resource (optional).

Here is the caller graph for this function:

5.31.3.3 calculateTax()

```
void Government::calculateTax ( )
```

Calculates the tax revenue based on current settings.

Calculates the income tax based on the current population and tax rate. Here is the caller graph for this function:

5.31.3.4 decreasePopulation()

```
void Government::decreasePopulation (
            int amount )
```

Decreases the population by a specified amount.

Parameters

amount	The amount to decrease the population.

Here is the caller graph for this function:

5.31.3.5 decrementPollutionLevel()

```
void Government::decrementPollutionLevel (
            int amount )
```

Decrements the pollution level by a specified amount.

Parameters

amount	The amount to decrement pollution.

5.31.3.6 displayGovernmentStats()

```
void Government::displayGovernmentStats ( )
```

Displays the current statistics of the government.

Here is the caller graph for this function:

5.31.3.7 getBuildingAmount()

```
int Government::getBuildingAmount (
             std::string type )
```

Gets the amount of a specified building type.

Parameters

type	The type of building.

Returns

int The amount of the specified building type.

Here is the caller graph for this function:

5.31.3.8 getCrimeRate()

```
double Government::getCrimeRate ( ) const
```

Gets the current crime rate.

Returns

double The current crime rate.

5.31.3.9 getIncomeTaxRate()

```
double Government::getIncomeTaxRate ( )
```

Gets the current income tax rate.

Returns

double The current income tax rate.

Here is the caller graph for this function:

5.31.3.10 getInstance()

```
static Government& Government::getInstance ( )  [inline], [static]
```

Retrieves the singleton instance of the Government class.

Returns

Government& Reference to the singleton instance.

Here is the caller graph for this function:

5.31.3.11 getMoney()

```
double Government::getMoney ( ) const
```

Gets the current amount of money the government has.

Returns

double The current money.

Here is the caller graph for this function:

5.31.3.12 getMortalityRate()

```
double Government::getMortalityRate ( ) const
```

Gets the current mortality rate.

Returns

double The current mortality rate.

5.31.3.13 getPollutionLevel()

```
int Government::getPollutionLevel ( )
```

Gets the current pollution level.

Returns

int The current pollution level.

Here is the caller graph for this function:

5.31.3.14 getPopulation()

```
int Government::getPopulation ( ) const
```

Gets the current population of the city.

Returns

int The current population.

Here is the caller graph for this function:

5.31.3.15 getPopulationGrowth()

```
int Government::getPopulationGrowth ( ) const
```

Gets the current population growth rate.

Returns

int The current population growth.

int The current population growth rate.

5.31.3.16 getProductionRate()

```
double Government::getProductionRate ( ) const
```

Gets the current production rate of the government.

Returns

double The current production rate.

Here is the caller graph for this function:

5.31.3.17 increaseCrimeRate()

```
void Government::increaseCrimeRate (
             double factor )
```

Increases the crime rate by a specified factor.

Parameters

factor	The factor by which to increase crime.

Here is the caller graph for this function:

5.31.3.18 increaseEmploymentRate()

```
void Government::increaseEmploymentRate (
             double rate )
```

Increases the employment rate by a specified amount.

Parameters

rate	The amount to increase the employment rate.

Here is the caller graph for this function:

5.31.3.19 increasePopulation()

```
void Government::increasePopulation (
             int amount )
```

Increases the population by a specified amount.

Parameters

amount	The amount to increase the population.

Here is the caller graph for this function:

5.31.3.20 increaseProduction()

```
void Government::increaseProduction (
             double factor )
```

Increases the production rate by a specified factor.

Parameters

factor	The factor by which to increase production.

Here is the caller graph for this function:

5.31.3.21 incrementPollutionLevel()

```
void Government::incrementPollutionLevel (
             int amount )
```

Increments the pollution level by a specified amount.

Parameters

amount	The amount to increment pollution.

Here is the caller graph for this function:

5.31.3.22 operator=()

```
Government& Government::operator= (
             const Government & )  [delete]
```

5.31.3.23 reduceMoney()

```
void Government::reduceMoney (
             double amount )
```

Reduces the government funds by a specified amount.

Parameters

amount	The amount of money to reduce.

Here is the caller graph for this function:

5.31.3.24 reduceProduction()

```
void Government::reduceProduction (
             double factor )
```

Reduces the production rate by a specified factor.

Parameters

factor	The factor by which to reduce production.

Here is the caller graph for this function:

5.31.3.25 setBuildingAmount()

```
void Government::setBuildingAmount (
             std::string type,
             int amount )
```

Sets the amount of a specified building type.

Parameters

type	The type of building.
amount	The amount of the building type to set.

Here is the caller graph for this function:

5.31.3.26 setCrimeRate()

```
void Government::setCrimeRate (
             int rate )
```

Sets the crime rate.

Parameters

rate	The new crime rate.
rate	The new crime rate as a percentage.

Here is the caller graph for this function:

5.31.3.27 setPopulationGrowth()

```
void Government::setPopulationGrowth (
             int growth )
```

Sets the population growth rate.

Parameters

growth	The new population growth.
growth	The new population growth rate.

Here is the caller graph for this function:

5.31.3.28 setProductionRate()

```
void Government::setProductionRate (
             double rate )
```

Sets the production rate of the government.

Parameters

rate	The new production rate.

5.31.3.29 setTax()

```
void Government::setTax (
             double rate )
```

Sets the tax rate.

Sets the income tax rate.

Parameters

rate	The new tax rate.
rate	The new income tax rate.

Here is the caller graph for this function:

5.31.3.30 updatePopulation()

```
void Government::updatePopulation ( )
```

Updates the population based on various factors.

Updates the population based on growth and mortality rate.

5.31.4 Member Data Documentation

5.31.4.1 crimeRate

```
double Government::crimeRate  [private]
```

Current crime rate.

5.31.4.2 EMPLOYMENT_RATE

```
double Government::EMPLOYMENT_RATE  [private]
```

Current employment rate.

5.31.4.3 incomeTax

```
double Government::incomeTax  [private]
```

Current income tax.

5.31.4.4 incomeTaxRate

```
double Government::incomeTaxRate  [private]
```

Current income tax rate.

5.31.4.5 money

```
double Government::money  [private]
```

Current money of the government.

5.31.4.6 mortalityRate

```
double Government::mortalityRate  [private]
```

Current mortality rate.

5.31.4.7 pollutionLevel

```
int Government::pollutionLevel  [private]
```

Current pollution level.

5.31.4.8 population

```
int Government::population  [private]
```

Current population of the city.

5.31.4.9 populationGrowth

```
int Government::populationGrowth  [private]
```

Current population growth rate.

5.31.4.10 processedResources

```
std::map<std::string, int> Government::processedResources  [private]
```

Store processed resources.

5.31.4.11 productionRate

```
double Government::productionRate  [private]
```

Current production rate.

5.31.4.12 publicServiceAmount

```
int Government::publicServiceAmount  [private]
```

Number of public service buildings.

5.31.4.13 residentialAmount

```
int Government::residentialAmount  [private]
```

Number of residential buildings.

5.31.4.14 resourceValues

```
std::map<std::string, double> Government::resourceValues  [private]
```

Store resource market values.

5.31.4.15 utilityAmount

```
int Government::utilityAmount  [private]
```

Number of utility buildings.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Government.h
- COS214-Poject/src/Government.cpp

5.32 Hospital Class Reference

Represents a hospital that provides medical services to citizens.

`#include <Hospital.h>`

Inheritance diagram for Hospital:

Collaboration diagram for Hospital:

Public Member Functions

- Hospital ()=default

    Default constructor.
- Hospital (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > electricity, std↩ ::shared_ptr< UtilityFlyweight > wasteManagement, std::shared_ptr< UtilityFlyweight > sewage, std::string buildingStatus)

    Parameterized constructor for initializing the hospital.
- ∼Hospital ()=default

    Destructor.
- void provideService () override

    Provides medical services to citizens.
- std::unique_ptr< PublicService > clone () const

    Clones the current Hospital object.
- void displayStatus () override

    Displays the current status of the hospital.
- char getDisplaySymbol () const override

    Gets the display symbol for the hospital.
- std::string getBuildingType () const

    Gets the building type of the hospital.

Private Attributes

- bool water

    Indicates if water supply is available.
- bool electricity

    Indicates if electricity supply is available.
- bool sewage

    Indicates if sewage system is connected.
- bool wasteManagement

    Indicates if waste management is operational.
- std::string status

    Current status of the hospital.

Additional Inherited Members

5.32.1 Detailed Description

Represents a hospital that provides medical services to citizens.

The Hospital class inherits from PublicService and includes utilities for water, electricity, sewage, and waste management.

5.32.2 Constructor & Destructor Documentation

5.32.2.1 Hospital() [1/2]

```
Hospital::Hospital ( )  [default]
```

Default constructor.

5.32.2.2 Hospital() [2/2]

```
Hospital::Hospital (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > electricity,
            std::shared_ptr< UtilityFlyweight > wasteManagement,
            std::shared_ptr< UtilityFlyweight > sewage,
            std::string buildingStatus )
```

Parameterized constructor for initializing the hospital.

Constructs a Hospital object with specified utilities and status.

Parameters

water	A shared pointer to the water utility.
electricity	A shared pointer to the electricity utility.
wasteManagement	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage utility.
buildingStatus	The status of the hospital.

5.32.2.3 ∼Hospital()

```
Hospital::∼Hospital ( )  [default]
```

Destructor.

5.32.3 Member Function Documentation

5.32.3.1 clone()

```
std::unique_ptr< PublicService > Hospital::clone ( ) const
```

Clones the current Hospital object.

Returns

A unique pointer to a cloned Hospital object.

5.32.3.2 displayStatus()

```
void Hospital::displayStatus ( )  [override], [virtual]
```

Displays the current status of the hospital.

Displays the current status of the hospital, including utility connections.

Implements PublicService.

5.32.3.3 getBuildingType()

```
std::string Hospital::getBuildingType ( ) const  [virtual]
```

Gets the building type of the hospital.

Returns

A string representing the building type.

Implements CityComponent.

5.32.3.4 getDisplaySymbol()

```
char Hospital::getDisplaySymbol ( ) const  [override], [virtual]
```

Gets the display symbol for the hospital.

Returns

The character symbol representing the hospital.

Implements PublicService.

5.32.3.5 provideService()

```
void Hospital::provideService ( )  [inline], [override], [virtual]
```

Provides medical services to citizens.

Implements PublicService.

5.32.4 Member Data Documentation

5.32.4.1 electricity

```
bool Hospital::electricity  [private]
```

Indicates if electricity supply is available.

5.32.4.2 sewage

```
bool Hospital::sewage  [private]
```

Indicates if sewage system is connected.

5.32.4.3 status

```
std::string Hospital::status  [private]
```

Current status of the hospital.

5.32.4.4 wasteManagement

```
bool Hospital::wasteManagement  [private]
```

Indicates if waste management is operational.

5.32.4.5 water

```
bool Hospital::water [private]
```

Indicates if water supply is available.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Hospital.h
- COS214-Poject/src/Hospital.cpp

5.33 HospitalFactory Class Reference

A factory class for creating Hospital instances.

```
#include <HospitalFactory.h>
```

Inheritance diagram for HospitalFactory:

Collaboration diagram for HospitalFactory:

Public Member Functions

- std::unique_ptr< PublicService > createPublicService ()
  Creates a new instance of a PublicService, specifically a Hospital.

5.33.1 Detailed Description

A factory class for creating Hospital instances.

The HospitalFactory class derives from PublicServiceFactory and implements a method to create Hospital objects. This allows for the encapsulation of the object creation logic, making it easier to manage dependencies and variations in the construction of Hospital objects.

5.33.2 Member Function Documentation

5.33.2.1 createPublicService()

```
std::unique_ptr< PublicService > HospitalFactory::createPublicService ( )  [virtual]
```

Creates a new instance of a PublicService, specifically a Hospital.

Creates a new Hospital instance.

Returns

A unique pointer to the newly created Hospital object.

This method creates and returns a unique pointer to a newly instantiated Hospital object. The Hospital is constructed using the default constructor, which initializes its attributes.

Returns

A unique pointer to the created Hospital object.

Implements PublicServiceFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/HospitalFactory.h
- COS214-Poject/src/HospitalFactory.cpp

5.34 House Class Reference

Represents a house, which is a type of residential building.

```
#include <House.h>
```

Inheritance diagram for House:

Collaboration diagram for House:

Public Member Functions

- House ()

  Default constructor for the House class.
- House (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_↩ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Parameterized constructor for the House class.
- char getDisplaySymbol () const override

  Gets the display symbol for the house.
- std::string getBuildingType () const override

  Gets the building type of the house.

Static Public Attributes

- static constexpr int BASE_COST = 100

  Base cost of the house.

Additional Inherited Members

5.34.1 Detailed Description

Represents a house, which is a type of residential building.

This class inherits from the ResidentialBuilding class and adds specific properties and methods related to a house.

5.34.2 Constructor & Destructor Documentation

5.34.2.1 House() [1/2]

```
House::House ( )
```

Default constructor for the House class.

Initializes a house with default values for its utilities.

5.34.2.2 House() [2/2]

```
House::House (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Parameterized constructor for the House class.

Parameters

water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste utility.
sewage	A shared pointer to the sewage utility.

5.34.3 Member Function Documentation

5.34.3.1 getBuildingType()

```
std::string House::getBuildingType ( ) const  [inline], [override], [virtual]
```

Gets the building type of the house.

Returns

A string representing the type of building.

Implements CityComponent.

5.34.3.2 getDisplaySymbol()

```
char House::getDisplaySymbol ( ) const  [inline], [override], [virtual]
```

Gets the display symbol for the house.

Returns

A character representing the house (H).

Implements ResidentialBuilding.

5.34.4 Member Data Documentation

5.34.4.1 BASE_COST

```
constexpr int House::BASE_COST = 100  [static], [constexpr]
```

Base cost of the house.

The documentation for this class was generated from the following files:

- COS214-Poject/src/House.h
- COS214-Poject/src/House.cpp

5.35 HouseFactory Class Reference

Factory class for creating House objects.

```
#include <HouseFactory.h>
```

Inheritance diagram for HouseFactory:

Collaboration diagram for HouseFactory:

Public Member Functions

- std::unique_ptr< ResidentialBuilding > createBuilding () override

  Creates a unique pointer to a new House object.

5.35.1 Detailed Description

Factory class for creating House objects.

The HouseFactory is a concrete factory class that implements the ResidentialBuildingFactory interface to produce House objects.

5.35.2 Member Function Documentation

5.35.2.1 createBuilding()

```
std::unique_ptr< ResidentialBuilding > HouseFactory::createBuilding ( )  [override], [virtual]
```

Creates a unique pointer to a new House object.

This function overrides the createBuilding method in ResidentialBuildingFactory to return a House, which is a specific type of ResidentialBuilding.

Returns

std::unique_ptr<ResidentialBuilding> A unique pointer to a newly created House object.

Implements ResidentialBuildingFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/HouseFactory.h
- COS214-Poject/src/HouseFactory.cpp

5.36 IncomeResourceProcessor Class Reference

Processes and manages storage for income-generating resources.

```
#include <IncomeResourceProcessor.h>
```

Inheritance diagram for IncomeResourceProcessor:

Collaboration diagram for IncomeResourceProcessor:

Public Member Functions

- IncomeResourceProcessor (std::shared_ptr< IncomeResourceProduct > res, int maxStorage=200)

  Constructs an IncomeResourceProcessor with a given resource and optional storage limit.
- void process (int amount)

  Processes a specified amount of the resource, deducting it from storage.
- void store (int amount)

  Stores a specified amount of resource, increasing the current storage up to max capacity.
- void display () const

  Displays the current storage status and calls the resource's status display.
- int getCurrentStorage () const

  Returns the current amount of resource in storage.

Private Attributes

- std::shared_ptr< IncomeResourceProduct > resource

  Pointer to the resource being processed.
- int currentStorage

  Current amount of resource in storage.
- const int maxStorage

  Maximum storage capacity for the resource.

5.36.1 Detailed Description

Processes and manages storage for income-generating resources.

The IncomeResourceProcessor class handles storing and processing resources that generate income. It maintains a current storage level and a maximum storage capacity. The processed resources contribute their market value to the Government instance.

5.36.2 Constructor & Destructor Documentation

5.36.2.1 IncomeResourceProcessor()

```
IncomeResourceProcessor::IncomeResourceProcessor (
            std::shared_ptr< IncomeResourceProduct > res,
            int maxStorage = 200 )
```

Constructs an IncomeResourceProcessor with a given resource and optional storage limit.

Constructs an IncomeResourceProcessor with a given resource and storage limit.

Parameters

res	Shared pointer to an IncomeResourceProduct to be processed.
maxStorage	Maximum storage capacity for the resource (default is 200).
res	Shared pointer to an IncomeResourceProduct to be processed.
maxStorage	Maximum storage capacity for the resource.

5.36.3 Member Function Documentation

5.36.3.1 display()

```
void IncomeResourceProcessor::display ( ) const  [virtual]
```

Displays the current storage status and calls the resource's status display.

Displays the current storage status and resource status information.

This function outputs the current storage usage and invokes the resource's displayStatus() function.

This function outputs the current storage usage and calls the resource's displayStatus() function.

Implements ResourceProcessor.

5.36.3.2 getCurrentStorage()

```
int IncomeResourceProcessor::getCurrentStorage ( ) const  [virtual]
```

Returns the current amount of resource in storage.

Returns

int Current storage level of the resource.

Implements ResourceProcessor.

5.36.3.3 process()

```
void IncomeResourceProcessor::process (
             int amount )  [virtual]
```

Processes a specified amount of the resource, deducting it from storage.

Processes a specified amount of resource, deducting it from storage and adding its monetary value to the government.

If the specified amount exists in storage, it is processed, and its monetary value is added to the Government instance.

Parameters

amount	The amount of resource to process.

If the current storage has at least the specified amount, it is processed by deducting the amount from the storage and calculating the monetary value based on the resource's market value. The processed amount and monetary value are added to the Government instance.

Parameters

amount	The amount of resource to process.

Implements ResourceProcessor.

Here is the call graph for this function:

5.36.3.4 store()

```
void IncomeResourceProcessor::store (
             int amount )  [virtual]
```

Stores a specified amount of resource, increasing the current storage up to max capacity.

Stores a specified amount of resource, increasing the current storage up to the maximum limit.

If adding the specified amount does not exceed the max storage capacity, it is added to the current storage level.

Parameters

amount	The amount of resource to store.

If adding the specified amount does not exceed the max storage capacity, it is added to the current storage. Otherwise, a warning is issued that the storage capacity has been exceeded.

Parameters

amount	The amount of resource to store.

Implements ResourceProcessor.

5.36.4 Member Data Documentation

5.36.4.1 currentStorage

```
int IncomeResourceProcessor::currentStorage  [private]
```

Current amount of resource in storage.

5.36.4.2 maxStorage

```
const int IncomeResourceProcessor::maxStorage [private]
```

Maximum storage capacity for the resource.

5.36.4.3 resource

```
std::shared_ptr<IncomeResourceProduct> IncomeResourceProcessor::resource [private]
```

Pointer to the resource being processed.

The documentation for this class was generated from the following files:

- COS214-Poject/src/IncomeResourceProcessor.h
- COS214-Poject/src/IncomeResourceProcessor.cpp

5.37 IncomeResourceProduct Class Reference

Represents an income-generating resource in the city.

```
#include <IncomeResourceProduct.h>
```

Inheritance diagram for IncomeResourceProduct:

Collaboration diagram for IncomeResourceProduct:

Public Member Functions

- IncomeResourceProduct (string name, int quantity, double marketValue)
  Constructs an IncomeResourceProduct with a specified name, quantity, and market value.
- ∼IncomeResourceProduct ()
  Destructor for IncomeResourceProduct.
- void consumeResources (int amount)
  Consumes a specified amount of the resource, reducing the available quantity.
- double calculateIncome () const
  Calculates the total income generated by the resource.
- double getMarketValue () const
  Gets the market value of the resource.
- void displayStatus () const
  Displays detailed information about the resource.
- int getQuantity () const
  Gets the current quantity of the resource.
- bool isReadyForCollection ()
  Checks if the resource is ready for collection based on a predefined threshold.
- void replenish (int amount)
  Adds a specified quantity to the resource.

- void accept (taxCollector ∗TC)

  Accepts a tax collector visitor.
- std::string getBuildingType ()

  Gets the type of building associated with this component.
- std::string getName () const

  Gets the name of the resource.
- void updateReadyStatus ()

  Updates the collection readiness status based on quantity.
- void setLocation (const Location &loc)

  Sets the location of the resource.
- Location getLocation () const

  Gets the location of the resource.

Private Attributes

- std::string name
- int quantity
- double marketValue
- bool readyForCollection
- Location location

5.37.1 Detailed Description

Represents an income-generating resource in the city.

The IncomeResourceProduct class defines an individual resource that can generate income when processed. Each resource has a name, quantity, market value, and a status indicating if it's ready for collection.

5.37.2 Constructor & Destructor Documentation

5.37.2.1 IncomeResourceProduct()

```
IncomeResourceProduct::IncomeResourceProduct (
             string name,
             int quantity,
             double marketValue )
```

Constructs an IncomeResourceProduct with a specified name, quantity, and market value.

Parameters

name	Name of the resource.
quantity	Initial quantity of the resource.
marketValue	Market value per unit of the resource.

5.37.2.2 ∼IncomeResourceProduct()

```
IncomeResourceProduct::∼IncomeResourceProduct ( )
```

Destructor for IncomeResourceProduct.

This destructor can handle any necessary cleanup upon object destruction.

5.37.3 Member Function Documentation

5.37.3.1 accept()

```
void IncomeResourceProduct::accept (
            taxCollector * TC )  [inline], [virtual]
```

Accepts a tax collector visitor.

Parameters

TC	Pointer to a taxCollector visitor.

Implements CityComponent.

5.37.3.2 calculateIncome()

```
double IncomeResourceProduct::calculateIncome ( ) const
```

Calculates the total income generated by the resource.

Calculates the total income generated by the current quantity of the resource.

Returns

double Total income generated by the resource.

double Total income generated by the resource (quantity ∗ market value).

Here is the caller graph for this function:

5.37.3.3 consumeResources()

```
void IncomeResourceProduct::consumeResources (
            int amount )
```

Consumes a specified amount of the resource, reducing the available quantity.

Parameters

amount	Amount of the resource to consume.

If the specified amount is available in quantity, it is deducted. Otherwise, an error message is displayed. The ready status is updated accordingly.

Parameters

amount	The amount of the resource to consume.

Here is the call graph for this function:

5.37.3.4 displayStatus()

```
void IncomeResourceProduct::displayStatus ( ) const
```

Displays detailed information about the resource.

This includes the name, quantity, market value, and total value of the resource. Here is the call graph for this function:

5.37.3.5 getBuildingType()

```
std::string IncomeResourceProduct::getBuildingType ( )  [inline]
```

Gets the type of building associated with this component.

Returns

std::string Building type as "IncomeResourceProduct".

5.37.3.6 getLocation()

```
Location IncomeResourceProduct::getLocation ( ) const  [inline]
```

Gets the location of the resource.

Returns

Location Current location of the resource.

5.37.3.7 getMarketValue()

```
double IncomeResourceProduct::getMarketValue ( ) const
```

Gets the market value of the resource.

Returns

double Market value per unit of the resource.

Here is the caller graph for this function:

5.37.3.8 getName()

```
std::string IncomeResourceProduct::getName ( ) const  [inline]
```

Gets the name of the resource.

Returns

std::string Name of the resource.

5.37.3.9 getQuantity()

```
int IncomeResourceProduct::getQuantity ( ) const
```

Gets the current quantity of the resource.

Returns

int Current quantity of the resource.

int Current quantity available.

Here is the caller graph for this function:

5.37.3.10 isReadyForCollection()

```
bool IncomeResourceProduct::isReadyForCollection ( )
```

Checks if the resource is ready for collection based on a predefined threshold.

Checks if the resource is ready for collection based on a threshold.

Returns

bool True if the resource is ready for collection, false otherwise.

The resource is considered ready for collection if the quantity is 150 or more.

Returns

bool True if the resource is ready for collection, false otherwise.

5.37.3.11 replenish()

```
void IncomeResourceProduct::replenish (
            int amount )
```

Adds a specified quantity to the resource.

Adds a specified quantity to the resource, increasing its total amount.

Parameters

amount	Amount of the resource to add.

The ready status is updated after replenishment.

Parameters

amount	Amount of the resource to add.

Here is the call graph for this function:

5.37.3.12 setLocation()

```
void IncomeResourceProduct::setLocation (
             const Location & loc )   [inline]
```

Sets the location of the resource.

Parameters

loc	Location object to set the resource's position.

5.37.3.13 updateReadyStatus()

```
void IncomeResourceProduct::updateReadyStatus ( )  [inline]
```

Updates the collection readiness status based on quantity.

Here is the caller graph for this function:

5.37.4 Member Data Documentation

5.37.4.1 location

```
Location IncomeResourceProduct::location  [private]
```

Location of the resource

5.37.4.2 marketValue

```
double IncomeResourceProduct::marketValue  [private]
```

Market value per unit of the resource

5.37.4.3 name

```
std::string IncomeResourceProduct::name  [private]
```

Name of the resource

5.37.4.4 quantity

```
int IncomeResourceProduct::quantity  [private]
```

Current quantity of the resource

5.37.4.5 readyForCollection

```
bool IncomeResourceProduct::readyForCollection  [private]
```

Status indicating if the resource is ready for collection

The documentation for this class was generated from the following files:

- COS214-Poject/src/IncomeResourceProduct.h
- COS214-Poject/src/IncomeResourceProduct.cpp

5.38 IndustrialFactory Class Reference

Abstract factory class for creating industrial facilities.

```
#include <IndustrialFactory.h>
```

Collaboration diagram for IndustrialFactory:

Public Member Functions

- virtual ~IndustrialFactory ()
    Virtual destructor for IndustrialFactory.

5.38.1 Detailed Description

Abstract factory class for creating industrial facilities.

The IndustrialFactory class serves as a base class for factories that create different types of industrial facilities in the city. It provides an interface for creating industrial instances, supporting extensions for various industrial types and customization.

5.38.2 Constructor & Destructor Documentation

5.38.2.1 ∼IndustrialFactory()

```
virtual IndustrialFactory::∼IndustrialFactory ( )  [inline], [virtual]
```

Virtual destructor for IndustrialFactory.

Ensures proper cleanup for derived classes when deleting through a pointer to IndustrialFactory.

The documentation for this class was generated from the following file:

- COS214-Poject/src/IndustrialFactory.h

5.39 Industry Class Reference

Represents an industrial facility in the city.

`#include <Industry.h>`

Inheritance diagram for Industry:

Collaboration diagram for Industry:

Public Member Functions

- Industry (const std::string &name, std::shared_ptr< ResourceProcessor > primary, std::shared_ptr< ResourceProcessor > secondary, MapGrid ∗grid, std::map< std::string, int > &collectedResources, int range=1)

  Constructs an Industry object with the specified parameters.
- bool isResourceInRange (const Location &industryLoc, const Location &resourceLoc) const

  Checks if a resource is within the collection range.
- int getCollectionRange () const

  Gets the collection range of the industry.
- Location findIndustryLocation () const

  Finds the location of the industry on the map.
- void showCollectionRange (const Location &loc) const

  Displays the collection range on the map.
- void processResources (int amount, bool isIncomeResource)

  Processes resources based on the specified amount.
- virtual void storeResources (int amount, bool toPrimary=true)

  Stores resources in the industry.
- virtual void displayStatus ()

  Displays the status of the industry, including pollution and resources.
- std::string getBuildingType () const override

  Gets the type of building as a string.
- virtual void accept (taxCollector ∗TC)

  Accepts a tax collector visitor.

Protected Member Functions

- void increasePollution (int amount)

  Increases pollution level by a specified amount.
- virtual ∼Industry ()=default

  Virtual destructor for Industry to allow proper cleanup of derived classes.

Protected Attributes

- std::string name

  Name of the industry.
- std::shared_ptr< ResourceProcessor > primaryProcessor

  Processor for primary resource.
- std::shared_ptr< ResourceProcessor > secondaryProcessor

  Processor for secondary resource.
- int pollutionLevel

  Pollution level caused by the industry.
- int collectionRange

  Range in which the industry can collect resources.
- MapGrid ∗ grid

  Pointer to the map grid for location purposes.
- std::shared_ptr< IncomeResourceProduct > incomeResource

  Income resource for the industry.
- std::shared_ptr< ConstructionResourceProduct > constructionResource

  Construction resource for the industry.
- std::map< std::string, int > & collectedResources

  Reference to collected resources map.

5.39.1 Detailed Description

Represents an industrial facility in the city.

The Industry class inherits from CityComponent and is responsible for processing resources, managing pollution, and interacting with resources in the vicinity. It supports income and construction resources and integrates with a city map grid.

5.39.2 Constructor & Destructor Documentation

5.39.2.1 Industry()

```
Industry::Industry (
            const std::string & name,
            std::shared_ptr< ResourceProcessor > primary,
            std::shared_ptr< ResourceProcessor > secondary,
            MapGrid * grid,
            std::map< std::string, int > & collectedResources,
            int range = 1 )
```

Constructs an Industry object with the specified parameters.

Constructs an Industry with specified parameters.

Parameters

name	Name of the industry
primary	Primary resource processor
secondary	Secondary resource processor
grid	Pointer to the map grid
collectedResources	Map of collected resources
range	Collection range of the industry
name	Name of the industry
primary	Primary resource processor
secondary	Secondary resource processor
grid	Pointer to the map grid
collectedResources	Map for tracking collected resources
range	Collection range for the industry

5.39.2.2 ∼Industry()

```
virtual Industry::∼Industry ( )  [protected], [virtual], [default]
```

Virtual destructor for Industry to allow proper cleanup of derived classes.

5.39.3 Member Function Documentation

5.39.3.1 accept()

```
void Industry::accept (
            taxCollector * TC )  [virtual]
```

Accepts a tax collector visitor.

Parameters

TC	Pointer to the tax collector visitor

Implements CityComponent.

5.39.3.2 displayStatus()

```
void Industry::displayStatus ( )  [virtual]
```

Displays the status of the industry, including pollution and resources.

Displays the status of the industry, including storage and pollution level.

Implements CityComponent.

5.39.3.3 findIndustryLocation()

```
Location Industry::findIndustryLocation ( ) const
```

Finds the location of the industry on the map.

Returns

Location of the industry
Location of the industry or (-1, -1) if not found

Here is the call graph for this function:

5.39.3.4 getBuildingType()

```
std::string Industry::getBuildingType ( ) const  [override], [virtual]
```

Gets the type of building as a string.

Gets the building type of the industry.

Returns

Type of building as a string
Building type as a string

Implements CityComponent.

5.39.3.5 getCollectionRange()

```
int Industry::getCollectionRange ( ) const
```

Gets the collection range of the industry.

Returns

Collection range as an integer

5.39.3.6 increasePollution()

```
void Industry::increasePollution (
             int amount )  [protected]
```

Increases pollution level by a specified amount.

Increases the pollution level of the industry.

Parameters

amount	Amount to increase pollution by

If pollution exceeds a certain threshold, a warning is displayed.

Parameters

amount	Amount to increase pollution by

5.39.3.7 isResourceInRange()

```
bool Industry::isResourceInRange (
            const Location & industryLoc,
            const Location & resourceLoc ) const
```

Checks if a resource is within the collection range.

Parameters

industryLoc	Location of the industry
resourceLoc	Location of the resource

Returns

True if the resource is within range, false otherwise

5.39.3.8 processResources()

```
void Industry::processResources (
            int amount,
            bool isIncomeResource )
```

Processes resources based on the specified amount.

Processes a specified amount of resources.

Parameters

amount	Amount of resource to process
isIncomeResource	True if processing income resources, false for construction resources

If resources are depleted, they are removed from the grid.

Parameters

amount	Amount of resource to process
isIncomeResource	True for income resources, false for construction resources

Here is the call graph for this function: Here is the caller graph for this function:

5.39.3.9 showCollectionRange()

```
void Industry::showCollectionRange (
             const Location & loc ) const
```

Displays the collection range on the map.

Displays the collection range of the industry on the map.

Parameters

loc	Location to display the range from

5.39.3.10 storeResources()

```
void Industry::storeResources (
             int amount,
             bool toPrimary = true )  [virtual]
```

Stores resources in the industry.

Stores a specified amount of resources in the specified processor.

Parameters

amount	Amount of resource to store
toPrimary	True to store in primary processor, false for secondary processor
amount	Amount of resource to store
toPrimary	True to store in the primary processor, false for secondary

5.39.4 Member Data Documentation

5.39.4.1 collectedResources

```
std::map<std::string, int>& Industry::collectedResources  [protected]
```

Reference to collected resources map.

5.39.4.2 collectionRange

```
int Industry::collectionRange [protected]
```

Range in which the industry can collect resources.

5.39.4.3 constructionResource

```
std::shared_ptr<ConstructionResourceProduct> Industry::constructionResource [protected]
```

Construction resource for the industry.

5.39.4.4 grid

```
MapGrid* Industry::grid [protected]
```

Pointer to the map grid for location purposes.

5.39.4.5 incomeResource

```
std::shared_ptr<IncomeResourceProduct> Industry::incomeResource [protected]
```

Income resource for the industry.

5.39.4.6 name

```
std::string Industry::name [protected]
```

Name of the industry.

5.39.4.7 pollutionLevel

```
int Industry::pollutionLevel [protected]
```

Pollution level caused by the industry.

5.39.4.8 primaryProcessor

```
std::shared_ptr<ResourceProcessor> Industry::primaryProcessor  [protected]
```

Processor for primary resource.

5.39.4.9 secondaryProcessor

```
std::shared_ptr<ResourceProcessor> Industry::secondaryProcessor  [protected]
```

Processor for secondary resource.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Industry.h
- COS214-Poject/src/Industry.cpp

5.40 LandMark Class Reference

Represents a landmark within the city.

```
#include <LandMark.h>
```

Inheritance diagram for LandMark:

Collaboration diagram for LandMark:

Public Member Functions

- LandMark ()=default
- LandMark (const std::string &type, int capacity, double price, std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Constructs a LandMark with specified parameters.
- virtual void displayStatus () override

  Displays the status of the landmark.
- double getPrice () const

  Gets the price of the landmark.
- int getUtilityCoverage () const

  Gets the number of utilities connected to the landmark.
- bool hasUtilities () const

  Checks if all utilities are connected.
- std::string getBuildingType () const override

  Gets the building type of the landmark.
- void accept (taxCollector *TC)

  Accepts a tax collector.

Protected Attributes

- std::string landMarkType

  The type of landmark.
- int visitorCapacity

  The maximum number of visitors the landmark can accommodate.
- double price

  The price associated with the landmark.
- std::shared_ptr< UtilityFlyweight > waterSupply

  Water utility connection.
- std::shared_ptr< UtilityFlyweight > powerSupply

  Power utility connection.
- std::shared_ptr< UtilityFlyweight > wasteManagement

  Waste management utility connection.
- std::shared_ptr< UtilityFlyweight > sewageManagement

  Sewage management utility connection.

5.40.1 Detailed Description

Represents a landmark within the city.

This class inherits from CityComponent and provides functionality specific to landmarks, including utility connections and visitor capacity.

5.40.2 Constructor & Destructor Documentation

5.40.2.1 LandMark() [1/2]

```
LandMark::LandMark ( )  [default]
```

5.40.2.2 LandMark() [2/2]

```
LandMark::LandMark (
             const std::string & type,
             int capacity,
             double price,
             std::shared_ptr< UtilityFlyweight > water,
             std::shared_ptr< UtilityFlyweight > power,
             std::shared_ptr< UtilityFlyweight > waste,
             std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a LandMark with specified parameters.

Parameters

type	The type of the landmark.
capacity	The visitor capacity of the landmark.
price	The price of the landmark.
water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage management utility.
type	The type of the landmark.
capacity	The visitor capacity of the landmark.
price	The price associated with the landmark.
water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage management utility.

5.40.3 Member Function Documentation

5.40.3.1 accept()

```
void LandMark::accept (
            taxCollector * TC )  [inline], [virtual]
```

Accepts a tax collector.

This method can be overridden by derived classes to implement specific tax collection behavior.

Parameters

TC	Pointer to the tax collector.

Implements CityComponent.

5.40.3.2 displayStatus()

```
void LandMark::displayStatus ( )  [override], [virtual]
```

Displays the status of the landmark.

This method overrides the displayStatus method in CityComponent to provide specific details for the landmark.

This method provides details about the landmark, including its type, visitor capacity, price, and the status of each connected utility.

Implements CityComponent.

Reimplemented in Park, Monument, and CulturalCenter.

Here is the caller graph for this function:

5.40.3.3 getBuildingType()

```
std::string LandMark::getBuildingType ( ) const  [inline], [override], [virtual]
```

Gets the building type of the landmark.

Returns

A string representing the type of building.

Implements CityComponent.

Reimplemented in Park, and Monument.

5.40.3.4 getPrice()

```
double LandMark::getPrice ( ) const  [inline]
```

Gets the price of the landmark.

Returns

The price of the landmark.

5.40.3.5 getUtilityCoverage()

```
int LandMark::getUtilityCoverage ( ) const  [inline]
```

Gets the number of utilities connected to the landmark.

Returns

The count of utility connections.

5.40.3.6 hasUtilities()

```
bool LandMark::hasUtilities ( ) const  [inline]
```

Checks if all utilities are connected.

Returns

True if all utilities are connected, false otherwise.

5.40.4 Member Data Documentation

5.40.4.1 landMarkType

```
std::string LandMark::landMarkType  [protected]
```

The type of landmark.

5.40.4.2 powerSupply

```
std::shared_ptr<UtilityFlyweight> LandMark::powerSupply  [protected]
```

Power utility connection.

5.40.4.3 price

```
double LandMark::price  [protected]
```

The price associated with the landmark.

5.40.4.4 sewageManagement

```
std::shared_ptr<UtilityFlyweight> LandMark::sewageManagement  [protected]
```

Sewage management utility connection.

5.40.4.5 visitorCapacity

`int LandMark::visitorCapacity [protected]`

The maximum number of visitors the landmark can accommodate.

5.40.4.6 wasteManagement

`std::shared_ptr<UtilityFlyweight> LandMark::wasteManagement [protected]`

Waste management utility connection.

5.40.4.7 waterSupply

`std::shared_ptr<UtilityFlyweight> LandMark::waterSupply [protected]`

Water utility connection.

The documentation for this class was generated from the following files:

- COS214-Poject/src/LandMark.h
- COS214-Poject/src/LandMark.cpp

5.41 LandMarkFactory Class Reference

Abstract factory class for creating landmark objects.

`#include <LandMarkFactory.h>`

Inheritance diagram for LandMarkFactory:

Collaboration diagram for LandMarkFactory:

Public Member Functions

- virtual std::unique_ptr< LandMark > createLandMark ()=0

    Pure virtual function to create a landmark.
- virtual ~LandMarkFactory ()

    Virtual destructor for the `LandMarkFactory` class.

5.41.1 Detailed Description

Abstract factory class for creating landmark objects.

The `LandMarkFactory` class provides an interface for creating different types of landmarks in the city simulation. Specific landmark types should be created by concrete factories that inherit from this interface.

5.41.2 Constructor & Destructor Documentation

5.41.2.1 ∼LandMarkFactory()

```
virtual LandMarkFactory::∼LandMarkFactory ( )  [inline], [virtual]
```

Virtual destructor for the `LandMarkFactory` class.

Ensures proper cleanup of derived classes.

5.41.3 Member Function Documentation

5.41.3.1 createLandMark()

```
virtual std::unique_ptr<LandMark> LandMarkFactory::createLandMark ( )  [pure virtual]
```

Pure virtual function to create a landmark.

This function should be implemented by derived classes to create specific types of landmarks.

Returns

A unique pointer to the created `LandMark` object.

Implemented in MonumentFactory.

The documentation for this class was generated from the following file:

- COS214-Poject/src/LandMarkFactory.h

5.42 Location Struct Reference

Represents a 2D coordinate location on a grid.

`#include <Location.h>`

Collaboration diagram for Location:

Public Member Functions

- Location ()

  Default constructor initializing location to (0, 0).
- Location (int xCoord, int yCoord)

  Parameterized constructor to initialize location with specific coordinates.
- bool operator== (const Location &other) const

  Equality operator to compare two locations.
- bool operator!= (const Location &other) const

  Inequality operator to compare two locations.
- bool operator< (const Location &other) const

  Less-than operator to order locations, primarily for use in sorted containers.
- double distanceTo (const Location &other) const

  Calculates the Euclidean distance to another location.
- bool isValid () const

  Checks if the location is valid (non-negative coordinates).

Public Attributes

- int x

  X-coordinate of the location.
- int y

  Y-coordinate of the location.

5.42.1 Detailed Description

Represents a 2D coordinate location on a grid.

The `Location` struct stores x and y coordinates, providing utilities to compare, validate, and calculate the distance between locations.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Location() [1/2]

```
Location::Location ( )  [inline]
```

Default constructor initializing location to (0, 0).

5.42.2.2 Location() [2/2]

```
Location::Location (
              int xCoord,
              int yCoord )  [inline]
```

Parameterized constructor to initialize location with specific coordinates.

Parameters

xCoord	The x-coordinate.
yCoord	The y-coordinate.

5.42.3 Member Function Documentation

5.42.3.1 distanceTo()

```
double Location::distanceTo (
            const Location & other ) const  [inline]
```

Calculates the Euclidean distance to another location.

Parameters

other	The other location.

Returns

The distance to the other location.

5.42.3.2 isValid()

```
bool Location::isValid ( ) const  [inline]
```

Checks if the location is valid (non-negative coordinates).

Returns

True if the location has non-negative x and y values.

5.42.3.3 operator"!=()

```
bool Location::operator!= (
            const Location & other ) const  [inline]
```

Inequality operator to compare two locations.

Parameters

other	Another location to compare with.

Returns

True if the locations have different coordinates.

5.42.3.4 operator<()

```
bool Location::operator< (
             const Location & other ) const  [inline]
```

Less-than operator to order locations, primarily for use in sorted containers.

Parameters

other	Another location to compare with.

Returns

True if this location is less than the other based on x and y coordinates.

5.42.3.5 operator==()

```
bool Location::operator== (
             const Location & other ) const  [inline]
```

Equality operator to compare two locations.

Parameters

other	Another location to compare with.

Returns

True if the locations have the same coordinates.

5.42.4 Member Data Documentation

5.42.4.1 x

```
int Location::x
```

X-coordinate of the location.

5.42.4.2 y

```
int Location::y
```

Y-coordinate of the location.

The documentation for this struct was generated from the following file:

- COS214-Poject/src/Location.h

5.43 Malls Class Reference

Represents a Mall, a type of CommercialBuilding.

```
#include <Malls.h>
```

Inheritance diagram for Malls:

Collaboration diagram for Malls:

Public Member Functions

- Malls ()

  Default constructor for Malls.
- Malls (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_↩ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Constructs a Mall with specified utility connections.
- char getDisplaySymbol () const

  Retrieves the display symbol for a Mall.
- std::string getBuildingType () const override

  Retrieves the building type of the Mall.

Additional Inherited Members

5.43.1 Detailed Description

Represents a Mall, a type of CommercialBuilding.

Malls are initialized with a base area and optional utility connections for water, power, waste management, and sewage.

5.43.2 Constructor & Destructor Documentation

5.43.2.1 Malls() [1/2]

```
Malls::Malls ( )
```

Default constructor for Malls.

Initializes a Mall with a base area and no utility connections.

Initializes a Mall with a base area of 400.0 and no utility connections.

5.43.2.2 Malls() [2/2]

```
Malls::Malls (
             std::shared_ptr< UtilityFlyweight > water,
             std::shared_ptr< UtilityFlyweight > power,
             std::shared_ptr< UtilityFlyweight > waste,
             std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a Mall with specified utility connections.

Parameters

water	Shared pointer to water utility.
power	Shared pointer to power utility.
waste	Shared pointer to waste management utility.
sewage	Shared pointer to sewage management utility.

5.43.3 Member Function Documentation

5.43.3.1 getBuildingType()

```
std::string Malls::getBuildingType ( ) const  [inline], [override], [virtual]
```

Retrieves the building type of the Mall.

Returns

A string "Mall" representing the type of the building.

Reimplemented from CommercialBuilding.

5.43.3.2 getDisplaySymbol()

```
char Malls::getDisplaySymbol ( ) const  [inline], [virtual]
```

Retrieves the display symbol for a Mall.

Returns

The character 'M' representing the Mall on a map.

Implements CommercialBuilding.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Malls.h
- COS214-Poject/src/Malls.cpp

5.44 MallsFactory Class Reference

Factory class for creating Mall buildings.

```
#include <MallsFactory.h>
```

Inheritance diagram for MallsFactory:

Collaboration diagram for MallsFactory:

Public Member Functions

- std::unique_ptr< CommercialBuilding > createBuilding ()

  Creates an instance of a Mall.

5.44.1 Detailed Description

Factory class for creating Mall buildings.

This class implements the factory pattern to create instances of `Mall`, a specific type of `CommercialBuilding`.

5.44.2 Member Function Documentation

5.44.2.1 createBuilding()

```
std::unique_ptr< CommercialBuilding > MallsFactory::createBuilding ( )  [virtual]
```

Creates an instance of a Mall.

Creates a new instance of a Mall.

Overrides the `createBuilding` function to provide a specific implementation for creating a `Mall` object.

Returns

A unique pointer to a `CommercialBuilding` instance representing a `Mall`.

This function overrides the `createBuilding` method from `CommercialBuildingFactory` to provide a specific implementation for creating a `Mall`.

Returns

A unique pointer to a `CommercialBuilding` instance representing a `Mall`.

Implements CommercialFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/MallsFactory.h
- COS214-Poject/src/MallsFactory.cpp

5.45 MapGrid Class Reference

```
#include <MapGrid.h>
```

Collaboration diagram for MapGrid:

Classes

- struct Cell
- struct PlacementResult
- struct ResourceSpot
- struct ZoneStyle

Public Member Functions

- Cell & getCell (int x, int y)
- void setCellSymbol (int x, int y, char symbol)
- void setTransportEndpoint (const Location &loc, char type)
- void setTransportPath (const Location &loc, char pathSymbol)
- Location getRandomEmptyLocation () const
- int getHeight () const
- int getWidth () const
- std::vector< std::shared_ptr< ZoneComposite > > getAllZones () const
- std::vector< std::shared_ptr< CityComponent > > getBuildingsInZone (const std::shared_ptr< ZoneComposite > &zone) const
- PlacementResult canPlaceZone (const Location &topLeft, const Location &bottomRight) const
- PlacementResult canPlaceBuilding (const Location &loc, const std::shared_ptr< CityComponent > &building) const
- MapGrid (int w, int h)
- bool createZone (const Location &topLeft, const Location &bottomRight, const std::string &zoneType)
- void setZone (const Location &loc, std::shared_ptr< ZoneComposite > newZone)
- std::shared_ptr< ZoneComposite > getZone (const Location &loc) const
- bool isValidLocation (const Location &loc) const
- bool placeComponent (const Location &loc, std::shared_ptr< CityComponent > component)
- bool connectLocations (const Location &start, const Location &end, std::shared_ptr< Transport > transport)
- bool placePublicService (const Location &loc, std::shared_ptr< PublicService > publicService)
- std::shared_ptr< CityComponent > getComponent (const Location &loc) const
- void addUtilityEffect (const Location &loc, std::shared_ptr< UtilityFlyweight > utility)
- void displayCollectionRange (const Location &industryLoc)
- void removeUtilityEffect (const Location &loc, std::shared_ptr< UtilityFlyweight > utility)
- bool isValidZoneForBuilding (const Location &loc, const std::shared_ptr< CityComponent > &building) const
- std::string getDisplayString () const
- void removeComponent (const Location &loc)
- void removeIncomeResource (const Location &loc)
- void removeConstructionResource (const Location &loc)

Private Member Functions

- bool isOverlappingZone (const Location &topLeft, const Location &bottomRight) const
- bool isCompatibleBuilding (const std::shared_ptr< CityComponent > &building, const std::shared_ptr< ZoneComposite > &zone) const
- void initializeZoneStyles ()
- bool isZoneBoundary (int x, int y) const

Private Attributes

- std::map< Location, std::shared_ptr< CityComponent > > gridMap
- std::map< Location, std::shared_ptr< IncomeResourceProduct > > incomeResourceMap
- std::map< Location, std::shared_ptr< ConstructionResourceProduct > > constructionResourceMap
- std::vector< std::vector< Cell > > grid
- int width
- int height
- std::map< std::string, ZoneStyle > zoneStyles

5.45.1 Constructor & Destructor Documentation

5.45.1.1 MapGrid()

```
MapGrid::MapGrid (
            int w,
            int h )  [inline]
```

Here is the call graph for this function:

5.45.2 Member Function Documentation

5.45.2.1 addUtilityEffect()

```
void MapGrid::addUtilityEffect (
            const Location & loc,
            std::shared_ptr< UtilityFlyweight > utility )
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.2 canPlaceBuilding()

```
PlacementResult MapGrid::canPlaceBuilding (
            const Location & loc,
            const std::shared_ptr< CityComponent > & building ) const  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.3 canPlaceZone()

```
PlacementResult MapGrid::canPlaceZone (
            const Location & topLeft,
            const Location & bottomRight ) const  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.4 connectLocations()

```
bool MapGrid::connectLocations (
            const Location & start,
            const Location & end,
            std::shared_ptr< Transport > transport )  [inline]
```

Here is the call graph for this function:

5.45.2.5 createZone()

```
bool MapGrid::createZone (
             const Location & topLeft,
             const Location & bottomRight,
             const std::string & zoneType )  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.6 displayCollectionRange()

```
void MapGrid::displayCollectionRange (
             const Location & industryLoc )
```

Here is the call graph for this function:

5.45.2.7 getAllZones()

```
std::vector<std::shared_ptr<ZoneComposite> > MapGrid::getAllZones ( ) const  [inline]
```

5.45.2.8 getBuildingsInZone()

```
std::vector< std::shared_ptr< CityComponent > > MapGrid::getBuildingsInZone (
             const std::shared_ptr< ZoneComposite > & zone ) const
```

5.45.2.9 getCell()

```
Cell& MapGrid::getCell (
             int x,
             int y )  [inline]
```

5.45.2.10 getComponent()

```
std::shared_ptr<CityComponent> MapGrid::getComponent (
             const Location & loc ) const  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.11 getDisplayString()

```
std::string MapGrid::getDisplayString ( ) const
```

Here is the caller graph for this function:

5.45.2.12 getHeight()

```
int MapGrid::getHeight ( ) const  [inline]
```

Here is the caller graph for this function:

5.45.2.13 getRandomEmptyLocation()

```
Location MapGrid::getRandomEmptyLocation ( ) const  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.14 getWidth()

```
int MapGrid::getWidth ( ) const  [inline]
```

Here is the caller graph for this function:

5.45.2.15 getZone()

```
std::shared_ptr<ZoneComposite> MapGrid::getZone (
             const Location & loc ) const  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.16 initializeZoneStyles()

```
void MapGrid::initializeZoneStyles ( )  [inline], [private]
```

Here is the caller graph for this function:

5.45.2.17 isCompatibleBuilding()

```
bool MapGrid::isCompatibleBuilding (
             const std::shared_ptr< CityComponent > & building,
             const std::shared_ptr< ZoneComposite > & zone ) const  [inline], [private]
```

Here is the caller graph for this function:

5.45.2.18 isOverlappingZone()

```
bool MapGrid::isOverlappingZone (
             const Location & topLeft,
             const Location & bottomRight ) const  [inline], [private]
```

Here is the caller graph for this function:

5.45.2.19 isValidLocation()

```
bool MapGrid::isValidLocation (
            const Location & loc ) const  [inline]
```

Here is the caller graph for this function:

5.45.2.20 isValidZoneForBuilding()

```
bool MapGrid::isValidZoneForBuilding (
            const Location & loc,
            const std::shared_ptr< CityComponent > & building ) const  [inline]
```

Here is the call graph for this function:

5.45.2.21 isZoneBoundary()

```
bool MapGrid::isZoneBoundary (
            int x,
            int y ) const  [inline], [private]
```

Here is the call graph for this function:

5.45.2.22 placeComponent()

```
bool MapGrid::placeComponent (
            const Location & loc,
            std::shared_ptr< CityComponent > component )
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.23 placePublicService()

```
bool MapGrid::placePublicService (
            const Location & loc,
            std::shared_ptr< PublicService > publicService )  [inline]
```

Here is the call graph for this function:

5.45.2.24 removeComponent()

```
void MapGrid::removeComponent (
            const Location & loc )
```

Here is the caller graph for this function:

5.45.2.25 removeConstructionResource()

```
void MapGrid::removeConstructionResource (
             const Location & loc )
```

Here is the caller graph for this function:

5.45.2.26 removeIncomeResource()

```
void MapGrid::removeIncomeResource (
             const Location & loc )
```

Here is the caller graph for this function:

5.45.2.27 removeUtilityEffect()

```
void MapGrid::removeUtilityEffect (
             const Location & loc,
             std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

Here is the call graph for this function:

5.45.2.28 setCellSymbol()

```
void MapGrid::setCellSymbol (
             int x,
             int y,
             char symbol )  [inline]
```

Here is the caller graph for this function:

5.45.2.29 setTransportEndpoint()

```
void MapGrid::setTransportEndpoint (
             const Location & loc,
             char type )  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.30 setTransportPath()

```
void MapGrid::setTransportPath (
             const Location & loc,
             char pathSymbol )  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.2.31 setZone()

```
void MapGrid::setZone (
            const Location & loc,
            std::shared_ptr< ZoneComposite > newZone )  [inline]
```

Here is the call graph for this function: Here is the caller graph for this function:

5.45.3 Member Data Documentation

5.45.3.1 constructionResourceMap

```
std::map<Location, std::shared_ptr<ConstructionResourceProduct> > MapGrid::construction←
ResourceMap  [private]
```

5.45.3.2 grid

```
std::vector<std::vector<Cell> > MapGrid::grid  [private]
```

5.45.3.3 gridMap

```
std::map<Location, std::shared_ptr<CityComponent> > MapGrid::gridMap  [private]
```

5.45.3.4 height

```
int MapGrid::height  [private]
```

5.45.3.5 incomeResourceMap

```
std::map<Location, std::shared_ptr<IncomeResourceProduct> > MapGrid::incomeResourceMap  [private]
```

5.45.3.6 width

```
int MapGrid::width  [private]
```

5.45.3.7 zoneStyles

```
std::map<std::string, ZoneStyle> MapGrid::zoneStyles  [private]
```

The documentation for this class was generated from the following files:

- COS214-Poject/src/MapGrid.h
- COS214-Poject/src/MapGrid.cpp

5.46 MetalFactory Class Reference

A factory class for creating metal-related resources for income generation and construction.

```
#include <MetalFactory.h>
```

Inheritance diagram for MetalFactory:

Collaboration diagram for MetalFactory:

Public Member Functions

- MetalFactory ()

  Constructor for MetalFactory.
- ∼MetalFactory ()

  Destructor for MetalFactory.
- std::unique_ptr< IncomeResourceProduct > createIncomeR (int quantity) override

  Creates an income-generating metal resource.
- std::unique_ptr< ConstructionResourceProduct > createConstructionR (int quantity) override

  Creates a construction metal resource.

5.46.1 Detailed Description

A factory class for creating metal-related resources for income generation and construction.

The MetalFactory is derived from ResourceFactory and specializes in creating resources related to metals, such as gold for income and steel for construction.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 MetalFactory()

```
MetalFactory::MetalFactory ( )
```

Constructor for MetalFactory.

Constructor for the MetalFactory.

Initializes a MetalFactory instance that can create metal-related resources.

Initializes a MetalFactory instance, used to produce metal-related resources for income generation and construction.

5.46.2.2 ∼MetalFactory()

```
MetalFactory::∼MetalFactory ( )
```

Destructor for MetalFactory.

Destructor for the MetalFactory.

Cleans up resources associated with the MetalFactory instance.

5.46.3 Member Function Documentation

5.46.3.1 createConstructionR()

```
std::unique_ptr< ConstructionResourceProduct > MetalFactory::createConstructionR (
            int quantity ) [override], [virtual]
```

Creates a construction metal resource.

Creates a construction resource.

Creates and returns a `Steel` object, a construction resource, with a specified quantity.

Parameters

quantity	The quantity of the construction resource to be produced.

Returns

A unique pointer to a `ConstructionResourceProduct` representing `Steel`.

This method creates and returns a `Steel` object, a construction resource.

Parameters

quantity	The quantity of the construction resource to be produced.

Returns

A unique pointer to a `ConstructionResourceProduct` representing `Steel`.

Implements ResourceFactory.

5.46.3.2 createIncomeR()

```
std::unique_ptr< IncomeResourceProduct > MetalFactory::createIncomeR (
            int quantity )  [override], [virtual]
```

Creates an income-generating metal resource.

Creates an income-generating resource.

Creates and returns a `Gold` object, an income-generating resource, with a specified quantity.

Parameters

quantity	The quantity of the income resource to be produced.

Returns

A unique pointer to an `IncomeResourceProduct` representing `Gold`.

This method creates and returns a `Gold` object, an income-generating resource.

Parameters

quantity	The quantity of the resource to be produced.

Returns

A unique pointer to an `IncomeResourceProduct` representing `Gold`.

Implements ResourceFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/MetalFactory.h
- COS214-Poject/src/MetalFactory.cpp

5.47 MetalWorkFacility Class Reference

A concrete implementation of an Industry for processing metal resources.

```
#include <MetalWorkFacility.h>
```

Inheritance diagram for MetalWorkFacility:

Collaboration diagram for MetalWorkFacility:

Public Member Functions

- MetalWorkFacility (std::shared_ptr< IncomeResourceProduct > gold, std::shared_ptr< ConstructionResourceProduct > steel, MapGrid *grid, std::map< std::string, int > &collectedResources)

  Constructs a MetalWorkFacility for processing gold and steel.
- void processGold (int amount)

  Processes a specified amount of gold.
- void processSteel (int amount)

  Processes a specified amount of steel.

Static Public Attributes

- static const int METAL_WORK_RANGE = 5

  The range within which the MetalWorkFacility can collect resources.

Additional Inherited Members

5.47.1 Detailed Description

A concrete implementation of an Industry for processing metal resources.

The MetalWorkFacility class handles the processing of income-generating and construction resources, specifically gold and steel. It maintains a collection range within which it can collect resources.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 MetalWorkFacility()

```
MetalWorkFacility::MetalWorkFacility (
            std::shared_ptr< IncomeResourceProduct > gold,
            std::shared_ptr< ConstructionResourceProduct > steel,
            MapGrid * grid,
            std::map< std::string, int > & collectedResources )
```

Constructs a MetalWorkFacility for processing gold and steel.

Constructs a MetalWorkFacility with specific resources and a processing range.

Initializes the facility to process income-generating gold resources and construction steel resources. It links to the map grid and a resource collection tracker.

Parameters

gold	A shared pointer to the income resource (gold) for processing.
steel	A shared pointer to the construction resource (steel) for processing.
grid	A pointer to the map grid where the facility is located.
collectedResources	A reference to a map tracking collected resource quantities.

Initializes the MetalWorkFacility as an `Industry` specializing in processing gold for income and steel for construction. Connects it to the `MapGrid` and tracks collected resources.

Parameters

gold	A shared pointer to the income-generating `IncomeResourceProduct` (Gold).
steel	A shared pointer to the construction `ConstructionResourceProduct` (Steel).
grid	Pointer to the `MapGrid` where the facility is located.
collectedResources	Reference to a map of collected resources for tracking processed items.

5.47.3 Member Function Documentation

5.47.3.1 processGold()

```
void MetalWorkFacility::processGold (
              int amount )
```

Processes a specified amount of gold.

Processes a specified amount of gold in the facility.

Decreases the stored quantity of gold by the specified amount and adds the processed amount to collected resources.

Parameters

amount	The amount of gold to process.

Initiates processing for a specified quantity of gold, reducing storage in the `IncomeResourceProcessor`.

Parameters

amount	The quantity of gold to process.

Here is the call graph for this function:

5.47.3.2 processSteel()

```
void MetalWorkFacility::processSteel (
            int amount )
```

Processes a specified amount of steel.

Processes a specified amount of steel in the facility.

Decreases the stored quantity of steel by the specified amount and adds the processed amount to collected resources.

Parameters

amount	The amount of steel to process.

Initiates processing for a specified quantity of steel, reducing storage in the `ConstructionResourceProcessor`.

Parameters

amount	The quantity of steel to process.

Here is the call graph for this function:

5.47.4 Member Data Documentation

5.47.4.1 METAL_WORK_RANGE

```
const int MetalWorkFacility::METAL_WORK_RANGE = 5  [static]
```

The range within which the MetalWorkFacility can collect resources.

The documentation for this class was generated from the following files:

- COS214-Poject/src/MetalWorkFacility.h
- COS214-Poject/src/MetalWorkFacility.cpp

5.48 ModerateCollectionStrategy Class Reference

Collection strategy for moderate collection rate.

`#include <ModerateCollectionStrategy.h>`

Inheritance diagram for ModerateCollectionStrategy:

Collaboration diagram for ModerateCollectionStrategy:

Public Member Functions

- int collect (int baseAmount)

  Collect resources at a moderate rate.
- int collect (int amount) const override

  Pure virtual function to collect resources based on a strategy.

5.48.1 Detailed Description

Collection strategy for moderate collection rate.

A strategy for collecting resources at a moderate rate.

Implements a 100% collection efficiency.

This class provides a method to collect resources at a normal collection rate. It is part of a strategy pattern where different collection strategies can be used based on the game's requirements.

5.48.2 Member Function Documentation

5.48.2.1 collect() `[1/2]`

```
int ModerateCollectionStrategy::collect (
            int baseAmount ) const  [inline], [override], [virtual]
```

Pure virtual function to collect resources based on a strategy.

Derived classes implement this method to adjust the amount collected based on a specific collection strategy.

Parameters

baseAmount	The base amount to be collected.

Returns

The adjusted amount collected based on the strategy.

< Returns the input amount unchanged.

Implements CollectionStrategy.

5.48.2.2 collect() [2/2]

```
int ModerateCollectionStrategy::collect (
            int baseAmount )
```

Collect resources at a moderate rate.

Collects resources at a moderate rate.

This method implements a standard collection rate by returning the base amount provided.

Parameters

baseAmount	The base amount of resources to collect.

Returns

The amount collected, which is equal to the base amount.

The ModerateCollectionStrategy implements a normal collection rate by simply returning the base amount specified.

Parameters

baseAmount	The base amount of resources to be collected.

Returns

The amount collected, which is equal to the base amount.

The documentation for this class was generated from the following files:

- COS214-Poject/src/ModerateCollectionStrategy.h
- COS214-Poject/src/NPCSystem.h
- COS214-Poject/src/ModerateCollectionStrategy.cpp

5.49 Monument Class Reference

Represents a Monument, a type of LandMark with specific characteristics and utilities.

`#include <Monument.h>`

Inheritance diagram for Monument:

Collaboration diagram for Monument:

Public Member Functions

- Monument ()=default

    Default constructor for Monument.

- Monument (const std::string &type, int capacity, double price, std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

    Constructs a Monument with specified type, visitor capacity, price, and utility connections.

- ∼Monument ()

    Destructor for Monument.

- void displayStatus ()

    Displays the status of the Monument, including its type, capacity, price, and utilities.

- std::unique_ptr< LandMark > clone () const

    Clones the Monument, creating a unique pointer to a new Monument with identical properties.

- std::string getBuildingType () const

    Retrieves the building type for this Monument.

Additional Inherited Members

5.49.1 Detailed Description

Represents a Monument, a type of LandMark with specific characteristics and utilities.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 Monument() [1/2]

```
Monument::Monument ( )  [default]
```

Default constructor for Monument.

5.49.2.2 Monument() [2/2]

```
Monument::Monument (
             const std::string & type,
             int capacity,
             double price,
             std::shared_ptr< UtilityFlyweight > water,
             std::shared_ptr< UtilityFlyweight > power,
             std::shared_ptr< UtilityFlyweight > waste,
             std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a Monument with specified type, visitor capacity, price, and utility connections.

Constructs a Monument object with specified type, capacity, price, and utility connections.

Parameters

type	The type of the monument.
capacity	The visitor capacity of the monument.
price	The price or cost associated with the monument.
water	Shared pointer to the water utility flyweight.
power	Shared pointer to the power utility flyweight.
waste	Shared pointer to the waste management utility flyweight.
sewage	Shared pointer to the sewage management utility flyweight.
type	The type of the monument.
capacity	The visitor capacity of the monument.
price	The price or cost associated with the monument.
water	Shared pointer to water utility flyweight.
power	Shared pointer to power utility flyweight.
waste	Shared pointer to waste management utility flyweight.
sewage	Shared pointer to sewage management utility flyweight.

5.49.2.3 ∼Monument()

```
Monument::∼Monument ( )
```

Destructor for Monument.

5.49.3 Member Function Documentation

5.49.3.1 clone()

```
std::unique_ptr< LandMark > Monument::clone ( ) const
```

Clones the Monument, creating a unique pointer to a new Monument with identical properties.

Creates a clone of the Monument object.

Returns

std::unique_ptr<LandMark> A unique pointer to the cloned Monument object.

This method returns a unique pointer to a new Monument object with the same properties as the current one.

Returns

A unique pointer to a cloned Monument object.

5.49.3.2 displayStatus()

```
void Monument::displayStatus ( )  [virtual]
```

Displays the status of the Monument, including its type, capacity, price, and utilities.

Displays the status of the monument, including its type, capacity, price, and connected utilities.

Reimplemented from LandMark.

Here is the call graph for this function:

5.49.3.3 getBuildingType()

```
std::string Monument::getBuildingType ( ) const  [virtual]
```

Retrieves the building type for this Monument.

Gets the building type of the monument.

Returns

std::string A string representing the building type, which is "Monument".

A string representing the building type, which is "Monument" for this class.

Reimplemented from LandMark.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Monument.h
- COS214-Poject/src/Monument.cpp

5.50 MonumentFactory Class Reference

Factory class for creating Monument instances.

```
#include <MonumentFactory.h>
```

Inheritance diagram for MonumentFactory:

Collaboration diagram for MonumentFactory:

Public Member Functions

- std::unique_ptr< LandMark > createLandMark () override

  Creates a new instance of a Monument.

5.50.1 Detailed Description

Factory class for creating Monument instances.

The MonumentFactory class inherits from LandMarkFactory and provides a method to create new Monument objects.

5.50.2 Member Function Documentation

5.50.2.1 createLandMark()

```
std::unique_ptr< LandMark > MonumentFactory::createLandMark ( )  [override], [virtual]
```

Creates a new instance of a Monument.

Creates a new instance of a Monument as a LandMark.

Returns

std::unique_ptr<LandMark> A unique pointer to the newly created Monument.

Implements LandMarkFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/MonumentFactory.h
- COS214-Poject/src/MonumentFactory.cpp

5.51 NeutralState Class Reference

Represents a neutral state in which no specific action is taken by the NPC.

```
#include <NeutralState.h>
```

Inheritance diagram for NeutralState:

Collaboration diagram for NeutralState:

Public Member Functions

- void handle () override
  Handles behavior specific to the neutral state.
- std::string getStateName () override
  Retrieves the name of the neutral state.
- NPCState ∗ clone () const override
  Clones the current NeutralState instance.
- ∼NeutralState () override
  Destructor for the NeutralState class.

5.51.1 Detailed Description

Represents a neutral state in which no specific action is taken by the NPC.

The NeutralState class is part of the NPC state management system and serves as a middle ground or idle state, preventing unnecessary state changes.

5.51.2 Constructor & Destructor Documentation

5.51.2.1 ∼NeutralState()

```
NeutralState::∼NeutralState ( )  [override], [default]
```

Destructor for the NeutralState class.

5.51.3 Member Function Documentation

5.51.3.1 clone()

```
NPCState * NeutralState::clone ( ) const  [override], [virtual]
```

Clones the current NeutralState instance.

Creates a copy of the current NeutralState object.

Returns

NPCState∗ A pointer to a new NeutralState instance cloned from the current instance.

NPCState∗ A pointer to a new NeutralState object cloned from the current instance.

Implements NPCState.

5.51.3.2 getStateName()

```
std::string NeutralState::getStateName ( )  [override], [virtual]
```

Retrieves the name of the neutral state.

Retrieves the name of the state.

Returns

std::string The name of the state as "NeutralState".

std::string The name of the Neutral state as "NeutralState".

Implements NPCState.

5.51.3.3 handle()

```
void NeutralState::handle ( ) [override], [virtual]
```

Handles behavior specific to the neutral state.

Handles behavior in the Neutral state.

In the neutral state, no specific action is performed. It serves as an idle state between other actionable states.

In the Neutral state, no specific action is taken, serving as a middle ground to prevent unnecessary state changes.

Implements NPCState.

The documentation for this class was generated from the following files:

- COS214-Poject/src/NeutralState.h
- COS214-Poject/src/NeutralState.cpp

5.52 Node Class Reference

Represents a node within the city grid, which holds a location, an optional city component, and connections to other nodes via transport methods.

```
#include <Node.h>
```

Collaboration diagram for Node:

Public Member Functions

- Node (int x, int y)

  Constructs a Node with specified x and y coordinates.
- Node (Node &&)=default
- Node & operator= (Node &&)=default
- Node (const Node &)=delete
- Node & operator= (const Node &)=delete
- void addConnection (Node ∗node, std::unique_ptr< Transport > transport)

  Adds a connection from this node to another node using a specified transport type.
- CityComponent ∗ getComponent () const

  Retrieves the CityComponent associated with this node.
- void setComponent (CityComponent ∗comp)

  Sets the CityComponent for this node.
- const Location & getLocation () const

  Gets the location of the node.

Public Attributes

- Location location

  Location of the node on the city grid.
- CityComponent ∗ component

  Pointer to a city component assigned to this node.
- std::unordered_map< Node ∗, std::unique_ptr< Transport > > connections

  Connections to neighboring nodes with transport types.

5.52.1 Detailed Description

Represents a node within the city grid, which holds a location, an optional city component, and connections to other nodes via transport methods.

5.52.2 Constructor & Destructor Documentation

5.52.2.1 Node() [1/3]

```
Node::Node (
             int x,
             int y )
```

Constructs a Node with specified x and y coordinates.

Constructs a Node with a specific location.

Parameters

x	The x-coordinate of the node.
y	The y-coordinate of the node.

5.52.2.2 Node() [2/3]

```
Node::Node (
             Node && ) [default]
```

5.52.2.3 Node() [3/3]

```
Node::Node (
             const Node & ) [delete]
```

5.52.3 Member Function Documentation

5.52.3.1 addConnection()

```
void Node::addConnection (
             Node * node,
             std::unique_ptr< Transport > transport )
```

Adds a connection from this node to another node using a specified transport type.

Adds a connection from this node to another node.

Parameters

node	Pointer to the node to connect to.
transport	Unique pointer to the Transport type for the connection.

This function establishes a connection between the current node and the given node using the specified transport method.

Parameters

node	Pointer to the destination node.
transport	Unique pointer to the Transport object used for the connection.

5.52.3.2 getComponent()

```
CityComponent * Node::getComponent ( ) const
```

Retrieves the CityComponent associated with this node.

Returns

CityComponent∗ Pointer to the component in this node, or nullptr if none is set.

CityComponent∗ Pointer to the component stored in this node.

5.52.3.3 getLocation()

```
const Location & Node::getLocation ( ) const
```

Gets the location of the node.

Returns

const Location& Reference to the node's location.

5.52.3.4 operator=() [1/2]

```
Node& Node::operator= (
            const Node & )  [delete]
```

5.52.3.5 operator=() [2/2]

```
Node& Node::operator= (
            Node && )  [default]
```

5.52.3.6 setComponent()

```
void Node::setComponent (
            CityComponent * comp )
```

Sets the CityComponent for this node.

Parameters

comp	Pointer to the CityComponent to be assigned to this node.
comp	Pointer to the CityComponent to be stored in this node.

5.52.4 Member Data Documentation

5.52.4.1 component

```
CityComponent* Node::component
```

Pointer to a city component assigned to this node.

5.52.4.2 connections

```
std::unordered_map<Node*, std::unique_ptr<Transport> > Node::connections
```

Connections to neighboring nodes with transport types.

5.52.4.3 location

```
Location Node::location
```

Location of the node on the city grid.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Node.h
- COS214-Poject/src/Node.cpp

5.53 NPCContext Class Reference

Factory class to hire Worker NPCs with specific collection strategies.

```
#include <NPCSystem.h>
```

Collaboration diagram for NPCContext:

Static Public Member Functions

- static std::unique_ptr< WorkerNPC > hireNPC (const std::string &npcType)
  Hires a Worker NPC of a specified type.

5.53.1 Detailed Description

Factory class to hire Worker NPCs with specific collection strategies.

Creates different types of Worker NPCs based on the specified type.

5.53.2 Member Function Documentation

5.53.2.1 hireNPC()

```
static std::unique_ptr<WorkerNPC> NPCContext::hireNPC (
            const std::string & npcType )  [static]
```

Hires a Worker NPC of a specified type.

The type determines the collection strategy and cost of the NPC.

Parameters

npcType	The type of NPC to hire ("Slow", "Moderate", "Fast").

Returns

std::unique_ptr<WorkerNPC> A unique pointer to the hired Worker NPC.

The documentation for this class was generated from the following file:

- COS214-Poject/src/NPCSystem.h

5.54 NPCManager Class Reference

Singleton class managing the state and statistics of NPCs.

```
#include <NPCManager.h>
```

Collaboration diagram for NPCManager:

Public Member Functions

- NPCManager (NPCManager const &)=delete
- void operator= (NPCManager const &)=delete
- int getHappinessLevel () const

  Gets the current happiness level of NPCs.
- void setHappinessLevel (int level)

  Sets or adjusts the NPC happiness level.
- int getDonationCount () const

  Retrieves the count of NPCs in the Donation (happy) state.
- int getNeutralCount () const

  Retrieves the count of NPCs in the Neutral state.
- int getRevoltCount () const

  Retrieves the count of NPCs in the Revolt state.
- int getProductiveCount () const

  Retrieves the count of NPCs in the Productive state.
- int getCrimeCount () const

  Retrieves the count of NPCs in the Crime state.
- void incrementCount (const std::string &stateName)

  Increments the count for a given NPC state.
- void decrementCount (const std::string &stateName)

  Decrements the count for a given NPC state.
- void resetCounts ()

  Resets all state counts to zero.
- std::string getHighestState () const

  Determines the state with the highest count.
- int getTotalNPCs () const

  Gets the total number of NPCs across all states.
- void EmployeedNpcs ()

  Calculates employment levels based on the dominant state.
- void getCrimeRate ()

  Updates and retrieves the crime rate based on NPC counts.

Static Public Member Functions

- static NPCManager & getInstance ()

  Retrieves the singleton instance of NPCManager.

Protected Member Functions

- NPCManager ()

  Default constructor for NPCManager.

Private Attributes

- int happinessLevel

  Overall happiness level of NPCs.
- int happyCount

  Count of NPCs in the Donation (happy) state.
- int neutralCount

  Count of NPCs in the Neutral state.
- int revoltCount

  Count of NPCs in the Revolt state.
- int productiveCount

  Count of NPCs in the Productive state.
- int crimeCount

  Count of NPCs in the Crime state.
- int employedNpcs

  Number of employed NPCs.
- int unemployedNpcs

  Number of unemployed NPCs.

5.54.1 Detailed Description

Singleton class managing the state and statistics of NPCs.

The NPCManager class tracks various NPC states, their counts, and calculates city-wide metrics such as happiness level, employment, and crime rate.

5.54.2 Constructor & Destructor Documentation

5.54.2.1 NPCManager() [1/2]

```
NPCManager::NPCManager (
             NPCManager const & )  [delete]
```

5.54.2.2 NPCManager() [2/2]

```
NPCManager::NPCManager ( )  [protected]
```

Default constructor for NPCManager.

Default constructor initializing NPCManager with default values.

5.54.3 Member Function Documentation

5.54.3.1 decrementCount()

```
void NPCManager::decrementCount (
             const std::string & stateName )
```

Decrements the count for a given NPC state.

Parameters

stateName	The name of the state to decrement.
stateName	Name of the state to decrement count for.

Here is the caller graph for this function:

5.54.3.2 EmployeedNpcs()

```
void NPCManager::EmployeedNpcs ( )
```

Calculates employment levels based on the dominant state.

Calculates employment based on the dominant state and updates the Government's employment rate. Here is the call graph for this function:

5.54.3.3 getCrimeCount()

```
int NPCManager::getCrimeCount ( ) const
```

Retrieves the count of NPCs in the Crime state.

Retrieves the number of NPCs in the Crime state.

Returns

int Count of NPCs in the Crime state.

int The count of Crime state NPCs.

Here is the caller graph for this function:

5.54.3.4 getCrimeRate()

```
void NPCManager::getCrimeRate ( )
```

Updates and retrieves the crime rate based on NPC counts.

Updates the Government's crime rate based on NPCs in the Crime state. Here is the call graph for this function:

5.54.3.5 getDonationCount()

```
int NPCManager::getDonationCount ( ) const
```

Retrieves the count of NPCs in the Donation (happy) state.

Retrieves the number of NPCs in the Donation state.

Returns

int Count of NPCs in the Donation state.

int The count of Donation state NPCs.

Here is the caller graph for this function:

5.54.3.6 getHappinessLevel()

```
int NPCManager::getHappinessLevel ( ) const
```

Gets the current happiness level of NPCs.

Retrieves the current happiness level.

Returns

int The current happiness level.

Here is the caller graph for this function:

5.54.3.7 getHighestState()

```
std::string NPCManager::getHighestState ( ) const
```

Determines the state with the highest count.

Determines the state with the highest count among NPCs.

Returns

std::string Name of the state with the highest count.

std::string The name of the state with the highest count.

Here is the caller graph for this function:

5.54.3.8 getInstance()

```
NPCManager & NPCManager::getInstance ( ) [static]
```

Retrieves the singleton instance of NPCManager.

Singleton instance getter for NPCManager.

Returns

NPCManager& Reference to the singleton instance.

NPCManager& Singleton instance of NPCManager.

Here is the caller graph for this function:

5.54.3.9 getNeutralCount()

```
int NPCManager::getNeutralCount ( ) const
```

Retrieves the count of NPCs in the Neutral state.

Retrieves the number of NPCs in the Neutral state.

Returns

int Count of NPCs in the Neutral state.

int The count of Neutral state NPCs.

Here is the caller graph for this function:

5.54.3.10 getProductiveCount()

```
int NPCManager::getProductiveCount ( ) const
```

Retrieves the count of NPCs in the Productive state.

Retrieves the number of NPCs in the Productive state.

Returns

int Count of NPCs in the Productive state.

int The count of Productive state NPCs.

Here is the caller graph for this function:

5.54.3.11 getRevoltCount()

```
int NPCManager::getRevoltCount ( ) const
```

Retrieves the count of NPCs in the Revolt state.

Retrieves the number of NPCs in the Revolt state.

Returns

int Count of NPCs in the Revolt state.

int The count of Revolt state NPCs.

Here is the caller graph for this function:

5.54.3.12 getTotalNPCs()

```
int NPCManager::getTotalNPCs ( ) const
```

Gets the total number of NPCs across all states.

Retrieves the total number of NPCs across all states.

Returns

int Total number of NPCs.

Here is the caller graph for this function:

5.54.3.13 incrementCount()

```
void NPCManager::incrementCount (
             const std::string & stateName )
```

Increments the count for a given NPC state.

Parameters

stateName	The name of the state to increment.
stateName	Name of the state to increment count for.

Here is the caller graph for this function:

5.54.3.14 operator=()

```
void NPCManager::operator= (
            NPCManager const & )  [delete]
```

5.54.3.15 resetCounts()

```
void NPCManager::resetCounts ( )
```

Resets all state counts to zero.

5.54.3.16 setHappinessLevel()

```
void NPCManager::setHappinessLevel (
            int level )
```

Sets or adjusts the NPC happiness level.

Sets and updates the happiness level within a defined range.

Parameters

level	The value to adjust the happiness level by.
level	The level to add to the current happiness.

Here is the call graph for this function: Here is the caller graph for this function:

5.54.4 Member Data Documentation

5.54.4.1 crimeCount

```
int NPCManager::crimeCount  [private]
```

Count of NPCs in the Crime state.

5.54.4.2 employedNpcs

```
int NPCManager::employedNpcs  [private]
```

Number of employed NPCs.

5.54.4.3 happinessLevel

```
int NPCManager::happinessLevel  [private]
```

Overall happiness level of NPCs.

5.54.4.4 happyCount

```
int NPCManager::happyCount  [private]
```

Count of NPCs in the Donation (happy) state.

5.54.4.5 neutralCount

```
int NPCManager::neutralCount  [private]
```

Count of NPCs in the Neutral state.

5.54.4.6 productiveCount

```
int NPCManager::productiveCount  [private]
```

Count of NPCs in the Productive state.

5.54.4.7 revoltCount

```
int NPCManager::revoltCount  [private]
```

Count of NPCs in the Revolt state.

5.54.4.8 unemployedNpcs

```
int NPCManager::unemployedNpcs  [private]
```

Number of unemployed NPCs.

The documentation for this class was generated from the following files:

- COS214-Poject/src/NPCManager.h
- COS214-Poject/src/NPCManager.cpp

5.55 NPCObserver Class Reference

Abstract base class for NPC observers.

```
#include <NPCObserver.h>
```

Inheritance diagram for NPCObserver:

Collaboration diagram for NPCObserver:

Public Member Functions

- virtual void update ()=0

    Pure virtual function to update the observer with new state information.
- virtual NPCObserver ∗ clone ()=0

    Pure virtual function to clone the observer.
- virtual ∼NPCObserver ()=default

    Virtual destructor.

5.55.1 Detailed Description

Abstract base class for NPC observers.

The NPCObserver class provides an interface for objects that observe changes in NPC states and respond to updates accordingly. It also supports cloning, allowing derived observer types to be duplicated.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 ∼NPCObserver()

```
virtual NPCObserver::∼NPCObserver ( )  [virtual], [default]
```

Virtual destructor.

Ensures that derived classes are properly destroyed.

5.55.3 Member Function Documentation

5.55.3.1 clone()

```
virtual NPCObserver∗ NPCObserver::clone ( )  [pure virtual]
```

Pure virtual function to clone the observer.

Allows derived observer types to create a copy of themselves.

Returns

NPCObserver∗ Pointer to the cloned observer instance.

Implemented in ReactingNPCS.

5.55.3.2 update()

```
virtual void NPCObserver::update ( )  [pure virtual]
```

Pure virtual function to update the observer with new state information.

Derived classes must implement this function to define their response to updates.

Implemented in ReactingNPCS.

The documentation for this class was generated from the following file:

- COS214-Poject/src/NPCObserver.h

5.56 NPCState Class Reference

Abstract base class representing a state in the NPC state machine.

```
#include <NPCState.h>
```

Inheritance diagram for NPCState:

Collaboration diagram for NPCState:

Public Member Functions

- virtual void handle ()=0

  Pure virtual function to handle the behavior associated with the state.
- virtual std::string getStateName ()=0

  Pure virtual function to get the name of the state.
- virtual NPCState ∗ clone () const =0

  Pure virtual function to clone the state.
- virtual ∼NPCState ()=default

  Virtual destructor.

5.56.1 Detailed Description

Abstract base class representing a state in the NPC state machine.

The NPCState class provides an interface for NPC states, allowing each state to define its own behavior, retrieve its name, and be cloned.

5.56.2 Constructor & Destructor Documentation

5.56.2.1 ∼NPCState()

```
virtual NPCState::∼NPCState ( )  [virtual], [default]
```

Virtual destructor.

Ensures that derived classes are properly destroyed.

5.56.3 Member Function Documentation

5.56.3.1 clone()

```
virtual NPCState∗ NPCState::clone ( ) const  [pure virtual]
```

Pure virtual function to clone the state.

Allows derived state types to create a copy of themselves.

Returns

NPCState∗ Pointer to the cloned state instance.

Implemented in RevoltState, ProductiveState, NeutralState, DonationState, and CrimeState.

Here is the caller graph for this function:

5.56.3.2 getStateName()

```
virtual std::string NPCState::getStateName ( )  [pure virtual]
```

Pure virtual function to get the name of the state.

Provides a string representing the name of the state.

Returns

std::string Name of the state.

Implemented in RevoltState, ProductiveState, NeutralState, DonationState, and CrimeState.

Here is the caller graph for this function:

5.56.3.3 handle()

```
virtual void NPCState::handle ( )  [pure virtual]
```

Pure virtual function to handle the behavior associated with the state.

Derived classes must implement this function to define the specific actions associated with each state.

Implemented in RevoltState, ProductiveState, NeutralState, DonationState, and CrimeState.

Here is the caller graph for this function:

The documentation for this class was generated from the following file:

- COS214-Poject/src/NPCState.h

5.57 Office Class Reference

Represents an office building, a type of commercial building with utility connections.

```
#include <Office.h>
```

Inheritance diagram for Office:

Collaboration diagram for Office:

Public Member Functions

- Office ()

  Default constructor for Office.
- Office (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_↩ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Constructs an Office with specified utility connections.
- char getDisplaySymbol () const

  Gets the display symbol for the Office.
- std::string getBuildingType () const override

  Gets the building type.

Additional Inherited Members

5.57.1 Detailed Description

Represents an office building, a type of commercial building with utility connections.

5.57.2 Constructor & Destructor Documentation

5.57.2.1 Office() [1/2]

```
Office::Office ( )
```

Default constructor for Office.

Initializes an Office building with a base capacity of 600.0 and no connected utilities.

5.57.2.2 Office() [2/2]

```
Office::Office (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs an Office with specified utility connections.

Initializes an Office building with a base capacity of 600.0 and connects to the specified water, power, waste, and sewage utilities.

Parameters

water	Shared pointer to the water utility.
power	Shared pointer to the power utility.
waste	Shared pointer to the waste utility.
sewage	Shared pointer to the sewage utility.

5.57.3 Member Function Documentation

5.57.3.1 getBuildingType()

```
std::string Office::getBuildingType ( ) const  [inline], [override], [virtual]
```

Gets the building type.

Returns

A string representing the building type ("Office").

Reimplemented from CommercialBuilding.

5.57.3.2 getDisplaySymbol()

```
char Office::getDisplaySymbol ( ) const  [inline], [virtual]
```

Gets the display symbol for the Office.

Returns

Character symbol representing the Office ('O').

Implements CommercialBuilding.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Office.h
- COS214-Poject/src/Office.cpp

5.58 OfficeFactory Class Reference

Factory class for creating Office buildings.

```
#include <OfficeFactory.h>
```

Inheritance diagram for OfficeFactory:

Collaboration diagram for OfficeFactory:

Public Member Functions

- std::unique_ptr< CommercialBuilding > createBuilding () override

    Creates a new Office building.

5.58.1 Detailed Description

Factory class for creating Office buildings.

The OfficeFactory class provides a method to create Office buildings, implementing the factory pattern for creating instances of the CommercialBuilding type.

5.58.2 Member Function Documentation

5.58.2.1 createBuilding()

`std::unique_ptr< CommercialBuilding > OfficeFactory::createBuilding ( ) [override], [virtual]`

Creates a new Office building.

Creates an instance of an Office building.

Returns

A unique pointer to a new Office building.

This method implements the createBuilding function defined in the OfficeFactory class. It creates and returns a unique pointer to a new Office instance, allowing the management of Office buildings within the city simulation.

Returns

std::unique_ptr<CommercialBuilding> A unique pointer to the newly created Office.

Implements CommercialFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/OfficeFactory.h
- COS214-Poject/src/OfficeFactory.cpp

5.59 Oil Class Reference

Represents an oil resource in the simulation.

`#include <Oil.h>`

Inheritance diagram for Oil:

Collaboration diagram for Oil:

Public Member Functions

- Oil (int quantity, double marketValue)

  Constructs an Oil resource.
- ∼Oil ()

  Destructor for the Oil resource.
- void displayStatus () override

  Displays the current status of the Oil resource.
- std::string getBuildingType () const

  Retrieves the building type of the resource.

5.59.1 Detailed Description

Represents an oil resource in the simulation.

The Oil class inherits from the IncomeResourceProduct and provides functionality to manage the quantity and market value of oil resources. It allows for displaying the current status of the oil resource.

5.59.2 Constructor & Destructor Documentation

5.59.2.1 Oil()

```
Oil::Oil (
            int quantity,
            double marketValue )
```

Constructs an Oil resource.

Constructor for the Oil resource.

Initializes the Oil resource with a specified quantity and market value.

Parameters

quantity	The initial quantity of the oil resource.
marketValue	The market value per unit of the oil resource.

Initializes an instance of the Oil resource with the specified quantity and market value. It sets the name of the resource to "Oil" and initializes the base class attributes using the constructor of IncomeResourceProduct.

Parameters

quantity	The initial quantity of the oil resource.
marketValue	The market value per unit of the oil resource.

5.59.2.2 ∼Oil()

```
Oil::∼Oil ( )
```

Destructor for the Oil resource.

Cleans up any resources held by the Oil object.

Cleans up any resources held by the Oil object. The destructor may be extended in the future to include additional cleanup logic.

5.59.3 Member Function Documentation

5.59.3.1 displayStatus()

```
void Oil::displayStatus ( )  [override], [virtual]
```

Displays the current status of the Oil resource.

Outputs the name, quantity, market value per unit, and total value of the oil resource.

Outputs the name, quantity, market value per unit, and total value of the oil resource to the standard output. This provides a comprehensive summary of the resource's current state.

Implements CityComponent.

Here is the call graph for this function:

5.59.3.2 getBuildingType()

```
std::string Oil::getBuildingType ( ) const  [inline], [virtual]
```

Retrieves the building type of the resource.

Returns

A string representing the building type, which is "Oil".

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Oil.h
- COS214-Poject/src/Oil.cpp

5.60 Park Class Reference

Represents a park in the city simulation.

```
#include <Park.h>
```

Inheritance diagram for Park:

Collaboration diagram for Park:

Public Member Functions

- Park ()=default

    Default constructor for the Park class.
- Park (const std::string &type, int capacity, double price, std::shared_ptr< UtilityFlyweight > water, std↩ ::shared_ptr< UtilityFlyweight > power, std::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

    Constructs a Park object with specified parameters.
- ∼Park ()

    Destructor for the Park object.
- void displayStatus () override

    Displays the current status of the Park.
- std::unique_ptr< LandMark > clone () const

    Creates a clone of the Park object.
- std::string getBuildingType () const

    Retrieves the building type of the park.

Additional Inherited Members

5.60.1 Detailed Description

Represents a park in the city simulation.

The Park class inherits from the LandMark class and includes details about the park's type, capacity, price, and utility connections.

5.60.2 Constructor & Destructor Documentation

5.60.2.1 Park() [1/2]

```
Park::Park ( )  [default]
```

Default constructor for the Park class.

5.60.2.2 Park() [2/2]

```
Park::Park (
             const std::string & type,
             int capacity,
             double price,
             std::shared_ptr< UtilityFlyweight > water,
             std::shared_ptr< UtilityFlyweight > power,
             std::shared_ptr< UtilityFlyweight > waste,
             std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a Park object with specified parameters.

Constructs a Park object.

Initializes the Park with a specified type, capacity, price, and utility connections.

Parameters

type	The type of the park.
capacity	The maximum visitor capacity of the park.
price	The cost associated with the park.
water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage management utility.

5.60.2.3 ∼Park()

```
Park::∼Park ( )
```

Destructor for the Park object.

Cleans up any resources held by the Park object.

5.60.3 Member Function Documentation

5.60.3.1 clone()

```
std::unique_ptr< LandMark > Park::clone ( ) const
```

Creates a clone of the Park object.

Returns

A unique pointer to a new Park object that is a copy of this one.

5.60.3.2 displayStatus()

```
void Park::displayStatus ( ) [override], [virtual]
```

Displays the current status of the Park.

Outputs the name, capacity, price, and status of the park's utilities.

Reimplemented from LandMark.

Here is the call graph for this function:

5.60.3.3 getBuildingType()

```
std::string Park::getBuildingType ( ) const  [virtual]
```

Retrieves the building type of the park.

Returns

A string representing the building type, which is "Park".

Reimplemented from LandMark.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Park.h
- COS214-Poject/src/Park.cpp

5.61 ParkFactory Class Reference

Factory class for creating Park objects.

```
#include <ParkFactory.h>
```

Inheritance diagram for ParkFactory:

Collaboration diagram for ParkFactory:

Public Member Functions

- std::unique_ptr< LandMark > createLandMark ()

    Creates a new Park object.

Additional Inherited Members

5.61.1 Detailed Description

Factory class for creating Park objects.

The ParkFactory class is responsible for creating instances of the Park class, allowing for encapsulation of the Park creation process within the city simulation.

5.61.2 Member Function Documentation

5.61.2.1 createLandMark()

```
std::unique_ptr< LandMark > ParkFactory::createLandMark ( )
```

Creates a new Park object.

This method creates and returns a unique pointer to a new Park instance.

Returns

A unique pointer to a newly created Park object.

This method creates a unique pointer to a new Park instance, which can be used to represent a park in the city simulation.

Returns

A unique pointer to a newly created Park object.

The documentation for this class was generated from the following files:

- COS214-Poject/src/ParkFactory.h
- COS214-Poject/src/ParkFactory.cpp

5.62 PetrochemicalFacility Class Reference

Represents a petrochemical facility that processes oil and concrete.

```
#include <PetroChemicalFacility.h>
```

Inheritance diagram for PetrochemicalFacility:

Collaboration diagram for PetrochemicalFacility:

Public Member Functions

- PetrochemicalFacility (std::shared_ptr< IncomeResourceProduct > oil, std::shared_ptr< ConstructionResourceProduct > concrete, MapGrid ∗grid, std::map< std::string, int > &collectedResources)

  Constructs a PetrochemicalFacility with specified resources and grid.
- void processOil (int amount)

  Processes a specified amount of oil in the facility.
- void processConcrete (int amount)

  Processes a specified amount of concrete in the facility.

Static Public Attributes

- static const int PETRO_CHEMICAL_RANGE = 5

  The collection range for the petrochemical facility.

Additional Inherited Members

5.62.1 Detailed Description

Represents a petrochemical facility that processes oil and concrete.

The PetrochemicalFacility class is derived from the Industry class and provides functionality to process oil and concrete resources, while managing the associated pollution levels and collection ranges.

5.62.2 Constructor & Destructor Documentation

5.62.2.1 PetrochemicalFacility()

```
PetrochemicalFacility::PetrochemicalFacility (
            std::shared_ptr< IncomeResourceProduct > oil,
            std::shared_ptr< ConstructionResourceProduct > concrete,
            MapGrid * grid,
            std::map< std::string, int > & collectedResources )
```

Constructs a PetrochemicalFacility with specified resources and grid.

Constructor for the PetrochemicalFacility class.

Parameters

oil	A shared pointer to an IncomeResourceProduct representing oil.
concrete	A shared pointer to a ConstructionResourceProduct representing concrete.
grid	A pointer to the MapGrid for managing resource locations.
collectedResources	A reference to a map tracking collected resources.

Initializes a PetrochemicalFacility with specified resources, processing capabilities, and a grid for resource management.

Parameters

oil	A shared pointer to an IncomeResourceProduct representing oil.
concrete	A shared pointer to a ConstructionResourceProduct representing concrete.
grid	A pointer to the MapGrid for resource allocation and management.
collectedResources	A reference to a map that tracks collected resources.

5.62.3 Member Function Documentation

5.62.3.1 processConcrete()

```
void PetrochemicalFacility::processConcrete (
             int amount )
```

Processes a specified amount of concrete in the facility.

This method handles the processing of the given amount of concrete.

Parameters

amount	The amount of concrete to be processed.

This method processes the given amount of concrete and updates the collected resources accordingly.

Parameters

amount	The amount of concrete to be processed.

Here is the call graph for this function:

5.62.3.2 processOil()

```
void PetrochemicalFacility::processOil (
             int amount )
```

Processes a specified amount of oil in the facility.

This method handles the processing of the given amount of oil.

Parameters

amount	The amount of oil to be processed.

This method processes the given amount of oil and updates the collected resources accordingly.

Parameters

amount	The amount of oil to be processed.

Here is the call graph for this function:

5.62.4 Member Data Documentation

5.62.4.1 PETRO_CHEMICAL_RANGE

```
const int PetrochemicalFacility::PETRO_CHEMICAL_RANGE = 5  [static]
```

The collection range for the petrochemical facility.

The documentation for this class was generated from the following files:

- COS214-Poject/src/PetroChemicalFacility.h
- COS214-Poject/src/PetroChemicalFacility.cpp

5.63 PlaceComponentCommand Class Reference

Command to place a CityComponent on a MapGrid at a specified Location.

```
#include <Command.h>
```

Inheritance diagram for PlaceComponentCommand:

Collaboration diagram for PlaceComponentCommand:

Public Member Functions

- PlaceComponentCommand (MapGrid &grid, Location loc, std::shared_ptr< CityComponent > comp)

  Constructs a PlaceComponentCommand with specified MapGrid, Location, and component.
- void execute () override

  Executes the command, placing the component on the MapGrid.
- void undo () override

  Undoes the command, restoring the previous component at the location.
- std::unique_ptr< Command > clone () const override

  Clones the PlaceComponentCommand.

Private Attributes

- MapGrid & grid

  Reference to the MapGrid where the component is placed.
- Location location

  The location on the MapGrid for the component.
- std::shared_ptr< CityComponent > component

  The component to place on the MapGrid.
- std::shared_ptr< CityComponent > previousComponent

  Stores the previous component at the location.

5.63.1 Detailed Description

Command to place a CityComponent on a MapGrid at a specified Location.

5.63.2 Constructor & Destructor Documentation

5.63.2.1 PlaceComponentCommand()

```
PlaceComponentCommand::PlaceComponentCommand (
            MapGrid & grid,
            Location loc,
            std::shared_ptr< CityComponent > comp ) [inline]
```

Constructs a PlaceComponentCommand with specified MapGrid, Location, and component.

Parameters

grid	Reference to the MapGrid.
loc	The location to place the component.
comp	Shared pointer to the CityComponent to be placed.

5.63.3 Member Function Documentation

5.63.3.1 clone()

```
std::unique_ptr<Command> PlaceComponentCommand::clone ( ) const [inline], [override], [virtual]
```

Clones the PlaceComponentCommand.

Returns

A unique pointer to a new PlaceComponentCommand instance.

Implements Command.

5.63.3.2 execute()

```
void PlaceComponentCommand::execute ( ) [inline], [override], [virtual]
```

Executes the command, placing the component on the MapGrid.

Implements Command.

Here is the call graph for this function:

5.63.3.3 undo()

```
void PlaceComponentCommand::undo ( )  [inline], [override], [virtual]
```

Undoes the command, restoring the previous component at the location.

Implements Command.

Here is the call graph for this function:

5.63.4 Member Data Documentation

5.63.4.1 component

```
std::shared_ptr<CityComponent> PlaceComponentCommand::component  [private]
```

The component to place on the MapGrid.

5.63.4.2 grid

```
MapGrid& PlaceComponentCommand::grid  [private]
```

Reference to the MapGrid where the component is placed.

5.63.4.3 location

```
Location PlaceComponentCommand::location  [private]
```

The location on the MapGrid for the component.

5.63.4.4 previousComponent

```
std::shared_ptr<CityComponent> PlaceComponentCommand::previousComponent  [private]
```

Stores the previous component at the location.

The documentation for this class was generated from the following file:

- COS214-Poject/src/Command.h

5.64 MapGrid::PlacementResult Struct Reference

`#include <MapGrid.h>`

Collaboration diagram for MapGrid::PlacementResult:

Public Attributes

- bool success
- std::string message

5.64.1 Member Data Documentation

5.64.1.1 message

```
std::string MapGrid::PlacementResult::message
```

5.64.1.2 success

```
bool MapGrid::PlacementResult::success
```

The documentation for this struct was generated from the following file:

- COS214-Poject/src/MapGrid.h

5.65 PlantFactory Class Reference

A factory class for creating resources related to plants.

`#include <PlantFactory.h>`

Inheritance diagram for PlantFactory:

Collaboration diagram for PlantFactory:

Public Member Functions

- PlantFactory ()

  Constructs a PlantFactory.
- ∼PlantFactory ()

  Destroys the PlantFactory.
- std::unique_ptr< IncomeResourceProduct > createIncomeR (int quantity) override

  Creates an income-generating resource.
- std::unique_ptr< ConstructionResourceProduct > createConstructionR (int quantity) override

  Creates a construction resource.

5.65.1 Detailed Description

A factory class for creating resources related to plants.

The PlantFactory class is responsible for creating income-generating and construction resources, specifically Coal and Wood.

5.65.2 Constructor & Destructor Documentation

5.65.2.1 PlantFactory()

```
PlantFactory::PlantFactory ( )
```

Constructs a PlantFactory.

Initializes the PlantFactory instance.

This constructor initializes the PlantFactory and outputs a message indicating that the factory has been created.

5.65.2.2 ∼PlantFactory()

```
PlantFactory::∼PlantFactory ( )
```

Destroys the PlantFactory.

Cleans up resources used by the PlantFactory.

This destructor cleans up the PlantFactory resources and outputs a message indicating that the factory has been deleted.

5.65.3 Member Function Documentation

5.65.3.1 createConstructionR()

```
std::unique_ptr< ConstructionResourceProduct > PlantFactory::createConstructionR (
            int quantity )  [override], [virtual]
```

Creates a construction resource.

Parameters

quantity	The quantity of the construction resource to create.

Returns

A unique pointer to the created ConstructionResourceProduct.

This method creates an instance of a construction resource (Wood) with the specified quantity.

Parameters

quantity	The quantity of the construction resource to create.

Returns

A unique pointer to the created ConstructionResourceProduct (Wood).

Implements ResourceFactory.

5.65.3.2 createIncomeR()

```
std::unique_ptr< IncomeResourceProduct > PlantFactory::createIncomeR (
            int quantity )  [override], [virtual]
```

Creates an income-generating resource.

Parameters

quantity	The quantity of the income-generating resource to create.

Returns

A unique pointer to the created IncomeResourceProduct.

This method creates an instance of an income-generating resource (Coal) with the specified quantity.

Parameters

quantity	The quantity of the income-generating resource to create.

Returns

A unique pointer to the created IncomeResourceProduct (Coal).

Implements ResourceFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/PlantFactory.h
- COS214-Poject/src/PlantFactory.cpp

5.66 PoliceStation Class Reference

Represents a police station public service in the city.

`#include <PoliceStation.h>`

Inheritance diagram for PoliceStation:

Collaboration diagram for PoliceStation:

Public Member Functions

- PoliceStation ()=default

    Default constructor for PoliceStation.
- PoliceStation (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > electricity, std↩ ::shared_ptr< UtilityFlyweight > wasteManagement, std::shared_ptr< UtilityFlyweight > sewage, std::string buildingStatus)

    Constructs a PoliceStation with specified utilities and status.
- ∼PoliceStation ()=default

    Default destructor for PoliceStation.
- void provideService () override

    Provides police services to ensure public safety.
- std::unique_ptr< PublicService > clone () const

    Creates a clone of the PoliceStation object.
- void displayStatus () override

    Displays the current status of the Police Station.
- char getDisplaySymbol () const override

    Retrieves the display symbol for the Police Station.

Private Attributes

- std::string status

    The operational status of the police station.

Additional Inherited Members

5.66.1 Detailed Description

Represents a police station public service in the city.

Represents a police station, which is a public service in the city.

The PoliceStation class inherits from the PublicService class and includes utilities required for its operation, along with its status.

The PoliceStation class inherits from the PublicService class and provides methods to manage its utilities and display its status.

5.66.2 Constructor & Destructor Documentation

5.66.2.1 PoliceStation() [1/2]

```
PoliceStation::PoliceStation ( )  [default]
```

Default constructor for PoliceStation.

5.66.2.2 PoliceStation() [2/2]

```
PoliceStation::PoliceStation (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > electricity,
            std::shared_ptr< UtilityFlyweight > wasteManagement,
            std::shared_ptr< UtilityFlyweight > sewage,
            std::string buildingStatus )
```

Constructs a PoliceStation with specified utilities and status.

Parameters

water	Shared pointer to the water utility.
electricity	Shared pointer to the electricity utility.
wasteManagement	Shared pointer to the waste management utility.
sewage	Shared pointer to the sewage utility.
buildingStatus	The operational status of the police station.

5.66.2.3 ∼PoliceStation()

```
PoliceStation::∼PoliceStation ( )  [default]
```

Default destructor for PoliceStation.

5.66.3 Member Function Documentation

5.66.3.1 clone()

```
std::unique_ptr< PublicService > PoliceStation::clone ( ) const
```

Creates a clone of the PoliceStation object.

Returns

A unique pointer to the cloned PoliceStation object.

5.66.3.2 displayStatus()

```
void PoliceStation::displayStatus ( )  [override], [virtual]
```

Displays the current status of the Police Station.

This includes the connection status of water, electricity, waste management, sewage, and the overall operational status of the police station.

Implements PublicService.

5.66.3.3 getDisplaySymbol()

```
char PoliceStation::getDisplaySymbol ( ) const  [override], [virtual]
```

Retrieves the display symbol for the Police Station.

Returns

The character symbol representing the Police Station ('P').

Implements PublicService.

5.66.3.4 provideService()

```
void PoliceStation::provideService ( )  [inline], [override], [virtual]
```

Provides police services to ensure public safety.

Implements PublicService.

5.66.4 Member Data Documentation

5.66.4.1 status

```
std::string PoliceStation::status [private]
```

The operational status of the police station.

The documentation for this class was generated from the following files:

- COS214-Poject/src/PoliceStation.h
- COS214-Poject/src/PoliceStation.cpp

5.67 PoliceStationFactory Class Reference

Factory class to create PoliceStation instances.

```
#include <PoliceStationFactory.h>
```

Inheritance diagram for PoliceStationFactory:

Collaboration diagram for PoliceStationFactory:

Public Member Functions

- std::unique_ptr< PublicService > createPublicService ()

    Creates a new PoliceStation object.

5.67.1 Detailed Description

Factory class to create PoliceStation instances.

This class inherits from PublicServiceFactory and provides the implementation for creating instances of PoliceStation.

5.67.2 Member Function Documentation

5.67.2.1 createPublicService()

```
std::unique_ptr< PublicService > PoliceStationFactory::createPublicService ( ) [virtual]
```

Creates a new PoliceStation object.

This method overrides the createPublicService function from the PublicServiceFactory interface, returning a unique pointer to a new instance of PoliceStation.

Returns

A unique pointer to the newly created PoliceStation object.

Implements PublicServiceFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/PoliceStationFactory.h
- COS214-Poject/src/PoliceStationFactory.cpp

5.68 PowerPlant Class Reference

Represents a Power Plant in the utility system.

```
#include <PowerPlant.h>
```

Inheritance diagram for PowerPlant:

Collaboration diagram for PowerPlant:

Public Member Functions

- PowerPlant ()

  Default constructor for PowerPlant.
- PowerPlant (const std::string &n, double c, int cap, double radius, bool operational, int l, int consumption, const std::map< std::string, int > &resources)

  Parameterized constructor for PowerPlant.
- void connect (double distance)

  Connects the power plant and checks its operational status.
- std::shared_ptr< UtilityFlyweight > clone () const

  Clones the PowerPlant instance.

Additional Inherited Members

5.68.1 Detailed Description

Represents a Power Plant in the utility system.

The PowerPlant class inherits from UtilityFlyweight and represents a power generation facility with characteristics such as cost, capacity, operational status, and the resources it consumes.

5.68.2 Constructor & Destructor Documentation

5.68.2.1 PowerPlant() `[1/2]`

```
PowerPlant::PowerPlant ( )
```

Default constructor for PowerPlant.

Initializes a PowerPlant instance with default values representing typical characteristics of a power plant.

Initializes a PowerPlant instance with default values representing a typical power plant's characteristics such as name, cost, capacity, coverage radius, operational status, level, consumption rate, and required resources.

5.68.2.2 PowerPlant() [2/2]

```
PowerPlant::PowerPlant (
            const std::string & n,
            double c,
            int cap,
            double radius,
            bool operational,
            int l,
            int consumption,
            const std::map< std::string, int > & resources )
```

Parameterized constructor for PowerPlant.

Initializes a PowerPlant instance with specified values.

Parameters

n	The name of the power plant.
c	The cost of the power plant.
cap	The capacity of the power plant.
radius	The coverage radius of the power plant.
operational	Indicates if the power plant is operational.
l	The level of the power plant.
consumption	The resource consumption rate of the power plant.
resources	A map of required resources and their quantities.

5.68.3 Member Function Documentation

5.68.3.1 clone()

```
std::shared_ptr< UtilityFlyweight > PowerPlant::clone ( ) const  [virtual]
```

Clones the PowerPlant instance.

Creates a shared pointer to a new instance of PowerPlant with the same properties.

Returns

A shared pointer to the cloned PowerPlant instance.

Implements UtilityFlyweight.

5.68.3.2 connect()

```
void PowerPlant::connect (
             double distance )  [virtual]
```

Connects the power plant and checks its operational status.

Connects the power plant and checks its operational status within a given distance.

If the power plant is operational and the distance is within the effect radius, it indicates that the power plant is providing power. Otherwise, it notifies that the connection is out of range.

Parameters

distance	The distance to check against the effect radius.

Implements UtilityFlyweight.

The documentation for this class was generated from the following files:

- COS214-Poject/src/PowerPlant.h
- COS214-Poject/src/PowerPlant.cpp

5.69 ProductiveState Class Reference

Represents the productive state of NPCs.

`#include <ProductiveState.h>`

Inheritance diagram for ProductiveState:

Collaboration diagram for ProductiveState:

Public Member Functions

- void handle () override

  Handles the logic for productivity boosts.
- std::string getStateName () override

  Retrieves the name of the productive state.
- NPCState ∗ clone () const override

  Clones the current ProductiveState instance.
- ∼ProductiveState ()

  Destructor for the ProductiveState class.

5.69.1 Detailed Description

Represents the productive state of NPCs.

This class handles the logic for NPCs that are in a productive state, including the calculation of productivity boosts based on the number of productive NPCs.

5.69.2 Constructor & Destructor Documentation

5.69.2.1 ∼ProductiveState()

```
ProductiveState::∼ProductiveState ( )  [default]
```

Destructor for the ProductiveState class.

5.69.3 Member Function Documentation

5.69.3.1 clone()

```
NPCState * ProductiveState::clone ( ) const  [override], [virtual]
```

Clones the current ProductiveState instance.

Returns

A pointer to the newly created ProductiveState instance.

This method creates a new instance of ProductiveState that is a copy of the current instance.

Returns

A pointer to the newly created ProductiveState instance.

Implements NPCState.

5.69.3.2 getStateName()

```
std::string ProductiveState::getStateName ( )  [override], [virtual]
```

Retrieves the name of the productive state.

Retrieves the name of the state.

Returns

A string representing the name of the state.

Implements NPCState.

5.69.3.3 handle()

```
void ProductiveState::handle ( )  [override], [virtual]
```

Handles the logic for productivity boosts.

Handles the logic for the productive state of NPCs.

This method calculates the chance of a productivity boost and applies it to the production rate in the Government class if applicable.

This method calculates the chance of a productivity boost based on the number of NPCs in the productive state. If the productivity boost occurs, it increases the production rate in the Government class.

Implements NPCState.

Here is the call graph for this function:

The documentation for this class was generated from the following files:

- COS214-Poject/src/ProductiveState.h
- COS214-Poject/src/ProductiveState.cpp

5.70 PublicService Class Reference

Abstract base class representing a public service.

`#include <PublicService.h>`

Inheritance diagram for PublicService:

Collaboration diagram for PublicService:

Public Member Functions

- PublicService ()=default

  Default constructor.
- PublicService (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std↩ ::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Parameterized constructor for initializing a public service.
- virtual void provideService ()=0

  Pure virtual function to provide a specific service.
- virtual ∼PublicService ()=default

  Virtual destructor.
- virtual void displayStatus ()=0

  Pure virtual function to display the status of the public service.
- std::string getBuildingType () const override

  Returns the type of the building.
- virtual char getDisplaySymbol () const =0

  Pure virtual function to get the display symbol for the public service on a map.
- int getUtilityCoverage () const

  Calculates the coverage of utilities connected to the public service.
- bool hasUtilities () const

  Checks if all necessary utilities are connected.
- void accept (taxCollector ∗TC) override

  Accepts a tax collector visitor.

Protected Attributes

- std::shared_ptr< UtilityFlyweight > waterSupply

  Connection to water supply utility.
- std::shared_ptr< UtilityFlyweight > powerSupply

  Connection to power supply utility.
- std::shared_ptr< UtilityFlyweight > wasteManagement

  Connection to waste management utility.
- std::shared_ptr< UtilityFlyweight > sewageManagement

  Connection to sewage management utility.

5.70.1 Detailed Description

Abstract base class representing a public service.

This class is responsible for managing utility connections and providing services to the city. It serves as a base for specific public service types (e.g., PoliceStation, FireStation).

5.70.2 Constructor & Destructor Documentation

5.70.2.1 PublicService() [1/2]

```
PublicService::PublicService ( )  [default]
```

Default constructor.

5.70.2.2 PublicService() [2/2]

```
PublicService::PublicService (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Parameterized constructor for initializing a public service.

Constructor for the PublicService class.

Parameters

water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage management utility.

Initializes the public service with the specified utility connections.

Parameters

water	A shared pointer to the water utility.
power	A shared pointer to the power utility.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage management utility.

5.70.2.3 ∼PublicService()

```
virtual PublicService::∼PublicService ( )  [virtual], [default]
```

Virtual destructor.

5.70.3 Member Function Documentation

5.70.3.1 accept()

```
void PublicService::accept (
            taxCollector * collector )  [inline], [override], [virtual]
```

Accepts a tax collector visitor.

Part of the Visitor pattern, allowing tax collectors to visit city components and perform operations based on the component's type.

Parameters

collector	A pointer to a taxCollector object.

Implements CityComponent.

5.70.3.2 displayStatus()

```
virtual void PublicService::displayStatus ( )  [pure virtual]
```

Pure virtual function to display the status of the public service.

Implements CityComponent.

Implemented in School, PoliceStation, Hospital, and FireStation.

5.70.3.3 getBuildingType()

```
std::string PublicService::getBuildingType ( ) const  [inline], [override], [virtual]
```

Returns the type of the building.

Returns

A string representing the building type.

Implements CityComponent.

5.70.3.4 getDisplaySymbol()

```
virtual char PublicService::getDisplaySymbol ( ) const  [pure virtual]
```

Pure virtual function to get the display symbol for the public service on a map.

Implemented in School, PoliceStation, Hospital, and FireStation.

5.70.3.5 getUtilityCoverage()

```
int PublicService::getUtilityCoverage ( ) const  [inline]
```

Calculates the coverage of utilities connected to the public service.

Returns

The number of connected utilities.

Here is the caller graph for this function:

5.70.3.6 hasUtilities()

```
bool PublicService::hasUtilities ( ) const  [inline]
```

Checks if all necessary utilities are connected.

Returns

True if all utilities are connected, false otherwise.

Here is the caller graph for this function:

5.70.3.7 provideService()

```
virtual void PublicService::provideService ( )  [pure virtual]
```

Pure virtual function to provide a specific service.

This function must be implemented by derived classes.

Implemented in School, PoliceStation, Hospital, and FireStation.

5.70.4 Member Data Documentation

5.70.4.1 powerSupply

```
std::shared_ptr<UtilityFlyweight> PublicService::powerSupply  [protected]
```

Connection to power supply utility.

5.70.4.2 sewageManagement

```
std::shared_ptr<UtilityFlyweight> PublicService::sewageManagement  [protected]
```

Connection to sewage management utility.

5.70.4.3 wasteManagement

```
std::shared_ptr<UtilityFlyweight> PublicService::wasteManagement  [protected]
```

Connection to waste management utility.

5.70.4.4 waterSupply

```
std::shared_ptr<UtilityFlyweight> PublicService::waterSupply  [protected]
```

Connection to water supply utility.

The documentation for this class was generated from the following files:

- COS214-Poject/src/PublicService.h
- COS214-Poject/src/PublicService.cpp

5.71 PublicServiceFactory Class Reference

An abstract factory class for creating instances of PublicService.

```
#include <PublicServiceFactory.h>
```

Inheritance diagram for PublicServiceFactory:

Collaboration diagram for PublicServiceFactory:

Public Member Functions

- virtual std::unique_ptr< PublicService > createPublicService ()=0

    Pure virtual function to create a PublicService object.
- virtual ∼PublicServiceFactory ()

    Virtual destructor for PublicServiceFactory.

5.71.1 Detailed Description

An abstract factory class for creating instances of PublicService.

This factory provides an interface for creating public service buildings in the city simulation. Derived classes must implement the `createPublicService` function to provide specific types of public services.

5.71.2 Constructor & Destructor Documentation

5.71.2.1 ∼PublicServiceFactory()

```
virtual PublicServiceFactory::∼PublicServiceFactory ( )  [inline], [virtual]
```

Virtual destructor for PublicServiceFactory.

5.71.3 Member Function Documentation

5.71.3.1 createPublicService()

```
virtual std::unique_ptr<PublicService> PublicServiceFactory::createPublicService ( )  [pure
virtual]
```

Pure virtual function to create a PublicService object.

Returns

A unique pointer to the created PublicService object.

Implemented in PoliceStationFactory, HospitalFactory, and FireStationFactory.

The documentation for this class was generated from the following file:

- COS214-Poject/src/PublicServiceFactory.h

5.72 ReactingNPCS Class Reference

Concrete observer class representing an NPC that reacts to state changes.

`#include <ReactingNPCS.h>`

Inheritance diagram for ReactingNPCS:

Collaboration diagram for ReactingNPCS:

Public Member Functions

- ReactingNPCS ()

  Default constructor initializing the NPC in a neutral state.
- virtual ∼ReactingNPCS ()

  Destructor for cleaning up dynamically allocated state.
- ReactingNPCS (const ReactingNPCS &other)

  Copy constructor for deep copying ReactingNPCS objects.
- void update () override

  Updates the NPC's state based on observed changes.
- void changeState (NPCState ∗newState)

  Changes the current state of the NPC.
- NPCState ∗ getState ()
- NPCObserver ∗ clone () override

  Clones the NPC, creating a copy with the same state.

Private Attributes

- NPCState ∗ state

  Current state of the NPC.

5.72.1 Detailed Description

Concrete observer class representing an NPC that reacts to state changes.

ReactingNPCS observes changes in the game environment and responds by changing its state. It utilizes a state pattern to manage different NPC states and behaviors.

5.72.2 Constructor & Destructor Documentation

5.72.2.1 ReactingNPCS() [1/2]

`ReactingNPCS::ReactingNPCS ( )`

Default constructor initializing the NPC in a neutral state.

Constructs a ReactingNPCS object in the NeutralState and increments the NPCManager count. Here is the call graph for this function: Here is the caller graph for this function:

5.72.2.2 ∼ReactingNPCS()

```
ReactingNPCS::∼ReactingNPCS ( )  [virtual]
```

Destructor for cleaning up dynamically allocated state.

Destructor for ReactingNPCS, ensuring the NPCManager count is decremented for the current state. Here is the call graph for this function:

5.72.2.3 ReactingNPCS() `[2/2]`

```
ReactingNPCS::ReactingNPCS (
              const ReactingNPCS & other )
```

Copy constructor for deep copying ReactingNPCS objects.

Copy constructor to create a deep copy of the ReactingNPCS object.

Parameters

other	The other ReactingNPCS instance to copy.
other	The ReactingNPCS object to copy.

Here is the call graph for this function:

5.72.3 Member Function Documentation

5.72.3.1 changeState()

```
void ReactingNPCS::changeState (
              NPCState * newState )
```

Changes the current state of the NPC.

Changes the state of the NPC, managing NPCManager counts for each state.

Parameters

newState	Pointer to the new NPCState to transition into.
newState	The new state for the NPC.

Here is the call graph for this function: Here is the caller graph for this function:

5.72.3.2 clone()

```
NPCObserver * ReactingNPCS::clone ( )  [override], [virtual]
```

Clones the NPC, creating a copy with the same state.

Clones the NPC by creating a deep copy using the copy constructor.

Returns

A pointer to the cloned NPCObserver object.

Implements NPCObserver.

Here is the call graph for this function: Here is the caller graph for this function:

5.72.3.3 getState()

```
NPCState* ReactingNPCS::getState ( )  [inline]
```

5.72.3.4 update()

```
void ReactingNPCS::update ( )  [override], [virtual]
```

Updates the NPC's state based on observed changes.

Updates the NPC's state based on the current happiness and pollution levels, with a chance of following consensus.

This method is called when there is a change in the observed subject, allowing the NPC to respond accordingly.

Implements NPCObserver.

Here is the call graph for this function:

5.72.4 Member Data Documentation

5.72.4.1 state

```
NPCState* ReactingNPCS::state  [private]
```

Current state of the NPC.

The documentation for this class was generated from the following files:

- COS214-Poject/src/ReactingNPCS.h
- COS214-Poject/src/ReactingNPCS.cpp

5.73 ResidentialBuilding Class Reference

Represents a residential building within the city.

`#include <ResidentialBuilding.h>`

Inheritance diagram for ResidentialBuilding:

Collaboration diagram for ResidentialBuilding:

Public Member Functions

- ResidentialBuilding (int bedrooms, double cost, std::shared_ptr< UtilityFlyweight > water, std::shared_↩ptr< UtilityFlyweight > power, std::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Constructs a ResidentialBuilding with specified utilities and attributes.
- void setTaxPaid (bool paid)

  Sets the tax-paid status.
- bool isTaxPaid () const

  Checks if taxes have been paid.
- void accept (taxCollector ∗TC) override

  Accepts a taxCollector visitor to collect taxes from this building.
- double getPrice () const

  Gets the price of the building.
- void displayStatus () override

  Displays the status of the residential building.
- std::string getBuildingType () const override

  Gets the building type.
- virtual char getDisplaySymbol () const =0

  Gets the symbol used for map display.
- int getUtilityCoverage () const

  Calculates the utility coverage for this building.
- bool hasUtilities () const

  Checks if the building has all essential utilities connected.
- void setWaterSupply (std::shared_ptr< UtilityFlyweight > utility)
- void setPowerSupply (std::shared_ptr< UtilityFlyweight > utility)
- void setWasteManagement (std::shared_ptr< UtilityFlyweight > utility)
- void setSewageManagement (std::shared_ptr< UtilityFlyweight > utility)
- virtual std::string getDisplayColor () const

  Gets the display color based on utility coverage.

Protected Attributes

- int bedrooms

  Number of bedrooms in the building.
- double price

  Price of the residential building.
- bool taxPaid

  Flag indicating whether taxes are paid.
- std::shared_ptr< UtilityFlyweight > waterSupply

  Water utility connection.
- std::shared_ptr< UtilityFlyweight > powerSupply

  Power utility connection.
- std::shared_ptr< UtilityFlyweight > wasteManagement

  Waste management utility connection.
- std::shared_ptr< UtilityFlyweight > sewageManagement

  Sewage management utility connection.

5.73.1 Detailed Description

Represents a residential building within the city.

The ResidentialBuilding class provides functionality for managing utility connections, displaying building status, and handling tax-related actions.

5.73.2 Constructor & Destructor Documentation

5.73.2.1 ResidentialBuilding()

```
ResidentialBuilding::ResidentialBuilding (
             int bedrooms,
             double price,
             std::shared_ptr< UtilityFlyweight > water,
             std::shared_ptr< UtilityFlyweight > power,
             std::shared_ptr< UtilityFlyweight > waste,
             std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a ResidentialBuilding with specified utilities and attributes.

Constructs a ResidentialBuilding with specified attributes and utilities.

Parameters

bedrooms	Number of bedrooms in the building.
cost	Price of the building.
water	Water supply connection.
power	Power supply connection.
waste	Waste management connection.
sewage	Sewage management connection.
bedrooms	Number of bedrooms in the building.
price	Price of the building.
water	Shared pointer to the water utility.
power	Shared pointer to the power utility.
waste	Shared pointer to the waste management utility.
sewage	Shared pointer to the sewage management utility.

5.73.3 Member Function Documentation

5.73.3.1 accept()

```
void ResidentialBuilding::accept (
             taxCollector * TC ) [override], [virtual]
```

Accepts a taxCollector visitor to collect taxes from this building.

Accepts a taxCollector visitor to collect taxes.

Parameters

TC	Pointer to the tax collector.
TC	Pointer to the taxCollector visiting the building.

Implements CityComponent.

Here is the call graph for this function:

5.73.3.2 displayStatus()

```
void ResidentialBuilding::displayStatus ( )  [override], [virtual]
```

Displays the status of the residential building.

Displays the status of the residential building, including utility connections and tax status.

Implements CityComponent.

Here is the call graph for this function:

5.73.3.3 getBuildingType()

```
std::string ResidentialBuilding::getBuildingType ( ) const  [inline], [override], [virtual]
```

Gets the building type.

Returns

A string representing the type of the building.

Implements CityComponent.

Reimplemented in Townhouse.

Here is the caller graph for this function:

5.73.3.4 getDisplayColor()

```
virtual std::string ResidentialBuilding::getDisplayColor ( ) const  [inline], [virtual]
```

Gets the display color based on utility coverage.

Returns

ANSI color code string for the building's utility coverage status.

Here is the call graph for this function:

5.73.3.5 getDisplaySymbol()

```
virtual char ResidentialBuilding::getDisplaySymbol ( ) const  [pure virtual]
```

Gets the symbol used for map display.

Returns

A character symbol representing the building.

Implemented in Townhouse, House, Flat, and Estate.

5.73.3.6 getPrice()

```
double ResidentialBuilding::getPrice ( ) const  [inline]
```

Gets the price of the building.

Returns

The price of the building.

Here is the caller graph for this function:

5.73.3.7 getUtilityCoverage()

```
int ResidentialBuilding::getUtilityCoverage ( ) const  [inline]
```

Calculates the utility coverage for this building.

Returns

The number of connected utilities.

Here is the caller graph for this function:

5.73.3.8 hasUtilities()

```
bool ResidentialBuilding::hasUtilities ( ) const  [inline]
```

Checks if the building has all essential utilities connected.

Returns

True if all utilities are connected, false otherwise.

Here is the caller graph for this function:

5.73.3.9 isTaxPaid()

```
bool ResidentialBuilding::isTaxPaid ( ) const
```

Checks if taxes have been paid.

Checks if the tax has been paid for this building.

Returns

True if taxes are paid, false otherwise.

True if the tax is paid, false otherwise.

5.73.3.10 setPowerSupply()

```
void ResidentialBuilding::setPowerSupply (
            std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

5.73.3.11 setSewageManagement()

```
void ResidentialBuilding::setSewageManagement (
            std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

5.73.3.12 setTaxPaid()

```
void ResidentialBuilding::setTaxPaid (
            bool paid )
```

Sets the tax-paid status.

Sets the tax-paid status and affects NPC happiness if applicable.

Parameters

paid	True if tax has been paid, false otherwise.

Here is the call graph for this function: Here is the caller graph for this function:

5.73.3.13 setWasteManagement()

```
void ResidentialBuilding::setWasteManagement (
            std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

5.73.3.14 setWaterSupply()

```
void ResidentialBuilding::setWaterSupply (
            std::shared_ptr< UtilityFlyweight > utility )  [inline]
```

5.73.4 Member Data Documentation

5.73.4.1 bedrooms

```
int ResidentialBuilding::bedrooms  [protected]
```

Number of bedrooms in the building.

5.73.4.2 powerSupply

```
std::shared_ptr<UtilityFlyweight> ResidentialBuilding::powerSupply  [protected]
```

Power utility connection.

5.73.4.3 price

```
double ResidentialBuilding::price  [protected]
```

Price of the residential building.

5.73.4.4 sewageManagement

```
std::shared_ptr<UtilityFlyweight> ResidentialBuilding::sewageManagement  [protected]
```

Sewage management utility connection.

5.73.4.5 taxPaid

```
bool ResidentialBuilding::taxPaid [protected]
```

Flag indicating whether taxes are paid.

5.73.4.6 wasteManagement

```
std::shared_ptr<UtilityFlyweight> ResidentialBuilding::wasteManagement [protected]
```

Waste management utility connection.

5.73.4.7 waterSupply

```
std::shared_ptr<UtilityFlyweight> ResidentialBuilding::waterSupply [protected]
```

Water utility connection.

The documentation for this class was generated from the following files:

- COS214-Poject/src/ResidentialBuilding.h
- COS214-Poject/src/ResidentialBuilding.cpp

5.74 ResidentialBuildingFactory Class Reference

Abstract Factory class for creating residential buildings.

```
#include <ResidentialBuildingFactory.h>
```

Inheritance diagram for ResidentialBuildingFactory:

Collaboration diagram for ResidentialBuildingFactory:

Public Member Functions

- virtual std::unique_ptr< ResidentialBuilding > createBuilding ()=0

  Creates a residential building.
- virtual ∼ResidentialBuildingFactory ()

  Virtual destructor for ResidentialBuildingFactory.

5.74.1 Detailed Description

Abstract Factory class for creating residential buildings.

The ResidentialBuildingFactory class provides an interface for creating residential building objects. Concrete factories derived from this class will implement the specific creation logic for different types of residential buildings.

5.74.2 Constructor & Destructor Documentation

5.74.2.1 ∼ResidentialBuildingFactory()

```
virtual ResidentialBuildingFactory::∼ResidentialBuildingFactory ( )  [inline], [virtual]
```

Virtual destructor for ResidentialBuildingFactory.

Ensures proper cleanup of derived classes.

5.74.3 Member Function Documentation

5.74.3.1 createBuilding()

```
virtual std::unique_ptr<ResidentialBuilding> ResidentialBuildingFactory::createBuilding ( )
[pure virtual]
```

Creates a residential building.

Returns

A unique pointer to a ResidentialBuilding.

This function is intended to be overridden in derived classes to create specific types of residential buildings.

Implemented in TownhouseFactory, HouseFactory, FlatFactory, and EstateFactory.

The documentation for this class was generated from the following file:

- COS214-Poject/src/ResidentialBuildingFactory.h

5.75 ResourceFactory Class Reference

Abstract Factory class for creating resources.

```
#include <ResourceFactory.h>
```

Inheritance diagram for ResourceFactory:

Collaboration diagram for ResourceFactory:

Public Member Functions

- ResourceFactory ()=default

  Default constructor for ResourceFactory.
- virtual ∼ResourceFactory ()=default

  Virtual destructor for ResourceFactory, allowing derived classes to clean up resources.
- virtual std::unique_ptr< IncomeResourceProduct > createIncomeR (int quantity)=0

  Creates an income-generating resource.
- virtual std::unique_ptr< ConstructionResourceProduct > createConstructionR (int quantity)=0

  Creates a construction resource.

5.75.1 Detailed Description

Abstract Factory class for creating resources.

The ResourceFactory class defines the interface for creating income-generating and construction-based resources. Derived classes will implement the specific creation logic for different types of resources.

5.75.2 Constructor & Destructor Documentation

5.75.2.1 ResourceFactory()

```
ResourceFactory::ResourceFactory ( )  [default]
```

Default constructor for ResourceFactory.

5.75.2.2 ∼ResourceFactory()

```
virtual ResourceFactory::∼ResourceFactory ( )  [virtual], [default]
```

Virtual destructor for ResourceFactory, allowing derived classes to clean up resources.

5.75.3 Member Function Documentation

5.75.3.1 createConstructionR()

```
virtual std::unique_ptr<ConstructionResourceProduct> ResourceFactory::createConstructionR (
            int quantity )  [pure virtual]
```

Creates a construction resource.

Parameters

quantity	The quantity of the construction resource to create.

Returns

A unique pointer to a ConstructionResourceProduct.

Implemented in StoneFactory, PlantFactory, MetalFactory, and ChemicalFactory.

5.75.3.2 createIncomeR()

```
virtual std::unique_ptr<IncomeResourceProduct> ResourceFactory::createIncomeR (
            int quantity )  [pure virtual]
```

Creates an income-generating resource.

Parameters

quantity	The quantity of the income resource to create.

Returns

A unique pointer to an IncomeResourceProduct.

Implemented in StoneFactory, PlantFactory, MetalFactory, and ChemicalFactory.

The documentation for this class was generated from the following file:

- COS214-Poject/src/ResourceFactory.h

5.76 ResourceProcessor Class Reference

Abstract base class for processing and managing resources.

```
#include <ResourceProcessor.h>
```

Inheritance diagram for ResourceProcessor:

Collaboration diagram for ResourceProcessor:

Public Member Functions

- virtual void process (int amount)=0

  Processes a specified amount of resource.
- virtual void store (int amount)=0

  Stores a specified amount of resource.
- virtual void display () const =0

  Displays the current status of the resource processor.
- virtual int getCurrentStorage () const =0

  Retrieves the current amount of resource in storage.
- virtual ∼ResourceProcessor ()=default

  Virtual destructor for ResourceProcessor, allowing derived classes to clean up resources.

5.76.1 Detailed Description

Abstract base class for processing and managing resources.

The ResourceProcessor class defines the interface for processing, storing, and displaying resource-related information. Derived classes will implement the specific processing and storage logic for different types of resources.

5.76.2 Constructor & Destructor Documentation

5.76.2.1 ∼ResourceProcessor()

```
virtual ResourceProcessor::∼ResourceProcessor ( )  [virtual], [default]
```

Virtual destructor for ResourceProcessor, allowing derived classes to clean up resources.

5.76.3 Member Function Documentation

5.76.3.1 display()

```
virtual void ResourceProcessor::display ( ) const  [pure virtual]
```

Displays the current status of the resource processor.

This function provides information about the processor's current state, typically including storage and processing details.

Implemented in IncomeResourceProcessor, and ConstructionResourceProcessor.

5.76.3.2 getCurrentStorage()

```
virtual int ResourceProcessor::getCurrentStorage ( ) const  [pure virtual]
```

Retrieves the current amount of resource in storage.

Returns

int The current storage level.

Implemented in IncomeResourceProcessor, and ConstructionResourceProcessor.

5.76.3.3 process()

```
virtual void ResourceProcessor::process (
             int amount )  [pure virtual]
```

Processes a specified amount of resource.

Parameters

amount	The amount of resource to process.

Implemented in IncomeResourceProcessor, and ConstructionResourceProcessor.

5.76.3.4 store()

```
virtual void ResourceProcessor::store (
             int amount )  [pure virtual]
```

Stores a specified amount of resource.

Parameters

amount	The amount of resource to store.

Implemented in IncomeResourceProcessor, and ConstructionResourceProcessor.

The documentation for this class was generated from the following file:

- COS214-Poject/src/ResourceProcessor.h

5.77 ResourceRequirement Struct Reference

Defines the resource requirements for constructing a building.

```
#include <BuildingRequirements.h>
```

Collaboration diagram for ResourceRequirement:

Public Member Functions

- ResourceRequirement (int w=0, int st=0, int se=0, int c=0)

  Constructor to initialize resource requirements.

Public Attributes

- int wood
- int stone
- int steel
- int concrete

5.77.1 Detailed Description

Defines the resource requirements for constructing a building.

This struct holds the amount of each resource (wood, stone, steel, concrete) needed to construct a specific type of building.

5.77.2 Constructor & Destructor Documentation

5.77.2.1 ResourceRequirement()

```
ResourceRequirement::ResourceRequirement (
              int w = 0,
              int st = 0,
              int se = 0,
              int c = 0 )  [inline]
```

Constructor to initialize resource requirements.

Parameters

w	Amount of wood required (default: 0).
st	Amount of stone required (default: 0).
se	Amount of steel required (default: 0).
c	Amount of concrete required (default: 0).

5.77.3 Member Data Documentation

5.77.3.1 concrete

```
int ResourceRequirement::concrete
```

Amount of concrete required

5.77.3.2 steel

```
int ResourceRequirement::steel
```

Amount of steel required

5.77.3.3 stone

```
int ResourceRequirement::stone
```

Amount of stone required

5.77.3.4 wood

```
int ResourceRequirement::wood
```

Amount of wood required

The documentation for this struct was generated from the following file:

- COS214-Poject/src/BuildingRequirements.h

5.78 MapGrid::ResourceSpot Struct Reference

Collaboration diagram for MapGrid::ResourceSpot:

Public Attributes

- std::shared_ptr< CityComponent > resource
- bool discovered

5.78.1 Member Data Documentation

5.78.1.1 discovered

```
bool MapGrid::ResourceSpot::discovered
```

5.78.1.2 resource

```
std::shared_ptr<CityComponent> MapGrid::ResourceSpot::resource
```

The documentation for this struct was generated from the following file:

- COS214-Poject/src/MapGrid.h

5.79 RevoltState Class Reference

Represents the state of NPCs when they are in revolt.

```
#include <RevoltState.h>
```

Inheritance diagram for RevoltState:

Collaboration diagram for RevoltState:

Public Member Functions

- void handle () override
  Executes actions specific to the revolt state.
- std::string getStateName () override
  Retrieves the name of this state.
- NPCState ∗ clone () const override
  Clones the current state, creating a new instance with the same properties.
- ∼RevoltState () override
  Destructor for the RevoltState class.

5.79.1 Detailed Description

Represents the state of NPCs when they are in revolt.

This state class handles the behavior and actions taken by NPCs when they are revolting due to low satisfaction or poor city conditions.

5.79.2 Constructor & Destructor Documentation

5.79.2.1 ∼RevoltState()

```
RevoltState::∼RevoltState ( )  [override], [default]
```

Destructor for the RevoltState class.

5.79.3 Member Function Documentation

5.79.3.1 clone()

```
NPCState ∗ RevoltState::clone ( ) const  [override], [virtual]
```

Clones the current state, creating a new instance with the same properties.

Creates a clone of the current RevoltState.

Returns

NPCState∗ A pointer to the cloned RevoltState object.

NPCState∗ A pointer to a new RevoltState object with the same properties.

Implements NPCState.

5.79.3.2 getStateName()

```
std::string RevoltState::getStateName ( )  [override], [virtual]
```

Retrieves the name of this state.

Returns the name of this state.

Returns

std::string The name of the state, typically "Revolt".

std::string The state name, "RevoltState".

Implements NPCState.

5.79.3.3 handle()

```
void RevoltState::handle ( )  [override], [virtual]
```

Executes actions specific to the revolt state.

Executes the handling logic for the revolt state.

This method defines what happens when NPCs are in revolt, potentially affecting the city's operations and resources.

This function calculates the chance of a revolt occurring based on the number of NPCs in the revolt state. If a revolt occurs, it reduces the city's production rate by a factor based on the number of revolting NPCs, up to a maximum limit.

Implements NPCState.

Here is the call graph for this function:

The documentation for this class was generated from the following files:

- COS214-Poject/src/RevoltState.h
- COS214-Poject/src/RevoltState.cpp

5.80 Roads Class Reference

A concrete implementation of Transport representing various types of roads.

```
#include <Roads.h>
```

Inheritance diagram for Roads:

Collaboration diagram for Roads:

Public Member Functions

- std::unique_ptr< Transport > clone () const override

    Creates a clone of this Roads object.
- double getSpeed () const override

    Gets the speed of the road.
- double getCost () const override

    Gets the construction cost of the road.
- double getMaintainanceCost () const override

    Gets the maintenance cost of the road.
- double getCapacity () const override

    Gets the traffic capacity of the road.
- std::string getType () const override

    Gets the type of road as a string.
- Roads ()

    Default constructor for Roads.
- ∼Roads ()

    Destructor for Roads.

Private Attributes

- int trafficCapacity
- roadType road

5.80.1 Detailed Description

A concrete implementation of Transport representing various types of roads.

The Roads class provides specific implementations for road-related attributes like traffic capacity and road type. It implements the Transport interface.

5.80.2 Constructor & Destructor Documentation

5.80.2.1 Roads()

```
Roads::Roads ( )  [inline]
```

Default constructor for Roads.

5.80.2.2 ∼Roads()

```
Roads::∼Roads ( )  [inline]
```

Destructor for Roads.

5.80.3 Member Function Documentation

5.80.3.1 clone()

```
std::unique_ptr< Transport > Roads::clone ( ) const  [override], [virtual]
```

Creates a clone of this Roads object.

Clones the current Roads object, creating a new instance with the same properties.

Returns

std::unique_ptr<Transport> A unique pointer to a cloned Roads object.

std::unique_ptr<Transport> A unique pointer to the cloned Roads object.

Implements Transport.

5.80.3.2 getCapacity()

```
double Roads::getCapacity ( ) const  [override], [virtual]
```

Gets the traffic capacity of the road.

Retrieves the capacity of the road.

Returns

double Capacity of the road.

double The traffic capacity of the road.

Implements Transport.

5.80.3.3 getCost()

```
double Roads::getCost ( ) const  [override], [virtual]
```

Gets the construction cost of the road.

Retrieves the construction cost of the road.

Returns

double Cost value for the road.

double The construction cost of the road.

Implements Transport.

5.80.3.4 getMaintainanceCost()

```
double Roads::getMaintainanceCost ( ) const  [override], [virtual]
```

Gets the maintenance cost of the road.

Retrieves the maintenance cost of the road.

Returns

double Maintenance cost for the road.

double The maintenance cost of the road.

Implements Transport.

5.80.3.5 getSpeed()

```
double Roads::getSpeed ( ) const  [override], [virtual]
```

Gets the speed of the road.

Retrieves the speed limit of the road.

Returns

double Speed value for the road.

double The speed limit on the road.

Implements Transport.

5.80.3.6 getType()

```
std::string Roads::getType ( ) const  [override], [virtual]
```

Gets the type of road as a string.

Retrieves the type of this transport as a string.

Returns

std::string Road type description.

std::string The string "Road" representing this transport type.

Implements Transport.

5.80.4 Member Data Documentation

5.80.4.1 road

```
roadType Roads::road  [private]
```

Type of the road, defined by roadType enum.

5.80.4.2 trafficCapacity

```
int Roads::trafficCapacity  [private]
```

Traffic capacity of the road.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Roads.h
- COS214-Poject/src/Roads.cpp

5.81 RoadsFactory Class Reference

Factory class for creating Road transport objects.

```
#include <RoadsFactory.h>
```

Inheritance diagram for RoadsFactory:

Collaboration diagram for RoadsFactory:

Public Member Functions

- std::unique_ptr< Transport > createTransport ()

  Creates a Road transport object.

5.81.1 Detailed Description

Factory class for creating Road transport objects.

RoadsFactory is a concrete implementation of the TransportationFactory. It provides a method to create Road transport objects, following the Factory Pattern.

5.81.2 Member Function Documentation

5.81.2.1 createTransport()

```
std::unique_ptr< Transport > RoadsFactory::createTransport ( )  [virtual]
```

Creates a Road transport object.

Creates a new Roads transport object.

Returns

std::unique_ptr<Transport> A unique pointer to a new Road transport object.

This method instantiates and returns a unique pointer to a new Roads object, which represents a type of transport infrastructure in the city simulation.

Returns

std::unique_ptr<Transport> A unique pointer to a Roads object.

Implements TransportationFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/RoadsFactory.h
- COS214-Poject/src/RoadsFactory.cpp

5.82 School Class Reference

```
#include <School.h>
```

Inheritance diagram for School:

Collaboration diagram for School:

Public Member Functions

- School (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_↩ ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage, const std::string &buildingStatus)
- ∼School () override=default
- void provideService () override

  Pure virtual function to provide a specific service.
- void displayStatus () override

  Pure virtual function to display the status of the public service.
- char getDisplaySymbol () const override

  Pure virtual function to get the display symbol for the public service on a map.

Private Attributes

- std::string status

Additional Inherited Members

5.82.1 Constructor & Destructor Documentation

5.82.1.1 School()

```
School::School (
          std::shared_ptr< UtilityFlyweight > water,
          std::shared_ptr< UtilityFlyweight > power,
          std::shared_ptr< UtilityFlyweight > waste,
          std::shared_ptr< UtilityFlyweight > sewage,
          const std::string & buildingStatus )
```

5.82.1.2 ∼School()

```
School::∼School ( ) [override], [default]
```

5.82.2 Member Function Documentation

5.82.2.1 displayStatus()

```
void School::displayStatus ( )  [override], [virtual]
```

Pure virtual function to display the status of the public service.

Implements PublicService.

Here is the call graph for this function:

5.82.2.2 getDisplaySymbol()

```
char School::getDisplaySymbol ( ) const  [override], [virtual]
```

Pure virtual function to get the display symbol for the public service on a map.

Implements PublicService.

5.82.2.3 provideService()

```
void School::provideService ( )  [override], [virtual]
```

Pure virtual function to provide a specific service.

This function must be implemented by derived classes.

Implements PublicService.

Here is the call graph for this function:

5.82.3 Member Data Documentation

5.82.3.1 status

```
std::string School::status  [private]
```

The documentation for this class was generated from the following files:

- COS214-Poject/src/School.h
- COS214-Poject/src/School.cpp

5.83 SewageSystem Class Reference

A concrete UtilityFlyweight that represents a sewage management utility.

`#include <SewageSystem.h>`

Inheritance diagram for SewageSystem:

Collaboration diagram for SewageSystem:

Public Member Functions

- SewageSystem ()
  Default constructor for SewageSystem.
- SewageSystem (const std::string &n, double c, int cap, double radius, bool operational, int l, int consumption, const std::map< std::string, int > &resources)
  Parameterized constructor for SewageSystem.
- void connect (double distance)
  Connects the sewage system to a building within a certain distance.
- std::shared_ptr< UtilityFlyweight > clone () const
  Clones the SewageSystem instance.

Additional Inherited Members

5.83.1 Detailed Description

A concrete UtilityFlyweight that represents a sewage management utility.

The SewageSystem class models a sewage utility, which provides sewage handling services for buildings within a defined range. It offers specific functionality such as connecting to nearby buildings and creating copies of itself.

5.83.2 Constructor & Destructor Documentation

5.83.2.1 SewageSystem() [1/2]

```
SewageSystem::SewageSystem ( )
```

Default constructor for SewageSystem.

Initializes a SewageSystem instance with preset attributes, including cost, capacity, and effect radius.

Initializes a SewageSystem instance with preset attributes:

- Name: "Sewage System"
- Cost: 175.0
- Capacity: 40 buildings
- Radius: 2.5 units
- Operational: true
- Level: 1
- Consumption: 8 resource units
- Resources: {"Steel": 30, "Plastic": 40, "Concrete": 20}

5.83.2.2 SewageSystem() [2/2]

```
SewageSystem::SewageSystem (
            const std::string & n,
            double c,
            int cap,
            double radius,
            bool operational,
            int l,
            int consumption,
            const std::map< std::string, int > & resources )
```

Parameterized constructor for SewageSystem.

Initializes a SewageSystem instance with specified attributes, including name, cost, capacity, effect radius, operational status, level, resource consumption, and required resources.

Parameters

n	Name of the sewage system.
c	Cost to build the sewage system.
cap	Capacity to serve buildings.
radius	Coverage radius of the sewage system.
operational	Operational status of the system.
l	Level of the sewage system.
consumption	Resources consumed per cycle.
resources	Map of resources needed for construction.

Initializes a SewageSystem instance with specified attributes.

Parameters

n	Name of the sewage system.
c	Cost of the sewage system.
cap	Capacity of the sewage system to serve buildings.
radius	Coverage radius of the sewage system.
operational	Initial operational status.
l	Level of the sewage system.
consumption	Resources consumed per cycle.
resources	Map of resources required for construction.

5.83.3 Member Function Documentation

5.83.3.1 clone()

```
std::shared_ptr< UtilityFlyweight > SewageSystem::clone ( ) const  [virtual]
```

Clones the SewageSystem instance.

Clones the current SewageSystem instance.

Implements the Prototype pattern to create a copy of the current SewageSystem instance.

Returns

std::shared_ptr<UtilityFlyweight> Shared pointer to the cloned SewageSystem.

Creates a copy of the SewageSystem instance, implementing the Prototype pattern.

Returns

std::shared_ptr<UtilityFlyweight> A shared pointer to the cloned SewageSystem instance.

Implements UtilityFlyweight.

5.83.3.2 connect()

```
void SewageSystem::connect (
             double distance )  [virtual]
```

Connects the sewage system to a building within a certain distance.

Connects the sewage system to a building within a specified distance.

Checks if the building is within range and, if so, connects to it, otherwise notifies that it is out of range.

Parameters

distance	Distance to the building attempting to connect.

If the building is within range, connects to it; otherwise, notifies that it is out of range.

Parameters

distance	Distance to the building attempting to connect.

Implements UtilityFlyweight.

The documentation for this class was generated from the following files:

- COS214-Poject/src/SewageSystem.h
- COS214-Poject/src/SewageSystem.cpp

5.84 Shops Class Reference

Represents a commercial building of type Shops.

```
#include <Shops.h>
```

Inheritance diagram for Shops:

Collaboration diagram for Shops:

Public Member Functions

- Shops ()

  Default constructor for Shops.
- Shops (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std::shared_↩ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Constructs a Shops instance with utility connections.
- char getDisplaySymbol () const override

  Gets the display symbol for Shops.
- std::string getBuildingType () const override

  Retrieves the building type as a string.

Additional Inherited Members

5.84.1 Detailed Description

Represents a commercial building of type Shops.

The Shops class inherits from CommercialBuilding and represents a specific type of commercial building. It includes methods for retrieving building details and display symbols.

5.84.2 Constructor & Destructor Documentation

5.84.2.1 Shops() [1/2]

```
Shops::Shops ( )
```

Default constructor for Shops.

Initializes a Shops instance with default utilities (no utility connections).

Initializes a Shops instance with a default cost and no utility connections.

5.84.2.2 Shops() [2/2]

```
Shops::Shops (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a Shops instance with utility connections.

Constructs a Shops instance with specified utility connections.

Parameters

water	Shared pointer to a water utility.
power	Shared pointer to a power utility.
waste	Shared pointer to a waste utility.
sewage	Shared pointer to a sewage utility.

This constructor initializes a Shops instance with a specified cost and connections to various utilities (water, power, waste, and sewage).

Parameters

water	Shared pointer to a water utility.
power	Shared pointer to a power utility.
waste	Shared pointer to a waste utility.
sewage	Shared pointer to a sewage utility.

5.84.3 Member Function Documentation

5.84.3.1 getBuildingType()

```
std::string Shops::getBuildingType ( ) const  [inline], [override], [virtual]
```

Retrieves the building type as a string.

Returns

std::string The building type, "Shops".

Reimplemented from CommercialBuilding.

5.84.3.2 getDisplaySymbol()

```
char Shops::getDisplaySymbol ( ) const  [inline], [override], [virtual]
```

Gets the display symbol for Shops.

Returns

char The display symbol representing Shops.

Implements CommercialBuilding.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Shops.h
- COS214-Poject/src/Shops.cpp

5.85 ShopsFactory Class Reference

Factory class for creating shop buildings.

`#include <ShopsFactory.h>`

Inheritance diagram for ShopsFactory:

Collaboration diagram for ShopsFactory:

Public Member Functions

- std::unique_ptr< CommercialBuilding > createBuilding ()

  Creates a new shop building.

5.85.1 Detailed Description

Factory class for creating shop buildings.

ShopsFactory is a concrete implementation of CommercialFactory that creates shop-type commercial buildings. This factory is used to encapsulate the creation of shop buildings, adhering to the Abstract Factory design pattern.

5.85.2 Member Function Documentation

5.85.2.1 createBuilding()

```
std::unique_ptr< CommercialBuilding > ShopsFactory::createBuilding ( )  [virtual]
```

Creates a new shop building.

This method creates and returns a unique pointer to a shop-type commercial building.

Returns

std::unique_ptr<CommercialBuilding> A unique pointer to the created shop building.

This method instantiates a Shops object, which represents a commercial building of type shop, and returns a unique pointer to the created building.

Returns

std::unique_ptr<CommercialBuilding> A unique pointer to the created Shops object.

Implements CommercialFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/ShopsFactory.h
- COS214-Poject/src/ShopsFactory.cpp

5.86 SlowCollectionStrategy Class Reference

Collection strategy for slow collection rate.

```
#include <NPCSystem.h>
```

Inheritance diagram for SlowCollectionStrategy:

Collaboration diagram for SlowCollectionStrategy:

Public Member Functions

- int collect (int amount) const override

  Pure virtual function to collect resources based on a strategy.
- int collect (int baseAmount)

  Collects resources based on a slow strategy.

5.86.1 Detailed Description

Collection strategy for slow collection rate.

Strategy for slow resource collection.

Implements a 50% collection efficiency.

This class defines a slow collection strategy that collects a specified amount of resources based on a base amount.

5.86.2 Member Function Documentation

5.86.2.1 collect() [1/2]

```
int SlowCollectionStrategy::collect (
             int baseAmount ) const  [inline], [override], [virtual]
```

Pure virtual function to collect resources based on a strategy.

Derived classes implement this method to adjust the amount collected based on a specific collection strategy.

Parameters

baseAmount	The base amount to be collected.

Returns

The adjusted amount collected based on the strategy.

< Returns half of the input amount.

Implements CollectionStrategy.

5.86.2.2 collect() [2/2]

```
int SlowCollectionStrategy::collect (
              int baseAmount )
```

Collects resources based on a slow strategy.

This method reduces the base amount to simulate a slower collection rate.

Parameters

baseAmount	The base amount of resources to be collected.

Returns

int Adjusted amount of resources collected according to the slow strategy.

This method simulates a slow collection rate by halving the base amount.

Parameters

baseAmount	The base amount of resources to be collected.

Returns

int Adjusted amount of resources collected according to the slow strategy.

The documentation for this class was generated from the following files:

- COS214-Poject/src/NPCSystem.h
- COS214-Poject/src/SlowCollectionStrategy.h
- COS214-Poject/src/SlowCollectionStrategy.cpp

5.87 Steel Class Reference

Represents the Steel resource in the city simulation.

```
#include <Steel.h>
```

Inheritance diagram for Steel:

Collaboration diagram for Steel:

Public Member Functions

- Steel (int quantity, int unitCost)

  Constructs a Steel object with specified quantity and unit cost.
- ∼Steel ()

  Destructor for Steel.
- void displayStatus () override

  Displays the current status of the Steel resource.
- std::string getBuildingType () const

  Gets the type of building resource.

5.87.1 Detailed Description

Represents the Steel resource in the city simulation.

This class is a concrete product derived from ConstructionResourceProduct. It encapsulates the properties and behavior of Steel used in construction.

5.87.2 Constructor & Destructor Documentation

5.87.2.1 Steel()

```
Steel::Steel (
            int quantity,
            int unitCost )
```

Constructs a Steel object with specified quantity and unit cost.

Initializes the Steel resource with a given quantity and cost per unit.

Parameters

quantity	The initial quantity of Steel.
unitCost	The cost per unit of Steel.

5.87.2.2 ∼Steel()

```
Steel::∼Steel ( )
```

Destructor for Steel.

Cleans up any resources or states when a Steel object is destroyed.

5.87.3 Member Function Documentation

5.87.3.1 displayStatus()

```
void Steel::displayStatus ( )  [override], [virtual]
```

Displays the current status of the Steel resource.

Provides detailed information about the quantity, unit cost, and total cost of the Steel.

Reimplemented from ConstructionResourceProduct.

Here is the call graph for this function:

5.87.3.2 getBuildingType()

```
std::string Steel::getBuildingType ( ) const  [inline], [virtual]
```

Gets the type of building resource.

Returns

The type of the building resource as a string, in this case, "Steel".

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Steel.h
- COS214-Poject/src/Steel.cpp

5.88 Stone Class Reference

Represents a construction resource product of type Stone.

```
#include <Stone.h>
```

Inheritance diagram for Stone:

Collaboration diagram for Stone:

Public Member Functions

- Stone (int quantity, int unitCost)

  Constructs a Stone object with a specified quantity and unit cost.
- ∼Stone ()

  Destructor for Stone.
- void displayStatus () override

  Displays the current status of the Stone resource.
- std::string getBuildingType () const override

  Gets the type of building resource.

5.88.1 Detailed Description

Represents a construction resource product of type Stone.

Inherits from `ConstructionResourceProduct` and provides additional functionality specific to stone resources, including cost and quantity tracking.

5.88.2 Constructor & Destructor Documentation

5.88.2.1 Stone()

```
Stone::Stone (
              int quantity,
              int unitCost )
```

Constructs a Stone object with a specified quantity and unit cost.

Constructs a Stone object with specified quantity and unit cost.

Parameters

quantity	The initial quantity of Stone.
unitCost	The cost per unit of Stone.

Initializes the Stone resource with a name, quantity, and unit cost.

Parameters

quantity	The initial quantity of Stone.
unitCost	The cost per unit of Stone.

5.88.2.2 ∼Stone()

```
Stone::∼Stone ( )
```

Destructor for Stone.

Outputs a message to the console when the Stone object is destroyed.

This is where any specific cleanup would be done for Stone objects.

5.88.3 Member Function Documentation

5.88.3.1 displayStatus()

```
void Stone::displayStatus ( )  [override], [virtual]
```

Displays the current status of the Stone resource.

Overrides the `displayStatus` method from `ConstructionResourceProduct` to provide specific information related to Stone resources.

Outputs detailed information about the Stone resource, including its type, quantity, unit cost, and total cost.

Reimplemented from ConstructionResourceProduct.

Here is the call graph for this function:

5.88.3.2 getBuildingType()

```
std::string Stone::getBuildingType ( ) const  [inline], [override], [virtual]
```

Gets the type of building resource.

Returns

A string indicating the resource type, "Stone".

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Stone.h
- COS214-Poject/src/Stone.cpp

5.89 StoneFactory Class Reference

Factory class for creating stone resources, providing both income and construction products.

`#include <StoneFactory.h>`

Inheritance diagram for StoneFactory:

Collaboration diagram for StoneFactory:

Public Member Functions

- StoneFactory ()

  Constructor for StoneFactory.
- ∼StoneFactory ()

  Destructor for StoneFactory.
- std::unique_ptr< IncomeResourceProduct > createIncomeR (int quantity) override

  Creates an income resource product representing stone.
- std::unique_ptr< ConstructionResourceProduct > createConstructionR (int quantity) override

  Creates a construction resource product representing stone.

5.89.1 Detailed Description

Factory class for creating stone resources, providing both income and construction products.

The `StoneFactory` class inherits from `ResourceFactory` and overrides methods to create `IncomeResourceProduct` and `ConstructionResourceProduct` objects representing stone resources.

5.89.2 Constructor & Destructor Documentation

5.89.2.1 StoneFactory()

```
StoneFactory::StoneFactory ( )
```

Constructor for StoneFactory.

Initializes any necessary components for the factory to create stone products.

Initializes the StoneFactory and outputs a creation message to the console.

5.89.2.2 ∼StoneFactory()

```
StoneFactory::∼StoneFactory ( )
```

Destructor for StoneFactory.

Cleans up any allocated resources for the factory.

Outputs a deletion message to the console upon factory destruction.

5.89.3 Member Function Documentation

5.89.3.1 createConstructionR()

```
std::unique_ptr< ConstructionResourceProduct > StoneFactory::createConstructionR (
            int quantity )  [override], [virtual]
```

Creates a construction resource product representing stone.

Creates a construction resource.

Parameters

quantity	The quantity of stone for the construction resource.

Returns

A unique pointer to a `ConstructionResourceProduct` representing stone.

This method generates a resource product for construction, represented as Stone, with the specified quantity and a default unit price.

Parameters

quantity	The quantity of the construction resource to be created.

Returns

A unique pointer to a `ConstructionResourceProduct` representing Stone.

Implements ResourceFactory.

5.89.3.2 createIncomeR()

```
std::unique_ptr< IncomeResourceProduct > StoneFactory::createIncomeR (
            int quantity )  [override], [virtual]
```

Creates an income resource product representing stone.

Creates an income-generating resource.

Parameters

quantity	The quantity of stone for the income resource.

Returns

A unique pointer to an `IncomeResourceProduct` representing stone.

This method generates a resource product for income, represented as Diamonds, with the specified quantity and a default unit price.

Parameters

quantity	The quantity of the income-generating resource to be created.

Returns

A unique pointer to an `IncomeResourceProduct` representing Diamonds.

Implements ResourceFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/StoneFactory.h
- COS214-Poject/src/StoneFactory.cpp

5.90 taxCollector Class Reference

Abstract class representing a tax collector that visits various building types to collect taxes.

```
#include <taxCollector.h>
```

Inheritance diagram for taxCollector:

Collaboration diagram for taxCollector:

Public Member Functions

- taxCollector ()

    Default constructor.
- virtual ~taxCollector ()=default

    Virtual destructor.
- virtual void visit (ResidentialBuilding ∗RB)=0

    Visit a residential building to collect taxes.
- virtual void visit (CommercialBuilding ∗CB)=0

    Visit a commercial building to collect taxes.
- virtual void visit (ZoneComposite ∗Rzone)=0

    Visit a residential zone composite to collect taxes.

Protected Member Functions

- Government & getGovernment ()

    Provides access to the `Government` singleton instance.

5.90.1 Detailed Description

Abstract class representing a tax collector that visits various building types to collect taxes.

The `taxCollector` class defines an interface for visiting different types of buildings and zones in the city to collect taxes. Each specific type of building or zone has its own `visit` method. The class also provides access to the `Government` singleton, which allows interaction with central government functions or resources.

5.90.2 Constructor & Destructor Documentation

5.90.2.1 taxCollector()

```
taxCollector::taxCollector ( )  [inline]
```

Default constructor.

Initializes the taxCollector without needing to store a pointer to the `Government` instance.

5.90.2.2 ∼taxCollector()

```
virtual taxCollector::∼taxCollector ( )  [virtual], [default]
```

Virtual destructor.

Allows derived classes to clean up resources properly.

5.90.3 Member Function Documentation

5.90.3.1 getGovernment()

```
Government& taxCollector::getGovernment ( )  [inline], [protected]
```

Provides access to the `Government` singleton instance.

Returns

A reference to the `Government` singleton.

Here is the call graph for this function:

5.90.3.2 visit() [1/3]

```
virtual void taxCollector::visit (
            CommercialBuilding * CB )  [pure virtual]
```

Visit a commercial building to collect taxes.

Parameters

CB	A pointer to the commercial building.

Implemented in ConcreteTaxCollector.

5.90.3.3 visit() [2/3]

```
virtual void taxCollector::visit (
             ResidentialBuilding * RB )  [pure virtual]
```

Visit a residential building to collect taxes.

Parameters

RB	A pointer to the residential building.

Implemented in ConcreteTaxCollector.

Here is the caller graph for this function:

5.90.3.4 visit() [3/3]

```
virtual void taxCollector::visit (
             ZoneComposite * Rzone )  [pure virtual]
```

Visit a residential zone composite to collect taxes.

Parameters

Rzone	A pointer to the residential zone composite.

Implemented in ConcreteTaxCollector.

The documentation for this class was generated from the following file:

- COS214-Poject/src/taxCollector.h

5.91 Townhouse Class Reference

A concrete class representing a townhouse, derived from ResidentialBuilding.

```
#include <Townhouse.h>
```

Inheritance diagram for Townhouse:

Collaboration diagram for Townhouse:

Public Member Functions

- Townhouse ()

  Default constructor for Townhouse.
- Townhouse (std::shared_ptr< UtilityFlyweight > water, std::shared_ptr< UtilityFlyweight > power, std↵ ::shared_ptr< UtilityFlyweight > waste, std::shared_ptr< UtilityFlyweight > sewage)

  Constructs a Townhouse with specified utility connections.
- char getDisplaySymbol () const override

  Returns the display symbol for the townhouse.
- std::string getBuildingType () const override

  Returns the building type as a string.

Static Public Attributes

- static constexpr int BASE_COST = 200

Additional Inherited Members

5.91.1 Detailed Description

A concrete class representing a townhouse, derived from ResidentialBuilding.

The Townhouse class defines a residential building type with specific characteristics. It provides utility connections (e.g., water, power, waste, sewage) and has a display symbol.

5.91.2 Constructor & Destructor Documentation

5.91.2.1 Townhouse() [1/2]

```
Townhouse::Townhouse ( )
```

Default constructor for Townhouse.

Constructs a Townhouse with a default capacity of 4 occupants and a cost of 200.00, with no initial utility connections.

5.91.2.2 Townhouse() [2/2]

```
Townhouse::Townhouse (
            std::shared_ptr< UtilityFlyweight > water,
            std::shared_ptr< UtilityFlyweight > power,
            std::shared_ptr< UtilityFlyweight > waste,
            std::shared_ptr< UtilityFlyweight > sewage )
```

Constructs a Townhouse with specified utility connections.

Parameterized constructor for Townhouse.

Parameters

water	A shared pointer to the water utility connection.
power	A shared pointer to the power utility connection.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage system utility.

Constructs a Townhouse with specified utility connections and assigns it a capacity of 4 occupants and a cost of 200.00.

Parameters

water	A shared pointer to the water utility connection.
power	A shared pointer to the power utility connection.
waste	A shared pointer to the waste management utility.
sewage	A shared pointer to the sewage system utility.

5.91.3 Member Function Documentation

5.91.3.1 getBuildingType()

```
std::string Townhouse::getBuildingType ( ) const  [inline], [override], [virtual]
```

Returns the building type as a string.

Returns

std::string The type of the building ("Townhouse").

Reimplemented from ResidentialBuilding.

5.91.3.2 getDisplaySymbol()

```
char Townhouse::getDisplaySymbol ( ) const  [inline], [override], [virtual]
```

Returns the display symbol for the townhouse.

Returns

char The symbol representing a townhouse ('T').

Implements ResidentialBuilding.

5.91.4 Member Data Documentation

5.91.4.1 BASE_COST

```
constexpr int Townhouse::BASE_COST = 200  [static], [constexpr]
```

Base cost for constructing a townhouse.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Townhouse.h
- COS214-Poject/src/Townhouse.cpp

5.92 TownhouseFactory Class Reference

Factory class for creating Townhouse residential buildings.

```
#include <TownhouseFactory.h>
```

Inheritance diagram for TownhouseFactory:

Collaboration diagram for TownhouseFactory:

Public Member Functions

- std::unique_ptr< ResidentialBuilding > createBuilding () override

  Creates a new instance of a townhouse building.

5.92.1 Detailed Description

Factory class for creating Townhouse residential buildings.

This class implements the ResidentialBuildingFactory interface to produce instances of townhouses.

5.92.2 Member Function Documentation

5.92.2.1 createBuilding()

```
std::unique_ptr< ResidentialBuilding > TownhouseFactory::createBuilding ( ) [override], [virtual]
```

Creates a new instance of a townhouse building.

Implements the Factory Method to produce a specific type of residential building, which in this case is a townhouse.

Returns

std::unique_ptr<ResidentialBuilding> A unique pointer to the created townhouse.

This function overrides the createBuilding function in the ResidentialBuildingFactory interface. It creates and returns a unique pointer to a Townhouse instance.

Returns

std::unique_ptr<ResidentialBuilding> A unique pointer to the created townhouse.

Implements ResidentialBuildingFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/TownhouseFactory.h
- COS214-Poject/src/TownhouseFactory.cpp

5.93 Trains Class Reference

A concrete implementation of the Transport class representing a train.

```
#include <Trains.h>
```

Inheritance diagram for Trains:

Collaboration diagram for Trains:

Public Member Functions

- std::unique_ptr< Transport > clone () const override

  Creates a copy of the current Trains instance.
- double getSpeed () const override

  Retrieves the speed of the train.
- double getCost () const override

  Retrieves the cost associated with the train.
- double getMaintainanceCost () const override

  Retrieves the maintenance cost for the train.
- double getCapacity () const override

  Retrieves the capacity of the train.
- std::string getType () const override

  Returns the type of transport as a string.
- Trains ()

  Default constructor for Trains.
- ∼Trains ()

  Destructor for Trains.

Private Attributes

- int cargoCapacity

    Capacity of the train for carrying cargo.
- double railwayLength

    Total length of railway covered by the train.
- int numberOfStations

    Number of stations the train stops at.

5.93.1 Detailed Description

A concrete implementation of the Transport class representing a train.

The Trains class encapsulates the properties and behaviors of a train transport, including its capacity, speed, cost, and maintenance requirements.

5.93.2 Constructor & Destructor Documentation

5.93.2.1 Trains()

```
Trains::Trains ( )  [inline]
```

Default constructor for Trains.

5.93.2.2 ∼Trains()

```
Trains::∼Trains ( )  [inline]
```

Destructor for Trains.

5.93.3 Member Function Documentation

5.93.3.1 clone()

```
std::unique_ptr< Transport > Trains::clone ( ) const  [override], [virtual]
```

Creates a copy of the current Trains instance.

Creates a clone of the current Trains instance.

Implements the Prototype pattern, allowing for the creation of a new instance with the same properties as the current one.

Returns

std::unique_ptr<Transport> A unique pointer to the cloned Trains instance.

Implements the Prototype pattern to provide a new instance with the same properties as the existing Trains instance.

Returns

std::unique_ptr<Transport> A unique pointer to the cloned Trains instance.

Implements Transport.

5.93.3.2 getCapacity()

```
double Trains::getCapacity ( ) const  [override], [virtual]
```

Retrieves the capacity of the train.

Gets the cargo capacity of the train.

Returns

double Capacity of the train.

double Capacity of the train in terms of cargo.

Implements Transport.

5.93.3.3 getCost()

```
double Trains::getCost ( ) const  [override], [virtual]
```

Retrieves the cost associated with the train.

Gets the cost associated with the train.

Returns

double Cost of the train.

double Cost to acquire or build the train.

Implements Transport.

5.93.3.4 getMaintainanceCost()

```
double Trains::getMaintainanceCost ( ) const  [override], [virtual]
```

Retrieves the maintenance cost for the train.

Gets the maintenance cost of the train.

Returns

double Maintenance cost of the train.

double Ongoing maintenance cost of the train.

Implements Transport.

5.93.3.5 getSpeed()

```
double Trains::getSpeed ( ) const  [override], [virtual]
```

Retrieves the speed of the train.

Gets the speed of the train.

Returns

double Speed of the train.

Implements Transport.

5.93.3.6 getType()

```
std::string Trains::getType ( ) const  [override], [virtual]
```

Returns the type of transport as a string.

Gets the type of the transport.

Returns

std::string Type of transport, e.g., "Train".

std::string Type of transport, which is "Train".

Implements Transport.

5.93.4 Member Data Documentation

5.93.4.1 cargoCapacity

`int Trains::cargoCapacity [private]`

Capacity of the train for carrying cargo.

5.93.4.2 numberOfStations

`int Trains::numberOfStations [private]`

Number of stations the train stops at.

5.93.4.3 railwayLength

`double Trains::railwayLength [private]`

Total length of railway covered by the train.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Trains.h
- COS214-Poject/src/Trains.cpp

5.94 TrainsFactory Class Reference

Factory class for creating train transport instances.

`#include <TrainsFactory.h>`

Inheritance diagram for TrainsFactory:

Collaboration diagram for TrainsFactory:

Public Member Functions

- std::unique_ptr< Transport > createTransport ()
  Create a new train transport instance.

5.94.1 Detailed Description

Factory class for creating train transport instances.

TrainsFactory is a concrete factory that implements the TransportationFactory interface. This class is responsible for creating and returning instances of train transports using the Factory Method pattern.

5.94.2 Member Function Documentation

5.94.2.1 createTransport()

```
std::unique_ptr< Transport > TrainsFactory::createTransport ( )  [virtual]
```

Create a new train transport instance.

Creates a new instance of the Trains transport.

Implements the Factory Method to create a unique instance of a train transport. This method encapsulates the instantiation details of the transport and returns a pointer to the created object.

Returns

A unique pointer to a new Transport instance representing a train.

This method implements the Factory Method pattern by creating and returning a unique instance of the Trains class. The returned instance represents a train transport with all its specific properties.

Returns

std::unique_ptr<Transport> A unique pointer to the newly created Trains instance.

Implements TransportationFactory.

The documentation for this class was generated from the following files:

- COS214-Poject/src/TrainsFactory.h
- COS214-Poject/src/TrainsFactory.cpp

5.95 Transport Class Reference

Abstract class representing a generic transport.

```
#include <Transport.h>
```

Inheritance diagram for Transport:

Collaboration diagram for Transport:

Public Member Functions

- virtual double getCost () const =0
  Get the cost of the transport.
- virtual double getMaintainanceCost () const =0
  Get the maintenance cost of the transport.
- virtual double getSpeed () const =0
  Get the speed of the transport.
- virtual double getCapacity () const =0
  Get the capacity of the transport.
- virtual std::string getType () const =0
  Get the type of transport.
- virtual std::unique_ptr< Transport > clone () const =0
  Clone the transport object.
- virtual ∼Transport ()=default
  Virtual destructor for the Transport class.

5.95.1 Detailed Description

Abstract class representing a generic transport.

The Transport class defines a prototype for various types of transportation. Derived classes must implement the specific transportation details, such as cost, maintenance cost, speed, capacity, and type. The class also provides a clone method to support creating copies of transport objects.

5.95.2 Constructor & Destructor Documentation

5.95.2.1 ∼Transport()

```
virtual Transport::∼Transport ( )  [virtual], [default]
```

Virtual destructor for the Transport class.

Ensures proper cleanup of derived transport instances when deleted through a pointer to the base class.

5.95.3 Member Function Documentation

5.95.3.1 clone()

```
virtual std::unique_ptr<Transport> Transport::clone ( ) const  [pure virtual]
```

Clone the transport object.

Virtual function using the Prototype pattern to create a copy of the transport object. Derived classes should implement specific cloning logic.

Returns

A unique pointer to the cloned transport object.

Implemented in Trains, and Roads.

5.95.3.2 getCapacity()

```
virtual double Transport::getCapacity ( ) const  [pure virtual]
```

Get the capacity of the transport.

Pure virtual function to retrieve the capacity for transportation.

Returns

The capacity of the transport.

Implemented in Trains, and Roads.

5.95.3.3 getCost()

```
virtual double Transport::getCost ( ) const  [pure virtual]
```

Get the cost of the transport.

Pure virtual function to retrieve the cost associated with the transport.

Returns

The cost of the transport.

Implemented in Trains, and Roads.

5.95.3.4 getMaintainanceCost()

```
virtual double Transport::getMaintainanceCost ( ) const  [pure virtual]
```

Get the maintenance cost of the transport.

Pure virtual function to retrieve the ongoing maintenance cost.

Returns

The maintenance cost of the transport.

Implemented in Trains, and Roads.

5.95.3.5 getSpeed()

```
virtual double Transport::getSpeed ( ) const  [pure virtual]
```

Get the speed of the transport.

Pure virtual function to retrieve the speed.

Returns

The speed of the transport.

Implemented in Trains, and Roads.

5.95.3.6 getType()

```
virtual std::string Transport::getType ( ) const  [pure virtual]
```

Get the type of transport.

Pure virtual function to retrieve the type of transport as a string.

Returns

The type of transport.

Implemented in Trains, and Roads.

The documentation for this class was generated from the following file:

- COS214-Poject/src/Transport.h

5.96 TransportationFactory Class Reference

Abstract Factory class for creating transportation objects.

```
#include <TransportationFactory.h>
```

Inheritance diagram for TransportationFactory:

Collaboration diagram for TransportationFactory:

Public Member Functions

- virtual std::unique_ptr< Transport > createTransport ()=0

  Creates a transport object.
- virtual ∼TransportationFactory ()

  Virtual destructor for TransportationFactory.

5.96.1 Detailed Description

Abstract Factory class for creating transportation objects.

The TransportationFactory defines an interface for creating various types of Transport objects. Derived classes will implement the specific transport creation logic.

5.96.2 Constructor & Destructor Documentation

5.96.2.1 ∼TransportationFactory()

```
virtual TransportationFactory::∼TransportationFactory ( )  [inline], [virtual]
```

Virtual destructor for TransportationFactory.

Ensures proper cleanup of derived factory instances when deleted through a pointer to the base class.

5.96.3 Member Function Documentation

5.96.3.1 createTransport()

```
virtual std::unique_ptr<Transport> TransportationFactory::createTransport ( )  [pure virtual]
```

Creates a transport object.

This pure virtual function must be overridden by derived factory classes to instantiate specific types of transport objects.

Returns

A unique pointer to a Transport object.

Implemented in TrainsFactory, and RoadsFactory.

The documentation for this class was generated from the following file:

- COS214-Poject/src/TransportationFactory.h

5.97 UtilityFactory Class Reference

Factory class for creating and managing utility flyweight instances.

```
#include <UtilityFactory.h>
```

Collaboration diagram for UtilityFactory:

Public Member Functions

- std::shared_ptr< UtilityFlyweight > getUtility (const std::string &type)

  Retrieves a utility flyweight instance by its string name.
- std::shared_ptr< UtilityFlyweight > getUtilityByType (int type)

  Retrieves a utility flyweight instance by its numeric type.

Static Public Member Functions

- static std::string getUtilityName (int type)

  Gets the name of the utility based on its numeric type.
- static double getUtilityCost (int type)

  Gets the cost of the utility based on its numeric type.

Private Attributes

- std::unordered_map< std::string, std::shared_ptr< UtilityFlyweight > > flyweights

  Stores utility flyweight instances by name.

5.97.1 Detailed Description

Factory class for creating and managing utility flyweight instances.

5.97.2 Member Function Documentation

5.97.2.1 getUtility()

```
std::shared_ptr< UtilityFlyweight > UtilityFactory::getUtility (
            const std::string & type )
```

Retrieves a utility flyweight instance by its string name.

This method returns a shared pointer to the requested utility. If the utility doesn't exist, it is created and added to the map.

Parameters

type	The name of the utility (e.g., "PowerPlant", "WaterSupply").

Returns

A shared pointer to the requested UtilityFlyweight instance.

Exceptions

std::invalid_argument	if the utility type is invalid.

If a flyweight of the specified type already exists, it returns it. If not, it creates a new flyweight, adds it to the map, and returns a clone.

Parameters

type	The name of the utility (e.g., "PowerPlant", "WaterSupply").

Returns

A shared pointer to the requested UtilityFlyweight instance.

Exceptions

std::out_of_range	if the utility type is invalid.

Here is the caller graph for this function:

5.97.2.2 getUtilityByType()

```
std::shared_ptr< UtilityFlyweight > UtilityFactory::getUtilityByType (
              int type )
```

Retrieves a utility flyweight instance by its numeric type.

This method is useful for menu selection, allowing retrieval by integer type.

Parameters

type	The integer representing the utility type.

Returns

A shared pointer to the requested UtilityFlyweight instance.

Exceptions

std::out_of_range	if the type number is not recognized.

Allows retrieval of utilities by numeric type for ease of menu selection.

Parameters

type	The integer representing the utility type.

Returns

A shared pointer to the requested UtilityFlyweight instance.

Exceptions

std::out_of_range	if the utility type number is invalid.

Here is the call graph for this function: Here is the caller graph for this function:

5.97.2.3 getUtilityCost()

```
double UtilityFactory::getUtilityCost (
              int type ) [static]
```

Gets the cost of the utility based on its numeric type.

Parameters

type	The integer representing the utility type.

Returns

The cost of the utility as a double.

Here is the caller graph for this function:

5.97.2.4 getUtilityName()

```
std::string UtilityFactory::getUtilityName (
            int type ) [static]
```

Gets the name of the utility based on its numeric type.

Parameters

type	The integer representing the utility type.

Returns

A string containing the name of the utility.

Here is the caller graph for this function:

5.97.3 Member Data Documentation

5.97.3.1 flyweights

```
std::unordered_map<std::string, std::shared_ptr<UtilityFlyweight> > UtilityFactory::flyweights
[private]
```

Stores utility flyweight instances by name.

The documentation for this class was generated from the following files:

- COS214-Poject/src/UtilityFactory.h
- COS214-Poject/src/UtilityFactory.cpp

5.98 UtilityFlyweight Class Reference

Abstract base class for utility components, providing shared functionality.

```
#include <UtilityFlyweight.h>
```

Inheritance diagram for UtilityFlyweight:

Collaboration diagram for UtilityFlyweight:

Public Member Functions

- UtilityFlyweight ()=default

  Default constructor for UtilityFlyweight.
- UtilityFlyweight (const std::string &n, double c, int cap, double radius, bool operational, int l, int consumption, const std::map< std::string, int > &resources)

  Parameterized constructor for UtilityFlyweight.
- virtual void connect (double distance)=0

  Connect the utility to a component within a specific distance.
- virtual ∼UtilityFlyweight ()=default

  Virtual destructor for UtilityFlyweight.
- virtual std::shared_ptr< UtilityFlyweight > clone () const =0

  Creates a clone of the utility, implementing the Prototype pattern.
- std::string getName () const

  Getter for the utility's name.
- bool getOperationalStatus () const

  Getter for the operational status.
- double getCost () const

  Getter for the monetary cost.
- double getEffectRadius () const

  Getter for the area of effect radius.
- int getCapacity () const

  Getter for the maximum capacity.
- int getLevel () const

  Getter for the utility's current level.
- virtual int getResourceConsumption () const

  Getter for the resource consumption rate.
- std::map< std::string, int > getResourceNeeds () const

  Getter for resource requirements map.
- void activate ()

  Activates the utility, setting it to operational.
- void deactivate ()

  Deactivates the utility, setting it to non-operational.
- void displayStatus ()

  Displays the current status of the utility.
- double getRadius () const

  Gets the effect radius of the utility.
- std::string getBuildingType () const override

  Gets the type of the building, overridden from CityComponent.
- void accept (taxCollector ∗) override

  Accepts a taxCollector visitor, allowing it to interact with the utility.
- void incrementLevel ()

  Increments the utility's level, typically for upgrades.

Protected Member Functions

- void setResourceConsumption (int consumption)

  Sets the resource consumption rate.

Protected Attributes

- std::string name

  Name of the utility.
- double cost = 0.0

  Monetary cost to build or upgrade.
- int capacity = 0

  Maximum number of buildings it can support.
- double effectRadius = 0.0

  Area of effect for the utility.
- bool isOperational = false

  Operational status of the utility.
- int level = 0

  Level of the utility, upgradeable.
- int resourceConsumption = 0

  Resources required to keep the utility running.
- std::map< std::string, int > resourceNeeds

  Map of resources required for construction.

5.98.1 Detailed Description

Abstract base class for utility components, providing shared functionality.

The UtilityFlyweight class defines shared attributes and methods for various utility types. Implements the Prototype pattern for cloning and defines core utility attributes like cost, capacity, operational status, and resource requirements.

5.98.2 Constructor & Destructor Documentation

5.98.2.1 UtilityFlyweight() [1/2]

```
UtilityFlyweight::UtilityFlyweight ( )  [default]
```

Default constructor for UtilityFlyweight.

5.98.2.2 UtilityFlyweight() [2/2]

```
UtilityFlyweight::UtilityFlyweight (
            const std::string & n,
            double c,
            int cap,
            double radius,
            bool operational,
            int t,
            int consumption,
            const std::map< std::string, int > & resources )
```

Parameterized constructor for UtilityFlyweight.

Constructs a UtilityFlyweight with specified attributes.

Allows setting initial values for key attributes of the utility.

Parameters

n	Name of the utility.
c	Cost of the utility.
cap	Capacity of the utility.
radius	Effect radius of the utility.
operational	Operational status of the utility.
l	Level of the utility.
consumption	Resource consumption rate of the utility.
resources	Map of resources required for construction.
n	Name of the utility.
c	Cost of the utility.
cap	Capacity of the utility.
radius	Effect radius of the utility.
operational	Operational status of the utility.
t	Level of the utility.
consumption	Resource consumption rate of the utility.
resources	Map of resources required for construction.

5.98.2.3 ∼UtilityFlyweight()

```
virtual UtilityFlyweight::∼UtilityFlyweight ( )  [virtual], [default]
```

Virtual destructor for UtilityFlyweight.

5.98.3 Member Function Documentation

5.98.3.1 accept()

```
void UtilityFlyweight::accept (
            taxCollector * )  [inline], [override], [virtual]
```

Accepts a taxCollector visitor, allowing it to interact with the utility.

Currently, this function does not perform any operations.

Parameters

TC	Pointer to the taxCollector visitor.

Implements CityComponent.

5.98.3.2 activate()

```
void UtilityFlyweight::activate ( )
```

Activates the utility, setting it to operational.

Activates the utility, making it operational.

5.98.3.3 clone()

```
virtual std::shared_ptr<UtilityFlyweight> UtilityFlyweight::clone ( ) const  [pure virtual]
```

Creates a clone of the utility, implementing the Prototype pattern.

Pure virtual method to be implemented by derived classes for cloning.

Returns

A shared pointer to the cloned UtilityFlyweight instance.

Implemented in WaterSupply, WasteManagement, SewageSystem, and PowerPlant.

5.98.3.4 connect()

```
virtual void UtilityFlyweight::connect (
             double distance )  [pure virtual]
```

Connect the utility to a component within a specific distance.

Pure virtual function for establishing a connection based on distance. Must be implemented in derived classes.

Parameters

distance	Distance to the component being connected.

Implemented in WaterSupply, WasteManagement, SewageSystem, and PowerPlant.

5.98.3.5 deactivate()

```
void UtilityFlyweight::deactivate ( )
```

Deactivates the utility, setting it to non-operational.

Deactivates the utility, making it non-operational.

5.98.3.6 displayStatus()

```
void UtilityFlyweight::displayStatus ( ) [virtual]
```

Displays the current status of the utility.

Displays the current status of the utility, including attributes and requirements.

Implements CityComponent.

5.98.3.7 getBuildingType()

```
std::string UtilityFlyweight::getBuildingType ( ) const [inline], [override], [virtual]
```

Gets the type of the building, overridden from CityComponent.

Returns

The building type, which is "Utility" for UtilityFlyweight.

Implements CityComponent.

5.98.3.8 getCapacity()

```
int UtilityFlyweight::getCapacity ( ) const
```

Getter for the maximum capacity.

Retrieves the capacity of the utility.

Returns

The maximum number of buildings the utility can support.

5.98.3.9 getCost()

```
double UtilityFlyweight::getCost ( ) const
```

Getter for the monetary cost.

Retrieves the monetary cost of the utility.

Returns

The cost of the utility.

5.98.3.10 getEffectRadius()

```
double UtilityFlyweight::getEffectRadius ( ) const
```

Getter for the area of effect radius.

Retrieves the effect radius of the utility.

Returns

The area of effect radius.

5.98.3.11 getLevel()

```
int UtilityFlyweight::getLevel ( ) const
```

Getter for the utility's current level.

Retrieves the level of the utility.

Returns

The current level of the utility.

5.98.3.12 getName()

```
std::string UtilityFlyweight::getName ( ) const
```

Getter for the utility's name.

Retrieves the name of the utility.

Returns

The name of the utility.

5.98.3.13 getOperationalStatus()

```
bool UtilityFlyweight::getOperationalStatus ( ) const
```

Getter for the operational status.

Retrieves the operational status of the utility.

Returns

True if operational, false otherwise.

5.98.3.14 getRadius()

```
double UtilityFlyweight::getRadius ( ) const
```

Gets the effect radius of the utility.

Retrieves the radius of effect.

Returns

The effect radius.

The effect radius of the utility.

5.98.3.15 getResourceConsumption()

```
int UtilityFlyweight::getResourceConsumption ( ) const  [virtual]
```

Getter for the resource consumption rate.

Retrieves the resource consumption rate.

Returns

The rate of resource consumption per cycle.

5.98.3.16 getResourceNeeds()

```
std::map< std::string, int > UtilityFlyweight::getResourceNeeds ( ) const
```

Getter for resource requirements map.

Retrieves the map of resources required for construction.

Returns

A map of resources and quantities needed for construction.

5.98.3.17 incrementLevel()

```
void UtilityFlyweight::incrementLevel ( )  [inline]
```

Increments the utility's level, typically for upgrades.

5.98.3.18 setResourceConsumption()

```
void UtilityFlyweight::setResourceConsumption (
             int consumption )  [protected]
```

Sets the resource consumption rate.

Parameters

consumption	The new resource consumption rate.

5.98.4 Member Data Documentation

5.98.4.1 capacity

```
int UtilityFlyweight::capacity = 0  [protected]
```

Maximum number of buildings it can support.

5.98.4.2 cost

```
double UtilityFlyweight::cost = 0.0  [protected]
```

Monetary cost to build or upgrade.

5.98.4.3 effectRadius

```
double UtilityFlyweight::effectRadius = 0.0  [protected]
```

Area of effect for the utility.

5.98.4.4 isOperational

```
bool UtilityFlyweight::isOperational = false  [protected]
```

Operational status of the utility.

5.98.4.5 level

```
int UtilityFlyweight::level = 0  [protected]
```

Level of the utility, upgradeable.

5.98.4.6 name

```
std::string UtilityFlyweight::name  [protected]
```

Name of the utility.

5.98.4.7 resourceConsumption

```
int UtilityFlyweight::resourceConsumption = 0  [protected]
```

Resources required to keep the utility running.

5.98.4.8 resourceNeeds

```
std::map<std::string, int> UtilityFlyweight::resourceNeeds  [protected]
```

Map of resources required for construction.

The documentation for this class was generated from the following files:

- COS214-Poject/src/UtilityFlyweight.h
- COS214-Poject/src/UtilityFlyweight.cpp

5.99 WasteManagement Class Reference

Concrete class for managing waste utility in the city.

```
#include <WasteManagement.h>
```

Inheritance diagram for WasteManagement:

Collaboration diagram for WasteManagement:

Public Member Functions

- WasteManagement ()

  Default constructor for WasteManagement, initializing with preset values.
- WasteManagement (const std::string &n, double c, int cap, double radius, bool operational, int l, int consumption, const std::map< std::string, int > &resources)

  Parameterized constructor for WasteManagement, allowing customization.
- void connect (double distance)

  Connects the waste management service to a city component within a specified distance.
- std::shared_ptr< UtilityFlyweight > clone () const

  Clones the WasteManagement object, creating a new instance with the same attributes.

Additional Inherited Members

5.99.1 Detailed Description

Concrete class for managing waste utility in the city.

WasteManagement represents a specific utility that handles waste disposal and recycling operations. It can connect to components within a certain range and can be cloned to create new instances with the same attributes.

5.99.2 Constructor & Destructor Documentation

5.99.2.1 WasteManagement() [1/2]

```
WasteManagement::WasteManagement ( )
```

Default constructor for WasteManagement, initializing with preset values.

Default constructor for WasteManagement.

Sets values for name, cost, capacity, effect radius, operational status, level, consumption, and required resources such as recycling and disposal equipment.

Initializes WasteManagement with default values such as name, cost, capacity, radius, operational status, level, resource consumption, and required resources.

5.99.2.2 WasteManagement() [2/2]

```
WasteManagement::WasteManagement (
            const std::string & n,
            double c,
            int cap,
            double radius,
            bool operational,
            int l,
            int consumption,
            const std::map< std::string, int > & resources )
```

Parameterized constructor for WasteManagement, allowing customization.

Parameterized constructor for WasteManagement.

Allows specifying attributes such as the name, cost, capacity, effect radius, operational status, level, consumption rate, and required resources for the WasteManagement instance.

Parameters

n	The name of the waste management facility.
c	The cost associated with waste management services.
cap	The capacity of the waste management facility.

Parameters

radius	The effective radius within which waste management can operate.
operational	The operational status of the waste management service.
l	The level of the waste management service.
consumption	The resource consumption rate of the waste management service.
resources	A map of resources associated with the waste management facility.
n	The name of the utility.
c	The construction cost.
cap	The capacity, i.e., the number of buildings it can support.
radius	The effective radius of the utility.
operational	The operational status of the utility.
l	The level of the utility.
consumption	The resource consumption per cycle.
resources	The map of resources required for construction.

5.99.3 Member Function Documentation

5.99.3.1 clone()

```
std::shared_ptr< UtilityFlyweight > WasteManagement::clone ( ) const  [virtual]
```

Clones the WasteManagement object, creating a new instance with the same attributes.

Creates a clone of the WasteManagement utility.

Returns

A shared pointer to the cloned WasteManagement object.

Implements the Prototype pattern to return a new instance that is a copy of the current WasteManagement object.

Returns

A shared pointer to the cloned WasteManagement instance.

Implements UtilityFlyweight.

5.99.3.2 connect()

```
void WasteManagement::connect (
             double distance )  [virtual]
```

Connects the waste management service to a city component within a specified distance.

Connects the WasteManagement utility to a building based on distance.

Outputs a message indicating if the connection is successful based on the operational status and the distance relative to the effect radius.

Parameters

distance	The distance to the component being connected to.

Outputs a message indicating if the utility is operational and within range to provide waste management services to the target building.

Parameters

distance	The distance to the target building.

Implements UtilityFlyweight.

The documentation for this class was generated from the following files:

- COS214-Poject/src/WasteManagement.h
- COS214-Poject/src/WasteManagement.cpp

5.100 WaterSupply Class Reference

Represents a water supply utility that can be connected to city components within a specific radius.

```
#include <WaterSupply.h>
```

Inheritance diagram for WaterSupply:

Collaboration diagram for WaterSupply:

Public Member Functions

- WaterSupply ()

  Default constructor for WaterSupply.
- WaterSupply (const std::string &n, double c, int cap, double radius, bool operational, int l, int consumption, const std::map< std::string, int > &resources)

  Constructs a WaterSupply object with specified attributes.
- void connect (double distance)

  Connects the water supply to a city component within a specified distance.
- std::shared_ptr< UtilityFlyweight > clone () const

  Clones the WaterSupply object, creating a new instance with the same attributes.

Additional Inherited Members

5.100.1 Detailed Description

Represents a water supply utility that can be connected to city components within a specific radius.

5.100.2 Constructor & Destructor Documentation

5.100.2.1 WaterSupply() [1/2]

```
WaterSupply::WaterSupply ( )
```

Default constructor for WaterSupply.

Default constructor for WaterSupply, initializing it with preset values.

Sets moderate values for cost, capacity, radius, operational status, level, consumption, and required resources such as Steel and Plastic.

5.100.2.2 WaterSupply() [2/2]

```
WaterSupply::WaterSupply (
            const std::string & n,
            double c,
            int cap,
            double radius,
            bool operational,
            int l,
            int consumption,
            const std::map< std::string, int > & resources )
```

Constructs a WaterSupply object with specified attributes.

Parameterized constructor for WaterSupply with customizable attributes.

Parameters

n	The name of the water supply.
c	The cost associated with the water supply.
cap	The capacity of the water supply.
radius	The effective radius within which the water supply can operate.
operational	The operational status of the water supply.
l	The water supply level.
consumption	The consumption rate of the water supply.
resources	A map of resources associated with the water supply.

Allows specifying the name, cost, capacity, effect radius, operational status, level, consumption rate, and required resources for the WaterSupply instance.

Parameters

n	The name of the water supply.
c	The cost associated with the water supply.
cap	The capacity of the water supply.

Parameters

radius	The effective radius within which the water supply can operate.
operational	The operational status of the water supply.
l	The level of the water supply.
consumption	The consumption rate of the water supply.
resources	A map of resources associated with the water supply.

5.100.3 Member Function Documentation

5.100.3.1 clone()

```
std::shared_ptr< UtilityFlyweight > WaterSupply::clone ( ) const  [virtual]
```

Clones the WaterSupply object, creating a new instance with the same attributes.

Returns

A shared pointer to the cloned WaterSupply object.

Implements UtilityFlyweight.

5.100.3.2 connect()

```
void WaterSupply::connect (
             double distance )  [virtual]
```

Connects the water supply to a city component within a specified distance.

Parameters

distance	The distance to the component being connected to.

Outputs a message indicating if the connection is successful based on the operational status and the distance relative to the effect radius.

Parameters

distance	The distance to the component being connected to.

Implements UtilityFlyweight.

The documentation for this class was generated from the following files:

- COS214-Poject/src/WaterSupply.h
- COS214-Poject/src/WaterSupply.cpp

5.101 Wood Class Reference

Represents a construction resource product of type Wood.

```
#include <Wood.h>
```

Inheritance diagram for Wood:

Collaboration diagram for Wood:

Public Member Functions

- Wood (int quantity, int unitCost)

    Constructs a Wood object with specified quantity and unit cost.
- ∼Wood ()

    Destructor for Wood.
- void displayStatus () override

    Displays the current status of the wood resource.
- std::string getBuildingType () const

    Gets the building type associated with this resource.

5.101.1 Detailed Description

Represents a construction resource product of type Wood.

The Wood class inherits from ConstructionResourceProduct and provides specific functionality for handling wood resources, including displaying its status and identifying its type.

5.101.2 Constructor & Destructor Documentation

5.101.2.1 Wood()

```
Wood::Wood (
                int quantity,
                int unitCost )
```

Constructs a Wood object with specified quantity and unit cost.

Parameters

quantity	The quantity of wood.
unitCost	The unit cost of the wood.

5.101.2.2 ∼Wood()

```
Wood::∼Wood ( )
```

Destructor for Wood.

5.101.3 Member Function Documentation

5.101.3.1 displayStatus()

```
void Wood::displayStatus ( ) [override], [virtual]
```

Displays the current status of the wood resource.

Overrides the displayStatus method to provide wood-specific status details.

Provides specific information for wood, including quantity, unit cost, and total cost.

Reimplemented from ConstructionResourceProduct.

Here is the call graph for this function:

5.101.3.2 getBuildingType()

```
std::string Wood::getBuildingType ( ) const [inline], [virtual]
```

Gets the building type associated with this resource.

Returns

A string representing the building type, in this case, "Wood".

Implements CityComponent.

The documentation for this class was generated from the following files:

- COS214-Poject/src/Wood.h
- COS214-Poject/src/Wood.cpp

5.102 WoodAndCoalPlant Class Reference

Concrete Industry for processing lumber and coal resources.

`#include <WoodAndCoalPlant.h>`

Inheritance diagram for WoodAndCoalPlant:

Collaboration diagram for WoodAndCoalPlant:

Public Member Functions

- WoodAndCoalPlant (std::shared_ptr< IncomeResourceProduct > coal, std::shared_ptr< ConstructionResourceProduct > wood, MapGrid *grid, std::map< std::string, int > &collectedResources)

  Constructs a WoodAndCoalPlant with specified resource dependencies and map grid.
- void processCoal (int amount)

  Processes a specified amount of coal and updates resources.
- void processWood (int amount)

  Processes a specified amount of wood and updates resources.

Static Public Attributes

- static const int WOOD_COAL_RANGE = 5

  The processing range for wood and coal resources.

Additional Inherited Members

5.102.1 Detailed Description

Concrete Industry for processing lumber and coal resources.

This class represents a specialized industry for handling wood and coal processing. It performs specific functions to process these resources and interact with the game map grid for resource management.

5.102.2 Constructor & Destructor Documentation

5.102.2.1 WoodAndCoalPlant()

```
WoodAndCoalPlant::WoodAndCoalPlant (
             std::shared_ptr< IncomeResourceProduct > coal,
             std::shared_ptr< ConstructionResourceProduct > wood,
             MapGrid * grid,
             std::map< std::string, int > & collectedResources )
```

Constructs a WoodAndCoalPlant with specified resource dependencies and map grid.

Constructs a WoodAndCoalPlant with specified coal and wood resources, grid, and resource collection map.

Parameters

coal	Shared pointer to the coal income resource product.
wood	Shared pointer to the wood construction resource product.
grid	Pointer to the map grid where the plant is located.
collectedResources	Map reference to store the collected resources.
coal	Shared pointer to the coal income resource product.
wood	Shared pointer to the wood construction resource product.
grid	Pointer to the map grid where the plant is located.
collectedResources	Reference to the map storing collected resources.

Initializes the WoodAndCoalPlant with its specific name, income and construction resource processors, grid, resource collection map, and processing range.

5.102.3 Member Function Documentation

5.102.3.1 processCoal()

```
void WoodAndCoalPlant::processCoal (
             int amount )
```

Processes a specified amount of coal and updates resources.

Processes a specified amount of coal, updating resources accordingly.

Parameters

amount	The amount of coal to be processed.
amount	The amount of coal to be processed.

Outputs a message to indicate coal processing and uses the inherited processResources function to handle the processing logic. Here is the call graph for this function:

5.102.3.2 processWood()

```
void WoodAndCoalPlant::processWood (
             int amount )
```

Processes a specified amount of wood and updates resources.

Processes a specified amount of wood, updating resources accordingly.

Parameters

amount	The amount of wood to be processed.
amount	The amount of wood to be processed.

Outputs a message to indicate wood processing and uses the inherited processResources function to handle the processing logic. Here is the call graph for this function:

5.102.4 Member Data Documentation

5.102.4.1 WOOD_COAL_RANGE

```
const int WoodAndCoalPlant::WOOD_COAL_RANGE = 5  [static]
```

The processing range for wood and coal resources.

The documentation for this class was generated from the following files:

- COS214-Poject/src/WoodAndCoalPlant.h
- COS214-Poject/src/WoodAndCoalPlant.cpp

5.103 WorkerNPC Class Reference

Represents a Worker NPC that collects resources.

```
#include <NPCSystem.h>
```

Collaboration diagram for WorkerNPC:

Public Member Functions

- WorkerNPC (const std::string &workerName, double workerCost, std::unique_ptr< CollectionStrategy > collectionStrategy)

  Constructs a Worker NPC with a specific name, cost, and collection strategy.
- std::string getName () const

  Gets the name of the Worker NPC.
- double getCost () const

  Gets the cost of the Worker NPC.
- int collect (int amount) const

  Collects resources based on the collection strategy.

Private Attributes

- std::string name

  Name of the Worker NPC.
- double cost

  Cost of hiring the Worker NPC.
- std::unique_ptr< CollectionStrategy > strategy

  Strategy for resource collection.

5.103.1 Detailed Description

Represents a Worker NPC that collects resources.

Uses a collection strategy to determine the NPC's collection efficiency.

5.103.2 Constructor & Destructor Documentation

5.103.2.1 WorkerNPC()

```
WorkerNPC::WorkerNPC (
             const std::string & workerName,
             double workerCost,
             std::unique_ptr< CollectionStrategy > collectionStrategy )
```

Constructs a Worker NPC with a specific name, cost, and collection strategy.

Parameters

workerName	The name of the Worker NPC.
workerCost	The cost associated with the Worker NPC.
collectionStrategy	The strategy for resource collection.

5.103.3 Member Function Documentation

5.103.3.1 collect()

```
int WorkerNPC::collect (
             int amount ) const
```

Collects resources based on the collection strategy.

Parameters

amount	The base amount to be collected.

Returns

int The amount collected after applying the collection strategy.

5.103.3.2 getCost()

```
double WorkerNPC::getCost ( ) const
```

Gets the cost of the Worker NPC.

Returns

double The hiring cost of the Worker NPC.

5.103.3.3 getName()

```
std::string WorkerNPC::getName ( ) const
```

Gets the name of the Worker NPC.

Returns

std::string The NPC's name.

5.103.4 Member Data Documentation

5.103.4.1 cost

```
double WorkerNPC::cost  [private]
```

Cost of hiring the Worker NPC.

5.103.4.2 name

```
std::string WorkerNPC::name  [private]
```

Name of the Worker NPC.

5.103.4.3 strategy

```
std::unique_ptr<CollectionStrategy> WorkerNPC::strategy  [private]
```

Strategy for resource collection.

The documentation for this class was generated from the following file:

- COS214-Poject/src/NPCSystem.h

5.104 ZoneComposite Class Reference

A composite class representing a zone in the city, containing multiple buildings and managing their operations.

```
#include <ZoneComposite.h>
```

Inheritance diagram for ZoneComposite:

Collaboration diagram for ZoneComposite:

Public Member Functions

- ZoneComposite (const std::string &type, const Location &topLeft, const Location &bottomRight)

  Constructs a ZoneComposite with a specified type and boundary locations.
- void add (std::shared_ptr< CityComponent > component)

  Adds a building component to the zone.
- void remove (std::shared_ptr< CityComponent > component)

  Removes a building component from the zone.
- void displayStatus ()

  Displays the status of all buildings within the zone.
- std::string getBuildingType () const

  Returns the building type for the zone.
- void accept (taxCollector ∗TC) override

  Accepts a taxCollector visitor to apply tax operations within the zone.
- const std::vector< std::shared_ptr< CityComponent > > & getBuildings () const

  Retrieves all buildings in the zone.
- const Location & getTopLeft () const

  Gets the top-left boundary location of the zone.
- const Location & getBottomRight () const

  Gets the bottom-right boundary location of the zone.

Private Attributes

- std::vector< std::shared_ptr< CityComponent > > buildings

  Vector storing shared pointers to buildings in the zone.
- std::string zoneType

  Type of zone (e.g., Residential, Commercial, Industrial).
- Location bounds [2]

  Boundaries of the zone: bounds[0] = top-left, bounds[1] = bottom-right.

5.104.1 Detailed Description

A composite class representing a zone in the city, containing multiple buildings and managing their operations.

5.104.2 Constructor & Destructor Documentation

5.104.2.1 ZoneComposite()

```
ZoneComposite::ZoneComposite (
             const std::string & type,
             const Location & topLeft,
             const Location & bottomRight )
```

Constructs a ZoneComposite with a specified type and boundary locations.

Constructs a ZoneComposite object with specified type and boundary coordinates.

Parameters

type	The type of the zone (e.g., Residential, Commercial).
topLeft	The top-left boundary of the zone.
bottomRight	The bottom-right boundary of the zone.
type	The type of zone (e.g., Residential, Commercial).
topLeft	The top-left boundary of the zone.
bottomRight	The bottom-right boundary of the zone.

5.104.3 Member Function Documentation

5.104.3.1 accept()

```
void ZoneComposite::accept (
             taxCollector * TC )  [override], [virtual]
```

Accepts a taxCollector visitor to apply tax operations within the zone.

Parameters

TC	Pointer to a taxCollector object.

Implements CityComponent.

Here is the call graph for this function:

5.104.3.2 add()

```
void ZoneComposite::add (
             std::shared_ptr< CityComponent > component )  [virtual]
```

Adds a building component to the zone.

Parameters

component	Shared pointer to the building component to add.

Reimplemented from CityComponent.

5.104.3.3 displayStatus()

```
void ZoneComposite::displayStatus ( )  [virtual]
```

Displays the status of all buildings within the zone.

Implements CityComponent.

5.104.3.4 getBottomRight()

```
const Location& ZoneComposite::getBottomRight ( ) const  [inline]
```

Gets the bottom-right boundary location of the zone.

Returns

A constant reference to the bottom-right Location.

5.104.3.5 getBuildings()

```
const std::vector<std::shared_ptr<CityComponent> >& ZoneComposite::getBuildings ( ) const
[inline]
```

Retrieves all buildings in the zone.

Returns

A constant reference to the vector of buildings.

Here is the caller graph for this function:

5.104.3.6 getBuildingType()

```
std::string ZoneComposite::getBuildingType ( ) const  [virtual]
```

Returns the building type for the zone.

Returns

The type of building as a string.

Implements CityComponent.

5.104.3.7 getTopLeft()

```
const Location& ZoneComposite::getTopLeft ( ) const  [inline]
```

Gets the top-left boundary location of the zone.

Returns

A constant reference to the top-left Location.

5.104.3.8 remove()

```
void ZoneComposite::remove (
             std::shared_ptr< CityComponent > component )  [virtual]
```

Removes a building component from the zone.

Parameters

component	Shared pointer to the building component to remove.

Reimplemented from CityComponent.

5.104.4 Member Data Documentation

5.104.4.1 bounds

```
Location ZoneComposite::bounds[2]  [private]
```

Boundaries of the zone: bounds[0] = top-left, bounds[1] = bottom-right.

5.104.4.2 buildings

```
std::vector<std::shared_ptr<CityComponent> > ZoneComposite::buildings  [private]
```

Vector storing shared pointers to buildings in the zone.

5.104.4.3 zoneType

```
std::string ZoneComposite::zoneType  [private]
```

Type of zone (e.g., Residential, Commercial, Industrial).

The documentation for this class was generated from the following files:

- COS214-Poject/src/ZoneComposite.h
- COS214-Poject/src/ZoneComposite.cpp

5.105 MapGrid::ZoneStyle Struct Reference

Collaboration diagram for MapGrid::ZoneStyle:

Public Member Functions

- ZoneStyle (const std::array< std::string, 9 > &borders, const std::string &bgColor)

Public Attributes

- std::array< std::string, 9 > borderChars
- std::string backgroundColor

5.105.1 Constructor & Destructor Documentation

5.105.1.1 ZoneStyle()

```
MapGrid::ZoneStyle::ZoneStyle (
             const std::array< std::string, 9 > & borders,
             const std::string & bgColor )  [inline]
```

5.105.2 Member Data Documentation

5.105.2.1 backgroundColor

```
std::string MapGrid::ZoneStyle::backgroundColor
```

5.105.2.2 borderChars

```
std::array<std::string, 9> MapGrid::ZoneStyle::borderChars
```

The documentation for this struct was generated from the following file:

- COS214-Poject/src/MapGrid.h

Chapter 6

File Documentation

6.1 COS214-Poject/src/AdvancedTechnologyDecorator.cpp File Reference

6.2 COS214-Poject/src/AdvancedTechnologyDecorator.h File Reference

6.3 COS214-Poject/src/BuildingRequirements.cpp File Reference

Implementation file that includes the definitions for resource management and building requirements.

```
#include "BuildingRequirements.h"
```
Include dependency graph for BuildingRequirements.cpp:

6.4 COS214-Poject/src/BuildingRequirements.h File Reference

```
#include <map>
#include <string>
#include <iostream>
```
Include dependency graph for BuildingRequirements.h: This graph shows which files directly or indirectly include this file:

Classes

- struct ResourceRequirement

  Defines the resource requirements for constructing a building.

Functions

- void initializeCollectedResources (std::map< std::string, int > &collectedResources)

  Initializes the collected resources map with zero values.
- bool checkResourceRequirements (const std::string &buildingType, const std::map< std::string, int > &collectedResources, bool displayInfo=true)

  Checks if the collected resources meet the requirements for a specified building.

Variables

- const std::map< std::string, ResourceRequirement > buildingResourceRequirements

    A map that associates building types with their respective resource requirements.

6.4.1 Function Documentation

6.4.1.1 checkResourceRequirements()

```
bool checkResourceRequirements (
            const std::string & buildingType,
            const std::map< std::string, int > & collectedResources,
            bool displayInfo = true ) [inline]
```

Checks if the collected resources meet the requirements for a specified building.

This function compares collected resource quantities against the requirements for a given building type. Optionally, it displays information about the requirements and current resource levels.

Parameters

buildingType	The type of building to check resources for.
collectedResources	A map containing the currently collected resource quantities.
displayInfo	Flag to control the display of requirement information (default: true).

Returns

true if resources meet or exceed the requirements; false otherwise.

Here is the caller graph for this function:

6.4.1.2 initializeCollectedResources()

```
void initializeCollectedResources (
            std::map< std::string, int > & collectedResources ) [inline]
```

Initializes the collected resources map with zero values.

This function populates the given map with resource types as keys and initializes their values to zero.

Parameters

collectedResources	A map to hold the quantities of collected resources.

Here is the caller graph for this function:

6.4.2 Variable Documentation

6.4.2.1 buildingResourceRequirements

```
buildingResourceRequirements
```

Initial value:

```
{

    {"House", ResourceRequirement(10, 5, 0, 0)},
    {"Flat", ResourceRequirement(15, 10, 5, 5)},
    {"Townhouse", ResourceRequirement(20, 15, 10, 10)},
    {"Estate", ResourceRequirement(25, 20, 15, 15)},


    {"Shop", ResourceRequirement(15, 10, 10, 10)},
    {"Mall", ResourceRequirement(30, 25, 20, 20)},
    {"Office", ResourceRequirement(25, 20, 25, 25)},


    {"MetalWorkFacility", ResourceRequirement(20, 30, 40, 20)},
    {"PetroChemical", ResourceRequirement(15, 25, 35, 40)},
    {"CrystalCraft", ResourceRequirement(20, 40, 30, 25)},
    {"WoodAndCoal", ResourceRequirement(40, 30, 25, 20)}
}
```

A map that associates building types with their respective resource requirements.

This map contains entries for various types of buildings (residential, commercial, and industrial) and their corresponding resource requirements for construction.

6.5 COS214-Poject/src/ChemicalFactory.cpp File Reference

Implementation of the ChemicalFactory class for creating resources.

```
#include "ChemicalFactory.h"
#include "Oil.h"
#include "Concrete.h"
```

Include dependency graph for ChemicalFactory.cpp:

6.5.1 Detailed Description

Implementation of the ChemicalFactory class for creating resources.

This file provides the implementation for methods in the ChemicalFactory class, which creates income-generating and construction resources.

6.6 COS214-Poject/src/ChemicalFactory.h File Reference

```
#include "ResourceFactory.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include <iostream>
#include <memory>
```

Include dependency graph for ChemicalFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class ChemicalFactory

  A factory class for creating chemical resources used in construction and income generation.

6.7 COS214-Poject/src/CityComponent.cpp File Reference

Implementation of the CityComponent class, which handles city component behaviors such as managing NPCs, location, and notifications.

```
#include "NPCObserver.h"
#include "CityComponent.h"
#include "NPCManager.h"
#include "Government.h"
#include "ReactingNPCS.h"
```

Include dependency graph for CityComponent.cpp:

6.7.1 Detailed Description

Implementation of the CityComponent class, which handles city component behaviors such as managing NPCs, location, and notifications.

This file provides the implementation for the CityComponent class, including functions for adding and removing NPCs, notifying observers, and setting locations.

6.8 COS214-Poject/src/CityComponent.h File Reference

Abstract base class for components within the city simulation.

```
#include "Location.h"
#include <iostream>
#include <vector>
#include <string>
#include <memory>
```

Include dependency graph for CityComponent.h: This graph shows which files directly or indirectly include this file:

Classes

- class CityComponent

  Abstract base class representing a component of a city.

6.8.1 Detailed Description

Abstract base class for components within the city simulation.

This file defines the CityComponent class, an abstract class for components in the city simulation, providing common functionality such as observer management, location handling, and status display.

6.9 COS214-Poject/src/CityComposite.cpp File Reference

Implementation of the CityComposite class, which manages zones, happiness, and tax adjustments for the city.

```
#include "CityComposite.h"
#include "CityComponent.h"
#include "NPCObserver.h"
#include "Government.h"
#include "NPCManager.h"
#include <iostream>
#include <vector>
#include <string>
```
Include dependency graph for CityComposite.cpp:

6.9.1 Detailed Description

Implementation of the CityComposite class, which manages zones, happiness, and tax adjustments for the city.

This file provides implementations for functions related to adding and removing zones, displaying city status, managing city conditions, and adjusting happiness based on components and tax rates.

6.10 COS214-Poject/src/CityComposite.h File Reference

Defines the CityComposite class, which represents a composite structure containing multiple city zones.

```
#include "CityComponent.h"
#include "Transport.h"
#include "Government.h"
#include "taxCollector.h"
#include "MapGrid.h"
#include <string>
#include <vector>
#include <algorithm>
#include <memory>
```
Include dependency graph for CityComposite.h: This graph shows which files directly or indirectly include this file:

Classes

- class CityComposite

    A composite class representing the entire city and containing multiple zones.

6.10.1 Detailed Description

Defines the CityComposite class, which represents a composite structure containing multiple city zones.

The CityComposite class is a concrete implementation of the CityComponent class, representing a collection of zones and managing high-level city attributes, such as zone management, budget, and happiness levels.

6.11 COS214-Poject/src/CityGame.cpp File Reference

```
#include "CityGame.h"
```
Include dependency graph for CityGame.cpp:

6.12 COS214-Poject/src/CityGame.h File Reference

```
#include "GameState.h"
#include "MapGrid.h"
#include "CityComponent.h"
#include "ResidentialBuilding.h"
#include "House.h"
#include "Flat.h"
#include "Townhouse.h"
#include "Estate.h"
#include "CommercialBuilding.h"
#include "Shops.h"
#include "Office.h"
#include "Malls.h"
#include "Industry.h"
#include "MetalWorkFacility.h"
#include "PetroChemicalFacility.h"
#include "CrystalCraftIndustry.h"
#include "WoodAndCoalPlant.h"
#include "LandMark.h"
#include "Park.h"
#include "Monument.h"
#include "CulturalCenter.h"
#include "UtilityFlyweight.h"
#include "UtilityFactory.h"
#include "WaterSupply.h"
#include "PowerPlant.h"
#include "SewageSystem.h"
#include "WasteManagement.h"
#include "concreteTaxCollector.h"
#include "taxCollector.h"
#include "NPCManager.h"
#include "Gold.h"
#include "Diamonds.h"
#include "Coal.h"
#include "Oil.h"
#include "Stone.h"
#include "Wood.h"
#include "Steel.h"
#include "Concrete.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include "IncomeResourceProcessor.h"
#include "ResourceProcessor.h"
#include "ConstructionResourceProcessor.h"
#include "BuildingRequirements.h"
#include "Roads.h"
#include "RoadsFactory.h"
#include "Trains.h"
#include "TrainsFactory.h"
```

```
#include <iostream>
#include <memory>
#include <vector>
#include <string>
#include <algorithm>
#include <iomanip>
#include <limits>
#include <map>
```
This graph shows which files directly or indirectly include this file:

Classes

- class CityGame

6.13 COS214-Poject/src/Coal.cpp File Reference

Implementation of the Coal class, representing a coal resource in the game.

```
#include "Coal.h"
```
Include dependency graph for Coal.cpp:

6.13.1 Detailed Description

Implementation of the Coal class, representing a coal resource in the game.

6.14 COS214-Poject/src/Coal.h File Reference

```
#include <iostream>
#include "IncomeResourceProduct.h"
```
Include dependency graph for Coal.h: This graph shows which files directly or indirectly include this file:

Classes

- class Coal

  Represents a coal resource in the game, derived from IncomeResourceProduct.

6.15 COS214-Poject/src/CollectionStrategy.cpp File Reference

Implementation file for the CollectionStrategy class.

```
#include "CollectionStrategy.h"
```
Include dependency graph for CollectionStrategy.cpp:

6.15.1 Detailed Description

Implementation file for the CollectionStrategy class.

This file provides the base functionality for resource collection strategies. Since CollectionStrategy is an abstract class, it serves as the interface for various concrete collection strategies in the game.

6.16 COS214-Poject/src/CollectionStrategy.h File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class CollectionStrategy

  Base class for collection strategies in the NPC system.

6.17 COS214-Poject/src/Command.cpp File Reference

Implementation of the Command interface and its concrete classes for executing and undoing actions on the map grid.

```
#include "Command.h"
```
Include dependency graph for Command.cpp:

6.17.1 Detailed Description

Implementation of the Command interface and its concrete classes for executing and undoing actions on the map grid.

6.18 COS214-Poject/src/Command.h File Reference

```
#include <memory>
#include "MapGrid.h"
#include "Location.h"
#include "CityComponent.h"
```
Include dependency graph for Command.h: This graph shows which files directly or indirectly include this file:

Classes

- class Command

  Abstract base class representing a command in the command pattern.
- class PlaceComponentCommand

  Command to place a CityComponent on a MapGrid at a specified Location.

6.19 COS214-Poject/src/CommercialBuilding.cpp File Reference

Implementation of the CommercialBuilding class, handling commercial building attributes, utilities, and tax functionalities.

```
#include "CommercialBuilding.h"
#include "NPCManager.h"
#include "taxCollector.h"
#include <iostream>
```
Include dependency graph for CommercialBuilding.cpp:

6.19.1 Detailed Description

Implementation of the CommercialBuilding class, handling commercial building attributes, utilities, and tax functionalities.

6.20 COS214-Poject/src/CommercialBuilding.h File Reference

```
#include "CityComponent.h"
#include <memory>
#include <string>
```
Include dependency graph for CommercialBuilding.h: This graph shows which files directly or indirectly include this file:

Classes

- class CommercialBuilding

  Represents a commercial building in the city, capable of interacting with utilities and tax collection.

6.21 COS214-Poject/src/CommercialFactory.cpp File Reference

Implementation of the CommercialFactory interface, providing factory methods for creating commercial buildings.

```
#include "CommercialFactory.h"
```
Include dependency graph for CommercialFactory.cpp:

6.21.1 Detailed Description

Implementation of the CommercialFactory interface, providing factory methods for creating commercial buildings.

6.22 COS214-Poject/src/CommercialFactory.h File Reference

```
#include <memory>
#include "CommercialBuilding.h"
```
Include dependency graph for CommercialFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class CommercialFactory

  Abstract factory class for creating commercial buildings.

6.23 COS214-Poject/src/Concrete.cpp File Reference

Implementation of the Concrete class, representing a construction resource for use within the game.

```
#include "Concrete.h"
```

Include dependency graph for Concrete.cpp:

6.23.1 Detailed Description

Implementation of the Concrete class, representing a construction resource for use within the game.

6.24 COS214-Poject/src/Concrete.h File Reference

```
#include "ConstructionResourceProduct.h"
#include <iostream>
```

Include dependency graph for Concrete.h: This graph shows which files directly or indirectly include this file:

Classes

- class Concrete

  Represents a concrete resource used for construction within the game.

6.25 COS214-Poject/src/concreteTaxCollector.cpp File Reference

Implementation of the ConcreteTaxCollector class, defining tax collection behaviors for residential and commercial buildings, as well as zones.

```
#include "concreteTaxCollector.h"
#include "ZoneComposite.h"
```

Include dependency graph for concreteTaxCollector.cpp:

6.25.1 Detailed Description

Implementation of the ConcreteTaxCollector class, defining tax collection behaviors for residential and commercial buildings, as well as zones.

6.26 COS214-Poject/src/concreteTaxCollector.h File Reference

```
#include "taxCollector.h"
#include "Government.h"
#include "ZoneComposite.h"
#include "ResidentialBuilding.h"
#include "CommercialBuilding.h"
#include "NPCManager.h"
```

Include dependency graph for concreteTaxCollector.h: This graph shows which files directly or indirectly include this file:

Classes

- class ConcreteTaxCollector

  A concrete implementation of the taxCollector interface, responsible for collecting taxes from various types of buildings and zones.

6.27 COS214-Poject/src/ConstructionResourceProcessor.cpp File Reference

Implementation of the ConstructionResourceProcessor class, managing processing and storage of construction resources.

```
#include "ConstructionResourceProcessor.h"
```

Include dependency graph for ConstructionResourceProcessor.cpp:

6.27.1 Detailed Description

Implementation of the ConstructionResourceProcessor class, managing processing and storage of construction resources.

6.28 COS214-Poject/src/ConstructionResourceProcessor.h File Reference

```
#include "ResourceProcessor.h"
#include "ConstructionResourceProduct.h"
#include "Government.h"
#include <memory>
```

Include dependency graph for ConstructionResourceProcessor.h: This graph shows which files directly or indirectly include this file:

Classes

- class ConstructionResourceProcessor

  Manages the processing and storage of construction resources within the game.

6.29 COS214-Poject/src/ConstructionResourceProduct.cpp File Reference

```
#include "ConstructionResourceProduct.h"
```
Include dependency graph for ConstructionResourceProduct.cpp:

6.30 COS214-Poject/src/ConstructionResourceProduct.h File Reference

```
#include <string>
#include <iostream>
#include "CityComponent.h"
```
Include dependency graph for ConstructionResourceProduct.h: This graph shows which files directly or indirectly include this file:

Classes

- class ConstructionResourceProduct

  Represents a construction resource product in the city-building simulation.

6.31 COS214-Poject/src/CrimeState.cpp File Reference

Implementation of the CrimeState class, representing the behavior of NPCs in a crime state.

```
#include "Government.h"
#include "NPCManager.h"
#include "CrimeState.h"
#include <random>
#include <iostream>
#include <string>
```
Include dependency graph for CrimeState.cpp:

6.31.1 Detailed Description

Implementation of the CrimeState class, representing the behavior of NPCs in a crime state.

6.32 COS214-Poject/src/CrimeState.h File Reference

Declaration of the CrimeState class, representing a state of crime among NPCs.

```
#include "NPCManager.h"
#include "NPCState.h"
#include <string>
```
Include dependency graph for CrimeState.h: This graph shows which files directly or indirectly include this file:

Classes

- class CrimeState

  Represents the crime state for NPCs, impacting their behavior and the city's overall conditions.

6.32.1 Detailed Description

Declaration of the CrimeState class, representing a state of crime among NPCs.

6.33 COS214-Poject/src/CrystalCraftIndustry.cpp File Reference

Implementation of the CrystalCraftIndustry class for processing diamond and stone resources.

```
#include "CrystalCraftIndustry.h"
```
Include dependency graph for CrystalCraftIndustry.cpp:

6.33.1 Detailed Description

Implementation of the CrystalCraftIndustry class for processing diamond and stone resources.

6.34 COS214-Poject/src/CrystalCraftIndustry.h File Reference

Declaration of the CrystalCraftIndustry class, a specific type of Industry that processes diamonds and stone.

```
#include "Industry.h"
#include "IncomeResourceProcessor.h"
#include "ConstructionResourceProcessor.h"
#include <memory>
#include <map>
```
Include dependency graph for CrystalCraftIndustry.h: This graph shows which files directly or indirectly include this file:

Classes

- class CrystalCraftIndustry

  Represents an industry focused on processing diamonds and stone for various uses.

6.34.1 Detailed Description

Declaration of the CrystalCraftIndustry class, a specific type of Industry that processes diamonds and stone.

6.35 COS214-Poject/src/CulturalCenter.cpp File Reference

Implements the CulturalCenter class methods.

```
#include "CulturalCenter.h"
```
Include dependency graph for CulturalCenter.cpp:

6.35.1 Detailed Description

Implements the CulturalCenter class methods.

6.36 COS214-Poject/src/CulturalCenter.h File Reference

Defines the CulturalCenter class, representing a landmark structure within the city.

```
#include "LandMark.h"
```
Include dependency graph for CulturalCenter.h: This graph shows which files directly or indirectly include this file:

Classes

- class CulturalCenter

  Represents a cultural center landmark in the city, providing cultural activities and boosting nearby happiness.

6.36.1 Detailed Description

Defines the CulturalCenter class, representing a landmark structure within the city.

6.37 COS214-Poject/src/CulturalCenterFactory.cpp File Reference

Implementation of the CulturalCenterFactory class for creating CulturalCenter landmarks.

```
#include "CulturalCenterFactory.h"
#include "CulturalCenter.h"
```
Include dependency graph for CulturalCenterFactory.cpp:

6.37.1 Detailed Description

Implementation of the CulturalCenterFactory class for creating CulturalCenter landmarks.

6.38 COS214-Poject/src/CulturalCenterFactory.h File Reference

Defines the CulturalCenterFactory class for creating CulturalCenter landmarks.

```
#include "LandMarkFactory.h"
#include "LandMark.h"
```
Include dependency graph for CulturalCenterFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class CulturalCenterFactory

  Factory class responsible for creating CulturalCenter landmarks.

6.38.1 Detailed Description

Defines the CulturalCenterFactory class for creating CulturalCenter landmarks.

6.39 COS214-Poject/src/DeliveryTruck.h File Reference

6.40 COS214-Poject/src/Diamonds.cpp File Reference

```
#include "Diamonds.h"
```
Include dependency graph for Diamonds.cpp:

6.41 COS214-Poject/src/Diamonds.h File Reference

Defines the Diamonds class representing an income-generating resource in the city simulation.

```
#include "IncomeResourceProduct.h"
#include <iostream>
#include <string>
```
Include dependency graph for Diamonds.h: This graph shows which files directly or indirectly include this file:

Classes

- class Diamonds

  Represents diamonds as an income-generating resource in the city simulation.

6.41.1 Detailed Description

Defines the Diamonds class representing an income-generating resource in the city simulation.

6.42 COS214-Poject/src/DonationState.cpp File Reference

```
#include "DonationState.h"
#include "NPCManager.h"
#include "Government.h"
#include <iostream>
#include <random>
#include <string>
```

Include dependency graph for DonationState.cpp:

6.43 COS214-Poject/src/DonationState.h File Reference

```
#include "NPCManager.h"
#include "NPCState.h"
#include <string>
```

Include dependency graph for DonationState.h: This graph shows which files directly or indirectly include this file:

Classes

- class DonationState

  Represents the state where NPCs are in a donation mood.

6.44 COS214-Poject/src/Estate.cpp File Reference

```
#include "Estate.h"
```

Include dependency graph for Estate.cpp:

6.45 COS214-Poject/src/Estate.h File Reference

```
#include "ResidentialBuilding.h"
#include "taxCollector.h"
#include "CityComposite.h"
```

Include dependency graph for Estate.h: This graph shows which files directly or indirectly include this file:

Classes

- class Estate

  Represents an estate building in the city simulation.

6.46 COS214-Poject/src/EstateFactory.cpp File Reference

```
#include "EstateFactory.h"
#include "Estate.h"
```

Include dependency graph for EstateFactory.cpp:

6.47 COS214-Poject/src/EstateFactory.h File Reference

```
#include "ResidentialBuildingFactory.h"
#include "ResidentialBuilding.h"
```

Include dependency graph for EstateFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class EstateFactory

  The EstateFactory class is responsible for creating Estate buildings.

6.48 COS214-Poject/src/FastCollectionStrategy.cpp File Reference

```
#include "FastCollectionStrategy.h"
```

Include dependency graph for FastCollectionStrategy.cpp:

6.49 COS214-Poject/src/FastCollectionStrategy.h File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class FastCollectionStrategy

  Collection strategy for fast collection rate.

6.50 COS214-Poject/src/FireStation.cpp File Reference

```
#include "FireStation.h"
#include <iostream>
```

Include dependency graph for FireStation.cpp:

6.51 COS214-Poject/src/FireStation.h File Reference

```
#include "PublicService.h"
```

Include dependency graph for FireStation.h: This graph shows which files directly or indirectly include this file:

Classes

- class FireStation

  Represents a fire station in the city.

6.52 COS214-Poject/src/FireStationFactory.cpp File Reference

```
#include "FireStationFactory.h"
#include "FireStation.h"
```
Include dependency graph for FireStationFactory.cpp:

6.53 COS214-Poject/src/FireStationFactory.h File Reference

```
#include "PublicServiceFactory.h"
#include "PublicService.h"
```
Include dependency graph for FireStationFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class FireStationFactory

  Factory class for creating FireStation objects.

6.54 COS214-Poject/src/Flat.cpp File Reference

```
#include "Flat.h"
```
Include dependency graph for Flat.cpp:

6.55 COS214-Poject/src/Flat.h File Reference

```
#include "ResidentialBuilding.h"
#include "taxCollector.h"
#include "CityComposite.h"
```
Include dependency graph for Flat.h: This graph shows which files directly or indirectly include this file:

Classes

- class Flat

  Represents a residential building of type Flat.

6.56 COS214-Poject/src/FlatFactory.cpp File Reference

```
#include "FlatFactory.h"
#include "Flat.h"
```
Include dependency graph for FlatFactory.cpp:

6.57 COS214-Poject/src/FlatFactory.h File Reference

```
#include "ResidentialBuildingFactory.h"
```

Include dependency graph for FlatFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class FlatFactory

  Factory class for creating Flat residential buildings.

6.58 COS214-Poject/src/ForwardDeclarations.cpp File Reference

```
#include "ForwardDeclarations.h"
```

Include dependency graph for ForwardDeclarations.cpp:

6.59 COS214-Poject/src/ForwardDeclarations.h File Reference

This graph shows which files directly or indirectly include this file:

6.60 COS214-Poject/src/GameState.cpp File Reference

Implementation of the GameState class for managing game states and command history.

```
#include "GameState.h"
```

Include dependency graph for GameState.cpp:

6.60.1 Detailed Description

Implementation of the GameState class for managing game states and command history.

This file contains the definitions of the methods declared in the GameState class, including creating mementos, restoring state, executing commands, and handling undo/redo functionality.

6.61 COS214-Poject/src/GameState.h File Reference

```
#include <vector>
#include <memory>
#include <algorithm>
#include "Command.h"
#include "GameStateMemento.h"
```

Include dependency graph for GameState.h: This graph shows which files directly or indirectly include this file:

Classes

- class GameState

  Manages the game state and command history.

6.62 COS214-Poject/src/GameStateMemento.cpp File Reference

```
#include "GameStateMemento.h"
```
Include dependency graph for GameStateMemento.cpp:

6.63 COS214-Poject/src/GameStateMemento.h File Reference

```
#include <vector>
#include <memory>
#include "Command.h"
```
Include dependency graph for GameStateMemento.h: This graph shows which files directly or indirectly include this file:

Classes

- class GameStateMemento

  Represents a memento for the GameState, capturing command history and the current command index.

6.64 COS214-Poject/src/Gold.cpp File Reference

```
#include "Gold.h"
```
Include dependency graph for Gold.cpp:

6.65 COS214-Poject/src/Gold.h File Reference

```
#include "IncomeResourceProduct.h"
#include <iostream>
#include <string>
```
Include dependency graph for Gold.h: This graph shows which files directly or indirectly include this file:

Classes

- class Gold

  Class representing the Gold resource.

6.66 COS214-Poject/src/Government.cpp File Reference

```
#include "Government.h"
#include <string>
#include <iostream>
```
Include dependency graph for Government.cpp:

6.67 COS214-Poject/src/Government.h File Reference

```
#include <vector>
#include <string>
#include <map>
```
Include dependency graph for Government.h: This graph shows which files directly or indirectly include this file:

Classes

- class Government

  The Government class manages the state and functionality of the government in the simulation.

6.68 COS214-Poject/src/GreenTechnologyDecorator.cpp File Reference

6.69 COS214-Poject/src/GreenTechnologyDecorator.h File Reference

6.70 COS214-Poject/src/Hospital.cpp File Reference

```
#include "Hospital.h"
#include <iostream>
```
Include dependency graph for Hospital.cpp:

6.71 COS214-Poject/src/Hospital.h File Reference

```
#include "PublicService.h"
#include "taxCollector.h"
```
Include dependency graph for Hospital.h: This graph shows which files directly or indirectly include this file:

Classes

- class Hospital

  Represents a hospital that provides medical services to citizens.

6.72 COS214-Poject/src/HospitalFactory.cpp File Reference

Implementation of the HospitalFactory class.

```
#include "HospitalFactory.h"
#include "Hospital.h"
```

Include dependency graph for HospitalFactory.cpp:

6.72.1 Detailed Description

Implementation of the HospitalFactory class.

This file contains the implementation of the methods defined in the HospitalFactory class, specifically the method for creating Hospital objects.

6.73 COS214-Poject/src/HospitalFactory.h File Reference

Declaration of the HospitalFactory class for creating Hospital objects.

```
#include "PublicServiceFactory.h"
#include "PublicService.h"
```

Include dependency graph for HospitalFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class HospitalFactory

  A factory class for creating Hospital instances.

6.73.1 Detailed Description

Declaration of the HospitalFactory class for creating Hospital objects.

This file defines the HospitalFactory class, which is responsible for creating instances of the Hospital class, a type of public service in the game.

6.74 COS214-Poject/src/House.cpp File Reference

```
#include "House.h"
```

Include dependency graph for House.cpp:

6.75 COS214-Poject/src/House.h File Reference

Header file for the House class, representing a residential building.

```
#include "ResidentialBuilding.h"
#include "taxCollector.h"
#include "CityComposite.h"
```

Include dependency graph for House.h: This graph shows which files directly or indirectly include this file:

Classes

- class House

  Represents a house, which is a type of residential building.

6.75.1 Detailed Description

Header file for the House class, representing a residential building.

6.76 COS214-Poject/src/HouseFactory.cpp File Reference

```
#include "HouseFactory.h"
#include "House.h"
```

Include dependency graph for HouseFactory.cpp:

6.77 COS214-Poject/src/HouseFactory.h File Reference

```
#include "ResidentialBuildingFactory.h"
#include "ResidentialBuilding.h"
```

Include dependency graph for HouseFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class HouseFactory

  Factory class for creating House objects.

6.78 COS214-Poject/src/IncomeResourceProcessor.cpp File Reference

```
#include "IncomeResourceProcessor.h"
```

Include dependency graph for IncomeResourceProcessor.cpp:

6.79 COS214-Poject/src/IncomeResourceProcessor.h File Reference

```
#include "ResourceProcessor.h"
#include "IncomeResourceProduct.h"
#include "Government.h"
#include <memory>
```

Include dependency graph for IncomeResourceProcessor.h: This graph shows which files directly or indirectly include this file:

Classes

- class IncomeResourceProcessor

  Processes and manages storage for income-generating resources.

6.80 COS214-Poject/src/IncomeResourceProduct.cpp File Reference

```
#include "IncomeResourceProduct.h"
```
Include dependency graph for IncomeResourceProduct.cpp:

6.81 COS214-Poject/src/IncomeResourceProduct.h File Reference

```
#include "CityComponent.h"
#include <iostream>
```
Include dependency graph for IncomeResourceProduct.h: This graph shows which files directly or indirectly include this file:

Classes

- class IncomeResourceProduct

  Represents an income-generating resource in the city.

6.82 COS214-Poject/src/IndustrialFactory.h File Reference

```
#include <memory>
#include "Industry.h"
```
Include dependency graph for IndustrialFactory.h:

Classes

- class IndustrialFactory

  Abstract factory class for creating industrial facilities.

6.83 COS214-Poject/src/Industry.cpp File Reference

```
#include "Industry.h"
#include "MapGrid.h"
#include <iostream>
#include <algorithm>
```
Include dependency graph for Industry.cpp:

6.84 COS214-Poject/src/Industry.h File Reference

```
#include "CityComponent.h"
#include "Location.h"
#include "ResourceProcessor.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include <memory>
#include <string>
#include <map>
```
Include dependency graph for Industry.h: This graph shows which files directly or indirectly include this file:

Classes

- class Industry

  Represents an industrial facility in the city.

6.85 COS214-Poject/src/LandMark.cpp File Reference

```
#include "LandMark.h"
```
Include dependency graph for LandMark.cpp:

6.86 COS214-Poject/src/LandMark.h File Reference

Header file for the LandMark class, representing a landmark in the city.

```
#include "CityComponent.h"
#include "UtilityFlyweight.h"
#include <iostream>
#include <string>
#include <memory>
```
Include dependency graph for LandMark.h: This graph shows which files directly or indirectly include this file:

Classes

- class LandMark

  Represents a landmark within the city.

6.86.1 Detailed Description

Header file for the LandMark class, representing a landmark in the city.

6.87 COS214-Poject/src/LandMarkFactory.cpp File Reference

```
#include "LandMarkFactory.h"
```
Include dependency graph for LandMarkFactory.cpp:

6.88 COS214-Poject/src/LandMarkFactory.h File Reference

```
#include <memory>
#include "LandMark.h"
```
Include dependency graph for LandMarkFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class LandMarkFactory

  Abstract factory class for creating landmark objects.

6.89 COS214-Poject/src/Location.cpp File Reference

```
#include "Location.h"
```
Include dependency graph for Location.cpp:

6.90 COS214-Poject/src/Location.h File Reference

```
#include <cmath>
```
Include dependency graph for Location.h: This graph shows which files directly or indirectly include this file:

Classes

- struct Location

  Represents a 2D coordinate location on a grid.

6.91 COS214-Poject/src/main.cpp File Reference

```
#include "CityGame.h"
```
Include dependency graph for main.cpp:

Functions

- int main ()

6.91.1 Function Documentation

6.91.1.1 main()

```
int main ( )
```

Here is the call graph for this function:

6.92 COS214-Poject/src/Malls.cpp File Reference

```
#include "Malls.h"
```
Include dependency graph for Malls.cpp:

6.93 COS214-Poject/src/Malls.h File Reference

```
#include <string>
#include "CommercialBuilding.h"
```

Include dependency graph for Malls.h: This graph shows which files directly or indirectly include this file:

Classes

- class Malls

  Represents a Mall, a type of CommercialBuilding.

6.94 COS214-Poject/src/MallsFactory.cpp File Reference

```
#include "MallsFactory.h"
#include "Malls.h"
```

Include dependency graph for MallsFactory.cpp:

6.95 COS214-Poject/src/MallsFactory.h File Reference

```
#include "CommercialFactory.h"
#include "CommercialBuilding.h"
```

Include dependency graph for MallsFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class MallsFactory

  Factory class for creating Mall buildings.

6.96 COS214-Poject/src/MapGrid.cpp File Reference

```
#include "MapGrid.h"
#include "Industry.h"
#include "ZoneComposite.h"
#include "CityComponent.h"
#include "ResidentialBuilding.h"
#include "CommercialBuilding.h"
#include "Transport.h"
#include "Gold.h"
#include "Diamonds.h"
#include "Coal.h"
#include "Oil.h"
#include "Stone.h"
#include "Wood.h"
#include "Steel.h"
#include "Concrete.h"
#include <sstream>
#include <iomanip>
#include <algorithm>
#include <map>
#include <set>
#include <unordered_set>
#include <random>
```

Include dependency graph for MapGrid.cpp:

6.97 COS214-Poject/src/MapGrid.h File Reference

```
#include "CityComponent.h"
#include "UtilityFlyweight.h"
#include "ResidentialBuilding.h"
#include "CommercialBuilding.h"
#include "ZoneComposite.h"
#include "Industry.h"
#include "Transport.h"
#include "Hospital.h"
#include "FireStation.h"
#include "PublicService.h"
#include "PoliceStation.h"
#include "LandMark.h"
#include "Monument.h"
#include "Park.h"
#include "CulturalCenter.h"
#include <memory>
#include <vector>
#include <string>
#include <random>
#include <set>
#include <algorithm>
#include <map>
#include <array>
#include <sstream>
#include "ForwardDeclarations.h"
#include "Location.h"
```
Include dependency graph for MapGrid.h: This graph shows which files directly or indirectly include this file:

Classes

- class MapGrid
- struct MapGrid::ResourceSpot
- struct MapGrid::ZoneStyle
- struct MapGrid::Cell
- struct MapGrid::PlacementResult

6.98 COS214-Poject/src/MetalFactory.cpp File Reference

```
#include "MetalFactory.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include "Gold.h"
#include "Steel.h"
```
Include dependency graph for MetalFactory.cpp:

6.99 COS214-Poject/src/MetalFactory.h File Reference

```
#include "ResourceFactory.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include <iostream>
#include <memory>
```
Include dependency graph for MetalFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class MetalFactory

  A factory class for creating metal-related resources for income generation and construction.

6.100 COS214-Poject/src/MetalWorkFacility.cpp File Reference

```
#include "MetalWorkFacility.h"
#include "IncomeResourceProcessor.h"
#include "ConstructionResourceProcessor.h"
#include <iostream>
```
Include dependency graph for MetalWorkFacility.cpp:

6.101 COS214-Poject/src/MetalWorkFacility.h File Reference

```
#include "Industry.h"
#include <memory>
#include <map>
```
Include dependency graph for MetalWorkFacility.h: This graph shows which files directly or indirectly include this file:

Classes

- class MetalWorkFacility

  A concrete implementation of an Industry for processing metal resources.

6.102 COS214-Poject/src/ModerateCollectionStrategy.cpp File Reference

```
#include "ModerateCollectionStrategy.h"
```
Include dependency graph for ModerateCollectionStrategy.cpp:

6.103 COS214-Poject/src/ModerateCollectionStrategy.h File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class ModerateCollectionStrategy

  Collection strategy for moderate collection rate.

6.104 COS214-Poject/src/Monument.cpp File Reference

```
#include "Monument.h"
```

Include dependency graph for Monument.cpp:

6.105 COS214-Poject/src/Monument.h File Reference

```
#include "LandMark.h"
```

Include dependency graph for Monument.h: This graph shows which files directly or indirectly include this file:

Classes

- class Monument

  Represents a Monument, a type of LandMark with specific characteristics and utilities.

6.106 COS214-Poject/src/MonumentFactory.cpp File Reference

```
#include "MonumentFactory.h"
#include "Monument.h"
```

Include dependency graph for MonumentFactory.cpp:

6.107 COS214-Poject/src/MonumentFactory.h File Reference

```
#include "LandMarkFactory.h"
#include "LandMark.h"
```

Include dependency graph for MonumentFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class MonumentFactory

  Factory class for creating Monument instances.

6.108 COS214-Poject/src/NeutralState.cpp File Reference

```
#include "NeutralState.h"
#include "NPCManager.h"
#include <iostream>
#include <string>
```

Include dependency graph for NeutralState.cpp:

6.109 COS214-Poject/src/NeutralState.h File Reference

```
#include "NPCManager.h"
#include "NPCState.h"
#include <string>
```
Include dependency graph for NeutralState.h: This graph shows which files directly or indirectly include this file:

Classes

- class NeutralState

  Represents a neutral state in which no specific action is taken by the NPC.

6.110 COS214-Poject/src/Node.cpp File Reference

```
#include "Node.h"
#include <unordered_map>
#include <memory>
```
Include dependency graph for Node.cpp:

6.111 COS214-Poject/src/Node.h File Reference

```
#include "CityComponent.h"
#include "Location.h"
#include "Transport.h"
#include <vector>
#include <unordered_map>
#include <memory>
```
Include dependency graph for Node.h: This graph shows which files directly or indirectly include this file:

Classes

- class Node

  Represents a node within the city grid, which holds a location, an optional city component, and connections to other nodes via transport methods.

6.112 COS214-Poject/src/NPCContext.cpp File Reference

6.113 COS214-Poject/src/NPCContext.h File Reference

6.114 COS214-Poject/src/NPCManager.cpp File Reference

```
#include "NPCManager.h"
#include "Government.h"
#include <utility>
#include <string>
#include <algorithm>
#include <iostream>
```
Include dependency graph for NPCManager.cpp:

6.115 COS214-Poject/src/NPCManager.h File Reference

```
#include <string>
```
Include dependency graph for NPCManager.h: This graph shows which files directly or indirectly include this file:

Classes

- class NPCManager

  Singleton class managing the state and statistics of NPCs.

6.116 COS214-Poject/src/NPCObserver.cpp File Reference

```
#include "NPCObserver.h"
```
Include dependency graph for NPCObserver.cpp:

6.117 COS214-Poject/src/NPCObserver.h File Reference

```
#include <string>
#include <vector>
```
Include dependency graph for NPCObserver.h: This graph shows which files directly or indirectly include this file:

Classes

- class NPCObserver

  Abstract base class for NPC observers.

6.118 COS214-Poject/src/NPCState.h File Reference

```
#include <string>
```
Include dependency graph for NPCState.h: This graph shows which files directly or indirectly include this file:

Classes

- class NPCState

  Abstract base class representing a state in the NPC state machine.

6.119 COS214-Poject/src/NPCSystem.cpp File Reference

```
#include "NPCSystem.h"
```
Include dependency graph for NPCSystem.cpp:

6.120 COS214-Poject/src/NPCSystem.h File Reference

```
#include <string>
#include <memory>
```
Include dependency graph for NPCSystem.h: This graph shows which files directly or indirectly include this file:

Classes

- class CollectionStrategy

  Base class for collection strategies in the NPC system.
- class SlowCollectionStrategy

  Collection strategy for slow collection rate.
- class ModerateCollectionStrategy

  Collection strategy for moderate collection rate.
- class FastCollectionStrategy

  Collection strategy for fast collection rate.
- class WorkerNPC

  Represents a Worker NPC that collects resources.
- class NPCContext

  Factory class to hire Worker NPCs with specific collection strategies.

6.121 COS214-Poject/src/Office.cpp File Reference

```
#include "Office.h"
```
Include dependency graph for Office.cpp:

6.122 COS214-Poject/src/Office.h File Reference

```
#include <string>
#include "CommercialBuilding.h"
```
Include dependency graph for Office.h: This graph shows which files directly or indirectly include this file:

Classes

- class Office

  Represents an office building, a type of commercial building with utility connections.

6.123 COS214-Poject/src/OfficeFactory.cpp File Reference

```
#include "OfficeFactory.h"
#include "Office.h"
```
Include dependency graph for OfficeFactory.cpp:

6.124 COS214-Poject/src/OfficeFactory.h File Reference

```
#include "CommercialFactory.h"
#include "CommercialBuilding.h"
```

Include dependency graph for OfficeFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class OfficeFactory

  Factory class for creating Office buildings.

6.125 COS214-Poject/src/Oil.cpp File Reference

```
#include "Oil.h"
```

Include dependency graph for Oil.cpp:

6.126 COS214-Poject/src/Oil.h File Reference

```
#include "IncomeResourceProduct.h"
#include <iostream>
#include <string>
```

Include dependency graph for Oil.h: This graph shows which files directly or indirectly include this file:

Classes

- class Oil

  Represents an oil resource in the simulation.

6.127 COS214-Poject/src/Park.cpp File Reference

```
#include "Park.h"
```

Include dependency graph for Park.cpp:

6.128 COS214-Poject/src/Park.h File Reference

```
#include "LandMark.h"
```

Include dependency graph for Park.h: This graph shows which files directly or indirectly include this file:

Classes

- class Park

  Represents a park in the city simulation.

6.129 COS214-Poject/src/ParkFactory.cpp File Reference

```
#include "ParkFactory.h"
#include "Park.h"
```
Include dependency graph for ParkFactory.cpp:

6.130 COS214-Poject/src/ParkFactory.h File Reference

```
#include "LandMarkFactory.h"
#include "LandMark.h"
```
Include dependency graph for ParkFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class ParkFactory

  Factory class for creating Park objects.

6.131 COS214-Poject/src/PetroChemicalFacility.cpp File Reference

```
#include "PetroChemicalFacility.h"
```
Include dependency graph for PetroChemicalFacility.cpp:

6.132 COS214-Poject/src/PetroChemicalFacility.h File Reference

```
#include "Industry.h"
#include "IncomeResourceProcessor.h"
#include "ConstructionResourceProcessor.h"
```
Include dependency graph for PetroChemicalFacility.h: This graph shows which files directly or indirectly include this file:

Classes

- class PetrochemicalFacility

  Represents a petrochemical facility that processes oil and concrete.

6.133 COS214-Poject/src/PlantFactory.cpp File Reference

```
#include "PlantFactory.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include "Wood.h"
#include "Coal.h"
```
Include dependency graph for PlantFactory.cpp:

6.134 COS214-Poject/src/PlantFactory.h File Reference

```
#include "ResourceFactory.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include <iostream>
#include <memory>
```
Include dependency graph for PlantFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class PlantFactory

  A factory class for creating resources related to plants.

6.135 COS214-Poject/src/PoliceStation.cpp File Reference

```
#include "PoliceStation.h"
#include <iostream>
```
Include dependency graph for PoliceStation.cpp:

6.136 COS214-Poject/src/PoliceStation.h File Reference

```
#include "PublicService.h"
```
Include dependency graph for PoliceStation.h: This graph shows which files directly or indirectly include this file:

Classes

- class PoliceStation

  Represents a police station public service in the city.

6.137 COS214-Poject/src/PoliceStationFactory.cpp File Reference

```
#include "PoliceStationFactory.h"
#include "PoliceStation.h"
```
Include dependency graph for PoliceStationFactory.cpp:

6.138 COS214-Poject/src/PoliceStationFactory.h File Reference

```
#include "PublicServiceFactory.h"
#include "PublicService.h"
```
Include dependency graph for PoliceStationFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class PoliceStationFactory

  Factory class to create PoliceStation instances.

6.139 COS214-Poject/src/PowerPlant.cpp File Reference

```
#include "PowerPlant.h"
```
Include dependency graph for PowerPlant.cpp:

6.140 COS214-Poject/src/PowerPlant.h File Reference

```
#include "UtilityFlyweight.h"
```
Include dependency graph for PowerPlant.h: This graph shows which files directly or indirectly include this file:

Classes

- class PowerPlant

  Represents a Power Plant in the utility system.

6.141 COS214-Poject/src/ProductiveState.cpp File Reference

```
#include "Government.h"
#include "NPCManager.h"
#include "ProductiveState.h"
#include <iostream>
#include <random>
#include <string>
```
Include dependency graph for ProductiveState.cpp:

6.142 COS214-Poject/src/ProductiveState.h File Reference

```
#include "NPCManager.h"
#include "NPCState.h"
#include <string>
```
Include dependency graph for ProductiveState.h: This graph shows which files directly or indirectly include this file:

Classes

- class ProductiveState

  Represents the productive state of NPCs.

6.143 COS214-Poject/src/PublicService.cpp File Reference

```
#include "PublicService.h"
#include <iostream>
```
Include dependency graph for PublicService.cpp:

6.144 COS214-Poject/src/PublicService.h File Reference

```
#include "CityComponent.h"
#include <iostream>
#include <string>
#include <memory>
```
Include dependency graph for PublicService.h: This graph shows which files directly or indirectly include this file:

Classes

- class PublicService

    Abstract base class representing a public service.

6.145 COS214-Poject/src/PublicServiceFactory.cpp File Reference

```
#include "PublicServiceFactory.h"
```
Include dependency graph for PublicServiceFactory.cpp:

6.146 COS214-Poject/src/PublicServiceFactory.h File Reference

Abstract Factory class for creating PublicService objects.

```
#include <memory>
#include "PublicService.h"
```
Include dependency graph for PublicServiceFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class PublicServiceFactory

    An abstract factory class for creating instances of PublicService.

6.146.1 Detailed Description

Abstract Factory class for creating PublicService objects.

6.147 COS214-Poject/src/PublicTransit.cpp File Reference

6.148 COS214-Poject/src/PublicTransit.h File Reference

6.149 COS214-Poject/src/PublicTransitFactory.cpp File Reference

6.150 COS214-Poject/src/PublicTransitFactory.h File Reference

6.151 COS214-Poject/src/ReactingNPCS.cpp File Reference

```
#include "ReactingNPCS.h"
#include "NPCManager.h"
#include "NPCState.h"
#include "NeutralState.h"
#include "DonationState.h"
#include "RevoltState.h"
#include "ProductiveState.h"
#include "CrimeState.h"
#include "Government.h"
#include <string>
#include <random>
#include <iostream>
```

Include dependency graph for ReactingNPCS.cpp:

6.152 COS214-Poject/src/ReactingNPCS.h File Reference

```
#include "NPCObserver.h"
#include "NPCState.h"
#include "NeutralState.h"
#include <iostream>
#include <string>
```

Include dependency graph for ReactingNPCS.h: This graph shows which files directly or indirectly include this file:

Classes

- class ReactingNPCS

  Concrete observer class representing an NPC that reacts to state changes.

6.153 COS214-Poject/src/README.md File Reference

6.154 COS214-Poject/src/ResidentialBuilding.cpp File Reference

```
#include "ResidentialBuilding.h"
#include "UtilityFlyweight.h"
#include "taxCollector.h"
#include "NPCManager.h"
#include <iostream>
```

Include dependency graph for ResidentialBuilding.cpp:

6.155 COS214-Poject/src/ResidentialBuilding.h File Reference

```
#include "CityComponent.h"
#include <iostream>
#include <string>
#include <memory>
```
Include dependency graph for ResidentialBuilding.h: This graph shows which files directly or indirectly include this file:

Classes

- class ResidentialBuilding

  Represents a residential building within the city.

6.156 COS214-Poject/src/ResidentialBuildingFactory.cpp File Reference

```
#include "ResidentialBuildingFactory.h"
```
Include dependency graph for ResidentialBuildingFactory.cpp:

6.157 COS214-Poject/src/ResidentialBuildingFactory.h File Reference

```
#include <memory>
#include "ResidentialBuilding.h"
```
Include dependency graph for ResidentialBuildingFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class ResidentialBuildingFactory

  Abstract Factory class for creating residential buildings.

6.158 COS214-Poject/src/ResourceFactory.cpp File Reference

```
#include "ResourceFactory.h"
```
Include dependency graph for ResourceFactory.cpp:

6.159 COS214-Poject/src/ResourceFactory.h File Reference

```
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include <memory>
```
Include dependency graph for ResourceFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class ResourceFactory

  Abstract Factory class for creating resources.

6.160 COS214-Poject/src/ResourceProcessor.cpp File Reference

```
#include "ResourceProcessor.h"
```
Include dependency graph for ResourceProcessor.cpp:

6.161 COS214-Poject/src/ResourceProcessor.h File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class ResourceProcessor

  Abstract base class for processing and managing resources.

6.162 COS214-Poject/src/RevoltState.cpp File Reference

```
#include "RevoltState.h"
#include "NPCManager.h"
#include "Government.h"
#include <iostream>
#include <random>
#include <string>
```
Include dependency graph for RevoltState.cpp:

6.163 COS214-Poject/src/RevoltState.h File Reference

```
#include "NPCManager.h"
#include "NPCState.h"
#include <string>
```
Include dependency graph for RevoltState.h: This graph shows which files directly or indirectly include this file:

Classes

- class RevoltState

  Represents the state of NPCs when they are in revolt.

6.164 COS214-Poject/src/Roads.cpp File Reference

```
#include <iostream>
#include <string>
#include "Roads.h"
```

Include dependency graph for Roads.cpp:

6.165 COS214-Poject/src/Roads.h File Reference

```
#include <iostream>
#include <string>
#include "Transport.h"
```

Include dependency graph for Roads.h: This graph shows which files directly or indirectly include this file:

Classes

- class Roads

  A concrete implementation of Transport representing various types of roads.

Enumerations

- enum class roadType {
  Highway , Street , Avenue , Boulevard ,
  Alley , Roundabout , Bridge , Tunnel ,
  PedestrianPath }

  Enum representing different types of roads.

6.165.1 Enumeration Type Documentation

6.165.1.1 roadType

```
enum roadType  [strong]
```

Enum representing different types of roads.

Enumerator

Highway	
Street	
Avenue	
Boulevard	
Alley	
Roundabout	
Bridge	
Tunnel	
PedestrianPath	

6.166 COS214-Poject/src/RoadsFactory.cpp File Reference

```
#include "RoadsFactory.h"
#include "Roads.h"
```
Include dependency graph for RoadsFactory.cpp:

6.167 COS214-Poject/src/RoadsFactory.h File Reference

```
#include "TransportationFactory.h"
#include "Transport.h"
```
Include dependency graph for RoadsFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class RoadsFactory

  Factory class for creating Road transport objects.

6.168 COS214-Poject/src/School.cpp File Reference

```
#include "School.h"
#include <iostream>
```
Include dependency graph for School.cpp:

6.169 COS214-Poject/src/School.h File Reference

```
#include "PublicService.h"
#include <string>
#include <memory>
```
Include dependency graph for School.h: This graph shows which files directly or indirectly include this file:

Classes

- class School

6.170 COS214-Poject/src/SchoolFactory.cpp File Reference

6.171 COS214-Poject/src/SchoolFactory.h File Reference

6.172 COS214-Poject/src/SewageSystem.cpp File Reference

```
#include "SewageSystem.h"
```
Include dependency graph for SewageSystem.cpp:

6.173 COS214-Poject/src/SewageSystem.h File Reference

```
#include "UtilityFlyweight.h"
```
Include dependency graph for SewageSystem.h: This graph shows which files directly or indirectly include this file:

Classes

- class SewageSystem

  A concrete UtilityFlyweight that represents a sewage management utility.

6.174 COS214-Poject/src/Shops.cpp File Reference

```
#include "Shops.h"
```
Include dependency graph for Shops.cpp:

6.175 COS214-Poject/src/Shops.h File Reference

```
#include <string>
#include "CommercialBuilding.h"
```
Include dependency graph for Shops.h: This graph shows which files directly or indirectly include this file:

Classes

- class Shops

  Represents a commercial building of type Shops.

6.176 COS214-Poject/src/ShopsFactory.cpp File Reference

```
#include "ShopsFactory.h"
#include "Shops.h"
```
Include dependency graph for ShopsFactory.cpp:

6.177 COS214-Poject/src/ShopsFactory.h File Reference

```
#include "CommercialFactory.h"
#include "CommercialBuilding.h"
```
Include dependency graph for ShopsFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class ShopsFactory

  Factory class for creating shop buildings.

6.178 COS214-Poject/src/SlowCollectionStrategy.cpp File Reference

```
#include "SlowCollectionStrategy.h"
```
Include dependency graph for SlowCollectionStrategy.cpp:

6.179 COS214-Poject/src/SlowCollectionStrategy.h File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class SlowCollectionStrategy

  Collection strategy for slow collection rate.

6.180 COS214-Poject/src/Steel.cpp File Reference

```
#include "Steel.h"
```
Include dependency graph for Steel.cpp:

6.181 COS214-Poject/src/Steel.h File Reference

```
#include "ConstructionResourceProduct.h"
#include <iostream>
#include <string>
```
Include dependency graph for Steel.h: This graph shows which files directly or indirectly include this file:

Classes

- class Steel

  Represents the Steel resource in the city simulation.

6.182 COS214-Poject/src/Stone.cpp File Reference

```
#include "Stone.h"
```
Include dependency graph for Stone.cpp:

6.183 COS214-Poject/src/Stone.h File Reference

```
#include "ConstructionResourceProduct.h"
#include <iostream>
#include <string>
```
Include dependency graph for Stone.h: This graph shows which files directly or indirectly include this file:

Classes

- class Stone

  Represents a construction resource product of type Stone.

6.184 COS214-Poject/src/StoneFactory.cpp File Reference

```
#include "StoneFactory.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include "Diamonds.h"
#include "Stone.h"
```

Include dependency graph for StoneFactory.cpp:

6.185 COS214-Poject/src/StoneFactory.h File Reference

```
#include "ResourceFactory.h"
#include "IncomeResourceProduct.h"
#include "ConstructionResourceProduct.h"
#include <iostream>
#include <memory>
```

Include dependency graph for StoneFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class StoneFactory

  Factory class for creating stone resources, providing both income and construction products.

6.186 COS214-Poject/src/taxCollector.cpp File Reference

```
#include "taxCollector.h"
```

Include dependency graph for taxCollector.cpp:

6.187 COS214-Poject/src/taxCollector.h File Reference

```
#include "Government.h"
```

Include dependency graph for taxCollector.h: This graph shows which files directly or indirectly include this file:

Classes

- class taxCollector

  Abstract class representing a tax collector that visits various building types to collect taxes.

6.188 COS214-Poject/src/Townhouse.cpp File Reference

```
#include "Townhouse.h"
```
Include dependency graph for Townhouse.cpp:

6.189 COS214-Poject/src/Townhouse.h File Reference

```
#include "ResidentialBuilding.h"
#include "taxCollector.h"
#include "CityComposite.h"
```
Include dependency graph for Townhouse.h: This graph shows which files directly or indirectly include this file:

Classes

- class Townhouse

  A concrete class representing a townhouse, derived from ResidentialBuilding.

6.190 COS214-Poject/src/TownhouseFactory.cpp File Reference

```
#include "TownhouseFactory.h"
#include "Townhouse.h"
```
Include dependency graph for TownhouseFactory.cpp:

6.191 COS214-Poject/src/TownhouseFactory.h File Reference

```
#include "ResidentialBuildingFactory.h"
#include "ResidentialBuilding.h"
```
Include dependency graph for TownhouseFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class TownhouseFactory

  Factory class for creating Townhouse residential buildings.

6.192 COS214-Poject/src/Trains.cpp File Reference

```
#include <iostream>
#include <string>
#include "Trains.h"
```
Include dependency graph for Trains.cpp:

6.193 COS214-Poject/src/Trains.h File Reference

```
#include <memory>
#include <string>
#include "Transport.h"
```
Include dependency graph for Trains.h: This graph shows which files directly or indirectly include this file:

Classes

- class Trains

  A concrete implementation of the Transport class representing a train.

6.194 COS214-Poject/src/TrainsFactory.cpp File Reference

```
#include "TrainsFactory.h"
#include "Trains.h"
```
Include dependency graph for TrainsFactory.cpp:

6.195 COS214-Poject/src/TrainsFactory.h File Reference

```
#include "TransportationFactory.h"
#include "Transport.h"
```
Include dependency graph for TrainsFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class TrainsFactory

  Factory class for creating train transport instances.

6.196 COS214-Poject/src/Transport.cpp File Reference

```
#include <iostream>
#include "Transport.h"
```
Include dependency graph for Transport.cpp:

6.197 COS214-Poject/src/Transport.h File Reference

```
#include <iostream>
#include <memory>
#include <string>
#include "CityComponent.h"
```
Include dependency graph for Transport.h: This graph shows which files directly or indirectly include this file:

Classes

- class Transport

  Abstract class representing a generic transport.

6.198 COS214-Poject/src/TransportationFactory.cpp File Reference

```
#include "TransportationFactory.h"
```
Include dependency graph for TransportationFactory.cpp:

6.199 COS214-Poject/src/TransportationFactory.h File Reference

```
#include <memory>
#include "Transport.h"
```
Include dependency graph for TransportationFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class TransportationFactory

  Abstract Factory class for creating transportation objects.

6.200 COS214-Poject/src/UtilitiesTesting.cpp File Reference

6.201 COS214-Poject/src/UtilityDecorator.cpp File Reference

6.202 COS214-Poject/src/UtilityDecorator.h File Reference

6.203 COS214-Poject/src/UtilityFactory.cpp File Reference

```
#include "UtilityFactory.h"
```
Include dependency graph for UtilityFactory.cpp:

6.204 COS214-Poject/src/UtilityFactory.h File Reference

```
#include "UtilityFlyweight.h"
#include "PowerPlant.h"
#include "SewageSystem.h"
#include "WasteManagement.h"
#include "WaterSupply.h"
#include <unordered_map>
#include <string>
#include <memory>
#include <stdexcept>
```
Include dependency graph for UtilityFactory.h: This graph shows which files directly or indirectly include this file:

Classes

- class UtilityFactory

  Factory class for creating and managing utility flyweight instances.

6.205 COS214-Poject/src/UtilityFlyweight.cpp File Reference

```
#include "UtilityFlyweight.h"
```

Include dependency graph for UtilityFlyweight.cpp:

6.206 COS214-Poject/src/UtilityFlyweight.h File Reference

```
#include "CityComponent.h"
#include <iostream>
#include <string>
#include <map>
#include <memory>
```

Include dependency graph for UtilityFlyweight.h: This graph shows which files directly or indirectly include this file:

Classes

- class UtilityFlyweight

  Abstract base class for utility components, providing shared functionality.

6.207 COS214-Poject/src/WasteManagement.cpp File Reference

```
#include "WasteManagement.h"
```

Include dependency graph for WasteManagement.cpp:

6.208 COS214-Poject/src/WasteManagement.h File Reference

```
#include "UtilityFlyweight.h"
```

Include dependency graph for WasteManagement.h: This graph shows which files directly or indirectly include this file:

Classes

- class WasteManagement

  Concrete class for managing waste utility in the city.

6.209 COS214-Poject/src/WaterSupply.cpp File Reference

```
#include "WaterSupply.h"
```
Include dependency graph for WaterSupply.cpp:

6.210 COS214-Poject/src/WaterSupply.h File Reference

```
#include "UtilityFlyweight.h"
#include <memory>
#include <map>
#include <string>
```
Include dependency graph for WaterSupply.h: This graph shows which files directly or indirectly include this file:

Classes

- class WaterSupply

  Represents a water supply utility that can be connected to city components within a specific radius.

6.211 COS214-Poject/src/Wood.cpp File Reference

```
#include "Wood.h"
```
Include dependency graph for Wood.cpp:

6.212 COS214-Poject/src/Wood.h File Reference

```
#include "ConstructionResourceProduct.h"
#include <iostream>
#include <string>
```
Include dependency graph for Wood.h: This graph shows which files directly or indirectly include this file:

Classes

- class Wood

  Represents a construction resource product of type Wood.

6.213 COS214-Poject/src/WoodAndCoalPlant.cpp File Reference

```
#include "WoodAndCoalPlant.h"
```
Include dependency graph for WoodAndCoalPlant.cpp:

6.214 COS214-Poject/src/WoodAndCoalPlant.h File Reference

```
#include "Industry.h"
#include "IncomeResourceProcessor.h"
#include "ConstructionResourceProcessor.h"
```

Include dependency graph for WoodAndCoalPlant.h: This graph shows which files directly or indirectly include this file:

Classes

- class WoodAndCoalPlant

  Concrete Industry for processing lumber and coal resources.

6.215 COS214-Poject/src/WorkerNPC.cpp File Reference

6.216 COS214-Poject/src/WorkerNPC.h File Reference

6.217 COS214-Poject/src/ZoneComposite.cpp File Reference

```
#include "ZoneComposite.h"
#include "CityComponent.h"
#include <vector>
#include <string>
```

Include dependency graph for ZoneComposite.cpp:

6.218 COS214-Poject/src/ZoneComposite.h File Reference

```
#include "CityComponent.h"
#include "Location.h"
#include "taxCollector.h"
#include <vector>
#include <memory>
#include <string>
#include <algorithm>
```

Include dependency graph for ZoneComposite.h: This graph shows which files directly or indirectly include this file:

Classes

- class ZoneComposite

  A composite class representing a zone in the city, containing multiple buildings and managing their operations.

Index