Warehouse Class Documentation

Description:

The Warehouse class represents a storage facility for materials. It manages the storage, retrieval, and information about various materials stored within it. The warehouse has a fixed capacity and can store materials of different types.

Dependencies:

- Requires the AMaterial class for representing materials.
- Requires the Observer class for observing material changes. // not yet implemented

```
class Warehouse

{

private:

static size_t mIDCounter; // Counter for assigning unique IDs to warehouses

const size_t mId; // Unique ID of the warehouse

const std::string mName; // Name of the warehouse

size_t mSize; // Current size of the warehouse

size_t mCapacity; // Maximum capacity of the warehouse

std::unordered_map<MaterialType, std::vector<AMaterial*>> mWarehouse; //
        Storage container for materials

public:

// Constructor

Warehouse(std::string Name);

// Displays information about the materials in the warehouse

void Information();

// Adds a material to the warehouse

void SetMaterial(AMaterial* Material);

// Retrieves a specified quantity of material from the warehouse

AMaterial* GetMaterial(MaterialType Type, size_t Quantity);

// Returns the name of the warehouse

std::string GetName();
```

// Returns the number of empty places in the warehouse

size_t EmptyPlaces();

// Checks if the warehouse is empty

bool isEmpty();

// Checks if the warehouse is full

bool isFull();

// Destructor

~Warehouse();

};

Member Variables:

- mIDCounter: Static counter to assign unique IDs to warehouses.
- mId: Unique ID assigned to the warehouse.
- mName: Name of the warehouse.
- mSize: Current size of the warehouse (number of materials stored).
- mCapacity: Maximum capacity of the warehouse.
- mWarehouse: Unordered map storing materials categorized by their types.

Member Functions:

- **Constructor:**
  - Warehouse(std::string Name): Initializes a new instance of the Warehouse class with the given name. Increments the ID counter to assign a unique ID to the warehouse.
- **Information:**
  - void Information(): Displays information about the materials stored in the warehouse, including their types, quantities, and other relevant details.
- **SetMaterial:**
  - void SetMaterial(AMaterial* Material): Adds a material to the warehouse. If a material of the same type already exists, it attempts to combine them if possible, otherwise, it adds a new material.
- **GetMaterial:**
  - AMaterial* GetMaterial(MaterialType Type, size_t Quantity): Retrieves a specified quantity of material from the warehouse. If the requested quantity exceeds the available quantity, an exception is thrown.
- **GetName:**
  - std::string GetName(): Returns the name of the warehouse.
  -

- **EmptyPlaces:**
  - size_t EmptyPlaces(): Returns the number of empty places (capacity - size) in the warehouse.
- **isEmpty:**
  - bool isEmpty(): Checks if the warehouse is empty.
- **isFull:**
  - bool isFull(): Checks if the warehouse is full.
- **Destructor:**
  - ~Warehouse(): Decrements the ID counter when the warehouse is destroyed.
- **Error Handling:**
  - Throws exceptions or returns nullptr in case of errors such as insufficient materials, exceeding capacity, or an empty warehouse.

Notes:

- The warehouse has a temporary capacity defined by **TemporaryCapacity**, which can be modified as needed.
- The **AMaterial** class represents the materials stored in the warehouse and is assumed to be implemented elsewhere.

AMaterial Class Documentation

Description:

The AMaterial class represents a generic material with properties such as type, quantity, capacity, name, description, and icon. It provides methods for managing the material's quantity, accessing its properties, and printing information about the material.

Dependencies:

- Requires the <string> header for string manipulation.
- Relies on the MaterialType enumeration for specifying material types.

```
enum class MaterialType
{
Gold,
Silver,
Metal
};

class AMaterial
{
private:
MaterialType mType;
size_t mCapacity;
size_t mQuantity;
std::string mName;
std::string mDescription;
std::string mIcon;

public:
// Constructor
AMaterial(const MaterialType Type, const size_t Quantity = 0, const std::string     Name = "No Name",
        const std::string Description = "No Description", const std::string Icon = "No Icon");

// Sets the material quantity
size_t SetMaterial(const size_t& Quantity);

// Retrieves a specified quantity of material
size_t GetMaterial(const size_t& Quantity);
// Retrieves all available material
size_t GetMaterial();

// Returns the empty space available for storing more material
size_t GetEmptyPlace();

// Resets the material quantity to zero
void ResetMaterialQuantity();

// Returns the maximum capacity of the material
size_t GetCapacity();

// Returns the current quantity of the material
size_t GetQuantity();

// Returns the name of the material
std::string GetName();

// Returns the description of the material
std::string GetDescription();

// Returns the icon associated with the material
std::string GetIcon();

// Const versions of accessor methods
size_t GetCapacity() const;
size_t GetQuantity() const;
MaterialType GetType() const;
std::string GetName() const;
std::string GetDescription() const;
std::string GetIcon() const;

// Prints all information about the material
void HelperPrintAll();

// Returns a string representation of the material type
std::string MaterialIdentifier(MaterialType type);
};
```

Member Variables:

- **mType**: Type of the material (enumeration).
- **mCapacity**: Maximum capacity of the material.
- **mQuantity**: Current quantity of the material.
- **mName**: Name of the material.
- **mDescription**: Description of the material.
- **mIcon**: Icon associated with the material.

Member Functions:

- **Constructor:**
  - **AMaterial(const MaterialType Type, const size_t Quantity, const std::string Name, const std::string Description, const std::string Icon)**: Initializes a new instance of the **AMaterial** class with the given type, quantity, name, description, and icon.
- **SetMaterial:**
  - **size_t SetMaterial(const size_t& Quantity)**: Sets the quantity of the material. If the quantity exceeds the capacity, returns the excess quantity.
- **GetMaterial:**
  - **size_t GetMaterial(const size_t& Quantity)**: Retrieves a specified quantity of material. If the requested quantity exceeds the available quantity, an exception is thrown.
  - **size_t GetMaterial()**: Retrieves all available material.
- **GetEmptyPlace:**
  - **size_t GetEmptyPlace()**: Returns the empty space available for storing more material.
- **ResetMaterialQuantity:**
  - **void ResetMaterialQuantity()**: Resets the material quantity to zero.
- **GetCapacity, GetQuantity, GetName, GetDescription, GetIcon:**
  - Accessor methods for retrieving the capacity, quantity, name, description, and icon of the material.
- **Const Accessor Methods:**
  - Const versions of accessor methods to retrieve the properties without modifying the object.
- **HelperPrintAll:**
  - **void HelperPrintAll()**: Prints all information about the material, including its name, description, icon, type, quantity, and capacity.
- **MaterialIdentifier:**
  - **std::string MaterialIdentifier(MaterialType type)**: Returns a string representation of the material type.

Error Handling:

- Throws exceptions in case of errors such as exceeding capacity or insufficient materials.

Notes:

- The material type is specified using the **MaterialType** enumeration.
- The maximum capacity of the material is defined by **DMaxCapacity**, which can be modified as needed.
- The **HelperPrintAll** method provides a convenient way to print all information about the material.

Player Class Documentation

Description:

The **Player** class represents a player entity in a game context. Players can interact with materials, take materials from warehouses, transfer materials between warehouses, and manage their own inventory.

Dependencies:

- Relies on the **AMaterial** class for representing materials.
- Requires the **Warehouse** class for managing material storage.

```
class Player
{
private:
size_t mID;
std::string mName;
AMaterial* mMaterial; // Player's inventory material
static size_t mIDCounter;
public:
// Constructor
Player(std::string);

// Methods for taking materials from warehouses
void TakeMaterial(AMaterial*);
void TakeMaterial(AMaterial*, const size_t);

// Sets player's material to a warehouse
void SetToWerehouse(Warehouse*);

// Returns a copy instance of player's material
```

AMaterial GetMaterialCopyInstance();*

// Sends materials from one warehouse to another

void SMToAnotherWarehouse(Warehouse, Warehouse*, const MaterialType&, size_t = 0);*

};

Member Variables:

- **mID**: Unique identifier for the player.
- **mName**: Name of the player.
- **mMaterial**: Pointer to the material held by the player.
- **mIDCounter**: Static counter for assigning unique IDs to players.

Member Functions:

- **Constructor:**
  - **Player(std::string Name)**: Initializes a new instance of the **Player** class with the given name. Increments the ID counter to assign a unique ID to the player.
- **TakeMaterial:**
  - **void TakeMaterial(AMaterial*)**: Takes a material from a warehouse and stores it in the player's inventory.
  - **void TakeMaterial(AMaterial*, const size_t)**: Takes a specified quantity of a material from a warehouse and stores it in the player's inventory.
- **SetToWerehouse:**
  - **void SetToWerehouse(Warehouse*)**: Sets the player's material to a warehouse.
- **GetMaterialCopyInstance:**
  - **AMaterial* GetMaterialCopyInstance()**: Returns a copy instance of the player's material.

- **SMToAnotherWarehouse:**
  - **void SMToAnotherWarehouse(Warehouse*, Warehouse*, const MaterialType&, size_t)**: Sends materials from one warehouse to another. Transfers materials of a specified type and quantity from one warehouse to another.

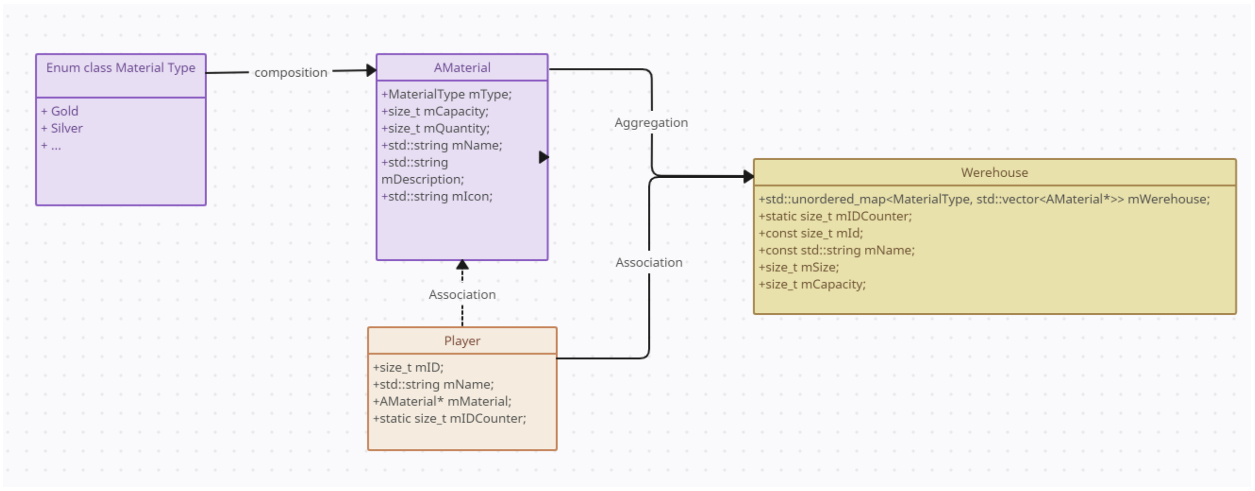
Error Handling:

- Throws exceptions in case of errors such as receiving a **nullptr** argument, attempting to hold multiple types of materials simultaneously, holding nothing while attempting to transfer to a warehouse, and attempting to transfer materials from an empty or **nullptr** warehouse.

Notes:

- The player's inventory is represented by the **mMaterial** member variable.
- Materials are transferred between warehouses using the **SMToAnotherWarehouse** method.
- The player's inventory can hold only one type of material at a time, ensuring simplicity and consistency in inventory management.

UML Diagram:


Enum class Material Type
+ Gold
+ Silver
+ ...
composition
AMaterial
+MaterialType mType;
+size_t mCapacity;
+size_t mQuantity;
+std::string mName;
+std::string mDescription;
+std::string mIcon;
Aggregation
Association
Association
Werehouse
+std::unordered_map<MaterialType, std::vector<AMaterial*>> mWerehouse;
+static size_t mIDCounter;
+const size_t mId;
+const std::string mName;
+size_t mSize;
+size_t mCapacity;
Player
+size_t mID;
+std::string mName;
+AMaterial* mMaterial;
+static size_t mIDCounter;