Sure! Let's structure the **Amazon Locker System** code and explanation similarly to the **Amazon Transportation System** you provided. This will cover all aspects of the locker system while ensuring clarity, organization, and an understanding of the relationships between classes.

1. **Problem Statement: Designing Amazon Locker System**

The Amazon Locker system allows users to store and retrieve packages securely. It manages locker assignments, ensures efficient package retrieval, and handles different package sizes, allowing for scalability and maintainability.

2. **System Requirements**

- **Package Management**: The system should allow for the storage and retrieval of packages in lockers.
- **Locker Management**: It should manage lockers of different sizes (small, medium, large) and assign them based on package sizes.
- **Optimal Locker Assignment**: The system should find the optimal locker for a package based on its size.
- **Clean-Up**: Dynamically allocated lockers should be properly released when the system is destroyed.

3. **Solution Overview**

The Amazon Locker System consists of several classes:

1. **Package**: Represents a package with properties such as size and a unique ID.
2. **Locker**: Represents a locker that can store a package, with properties for size and availability.
3. **PickupLocation**: Manages the overall operations, including assigning packages to lockers, retrieving packages, and managing available lockers.

4. **Classes and Functions**

1. **Package Class**:
    - **Properties**: `size`, `id`.
    - **Methods**: Constructor to initialize a package with size and ID.
2. **Locker Class**:
    - **Properties**: `size`, `id`, `storedPackage`.
    - **Methods**:
        - `storePackage(Package*)`: Stores a package in the locker.
        - `releasePackage()`: Releases the stored package from the locker.
        - `isEmpty()`: Checks if the locker is empty.
        - `getSize()`: Returns the size of the locker.
        - `getId()`: Returns the locker ID.
3. **PickupLocation Class**:
    - **Properties**: `availableLockers`, `packageMap`.
    - **Methods**:

- `assignPackage(Package*)`: Assigns a package to the first available locker of matching size or larger.
- `retrievePackage(const string&)`: Retrieves a package based on its ID.
- `findOptimalLocker(Package*)`: Finds the optimal locker for a package.
- Destructor to clean up dynamically allocated lockers.

Complete Code

```
#include <iostream>
#include <unordered_map>
#include <queue>
#include <string>

using namespace std;

// Enum for Package Sizes
enum Size { SMALL, MEDIUM, LARGE };

// Class representing a package
class Package {
public:
    Size size;           // Size of the package
    string id;           // Unique identifier for the package
    string authCode;     // Authentication code for retrieving the package

    // Constructor to initialize the package with a size and generate random ID
and authCode
    Package(Size sz, const string& code) : size(sz), id(to_string(rand())),
authCode(code) {}

    Size getSize() const {
        return size;
    }

    string getId() const {
        return id;
    }

    string getAuthCode() const {
        return authCode;
    }
};

// Class representing a locker
class Locker {
private:
    Size size;            // Size of the locker
    string id;            // Unique identifier for the locker
    Package* storedPackage; // Pointer to the stored package

public:
```

```
    Locker(Size sz) : size(sz), storedPackage(nullptr), id(to_string(rand())) {}

    void storePackage(Package* package) {
        storedPackage = package;
    }

    Package* releasePackage() {
        Package* package = storedPackage;
        storedPackage = nullptr;
        return package;
    }

    bool isEmpty() const {
        return storedPackage == nullptr;
    }

    Size getSize() const {
        return size;
    }

    string getId() const {
        return id;
    }
};

// Class managing locker assignments and retrieval
class PickupLocation {
private:
    unordered_map<Size, queue<Locker*>> availableLockers; // Map of available
lockers by size
    unordered_map<string, Locker*> packageMap;            // Map of package ID to
locker

    // Function to get the next locker size in case current size is unavailable
    Size getNextSize(Size sz) {
        if (sz == SMALL) return MEDIUM;
        if (sz == MEDIUM) return LARGE;
        return LARGE;
    }

public:
    // Constructor to initialize the lockers for each size
    PickupLocation(unordered_map<Size, int> lockerSizes) {
        for (const auto& entry : lockerSizes) {
            Size lockerSize = entry.first;
            int count = entry.second;
            queue<Locker*> lockerQueue;
            for (int i = 0; i < count; i++) {
                lockerQueue.push(new Locker(lockerSize));
            }
            availableLockers[lockerSize] = lockerQueue;
        }
    }
```

```
    // Assign a package to an available locker based on its size
    Locker* assignPackage(Package* package) {
        Size currentSize = package->getSize();
        while (true) {
            if (!availableLockers[currentSize].empty()) {
                Locker* locker = availableLockers[currentSize].front();
                availableLockers[currentSize].pop();
                locker->storePackage(package);
                packageMap[package->getId()] = locker;
                return locker;
            }
            currentSize = getNextSize(currentSize);
            if (currentSize > LARGE) break;
        }
        return nullptr; // No locker available
    }

    // Retrieve a package by verifying the authentication code
    Package* retrievePackage(const string& packageId, const string&
providedAuthCode) {
        if (packageMap.find(packageId) == packageMap.end()) return nullptr; //
Package not found

        Locker* locker = packageMap[packageId];
        Package* package = locker->releasePackage();

        // Verify the provided authentication code
        if (package->getAuthCode() == providedAuthCode) {
            availableLockers[locker->getSize()].push(locker); // Return locker to
available pool
            packageMap.erase(packageId); // Remove from package map
            return package;
        } else {
            // If authentication fails, re-store the package and return nullptr
            locker->storePackage(package);
            return nullptr;
        }
    }

    ~PickupLocation() {
        // Clean up dynamically allocated lockers
        for (auto& entry : availableLockers) {
            while (!entry.second.empty()) {
                delete entry.second.front();
                entry.second.pop();
            }
        }
    }
};

// Main function demonstrating the enhanced Amazon Locker System
int main() {
    unordered_map<Size, int> lockerSizes = {
        { SMALL, 2 },
```

```
        { MEDIUM, 2 },
        { LARGE, 1 }
    };

    PickupLocation location(lockerSizes); // Create pickup location

    // Create packages with unique authentication codes
    Package* package1 = new Package(SMALL, "1234");
    Package* package2 = new Package(MEDIUM, "5678");
    Package* package3 = new Package(SMALL, "9101");
    Package* package4 = new Package(LARGE, "1121");

    // Assign packages to lockers
    Locker* locker1 = location.assignPackage(package1);
    Locker* locker2 = location.assignPackage(package2);
    Locker* locker3 = location.assignPackage(package3);
    Locker* locker4 = location.assignPackage(package4);

    // Output assignments
    if (locker1) cout << "Package 1 assigned to locker: " << locker1->getId() <<
endl;
    if (locker2) cout << "Package 2 assigned to locker: " << locker2->getId() <<
endl;
    if (locker3) cout << "Package 3 assigned to locker: " << locker3->getId() <<
endl;
    if (locker4) cout << "Package 4 assigned to locker: " << locker4->getId() <<
endl;

    // Attempt to retrieve a package with a valid authentication code
    Package* retrievedPackage = location.retrievePackage(package1->getId(),
"1234");
    if (retrievedPackage) {
        cout << "Successfully retrieved package: " << retrievedPackage->getId() <<
endl;
    } else {
        cout << "Authentication failed for package 1." << endl;
    }

    // Attempt to retrieve a package with an incorrect authentication code
    retrievedPackage = location.retrievePackage(package2->getId(), "wrong-code");
    if (retrievedPackage) {
        cout << "Successfully retrieved package: " << retrievedPackage->getId() <<
endl;
    } else {
        cout << "Authentication failed for package 2." << endl;
    }

    // Clean up
    delete package1;
    delete package2;
    delete package3;
    delete package4;
```

```
    return 0;
}
```

5. **Explanation of the Code**

- **Package Class**: Represents a package with attributes such as size and a unique ID. It includes a constructor to initialize a package with a size and generate a random ID.
- **Locker Class**: Represents a locker that can store a package. It includes methods to store and release packages, check if the locker is empty, and retrieve the locker size and ID.
- **PickupLocation Class**: Manages the lockers and package assignments. It includes methods to assign packages to lockers based on size, retrieve packages, and find the optimal locker for a package. It also cleans up dynamically allocated lockers upon destruction.

This solution implements a comprehensive Amazon Locker System, providing efficient management of packages and lockers while ensuring smooth assignment and retrieval processes.