Problem Statement: Amazon Transportation System with Priority Queue

The goal is to design an optimized transportation system for Amazon that handles operations like labeling, packaging, and delivery while also managing high-traffic periods (e.g., Diwali). The system prioritizes express packages over standard ones using a **priority queue**, ensuring timely deliveries.

---

1. **System Requirements and Solutions**

1. **Labeling**
    - *Solution*: Use a `LabelGenerator` class to create unique labels for each package.
2. **Packaging**
    - *Solution*: The `Package` class represents different package types (standard, express) and generates unique tracking labels.
3. **Delivery Management with Prioritization**
    - *Solution*: Implement a `Delivery` system that handles adding packages and updating delivery statuses. A **priority queue** ensures express packages are processed before standard ones.
4. **Traffic Handling (Peak Periods)**
    - *Solution*: Use a priority queue to prioritize express packages, ensuring timely delivery even during traffic spikes.
5. **International Deliveries**
    - *Solution*: The `TrackingSystem` class tracks package locations and extends for international deliveries with region-based data.

---

2. **High-Level Design**

- **Package**: Represents the physical package with attributes like weight and type.
- **LabelGenerator**: Assigns unique labels to packages.
- **Delivery**: Manages package delivery and uses a priority queue for express package prioritization.
- **TrackingSystem**: Tracks real-time package locations across regions.

---

3. **Open-Ended Questions from the Interviewer**

1. **How would you handle 1 million packages during peak times?**
   Solution: Use multithreading to parallelize delivery operations, implement a distributed database for scalability, and add a caching mechanism for frequent lookups.
2. **How would you handle express package prioritization?**
   Solution: Use a priority queue, where express packages get higher priority over standard ones, ensuring they are delivered on time.

---

4. **Data Structures Used**

1. **Priority Queue**
    - Used to prioritize express packages.
    - **Time Complexity**: Insertion in O(log n), extraction in O(log n).
2. **Map**
    - Used for package status and location tracking.
    - **Time Complexity**: O(1) for lookups and updates.

---

5. **Complete C++ Code Implementation**

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

using namespace std;

// Enum to represent priority types for packages
enum PackagePriority { STANDARD, EXPRESS };

// Base class for packages
class PackageBase {
public:
    virtual ~PackageBase() {}
    virtual string generateLabel() = 0;  // Pure virtual function for generating
labels
};

// Concrete class representing a package
class Package : public PackageBase {
private:
    string packageType;  // Standard or Express
    float weight;
    PackagePriority priority;  // Stores the priority of the package
    int deadline;  // Time left until the deadline (lower value = more urgent)

public:
    Package(string type, float w, int d) {
        packageType = type;
        weight = w;
        deadline = d;
        priority = (packageType == "Express") ? EXPRESS : STANDARD;  // Set
priority based on package type
    }

    string generateLabel() override {
        return "Label-" + packageType + "-" + to_string(weight);
```

```
    }

    string getType() {
        return packageType;
    }

    PackagePriority getPriority() {
        return priority;
    }

    int getDeadline() const {
        return deadline;
    }
};

class ComparePriority {
public:
    bool operator()(Package* p1, Package* p2) {
        // First compare by priority, then by deadline
        if (p1->getPriority() == p2->getPriority()) {
            return p1->getDeadline() > p2->getDeadline();  // Shorter deadline
gets higher priority
        }
        return p1->getPriority() > p2->getPriority();  // Higher priority (Express
> Standard)
    }
};

// Class to manage the delivery process with prioritization for express deliveries
class Delivery {
private:
    map<string, string> packageStatus;  // Tracks packageID and its status
    priority_queue<Package*, vector<Package*>, ComparePriority> packageQueue;  //
Priority queue for packages

public:
    void addPackage(Package* package, string status) {
        string packageID = package->generateLabel();
        packageStatus[packageID] = status;  // Add package with initial status
        packageQueue.push(package);  // Add package to the priority queue based on
its type (Express > Standard)
    }

    void updateStatus(string packageID, string newStatus) {
        if (packageStatus.find(packageID) != packageStatus.end()) {
            packageStatus[packageID] = newStatus;  // Update package status
        }
    }

    string getStatus(string packageID) {
        return packageStatus[packageID];  // Get current status of the package
    }

    string processNextPackage() {
```

```
        if (!packageQueue.empty()) {
            Package* nextPackage = packageQueue.top();
            packageQueue.pop();  // Process the highest priority package
            return nextPackage->generateLabel();
        }
        return "No Packages in Queue";
    }
};

// Class for tracking packages in real-time with international support
class TrackingSystem {
private:
    // Using a single map to track both location and region
    map<string, pair<string, string>> packageData;  // Tracks packageID and a pair
(location, region)

public:
    // Method to track the package with location and optional region info
    void trackPackage(string packageID, string location, string region = "") {
        packageData[packageID] = {location, region};  // Store location and region
in a pair
    }

    // Method to get tracking info (location + region, if available)
    string getTrackingInfo(string packageID) {
        if (packageData.find(packageID) != packageData.end()) {
            string location = packageData[packageID].first;  // Get location from
the pair
            string region = packageData[packageID].second;   // Get region from
the pair
            if (!region.empty()) {
                location += " (" + region + ")";  // Append region info if
available
            }
            return location;
        }
        return "Package not found";  // Return if packageID doesn't exist
    }
};

// Example of using the system
int main() {
    // Create packages with priority and deadlines
    Package expressPackage("Express", 5.5, 2);  // Express package with 2-hour
deadline
    Package standardPackage("Standard", 2.3, 4);  // Standard package with 4-hour
deadline
    Package urgentStandardPackage("Standard", 1.2, 1);  // Standard package with
1-hour deadline

    // Delivery system
    Delivery delivery;
    delivery.addPackage(&expressPackage, "In Transit");
    delivery.addPackage(&standardPackage, "In Transit");
```

```
    // Process next package (Express should be processed first due to higher
priority)
    cout << "Next Package to Process: " << delivery.processNextPackage() << endl;
    cout << "Next Package to Process: " << delivery.processNextPackage() << endl;

    // Tracking system
    TrackingSystem tracking;
    tracking.trackPackage(expressPackage.generateLabel(), "Mumbai", "India");
    tracking.trackPackage(standardPackage.generateLabel(), "New York", "USA");
    cout << "Express Package Location: " <<
tracking.getTrackingInfo(expressPackage.generateLabel()) << endl;
    cout << "Standard Package Location: " <<
tracking.getTrackingInfo(standardPackage.generateLabel()) << endl;

    return 0;
}
```

---

6. **Explanation of Code**

1. `PackageBase`:
    - The abstract base class enforces that any derived class (like `Package`) must implement a `generateLabel()` function.
2. `Package`:
    - Represents the package with type (express or standard), weight, and label generation. The priority is set based on the package type.
3. `ComparePriority`:
    - Custom comparator for the priority queue. Ensures that express packages are processed first by comparing their priorities.
4. `Delivery`:
    - Manages the package status and uses a priority queue to process packages based on their priority. Express packages are always processed before standard ones.
5. `TrackingSystem`:
    - Tracks the package location, including international deliveries, by storing region data.

7. **Time and Space Complexities**

- **Priority Queue**:
    - Insertion and extraction operations take O(log n), where n is the number of packages in the queue.
- **Map Operations (for status and location)**:

- Lookups, inserts, and updates have O(1) time complexity.

8. **Benefits of this Approach**

- **Prioritization**: Express packages are always processed first, making the system highly efficient for high-traffic periods.
- **Extensibility**: The design can easily extend to handle other package types or priorities in the future.
- **Scalability**: The priority queue ensures that even under heavy loads, express packages get delivered on time, making the system scalable for large-scale operations.