1. **Implement a Multimap**

**Problem Statement**:  
Create a custom MultiMap class that allows multiple values for a single key, using a HashMap<K, List<V>> as the underlying data structure. Implement the following methods:

* put(K key, V value): Adds a value to the list of values associated with the specified key.
* get(K key): Returns the list of values associated with the specified key, or an empty list if no values are found.
* removeValue(K key, V value): Removes a specific value associated with a key, leaving the key if other values remain.

**Example**: If you call put("color", "red") and then put("color", "blue"), calling get("color") should return ["red", "blue"].

**Goal**:  
This exercise is about managing complex key-value mappings, where each key can be associated with multiple values.

**2. Frequency Analysis of Consecutive Elements**

**Problem Statement**:  
Given an array of integers, create a HashMap that tracks the frequency of consecutive pairs. Each key in the map should be a pair of integers (like [a, b]), and the value should be the count of occurrences of that consecutive pair in the array.

**Example**: Input: [1, 2, 1, 2, 3, 1, 2, 3, 4] Output: { [1, 2] -> 3, [2, 1] -> 1, [2, 3] -> 2, [3, 4] -> 1 }

**Goal**:  
This problem requires creating composite keys and tracking specific patterns in a sequence using HashMap.

**3. Design a Stock Price Tracker**

**Problem Statement**:  
Design a StockPriceTracker class that uses a HashMap to store stock prices with timestamps. Implement the following methods:

* addPrice(String stock, double price, long timestamp): Adds a stock price for a given stock at a specific timestamp.
* getLatestPrice(String stock): Returns the latest price for a given stock.
* getPriceAt(String stock, long timestamp): Returns the price at a specific timestamp, or -1 if not available.
* getMaxPrice(String stock): Returns the maximum price recorded for the given stock.

**Goal**:  
This exercise tests your ability to handle time-series data and manage multiple attributes for each entry.

**4. Collaborative Filtering for Recommender Systems**

**Problem Statement**:  
Implement a basic collaborative filtering system for a movie recommender using a HashMap<String, HashMap<String, Integer>>. Each outer key is a user, and the value is another HashMap where keys are movie names, and values are ratings. Implement the following methods:

* addRating(String user, String movie, int rating): Adds a movie rating for a specific user.
* getAverageRating(String movie): Returns the average rating for a specific movie.
* getRecommendations(String user): Returns a list of movie recommendations for a user based on movies that other similar users have rated.

**Example**: If user1 and user2 have similar tastes, movies rated highly by user2 but not watched by user1 should be recommended to user1.

**Goal**:  
This task covers collaborative filtering, a basic concept in recommendation systems, and requires efficient use of nested HashMaps.

**5. Implement a Leaderboard System**

**Problem Statement**:  
Design a leaderboard system using a HashMap that tracks scores for players. Implement the following methods:

* addScore(String player, int score): Adds a score for a player, updating their total score if they already exist.
* topKPlayers(int k): Returns the names of the top k players by score in descending order.
* reset(String player): Resets the score for a player.

**Example**: If addScore("Alice", 50) and addScore("Bob", 70) are called, topKPlayers(1) should return ["Bob"].

**Goal**:  
This exercise focuses on ranking, aggregation, and sorting data efficiently with HashMap.

**6. Dependency Resolver for Package Management**

**Problem Statement**:  
You are given a list of software packages and their dependencies. Create a DependencyResolver class using a HashMap<String, List<String>>, where each key is a package name and the value is a list of dependencies for that package. Implement the following methods:

* addDependency(String package, String dependency): Adds a dependency to a package.
* getInstallOrder(String package): Returns a list representing the correct installation order for the specified package, with dependencies first.

**Example**: If package A depends on B and B depends on C, getInstallOrder("A") should return [C, B, A].

**Goal**:  
This task challenges you to use HashMap for dependency resolution, a foundational concept in package management.

**7. Count Paths in a Graph Using Adjacency List**

**Problem Statement**:  
Given a directed graph represented by an adjacency list (using a HashMap<String, List<String>>), create a method to count the number of distinct paths from a start node to an end node.

**Example**: For the graph:

css

Copy code

A -> [B, C]

B -> [D]

C -> [D]

D -> []

The number of paths from A to D would be 2 (A->B->D and A->C->D).

**Goal**:  
This task uses HashMap to represent graph structures and tests your understanding of pathfinding and recursive algorithms.

**8. Character Pair Frequency in Text**

**Problem Statement**:  
Write a function that takes a long text and returns a HashMap<String, Integer> where each key is a pair of consecutive characters (like "ab", "bc") found in the text, and the value is the frequency of that pair.

**Example**: Input: "aabbcc" Output: { "aa" -> 1, "ab" -> 1, "bb" -> 1, "bc" -> 1, "cc" -> 1 }

**Goal**:  
This task requires handling large text input and tracking the frequency of all consecutive character pairs efficiently.