Hashmap Package Documentation

May 6, 2025

Introduction

This document provides detailed documentation for the `Hashmap` package, which includes two Java classes: `SimpleHashMap` and `HashMapExperiment`. The `SimpleHashMap` class implements a hash map data structure with dynamic resizing and visualization of capacity growth using JFreeChart. The `HashMapExperiment` class demonstrates the usage of `SimpleHashMap` by processing a CSV dataset of car information, storing it in the hash map, and generating visualizations for capacity growth and average car prices by brand. This documentation covers the purpose, methods, parameters, and example usage of each class, along with their dependencies and output.

1 SimpleHashMap Class

The `SimpleHashMap` class implements a hash map data structure with key-value pairs, supporting insertion, lookup, and dynamic resizing. It tracks capacity growth and provides a visualization of capacity changes using JFreeChart.

Class Overview

- **Package**: `Hashmap`
- **Purpose**: Provide a hash map implementation with dynamic resizing and capacity growth visualization.
- **Dependencies**: `org.jfree.chart.*`, `javax.swing.*`, `java.util.*`
- **Inner Class**: `Node` (stores key-value pairs)
- **Constructor**: `SimpleHashMap(int initialCapacity)`

Inner Class: Node

- **Purpose**: Represents a key-value pair in the hash map.
- **Fields**:
  - `String key`: The key of the pair.
  - `String value`: The value associated with the key.
- **Constructor**: `Node(String key, String value)`

Methods

- **`public SimpleHashMap(int initialCapacity)`**
  - **Description**: Initializes the hash map with the specified capacity.
  - **Parameters**:
    - `initialCapacity`: Initial number of buckets.

- **`private int hash(String key)`**
  - **Description**: Computes the hash index for a given key by summing character values and applying modulo capacity.
  - **Parameters**:
    - `key`: The key to hash.
  - **Returns**: Index in the range [0, capacity).
- **`public void put(String key, String value)`**
  - **Description**: Inserts or updates a key-value pair. Resizes the hash map if the load factor exceeds 0.75. Logs entry count and capacity.
  - **Parameters**:
    - `key`: The key to insert or update.
    - `value`: The value associated with the key.
- **`public boolean contains(String key)`**
  - **Description**: Checks if the hash map contains the specified key.
  - **Parameters**:
    - `key`: The key to check.
  - **Returns**: `true` if the key exists, `false` otherwise.
- **`private void resize()`**
  - **Description**: Doubles the capacity and rehashes all existing key-value pairs.
- **`public double averageLoad()`**
  - **Description**: Calculates the average number of entries per non-empty bucket.
  - **Returns**: Average load as a double.
- **`public int maxBucketSize()`**
  - **Description**: Returns the size of the largest bucket (maximum number of entries in any bucket).
  - **Returns**: Maximum bucket size as an integer.
- **`public void showCapacityGrowthChart()`**
  - **Description**: Displays a line chart showing capacity growth as entries are inserted, using JFreeChart.
  - **Output**: A JFrame window with a line chart.

Example Usage

```
SimpleHashMap map = new SimpleHashMap(16);
map.put("VIN123", "Brand: Toyota, Model: Corolla, Price: $20000");
map.put("VIN456", "Brand: Honda, Model: Civic, Price: $22000");
System.out.println("Contains VIN123: " + map.contains("VIN123")); // true
System.out.println("Average Load: " + map.averageLoad());
System.out.println("Max Bucket Size: " + map.maxBucketSize());
map.showCapacityGrowthChart(); // Displays chart
```

2 HashMapExperiment Class

The `HashMapExperiment` class demonstrates the usage of `SimpleHashMap` by processing a CSV dataset of car information, storing data in the hash map, and generating visualizations. It reads car data from a specified CSV file, measures insertion performance, and displays a bar chart of average car prices by brand.

Class Overview

- **Package**: Default package
- **Purpose**: Run an experiment using `SimpleHashMap` to store car data and visualize average car prices by brand.
- **Dependencies**: `Hashmap.SimpleHashMap`, `org.jfree.chart.*`, `javax.swing.*`, `java.io.*`, `java.util.*`
- **Main Method**: `public static void main(String[] args)`
- **Static Method**: `showAveragePriceByBrand(Map¡String, Integer¿, Map¡String, Integer¿)`

Methods

- **`public static void main(String[] args)`**
  - **Description**: Executes the experiment by reading a CSV file, populating a `SimpleHashMap`, and generating visualizations.
  - **Functionality**:
    * Initializes a `SimpleHashMap` with an initial capacity of 16.
    * Reads a CSV file (`cleaned`$_{d}ataset.csv)containingcardata, skippingtheheaderrow.Foreachvalidrow(with$
    * * `VIN` (column 9, index 8): Used as the key.
      * `Brand` (column 3, index 2): Car brand.
      * `Model` (column 4, index 3): Car model.
      * `Price` (column 2, index 1): Car price (parsed as an integer).
  - Stores each car's details in the hash map as a string: `"Brand: ¡brand¿, Model: ¡model¿, Price:` $<price>".MaintainsalistofVINs($`vinList`$)forrandomlookuptests.$
  - Tracks brand price sums (`brandPriceSum`) and car counts (`brandCarCount`) using standard Java `HashMap`s.
  - Measures insertion time using `System.currentTimeMillis()`.
  - Outputs experiment results, including:
    * Number of entries inserted.
    * Insertion time (in milliseconds).
    * Final hash map capacity.
    * Average load per non-empty bucket.
    * Maximum bucket size.
  - Performs five random `contains` checks on VINs from `vinList`.
  - Calls `showAveragePriceByBrand` to display a bar chart of average car prices by brand.
  - Implicitly relies on `SimpleHashMap.showCapacityGrowthChart()` (not called in this code but available for use).

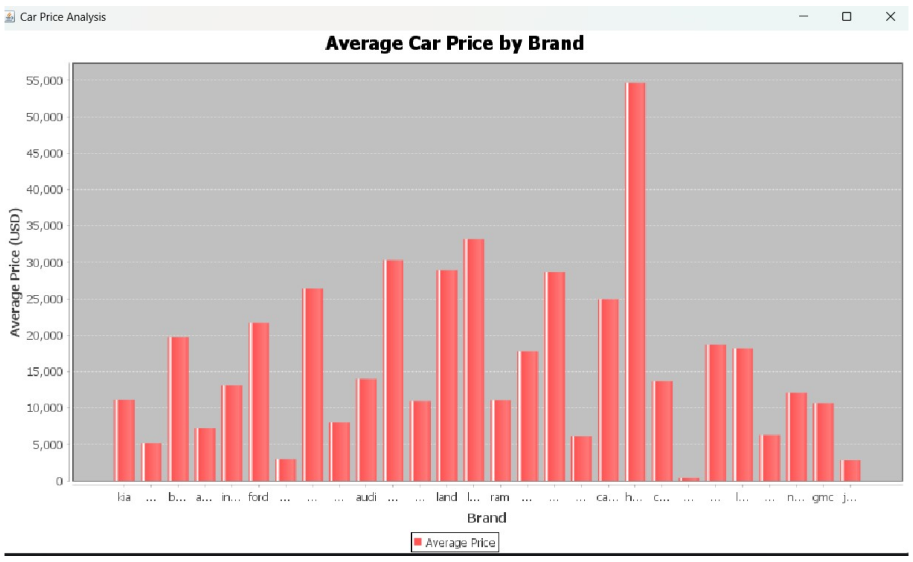

Figure 1: HasMap GUI for Average Price By Brands

- **Exceptions**: Catches `IOException` for file reading errors and prints the stack trace.

`private static void showAveragePriceByBrand(Map¡String, Integer¿ brandPriceSum, Map¡String, Integer¿ brandCarCount)`

- **Description**: Creates a bar chart showing the average car price by brand using JFreeChart.
- **Parameters**:
  - `brandPriceSum`: Map of brands to total prices.
  - `brandCarCount`: Map of brands to car counts.
- **Functionality**:
  - Calculates the average price for each brand as `total / count`.
  - Creates a `DefaultCategoryDataset` with average prices.
  - Generates a bar chart titled ”Average Car Price by Brand” with brands on the x-axis and average prices (USD) on the y-axis.
  - Displays the chart in a `JFrame` window (1000x600 pixels).

Example Usage

```
java HashMapExperiment
```

This will:

- Read the CSV file and populate the `SimpleHashMap` with car data.
- Output statistics (e.g., insertion time, capacity, average load).
- Perform random `contains` checks and print results.
- Display a bar chart of average car prices by brand in a JFrame window.

Example Output

```
Resizing from 16 to 32
Resizing from 32 to 64
...
```

```
=== Experiment Results ===
Inserted 1000 entries.
Time taken for insertion: 150 ms
Final Capacity: 64
Average Load per Bucket: 2.5
Max Bucket Size: 7
Contains VIN123: true
Contains VIN456: true
...
```

A JFrame window appears with a bar chart titled "Average Car Price by Brand," showing average prices for each brand.

3 Dependencies

- **JFreeChart**: For creating line and bar charts (`org.jfree.chart.*`, `org.jfree.data.category.*`).
- **Swing**: For displaying charts in JFrame windows (`javax.swing.*`).
- **Java Standard Library**: For data structures and file I/O (`java.util.*`, `java.io.*`).

4 Usage Notes

- Resizing occurs when the load factor exceeds 0.75, doubling the capacity, which may trigger multiple times for large datasets.
- The bar chart is displayed in a JFrame window; ensure a graphical environment is available.
- The `entryCounts` and `capacities` lists in `SimpleHashMap` grow with each insertion, which may consume memory for very large datasets.
- The `showCapacityGrowthChart` method is available in `SimpleHashMap` but not called in `HashMapExperiment`; it can be added if capacity growth visualization is needed.