



**FAKULTAS TEKNOLOGI
INFORMASI
UNIVERSITAS ANDALAS
PADANG**

**PEMROGRAMAN
BERORIENTASI OBJEK**

**Rehan Khairuno
(2211533003)**

**Poalca Valusio
(221153300)**

**Muhammad Zhafarul Maahiy
(2211533009)**

**Najwa Azka Thalita Mehdi
(2211533012)**

**Rafano Alexander
(2211533016)**

**Humayra Fahreri
(2211533019)**

**Dhiya Safira Andini
(2211533023)**

**Khoiri Putra Mujiza
(2211537001)**

DOSEN PENGAMPU :

Jefril Rahmadoni, M.Kom.

KATA PENGANTAR

Puji syukur kami panjatkan kehadirat Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan makalah ini dengan judul "Eksplorasi Mendalam tentang HashMap, LinkedHashMap, dan TreeMap dalam Pemrograman Java." Makalah ini disusun sebagai salah satu tugas akhir dalam rangka memenuhi tugas mata kuliah Struktur Data dan Algoritma.

Dalam makalah ini, penulis membahas secara mendalam tentang tiga struktur data khusus dalam pemrograman Java, yaitu HashMap, LinkedHashMap, dan TreeMap. Setiap struktur data memiliki karakteristik, kelebihan, dan kekurangan yang perlu dipahami dengan baik untuk memilih struktur data yang sesuai dengan kebutuhan aplikasi.

Penulis menyadari bahwa makalah ini tidak luput dari kekurangan, baik dari segi pemahaman maupun implementasi. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan demi perbaikan ke depannya. Akhir kata, penulis berharap makalah ini dapat memberikan kontribusi dan manfaat yang baik bagi pembaca, khususnya para pembaca yang sedang memperdalam pemahaman mengenai struktur data dalam pemrograman Java.

Terima kasih kepada semua pihak yang telah membantu dan mendukung penulis dalam menyelesaikan makalah ini. Semoga makalah ini dapat bermanfaat dan menjadi referensi yang berguna bagi pengembangan ilmu pengetahuan dan teknologi. Wassalamu'alaikum Wr. Wb.

DAFTAR ISI

KATA PENGANTAR	2
BAB I.....	4
1.1 Latar Belakang	4
1.2 Rumusan Masalah	4
1.3 Tujuan dan Manfaat	5
BAB II.....	6
2.1 HashMap	6
2.1.1 Pengertian	6
2.1.2 Fungsi/ Method yang Digunakan Pada HashMap	6
2.1.3 Kelebihan dan Kekurangan HashMap	6
2.2 Linked HashMap	9
2.2.1 Pengertian	9
2.2.2 Kelebihan dan Kekurangan	9
2.3 Perbedaan HashMap dan Linked HashMap	12
2.4 TreeMap	13
2.4.1 Pengertian	13
2.4.2 Kelebihan dan Kekurangan	13
BAB III	17
3.1 Kesimpulan.....	17
3.2 Saran.....	18
DAFTAR PUSTAKA	20

BAB I

PENDAHULUAN

1.1 Latar Belakang

Makalah ini membahas tentang struktur data HashMap, LinkedHashMap, dan TreeMap dalam pemrograman Java. Struktur data ini merupakan implementasi array asosiatif dalam Java yang memiliki kelebihan dan kekurangan masing-masing. HashMap adalah kelas yang berisi kumpulan pasangan nilai dan kunci, dengan beberapa method yang digunakan untuk manipulasi data. Kelebihan HashMap antara lain performa tinggi, fleksibilitas, pemindaian iteratif yang efisien, implementasi di Java yang matang, dan kapasitas dinamis. Namun, kekurangannya adalah tidak terurut, tidak thread-safe, kinerja buruk jika terlalu penuh, penggunaan memori lebih tinggi, dan tidak cocok untuk struktur data terbatas.

LinkedHashMap adalah implementasi Map dengan dukungan hash dan list terkait untuk meningkatkan fungsionalitas dari HashMap. Kelebihannya adalah urutan penyisipan dijaga, performa cukup baik, dan iterasi efisien, namun memiliki kekurangan dalam penggunaan memori lebih besar, performa menurun untuk operasi kunci tertentu, tidak thread-safe secara alami, dan tidak cocok untuk beberapa kasus penggunaan.

TreeMap adalah kelas dalam Java yang mengimplementasikan antarmuka SortedMap dan menyimpan pasangan kunci-nilai dalam urutan terurut berdasarkan kunci. Kelebihannya adalah urutan alami, implementasi thread-safe, operasi cepat untuk kunci terbatas, dan fleksibilitas comparator, namun memiliki kekurangan dalam kinerja yang tergantung pada struktur pohon, kompleksitas pemrograman, dan pemakaian memori lebih besar.

1.2 Rumusan Masalah

1. bagaimana memahami karakteristik dan kebutuhan aplikasi untuk memilih struktur data yang tepat antara HashMap, LinkedHashMap, dan TreeMap dalam pemrograman Java?

2. Bagaimana performa, kelebihan, dan kekurangan dari masing-masing struktur data ini?
3. Bagaimana pengaruh penggunaan memori, thread safety, dan kompleksitas pemrograman dalam pemilihan struktur data yang tepat?
4. Bagaimana cara memilih struktur data yang sesuai dengan kebutuhan aplikasi?

1.3 Tujuan dan Manfaat

a. Tujuan

Pada Bahasa pemrograman java ada beberapa struktur data bawaan yang telah disediakan oleh java dan diantaranya yang pertama ada hashmap yang digunakan untuk menyimpan nilai kunci, hashmap digunakan untuk membuat algoritma pencarian yang memiliki performa yang tinggi dan efisien. Struktur data berikutnya ada linkedhashmap yang mana linked hashmap memiliki kegunaan yang sama dengan hashmap tetapi juga mempertahankan urutan elemen, linkedhashmap menggabungkan keunggulan dari dua struktur data yaitu linkedlist dan hashmap. Struktur data yang ketiga ada tree map yang bertujuan untuk mengatur data yang memerlukan urutan.

b. Manfaat

Pada struktur data java hashmap dapat berguna untuk keefisienan pencarian data, linkedHashMap dapat memilih urutan data dan performa yang baik dan mempertahankan urutan elemen, serta treeMap yang menggunakan urutan otomatis serta penelusuran yang efisien.

BAB II

PEMBAHASAN

2.1 HashMap

2.1.1 Pengertian

Ini memungkinkan pengambilan, penghapusan, dan pembaruan pasangan nilainya melalui kuncinya. Dalam pemrograman Java, HashMap adalah sebuah class yang berisi kumpulan pasangan nilai (value) dan kunci (key). Nilai bisa dalam bentuk string, integer, boolean, float, double, dan objek. HashMap memungkinkan penambahan, penghapusan, dan pencarian nilai berdasarkan kuncinya..

2.1.2 Fungsi/ Method yang Digunakan Pada HashMap

- **.put()**
- **.get()**
- **.remove()**
- **.containsKey()**
- **.containsValue()**
- **.keySet()**
- **.values()**
- **.entrySet()**
- **.size()**
- **.isEmpty()**

2.1.3 Kelebihan dan Kekurangan HashMap

Kelebihan:

- **Performa Tinggi:** Pencarian, penyisipan, dan penghapusan data dapat dilakukan dalam waktu konstan ($O(1)$).

- **Fleksibilitas:** Data dapat diakses dengan menggunakan kunci, dan kita dapat menyimpan dan mengakses data dengan cara yang sangat efisien.
- **Pemindaian Iteratif yang Efisien:** Untuk melakukan iterasi melalui seluruh elemen dalam HashMap, Anda dapat menggunakan entrySet untuk mendapatkan set dari seluruh pasangan kunci-nilai. Kemudian, Anda dapat menggunakan loop for-each untuk mengakses setiap elemen dengan cara yang efisien.
- **Implementasi di Java yang Matang:** Implementasi HashMap di Java sangat matang dan dioptimalkan dengan baik. Seiring dengan pembaruan Java, perbaikan dan optimalisasi terus dilakukan untuk memastikan kinerjanya yang baik.
- **Kapasitas Dinamis:** HashMap dapat menyesuaikan kapasitasnya secara dinamis saat elemen-elemen ditambahkan atau dihapus, yang mengurangi kompleksitas administrasi dari segi pengelolaan ruang dan efisiensi penyimpanan.

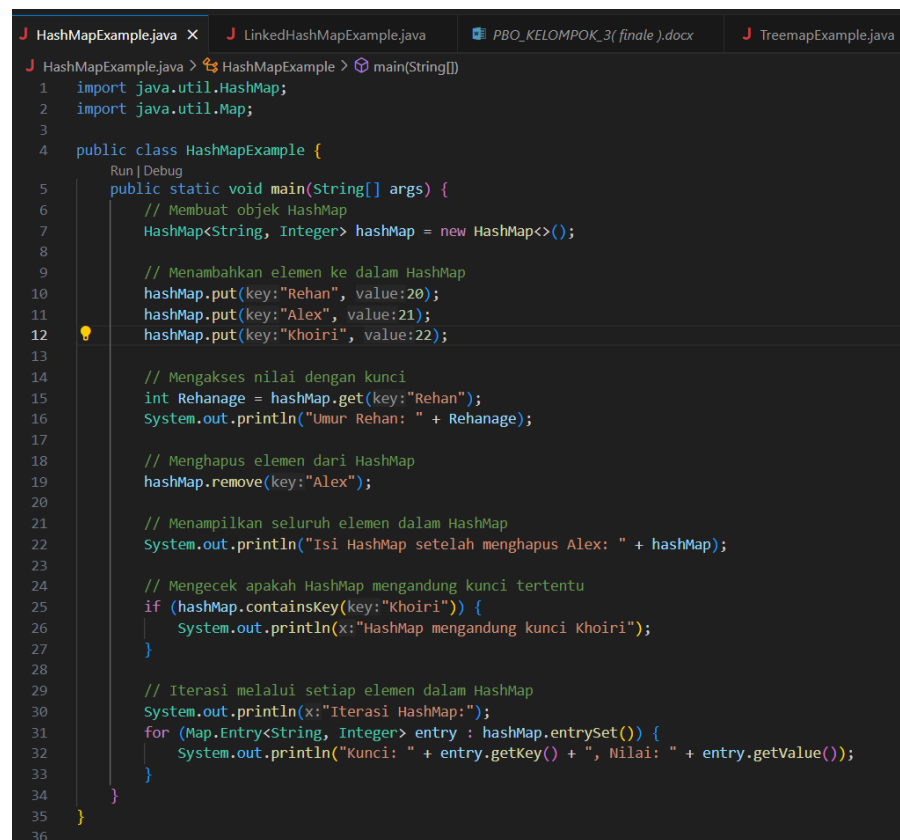
Adapun, kekurangannya antara lain:

- **Tidak Terurut (Unordered):** Elemen-elemen dalam HashMap tidak diurutkan berdasarkan urutan penambahan. Jika Anda membutuhkan iterasi berdasarkan urutan tertentu, Anda mungkin harus menggunakan struktur data lain, seperti LinkedHashMap.
- **Synchronicity:** HashMap tidak bersifat thread-safe. Jika digunakan dalam lingkungan multi-threading tanpa tindakan pencegahan tambahan, itu dapat menyebabkan masalah keamanan. Jika diperlukan keamanan thread, Anda bisa menggunakan ConcurrentHashMap atau melakukan sinkronisasi manual.
- **Kinerja Buruk Jika Terlalu Penuh (Load Factor):** Kinerja HashMap dapat menurun jika terlalu penuh. Faktor muatan (load factor) yang buruk dapat menyebabkan banyak tabrakan hash dan mengurangi kinerja operasi dasar seperti put dan get. Faktor muatan yang baik dapat diatur dengan konstruktor HashMap untuk mengoptimalkan kinerja.

- **Penggunaan Memori Lebih Tinggi:** HashMap dapat menggunakan lebih banyak memori dibandingkan dengan struktur data lain jika tidak dikelola dengan baik, terutama karena alokasi awal yang besar dan faktor muatan yang tinggi.
- **Tidak Cocok untuk Struktur Data Terbatas (Sparse Data):** Jika hanya sedikit kunci-nilai yang digunakan dalam HashMap dan kunci memiliki rentang yang sangat besar, maka HashMap mungkin tidak menjadi struktur data yang paling efisien.

Contoh Kodingan HashMap:

Kodingan:



```

J HashMapExample.java X J LinkedHashMapExample.java PBO_KELOMPOK_3( finale ).docx J TreemapExample.java
J HashMapExample.java > HashMapExample > main(String[])
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class HashMapExample {
5     public static void main(String[] args) {
6         // Membuat objek HashMap
7         HashMap<String, Integer> hashMap = new HashMap<>();
8
9         // Menambahkan elemen ke dalam HashMap
10        hashMap.put(key:"Rehan", value:20);
11        hashMap.put(key:"Alex", value:21);
12        hashMap.put(key:"Khoiri", value:22);
13
14        // Mengakses nilai dengan kunci
15        int Rehanage = hashMap.get(key:"Rehan");
16        System.out.println("Umur Rehan: " + Rehanage);
17
18        // Menghapus elemen dari HashMap
19        hashMap.remove(key:"Alex");
20
21        // Menampilkan seluruh elemen dalam HashMap
22        System.out.println("Isi HashMap setelah menghapus Alex: " + hashMap);
23
24        // Mengecek apakah HashMap mengandung kunci tertentu
25        if (hashMap.containsKey(key:"Khoiri")) {
26            System.out.println(x:"HashMap mengandung kunci Khoiri");
27        }
28
29        // Iterasi melalui setiap elemen dalam HashMap
30        System.out.println(x:"Iterasi HashMap:");
31        for (Map.Entry<String, Integer> entry : hashMap.entrySet()) {
32            System.out.println("Kunci: " + entry.getKey() + ", Nilai: " + entry.getValue());
33        }
34    }
35 }
36
  
```


Output:

```
29 // Iterasi melalui setiap elemen dalam HashMap

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

InExceptionMessages' '-cp' 'C:\Users\ASUS\AppData\Roaming\Code\User\workspaceS
hMapExample'
Umur Rehan: 20
Isi HashMap setelah menghapus Alex: {Rehan=20, Khoiri=22}
HashMap mengandung kunci Khoiri
Iterasi HashMap:
Kunci: Rehan, Nilai: 20
Kunci: Khoiri, Nilai: 22
PS D:\1.Tugas Semester 3\PBO\Tugas Kelompok>
```

Penjelasan terkait kodingan tersebut:

- **HashMap<String, Integer>:** Mendeklarasikan sebuah objek HashMap yang memiliki kunci bertipe String dan nilai bertipe Integer.
- **put:** Menambahkan elemen ke dalam HashMap dengan kunci dan nilai tertentu.
- **get:** Mengakses nilai berdasarkan kunci.
- **remove:** Menghapus elemen dari HashMap berdasarkan kunci.
- **containsKey:** Mengecek apakah HashMap mengandung kunci tertentu.
- **entrySet:** Mengembalikan set dari seluruh elemen dalam HashMap, yang kemudian dapat diiterasi menggunakan loop for-each.

2.2 Linked HashMap

2.2.1 Pengertian

kelas dalam Java yang menyediakan implementasi Map dengan dukungan hash dan list terkait (linked list) untuk meningkatkan fungsionalitas dari HashMap. LinkedHashMap memiliki sebagian besar fitur dari HashMap, tetapi menambahkan fitur penjagaan urutan elemen yang diinsertkan. Elemen dalam LinkedHashMap dapat diakses dalam urutan yang diberikan saat insertion.

2.2.2 Kelebihan dan Kekurangan

Kelebihan:

- **Urutan Penyisipan Dijaga:** Elemen-elemen dalam LinkedHashMap dipesan berdasarkan urutan penyisipan. Ini bermanfaat jika Anda perlu mempertahankan urutan penyisipan elemen.
- **Performa Cukup Baik:** Meskipun sedikit lebih lambat daripada HashMap dalam beberapa operasi, performa LinkedHashMap masih cukup baik.
- **Iterasi Efisien:** Iterasi melalui elemen-elemen menggunakan iterator atau foreach loop dilakukan dengan efisien karena elemen-elemen diatur dalam urutan penyisipan.

Kekurangan:

- **Penggunaan Memori Lebih Besar:** LinkedHashMap menggunakan lebih banyak memori daripada HashMap. Ini disebabkan oleh adanya penambahan overhead untuk menyimpan informasi tambahan tentang urutan penyisipan.
- **Performa Menurun untuk Operasi Kunci Tertentu:** Meskipun iterasi secara umum dilakukan dengan efisien, beberapa operasi tertentu, seperti penghapusan atau penyisipan di tengah-tengah, dapat menyebabkan performa LinkedHashMap menurun. Ini karena harus memperbarui rangkaian terhubung.
- **Tidak Thread-Safe Secara Alami:** Seperti HashMap, LinkedHashMap juga bukan thread-safe. Jika digunakan dalam lingkungan multi-threading tanpa tindakan pencegahan tambahan, itu dapat menyebabkan masalah keamanan. Untuk penggunaan multi-threading, disarankan untuk menggunakan ConcurrentHashMap atau melakukan sinkronisasi manual.
- **Tidak Cocok untuk Beberapa Kasus Penggunaan:** Jika urutan penyisipan tidak penting dalam aplikasi Anda, menggunakan LinkedHashMap mungkin membuang-buang sumber daya karena memiliki kompleksitas memori dan waktu yang lebih tinggi dibandingkan dengan HashMap.

Contoh Kodingan Linked HashMap

Kodingan:

```
J HashMapExample.java  J LinkedHashMapExample.java  PBO_KELOMPOK_3(finale ).docx  J TreemapExample.java
J LinkedHashMapExample.java > LinkedHashMapExample > main(String[])
1  import java.util.LinkedHashMap;
2  import java.util.Map;
3
4  public class LinkedHashMapExample {
5      public static void main(String[] args) {
6          // Membuat objek LinkedHashMap
7          LinkedHashMap<String, Integer> linkedHashMap = new LinkedHashMap<>();
8
9          // Menambahkan elemen ke dalam LinkedHashMap
10         linkedHashMap.put(key:"Yaya", value:20);
11         linkedHashMap.put(key:"Poa", value:30);
12         linkedHashMap.put(key:"Zhafa", value:22);
13
14         // Mengakses nilai dengan kunci
15         int Yayaage = linkedHashMap.get(key:"Yaya");
16         System.out.println("Umur yaya: " + Yayaage);
17
18         // Menghapus elemen dari LinkedHashMap
19         linkedHashMap.remove(key:"Poa");
20
21         // Menampilkan seluruh elemen dalam LinkedHashMap
22         System.out.println("Isi LinkedHashMap setelah menghapus Poa: " + linkedHashMap);
23
24         // Mengecek apakah LinkedHashMap mengandung kunci tertentu
25         if (linkedHashMap.containsKey(key:"Zhafa")) {
26             System.out.println(x:"LinkedHashMap mengandung kunci Zhafa");
27         }
28
29         // Iterasi melalui setiap elemen dalam LinkedHashMap
30         System.out.println(x:"Iterasi LinkedHashMap:");
31         for (Map.Entry<String, Integer> entry : linkedHashMap.entrySet()) {
32             System.out.println("Kunci: " + entry.getKey() + ", Nilai: " + entry.getValue());
33         }
34     }
35 }
36
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
:~ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\ASUS\AppData\Roaming\Code\User\workspaceStorage\0c70eac0233d6349445ded6af860326d
\tugas kelompok_a3e8a47c\bin' 'LinkedHashMapExample'
Umur yaya: 20
Isi LinkedHashMap setelah menghapus Poa: {Yaya=20, Zhafa=22}
LinkedHashMap mengandung kunci Zhafa
Iterasi LinkedHashMap:
Kunci: Yaya, Nilai: 20
Kunci: Zhafa, Nilai: 22
PS D:\1.Tugas Semester 3\PBO\tugas kelompok>
```

Penjelasan terkait kodingan tersebut:

- **LinkedHashMap<String, Integer>:** Mendeklarasikan objek LinkedHashMap dengan kunci bertipe String dan nilai bertipe Integer.
- **put:** Menambahkan elemen ke dalam LinkedHashMap dengan kunci dan nilai tertentu.
- **get:** Mengakses nilai berdasarkan kunci.
- **remove:** Menghapus elemen dari LinkedHashMap berdasarkan kunci.

- **containsKey:** Mengecek apakah LinkedHashMap mengandung kunci tertentu.
- **entrySet:** Mengembalikan set dari seluruh elemen dalam LinkedHashMap, yang kemudian dapat diiterasi menggunakan loop for-each.

2.3 Perbedaan HashMap dan Linked HashMap

➤ Jenis Struktur Data:

- Kode pertama menggunakan HashMap, yang merupakan struktur data hash table. Elemen dalam HashMap tidak memiliki urutan tertentu.
- Kode kedua menggunakan LinkedHashMap, yang juga merupakan struktur data hash table tetapi menyimpan urutan penyisipan elemen.

➤ Urutan Penyisipan:

- Pada HashMap, urutan penyisipan elemen tidak dijamin. Saat Anda melakukan iterasi, elemen dapat muncul dalam urutan yang berbeda dari urutan penyisipan.
- Pada LinkedHashMap, urutan penyisipan elemen dijaga. Saat Anda melakukan iterasi, elemen akan muncul dalam urutan penyisipan.

➤ Pengaruh pada Operasi Penghapusan:

- Penghapusan elemen dalam HashMap tidak memengaruhi urutan elemen yang tersisa.
- Penghapusan elemen dalam LinkedHashMap dapat memengaruhi urutan elemen, karena elemen-elemen diatur dalam suatu rangkaian terhubung.

➤ **Pengaruh pada Penggunaan Memori:**

- LinkedHashMap memiliki overhead memori lebih besar daripada HashMap karena harus menyimpan informasi tambahan tentang urutan penyisipan.

➤ **Tujuan Penggunaan:**

- Pilih HashMap jika urutan elemen tidak penting dan Anda hanya ingin efisiensi dalam operasi pencarian dan penyisipan.
- Pilih LinkedHashMap jika urutan penyisipan elemen harus dijaga atau jika Anda perlu melakukan operasi iterasi yang efisien.

2.4 TreeMap

2.4.1 Pengertian

TreeMap adalah kelas dalam Java yang mengimplementasikan antarmuka SortedMap dan menyimpan pasangan kunci-nilai dalam urutan yang terurut berdasarkan kunci. TreeMap menyimpan elemen dalam struktur pohon merah-hitam, yang memungkinkan pencarian, penambahan, dan penghapusan elemen dalam waktu logaritmik.

2.4.2 Kelebihan dan Kekurangan

Kelebihan:

- **Urutan Alami:** Elemen-elemen dalam TreeMap disimpan dalam urutan alami (ascending order) berdasarkan kunci. Ini memudahkan untuk melakukan operasi seperti pencarian, iterasi, dan rentang kunci tertentu.
- **Implementasi Thread-Safe:** Secara alami, TreeMap bukan struktur data thread-safe. Namun, jika Anda membutuhkan versi thread-safe, Anda dapat menggunakan `Collections.synchronizedSortedMap` untuk membuat TreeMap yang bersifat thread-safe.

- **Operasi Cepat untuk Kunci Terbatas:** Operasi pencarian, penambahan, dan penghapusan memiliki kompleksitas waktu $O(\log N)$, di mana N adalah jumlah elemen dalam TreeMap.
- **Fleksibilitas Comparator:** Selain menggunakan kunci yang mengimplementasikan Comparable, Anda dapat membuat TreeMap dengan memberikan comparator eksternal saat pembuatan objek TreeMap. Ini memungkinkan Anda mengurutkan elemen berdasarkan kriteria khusus.

Kekurangan:

- **Kinerja Tergantung pada Struktur Pohon:** Performa operasi put, get, dan remove dapat bervariasi tergantung pada struktur pohon merah-hitam dan tingkat seimbangnya. Dalam beberapa kasus, TreeMap mungkin tidak secepat struktur data hash table seperti HashMap.
- **Kompleksitas Pemrograman:** Menggunakan TreeMap memerlukan implementasi dari antarmuka Comparable atau pemberian comparator eksternal. Ini memerlukan sedikit lebih banyak pekerjaan dari sisi pengkodean dibandingkan dengan struktur data yang tidak memerlukan perbandingan.
- **Pemakaian Memori Lebih Besar:** TreeMap menggunakan lebih banyak memori dibandingkan dengan struktur data yang lebih sederhana seperti HashMap atau LinkedHashMap karena menyimpan struktur pohon merah-hitam.

Contoh Kodingan TreeMap

Kodingan:

```
J TreemapExample.java > TreemapExample > main(String[])
1  import java.util.Map;
2  import java.util.TreeMap;
3
4  public class TreemapExample {
5      public static void main(String[] args) {
6          Map<String, Integer> treeMap = new TreeMap<>();
7
8          // Menambah elemen pada tree map
9          treeMap.put(key:"A", value:1);
10         treeMap.put(key:"C", value:3);
11         treeMap.put(key:"B", value:2);
12
13         // Mendapatkan nilai dari tree map
14         int valueA = treeMap.get(key:"A");
15         System.out.println("Value of A: " + valueA);
16
17         // Menghilangkan elements pada tree map
18         treeMap.remove(key:"B");
19
20         // Iterasi elements pada tree map
21         for (String key : treeMap.keySet()) {
22             System.out.println("Key: " + key + ", Value: " + treeMap.get(key));
23         }
24     }
25 }
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS D:\1.Tugas Semester 3\PBO\Tugas Kelompok> & 'C:\Program Files\Java\jdk-18.0.2.1\bin\java.exe' -cp 'D:\1.Tugas Semester 3\PBO\Tugas Kelompok\TreemapExample.jar' -Djava.class.path=. TreemapExample
Value of A: 1
Key: A, Value: 1
Key: C, Value: 3
PS D:\1.Tugas Semester 3\PBO\Tugas Kelompok>
```

Penjelasan terkait kodingan tersebut:

- **TreeMap<String, Integer>:** Mendeklarasikan objek TreeMap dengan kunci bertipe String dan nilai bertipe Integer.
- **put:** Menambahkan elemen ke dalam TreeMap dengan kunci dan nilai tertentu.

- **get:** Mengakses nilai berdasarkan kunci.
- **remove:** Menghapus elemen dari TreeMap berdasarkan kunci.
- **containsKey:** Mengecek apakah TreeMap mengandung kunci tertentu.

BAB III

KESIMPULAN DAN SARAN

3.1 Kesimpulan

1. Hash Map

struktur data yang mengimplementasikan array asosiatif, juga disebut kamus, yang merupakan tipe data abstrak yang memetakan kunci ke nilai. HashMap menggunakan fungsi hash untuk menghitung indeks ke dalam array dari mana nilai yang diinginkan dapat ditemukan. Namun, selain itu HashMap sendiri juga memiliki kelebihan serta kekurangannya tersendiri, yang mana kelebihan HashMap meliputi performa tinggi, fleksibilitas, pemindaian iteratif yang efisien, implementasi Java yang matang, serta kapasitas dinamis. Namun sayangnya hashmap ini tak terurut, synchronicity, kinerja buruk, penggunaan memori lebih tinggi, serta tidak cocok untuk struktur data terbatas.

2. Linked Hashmap

Linked HashMap mengalihi sebagian besar fitur dari HashMap, tetapi menambahkan fitur penjagaan urutan elemen yang diinsertkan. Elemen dalam LinkedHashMap dapat diakses dalam urutan yang diberikan saat insertion. Disamping itu, Linked Hashmap juga memiliki kelebihan dan kekurangannya tersendiri, yang mana kelebihan Linked Hashmap ini meliputi urutan penyisipan dijaga, performa cukup baik, dan iterasi efisien, namun disamping kelebihan Linked Hashmap yang sangat banyak, Linked Hashmap juga memiliki kekurangan berupa penggunaan memori yang lebih besar, performa menurun untuk operasi kunci tertentu, dan tidak thread safe secara alami serta tidak cocok untuk beberapa kasus.

3. TreeMap

Tree Map adalah kelas dalam Java yang mengimplementasikan antarmuka SortedMap dan menyimpan pasangan kunci-nilai dalam urutan yang terurut

berdasarkan kunci. pada Tree Map ini sendiripun memiliki kelebihan berupa urutan alami, implementasi thread safe, operasi cepat untuk kunci terbatas, serta fleksibilitas comparator, dan kekurangannya berupa kinerja yang bergantung pada struktur pohon, kompleksitas pemrograman, dan pemakaian memori lebih besar.

3.2 Saran

Penting untuk menyadari bahwa makalah ini memiliki keterbatasan tertentu, dan rekomendasi di masa depan harus mempertimbangkan keterbatasan ini, termasuk keterbatasan pemahaman masyarakat umum tentang konsep struktur data. Oleh karena itu, penulis merekomendasikan penelitian di masa depan untuk mencakup upaya mengedukasi dan mengkomunikasikan informasi tentang kelebihan dan kekurangan struktur data HashMap, LinkedHashMap, dan TreeMap dalam pemrograman Java kepada khalayak yang lebih luas, termasuk pendatang baru di dunia pemrograman.

Selain itu, pembaca harus mempertimbangkan bahwa aspek praktis penggunaan ketiga struktur data ini mungkin berbeda tergantung pada konteks pengembangan perangkat lunak. Rekomendasi di masa depan harus mencakup penelitian lebih lanjut mengenai situasi dan jenis proyek di mana setiap struktur data akan lebih efektif dan efisien. Meskipun penelitian ini memberikan pemahaman yang lebih mendalam tentang karakteristik masing-masing struktur data, penelitian berikutnya mungkin akan menggali lebih dalam penerapan di dunia nyata, termasuk skenario penggunaan umum di industri.

Selain itu, pembaca harus menyadari bahwa pengembangan dan pemeliharaan struktur data di lingkungan pemrograman Java juga dapat berkaitan erat dengan kinerja aplikasi. Oleh karena itu, penelitian lebih lanjut dapat mencakup evaluasi kinerja yang lebih rinci dari ketiga struktur data ini dalam situasi penggunaan tertentu. Hasil evaluasi tersebut tentu akan berfungsi sebagai dasar bagi pengembang perangkat lunak untuk membuat keputusan yang lebih tepat ketika memilih struktur data yang paling sesuai dengan kebutuhan proyek.

Terakhir, penulis mendorong untuk menjajaki kemungkinan kolaborasi dengan industri atau komunitas pengembang untuk memastikan hasil studi ini memiliki

dampak yang lebih luas. Kolaborasi ini memungkinkan transfer pengetahuan yang lebih efektif dan membuka peluang untuk menerapkan temuan penelitian pada situasi dunia nyata. Akhir kata, makalah ini diharapkan dapat memberikan kontribusi yang signifikan terhadap pengembangan dan peningkatan teknik pemrograman Java yang berkelanjutan.

DAFTAR PUSTAKA

Muhardian, A. (2018). Belajar Java: Mengenal dan Memahami Class HashMap di Java.

<https://www.geeksforgeeks.org/linkedhashmap-class-in-java/>

<https://www.geeksforgeeks.org/treemap-in-java/>

<https://muhammadfikri163.wordpress.com/2014/06/28/map/>

https://www.w3schools.com/java/java_hashmap.asp

<https://stackoverflow.com/questions/2592043/what-is-a-hash-map-in-programming-and-where-can-it-be-used>