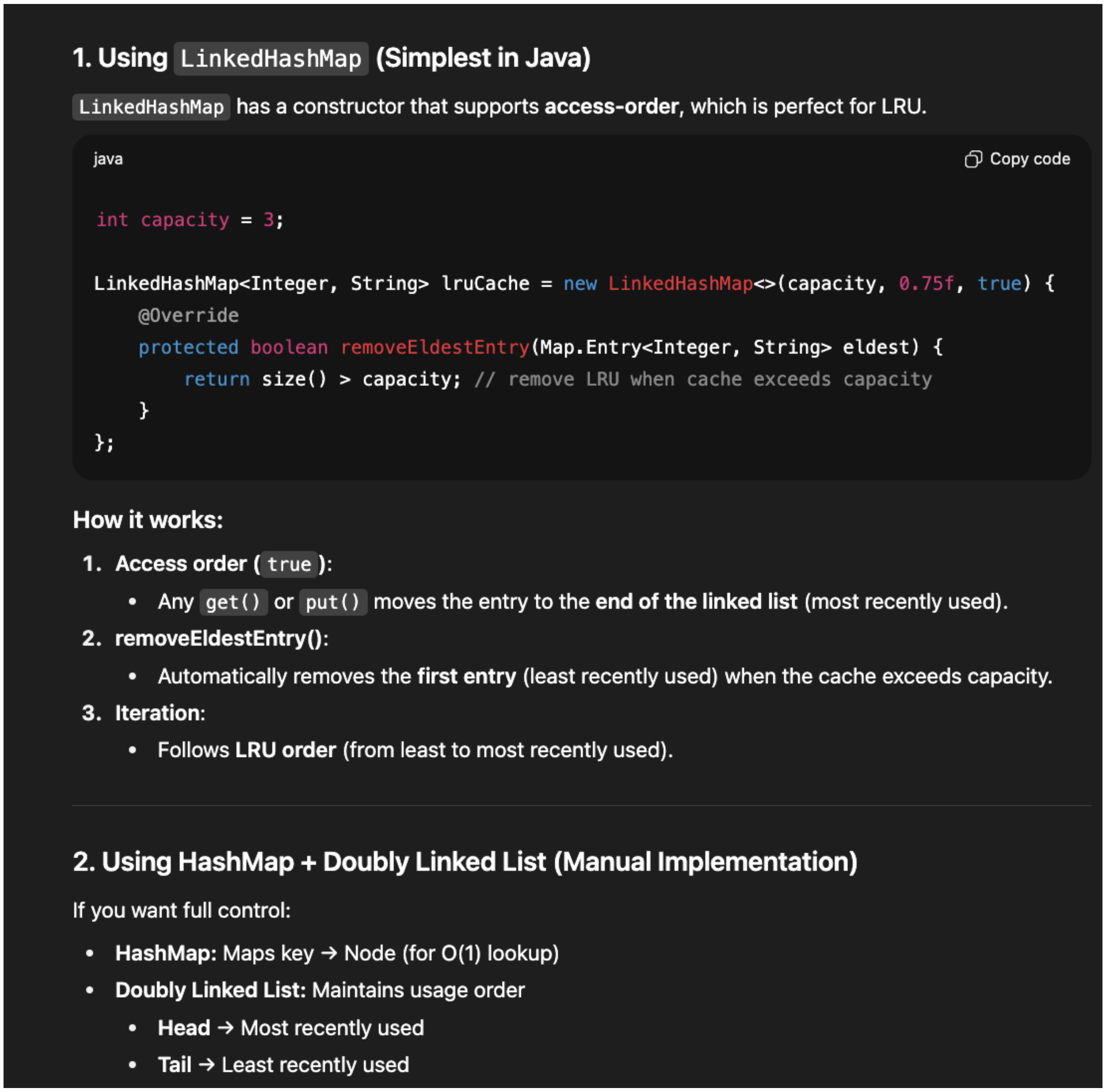
1. LRU:



1. Using `LinkedHashMap` (Simplest in Java)

`LinkedHashMap` has a constructor that supports **access-order**, which is perfect for LRU.

```java
int capacity = 3;

LinkedHashMap<Integer, String> lruCache = new LinkedHashMap<>(capacity, 0.75f, true) {
    @Override
    protected boolean removeEldestEntry(Map.Entry<Integer, String> eldest) {
        return size() > capacity; // remove LRU when cache exceeds capacity
    }
};
```

How it works:

1. **Access order (`true`):**
   - Any `get()` or `put()` moves the entry to the **end of the linked list** (most recently used).
2. **removeEldestEntry():**
   - Automatically removes the **first entry** (least recently used) when the cache exceeds capacity.
3. **Iteration:**
   - Follows **LRU order** (from least to most recently used).

2. Using HashMap + Doubly Linked List (Manual Implementation)

If you want full control:

- **HashMap:** Maps key → Node (for O(1) lookup)
- **Doubly Linked List:** Maintains usage order
  - **Head** → Most recently used
  - **Tail** → Least recently used

2. LinkedHashMap: Double-linked List + HashMap

3. PriorityQueue: Heap