ArrayList

Array

list

dynamic

10 default size

50%

indexed

null allowed

it allows duplicates

no thread

faster acess when no multi threading

LinkedList

List dequeue

double linked list

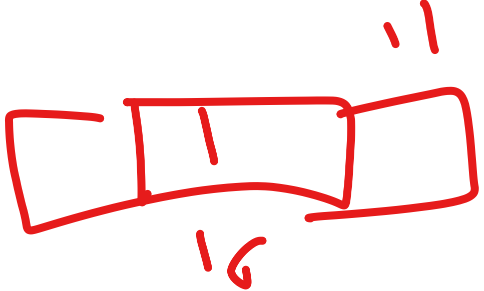
index

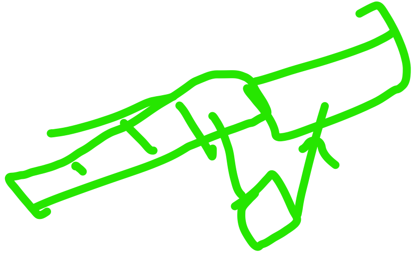
yes null values

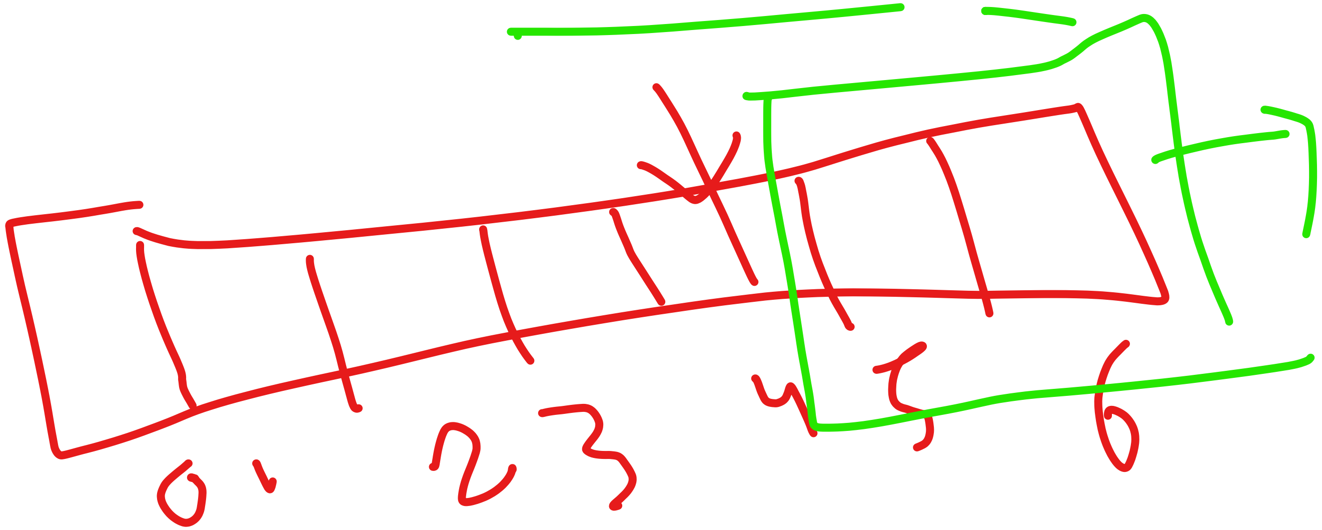
yes duplicateds allowed

no thread safe

faster insertion and deletion







DataStrutures Vs Collections

From Stratch

Code Complex nature

Predefined

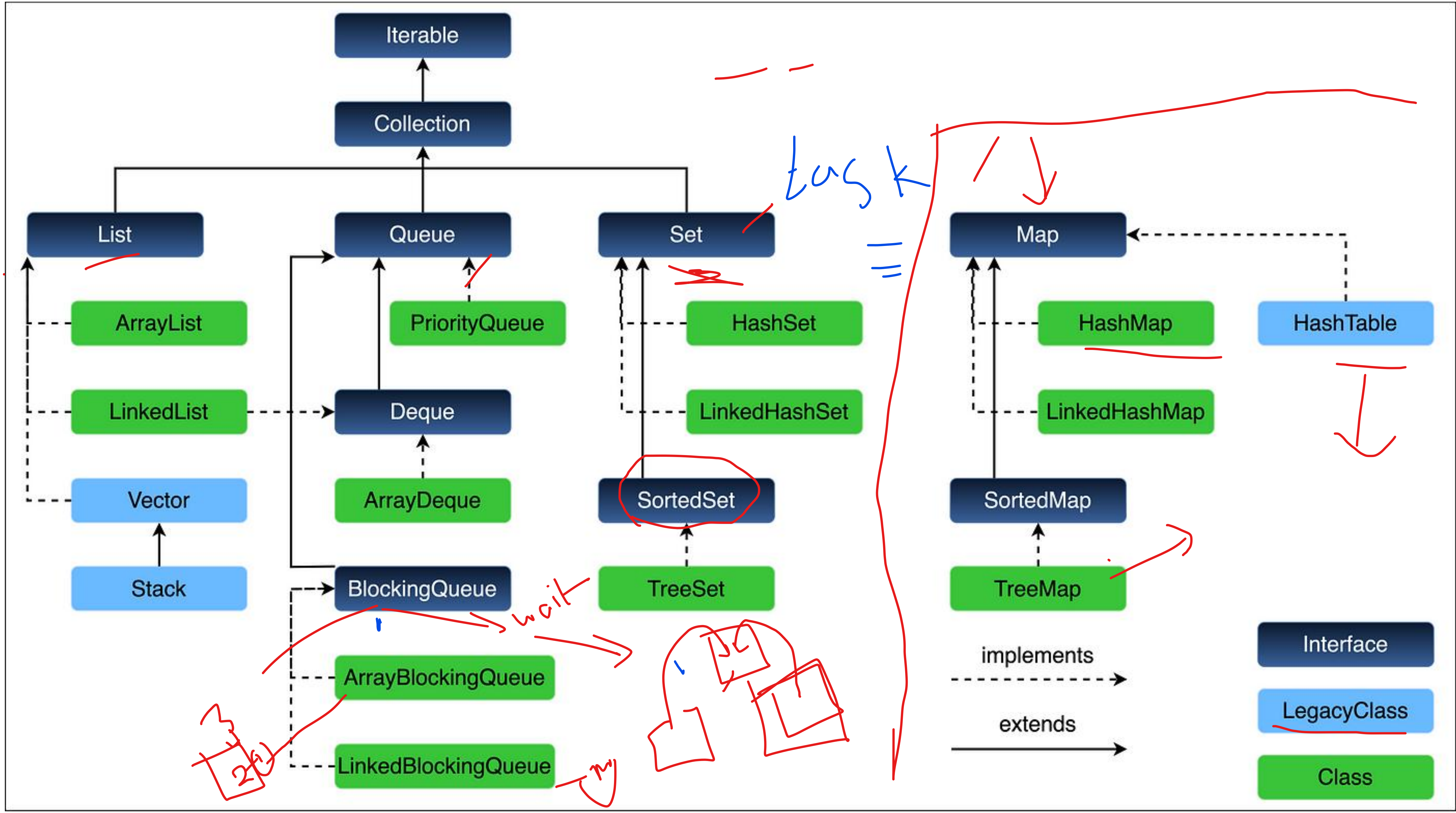
More Built class

Daynimic

flexable

◆ Legacy Classes (Old before Java Collections)

- `Vector`, `Stack`, `Hashtable` – these are older and synchronized (thread-safe).
- We now prefer `ArrayList`, `Deque`, `HashMap` unless thread-safety is needed.

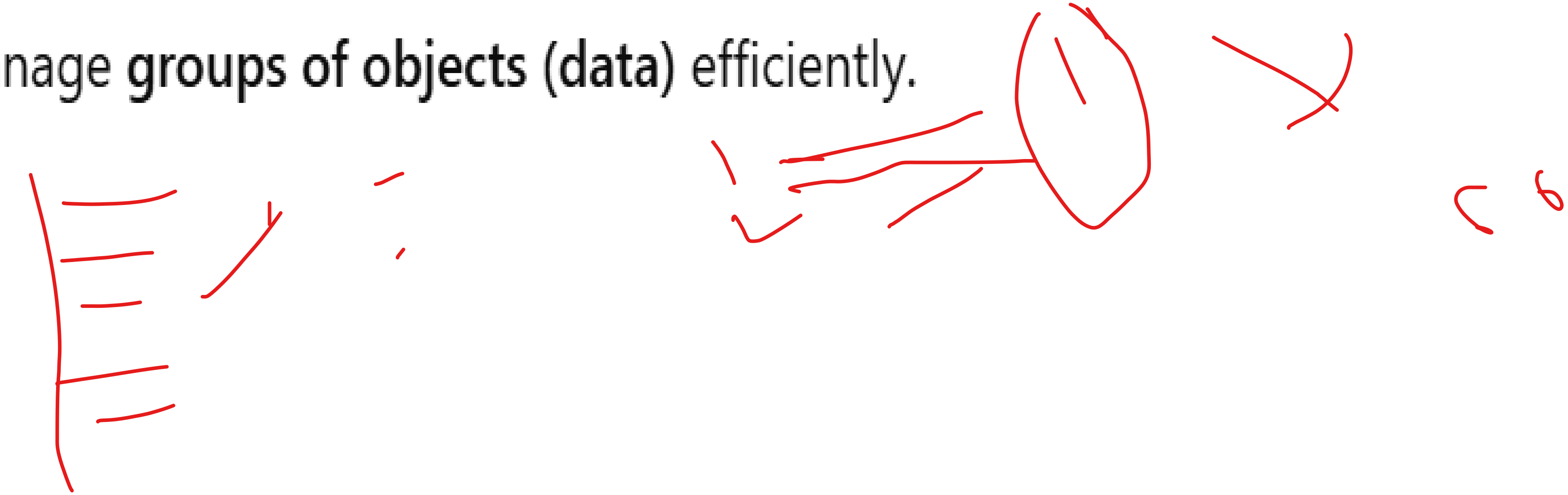


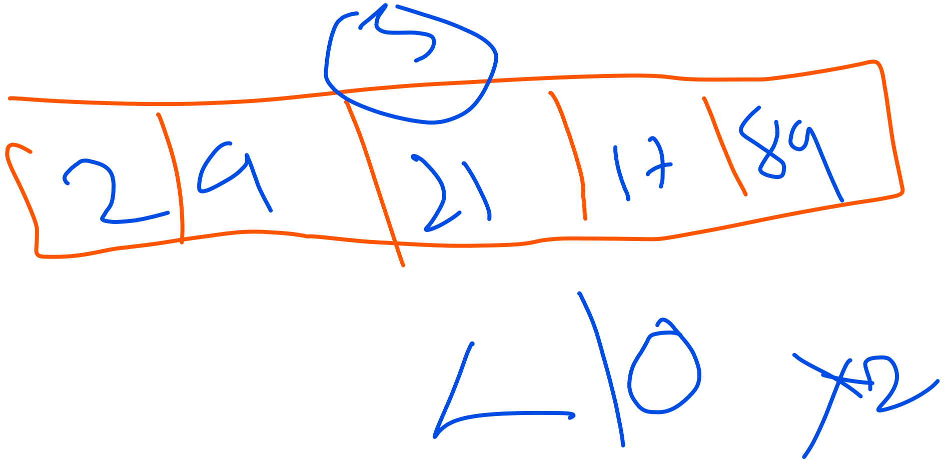
Iterable
Collection
List
Queue
Set
Map
ArrayList
PriorityQueue
HashSet
HashMap
HashTable
LinkedList
Deque
LinkedHashSet
LinkedHashMap
Vector
ArrayDeque
SortedSet
SortedMap
Stack
BlockingQueue
TreeSet
TreeMap
ArrayBlockingQueue
LinkedBlockingQueue
implements
extends
Interface
LegacyClass
Class
task
wait
3
2

- What is Java CollectionsFramework?

➤ Definition:

Java Collection Framework is a **set of classes and interfaces** that helps you store, retrieve, and manage **groups of objects (data)** efficiently.



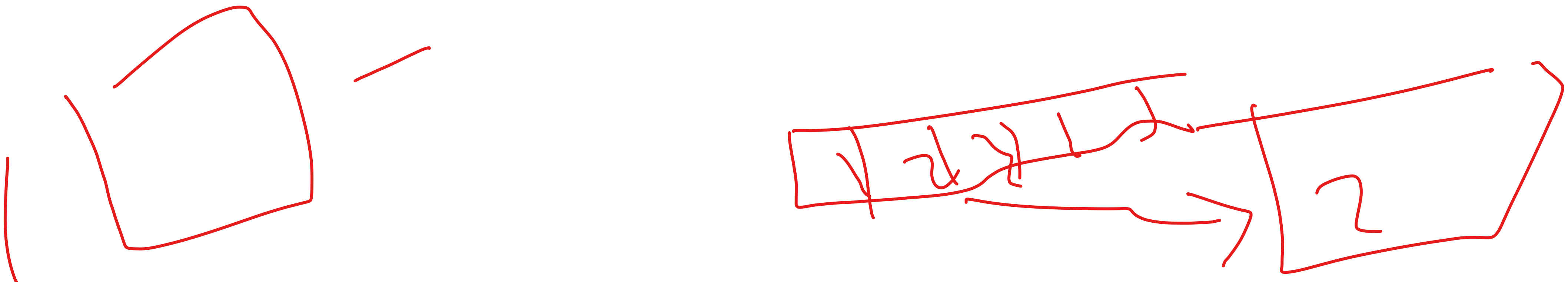


- **Why do we need it?**

Queue

FIFO

push
pop
peek
empty
poll



Java Collections vs Data Structures

hash table

no order

no null values

hash

hashmap

no order

one key allowed

linkedhashmap

order

1 null key

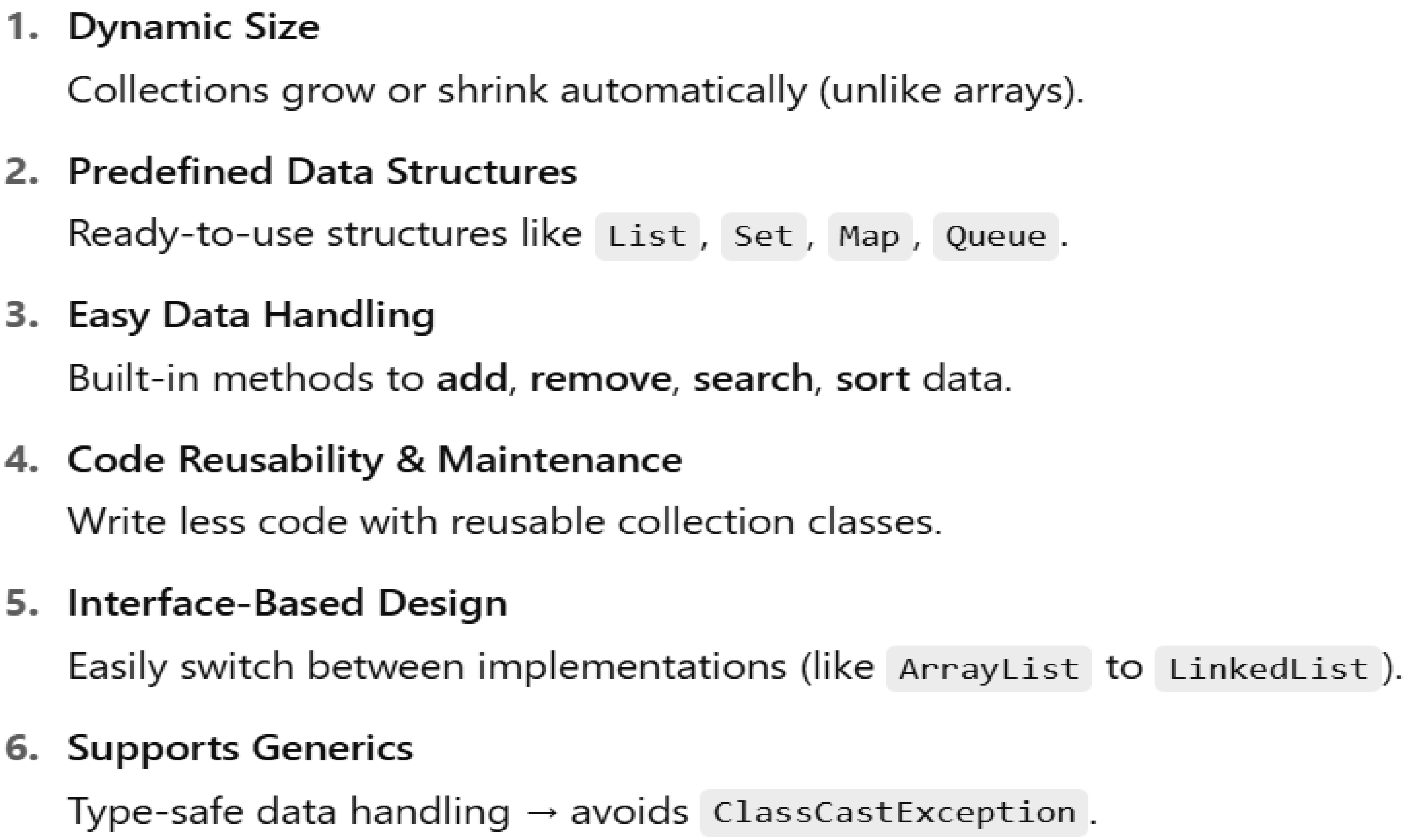
treemap

no order

no null

◆ Java Collections vs Data Structures

Feature	Data Structures (Manual)	Java Collection Framework
Write from scratch?	Yes	No – prebuilt classes
Dynamic size?	No (Arrays fixed)	Yes
Easy to use?	Complex	Very easy (`.add()` , `.remove()`)
Sorting/searching	Manual	Built-in (`Collections.sort`)
Performance optimized	You write logic	Already optimized



1. **Dynamic Size**

   Collections grow or shrink automatically (unlike arrays).

2. **Predefined Data Structures**

   Ready-to-use structures like `List`, `Set`, `Map`, `Queue`.

3. **Easy Data Handling**

   Built-in methods to **add**, **remove**, **search**, **sort** data.

4. **Code Reusability & Maintenance**

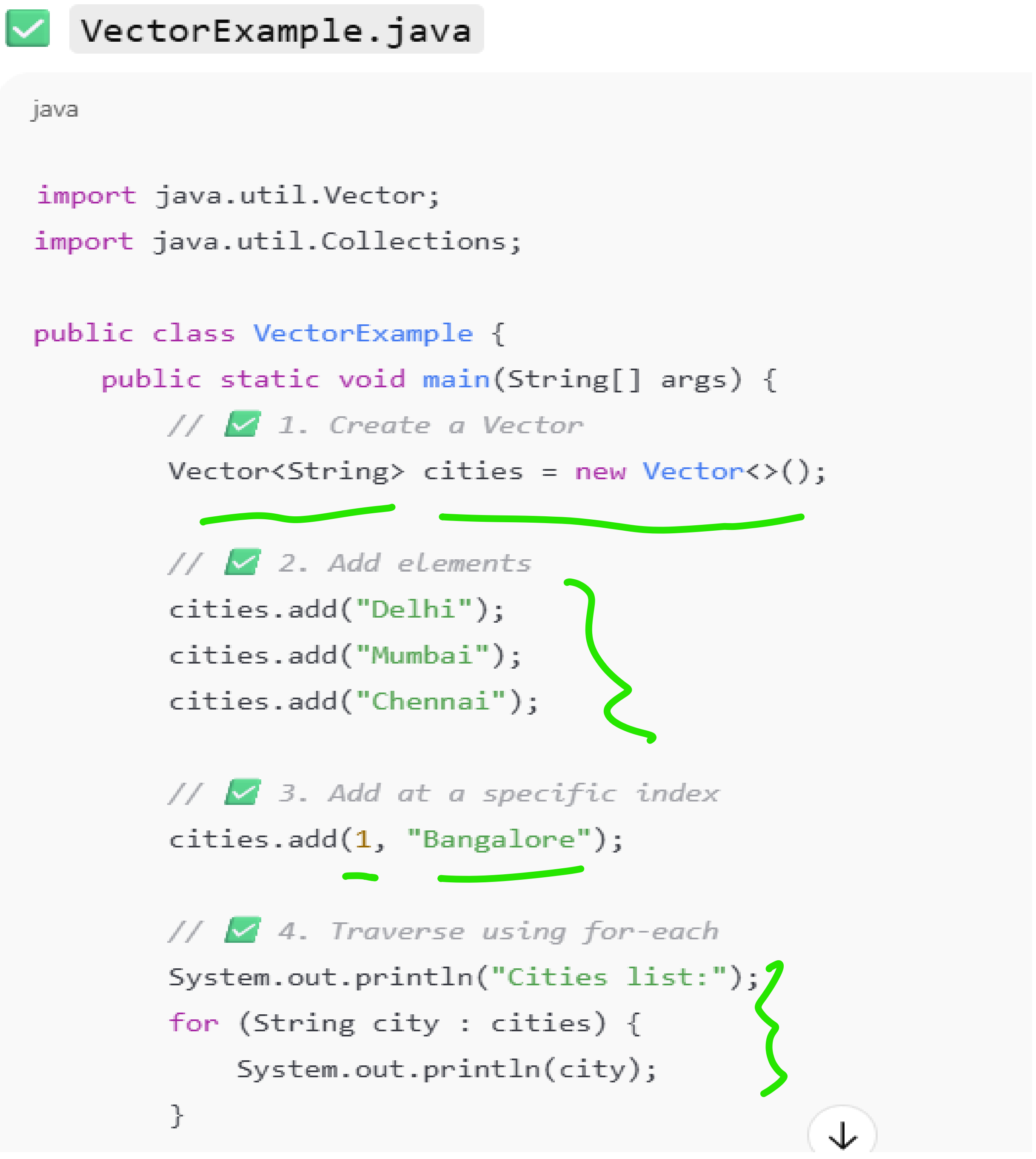
   Write less code with reusable collection classes.

5. **Interface-Based Design**

   Easily switch between implementations (like `ArrayList` to `LinkedList`).

6. **Supports Generics**

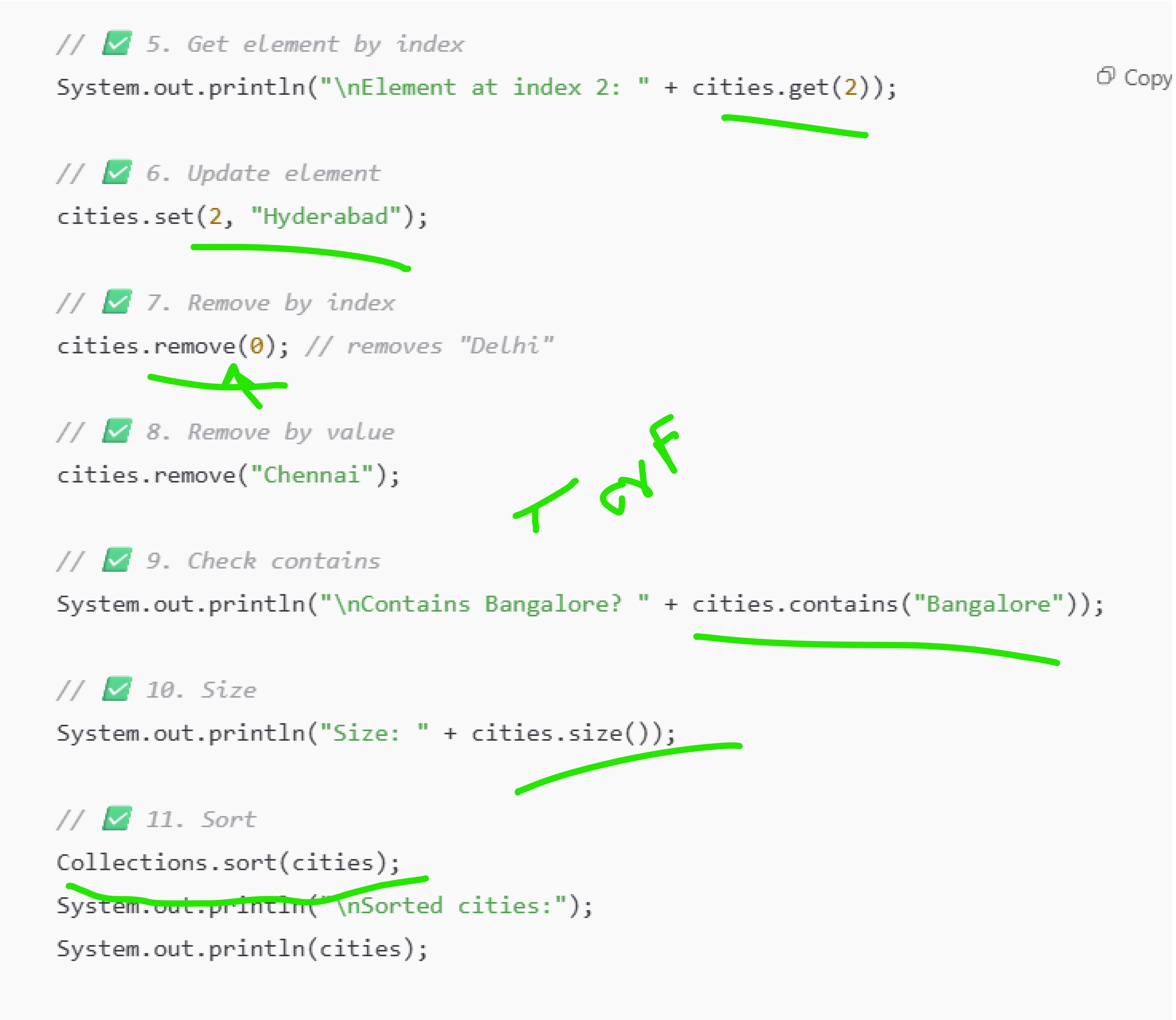
   Type-safe data handling → avoids `ClassCastException`.



✅ VectorExample.java

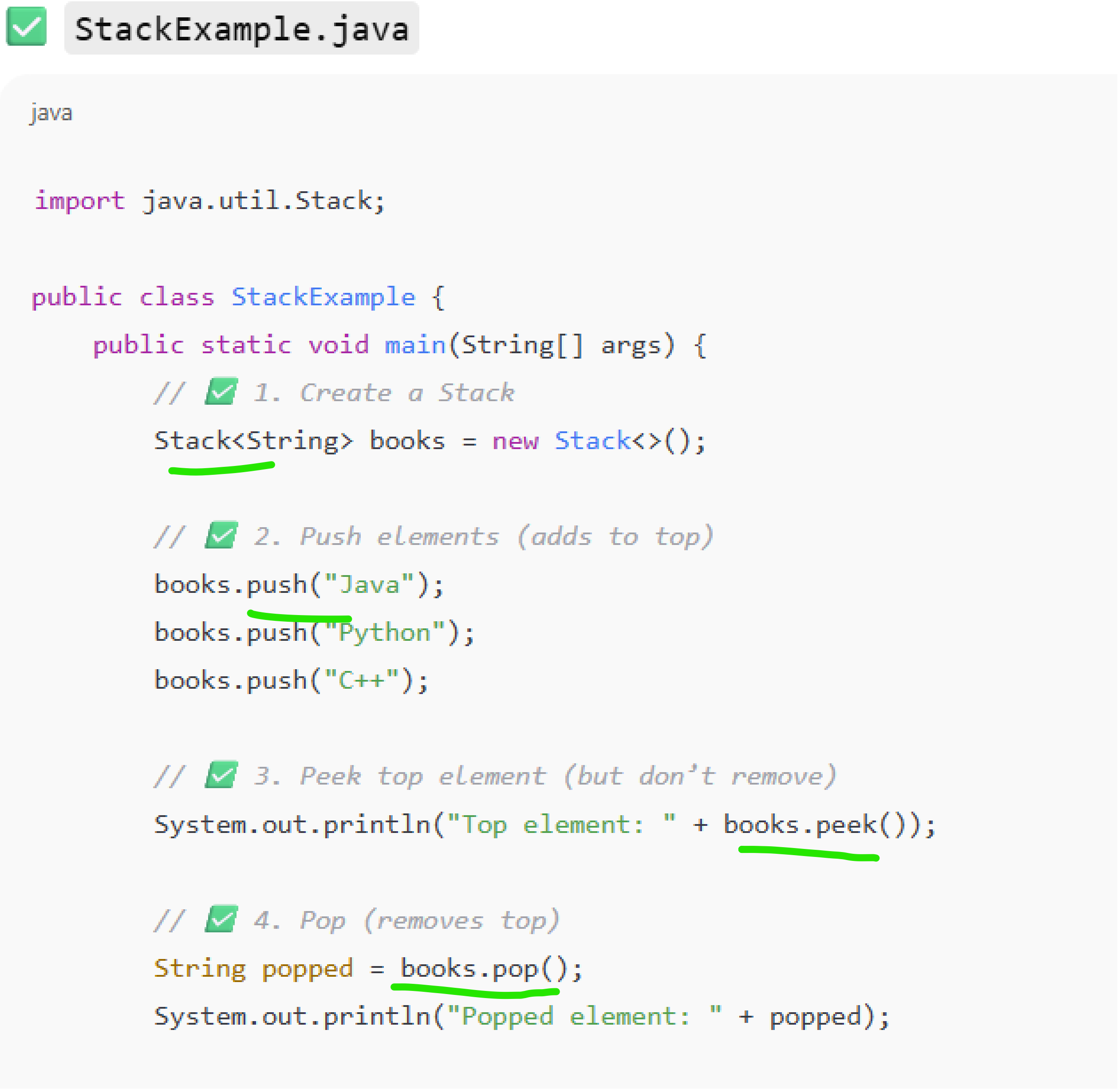
java

```java
import java.util.Vector;
import java.util.Collections;

public class VectorExample {
    public static void main(String[] args) {
        // ✅ 1. Create a Vector
        Vector<String> cities = new Vector<>();

        // ✅ 2. Add elements
        cities.add("Delhi");
        cities.add("Mumbai");
        cities.add("Chennai");

        // ✅ 3. Add at a specific index
        cities.add(1, "Bangalore");

        // ✅ 4. Traverse using for-each
        System.out.println("Cities list:");
        for (String city : cities) {
            System.out.println(city);
        }
```



```
// ✅ 5. Get element by index
System.out.println("\nElement at index 2: " + cities.get(2));

// ✅ 6. Update element
cities.set(2, "Hyderabad");

// ✅ 7. Remove by index
cities.remove(0); // removes "Delhi"

// ✅ 8. Remove by value
cities.remove("Chennai");

// ✅ 9. Check contains
System.out.println("\nContains Bangalore? " + cities.contains("Bangalore"));

// ✅ 10. Size
System.out.println("Size: " + cities.size());

// ✅ 11. Sort
Collections.sort(cities);
System.out.println("\nSorted cities:");
System.out.println(cities);
```

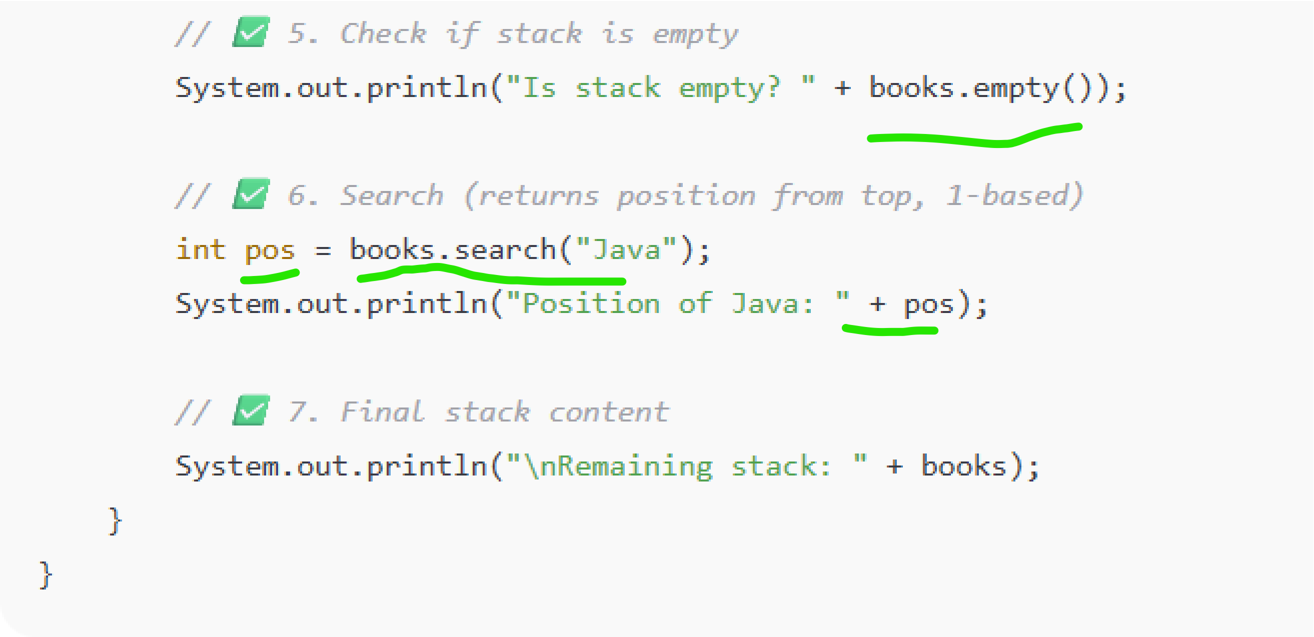
T GYF



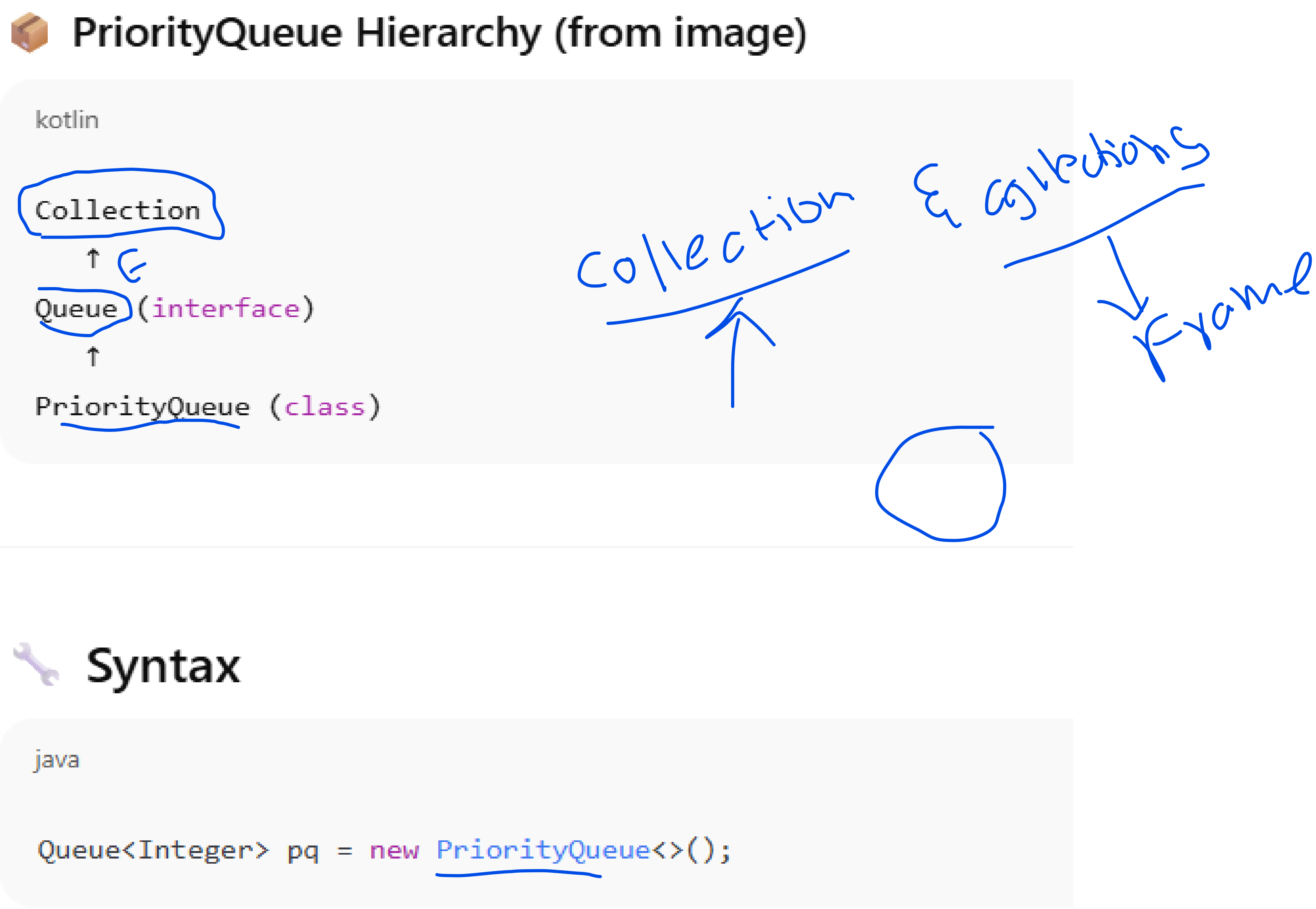
✅ `StackExample.java`

java

```java
import java.util.Stack;

public class StackExample {
    public static void main(String[] args) {
        // ✅ 1. Create a Stack
        Stack<String> books = new Stack<>();

        // ✅ 2. Push elements (adds to top)
        books.push("Java");
        books.push("Python");
        books.push("C++");

        // ✅ 3. Peek top element (but don't remove)
        System.out.println("Top element: " + books.peek());

        // ✅ 4. Pop (removes top)
        String popped = books.pop();
        System.out.println("Popped element: " + popped);
```



```java
        // ✅ 5. Check if stack is empty
        System.out.println("Is stack empty? " + books.empty());

        // ✅ 6. Search (returns position from top, 1-based)
        int pos = books.search("Java");
        System.out.println("Position of Java: " + pos);

        // ✅ 7. Final stack content
        System.out.println("\nRemaining stack: " + books);
    }
}
```



📦 PriorityQueue Hierarchy (from image)

kotlin

```kotlin
Collection
    ↑
Queue (interface)
    ↑
PriorityQueue (class)
```

E

Collection & collections

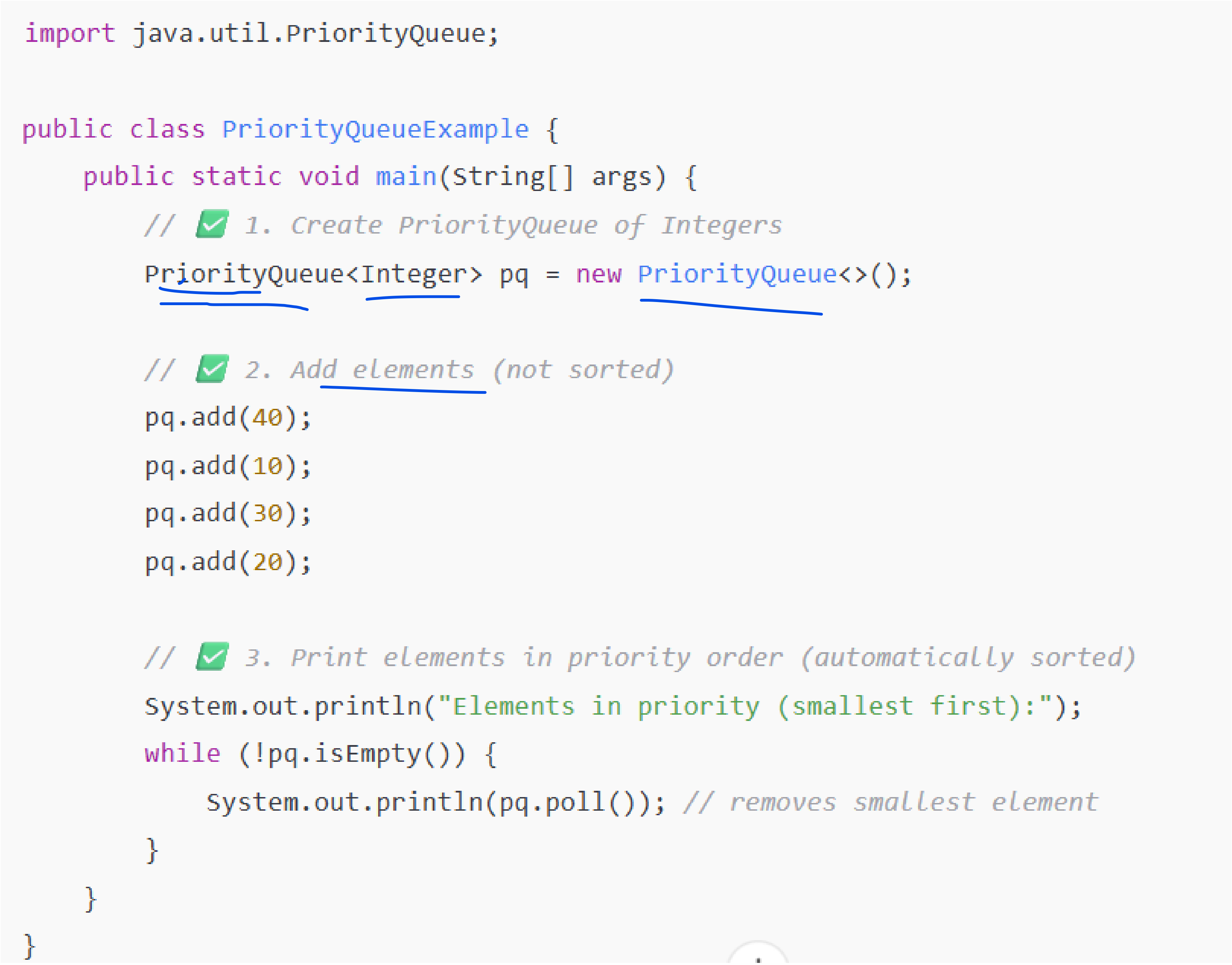
↓

Frame

🔧 Syntax

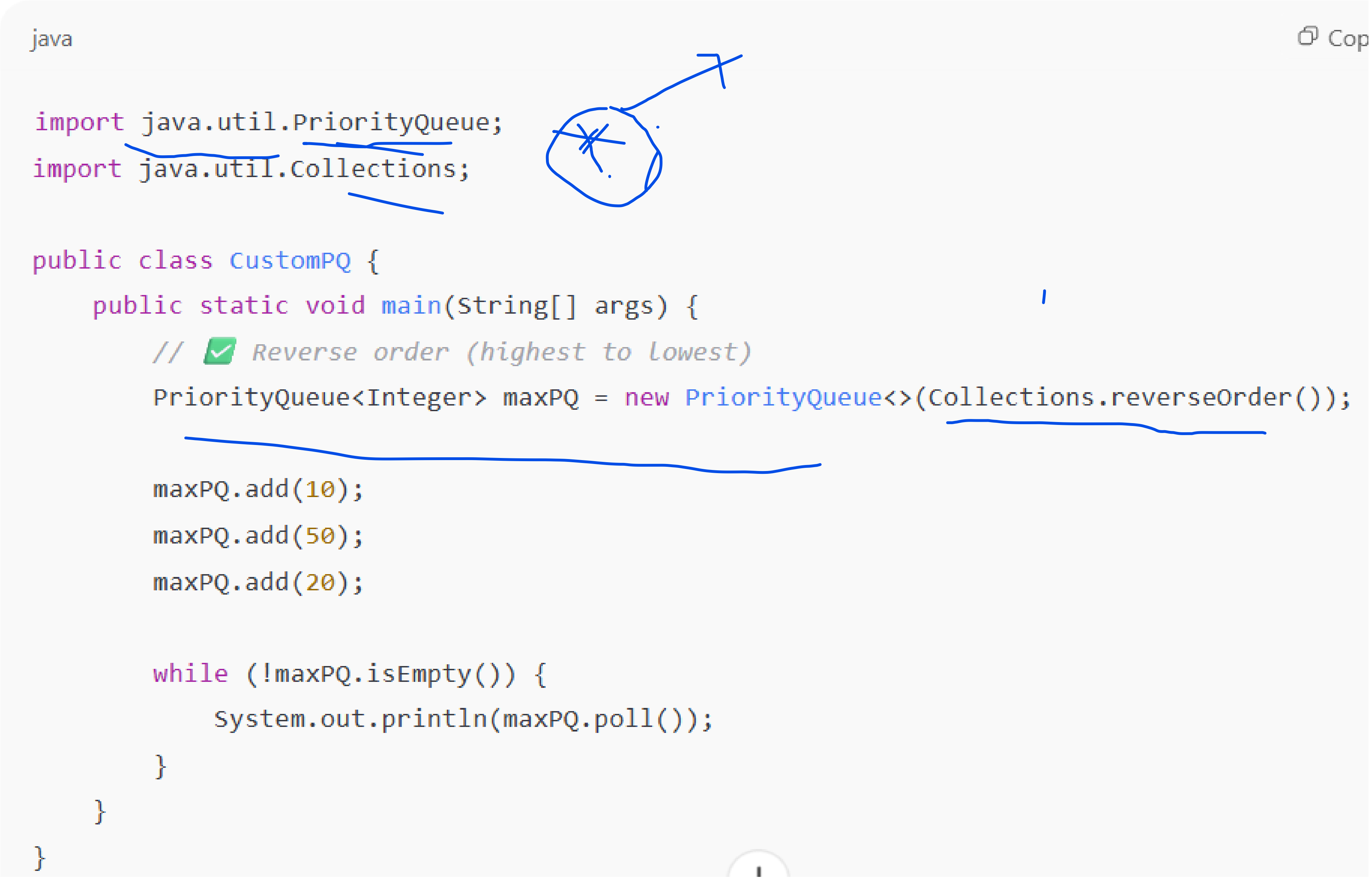
java

```java
Queue<Integer> pq = new PriorityQueue<>();
```



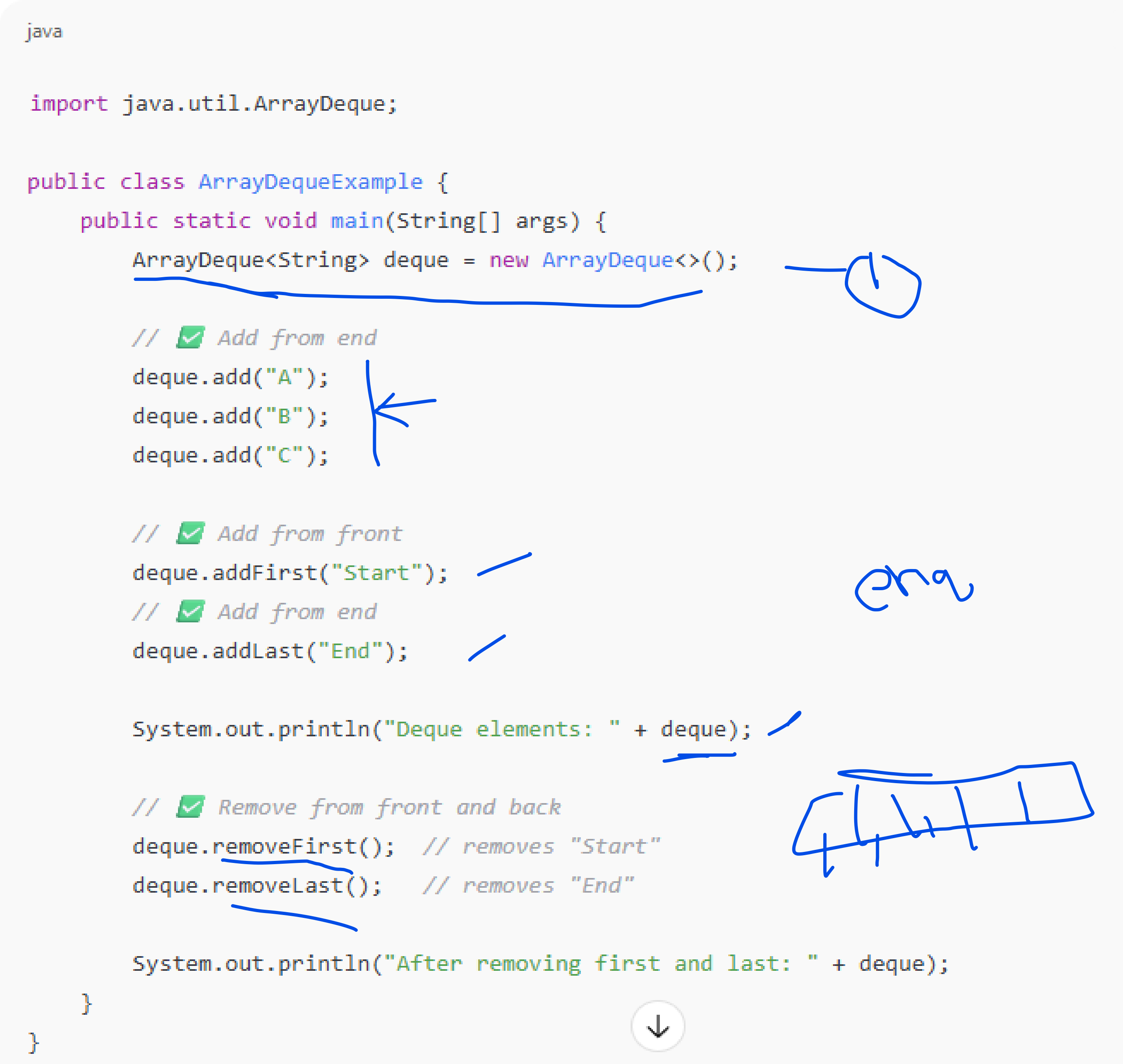
```java
import java.util.PriorityQueue;

public class PriorityQueueExample {
    public static void main(String[] args) {
        // ✅ 1. Create PriorityQueue of Integers
        PriorityQueue<Integer> pq = new PriorityQueue<>();

        // ✅ 2. Add elements (not sorted)
        pq.add(40);
        pq.add(10);
        pq.add(30);
        pq.add(20);

        // ✅ 3. Print elements in priority order (automatically sorted)
        System.out.println("Elements in priority (smallest first):");
        while (!pq.isEmpty()) {
            System.out.println(pq.poll()); // removes smallest element
        }
    }
}
```

🔁 Custom Priority: Descending Order



```java
import java.util.PriorityQueue;
import java.util.Collections;

public class CustomPQ {
    public static void main(String[] args) {
        // ✅ Reverse order (highest to lowest)
        PriorityQueue<Integer> maxPQ = new PriorityQueue<>(Collections.reverseOrder());

        maxPQ.add(10);
        maxPQ.add(50);
        maxPQ.add(20);

        while (!maxPQ.isEmpty()) {
            System.out.println(maxPQ.poll());
        }
    }
}
```

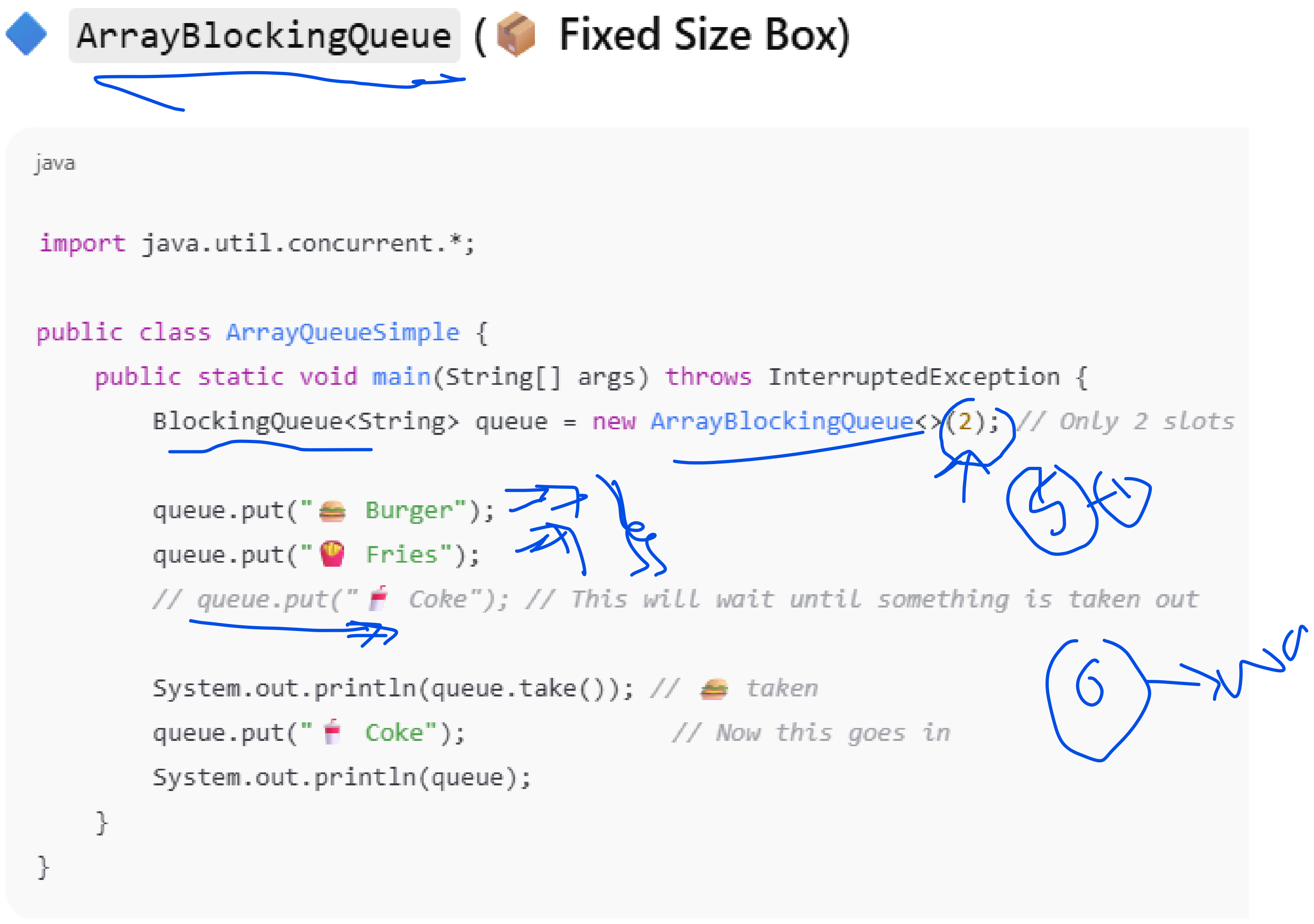
✅ Basic Usage (Like Stack and Queue)



java

```java
import java.util.ArrayDeque;

public class ArrayDequeExample {
    public static void main(String[] args) {
        ArrayDeque<String> deque = new ArrayDeque<>();

        // ✅ Add from end
        deque.add("A");
        deque.add("B");
        deque.add("C");

        // ✅ Add from front
        deque.addFirst("Start");
        // ✅ Add from end
        deque.addLast("End");

        System.out.println("Deque elements: " + deque);

        // ✅ Remove from front and back
        deque.removeFirst();  // removes "Start"
        deque.removeLast();   // removes "End"

        System.out.println("After removing first and last: " + deque);
    }
}
```

end



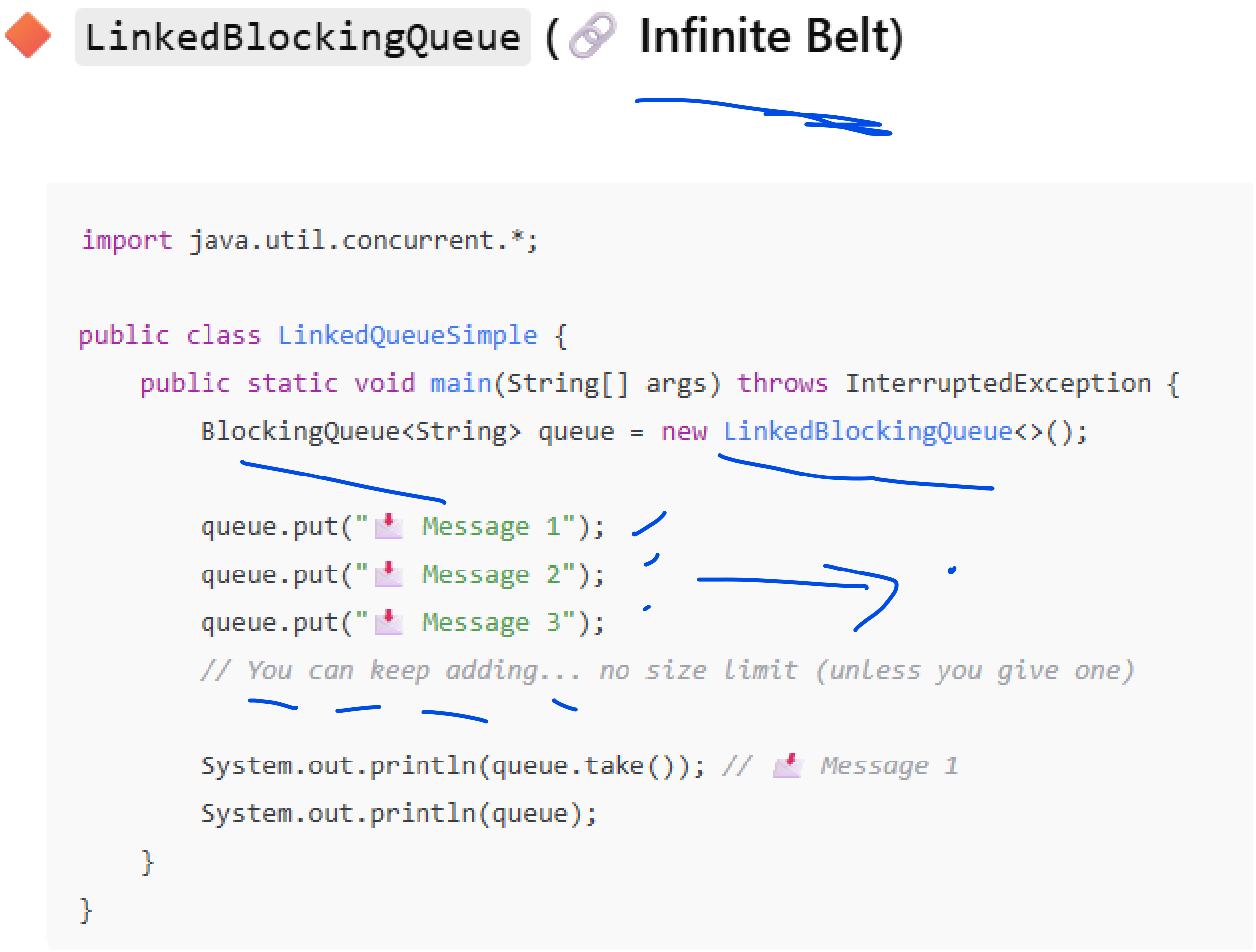
🔷 `ArrayBlockingQueue` (📦 Fixed Size Box)

java

```java
import java.util.concurrent.*;

public class ArrayQueueSimple {
    public static void main(String[] args) throws InterruptedException {
        BlockingQueue<String> queue = new ArrayBlockingQueue<>(2); // Only 2 slots

        queue.put("🍔 Burger");
        queue.put("🍟 Fries");
        // queue.put("🥤 Coke"); // This will wait until something is taken out

        System.out.println(queue.take()); // 🍔 taken
        queue.put("🥤 Coke");              // Now this goes in
        System.out.println(queue);
    }
}
```

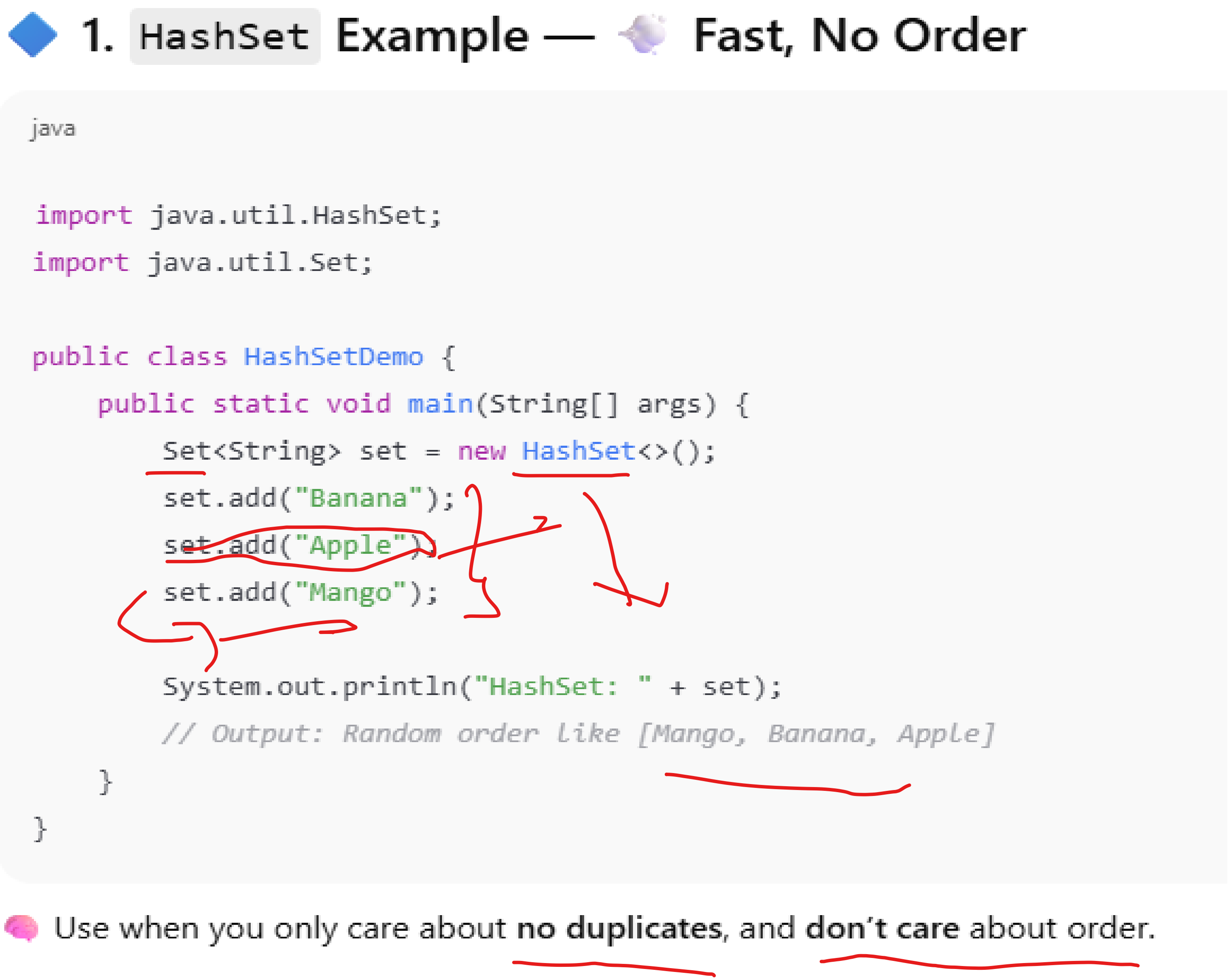
5 +1

6 → wa



🔶 `LinkedBlockingQueue` (🔗 Infinite Belt)

```java
import java.util.concurrent.*;

public class LinkedQueueSimple {
    public static void main(String[] args) throws InterruptedException {
        BlockingQueue<String> queue = new LinkedBlockingQueue<>();

        queue.put("📥 Message 1");
        queue.put("📥 Message 2");
        queue.put("📥 Message 3");
        // You can keep adding... no size limit (unless you give one)

        System.out.println(queue.take()); // 📥 Message 1
        System.out.println(queue);
    }
}
```

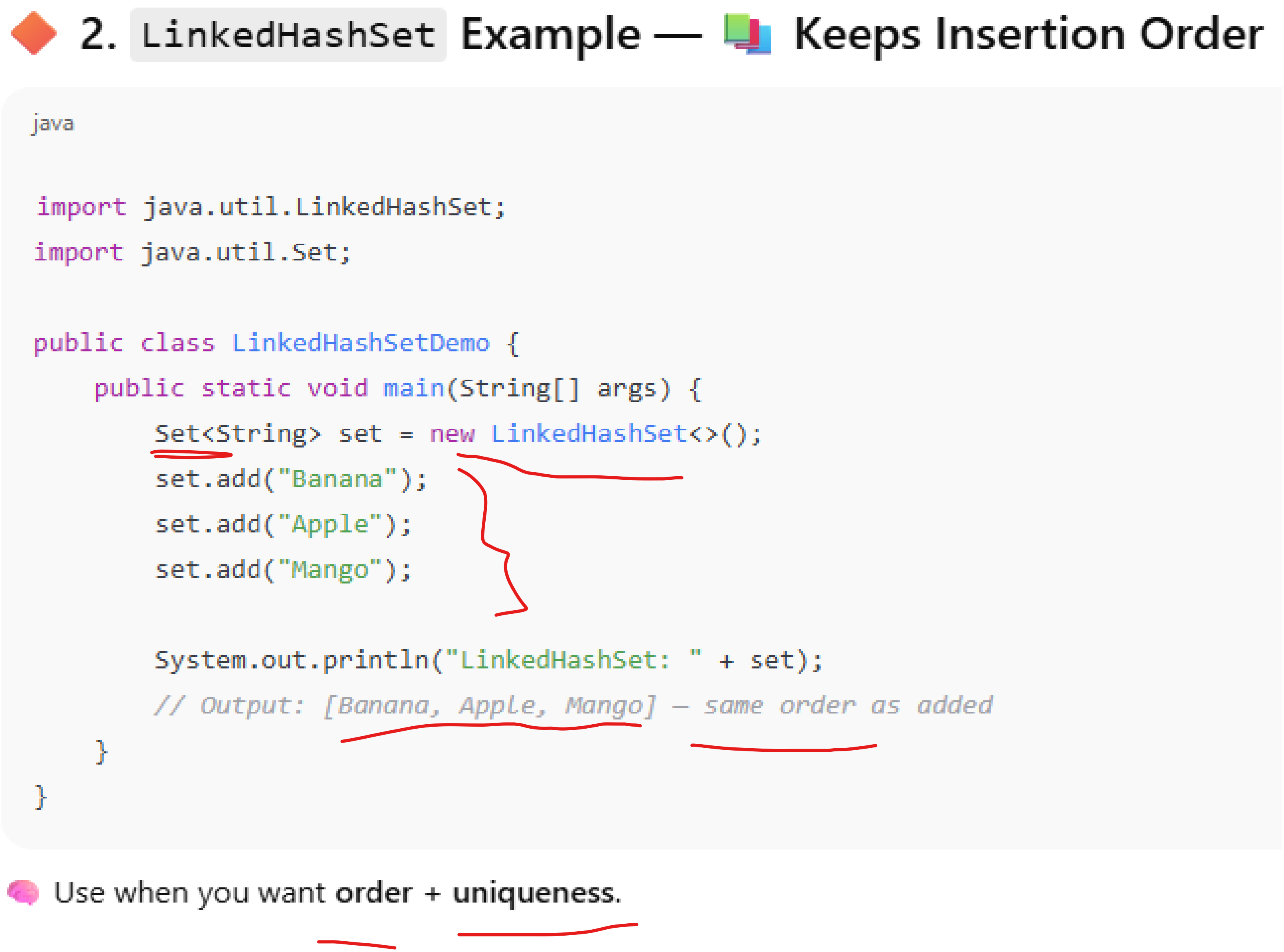


🔷 1. `HashSet` Example — Fast, No Order

java

```java
import java.util.HashSet;
import java.util.Set;

public class HashSetDemo {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        set.add("Banana");
        set.add("Apple");
        set.add("Mango");

        System.out.println("HashSet: " + set);
        // Output: Random order like [Mango, Banana, Apple]
    }
}
```

🧠 Use when you only care about **no duplicates**, and **don't care** about order.

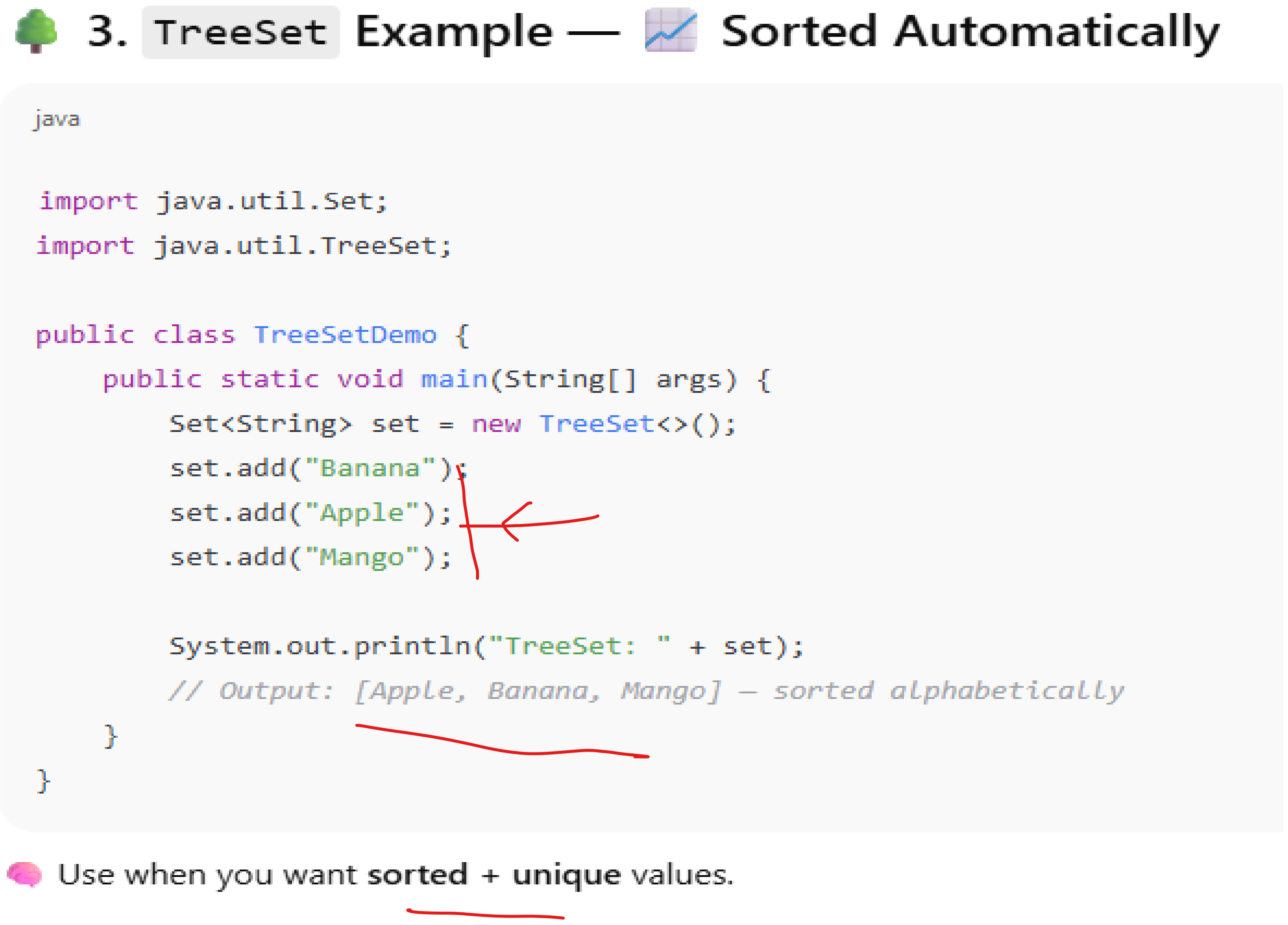


🔶 2. `LinkedHashSet` Example — 📚 Keeps Insertion Order

java

```java
import java.util.LinkedHashSet;
import java.util.Set;

public class LinkedHashSetDemo {
    public static void main(String[] args) {
        Set<String> set = new LinkedHashSet<>();
        set.add("Banana");
        set.add("Apple");
        set.add("Mango");

        System.out.println("LinkedHashSet: " + set);
        // Output: [Banana, Apple, Mango] — same order as added
    }
}
```

🧠 Use when you want **order** + **uniqueness.**

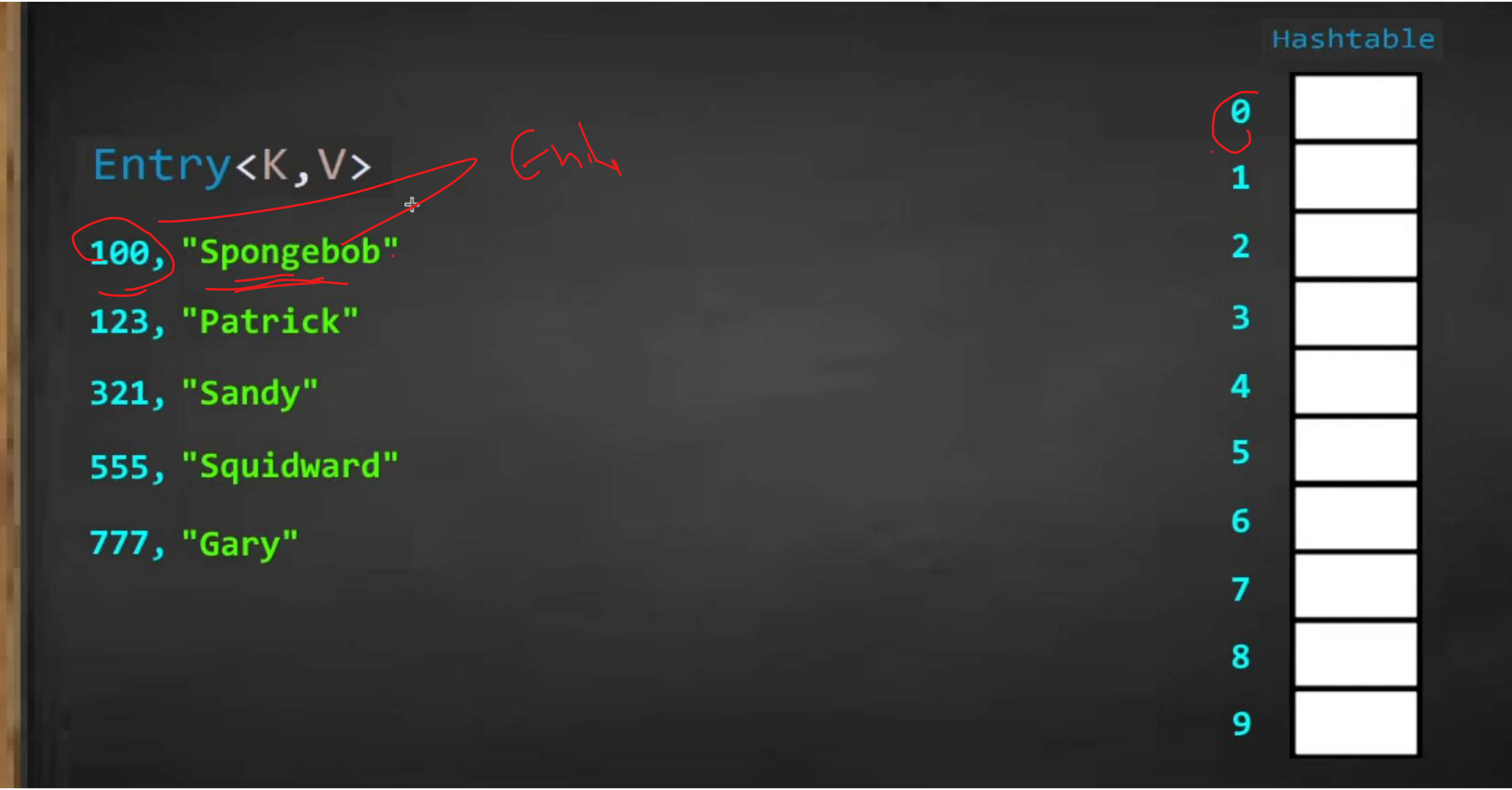


🌳 3. `TreeSet` Example — 📈 Sorted Automatically

java

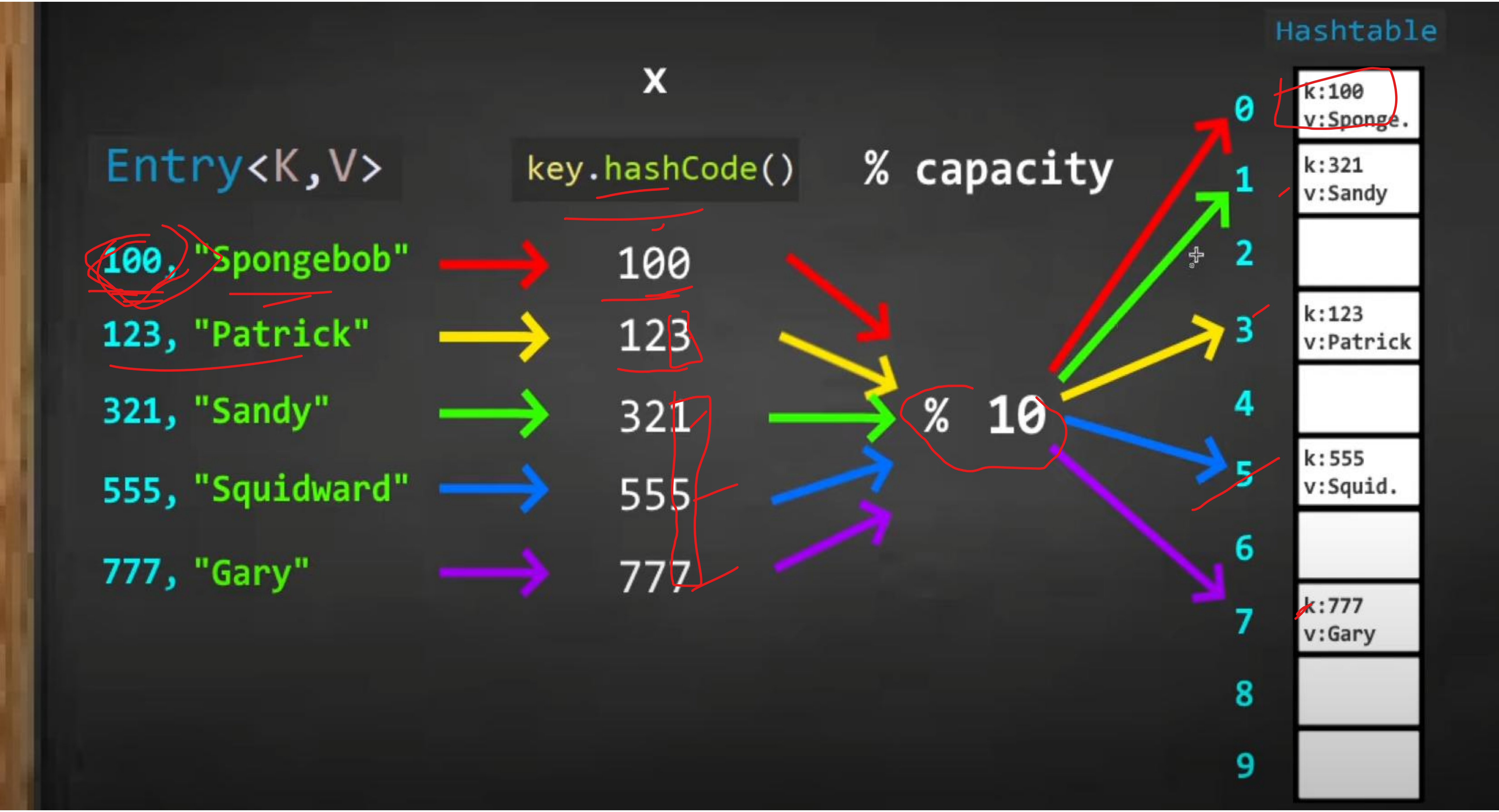
```java
import java.util.Set;
import java.util.TreeSet;

public class TreeSetDemo {
    public static void main(String[] args) {
        Set<String> set = new TreeSet<>();
        set.add("Banana");
        set.add("Apple");
        set.add("Mango");

        System.out.println("TreeSet: " + set);
        // Output: [Apple, Banana, Mango] — sorted alphabetically
    }
}
```

🧠 Use when you want **sorted + unique** values.



Entry<K,V>

```
100, "Spongebob"
123, "Patrick"
321, "Sandy"
555, "Squidward"
777, "Gary"
```



Hashtable

x

Entry<K,V>

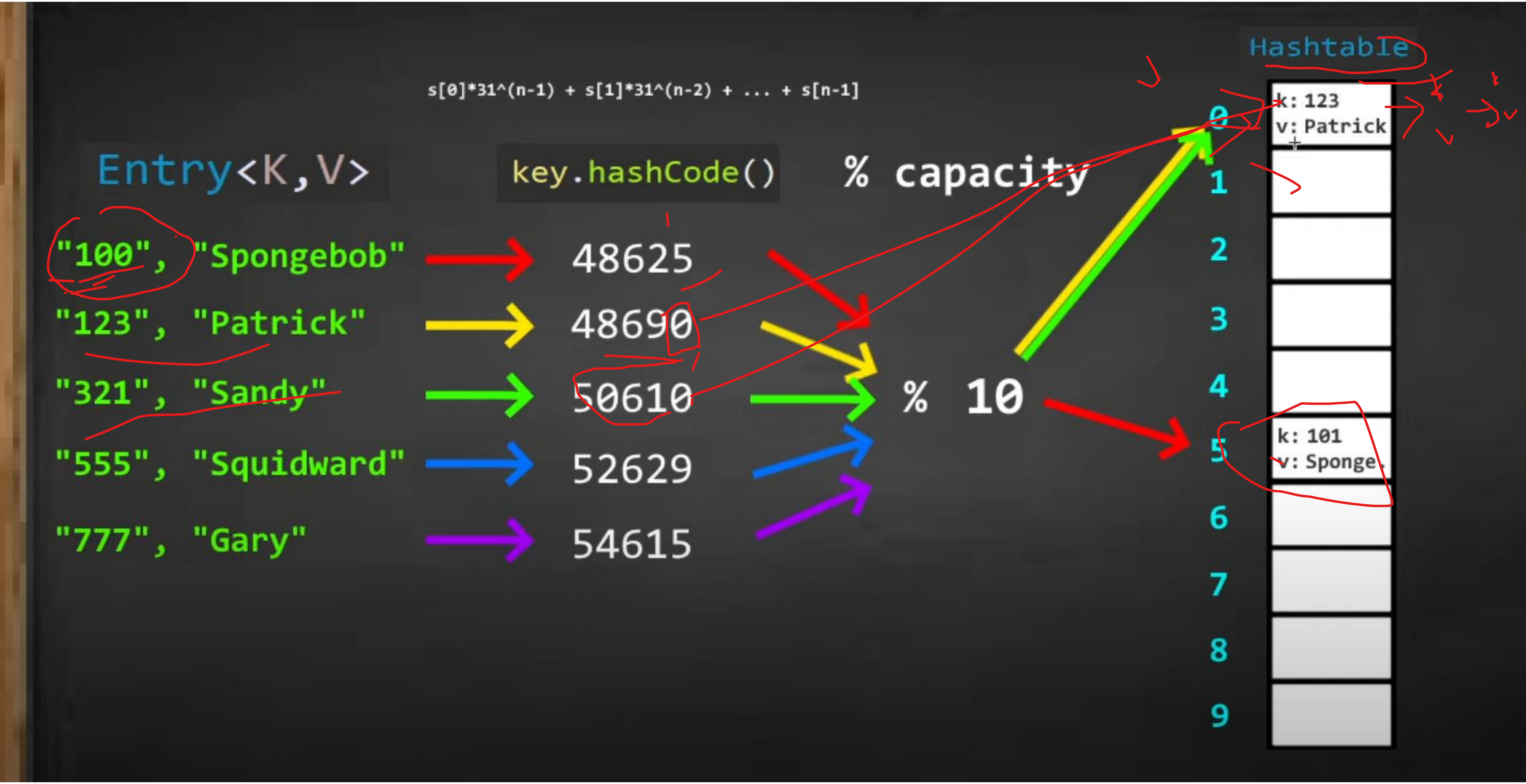
key.hashCode()

% capacity

Entry<K,V>	key.hashCode()
100, "Spongebob"	100
123, "Patrick"	123
321, "Sandy"	321
555, "Squidward"	555
777, "Gary"	777

% 10

	Hashtable
0	k:100 v:Sponge.
1	k:321 v:Sandy
2	
3	k:123 v:Patrick
4	
5	k:555 v:Squid.
6	
7	k:777 v:Gary
8	
9	



Hashtable

s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]

Entry<K,V>

key.hashCode()

% capacity

"100", "Spongebob" → 48625

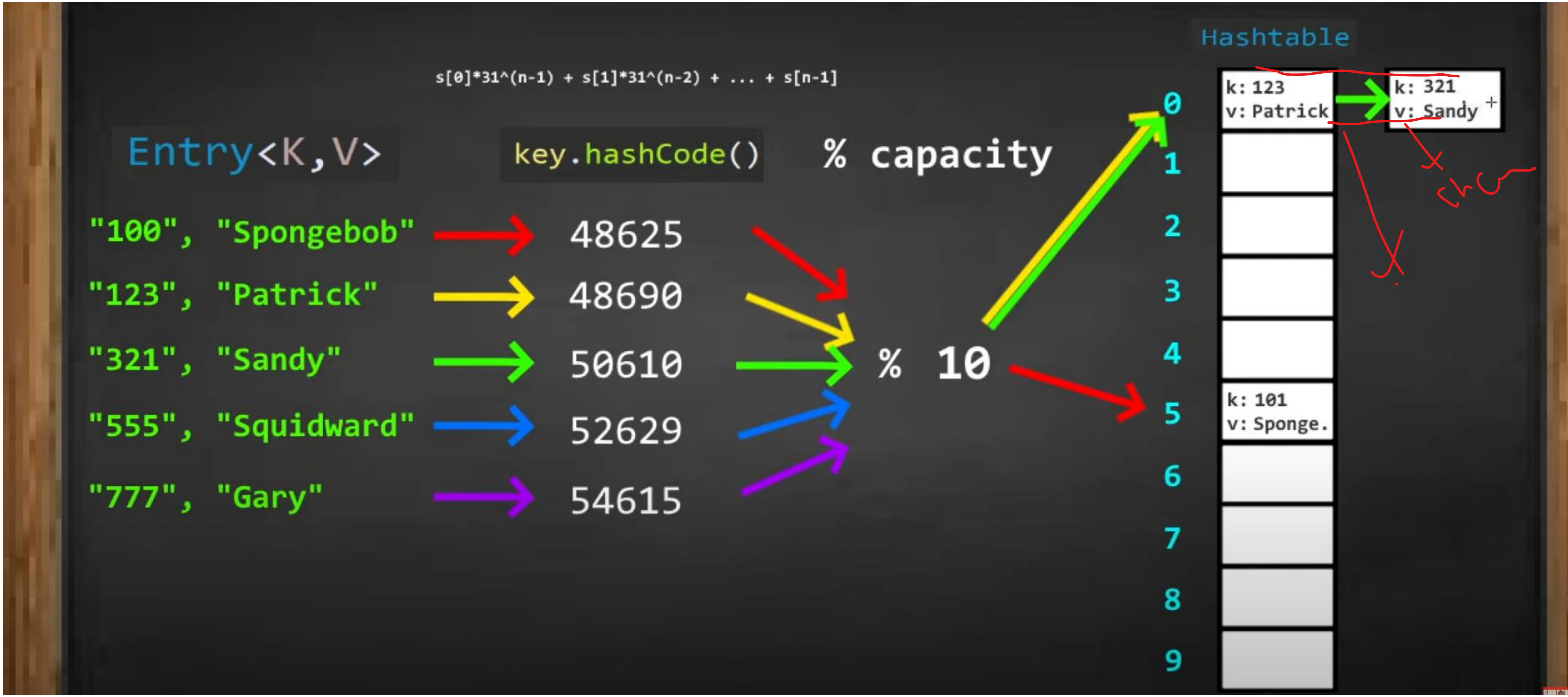
"123", "Patrick" → 48690

"321", "Sandy" → 50610

"555", "Squidward" → 52629

"777", "Gary" → 54615

% 10



Hashtable

s[0]*31^(n-1) + s[1]*31^(n-2) + ... + s[n-1]

Entry<K,V>   key.hashCode()   % capacity

"100", "Spongebob" → 48625

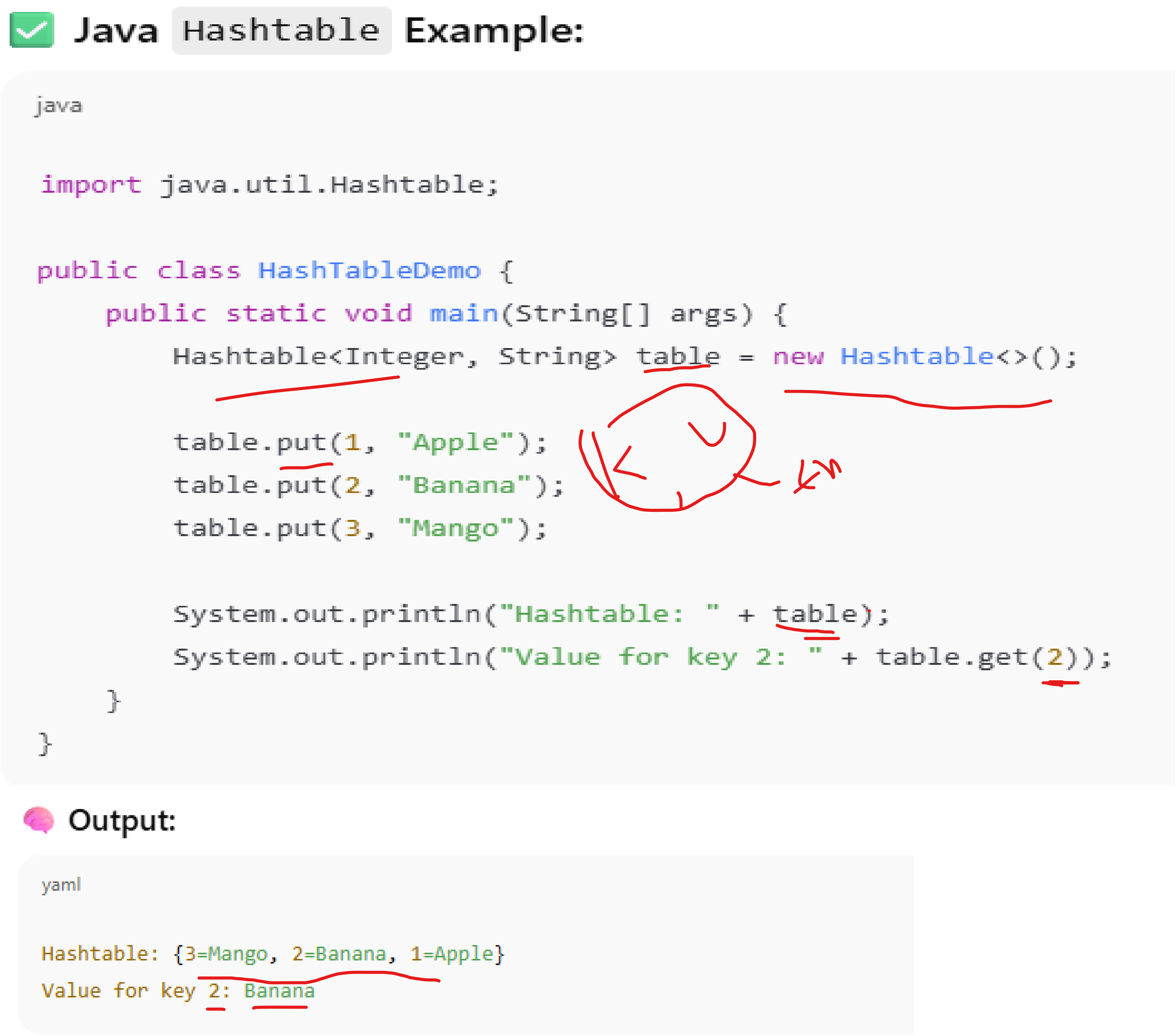
"123", "Patrick" → 48690

"321", "Sandy" → 50610

"555", "Squidward" → 52629

"777", "Gary" → 54615

% 10



✅ Java `Hashtable` Example:

java

```java
import java.util.Hashtable;

public class HashTableDemo {
    public static void main(String[] args) {
        Hashtable<Integer, String> table = new Hashtable<>();

        table.put(1, "Apple");
        table.put(2, "Banana");
        table.put(3, "Mango");

        System.out.println("Hashtable: " + table);
        System.out.println("Value for key 2: " + table.get(2));
    }
}
```

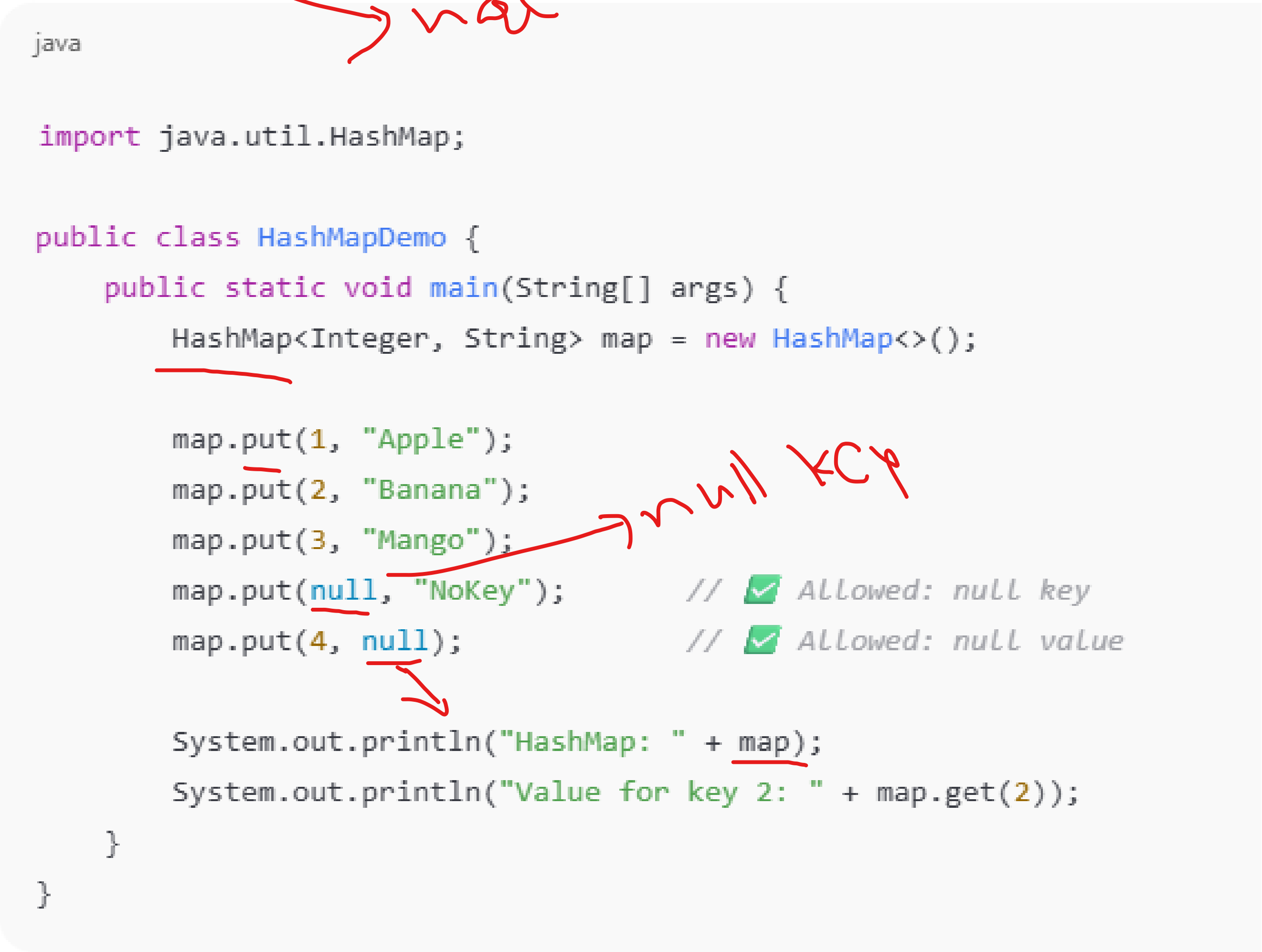
🧠 Output:

yaml

```
Hashtable: {3=Mango, 2=Banana, 1=Apple}
Value for key 2: Banana
```

✅ HashMap Example:

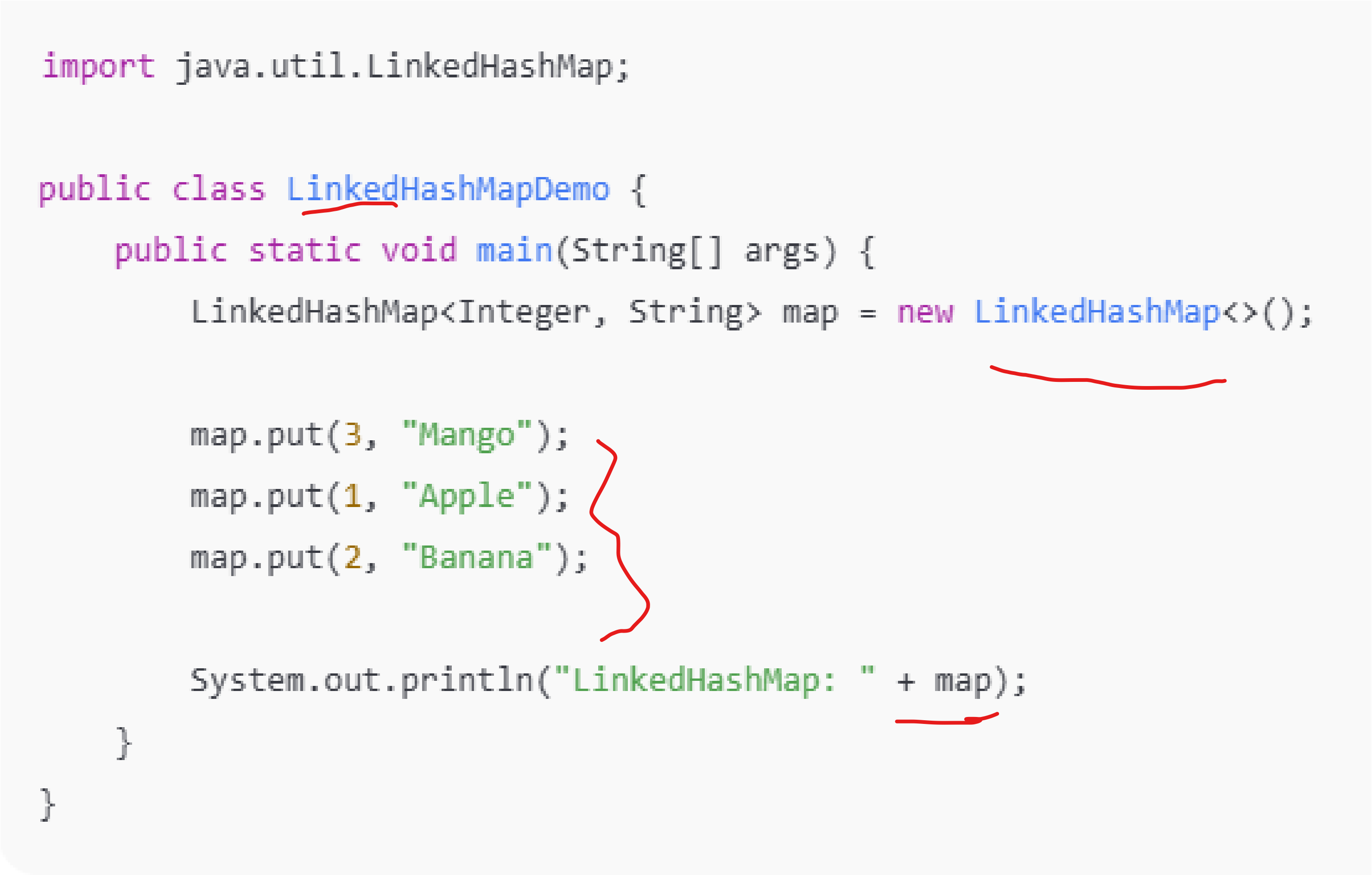
nau -



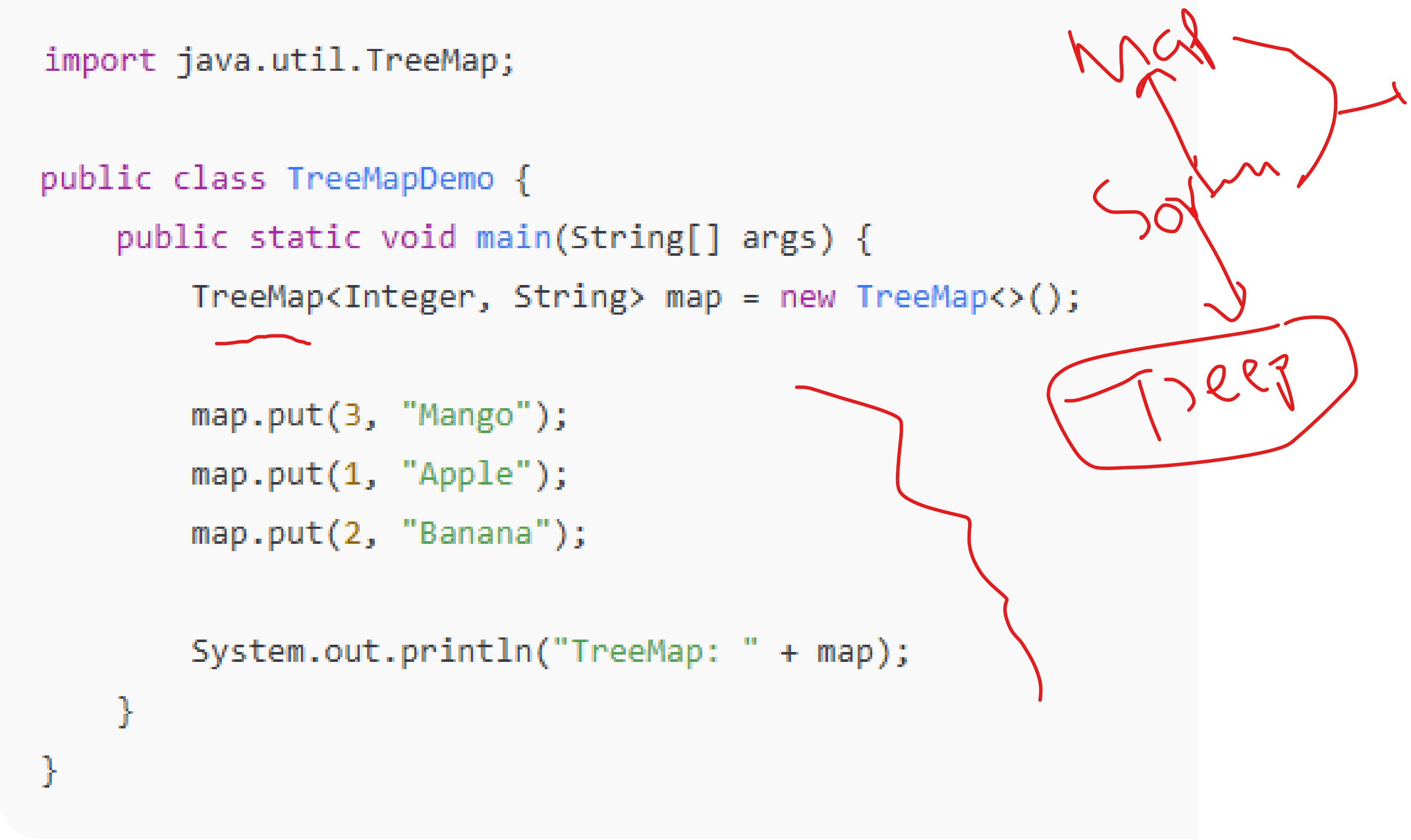
java

```java
import java.util.HashMap;

public class HashMapDemo {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();

        map.put(1, "Apple");
        map.put(2, "Banana");
        map.put(3, "Mango");
        map.put(null, "NoKey");        // ✅ Allowed: null key
        map.put(4, null);              // ✅ Allowed: null value

        System.out.println("HashMap: " + map);
        System.out.println("Value for key 2: " + map.get(2));
    }
}
```

null Key



```
import java.util.LinkedHashMap;

public class LinkedHashMapDemo {
    public static void main(String[] args) {
        LinkedHashMap<Integer, String> map = new LinkedHashMap<>();

        map.put(3, "Mango");
        map.put(1, "Apple");
        map.put(2, "Banana");

        System.out.println("LinkedHashMap: " + map);
    }
}
```



```java
import java.util.TreeMap;

public class TreeMapDemo {
    public static void main(String[] args) {
        TreeMap<Integer, String> map = new TreeMap<>();

        map.put(3, "Mango");
        map.put(1, "Apple");
        map.put(2, "Banana");

        System.out.println("TreeMap: " + map);
    }
}
```