$$\underbrace{\text{Collection}}_{\text{Group of Objects}}\ \underbrace{\text{Frameworks}}_{\text{Set of classes \& interfaces which are pre-built}} \Rightarrow \text{Java API} = \text{Package} :\rightarrow \text{java.Util}$$

(I) Iterable   forEach loop   for(T data : collectn)

$$\begin{tikzcd}
 & \text{(I) Iterable} & \\
\text{Iterator} \arrow[r, Rightarrow, "\text{hasA}"] & \text{(I) Collection} \arrow[u, "\text{extends}"] & \text{(I) Queue} \arrow[l] \\
\text{ListIterator} \arrow[u, "\text{extends}"] \arrow[r, Rightarrow] & \text{(I) List} \arrow[u] & \text{(I) Set} \arrow[ul]
\end{tikzcd}$$

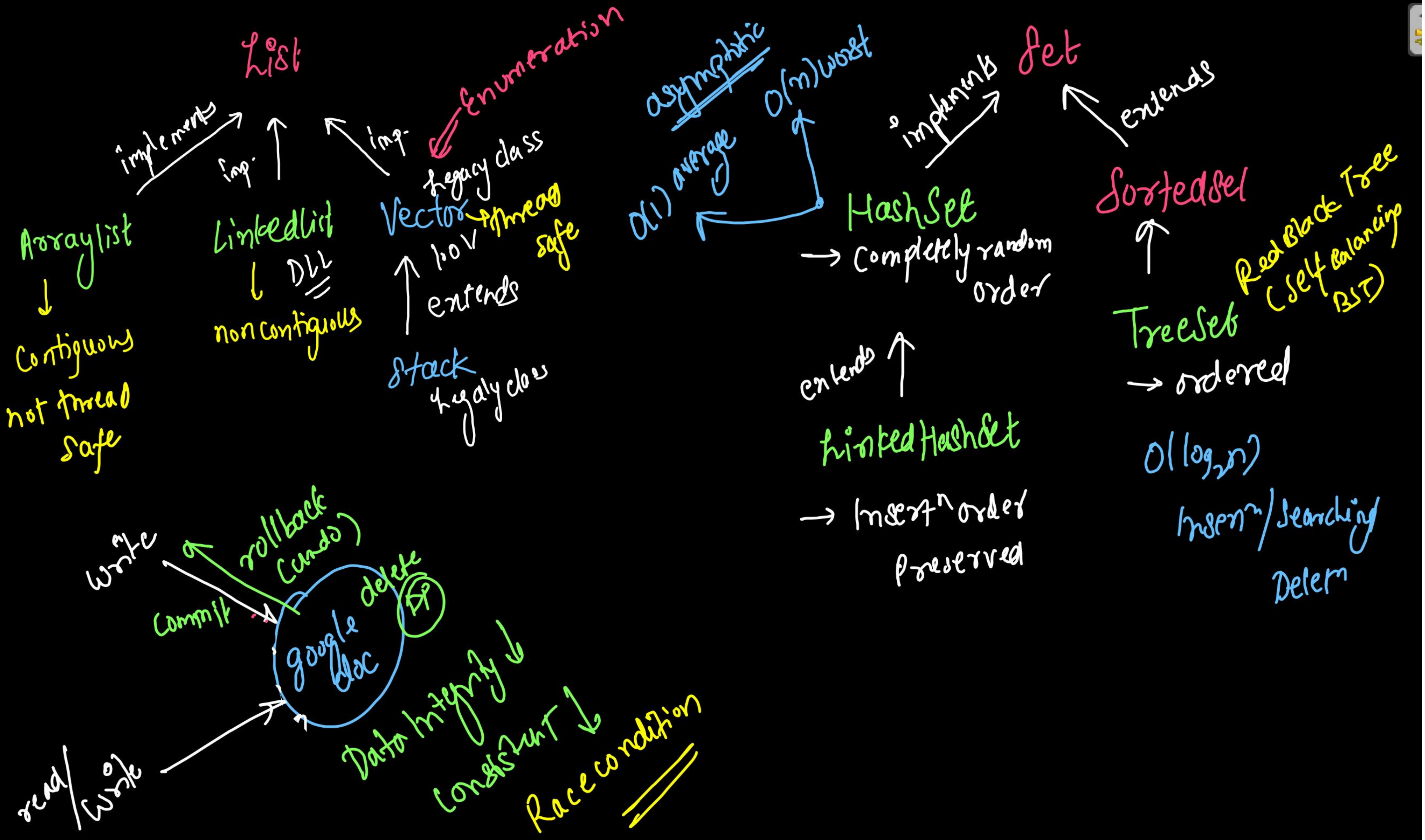
forward iteratn

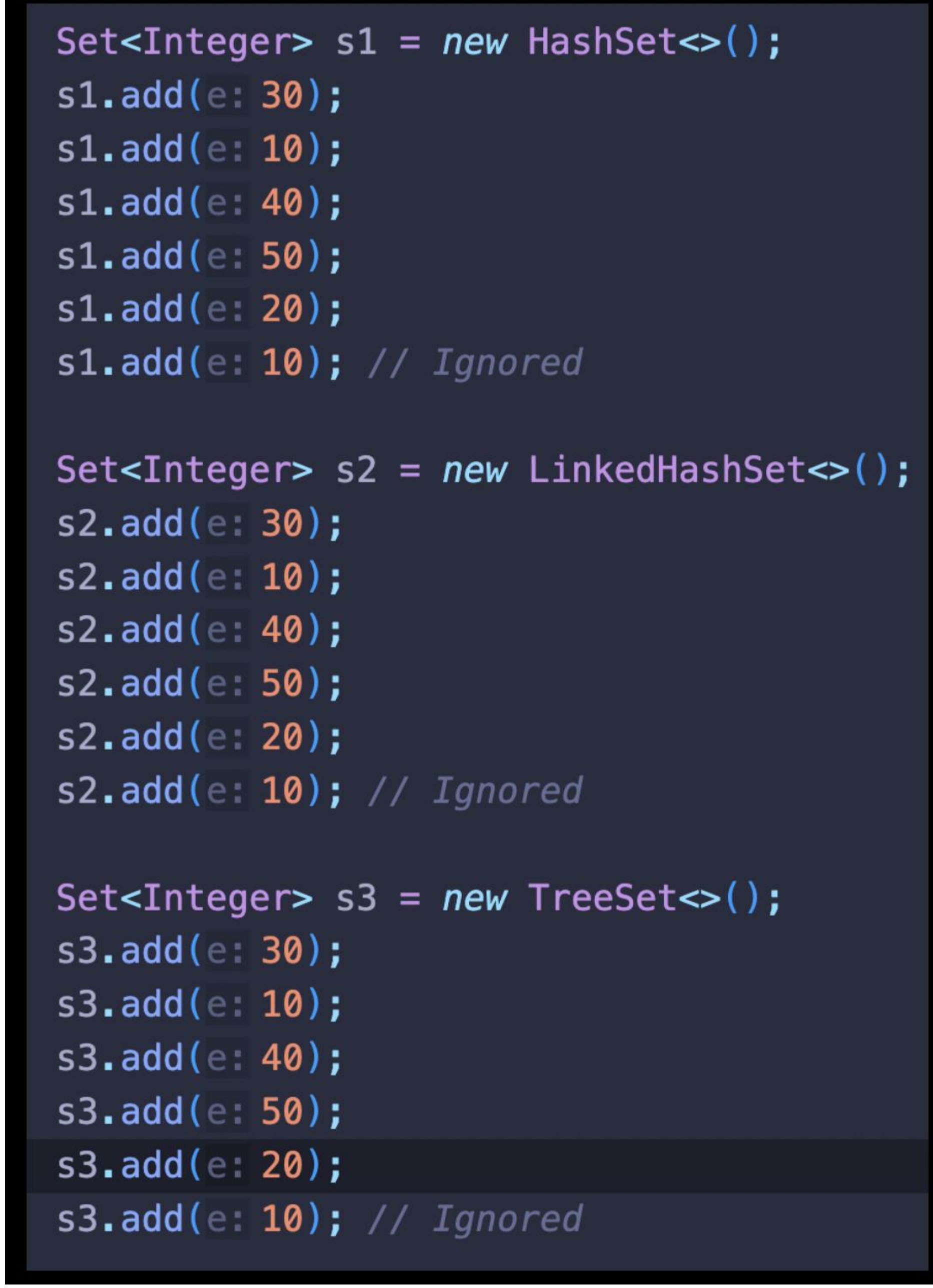
forward & backward traversal

List
- → linear data structure
- → duplicate keys
- → Forword & Backward
- → indexed
- → Insertion order → preserved

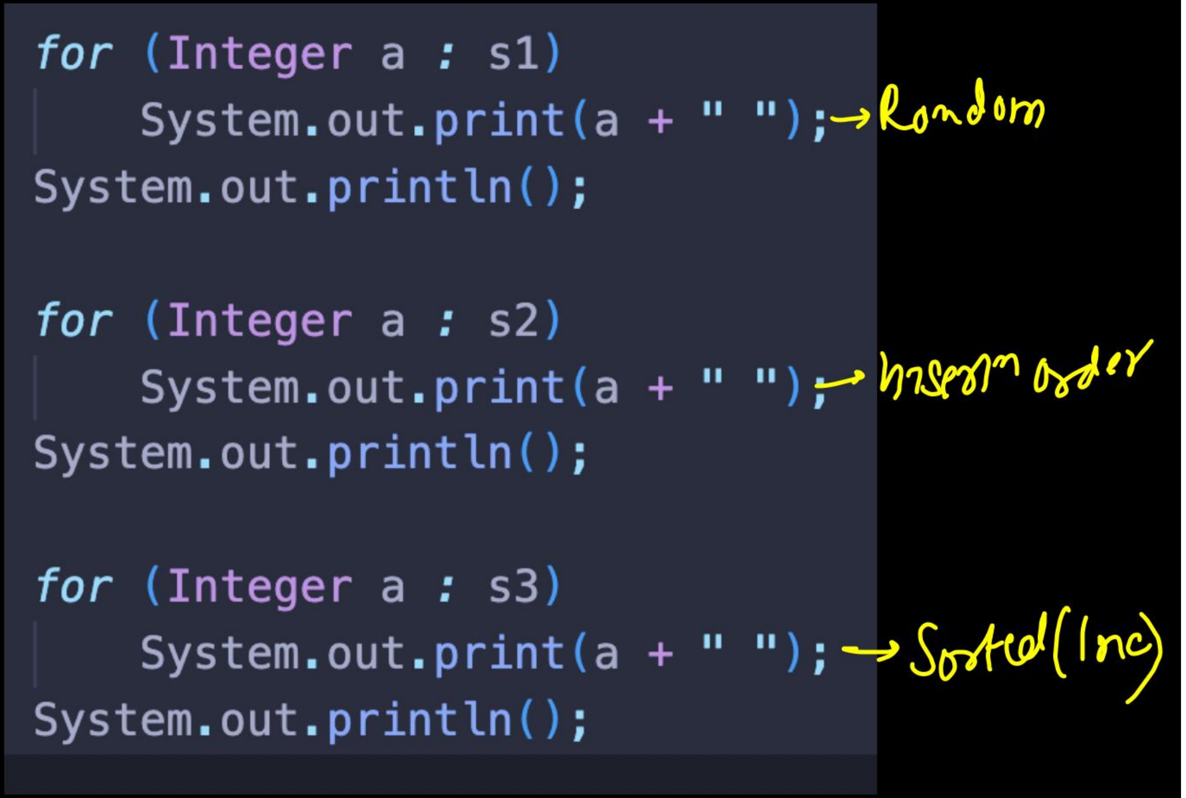
Set
- → hashing/heirarchial
- → unique keys
- → Forward Traversal
- → no indexing
- → insertion order → not preserved

Java 8+ # forEach method (lambda expression)


List
implements
inp.
imp.
Enumeration
Arraylist
Contiguous
not thread safe
Linkedlist
DLL
non contiguous
Vector
legacy class
thread safe
10V
extends
Stack
legacy class
asymptotic
O(1) average
O(n) worst
implements
Set
extends
HashSet
→ completely random order
extends
LinkedHashSet
→ insertn order preserved
Sortedset
TreeSet
Red Black Tree (self balancing BST)
→ ordered
O(log₂n)
Insertn/searching
Deletn
write
rollback (undo)
commit
google doc
delete
read/write
Data Integrity
consistent
Race condition



```java
Set<Integer> s1 = new HashSet<>();
s1.add(e: 30);
s1.add(e: 10);
s1.add(e: 40);
s1.add(e: 50);
s1.add(e: 20);
s1.add(e: 10); // Ignored

Set<Integer> s2 = new LinkedHashSet<>();
s2.add(e: 30);
s2.add(e: 10);
s2.add(e: 40);
s2.add(e: 50);
s2.add(e: 20);
s2.add(e: 10); // Ignored

Set<Integer> s3 = new TreeSet<>();
s3.add(e: 30);
s3.add(e: 10);
s3.add(e: 40);
s3.add(e: 50);
s3.add(e: 20);
s3.add(e: 10); // Ignored
```

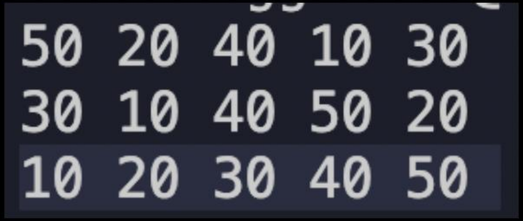


```java
for (Integer a : s1)
    System.out.print(a + " ");
System.out.println();

for (Integer a : s2)
    System.out.print(a + " ");
System.out.println();

for (Integer a : s3)
    System.out.print(a + " ");
System.out.println();
```

→ Random

→ Insertion order

→ Sorted(Inc)



```
50 20 40 10 30
30 10 40 50 20
10 20 30 40 50
```

$$\begin{tikzcd}
 & \text{Queue} & \\
\text{Priority Queue} \arrow[ur, "\text{implements}"] & & \text{Deque} \arrow[ul, "\text{extends}"'] \\
 & & \text{ArrayDeque} \arrow[u]
\end{tikzcd}$$

duplicates can be stored

Priority Queue

Heap

Insertn :→ O(log n)

Get Highest Priority :→ O(1)

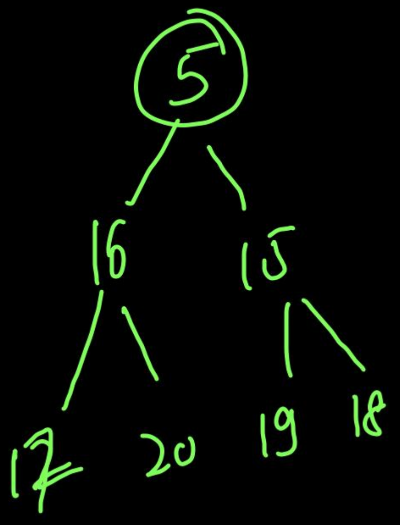
delete Highest Priority :→ O(log₂ n)

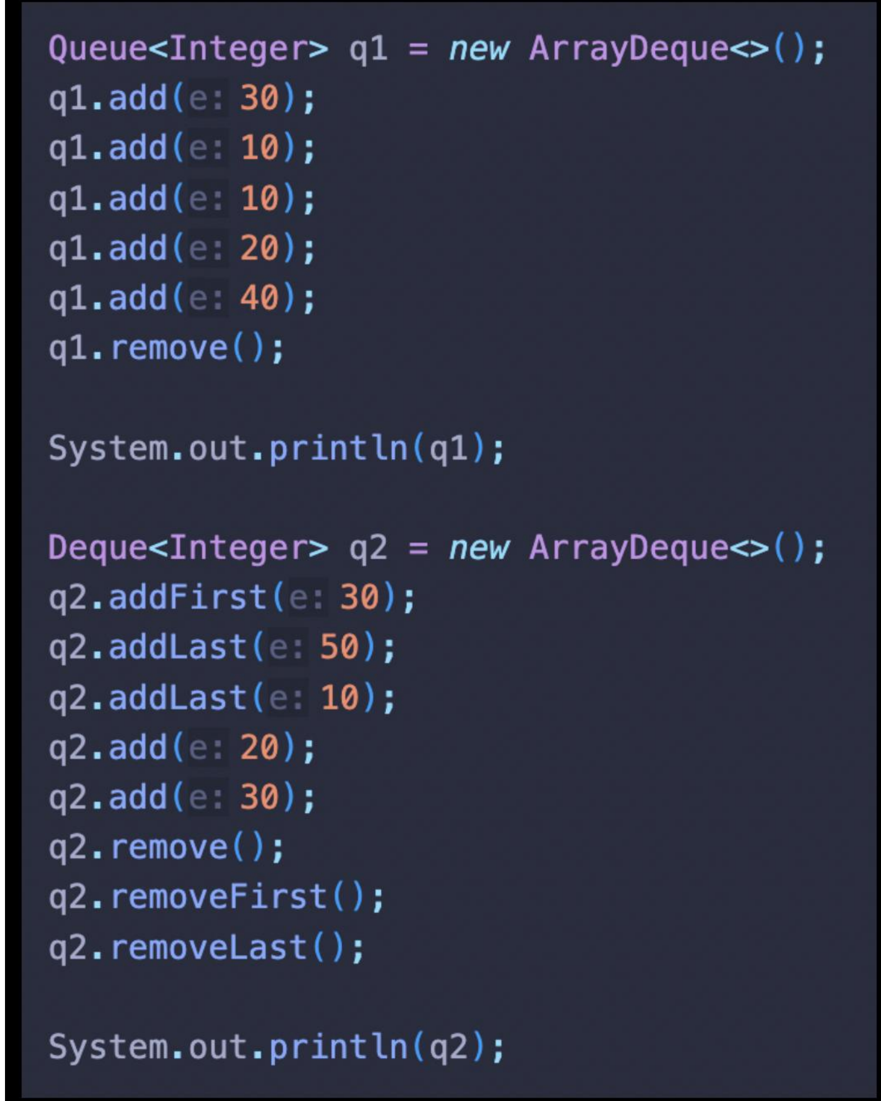
Searching :→ O(n)

ArrayDeque

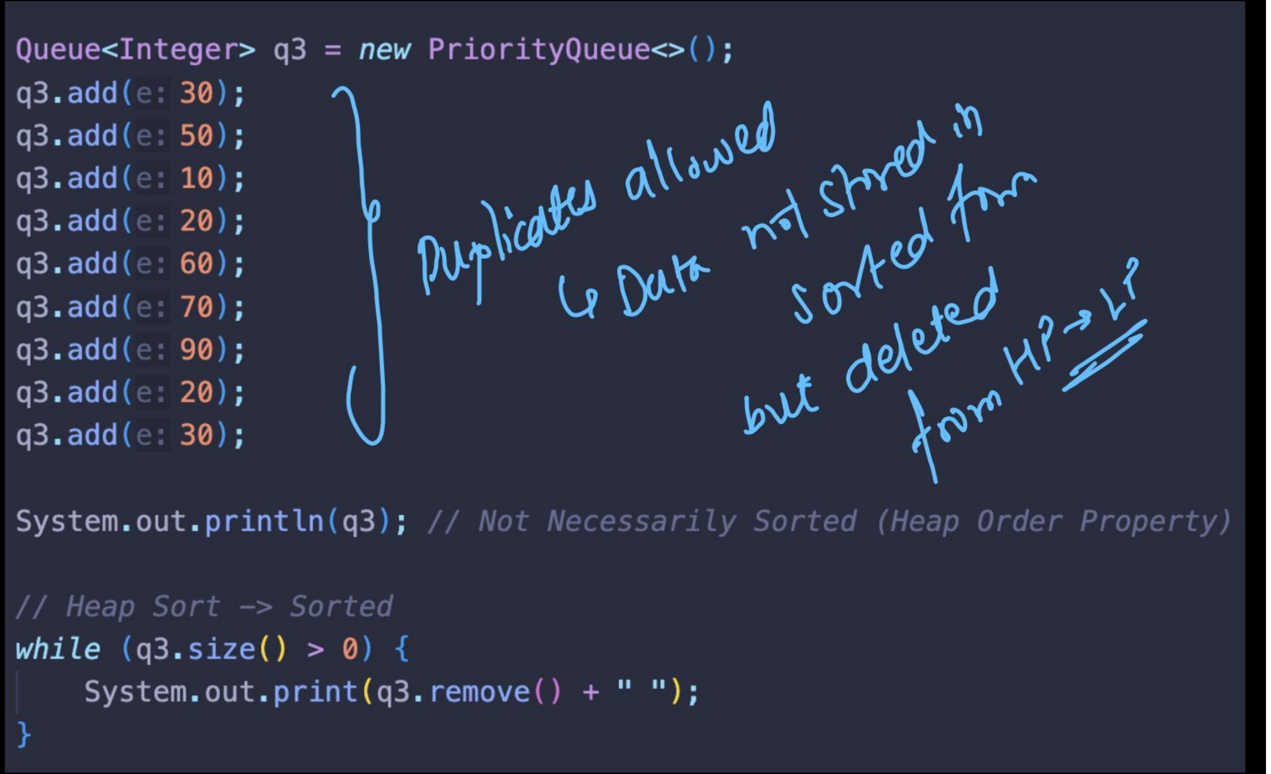
- addFirst – O(1)
- addLast ✓ – O(1)
- removeFirst ✓ – O(1)
- removeLast – O(1)

$\left.\begin{array}{l}\text{addFirst} \\ \left.\begin{array}{l}\text{addLast} \\ \text{removeFirst}\end{array}\right\}\text{queue} \\ \text{removeLast}\end{array}\right\}\text{Deque}$





```java
Queue<Integer> q1 = new ArrayDeque<>();
q1.add(e: 30);
q1.add(e: 10);
q1.add(e: 10);
q1.add(e: 20);
q1.add(e: 40);
q1.remove();

System.out.println(q1);

Deque<Integer> q2 = new ArrayDeque<>();
q2.addFirst(e: 30);
q2.addLast(e: 50);
q2.addLast(e: 10);
q2.add(e: 20);
q2.add(e: 30);
q2.remove();
q2.removeFirst();
q2.removeLast();

System.out.println(q2);
```



```java
Queue<Integer> q3 = new PriorityQueue<>();
q3.add(e: 30);
q3.add(e: 50);
q3.add(e: 10);
q3.add(e: 20);
q3.add(e: 60);
q3.add(e: 70);
q3.add(e: 90);
q3.add(e: 20);
q3.add(e: 30);

System.out.println(q3); // Not Necessarily Sorted (Heap Order Property)

// Heap Sort -> Sorted
while (q3.size() > 0) {
    System.out.print(q3.remove() + " ");
}
```

Duplicates allowed
↳ Data not stored in Sorted form
but deleted
from HP → LP

$$\begin{tikzcd}
 & \text{Map (I)} & \\
\text{(class) Hashmap} \arrow[ur, "\text{implements}"] & & \text{SortedSet} \arrow[ul, "\text{extends}"'] \\
\text{LinkedHashmap} \arrow[u] & & \text{Tree map} \arrow[u]
\end{tikzcd}$$

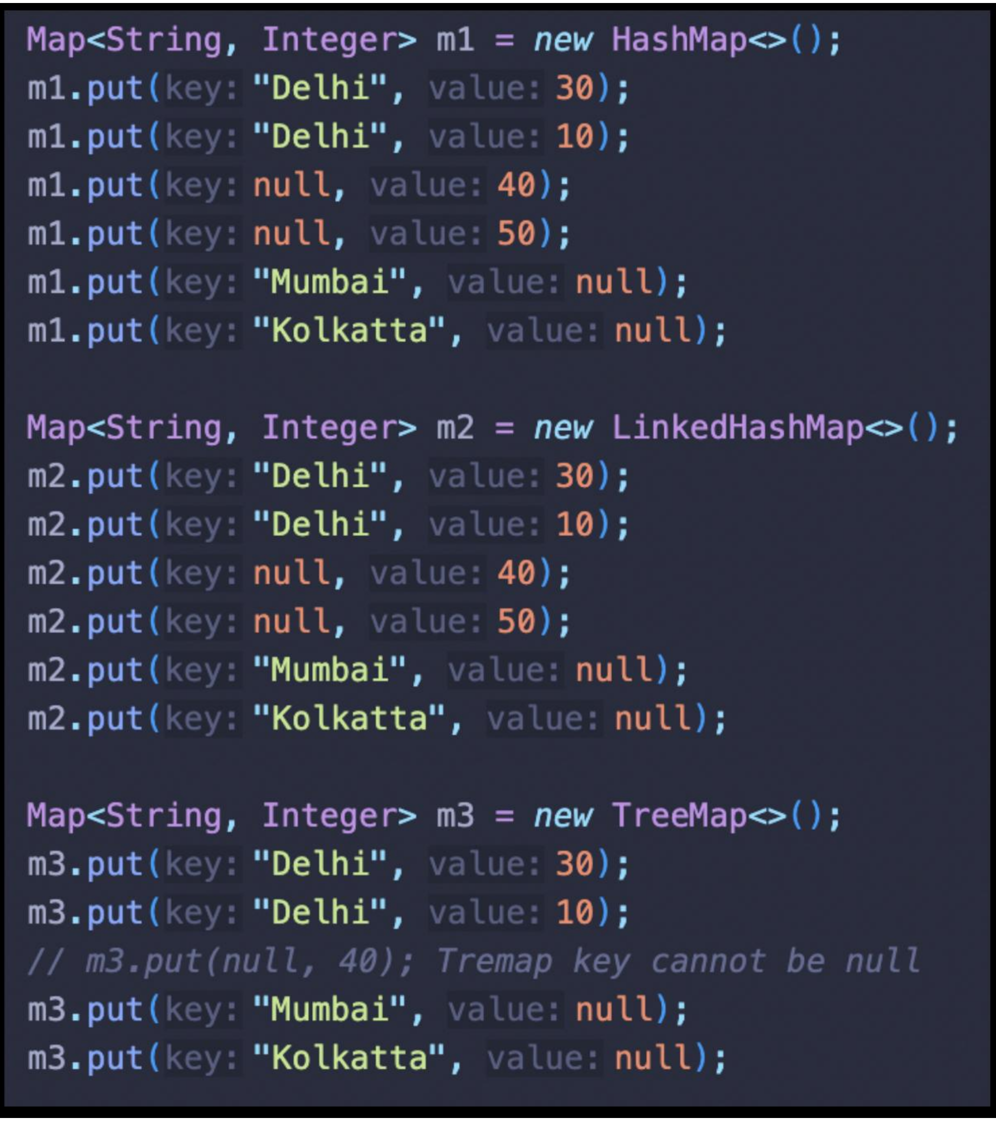
key-value pairs
- ↳ unique keys
- ↳ null can be inserted as key (once) and as values (many)
- ↳ duplicate values will be there

Hashing
→ getKey - O(1) avg
  - O(n) worst
→ getValue - O(n) avg/worst

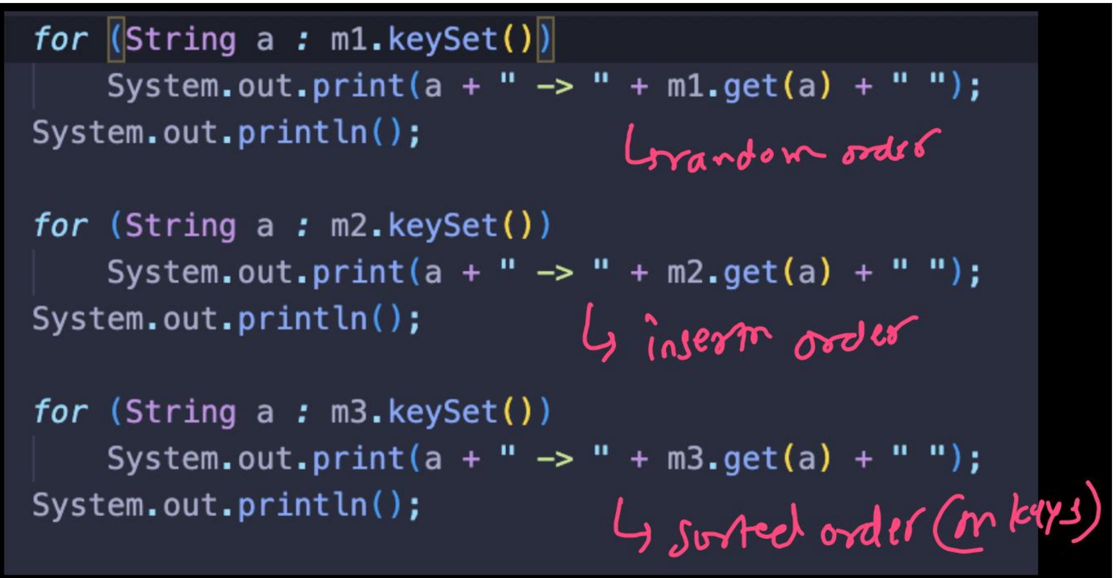
rest all operations same as HashSet

LinkedHashmap
↳ order preserved

Tree map
Red Black BST
O(log n)
Insertion/searching/deletion
↳ Sorting on keys



```java
Map<String, Integer> m1 = new HashMap<>();
m1.put(key: "Delhi", value: 30);
m1.put(key: "Delhi", value: 10);
m1.put(key: null, value: 40);
m1.put(key: null, value: 50);
m1.put(key: "Mumbai", value: null);
m1.put(key: "Kolkatta", value: null);

Map<String, Integer> m2 = new LinkedHashMap<>();
m2.put(key: "Delhi", value: 30);
m2.put(key: "Delhi", value: 10);
m2.put(key: null, value: 40);
m2.put(key: null, value: 50);
m2.put(key: "Mumbai", value: null);
m2.put(key: "Kolkatta", value: null);

Map<String, Integer> m3 = new TreeMap<>();
m3.put(key: "Delhi", value: 30);
m3.put(key: "Delhi", value: 10);
// m3.put(null, 40); Tremap key cannot be null
m3.put(key: "Mumbai", value: null);
m3.put(key: "Kolkatta", value: null);
```



```java
for (String a : m1.keySet())
    System.out.print(a + " -> " + m1.get(a) + " ");
System.out.println();
```

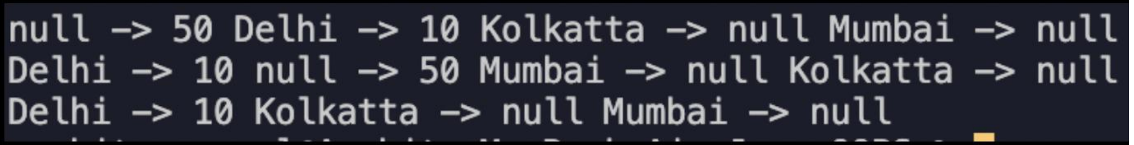
↳ random order

```java
for (String a : m2.keySet())
    System.out.print(a + " -> " + m2.get(a) + " ");
System.out.println();
```

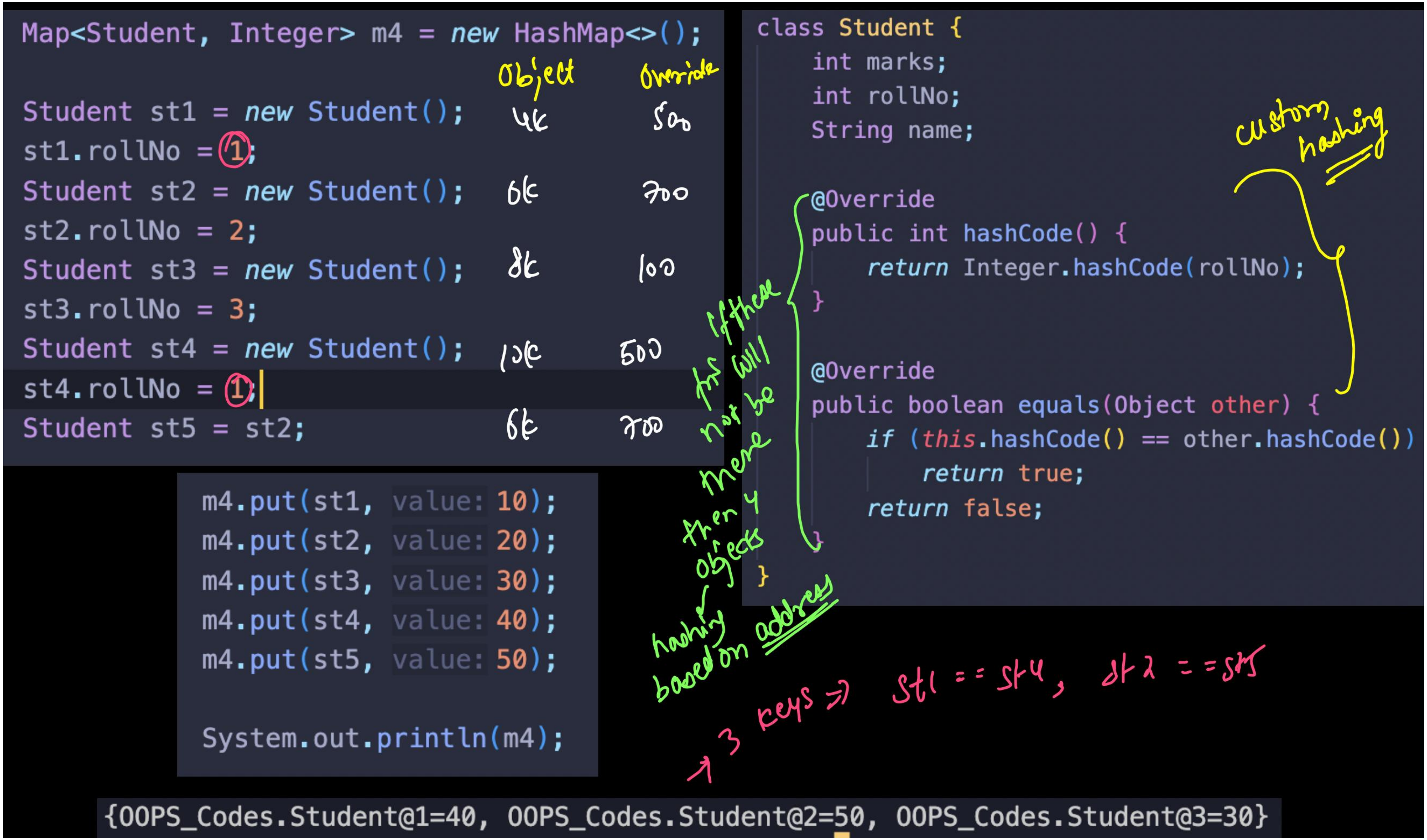
↳ insertn order

```java
for (String a : m3.keySet())
    System.out.print(a + " -> " + m3.get(a) + " ");
System.out.println();
```

↳ sorted order (on keys)



```
null -> 50 Delhi -> 10 Kolkatta -> null Mumbai -> null
Delhi -> 10 null -> 50 Mumbai -> null Kolkatta -> null
Delhi -> 10 Kolkatta -> null Mumbai -> null
```



```java
Map<Student, Integer> m4 = new HashMap<>();

Student st1 = new Student();
st1.rollNo = 1;
Student st2 = new Student();
st2.rollNo = 2;
Student st3 = new Student();
st3.rollNo = 3;
Student st4 = new Student();
st4.rollNo = 1;
Student st5 = st2;
```

	Object	Override
st1	4k	500
st2	6k	700
st3	8k	100
st4	10k	500
st5	6k	700

```java
m4.put(st1, value: 10);
m4.put(st2, value: 20);
m4.put(st3, value: 30);
m4.put(st4, value: 40);
m4.put(st5, value: 50);

System.out.println(m4);
```

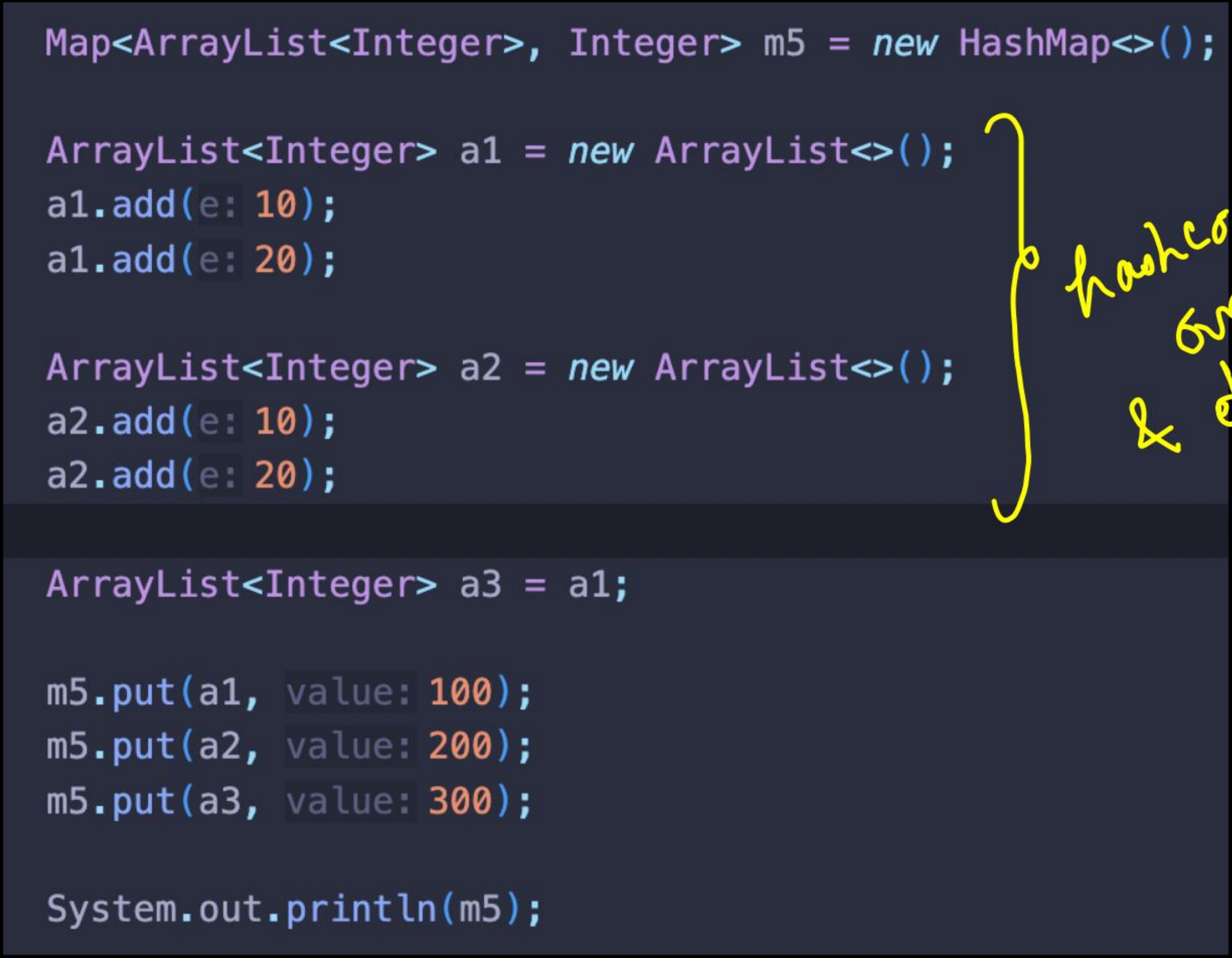
```java
class Student {
    int marks;
    int rollNo;
    String name;

    @Override
    public int hashCode() {
        return Integer.hashCode(rollNo);
    }

    @Override
    public boolean equals(Object other) {
        if (this.hashCode() == other.hashCode())
            return true;
        return false;
    }
}
```

custom hashing

If these fns will not be there then 4 objects hashing based on address

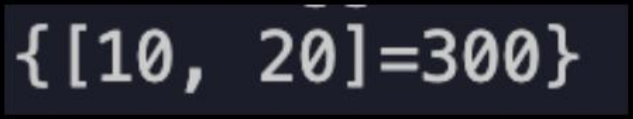
→ 3 keys ⇒ st1 == st4, st2 == st5

```
{OOPS_Codes.Student@1=40, OOPS_Codes.Student@2=50, OOPS_Codes.Student@3=30}
```



```java
Map<ArrayList<Integer>, Integer> m5 = new HashMap<>();

ArrayList<Integer> a1 = new ArrayList<>();
a1.add(e: 10);
a1.add(e: 20);

ArrayList<Integer> a2 = new ArrayList<>();
a2.add(e: 10);
a2.add(e: 20);

ArrayList<Integer> a3 = a1;

m5.put(a1, value: 100);
m5.put(a2, value: 200);
m5.put(a3, value: 300);

System.out.println(m5);
```

hashcode is overrided & data is compares



```
{[10, 20]=300}
```