ORDERED ASSOCIATIVE CONTAINERS AKA TREEMAPS

MAPS ARE ASSOCIATIVE
CONTAINERS

THEY ARE USED TO MAP KEYS TO CORRESPONDING VALUES, AND TO ACCESS THEM QUICKLY

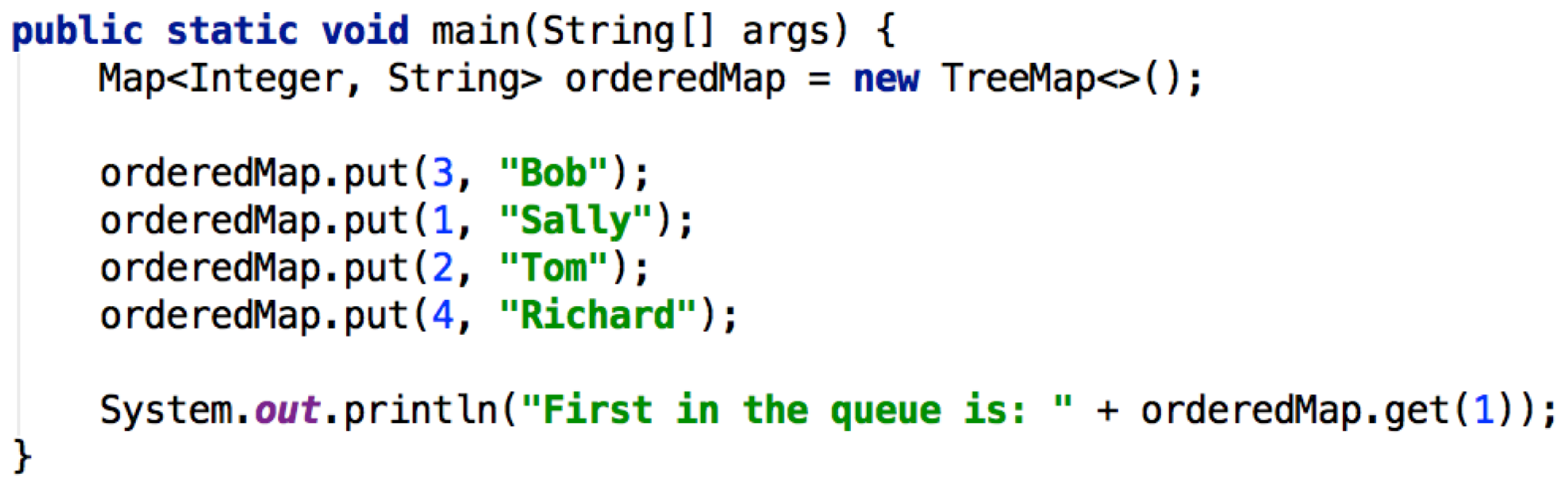
THEY CAN'T REALLY BE USED FOR ACCESSING ELEMENTS IN ANY ORDER

ITERATING THROUGH THE MAP GIVES YOU ELEMENTS IN RANDOM ORDER

TURNS OUT, THIS STATEMENT IS NOT ENTIRELY TRUE

MEET THE TREE MAP, AN ORDERED ASSOCIATIVE CONTAINER!

THE TREE MAP WORKS JUST LIKE ANY OTHER MAP - IT HAS ALL THE BASIC METHODS WHICH APPLY TO MAPS IN GENERAL

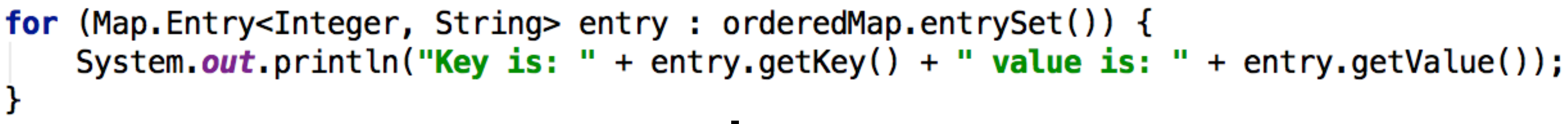


```
public static void main(String[] args) {
    Map<Integer, String> orderedMap = new TreeMap<>();

    orderedMap.put(3, "Bob");
    orderedMap.put(1, "Sally");
    orderedMap.put(2, "Tom");
    orderedMap.put(4, "Richard");

    System.out.println("First in the queue is: " + orderedMap.get(1));
}
```

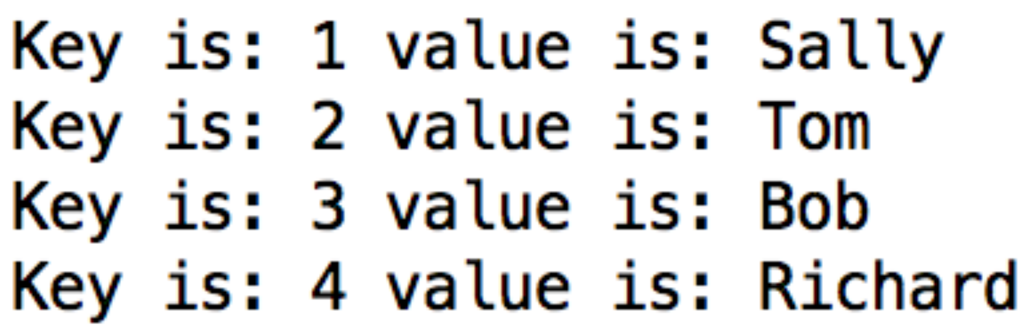
YOU CAN ADD KEY-VALUE PAIRS TO THE MAP AND RETRIEVE VALUES USING KEYS

THE INTERESTING PART IS THAT WHEN YOU ITERATE THROUGH THIS MAP, YOU CAN ACCESS THE KEY-VALUE PAIRS IN ORDER

THE PAIRS IN THE MAP WILL BE RETURNED IN THE "NATURAL ORDER" OF THE KEYS



```java
for (Map.Entry<Integer, String> entry : orderedMap.entrySet()) {
    System.out.println("Key is: " + entry.getKey() + " value is: " + entry.getValue());
}
```



```
Key is: 1 value is: Sally
Key is: 2 value is: Tom
Key is: 3 value is: Bob
Key is: 4 value is: Richard
```

NATURAL ORDER IS MAINTAINED FOR THE KEY TYPES

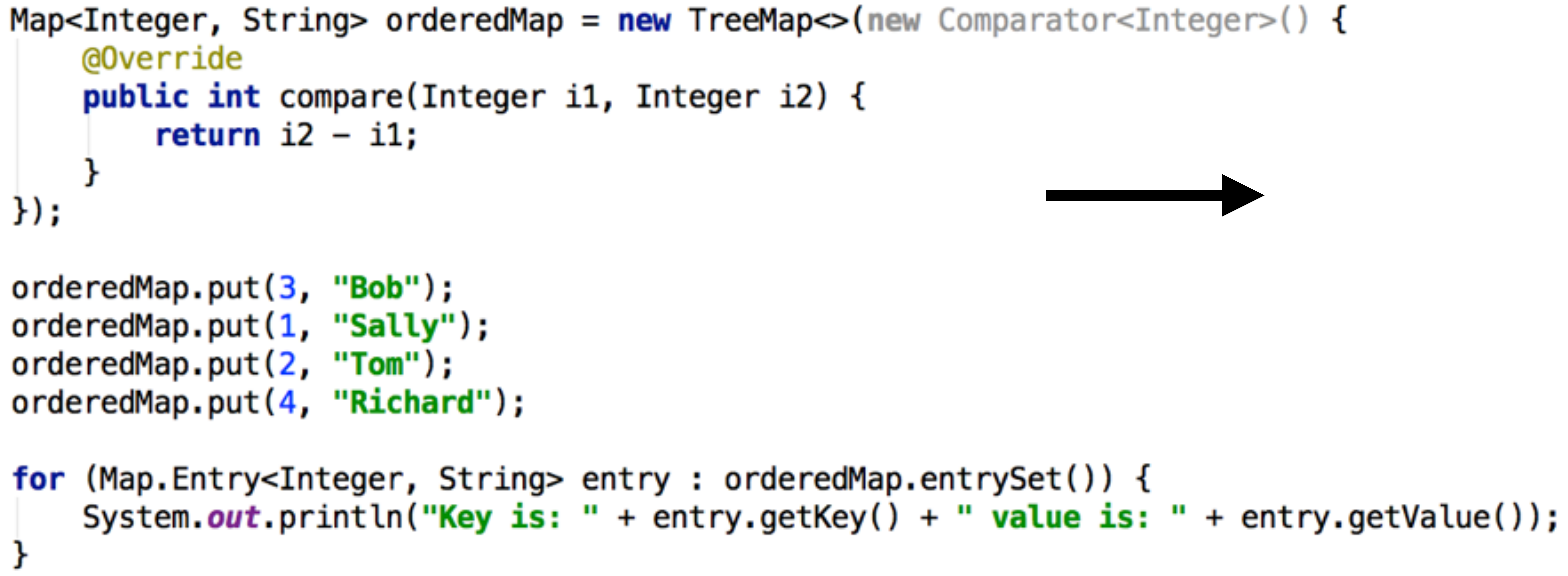
THIS IS ASCENDING ORDER FOR INTEGERS

WHAT IF YOU DO NOT WANT THE NATURAL ORDER?

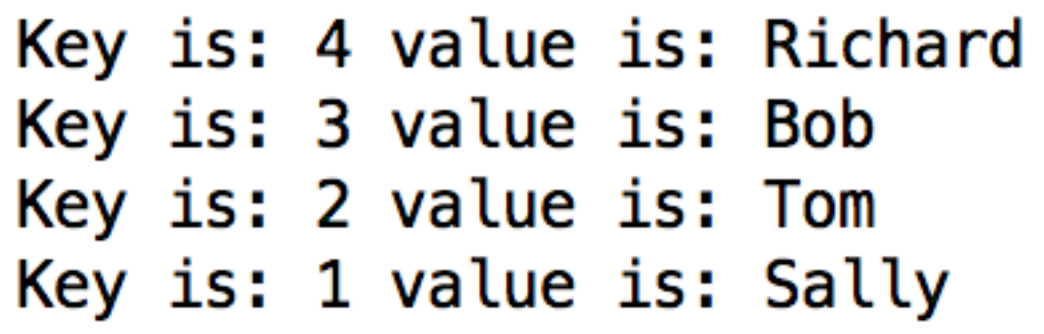
INSTEAD YOU WANT SPECIFY YOUR OWN ORDER ON HOW THE ENTRIES ARE RETRIEVED?

IT'S POSSIBLE!

WE CAN SPECIFY OUR OWN ORDER ON THE KEYS BY USING A COMPARATOR



```java
Map<Integer, String> orderedMap = new TreeMap<>(new Comparator<Integer>() {
    @Override
    public int compare(Integer i1, Integer i2) {
        return i2 - i1;
    }
});

orderedMap.put(3, "Bob");
orderedMap.put(1, "Sally");
orderedMap.put(2, "Tom");
orderedMap.put(4, "Richard");

for (Map.Entry<Integer, String> entry : orderedMap.entrySet()) {
    System.out.println("Key is: " + entry.getKey() + " value is: " + entry.getValue());
}
```

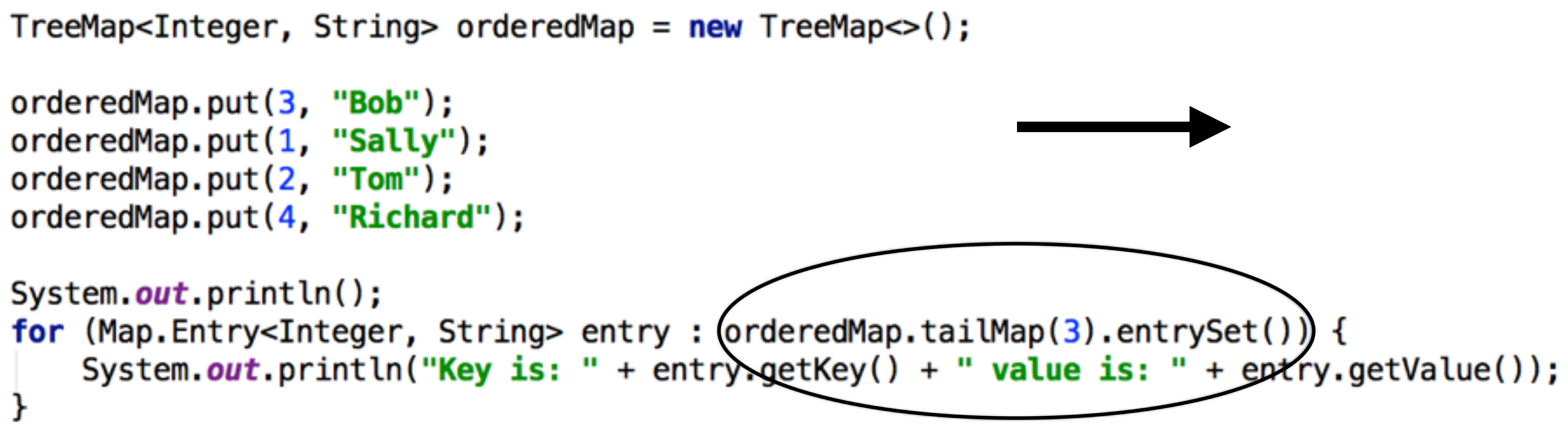


```
Key is: 4 value is: Richard
Key is: 3 value is: Bob
Key is: 2 value is: Tom
Key is: 1 value is: Sally
```

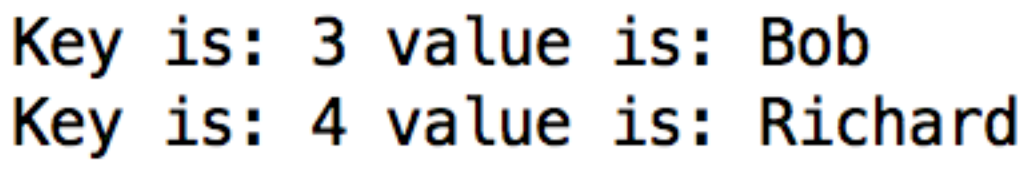
THIS COMPARATOR SPECIFIES A DESCENDING ORDER ON THE KEYS

WHICH MEANS ITERATING THROUGH THE ENTRIES RETRIEVES THEM IN DESCENDING ORDER

TREE MAP HAS SOME OTHER UNIQUE METHODS WHICH ARE NOT AVAILABLE IN A REGULAR MAP



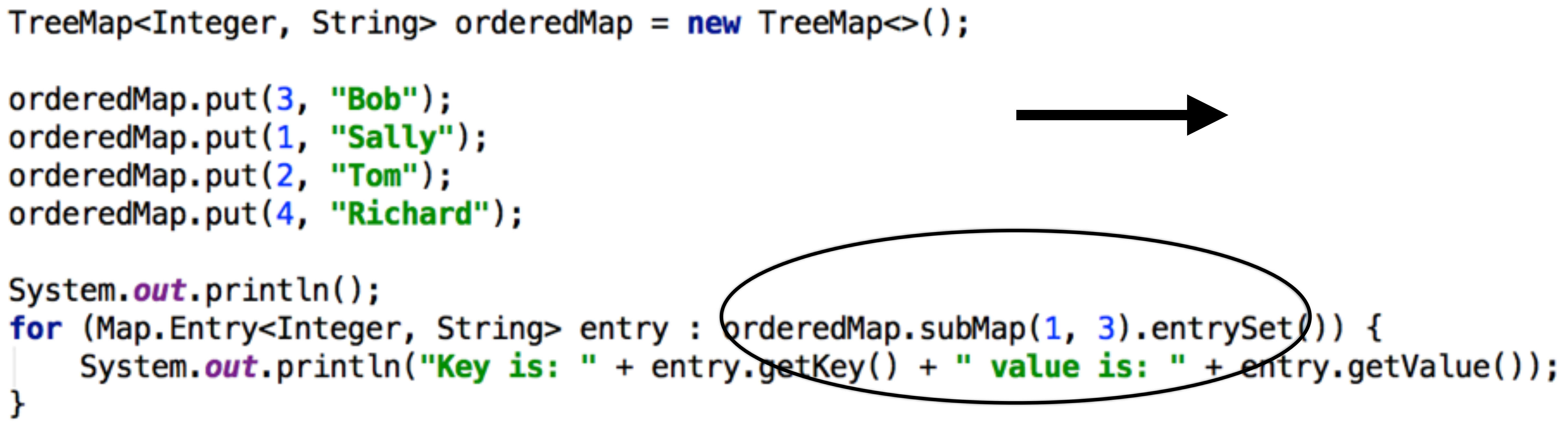
```
TreeMap<Integer, String> orderedMap = new TreeMap<>();

orderedMap.put(3, "Bob");
orderedMap.put(1, "Sally");
orderedMap.put(2, "Tom");
orderedMap.put(4, "Richard");

System.out.println();
for (Map.Entry<Integer, String> entry : orderedMap.tailMap(3).entrySet()) {
    System.out.println("Key is: " + entry.getKey() + " value is: " + entry.getValue());
}
```



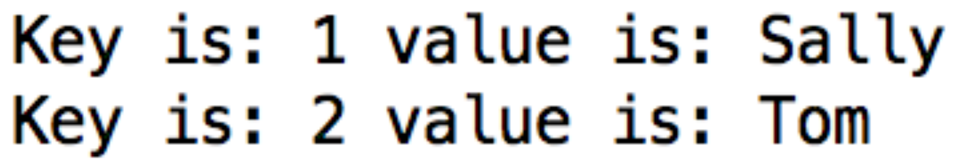
```
Key is: 3 value is: Bob
Key is: 4 value is: Richard
```

THE TAILMAP() RETURNS A VIEW OF THE ORIGINAL MAP WHICH CONTAINS ALL KEYS >= SPECIFIED KEY

HERE IS ANOTHER VIEW OF A TREE MAP



```
TreeMap<Integer, String> orderedMap = new TreeMap<>();

orderedMap.put(3, "Bob");
orderedMap.put(1, "Sally");
orderedMap.put(2, "Tom");
orderedMap.put(4, "Richard");

System.out.println();
for (Map.Entry<Integer, String> entry : orderedMap.subMap(1, 3).entrySet()) {
    System.out.println("Key is: " + entry.getKey() + " value is: " + entry.getValue());
}
```



```
Key is: 1 value is: Sally
Key is: 2 value is: Tom
```

THE SUBMAP() RETURNS A VIEW WHICH INCLUDES KEYS BETWEEN THE START INDEX (INCLUSIVE) AND END INDEX (EXCLUSIVE)