AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

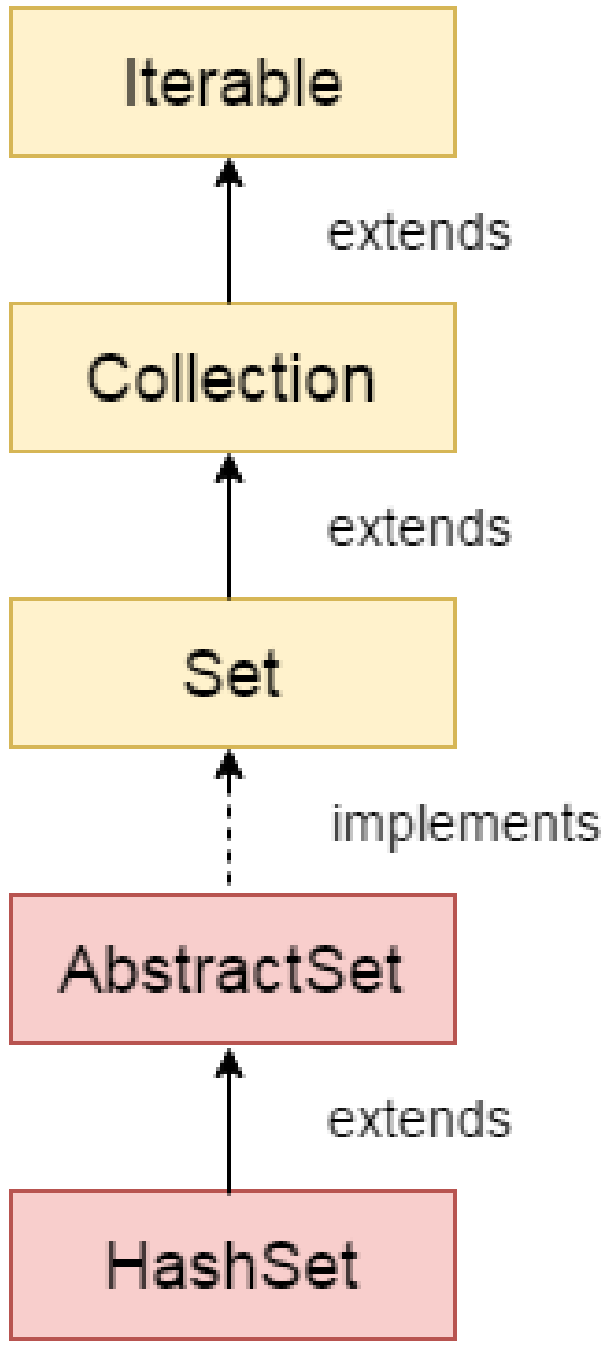
19CSE204
Object Oriented Paradigm
2-0-3-3

Java Collections
Set Interface

- HashSet
- LinkedHashSet
- TreeSet

Java HashSet

- A Set is a Collection that cannot contain duplicate elements. It models the mathematical set abstraction.
- The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
- Set has its implementation in various classes like HashSet, TreeSet, LinkedHashSet



Java HashSet

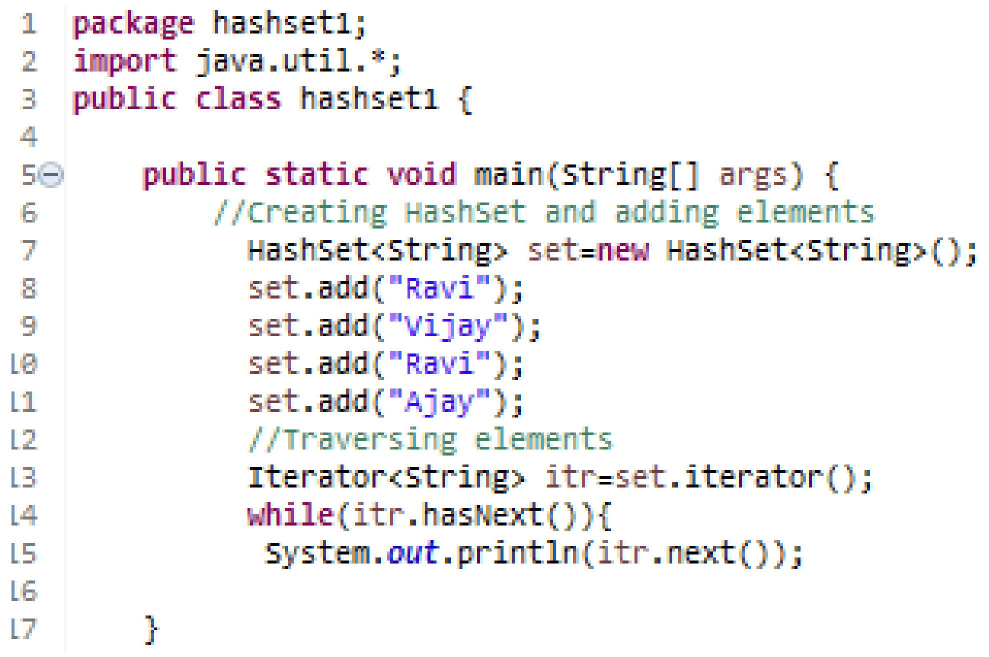
- Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.
  - HashSet stores the elements by using a mechanism called **hashing.**
  - HashSet contains unique elements only.
  - HashSet allows null value.
  - HashSet class is non synchronized.
  - HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
  - HashSet is the best approach for search operations.
  - The initial default capacity of HashSet is 16, and the load factor is 0.75.

A list can contain duplicate elements whereas Set contains unique elements only.

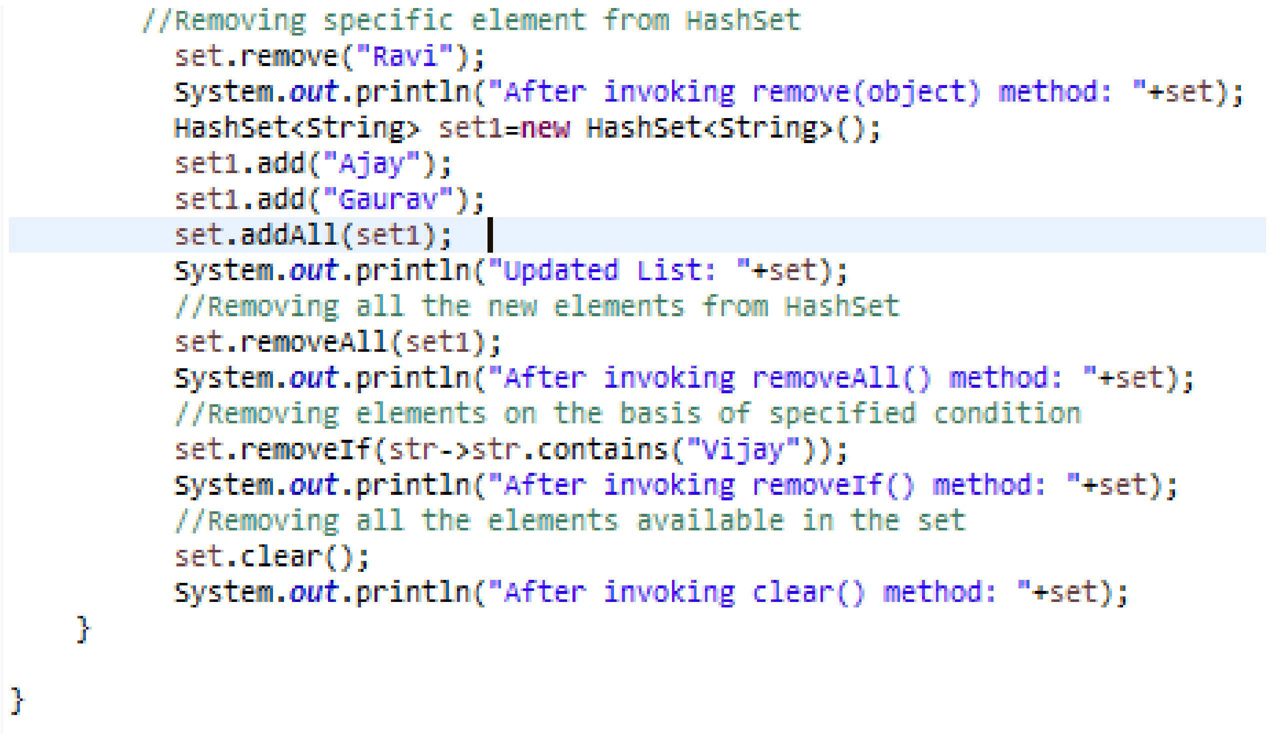
HashMap's capacity will double in **size** by recomputing the hashcodes of the existing data structure elements any time the **HashMap** reaches 75% (in this case 12) of its current **size** (**16**).

HashSet methods

SN	Modifier & Type	Method	Description
1)	boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
2)	void	clear()	It is used to remove all of the elements from the set.
3)	object	clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	contains(Object o)	It is used to return true if this set contains the specified element.
5)	boolean	isEmpty()	It is used to return true if this set contains no elements.
6)	Iterator<E>	iterator()	It is used to return an iterator over the elements in this set.
7)	boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.
8)	int	size()	It is used to return the number of elements in the set.
9)	Spliterator<E>	spliterator()	It is used to create a late-binding and fail-fast Spliterator over the elements in the set.



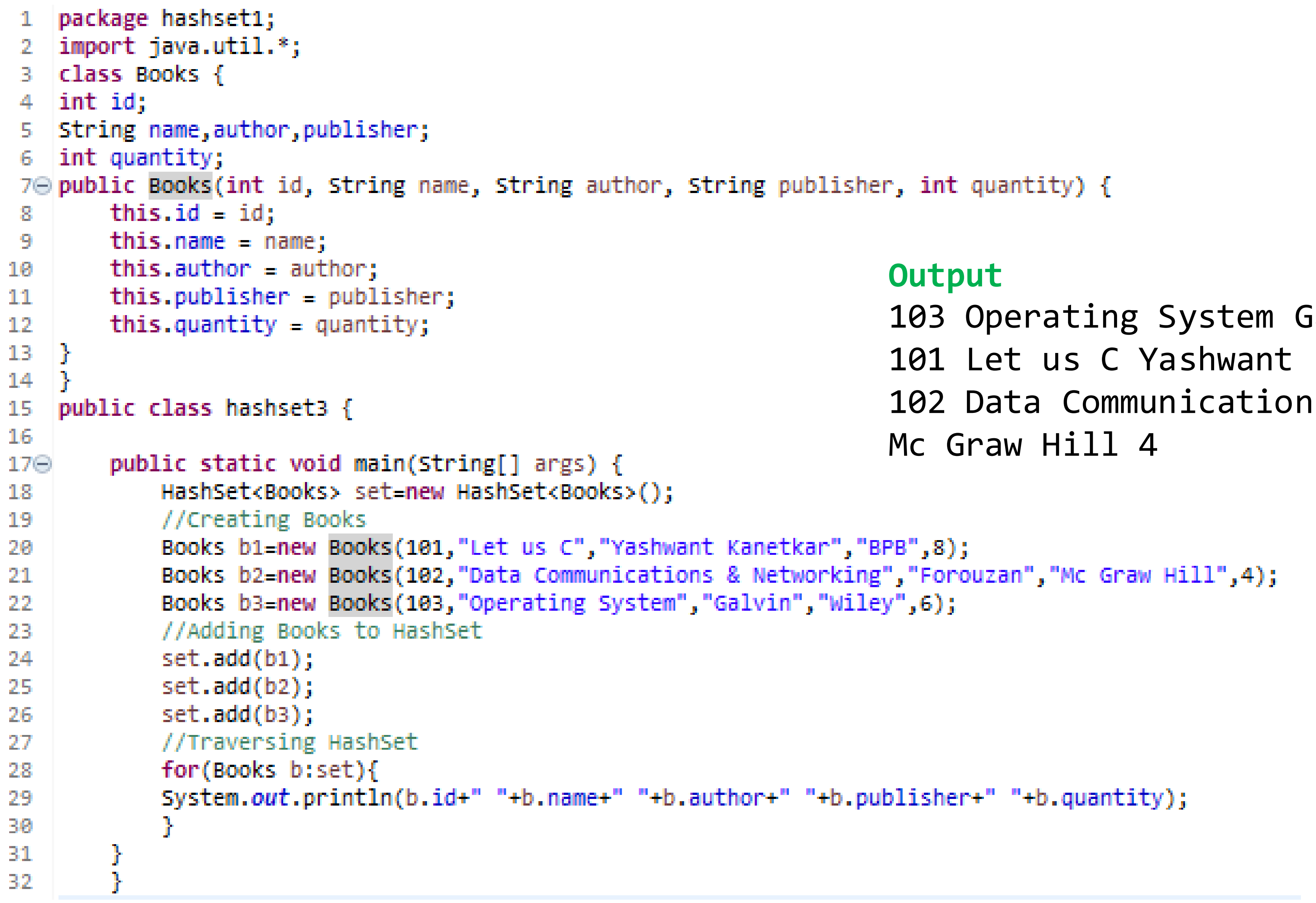
```java
package hashset1;
import java.util.*;
public class hashset1 {

    public static void main(String[] args) {
        //Creating HashSet and adding elements
          HashSet<String> set=new HashSet<String>();
          set.add("Ravi");
          set.add("Vijay");
          set.add("Ravi");
          set.add("Ajay");
          //Traversing elements
          Iterator<String> itr=set.iterator();
          while(itr.hasNext()){
           System.out.println(itr.next());

    }
```



```java
        //Removing specific element from HashSet
          set.remove("Ravi");
          System.out.println("After invoking remove(object) method: "+set);
          HashSet<String> set1=new HashSet<String>();
          set1.add("Ajay");
          set1.add("Gaurav");
          set.addAll(set1);
          System.out.println("Updated List: "+set);
          //Removing all the new elements from HashSet
          set.removeAll(set1);
          System.out.println("After invoking removeAll() method: "+set);
          //Removing elements on the basis of specified condition
          set.removeIf(str->str.contains("Vijay"));
          System.out.println("After invoking removeIf() method: "+set);
          //Removing all the elements available in the set
          set.clear();
          System.out.println("After invoking clear() method: "+set);
    }

}
```

Output

```
Vijay
Ravi
Ajay
After invoking remove(object) method: [Vijay, Ajay]
Updated List: [Vijay, Gaurav, Ajay]
After invoking removeAll() method: [Vijay]
After invoking removeIf() method: []
After invoking clear() method: []
```



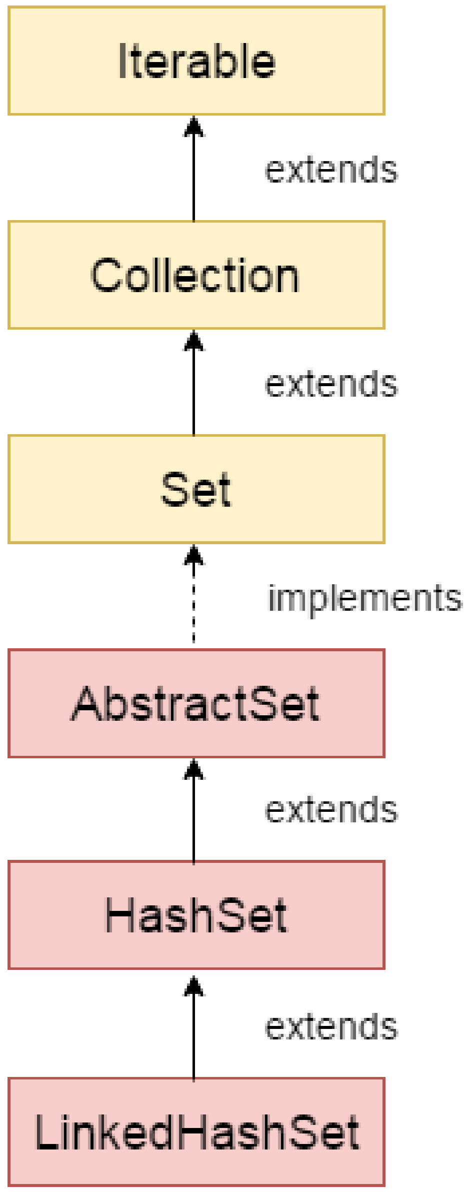
```java
package hashset1;
import java.util.*;
class Books {
int id;
String name,author,publisher;
int quantity;
public Books(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class hashset3 {

    public static void main(String[] args) {
        HashSet<Books> set=new HashSet<Books>();
        //Creating Books
        Books b1=new Books(101,"Let us C","Yashwant Kanetkar","BPB",8);
        Books b2=new Books(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
        Books b3=new Books(103,"Operating System","Galvin","Wiley",6);
        //Adding Books to HashSet
        set.add(b1);
        set.add(b2);
        set.add(b3);
        //Traversing HashSet
        for(Books b:set){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
        }
    }
    }
```

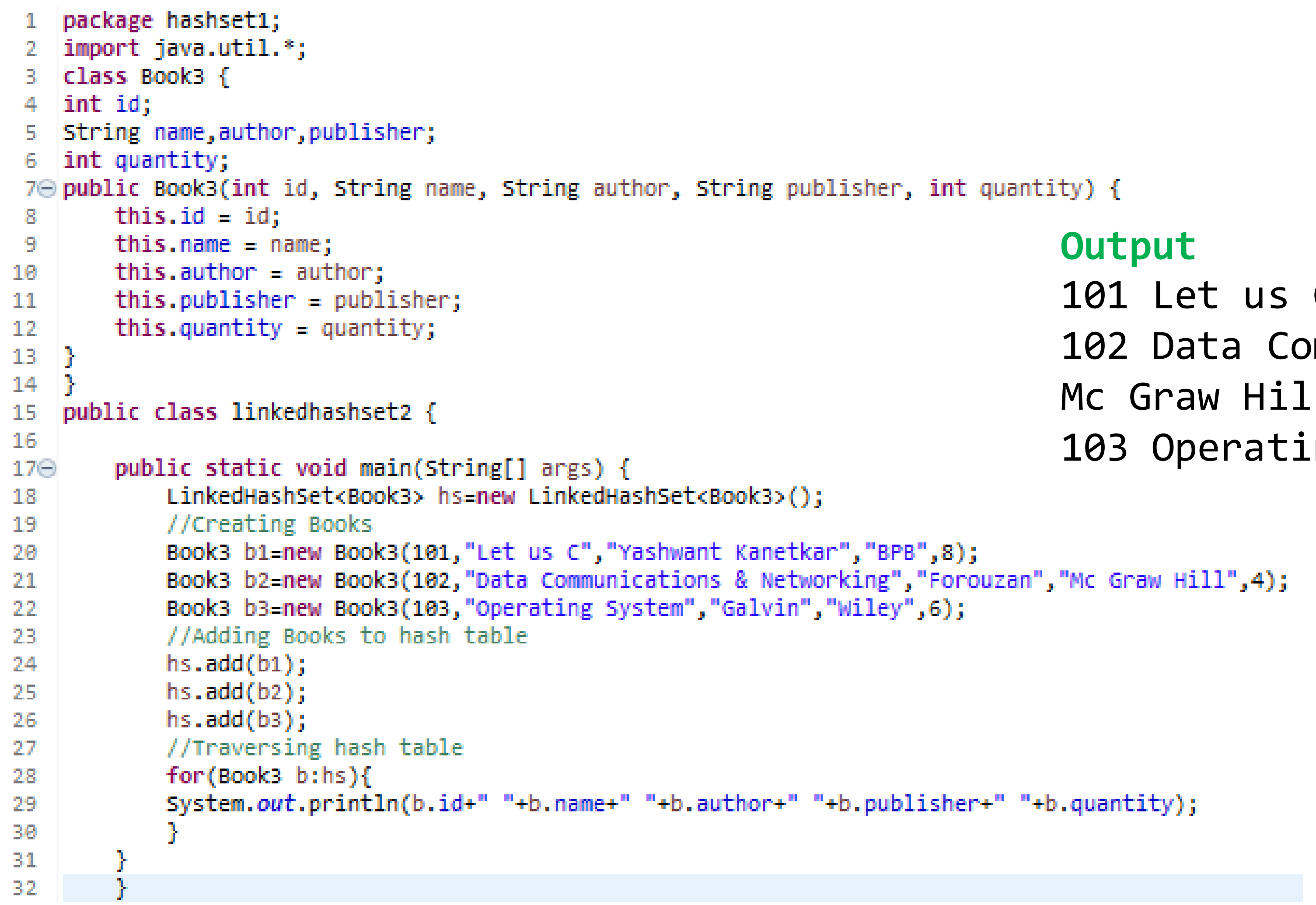
Output

```
103 Operating System Galvin Wiley 6
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
```

LinkedHashSet

- Java LinkedHashSet class is a Hashtable and Linked list implementation of the set interface. It inherits HashSet class and implements Set interface.
  - Java LinkedHashSet class contains unique elements only like HashSet.
  - Java LinkedHashSet class provides all optional set operation and permits null elements.
  - Java LinkedHashSet class is non synchronized.
  - Java LinkedHashSet class maintains insertion order.





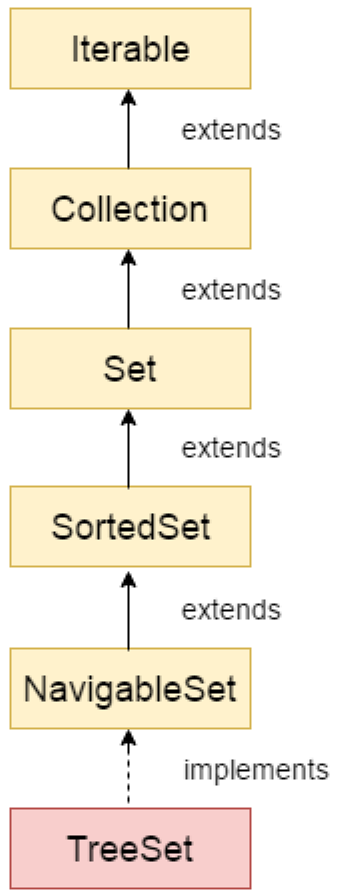
```java
package hashset1;
import java.util.*;
class Book3 {
int id;
String name,author,publisher;
int quantity;
public Book3(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class linkedhashset2 {

    public static void main(String[] args) {
        LinkedHashSet<Book3> hs=new LinkedHashSet<Book3>();
        //Creating Books
        Book3 b1=new Book3(101,"Let us C","Yashwant Kanetkar","BPB",8);
        Book3 b2=new Book3(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
        Book3 b3=new Book3(103,"Operating System","Galvin","Wiley",6);
        //Adding Books to hash table
        hs.add(b1);
        hs.add(b2);
        hs.add(b3);
        //Traversing hash table
        for(Book3 b:hs){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
        }
    }
    }
```

Output

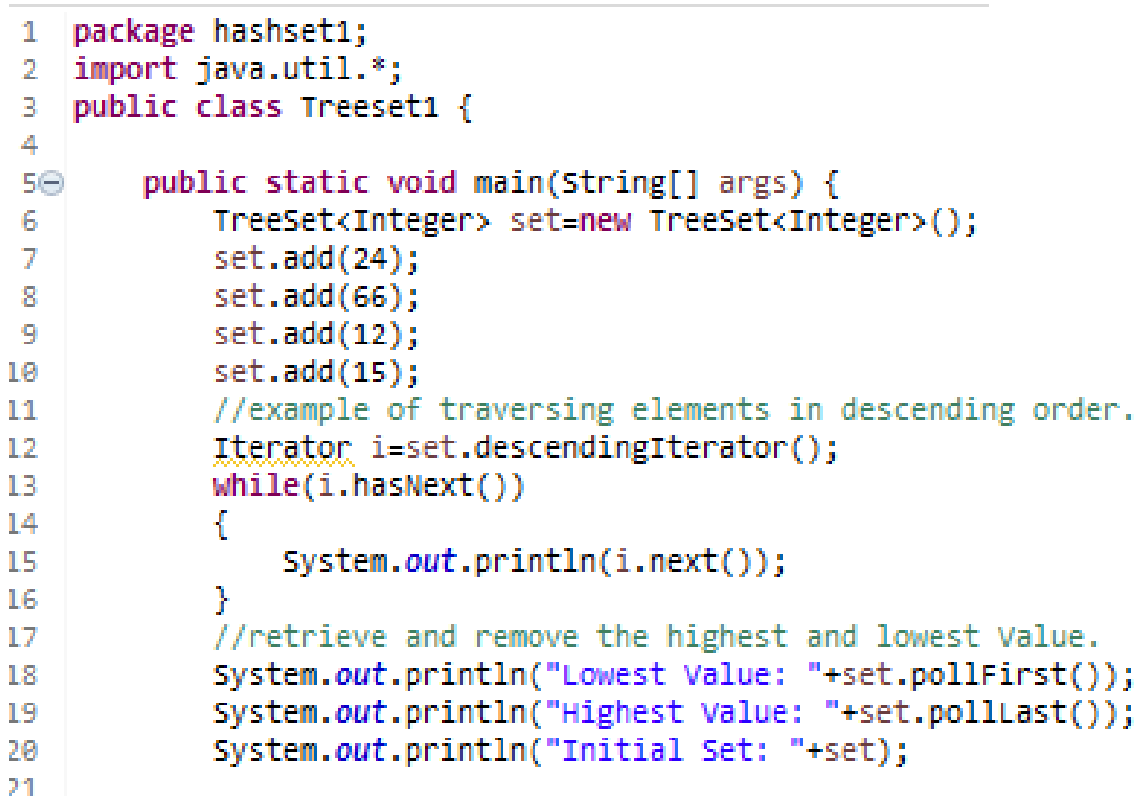
```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan
Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Java TreeSet

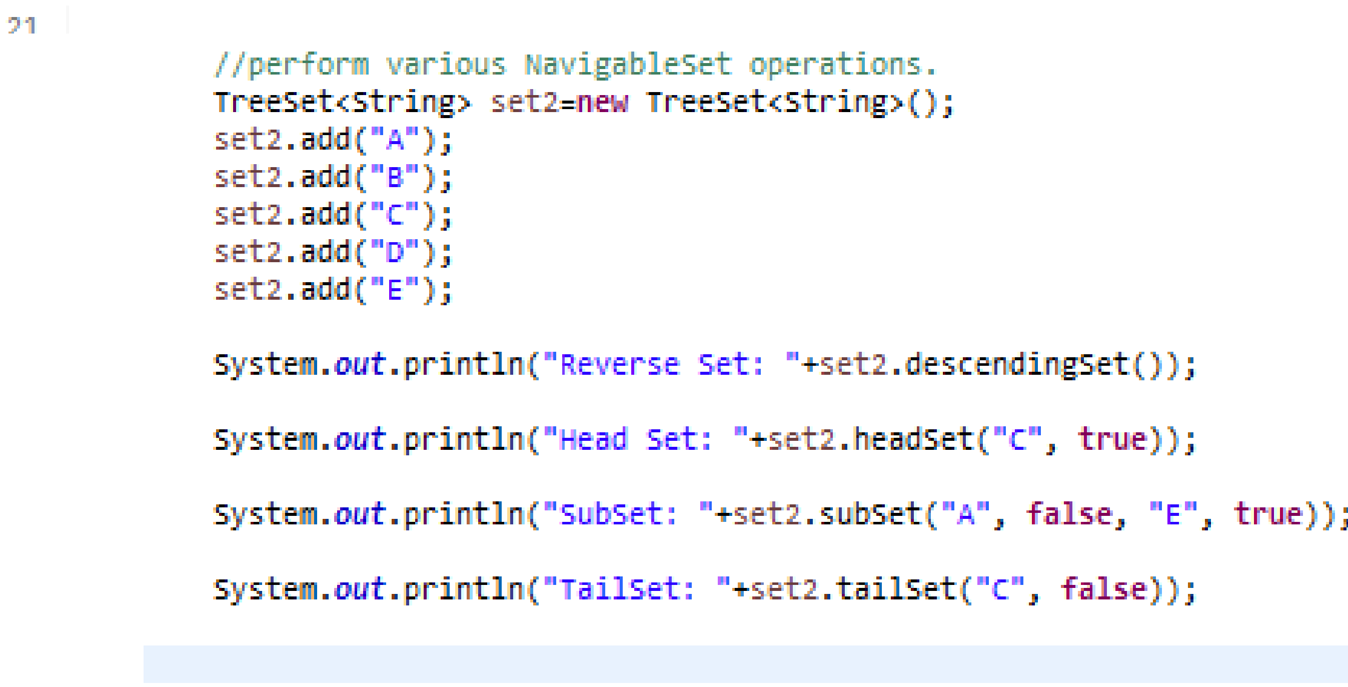
- Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.
  - Java TreeSet class contains unique elements only like HashSet.
  - Java TreeSet class access and retrieval times are quiet fast.
  - Java TreeSet class doesn't allow null element.
  - Java TreeSet class is non synchronized.
  - Java TreeSet class maintains ascending order.



TreeSet example



```java
package hashset1;
import java.util.*;
public class Treeset1 {

    public static void main(String[] args) {
        TreeSet<Integer> set=new TreeSet<Integer>();
        set.add(24);
        set.add(66);
        set.add(12);
        set.add(15);
        //example of traversing elements in descending order.
        Iterator i=set.descendingIterator();
        while(i.hasNext())
        {
            System.out.println(i.next());
        }
        //retrieve and remove the highest and lowest Value.
        System.out.println("Lowest Value: "+set.pollFirst());
        System.out.println("Highest Value: "+set.pollLast());
        System.out.println("Initial Set: "+set);
```



```java
//perform various NavigableSet operations.
TreeSet<String> set2=new TreeSet<String>();
set2.add("A");
set2.add("B");
set2.add("C");
set2.add("D");
set2.add("E");

System.out.println("Reverse Set: "+set2.descendingSet());

System.out.println("Head Set: "+set2.headSet("C", true));

System.out.println("SubSet: "+set2.subSet("A", false, "E", true));

System.out.println("TailSet: "+set2.tailSet("C", false));
```

```
66
24
15
12
Lowest Value: 12
Highest Value: 66
Initial Set: [15, 24]
```

```
Reverse Set: [E, D, C, B, A]
Head Set: [A, B, C]
SubSet: [B, C, D, E]
TailSet: [D, E]
```

The **headset**() method returns a view of the portion of this set whose elements are strictly less than toElement. The **tailSet**() method returns a view of the portion of this set whose elements are greater than or equal to fromElement. The **subset()** method of the TreeSet class returns a view of part of the TreeSet whose elements range from given start and end elements. Boolean is ‘inclusive’

Namah Shivaya