

Object Oriented Programming in Java

Er.Sital Prasad Mandal

BCA- 2nd sem

Mechi Campus

Bhadrapur, Jhapa, Nepal

(Email : info.sitalmandal@gmail.com)

<https://ctaljava.blogspot.com/>



Text Book

1. Deitel & Dietel. -Java: How to-program-. 9th Edition. TearsorrEducation. 2011, ISBN: 9780273759168
2. Herbert Schildt. "Java: The CoriviaeReferi4.ic e 61 Seventh Edition. McGraw -Hill 2006, ISBN; 0072263857



Holding Collection of Data

- 1. Arrays And Collection Classes/Interfaces**
- 2. Map/List/Set Implementations:**
 - i. Map Interface**
 - ii. List Interface**
 - iii. Set Interface**
- 3. Collection Classes:**
 - i. Array List**
 - ii. Linked List**
 - iii. Hash Set**
 - iv. Tree Set**
- 4. Accessing Collections/Use of An Iterator**
- 5. Comparator**

10. Holding Collection of Data



Holding Collection of Data

Initially, collection framework is simply called Java.util package or **Collection API**. Later on, Sun Microsystems had introduced the collection framework in Java 1.2. It was developed and designed by “Joshua Bloch”.

Later on, after Java 1.2, it is known as **collections framework**. From Java 1.5 onwards, The Sun Microsystems added some more new concepts called **Generics**.

10. Holding Collection of Data

Holding Collection of Data

What is Collection in Java

A collection is a group of objects. In Java, these objects are called elements of the collection.

Let's understand it with some realtime examples.

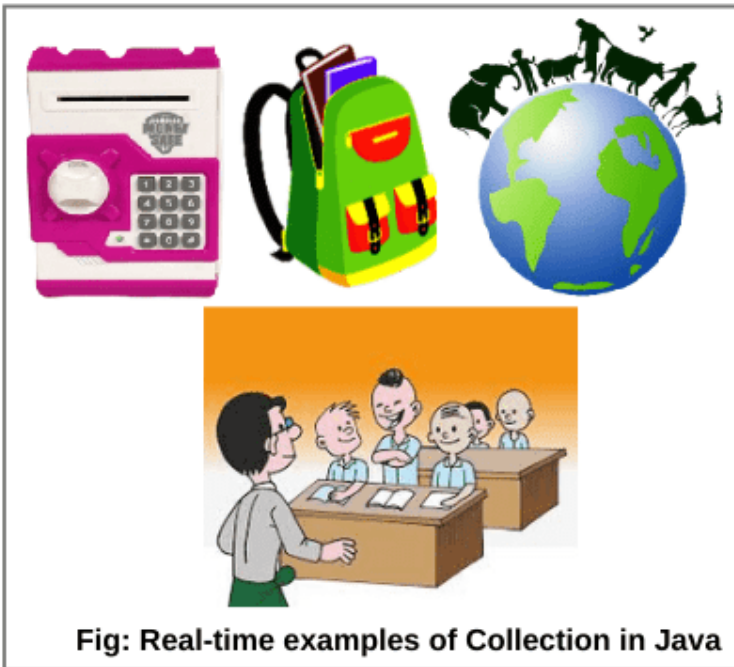


Fig: Real-time examples of Collection in Java

1. This kiddy bank is called collection and the coins are nothing but objects.
2. Here Schoolbag is a collection and books are objects.
3. So, the classroom is nothing but a collection and students are objects.
4. The world is a collection and humans, animals and different things are different objects.

10. Holding Collection of Data

Holding Collection of Data

- Technically, a collection is an object or container which stores a group of other objects as a single unit or single entity. Therefore, it is also known as ***container object or collection object in java.***
- A container object means it contains other objects. In simple words, a *collection is a container that stores multiple elements together.*
- JVM (Java Virtual Machine) stores the reference of other objects into a collection object.

A collection object has a class that is known ***as collection class or container class.*** All collection classes are present in java.util package.

Here, util stands for utility. A group of collection classes is called ***collections framework in java.***

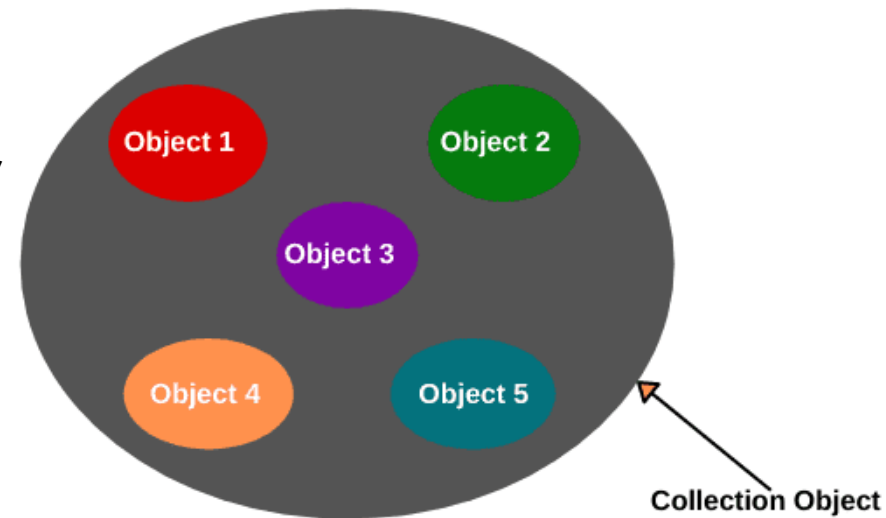


Fig: A group of objects stored in a collection object

10. Holding Collection of Data

Types of Objects Stored in Collection (Container) Object

There are two types of objects that can be stored in a collection or container object.

1. Homogeneous objects (same object):

Homo means same. Homogeneous objects are a group of multiple objects that belong to the same class.

For example, suppose we have created three objects Student s1, Student s2, and Student s3 of the same class 'Student'. Since these three objects belong to the same class that's why they are called homogeneous objects.

2. Heterogeneous objects (different objects):

Hetero means different. Heterogeneous objects are a group of different objects that belong to different classes.

For example, suppose we have created two different objects of different classes such as one object Student s1, and another one object Employee e1. Here, student and employee objects together are called a collection of heterogeneous objects.

Keep in Mind

These objects can also be further divided into two types. They are as follows:

1. Duplicate objects:

```
Person p1 = new Person( "Ram");  
Person p2 = new Person("Ram");
```

2. Unique objects: (different data)

```
Person p1 = new Person("Ram");  
Person p2 = new Person("Hari");
```

What is need for Collections in Java?

1. Using variable approach:

```
int x = 10;  
int y = 20;
```

2. Using class object approach:

```
class Employee {  
    int eNo;  
    String eName;  
}
```

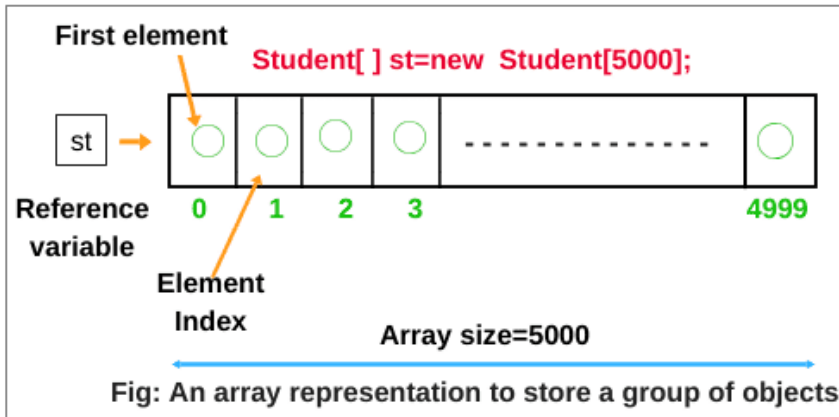
```
// Creating an object of Employee class.  
Employee e1 = new Employee();
```


Keep in Mind

What is need for Collections in Java?

3. Using array object approach:

```
Student[ ] st = new Student[5000];
```



// It will hold only employee type objects.
`Employee[] emp = new Employee[5000];`

For example:

```
emp[0] = new Employee(); // valid.
```

```
// invalid because here, we are providing the customer type object.  
emp[1] = new Customer();
```

```
Object[ ] ob = new Object[5000];  
ob[0] = new Employee(); // valid.  
ob[1] = new Customer(); // valid.
```

4. Using collection object:

By using collection object, we can store the same or different data without any size limitation. Thus, technically, we can define the collections as:

A **collection in java** is a container object that is used for storing multiple homogeneous and heterogeneous, duplicate, and unique elements without any size limitation.

10. Holding Collection of Data



Do U Know... !

Accessor and Mutator methods

The accessor methods are what allow an outside program to get the value of an instance variable. This is why accessor methods are often referred to as "**getter**" methods.

Mutator methods are used to set the value of an instance variable. Therefore, mutators are often referred to as "**setter**" methods.

Q. Does a collection object store copies of other objects?

A: No, a collection object works with reference types. It stores references of other objects, not copies of other objects.

Q. Can we store a primitive data type into a collection?

A: No, collections store only objects.

Q. What is a framework in Java

- It provides readymade architecture.

- It represents a set of classes and interfaces.

- It is optional.

10. Holding Collection of Data



What is Collection framework

A framework in java is a set of several classes and interfaces which provide a ready-made architecture.

A collections framework is a class library to handle groups of objects. It is present in java.util package.

Java Collection Framework enables the user to perform various data manipulation operations like storing data, searching, sorting, insertion, deletion, and updating of data on the group of elements.

Collections framework in Java supports two types of containers:

1. One for storing a collection of elements (objects), that is simply called a collection.
2. The other, for storing key/value pairs, which is called a map.

Collections framework has:

- ❖ Interfaces and its implementations, i.e., classes
- ❖ Algorithm

10. Holding Collection of Data

Advantage Collection framework

- 1.The collections framework reduces the development time and the burden of designers, programmers, and users.
- 2.Your code is easier to maintain because it provides useful data structure and interfaces which reduce programming efforts.
- 3.The size of the container is growable in nature.
- 4.It implements high-performance of useful data structures and algorithms that increase the performance.
- 5.It enables software reuse.

10. Holding Collection of Data



What is Collection framework

List of Interfaces defined in java.util package

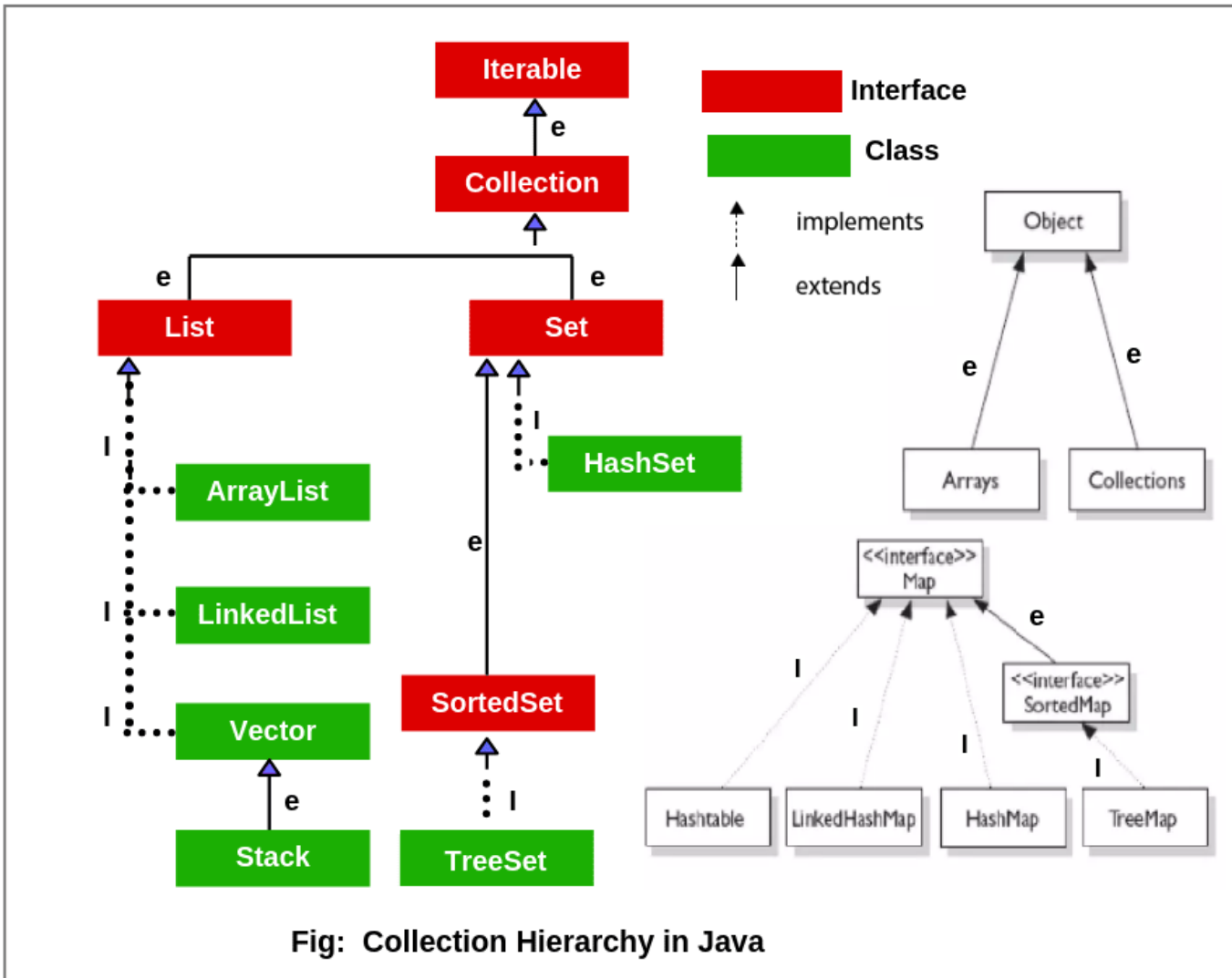
List
Comparator
Map
Set

List of classes defined in java.util package

Array List
Linked List
Hash Set
Tree Set

10. Holding Collection of Data

Java Collection Framework Hierarchy



10. Holding Collection of Data



What is Collection framework

Arrays	Collections
1) Fixed in size.	1) Growable in nature.
2) Memory point of view → Not Recommend	2) Highly recommended to use.
3) Performance point of view → Recommend	3) Not recommended to use.
4) Hold only Homogeneous data type element	4) Hold both Homo & Heterogeneous data types.
5) Hold both primitives & object types.	5) Holds only Objects but not primitives.

10. Holding Collection of Data

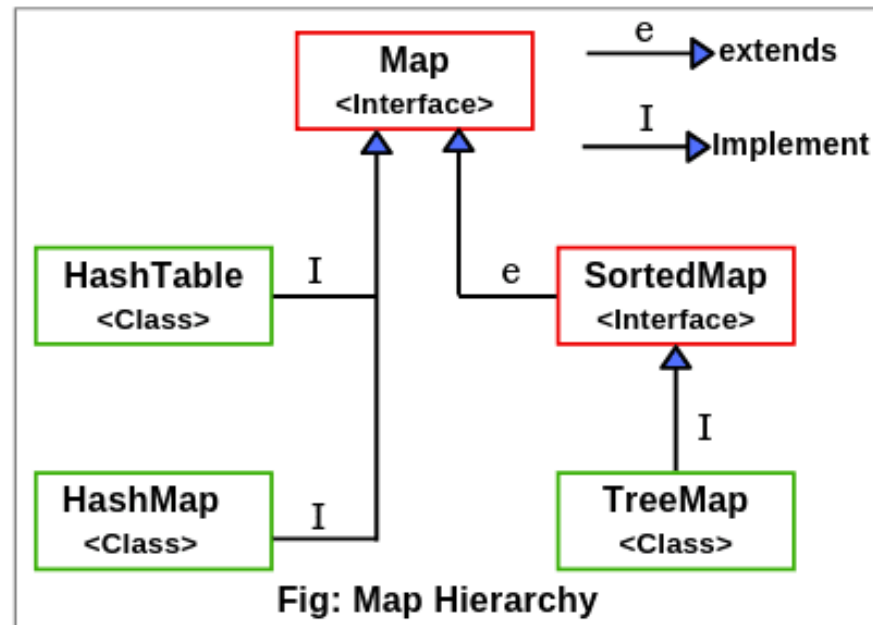
Map Interface

- Map interface is not inherited by the collection interface.
- It represents an object that stores and retrieves elements in the form of a Key/Value pairs and their location within the Map are determined by a Key.
- Map uses a hashing technique for storing key-value pairs.
- It doesn't allow to store the duplicate keys but duplicate values are allowed.
- *HashMap, Hashtable, LinkedHashMap, TreeMap classes implements Map interface.*

● HashMap

● TreeMap

● LinkedHashMap



A Map is an object that maps **keys** to **values**. A map cannot contain duplicate keys.

10. Holding Collection of Data



Map Interface [HashMap]

HashMap:

- A HashMap contains values based on the key.
- It contains only **unique elements**.
- It may have one null key and **multiple null values**.
- It maintains **no order**.

10. Holding Collection of Data

Map Interface [HashMap]

Methods of Java HashMap class

Method	Description
void clear ()	It is used to remove all of the mappings from this map.
boolean containsKey (Object key)	It is used to return true if this map contains a mapping for the specified key.
boolean containsValue (Object value)	It is used to return true if this map maps one or more keys to the specified value.
boolean isEmpty ()	It is used to return true if this map contains no key-value mappings.
Set entrySet ()	It is used to return a collection view of the mappings contained in this map.
Set keySet ()	It is used to return a set view of the keys contained in this map.
Object put (Object key, Object value)	It is used to associate the specified value with the specified key in this map.
int size ()	It is used to return the number of key-value mappings in this map.
Collection values ()	It is used to return a collection view of the values contained in this map.

10. Holding Collection of Data

Map Interface [HashMap]

```
HashMap<Integer, String> map = new HashMap<Integer, String>();
map.put(101, "Let us C");
map.put(102, "Operating System");
map.put(103, "Data Communication and Networking");
System.out.println("Values before remove: " + map);
// Remove value for key 102
map.remove(102);
System.out.println("Values after remove: " + map);
```

```
Map<Integer, Book> map = new HashMap<Integer, Book>();
//Creating Books
Book b1 = new Book(101, "Let us C", "Yashwant Kanetkar", "BPB", 8);
Book b2 = new Book(102, "Data Communications & Networking", "Forouzan", "Mc Graw Hill", 4);
Book b3 = new Book(103, "Operating System", "Galvin", "Wiley", 6);
//Adding Books to map
map.put(1, b1);
map.put(2, b2);
map.put(3, b3);
```

Traversing map

```
for (Map.Entry<Integer, Book> myEntry : map.entrySet()) {
    int key = myEntry.getKey();
    Book b = myEntry.getValue();
    System.out.println(key + " Details:");
    System.out.println(b.id + " " + b.name + " " + b.author + " " + b.publisher + " " + b.quantity);
}
```

10. Holding Collection of Data



Map Interface [TreeMap]

TreeMap is unsynchronized collection class which means it is not suitable for thread-safe operations until unless synchronized explicitly:

- It contains only **unique elements**.
- It **cannot have null key** but **can have multiple null values**.
- It maintains data in **ascending order**.

10. Holding Collection of Data

Map Interface [TreeMap]

Methods of Java TreeMap class

Method	Description
boolean containsKey (Object key)	It is used to return true if this map contains a mapping for the specified key.
boolean containsValue (Object value)	It is used to return true if this map maps one or more keys to the specified value.
Object firstKey ()	It is used to return the first (lowest) key currently in this sorted map.
Object get (Object key)	It is used to return the value to which this map maps the specified key.
Object lastKey ()	It is used to return the last (highest) key currently in this sorted map.
Object remove (Object key)	It is used to remove the mapping for this key from this TreeMap if present.
void putAll (Map map)	It is used to copy all of the mappings from the specified map to this map.
Set entrySet ()	It is used to return a set view of the mappings contained in this map.
int size ()	It is used to return the number of key-value mappings in this map.
Collection values ()	It is used to return a collection view of the values contained in this map.

10. Holding Collection of Data

Map Interface [TreeMap]

```
Map<Integer, String> map = new TreeMap<Integer, String>();
map.put(102, "Let us C");
map.put(103, "Operating System");
map.put(101, "Data Communication and Networking");
System.out.println("Values before remove: "+ map);
// Remove value for key 102
map.remove(102);
System.out.println("Values after remove: "+ map);
```

What is difference between HashMap and TreeMap?

HashMap	TreeMap
1) HashMap can contain one null key.	TreeMap can not contain any null key.
2) HashMap maintains no order.	TreeMap maintains ascending order.

10. Holding Collection of Data



Map Interface [LinkedHashMap]

Java LinkedHashMap class is Hash table and Linked list implementation of the Map interface, with predictable iteration order.

- A LinkedHashMap contains **values based on the key**.
- It contains only **unique elements**.
- It may have **one null key** and **multiple null values**.
- It maintains data in **insertion order**.

10. Holding Collection of Data

Map Interface [LinkedHashMap]

//Creating map of Books

```
Map<Integer,Book> map=new LinkedHashMap<Integer,Book>();
```

//Creating Books

```
Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
```

```
Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
```

```
Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
```

//Adding Books to map

```
map.put(2,b2);
```

```
map.put(1,b1);
```

```
map.put(3,b3);
```

//Traversing map

```
for(Map.Entry<Integer, Book> entry:map.entrySet()){
```

```
    int key=entry.getKey();
```

```
    Book b=entry.getValue();
```

```
    System.out.println(key+" Details:");
```

```
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
```

```
}
```


10. Holding Collection of Data

List Interface

- This interface represents a collection of elements whose elements are arranged sequentially ordered.
- List maintains an order of elements means the order is preserved in which we add elements, and the same sequence we will get while retrieving elements.
- We can insert elements into the list at any location. The list allows storing duplicate elements in Java.
- [ArrayList](#), [vector](#), and [LinkedList](#) are three concrete subclasses that implement the list interface.

10. Holding Collection of Data

List Interface

```
import java.util.*;
public class ListInterface {
    public static void main(String args[]) {
        List list = new ArrayList();
        list.add("David");
        list.add("Jhon");
        list.add("Yam");

        System.out.println(list);
        list.remove(2);
        System.out.println(list.size()); //2
    }
}
```

10. Holding Collection of Data



Set Interface

- Set is a child interface of Collection.
- Insertion order not preserved i.e., They appear in the different order in which we inserted.
- Duplicate elements are not allowed.
- Heterogeneous objects are allowed.
- Set Interface is implemented by using LinkedHashSet and HashSet class.

The set interface is inherited from the Java collections Interface. A Set interface cannot store duplicate/redundant elements in it.

10. Holding Collection of Data



Set Interface

- This interface represents a collection of elements that contains unique elements. i.e, It is used to store the collection of unique elements.
- Set interface does not maintain any order while storing elements and while retrieving, we may not get the same order as we put elements. All the elements in a set can be in any order.
- Set does not allow any duplicate elements.
- HashSet, LinkedHashSet, TreeSet classes implements the set interface and sorted interface extends a set interface.

10. Holding Collection of Data



Set Interface

```
public class SetInterface {  
  
    public static void main ( String[] args ) {  
  
        int[] count = {21, 23, 43, 53, 22, 65};  
  
        Set<Integer> set = new HashSet<Integer>();  
        try {  
            for (int i = 0; i <= 5; i++) {  
                set.add(count[i]);  
            }  
  
            System.out.println(set);  
  
            TreeSet<Integer> sortedSet = new TreeSet<Integer>(set);  
  
            System.out.println("The sorted list is:");  
  
            System.out.println(sortedSet);  
  
            System.out.println("First element of the set is: " + sortedSet.first());  
  
            System.out.println("last element of the set is: " + sortedSet.last());  
  
        } catch (Exception e) {  
        }  
    }  
}
```

10. Holding Collection of Data

ArrayList Class

ArrayList is a **resizable-array** implementation of the **List** interface. It implements all optional list operations, and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list.

ArrayList [ArrayList class implements List interface] →

- ✓ Supports dynamic array that can grow dynamically.
- ✓ Can contain duplicate elements.
- ✓ Heterogeneous objects are allowed. (Except TreeSet & TreeMap)
- ✓ Insertion order preserved & Provides more powerful insertion and search mechanisms than arrays.
- ✓ Null insertion is possible.

10. Holding Collection of Data

ArrayList Class

Example1:

```
ArrayList<String> books = new ArrayList<String>();  
books.add("Java Book1");  
books.add("Java Book2");  
books.add("Java Book3");  
System.out.println("Books stored in array list are:"+books);
```

Example2:

```
ArrayList<String> cities = new ArrayList<String>(){  
    {  
        add(" Ktm ");  
        add(" Bhd ");  
        add(" Btm ");  
    }  
};  
System.out.println("Content of Array list cities:"+cities);
```

Example3:

```
ArrayList<String> obj = new ArrayList<String>(  
    Arrays.asList("Ram ", "Peter", "Hari"));  
  
System.out.println("Elements are:"+obj);
```

10. Holding Collection of Data

Linked List Class

- ✓ Usually used to Implement List and also the Stack(LIFO) & Queue(FIFO).
- ✓ Frequent Insertion / deletion in Middle then LinkedList - Best choice.
- ✓ Insertion order is Preserved.
- ✓ Null Insertion – Possible.
- ✓ Frequent Retrieval operation then LinkedList - Worst choice.
- ✓ Allow Duplicate objects.
- ✓ Allow Heterogeneous objects.
- ✓ Implements Serializable and Cloneable interfaces but not RandomAccess.

Constructors:

- ✓ Creates an empty LinkedList object.

```
LinkedList l=new LinkedList();
```

LinkedList IMPORTANT METHODS:

- ✓ void addFirst(Object x)
- ✓ void addLast(Object x)
- ✓ Object getFirst()
- ✓ Object getLast()
- ✓ Object removeFirst()
- ✓ Object removeLast()

[LinkedList class](#) is an implementation of a list interfaces. *Linked List is similar to an array, but it does not store data in sequential data addresses, but connects the memory blocks in the sequential order and allows null elements.*

10. Holding Collection of Data

Linked List Class

```
import java.util.LinkedList;
public class LinkedListExample {
    public static void main ( String[] args ) {
        LinkedList lst = new LinkedList();
        lst.add("ram");
        lst.add("hari");
        lst.add("Rima");
        System.out.println(lst);
        System.out.println(lst.size());
        System.out.println(lst.remove(1));
        System.out.println(lst.size());
        lst.addFirst("Gita");
        lst.addLast("Ramesh");
        System.out.println(lst.getFirst());
        System.out.println(lst.getLast());
        System.out.println(lst);
        System.out.println(lst.removeFirst());
    }
}
```

[ram, hari, Rima]

3

hari

2

Gita

Ramesh

[Gita, ram, Rima, Ramesh]

Gita

10. Holding Collection of Data



Hash Set

Java HashSet is used to create a collection that uses a **hash table** for storage.

The important points about Java HashSet class are:

- 1) **HashSet doesn't maintain any order**, the elements would be returned in any random order.
- 2) **HashSet doesn't allow duplicates**. If you try to add a duplicate element in HashSet, the old value would be overwritten.
- 3) HashSet allows **null values**.
- 4) HashSet is **non-synchronized**.

10. Holding Collection of Data

Hash Set

Methods of Java HashSet class:

Method	Description
void clear ()	It is used to remove all of the elements from this set.
boolean contains (Object o)	It is used to return true if this set contains the specified element.
boolean add (Object o)	It is used to adds the specified element to this set if it is not already present.
boolean isEmpty ()	It is used to return true if this set contains no elements.
boolean remove (Object o)	It is used to remove the specified element from this set if it is present.
Object clone ()	It is used to return a shallow copy of this HashSet instance: the elements then
Iterator iterator ()	It is used to return an iterator over the elements in this set.
int size ()	It is used to return the number of elements in this set.

10. Holding Collection of Data

Hash Set

//HashSet Class

```
import java.util.HashSet;
public class HashSetClass {
    public static void main ( String[] args ) {
        HashSet<String> hset = new HashSet<String>();
        hset.add("Suzuki");
        hset.add("Kawasaki");
        hset.add("Honda");
        hset.add("Ducati");
        hset.add("Yamaha");
        hset.add("Yamaha");
        hset.add("Suzuki");
        hset.add(null);
        hset.add(null);
        // Displaying HashSet elements
        System.out.println(hset);
        System.out.println(hset.size());
    }
}
```

[null, Suzuki, Ducati, Yamaha, Kawasaki, Honda]

10. Holding Collection of Data

Hash Set

Example 1:

```
HashSet<String> hset = new HashSet<String>();

// Adding elements to the HashSet
hset.add("Apple");
hset.add("Orange");
hset.add("Fig");

//Addition of duplicate elements
hset.add("Apple");
hset.add("Mango");

//Addition of null values
hset.add(null);

//Displaying HashSet elements
System.out.println(hset);
```

10. Holding Collection of Data

Hash Set

Example 2:

```
public class Book {
    int id, quantity;
    String name, author, publisher;
    public Book(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;
    }
}

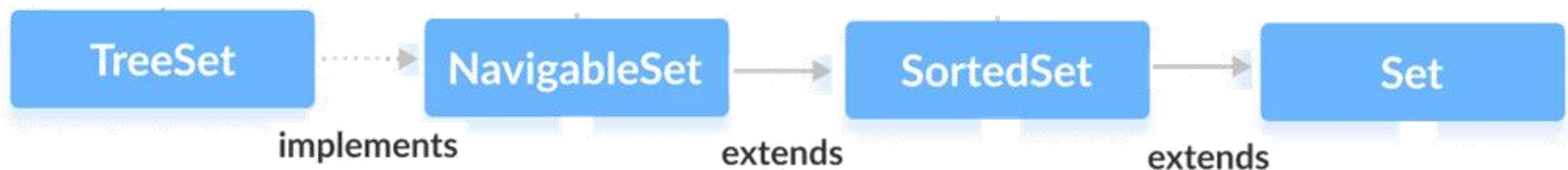
public class HashSetExample {
    public static void main(String[] args) {

        HashSet<Book> hset=new HashSet<Book>();
        //Creating Books
        Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
        Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
        Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
        //Adding Books to HashSet
        hset.add(b1);
        hset.add(b2);
        hset.add(b3);
    }
}
```

10. Holding Collection of Data

Tree Set

- TreeSet is similar to HashSet except that it sorts the elements in the **ascending order** while HashSet doesn't maintain any order.
- TreeSet allows null element but like HashSet it doesn't allow.
- TreeSet class implements the SortedSet interface. It doesn't allow duplicate elements.
- TreeSet class is not synchronized.
- TreeSet does not preserve the insertion order but the elements in TreeSet are sorted as per the natural ordering.
- TreeSet can be ordered by using a custom comparator while creating a TreeSet object.
- TreeSet is normally used for storing huge amounts of information that is naturally sorted. This aids in easy and faster access.

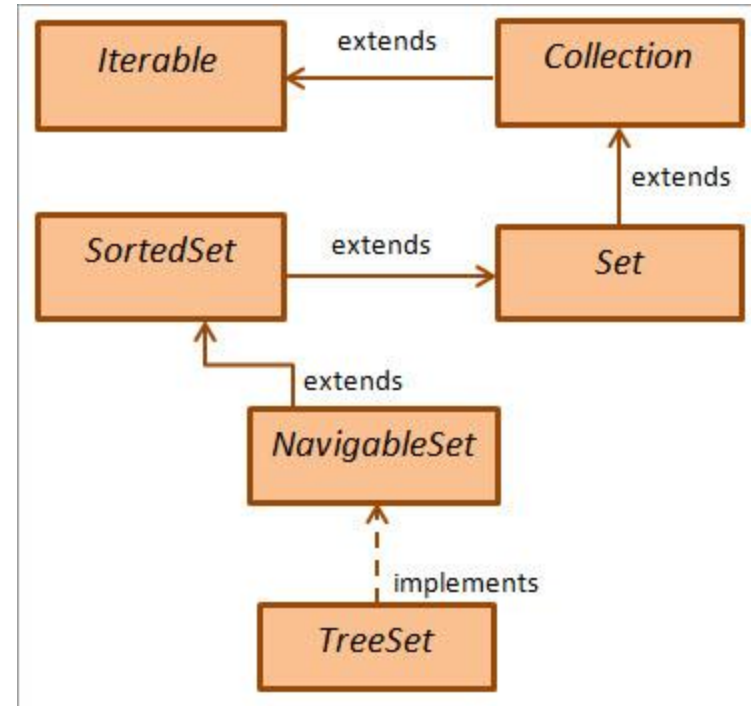


10. Holding Collection of Data

Tree Set

```
import java.util.Set;
import java.util.TreeSet;
public class TreeSetClass {
    public static void main(String args[]) {
        //Set treeset = new TreeSet();
        TreeSet treeset = new TreeSet();
        treeset.add(8476);
        treeset.add(748);
        treeset.add(88);
        treeset.add(983);
        treeset.add(18);
        treeset.add(0);
        System.out.println(treeset);
    }
}
```

[0, 18, 88, 748, 983, 8476]



Class hierarchy for TreeSet class

10. Holding Collection of Data



Tree Set

The objects of TreeSet class are stored in ascending order.

Example:

```
LinkedHashSet<Book> hs=new LinkedHashSet<Book>();  
//Creating Books  
Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);  
Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);  
Book b3=new Book(103,"Operating System","Galvin","Wiley",6);  
//Adding Books to hash table  
hs.add(b1);  
hs.add(b2);  
hs.add(b3);
```

10. Holding Collection of Data

Accessing Collections

Collection Interface Methods → [No concrete class to implement interface directly]

1. `int size();` - Return no. of elements in collection.
2. `boolean add(Object ele);` - Add ele to invoking collection.
3. `boolean contains(Object ele);` - Return true if element is ele of invoking collection.
4. `Iterator iterator();` - Returns iterator from invoking collection.
5. `boolean remove(Object ele);` - Remove one instance of element from invoking collection.
6. `boolean addAll(Collection c);` - Add all element of c to invoking collection.
7. `boolean containsAll(Collection c);` - Return true if invoking collection contains all elements of c, else false.
8. `boolean removeAll(Collection c);` - Remove all element of c from invoking collection.
9. `boolean retainAll(Collection c);` - Remove all objects/ele except those present in c.
10. `void clear();` - Remove all ele from invoking collection.
11. `boolean isEmpty();` - Returns true if invoking collection is empty.
12. `Object[] toArray();` - Return array that contains all ele stored in invoking collection.

10. Holding Collection of Data

Accessing Collections Use of An Iterator

The java collection framework often we want to ***cycle through the elements***. For example, we might want to display each element of a collection. The java provides an interface **Iterator** that is available inside the **java.util** package to cycle through each element of a collection.

- The **Iterator** allows us to move only forward direction.
- The **Iterator** does not support the replacement and addition of new elements.

We use the following steps to access a collection of elements using the Iterator.

Step-1: Create an object of the Iterator by calling **collection.iterator()** method.

Step-2: Use the method **hasNext()** to access to check does the collection has the next element. (Use a loop).

Step-3: Use the method **next()** to access each element from the collection. (use inside the loop).

Method	Description
Iterator iterator()	Used to obtain an iterator to the start of the collection.
boolean hasNext()	Returns true if the collection has the next element, otherwise, it returns false.
E next()	Returns the next element available in the collection.

10. Holding Collection of Data

Accessing Collections Use of An Iterator

Iterator - [Legacy – No]

- ✓ Returns an iterator to a collection.
- ✓ It's an object that enables to traverse through collection.
- ✓ Can be used to remove elements from collection selectively.
- ✓ **iterator()** method of Collection interface to create Iterator object.

```
public Iterator iterator();  
Iterator itr = c.iterator();
```

Enumeration() Methods →

1. **hasMoreElements()**
2. **nextElement()**
3. (Not Available)

Iterator Methods →

1. **hasNext()** - Returns true if more ele exist.
2. **next()** - Returns next element.
3. **remove()** - Removes current element.

Advantage of Iterator over for-each(Also used for iterating) method →

- ✓ The for-each construct hides the iterator, so you cannot call remove
- ✓ Iterate over multiple collections in parallel.

```
for(Object o : oa) {           // for-each construct  
    Fruit d2= (Fruit)o;  
    System.out.println(d2.name);  
}
```

10. Holding Collection of Data

Accessing Collections Use of An Iterator

```
TreeSetExample.java
1 import java.util.*;
2
3 public class TreeSetExample {
4
5     public static void main(String[] args) {
6
7         TreeSet set = new TreeSet();
8
9         Random num = new Random();
10        for(int i = 0; i < 10; i++)
11            set.add(num.nextInt(100));
12
13        Iterator collection = set.iterator();
14
15        System.out.println("All the elements of TreeSet collection:");
16        while(collection.hasNext())
17            System.out.print(collection.next() + ", ");
18
19    }
20 }
```

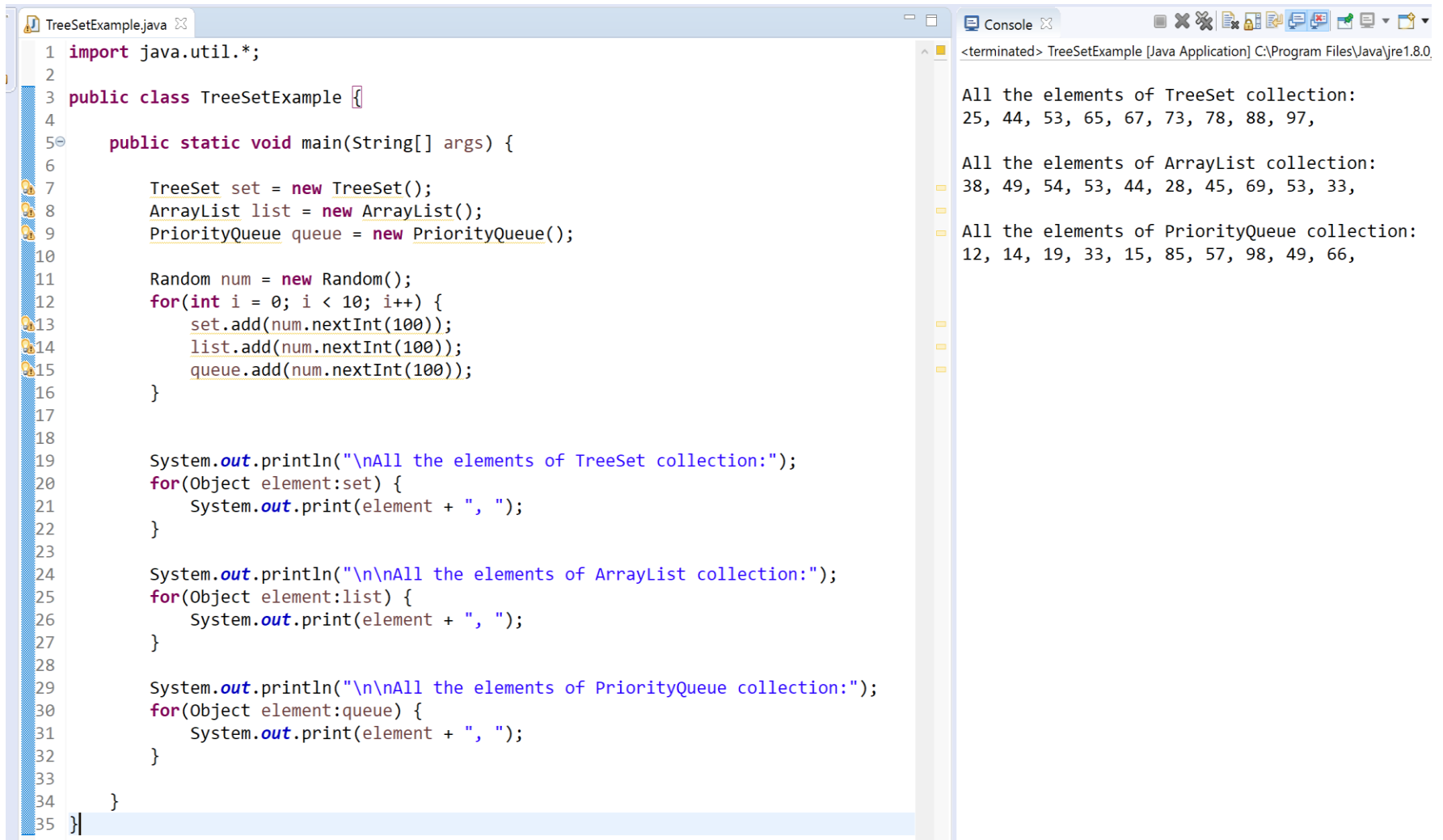
Console
<terminated> TreeSetExample [Java Application] C:\Program Files\Java

All the elements of TreeSet collection:
2, 12, 32, 36, 64, 65, 78, 82, 90,

10. Holding Collection of Data

Accessing Collections Use of An Iterator

Accessing a collection using for-each



The screenshot displays a Java IDE with two panes. The left pane shows the source code for `TreeSetExample.java`, and the right pane shows the console output.

Source Code (TreeSetExample.java):

```
1 import java.util.*;
2
3 public class TreeSetExample {
4
5     public static void main(String[] args) {
6
7         TreeSet set = new TreeSet();
8         ArrayList list = new ArrayList();
9         PriorityQueue queue = new PriorityQueue();
10
11         Random num = new Random();
12         for(int i = 0; i < 10; i++) {
13             set.add(num.nextInt(100));
14             list.add(num.nextInt(100));
15             queue.add(num.nextInt(100));
16         }
17
18         System.out.println("\nAll the elements of TreeSet collection:");
19         for(Object element:set) {
20             System.out.print(element + ", ");
21         }
22
23         System.out.println("\n\nAll the elements of ArrayList collection:");
24         for(Object element:list) {
25             System.out.print(element + ", ");
26         }
27
28         System.out.println("\n\nAll the elements of PriorityQueue collection:");
29         for(Object element:queue) {
30             System.out.print(element + ", ");
31         }
32     }
33 }
34
35 }
```

Console Output:

```
<terminated> TreeSetExample [Java Application] C:\Program Files\Java\jre1.8.0_
All the elements of TreeSet collection:
25, 44, 53, 65, 67, 73, 78, 88, 97,

All the elements of ArrayList collection:
38, 49, 54, 53, 44, 28, 45, 69, 53, 33,

All the elements of PriorityQueue collection:
12, 14, 19, 33, 15, 85, 57, 98, 49, 66,
```

10. Holding Collection of Data



Accessing Collections Use of An Iterator

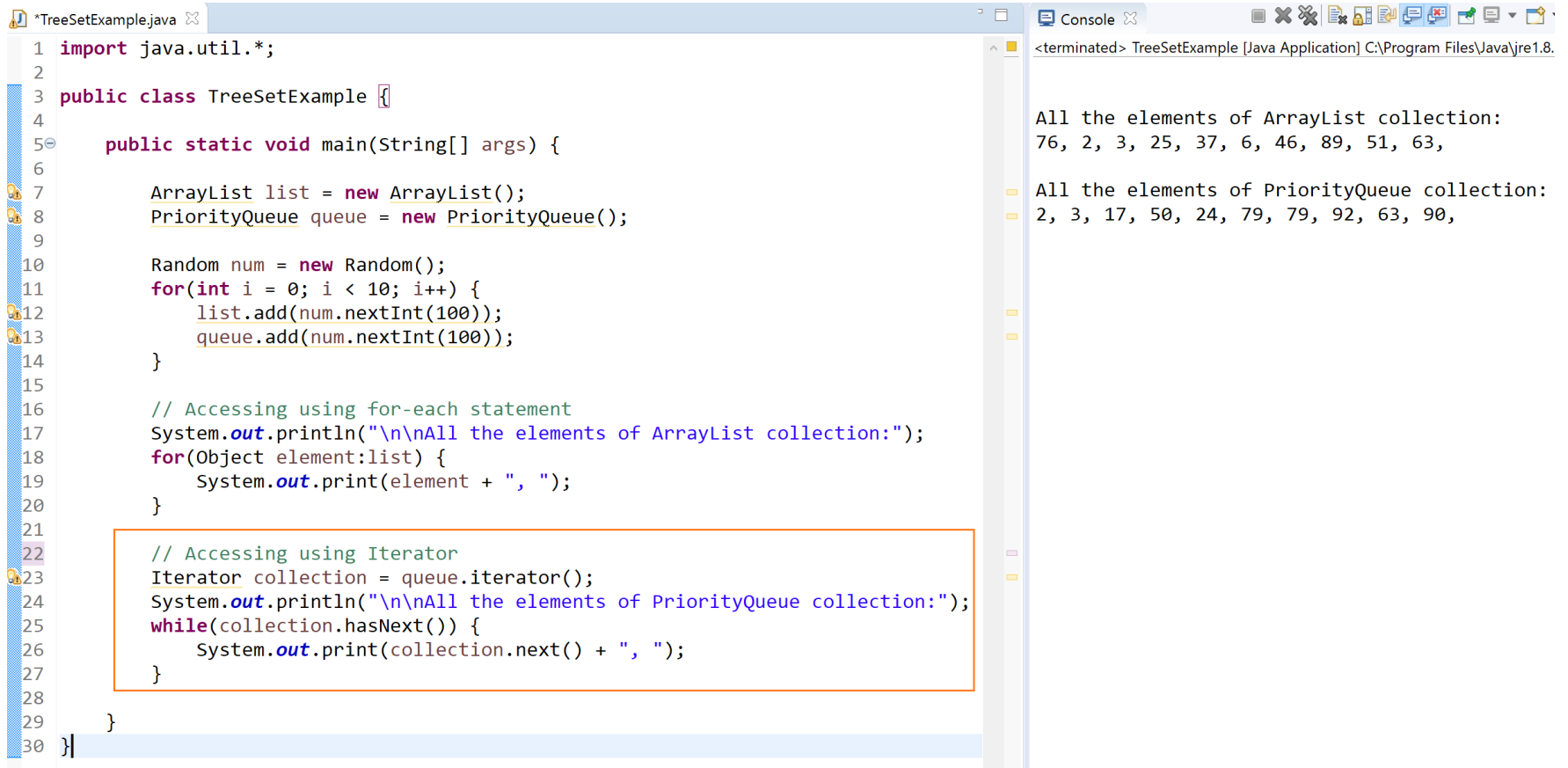
The For-Each alternative

Using for-each, we can access the elements of a collection. But for-each can only be used if we don't want to modify the contents of a collection, and we don't want any reverse access.

Alternatively, we can use the ***Iterator*** to access or cycle through a collection of elements.

10. Holding Collection of Data

Accessing Collections Use of An Iterator



The screenshot displays an IDE with a Java file named `*TreeSetExample.java` and a console window. The code in the editor defines a `TreeSetExample` class with a `main` method. It initializes an `ArrayList` and a `PriorityQueue`, populates them with random numbers, and then prints their contents. The `ArrayList` is printed using a `for`-each loop, and the `PriorityQueue` is printed using an `Iterator`. The console output shows the elements of both collections.

```
1 import java.util.*;
2
3 public class TreeSetExample {
4
5     public static void main(String[] args) {
6
7         ArrayList list = new ArrayList();
8         PriorityQueue queue = new PriorityQueue();
9
10        Random num = new Random();
11        for(int i = 0; i < 10; i++) {
12            list.add(num.nextInt(100));
13            queue.add(num.nextInt(100));
14        }
15
16        // Accessing using for-each statement
17        System.out.println("\n\nAll the elements of ArrayList collection:");
18        for(Object element:list) {
19            System.out.print(element + ", ");
20        }
21
22        // Accessing using Iterator
23        Iterator collection = queue.iterator();
24        System.out.println("\n\nAll the elements of PriorityQueue collection:");
25        while(collection.hasNext()) {
26            System.out.print(collection.next() + ", ");
27        }
28    }
29 }
30 }
```

Console Output:

```
<terminated> TreeSetExample [Java Application] C:\Program Files\Java\jre1.8.
All the elements of ArrayList collection:
76, 2, 3, 25, 37, 6, 46, 89, 51, 63,
All the elements of PriorityQueue collection:
2, 3, 17, 50, 24, 79, 79, 92, 63, 90,
```


10. Holding Collection of Data



Accessing Collections Use of An Iterator

Accessing elements using ListIterator

The ListIterator interface is used to traverse through a list in both forward and backward directions. It does not support all types of collections. It supports only the collection which implements the List interface.

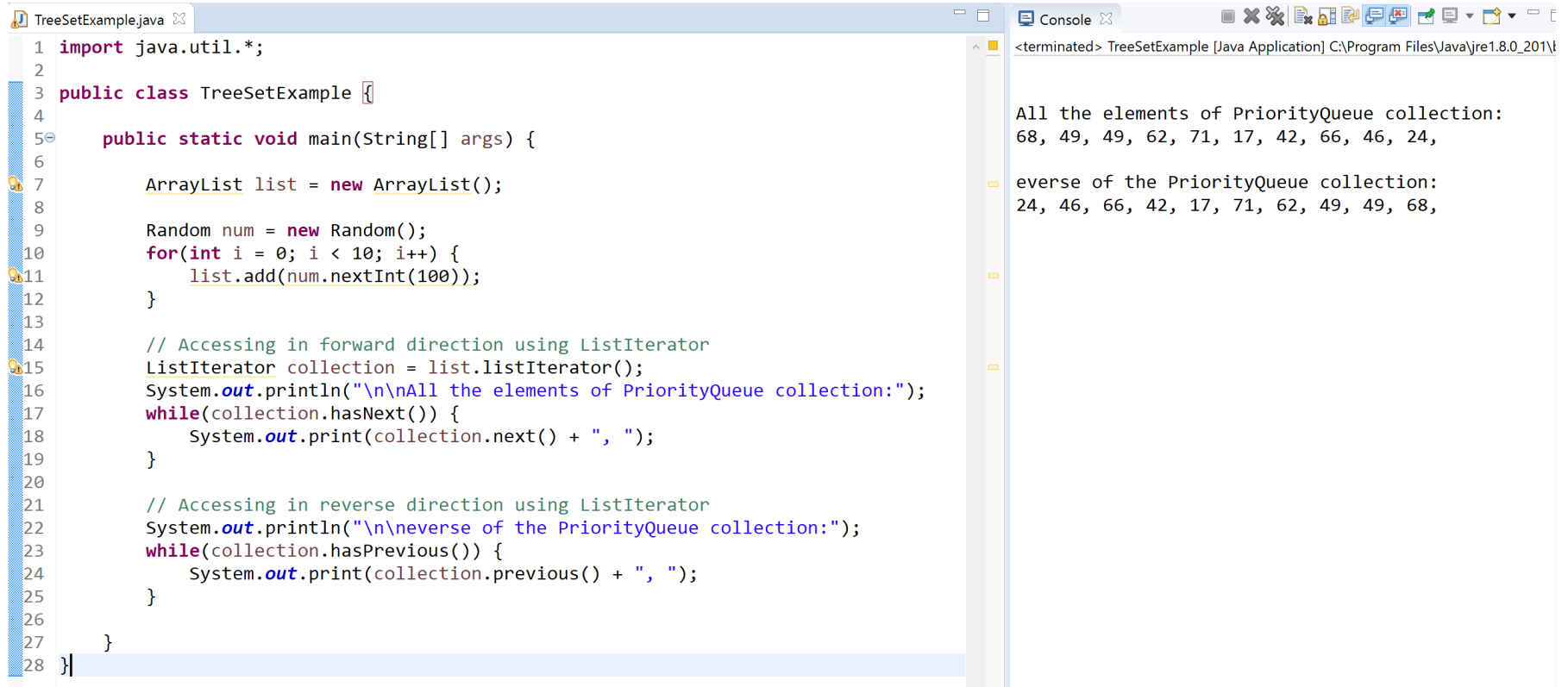
The ListIterator provides the following methods to traverse through a list of elements.

Method	Description
ListIterator listIterator()	Used to obtain an iterator of the list collection.
boolean hasNext()	Returns true if the list collection has next element, otherwise it returns false.
E next()	Returns the next element available in the list collection.
boolean hasPrevious()	Returns true if the list collection has previous element, otherwise it returns false.
E previous()	Returns the previous element available in the list collection.
int nextIndex()	Returns the index of the next element. If there is no next element, returns the size of the list.
E previousIndex()	Returns the index of the previous element. If there is no previous element, returns -1.

10. Holding Collection of Data

Accessing Collections Use of An Iterator

Accessing elements using ListIterator



The screenshot shows a Java IDE with a code editor on the left and a console window on the right. The code editor displays the following Java code:

```
1 import java.util.*;
2
3 public class TreeSetExample {
4
5     public static void main(String[] args) {
6
7         ArrayList list = new ArrayList();
8
9         Random num = new Random();
10        for(int i = 0; i < 10; i++) {
11            list.add(num.nextInt(100));
12        }
13
14        // Accessing in forward direction using ListIterator
15        ListIterator collection = list.listIterator();
16        System.out.println("\n\nAll the elements of PriorityQueue collection:");
17        while(collection.hasNext()) {
18            System.out.print(collection.next() + ", ");
19        }
20
21        // Accessing in reverse direction using ListIterator
22        System.out.println("\n\nreverse of the PriorityQueue collection:");
23        while(collection.hasPrevious()) {
24            System.out.print(collection.previous() + ", ");
25        }
26    }
27 }
28 }
```

The console window on the right shows the output of the program:

```
<terminated> TreeSetExample [Java Application] C:\Program Files\Java\jre1.8.0_201\bin\java.exe
All the elements of PriorityQueue collection:
68, 49, 49, 62, 71, 17, 42, 66, 46, 24,
reverse of the PriorityQueue collection:
24, 46, 66, 42, 17, 71, 62, 49, 49, 68,
```

10. Holding Collection of Data

Comparator Interface

- Java Comparator interface is used to order the objects inside a user-defined class.
- This interface is available in **java.util package** and includes two methods known as compare(Object obj1, Object obj2) and equals(Object element).
- Using the java Comparator, we can sort the elements based on data members of a class. For example, we can sort based on rollNo, age, salary, marks, etc.
- **Comparable interface** sorts the list structures like Arrays and ArrayLists containing custom objects.
- **Once the list objects implement Comparable interface, we can then use the Collections.sort () method or Arrays.sort () in case of the arrays to sort the contents.**

10. Holding Collection of Data



Comparator Interface

So how exactly can we write the Comparators?

Consider an example of a Class Student with name and age as its field.

Consider that we want to sort Student objects on name and age fields.

For this purpose, we will have to first write Comparator classes, StudentAgeComparator, and StudentNameComparator. In these classes, we will override the *compare()* method of the Comparator interface, and then we will call the *Collections.sort()* method using each of these comparators to sort student objects.

The compare() method returns an integer value that can have one of the following values:

Positive (> 1) integer=> the current object > the object parameter passed.

Negative (< -1) integer => the current object < the specified object.

Zero (= 0) => the current object and specified object are both equal.

We can use the compare() method to sort:

String type objects

Wrapper class objects

User-defined or custom objects

10. Holding Collection of Data

Comparator Interface

```
import java.util.*;
//class student whose objects are to be sorted
class Student{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
        this.rollno=rollno;
        this.name=name;
        this.age=age;
    }
}
//AgeComparator class implementing Comparator to compare objects
class AgeComparator implements Comparator <Student> {
    public int compare(Student s1,Student s2){
        if(s1.age==s2.age)
            return 0;
        else if(s1.age>s2.age)
            return 1;
        else
            return -1;
    }
}
```

10. Holding Collection of Data

Comparator Interface

```
class ComparatorExample1{
    public static void main(String args[]){
        //create an ArrayList of Students
        ArrayList<Student> myList=new ArrayList<Student>();
        myList.add(new Student(101,"Ram",25));
        myList.add(new Student(106,"Hari",22));
        myList.add(new Student(105,"Gita",27));

        //call Collections.sort method with AgeComparator object to sort ArrayList
        Collections.sort(myList,new AgeComparator());
        System.out.println("Sorted Student objects by Age are as follows:");

        for(Student stud: myList) {
            System.out.println(stud.age + " --> " + stud.name);
        }
    }
}
```

Avg % --> Name

90.61 --> Gouthami
85.55 --> Honey
83.55 --> Raja
80.89 --> Varshith
77.56 --> Teja

10. Holding Collection of Data



Comparator Interface

Comparable Interface	Comparator Interface
The comparable interface provides single field sorting.	Comparator interface provides multiple fields sorting.
Comparable interface sorts object as per natural ordering.	Comparator interface sorts various attributes of different objects.
Using a comparable interface we can compare the current object 'this' with the specified object.	Using a comparator interface, we can compare objects of different classes.
Part of the java.lang package.	Part of java.util package.
The use of a Comparable interface modifies the actual class.	Comparator does not alter the original class.
Provides compareTo () method to sort elements.	Provides compare () method to sort elements.
Uses Collections.sort (List) to sort elements.	Uses Collections.sort (List, Comparator) to sort the elements.

10. Holding Collection of Data

Question

1	What is the difference between ArrayList and Vector?
2	What is the difference between ArrayList and LinkedList?
3	What is the difference between Iterator and ListIterator?
4	What is the difference between Iterator and Enumeration?
5	What is the difference between List and Set?
6	What is the difference between HashSet and TreeSet?
7	What is the difference between Set and Map?
8	What is the difference between HashSet and HashMap?
9	What is the difference between HashMap and TreeMap?
10	What is the difference between HashMap and Hashtable?
11	What is the difference between Collection and Collections?
12	What is the difference between Comparable and Comparator?
13	What is the advantage of Properties file?
14	What are the classes implementing List interface?
15	What are Collection related features in Java 8?
16	What is Java Collections Framework? List out some benefits of Collections framework?

10. Holding Collection of Data

U should Know

Comparable	Comparator
1) For default Natural sorting order.	1) For Customized sorting order.
2) Present in java.lang package.	2) Present in java.util package.
3) Contains only 1 method compareTo()	3) Contains 2 methods Compare() & Equals().
4) String class and all wrapper Classes implements Comparable interface.	4) Only implemented classes of Comparator are Collator & RuleBasedCollator. (used in GUI)

Property	HashSet	LinkedHashSet	TreeSet
1) Underlying DS	Hashtable.	LinkedList +Hashtable	Balanced Tree.
2) Insertion order.	Not preserved.	Preserved.	Not preserved (by default).
3) Duplicate objects.	Not allowed.	Not allowed.	Not allowed.
4) Sorting order.	Not applicable	Not applicable.	Applicable.
5) Heterogeneous obj.	Allowed.	Allowed.	Not allowed.
6) Null insertion.	Allowed.	Allowed.	For empty TreeSet, 1st element null insertion is possible in all other cases get NullPointerException.

10. Holding Collection of Data

Keep in Mind

The 3 cursors of java →

✓ *To get objects one by one from the collection then - use cursor.*

1. Enumeration
2. Iterator
3. ListIterator

Collection and Collections ?

- ✓ Collection – **Interface**, used to represent group of objects as single entity.
- ✓ Collections – **Utility class**, present in java.util package to define several utility methods for collection objects.

10. Holding Collection of Data

9 Key Collection Interface framework

1. **Collection** - Collection interface defines the most common methods which can be applicable for any collection object.
 2. **List** [child interface of Collection] - Represent group of individual objects as single entity where *"Duplicates are allow & Insertion order must be preserved"*.
 3. **Set** [child interface of Collection] - Maintain unique ele. *"Duplicates not allow & insertion order is not preserved"*.
 4. **SortedSet** [child interface of Set] - Sets that maintain elements in sorted order. *"Duplicates not allow but all objects insertion according to some sorting order"*.
 5. **NavigableSet** [child interface of SortedSet] - Defines several methods for navigation purposes.
 6. **Queue** [child interface of Collection] - Objects *arranged in order* in which they are to be processed.
 7. **Map** [NOT child interface of Collection] - Represent group of objects as key-value pairs. *"Duplicate keys not allowed but values can be duplicated"*.
 8. **SortedMap** [child interface of Map] - Represent group of object as key value pair *"according to some sorting order of keys"*.
 9. **NavigableMap** [child interface of SortedMap] - Defines several methods for navigation purposes.
- Interfaces (Collection, List, Set, SortedSet, NavigableSet, and Queue) meant for representing a group of individual objects.

3. Object Oriented Programming Concepts



Motivate

How important it is?

Without using collection concepts, you cannot develop any production level software application in Java. This is because each java collection implementation is created and optimized for a specific type of requirement.

3. Object Oriented Programming Concepts



Motivate Pro. Real World

Best Practices for Collection

- **Code for Interface, not for Implementation**

By declaring a collection using an interface type, the code would be more flexible as you can change the concrete implementation easily when needed, for example:

```
// Better
List<String> list = new ArrayList<>();

List<String> list = new LinkedList<>();

// Avoid
ArrayList<String> list = new ArrayList<>();
```