Java Collections

Hexavarsity

Objective

- Collection introduction
- List
- Set
- Map
- Iterator
- Comparator & Comparable interface

Collection Introduction

What is Exception?

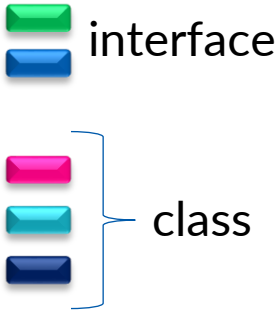
- The Java language API provides many of the data structures from this class for you.
- It defines a “collection” as “an object that represents a group of objects”.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Collection Hierarchy


interface


class

- Collection
  - List
    - ArrayList — Not Sync
    - Vector — Sync
    - LinkedList — Not Sync
    - Stack — Not Sync
  - Set
    - HashSet — Not Sync, Unorder, Allow Null
    - Linked HashSet — Not Sync, Unorder, Allow Null, use DoubleLinkedList
    - SortedSet — NavigableSet — TreeSet
      - TreeSet: Not Sync, Ascending Order
  - Queue
    - Priority Queue
    - Blocking Queue



Collection

Map

Key, Value pair

HashTable

Null not allowed
Sync
Insertion order not maintained

LinkedHashMap

Null allowed
Sync
Insertion Order maintained

HashMap

Null allowed
Not Sync
Insertion order not maintained

SortedMap

TreeMap

Sorted Order, Null allowed

Additional Interface and classs

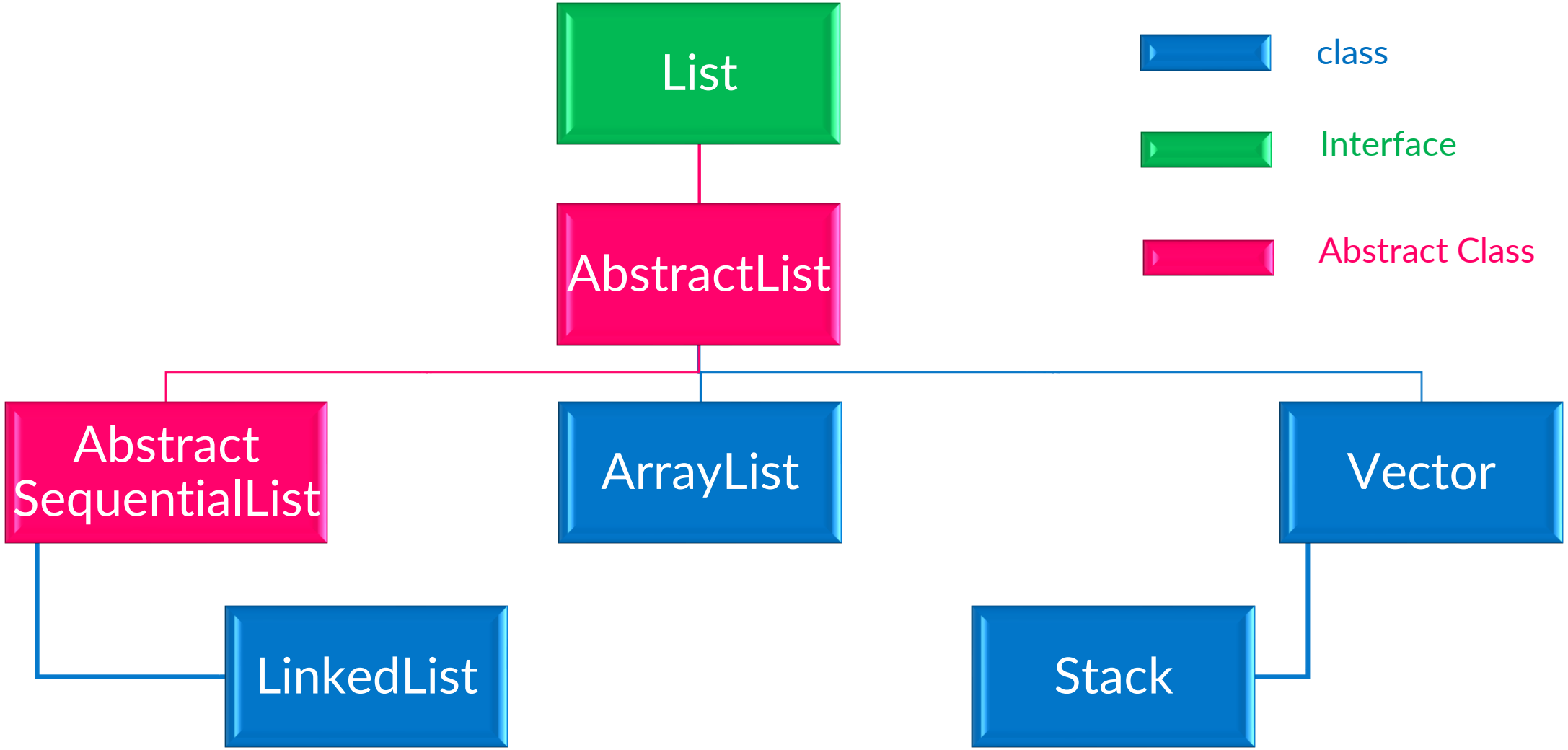
Sorting	Cursor	Utility Class
❖Comparable(I) ❖Comparator(I)	❖Iterator(I) ❖ListIterator(I) ❖Enumeration(I)	❖Arrays ❖Collections

List Interface

List Interface

- List in Java provides the facility to maintain the ordered collection of objects.
- It contains the index-based methods to insert, update, delete and search the elements.
- List contains duplicate elements also store the null elements in the list.

classes implementing List interface



Features of ArrayList

- ArrayList use the array as an internal data structure to store element
- ArrayList size can grow and sink dynamically when we add/remove elements from it.
- ArrayList class maintains insertion order.
- ArrayList class is non synchronized.
- ArrayList allows random access because array works at the index basis
- ArrayList allows to store duplicate values including “null” values

Methods in ArrayList

Method	Description	Usage
boolean add(Object element)	adds an object to the arraylist at last position	obj.add("hello");
void add(int index, Object e)	adds the object to the array list at the given index	obj.add(2, "bye");
remove(Object o)	Removes the object from the ArrayList.	obj.remove("hello");
remove(int index)	Removes element from a given index	obj.remove(3);
set(int index, Object o)	❖ Used for updating an element. ❖ It replaces the element present at the specified index with the object o.	obj.set(2, "Tom");
int indexOf(Object o)	Returns the index of the object o	int pos = obj.indexOf("Tom");
Object get(int index)	returns the object from list, which is present at the specified index.	String str= obj.get(2);
int size()	Returns the size of the ArrayList	int n = obj.size();
boolean contains(Object o)	It checks whether the given object o is present in the array list or not	obj.contains("Steve");

ArrayList Example

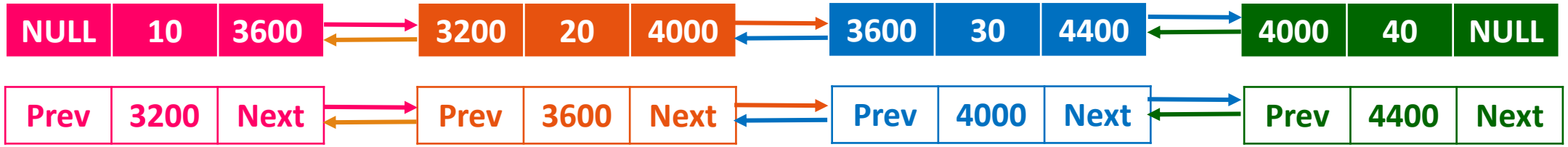
Example:

```
import java.util.*;
class ArrayListDemo
{
    public static void main(String[] args)
    {
        ArrayList al = new ArrayList();
        ArrayList a = new ArrayList();
        al.add(10);
        al.add(20.45f);
        al.add(67.987);
        al.add("welcome");
        System.out.println(al);
        al.remove(3);
        System.out.println(al);
        al.add(2,900);
        a.add("ji");
        System.out.println(al);
        al.set(3,"silent");
        System.out.println(al);
        System.out.println("----"+al.get(3));
    }
}
```

```
[10, 20.45, 67.987, welcome]
[10, 20.45, 67.987]
[10, 20.45, 900, 67.987]
[10, 20.45, 900, silent]
----silent
```

Features of LinkedList

- It uses a doubly linked list internally to store the elements. It can store the duplicate elements.
- It maintains the insertion order
- It is not synchronized.
- LinkedList class can contain duplicate elements.



Methods in LinkedList

Method	Description	Usage
boolean add(Object element)	adds an object to the arraylist at last position	obj.add("hello");
void add(int index, Object e)	adds the object to the array list at the given index	obj.add(2, "bye");
void addFirst(Object o)	inserts the given element at the beginning of a list	obj.addFirst("text");
void addLast(Object item)	It inserts the specified item at the end of the list	obj.addLast("Chat");
Object get(int index)	returns the object from list, present at the specified index.	String str= obj.get(2);
Object getFirst()	returns the first element in a list.	String var= obj.getFirst();
Object getLast()	returns the last element in a list.	String var= obj.getLast();
Object removeFirst()	It removes the first item from the list.	obj.removeFirst();
Object removeLast()	It removes the last item of the list.	obj.removeLast();
Object set(int index, Object item)	It updates the item of specified index with the give value.	obj.set(2, "Test");

Methods in LinkedList

Method	Description	Usage
Object remove(Object obj)	It removes the specified object from the list.	obj.remove("text");
Object poll()	It returns and removes the first item of the list.	String var = obj.poll();
Object pollFirst()	same as poll() method. Removes the first item of the list.	String v = obj.pollFirst()
Object pollLast()	It returns and removes the last element of the list.	String v = obj.pollLast();
int size()	Returns the size of the ArrayList	int n = obj.size();
boolean contains(Object o)	It checks whether the given object o is present in the array list or not	obj.contains("Steve");
clear()	used for removing all the elements of the array list	obj.clear();
List subList(int fromIndex, int toIndex)	Returns list of objects from this list between the specified fromIndex, toIndex **IndexOutOfBoundsException** - Index > size **IllegalArgumentException** - fromIndex > toIndex	List<String> arrlist2 = obj.subList(2, 4)
Object[] toArray()	used for converting ArrayList to Array	Integer[] objects = al.toArray();

LinkedList Example

Example:

```
import java.util.LinkedList;
import java.util.ListIterator;
public class LinkedListExample
{
   public static void main(String[] args)
   {
      LinkedList<String> linkedList = new LinkedList<>(); //Create linked list
      linkedList.add("A");  //Add elements
      linkedList.add("B");
      linkedList.add("C");       linkedList.add("D");
      System.out.println(linkedList);
      linkedList.add(4, "A"); //Add elements at specified position
      linkedList.add(5, "A");
      System.out.println(linkedList); //Remove element
      linkedList.remove("A");     //removes A
      linkedList.remove(0);       //removes B
      System.out.println(linkedList);
       ListIterator<String> itrator = linkedList.listIterator(); //Iterate
            while (itrator.hasNext())
                     System.out.println(itrator.next());

}
}
```

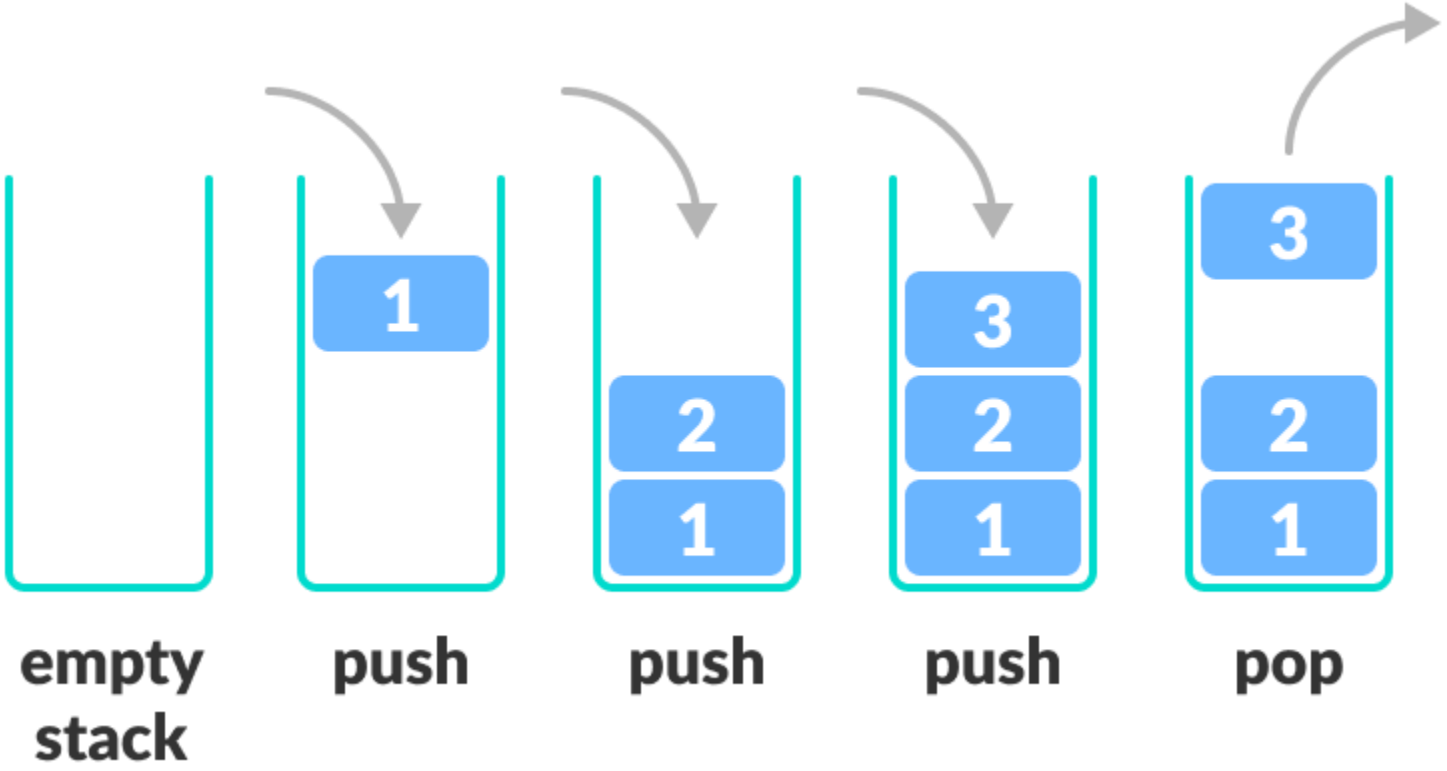
Output:
[A, B, C, D]
[A, B, C, D, A, A]
[C, D, A, A]
C
D
A
A

Features of Vector

- Vector implements a dynamic array that means it can grow or shrink as required
- They are very similar to ArrayList
- Vector is synchronized. ArrayList is not synchronized.
- Methods are same as to arraylist.

Features of Stack

- Based on the basic principle of last-in-first-out
- Child class of vector
- Insertion order is preserved



Methods in Stack

Method	Description	Usage
Object push(Object element)	Pushes an element on the top of the stack.	obj.push("hello");
Object pop()	❖ Removes and returns the top element of the stack ❖ EmptyStackException exception is thrown when stack is empty	obj.pop();
Object peek()	Returns the element on the top of the stack	obj.peek();
int search(Object element)	❖ Check whether an object exists in the stack ❖ If found, it returns the position of the element ❖ If not found, it returns -1	obj.search("hello");
boolean empty()	It returns true if nothing is on the top of the stack Else return -1.	Obj.empty()

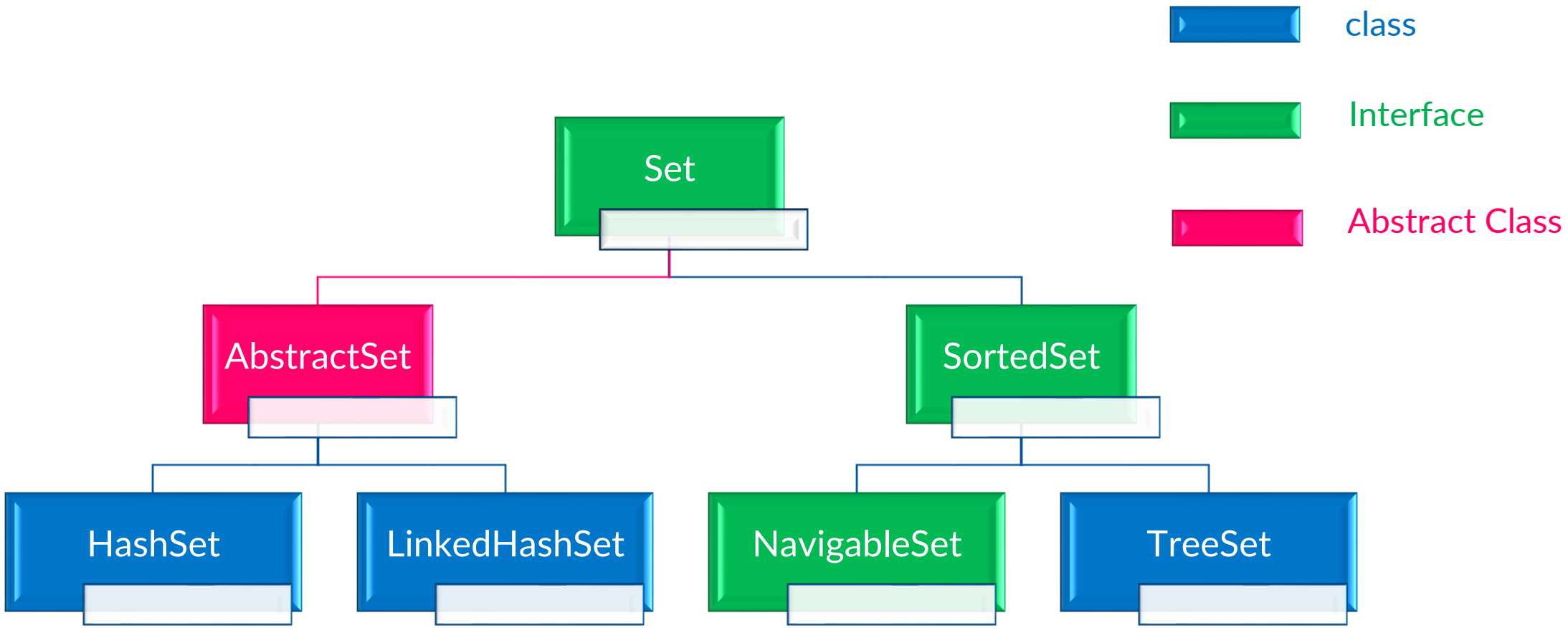
Demo

Set Interface

Set Interface

- Unordered collection of objects, Doesn’t allow duplicate elements
- You cannot access elements by their index
- Also search elements in the list.
- Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).

classes implementing Set interface



Features of HashSet

- 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 based on 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.

Methods in HashSet

Method	Description
boolean add(Object o)	Adds the specified element to this set if it is not already present.
void clear()	Removes all the elements from this set.
Object clone()	Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean contains(Object o)	Returns true if this set contains the specified element.
boolean isEmpty()	Returns true if this set contains no elements.
boolean remove(Object o)	Removes the specified element from this set if it is present.
int size()	Returns the number of elements in this set (its cardinality).
Iterator iterator()	Returns an iterator over the elements in this set.
boolean contains(Object o)	It checks whether the given object o is present in the array list or not

HashSet Example

Example:

```
import java.util.*;
class HashSet3{
 public static void main(String args[]){
  HashSet<String> set=new HashSet<String>();
        set.add("Ravi");        set.add("Vijay");        set.add("Arun");        set.add("Sumit");
        System.out.println("An initial list of elements: "+set);
        //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);
 }
}
```

```
An initial list of elements: [Vijay, Ravi, Arun, Sumit]
After invoking remove(object) method: [Vijay, Arun, Sumit]
Updated List: [Vijay, Arun, Gaurav, Sumit, Ajay]
After invoking removeAll() method: [Vijay, Arun, Sumit]
After invoking removeIf() method: [Arun, Sumit]
After invoking clear() method: []
```

Features of HashSet

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

Example

Example:

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

Example

Example:

```
public class LinkedHashSetExample {
   public static void main(String[] args) {
      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);
      //Traversing hash table
      for(Book b:hs){
      System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
      }
   }
   }
```

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

Features of TreeSet

- TreeSet is a sorted collection that extends the AbstractSet class and implements the NavigableSet interface.
- It stores unique elements
- It doesn't preserve the insertion order of the elements
- It sorts the elements in ascending order
- It's not thread-safe
- The TreeSet can only allow those generic types that are comparable. For example The Comparable interface is being implemented by the StringBuffer class.

Features of TreeSet

Sr.No.	Method & Description
1	**void add(Object o)** Adds the specified element to this set if it is not already present.
2	**boolean addAll(Collection c)** Adds all the elements in the specified collection to this set.
3	**void clear()** Removes all of the elements from this set.
4	**boolean remove(Object o)** Removes the specified element from this set if it is present.
5	**int size()** Returns the number of elements in this set (its cardinality).
6	**Object first()** Returns the first (lowest) element currently in this sorted set.
7	**Object last()** Returns the last (highest) element currently in this sorted set.

Example

Example:

```
import java.util.TreeSet;
import java.util.Iterator;
class Main {
   public static void main(String[] args) {
      //create and initialize TreeSet
      TreeSet<Integer> num_Treeset = new TreeSet<>();
      num_Treeset.add(20);
      num_Treeset.add(5);
      num_Treeset.add(15);
      num_Treeset.add(25);
      num_Treeset.add(10);
      System.out.println("TreeSet: " + num_Treeset);

      // Call iterator() method to define Iterator for TreeSet
      Iterator<Integer> iter_set = num_Treeset.iterator();
      System.out.print("TreeSet using Iterator: ");
      // Access TreeSet elements using Iterator
      while(iter_set.hasNext()) {
         System.out.print(iter_set.next());
         System.out.print(", ");
      }
   }
}
```

```
TreeSet: [5, 10, 15, 20, 25]
TreeSet using Iterator: 5, 10, 15, 20, 25,
```

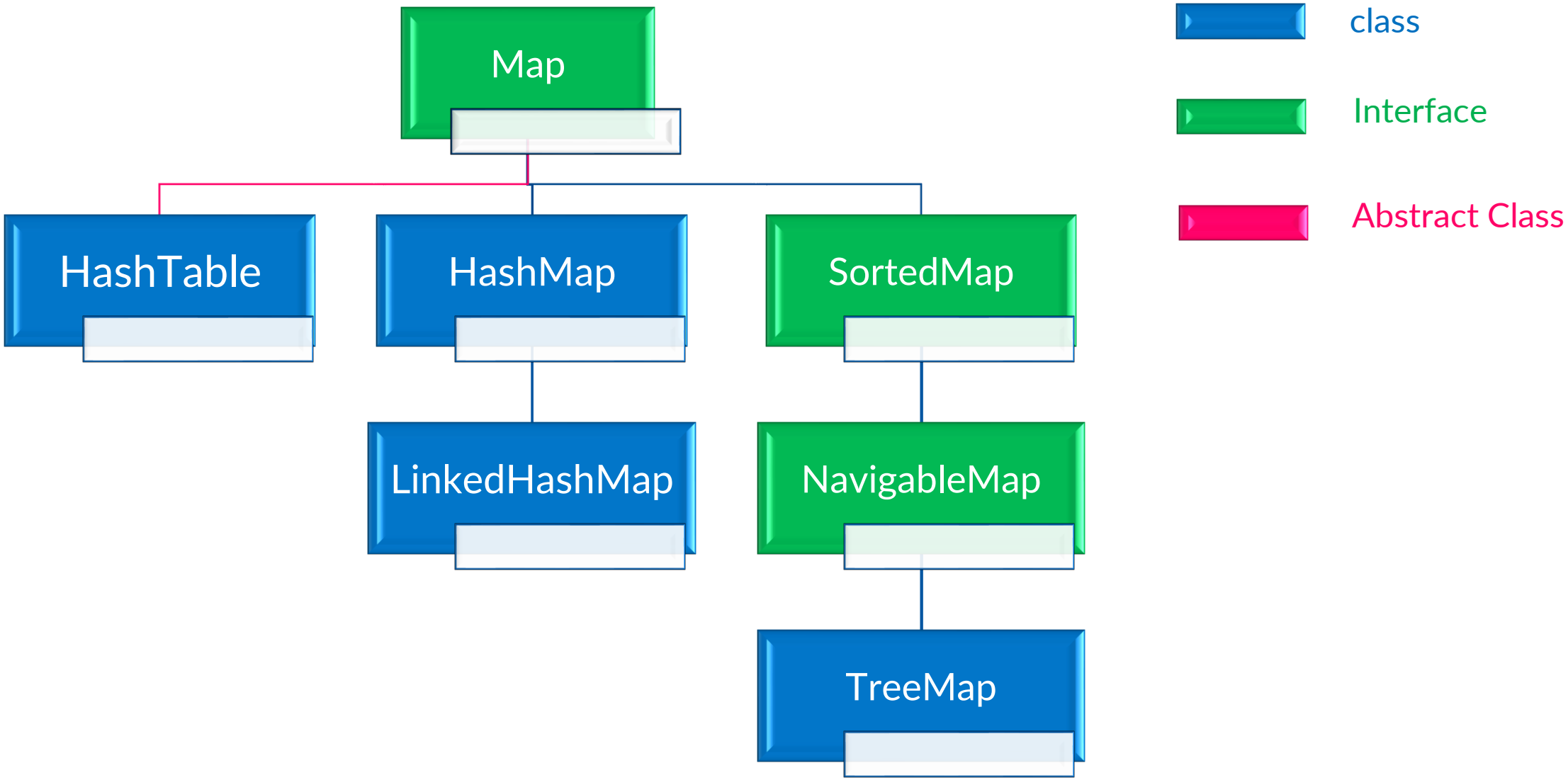
Demo

Map Interface

Features of Map Interface

- A map contains values on the basis of key, i.e. key and value pair.
- Each key and value pair is known as an entry. A Map contains unique keys.
- A Map doesn't allow duplicate keys, but you can have duplicate values.
- A Map can't be traversed, so you need to convert it into Set using keySet() or entrySet() method.

classes implementing Set interface



Features of HashMap

- HashMap class implements the Map interface which allows us to store key and value pair, where keys should be unique.
- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap is non synchronized.
- Java HashMap maintains no order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Features of HashMap

Method	Description
void clear()	It is used to remove all of the mappings from this map.
boolean isEmpty()	It is used to return true if this map contains no key-value mappings.
Object clone()	It is used to return a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
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 insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
void remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated specified keys from the map.
boolean containsValue(Object value)	This method returns true if some value equal to the value exists within the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the key exists within the map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the Map.
get(Object key)	This method returns the object that contains the value associated with the key.

Features of HashMap

Method	Description
boolean isEmpty()	This method returns true if the map is empty; returns false if it contains at least one key.
replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a specified key.
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
Collection<V> values()	It returns a collection view of the values contained in the map.
int size()	This method returns the number of entries in the map.

IOException Example

Example:

```
import java.util.*;
class HashMap3{
 public static void main(String args[]){
  HashMap<Integer,String> hm=new

            HashMap<Integer,String>();
   hm.put(100,"Amit");
   hm.put(101,"Vijay");
   hm.put(102,"Rahul");
   System.out.println("Initial list of elements:");
  for(Map.Entry m:hm.entrySet())
  {
    System.out.println(m.getKey()+" "+m.getValue());
  }
  System.out.println("Updated list of elements:");
  hm.replace(102, "Gaurav");
  for(Map.Entry m:hm.entrySet())
  {
    System.out.println(m.getKey()+" "+m.getValue());
  }
  System.out.println("Updated list of elements:");
  hm.replace(101, "Vijay", "Ravi");
  for(Map.Entry m:hm.entrySet())
  {
    System.out.println(m.getKey()+" "+m.getValue());
  }
  System.out.println("Updated list of elements:");
  hm.replaceAll((k,v) -> "Ajay");
  for(Map.Entry m:hm.entrySet())
  {
    System.out.println(m.getKey()+" "+m.getValue());
  }
 }
}
```

```
Initial list of elements:
100 Amit
101 Vijay
102 Rahul
Updated list of elements:
100 Amit
101 Vijay
102 Gaurav
```

```
Updated list of elements:
100 Amit
101 Ravi
102 Gaurav
Updated list of elements:
100 Ajay
101 Ajay
102 Ajay
```

Features of LinkedHashMap class

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains unique elements.
- Java LinkedHashMap may have one null key and multiple null values.
- Java LinkedHashMap is non synchronized.
- Java LinkedHashMap maintains insertion order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Features of LinkedHashMap class

- Java LinkedHashMap contains values based on the key.
- Java LinkedHashMap contains unique elements.
- Java LinkedHashMap may have one null key and multiple null values.
- Java LinkedHashMap is non synchronized.
- Java LinkedHashMap maintains insertion order.
- The initial default capacity of Java HashMap class is 16 with a load factor of 0.75.

Features of TreeMap class

- It is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.
- It contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- It contains only unique elements.
- It cannot have a null key but can have multiple null values.
- It is non synchronized.
- It maintains ascending order.

HashMap	TreeMap
HashMap can contain one null key.	TreeMap cannot contain any null key.
HashMap maintains no order.	TreeMap maintains ascending order.

TreeMap Example

Example:

```
import java.util.*;
public class TreeMap2 {
  public static void main(String args[]) {
   TreeMap<Integer,String> map=new TreeMap<Integer,String>();
    map.put(100,"Amit");
    map.put(102,"Ravi");
    map.put(101,"Vijay");
    map.put(103,"Rahul");
    System.out.println("Before invoking remove() method");
    for(Map.Entry m:map.entrySet())
    {
       System.out.println(m.getKey()+" "+m.getValue());
    }
    map.remove(102);
    System.out.println("After invoking remove() method");
    for(Map.Entry m:map.entrySet())
    {
       System.out.println(m.getKey()+" "+m.getValue());
    }
    }
}
```

```
Before invoking remove() method
100 Amit
101 Vijay
102 Ravi
103 Rahul
After invoking remove() method
100 Amit
101 Vijay
103 Rahul
```

Demo on Set

Comparable and Comparator Interface

Comparable Interface

- If we want to sort a List of elements then we can Collections.sort() method. It sorts the list items according to the natural ordering.
- The Comparable interface is used to compare an object of the same class with an instance of that class, it provides ordering of data for objects of the user-defined class.
- it provides the compareTo method that takes a parameter of the object of that class.

Implementing Comparable

- The objects must be mutually comparable and must not throw ClassCastException for any key in the collection.
- The compareTo() method must return a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.
- Note that compareTo() must throw an exception if y.compareTo(x) throws an exception.
- Also, the relationship between the comparable objects must be transitive i.e. (x.compareTo(y) > 0 && y.compareTo(z) > 0) implies x.compareTo(z)>0.
- null is not an instance of any class so e.compareTo(null) should throw a NullPointerException.

Exception Handling with Method Overriding

Student.java

```
import java.util.*;
import java.io.*;
class Student implements Comparable<Student>{
int rollno;
String name;
int age;
Student(int rollno,String name,int age){
        this.rollno=rollno;
        this.name=name;
        this.age=age;
}
public int compareTo(Student st){
if(age==st.age)
                return 0;
else if(age>st.age)
                return 1;
else
                return -1;
 }
}
```

Main.java

```
public class Main{
    public static void main(String args[]){
    ArrayList<Student> al=new ArrayList<Student>();
    al.add(new Student(101,"Vijay",23));
    al.add(new Student(106,"Ajay",27));
    al.add(new Student(105,"Jai",21));

                Collections.sort(al);
    for(Student st:al){
    System.out.println(st.rollno+" "+st.name+" "+st.age);
    }
    }
    }
```

```
105 Jai 21
101 Vijay 23
106 Ajay 27
```

Comparator Interface

- A comparator interface is used to order the objects of user-defined classes. A comparator object can compare two objects of the same class. Following function compare obj1 with obj2.

- Syntax:
  - public int compare(Object obj1, Object obj2):

Comparator Interface example

```
class Student{
    int rollno;
    String name;
    int age;
    Student(int rollno,String name,int age){
    this.rollno=rollno;
    this.name=name;
    this.age=age;
    }
    }
```

Student.java

```
import java.util.*;
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;
}
}
```

AgeComparator.java

```
import java.util.*;
class NameComparator implements
Comparator<Student>{
public int compare(Student s1,Student s2){
return s1.name.compareTo(s2.name);
}
}
```

NameComparator.java

Comparator Interface example

```
import java.util.*;
import java.io.*;
class TestComparator{
public static void main(String args[]){
ArrayList<Student> al=new ArrayList<Student>();
al.add(new Student(101,"Vijay",23));
al.add(new Student(106,"Ajay",27));
al.add(new Student(105,"Jai",21));

System.out.println("Sorting by Name");
Collections.sort(al,new NameComparator());
for(Student st: al){
            System.out.println(st.rollno+" "+st.name+" "+st.age);
}
System.out.println("sorting by Age");
Collections.sort(al,new AgeComparator());  //Travering the list again
for(Student st: al){
System.out.println(st.rollno+" "+st.name+" "+st.age);
}

}
}
```

TestComparator.java

```
Sorting by Name
106 Ajay 27
105 Jai 21
101 Vijay 23

Sorting by Age
105 Jai 21
101 Vijay 23
106 Ajay 27
```

Demo

Quiz

1

Which interface provides key-value pair?

a. List
b. Set
c. Map
d. Collection

c. Map

2

The Comparable interface contains which called?

a.toCompare
b.compare
c.compareTo
d.compareWith

c.compareTo

3

What is the output of the following code snippet?

```
import java.util.*;

public class Treeset {
   public static void main(String[] args)
   {
      TreeSet<String> treeSet = new TreeSet<>();

      treeSet.add("Geeks");
      treeSet.add("For");
      treeSet.add("Geeks");
      treeSet.add("GeeksforGeeks");

      for (String temp : treeSet)
         System.out.printf(temp + " ");

      System.out.println("\n");
   }
}
```

A. Geeks For Geeks GeeksforGeeks
B. Geeks For GeeksforGeeks
C. For Geeks GeeksforGeeks
D. For GeeksforGeeks Geeks

C. For Geeks GeeksforGeeks

3

What is the output of the following code snippet?

```
import java.util.*;
public class stack {
    public static void main(String[] args)
    {
        List<String> list = new LinkedList<>();
        list.add(“hexa");
        list.add("For");
        list.add(“hexa");
        list.add(“hexaforhexa");
        Iterator<Integer> iter = list.iterator();

        while (iter.hasNext())
            System.out.printf(iter.next() + " ");

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

a) hexaforhexa
b) Hexaforhexahexaforhexa
c) Runtime Error
d) Compilation Error

d) Compilation Error

5

Which of this interface must contain a unique element?

a) Set
b) List
c) Array
d) Collection

a) Set

Queries

References

1. https://www.geeksforgeeks.org/java/
2. https://www.tutorialspoint.com/java/index.htm
3. https://www.edureka.co/blog/java-tutorial/
4. https://www.javatpoint.com/java-tutorial
5. https://www.programiz.com/java-programming

Thank you

Innovative Services

Passionate Employees

Delighted Customers