CAP615
PROGRAMMING IN JAVA

Unit-2

Created By:
Kumar Vishal
(SCA), LPU

Collection Framework topics covered:

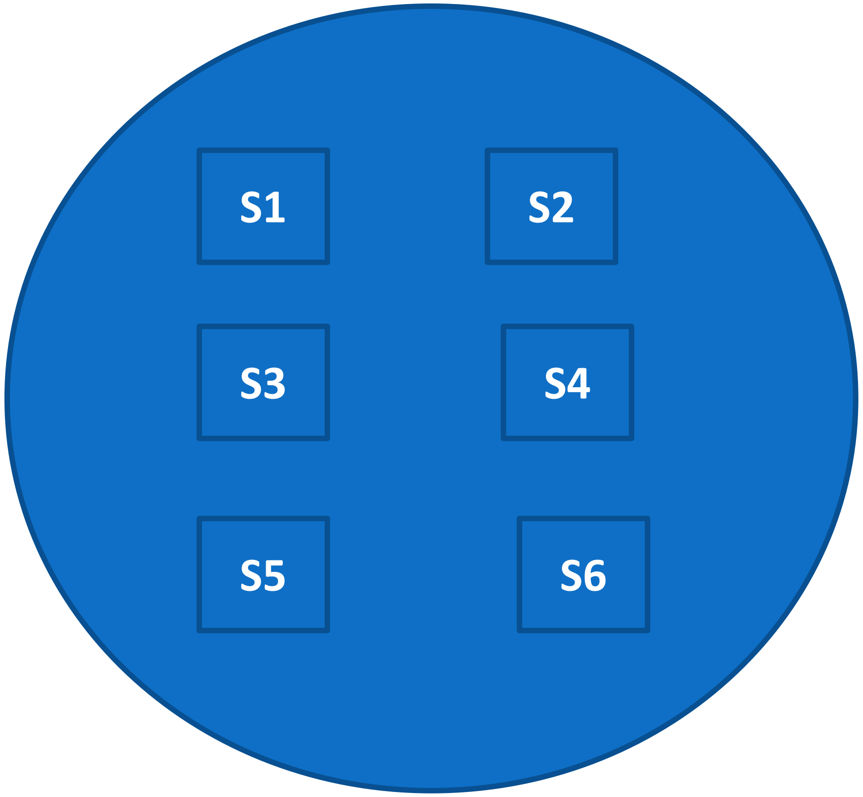
- ✓ArrayList class,
- ✓ListIterator interface,
- ✓ Linkedlist class,
- ✓TreeSet class,
- ✓PriorityQueue class,
- ✓comparable and comparator,
- ✓Properties class,
- ✓Lambda expressions

Collection

group of individual objects.

Student S1=new

Student()

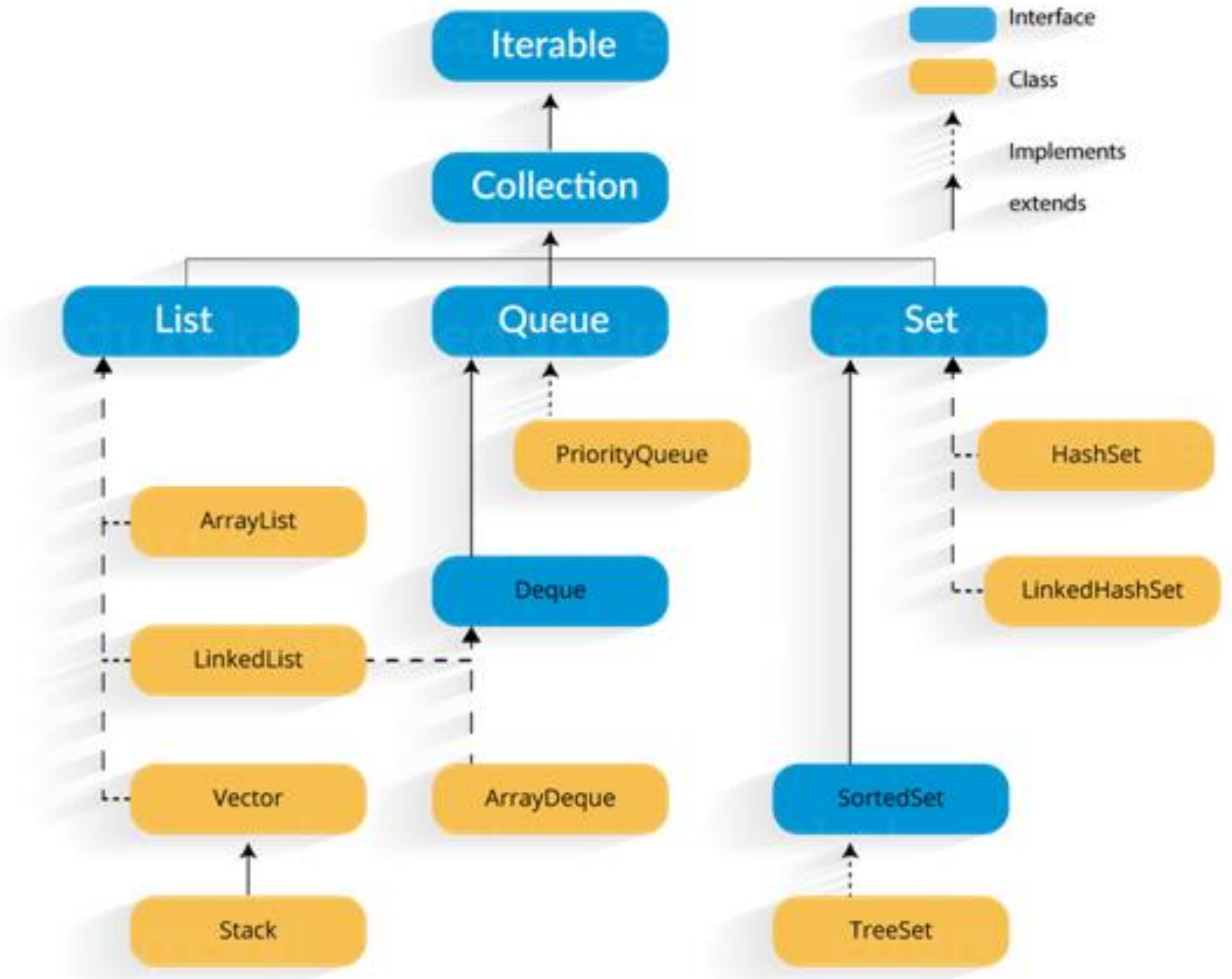


Collection framework

Several classes and interfaces which can be used a group of objects.

Package: util

import java.util.*;


Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet
Interface
Class
Implements
extends

Keyword: extends and implements

- Used in case of inheritance:
- Class to class- extends

(class classA extends classB)

- Interface to interface: extends

interface I1 extends I2

- Class to interfcae: implements

(class classA implements interfaceA)

ArrayList class

- **ArrayList** class uses a concept of *dynamic array* for storing the elements.
- It is like an array, with *no size limit*. We can add or remove elements anytime.
- It is found in the *java.util* package
- ArrayList class can contain duplicate elements also.

ArrayList al=new ArrayList();

//creating old non-generic arraylist

ArrayList<String> al=new ArrayList<String>();

//creating new generic arraylist

Java new generic collection allows you to have only one type of object in a collection. Now it is type safe so typecasting is not required at runtime.

Methods in ArrayList:

1. Add():

Add new elements to an ArrayList using the add() method.

Syntax:

arrayListObj.add(arrayListElement)

Ex:

List.add(“java”)

2. Adding an element at a particular index in an ArrayList.

Syntax:

arrayListObj.add(arrayListIndex, arrayListElement)

Ex:

List.add(2, “java”)

3. size():

to find the size of an ArrayList using the size() method.

Syntax:

arrayListObj.size()

Ex:

List.size()

4. get():

access the element at a particular index in an ArrayList using the get() method.

Syntax:

arrayListObj.get(0)

Ex:

List.get(0)

5. Set():

to modify the element at a particular index in an ArrayList using the set() method.

Syntax:

arrayListObj.set(index,element)

Ex:

List.set(4, “java”)

6. isEmpty():

To check if an ArrayList is empty using the isEmpty() method.

It will return true or false

List.isEmpty()

7. contains():

This method returns true if this list contains the specified element.

Ex:

boolean retval = arrlist.contains(10);

8. remove():

to remove the element at a given index in an ArrayList

Syntax:

arrayListObj.remove(int index)

9. removeAll():

to remove all the elements from an ArrayList.

10. indexOf():

The indexOf() method of ArrayList returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.

Iterator interface

- Iterator is an interface that iterates the elements.
- **Iterator** can traverse elements in a collection only in **forward direction**.
- It is used to traverse the list and modify the elements. **Iterator interface** has three methods:
  - **public boolean hasNext()** – This method returns true if the iterator has more elements.
  - **public object next()** – It returns the element and moves the cursor pointer to the next element.
  - **public void remove()** – This method removes the last elements returned by the iterator.

Collection classes can hold which type of values?

A. Only Primitive values

B. Only Object type values

C. Both Primitive & Objects type values

D. None of these

ListIterator

- ListIterator is an interface in a **Collection framework**, and it extends the **Iterator** interface.
- **Using ListIterator**, you can traverse the elements of the collection in both **forward** and **backwards** directions.

Methods in ListIterator

- **void add(Object object)**: It inserts object immediately before the element that is returned by the next( ) function.
- **boolean hasNext( )**: It returns true if the list has a next element.
- **boolean hasPrevious( )**: It returns true if the list has a previous element.
- **Object next( )**: It returns the next element of the list. It throws 'NoSuchElementException' if there is no next element in the list.
- **Object previous( )**: It returns the previous element of the list. It throws 'NoSuchElementException' if there is no previous element.
- **void remove( )**: It removes the current element from the list. It throws 'IllegalStateException' if this function is called before next( ) or previous( ) is invoked.

Linkedlist class

LinkedList class uses a doubly linked list to store the elements.

Methods:

void addFirst(E e)	It is used to insert the given element at the beginning of a list.
void addLast(E e)	It is used to append the given element to the end of a list.
getFirst()	It is used to return the first element in a list.
getLast()	It is used to return the last element in a list.

Linkedlist class

peek()	It retrieves the first element of a list
peekFirst()	It retrieves the first element of a list or returns null if a list is empty.
peekLast()	It retrieves the last element of a list or returns null if a list is empty.
poll()	It retrieves and removes the first element of a list.
pollFirst()	It retrieves and removes the first element of a list, or returns null if a list is empty.
pollLast()	It retrieves and removes the last element of a list, or returns null if a list is empty.
pop()	It pops an element from the stack represented by a list.
void push(E e)	It pushes an element onto the stack represented by a list.
remove()	It is used to retrieve and removes the first element of a list.
remove(int index)	It is used to remove the element at the specified position in a list.

TreeSet class

TreeSet class implements the Set interface that uses a tree for storage.

- ✓ The objects of the TreeSet class are stored in ascending order.
- ✓ Java TreeSet class contains unique elements means does not allow duplicate elements
- ✓ Java TreeSet class doesn't allow null element

Methods of TreeSet class:

add(Object o): This method will add the specified element according to the same sorting order mentioned during the creation of the TreeSet.

addAll(Collection c): This method will add all elements of the specified Collection to the set. Elements in the Collection should be homogeneous

clear(): This method will remove all the elements.

contains(Object o): This method will return true if a given element is present in TreeSet else it will return false.

first(): This method will return the first element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.

last(): This method will return the last element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.

size(): This method is used to return the size of the set or the number of elements present in the set.

```
import java.util.*;
public class Main
{
		public static void main(String[] args)
		{
				TreeSet<String> t1=new TreeSet<String>();
				t1.add("Kumar");
				t1.add("Vishal");
				t1.add("Rahul");
				t1.add("Abhinav");
				t1.add("Vishal");
				for(String str:t1)
				{
				   System.out.println(str);
				}
		}
}
```

PriorityQueue Class

A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a Queue follows the First-In-First-Out algorithm.

Operations on PriorityQueue:

1. **Adding Elements:** In order to add an element in a priority queue, we can use the add() method.
2. **Removing Elements:** In order to remove an element from a priority queue, we can use the remove() method.
3. **Accessing the elements:** Since Queue follows the First In First Out principle, we can access only the head of the queue.
4. **Iterating the PriorityQueue:** There are multiple ways to iterate through the PriorityQueue. The most famous way is converting the queue to the array and traversing using the for loop.

Example:

```
PriorityQueue<String> pq = new PriorityQueue<>();
    pq.add("Samsung");
    pq.add("Nokia");
    pq.add("RealMe");
     Iterator iterator = pq.iterator();
     while (iterator.hasNext())
       {
      System.out.print(iterator.next() + " ");
      }
```

Methods in Stack class

- **Object push(*Object element*)** : Pushes an element on the top of the stack.
- **Object pop()** : Removes and returns the top element of the stack. An ‘EmptyStackException’ exception is thrown if we call pop() when the invoking stack is empty.
- **Object peek()** : Returns the element on the top of the stack, but does not remove it.
- **boolean empty()** : It returns true if nothing is on the top of the stack. Else, returns false.
- **int search(*Object element*)** : It determines whether an object exists in the stack. If the element is found, it returns the position of the element from the top of the stack. Else, it returns -1.

Comparable and Comparator

Comparable and Comparator both are interfaces and can be used to sort collection elements.

Comparable	Comparator
1) Comparable provides a **single sorting sequence**. In other words, we can sort the collection on the basis of a single element such as id, name, and price.	The Comparator provides **multiple sorting sequences**. In other words, we can sort the collection on the basis of multiple elements such as id, name, and price etc.
2) Comparable provides **compareTo() method** to sort elements.	Comparator provides **compare() method** to sort elements.
3) Comparable is present in **java.lang** package.	A Comparator is present in the **java.util** package.

Properties class in Java

The **properties** object contains key and value pair both as a string.

An Advantage of the properties file

Recompilation is not required if the information is changed from a properties file: If any information is changed from the properties file, you don't need to recompile the java class.

Lambda expressions

- In programming, a Lambda expression (or function) is just an anonymous function, i.e., a function with no name
- The Lambda expression is used to provide the implementation of an functional interface ,we don't need to define the method again for providing the implementation.

Note: An interface which has only one abstract method is called functional interface.

Lambda Expression Syntax

(argument-list) -> {body};

1) **Argument-list:** It can be empty or non-empty

2) **Arrow-token:** It is used to link arguments-list and body of expression.

3) **Body:** It contains expressions and statements for lambda expression.