Object Oriented Programming Through Java

Lecture Notes

Unit-IV

Multithreading, The Collections Framework (java.util)

Prepared by

Mr. P.Bhaskar

Assistant Professor (Dept of CSE)

ADITYA COLLEGE OF ENGINEERING

MADANAPALLI

Unit-IV

Multithreading, The Collections Framework (java.util)

Multithreading: The Java thread model, Creating threads, Thread priorities, Synchronizing threads,Interthread communication.

The Collections Framework (java.util): Collections overview, Collection Interfaces, The Collection classes- Array List, Linked List, Hash Set, Tree Set, Priority Queue, Array Deque. Hashtable, Properties, Stack, Vector, String Tokenizer, Bit Set, Date, Calendar, Random, Formatter, Scanner.

CONTENTS

Multithreading in java

- The java programming language allows us to create a program that contains one or more parts that can run simultaneously at the same time. This type of program is known as a multithreading program.
- Each part of this program is called a thread. Every thread defines a separate path of execution in java.

A thread is explained in different ways, and a few of them are as specified below.

- A thread is **a light weight process**. (or) A thread is a **subpart of a process** that can **run individually**.
- In java, multiple threads can run at a time, which enables the java to write multitasking programs.

- The multithreading is a specialized form of multitasking.
- All modern operating systems support multitasking.

There are two types of multitasking, and they are as follows.

- Process-based multitasking
- Thread-based multitasking

It is important to know the difference between process-based and thread-based multitasking.

Process-based multitasking	**Thread-based multitasking**
It allows the computer to run two or more programs concurrently	It allows the computer to run two or more threads concurrently
In this process is the smallest unit.	In this thread is the smallest unit.
Process is a larger unit.	Thread is a part of process.
Process is heavy weight.	Thread is light weight.
Process requires separate address space for each.	Threads share same address space.
Process never gain access over idle time of CPU.	Thread gain access over idle time of CPU.
Inter process communication is expensive.	Inter thread communication is not expensive.

Thread class

Thread class provide constructors and methods to create and perform operations on a thread.

Thread class extends Object class and implements Runnable interface.

The Thread class has the following consructors.

- Thread( )
- Thread(String threadName)
- Thread(Runnable objectName)
- Thread(Runnable objectName, String threadName)

The Thread class contains the following methods.

Method	Description	Return Value
run( )	Defines actual task of the thread.	void
start( )	It moves there thread from Ready state to Running state by calling	Void
setName(String)	Assigns a name to the thread.	Void
getName( )	Returns the name of the thread.	String
setPriority(int)	Assigns priority to the thread.	Void
getPriority( )	Returns the priority of the thread.	Int
getId( )	Returns the ID of the thread.	Long
activeCount()	Returns total number of thread under active.	Int
currentThread( )	Returns the reference of the thread that currently in running state.	Void
sleep(long)	moves the thread to blocked state till the specified number of milliseconds.	Void
isAlive( )	Tests if the thread is alive.	Boolean
yield( )	Tells to the scheduler that the current thread is willing to yield its current use of a processor.	Void
join( )	Waits for the thread to end.	Void

- The Thread class in java also contains methods like stop( ), destroy( ), suspend( ), and resume( ). But they are deprecated.
- Other thread methods are

  isDaemon(): tests if the thread is a daemon thread.

  getId(): returns the id of the thread.

  interrupt(): interrupts the thread.

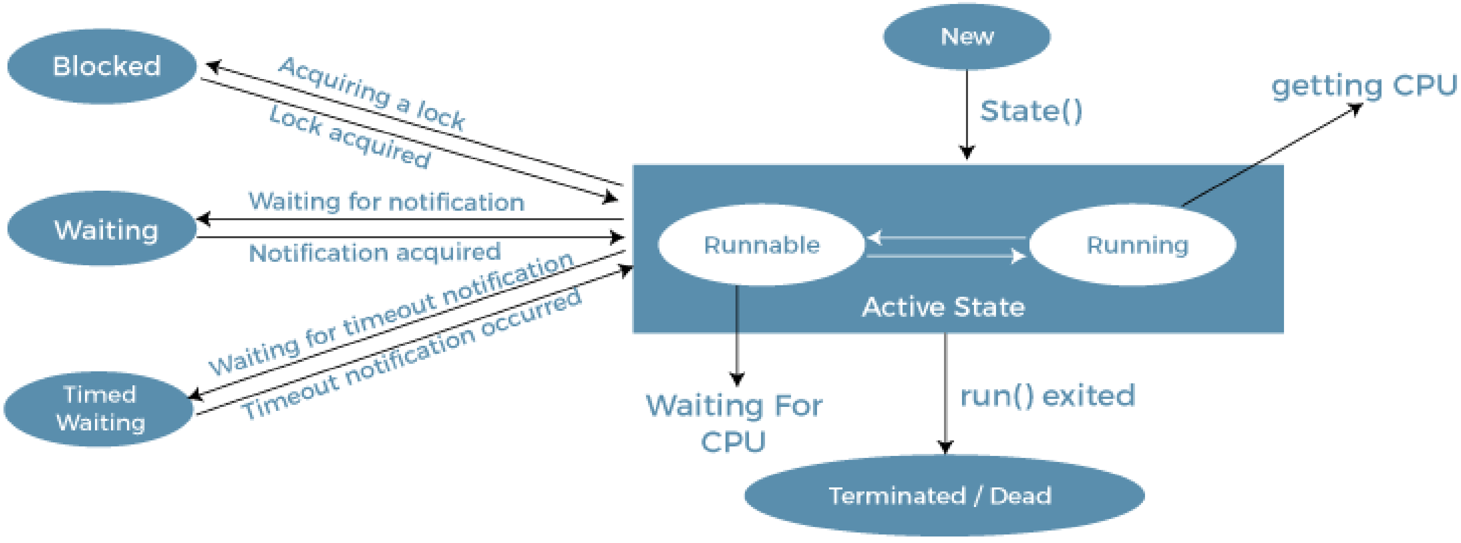
  isInterrupted(): tests if the thread has been interrupted.

Note: Deprecated is one that programmers are discouraged from using, typically because it is dangerous, or because a better alternative exists.

4.1 The Java thread model

- ➢ In java, a thread goes through different states throughout its execution. These stages are called thread life cycle states or phases.
- ➢ A thread may in any of the states like new, ready or runnable, running, blocked or wait, and dead or terminated state.

The life cycle of a thread in java is shown in the following figure.



Life Cycle of a Thread

New

When a thread object is created using new, then the thread is said to be in the New state.

This state is also known as Born state.

Example

```
Thread t1 = new Thread();
```

Runnable / Ready

When a thread calls start( ) method, then the thread is said to be in the Runnable state.

This state is also known as a Ready state.

Example

```
t1.start( );
```

Running

When a thread calls run( ) method, then the thread is said to be Running.

The run( ) method of a thread called automatically by the start( ) method.

Blocked / Waiting

A thread in the Running state may move into the blocked state due to various reasons like sleep( ) method called, wait( ) method called, suspend( ) method called, and join( ) method called, etc.

When a thread is in the blocked or waiting state, it may move to Runnable state due to reasons like sleep time completed, waiting time completed, notify( ) or notifyAll( ) method called, resume( ) method called, etc.

Example

```
Thread.sleep(1000);
wait(1000);
wait();
suspend();
notify();
notifyAll();
resume();
```

Dead / Terminated

A thread in the Running state may move into the dead state due to either its execution completed or the stop( ) method called. The dead state is also known as the terminated state.

4.2 Creating Threads

- In java, a thread is a lightweight process.
- Every java program executes by a thread called the main thread. When a java program gets executed, the main thread created automatically. All other threads called from the main thread.

  The java programming language provides two methods to create threads listed below.
- **Using Thread class (by extending Thread class)**
- **Using Runnable interface (by implementing Runnable interface)**

Extending Thread class

- The java contains a built-in class Thread inside the java.lang package.
- The Thread class contains all the methods that are related to the threads.

To create a thread using Thread class, follow the step given below.

- **Step-1**: Create a class as a child of Thread class. That means, create a class that extends Thread class.
- **Step-2**: Override the run( ) method with the code that is to be executed by the thread. The run( ) method must be public while overriding.
- **Step-3**: Create the object of the newly created class in the main( ) method.
- **Step-4**: Call the start( ) method on the object created in the above step.

Example

```
class SampleThread extends Thread
{
        public void run()
         {
                for(int i= 1; i<=4; i++)
                {
                        try
                        {
                                System.out.println("i = " + i);
                                Thread.sleep(1000);
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        }
}

public class ThreadCreate
{
        public static void main(String[] args)
         {
                SampleThread s1 = new SampleThread();
                SampleThread s2 = new SampleThread();
                Thread t1=new Thread(s1);
                Thread t2=new Thread(s2);
                System.out.println("Thread about to start...");
                t1.start();
                t2.start();
        }
}
```

Output:

```
D:\ACEM\II CSE  Bsection>javac ThreadCreate.java
D:\ACEM\II CSE  Bsection>java ThreadCreate
Thread about to start...
i = 1
i = 1
i = 2
i = 2
i = 3
i = 3
i = 4
i = 4
```

Implementng Runnable interface

- The java contains a built-in interface Runnable inside the java.lang package.
- The Runnable interface implemented by the Thread class that contains all the methods that are related to the threads.

To create a thread using Runnable interface, follow the step given below.

- **Step-1**: Create a class that implements Runnable interface.
- **Step-2**: Override the run( ) method with the code that is to be executed by the thread. The run( ) method must be public while overriding.
- **Step-3**: Create the object of the newly created class in the main( ) method.
- **Step-4**: Create the Thread class object by passing above created object as parameter to the Thread class constructor.
- **Step-5**: Call the start( ) method on the Thread class object created in the above step.

Example

```
class SampleThread implements Runnable
{
        public void run()
         {
                for(int i= 1; i<=4; i++)
                {
                        try
                        {
                                System.out.println("i = " + i);
                                Thread.sleep(1000);

                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        }
}
```

```
public class ThreadCreate1
{
        public static void main(String[] args)
         {
                SampleThread s1 = new SampleThread();

                SampleThread s2 = new SampleThread();

                Thread t1=new Thread(s1);

                Thread t2=new Thread(s2);

                System.out.println("Thread about to start...");

                t1.start();

                t2.start();

        }
}
```

Output:

```
D:\ACEM\II CSE  Bsection>javac ThreadCreate1.java
D:\ACEM\II CSE  Bsection>java ThreadCreate1
Thread about to start...
i = 1
i = 1
i = 2
i = 2
i = 3
i = 3
i = 4
```

4.3 Thread priorities

- In a java programming language, every thread has a property called priority.
- Most of the scheduling algorithms use the thread priority to schedule the execution sequence.
- In java, the thread priority range from 1 to 10.
- Priority 1 is considered as the lowest priority, and priority 10 is considered as the highest priority.
- The thread with more priority allocates the processor first.
- Thread class provides two methods setPriority(int),and getPriority( ) to handle thread priorities.

- The Thread class also contains three constants that are used to set the thread priority, and they are

    MAX_PRIORITY - It has the value 10 and indicates highest priority.

    NORM_PRIORITY - It has the value 5 and indicates normal priority.

    MIN_PRIORITY - It has the value 1 and indicates lowest priority.

The default priority of any thread is 5 (i.e. NORM_PRIORITY).

setPriority( ) method

The setPriority( ) method of Thread class used to set the priority of a thread.

It takes an integer range from 1 to 10 as an argument and returns nothing (void).

The regular use of the setPriority( ) method is as follows.

Example

threadObject.setPriority(4);   or   threadObject.setPriority(MAX_PRIORITY);

getPriority( ) method

The getPriority( ) method of Thread class used to access the priority of a thread. It does not takes any argument and returns name of the thread as String.

The regular use of the getPriority( ) method is as follows.

Example

String threadName = threadObject.getPriority();

Example

```
class SampleThread extends Thread
{
        public void run()
        {
                System.out.println("Inside SampleThread");
                System.out.println("Current Thread: " + Thread.currentThread().getName());
        }
}
public class PriorotyThreds
{
        public static void main(String[] args)
        {
                SampleThread   threadObject1  =  new     SampleThread();
                SampleThread   threadObject2  =  new     SampleThread();
                threadObject1.setName("first");
                threadObject2.setName("second");
                threadObject1.setPriority(4);
                threadObject2.setPriority(Thread.MAX_PRIORITY);
                threadObject1.start();
                threadObject2.start();
        }
}
```

Output:

```
D:\ACEM\II CSE  Bsection>javac PriorotyThreds.java
D:\ACEM\II CSE  Bsection>java PriorotyThreds
Inside SampleThread
Inside SampleThread
Current Thread: second
Current Thread:first
```

Note: In java, it is not guaranteed that threads execute according to their priority because it depends on JVM specification that which scheduling it chooses.

4.4 Synchronizing threads

- The java programming language supports multithreading.
- The problem of shared resources occurs when two or more threads get execute at the same time.
- In such a situation, we need some way to ensure that the shared resource will be accessed by only one thread at a time, and this is performed by using the concept called synchronization.

The synchronization is **the process of allowing only one thread to access a shared resource at a time**.

In java, the synchronization is achieved using the following concepts.

- Mutual Exclusion
- Inter thread communication

Mutual Exclusion

Using the mutual exclusion process, we keep threads from interfering with one another while they accessing the shared resource.

In java, mutual exclusion is achieved using the following concepts.

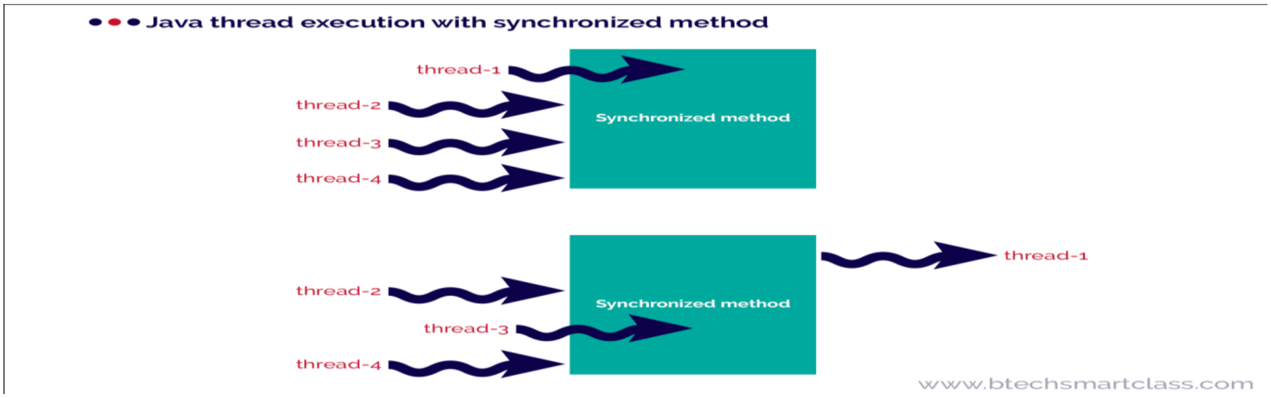
Synchronized method

Synchronized block

Synchronized method

When a method created using a synchronized keyword, it allows only one object to access it at a time.

- When an object calls a synchronized method, it put a lock on that method so that other objects or thread that are trying to call the same method must wait, until the lock is released.
- Once the lock is released on the shared resource, one of the threads among the waiting threads will be allocated to the shared resource.



- In the above image, initially the thread-1 is accessing the synchronized method and other threads (thread-2, thread-3, and thread-4) are waiting for the resource (synchronized method).
- When thread-1 completes it task, then one of the threads that are waiting is allocated with the synchronized method, in the above it is thread-3.

Example

```
class Table
{
        synchronized void printTable(int n)    // Here, method is synchronized.
        {
                 for(int i = 1; i <= 5; i++)
                {
                        System.out.println(n * i);
                        try
                        {
                                Thread.sleep(400);
                        }
                        catch(InterruptedException ie)
                        {
                                 System.out.println(ie);
                        }
                 }
        }
}
class Thread1 extends Thread
{
         Table t;            // Declaration of variable t of class type Table.
          Thread1(Table t)       // Declare one parameterized constructor and pass variable t as a parameter.
         {
                this.t = t;
         }
         public void run()
        {
                t.printTable(2);
        }
}
```

```
 class Thread2 extends Thread
{
        Table t;

         Thread2(Table t)
         {
                this.t = t;
         }

        public void run()
        {
                t.printTable(10);
        }
}


public class SynchronizedMethod
{
        public static void main(String[] args)
        {
                Table obj = new Table();     // Create an object of class Table.
                Thread1 t1 = new Thread1(obj);
               Thread2 t2 = new Thread2(obj);
                t1.start();
                t2.start();
        }
}
```

Output:

```
        D:\ACEM\II CSE  Bsection>javac SynchronizedMethod.java
        D:\ACEM\II CSE  Bsection>java SynchronizedMethod
2
4
6
8
10
10
20
30
40
50
```

Synchronized block

The synchronized block is used when we want to synchronize only a specific sequence of lines in a method.

For example, let's consider a method with 20 lines of code where we want to synchronize only a sequence of five lines code, we use the synchronized block.

Syntax

```
synchronized(object)
{                    ...
              block code
                     ...
}
```

Note: The complete code of a method may be written inside the synchronized block, where it works similarly to the synchronized method.

4.5 Interthread communication

- Inter thread communication is the concept where two or more threads communicate to solve the problem of **polling**.
- In java, polling is the situation to check some condition repeatedly, to take appropriate action, once the condition is true. That means, in inter-thread communication, a thread waits until a condition becomes true such that other threads can execute its task.
- The inter-thread communication allows the synchronized threads to communicate with each other.

Java provides the following methods to achieve inter thread communication.

Method	Description
void wait( )	It makes the current thread to pause its execution until other thread in the same monitor calls notify( )
void notify( )	It wakes up the thread that called wait( ) on the same object.
void notifyAll()	It wakes up all the threads that called wait( ) on the same object.

calling notify( ) or notifyAll( ) does not actually give up a lock on a resource.

Let's look at an example problem of producer and consumer.

The producer produces the item and the consumer consumes the same. But here, the consumer can not consume until the producer produces the item, and producer can not produce until the consumer consumes the item that already been produced.

So here, the consumer has to wait until the producer produces the item, and the producer also needs to wait until the consumer consumes the same. Here we use the inter-thread communication to implement the producer and consumer problem.

Example

```
import   java.util.LinkedList;

class ProducerConsumer
{
        LinkedList<Integer>    list = new LinkedList<>();

        public void produce() throws InterruptedException
        {
                int value=0;

                while(true)

                {
                        synchronized(this)
                        {
                                while(list.size()>0)
                                wait();

                                System.out.println("producer data "+value);
                                list.add(value);

                                value++;
                                notify();
                        }
                }
        }


        public void consume()  throws InterruptedException
        {
                while(true)
                {
                        synchronized(this)
                        {
                                while(list.size() ==0)
                                wait();

                                int  value=list.removeFirst();

                                System.out.println("Consumer data "+value);
                                notify();

                                Thread.sleep(2000);
                        }
                }
        }
}
```

```
class  ExeLogics
{

       public   static  void main(String    args[])throws InterruptedException
      {
        ProducerConsumer      pc  = new  ProducerConsumer();

        Thread     t1  = new  Thread(new Runnable()
        {
                public void run()
                {
                        try
                        {
                                pc.produce();
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        });

        Thread     t2  = new  Thread(new Runnable()
        {
                public void run()
                {
                        try
                        {
                                pc.consume();
                        }
                        catch(Exception e)
                        {
                                System.out.println(e);
                        }
                }
        });

                        t1.start();
                        t2.start();

                        t1.join();
                        t2.join();
     }
}
```

Output

```
D:\ACEM\II  AI   DS>javac ExeLogics.java
D:\ACEM\II  AI   DS>java ExeLogics

producer data 0
Consumer data 0
producer data 1
Consumer data 1
producer data 2
Consumer data 2
producer data 3
Consumer data 3
```

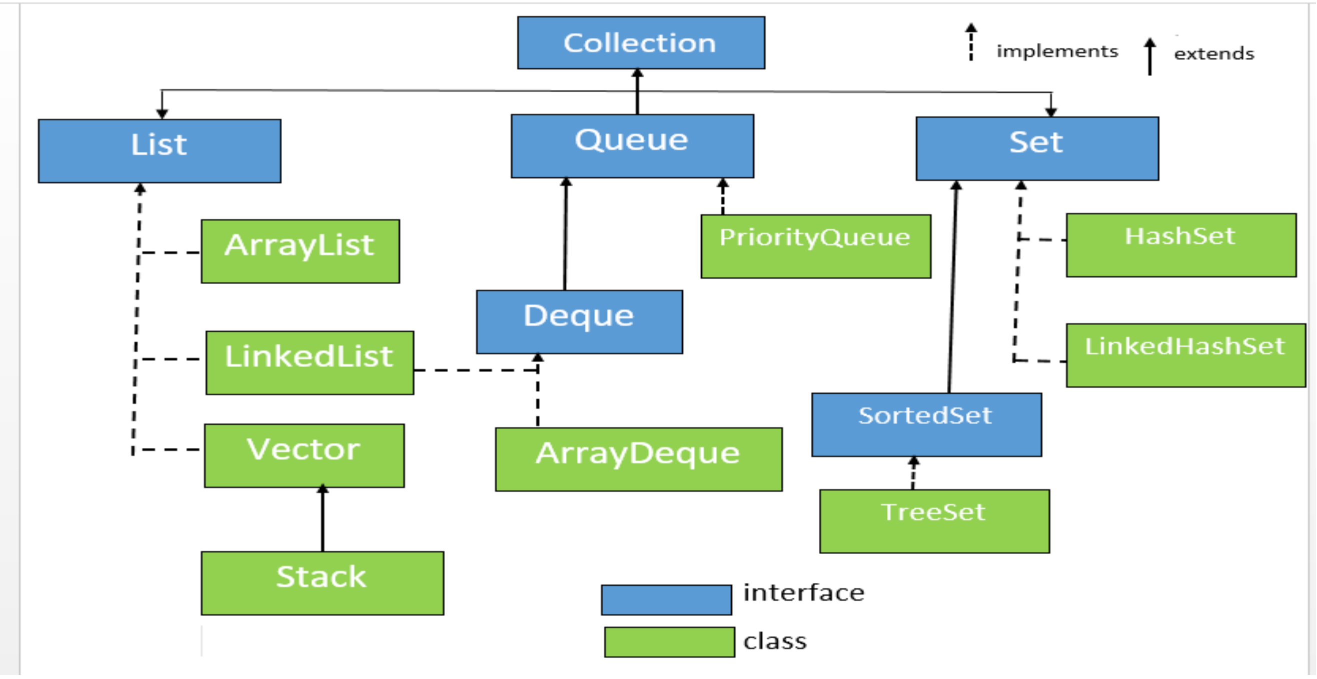
Note: while running this program press CTRL+C to break the execution

4.6 Java Collection Framework Overview

- Java collection framework is a collection of interfaces and classes used to storing and processing a group of individual objects as a single unit.
- The java collection framework holds several classes that provide a large number of methods to store and process a group of objects.

These classes make the programmer task super easy and fast.

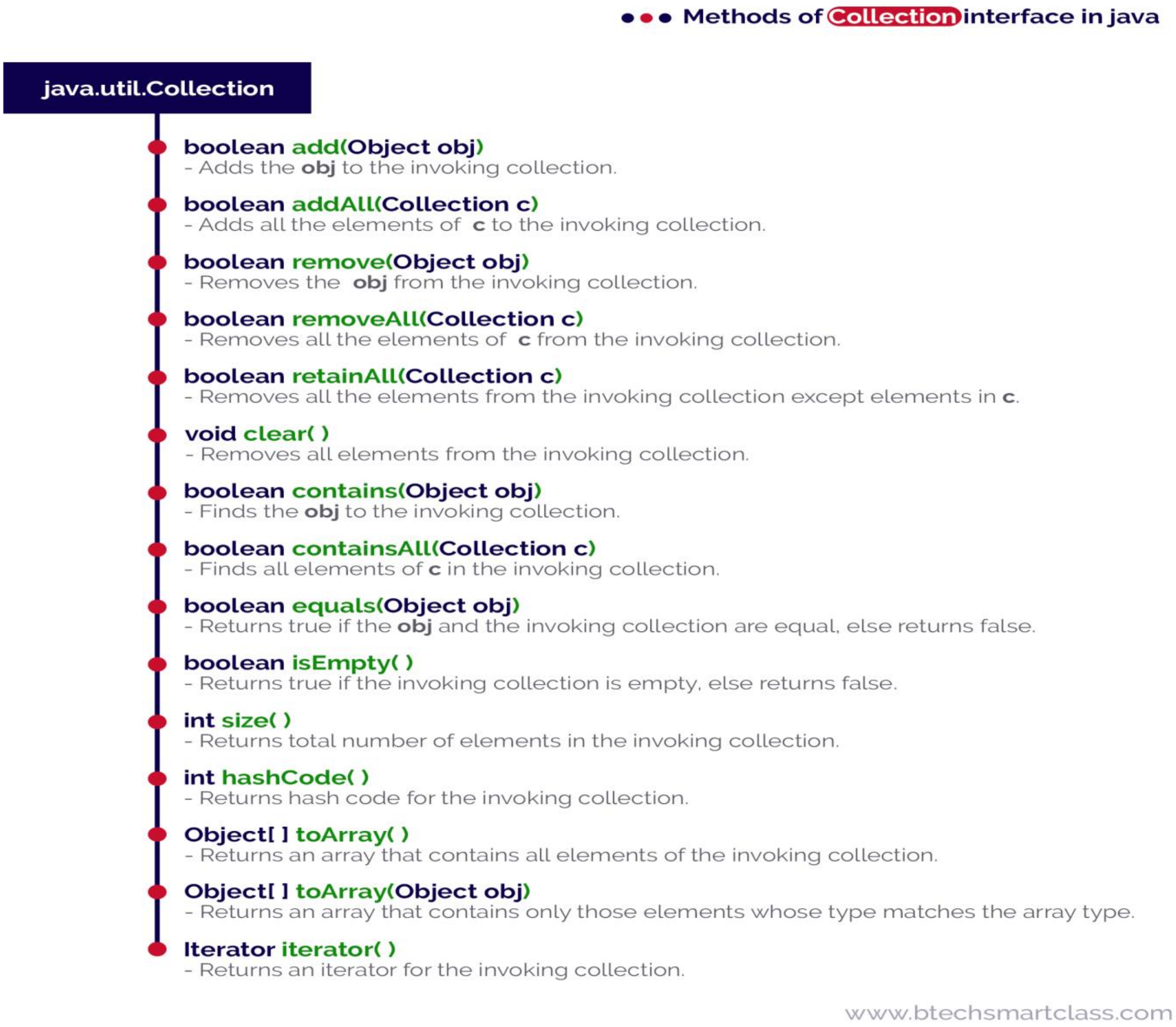
- Java collection framework was introduced in java 1.2 version.
- Java collection framework has the following hierarchy.



- Before the collection framework in java (before java 1.2 version), there was a set of classes like **Array**, **Vector**, **Stack**, **HashTable**. These classes are known as **legacy classes**.
- The java collection framework contains List, Queue, Set, and Map as top-level interfaces.
- The List, Queue, and Set stores single value as its element, whereas Map stores a pair of a key and value

4.7 Java Collection Interface

- The **Collection** interface is the root interface for most of the interfaces and classes of collection framework.
- The Collection interface is available inside the **java.util** package.
- It defines the methods that are commonly used by almost all the collections.
- The Collection interface extends **Iterable** interface.
- The Collection interface defines the following methods.



Methods of Collection interface in java

java.util.Collection

boolean add(Object obj)
- Adds the **obj** to the invoking collection.

boolean addAll(Collection c)
- Adds all the elements of **c** to the invoking collection.

boolean remove(Object obj)
- Removes the **obj** from the invoking collection.

boolean removeAll(Collection c)
- Removes all the elements of **c** from the invoking collection.

boolean retainAll(Collection c)
- Removes all the elements from the invoking collection except elements in **c**.

void clear( )
- Removes all elements from the invoking collection.

boolean contains(Object obj)
- Finds the **obj** to the invoking collection.

boolean containsAll(Collection c)
- Finds all elements of **c** in the invoking collection.

boolean equals(Object obj)
- Returns true if the **obj** and the invoking collection are equal, else returns false.

boolean isEmpty( )
- Returns true if the invoking collection is empty, else returns false.

int size( )
- Returns total number of elements in the invoking collection.

int hashCode( )
- Returns hash code for the invoking collection.

Object[ ] toArray( )
- Returns an array that contains all elements of the invoking collection.

Object[ ] toArray(Object obj)
- Returns an array that contains only those elements whose type matches the array type.

Iterator iterator( )
- Returns an iterator for the invoking collection.

www.btechsmartclass.com

The Collection classes

4.8 ArrayList class

The **ArrayList** class is a part of java collection framework and It is available inside the **java.util** package.

- The ArrayList class **extends *AbstractList*** class and **implements *List*** interface.
- The elements of ArrayList are organized as an array internally. The **default size** of an ArrayList is **10**.
- The ArrayList class is used to create a **dynamic array** that can **grow or shrunk as needed**.
- ArrayList is is a child class of **AbstractList** implements List, Serializable, Cloneable, RandomAccess.
- The ArrayList **allows** to store **duplicate data values**.
- The ArrayList **allows to access** elements **randomly using index-**based accessing.
- The ArrayList maintains the order of insertion.

ArrayList class declaration

public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable, Serializable

ArrayList class constructors

The ArrayList class has the following constructors.

- **ArrayList( )** - Creates an empty ArrayList.
- **ArrayList(Collection c)** - Creates an ArrayList with given collection of elements.
- **ArrayList(int size)** - Creates an empty ArrayList with given size (capacity).

Operations on ArrayList

The ArrayList class allow us to perform several operations like adding, accessing, deleting, updating, etc.

Adding Items

The ArrayList class has the following methods to add items.

- **boolean add(E element)** - Appends given element to the ArrayList.
- **boolean addAll(Collection c)** - Appends given collection of elements to the ArrayList.
- **void add(int index, E element)** - Inserts the given element at specified index.
- **boolean addAll(int index, Collection c)** - Inserts the given collection of elements at specified index.

Accessing Items

The ArrayList class has the following methods to access items.

- **E get(int index)** - Returns element at specified index from the ArrayList.
- **ArrayList subList(int startIndex, int lastIndex)** - Returns an ArrayList that contains elements from specified startIndex to lastIndex-1 from the invoking ArrayList.
- **int indexOf(E element)** - Returns the index value of given element first occurrence in the ArrayList.
- **int lastIndexOf(E element)** - Returns the index value of given element last occurrence in the ArrayList.

Updating Items

The ArrayList class has the following methods to update or change items.

- **E set(int index, E newElement)** - Replace the element at specified index with new Element in the invoking ArrayList.

Removing Items

The ArrayList class has the following methods to remove items.

- **E remove(int index)** - Removes the element at specified index in the invoking ArrayList.
- **boolean remove(Object element)** - Removes the first occurrence of the given element from the invoking ArrayList.
- **boolean removeAll(Collection c)** - Removes the given collection of elements from the invoking ArrayList.
- **void retainAll(Collection c)** - Removes all the elements except the given collection of elements from the invoking ArrayList..
- **void clear( )** - Removes all the elements from the ArrayList.

Example

```
import java.util.ArrayList;
class JavaExample1
{
        public static void main(String args[])
        {
                  ArrayList<String> arrList=new ArrayList<>();

                 //adding few elements

                  arrList.add("Cricket");        //list: ["Cricket"]
                  arrList.add("Hockey");         //list: ["Cricket", "Hockey"]
                  arrList.add(0, "BasketBall");    //list: ["BasketBall", "Cricket", "Hockey"]

                  System.out.println("ArrayList Elements: ");

                  /*  Traversing ArrayList using enhanced for loop
                                for(String str:arrList)

                               System.out.println(str);        */
                    for(int i=0;i<arrList.size();i++)
                    {
                               System.out.println(arrList(i));
                    }
        }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac JavaExample1.java

D:\ACEM\II  AI   DS>java JavaExample1

ArrayList Elements:

BasketBall

Cricket

Hockey
```

4.9 Linked List

- The **LinkedList** class is a part of java collection framework. It is available inside the **java.util** package.
- The LinkedList class extends **AbstractSequentialList** class and implements **List** and **Deque** interface.
- The elements of LinkedList are organized as the elements of linked list data structure.
- The LinkedList class is used to create a **dynamic list** of elements that can **grow or shrunk** as needed.
- The LinkedList is a child class of **AbstractSequentialList**
- The LinkedList implements interfaces like **List**, **Deque**, **Cloneable**, and **Serializable**.
- The LinkedList **allows to store duplicate data values**.
- The LinkedList maintains the order of insertion.

LinkedList class declaration

public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable

Example

```
import java.util.LinkedList;
import java.util.Iterator;
public class JavaExample2
{
	public static void main(String args[])
	{
		LinkedList<String> linkList=new LinkedList<>();

		linkList.add("Apple");		//["Apple"]
		linkList.add("Orange");		//["Apple", "Orange"]

		//inserting element at first position

		linkList.add(0, "Banana");		//["Banana", "Apple", "Orange"]

		System.out.println("LinkedList elements: ");
```

```
                //iterating LinkedList using iterator

                Iterator<String> it=linkList.iterator();

                        while(it.hasNext())
                         {
                                System.out.println(it.next());
                        }
        }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac JavaExample2.java

D:\ACEM\II  AI   DS>java JavaExample2

LinkedList elements:

Banana

Apple

Orange
```

4.10 Hash Set

- The HashSet class extends **AbstractSet** class and implements **Set** interface.
- The elements of HashSet are organized using a mechanism called hashing.
- The HashSet is used to create hash table for storing set of elements.
- The HashSet class is used to create a collection that uses a hash table for storing set of elements.
- The HashSet does not allows to store duplicate data values, but null values are allowed.
- The HashSet does not maintains the order of insertion.
- The HashSet initial capacity is 16 elements.
- The HashSet is best suitable for search operations.

HashSet class declaration

```
        public class HashSet<E> extends AbstractSet<E> implements Set<E>, Cloneable, Serializable
```

Program:

```
        import java.util.HashSet;
        import java.util.Iterator;

        public class JavaExample3
        {
                public static void main(String args[])
                {
                        HashSet<String> set=new HashSet<>();
```

```
                    set.add("honey cake");
                    set.add("stawberry");
                    set.add("vennela cake");
                    set.add("ice cake");

                   Iterator<String> it=set.iterator();

                   while(it.hasNext())
                   {
                           System.out.println(it.next());
                    }
           }
      }
```

Output:

```
D:\ACEM\II  AI   DS>javac JavaExample3.java
D:\ACEM\II  AI   DS>java JavaExample3
ice cake
stawberry
honey cake
vennela cake
```

4.11 Tree Set

- The **TreeSet** class is a part of java collection framework. It is available inside the **java.util** package.
- The TreeSet class extends **AbstractSet** class and implements **NavigableSet**, **Cloneable**, and **Serializable** interfaces.
- The elements of TreeSet are organized using a mechanism called tree.
- The TreeSet class internally uses a TreeMap to store elements. The elements in a TreeSet are sorted according to their natural ordering.
- The TreeSet is a child class of **AbstractSet and** implements **NavigableSet**, **Cloneable** , **Serializable**.
- The TreeSet does not allows to store duplicate data values, but null values are allowed.
- The elements in a TreeSet are sorted according to their natural ordering.
- The TreeSet initial capacity is 16 elements and it is best suitable for search operations.

TreeSet class declaration

```
public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable, Serializable
```

Example:

```
import java.util.TreeSet;

import java.util.Iterator;

public class JavaExample4
{
        public static void main(String args[])
        {
                TreeSet<String> set=new TreeSet<>();

                set.add("airtel");
                set.add("idea");
                set.add("jio");
                set.add("voda phone");

                Iterator<String> it=set.iterator();

                while(it.hasNext())
                {
                        System.out.println(it.next());
                }
        }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac JavaExample4.java
D:\ACEM\II  AI   DS>java JavaExample4
airtel

idea

jio

voda phone
```

4.12 Priority Queue

- The **PriorityQueue** class is a part of java collection framework. It is available inside the **java.util** package.
- The PriorityQueue class extends **AbstractQueue** class and implements **Serializable** interface.
- The elements of PriorityQueue are organized as the elements of queue , but it does not follow FIFO.
- The PriorityQueue elements are organized based on the priority heap and maintains the order of insertion
- The PriorityQueue class is used to create a dynamic queue of elements that can grow or shrunk as needed.
- The PriorityQueue is a child class of **AbstractQueue and** implements **Serializable interface**
- The PriorityQueue allows to store duplicate data values, but not null values.

PriorityQueue class declaration

```
public class PriorityQueue<E> extends AbstractQueue<E> implements Serializable
```

Example

```
import java.util.*;

public class PriorityQueueExample1
{
        public static void main(String[] args)
        {
                PriorityQueue queue = new PriorityQueue();

                PriorityQueue anotherQueue = new PriorityQueue();


                queue.add(10);
                queue.add(20);
                queue.add(15);


                System.out.println("\nQueue is " + queue);

                anotherQueue.addAll(queue);


                System.out.println("\nanotherQueue is " + anotherQueue);

                anotherQueue.offer(25);


                System.out.println("\nanotherQueue is " + anotherQueue);

        }
}
```

Output:

```
        D:\sasi>javac PriorityQueueExample1.java

        D:\sasi>java PriorityQueueExample1

        Queue is [10, 20, 15]

        anotherQueue is [10, 20, 15]

        anotherQueue is [10, 20, 15, 25]
```

4.13 Array Deque

- The **ArrayDeque** class is a part of java collection framework. It is available inside the **java.util** package.
- The ArrayDeque class extends **AbstractCollection** class and implements **Deque**, **Cloneable**, and **Serializable** interfaces.
- The elements of ArrayDeque are organized as the elements of double ended queue data structure.
- The ArrayDeque is a special kind of array that grows and allows users to add or remove an element from both the sides of the queue.

- The ArrayDeque class is used to create a dynamic double ended queue of elements that can grow or shrunk as needed.
- The ArrayDeque is a child class of **AbstractCollection**
- The ArrayDeque implements interfaces like **Deque**, **Cloneable**, and **Serializable**.
- The ArrayDeque allows to store duplicate data values, but not null values.
- The ArrayDeque maintains the order of insertion.
- The ArrayDeque allows to add and remove elements at both the ends.
- The ArrayDeque is faster than LinkedList and Stack.

ArrayDeque class declaration

public class ArrayDeque<E> extends AbstractCollection<E> implements Deque<E>, Cloneable, Serializable

Example:

```
import java.util.Deque;
import java.util.ArrayDeque;
public class TestJavaCollection6
{
        public static void main(String[] args)
        {
                //Creating Deque and adding elements

                Deque<String> deque = new ArrayDeque<String>();

                deque.add("Gautam");
                deque.add("Karan");
                deque.add("Ajay");

                //Traversing elements

                for (String str : deque)
                {
                        System.out.println(str);
                }
        }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac TestJavaCollection6.java
D:\ACEM\II  AI   DS>java TestJavaCollection6
Gautam
Karan
Ajay
```

4.14 Hashtable

- In java, the package **java.util** contains a class called **Hashtable** which works like a HashMap but it is synchronized.
- The Hashtable is a concrete class of Dictionary.
- It is used to store and manage elements in the form of a pair of key and value.
- The Hashtable stores data as a pair of key and value. In the Hashtable, each key associates with a value.
- Any non-null object can be used as a key or as a value. We can use the key to retrieve the value back when needed.
- The Hashtable class is no longer in use, it is obsolete. The alternate class is HashMap.
- The Hashtable class is a concrete class of Dictionary.
- The Hashtable class is synchronized.
- The Hashtable does no allow null key or value and Hashtable has the initial default capacity 11.

Example:

```
import java.util.Hashtable;
import java.util.Map;
class Hashtable1
{
         public static void main(String args[])
        {
                 Hashtable<Integer,String> hm=new Hashtable<Integer,String>();

                hm.put(100,"Amit");
                hm.put(102,"Ravi");
                hm.put(101,"Vijay");
                hm.put(103,"Rahul");
                for(Map.Entry m:hm.entrySet())
                {
                        System.out.println(m.getKey()+" "+m.getValue());
                }
        }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac Hashtable1.java
D:\ACEM\II  AI   DS>java Hashtable1
        103 Rahul
        102 Ravi
        101 Vijay
        100 Amit
```

4.15 Properties

- In java, the package **java.util** contains a class called **Properties** which is a child class of Hashtable class.
- It implements interfaces like Map, Cloneable, and Serializable.
- The Properties class used to store configuration values managed as key, value pairs. In each pair, both key and value are String values. We can use the key to retrieve the value back when needed.
- The Properties class provides methods to get data from the properties file and store data into the properties file. It can also be used to get the properties of a system.
- The Properties class is child class of Hashtable class.
- The Properties class implements Map, Cloneable, and Serializable interfaces.
- The Properties class used to store configuration values.
- The Properties class stores the data as key, value pairs.
- In Properties class both key and value are String data type.
- Using Properties class, we can load key, value pairs into a Properties object from a stream.
- Using Properties class, we can save the Properties object to a stream.

Example: **db.properties (save in separate notepad)**

```
user=system
password=oracle
```

Program:

```
import java.util.*;
import java.io.*;
public class TestProperties
{
    public static void main(String[] args)throws Exception
    {
            FileReader reader=new FileReader("db.properties");
            Properties p=new Properties();
            p.load(reader);
            System.out.println(p.getProperty("user"));
            System.out.println(p.getProperty("password"));
    }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac TestProperties.java
D:\ACEM\II  AI   DS>java TestProperties
system
oracle
```

4.16 Stack

- In java, the package **java.util** contains a class called **Stack** which is a child class of Vector class.
- It implements the standard principle Last-In-First-Out of stack data structure.

The Stack has push method for insertion and pop method for deletion. It also has other utility methods.

In Stack, the elements are added to the top of the stack and removed from the top of the stack.

Example

```
import java.util.*;
public class StackClassExample
{
        public static void main(String[] args)
        {
                Stack stack = new Stack();
                Random num = new Random();
                for(int i = 0; i < 5; i++)
                        stack.push(num.nextInt(100));

                System.out.println("Stack elements => " + stack);
                System.out.println("Top element is " + stack.peek());
                System.out.println("Removed element is " + stack.pop());
                System.out.println("Element 50 availability => " + stack.search(50));
                System.out.println("Stack is empty? - " + stack.isEmpty());
        }
}
```

Output:

```
D:\SASI\sasijava>javac StackClassExample.java
Note: StackClassExample.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\SASI\sasijava>java StackClassExample
        Stack elements => [14, 42, 15, 14, 19]
        Top element is 19
        Removed element is 19
        Element 50 availability => -1
        Stack is empty? - false
```

4.17 Vector

- In java, the package **java.util** contains a class called **Vector** which implements the **List** interface.
- The Vector is similar to an ArrayList. Like ArrayList Vector also maintains the insertion order. But Vector is synchronized, due to this reason, it is rarely used in the non-thread application. It also lead to poor performance.
- The Vector is a class in the java.util package.
- The Vector implements List interface.
- The Vector is a legacy class.
- The Vector is synchronized.

Example:

```
import java.util.*;
public class VectorExample
{
      public static void main(String args[])
    {

         Vector<String> vec = new Vector<String>();

         //Adding elements using add() method of List

         vec.add("Tiger");
         vec.add("Lion");
         vec.add("Dog");
         vec.add("Elephant");

         //Adding elements using addElement() method of Vector

         vec.addElement("Rat");
         vec.addElement("Cat");
         vec.addElement("Deer");

         System.out.println("Elements are: "+vec);
      }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac VectorExample.java

D:\ACEM\II  AI   DS>java VectorExample

Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]
```

4.18 String Tokenizer

- The StringTokenizer is a built-in class in java used to break a string into tokens. The StringTokenizer class is available inside the java.util package.
- The StringTokenizer class object internally maintains a current position within the string to be tokenized.
- A token is returned by taking a substring of the string that was used to create the StringTokenizer object.

Example:

```
import java.util.StringTokenizer;

public class StringTokenizerExample
{

        public static void main(String args[])
        {
                StringTokenizer st = new StringTokenizer("my name is sachin"," ");

                while (st.hasMoreTokens())
                {
                        System.out.println(st.nextToken());
                  }
  }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac StringTokenizerExample.java
D:\ACEM\II  AI   DS>java StringTokenizerExample
my
name
is
sachin
```

4.19 Bit Set

- The BitSet is a built-in class in java used to create a dynamic array of bits represented by boolean values. The BitSet class is available inside the java.util package.
- The BitSet array can increase in size as needed. This feature makes the BitSet similar to a Vector of bits.
- The bit values can be accessed by non-negative integers as an index.
- The size of the array is flexible and can grow to accommodate additional bit as needed.
- The default value of the BitSet is boolean false with a representation as 0 (off).
- BitSet uses 1 bit of memory per each boolean value.

Example:

```
import java.util.BitSet;

public class BitSetAndExample1
{
   public static void main(String[] args)
  {
        // create 2 bitsets

        BitSet bitset1 = new BitSet();
        BitSet bitset2 = new BitSet();

        // assign values to bitset1

        bitset1.set(0);
        bitset1.set(1);
        bitset1.set(2);
        bitset1.set(3);
        bitset1.set(4);

        // assign values to bitset2

        bitset2.set(2);
        bitset2.set(4);
        bitset2.set(6);
        bitset2.set(8);
        bitset2.set(10);

        // print the sets

        System.out.println("bitset1: " + bitset1);
        System.out.println("bitset2: " + bitset2);

        // perform and operation between two bitsets

        bitset1.and(bitset2);

        // print the new bitset1

        System.out.println("result bitset: " + bitset1);
   }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac BitSetAndExample1.java

D:\ACEM\II  AI   DS>java BitSetAndExample1

bitset1: {0, 1, 2, 3, 4}

bitset2: {2, 4, 6, 8, 10}

result bitset: {2, 4}
```

4.20 Date

- The **Date** is a built-in class in java used to work with date and time in java. The Date class is available inside the **java.util** package. The Date class represents the date and time with millisecond precision.
- The Date class implements **Serializable**, **Cloneable** and **Comparable** interface.
- Most of the constructors and methods of Date class has been deprecated after Calendar class introduced.

Example:

```
import java.time.LocalDate;
public class LocalDateExample1
{
        public static void main(String[] args)
        {
                LocalDate date = LocalDate.now();
                LocalDate yesterday = date.minusDays(1);
                 LocalDate tomorrow = yesterday.plusDays(2);


                System.out.println("Today date: "+date);
                System.out.println("Yesterday date: "+yesterday);
                 System.out.println("Tomorrow date: "+tomorrow);
         }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac LocalDateExample1.java
D:\ACEM\II  AI   DS>java LocalDateExample1
        Today date: 2022-12-25
        Yesterday date: 2022-12-24
        Tomorrow date: 2022-12-26
```

4.21 Calendar

- The **Calendar** is a built-in abstract class in java used to convert date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc.
- The Calendar class is available inside the **java.util** package.
- The Calendar class implements **Serializable**, **Cloneable** and **Comparable** interface.
- As the Calendar class is an abstract class, we can not create an object using it.
- We will use the static method **Calendar.getInstance()** to instantiate and implement a sub-class.

Example:

```
import java.util.Calendar;
public class CalendarExample1
{
        public static void main(String[] args)
        {
                Calendar calendar = Calendar.getInstance();
                System.out.println("The current date is : " + calendar.getTime());

               calendar.add(Calendar.DATE, -15);
               System.out.println("15 days ago: " + calendar.getTime());

               calendar.add(Calendar.MONTH, 4);
               System.out.println("4 months later: " + calendar.getTime());

               calendar.add(Calendar.YEAR, 2);
               System.out.println("2 years later: " + calendar.getTime());
  }
}
```

Output:

```
        D:\ACEM\II  AI   DS>javac CalendarExample1.java
        D:\ACEM\II  AI   DS>java CalendarExample1
                The current date is : Sun Dec 25 22:08:47 IST 2022
                15 days ago: Sat Dec 10 22:08:47 IST 2022
                4 months later: Mon Apr 10 22:08:47 IST 2023
                2 years later: Thu Apr 10 22:08:47 IST 2025
```

4.22 Random

The **Random** is a built-in class in java used to generate a stream of pseudo-random numbers in java programming. The Random class is available inside the **java.util** package.

The Random class implements **Serializable**, **Cloneable** and **Comparable** interface.

- The **Random** class is a part of java.util package.
- The **Random** class provides several methods to generate random numbers of type integer, double, long, float etc.
- The **Random** class is thread-safe.
- Random number generation algorithm works on the seed value. If not provided, seed value is created from system nano time.

Example:

```
import java.util.Random;
public class JavaRandomExample
 {
        public static void main(String[] args)
        {
                        Random random = new Random();

                        //return the next pseudorandom integer value
                        System.out.println("Random Integer value : "+random.nextInt());

                        // setting seed

                        long seed =20;
                        random.setSeed(seed);

                        //value after setting seed
                        System.out.println("Seed value : "+random.nextInt());


                        //return the next pseudorandom long value

                         Long val = random.nextLong();
                         System.out.println("Random Long value : "+val);
  }

}
```

Output:

```
D:\ACEM\II  AI   DS>javac JavaRandomExample.java
D:\ACEM\II  AI   DS>java JavaRandomExample
        Random Integer value : -1652727993
        Seed value : -1150867590
        Random Long value : -7322354119883315205
```

4.23 Formatter class

- The **Formatter** is a built-in class in java used for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output in java programming.
- The Formatter class is defined as final class inside the **java.util** package.
- The Formatter class implements **Cloneable** and **Flushable** interface.
- The Formatter class in java has the following constructors.

Example:

```
import java.util.Formatter;
public class FomatterExample
{
        public static void main(String args[]) throws Exception
        {
                int num[] = {10, 21, 13, 4, 15, 6, 27, 8, 19};

                Formatter fmt = new Formatter();

                fmt.format("%15s %15s %15s\n", "Number", "Square", "Cube");

                for (int n : num)
                {
                        fmt.format("%14s %14s %17s\n", n, (n*n), (n*n*n));
                }
                System.out.println(fmt);

        }

}
```

Output:

```
D:\ACEM\II  AI   DS>javac FomatterExample.java
D:\ACEM\II  AI   DS>java FomatterExample
      Number        Square         Cube
        10         100           1000
        21         441           9261
        13         169           2197
         4         16            64
        15         225           3375
         6         36           216
        27         729          19683
         8         64           512
        19         361           6859
```

4.24 Scanner

The **Scanner** is a built-in class in java used for read the input from the user in java programming.

The Scanner class is defined inside the **java.util** package and implements **Iterator** interface.

- The Scanner class provides the easiest way to read input in a Java program.
- The Scanner object breaks its input into tokens using a delimiter pattern, the default delimiter is whitespace.

Example:

```
import java.util.*;
public class ScannerClassExample1
 {
        public static void main(String args[])
        {
                Scanner in = new Scanner(System.in);

                System.out.print("Enter your name: ");
                String name = in.next();
                System.out.println("Name: " + name);


                System.out.print("Enter your age: ");
                int i = in.nextInt();
               System.out.println("Age: " + i);


               System.out.print("Enter your salary: ");
               double d = in.nextDouble();
                System.out.println("Salary: " + d);

                in.close();
         }
}
```

Output:

```
D:\ACEM\II  AI   DS>javac ScannerClassExample1.java
D:\ACEM\II  AI   DS>java ScannerClassExample1
        Enter your name: Aditya
        Name: Aditya
        Enter your age: 32
        Age: 32
        Enter your salary: 35000
        Salary: 35000.0
```