Collections

Limitations of array:

- Array is indexed collection o fixed number of homogeneous data elements
- Arrays can hold homogeneous data only
- Once we created an array no chance of increasing o decreasing size of array

Ex:

```
Student[ ] s=new Student[100];
   S[0]=new Student();
   S[1]=new Student();
   S[2]=new Customer();  compilation error
```

- To overcome the above limitations of array the sun peoples are introduced collections concept

- To overcome the limitations in Array we should go for collections concept.

- Collections are growable in nature that is based on our requirement we can increase (or) decrease the size hence memory point of view collections concept is recommended to use.

- Collections can hold both homogeneous and heterogeneous objects.

- Every collection class is implemented based on some standard data structure

- Hence for every requirement ready-made method support is available being a programmer we can use these methods directly without writing the functionality on our own

Collections:

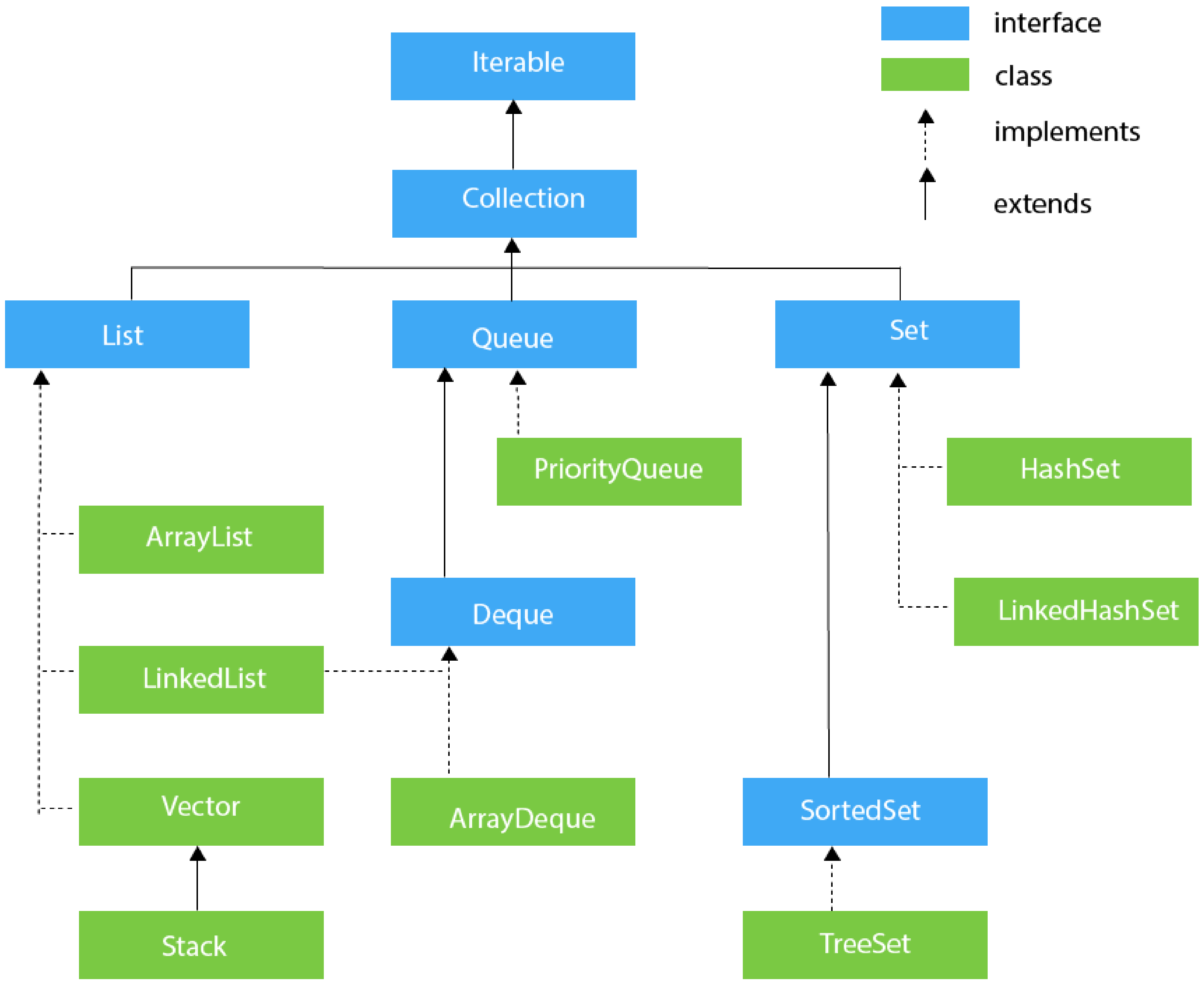
- ➢collection can hold both homogeneous data and heterogeneous data
- ➢collections are growable in nature
- ➢Memory wise collections are good. Recommended to use.
- ➢Performance wise collections are not recommended to use .

Arrays	Collections
1) Arrays are fixed in size.	1) Collections are growable in nature.
2) Memory point of view arrays are not recommended to use.	2) Memory point of view collections are highly recommended to use.
3) Performance point of view arrays are recommended to use.	3) Performance point of view collections are not recommended to use.
4) Arrays can hold only homogeneous data type elements.	4) Collections can hold both homogeneous and heterogeneous elements.
5) There is no underlying data structure for arrays and hence there is no readymade method support.	5) Every collection class is implemented based on some standard data structure and hence readymade method support is available.
6) Arrays can hold both primitives and object types.	6) Collections can hold only objects but not primitives.

Introduction to Collections Framework

- “The Collections Framework provides a well-designed set of interfaces and classes for storing and manipulating groups of data as a single unit, a collection.” -java.sun.com
- The standard data structure in Java can be implemented in Java using some library classes and methods. These classes are present in **java.util** package.
- The collection framework is comprised of collection classes and collection interfaces.
- Collection is a group of objects which are designed to perform certain task. These tasks are associated with data structures.

- The collection classes are the group of classes used to implement the collection interfaces. Various collection classes are…
  - LinkedList
  - ArrayList
  - AbstractSet
  - EnumSet
  - HashSet
  - PriorityQueue
  - TreeSet
  - Vector
  - HashTable ..etc


interface
class
implements
extends
Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet

Collection - interface

- If we want to represent a group of "individual objects" as a single entity then we should go for collection.

- In general we can consider collection as root interface of entire collection framework.

- Collection interface defines the most common methods which can be applicable for any collection object.

- There is no concrete class which implements Collection interface directly.

Collection Interface Methods

Method	Description
add(Object)	This method is used to add an object to the collection.
addAll(Collection c)	This method adds all the elements in the given collection to this collection.
clear()	This method removes all of the elements from this collection.
contains(Object o)	This method returns true if the collection contains the specified element.
containsAll(Collection c)	This method returns true if the collection contains all of the elements in the given collection.
equals(Object o)	This method compares the specified object with this collection for equality.
hashCode()	This method is used to return the hash code value for this collection.
isEmpty()	This method returns true if this collection contains no elements.
iterator()	This method returns an iterator over the elements in this collection.
max()	This method is used to return the maximum value present in the collection.
parallelStream()	This method returns a parallel Stream with this collection as its source.
remove(Object o)	This method is used to remove the given object from the collection. If there are duplicate values, then this method removes the first occurrence of the object.
removeAll(Collection c)	This method is used to remove all the objects mentioned in the given collection from the collection.
removeIf(Predicate filter)	This method is used to remove all the elements of this collection that satisfy the given predicate.
retainAll(Collection c)	This method is used to retain only the elements in this collection that are contained in the specified collection.
size()	This method is used to return the number of elements in the collection.
spliterator()	This method is used to create a Spliterator over the elements in this collection.
stream()	This method is used to return a sequential Stream with this collection as its source.
toArray()	This method is used to return an array containing all of the elements in this collection.

List -interface

- It is the child interface of Collection.
- If we want to represent a group of individual objects as a single entity where "duplicates are allow and insertion order must be preserved" then we should go for List interface.
- We can differentiate duplicate objects and we can maintain insertion order by means of index hence "index play very important role in List“
- All list interface methods build based on index.

Methods in List interface

- ❖boolean add(int index,Object o);
- ❖boolean addAll(int index,Collectio c);
- ❖Object get(int index);
- ❖Object remove(int index);
- ❖Object set(int index,Object new);//to replace
- ❖Int indexOf(Object o);
- ❖Returns index of first occurrence of "o".
- ❖Int lastIndexOf(Object o);
- ❖ListIterator listIterator();

ArrayList

ArrayList

- Introduced in 1.2 version.
- ArrayList supports dynamic array that can be grow as needed.it can dynamically increase and decrease the size.
- Duplicate objects are allowed.
- Null insertion is possible.
- Heterogeneous objects are allowed.
- The under laying data structure is growable array.
- Insertion order is preserved.

Constructors:

1) ArrayList a=new ArrayList();

Creates an empty ArrayList object with default initial capacity "10" if ArrayList reaches its max capacity then a new ArrayList object will be created with

New capacity=(current capacity*3/2)+1

- 2)ArrayList a=new ArrayList(int initialcapacity);

Creates an empty ArrayList object with the specified initial capacity

Vector

Vector

- ➢The underlying data structure is resizable array (or) growable array.
- ➢Duplicate objects are allowed.
- ➢Insertion order is preserved.
- ➢Heterogeneous objects are allowed.
- ➢Null insertion is possible.
- ➢Implements Serializable, Cloneable and RandomAccess interfaces.
- ➢Every method present in Vector is synchronized and hence Vector is Thread safe.

Vector specific methods:

To add objects:

- ❖add(Object o);-----Collection
- ❖add(int index,Object o);-----List
- ❖addElement(Object o);-----Vector

To remove elements:

- ❖remove(Object o);--------Collection
- ❖remove(int index);--------------List
- ❖removeElement(Object o);----Vector
- ❖removeElementAt(int index);-----Vector
- ❖removeAllElements();-----Vector
- ❖clear();-------Collection

To get objects:

- ❖Object get(int index);---------------List
- ❖Object elementAt(int index);-----Vector
- ❖Object firstElement();--------------Vector
- ❖Object lastElement();---------------Vector

Constructors:

1) Vector v=new Vector();

- Creates an empty Vector object with default initial capacity 10.
- Once Vector reaches its maximum capacity then a new Vector object will be created with double capacity.
- That is "newcapacity=currentcapacity*2"

- ➢2) Vector v=new Vector(int initialcapacity);
- ➢3) Vector v=new Vector(int initialcapacity, int incrementalcapacity);
- ➢4) Vector v=new Vector(Collection c);

Stack

Stack

- It is the child class of Vector.
- Whenever last in first out(LIFO) order required then we should go for Stack.

Constructor:

- It contains only one constructor.

```
Stack s= new Stack();
```

Methods:

- **Object push(Object o)**

  To insert an object into the stack.
- **Object pop()**

  To remove and return top of the stack.
- **Object peek()**

  To return top of the stack without removal.
- **boolean empty()**

  Returns true if Stack is empty.

```java
import java.util.Stack;

public class StackDemo
{
    public static void main(String[] args) {
        Stack s=new Stack();
        System.out.println("Elements in stack :"+s);
        s.push("A");
        s.push("B");
        s.push("C");
        s.push("D");
        s.push("E");
        System.out.println("Elements in stack :"+s);
        s.pop();
        System.out.println("Elements in stack :"+s);
        System.out.println("Top element in stack :"+s.peek());
        System.out.println("Search A element in stack :"+s.search("A"));
        System.out.println("Search F element in stack :"+s.search("F"));
        System.out.println("Search A element in stack :"+s);

    }
}
```

Output:

```
Elements in stack :[]
Elements in stack :[A, B, C, D, E]
Elements in stack :[A, B, C, D]
Top element in stack :D
Search A element in stack :4
Search F element in stack :-1
Search A element in stack :[A, B, C, D]
```

```java
import java.util.Scanner;
import java.util.Stack;
public class Ex5
{
    public static void main(String[] args) {
        Stack<Integer> st=new Stack<Integer>();
        int choice=0;
        int position;
        Scanner scr=new Scanner(System.in);
        while(true)    {
            System.out.println("Stack Operations");
            System.out.println("1.Push an element");
            System.out.println("2.Display stack");
            System.out.println("3.Pop an element");
            System.out.println("4.Search an element");
            System.out.println("Enter your choice");
            choice=scr.nextInt();
            switch(choice)  {
                case 1:
                    System.out.println("Enter an element");
                    Integer i=scr.nextInt();
                    st.push(i);
                   break;
                case 2:
                    System.out.println("Elements in stack :"+st);
                    break;
                case 3:
                    System.out.println("Top element popped..");
                    Integer obj = st.pop();
                    System.out.println("Popped element= "+obj);
                    break;
                case 4:
                    System.out.println("Which an element ? ");
                    Integer ele=scr.nextInt();
                    position = st.search(ele);
                    if(position==-1)
                        System.out.println("Element not found");
                    else
                        System.out.println("Position of the element is:= "+position);
                    break;
                default:
                    return;     }    }    }
}
```

Output:

```
Stack Operations
1.Push an element
2.Display stack
3.Pop an element
4.Search an element
Enter your choice
1
Enter an element
10
Stack Operations
1.Push an element
2.Display stack
3.Pop an element
4.Search an element
Enter your choice
1
Enter an element
20
Stack Operations
1.Push an element
2.Display stack
3.Pop an element
4.Search an element
Enter your choice
2
Elements in stack :[10, 20]
Stack Operations
1.Push an element
2.Display stack
3.Pop an element
4.Search an element
Enter your choice
```

LinkedList:

- The underlying data structure is double LinkedList.
- If our frequent operation is insertion (or) deletion in the middle then LinkedList is the best choice.
- If our frequent operation is retrieval operation then LinkedList is worst choice.
- Duplicate objects are allowed.
- Insertion order is preserved.
- Heterogeneous objects are allowed.
- Null insertion is possible.

Methods in LinkedList

- ❖void addFirst(Object o);
- ❖void addLast(Object o);
- ❖Object getFirst();
- ❖Object getLast();
- ❖Object removeFirst();
- ❖Object removeLast();

Constructors:

- LinkedList l=new LinkedList();

  Creates an empty LinkedList object.

- LinkedList l=new LinkedList(Collection c);

  To create an equivalent LinkedList object for the given collection.

Cursors

The 3 cursors of java:

If we want to get objects one by one from the collection then we should go for cursor.

There are 3 types of cursors available in java. They are:

1. Enumeration
2. Iterator
3. ListIterator

Enumeration:

- We can use Enumeration to get objects one by one from the legacy collection objects.
- We can create Enumeration object by using elements() method.
  - public Enumeration elements();
  - Enumeration e=v.elements();

Enumeration interface defines the following two methods

- ➢public boolean hasMoreElements();
- ➢public Object nextElement();

Example:

```
import java.util.*;
class  EnumerationDemo
{
    public static void main(String[] args)
    {
        Vector v=new Vector();
        for(int i=0;i<=10;i++)
        {
            v.addElement(i);
        }
        System.out.println(v);//[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        Enumeration e=v.elements();
        while(e.hasMoreElements())
        {
            Integer i=(Integer)e.nextElement();
            if(i%2==0)
                System.out.println(i);//0 2 4 6 8 10
        }
    }
}
```

Limitations of Enumeration:

- ➢We can apply Enumeration concept only for legacy classes and it is not a universal cursor.
- ➢By using Enumeration we can get only read access and we can't perform remove operations.
- ➢To overcome these limitations they introduced Iterator concept in 1.2v.

Iterator:

- We can use Iterator to get objects one by one from any collection object.
- We can apply Iterator concept for any collection object and it is a universal cursor.
- While iterating the objects by Iterator we can perform both read and remove operations.
- We can get Iterator object by using iterator() method of Collection interface.

```
public Iterator iterator();
```

Ex: `Iterator itr=c.iterator();`

Iterator interface defines the following 3 methods.

- ❖public boolean hasNext();
- ❖public object next();
- ❖public void remove();

Ex:

```
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorEx
{
    public static void main(String[] args) {
        ArrayList al=new ArrayList();
        for (int i=0;i<100;i++)
        {
            al.add(i);
        }
        System.out.println("ArrayList Elements:"+al);
        Iterator i=al.iterator();

        while (i.hasNext())
        {
            int n=(int)i.next();
            if(n%2!=0)
                i.remove();
        }
        System.out.println("Even ArrayList
Elements:"+al);

    }
}
```

Output:

```
ArrayList Elements:[0, 1, 2, 3, 4, 5, 6, 7, 8, .. 99]
Even ArrayList Elements:[0, 2, 4, 6, 8, 10,   ..98]
```

Limitations of Iterator:

- ➢Both enumeration and Iterator are single direction cursors only. That is we can always move only forward direction and we can't move to the backward direction.
- ➢While iterating by Iterator we can perform only read and remove operations and we can't perform replacement and addition of new objects.
- ➢To overcome these limitations sun people introduced listIterator concept

Enumeration	Iterator
Introduced in Java 1.0	Introduced in Java 1.2
Legacy Interface	Not Legacy Interface
It is used to iterate only Legacy Collection classes.	We can use it for any Collection class.
It supports only READ operation.	It supports both READ and DELETE operations.
It’s not Universal Cursor.	It is a Universal Cursor.
Lengthy Method names.	Simple and easy-to-use method names.

ListIterator:

- ListIterator is the child interface of Iterator.
- By using listIterator we can move either to the forward direction (or) to the backward direction that is it is a bi-directional cursor.
- While iterating by listIterator we can perform replacement and addition of new objects in addition to read and remove operations

By using listIterator method we can create listIterator object.

```
public ListIterator listIterator();
ListIterator itr=l.listIterator();
```

ListIterator interface defines the following 9 methods.

- ❖public boolean hasNext();
- ❖public Object next(); forward
- ❖public int nextIndex();
- ❖public boolean hasPrevious();
- ❖public Object previous(); backward
- ❖public int previousIndex();
- ❖public void remove();
- ❖public void set(Object new);
- ❖public void add(Object new);

Ex:

```java
import java.util.LinkedList;
import java.util.ListIterator;
public class MyLinkedList
{
    public static void main(String[] args)
    {
        LinkedList<String> ll=new LinkedList<String>();
        ll.add("A");
        ll.add("B");
        ll.add("C");
        ll.add("D");
        ll.add("E");
        ListIterator li=ll.listIterator();
        System.out.println("LinkedList elements in forward direction..");
        while (li.hasNext())
        {
            System.out.println(li.next());
        }
        System.out.println("LinkedList elements in backward direction..");
        while (li.hasPrevious())
        {
            System.out.println(li.previous());
        }
    }
}
```

Output:

```
LinkedList elements in forward direction..
A
B
C
D
E
LinkedList elements in backward direction..
E
D
C
B
A
```

Comparison of Enumeration Iterator and ListIterator ?

Property	Enumeration	Iterator	ListIterator
1) Is it legacy ?	Yes	no	no
2) It is applicable for ?	Only legacy classes.	Applicable for any collection object.	Applicable for only list objects.
3) Moment?	Single direction cursor(forward)	Single direction cursor(forward)	Bi-directional.
4) How to get it?	By using elements() method.	By using iterator()method.	By using listIterator() method.
5) Accessibility?	Only read.	Both read and remove.	Read/remove/replace/a dd.

6) Methods	hasMoreElement()
	nextElement()
		hasNext()
		next()
		remove()
			9 methods.

Generic classes

Generic Class

- JDK 1.5 introduces several extensions to the Java programming language. One of these is the introduction of generics.
- Using generics it is possible to create a single class that automatically works with different types of data.
- A Generic class simply means that the items or functions in that class can be generalized with the parameter(example T) to specify that we can add any type as a parameter in place of T like Integer, Character, String, Double or any other user-defined type.

Generics

Advantage of Java Generics:

There are mainly 3 advantages of generics. They are as follows

1) Type-safety:

- We can hold only a single type of objects in generics. It doesn’t allow to store other objects.

Ex:
```
List list = new ArrayList();
    list.add(10);
    list.add("10");
    With Generics, it is required to specify the type of object we need to store.
    List<Integer> list = new ArrayList<Integer>();
    list.add(10);
    list.add("10");// compile-time error
```

2) Type casting is not required: There is no need to typecast the object.

Ex:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);//typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

3) Compile-Time Checking:

- It will check the type at compile time so problem will not occur at runtime.
- The good programming strategy says it is far better to handle the problem at compile time than runtime.

Example:

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32);//Compile Time Error
```

Example:

```
import java.util.ArrayList;
import java.util.Collections;
public class SortNames
{
    public static void main(String[] args)
    {
        ArrayList<String> al=new ArrayList<String>();
        al.add("Bhaanu");
        al.add("Chandhu");
        al.add("Divya");
        al.add("Abhi");
        al.add("Eesha");
        System.out.println("Before sorting");
        System.out.println("---------------------");
        for (String name:al)
            System.out.println(name);
        Collections.sort(al);
        System.out.println("Before sorting");
        System.out.println("---------------------");
        for (String name:al)
            System.out.println(name);

    }
}
```

User defined Generic class

Example:

```
public class Ex1
{
    public static void main(String[] args)
    {
        MyClass<String> m1=new MyClass<String>("100");
        System.out.println("Generic class is returning :"+m1.getInfo());
        MyClass<Double> m2=new MyClass<Double>(10.25);
        System.out.println("Generic class is returning :"+m2.getInfo().getClass());
    }
}
class MyClass<T>
{
    T t;
    MyClass(T t)
    {
        this.t=t;
    }
    public T getInfo()
    {

        return t;
    }
}
```

Example:

```
public class Ex2
{
    public static void main(String[] args)
    {
        Show<Float> s=new Show<Float>(10.5f);
        System.out.println(s.getInfo());
        Show<Integer> s1=new Show<Integer>(10);
        System.out.println(s1.getInfo());
    }
}
class Show<T extends Number>
{
    T t;
    Show(T t)
    {
        this.t=t;
    }
    public T getInfo()
    {
        return t;
    }
}
```

Random class

Random class in Java?

- In Java, Random class is a part of **java.util** package.

- The generation of random numbers takes place by using an instance of the Java Random Class.

- This class provides different methods in order to produce random numbers of type boolean, integer, double, long, float, etc.

Constructors used in a Java Random class

This class contains two constructors that are mentioned below:

- ➢**Random():** this constructor helps in creating a new random generator
- ➢**Random(long seed):** this constructor helps in creating a new random generator using specified seed

Methods

Method	Functionality
nextDouble()	Returns the next pseudo-random number that is a double value between the range of 0.0 to 1.0.
nextBoolean()	Returns the next pseudo-random which is a Boolean value from random number generator sequence
nextFloat()	Returns the next pseudo-random which is a float value between 0.0 to 1.0
nextInt()	Returns the next pseudo-random which is an integer value from random number generator sequence
nextInt(Int n)	Returns the next pseudo-random which is an integer value between 0 and the specified value from random number generator sequence
nextBytes(byte[] bytes)	Generates random bytes and places them into a byte array supplied by the user
Longs()	Returns an unlimited stream of pseudorandom long values
nextGaussian()	Helps in returning the next pseudo-random, Gaussian (precisely) distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence

Example:

```
import java.util.Random;
public class RandomDemo
{
    public static void main(String[] args) {
        Random rnd=new Random();
        System.out.println("Random boolean :"+rnd.nextBoolean());
        byte b[]=new byte[10];
        rnd.nextBytes(b);
        System.out.print("Random bytes    : ");
        for (byte n:b)
        {
            System.out.print(n+" ");
        }
    }
}
```

Output:

```
Random boolean :false
Random bytes   : -36 74 119 70 -65 117 -58 127 121 -5
```

StringTokenizer

StringTokenizer

- The string tokenizer class allows an application to break a string into tokens.
- The tokenization method is much simpler than the one used by the StreamTokenizer class
- The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.
- An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the returnDelims flag having the value true or false:

StringTokenizer methods

int	countTokens()Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.
boolean	hasMoreElements()Returns the same value as the hasMoreTokens method.
boolean	hasMoreTokens()Tests if there are more tokens available from this tokenizer's string.
Object	nextElement()Returns the same value as the nextToken method, except that its declared return value is Object rather than String.
String	nextToken()Returns the next token from this string tokenizer.
String	nextToken(String delim)Returns the next token in this string tokenizer's string.

Example:

```
 StringTokenizer st = new StringTokenizer("this is a test");
 while (st.hasMoreTokens())
{
     System.out.println(st.nextToken());
 }
```

prints the following output:

```
 this
 is
 a
 test
```

- **StringTokenizer** is a legacy class that is retained for compatibility reasons although its use is discouraged in new code.

- It is recommended that anyone seeking this functionality use the split method of String or the java.util.regex package instead.

- The following example illustrates how the **String.split()** can be used to break up a string into its basic tokens:

Example:

```
String[] result = "this is a test".split("\\s");
for (int x=0; x<result.length; x++)
    System.out.println(result[x]);
```

Output:

```
this
is
a
test
```

Scanner Class

Scanner class

- ➢Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings.
- ➢It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint.
- ➢To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.
- ➢To read entire line of Strings we can use nextLine().
- ➢To read a single character, we use next().charAt(0). next() function returns the next token/word in the input as a string and charAt(0) function returns the first character in that string.

Methods

boolean hasNext( )
boolean hasNext(String pattern)
boolean hasNextBigInteger( )
boolean hasNextBoolean( )
boolean hasNextByte(int radix)
boolean hasNextFloat( )
boolean hasNextInt(int radix)
boolean hasNextLong( )
boolean hasNextShort( )
String next( )
boolean nextBoolean( )
double nextDouble( )
String nextLine( )
short nextShort( )

boolean hasNext(Pattern pattern)
boolean hasNextBigDecimal( )
boolean hasNextBigInteger(int radix)
boolean hasNextByte( )
boolean hasNextDouble( )
boolean hasNextInt( )
boolean hasNextLine( )
boolean hasNextLong(int radix)
boolean hasNextShort(int radix)
String next(String pattern)
byte nextByte( )
int nextInt( )
long nextLong()

Calendar class

Calendar class

- Calendar class in Java is an abstract class that provides methods for converting date between a specific instant in time and a set of calendar fields such as MONTH, YEAR, HOUR, etc.
- It inherits Object class and implements the Comparable, Serializable, Cloneable interfaces.
- As it is an Abstract class, so we cannot use a constructor to create an instance. Instead, we will have to use the static method **Calendar.getInstance()** to instantiate and implement a sub-class..
- java.util.Calendar is an abstract base class for extracting detailed information such as year, month, date, hour, minute and second from a Date object.
- Subclasses of Calendar can implement specific calendar systems such as Gregorian calendar, Lunar Calendar and Jewish calendar.
- Currently, java.util.GregorianCalendar for the Gregorian calendar is supported in the Java API.

Methods

- Calendar provides no public constructors.
- **static Calendar getInstance( ) :** Returns a Calendar object for the default locale and time zone.
- **int get(int calendarField) :** Returns the value of one component of the invoking object.
- The component is indicated by calendarField.
- Examples of the components that can be requested are Calendar.YEAR, Calendar.MONTH, Calendar.MINUTE, and so forth.

- **final Date getTime( ) :** Returns a Date object equivalent to the time of the invoking object.

- **final void set(int year, int month, int dayOfMonth) :** Sets various date and time components of the invoking object.

- **final void setTime(Date d) :** Sets various date and time components of the invoking object. This information is obtained from the Date object d.

METHOD	**DESCRIPTION**
abstract void add(int field, int amount)	It is used to add or subtract the specified amount of time to the given calendar field, based on the calendar’s rules.
int get(int field)	It is used to return the value of the given calendar field.
abstract int getMaximum(int field)	It is used to return the maximum value for the given calendar field of this Calendar instance.
abstract int getMinimum(int field)	It is used to return the minimum value for the given calendar field of this Calendar instance.
Date getTime()	It is used to return a Date object representing this Calendar’s time value.</td

Example:

```
import java.util.*;
public class Calendar1 {
    public static void main(String args[])
    {
        Calendar c = Calendar.getInstance();
        System.out.println("The Current Date is:" + c.getTime());
    }
}
```

Output:

```
The Current Date is:Tue Aug 28 11:10:40 UTC 2022
```

Example:

```
import java.util.*;
public class Calendar3 {
    public static void main(String[] args)
    {
        // creating calendar object
        Calendar calendar = Calendar.getInstance();

        int max = calendar.getMaximum(Calendar.DAY_OF_WEEK);
        System.out.println("Maximum number of days in a week: " + max);

        max = calendar.getMaximum(Calendar.WEEK_OF_YEAR);
        System.out.println("Maximum number of weeks in a year: " + max);
    }
}
```

Output:

```
Maximum number of days in a week: 7
Maximum number of weeks in a year: 53
```

Example:

```
import java.util.Calendar;

public class Ex20
{
    public static void main(String[] args)
    {
        String[]
month={"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"};
        Calendar calendar=Calendar.getInstance();
        System.out.println("Calendar Type:"+calendar.getCalendarType());
        System.out.println("Time Zone :"+calendar.getTimeZone().getID());
        System.out.print("Date:");
        System.out.print(month[calendar.get(Calendar.MONTH)]);
        System.out.print(" "+calendar.get(Calendar.DAY_OF_MONTH));
        System.out.print(" "+calendar.get(Calendar.YEAR));
    }

}
```

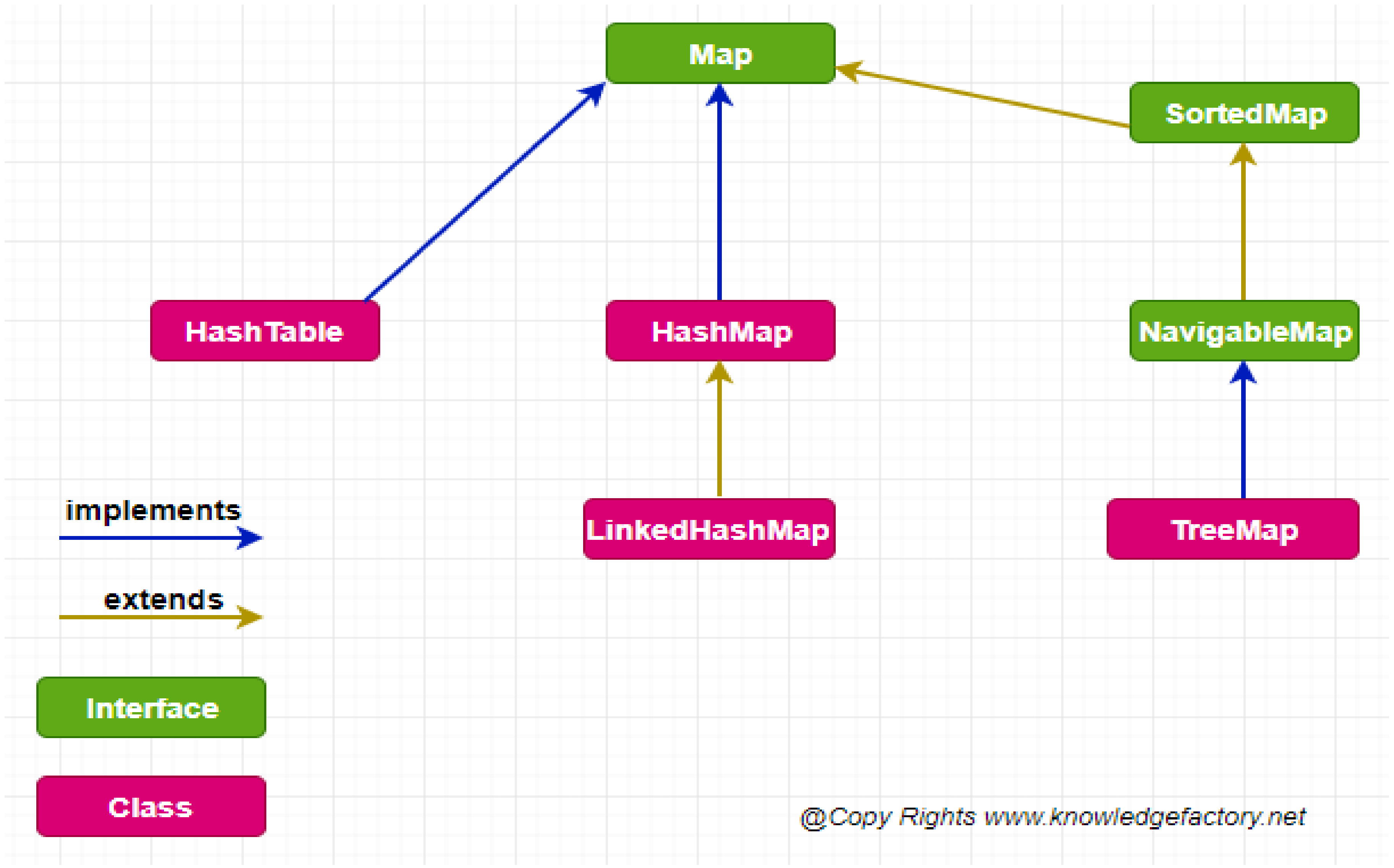
Output:

```
Calendar Type:gregory
Time Zone :Asia/Calcutta
Date:Jun 16 2022
```

MAP

Map properties

- ➢If we want to represent a group of objects as "key-value" pair then we should use Map interface.
- ➢Both key and value are objects only.
- ➢Duplicate keys are not allowed but values can be duplicated
- ➢Each key-value pair is called "one entry".
- ➢Map interface is not child interface of Collection and hence we can't apply Collection interface methods here.
- ➢Map interface defines the following specific methods.


Map
SortedMap
HashTable
HashMap
NavigableMap
LinkedHashMap
TreeMap
implements
extends
Interface
Class
@Copy Rights www.knowledgefactory.net

HashMap:

HashMap:

- The underlying data structure is Hashtable.
- Duplicate keys are **not allowed** but values can be duplicated.
- Insertion order is not preserved and it is based on hash code of the keys.
- Heterogeneous objects are allowed for both key and value.
- Null is allowed for keys(only once) and for values(any number of times).
- It is best suitable for **Search** operations.