This font is 600 weight.

This font is 700 weight.

This font is 800 weight.

This font is 900 weight.

Java script

JavaScript is *an object-based scripting language* that is lightweight and cross-platform.

JavaScript is not compiled but translated. The JavaScript Translator (embedded in browser) is responsible to translate the JavaScript code.

Where JavaScript is used

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation
- Dynamic drop-down menus
- Displaying data and time

- Displaying popup windows and dialog boxes (like alert dialog box, confirm dialog box and prompt dialog box)

Displaying clocks etc.

JavaScript Example

```
1. <h2>Welcome to JavaScript</h2>
2. <script>
3. document.write("Hello JavaScript by JavaScript");

   </script>
```

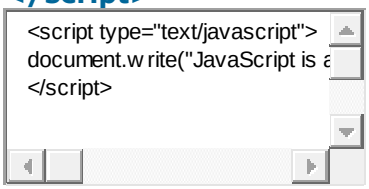
JavaScript Example

1. JavaScript Example
2. Within body tag
3. Within head tag

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let's create the first JavaScript example.

```
1. <script type="text/javascript">
2. document.write("JavaScript is a simple language for javatpoint learners");
3. </script>
```



Test it Now

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through

JavaScript. We will learn about document object in detail later.

---

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

---

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

```
<script type="text/javascript">
 alert("Hello Javatpoint");
</script>
```

Test it Now

---

2) JavaScript Example : code between the head tag

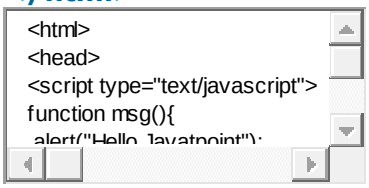
Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.

To call function, you need to work on event. Here we are using onclick event to

call msg() function.

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```



External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

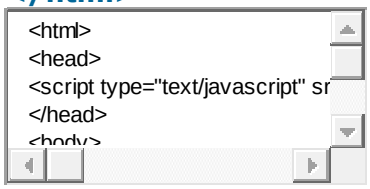
message.js

```
1. function msg(){
2.  alert("Hello Javatpoint");
3. }
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
1. <html>
2. <head>
3. <script type="text/javascript" src="message.js"></script>
4. </head>
5. <body>
6. <p>Welcome to JavaScript</p>
7. <form>
8. <input type="button" value="click" onclick="msg()"/>
9. </form>
10. </body>
11. </html>
```



Document Object Model

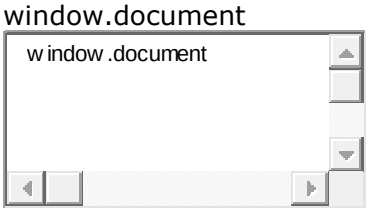
1. Document Object
2. Properties of document object
3. Methods of document object
4. Example of document object

The **document object** represents the whole html document.

When html document is loaded in the browser, it becomes a document object. It is the **root element** that represents the html document. It has properties and methods. By the help of document object, we can add dynamic content to our web page.

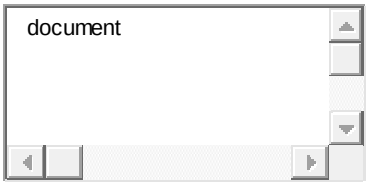
As mentioned earlier, it is the object of window. So

1. window.document



Is same as

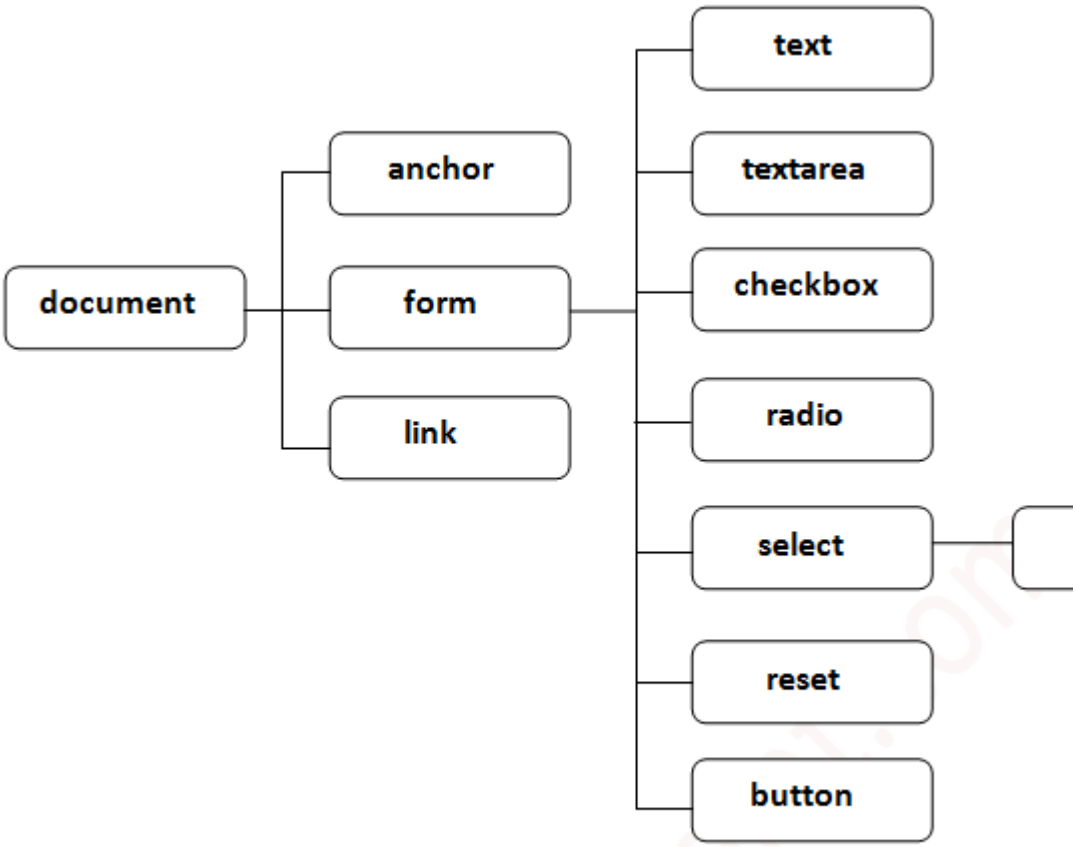
1. document



According to W3C - *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

Properties of document object

Let's see the properties of document object that can be accessed and modified by the document object.



Methods of document object

We can access and change the contents of document by its methods.

The important methods of document object are as follows:

Method	Description
write("string")	writes the given string on the doucment.
writeln("string")	writes the given string on the doucmer with newline character at the end.

getElementById()	returns the element having the given id value.
getElementsByName()	returns all the elements having the given name value.
getElementsByTagName()	returns all the elements having the given tag name.
getElementsByClassName()	returns all the elements having the given class name.

Accessing field value by document object

In this example, we are going to get the value of input text by user. Here, we are using **document.form1.name.value** to get the value of name field.

Here, **document** is the root element that represents the html document.

form1 is the name of the form.

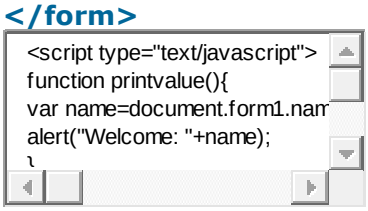
name is the attribute name of the input text.

value is the property, that returns the value of the input text.

Let's see the simple example of document object that prints name with welcome message.

```
<script type="text/javascript">
function printvalue(){
var name=document.form1.name.value;
alert("Welcome: "+name);
}
</script>

<form name="form1">
Enter Name:<input type="text" name="name"/>
```

```
10. <input type="button" onclick="printvalue()" value="print name"/>
11. </form>
```



Output of the above example

Enter Name:

Javascript - document.getElementById() method

1. getElementById() method
2. Example of getElementById()

The **document.getElementById()** method returns the element of specified id.

In the previous page, we have used **document.form1.name.value** to get the value of the input value. Instead of this, we can use document.getElementById() method to get value of the input text. But we need to define id for the input field.

Let's see the simple example of document.getElementById() method that prints cube of the given number.

```
1. <script type="text/javascript">
2. function getcube(){
3. var number=document.getElementById("number").value;
4. alert(number*number*number);
5. }
5. </script>
7. <form>
3. Enter No:<input type="text" id="number" name="number"/><br/>
```

```
9. <input type="button" value="cube" onclick="getcube()"/>
10. </form>
```

```
<script type="text/javascript">
function getcube(){
var number=document.getEleme
alert(number*number*number);
}
```

Output of the above example

Enter No:

Javascript - document.getElementsByName() method

1. getElementsByName() method
2. Example of getElementsByName()

The **document.getElementsByName()** method returns all the element of specified name.

The syntax of the getElementsByName() method is given below:

1. document.getElementsByName("name")

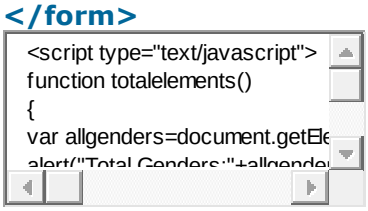
```
document.getElementsByName(
```

Here, name is required.

Example of document.getElementsByName() method

In this example, we going to count total number of genders. Here, we are using getElementsByName() method to get all the genders.

```
<script type="text/javascript">
function totalelements()
{
var allgenders=document.getElementsByName("gender");
alert("Total Genders:"+allgenders.length);
}
</script>
<form>
Male:<input type="radio" name="gender" value="male">
Female:<input type="radio" name="gender" value="female">

<input type="button" onclick="totalelements()" value="Total Genders">
</form>
```



Output of the above example

Male: Female:

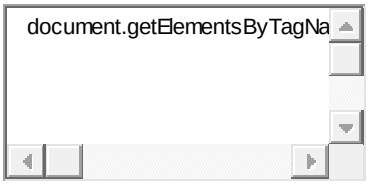
Bottom of FoJavascript - document.getElementsByTagName() method

1. getElementsByTagName() method
2. Example of getElementsByTagName()

The **document.getElementsByTagName()** method returns all the element of specified tag name.

The syntax of the getElementsByTagName() method is given below:

```
1. document.getElementsByTagName("name")
```

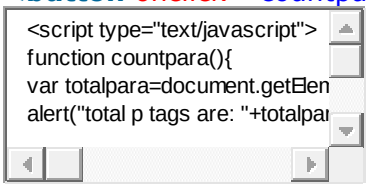


Here, name is required.

Example of document.getElementsByTagName() method

In this example, we going to count total number of paragraphs used in the document. To do this, we have called the document.getElementsByTagName("p") method that returns the total paragraphs.

```
<script type="text/javascript">
function countpara(){
var totalpara=document.getElementsByTagName("p");
alert("total p tags are: "+totalpara.length);

}
</script>
<p>This is a pragraph</p>
<p>Here we are going to count total number of paragraphs by getElementByTagName() method.</p>
<p>Let's see the simple example</p>
<button onclick="countpara()">count paragraph</button>
```



Output of the above example

This is a pragraph

Here we are going to count total number of paragraphs by getElementByTagName() method.

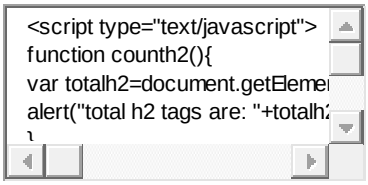
Let's see the simple example

count paragraph

Another example of document.getElementsByTagName() method

In this example, we going to count total number of h2 and h3 tags used in the document.

```
<script type="text/javascript">
function counth2(){
var totalh2=document.getElementsByTagName("h2");
alert("total h2 tags are: "+totalh2.length);
}
function counth3(){
var totalh3=document.getElementsByTagName("h3");
alert("total h3 tags are: "+totalh3.length);
}
</script>
<h2>This is h2 tag</h2>
<h2>This is h2 tag</h2>
<h3>This is h3 tag</h3>
<h3>This is h3 tag</h3>
<h3>This is h3 tag</h3>
<button onclick="counth2()">count h2</button>
<button onclick="counth3()">count h3</button>
```



Output of the above example

This is h2 tag

This is h2 tag

This is h3 tag
This is h3 tag
This is h3 tag

count h2 count h3

Note: Output of the given examples may differ on this page because it will count the total number of para , total number of h2 and total number of h3 tags used in this document

---

Javascript - innerHTML

1. javascript innerHTML
2. Example of innerHTML property

The **innerHTML** property can be used to write the dynamic html on the html document.

It is used mostly in the web pages to generate the dynamic html such as registration form, comment form, links etc.

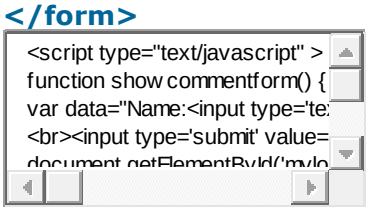
Example of innerHTML property

In this example, we are going to create the html form when user clicks on the button.

In this example, we are dynamically writing the html form inside the div name having the id mylocation. We are identifing this position by calling the document.getElementById() method.

```
<script type="text/javascript" >
function showcommentform() {
var data="Name:<input type='text' name='name'><br>Comment:<br><textarea rows='5' cols='80'></textarea>
<br><input type='submit' value='Post Comment'>";
document.getElementById('mylocation').innerHTML=data;
}
```

```
</script>
<form name="myForm">
<input type="button" value="comment" onclick="showcommentform()">
<div id="mylocation"></div>
</form>
```



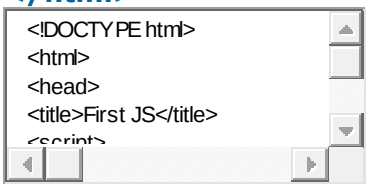
Test it Now

Output of the above example

Show/Hide Comment Form Example using innerHTML

```
<!DOCTYPE html>
<html>
<head>
<title>First JS</title>
<script>
var flag=true;
function commentform(){
var cform="<form action='Comment'>Enter Name:<br><input type='text' name='name'/><br/>
Enter Email:<br><input type='email' name='email'/><br>Enter Comment:<br/>
<textarea rows='5' cols='70'></textarea><br><input type='submit' value='Post Comment'/></form>";
if(flag){
document.getElementById("mylocation").innerHTML=cform;
flag=false;
}else{
```

```
15. document.getElementById("mylocation").innerHTML="";
16. flag=true;
17. }
18. }
19. </script>
20. </head>
21. <body>
22. <button onclick="commentform()">Comment</button>
23. <div id="mylocation"></div>
24. </body>
25. </html>
```



Javascript - innerText

1. javascript innerText
2. Example of innerText property

The **innerText** property can be used to write the dynamic text on the html document. Here, text will not be interpreted as html text but a normal text.

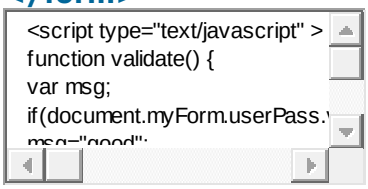
It is used mostly in the web pages to generate the dynamic content such as writing the validation message, password strength etc.

Javascript innerText Example

In this example, we are going to display the password strength when releases the key after press.

```
1. <script type="text/javascript" >
2. function validate() {
3. var msg;
4. if(document.myForm.userPass.value.length>5){
5. msg="good";
6. }
7. else{
```

```
msg="poor";
}
document.getElementById('mylocation').innerText=msg;
}

</script>
<form name="myForm">
<input type="password" value="" name="userPass" onkeyup="validate()">
Strength:<span id="mylocation">no strength</span>
</form>
```



Test it Now

Output of the above example

Strength:no strength

JavaScript Form Validation

1. JavaScript form validation
2. Example of JavaScript validation
3. JavaScript email validation

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.

The JavaScript provides you the facility the validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.

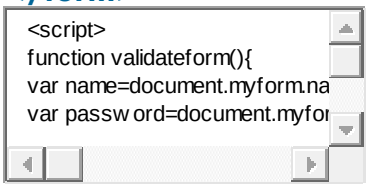
Through JavaScript, we can validate name, password, email, date, mobile number etc fields.

JavaScript form validation example

In this example, we are going to validate the name and password. The name can't be empty and password can't be less than 6 characters long.

Here, we are validating the form on form submit. The user will not be forwarded to the next page until given values are correct.

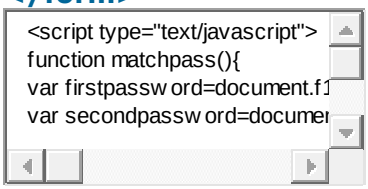
```
1. <script>
2. function validateform(){
3. var name=document.myform.name.value;
4. var password=document.myform.password.value;
5. 
6. if (name==null || name==""){
7.   alert("Name can't be blank");
8.   return false;
9. }else if(password.length<6){
10.  alert("Password must be at least 6 characters long.");
11.  return false;
12.  }
13. }
14. </script>
15. <body>
16. <form name="myform" method="post" action="abc.jsp" onsubmit="return v
    alidateform()" >
17. Name: <input type="text" name="name"><br/>
18. Password: <input type="password" name="password"><br/>
19. <input type="submit" value="register">
20. </form>
```



Test it Now

JavaScript Retype Password Validation

```
1. <script type="text/javascript">
2. function matchpass(){
3. var firstpassword=document.f1.password.value;
4. var secondpassword=document.f1.password2.value;
5. 
6. if(firstpassword==secondpassword){
7. return true;
8. }
9. else{
10. alert("password must be same!");
11. return false;
12. }
13. }
14. </script>
15. 
16. <form name="f1" action="register.jsp" onsubmit="return matchpass()">
17. Password:<input type="password" name="password" /><br/>
18. Re-enter Password:<input type="password" name="password2"/><br/>
19. <input type="submit">
20. </form>
```



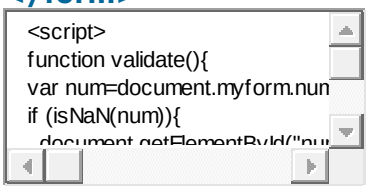
Test it Now

JavaScript Number Validation

Let's validate the textfield for numeric value only. Here, we are using isNaN() function.

```
1. <script>
2. function validate(){
3. var num=document.myform.num.value;
```

```
4. if (isNaN(num)){
5.   document.getElementById("numloc").innerHTML="Enter Numeric value only"
   ;
6.   return false;
7. }else{
8.   return true;
9.   }
10. }
11. </script>
12. <form name="myform" onsubmit="return validate()" >
13. Number: <input type="text" name="num"><span id="numloc"></span><
   br/>
14. <input type="submit" value="submit">
15. </form>
```



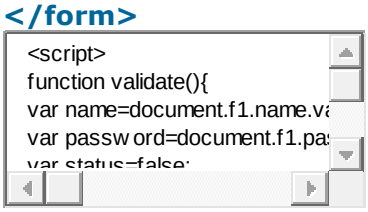
Test it Now

JavaScript validation with image

Let's see an interactive JavaScript form validation example that displays correct and incorrect image if input is correct or incorrect.

```
1. <script>
2. function validate(){
3. var name=document.f1.name.value;
4. var password=document.f1.password.value;
5. var status=false;
6.
7. if(name.length<1){
8. document.getElementById("nameloc").innerHTML=
9. " <img src='unchecked.gif'/> Please enter your name";
10. status=false;
11. }else{
12. document.getElementById("nameloc").innerHTML=" <img src='checked.gif'/>
```

```
";
13. status=true;
14. }
15. if(password.length<6){
16. document.getElementById("passwordloc").innerHTML=
17. " <img src='unchecked.gif'/> Password must be at least 6 char long";
18. status=false;
19. }else{
20. document.getElementById("passwordloc").innerHTML=" <img src='checked.gi
    f'/>";
21. }
22. return status;
23. }
24. </script>
25.
26. <form name="f1" action="#" onsubmit="return validate()">
27. <table>
28. <tr><td>Enter Name:</td><td><input type="text" name="name"/>
29. <span id="nameloc"></span></td></tr>
30. <tr><td>Enter Password:</td><td><input type="password" name="pass
    word"/>
31. <span id="passwordloc"></span></td></tr>
32. <tr><td colspan="2"><input type="submit" value="register"/></td></tr
    >
33. </table>
34. </form>
```



Test it Now

Output:

Enter Name:

Enter Password:

register

JavaScript email validation

We can validate the email by the help of JavaScript.

There are many criteria that need to be follow to validate the email id such as:

- email id must contain the @ and . character
- There must be at least one character before and after the @.
- There must be at least two characters after . (dot).

Let's see the simple example to validate the email field.

```
<script>
function validateemail()
{
var x=document.myform.email.value;
var atposition=x.indexOf("@");
var dotposition=x.lastIndexOf(".");
if (atposition<1 || dotposition<atposition+2 || dotposition+2>=x.length){
  alert("Please enter a valid e-
mail address \n atpostion:"+atposition+"\n dotposition:"+dotposition);
  return false;
  }
}
</script>
<body>
<form name="myform"  method="post" action="#" onsubmit="return validat
eemail();">
Email: <input type="text" name="email"><br/>

<input type="submit" value="register">
</form>
```

HTML/DOM events for JavaScript

HTML or DOM events are widely used in JavaScript code. JavaScript code is executed with HTML/DOM events. So before learning JavaScript, let's have some idea about events.

Events	Description
onclick	occurs when element is clicked.
ondblclick	occurs when element is double-clicked.
onfocus	occurs when an element gets focus such as buttor input, textarea etc.
onblur	occurs when form looses the focus from an element.
onsubmit	occurs when form is submitted.
onmouseover	occurs when mouse is moved over an element.
onmouseout	occurs when mouse is moved out from an element (afte moved over).
onmousedown	occurs when mouse button is pressed over an element.
onmouseup	occurs when mouse is released from an element (afte mouse is pressed).
onload	occurs when document, object or frameset is loaded.

onunload	occurs when body or frameset is unloaded.
onscroll	occurs when document is scrolled.
onresized	occurs when document is resized.
onreset	occurs when form is reset.
onkeydown	occurs when key is being pressed.
onkeypress	occurs when user presses the key.
onkeyup	occurs when key is released.

COLLECTIONS

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

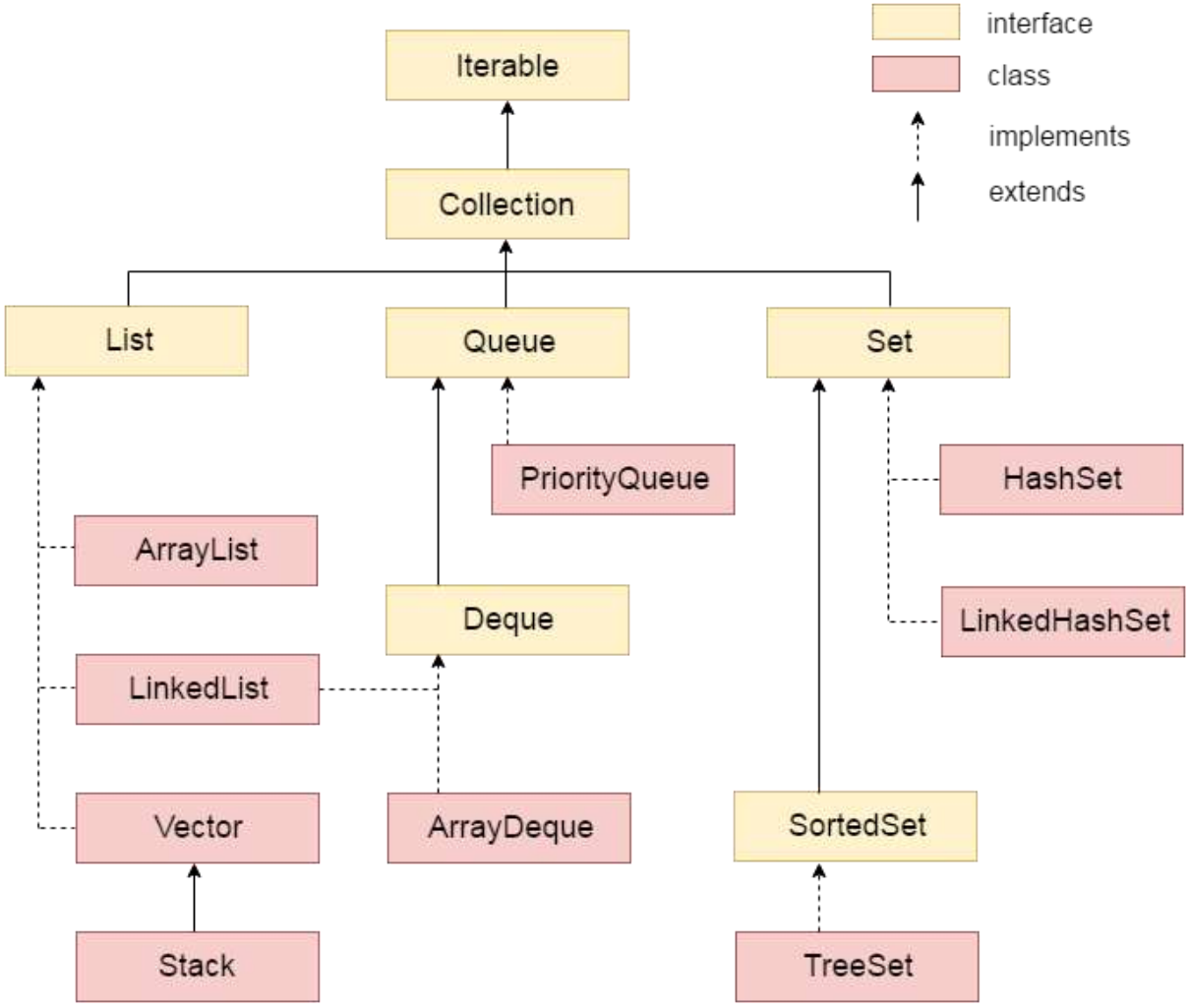
What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm.

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework.The **java.util** package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.
2	public boolean addAll(Collection	is used to insert the specified collection elements in the

	c)	invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

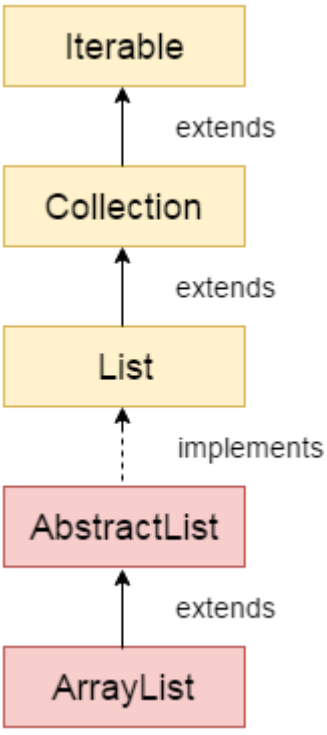
No.	Method	Description
1	public boolean hasNext()	It returns true if iterator has more elements.

2	public Object next()	It returns the element and moves the cursor pointer to the next element.
3	public void remove()	It removes the last elements returned by the iterator. It is rarely used.

What we are going to learn in Java Collections Framework

1. ArrayList class
2. LinkedList class
3. List interface
4. HashSet class
5. LinkedHashSet class
6. TreeSet class
7. PriorityQueue class
8. Map interface
9. HashMap class
10. LinkedHashMap class
11. TreeMap class
12. Hashtable class
13. Sorting
14. Comparable interface
15. Comparator interface
16. Properties class in Java

Java ArrayList class



Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Hierarchy of ArrayList class

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

Let's see the declaration for java.util.ArrayList class.

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

Constructors of Java ArrayList

ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of Java ArrayList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

void clear()	It is used to remove all of the elements from this list.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

Java Non-generic Vs Generic Collection

Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.

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

Let's see the old non-generic example of creating java collection.

```
1. ArrayList al=new ArrayList();//creating old non-generic arraylist
```

Let's see the new generic example of creating java collection.

```
1. ArrayList<String> al=new ArrayList<String>();//creating new generic arraylist
```

In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

For more information of java generics, click here Java Generics Tutorial.

Java ArrayList Example

```
import java.util.*;

class TestCollection1{

 public static void main(String args[]){

  ArrayList<String> list=new ArrayList<String>();//Creating arraylist

  list.add("Ravi");//Adding object in arraylist

  list.add("Vijay");

  list.add("Ravi");

  list.add("Ajay");

  //Traversing list through Iterator

  Iterator itr=list.iterator();

  while(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}
```

Output:-

```
Ravi
Vijay
Ravi
Ajay
```

Two ways to iterate the elements of collection in java

There are two ways to traverse collection elements:

1. By Iterator interface.
2. By for-each loop.

In the above example, we have seen traversing ArrayList by Iterator. Let's see the example to traverse ArrayList elements using for-each loop.

Iterating Collection through for-each loop

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

```
class TestCollection2{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  for(String obj:al)
    System.out.println(obj);
 }
}
```

OUTPUT:-

```
Ravi
Vijay
Ravi
Ajay
```

User-defined class objects in Java ArrayList

Let's see an example where we are storing Student class object in array list.

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

```
}

import java.util.*;

public class TestCollection3{

 public static void main(String args[]){

  //Creating user-defined class objects

  Student s1=new Student(101,"Sonoo",23);

  Student s2=new Student(102,"Ravi",21);

  Student s2=new Student(103,"Hanumat",25);

  //creating arraylist

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

al.add(s1);//adding Student class object

   al.add(s2);

      al.add(s3);
    //Getting Iterator
      Iterator itr=al.iterator();
     //traversing elements of ArrayList object
      while(itr.hasNext()){
      Student st=(Student)itr.next();
      System.out.println(st.rollno+" "+st.name+" "+st.age);
     }
`    }

      }
```

Output:-

```
101 Sonoo 23
102 Ravi 21
103 Hanumat 25
```

Example of addAll(Collection c) method

```
import java.util.*;

class TestCollection4{

 public static void main(String args[]){
```

```
 ArrayList<String> al=new ArrayList<String>();
 al.add("Ravi");
 al.add("Vijay");
 al.add("Ajay");
 ArrayList<String> al2=new ArrayList<String>();
 al2.add("Sonoo");
 al2.add("Hanumat");
 al.addAll(al2);//adding second list in first list
 Iterator itr=al.iterator();
 while(itr.hasNext()){
  System.out.println(itr.next());
 }
}
}
```

Output:-

```
Ravi
Vijay
Ajay
Sonoo
Hanumat
```

Example of removeAll() method

```
import java.util.*;
class TestCollection5{
 public static void main(String args[]){
  ArrayList<String> al=new ArrayList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ajay");
```

```
ArrayList<String> al2=new ArrayList<String>();
al2.add("Ravi");
al2.add("Hanumat");
al.removeAll(al2);
System.out.println("iterating the elements after removing the elements of al2...");
Iterator itr=al.iterator();
while(itr.hasNext()){
 System.out.println(itr.next());
}


}
}
```

Output:-

```
iterating the elements after removing the elements of al2...
Vijay
Ajay
```

Example of retainAll() method

```
import java.util.*;
class TestCollection6{
 public static void main(String args[]){
 ArrayList<String> al=new ArrayList<String>();
 al.add("Ravi");
 al.add("Vijay");
 al.add("Ajay");
 ArrayList<String> al2=new ArrayList<String>();
 al2.add("Ravi");
```

```
 al2.add("Hanumat");

 al.retainAll(al2);

 System.out.println("iterating the elements after retaining the elements of al2...");

 Iterator itr=al.iterator();

 while(itr.hasNext()){

  System.out.println(itr.next());

 }

}

}
```

Output:-

```
iterating the elements after retaining the elements of al2...
Ravi
```

Java ArrayList Example: Book

Let's see an ArrayList example where we are adding books to list and printing all the books.

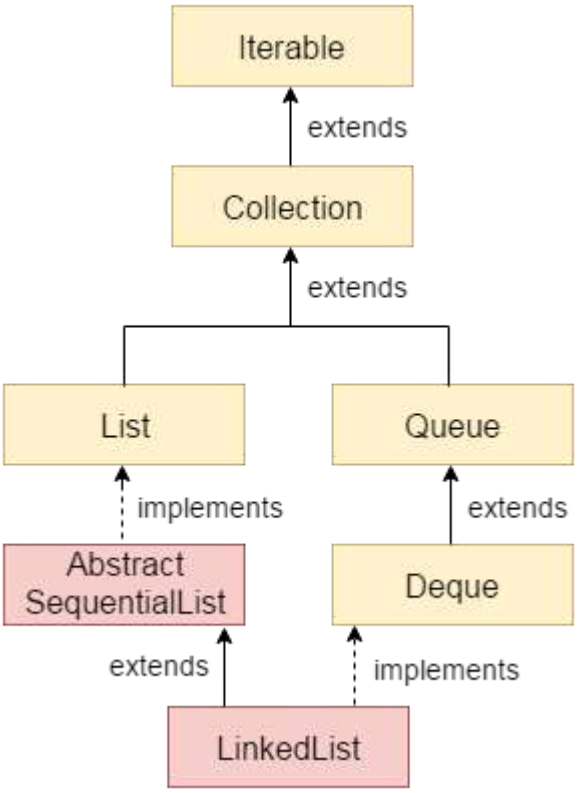
```
import java.util.*;

class Book {

int id;

String name,author,publisher;

int quantity;

public Book(int id, String name, String author, String publisher, int quantity) {

   this.id = id;

   this.name = name;

   this.author = author;

   this.publisher = publisher;

   this.quantity = quantity;

}
```

```
    }

    public class ArrayListExample {

    public static void main(String[] args) {

       //Creating list of Books

       List<Book> list=new ArrayList<Book>();

       //Creating Books

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

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

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

       //dding Books to list

    list.dd(b1);

       list.add(b2);

      list.add(b3);

       //Traversing list

       for(Book b:list){

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

       }

    }

}
```

Output:

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

Java LinkedList class



Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

Hierarchy of LinkedList class

As shown in above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

Doubly Linked List

In case of doubly linked list, we can add or remove elements from both side.

fig- doubly linked list

LinkedList class declaration

Let's see the declaration for java.util.LinkedList class.

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

Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection c)	It is used to construct a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Methods of Java LinkedList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
void addFirst(Object o)	It is used to insert the given element at the beginning of a list.
void addLast(Object o)	It is used to append the given element to the end of a list.
int size()	It is used to return the number of elements in a list
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean contains(Object o)	It is used to return true if the list contains a specified element.
boolean remove(Object o)	It is used to remove the first occurence of the specified element in a list.
Object getFirst()	It is used to return the first element in a list.
Object getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the

specified element, or -1 if the list does not contain any element.

Java LinkedList Example

```
import java.util.*;
public class TestCollection7{
 public static void main(String args[]){

  LinkedList<String> al=new LinkedList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");

  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
```

```
Output:
        Ravi
        Vijay
        Ravi
        Ajay
```

Java LinkedList Example: Book

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

```
class Book {

int id;

String name,author,publisher;

int quantity;

public Book(int id, String name, String author, String publisher, int quantity) {

    this.id = id;

    this.name = name;

    this.author = author;

    this.publisher = publisher;

    this.quantity = quantity;

}

}

public class LinkedListExample {

public static void main(String[] args) {

    //Creating list of Books

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

    //Creating Books

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

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

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

    //Adding Books to list

    list.add(b1);

    list.add(b2);

    list.add(b3);

    //Traversing list
```

```
        for(Book b:list){
       System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
        }
      }
  }
```

Output:

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

Difference between ArrayList and LinkedList

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

But there are many differences between ArrayList and LinkedList classes that are given below.

ArrayList	LinkedList
1) ArrayList internally uses **dynamic array** to store the elements.	LinkedList internally uses **doubly linked list** to store the elements.
2) Manipulation with ArrayList is **slow** because it internally uses array. If any element is removed from the array, all the bits are shifted in memory.	Manipulation with LinkedList is **faster** than ArrayList because it uses doubly linked list so no bit shifting is required in memory.
3) ArrayList class can **act as a list** only because it implements List only.	LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces.
4) ArrayList is **better for storing and accessing** data.	LinkedList is **better for manipulating** data.

Example of ArrayList and LinkedList in Java

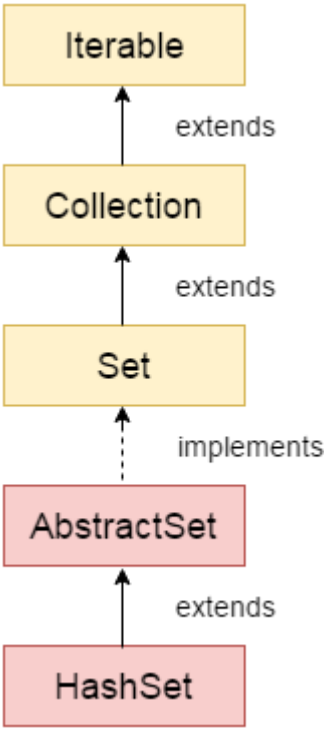
```
import java.util.*;
class TestArrayLinked{
 public static void main(String args[]){

  List<String> al=new ArrayList<String>();//creating arraylist
```

```
    al.add("Ravi");//adding object in arraylist
    al.add("Vijay");
    al.add("Ravi");
    al.add("Ajay");

    List<String> al2=new LinkedList<String>();//creating linkedlist
    al2.add("James");//adding object in linkedlist
    al2.add("Serena");
    al2.add("Swati");
    al2.add("Junaid");

    System.out.println("arraylist: "+al);
    System.out.println("linkedlist: "+al2);
  }

}
```

Output:

```
arraylist: [Ravi,Vijay,Ravi,Ajay]
linkedlist: [James,Serena,Swati,Junaid]
```

Java HashSet class



Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.

Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class

The HashSet class extends AbstractSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

HashSet class declaration

Let's see the declaration for java.util.HashSet class.

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

Constructors of Java HashSet class:

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.

Methods of Java HashSet class:

Method	Description
void clear()	It is used to remove all of the elements from this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean add(Object o)	It is used to adds the specified element to this set if it is not already present.

boolean isEmpty()	It is used to return true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
Object clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
Iterator iterator()	It is used to return an iterator over the elements in this set.
int size()	It is used to return the number of elements in this set.

Java HashSet Example

```
import java.util.*;
class TestCollection9{
 public static void main(String args[]){
  //Creating HashSet and adding elements
  HashSet<String> set=new HashSet<String>();
  set.add("Ravi");
  set.add("Vijay");
  set.add("Ravi");
  set.add("Ajay");
  //Traversing elements
 Iterator<String> itr=set.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Output:-

```
Ajay
Vijay
Ravi
```

Java HashSet Example: Book

Let's see a HashSet example where we are adding books to set and printing all the books.

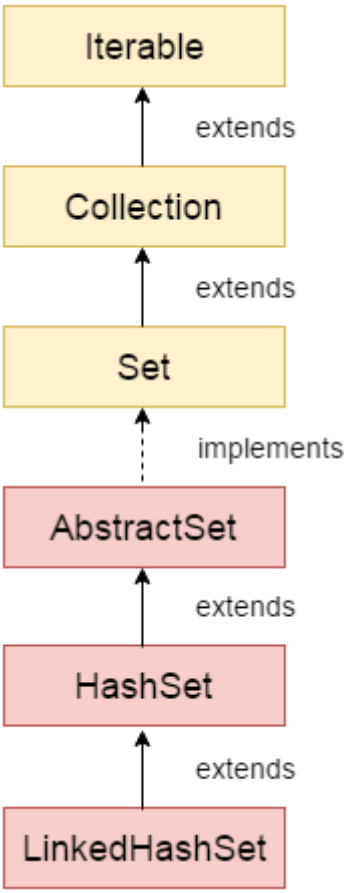
```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
   this.id = id;
   this.name = name;
   this.author = author;
   this.publisher = publisher;
   this.quantity = quantity;
  }
}
  public class HashSetExample {
  public static void main(String[] args) {
     HashSet<Book> set=new HashSet<Book>();
     //Creating Books
     Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);

  Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4)
  ;
     Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
     //Adding Books to HashSet
     set.add(b1);
```

```
            set.add(b2);
            set.add(b3);
            //Traversing HashSet
            for(Book b:set){
            System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
            }
        }
    }
```

Output:

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

Java LinkedHashSet class



Java LinkedHashSet class is a Hash table and Linked list implementation of the set interface. It inherits HashSet class and implements Set interface.

The important points about Java LinkedHashSet class are:

- Contains unique elements only like HashSet.
- Provides all optional set operations, and permits null elements.
- Maintains insertion order.

Hierarchy of LinkedHashSet class

The LinkedHashSet class extends HashSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

LinkedHashSet class declaration

Let's see the declaration for java.util.LinkedHashSet class.

```
1. public class LinkedHashSet<E> extends HashSet<E> implements Set<E>, Cloneable, Serializab
   le
```

Constructors of Java LinkedHashSet class

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
LinkedHashSet(int capacity)	It is used initialize the capacity of the linkedhashset to the given integer value capacity.
LinkedHashSet(int capacity, float fillRatio)	It is used to initialize both the capacity and the fill ratio (also called load capacity) of the hash set from its argument.

Example of LinkedHashSet class:

```
import java.util.*;

class TestCollection10{

 public static void main(String args[]){

  LinkedHashSet<String> al=new LinkedHashSet<String>();
```

```
        al.add("Ravi");
        al.add("Vijay");
        al.add("Ravi");
        al.add("Ajay");
        Iterator<String> itr=al.iterator();
        while(itr.hasNext()){
         System.out.println(itr.next());
        }
       }
    }
```

Output:-

```
Ravi
Vijay
Ajay
```

Java LinkedHashSet Example: Book

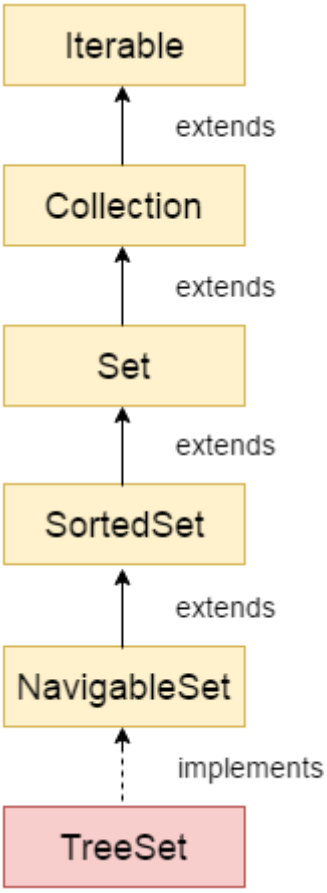
```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
   this.id = id;
   this.name = name;
   this.author = author;
   this.publisher = publisher;
```

```
    this.quantity = quantity;
  }
  }
  public class LinkedHashSetExample {
  public static void main(String[] args) {
    LinkedHashSet<Book> hs=new LinkedHashSet<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill”4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to hash table
    hs.add(b1);
    hs.add(b2);
    hs.add(b3);
    //Traversing hash table
   for(Book b:hs){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
  }
}
```

Output:

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

Java TreeSet class



Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements NavigableSet interface. The objects of TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

- Contains unique elements only like HashSet.
- Access and retrieval times are quiet fast.
- Maintains ascending order.

Hierarchy of TreeSet class

As shown in above diagram, Java TreeSet class implements NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

TreeSet class declaration

Let's see the declaration for java.util.TreeSet class.

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

Constructors of Java TreeSet class

Constructor	Description
TreeSet()	It is used to construct an empty tree set that will be sorted in an ascending order according to the natural order of the tree set.
TreeSet(Collection c)	It is used to build a new tree set that contains the elements of the collection c.
TreeSet(Comparator comp)	It is used to construct an empty tree set that will be sorted according to given comparator.
TreeSet(SortedSet ss)	It is used to build a TreeSet that contains the elements of the given SortedSet.

Methods of Java TreeSet class

Method	Description
boolean addAll(Collection c)	It is used to add all of the elements in the specified collection to this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean isEmpty()	It is used to return true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void add(Object o)	It is used to add the specified element to this set if it is not already present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It is used to return a shallow copy of this TreeSet instance.
Object first()	It is used to return the first (lowest) element currently in this sorted set.
Object last()	It is used to return the last (highest) element currently in this sorted

	set.
int size()	It is used to return the number of elements in this set.

Java TreeSet Example

```
import java.util.*;
class TestCollection11{
 public static void main(String args[]){
  //Creating and adding elements
TreeSet<String> al=new TreeSet<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  //Traversing elements
  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Test it Now

Output:

```
Ajay
Ravi
Vijay
```

Java TreeSet Example: Book

Let's see a TreeSet example where we are adding books to set and printing all the books. The elements in TreeSet must be of Comparable type. String and Wrapper classes are Comparable

by default. To add user-defined objects in TreeSet, you need to implement Comparable interface.

```
import java.util.*;

class Book implements Comparable<Book>{

int id;

String name,author,publisher;

int quantity;

public Book(int id, String name, String author, String publisher, int quantity) {

   this.id = id;

   this.name = name;

   this.author = author;

   this.publisher = publisher;

   this.quantity = quantity;

}

public int compareTo(Book b) {

   if(id>b.id){

      return 1;

   }else if(id<b.id){

      return -1;

   }else{

   return 0;

   }

}

}

public class TreeSetExample {

public static void main(String[] args) {
```

```
        Set<Book> set=new TreeSet<Book>();

        //Creating Books

        Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);

        Book b2=new Book(233,"Operating System","Galvin","Wiley",6);

        Book b3=new Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill"
    ,4);

        //Adding Books to TreeSet

        set.add(b1);

        set.add(b2);

        set.add(b3);

        //Traversing TreeSet

        for(Book b:set){

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

        }

    }

}
```

Output:

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

Java Queue Interface

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

Queue Interface declaration

1. public interface Queue<E> extends Collection<E>

Methods of Java Queue Interface

Method	Description
boolean add(object)	It is used to insert the specified element into this queue and return true upon success.
boolean offer(object)	It is used to insert the specified element into this queue.
Object remove()	It is used to retrieves and removes the head of this queue.
Object poll()	It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.
Object element()	It is used to retrieves, but does not remove, the head of this queue.
Object peek()	It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

PriorityQueue class

The PriorityQueue class provides the facility of using queue. But it does not orders the elements in FIFO manner. It inherits AbstractQueue class.

PriorityQueue class declaration

Let's see the declaration for java.util.PriorityQueue class.

1. public class PriorityQueue<E> extends AbstractQueue<E> implements Serializable

Java PriorityQueue Example

```
import java.util.*;

class TestCollection12{

public static void main(String args[]){

PriorityQueue<String> queue=new PriorityQueue<String>();

queue.add("Amit");

queue.add("Vijay");

queue.add("Karan");

queue.add("Jai");
```

```
queue.add("Rahul");

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

Iterator itr=queue.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

Iterator<String> itr2=queue.iterator();

while(itr2.hasNext()){

System.out.println(itr2.next());

}

}

}
```

```
Output:-
        head:Amit
        head:Amit
        iterating the queue elements:
        Amit
        Jai
        Karan
        Vijay
        Rahul
        after removing two elements:
        Karan
        Rahul
        Vijay
```

Java PriorityQueue Example: Book

Let's see a PriorityQueue example where we are adding books to queue and printing all the

books. The elements in PriorityQueue must be of Comparable type. String and Wrapper classes are Comparable by default. To add user-defined objects in PriorityQueue, you need to implement Comparable interface.

```
import java.util.*;

class Book implements Comparable<Book>{

int id;

String name,author,publisher;

int quantity;

public Book(int id, String name, String author, String publisher, int quantity) {

   this.id = id;

   this.name = name;

   this.author = author;

   this.publisher = publisher;

   this.quantity = quantity;

}

public int compareTo(Book b) {

   if(id>b.id){

      return 1;

   }else if(id<b.id){

      return -1;

   }else{

   return 0;

   }

}

}

public class LinkedListExample {
```

```
public static void main(String[] args) {

   Queue<Book> queue=new PriorityQueue<Book>();

   //Creating Books

   Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);

   Book b2=new Book(233,"Operating System","Galvin","Wiley",6);

   Book b3=new Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill"
,4);

   //Adding Books to the queue

   queue.add(b1);

   queue.add(b2);

   queue.add(b3);

   System.out.println("Traversing the queue elements:");

   //Traversing queue elements

   for(Book b:queue){

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

   }

   queue.remove();

   System.out.println("After removing one book record:");

   for(Book b:queue){

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

      }

}

}
```

Output:

```
Traversing the queue elements:
101 Data Communications & Networking Forouzan Mc Graw Hill 4
233 Operating System Galvin Wiley 6
121 Let us C Yashwant Kanetkar BPB 8
```

```
After removing one book record:
121 Let us C Yashwant Kanetkar BPB 8
233 Operating System Galvin Wiley 6
```

Java Deque Interface

Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for **"double ended queue".**

Deque Interface declaration

1. public interface Deque<E> extends Queue<E>

Methods of Java Deque Interface

Method	Description
boolean add(object)	It is used to insert the specified element into this deque and return true upon success.
boolean offer(object)	It is used to insert the specified element into this deque.
Object remove()	It is used to retrieves and removes the head of this deque.
Object poll()	It is used to retrieves and removes the head of this deque, or returns null if this deque is empty.
Object element()	It is used to retrieves, but does not remove, the head of this deque.
Object peek()	It is used to retrieves, but does not remove, the head of this deque, or returns null if this deque is empty.

ArrayDeque class

The ArrayDeque class provides the facility of using deque and resizable-array. It inherits AbstractCollection class and implements the Deque interface.

The important points about ArrayDeque class are:

- Unlike Queue, we can add or remove elements from both sides.
- Null elements are not allowed in the ArrayDeque.
- ArrayDeque is not thread safe, in the absence of external synchronization.
- ArrayDeque has no capacity restrictions.

- ArrayDeque is faster than LinkedList and Stack.

ArrayDeque Hierarchy

The hierarchy of ArrayDeque class is given in the figure displayed at the right side of the page.

ArrayDeque class declaration

Let's see the declaration for java.util.ArrayDeque class.

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

---

Java ArrayDeque Example

```
import java.util.*;

public class ArrayDequeExample {

  public static void main(String[] args) {

  //Creating Deque and adding elements

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

  deque.add("Ravi");

  deque.add("Vijay");

  deque.add("Ajay");

  //Traversing elements

  for (String str : deque) {

  System.out.println(str);

  }

  }

}
```

Output:

```
Ravi
Vijay
```

```
Ajay
```

Java ArrayDeque Example: offerFirst() and pollLast()

```
import java.util.*;

public class DequeExample {

public static void main(String[] args) {

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

   deque.offer("arvind");

   deque.offer("vimal");

   deque.add("mukul");

   deque.offerFirst("jai");

   System.out.println("After offerFirst Traversal...");

   for(String s:deque){

      System.out.println(s);

   }

   //deque.poll();

   //deque.pollFirst();//it is same as poll()

   deque.pollLast();

   System.out.println("After pollLast() Traversal...");

   for(String s:deque){

      System.out.println(s);

   }

}

}
```

Output:

```
After offerFirst Traversal...
jai
arvind
```

```
vimal
mukul
After pollLast() Traversal...
jai
arvind
vimal
```

Java ArrayDeque Example: Book

```
import java.util.*;

class Book {

int id;

String name,author,publisher;

int quantity;

public Book(int id, String name, String author, String publisher, int quantity) {

   this.id = id;

   this.name = name;

   this.author = author;

   this.publisher = publisher;

   this.quantity = quantity;

}

}

public class ArrayDequeExample {

public static void main(String[] args) {

   Deque<Book> set=new ArrayDeque<Book>();

   //Creating Books

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

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

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

```
        //Adding Books to Deque
        set.add(b1);
    set.add(b2);
        set.add(b3);
        //Traversing ArrayDeque
        for(Book b:set){
        System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
        }
      }
      }
```

Output:

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

Difference between ArrayList and Vector

ArrayList and Vector both implements List interface and maintains insertion order.

But there are many differences between ArrayList and Vector classes that are given below.

ArrayList	Vector
1) ArrayList is **not synchronized**.	Vector is **synchronized**.
2) ArrayList **increments 50%** of current array size if number of element exceeds from its capacity.	Vector **increments 100%** means doubles the array size if total number of element exceeds than its capacity.
3) ArrayList is **not a legacy** class, it is introduced in JDK 1.2.	Vector is a **legacy** class.
4) ArrayList is **fast** because it is non-synchronized.	Vector is **slow** because it is synchronized i.e. in multithreading environment, it will hold the other threads in runnable or non-runnable state until current thread releases the lock of object.

5) ArrayList uses **Iterator** interface to traverse the elements.	Vector uses **Enumeration** interface to traverse the elements. But it can use Iterator also.

Example of Java ArrayList

```
import java.util.*;

class TestArrayList21{

 public static void main(String args[]){


  List<String> al=new ArrayList<String>();//creating arraylist

  al.add("Sonoo");//adding object in arraylist

  al.add("Michael");

  al.add("James");

  al.add("Andy");

  //traversing elements using Iterator

  Iterator itr=al.iterator();

  while(itr.hasNext()){

   System.out.println(itr.next());

  }

 }

}
```

Output:

```
Sonoo
Michael
James
Andy
```

Example of Java Vector

Let's see a simple example of java Vector class that uses Enumeration interface.

```
import java.util.*;

class TestVector1{

 public static void main(String args[]){

  Vector<String> v=new Vector<String>();//creating vector

  v.add("umesh");//method of Collection

  v.addElement("irfan");//method of Vector

  v.addElement("kumar");

  //traversing elements using Enumeration

  Enumeration e=v.elements();

  while(e.hasMoreElements()){

   System.out.println(e.nextElement());

  }

 }

}
```

Output:

```
umesh
irfan
kumar
```