UNIT-1

HTML stands for Hyper Text Markup Language, which is the most widely used language on Web to develop web pages.

HTML was created by Berners-Lee in late 1991 but "HTML 2.0" was the first standard HTML specification which was published in 1995. HTML 4.01 was a major version of HTML and it was published in late 1999. Though HTML 4.01 version is widely used but currently we are having HTML-5 version which is an extension to HTML 4.01, and this version was published in 2012.

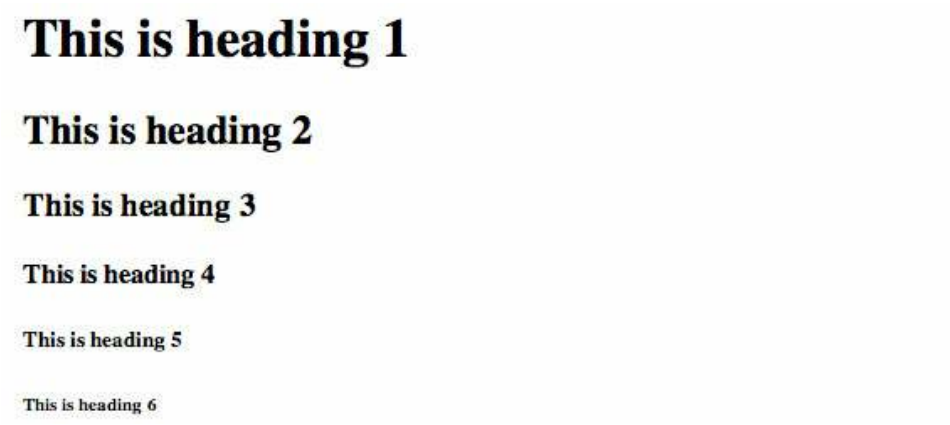
Basic Tags

Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>, <h2>, <h3>, <h4>, <h5>, and <h6>**. While displaying any heading, browser adds one line before and one line after that heading.

Example

```
<html>

<head>

<title>Heading Example</title>

</head>

<body>

<h1>This is heading 1</h1>

<h2>This is heading 2</h2>

<h3>This is heading 3</h3>

<h4>This is heading 4</h4>

<h5>This is heading 5</h5>

<h6>This is heading 6</h6>

</body>

</html>
```



This will produce following result:

Paragraph Tag:

The ** ** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening and a closing tag as shown below in the example:

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Paragraph Example</title>
</head>
<body>
<p>Here is a first paragraph of text.</p>
<p>Here is a second paragraph of text.</p>
<p>Here is a third paragraph of text.</p>
</body>
</html>
```

This will produce following result:

Here is a first paragraph of text.

Here is a second paragraph of text.

Here is a third paragraph of text.

Line Break Tag:

Whenever you use the **
** element, anything following it starts from the next line. This tag is an example of an **empty**element, where you do not need opening and closing tags, as there is nothing to go in between them.

The
 tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use
 it is not valid in XHTML

Example

```
<!DOCTYPE html>
<html>
<head>
<title>Line Break  Example</title>
</head>
<body>
<p>Hello<br />
You delivered your assignment ontime.<br />
Thanks<br />
Mahnaz</p>
</body>
</html>
```

HTML ATTRIBUTES

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts: a **name** and a **value**:

- The **name** is the property you want to set. For example, the paragraph <p> element in the example carries an attribute whose name is **align**, which you can use to indicate the alignment of paragraph on the page.
- The **value** is what you want the value of the property to be set and always put within quotations. The below example shows three possible values of align

attribute:**left, center** and **right**.

```
<!DOCTYPE html>
<html>
<head>
<title>Align Attribute  Example</title>
</head>
<body>
<p align="left">This is left aligned</p>
<p align="center">This is center aligned</p>
<p align="right">This is right aligned</p>
</body>
</html>
```

This will display following result:

This is left aligned

This is center aligned

This is right aligned

XML stands for Extensible Markup Language. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

There are three important characteristics of XML that make it useful in a variety of systems and solutions:

- **XML is extensible:** XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it:** XML allows you to store the data

irrespective of how it will be presented.

- **XML is a public standard:** XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

XML Usage:

A short list of XML usage says it all:

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

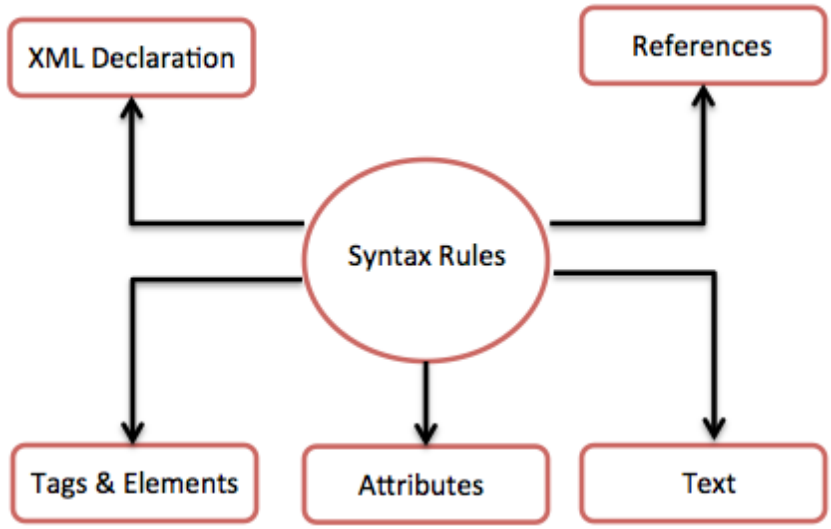
XML SYNTAX

```
<?xml version="1.0"?>
<contact-info>
<name>Tanmay Patil</name>
<company>TutorialsPoint</company>
<phone>(011) 123-4567</phone>
</contact-info>
```

You can notice there are two kinds of information in the above example:

- markup, like *<contact-info>* and
- the text, or the character data, *Tutorials Point* and *(040) 123-4567*.

The following diagram depicts the syntax rules to write different types of markup and text in an XML document.



Let us see each component of the above diagram in detail:

XML Declaration

The XML document can optionally have an XML declaration. It is written as below:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Where *version* is the XML version and *encoding* specifies the character encoding used in the document.

Syntax Rules for XML declaration:

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lower-case.
- If document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs be the first statement in the XML document.
- An HTTP protocol can override the value of *encoding* that you put in the XML declaration.

Tags and Elements;

An XML file is structured by several XML-elements, also called XML-nodes or XML-tags. XML-elements' names are enclosed by triangular brackets < > as shown below:

```
<element>
```

Syntax Rules for Tags and Elements:

Element Syntax: Each XML-element needs to be closed either with start or with end elements as shown below:

```
<element>....</element>
```

or in simple-cases, just this way:

```
<element/>
```

Nesting of elements: An XML-element can contain multiple XML-elements as its children, but the children elements must not overlap. i.e., an end tag of an element must have the same name as that of the most recent unmatched start tag.

Following example shows incorrect nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>TutorialsPoint
<contact-info>
</company>
```

Following example shows correct nested tags:

```
<?xml version="1.0"?>
<contact-info>
<company>TutorialsPoint</company>
<contact-info>
```

Root element: An XML document can have only one root element. For example, following is not a correct XML document, because both the x and y elements occur at the top level without a root element:

```
<x>...</x>
```

```
<y>...</y>
```

The following example shows a correctly formed XML document:

```
<root>
  <x>...</x>
  <y>...</y>
</root>
```

Case sensitivity: The names of XML-elements are case-sensitive. That means the name of the start and the end elements need to be exactly in the same case.

For example **<contact-info>** is different from **<Contact-Info>**.

XHTML

XHTML stands for E**X**tensible **H**yper**T**ext **M**arkup **L**anguage. It is the next step in the evolution of the internet. The XHTML 1.0 is the first document type in the XHTML family.

XHTML is almost identical to HTML 4.01 with only few differences. This is a cleaner and stricter version of HTML 4.01

Why Use XHTML?

Developers who migrate their content to XHTML 1.0 get the following benefits −

- XHTML documents are XML conforming as they are readily viewed, edited, and validated with standard XML tools.
- XHTML documents can be written to operate better than they did before in existing browsers as well as in new browsers.
- XHTML documents can utilize applications such as scripts and applets that rely upon either the HTML Document Object Model or the XML Document Object Model.
- XHTML gives you a more consistent, well-structured format so that your webpages can be easily parsed and processed by present and future web browsers.

XHTML syntax is very similar to HTML syntax and almost all the valid HTML elements

are valid in XHTML as well. But when you write an XHTML document, you need to pay a bit extra attention to make your HTML document compliant to XHTML.

Here are the important points to remember while writing a new XHTML document or converting existing HTML document into XHTML document −

- Write a DOCTYPE declaration at the start of the XHTML document.
- Write all XHTML tags and attributes in lower case only.
- Close all XHTML tags properly.
- Nest all the tags properly.
- Quote all the attribute values.
- Forbid Attribute minimization.
- Replace the **name** attribute with the **id** attribute.
- Deprecate the **language** attribute of the script tag.

Here is the detail explanation of the above XHTML rules −

DOCTYPE Declaration

All XHTML documents must have a DOCTYPE declaration at the start. There are three types of DOCTYPE declarations, which are discussed in detail in XHTML Doctypes chapter. Here is an example of using DOCTYPE −

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Case Sensitivity

XHTML is case sensitive markup language. All the XHTML tags and attributes need to be written in lower case only.

```
<!-- This is invalid in XHTML -->
<A Href="/xhtml/xhtml_tutorial.html">XHTML Tutorial</A>

<!-- Correct XHTML way of writing this is as follows -->
<a href="/xhtml/xhtml_tutorial.html">XHTML Tutorial</a>
```

In the example, **Href** and anchor tag **A** are not in lower case, so it is incorrect.

Closing the Tags

Each and every XHTML tag should have an equivalent closing tag, even empty elements should also have closing tags. Here is an example showing valid and invalid ways of using tags −

```
<!-- This is invalid in XHTML -->
<p>This paragraph is not written according to XHTML syntax.

<!-- This is also invalid in XHTML -->
<img src="/images/xhtml.gif" >
```

The following syntax shows the correct way of writing above tags in XHTML. Difference is that, here we have closed both the tags properly.

```
<!-- This is valid in XHTML -->
<p>This paragraph is not written according to XHTML syntax.</p>

<!-- This is also valid now -->
<img src="/images/xhtml.gif" />
```

Attribute Quotes

All the values of XHTML attributes must be quoted. Otherwise, your XHTML document is assumed as an invalid document. Here is the example showing syntax −

```
<!-- This is invalid in XHTML -->
<img src="/images/xhtml.gif" width=250 height=50 />

<!-- Correct XHTML way of writing this is as follows -->
<img src="/images/xhtml.gif" width="250" height="50" />
```

Attribute Minimization

XHTML does not allow attribute minimization. It means you need to explicitly state the attribute and its value. The following example shows the difference –

```
<!-- This is invalid in XHTML -->
<option selected>

<!-- Correct XHTML way of writing this is as follows -->
<option selected="selected">
```

Here is a list of the minimized attributes in HTML and the way you need to write them in XHTML –

HTML Style	XHTML Style
compact	compact="compact"
checked	checked="checked"
declare	declare="declare"
readonly	readonly="readonly"
disabled	disabled="disabled"
selected	selected="selected"
defer	defer="defer"
ismap	ismap="ismap"
nohref	nohref="nohref"
noshade	noshade="noshade"

nowrap	nowrap="nowrap"
multiple	multiple="multiple"
noresize	noresize="noresize"

The *id* Attribute

The id attribute replaces the name attribute. Instead of using name = "name", XHTML prefers to use id = "id". The following example shows how –

```
<!-- This is invalid in XHTML -->
<img src="/images/xhtml.gif" name="xhtml_logo" />

<!-- Correct XHTML way of writing this is as follows -->
<img src="/images/xhtml.gif" id="xhtml_logo" />
```

The *language* Attribute

The language attribute of the script tag is deprecated. The following example shows this difference –

```
<!-- This is invalid in XHTML -->

<script language="JavaScript" type="text/JavaScript">
  document.write("Hello XHTML!");
</script>

<!-- Correct XHTML way of writing this is as follows -->

<script type="text/JavaScript">
  document.write("Hello XHTML!");
</script>
```

Nested Tags

You must nest all the XHTML tags properly. Otherwise your document is assumed as

an incorrect XHTML document. The following example shows the syntax −

```
<!-- This is invalid in XHTML -->
<b><i> This text is bold and italic</b></i>

<!-- Correct XHTML way of writing this is as follows -->
<b><i> This text is bold and italic</i></b>
```

Element Prohibitions

The following elements are not allowed to have any other element inside them. This prohibition applies to all depths of nesting. Means, it includes all the descending elements.

Element	Prohibition
<a>	Must not contain other <a> elements.
<pre>	Must not contain the <img>, <object>, <big>, <small>, <sub>, or <sup> elements.
<button>	Must not contain the <input>, <select>, <textarea>, <label>, <button>, <form>, <fieldset>, <iframe> or <isindex> elements.
<label>	Must not contain other <label> elements.
<form>	Must not contain other <form> elements.

A Minimal XHTML Document

The following example shows you a minimum content of an XHTML 1.0 document −

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/TR/xhtml1" xml:lang="en" lang="en">
  <head>
    <title>Every document must have a title</title>
  </head>

  <body>
    ...your content goes here...
  </body>
</html>
```

APPLET CONTEXT

An applet runs inside a browser or applet viewer.
An applet running within a browser can ask the browser to do things for it:

Fetch an audio clip
Show a short message in the status line
Show a different web page

The browser can perform these requests or ignore them.

- To communicate with the browser, an applet calls the **java.applet.Applet.getAppletContext()** method, which returns an object that implements an interface of type **java.applet.AppletContext**.
- Think of **AppletContext** as a communication path between the applet and the browser.
- **AppletContext** provides the following methods:

Method	Description
void showStatus(String message)	Shows the message in the status line of the browser
Enumeration getApplets()	Returns an enumeration of all the applets in the same context (same web page)
Applet getApplet(String name)	Returns the applet in the current context with the specified name (null if none exists)
void showDocument(URL url)	Shows a new web page in the browser, displacing the current page.

void showDocument(URL url, String target)	Shows a new web page in the browser, specifying the target frame ("_self", "_parent", "_top", "_blank", or <frame-name>)
Image getImage(URL url)	Returns an image object that encapsulates the image specified by the URL
AudioClip getAudioClip(URL url)	Returns an AudioClip object that encapsulates the sound file specified by the URL.

```
public class LoadNewHTMLFileExample extends Applet{
 public void start(){

            URL codeBase = getCodeBase();

          AppletContext context = getAppletContext();



          try{

               URL url = new URL(codeBase + "AppletContextExample.html");
              Context .showDocument(url);

          }catch(MalformedURLException me){
          }
    }
}
```

Java Swing Tutorial

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.Unlike AWT, Java Swing provides platform-independent and lightweight components.The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox,

JMenu, JColorChooser etc.

```
import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame

JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height

f.add(b);//adding button in JFrame

f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}
```

JColorChooser class:

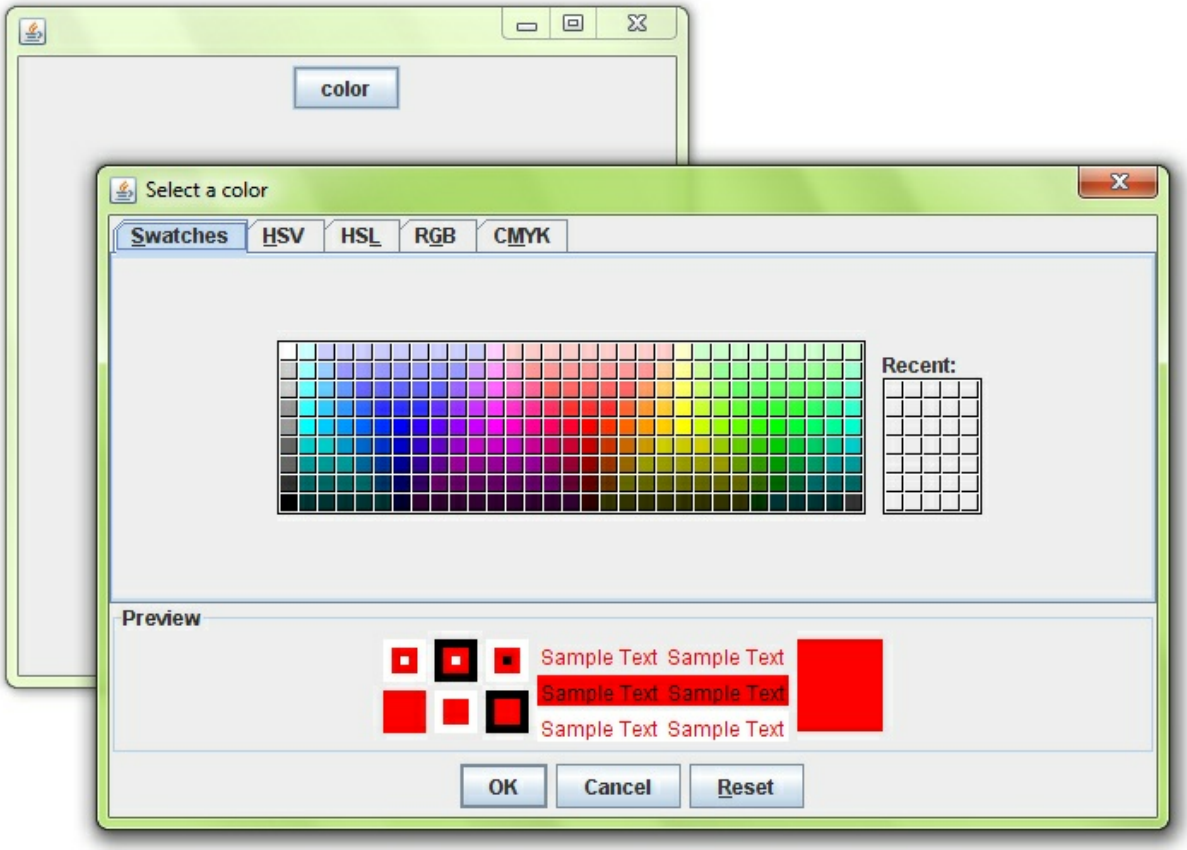
The JColorChooser class is used to create a color chooser dialog box so that user can select any color.

Commonly used Constructors of JColorChooser class:

- **JColorChooser():** is used to create a color chooser pane with white color initially.
- **JColorChooser(Color initialColor):** is used to create a color chooser pane with the specified color initially.

Commonly used methods of JColorChooser class:

public static Color showDialog(Component c, String title, Color initialColor): is used to show the color-chooser dialog box.




```
import java.awt.event.*;
1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class JColorChooserExample extends JFrame implements ActionListener{
5. JButton b;
6. Container c;
7.
8. JColorChooserExample(){
9.     c=getContentPane();
10.    c.setLayout(new FlowLayout());
11.
12.    b=new JButton("color");
13.    b.addActionListener(this);
14.
15.    c.add(b);
16.}
17.
18.public void actionPerformed(ActionEvent e) {
19.Color initialcolor=Color.RED;
20.Color color=JColorChooser.showDialog(this,"Select a color",initialcolor);
```

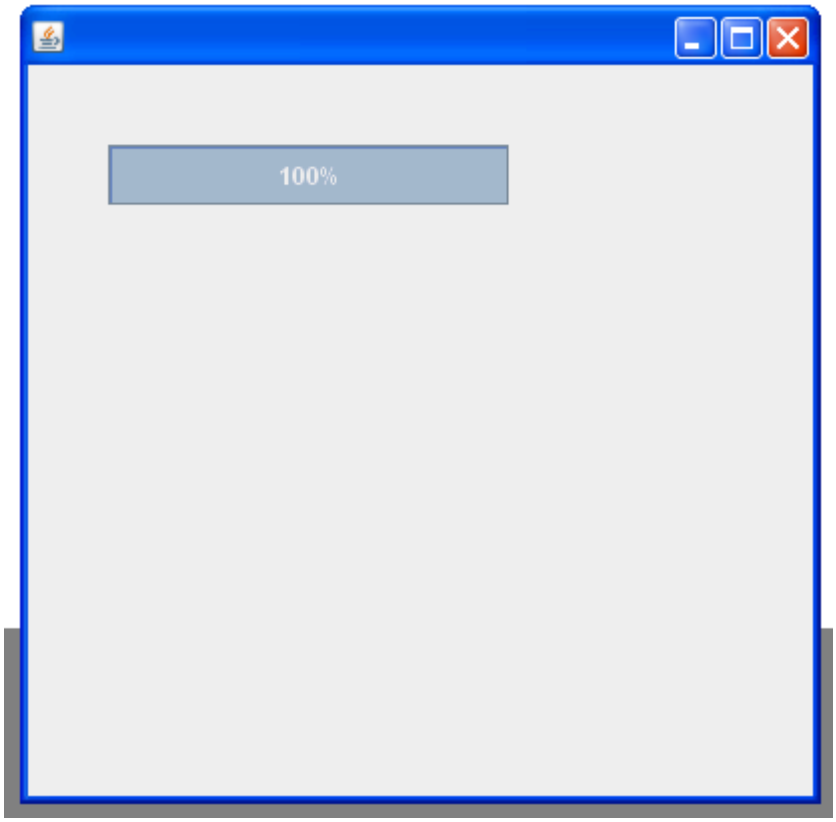
```
21. c.setBackground(color);
22. }
23.
24. public static void main(String[] args) {
25.     JColorChooserExample ch=new JColorChooserExample();
26.     ch.setSize(400,400);
27.     ch.setVisible(true);
28.     ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
29. }
30. }
```

JProgressBar class:

The JProgressBar class is used to display the progress of the task.

```
1.  import javax.swing.*;
2.  public class MyProgress extends JFrame{
3.  JProgressBar jb;
4.  int i=0,num=0;
5.
6.  MyProgress(){
7.  jb=new JProgressBar(0,2000);
8.  jb.setBounds(40,40,200,30);
9.
10. jb.setValue(0);
11. jb.setStringPainted(true);
12.
13. add(jb);
14. setSize(400,400);
15. setLayout(null);
16. }
17.
18. public void iterate(){
19. while(i<=2000){
20.   jb.setValue(i);
21.   i=i+20;
22.   try{Thread.sleep(150);}catch(Exception e){}
23. }
24. }
25. public static void main(String[] args) {
26.     MyProgress m=new MyProgress();
27.     m.setVisible(true);
28.     m.iterate();
29. }
```

30.}



Signed applet:

If an applet attempts to access local system resources, the applet must be signed and the local system must have a policy file configured to allow the access.

For an applet to access local system resources outside the directory from which the applet is launched, the applet must be granted explicit access to those resources. An applet is granted access to specific resources by setting up a policy file that contains the URLs to one or more resources that specifies the required permissions. When the applet is launched, it must be launched with the policy file to gain the access. Sometimes the permissions in the policy file require an applet be signed to gain access to some URLs and not signed for access to others.

If a signature is needed for the access, the applet has to be bundled into a Java ARchive (JAR) file before it can be signed.

Usually an applet is bundled and signed by one person and handed off to another who verifies the signature and runs the applet. In this example, Susan performs Steps 1 through 5 and Ray performs Steps 6 through 8. But, to keep things simple, all steps occur in the same working directory.

1. Compile the applet
2. Create a JAR file
3. Generate Keys
4. Sign the JAR file
5. Export the Public Key Certificate
6. Import the Certificate as a Trusted Certificate
7. Create the policy file
8. Run the applet

System class:

The java.lang.System class contains several useful class fields and methods. It cannot be instantiated.Facilities provided by System −

- standard output
- error output streams
- standard input and access to externally defined properties and environment variables.
- A utility method for quickly copying a portion of an array.
- a means of loading files and libraries

Class Declaration

Following is the declaration for java.lang.System class −

public final class System
  extends Object

Field

Following are the fields for java.lang.System class −

- static PrintStream err − This is the "standard" error output stream.
- static InputStream in − This is the "standard" input stream.
- static PrintStream out − This is the "standard" output stream.

Class methods:

1.
static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

This method copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array.

2.static String clearProperty(String key)

This method removes the system property indicated by the specified key.

3.static Console console()

This method returns the unique Console object associated with the current Java virtual machine, if any.

4.static long currentTimeMillis()

This method returns the current time in milliseconds.

5.**static void exit(int status)**

This method terminates the currently running Java Virtual Machine.

6.static void gc()

This method runs the garbage collector.

7.static Properties getProperties()

This method determines the current system properties.

8.**static void load(String filename)**

This method loads a code file with the specified filename from the local file system as a dynamic library

9.static void loadLibrary(String libname)

This method loads the system library specified by the libname argument.

```
package com.tutorialspoint;
```

```
import java.lang.*;

public class SystemDemo {

  public static void main(String[] args) {

    int arr1[] = { 0, 1, 2, 3, 4, 5 };
    int arr2[] = { 5, 10, 20, 30, 40, 50 };

    // copies an array from the specified source array
    System.arraycopy(arr1, 0, arr2, 0, 1);
    System.out.print("array2 = ");
    System.out.print(arr2[0] + " ");
    System.out.print(arr2[1] + " ");
    System.out.print(arr2[2] + " ");
    System.out.print(arr2[3] + " ");
    System.out.print(arr2[4] + " ");
```

JAVA COLLECTIONS:

The **Collection in Java** is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

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

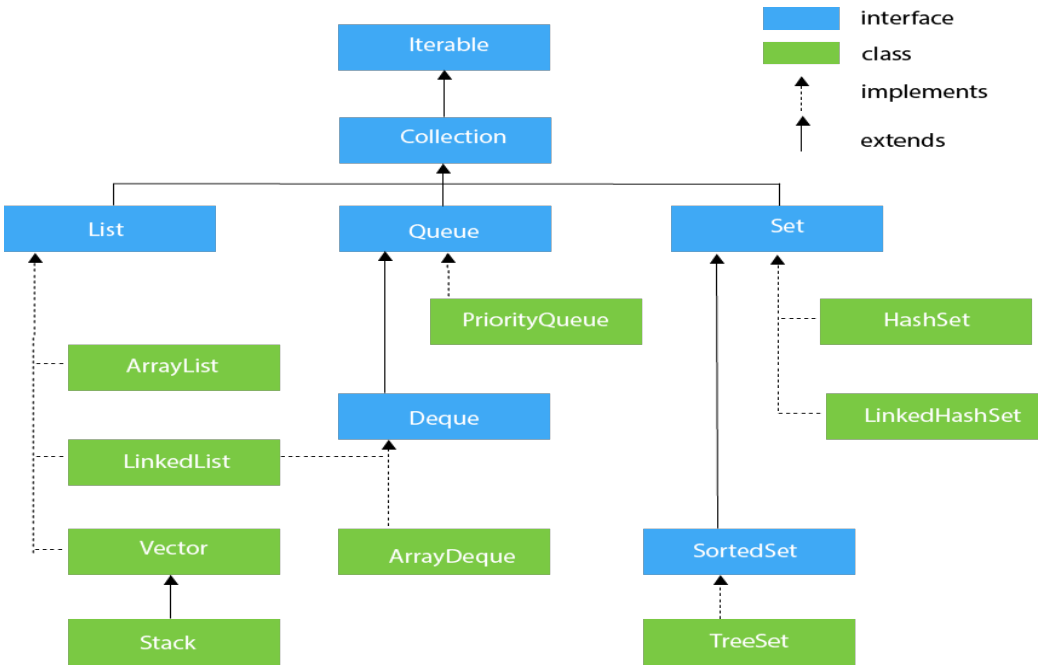
The Collection framework represents a unified architecture for storing and manipulating

a 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 the 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(E e)	It is used to insert an element in this

		collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.
9	public boolean contains(Object element)	It is used to search an element.
10	public boolean containsAll(Collection<?> c)	It is used to search the specified collection in the collection.
11	public Iterator iterator()	It returns an iterator.
12	public Object[] toArray()	It converts collection into array.
13	public <T> T[] toArray(T[] a)	It converts collection into array. Here, the runtime type of the returned array is

		that of the specified array.
14	public boolean isEmpty()	It checks if collection is empty.
15	default Stream<E> parallelStream()	It returns a possibly parallel Stream with the collection as its source.
16	default Stream<E> stream()	It returns a sequential Stream with the collection as its source.
17	default Spliterator<E> spliterator()	It generates a Spliterator over the specified elements in the collection.
18	public boolean equals(Object element)	It matches two collections.
19	public int hashCode()	It returns the hash code number of the collection.

ITERATORS:

Iterator' is an interface which belongs to collection framework. It allows us to traverse the collection, access the data element and remove the data elements of the collection. **java.util** package has **public interface Iterator** and contains three methods:

1. **boolean hasNext()**: It returns true if Iterator has more element to iterate.
2. **Object next()**: It returns the next element in the collection until the hasNext()method return true. This method throws 'NoSuchElementException' if there is no next element.
3. **void remove()**: It removes the current element in the collection. This method throws 'IllegalStateException' if this function is called before next( ) is invoked.

```
// Java code to illustrate the use of iterator
import java.io.*;
import java.util.*;

class Test {
   public static void main(String[] args)
   {
      ArrayList<String> list = new ArrayList<String>();

      list.add("A");
      list.add("B");
      list.add("C");
```

```
        list.add("D");
        list.add("E");

        // Iterator to traverse the list
        Iterator iterator = list.iterator();

        System.out.println("List elements : ");

        while (iterator.hasNext())
           System.out.print(iterator.next() + " ");

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

Output:

```
List elements :
A B C D E
```

ListIterator

'ListIterator' in Java is an Iterator which allows users to traverse Collection in both direction. It contains the following methods:

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

```
// Java code to illustrate the use of ListIterator
import java.io.*;
import java.util.*;

class Test {
    public static void main(String[] args)
    {
        ArrayList<String> list = new ArrayList<String>();

        list.add("A");
        list.add("B");
        list.add("C");
```

```
        list.add("D");
        list.add("E");

        // ListIterator to traverse the list
        ListIterator iterator = list.listIterator();

        // Traversing the list in forward direction
        System.out.println("Displaying list elements in forward direction : ");

        while (iterator.hasNext())
            System.out.print(iterator.next() + " ");

        System.out.println();

        // Traversing the list in backward direction
        System.out.println("Displaying list elements in backward direction : ");

        while (iterator.hasPrevious())
            System.out.print(iterator.previous() + " ");

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

Output:

```
Displaying list elements in forward direction :
A B C D E
Displaying list elements in backward direction :
E D C B A
```

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

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

Constructor	Description
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.

Example program:

```
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());
```

```
 }
} }
```

SETS:

- Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
- Basically, Set is implemented by HashSet, LinkedHashSet or TreeSet (sorted representation).
- Set has various methods to add, remove clear, size, etc to enhance the usage of this interface

```
import java.util.*;
public class SetDemo {

  public static void main(String args[]) {
     int count[] = {34, 22,10,60,30,22};
     Set<Integer> set = new HashSet<Integer>();
     try {
        for(int i = 0; i < 5; i++) {
           set.add(count[i]);
        }
        System.out.println(set);
        TreeSet sortedSet = new TreeSet<Integer>(set);
        System.out.println("The sorted list is:");
        System.out.println(sortedSet);

        System.out.println("The First element of the set is: "+ (Integer)sortedSet.first());
        System.out.println("The last element of the set is: "+ (Integer)sortedSet.last());
     }
     catch(Exception e) {}
  }
```

This will produce the following result –

Output

```
[34, 22, 10, 60, 30]
The sorted list is:
[10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60
```

HashSet :

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.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.

Example:

```
1. import java.util.*;
2. class HashSet1{
3.  public static void main(String args[]){
4.   //Creating HashSet and adding elements
5.     HashSet<String> set=new HashSet();
6.           set.add("One");
7.           set.add("Two");
8.           set.add("Three");
9.           set.add("Four");
10.          set.add("Five");
11.          Iterator<String> i=set.iterator();
12.          while(i.hasNext())
13.          {
14.          System.out.println(i.next());
15.          }
16. }
17.}
```

```
Five
One
Four
Two
Three
```

Hashtable

Java Hashtable class implements a hashtable, which maps keys to values. It inherits Dictionary class and implements the Map interface.

- A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashcode() method. A Hashtable contains values based on the key.
- Java Hashtable class contains unique elements.
- Java Hashtable class doesn't allow null key or value.
- Java Hashtable class is synchronized.
- The initial default capacity of Hashtable class is 11 whereas loadFactor is 0.75.

Hashtable class declaration

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

```
1. public class Hashtable<K,V> extends Dictionary<K,V> implements Map<K,V>, Cloneable,
   Serializable
```

```
2. import java.util.*;
3. class Hashtable1{
4.  public static void main(String args[]){
5.   Hashtable<Integer,String> hm=new Hashtable<Integer,String>();
6.
7.   hm.put(100,"Amit");
8.   hm.put(102,"Ravi");
9.   hm.put(101,"Vijay");
10.  hm.put(103,"Rahul");
11.
12.  for(Map.Entry m:hm.entrySet()){
13.   System.out.println(m.getKey()+" "+m.getValue());
14.  }
15. }
16.}
```

Output:

```
103 Rahul
102 Ravi
101 Vijay
100 Amit
```

Queue

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.

```
import java.util.LinkedList;
import java.util.Queue;

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

```
 {
   Queue<Integer> q = new LinkedList<>();

   // Adds elements {0, 1, 2, 3, 4} to queue
   for (int i=0; i<5; i++)
    q.add(i);

   // Display contents of the queue.
   System.out.println("Elements of queue-"+q);

   // To remove the head of queue.
   int removedele = q.remove();
   System.out.println("removed element-" +
removedele);

   System.out.println(q);

   // To view the head of queue
   int head = q.peek();
   System.out.println("head of queue-" + head);

   // Rest all methods of collection interface,
   // Like size and contains can be used with this
   // implementation.
   int size = q.size();
   System.out.println("Size of queue-" + size);
 }
}
```

Output:

```
Elements of queue-[0, 1, 2, 3, 4]

removed element-0

[1, 2, 3, 4]

head of queue-1

Size of queue-4
```

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`

Vectors:

Vector implements a dynamic array. It is similar to Array List, but with two differences –

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

Vector( )

This constructor creates a default vector, which has an initial size of 10.

Vector(int size)

This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.

Vector(int size, int incr)

This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.

int size()

Returns the number of components in this vector.

int capacity()

Returns the current capacity of this vector.

Enumeration elements()

Returns an enumeration of the components of this vector.

void addElement(Object obj)

Adds the specified component to the end of this vector, increasing its size by one.

Object elementAt(int index)

Returns the component at the specified index.

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

```
2. class TestVector1{
3.  public static void main(String args[]){
4.   Vector<String> v=new Vector<String>();//creating vector
5.   v.add("umesh");//method of Collection
6.   v.addElement("irfan");//method of Vector
7.   v.addElement("kumar");
8.   //traversing elements using Enumeration
9.   Enumeration e=v.elements();
10.  while(e.hasMoreElements()){
11.   System.out.println(e.nextElement());
12.  }
13. }
14.}
```

Comparable interface

Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.

compareTo(Object obj) method

public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

Java Comparable Example

Let's see the example of the Comparable interface that sorts the list elements on the basis of age.

File: Student.java

```
1. class Student implements Comparable<Student>{
2. int rollno;
3. String name;
4. int age;
5. Student(int rollno,String name,int age){
6. this.rollno=rollno;
```

```
7.  this.name=name;
8.  this.age=age;
9.  }
10.
11. public int compareTo(Student st){
12. if(age==st.age)
13. return 0;
14. else if(age>st.age)
15. return 1;
16. else
17. return -1;
18. }
19. }
```

File: TestSort1.java

```
1.  import java.util.*;
2.  public class TestSort1{
3.  public static void main(String args[]){
4.  ArrayList<Student> al=new ArrayList<Student>();
5.  al.add(new Student(101,"Vijay",23));
6.  al.add(new Student(106,"Ajay",27));
7.  al.add(new Student(105,"Jai",21));
8.
9.  Collections.sort(al);
10. for(Student st:al){
11. System.out.println(st.rollno+" "+st.name+" "+st.age);
12. }
13. }
14. }
```

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