UNIT 4

Java Packages: I/O classes: Byte Stream and Character Stream, Predefined Stream, reading console input, writing console output. **Applet:** Applet Life Cycle, creating an applet, Using image and sound in applet. **Lang:** Various classes, Importance class Definition, **Util:** Framework, Event Model, Scanner Class **AWT:** Exploring AWT, Event handling – The delegation-event model, Event classes, Source of event, Event listener interfaces, handling mouse and keyboard event, Adapter class

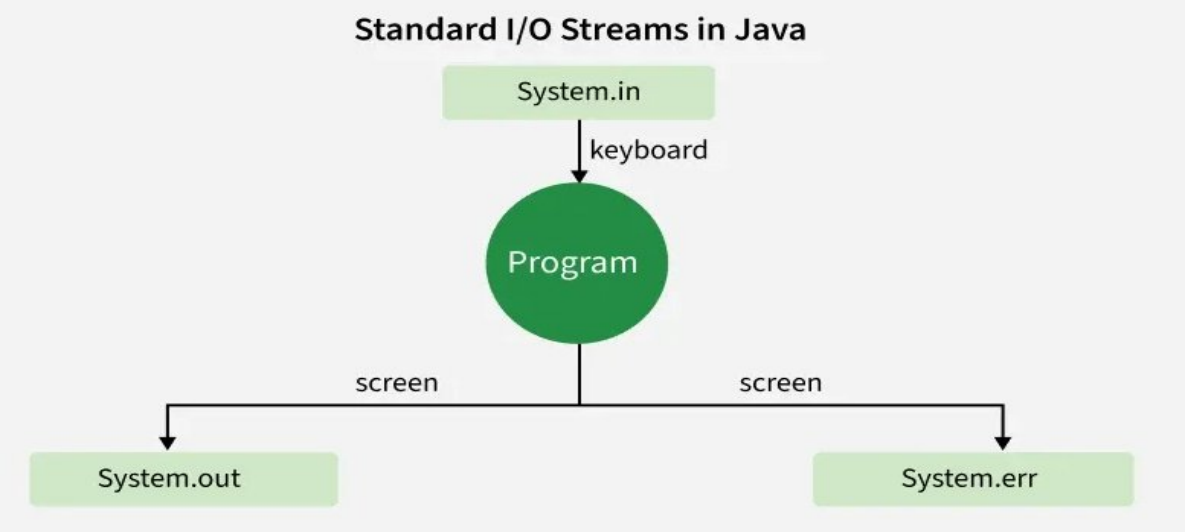
Types of Streams

Operations के प्रकार के आधार पर, **streams** को दो मुख्य वर्गों में बाँटा जा सकता है:

1. Input Stream: ये streams उन डेटा को पढ़ने के लिए उपयोग किए जाते हैं, जिन्हें किसी **source array**, **file**, या किसी **peripheral device** से **input** के रूप में लिया जाना होता है। उदाहरण के लिए: **FileInputStream, BufferedInputStream, ByteArrayInputStream** आदि।

InputStream

2. Output Stream: ये streams उन डेटा को **write** करने के लिए उपयोग किए जाते हैं, जिन्हें किसी **array**, **file**, या किसी **output peripheral device** में **output** के रूप में भेजा जाना होता है। उदाहरण के लिए: **FileOutputStream, BufferedOutputStream, ByteArrayOutputStream** आदि।

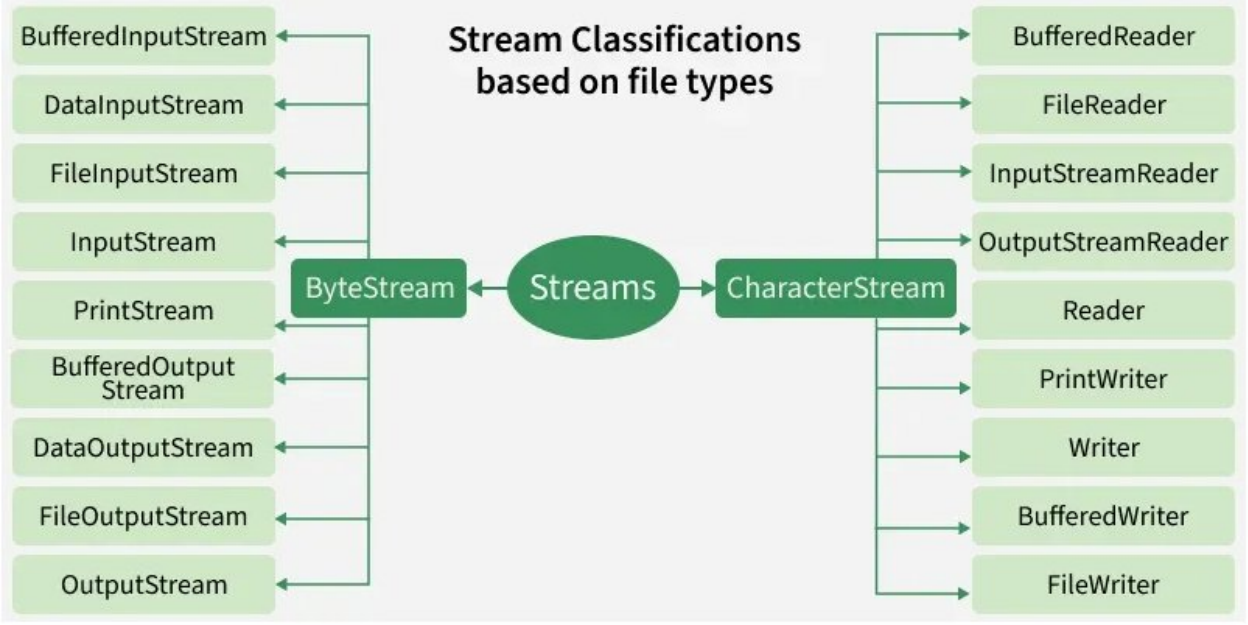


Output Stream

Streams के प्रकारों के बारे में अधिक जानकारी के लिए आप देख सकते हैं: **Java.io.InputStream Class** और **Java.io.OutputStream Class**।

Types of Streams Depending on the Types of File

File के प्रकारों के आधार पर, **Streams** को दो मुख्य वर्गों में बाँटा जा सकता है, जिन्हें आगे अन्य वर्गों में विभाजित किया जा सकता है। नीचे दिए गए diagram और उसके बाद के explanations में इसे समझाया गया है।



Classification

1. ByteStream:

Java में **Byte streams** का उपयोग **8-bit bytes** के input और output के लिए किया जाता है। ये **raw binary data** जैसे images, audio, और video को handle करने के लिए उपयुक्त हैं, और इसके लिए **InputStream** और **OutputStream** जैसी classes का उपयोग किया जाता है।

Here is the list of various ByteStream Classes:

Stream class	Description
BufferedInputStream	Used to read data more efficiently with buffering.
DataInputStream	Provides methods to read Java primitive data types.
FileInputStream	This is used to read from a file.
InputStream	This is an abstract class that describes stream input.
PrintStream	This contains the most used print() and println() method
BufferedOutputStream	This is used for Buffered Output Stream.

Stream class	Description
DataOutputStream	This contains method for writing java standard data types.
FileOutputStream	This is used to write to a file.
OutputStream	This is an abstract class that describes stream output.

Example: Java Program illustrating the Byte Stream to copy contents of one file to another file.

```
import java.io.*;
public class Geeks {
   public static void main(
      String[] args) throws IOException
   {

      FileInputStream sourceStream = null;
      FileOutputStream targetStream = null;

      try {
         sourceStream
            = new FileInputStream("sorcefile.txt");
         targetStream
            = new FileOutputStream("targetfile.txt");

         // Reading source file and writing content to target file byte by byte
         int temp;
         while ((
                  temp = sourceStream.read())
               != -1)
            targetStream.write((byte)temp);
      }
      finally {
         if (sourceStream != null)
            sourceStream.close();
         if (targetStream != null)
            targetStream.close();
      }
   }
}
```

Output:

Shows contents of file test.txt

2. Character Stream:

Java में **Character streams** का उपयोग **16-bit Unicode characters** के input और output के लिए किया जाता है।
ये **text data** को handle करने के लिए सबसे उपयुक्त होते हैं, और इसके लिए **Reader** और **Writer** जैसी classes का उपयोग किया जाता है, जो **character encoding** और **decoding** को स्वचालित रूप से संभालती हैं।

Here is the list of various CharacterStream Classes:

Stream class	Description
BufferedReader	It is used to handle buffered input stream.
FileReader	This is an input stream that reads from file.
InputStreamReader	This input stream is used to translate byte to character.
OutputStreamWriter	Converts character stream to byte stream.
Reader	This is an abstract class that define character stream input.
PrintWriter	This contains the most used print() and println() method
Writer	This is an abstract class that define character stream output.
BufferedWriter	This is used to handle buffered output stream.
FileWriter	This is used to output stream that writes to file.

Example:

```
import java.io.*;

public class Geeks
{
   public static void main(String[] args) throws IOException
   {
      FileReader sourceStream = null;

            try {
```

```
            sourceStream = new FileReader("test.txt");

            // Reading sourcefile and writing content to target file character by character.
            int temp;

            while (( temp = sourceStream.read())!= -1 )
               System.out.println((char)temp);
        }
        finally {

            // Closing stream as no longer in use
            if (sourceStream != null)
               sourceStream.close();
        }
    }
}
```

Reading console input in java

Java में, कंसोल से इनपुट पढ़ने के तीन तरीके हैं। इन तीन तरीकों से हम कंसोल से इनपुट डेटा पढ़ सकते हैं:

- **BufferedReader class** का उपयोग करना
- **Scanner class** का उपयोग करना
- **Console class** का उपयोग करना

आइए, प्रत्येक method को एक उदाहरण के साथ देखें।

1. Reading console input using BufferedReader class in java

BufferedReader class का उपयोग करके इनपुट डेटा पढ़ना पारंपरिक तरीका है। इस तरीके में **System.in** (standard input stream) को **InputStreamReader** में लपेटा जाता है, और फिर उसे **BufferedReader** में लपेटा जाता है, जिससे हम कंसोल से इनपुट पढ़ सकते हैं।
BufferedReader class **java.io** पैकेज में परिभाषित है।

नीचे दिया गया उदाहरण कोड समझने के लिए है कि BufferedReader class का उपयोग करके कंसोल से इनपुट कैसे पढ़ा जा सकता है।

Example

```
import java.io.*;
```

```
public class ReadingDemo {

	public static void main(String[] args) throws IOException {

		BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));

		String name = "";

		try {
			System.out.print("Please enter your name : ");
			name = in.readLine();
			System.out.println("Hello, " + name + "!");
		}
		catch(Exception e) {
			System.out.println(e);
		}
		finally {
			in.close();
		}
	}
}
```

When we run the above program, it produce the following output.

2. Reading console input using Scanner class in java

Scanner class का उपयोग करके इनपुट डेटा पढ़ना सबसे सामान्य तरीका है। इस तरीके में **System.in** (standard input stream) को **Scanner** में लपेटा जाता है, जिससे हम कंसोल से इनपुट पढ़ सकते हैं।
Scanner class **java.util** पैकेज में परिभाषित है।

नीचे दिया गया उदाहरण कोड यह समझने के लिए है कि Scanner class का उपयोग करके कंसोल से इनपुट कैसे पढ़ा जा सकता है।

Example

```
import java.util.Scanner;

public class ReadingDemo {

	public static void main(String[] args) {

		Scanner in = new Scanner(System.in);

		String name = "";
			System.out.print("Please enter your name : ");
			name = in.next();
			System.out.println("Hello, " + name + "!");

	}
}
```

3. Reading console input using Console class in java

Console class का उपयोग करके इनपुट डेटा पढ़ना सबसे सामान्य तरीका है। इस class को Java 1.6 संस्करण में प्रस्तुत किया गया था।
Console class **java.io** पैकेज में परिभाषित है।

नीचे दिया गया उदाहरण कोड यह समझने के लिए है कि Console class का उपयोग करके कंसोल से इनपुट कैसे पढ़ा जा सकता है।

Example

```
import java.io.*;

public class ReadingDemo {

        public static void main(String[] args) {

                String name;
    Console con = System.console();

                if(con != null) {
                        name = con.readLine("Please enter your name : ");
                        System.out.println("Hello, " + name + "!!");
                }
                else {
                        System.out.println("Console not available.");
                }
        }
}
```

Writing console output in java

Java में, कंसोल आउटपुट लिखने के दो तरीके हैं। इन दो तरीकों से हम कंसोल पर आउटपुट डेटा लिख सकते हैं:

• **print() और println() methods** का उपयोग करना
• **write() method** का उपयोग करना

आइए, हम प्रत्येक method को एक उदाहरण के साथ विस्तार से समझते हैं।

1. Writing console output using print() and println() methods

PrintStream एक built-in class है जो **print()** और **println()** methods प्रदान करती है, जो कंसोल पर आउटपुट लिखने के लिए उपयोग की जाती हैं। **print()** और **println()** methods कंसोल आउटपुट के लिए सबसे अधिक उपयोग की जाने वाली methods हैं।
दोनों **print()** और **println()** methods **System.out** stream के साथ उपयोग होती हैं।

- **print()** method कंसोल पर आउटपुट को उसी पंक्ति में लिखता है। इस method का उपयोग केवल कंसोल आउटपुट के लिए किया जा सकता है।
- **println()** method कंसोल पर आउटपुट को एक नई पंक्ति (new line) में लिखता है। इस method का उपयोग कंसोल के साथ-साथ अन्य आउटपुट स्रोतों के लिए भी किया जा सकता है।

आइए, हम नीचे दिए गए कोड को देखें जो **print()** और **println()** methods को दर्शाता है:

Example

```
public class WritingDemo {

	public static void main(String[] args) {

		int[] list = new int[5];

		for(int i = 0; i < 5; i++)
			list[i] = i*10;
		for(int i:list)
			System.out.print(i);		//prints in same line

		System.out.println("");
		for(int i:list)
			System.out.println(i);		//Prints in separate lines


	}
}
```

2. Writing console output using write() method

वैकल्पिक रूप से, **PrintStream** class **write()** method प्रदान करती है, जिससे कंसोल पर आउटपुट लिखा जा सकता है।
write() method एक integer को argument के रूप में लेता है और इसके ASCII equivalent character को कंसोल पर लिखता है। यह method **escape sequences** को भी स्वीकार करता है।

आइए, नीचे दिए गए कोड को देखें जो **write()** method को दर्शाता है:

Example

```
public class WritingDemo {

	public static void main(String[] args) {

		int[] list = new int[26];

		for(int i = 0; i < 26; i++) {
			list[i] = i + 65;
		}

		for(int i:list) {
			System.out.write(i);
			System.out.write('\n');
		}
	}
}
```

Applet

Introduction

एक **Applet** एक विशेष प्रकार का Java प्रोग्राम है जो वेब ब्राउज़र पर चलाया जाता है। **Applet class** ब्राउज़र और applet के बीच मानक इंटरफ़ेस प्रदान करती है।

एक applet class में **main method** नहीं होती है और यह **java.applet.Applet** class को एक्सटेंड करती है। एक applet प्रोग्राम **applet viewer** के माध्यम से चलता है।

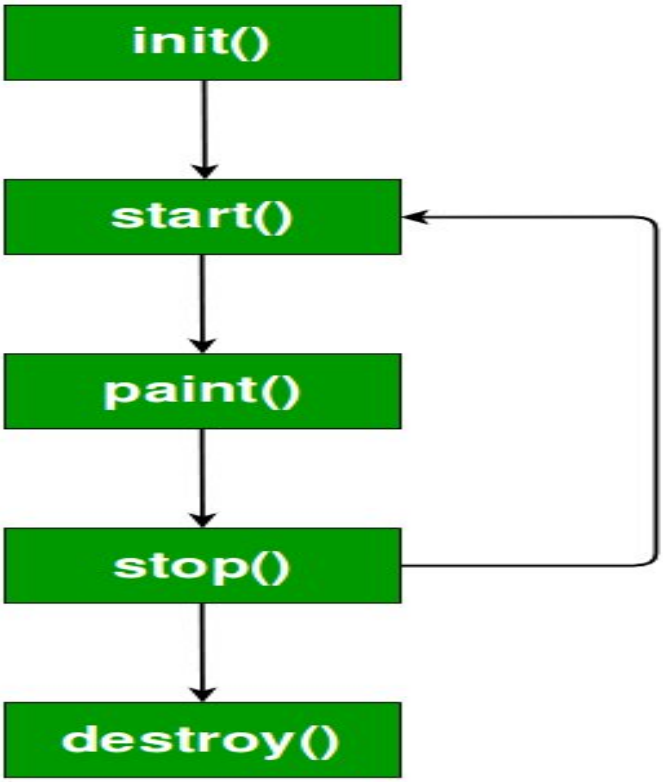
Advantages of Applets

• Applets अधिकांश वेब ब्राउज़रों द्वारा समर्थित होते हैं।
• Applets का प्रतिक्रिया समय बहुत कम होता है, क्योंकि यह क्लाइंट साइड पर काम करते हैं।
• Applets सुरक्षा के मामले में काफी मजबूत होते हैं क्योंकि इनका संसाधनों तक सीमित पहुंच होती है।
• एक अप्रतिष्ठित applet को स्थानीय मशीन तक कोई भी पहुंच नहीं होती है।
• Applets का आकार छोटा होता है, जिससे इसे नेटवर्क पर स्थानांतरित करना आसान होता है।

Limitations of Applets

• Applets को चलाने के लिए एक Java प्लगइन की आवश्यकता होती है।
• Applets फाइल सिस्टम का समर्थन नहीं करते हैं।
• Applet स्वयं स्थानीय सिस्टम पर किसी भी एप्लिकेशन को नहीं चला सकता या संशोधित नहीं कर सकता।
• यह सिस्टम के **native methods** के साथ काम नहीं कर सकता।
• एक applet क्लाइंट-साइड संसाधनों तक पहुँच नहीं सकता।

Applet Life Cycle



It is derived from the Applet class. When an Applet is created, it goes through different stages; it is known as applet life cycle.

1. Born or initialization state
2. Running state
3. Stopped state
4. Destroyed state
5. Display state

Applet Life Cycle Methods

1. init ( )
init() method applet के लोड होते समय उसे इनिशियलाइज करने के लिए जिम्मेदार होती है। यह method केवल एक बार, ब्राउज़र या applet viewer द्वारा कॉल की जाती है।

Syntax

```
public void init( )
{
    //actions
}
```

2. start ( )
यह **init()** method के बाद कॉल किया जाता है, जब applet initialization state से running state में प्रवेश कर जाता है। यह applet के execution को प्रारंभ करता है और applet सक्रिय (active) स्थिति में आ जाता है।

Syntax

```
public void start()
{
    //actions
}
```

3. paint( )
यह **paint()** method स्क्रीन पर आउटपुट दिखाने के लिए उपयोग की जाती है। यह **java.awt.Graphics** ऑब्जेक्ट को एक पैरामीटर के रूप में लेती है। यह method Applet का GUI (जैसे रंगीन बैकग्राउंड, चित्रण आदि) बनाने में मदद करती है।

Syntax

```
public void paint(Graphics g )
{
     //Display statement
}
```

4. stop( )
यह **stop()** method execution को रोकने के लिए जिम्मेदार होती है और applet को अस्थायी रूप से निष्क्रिय (inactive) स्थिति में बदल देती है। इसे तब कॉल किया जाता है जब applet को रोका जाता है।

Syntax

```
public void stop( )
{
    //actions
}
```

5. destroy( )
यह **destroy()** method applet को मेमोरी से नष्ट और हटा देती है। इसे केवल एक बार कॉल किया जाता है। यह applet के जीवन चक्र का अंत होता है और इसके बाद applet मृत (dead) हो जाता है।

Syntax

```
public void destroy()
{
    //actions
}
```

Developing an Applet Program

Applet प्रोग्राम विकसित करने के चरण:

1. एक applet (java) कोड बनाएं, यानी .java फ़ाइल लिखें।
2. कोड को संकलित (compile) करें ताकि .class फ़ाइल उत्पन्न हो सके।
3. एक HTML फ़ाइल डिज़ाइन करें जिसमें `<APPLET>` टैग का उपयोग किया जाए।

Running the Applet Program

There are two ways to run the applet program
1. By HTML
2. Appletviewer tool

Example : Getting a message printed through Applet

```
import java.awt.*;
import java.applet.*;
public class AppletDemo
{
    public void paint(Graphics g)
    {
        g.drawString("Welcome to TutorialRide", 50, 50);
    }
}
```

1. Save the above program: **AppletDemo.java**
2. Compile: > javac AppletDemo.java

//AppletDemo.html

```
<html>
    <head>
        <title> AppletDemo </title>
    </head>
    <body>
        <applet code="AppletDemo.class" width="200" height="250">
        </applet>
    </body>
</html>
```

Run the complete program:
> appletviewer AppletDemo.html

NOTE: To execute the applet program through **appletviewer** tool, write it on command prompt.
C :\> javac AppletDemo.java
C :\> appletviewer AppletDemo.java

Displaying Images in Java Applets

Java Applets इमेजेस को **Image class** और **paint()** method का उपयोग करके प्रदर्शित कर सकते हैं। यह इस प्रकार काम करता है:

इमेज़ दिखाने के चरण:

- **getImage()** method का उपयोग करके applet class के माध्यम से इमेज को लोड करें।
- **paint()** method का उपयोग करें और **Graphics class** के **drawImage()** method का उपयोग करके इमेज को applet विंडो पर ड्रॉ करें।
- **Code Example:**

```
Copy code
import java.applet.Applet;
import java.awt.*;

public class ImageApplet extends Applet {
    Image img;

    public void init() {
        img = getImage(getDocumentBase(), "example.jpg"); // Load image
    }

    public void paint(Graphics g) {
        g.drawImage(img, 50, 50, this); // Draw image at (50, 50)
    }
}
```

Playing Audio in Java Applets

Java Applets **AudioClip interface** का उपयोग करके ऑडियो फ़ाइलें चला सकते हैं। ऑडियो फ़ाइलें **.au** फ़ॉर्मेट या अन्य समर्थित फ़ॉर्मेट्स में होनी चाहिए।

ऑडियो चलाने के चरण:

- **getAudioClip()** method का उपयोग करके applet class के माध्यम से ऑडियो फ़ाइल लोड करें।
- **play()**, **loop()**, या **stop()** जैसे methods का उपयोग करके ऑडियो प्लेबैक को नियंत्रित करें।
- **Code Example:**

```
Copy code
import java.applet.Applet;
import java.applet.AudioClip;

public class AudioApplet extends Applet {
    AudioClip audio;

    public void init() {
        audio = getAudioClip(getDocumentBase(), "example.au"); // Load audio
    }

    public void start() {
        audio.play(); // Play audio once
    }

    public void stop() {
        audio.stop(); // Stop audio
    }
}
```

Lang Package (java.lang)

Overview of the java.lang Package

java.lang पैकेज Java प्रोग्रामिंग भाषा का एक बुनियादी हिस्सा है, जो आवश्यक क्लासों और इंटरफेसों को प्रदान करता है जिन्हें हर Java प्रोग्राम में स्वचालित रूप से इंपोर्ट किया जाता है। इसमें निम्नलिखित वर्ग शामिल हैं:

- **बुनियादी डेटा प्रकार** (Basic data types) के लिए क्लासेस।
- **गणितीय संचालन** (Mathematical operations) के लिए क्लासेस।
- **थ्रेडिंग** (Threading) के लिए क्लासेस।
- **स्ट्रिंग मैनिपुलेशन** (String manipulation) के लिए क्लासेस।
- **सिस्टम-स्तरीय संचालन** (System-level operations) के लिए क्लासेस।

.

Key Classes in java.lang

1. **Object**: क्लास हायरार्की का रूट। Java में हर क्लास **Object** को अप्रत्यक्ष रूप से एक्सटेंड करती है, जो **equals()**, **hashCode()**, और **toString()** जैसी मेथड्स प्रदान करती है।
2. **String**: अपरिवर्तनीय (immutable) चरित्रों की अनुक्रमणिका। यह टेक्स्ट मैनिपुलेशन के लिए व्यापक रूप से उपयोग किया जाता है और इसमें **substring()**, **charAt()**, और **concat()** जैसी मेथड्स शामिल हैं।

3. **StringBuilder और StringBuffer**: **String** के लिए परिवर्तनीय (mutable) विकल्प। **StringBuilder** तेज़ होता है लेकिन थ्रेड-सुरक्षित नहीं है, जबकि **StringBuffer** थ्रेड-सुरक्षित होता है।
4. **Math और StrictMath**: गणितीय संचालन के लिए मेथड्स प्रदान करते हैं जैसे **sqrt()**, **log()**, और त्रिकोणमितीय कार्य।
5. **Wrapper Classes**: इसमें **Integer**, **Double**, **Boolean**, और अन्य शामिल हैं, जो मूल प्रकारों (primitive types) को ऑब्जेक्ट्स में लपेटते हैं। ये क्लासेस टाइप कन्वर्शन और पार्सिंग के लिए यूटिलिटी मेथड्स भी प्रदान करती हैं।
6. **Thread और ThreadGroup**: मल्टीथ्रेडिंग को सुविधाजनक बनाते हैं, थ्रेड्स के निर्माण और प्रबंधन की अनुमति देते हैं। **ThreadGroup** थ्रेड्स को समूहों में व्यवस्थित करता है ताकि प्रबंधन आसान हो।
7. **Runtime**: Java रनटाइम वातावरण के साथ इंटरैक्ट करने के लिए मेथड्स प्रदान करता है, जैसे कि मेमोरी प्रबंधन और प्रक्रिया निष्पादन।
8. **Class और ClassLoader**: रनटाइम पर क्लासेस और इंटरफेस को प्रदर्शित करते हैं। **ClassLoader** क्लासेस को डायनेमिक रूप से लोड करने के लिए जिम्मेदार होता है।
9. **Throwable**: Java में सभी अपवादों (exceptions) और त्रुटियों (errors) के लिए सुपरक्लास। इसके उपक्लासेस में **Exception** और **Error** शामिल हैं।

Common Use Cases

• **गणितीय संचालन**: गणना जैसे लॉगारिथ्म, त्रिकोणमिति और राउंडिंग के लिए **Math** या **StrictMath** का उपयोग करें।
• **स्ट्रिंग मैनिपुलेशन**: टेक्स्ट डेटा को संभालने के लिए **String**, **StringBuilder**, या **StringBuffer** का उपयोग करें।
• **थ्रेडिंग**: समवर्ती प्रोग्रामिंग (concurrent programming) के लिए **Thread** और संबंधित क्लासेस का उपयोग करें।
• **सिस्टम संचालन**: पर्यावरण चर (environment variables) पढ़ने या बाहरी प्रक्रियाओं (external processes) को निष्पादित करने जैसे कार्यों के लिए **System** और **Runtime** का उपयोग करें।

java.util package

java.util पैकेज Java Standard Library का एक महत्वपूर्ण हिस्सा है, जो उपयोगिता क्लासेस का एक संग्रह प्रदान करता है जो विभिन्न डेटा संरचनाओं, एल्गोरिदम और अन्य कार्यक्षमताओं का समर्थन करता है। यह पैकेज Java एप्लिकेशन्स में प्रभावी डेटा प्रबंधन और हेरफेर (manipulation) के लिए अत्यंत महत्वपूर्ण है। यहाँ इसके मुख्य घटकों का एक व्यापक अवलोकन है:

1. Collections Framework

java.util पैकेज में Collections Framework के कोर इंटरफेस और क्लासेस शामिल हैं, जिसका उपयोग वस्तुओं के समूहों को स्टोर और हेरफेर करने के लिए किया जाता है। इसके प्रमुख इंटरफेस हैं:

- **Collection**: यह कलेक्शन हायरार्की का रूट इंटरफेस है। यह वस्तुओं का एक समूह (elements) प्रतिनिधित्व करता है। सामान्य मेथड्स में **add**, **remove**, और **size** शामिल हैं।
- **List**: **Collection** को एक्सटेंड करता है और एक क्रमबद्ध कलेक्शन (ordered collection) को प्रतिनिधित्व करता है। **ArrayList** और **LinkedList** जैसी इंप्लीमेंटेशंस सूची संचालन के लिए अलग-अलग प्रदर्शन विशेषताएँ प्रदान करती हैं।
- **Set**: **Collection** को एक्सटेंड करता है और यह एक कलेक्शन को प्रतिनिधित्व करता है जो डुप्लिकेट तत्वों (duplicates) की अनुमति नहीं देता। **HashSet** और **TreeSet** जैसी इंप्लीमेंटेशंस अद्वितीय तत्वों (unique elements) को स्टोर करने के लिए अलग-अलग दृष्टिकोण प्रदान करती हैं।
- **Queue**: **Collection** को एक्सटेंड करता है और यह कलेक्शन को प्रस्तुत करता है जो प्रसंस्करण (processing) से पहले तत्वों को होल्ड करने के लिए डिज़ाइन किया गया है। **LinkedList** और **PriorityQueue** जैसी इंप्लीमेंटेशंस क्यू ऑपरेशंस को सपोर्ट करती हैं।
- **Map**: कुंजी-मूल्य (key-value) जोड़ियों के संग्रह को प्रस्तुत करता है। यह **Collection** को एक्सटेंड नहीं करता लेकिन कलेक्शन फ्रेमवर्क का हिस्सा है। **HashMap**, **TreeMap**, और **LinkedHashMap** जैसी इंप्लीमेंटेशंस कुंजी और मान (key-value) के बीच मैपिंग को संभालने के विभिन्न तरीके प्रदान करती हैं।

2. मुख्य क्लासेस:

- **ArrayList**: **List** इंटरफेस का एक रीसाइज़ेबल ऐरे इंप्लीमेंटेशन। यह तत्वों तक तेज़ रैंडम एक्सेस प्रदान करता है और उन परिदृश्यों के लिए उपयुक्त है जहाँ बार-बार रिट्रीवल की आवश्यकता होती है।
- **LinkedList**: **List** और **Deque** दोनों इंटरफेस को इंप्लीमेंट करता है, और एक डबल लिंक्ड लिस्ट संरचना प्रदान करता है। यह इन्सर्शन और डिलीशन के लिए कुशल है, लेकिन **ArrayList** की तुलना में रैंडम एक्सेस के लिए धीमा होता है।
- **HashSet**: **Set** इंटरफेस का एक हैश टेबल-आधारित इंप्लीमेंटेशन जो डुप्लिकेट तत्वों की अनुमति नहीं देता। यह **add**, **remove**, और **contains** जैसे मूल ऑपरेशंस के लिए स्थिर समय जटिलता (constant time complexity) प्रदान करता है।
- **TreeSet**: **Set** इंटरफेस को इंप्लीमेंट करता है और एक रेड-ब्लैक ट्री (Red-Black Tree) द्वारा समर्थित है। यह तत्वों को एक क्रमबद्ध (sorted) आदेश में बनाए रखता है और रेंज-आधारित ऑपरेशंस के लिए कुशल क्वेरींग (querying) प्रदान करता है।
- **PriorityQueue**: **Queue** इंटरफेस का एक इंप्लीमेंटेशन है जो प्राथमिकता हीप (priority heap) का उपयोग करता है। यह तत्वों को उनकी प्राकृतिक आदेश (natural ordering) या एक निर्दिष्ट तुलना करने वाले (comparator) के आधार पर क्रमबद्ध करता है।

- **HashMap**: **Map** इंटरफेस का एक इंप्लीमेंटेशन है जो हैश टेबल का उपयोग करता है। यह **null** मानों और कुंजियों (keys) की अनुमति देता है और **get** और **put** ऑपरेशंस के लिए स्थिर समय जटिलता प्रदान करता है।
- **TreeMap**: **Map** इंटरफेस का एक रेड-ब्लैक ट्री-आधारित इंप्लीमेंटेशन है जो कुंजियों को क्रमबद्ध (sorted) क्रम में बनाए रखता है। यह क्रमबद्ध कुंजियों (sorted keys) के साथ काम करने के लिए कई ऑपरेशंस प्रदान करता है।

3. Collections Utility क्लास मेथड्स

Collections क्लास कलेक्शन्स के साथ काम करने के लिए स्थैतिक (static) मेथड्स प्रदान करता है। मुख्य मेथड्स में शामिल हैं:

- **sort(List list)**: निर्दिष्ट सूची को आरोही (ascending) क्रम में सॉर्ट करता है।
- **shuffle(List<?> list)**: निर्दिष्ट सूची के तत्वों को यादृच्छिक (randomly) रूप से परिवर्तित करता है।
- **reverse(List<?> list)**: निर्दिष्ट सूची के तत्वों का क्रम उलट (reverse) करता है।
- **max(Collection<? extends T> coll)**: दिए गए कलेक्शन का अधिकतम तत्व लौटाता है, इसके तत्वों के प्राकृतिक आदेश (natural ordering) के अनुसार।

Java Event Model

Introduction

इवेंट डेलीगेशन मॉडल (Event Delegation Model) (या केवल **इवेंट मॉडल**) Java में **Abstract Window Toolkit (AWT)** और **Swing** का एक हिस्सा है। यह GUI घटकों जैसे बटन्स, टेक्स्ट फ़ील्ड्स, चेकबॉक्स, आदि द्वारा उत्पन्न इवेंट्स को संभालने के लिए एक तंत्र प्रदान करता है।

जब एक उपयोगकर्ता किसी GUI घटक (जैसे बटन पर क्लिक करना या कीबोर्ड दबाना) के साथ इंटरैक्ट करता है, तो एक इवेंट उत्पन्न होता है। यह इवेंट किसी ऐसे ऑब्जेक्ट द्वारा हैंडल किया जाना चाहिए जो उचित रूप से प्रतिक्रिया दे सके।

इवेंट डेलीगेशन मॉडल में मुख्य बातें:

- **इवेंट्स का उत्पन्न होना**: GUI घटक से उपयोगकर्ता के इंटरैक्शन के कारण इवेंट उत्पन्न होते हैं।
- **इवेंट हैंडलिंग**: इवेंट को संभालने के लिए एक ऑब्जेक्ट को असाइन किया जाता है, जिसे "इवेंट लिसनर→ कहा जाता है। यह ऑब्जेक्ट इवेंट के आधार पर एक प्रतिक्रिया (event handler) प्रदान करता है।

इवेंट डेलीगेशन मॉडल की संरचना:

1. **इवेंट स्रोत (Event Source)**: यह वह GUI घटक होता है जो इवेंट उत्पन्न करता है (जैसे बटन, टेक्स्ट फ़ील्ड, चेकबॉक्स)।
2. **इवेंट लिसनर (Event Listener)**: यह एक इंटरफेस होता है जिसे इवेंट को सुनने के लिए इम्प्लीमेंट किया जाता है। जब इवेंट उत्पन्न होता है, तो इवेंट लिसनर उस इवेंट को हैंडल करता है। उदाहरण: `ActionListener`, `MouseListener`, `KeyListener`, आदि।
3. **इवेंट ऑब्जेक्ट (Event Object)**: इवेंट लिसनर को एक इवेंट ऑब्जेक्ट प्राप्त होता है, जो इवेंट की जानकारी रखता है, जैसे कि बटन पर क्लिक किया गया या कीबोर्ड पर कौन सा कुंजी दबाया गया।
4. **इवेंट हैंडलिंग**: इवेंट लिसनर को इवेंट स्रोत के साथ जोड़ने के लिए, हम इवेंट स्रोत पर `addEventListener` या किसी अन्य संबंधित मेथड का उपयोग करते हैं। जब इवेंट होता है, तो लिसनर संबंधित मेथड को कॉल करता है जो इवेंट को संभालता है।

Basic Concepts

Event

एक **event** एक **object** होता है जो **source component** में **state** के **change** का वर्णन करता है।

Examples of events:

- Mouse click
- Key press
- Button click
- Window closing
- Text change in a text field

Each event is an instance of a class that inherits from **java.util.EventObject**.

Event Source

The **source** is the object on which the event occurs.

Example:

- A Button is the source of an ActionEvent.
- A TextField is the source of a TextEvent.

Event Listener

एक **listener** एक **object** होता है जिसे तब **notify** किया जाता है जब कोई **event** होती है। **Listeners** कुछ विशेष **event listener interfaces** को **implement** करते हैं, जो एक या अधिक **methods** को **define** करते हैं।

उदाहरण:
• **ActionListener** – बटन क्लिक के लिए
• **MouseListener** – माउस ईवेंट्स के लिए
• **KeyListener** – कीबोर्ड इनपुट के लिए

जब **source** पर कोई **event** होती है, तो **listener** की **method** अपने आप **call** हो जाती है।

Delegation Event Model

परिभाषा (Definition):
Delegation Event Model एक ऐसा **mechanism** है जिसके माध्यम से **Java** में **events** को **handle** किया जाता है।

इस **model** में:
• **Event source** एक **event** को **generate** करता है।
• **Event listener** उस **event** को **receive** करके **handle** करता है।
• **Source** उस **event** को **handle** करने की जिम्मेदारी **listener** को **delegate** करता है।

Diagram (Conceptual)

User Action → Event Source → Event Object → Event Listener
(Click) (Button) (ActionEvent) (ActionListener)

Steps to Handle an Event

1. **Implement the listener interface**
   - Create a class that implements an appropriate listener interface (e.g., ActionListener).
2. **Register the listener with the event source**
   - Use a method like addActionListener().
3. **Override the listener method**
   - Define what should happen when the event occurs.

Example: Button Click Event

```
import java.awt.*;
```

```
import java.awt.event.*;

class ButtonExample extends Frame implements ActionListener {
   Button b;

   ButtonExample() {
      b = new Button("Click Me");
      b.setBounds(100, 100, 80, 30);
      b.addActionListener(this);  // Step 2: Register listener
      add(b);

      setSize(300, 300);
      setLayout(null);
      setVisible(true);
   }

   // Step 3: Implement the listener method
   public void actionPerformed(ActionEvent e) {
      System.out.println("Button Clicked!");
   }

   public static void main(String[] args) {
      new ButtonExample();  // Step 1: Create frame
   }
}
```

Explanation:

- Button → Event Source
- ActionEvent → Event Object
- ActionListener → Event Listener
- addActionListener(this) → Registers the listener

Event Classes

All event classes inherit from java.util.EventObject.
Most GUI-related events extend from **java.awt.AWTEvent**.

Event Class	Description
ActionEvent	Generated by buttons, menu items, etc.

Event Class	Description
MouseEvent	Generated by mouse actions (click, press, move)
KeyEvent	Generated by keyboard actions
ItemEvent	Generated by selecting/deselecting items (like checkbox)
TextEvent	Generated when text in a text field changes
WindowEvent	Generated by window actions (open, close, minimize)
FocusEvent	Generated when a component gains or loses focus

Event Listener Interfaces

Each event type has a corresponding listener interface.

Listener Interface	Methods	Used For
ActionListener	actionPerformed(ActionEvent e)	Buttons, menu items
ItemListener	itemStateChanged(ItemEvent e)	Checkboxes, lists
MouseListener	mouseClicked(), mousePressed(), mouseReleased(), mouseEntered(), mouseExited()	Mouse actions
MouseMotionListener	mouseDragged(), mouseMoved()	Mouse movement
KeyListener	keyPressed(), keyReleased(), keyTyped()	Keyboard input
WindowListener	windowOpened(), windowClosing(), etc.	Window events

Listener Interface	Methods	Used For
FocusListener	focusGained(), focusLost()	Focus change
TextListener	textValueChanged(TextEvent e)	Text field changes

Adapter Classes

ऐसे **interfaces** जिनमें कई **methods** होती हैं, उनमें सभी **methods** को **implement** करना कभी-कभी **tedious** (थकाऊ) हो सकता है।
इसे आसान बनाने के लिए, **Java** कुछ **adapter classes** प्रदान करता है जिनमें सभी **interface methods** की **empty implementations** होती हैं।
आप किसी **adapter class** को **extend** करके केवल वही **methods override** कर सकते हैं जिनकी आपको आवश्यकता होती है।

Interface	Adapter Class
MouseListener	MouseAdapter
KeyListener	KeyAdapter
WindowListener	WindowAdapter
FocusListener	FocusAdapter

Example:

```
import java.awt.*;
import java.awt.event.*;

class MouseExample extends Frame {
   MouseExample() {
      addMouseListener(new MouseAdapter() {
         public void mouseClicked(MouseEvent e) {
            System.out.println("Mouse clicked at " + e.getX() + ", " + e.getY());
         }
```

```
        });
        setSize(300, 300);
        setVisible(true);
    }

    public static void main(String[] args) {
        new MouseExample();
    }
}
```

Handling Multiple Listeners

Multiple listeners can be registered to a single event source.
Each listener will be notified in the order of registration.

Example:

```
button.addActionListener(listener1);
button.addActionListener(listener2);
```

Custom Events

You can also create **user-defined (custom)** events by extending EventObject and defining your own listener interface.

Example:

```
class MyEvent extends EventObject {
    MyEvent(Object source) {
        super(source);
    }
}
```