

JAVA I/O, AWT & EVENT HANDLING

Streams, Applets एवं Packages

सरल Hindi-English व्याख्या के साथ

12 अध्यायों में Unit IV की व्यवस्थित शिक्षा

I/O

AWT EVENTS

MCQs

पुस्तक का परिचय

How to Use This Book

यह पुस्तक Unit IV syllabus को 12 अध्यायों में समझाती है: byte और character streams, console I/O, applet lifecycle, image और sound, java.lang classes, java.util framework एवं Scanner, AWT components, delegation event model, event classes, listener interfaces, mouse/keyboard events और adapter classes।

Hindi-English explanations, complete programs, output examples, comparisons, revision और chapter exercises के साथ concepts को practice से जोड़ें। अंतिम अध्याय में 40 MCQs, answer key और Student Result mini project है।

सीखने का क्रम

Concept पढ़ें, example type करें, input बदलें और result explain करें। MCQs पहले बिना answer key हल करें। Chapter examples स्वतंत्र folders में रखें।

Version एवं Execution

Console और AWT examples Java 17-compatible हैं। AWT windows को graphical desktop चाहिए। Applet section historical syllabus study है: JDK 11 में appletviewer हट गया और JDK 26 में Applet API हट गई। Modern mainstream browsers Java applets नहीं चलाते। Legacy applet code को Java 17 API के संदर्भ में पढ़ें; वह modern browser demo नहीं है।

Public class का नाम और .java filename समान रखें। Compile: `javac ClassName.java`; run: `java ClassName`। Java code fragments को उचित class/method में रखें। GUI examples की expected interactions बताई गई हैं; console output और window appearance अलग चीजें हैं।

Design एवं Practice

Blue code panels, teal comparison tables और yellow exam notes learning को आसान बनाते हैं। Diagrams conceptual flow दिखाते हैं। File examples temporary files का उपयोग करते हैं; project save existing file को overwrite नहीं करता।

विषय-सूची

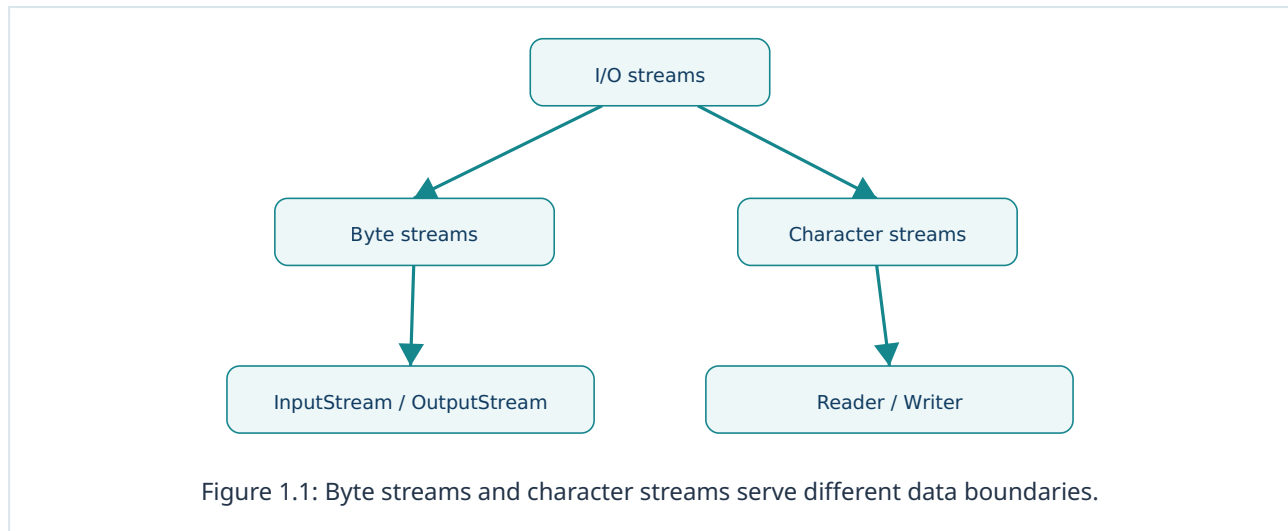
Contents / Chapters 1-12

01. Java I/O एवं Streams	4
02. Byte Streams एवं Character Streams	7
03. Predefined Streams एवं Console I/O	10
04. Applets का परिचय एवं Life Cycle	13
05. Applet बनाना, Images एवं Sound	16
06. java.lang की महत्वपूर्ण Classes	19
07. java.util Framework एवं Scanner	22
08. AWT Components एवं Layout Managers	25
09. Delegation Event Model	28
10. Event Classes एवं Listener Interfaces	31
11. Mouse, Keyboard Events एवं Adapter Classes	34
12. Complete Revision, Test एवं Mini Project	37

UNIT IV / CHAPTER 01

अध्याय 1: Java I/O एवं Streams

Input, Output और Data Flow



1.1 Learning Objectives

Input/output का अर्थ, stream abstraction, java.io classes, EOF, buffering और resource management समझेंगे।

1.2 I/O क्या है?

Input वह data है जो program प्राप्त करता है; output वह data है जो program किसी destination को भेजता है। Keyboard, file, network connection और memory buffer input/output sources या destinations हो सकते हैं।

I/O का अर्थ केवल console messages नहीं है। Image file पढ़ना, text लिखना या memory में binary data process करना भी I/O operations हैं।

1.3 Stream की अवधारणा

Stream data का sequential flow represent करती है। Input stream program की ओर data लाती है; output stream program से data बाहर ले जाती है। Byte और character abstractions data handling के अलग स्तर हैं।

श्रेणी	Data unit	प्रमुख base classes
Byte stream	Bytes	InputStream, OutputStream
Character stream	UTF-16 character units	Reader, Writer

Character stream को केवल “2 bytes की file stream” कहना गलत है। File के encoded bytes और Java character representation का conversion encoding के अनुसार होता है।

1.4 java.io का परिचय

Class	मुख्य उपयोग
FileInputStream	File से bytes
FileOutputStream	File में bytes
BufferedInputStream	Buffered byte reading
BufferedReader	Buffered character reading
InputStreamReader	Bytes को characters में decode करना
OutputStreamWriter	Characters को bytes में encode करना
PrintWriter	Text-oriented formatted output
StringReader	String से character input

File/Path destination की location represent कर सकते हैं; स्वयं data stream नहीं हैं। `java.nio.file.Files` file operations के convenient methods देती है।

1.5 read(), write() और EOF

Single-byte `InputStream.read()` int देता है। Valid byte values 0 से 255 और EOF indicator -1 है। `Reader.read()` भी int देता है ताकि character value तथा EOF अलग पहचाने जा सकें।

```
int value;
while ((value = input.read()) != -1) {
    // Process value
}
```

EOF check से पहले byte में cast करने पर 255 और -1 में भ्रम हो सकता है। Bulk read returned count देता है; केवल उतने valid elements process करें।

1.6 Memory Stream Example

```
import java.io.ByteArrayInputStream;

public class MemoryInputDemo {
    public static void main(String[] args) {
        byte[] data = {65, 66, 67};
        ByteArrayInputStream input = new ByteArrayInputStream(data);
        int value;
        while ((value = input.read()) != -1) {
            System.out.print((char) value);
        }
        System.out.println();
    }
}
```

Output: ABC। यह ASCII-range bytes का उदाहरण है; arbitrary UTF-8 bytes को individually char में cast करना text decoding का सामान्य सही तरीका नहीं है।

1.7 Buffering

Buffer एक intermediate memory area है। Buffered I/O कई छोटे reads/writes की overhead घटा सकती है। यह destination durability की guarantee नहीं है। Flush buffered data आगे भेजती है; close resource cleanup करती है और applicable buffered output flush कर सकती है।

1.8 Resource Management

Files और streams को close करना जरूरी हो सकता है। AutoCloseable resources के लिए try-with-resources उपयोग करें।

```
try (java.io.StringReader reader =  
    new java.io.StringReader("Java")) {  
    System.out.println((char) reader.read());  
}
```

यह fragment IOException obligation के अनुसार suitable method/catch में रखें। Resources reverse declaration order में close होती हैं।

1.9 Common Mistakes

- EOF को normal data value समझना।
- Character encoding implicitly मान लेना।
- Byte streams को text-only समझना।
- Read buffer की पूरी length process करना जबकि count कम मिला हो।
- Resource close न करना या shared console stream अनजाने में close करना।

1.10 Revision और अभ्यास

InputStream/OutputStream byte bases हैं। Reader/Writer character bases हैं। EOF -1 है। Buffering efficiency और resource management अलग concerns हैं।

अभ्यास: Memory input program का trace बनाइए। Keyboard, file और network I/O के examples लिखिए।

MCQs:

1. Byte input base: A. Reader B. InputStream C. Writer D. Frame
2. EOF indicator: A. 1 B. 0 C. 255 D. -1
3. Character output base: A. Button B. Scanner only C. Writer D. InputStream

उत्तर: 1-B, 2-C, 3-C.

UNIT IV / CHAPTER 02

अध्याय 2: Byte Streams एवं Character Streams

Binary Data, Text और Encoding

2.1 Learning Objectives

Binary copy और text processing के सही tools चुनेंगे। Explicit charset, stream bridges और buffering का उपयोग सीखेंगे।

2.2 Byte Streams

Raw binary data, image files और exact byte copying में byte streams उपयोगी हैं। Bytes का अर्थ application format निर्धारित करता है। InputStream single read या bulk read करती है; OutputStream bytes write करती है।

2.3 Byte Copy Program

```
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.Arrays;

public class ByteCopyDemo {
    public static void main(String[] args) throws Exception {
        Path source = Files.createTempFile("source-", ".bin");
        Path target = Files.createTempFile("target-", ".bin");
        try {
            Files.write(source, new byte[]{10, 20, 30, 40});
            try (InputStream input = Files.newInputStream(source);
                OutputStream output = Files.newOutputStream(target)) {
                byte[] buffer = new byte[1024];
                int count;
                while ((count = input.read(buffer)) != -1) {
                    output.write(buffer, 0, count);
                }
            }
            System.out.println(Arrays.toString(Files.readAllBytes(target)));
        } finally {
            Files.deleteIfExists(source);
            Files.deleteIfExists(target);
        }
    }
}
```

Output: [10, 20, 30, 40]। Temporary files example के अंत में हटाई जाती हैं। Real copy में input और output वही file न हों; truncate होने से source data खो सकता है।

2.4 Character Streams

Reader और Writer text-oriented processing देती हैं। Java chars UTF-16 code units हैं। External file encoding UTF-8, UTF-16 या अन्य हो सकती है। Encoding और decoding mismatch से unreadable text या replacement characters आ सकते हैं।

2.5 Bridge Classes

Bridge	दिशा
InputStreamReader	Bytes से characters
OutputStreamWriter	Characters से bytes

```
new java.io.InputStreamReader(
    System.in, java.nio.charset.StandardCharsets.UTF_8);
```

Chosen charset को actual input encoding से match करना चाहिए। हर terminal UTF-8 हो, यह स्वतः नहीं मानें।

2.6 UTF-8 Text File Program

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;

public class TextFileDemo {
    public static void main(String[] args) throws Exception {
        Path file = Files.createTempFile("java-notes-", ".txt");
        try {
            try (BufferedWriter writer =
                Files.newBufferedWriter(file, StandardCharsets.UTF_8)) {
                writer.write("Java Programming");
                writer.newLine();
                writer.write("Hindi-English Notes");
            }
            try (BufferedReader reader =
                Files.newBufferedReader(file, StandardCharsets.UTF_8)) {
                String line;
                while ((line = reader.readLine()) != null) {
                    System.out.println(line);
                }
            }
        } finally {
            Files.deleteIfExists(file);
        }
    }
}
```

Output: Java Programming; Hindi-English Notes अलग lines में। `BufferedReader.readLine()` line terminator को returned text में शामिल नहीं करती। EOF पर null देती है।

2.7 FileReader और FileWriter

ये file-based character convenience classes हैं। Charset न देने वाले constructors environment default पर निर्भर हो सकते हैं। Supported constructors या stream bridges से explicit charset देना portable text processing में उपयोगी है। Newer Files methods भी convenient हैं।

2.8 Byte बनाम Character

आधार	Byte	Character
उद्देश्य	Raw bytes	Text units
Base	InputStream/OutputStream	Reader/Writer
Typical use	Binary copy	Text lines
Encoding	Raw byte interpretation अलग	Encoding bridge आवश्यक हो सकती है

2.9 Revision एवं अभ्यास

Binary file character conversion से copy न करें। Bulk count सही उपयोग करें। Explicit encoding रखें। ReadLine का EOF null है; read का EOF -1 है।

अभ्यास: पाँच lines UTF-8 file में लिखें और पढ़ें। Byte-copy buffer count explain करें।

MCQs:

1. Bytes-to-characters bridge: A. OutputStreamWriter B. InputStreamReader C. Frame D. StringBuilder
2. readLine EOF: A. empty हमेशा B. -1 C. 0 D. null
3. Binary copy में write count: A. केवल 1 B. filename length C. पूरा buffer हमेशा D. actual bytes read

उत्तर: 1-B, 2-C, 3-B.

UNIT IV / CHAPTER 03

अध्याय 3: Predefined Streams एवं Console I/O

System.in, System.out और System.err

3.1 Learning Objectives

Standard streams पहचानेंगे, console input पढ़ेंगे, output format करेंगे और Scanner/BufferedReader approaches की तुलना करेंगे।

3.2 Standard Streams

Stream	Declared type	सामान्य भूमिका
System.in	InputStream	Standard input
System.out	PrintStream	Standard output
System.err	PrintStream	Standard error

ये streams keyboard/screen तक सीमित नहीं; shell या environment redirect कर सकता है। Out और err अलग destinations/order रख सकते हैं।

3.3 Output Methods

```
System.out.print("Hello ");
System.out.println("Students");
System.out.printf("Marks: %d\n", 85);
System.err.println("Example diagnostic message");
```

Print नई line नहीं जोड़ती; println जोड़ती है। Printf format placeholders उपयोग करती है। %d integer, %s string और %.2f दो fractional digits वाला floating format है। Locale decimal separator बदल सकता है।

3.4 Scanner Console Program

```
import java.util.Scanner;

public class ConsoleInputDemo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter name:");
        String name = input.nextLine();
        System.out.println("Enter marks:");
        if (!input.hasNextInt()) {
            System.out.println("Invalid integer input");
            return;
        }
    }
}
```

```

    }
    int marks = input.nextInt();
    System.out.println(name + ": " + marks);
}
}

```

Sample input: Amit, फिर 85। **Output:** prompts के बाद Amit: 85। Example shared System.in close नहीं करता। Dedicated short application के अंत में close policy अलग हो सकती है।

3.5 nextInt और nextLine Trap

nextInt token पढ़ती है, उसके बाद line का remainder बच सकता है। तुरंत nextLine empty remainder लौटाकर अगली intended line न पढ़े।

```

int age = input.nextInt();
input.nextLine(); // Consume current line remainder
String name = input.nextLine();

```

Alternative: सारी input lines पढ़कर parse करें। Consume call को बिना context हमेशा लगाने से valid content skip हो सकता है।

3.6 BufferedReader Input

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.nio.charset.StandardCharsets;

public class BufferedConsoleDemo {
    public static void main(String[] args) throws Exception {
        BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in, StandardCharsets.UTF_8));
        System.out.println("Enter an integer:");
        String line = reader.readLine();
        if (line == null) {
            System.out.println("No input");
            return;
        }
        try {
            int number = Integer.parseInt(line.trim());
            System.out.println("Square: " + (number * number));
        } catch (NumberFormatException exception) {
            System.out.println("Invalid number");
        }
    }
}

```

Input 12 दे तो Square: 144। Large integer square int overflow कर सकती है; example छोटे inputs के लिए है।

3.7 Console Class

System.console() terminal interaction की सुविधा दे सकती है। IDEs, redirected execution या automated environments में null हो सकती है। Null check करें। ReadPassword char array देती है; sensitive input display न करने की सुविधा है।

3.8 Resource Ownership

Scanner close करने पर Closeable underlying source भी close होती है। System.in पर Scanner बंद करने से later input प्रभावित हो सकती है। एक source पर multiple independently buffering wrappers रखकर reads interleave करना confusing हो सकता है।

3.9 Comparison एवं अभ्यास

आधार	Scanner	BufferedReader
Parsing	Token methods	Manual parsing
Line reading	nextLine	readLine
Failure example	InputMismatchException	IOException/parse failure
उपयोग	Structured simple input	Explicit line processing

अभ्यास: Name, class और marks पढ़ें। Invalid marks type handle करें। Formatting table बनाइए।

MCQs:

1. System.in type: A. InputStream B. Writer C. Frame D. PrintWriter
2. System.out type: A. Button B. Reader C. PrintStream D. Scanner
3. System.console(): A. केवल applet B. always String C. कभी null नहीं D. null हो सकती है

उत्तर: 1-A, 2-B, 3-B.

UNIT IV / CHAPTER 04

अध्याय 4: Applets का परिचय एवं Life Cycle

Historical Java Browser Programs

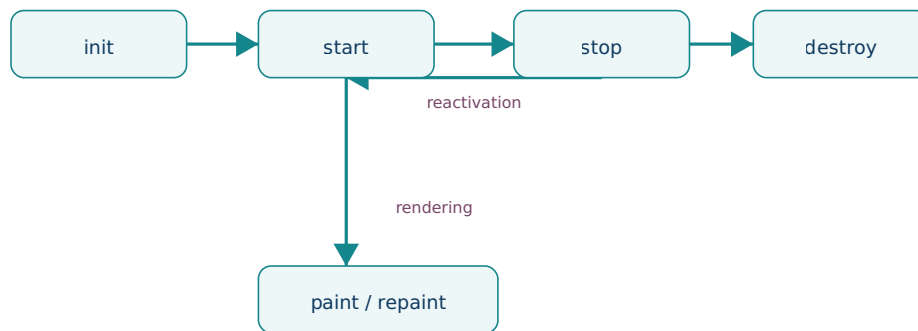


Figure 4.1: Simplified historical lifecycle. Rendering may repeat; host controls lifecycle callbacks.

4.1 Learning Objectives

Applet की historical role, host-controlled lifecycle और painting समझेंगे। Legacy environment तथा modern compatibility का अंतर स्पष्ट करेंगे।

4.2 Applet क्या था?

Applet Java का छोटा hosted program था जिसे historical browser plugin या applet viewer manage करता था। यह normal standalone application की तरह main() entry point पर निर्भर नहीं होता था। Host lifecycle callbacks invoke करता था।

इस syllabus topic को examination और historical understanding के लिए पढ़ें। Modern browser Java applets run करने का practical environment नहीं है।

4.3 Version Context

Environment	स्थिति
Legacy JDK 8 lab	appletviewer वाले पुराने educational workflows
JDK 11	Applet viewer/deployment tools हटाए गए
JDK 17	Applet API deprecated for removal
JDK 26	Applet API हटाई गई

Old HTML tag लिखने से modern browser support वापस नहीं आती। Applet class compile होना और actual host उपलब्ध होना अलग बातें हैं।

4.4 Lifecycle Methods

Method	Host-controlled उद्देश्य
init()	Initial setup
start()	Active execution शुरू/फिर शुरू
stop()	Active execution रोकना
destroy()	Final cleanup

Start/stop callbacks अनेक बार हो सकती हैं। Paint(Graphics) rendering callback है, core applet lifecycle की distinct one-time stage नहीं। Repainting अनेक बार हो सकती है।

4.5 Historical Lifecycle Example

```
import java.applet.Applet;
import java.awt.Graphics;

public class LifeCycleApplet extends Applet {
    private static final long serialVersionUID = 1L;
    private String message;

    @Override
    public void init() {
        message = "Applet initialized";
    }
    @Override
    public void start() {
        message = "Applet active";
        repaint();
    }
    @Override
    public void paint(Graphics graphics) {
        graphics.drawString(message, 20, 40);
    }
    @Override
    public void stop() {
        message = "Applet inactive";
    }
    @Override
    public void destroy() {
        message = null;
    }
}
```

Historical display: Applet active। यह main-based console program नहीं है। Example modern browser में directly runnable नहीं। JDK 17 में compilation deprecation warning दे सकती है; JDK 26 में Applet API नहीं है।

4.6 Legacy HTML Host

```
<applet code="LifeCycleApplet.class" width="400" height="150">
</applet>
```

इस tag को modern website solution के रूप में उपयोग न करें। पुराने isolated educational environment में historical command `appletviewer page.html` थी; current JDK installations में tool मिलने की अपेक्षा न करें।

4.7 Application बनाम Applet

आधार	Application	Historical Applet
Entry	main या supported launcher form	Host callbacks
Host	Java runtime launcher	Browser plugin/viewer
Lifecycle	Application-controlled	Host-controlled
Current browser use	Applet plugin जैसा नहीं	Unsupported obsolete approach

4.8 Applet stop बनाम Thread.stop

`Applet.stop()` lifecycle callback है। `Thread.stop()` legacy unsafe thread-control API थी। समान नाम होने से दोनों समान operation नहीं हैं। Applet callback में worker tasks को cooperative policy से pause/stop किया जाता था।

4.9 Revision एवं अभ्यास

Init setup; start active phase; stop inactivity; destroy cleanup। Paint repeated rendering है। Applet को main application समझकर `java ClassName` चलाना पर्याप्त नहीं है।

अभ्यास: Lifecycle methods का sequence समझाएँ। Application और applet का difference table बनाइए।

MCQs:

1. Initial setup callback: A. init B. main C. throws D. notify
2. Paint call count: A. केवल init में B. हमेशा एक C. अनेक बार हो सकती है D. कभी नहीं
3. Modern browser applet plugin: A. Scanner B. File stream C. सामान्य supported option D. historical obsolete approach

उत्तर: 1-A, 2-B, 3-B.

UNIT IV / CHAPTER 05

अध्याय 5: Applet बनाना, Images एवं Sound

Graphics, Code Base एवं AudioClip

5.1 Learning Objectives

Historical Applet creation, drawing, image/audio resource location और lifecycle cleanup समझेंगे।
Modern desktop alternatives का परिचय भी मिलेगा।

5.2 Basic Graphics

Graphics drawing operations rendering callback में होती हैं। Coordinates सामान्यतः component के top-left से relative हैं; x दाईं ओर और y नीचे बढ़ता है। Font metrics तथा platform rendering appearance को प्रभावित करती हैं।

```
graphics.drawString("Java", 20, 40);  
graphics.drawRect(20, 60, 100, 50);  
graphics.fillOval(150, 60, 80, 80);
```

Draw outline बनाती है; fill interior भरती है। Component graphics में drawing को stored state के आधार पर repaintable रखें।

5.3 Image Loading का Historical Pattern

getImage(baseUrl, filename) image resource देती थी। getCodeBase() applet code की base location तथा getDocumentBase() containing document की location देती थी। Resource fetch asynchronous हो सकती थी; तुरंत dimensions/data ready होना आवश्यक नहीं।

5.4 Historical Media Applet

```
import java.applet.Applet;  
import java.applet.AudioClip;  
import java.awt.Graphics;  
import java.awt.Image;  
  
public class MediaApplet extends Applet {  
    private static final long serialVersionUID = 1L;  
    private Image picture;  
    private AudioClip audio;  
  
    @Override  
    public void init() {  
        picture = getImage(getCodeBase(), "picture.png");  
        audio = getAudioClip(getCodeBase(), "sound.wav");  
    }  
  
    @Override  
    public void start() {  
        if (audio != null) {  
            audio.play();  
        }  
    }  
}
```

```

    }
    @Override
    public void paint(Graphics graphics) {
        graphics.drawString("Image and Sound Applet", 20, 30);
        if (picture != null) {
            graphics.drawImage(picture, 20, 50, 200, 120, this);
        }
    }
    @Override
    public void stop() {
        if (audio != null) {
            audio.stop();
        }
    }
    @Override
    public void destroy() {
        if (audio != null) { audio.stop(); }
        picture = null;
        audio = null;
    }
}

```

Required historical assets: picture.png और supported sound.wav। Image display/audio playback resource availability, supported format और host behavior पर निर्भर थे। Null check successful decoding/playback की पूरी guarantee नहीं है।

5.5 AudioClip Methods

Method	उद्देश्य
play()	एक playback शुरू
loop()	Repeated playback
stop()	Playback रोकना

इस chapter की Applet/AudioClip APIs legacy हैं। Existing old code पढ़ने और exam syntax के लिए हैं, new production development के लिए नहीं।

5.6 ImageObserver

DrawImage में final observer argument asynchronous image updates की notification से संबंधित है। Applet/Component ImageObserver role निभा सकती थी। Repaint से UI updated content draw कर सकती है।

5.7 Painting Rules

- Long-running work paint() में न रखें।
- Paint को केवल एक बार चलने वाली initialization न मानें।
- Drawing के लिए external state रखें; resize/exposure repaint कर सकते हैं।
- Graphics argument को permanently store न करें।
- Repaint rendering request है, direct immediate drawing guarantee नहीं।

5.8 Modern In-Memory Image Example

Applet की आवश्यकता के बिना BufferedImage और Graphics2D drawing की जा सकती है।

```
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;

public class ImageDrawingDemo {
    public static void main(String[] args) {
        BufferedImage image = new BufferedImage(
            100, 60, BufferedImage.TYPE_INT_RGB);
        Graphics2D graphics = image.createGraphics();
        try {
            graphics.setColor(Color.BLUE);
            graphics.fillRect(0, 0, 100, 60);
        } finally {
            graphics.dispose();
        }
        System.out.println(image.getWidth() + " x " + image.getHeight());
    }
}
```

Output: 100 x 60। यह offscreen example graphical desktop window नहीं खोलता। Sound के modern APIs अलग javax.sound packages में उपलब्ध हैं।

5.9 Revision एवं अभ्यास

CodeBase और documentBase अलग resource anchors हैं। Audio play/loop/stop याद रखें। Painting repeated है। Media assets तथा historical host के बिना applet display की guarantee नहीं।

अभ्यास: Applet में rectangle और oval draw करने का historical code लिखें। Media cleanup lifecycle explain करें।

MCQs:

1. Audio repeat method: A. loop B. draw C. read D. next
2. Code resource base: A. readLine B. getCodeBase C. getPriority D. getState
3. Repaint: A. File copy B. Scanner delimiter C. Rendering request D. JVM restart

उत्तर: 1-A, 2-B, 3-C.

UNIT IV / CHAPTER 06

अध्याय 6: java.lang की महत्वपूर्ण Classes

Object, String, Wrappers, Math एवं System

6.1 Learning Objectives

Automatically imported java.lang types, wrapper conversions, Object methods और Math/System utilities उपयोग कर सकेंगे।

6.2 java.lang Package

Java.lang की accessible types ordinary source में automatically imported होती हैं। Subpackages automatic नहीं। String, System, Object, Math, Thread और exception base types इसके examples हैं।

6.3 Object Class

सामान्य class hierarchy का root Object है। ToString textual representation देती है। Equals logical equality का contract दे सकती है; default Object equality reference identity है। hashCode hash-based collections में उपयोगी है। getClass runtime class देती है।

Equals override करने पर hashCode contract भी ध्यान रखें: equal objects की hash codes समान होनी चाहिए; same hash होने से objects equal होना आवश्यक नहीं।

6.4 Wrapper Classes

Primitive	Wrapper
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Autoboxing primitive से wrapper और unboxing wrapper से primitive conversion देती है। Null wrapper unbox करने पर NullPointerException हो सकती है।

6.5 Parsing और valueOf

```
int marks = Integer.parseInt("85");
Integer wrapped = Integer.valueOf("85");
int unboxed = wrapped;
```

parseInt primitive int देती है। ValueOf Integer reference देती है। Wrapper objects के numerical equality के लिए inappropriate reference == पर निर्भर न रहें।

6.6 Math Class

Method	उदाहरण	Result
abs	Math.abs(-5)	5
max	Math.max(10,20)	20
min	Math.min(10,20)	10
sqrt	Math.sqrt(81)	9.0
pow	Math.pow(2,3)	8.0
ceil	Math.ceil(2.3)	3.0
floor	Math.floor(2.8)	2.0
round	Math.round(2.6)	3

Floating special values और integer minimum abs जैसे edge cases अलग behavior रख सकते हैं। Table सामान्य small finite inputs के लिए है।

6.7 Complete Utility Program

```
public class LangDemo {
    public static void main(String[] args) {
        int number = Integer.parseInt("81");
        System.out.println("Square root: " + Math.sqrt(number));
        System.out.println("Digit: " + Character.isDigit('7'));
        System.out.println("Uppercase: " + Character.toUpperCase('a'));
        String text = "Java";
        System.out.println("Length: " + text.length());
    }
}
```

Output: Square root: 9.0; Digit: true; Uppercase: A; Length: 4।

6.8 System और Runtime

System standard streams, properties और arraycopy जैसी utilities देती है। CurrentTimeMillis wall-clock time तथा nanoTime elapsed durations measure करने में उपयोगी है। NanoTime absolute calendar timestamp नहीं है।

Runtime.getRuntime() runtime information/services देता है। Free-memory snapshot को total future allocation capacity न समझें। Garbage collection request exact timing guarantee नहीं देती।

6.9 Class और Class Literals

```
Class<?> type = String.class;  
System.out.println(type.getName()); // java.lang.String
```

Class runtime type metadata represent करती है। Reflection advanced topic है; Class को ordinary object instance के समान नाम वाला keyword न समझें।

6.10 Revision एवं अभ्यास

java.lang import automatically है। Wrapper class और primitive type अलग हैं। Math static methods class name से call होती हैं। String immutable है।

अभ्यास: Wrapper mapping लिखें। Text से marks parse करें। Math methods से numerical outputs निकालें।

MCQs:

1. int wrapper: A. Int B. Integer C. Number only D. Long
2. Math.pow(2,3): A. 5 B. 6 C. 8.0 D. 9
3. Null wrapper unboxing: A. false B. Class बनती है C. 0 हमेशा D. NullPointerException हो सकती है

उत्तर: 1-B, 2-B, 3-B.

UNIT IV / CHAPTER 07

अध्याय 7: java.util Framework एवं Scanner

Collections, Utilities एवं Input Parsing

7.1 Learning Objectives

Collection interfaces और common implementations, Arrays/Collections utilities तथा Scanner की token/line methods समझेंगे।

7.2 java.util का Scope

Java.util में collections framework, scanners, random utilities और अन्य helpers हैं। Event delegation का AWT-specific portion java.awt.event में है; उसे केवल java.util feature न समझें। Syllabus के event topics chapters 9-11 में हैं।

7.3 Collections Framework

Interface	उद्देश्य	Example implementation
List	Ordered sequence; duplicates possible	ArrayList
Set	Unique elements	HashSet
Queue	Queue-based processing	ArrayDeque
Map	Key-value associations	HashMap

Map, Collection interface की subtype नहीं है। Collection और Collections अलग हैं: पहला interface, दूसरा utility class।

7.4 ArrayList Example

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ListDemo {
    public static void main(String[] args) {
        List<String> names = new ArrayList<>();
        names.add("Neha");
        names.add("Amit");
        names.add("Ravi");
        Collections.sort(names);
        System.out.println(names);
        System.out.println("Size: " + names.size());
    }
}
```

Output: [Amit, Neha, Ravi]; Size: 3 | Generic type String compile-time element checking में सहायता देती है।

7.5 Set और Map

HashSet duplicate equality-based elements नहीं रखती। HashMap unique keys को values से जोड़ती है; same key पर put existing mapping replace कर सकती है। HashSet/HashMap iteration order को insertion order न मानें। Ordered needs के लिए appropriate implementations चुनें।

```
java.util.Map<String, Integer> marks = new java.util.HashMap<>();
marks.put("Amit", 85);
System.out.println(marks.get("Amit")); // 85
```

7.6 Arrays Utilities

```
int[] values = {30, 10, 20};
java.util.Arrays.sort(values);
System.out.println(java.util.Arrays.toString(values));
```

Output: [10, 20, 30] | BinarySearch के लिए input appropriate sorted order में होना चाहिए। Arrays.asList fixed-size list view देती है; add/remove unsupported हो सकता है।

7.7 Scanner Class

Scanner whitespace-delimited tokens को parse कर सकती है। Input String, stream या file आदि से आ सकती है। Token parsing के लिए nextInt, nextDouble, nextBoolean; text के लिए next और nextLine उपलब्ध हैं।

Method	उद्देश्य
next()	Next token
nextLine()	Current line remainder
nextInt()	Integer token
hasNextInt()	Compatible next token?
useDelimiter()	Token delimiter pattern
useLocale()	Number-format locale

7.8 Self-Contained Scanner Example

```
import java.util.Scanner;

public class ScannerTextDemo {
    public static void main(String[] args) {
        try (Scanner input = new Scanner("Amit 85 90")) {
```

```

        String name = input.next();
        int first = input.nextInt();
        int second = input.nextInt();
        System.out.println(name + ": " + (first + second));
    }
}

```

Output: Amit: 175 | String source close करने से shared console बंद नहीं होती।

7.9 Parsing Failures

Invalid token `InputMismatchException` दे सकती है। Token अपने आप discard नहीं होती; recovery loop को bad token consume करना पड़ सकता है। Input समाप्त होने पर next methods `NoSuchElementException` दे सकती हैं। Closed scanner `IllegalStateException` दे सकती है।

7.10 Important Utility Distinctions

Pair	अंतर
Array length / List size()	Field / method
Collection / Collections	Interface / utility
List / Set	Duplicates allowed / uniqueness
Map / Collection	Map अलग interface branch
next / nextLine	Token / line remainder

अभ्यास: Names sort करें। Unique subjects का set बनाएँ। Scanner से numerical text का total निकालें।

MCQs:

1. Map Collection का subtype? A. हाँ B. नहीं C. केवल HashMap D. केवल TreeMap
2. Collection utility class: A. Reader B. Collections C. Scanner D. Frame
3. Next token: A. readLine B. paint C. next D. nextLine हमेशा

उत्तर: 1-B, 2-B, 3-C.

UNIT IV / CHAPTER 08

अध्याय 8: AWT Components एवं Layout Managers

Desktop GUI का निर्माण

8.1 Learning Objectives

AWT toolkit, Component/Container, Frame, controls और layout managers समझेंगे। Desktop display तथा event-dispatch-thread requirements स्पष्ट करेंगे।

8.2 AWT क्या है?

AWT = Abstract Window Toolkit. Java.awt graphical controls, windows, graphics और layout facilities देती है। कई classic AWT controls native peers उपयोग करती हैं, इसलिए appearance operating system पर निर्भर हो सकती है।

AWT desktop code के लिए graphical display चाहिए। Headless server में Frame creation HeadlessException दे सकती है। Offscreen BufferedImage processing कई headless workflows में चल सकती है।

8.3 Component और Container

Component visual/UI element का आधार है। Container अन्य components रख सकती है। Panel, Window और Frame container-related classes हैं। Frame title और window decoration वाला top-level window है।

Component	कार्य
Label	Text display
Button	Click action
TextField	Single-line input
TextArea	Multiline text
Checkbox	Selection toggle
Choice	Drop-down choice
List	Selectable items
Canvas	Custom drawing surface

8.4 Layout Managers

Layout	व्यवस्था
FlowLayout	Sequential flowing components
BorderLayout	NORTH, SOUTH, EAST, WEST, CENTER regions
GridLayout	Equal-sized grid cells
CardLayout	One card visible at a time
GridBagLayout	Flexible constraint-based grid

Frame default BorderLayout और Panel default FlowLayout है। Layout managers resizing और platform differences handle करने में सहायता करती हैं। Null layout और hardcoded positions beginner demonstrations में दिख सकती हैं, लेकिन responsive arrangement के लिए managers बेहतर हैं।

8.5 Basic AWT Window

```
import java.awt.Button;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class AwtWindowDemo {
    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            Frame frame = new Frame("Java AWT Demo");
            frame.setLayout(new FlowLayout());
            frame.add(new Label("Name:"));
            frame.add(new TextField(20));
            frame.add(new Button("Submit"));
            frame.addWindowListener(new WindowAdapter() {
                @Override
                public void windowClosing(WindowEvent event) {
                    frame.dispose();
                }
            });
            frame.setSize(420, 180);
            frame.setVisible(true);
        });
    }
}
```

Desktop result: Window में Name label, text field और Submit button। Button action अभी registered नहीं है। Close event window dispose करती है।

8.6 Event Dispatch Thread

EventQueue.invokeLater task को AWT Event Dispatch Thread (EDT) पर schedule करती है। GUI creation/update को EDT पर व्यवस्थित करें। Long I/O या heavy computation EDT पर करने से UI unresponsive हो सकती है।

8.7 Drawing with Canvas

```
java.awt.Canvas canvas = new java.awt.Canvas() {  
    @Override  
    public void paint(java.awt.Graphics graphics) {  
        graphics.drawString("Java AWT", 20, 30);  
        graphics.drawRect(20, 50, 120, 60);  
    }  
};
```

यह fragment Canvas को visible container में add करने पर rendering देती है। Graphics directly screen state नहीं बचाती; paint stored model के आधार पर पुनः draw करती है।

8.8 Size, pack और Insets

SetSize explicit window size देती है। Pack layout के preferred sizes के आधार पर size निर्धारित कर सकती है। Top-level decorations/insets client area कम कर सकते हैं। Exact pixel appearance platform-dependent है।

8.9 Common Mistakes और अभ्यास

Java.awt.List और java.util.List अलग types हैं। Both packages के wildcard imports ambiguous List reference दे सकते हैं। Explicit imports रखें। Window close handling लिखें; button बनाना listener registration नहीं है।

अभ्यास: GridLayout में four buttons रखें। BorderLayout regions में labels रखें। Basic form बनाइए।

MCQs:

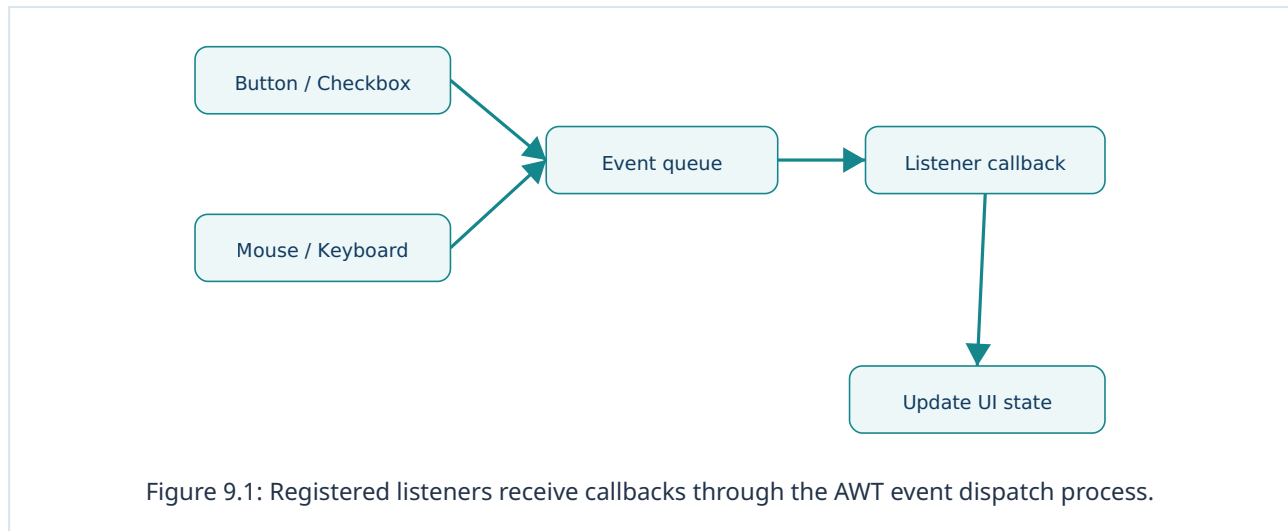
1. AWT full form: A. Abstract Window Toolkit B. Automatic Web Tool C. Applet Writer D. Abstract Word Table
2. Frame default layout: A. कोई नहीं B. BorderLayout C. FlowLayout D. GridLayout
3. Panel default layout: A. CardLayout B. GridBag always C. FlowLayout D. BorderLayout

उत्तर: 1-A, 2-B, 3-C.

UNIT IV / CHAPTER 09

अध्याय 9: Delegation Event Model

Event Sources, Objects और Listeners



9.1 Event क्या है?

Event user/system action का notification object है। Button click, key press, mouse motion, selection change और window close events के examples हैं।

9.2 Delegation Model

Source event generate करती है। Registered listener event receive करके callback method में response देती है। Listener source के event handling को delegated object में रखती है।

भाग	उदाहरण
Source	Button
Event object	ActionEvent
Listener	ActionListener implementation
Registration	<code>addActionListener(listener)</code>
Callback	<code>actionPerformed(event)</code>

Source में listener register न हो, तो button का application action callback नहीं आएगा।

9.3 Complete Button Counter

```
import java.awt.Button;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class ButtonCounterDemo {
    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            Frame frame = new Frame("Button Counter");
            frame.setLayout(new FlowLayout());
            Label output = new Label("Clicks: 0");
            Button button = new Button("Click Me");
            int[] count = {0};
            button.addActionListener(event -> {
                count[0]++;
                output.setText("Clicks: " + count[0]);
            });
            frame.add(button);
            frame.add(output);
            frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent event) {
                    frame.dispose();
                }
            });
            frame.setSize(320, 150);
            frame.setVisible(true);
        });
    }
}
```

Desktop behavior: Initial Clicks: 0। हर click पर 1 बढ़ेगा। Count access इस example में EDT-confined है; array होना general thread safety नहीं देता।

9.4 Listener Registration

AddXListener registration करती है; removeXListener deregistration कर सकती है। एक source multiple listeners रख सकती है। Long-lived source अनावश्यक references रखे, तो listener lifecycle manage करना उपयोगी है।

9.5 Callback Styles

- Named class implements listener interface।
- Anonymous class implementation।
- Single-abstract-method compatible interface के लिए lambda।
- Multi-method interface के लिए adapter subclass।

ActionListener single abstract method रखती है, इसलिए lambda उपयोगी है। MouseListener की पाँच methods हैं; सीधे lambda target नहीं है।

9.6 Event Object से Information

Event object source, event id और event-specific details दे सकती है। `ActionEvent.getActionCommand()` command text और `getSource()` originating object देता है। Button labels बदलने पर command assumptions भी जाँचें; explicit action commands रख सकते हैं।

9.7 GUI Event और Threading

AWT callbacks सामान्यतः event queue पर dispatch होती हैं। Handler को short रखें। Blocking wait या long file read EDT पर UI freeze कर सकती है। Worker result आने पर UI update EDT पर schedule करें।

9.8 Syllabus Clarification

Delegation event model Java AWT event handling का केंद्रीय concept है। यह old Applet programs और standalone AWT windows दोनों में उपयोग किया जा सकता है; browser-hosting requirement नहीं है।

9.9 Revision और अभ्यास

Source generates, event carries details, listener responds। Registration callback activation के लिए आवश्यक है। Button add करना handling नहीं है।

अभ्यास: Button click पर text field content label में दिखाएँ। Add/remove listener का effect समझाएँ।

MCQs:

1. Button action callback: A. `actionPerformed` B. `paint` C. `run` D. `readLine`
2. Registration: A. import only B. `addActionListener` C. throws D. init always
3. Event source: A. केवल file B. केवल listener C. Originating component/object D. हमेशा JVM only

उत्तर: 1-A, 2-B, 3-C.

UNIT IV / CHAPTER 10

अध्याय 10: Event Classes एवं Listener Interfaces

Matching Events to Callbacks

10.1 Learning Objectives

Event classes, sources, corresponding listener interfaces और callback methods पहचानेंगे। Low-level और semantic events के examples पढ़ेंगे।

10.2 Event Mapping Table

Event Class	Typical source/action	Listener
ActionEvent	Button click, TextField enter	ActionListener
ItemEvent	Checkbox/Choice selection	ItemListener
AdjustmentEvent	Scrollbar value change	AdjustmentListener
TextEvent	Text change	TextListener
WindowEvent	Window lifecycle	WindowListener आदि
MouseEvent	Mouse click/motion	MouseListener/MouseMotionListener
KeyEvent	Keyboard activity	KeyListener
FocusEvent	Focus change	FocusListener
ComponentEvent	Move/resize/show/hide	ComponentListener

Sources और event types exact component behavior के अनुसार होंगी। केवल class का नाम देखकर हर source से हर event expect न करें।

10.3 Low-Level और Semantic Events

Mouse/key/component events low-level interaction बता सकती हैं। Action/item events higher-level semantic operations represent करती हैं। Button activate करने का application meaning ActionEvent में व्यक्त करना mouseClicked पर exclusively निर्भर होने से बेहतर हो सकता है।

10.4 ActionListener

```
java.awt.event.ActionListener listener = event -> {
    System.out.println(event.getActionCommand());
};
```

Registration उचित Button/TextField आदि पर की जाती है। GetSource return Object है; जरूरत हो तो verified type के अनुसार cast करें।

10.5 Checkbox Item Example

```
import java.awt.Checkbox;
import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class CheckboxDemo {
    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            Frame frame = new Frame("Selection Demo");
            frame.setLayout(new FlowLayout());
            Checkbox choice = new Checkbox("Java");
            Label result = new Label("Not selected");
            choice.addItemListener(event -> result.setText(
                choice.getState() ? "Selected" : "Not selected"));
            frame.add(choice);
            frame.add(result);
            frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent event) {
                    frame.dispose();
                }
            });
            frame.setSize(340, 160);
            frame.setVisible(true);
        });
    }
}
```

Desktop behavior: Checkbox select करने पर Selected; deselect करने पर Not selected।

10.6 WindowListener Methods

WindowOpened, windowClosing, windowClosed, windowIconified, windowDeiconified, windowActivated और windowDeactivated callbacks हैं। WindowClosing user close request है; windowClosed actual disposal-related notification है। दोनों समान stage नहीं।

WindowAdapter unnecessary methods के empty implementations से boilerplate कम करती है। केवल windowClosing override करके dispose करना basic cleanup है।

10.7 Focus और Keyboard

Keyboard events सामान्यतः focused component को मिलती हैं। Window दिखाने से हर arbitrary component को focus guarantee नहीं मिलती। Focus traversal key handling से अलग subsystem है।

10.8 EventObject और AWTEvent

Java.util.EventObject generic event source support देती है। Java.awt.AWTEvent AWT event hierarchy का आधार है। Concrete AWT event classes details जोड़ती हैं। EventListener java.util में marker interface है, जबकि concrete listener contracts java.awt.event में हैं।

10.9 Common Mistakes और अभ्यास

ActionEvent को MouseEvent समझकर getX call नहीं कर सकते। Wrong listener registration compile या behavior problems पैदा करती है। WindowClosing और closed अलग callbacks हैं।

अभ्यास: Source-event-listener mapping याद करें। TextField Enter पर entered text label में दिखाएँ।

MCQs:

1. Checkbox selection: A. ItemEvent B. IOException C. ThreadState D. Class
2. Window close request: A. read B. windowClosing C. windowClosed हमेशा पहले D. init
3. Generic event source base: A. InputStream B. StringBuilder C. EventObject D. Scanner

उत्तर: 1-A, 2-B, 3-C.

UNIT IV / CHAPTER 11

अध्याय 11: Mouse, Keyboard Events एवं Adapter Classes

Interaction Handling

11.1 Learning Objectives

Mouse clicks/movement और keyboard activity handle करेंगे। Adapter classes से केवल आवश्यक callbacks override करना सीखेंगे।

11.2 MouseListener Methods

Callback	अर्थ
mouseClicked	Click event
mousePressed	Button press
mouseReleased	Button release
mouseEntered	Component area में entry
mouseExited	Component area से exit

MouseMotionListener में mouseMoved और mouseDragged हैं। MouseWheelListener wheel movement के लिए है। Coordinates event source component के relative होती हैं।

11.3 MouseAdapter Example

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Frame;
import java.awt.Label;
import java.awt.BorderLayout;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class MouseDemo {
    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            Frame frame = new Frame("Mouse Demo");
            Canvas canvas = new Canvas();
            canvas.setBackground(Color.WHITE);
            Label output = new Label("Click inside the white area");
            canvas.addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent event) {
                    output.setText("x=" + event.getX()
                        + ", y=" + event.getY());
                }
            });
        });
    }
}
```

```

    });
    frame.add(canvas, BorderLayout.CENTER);
    frame.add(output, BorderLayout.SOUTH);
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent event) {
            frame.dispose();
        }
    });
    frame.setSize(420, 280);
    frame.setVisible(true);
}
}
}

```

Desktop behavior: Canvas पर click coordinates label में दिखती हैं। Exact numbers click location के अनुसार हैं।

11.4 Mouse Motion Registration

MouseAdapter modern API में motion/wheel listener roles भी implement करती है। Required registration अलग है: movement के लिए addMouseMotionListener और wheel के लिए addMouseWheelListener। केवल addMouseListener से mouseMoved callbacks नहीं मिलेंगी।

11.5 KeyListener Methods

Callback	उद्देश्य
keyPressed	Key press; virtual key code
keyReleased	Key release
keyTyped	Character-oriented typing

GetKeyCode pressed/released में उपयोगी है। GetKeyChar character typing में उपयोगी है। हर key printable character नहीं देती; modifiers और special keys अलग हैं।

11.6 KeyAdapter Program

```

import java.awt.EventQueue;
import java.awt.FlowLayout;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class KeyboardDemo {
    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            Frame frame = new Frame("Keyboard Demo");
            frame.setLayout(new FlowLayout());
            TextField input = new TextField(20);
            Label output = new Label("Press a key in the text field");
            input.addKeyListener(new KeyAdapter() {
                public void keyPressed(KeyEvent event) {
                    output.setText("Key: "
                        + KeyEvent.getKeyText(event.getKeyCode()));
                }
            });
            frame.add(input);
        });
    }
}

```

```

        frame.add(output);
        frame.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent event) {
                frame.dispose();
            }
        });
        frame.setSize(450, 170);
        frame.setVisible(true);
        input.requestFocusInWindow();
    });
}

```

Desktop behavior: Focused text field में pressed key का description दिखता है। Focus request environment के अनुसार fail हो सकती है; field click करके focus दें।

11.7 Adapter Classes

Adapter	Related interface(s)
MouseAdapter	Mouse, motion और wheel roles
KeyAdapter	KeyListener
WindowAdapter	Window listener roles
FocusAdapter	FocusListener
ComponentAdapter	ComponentListener

Adapter abstract classes empty callbacks देती हैं। Subclass केवल required methods override करती है। ActionListener single-method interface है; standard ActionListener class नहीं है।

11.8 Revision एवं अभ्यास

Adapter implements callbacks का boilerplate घटाती है। Correct registration और focus आवश्यक हैं। Click coordinates component-relative हैं। Key typed और key pressed अलग semantics हैं।

अभ्यास: Mouse movement coordinates दिखाएँ। Enter key पर form action trigger करें। WindowAdapter lifecycle callbacks compare करें।

MCQs:

1. Mouse coordinates: A. Event source relative B. हमेशा screen absolute C. हमेशा window border D. file-relative
2. Keyboard callbacks के लिए: A. Applet आवश्यक B. Focus महत्वपूर्ण C. हमेशा हर component को event D. Scanner आवश्यक
3. ActionListener standard class: A. Scanner की subclass B. Window की subclass C. नहीं D. हमेशा

उत्तर: 1-A, 2-B, 3-C.

UNIT IV / CHAPTER 12

अध्याय 12: Complete Revision, Test एवं Mini Project

पूरे Unit को जोड़कर सीखें

12.1 One-page Concept Map

Topic	याद रखने वाला विचार	सामान्य गलती
Byte streams	Binary data, read() returns int	-1 को byte में बदलकर EOF खोना
Character streams	Text + explicit charset	Platform default पर निर्भर होना
Console	System.in/out/err	nextInt के बाद newline भूलना
Applet	Historical hosted lifecycle	Modern browser में runnable मानना
java.lang	Object, String, wrappers, Math	String comparison के लिए ==
java.util	Collections, Scanner	Map को Collection subtype मानना
AWT	Components + layout managers	UI updates background thread से करना
Events	Source → event → registered listener	Listener register न करना
Adapters	Required callbacks override करें	हर event के लिए adapter मानना

12.2 Solved Output Questions

Q1: InputStream में single byte 255 है। लगातार दो read() results क्या होंगे? **Answer:** 255, फिर -1। Return type int valid unsigned byte और EOF को अलग रखती है।

Q2: new String("Java").equals("Java")? **Answer:** true। equals content compare करता है। == object identity compare करता है।

Q3: ArrayList में दो बार "A" add करने पर size? **Answer:** 2। List duplicates रख सकती है।

Q4: Scanner input 85\nAmit\n पर nextInt(), फिर nextLine()? **Answer:** int 85 और empty string; nextLine current line का शेष भाग पढ़ती है। अगली nextLine Amit पढ़ती है।

Q5: Applet paint() केवल एक बार execute होती है? **Answer:** नहीं। Repaint/exposure के कारण कई calls संभव हैं; persistent state fields में रखें।

Q6: Button click पर actionPerformed क्यों नहीं चला? **Answer:** Registration, enabled state और correct source check करें। EDT को लंबे काम से block करना भी responsiveness खराब करता है।

12.3 Unit Test: 40 MCQs

पहले स्वयं हल करें। हर प्रश्न का एक सबसे उपयुक्त उत्तर है।

1. Binary file के लिए कौन-सा abstraction उचित है? A. InputStream B. Reader C. String D. Label
2. read() का EOF result क्या है? A. null B. -1 C. 0 D. 255
3. Reader किस data पर केंद्रित है? A. Threads B. Images only C. Characters D. Window events
4. UTF-8 कहाँ specify करें? A. Mouse coordinate B. Button color C. Array index D. Text encoding/decoding boundary
5. Console input stream कौन-सी है? A. System.in B. System.out C. System.err D. System.consoleOut
6. PrintStream example कौन-सा है? A. Frame B. System.out C. FileReader D. Scanner
7. nextInt के बाद nextLine empty क्यों हो सकती है? A. AWT आवश्यक है B. int invalid हमेशा है C. Newline शेष है D. JVM बंद है
8. BufferedReader line input method? A. nextInt B. drawString C. addActionListener D. readLine
9. Applet initialization callback? A. init B. main only C. click D. scan
10. Applet rendering callback? A. hash B. paint C. next D. close
11. Applet lifecycle का paused-to-active callback? A. equals B. parse C. start D. destroy
12. Modern mainstream browser support for Java applets? A. हमेशा built-in B. HTML5 का अनिवार्य भाग C. Scanner आधारित D. उपलब्ध नहीं
13. Object किस package में है? A. java.lang B. java.awt.event C. java.io D. java.net
14. String content compare कैसे करें? A. sleep B. equals C. == हमेशा D. repaint
15. Integer.parseInt returns? A. boolean B. Reader C. int D. Frame
16. Math.sqrt(81) result? A. 81 B. 8 C. -1 D. 9.0
17. Collection framework package? A. java.util B. java.applet only C. java.sql only D. java.lang.event
18. List duplicates? A. केवल null B. Allow हो सकते हैं C. कभी नहीं D. हमेशा exception
19. Set सामान्यतः क्या enforce करता है? A. Window size B. Byte encoding C. Unique elements D. Duplicate count
20. Map represents? A. केवल ordered characters B. Mouse movement C. Applet drawing D. Key-value mappings
21. Scanner किससे पढ़ सकती है? A. String input B. केवल keyboard C. केवल AWT D. केवल applet
22. hasNextInt किसलिए? A. Thread start B. अगला token int-check C. File delete D. Window close
23. AWT top-level window class? A. Math B. Reader C. Frame D. Scanner
24. Label का काम? A. Binary copy B. Audio decode C. Hashing D. Text display
25. TextField का सामान्य उपयोग? A. Single-line input B. Thread lock C. Applet host D. File stream
26. Frame का default layout? A. कोई नहीं हमेशा B. BorderLayout C. FlowLayout D. GridBagLayout
27. Panel का default layout? A. CardLayout B. कोई नहीं हमेशा C. FlowLayout D. BorderLayout
28. AWT UI task queue पर schedule करने के लिए? A. Scanner.next B. System.exit C. Math.abs D. EventQueue.invokeLater
29. Button का semantic click event? A. ActionEvent B. KeyEvent only C. WindowEvent D. IOException
30. Listener registration example? A. readLine B. addActionListener C. parseInt D. drawRect

31. Checkbox state listener? A. KeyListener only B. Runnable C. ItemListener D. WindowListener
32. Window close request callback? A. windowClosed हमेशा पहले B. init C. read D. windowClosing
33. Mouse click callback? A. mouseClicked B. keyTyped C. actionPerformed D. drawMouse
34. Mouse movement listener? A. Scanner B. MouseMotionListener C. ActionListener D. ItemListener
35. KeyEvent handling में क्या आवश्यक है? A. Applet plugin B. AudioClip C. Appropriate focus D. UTF-8 file
36. keyTyped मुख्यतः किससे संबंधित है? A. Physical key code only B. Window closing C. File EOF D. Character input
37. Adapter का लाभ? A. Optional callbacks का boilerplate घटाना B. हर exception हटाना C. File encrypt करना D. JVM install करना
38. Standard ActionAdapter? A. Label subtype B. नहीं है C. सभी JDK में है D. Scanner subtype
39. Long work EDT पर करने से? A. Charset बदलता है B. File unique होती है C. UI freeze हो सकती है D. UI हमेशा तेज
40. Resource cleanup के लिए? A. repaint only B. equals only C. Thread priority D. try-with-resources

उत्तर-कुंजी:

1-A, 2-B, 3-C, 4-D, 5-A, 6-B, 7-C, 8-D, 9-A, 10-B, 11-C, 12-D, 13-A, 14-B, 15-C, 16-D, 17-A, 18-B, 19-C, 20-D, 21-A, 22-B, 23-C, 24-D, 25-A, 26-B, 27-C, 28-D, 29-A, 30-B, 31-C, 32-D, 33-A, 34-B, 35-C, 36-D, 37-A, 38-B, 39-C, 40-D.

12.4 Short एवं Long Answer Practice

1. Byte और character stream compare करें; प्रत्येक का practical example दें।
2. InputStreamReader encoding bridge कैसे बनाती है?
3. Scanner के token-based और BufferedReader के line-based input compare करें।
4. Applet lifecycle explain करें और modern execution limitations लिखें।
5. java.lang और java.util की पाँच useful classes/interfaces समझाएँ।
6. Delegation event model को source, event object, listener और registration के साथ explain करें।
7. MouseListener, MouseMotionListener और KeyListener compare करें।
8. Adapter class का example दें; ActionListener के लिए lambda कब संभव है?
9. Layout managers resizing में क्या सहायता करते हैं?
10. Console text-file program और AWT form के resource/error handling compare करें।

12.5 Mini Project: Student Result Form

यह desktop project name और marks लेता है। Calculate validation करके result दिखाता है। Save नया UTF-8 text file बनाता है; existing file overwrite नहीं होती। Pass threshold 40 इस example का चुना हुआ rule है।

```
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;
```

```

public class StudentResultApp {
    static String calculate(String name, String text) {
        String cleanName = name.trim();
        if (cleanName.isEmpty()) {
            throw new IllegalArgumentException("Enter a name");
        }
        int marks = Integer.parseInt(text.trim());
        if (marks < 0 || marks > 100) {
            throw new IllegalArgumentException("Marks: 0 to 100");
        }
        return cleanName + ": " + marks + " ("
            + (marks >= 40 ? "PASS" : "FAIL") + ")";
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            Frame frame = new Frame("Student Result");
            frame.setLayout(new FlowLayout());
            TextField name = new TextField(15);
            TextField marks = new TextField(5);
            Label status = new Label("Enter details", Label.LEFT);
            Button calculate = new Button("Calculate");
            Button save = new Button("Save");
            String[] latest = {" "};
            frame.add(new Label("Name"));
            frame.add(name);
            frame.add(new Label("Marks"));
            frame.add(marks);
            frame.add(calculate);
            frame.add(save);
            frame.add(status);
            calculate.addActionListener(event -> {
                latest[0] = " ";
                try {
                    latest[0] = calculate(
                        name.getText(), marks.getText());
                    status.setText(latest[0]);
                } catch (IllegalArgumentException ex) {
                    status.setText("Invalid name or marks (0-100)");
                }
                frame.validate();
            });
            save.addActionListener(event -> {
                if (latest[0].isEmpty()) {
                    status.setText("Calculate a valid result first");
                    return;
                }
                try {
                    Files.writeString(Path.of("student_result.txt"),
                        latest[0] + System.lineSeparator(),
                        StandardCharsets.UTF_8,
                        StandardOpenOption.CREATE_NEW);
                    status.setText("Saved student_result.txt");
                } catch (FileAlreadyExistsException ex) {
                    status.setText("File exists: choose a new file name");
                } catch (IOException ex) {
                    status.setText("Could not save result");
                }
                frame.validate();
            });
            frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent event) {
                    frame.dispose();
                }
            });
        });
    }
}

```

```

    });
    frame.setSize(650, 180);
    frame.setVisible(true);
});
}
}

```

Expected interaction: Amit + 85 → Amit: 85 (PASS) | Marks 140 या nonnumeric text → validation message | Save creates `student_result.txt` in working directory; existing file होने पर clear message दिखती है। Save last calculated result लिखता है; input बदलने के बाद Calculate फिर दबाएँ।

Project limitation: यह छोटा teaching example है। बड़ी file/network operations worker thread में करें और final UI update EDT पर लौटाएँ। Actual widget appearance platform पर निर्भर है। Name/marks validation rules real institution policy के अनुसार बदलें।

Extend it: FileChooser से destination चुनें; user-approved overwrite flow जोड़ें; multiple subjects और average calculate करें; JTextField/JFrame-based Swing version से compare करें।

12.6 Practical Assessment Checklist

Task	स्वयं जाँचें
Binary copy	Exact bytes और EOF handling correct?
Text I/O	Charset explicit और resources closed?
Console	Invalid input और leftover newline handled?
GUI	Correct listener registration और focus?
Close	dispose callback मौजूद?
Project	0, 39, 40, 100, -1, 101 और nonnumeric marks tested?
Applet	Historical lifecycle और present-day limitations स्पष्ट?

12.7 Official Further Reading

- Oracle Java 17 Scanner API: <https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Scanner.html>
- Oracle Java 17 AWT event package: <https://docs.oracle.com/en/java/javase/17/docs/api/java.desktop/java/awt/event/package-summary.html>
- Oracle JDK 11 Migration Guide, removed deployment tools: <https://docs.oracle.com/en/java/javase/11/migrate/removed-tools-and-components.html>
- Oracle JDK 26 Migration Guide, removed APIs: <https://docs.oracle.com/en/java/javase/26/migrate/removed-apis.html>

Technical documentation पढ़ते समय method signature, return type, exceptions और version को साथ देखें। यह पुस्तक syllabus learning guide है; applet section historical study के लिए है।