

Chapter 3:

JAVA GUI Programming

Revision Tour -I

Informatics Practices

Class XII

By- Rajesh Kumar Mishra

PGT (Comp.Sc.)

KV No.1, AFS, Suratgarh

e-mail : rkmalld@gmail.com

What is JAVA?



- ❑ JAVA is an Object Oriented programming language as well a platform.
 - ❑ By using JAVA, we can write various types of Application program for any type of OS and Hardware.
 - ❑ JAVA is designed to build interactive, dynamic and secure applications on network computer system.
-

History of JAVA

JAVA was started with a project (Green) to find to write applications for electronic devices like TV-Set top Box etc. Which was originally named Oak. Later renamed with JAVA.

1991	James Gosling developed Oak to program consumer electronic devices
1995	JAVA formally announced as a part of Netscape web browser.
1996	Java Development Kit (JDK) 1.0 by Sun Microsystems.
1997	JDK 1.1 launched with JAVA Servlet API
1998	Sun introduced community source "Open" and produces JDK 1.2 for Linux
1999	JDK 1.3 released and J2EE, J2SE, J2ME appeared
2002	JAVA Web Services Developer Pack released for Web Development.
2005	JAVA Enterprise System 2005Q4 released with integration of various features like monitoring, security etc. for Solaris, Windows etc.
2006	The Netbeans IDE 5.0 is released. Sun Open sourced Java EE component as the Glassfish project to JAVA.net.

Characteristics of JAVA

❑ Write Once Run Anywhere (WORA)

JAVA Program can be run on different platforms without any changes.

❑ Light Weight Code

Big applications can be developed with small code.

❑ Security

JAVA Programs are safe and secure.

❑ Built-in Graphics & Supports Multimedia

JAVA is equipped with Graphics feature. It is best for integration of Audio, Video and graphics & animation.

❑ Object Oriented Language

Java is Object Oriented Language, near to real world.

❑ Platform Independent

Change of H/W and OS platform does not effect JAVA program.

❑ Open Product

It is open i.e. freely available to all with no cost.

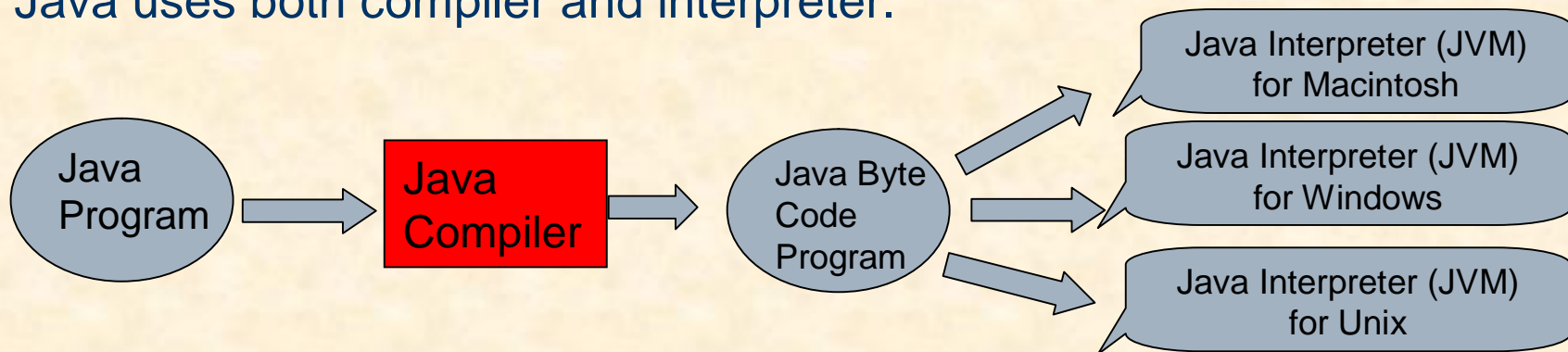
Why JAVA is Platform Independent?

A program written in HLL must be converted into its equivalent Machine code, so that computer can understand and execute. This conversion is known as Compilation. Generally, the converted machine code depends on the H/w and OS platform. So, that a Windows program will not work on UNIX, or LINUX or Mac platform etc. Since they are Platform dependent.

A program written in JAVA is platform-independent i.e. they are not affected with changing of OS. This magic is done by using Byte code. Byte code is independent of the computer system it has to run upon.

Java compiler does not produces **Byte code** instead of native executable code which is interpreted by Java Virtual Machine (**JVM**) at the time of execution.

Java uses both compiler and interpreter.



Basics of GUI

□ How GUI application works ?

Graphical User Interface (GUI) based application contains Windows, Buttons, Text boxes, Dialogue boxes and Menus etc. known as GUI components. While using a GUI application, when user performs an action, an **Event** is generated. Each time an Event occurs, it causes a **Message** which sent to OS to take action.

□ What is Event ?

An Events refers to the occurrence of an activity.

□ What is Message ?

A Message is the information/request sent to the application.

GUI in JAVA

In Java, GUI features are supported through **JFC** (Java Foundation Classes). JFC comprises all the features which are needed to build a GUI application.

Prior to Java 1.2 JFC components was called **Abstract Windows Tools** (AWT). After Java 1.2, a more flexible **Swing Components** was introduces.

A GUI application in JAVA contains **three** basic elements.-

1. Graphical Component:

It is an object that defines a screen element such as Button, Text field, Menus etc. Each component has certain properties. They are source of the Events. In java It is also known as **Widget** (Window Gadget). It can be **container** control or **child** control.

2. Event:

An Event (occurrence of an activity) is generated, when user does something like mouse click, dragging, pressing a key on the keyboard etc.

3. Event Listener:

It contains method/functions which is attached to a component and executed in response to an event. In Java, Listener Interface stores all Event-response-methods or Event-Handler methods.

Basic Graphical Controls of JAVA

(Swing controls)

The palette of controls in NetBeans IDE offers various Java Swing, that can be used in Application frames/ Window/ Form. Commonly used controls are-

- ❑ **JFrame**: Used as a Basic Window or form.
 - ❑ **JLabel**: Allows Non-editable text or icon to displayed.
 - ❑ **TextField**: allows user input. It is editable through text box.
 - ❑ **Button**: An action is generated when pushed.
 - ❑ **CheckBox**: Allow user to select multiple choices.
 - ❑ **RadioButton**: They are option button which can be turned on or off. These are suitable for single selection.
 - ❑ **JList**: Gives a list of items from which user can select one or more items.
 - ❑ **ComboBox**: gives dropdown list of items or new item can be added. It is combination of JList + TextField.
 - ❑ **Panel**: It is container controls which contains other controls using a frame.
-

Events Handling in JAVA GUI Application

An event is occurrence of some activities either initiated by user or by the system. In order to react, you need to implement some Event handling system in your Application. Three things are important in Even Handling-

❑ **Event Source:**

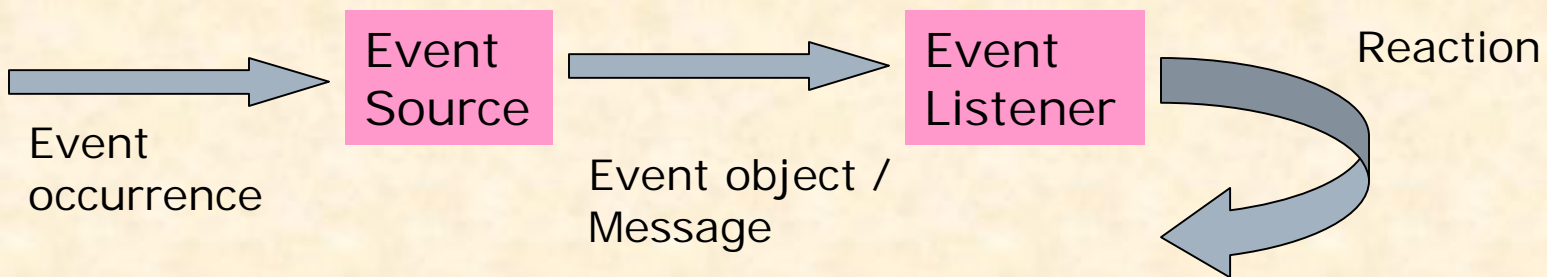
It is the GUI component that generates the event, e.g. Button.

❑ **Event Handler or Event Listener:**

It is implemented as in the form of code. It receives and handles events through Listener Interface.

❑ **Event Object or Message:**

It is created when event occurs. It contains all the information about the event which includes Source of event and type of event etc.



Commonly used Events & Listeners

Java API offers numerous types of events and event listeners. The commonly used events are-

- ❑ **Action Event:**

The Action events occurred when user completes an action on components like JButton, JCheckBox, JTextField etc. To handle Action Event, **ActionListener** interface is used.

- ❑ **Focus Event:**

The Focus event occurred when any components gains or loses focus on components like JButton, JCheckBox, JTextField etc. To handle Focus Event, **FocusListener** interface is used.

- ❑ **Key Event:**

The Key Event occurred when a key is pressed on the keyboard on input enabled components, JTextField etc. To handle Key Event, **KeyListener** interface is used.

- ❑ **Mouse Events:**

When a mouse is clicked, entered or leaved a component area of a control then Mouse event is generated. Any component can generate this event. To handle Mouse Event, **MouseListener** interface is used.

- ❑ **Window Event:**

Window Event occurred when user opens or closes a Window object. This event is generated internally by the system. To handle Window Event, **WindowListener** interface is used.

JAVA character set

- ❑ Character set is a set of valid characters that a language can recognize. It may be any letter, digit or any symbol or sign.
 - ❑ JAVA uses 2-Byte UNICODE character set, which supports almost all characters in almost all languages like English, Chinese, Arabic etc.
 - ❑ In Unicode, first 128 characters are similar to ASCII character set. Next 128 characters equal to Extended ASCII code. Rest are capable to support other languages.
 - ❑ Any character in Unicode can be represented by \u followed by 4 digit Hexadecimal number. E.g. \u0394 to represent Delta Symbol.
-

JAVA Tokens

- ❑ The smallest individual unit in a program is known as Token. It may any word, symbols or punctuation mark etc.
 - ❑ Following types of tokens used in Java-
 - ❖ Keywords
 - ❖ Identifiers
 - ❖ Literals
 - ❖ Punctuators (; [] etc)
 - ❖ Operators (+, -, /, *, =, == etc.)
-

Keywords in Java

- ❑ Keywords are the reserve words that have a special meaning to the compiler.
 - ❑ Key words can't be used as identifiers or variable name etc.
 - ❑ Commonly used key words are-
char, long, for, case, if, double, int, short, void, main, while , new etc.
-

Identifiers in Java

- ❑ Identifiers are fundamental building block of program and used as names given to variables, objects, classes and functions etc.
 - ❑ The following rules must be followed while using identifiers.
 - Identifiers may have alphabets, digits and dollar (\$), underscore (_) sign.
 - They must not be Java keywords.
 - They must not begin with digit.
 - They can be of any length.
 - They are Case Sensitive ie. Age is different from age.
 - ❑ Example of Valid identifiers-
MyFile, Date9_7_7, z2t09, A_2_Z, \$1_to_100, _chk etc.
 - ❑ Example of Invalid identifiers-
Date-RAC, 29abc, My.File, break, for
-

Literals in Java

- ❑ Literals or constants are data items that have fixed data value.
 - ❑ Java allows several types of literals like-
 - Integer Literals
 - Floating Literals
 - Boolean Literals
 - Character Literals
 - String Literals
 - The null literals
-

Integer Literals

- ❑ An integer constant or literals must have at least one +/- digit without decimal point.
 - ❑ Java allows three types of integer literals -
 - Decimal Integer Literals (Base 10)
e.g. 1234, 41, +97, -17 etc.
 - Octal Integer Literals (Base 8)
e.g. 010, 014 (Octal must start with 0)
 - Hexadecimal Integer Literals (Base 16)
e.g. 0xC, 0xab (Hex numbers must start with 0x)
 - ❑ L or U suffix can be used to represent long and unsigned literals respectively.
-

Floating / Real Literals

- ❑ A real literals are fractional numbers having at least one digit before and after decimal point with + or – sign.
The following are valid real numbers-
2.0, 17.5, -13.0. -0.00626
The following are invalid real numbers-
7, 7. , +17/2, 17,250.26 etc.

 - ❑ A real literals may be represented in Exponent form having Mantissa and exponent with base 10 (E). Mantissa may be a proper real numbers while exponent must be integer.
The following are valid real in exponent form-
152E05, 1.52E07, 0.152E08, -0.12E-3, 1.5E+8
The following are invalid real exponent numbers-
172.E5, 1.7E, 0.17E2.3, 17,22E05, .25E-7
-

Other Literals

- ❑ The Boolean Literals represents either TRUE or FALSE. It always Boolean type.
 - ❑ A null literals indicates nothing. It always null type.
 - ❑ Character Literals must contain one character and must enclosed in single quotation mark.
e.g. 'a', '%' , '9' , '\\' etc.
Java allows some non-graphic characters (which can not be typed directly through keyboard) by using Escape sequence (\) .
E.g.

\a (alert),	\b (backspace),	\f (Form feed),
\n (new line),	\r (return key),	\t (Horizontal tab),
\v (vertical tab),	\\ (back slash),	\' (single quote) ,
\\" (double quote) ,	\? (question mark),	\0 (null) etc.
 - ❑ String Literals is a sequence of zero or more characters enclosed in double quotes. E.g. "abs", "amit" , "1234" , "12 A" etc.
-

Data types in JAVA

- Data types are means to identify the type of data and associated operations of handling it.
Java offers two types of data types.

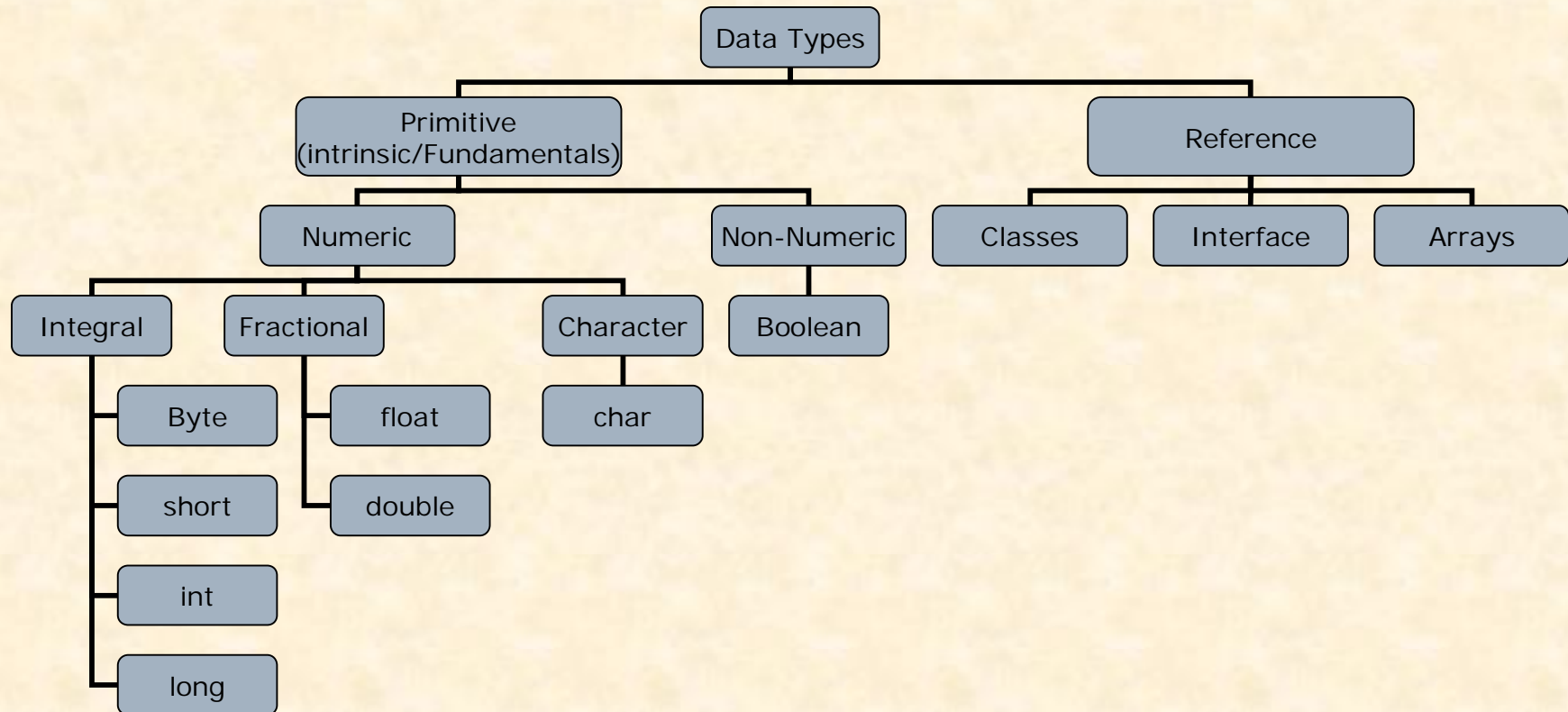
- **Primitive:**

These are in-built data types offered by the compiler. Java supports 8 primitive data types e.g. byte, short, int, long, float, double, char, boolean.

- **Reference:**

These are constructed by using primitive data types, as per user need. Reference data types store the memory address of an object. Class, Interface and Array are the example of Reference Data types.

Data Types in Java



String Data type is also used in Java as Reference data type

Primitive Data types

Type	Size	Description	Range
byte	1 Byte	Byte integer	-128 to +128
short	2 Byte	Short integer	-32768 to +32767
int	4 Byte	integer	-2^{31} to $2^{31}-1$
long	8 Byte	Long integer	-2^{63} to $2^{63}-1$
float	4 Byte	Single precision floating point (up to 6 digit)	$-3.4E+38$ to $+3.4E+38$
double	8 Byte	Double precision floating (up to 15 digit)	$-1.7E+308$ to $1.7E+308$
char	2 Byte	Single character	0 to 65536
Boolean	1 Byte	Logical Boolean values	True or False

- **L** suffix can used to indicate the value as long.
 - By default Java assume frictional value as double, **F** and **D** suffix can be used with number to indicate float and double values respectively.
-

Working with Variables

- ❑ A variable is named memory location, which holds a data value of a particular data type.
- ❑ Declaration and Initialization of variable-
<data type> <variable Name>;

Example:

```
int age;  
double amount;  
double price=214.70, discount =0.12;  
String name="Amitabh"  
long x=25L;  
byte a=3;  
float x= a+b;
```

- ❑ By default all Numeric variables initialized with 0, and character and reference variable with null, boolean with false, if it is not initialized.
 - ❑ The keyword **final** can be used with variable declaration to indicate **constant**.
E.g. final double SERVICE_TAX=0.020
-

Text interaction in JAVA GUIs

In GUI application often we require to store the values of text fields to variable or vice-versa. Java offers three method for this purpose-

□ **getText():**

It returns the text stored in the text based GUI components like Text Field, Text Area, Button, Label, Check Box and Radio Button etc. in string type.

e.g. `String str1=jTextField1.getText();`

□ **parse.....()**

This method convert textual data from GUI component in to numeric type.

<code>Byte.parseByte(String s)</code>	– string into byte.
<code>Short.parseShort(String s)</code>	– string into short.
<code>Integer.parseInt(string s)</code>	– string into integer.
<code>Long.parseLong(string s)</code>	– string into long.
<code>Float.parseFloat(string s)</code>	– string into float.
<code>Double.parseDouble(string s)</code>	– string into double.

e.g. `int age=Integer.parseInt(jTextField1.getText());`

□ **setText()**

This method stores string into GUI component.

e.g. `jTextField1.setText("Amitabh");`
`jLabel1.setText(""+payment);`

Displaying Dialogue Boxes in JAVA GUIs

In GUI application often we require to display a message in the Dialog Boxes containing OK button to close the Dialog Box. The following steps can be used to display a message in a dialog box.

- ❑ Firstly, you need to import JOptionPane swing control at the top of program code, by typing -

```
import javax.swing.JOptionPane;
```

- ❑ When required you may display a message by following code in a method-

```
JOptionPane.showMessageDialog(null, "Hello.. ");
```

- ❑ In non-GUI application or at console window you may use System.out.print() method to display a message.

e.g. `System.out.print("Hello...");`

```
System.out.println("How are you?");
```

A sample Java Program

// A non-GUI Program to calculate area of circle//

```
import java.io.*;
class MyProgram
{
    public static void main(String arg[]) throws IOException
    {
        final float PI= 3.14;
        float a;
        int r = 5;
        a=PI*(r*r);
        System.out.println("Area of circle =" + a);
        System.out.println("Bye...");
    }
}
```

Operators in Java

- ❑ The operators are symbols or words, which perform specified operation on its operands.
 - ❑ Operators may be Unary, Binary and Ternary as per number of operands it requires.
 - ❑ Java offers the following types of Operators: -
 - ❖ Arithmetic Operator
 - ❖ Increment/Decrement Operator
 - ❖ Relational or Comparison Operators
 - ❖ Logical Operators
 - ❖ Assignment Operators
 - ❖ Other Operators.
-

1. Arithmetic Operators

+	Unary plus	Represents positive values.	<code>int a = +25;</code>
-	Unary minus	Represents negative values.	<code>int a = -25;</code>
+	Addition	Adds two values	<code>int x = a + b;</code>
-	Subtraction	Subtract second operands from first.	<code>int x = a - b;</code>
*	Multiplication	Multiplies two values	<code>int x = a * b;</code>
/	Division	Divides first operand by second	<code>int x = a / b;</code>
%	Modulus (remainder)	Finds remainder after division.	<code>int x = a % b;</code>
+	Concatenate or String addition	Adds two strings	<code>"ab" + "cd" ⇨ "abcd"</code> <code>"25" + "12" ⇨ "2512"</code> <code>"" + 5 ⇨ "5"</code> <code>"" + 5 + "xyz" ⇨ "5xyz"</code>

2. Increment & Decrement Operator

- Java supports `++` and `--` operator which adds or subtract 1 from its operand. i.e.

`a=a+1` equivalent to `++a` or `a++`

`a=a-1` equivalent to `--a` or `a--`

- `++` or `--` operator may used in **Pre** or **Post** form.

`++a` or `--a` (increase/decrease before use)

`a++` or `a--` (increase/decrease after use)

Ex. Find value of P? (initially n=8 and p=4)

`p=p* --n; ⇨ 28`

`p=p* n--; ⇨ 32`

Ex. Evaluate `x=++y + 2y` if `y=6`.

$$= 7 + 14 = 21$$

3. Relational Operator

- ❑ Relational operators returns true or false as per the relation between operands.
- ❑ Java offers the following six relational operators.

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

Relational operators solved from left to right.

Ex: $3 >= 3$ true

$3 != 3$ false

$a == b$ true if a and b have the same value.

$a < b < c ==> (a < b) < c$ true if a is smallest.

4.Logical Operator

- ❑ Logical operators returns true or false as per the condition of operands. These are used to design more complex conditions.
- ❑ Java offers the following six (5 binary and 1 unary) logical operators.

Operator	Name	use	Returns true if
&&	And	x&&y	X and y both true
	Or	x y	Either x or y is true
!	Not	!x	X is false
&	Bitwise and	x&y	X and y both true
	Bitwise or	x y	Either x or y is true
^	Exclusive or	x^y	If x and y are different

Ex: 5>8 || 5<2 (false)

6<9 && 4>2 (true)

!(5!=0) (false)

1==0||0>1 (false)

6==3&&4==4 (false)

!(5>9) (true)

5. Assignment Operator

- In Java = operator is known as Assignment operator, it assigns right hand value to left hand variables.

Ex: `int x=5;`

`z= x+y;`

- Java offers some special shortened Assignment operators, which are used to assign values on a variable.

Operator	use	Equivalent to
<code>+=</code>	<code>X+=y</code>	<code>X=x+y</code>
<code>-=</code>	<code>X-=y</code>	<code>X=x-y</code>
<code>*=</code>	<code>X*=y</code>	<code>X=x*y</code>
<code>/=</code>	<code>x/=y</code>	<code>X=x/y</code>
<code>%=</code>	<code>X%=y</code>	<code>X=x%y</code>

Ex: `x-=10` => `x=x-10`

`x%=y` => `x=x%y`

6. Other Operators

- ❑ In Java some other operators are also used for various operations. Some operators are-

Operator	Equivalent to
? :	Shortcut of If condition (ternary operator) <Condition> ? <true action> : <false action>
[]	Used to declare array or access array element
.	Used to form qualified name (refer)
(type)	Converts values as per given data type
new	Creates a new object
instanceof	Determines whether the first operator is instance of other.
<<, >>	Performs bitwise left shift or right shift operation.
~	(compliment) Inverts each bit (0 to 1 or 1 to 0)

Ex. result = marks >= 50 ? 'P' : 'F'

6 > 4 ? 9 : 7 evaluates 9 because 6 > 4 is true.

Operator's Precedence

- Operator's precedence determines the order in which expressions are evaluated. There is certain rules for evaluating a complex expression.

e.g. $y=6+4/2$ (why 8 not 5 ?)

Operators	Remark	Associativity
. [] ()	() used to make a group, [] used for array and . Is used to access member of object	L to R
++ -- ! ~	Returns true or false based on operands	R to L
New (type)	New is used to create object and (type) is used to convert data into other types.	R to L
* / %	Multiplication, division and modulus	L to R
+ -	Addition and Subtraction	R to L
== !=	Equality and not equality	L to R
&	Bitwise And	L to R
^	Bitwise Exclusive Or	L to R
	Bitwise or	L to R
&&	Logical And	L to R
	Logical or	L to R
? :	Shortcut of IF	R to L
= += -= *= /= %=	Various Assignment operators	R to L

Expression in Java

- ❑ An expression is a valid combination of operators, constants and variable and keywords i.e. combination of Java tokens.
- ❑ In java, **three** types of expressions are used.

- ❑ **Arithmetic Expression**

Arithmetic expression may contain one or more numeric variables, literals and operators. Two operands or operators should not occur in continuation.

e.g. $x + *y$ and $q(a + b - z/4)$ is invalid expressions.

Pure expression: when all operands are of same type.

Mixed expressions: when operands are of different data types.

- ❑ **Compound Expression**

It is combination of two or more simple expressions.

e.g. $(a + b)/(c + d)$ $(a > b) || (b < c)$

- ❑ **Logical Expression**

Logical or Boolean expression may have two or more simple expressions joined with relational or logical operators.

e.g. $x > y$ $(y + z) >= (x/z)$ $x || y \ \&\& \ z$ (x) $(x - y)$

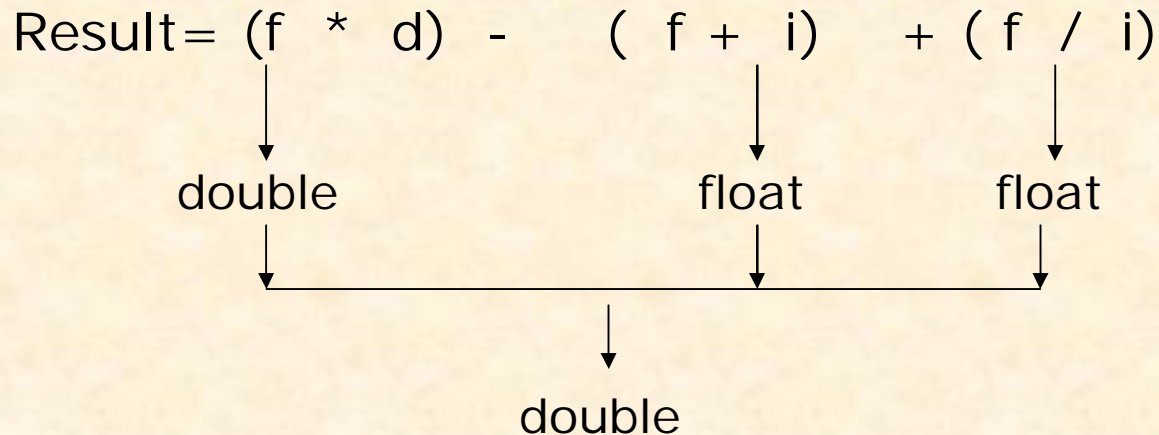
Type Conversion in JAVA

- ❑ The process of converting one predefined type into another is called type conversion.
 - ❑ In mixed expression, various types of constant and variables are converted into same type before evaluation.
 - ❑ Java facilitates two types of conversion.
 - ✓ Implicit type conversion
 - ✓ Explicit type conversion
-

Implicit Type Conversion

- ❑ It is performed by the compiler, when different data types are intermixed in an expression. It is also called **Coercion**.
- ❑ In Implicit conversion, all operands are promoted (Coercion) up to the **largest data** type in the expression.

Ex. Consider the given expression, where f is float, d is double and I is integer data type.



Explicit Conversion in JAVA

- ❑ An explicit conversion is user defined that forces to convert an operand to a specific data type by (type) cast.
Ex. (float) (x/2) suppose x is integer.
The result of x/2 is converted in float otherwise it will give integer result.
 - ❑ In pure expression the resultant is given as expression's data type.
e.g. 100/11 will give 9 not 9.999 (since both are integer)
 - ❑ In mixed expression the implicit conversion is applied (largest type promotion)
e.g. int a, mb=2, k=4 then evaluate
$$\begin{aligned} & a = mb * 3 / 4 + k / 4 + 8 - mb + 5 / 8 \\ & = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8 \\ & = 6 / 4 + 1 + 8 - 2 + 5 / 8 \\ & = 1 + 1 + 8 - 2 + 0 \quad (6/4 \text{ will give } 1) \\ & = 8 \end{aligned}$$
-

JAVA Statements

- ❑ A statement in Java is a complete unit of execution. It may consists of Expression, Declaration, Control flow statements and must be ended with semicolon (;)
- ❑ Statements forms a block enclosed within { }. Even a block may have no statement (empty).

E.g. If(a>b)
 {

 }

- ❑ Note:

System.out.print('h'+'a') will give 169

System.out.print(""+'h'+'a') will give ha

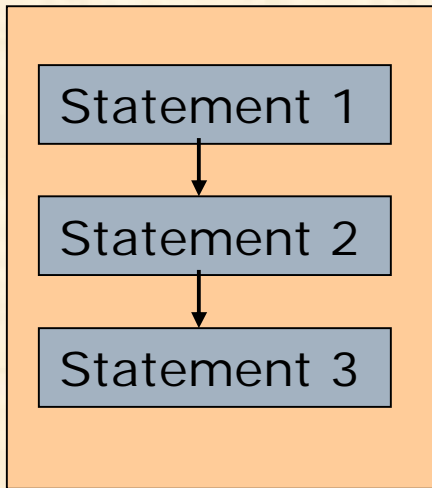
System.out.print("2+2=" + 2+2) will give 2+2=22

System.out.print("2+2=" + (2+2)) will give 2+2=4

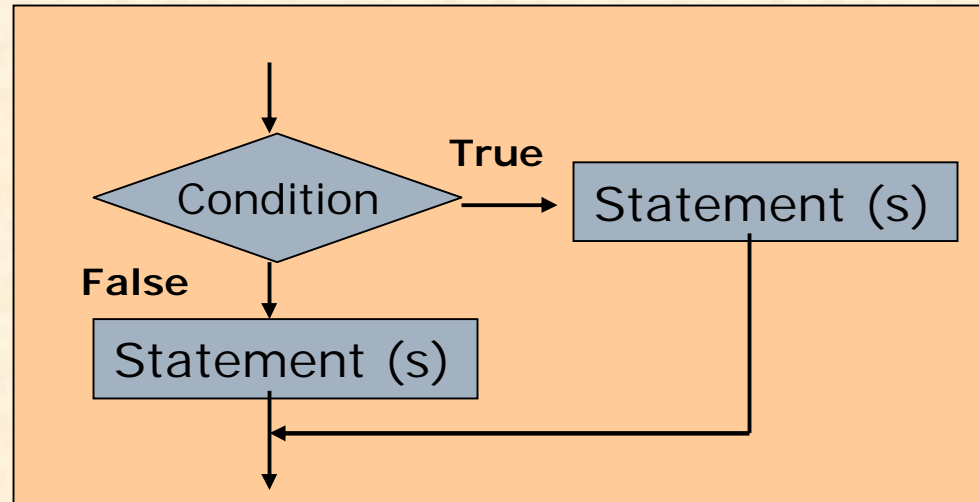
Programming Constructs in JAVA

- In general a program is executed from begin to end. But sometimes it required to execute the program selectively or repeatedly as per defined condition. Such constructs are called control statements.
 - The programming constructs in Java, are categorized into -
 - **Sequence:**
Statements are executed in top down sequence.
 - **Selection (conditional):**
Execution of statement depends on the condition, whether it is True or False.
(Ex. if.., if...else, switch constructs)
 - **Iteration (Looping):**
Statement is executed multiple times (repetition) till the defined condition True or False.
(Ex. for.. , while... , do..while loop constructs)
-

Diagrammatic Representation

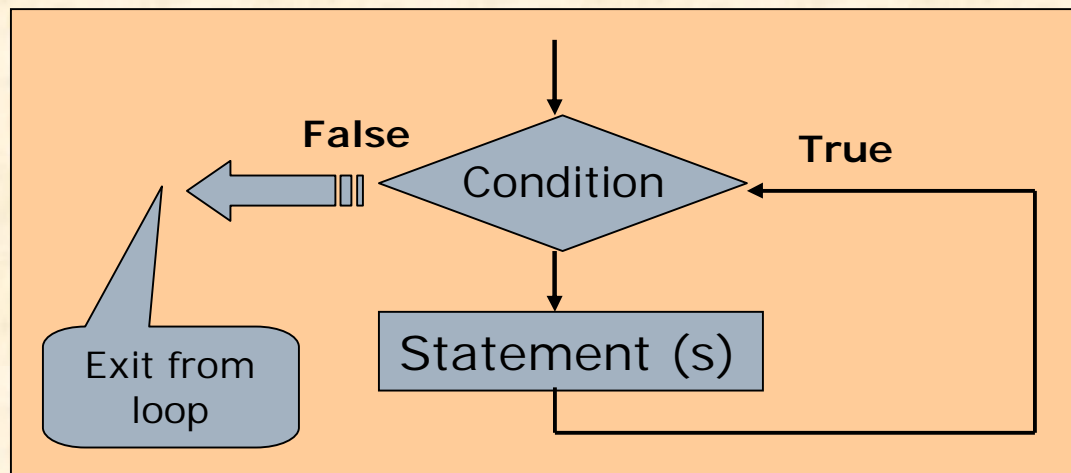


Sequence construct



Selection construct

Iteration Construct



Selection statement (if..)

- ❑ The if... statement is used to test a condition. If defined condition is true the statements are executed otherwise they are ignored.
- ❑ The condition may be simple or complex (using &&, || etc.) and must be enclosed in ().
- ❑ Expression defined as condition must be evaluated as True or False.

Syntax

```
if (condition)
{
    statement 1 ;
    .....
}
```

```
if ( num>0) {
    JLabel1.setText("Number is positive");
}

if ( ch>='0' && ch<='9' ) {
    JLabel1.setText("It is digit");
}
```

In case of single statement in if... the use of { } is optional.

Selection statement (if..else..)

- ❑ The if..else.. also tests condition. If defined condition is true the statements in **True block** are executed otherwise **else block** is executed.

```
if (condition)
{
    statement 1 ;
    .....
}
else
{
    statement 2;
    .....
}
```

```
if ( num>0) {
    jLabel1.setText("Number is positive");
}
else
{
    jLabel1.setText("Number is zero or negative");
}
```

```
if ( age >= 18)
    jLabel1.setText("Eligible to Vote");
else
    jLabel1.setText("Not eligible to Vote");
```



In case of single statement **{ }** is optional.

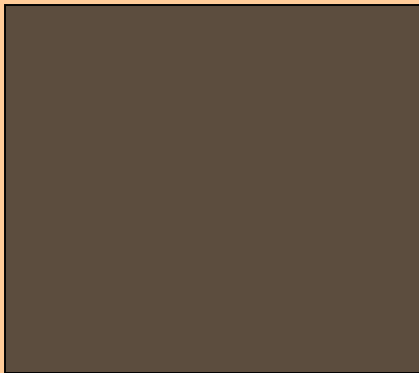
Nested if...

- An if... or if..else.. may have another if.. Statement in its true block or in false block. It is known as Nesting of if (placing an if inside another if).

if (condition1){



else{



}

```
if ( num>0)
{
    JLabel1.setText("Number is positive");
}
else
{ if (num<0)
    JLabel1.setText("Number is negative");
  else
    JLabel1.setText("Number is zero");
}
```



Nested if.. block

If...else...If ladder

- When a else block contains another if.. and this nested else block may have another if and so on. This nesting is often known as if..else..if ladder or staircase.

```
if (condition1)
    statement 1;
else
    if (condition 2)
        statement 2;
    else
        if (condition 3)
            statement 3;
        else
            if(condition 4)
                statement 4;
            .....
            .....
        else
            statement n;
```

```
if (day==1)
    JLabel.setText("Sunday");
else
    if (day==2)
        JLabel.setText("Monday");
    else
        if (day==3)
            JLabel.setText("Tuesday");
        else
            if(day==4)
                JLabel.setText("Wednesday ");
            else
                if(day==5)
                    JLabel.setText("Thrusday");
                else
                    if(day==6)
                        JLabel.setText("Friday");
                    else
                        JLabel.setText("Saturday");
```


Conditional operator and if.. statement

- ❑ The ? : (conditional operator) may be used as alternative to if..else.. statement in Java.
- ❑ In contrast to if..., ?: is more concise and compact code it is less functional.
- ❑ ?: produces an expression so that a single value may be incorporated whereas if.. is more flexible, whereas you may use multiple statements, expressions and assignments.
- ❑ When ?: is used as nested form, it becomes more complex and difficult to understand.

Syntax

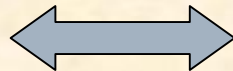
Expression 1 ? Expression 2: expression 3;

```
if ( a>b)
```

```
    c=a;
```

```
else
```

```
    c=b;
```



```
C = a>b ? a : b ;
```


The switch statement

- ❑ Multi-branching selection can be made in Java by using switch statement.
- ❑ It test successively, the value of an expression (short, int, long or char type), when match is found, the statements associated with constant is executed.

```
switch (<expression>
{ case <const 1> : statement (s);
                        break;
  case <const 2> : statement (s);
                        break;
  case <const 2> : statement (s);
                        break;

  .....
  [default : statement (s);]
}
```

```
switch (day)
{ case 1 : Dow="Sunday";
      break;
  case 2 : Dow="Monday";
      break;
  case 3 : Dow="Tuesday";
      break;
  case 4 : Dow="Wednesday";
      break;
  case 5 : Dow="Thursday";
      break;
  case 6 : Dow="Friday";
      break;
  case 7 : Dow="Saturday";
      break;
  default : Dow="Wrong Input";
}
jLabel.setText("Weak day"+Dow);
```

- 
1. No two identical constant can be used.
 2. Default.. is optional may be anywhere in switch block.

Switch and if..else statement

The switch and if..else both are used to make a selection construct in Java, but there some differences.

- ❑ Switch can test only equality whereas if.. Can evaluate any type of relational or logical expression.
- ❑ In switch a single value or constant can be tested but in if.. more versatile expression can be tested.
- ❑ The switch statement can handle only byte, short, int or char variable but If.. can test more data type like float, double or string etc.

Nesting of switch with switch or if.. may be used....

```
if ( condition 1)
{ switch (exp1)
  { .....
    .....
  }
}
else
{ .....;
}
```

```
switch (exp 1)
{ case <cont>:switch (exp2)
  { .....
    .....
  }
  .....
}
```


Iteration (looping) statements

- ❑ Iteration or looping allow a set of instructions to be executed repeatedly until a certain condition is true or false.
 - ❑ As per placing of condition to be tested, a loop may be **Entry-controlled** or **Exit-controlled** loop. In Entry controlled loop, a condition is tested (pre test) before executing the statements. Whereas in Exit-controlled statements are executed first then condition is tested (post test), which ensures at least on time execution of statements.
 - ❑ As per number of execution, a loop may be **Counter-controlled** or **Sentinel** loop. Counter controlled loops are executed fixed number of times, but sentinel loop may be stopped any time by giving its sentinel value. i.e. number of execution can not be forecasted.
 - ❑ A body of loop contains a block, having statements to be executed.
-

The for .. loop

In simple use, a for.. Loop is Entry-controlled and counter controlled loop having fixed number of iteration.

```
for (initialization exp (s) ; condition ; update exp (s) )  
{ .....  
  .....  
}
```



The diagram illustrates the components of a for loop. A callout box labeled "Looping statements" points to the body of the loop, which is enclosed in curly braces. The body contains two lines of code, each followed by a series of dots, representing the statements that are repeated during each iteration of the loop.

```
for (int i=1; i<=10 ; i++ ) {  
    System.out.println (""+i);  
}
```

```
//loop to find even nos. up to 50  
for (int i=0; i<=50 ; i=i+2)  
    System.out.println (""+i);
```

```
//loop to find factorial  
int fact=1,num=5;  
for (int i=1; i<=num ; i++)  
    fact=fact * i;  
System.out.println ("factorial="+fact);
```

```
//loop to get sum of first 10 nos.  
int sum=0;  
for (int i=1; i<=10 ; i++ ) {  
    sum=sum+ i;  
}  
System.out.println (""+sum);
```

Variation in for.. loop

❑ Multiple initialization and update expression

A for.. Loop may contain multiple initialization and/or update expressions separated by (,)

```
for (i=0,sum=0;i<=n; sum++, i++)
```

❑ Optional Expressions

In for loop initialization, condition and update section may be omitted. Note that (;) must be present.

```
for ( ; i<=n ; )
```

❑ Infinite loop

For.. Loop may be endless (infinite) when defined condition is always true or it is omitted.

```
for ( ; ; )
```

❑ Empty loop

for.. Loop may not contain any statement in looping body i.e. Empty loop. If (;) placed after for.. Loop then empty loop is created.

```
for (int i=1; i<=300 ; i++) ;
```

❑ Variable declared inside for.. Loop can't accessed outside the loop. Since it is out of scope.

The while.. loop

In simple use, a while .. Loop is Entry-controlled and counter controlled or sentinel looping statement, which executes statements until defined condition is true.

```
while (condition)
```

```
{ .....
```

```
.....
```

```
}
```

Looping statements

```
int i=1;
while ( i<=10) {
    i=i+1;
    System.out.println (" "+i);
}
```

//while loop to find factorial


```
int fact=1,num=5,i=1;
while (i<=num)
{ fact=fact * i;
  i++;
}
System.out.println ("factorial="+fact);
```

A while loop also may be empty or infinite

The do..while loop

Unlike for.. and while.. Loop, do..while loop is Exit-controlled and counter controlled or sentinel looping statement, which executes statements until defined condition is true. It always executes at least once.

```
do
{ .....
  .....
} while (condition);
```



```
// do.. Loop to print A – Z letters
char ch = 'A';
do {
    System.out.println (" "+i);
    ch++;
} while (ch <= 'Z');
```

```
//do.. loop fixed time
int i=1;
do
{ System.out.println (" "+i);
  i++;
} while (i <= 10);
```

A do...while loop also may be empty or infinite

Which loop is better ?

- ❑ Java offers three looping statements i.e. for..., while.. and do..while. There are some situation where one loop is more appropriate than others.
 - ❑ The for loop is best suited where number of execution is known in advance. (fixed execution)
 - ❑ The while and do.. Are more suitable in the situations where it is not known that when loop will terminate. (unpredictable times of execution).
 - ❑ The do.. Loop ensures at least one time of execution since it is Exit-controlled loop.
-


Jump statement in Java

- ❑ Java offers three jump statements (return, break and continue), which transfers the control else where unconditionally.
 - ❑ The **return** statement can be used any where in the program. It transfers the control to calling module or Operating System. However Java provides System.exit() method to stop the execution of program.
 - ❑ The **break** is used with for.., while, do.. and switch statements which enables the program to skip over some part of the code and places control just after the nearest closing of block. It is used to terminate the loop.
 - ❑ The **continue** statement is used within looping statement (not with switch) and works like break i.e. it also skips the statements. Unlike break, it forces the next iteration of loop by skipping the in between code and continues the loop.
-

Break and Continue the loop


```
While (condition 1)
{ statement 1;
  if (condition 2)
    break;

  .....
  statement 2;
}
Statement 3;
```




```
for (ini;cond;update)
{ statement 1;
  if (condition)
    break;

  .....
  statement 2;
}
Statement 3;
```



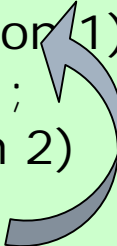
```
Do
{ statement 1;
  if (condition)
    break;

  .....
  statement 2;
} While (test condition)
Statement 3;
```




```
While (condition 1)
{ statement 1;
  if (condition 2)
    continue;

  .....
  statement 2;
}
Statement 3;
```




```
for (ini;cond;update)
{ statement 1;
  if (condition)
    continue;

  .....
  statement 2;
}
Statement 3;
```



```
Do
{ statement 1;
  if (condition)
    continue;

  .....
  statement 2;
} While (test condition)
Statement 3;
```

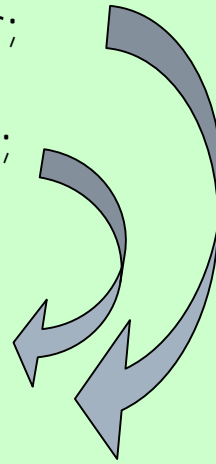


Using Labels with Break and Continue statement

Java also provides a mechanism for exiting nested loops. We can define a Label (flag). Label is an identifier followed by (:), which identifies a place where control is to be transferred. In such case we may use Label with break or continue statement.

```
Outer:
for (int i=0; i<=n; i++)
{ .....
    Inner:
    for(int j=1;j<=m;j++)
    { if (condition)
        break Outer;
      else
        break Inner;
      .....
    }
    statement 2;
}
Statement 3;
```

break [label name];
continue [label name];



Scope of a variable

- ❑ In Java, a variable can be declared anywhere in the program but before using them.
- ❑ The area of program within which a variable is accessible, known as its scope.
- ❑ A variable can be accessed within the block where it is declared.

