JAVA PROGRAMMING

CONTENTS

Contents

- Introduction
- History and Overview of Java
- Object Oriented Program
- Simple Program
- Select Statement
- Control Statements
- Looping Statements
- Jumping Statements

Introduction

- **Java** – It stands for **J**ust **A**nother **V**irtual **A**rchitecture
- It is a programming Language
- Java is a Platform Independent Language
- It is a purely object Oriented Language
- Developed by Sun Micro system
- Founder : James Gosling and Patrick Naughton
- Year: 1995
- Names : Oak Tree, Java

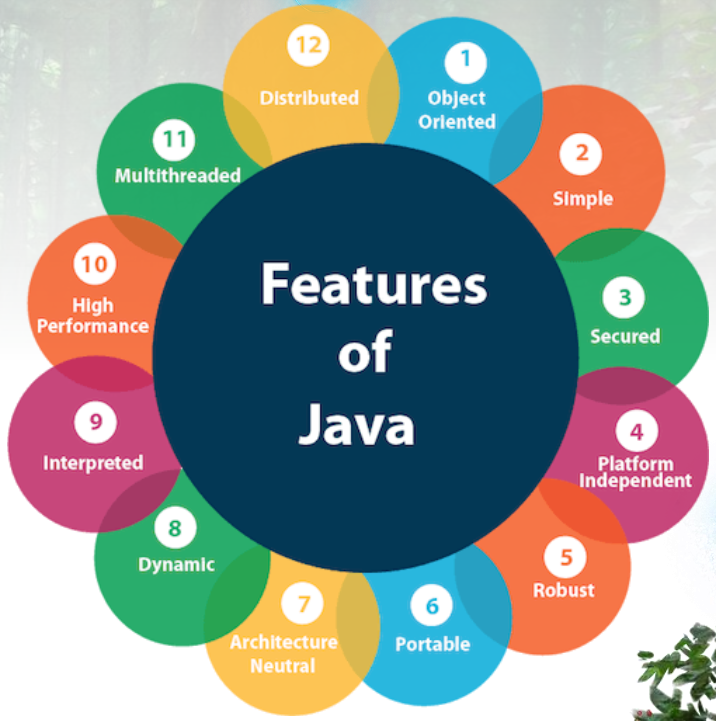
"Father of Java"
James Gosling
CTO, Liquid Robotics

History

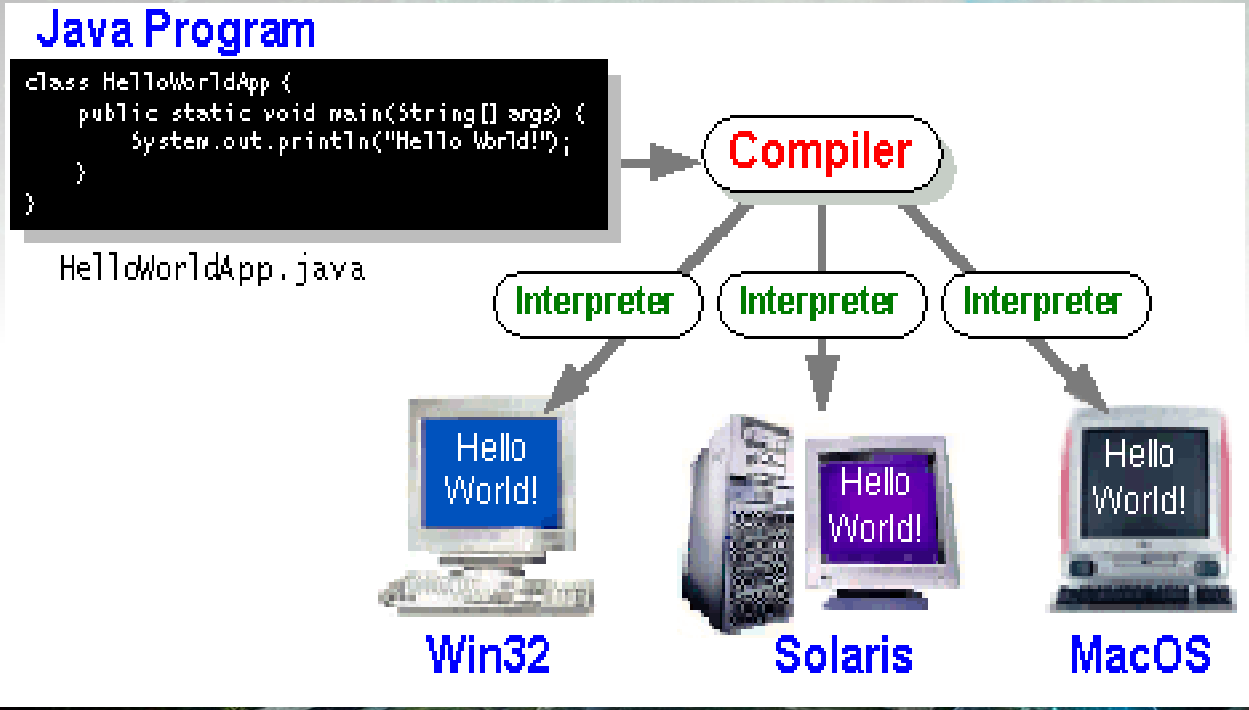
- James Gosling initiated the Java language project in June 1991 for use in one of his many set-top box projects.
- The language, initially called Oak after an oak tree that stood outside Gosling's office, also went by the name Green and ended up later being renamed as Java, from a list of random words.
- Sun released the first public implementation as Java 1.0 in 1995.
- It promised Write Once, Run Anywhere (WORA), providing no-cost run-times on popular platforms. On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).
- On 8 May 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

Features

- Object Oriented
- Platform independent
- Simple
- Secure
- Architectural-Neutral
- Portable
- Robust
- Multi Threaded
- **WORA – Write Once Read Anywhere**



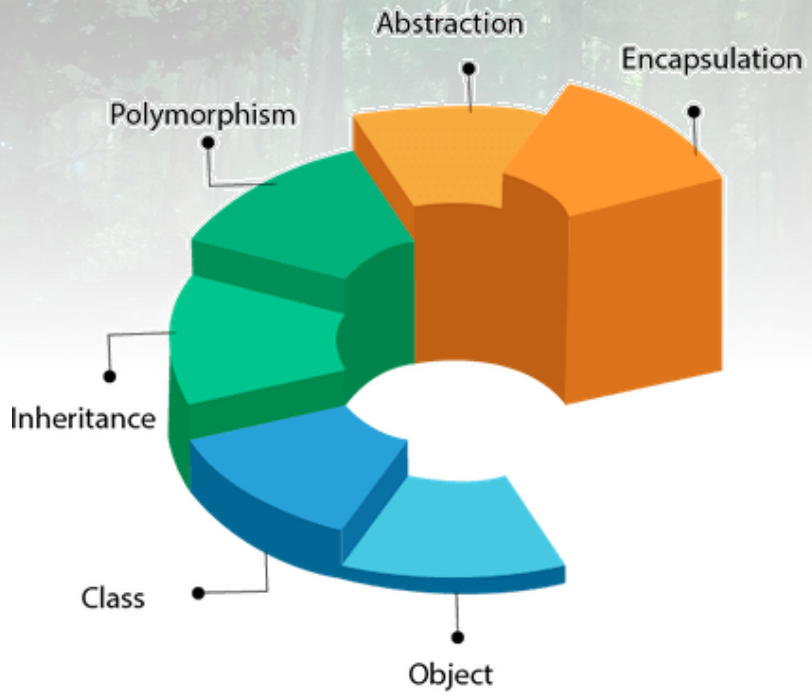
Platform Independance



Object Oriented Programming System

- Class and Objects
- Data Abstraction
- Data Encapsulation
- Inheritance
- Polymorphism

Object Oriented Programming System



Why OOPS important

- **Reduces complexity:** OOP promotes the reuse of data, helping reduce development time and complexity. Using OOP concepts in Java, you can write a functionality once and reuse it everywhere else.
- **Reduces maintenance time:** OOP makes projects modular, allowing you to isolate and solve issues easier. For example, if the bill amount is not right, it means that the problem is with the Bill class and one can go directly there and start debugging.
- **Widely applicable:** OOP can be used to model any scenario imaginable, making it highly useful and applicable in a variety of business use cases.

Object Oriented Programming System

Simple Program

```
class Hello
{
        public static void main (String args [] )
         {
             System.out.println("Welcome to the Java World");
          }
}
```

Explanation

class : It is a keyword. It should be in small case.

Hello : It is a class name. User defined name, user can give any name and the name of the class should start with a character and must not be a digit.

Ex: Suhan, r5 and not 5r

{} : **C**urly braces

; Terminator or Semicolen

Explanation

public: Anyone can access from anywhere and common to all.

static: It means fixed and without creating an object user can access the class.
This is common to all.

void: It is an empty data type and it will not return anything.

main: In every programming language, the action execution always start with the main function.

String: Collection of characters, in java everything will be considered as string.

a[]: It is an array type of argument. User can use any variable name here.
Ex: args,hai,ram etc

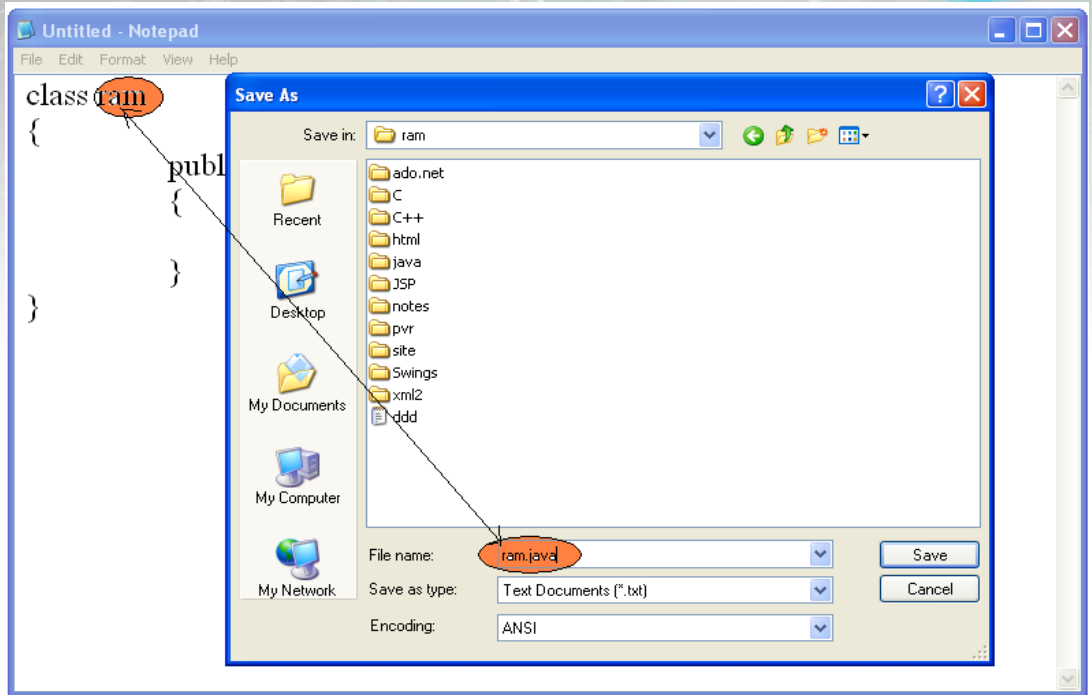
Explanation

Untitled - Notepad

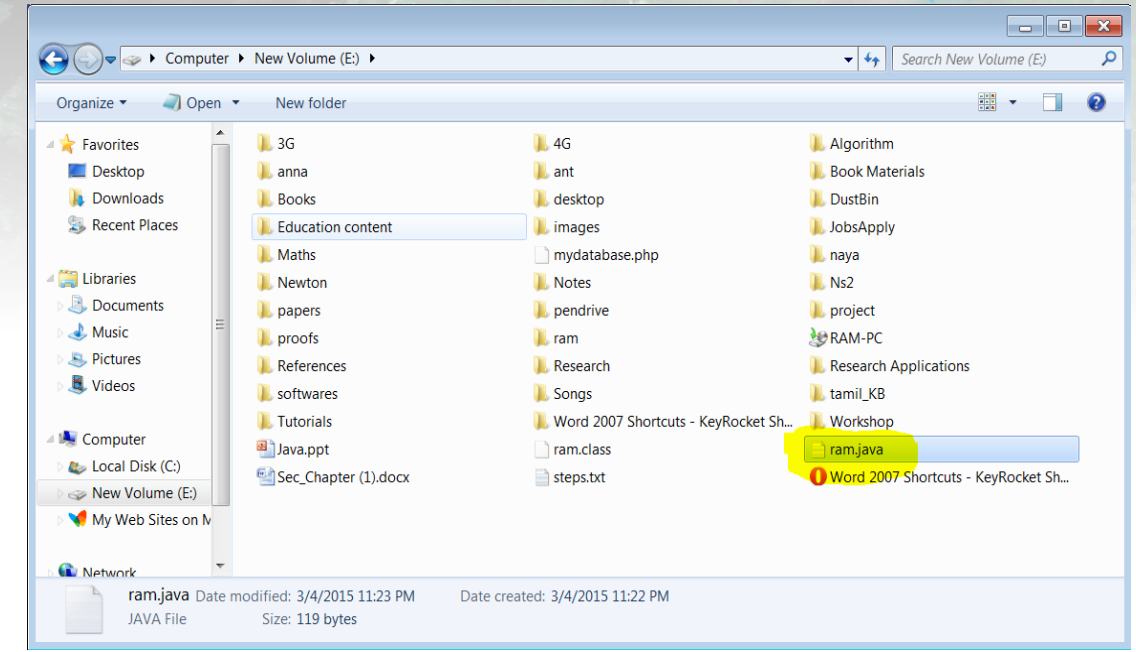
File Edit Format View Help

```
class ram
{
        public static void main (String a[])
        {
                System.out.println ("Welcome to Java");
        }
}
```

Explanation



Explanation



Execution

Execution Commands :

For Compilation	**javac [filename].java**
For Execution	**java [filename]**

Example: **javac ram.java**
java ram

Explanation

Visual Studio 2008 Command Prompt

```
D:\>set path=C:\Program Files\Java\jdk1.7.0\bin

D:\>cd alex

D:\alex>javac ram.java

D:\alex>java ram
Welcome to Java

D:\alex>
```

Editors

Command Prompt
Notepad
Notepad++
Sublime Text
VS Code
Word pad
C,C++
etc

Execution

source code

`Hello.java`

Hello.java

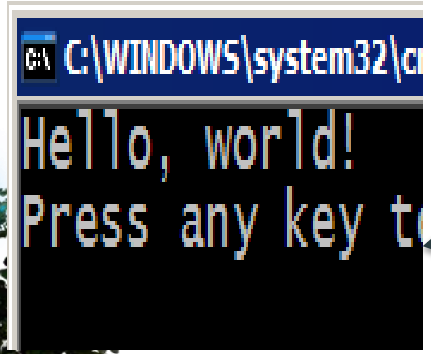
javac Hello.java

byte code

`Hello.class`

Hello.class

java Hello

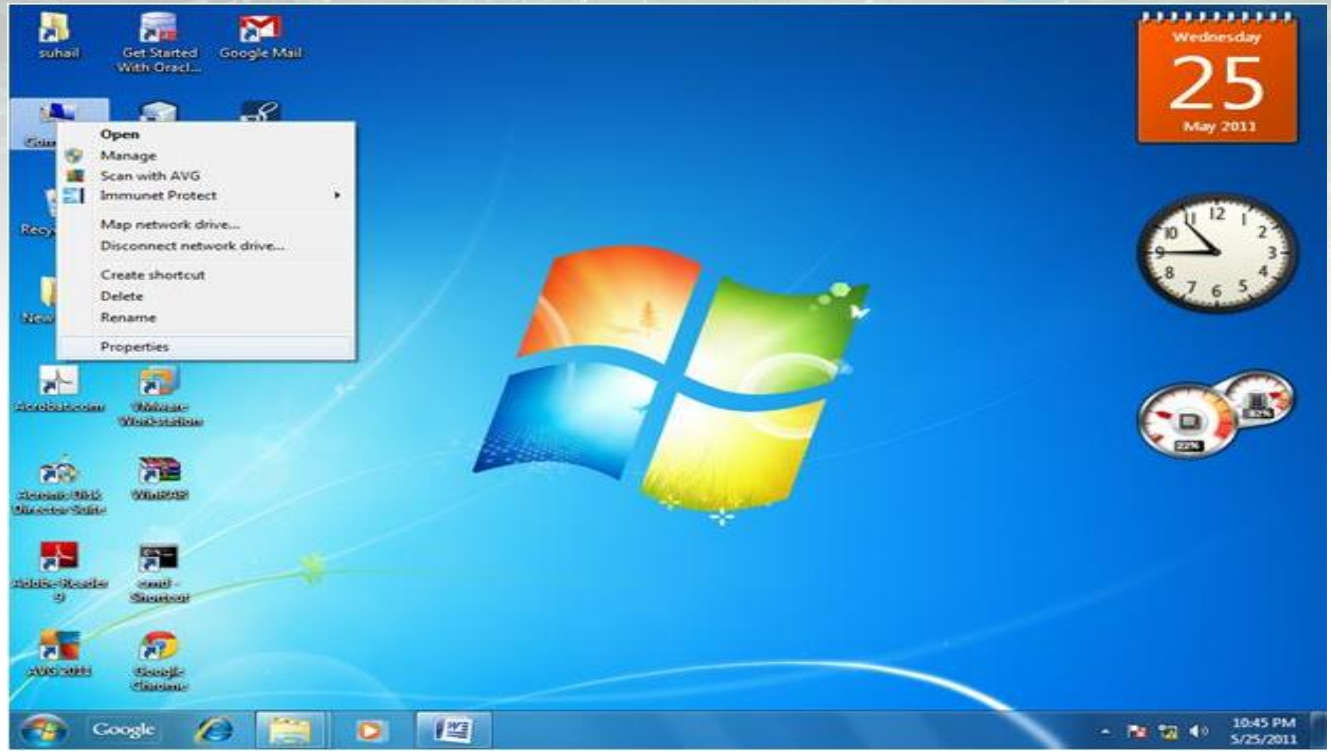


Set Path for Execution

C:\Windows\system32\cmd.exe

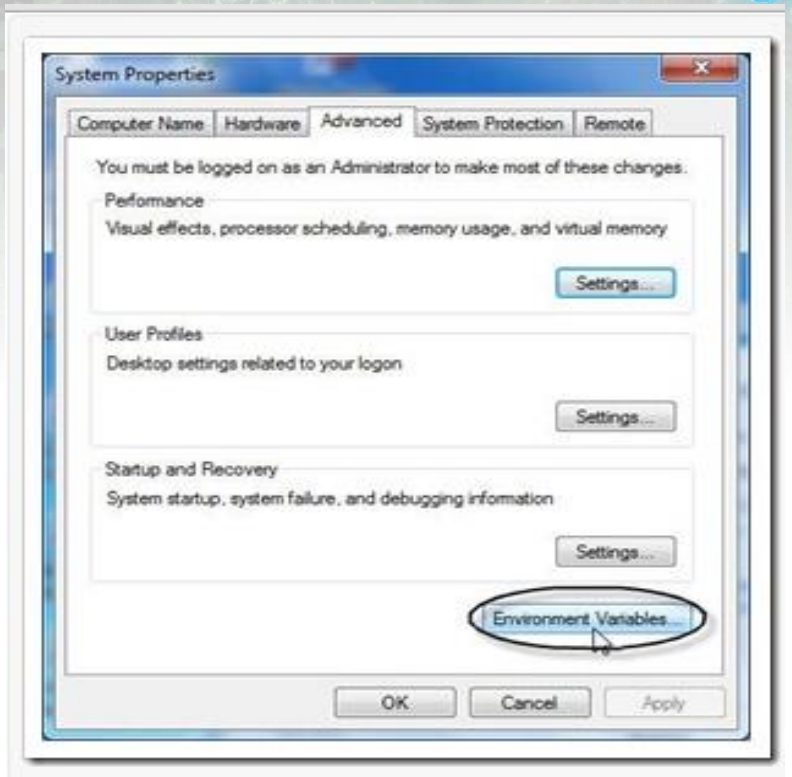
```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sonoo>cd\

C:\>cd new

C:\new>javac Simple.java
'javac' is not recognized as an internal or external command,
operable program or batch file.

C:\new>set path=C:\Program Files\Java\jdk1.6.0_03\bin

C:\new>javac Simple.java

C:\new>java Simple
Hello Java

C:\new>
```

Set Path for Execution

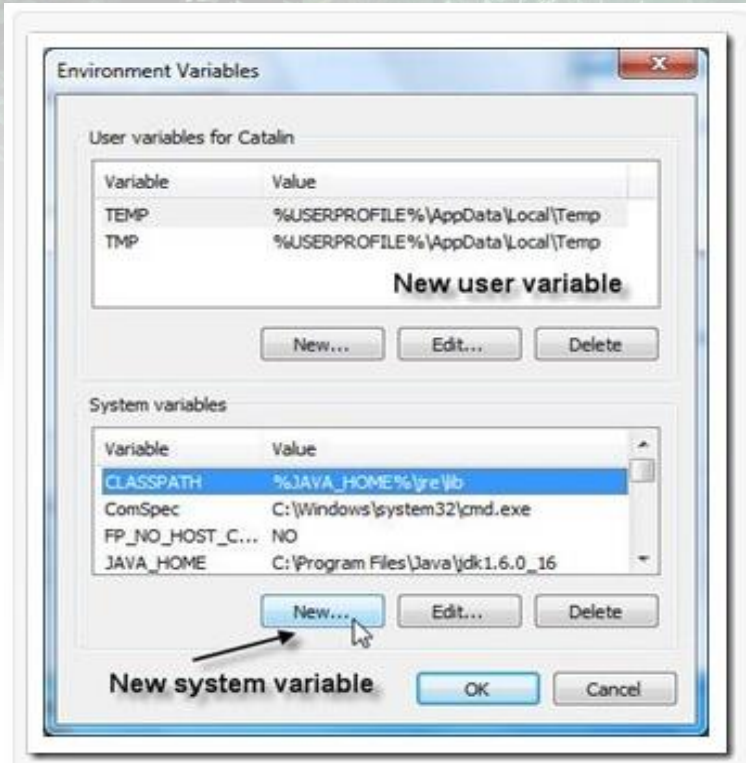


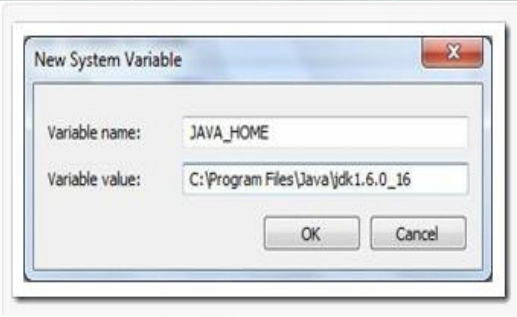
Set Path for Execution

Set Path for Execution

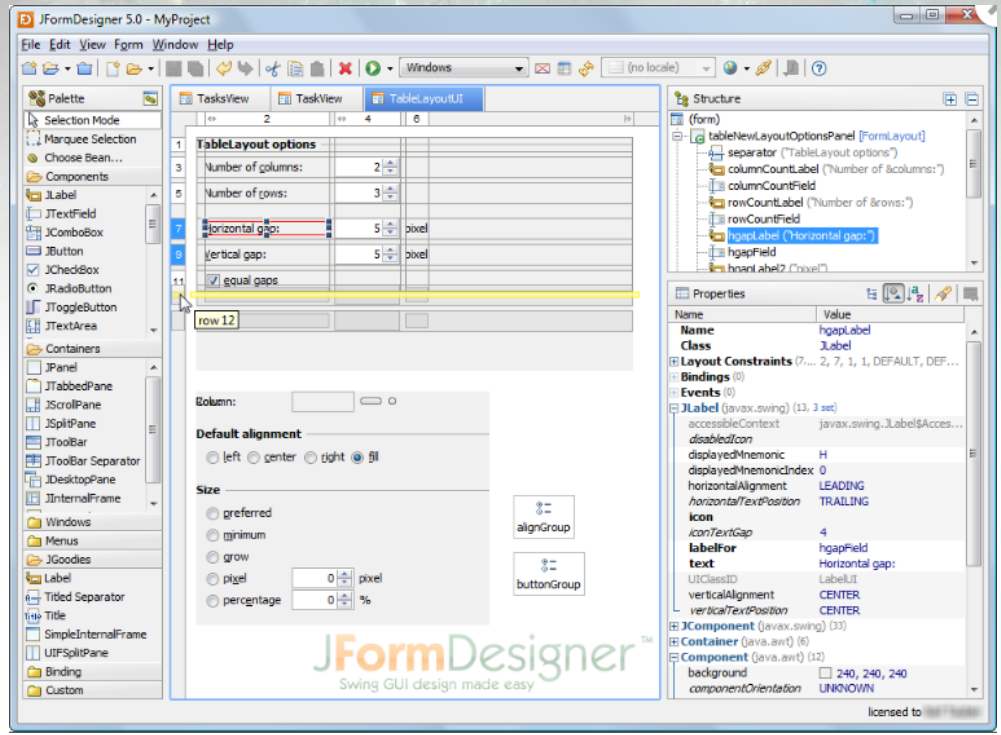


Set Path for Execution

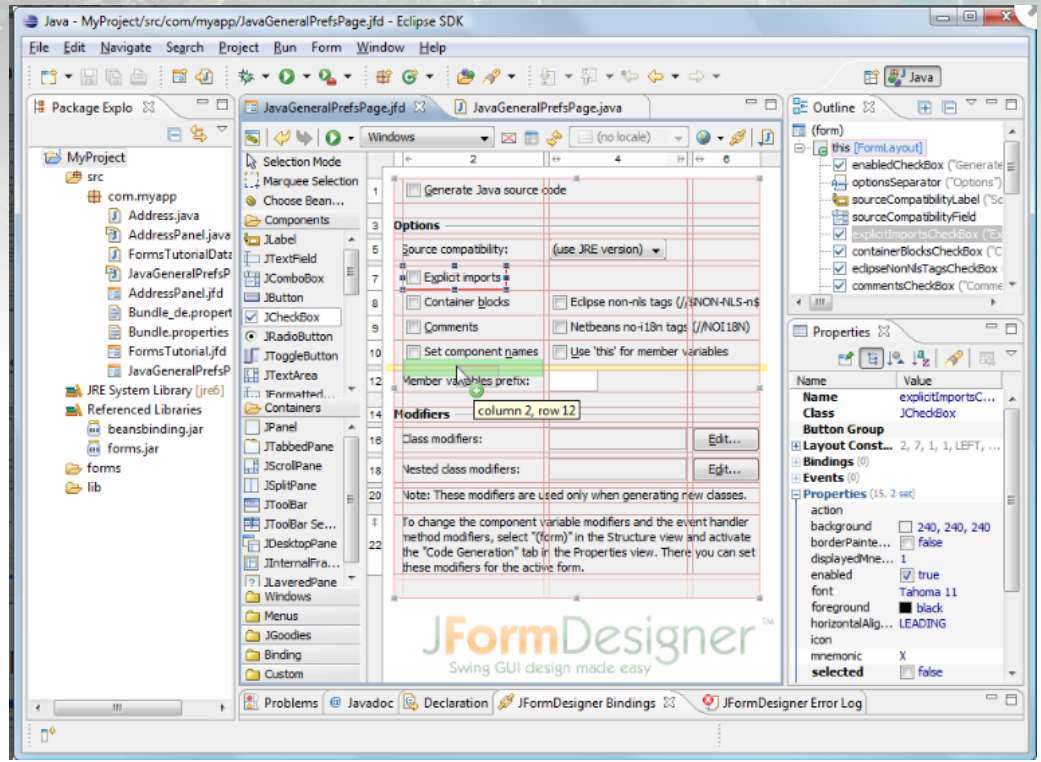




IDE’s – Sample


JFormDesigner 5.0 - MyProject
File Edit View Form Window Help
Windows
(no locale)
Palette
Selection Mode
Marquee Selection
Choose Bean...
Components
JLabel
JTextField
JComboBox
JButton
JCheckBox
JRadioButton
JToggleButton
JTextArea
Containers
JPanel
JTabbedPane
JScrollPane
JSplitPane
JToolBar
JToolBar Separator
JDesktopPane
JInternalFrame
Windows
Menus
JGoodies
Label
Titled Separator
Title
SimpleInternalFrame
UIFSplitPane
Binding
Custom
TasksView
TaskView
TableLayoutUI
TableLayout options
Number of columns:
Number of rows:
Horizontal gap:
Vertical gap:
equal gaps
row 12
Column:
Default alignment
left
center
right
fill
Size
preferred
minimum
grow
pixel
percentage
alignGroup
buttonGroup
JFormDesigner
Swing GUI design made easy
Structure
(form)
tableNewLayoutOptionsPanel [FormLayout]
separator ("TableLayout options")
columnCountLabel ("Number of &columns:")
columnCountField
rowCountLabel ("Number of &rows:")
rowCountField
hgapLabel ("Horizontal gap:")
hgapField
Properties
Name
Value
Name
hgapLabel
Class
JLabel
Layout Constraints
Bindings
Events
JLabel (javax.swing)
accessibleContext
disabledIcon
displayedMnemonic
displayedMnemonicIndex
horizontalAlignment
horizontalTextPosition
icon
iconTextGap
labelFor
text
Horizontal gap:
UIClassID
verticalAlignment
verticalTextPosition
JComponent (javax.swing)
Container (java.awt)
Component (java.awt)
background
componentOrientation
UNKNOWN
licensed to

IDE’s – Sample


Java - MyProject/src/com/myapp/JavaGeneralPrefsPage.jfd - Eclipse SDK
File Edit Navigate Search Project Run Form Window Help
Package Explo
MyProject
src
com.myapp
Address.java
AddressPanel.java
FormsTutorialData
JavaGeneralPrefsP
AddressPanel.jfd
Bundle_de.propert
Bundle.properties
FormsTutorial.jfd
JavaGeneralPrefsP
JRE System Library [jre6]
Referenced Libraries
beansbinding.jar
forms.jar
forms
lib
JavaGeneralPrefsPage.jfd
JavaGeneralPrefsPage.java
Windows
(no locale)
Selection Mode
Marquee Selection
Choose Bean...
Components
JLabel
JTextField
JComboBox
JButton
JCheckBox
JRadioButton
JToggleButton
JTextArea
Containers
JPanel
JTabbedPane
JScrollPane
JSplitPane
JToolBar
JToolBar Se...
JDesktopPane
JInternalFra...
JLayeredPane
Windows
Menus
JGoodies
Binding
Custom
Generate Java source code
Options
Source compatibility:
(use JRE version)
Explicit imports
Container blocks
Eclipse non-nls tags
Comments
Netbeans no-i18n tags
Set component names
Use 'this' for member variables
Member variables prefix:
column 2, row 12
Modifiers
Class modifiers:
Edit...
Nested class modifiers:
Edit...
Note: These modifiers are used only when generating new classes.
To change the component variable modifiers and the event handler method modifiers, select "(form)" in the Structure view and activate the "Code Generation" tab in the Properties view. There you can set these modifiers for the active form.
JFormDesigner
Swing GUI design made easy
Outline
(form)
this [FormLayout]
enabledCheckBox
optionsSeparator
sourceCompatibilityLabel
sourceCompatibilityField
explicitImportsCheckBox
containerBlocksCheckBox
eclipseNonNlsTagsCheckBox
commentsCheckBox
Properties
Name
Value
Name
Class
JCheckBox
Button Group
Layout Const...
Bindings
Events
Properties
action
background
240, 240, 240
borderPainte...
false
displayedMne...
1
enabled
true
font
Tahoma 11
foreground
black
horizontalAlig...
LEADING
icon
mnemonic
X
selected
false
Problems
Javadoc
Declaration
JFormDesigner Bindings
JFormDesigner Error Log

IDE's – Sample

Resource - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Navigator

- Example
  - bin
  - src
    - com
      - example
        - hw
          - HelloWorld
  - .classpath
  - .project

HelloWorld.java

```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] ar
        System.out.println("Hello world")
    }
}
```

Outline

- com.example.hw
- HelloWorld
  - main(String[]) : v

Properties

Property	Value
Info	
editable	true
last modified	8/28/02 9:10 PM
name	HelloWorld.java
path	/Example/src/com
size	144

Tasks (0 items)

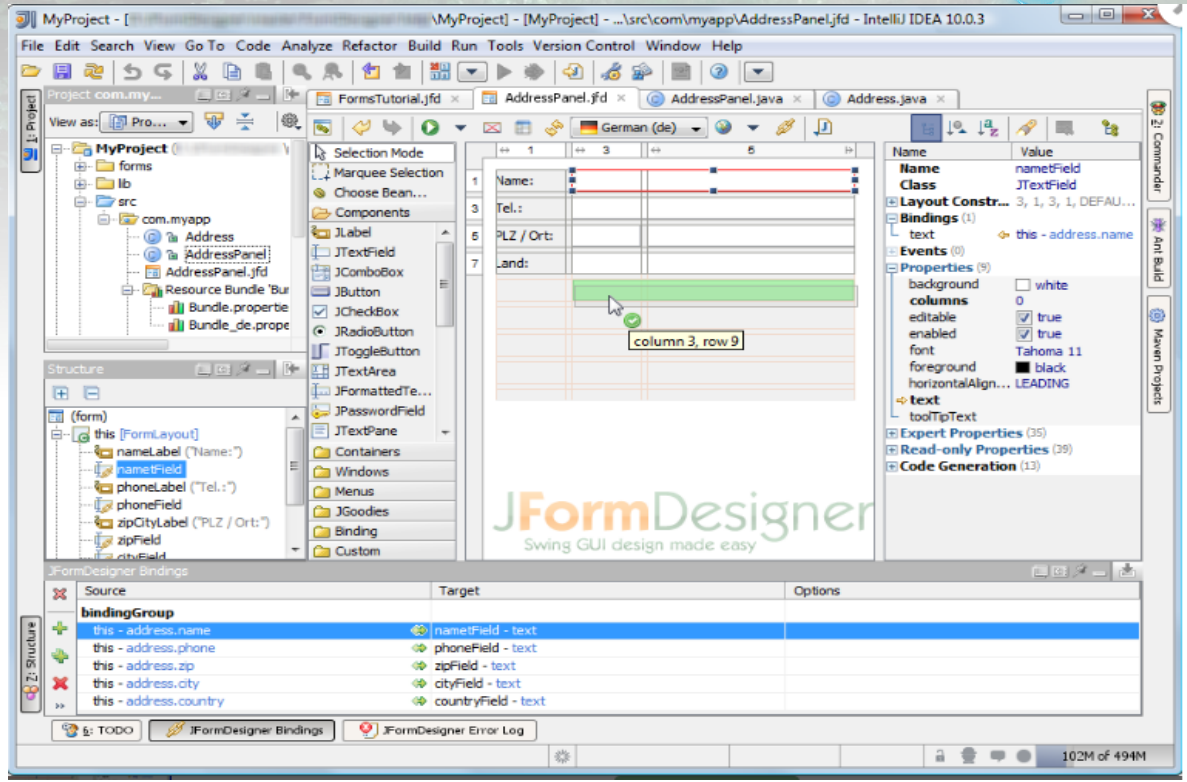
C	!	Description	Resource	In Folder

Tasks Search

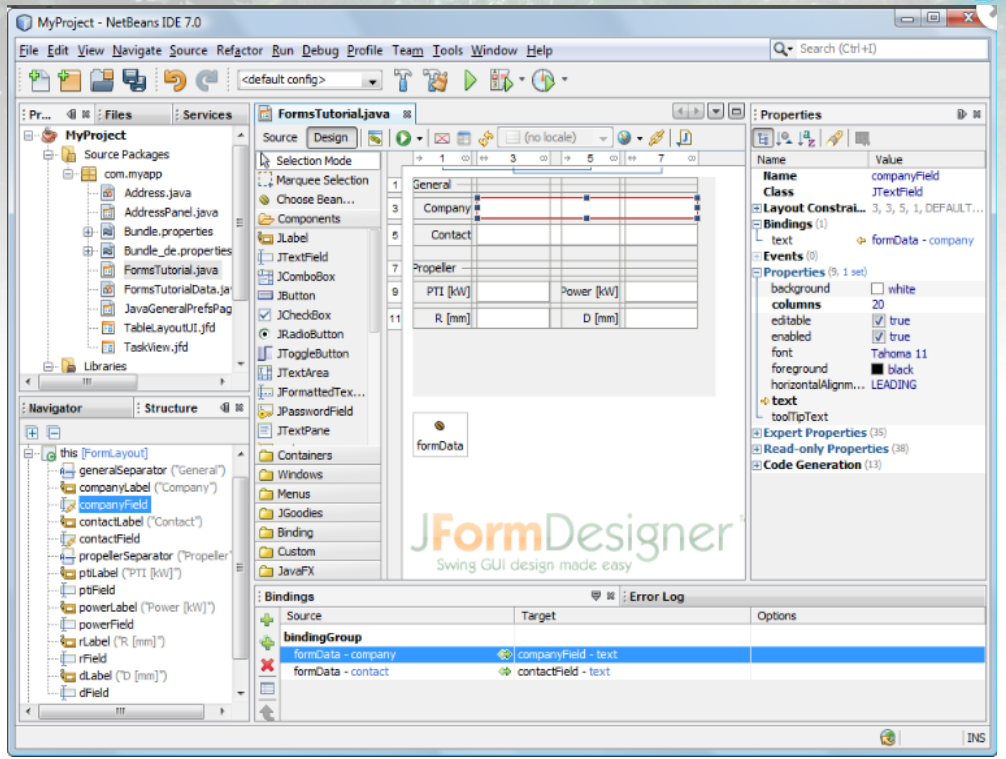
Bookmarks

Writable Insert 6 : 16

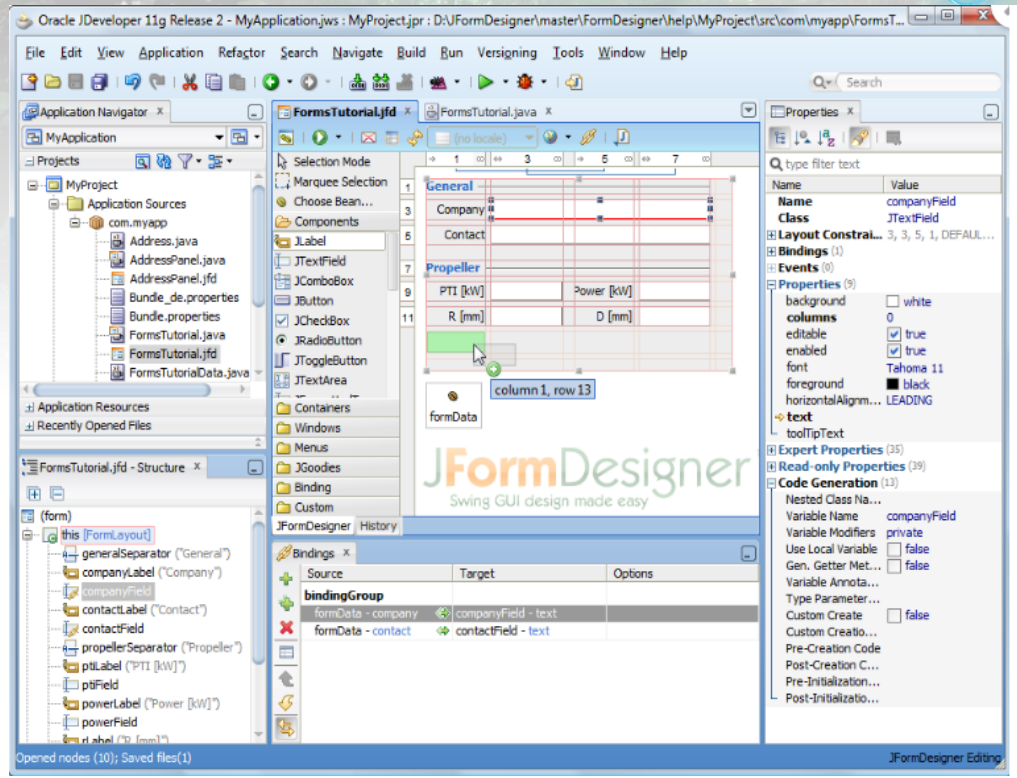
IDE’s – Sample


MyProject - [\MyProject] - [MyProject] - ...\src\com\myapp\AddressPanel.jfd - IntelliJ IDEA 10.0.3
File Edit Search View Go To Code Analyze Refactor Build Run Tools Version Control Window Help
FormsTutorial.jfd
AddressPanel.jfd
AddressPanel.java
Address.java
MyProject
forms
lib
src
com.myapp
Address
AddressPanel
AddressPanel.jfd
Resource Bundle 'Bur
Bundle.propertie
Bundle_de.prope
Selection Mode
Marquee Selection
Choose Bean...
Components
JLabel
JTextField
JComboBox
JButton
JCheckBox
JRadioButton
JToggleButton
JTextArea
JFormattedTe...
JPasswordField
JTextPane
Containers
Windows
Menus
JGoodies
Binding
Custom
German (de)
Name:
Tel.:
PLZ / Ort:
Land:
column 3, row 9
JFormDesigner
Swing GUI design made easy
Name
Value
Name
nametField
Class
JTextField
Layout Constr...
Bindings
text
this - address.name
Events
Properties
background
white
columns
0
editable
true
enabled
true
font
Tahoma 11
foreground
black
horizontalAlign...
LEADING
text
toolTipText
Expert Properties
Read-only Properties
Code Generation
Structure
(form)
this [FormLayout]
nameLabel ("Name:")
nametField
phoneLabel ("Tel.:")
phoneField
zipCityLabel ("PLZ / Ort:")
zipField
JFormDesigner Bindings
Source
Target
Options
bindingGroup
this - address.name
nametField - text
this - address.phone
phoneField - text
this - address.zip
zipField - text
this - address.city
cityField - text
this - address.country
countryField - text
6: TODO
JFormDesigner Bindings
JFormDesigner Error Log
102M of 494M

IDE’s – Sample


MyProject - NetBeans IDE 7.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Pr... Files Services
MyProject
Source Packages
com.myapp
Address.java
AddressPanel.java
Bundle.properties
Bundle_de.properties
FormsTutorial.java
FormsTutorialData.ja
JavaGeneralPrefsPag
TableLayoutUI.jfd
TaskView.jfd
Libraries
Navigator Structure
this [FormLayout]
generalSeparator ("General")
companyLabel ("Company")
companyField
contactLabel ("Contact")
contactField
propellerSeparator ("Propeller")
ptiLabel ("PTI [kW]")
ptiField
powerLabel ("Power [kW]")
powerField
rLabel ("R [mm]")
rField
dLabel ("D [mm]")
dField
FormsTutorial.java
Source Design
(no locale)
Selection Mode
Marquee Selection
Choose Bean...
Components
JLabel
JTextField
JComboBox
JButton
JCheckBox
JRadioButton
JToggleButton
JTextArea
JFormattedTex...
JPasswordField
JTextPane
Containers
Windows
Menus
JGoodies
Binding
Custom
JavaFX
General
Company
Contact
Propeller
PTI [kW]
Power [kW]
R [mm]
D [mm]
formData
JFormDesigner
Swing GUI design made easy
Properties
Name Value
Name companyField
Class JTextField
Layout Constrai... 3, 3, 5, 1, DEFAULT...
Bindings (1)
text formData - company
Events (0)
Properties (9, 1 set)
background white
columns 20
editable true
enabled true
font Tahoma 11
foreground black
horizontalAlignm... LEADING
text
toolTipText
Expert Properties (35)
Read-only Properties (38)
Code Generation (13)
Bindings Error Log
Source Target Options
bindingGroup
formData - company companyField - text
formData - contact contactField - text
INS

IDE’s – Sample


Oracle JDeveloper 11g Release 2 - MyApplication.jws : MyProject.jpr : D:\JFormDesigner\master\FormDesigner\help\MyProject\src\com\myapp\FormsT...
File Edit View Application Refactor Search Navigate Build Run Versioning Tools Window Help
Application Navigator
MyApplication
Projects
MyProject
Application Sources
com.myapp
Address.java
AddressPanel.java
AddressPanel.jfd
Bundle_de.properties
Bundle.properties
FormsTutorial.java
FormsTutorial.jfd
FormsTutorialData.java
Application Resources
Recently Opened Files
FormsTutorial.jfd - Structure
(form)
this [FormLayout]
generalSeparator ("General")
companyLabel ("Company")
companyField
contactLabel ("Contact")
contactField
propellerSeparator ("Propeller")
ptiLabel ("PTI [kW]")
ptiField
powerLabel ("Power [kW]")
powerField
FormsTutorial.jfd
FormsTutorial.java
Selection Mode
Marquee Selection
Choose Bean...
Components
JLabel
JTextField
JComboBox
JButton
JCheckBox
JRadioButton
JToggleButton
JTextArea
Containers
Windows
Menus
JGoodies
Binding
Custom
JFormDesigner
History
General
Company
Contact
Propeller
PTI [kW]
Power [kW]
R [mm]
D [mm]
column 1, row 13
formData
JFormDesigner
Swing GUI design made easy
Bindings
Source
Target
Options
bindingGroup
formData - company
companyField - text
formData - contact
contactField - text
Properties
type filter text
Name
Value
Name
companyField
Class
JTextField
Layout Constrai...
3, 3, 5, 1, DEFAUL...
Bindings (1)
Events (0)
Properties (9)
background
white
columns
0
editable
true
enabled
true
font
Tahoma 11
foreground
black
horizontalAlignm...
LEADING
text
toolTipText
Expert Properties (35)
Read-only Properties (39)
Code Generation (13)
Nested Class Na...
Variable Name
companyField
Variable Modifiers
private
Use Local Variable
false
Gen. Getter Met...
false
Variable Annota...
Type Parameter...
Custom Create
false
Custom Creatio...
Pre-Creation Code
Post-Creation C...
Pre-Initialization...
Post-Initializatio...
Opened nodes (10); Saved files(1)
JFormDesigner Editing

Some Sample Programs

```
class Var
{
      static public void main(String a[])
     {
            System.out.println("Name="+"Lakshan");
            System.out.println("Roll No="+22);
            System.out.println("Salary="+50000);
     }
}
```

Some Sample Programs

```
Class VarUsage
{
    public static void main(String k[])
    {
        String EmpName=”Dhanush.A”;
        int EmpId=555;
        float EmpSal=250000;

        System.out.println(“Name=”+EmpName);
        System.out.println(“ID=”+EmpId);
        System.out.println(“Salary=”+EmpSal);

    }
}
```

Java – Important Terminologies

JVM (Java Virtual Machine)

- JVM is also responsible for loads the code, verify the code, execute the code, and provides the runtime environment for run Java programs

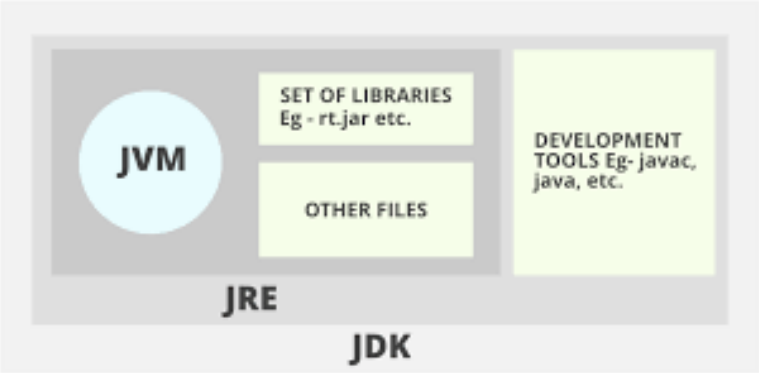
JRE (Java Runtime Environment)

- The JRE (Java Runtime Environment) provides the environment to run Java programs. JRE is used by the end user of your system. JRE consists of a set of libraries and other files that JVM uses at runtime

Java – Important Terminologies

JDK (Java Development Kit)

- The JDK (Java Development Kit) is a software environment which is used to develop and run Java applications.
- JDK contains two things
  - **Development tools:** Which is used to provide an environment to develop Java applications.
  - **JRE:** Which is used to execute Java applications.



Stream

- A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow
- In Java, 3 streams are created for us automatically. All these streams are attached with the console

System.out: standard output stream

System.in: standard input stream

System.err: standard error stream

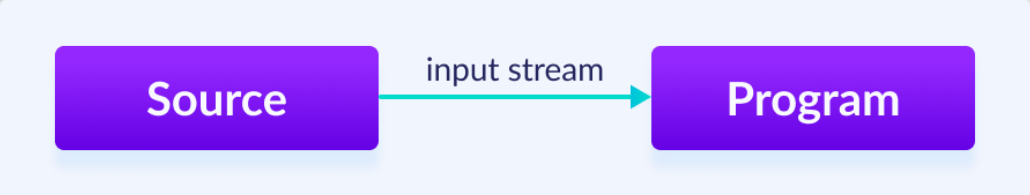
Input and Output Processing

- Java I/O (Input and Output) is used to process the input and produce the output.
- Java uses the concept of a stream to make I/O operation fast.
- The **java.io** package contains all the classes required for input and output operations
- There are various types

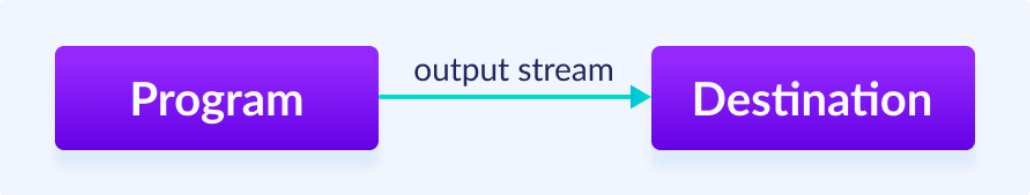
Scanner	
BufferedReader	BufferedWriter
DataInputStream	DataOututStream
FileInputStream	FileOutputStream

Input and Output Processing

Reading data from source



Writing data to destination



Scanner

- It is a predefined class.
- **import java.util.*;** (It must be declared before the class).
- Scanner is used to get or accept the input from the user.
- To use the scanner we must import **java.util** package because the scanner class is available on the particular util package.

Syntax:

```
Scanner obj=new Scanner(System.in);
```

Some built-in methods in scanner:

(next)Next=() It is a method for reading a string. It is a built-in method.
nextInt()-> Integer conversion
nextFloat()-> Float conversion
nextDouble()-> Double conversion

Scanner

```
import java.util.*;
Class Ex
 {
        public static void main(String k[])
        {
                Int x;
                String name;

                Scanner sr=new Scanner(System.in);
                System.out.println("Enter the x value:");
                x=sr.nextInt();
                System.out.println("Enter your name:");
                name=sr.next();
                System.out.println("Name::"+name);
                System.out.println("X value:"+x);
        }
 }
```

Output:

```
Enter the x value:
45
Enter your name:
Sayan
Name: Sayan
X value:45
```

Buffered Reader

- It is a predefined class.
- Used to read or get information from the user.
- It always associated with the input stream reader class.
- In order to use this class user has to import **java.io** package because that buffered reader class will be available(support)under io package.
- Also user has to use **IOException** or **Exception** to handling the inputs and outputs.
- Here the system.in is also used. It is a buffer.
- readLine() method used here for reading the user inputs or processing results.
- To use string data user can alone use readLine itself because of string data.

Buffered Reader

- To use integer,float,double,long the user have to do following:

Integer.parseInt(bufferedObject.readLine());

Float.parseFloat(bufferedObject.readLine());

Long.parseLong(bufferedObject.readLine());

Double.parseDouble(bufferedObject.readLine());

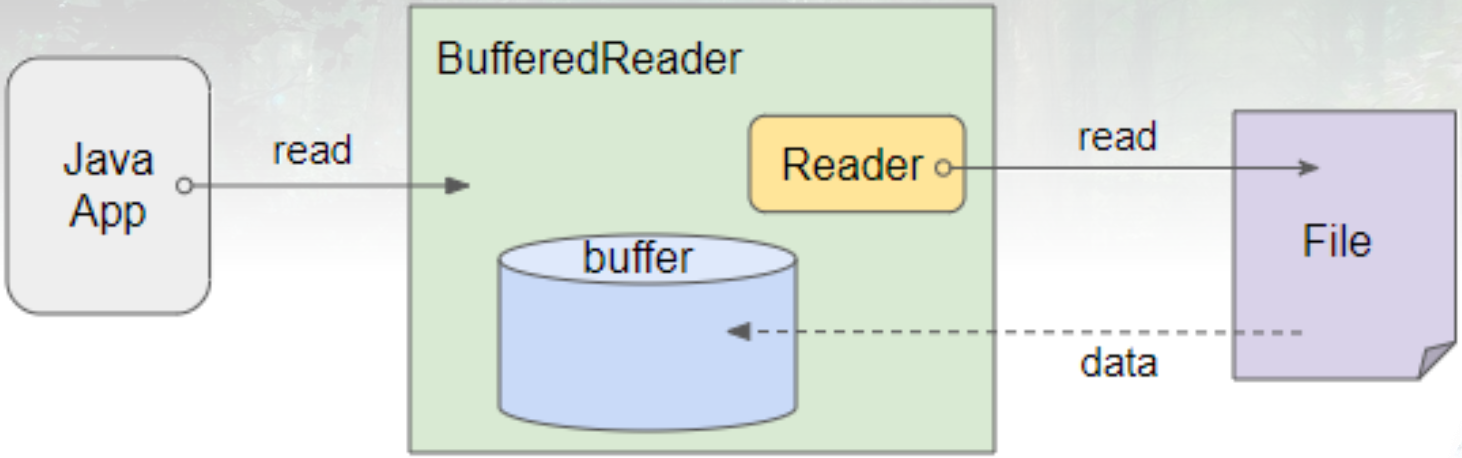
Syntax:

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

Note:

In above syntax the user can change only one thing that is br. Because br is an user defined object of the BufferedReader class, remaining things should not get changed.

Buffered Reader



Buffered Reader

```
import java.io.*;
class ExampleBR
{
        public static void main(String a[]) throws IOException
        {
                int a, b;

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                a = Integer.parseInt(br. readLine());
                b = Integer.parseInt(br. readLine());

                System.out.println("Addition:" + (a+b));
        }
}
```

Data Input Stream

- It is a predefined class.
- Used to read or get information from the user.
- In order to use this class user has to import java.io package because that DataInputStream class will be available(support)under io package.
- Also user has to use IOException or Exception to handling the inputs and outputs.
- Here the system.in is also used. It is a buffer.
- readLine() method used here for reading the user inputs or processing results.

```
Integer.parseInt(DataInputStream Object.readLine());
Float.parseFloat(DataInputStream Object.readLine());
Long.parseLong(DataInputStream Object.readLine());
Double.parseDouble(DataInputStream Object.readLine());
```

Data Input Stream

```
import java.io.*;
class DIS
{
        public static void main(String arg[])throws IOException
        {
                int a,b;
                DataInputStream br=new DataInputStream(System.in);
                System.out.println("Enter A & B:");
                a=Integer.parseInt(br.readLine());
                b=Integer.parseInt(br.readLine());
                System.out.println("Addition:"+(a+b));
        }
}
```

Data Input Stream

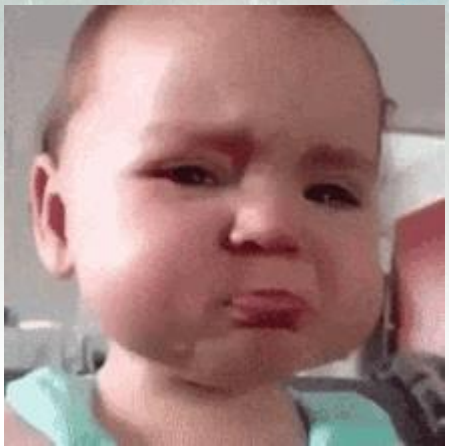
```
import java.io.*;
public class addition
{
  public static void main(String args[]) throws IOException
  {
        float a,b,c;
        DataInputStream input=new DataInputStream(System.in);
        System.out.println("enter two numbers for addition");
        a=Float.parseFloat(input.readLine());
        b=Float.parseFloat(input.readLine());
        c=a+b;
        System.out.println("the total is "+c);
  }
}
```

Scanner - Exercises

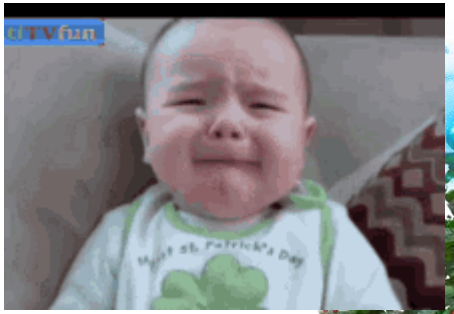
$$t_n = a + (n-1)d$$

$$S_n = \frac{n}{2}[2a + (n-1)d]$$

$$S_n = \frac{n}{2}[a + l]$$



$$S_n = \frac{n}{2}(a_1 + a_n)$$



Algebraic Questions

$$16 - 2t = 5t + 9$$

(1.) $17x + 12 = 54 - 4x.$

(2.) $6x - 16 = 5x - 9.$

(3.) $4x - 4 = 3x + 2.$

(4.) $13x - 25 = 9x - 13.$

(5.) $9x + 2 = 7x + 15.$

(6.) $3x + 39 = 2x + 42.$

(7.) $5x - 3x = 36 - x.$

(8.) $7x - 12 + 5 = 8x - 24.$

(9.) $30 - x + 12 = 3x + 2.$

(10.) $6x - 30 = 60 - 8x - 20.$

(11.) $9x - 16 = 3x - (x - 5).$

(12.) $5x - 10 + 2x = 6(x + 1) - x.$

(13.) $2(5x + 1) = x + 2.$

(14.) $5(x + 1) = 2(x - 5) + 6.$

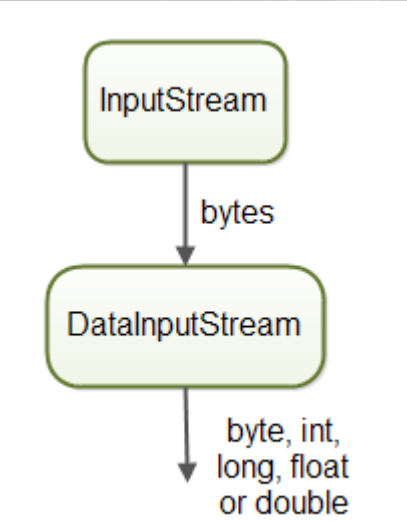
(15.) $3(x - 2) - 2(x - 3) - 5 = 0.$

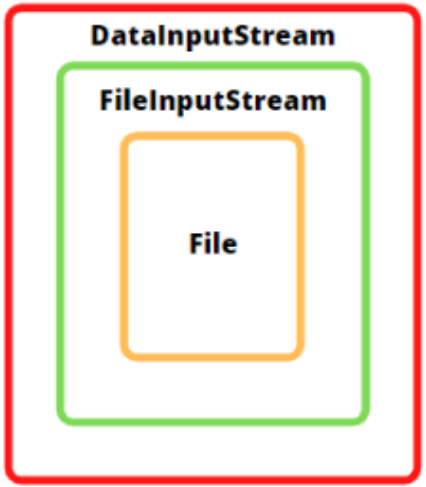
(16.) $(3 - 2x) + 3(1 - 4x) - 4(1 - 5x) = 2(1 - 3x.)$

Can You Solve This?

$$6 \div 2(1+2) =$$

DataInputStream





A File wrapped with a FileInputStream is wrapped with a DataInputStream

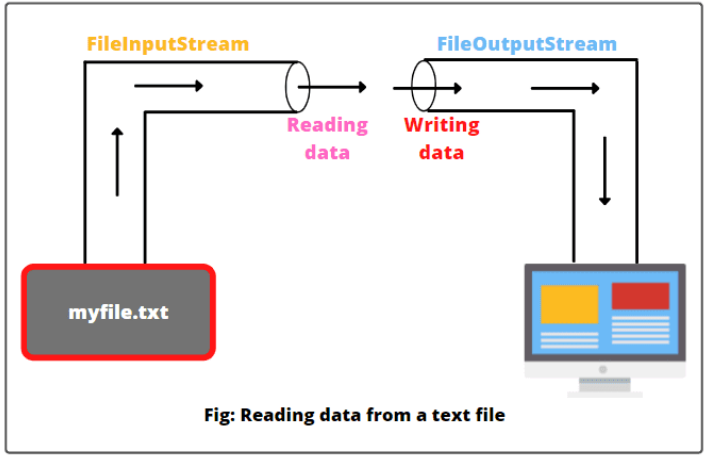


Fig: Reading data from a text file

Scanner vs. BufferedReader

Sr. No.	Key	Scanner Class	BufferReader Class
1	Synchronous	Scanner is not syncronous in nature and should be used only in single threaded case.	BufferReader is syncronous in nature. During multithreading environment, BufferReader should be used.
2	Buffer Memory	Scanner has little buffer of 1 KB char buffer.	BufferReader has large buffer of 8KB byte Buffer as compared to Scanner.
3	Processing Speed	Scanner is bit slower as it need to parse data as well.	BufferReader is faster than Scanner as it only reads a character stream.
4	Methods	Scanner has methods like nextInt(), nextShort() etc.	BufferReader has methods like parseInt(), parseShort() etc.
5	Read Line	Scanner has method nextLine() to read a line.	BufferReader has method readLine() to read a line.

Data Input Stream

- **DataInputStream** in Java is a filter input stream that provides methods for reading Java's standard data types.
- It enables user **conveniently to read strings** and all primitive data types such as int, float, long, double, etc from a stream.
- Java DataInputStream reads bytes from an underlying stream and converts them into suitable primitive-type values or strings.

Stretch Yourself

Diffie Hellman Algorithm

Diffie-Hellman Key Exchange

Alice

Bob

Bob and Alice know and have the following :
p = 23 (a prime number) g = 11 (a generator)

Alice chooses a secret random number a = 6	Bob chooses a secret random number b = 5
Alice computes : $A = g^a \bmod p$ $A = 11^6 \bmod 23 = 9$	Bob computes : $B = g^b \bmod p$ $B = 11^5 \bmod 23 = 5$
Alice receives B = 5 from Bob	**Bob receives A = 9 from Alice**
Secret Key = $K = B^a \bmod p$	Secret Key = $K = A^b \bmod p$
$K = 5^6 \bmod 23 = \boxed{8}$	$K = 9^5 \bmod 23 = \boxed{8}$

The common secret key is : 8

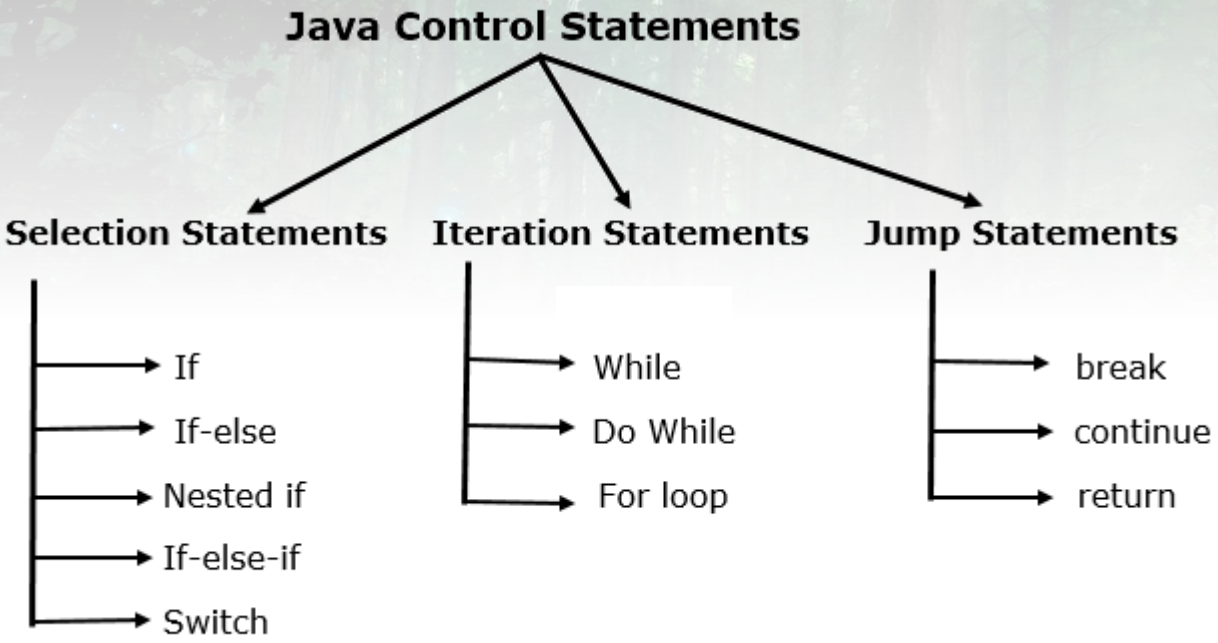
N.B. We could also have written : $K = g^{ab} \bmod$

©2007 Mat-D.com

Control Statements in Java

- A programming language uses control statements to control the flow of execution of a program.
- In Java programming, user can control the flow of execution of a program based on some conditions.
- Java control statements can be put into the following three categories: **Selection, Iteration, and Jump**.
- The selection statements allow user program to choose a different path of execution based on a certain condition.
- Iteration statements allow program execution to repeat one or more statements.
- Jump statements transfer the control to other parts of the program.

Control Statements in Java

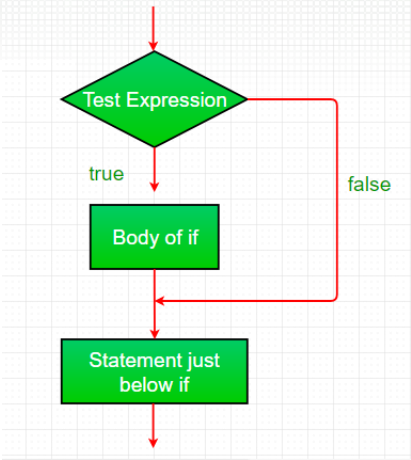


Selection – If Statement

Definition

- If statement in Java is a simple decision-making statement. Java if statement is used to decide whether a certain statement or block of the statement will be executed or not. If a certain condition is true, then the block of statement is executed, otherwise not.
- **Syntax**

```
if(condition)
{
        //statement to be execute
        //if condition is true
}
```



Selection – If Statement

Condition is true

```
int number = 10;

if (number > 0) {
  // code
}

// code after if
```

Condition is false

```
int number = 10;

if (number < 0) {
  // code
}

// code after if
```

Selection – If Statement

Example

```
class If_Statement
 {
        public static void main(String[] args)
        {
                int number = 10;
                                // checks if number is less than 0
               if (number < 0)
                {
                        System.out.println("The number is negative.");
                 }
}
```

Output

The number is negative

Selection – If Statement

```
class If_Demo
{
        public static void main(String[] args)
        {
                String language = "Java";
                 if (language == "Java")
                 {
                      System.out.println("Best Programming Language");
                 }
        }
}
```

Output

Best Programming Language

Selection – If else Statements

Definition

- Java if statement is used to decide whether a certain statement or block of the statement will be executed or not. If a certain condition is true, then the block of statement is executed, otherwise (false) else part will be executed.
- **Syntax**

```
if(condition)
{
        //statement to be execute
        //if condition is true
}
else
{
       //statement of false part
}
```

Selection – If else Statements

Condition is true

```
int number = 5;

if (number > 0) {
  // code
}
else {
  // code
}

// code after if...else
```

Condition is false

```
int number = 5;

if (number < 0) {
  // code
}
else {
  // code
}

// code after if...else
```

Selection – If else Statements

```
class IFELSE
{
        public static void main(String[] args)
        {
                int number = 10;
                if (number > 0) {
                        System.out.println("The number is positive.");
                }
                else {
                        System.out.println("The number is not positive.");
                }
        }
}
```

Output

The number is positive.

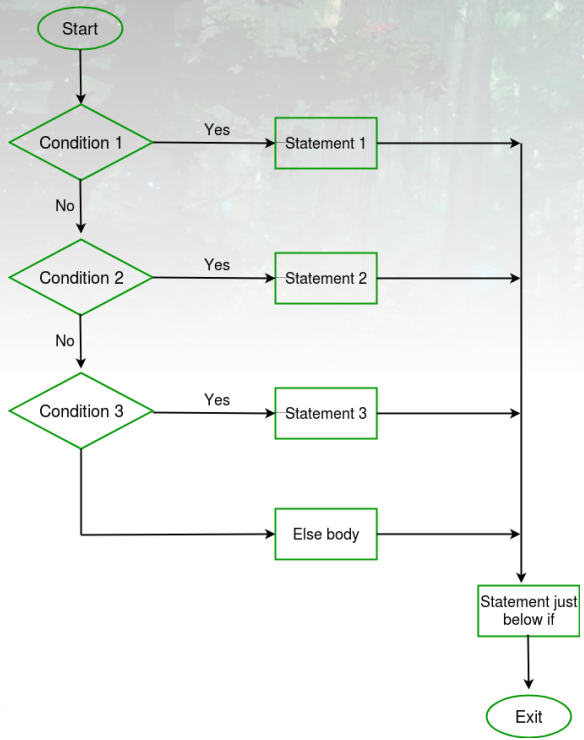
Selection – Else if Statements

Definition

- Java if-else-if ladder is used to work on multiple conditions. The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed
- **Syntax**

```
if(condition)
{
        //statement to be execute
}
else if (condition)
{
       //statement of false part
}
else {   //all if conditions met false
  }
```

Selection – Else if Statements




```
if(condition1){
//statement1;
}
else if(condition2){
//statement2;
}
else if(condition3){
//statement3;
}
.........
.........
.........
else{
//default statement;
}
```

Selection – Else if Statements

1st Condition is true

```
int number = 2;
if (number > 0) {
  // code
}
else if (number == 0){
  // code
}
else {
  //code
}

//code after if
```

2nd Condition is true

```
int number = 0;
if (number > 0) {
  // code
}
else if (number == 0){
  // code
}
else {
  //code
}

//code after if
```

All Conditions are false

```
int number = -2;
if (number > 0) {
  // code
}
else if (number == 0){
  // code
}
else {
  //code
}

//code after if
```

Selection – Else if Statements

```
class ELSEIF
{
    public static void main(String[] args)
    {
        int number = 0;
        if (number > 0) {
            System.out.println("The number is positive.");
        }
        else if (number < 0) {
            System.out.println("The number is negative.");
        }
        else {
            System.out.println("The number is 0.");
        }
    }
}
```

Output

The number is 0.

Selection – Nested if Statements

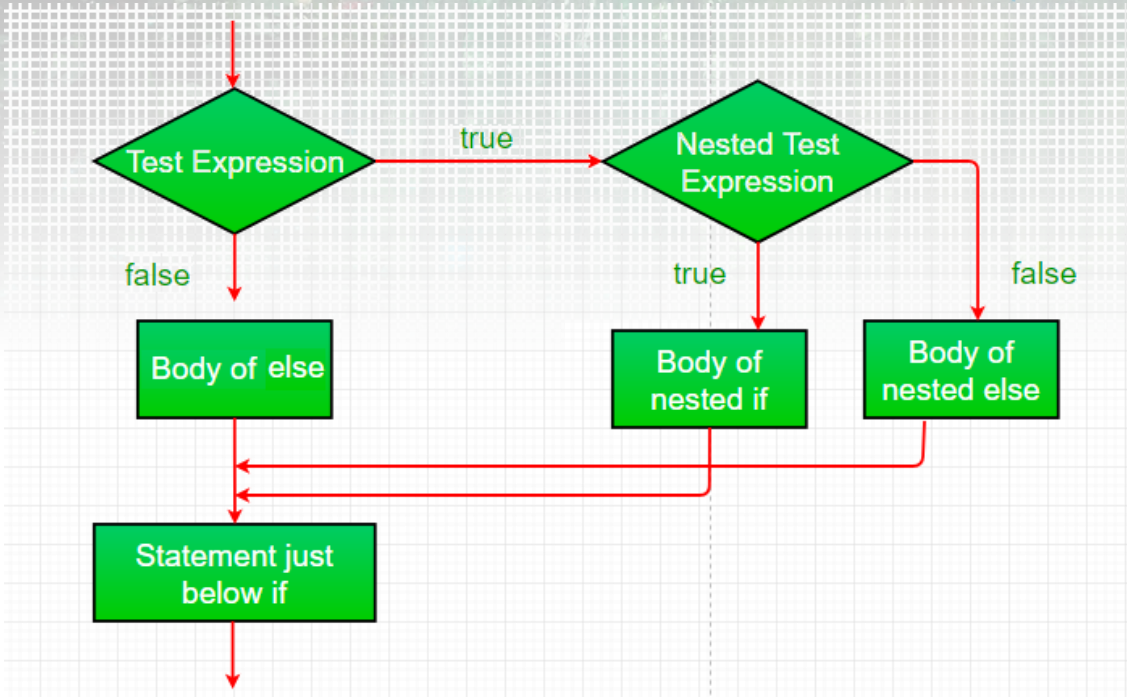
Definition

- Java nested-if statement means an if statement inside another if statement.
- A nested-if is an if statement that is the target of another if or else statement.
- In nested-if, the inner if block condition executes only when outer if block condition is true

Syntax

```
if(Outer_Condition)
{
    if(Inner_Condition) {
        // Executes Inner IF first then look for outer
    }
        // Executes Outer IF
 }
```

Selection – Nested if Statements



Selection – Nested if Statements

```
Class NestIF
{
    public static void main(String args[])
    {
        int x = 30;
        int y = 10;
        if(x == 30)
        {
            if(y == 10)
            {
                System.out.println("X = 30 and Y = 10");
            }
        }
    }
}
```

Output

X = 30 and Y = 10

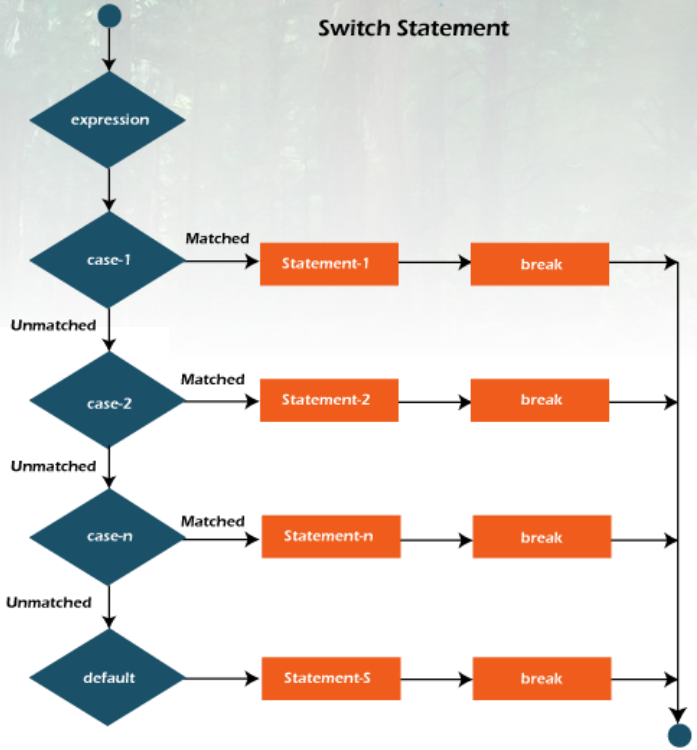
Selection – Switch Statements

Switch

- It is called multiway decision making statement
- The expression is evaluated once and compared with the values of each case.
- If expression matches with value1, the code of case value1 are executed. Similarly, the code of case value2 is executed if expression matches with value2.
- If there is no match, the code of the default case is executed.

Selection – Switch Statements

```
switch (expression)
{
   case value1:
                  // code
                    break;
   case value2:
                 // code
                    break;
     ...
     ...
   default:
                // default statements
 }
```



Selection – Switch Statements

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

    int number = 44;
    String size;

   switch (number) {
        case 29:
              size = "Small";
              break;
        case 42:
              size = "Medium";
              break;
         case 44:
              size = "Large";
              break;
        case 48:
              size = "Extra Large";
              break;
         default:
              size = "Unknown";
              break;
         }
           System.out.println("Size: " + size);
        }
    }
```

Output:
Size: Large

Iteration Statements

- Java Iteration statements allow program execution to repeat one or more statements.
- Java Iteration statements provide different type of statements such as:

1. While loop

2. Do While loop

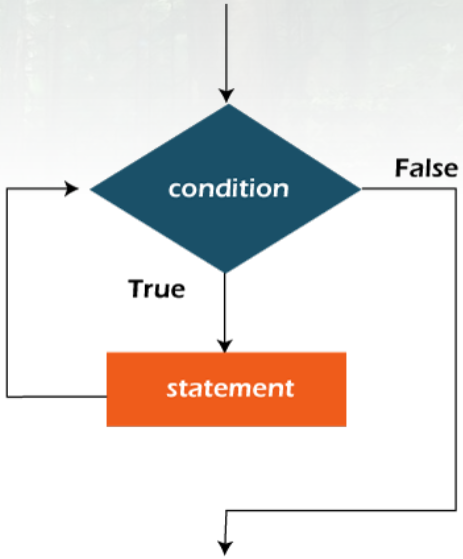
3. For loop

Iteration – While Loop

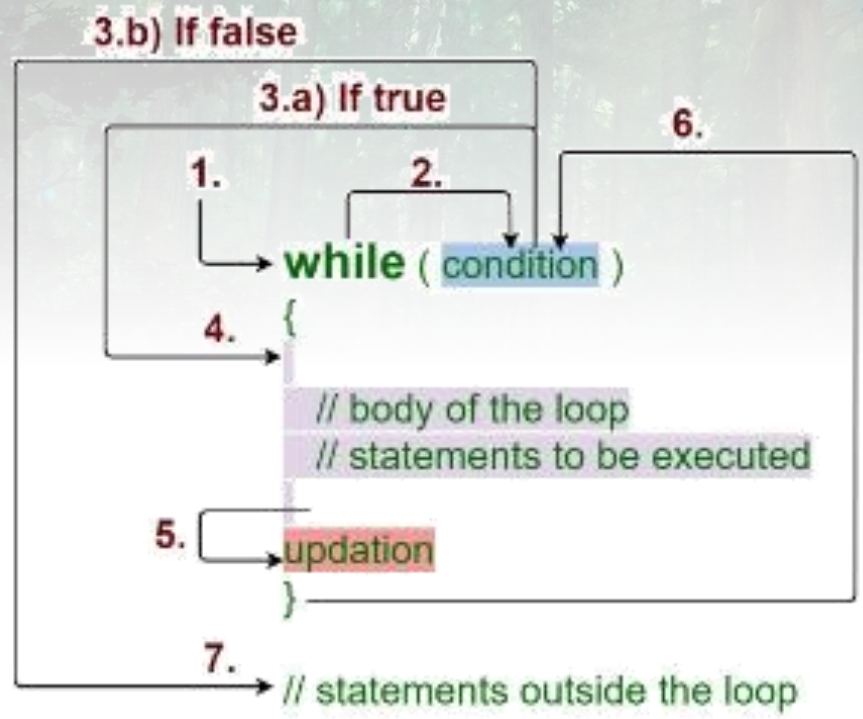
- The while loop in java is the most fundamental looping statement.
- Java while loop repeats a statement or a block while a particular condition is true.

Syntax

```
while(boolean condition)
{
        //body of the loop
}
```



Iteration – While Loop



Iteration – While Loop

```
class WhileLoop
{
        public static void main(String args[])
        {
                System.out.println("The values from 1 to 10 is: ");
                int num = 1;
                while(num <= 10)
                {
                        System.out.println(num);
                        num++;
                }
        }
}
```

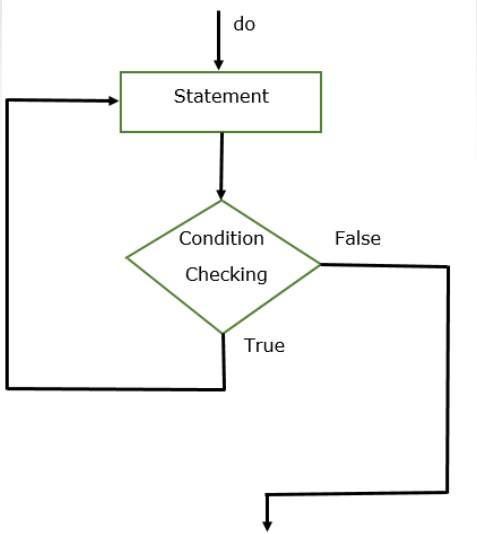
```
E:\>javac WhileLoop.java

E:\>java WhileLoop
The values from 1 to 10 is:
1
2
3
4
5
6
7
8
9
10
```

Iteration – Do While Loop

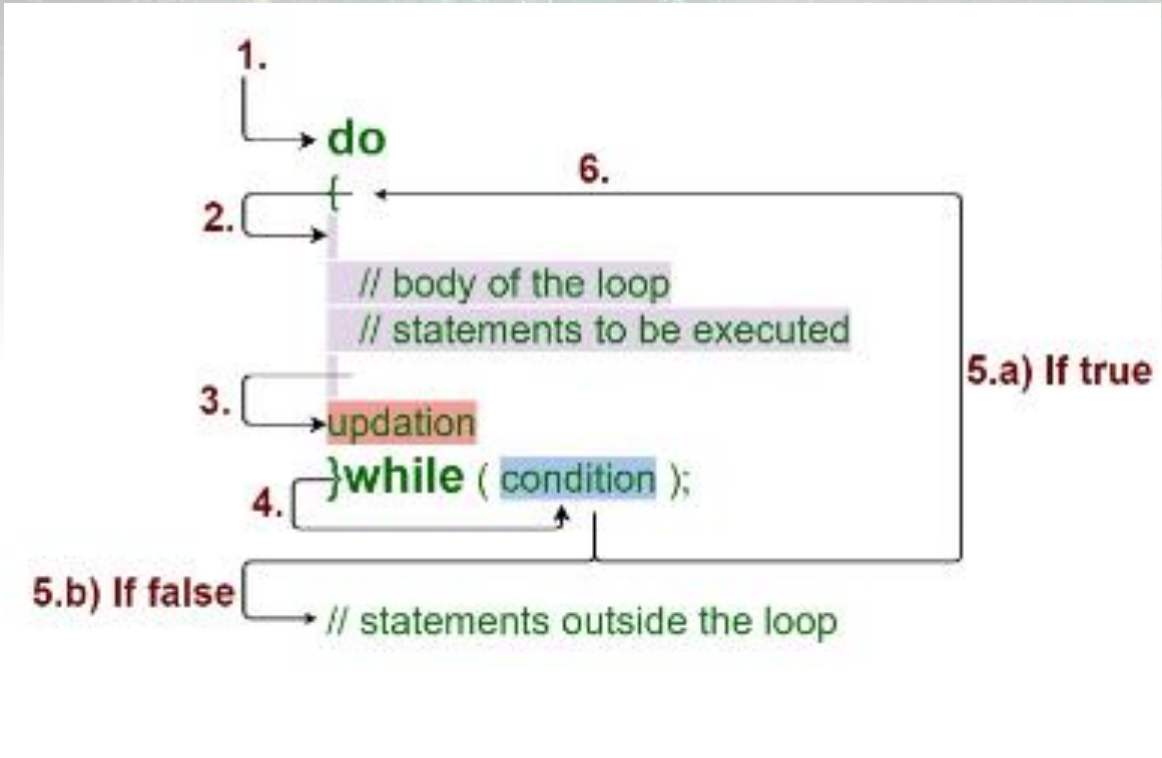
- Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression.
- If this expression is true, the loop will repeat. Otherwise, the loop terminates

Syntax

```
do
{
        // body of the loop.
} while(condition);
```



Iteration – Do While Loop



Iteration – Do While Loop

```
class DoWhileLoop
{
        public static void main(String args[])
         {
              System.out.println("The values from 1 to 10 is: ");
              int num = 1;
              do{
                        System.out.println(num);
                        num++;
              } while(num <= 10);
              }
}
```

```
E:\>javac DoWhileLoop.java

E:\>java DoWhileLoop
The values from 1 to 10 is:
1
2
3
4
5
6
7
8
9
10
```

Iteration – For Loop

- The java for loop is a compact way to iterate over a range of values. The Java for loop consumes initialization, condition, and increment/decrement in one line.
- If the number of iteration is fixed, then it is recommended to use for loop

Syntax

```
for(initialization; condition; incr/decr)
{
    //body of the loop.
}
```

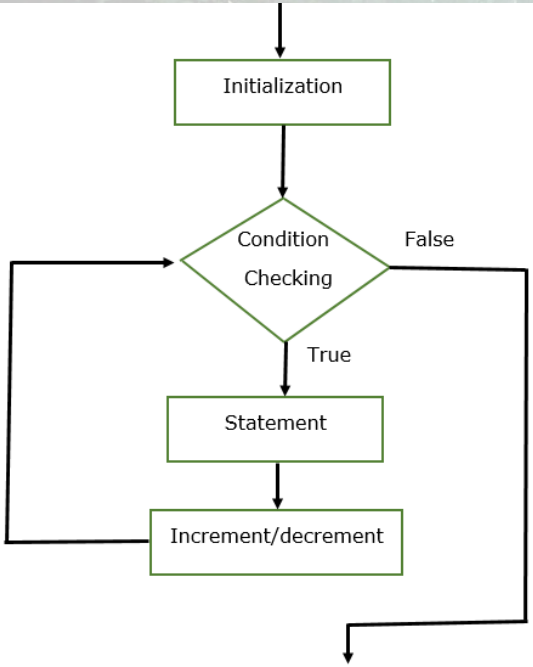
initialization: The initialization expression initializes the loop; it's executed once, as the loop begins..

condition: In this step, user can check the condition.

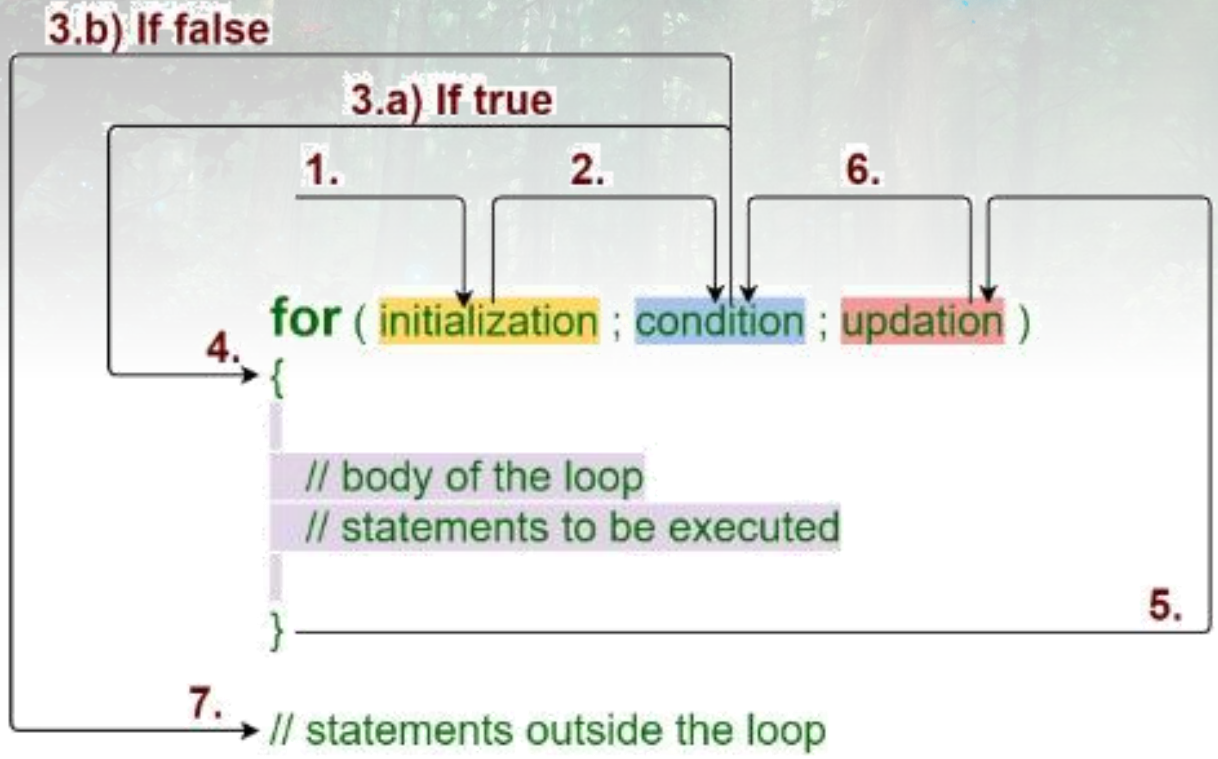
If the condition is true, then the body of the loop is executed otherwise not. If the termination expression evaluates to false, the loop terminates.

increment/decrement: user can increment or decrement the value of a variable.

Iteration – For Loop



Iteration – For Loop



Iteration – For Loop

```
class ForLoop
{
        public static void main(String args[])
        {
                System.out.println("The values from 1 to 10 is: ");

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

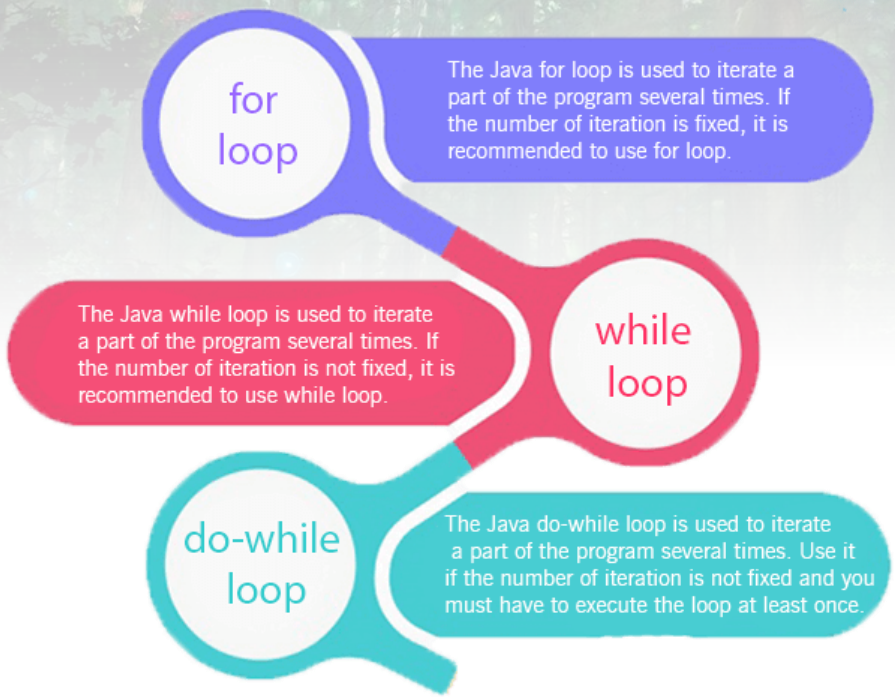
Declaring and Initializing loop control variable

Checking condition

Incrementing loop control variable

```
for (int i =0; i<10 ; i++) {

// Loop statements to be executed

}
```

Iteration – Loop Comparison



Iteration – Loop Comparison

BASIS FOR COMPARISON	WHILE	DO-WHILE

General Form	while (condition) {
	statements; //body of loop
	}
		do{
		.
		statements; // body of loop.
		.
		} while(Condition);

Controlling Condition	In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
Iterations	The iterations do not occur if, the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.

Jump Statements

- Java Jump statements transfer the control to other parts of the program.
- Java Jump statements provide different type of statements such as:

1. break statement.
2. continue statement.
3. return statement.

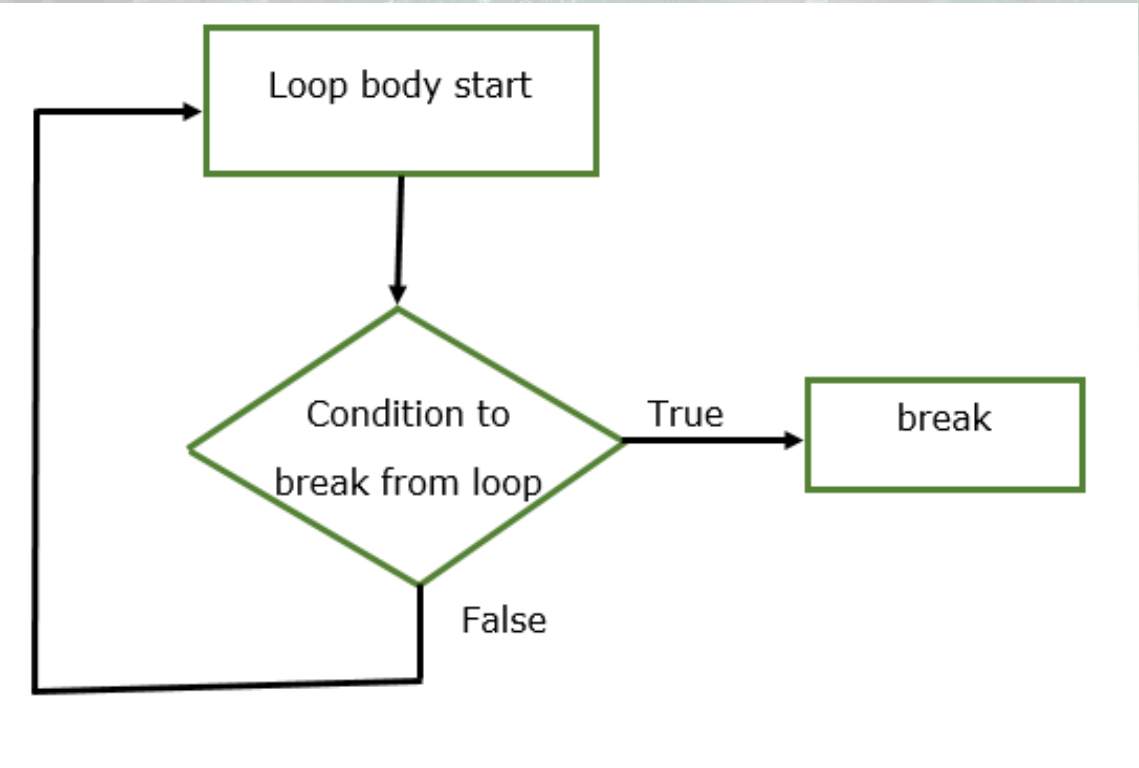
Jump Statements – Break

- Java break statement can be used to terminate the loop. It can be used inside a loop.
- Users can use a break statement in all types of loops such as: while loop, do-while loop, and for loop.
- When a break statement is encountered inside a loop, the loop is terminated and the program control goes to the next statement following the loop. In java, the break statement has three uses.
    1. It terminates a statement sequence in a switch statement.
    2. It can be used to exit a loop.
    3. It can be used as a "civilized" form of goto (labeled break).

Syntax

```
break;
```

Jump Statements – Break



Jump Statements – Break

```
class BreakLoop
{
        public static void main(String args[])
        {
                for(int i = 1; i<=10; i++)
                 {
                        if(i==6)
                        break;
                        System.out.println("i is:" +i);
                }
                System.out.println("Loop Complete !");
        }
}
```

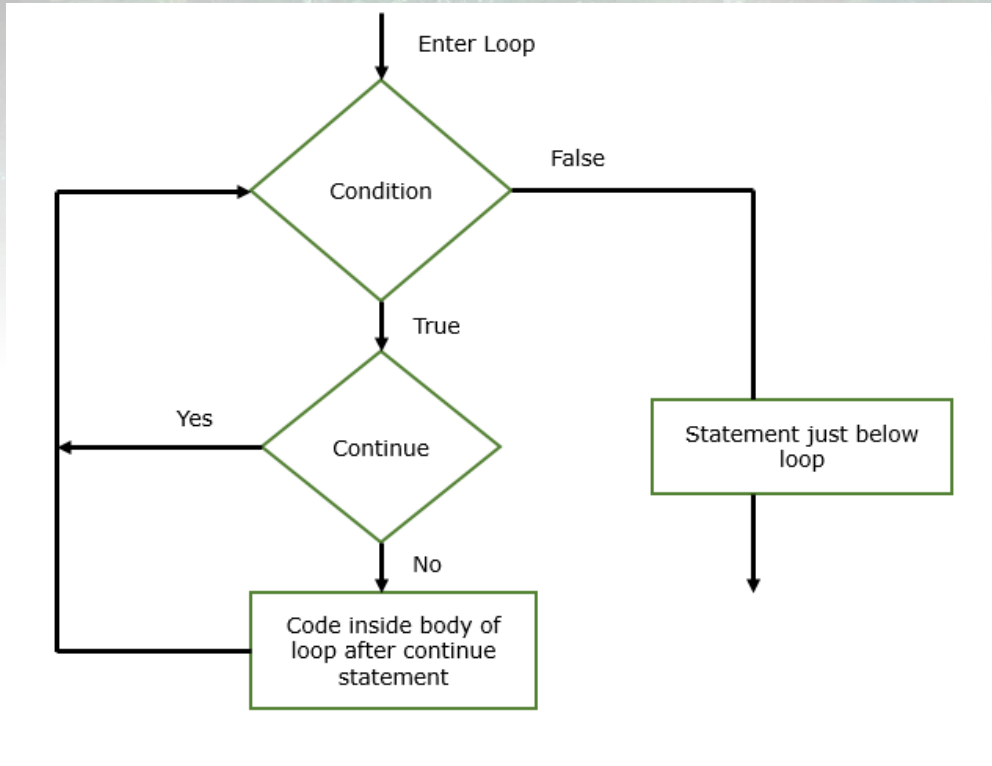
```
E:\>javac BreakLoop.java

E:\>java BreakLoop
i is: 1
i is: 2
i is: 3
i is: 4
i is: 5
Loop Complete !
```

Jump Statements – Continue

- The java continue statement skips the current iteration of a for loop, while loop, and do-while loop.
- User can use a continue statement with all types of loops such as: for loop, while loop, and do-while loop.
- In the while and do-while loops, a continue statement causes control to be transferred directly to the conditional expression that controls the loop.
- In for loop, the control goes first to the iteration portion of the for statement and then to the conditional expression.
- **Syntax**

```
continue;
```

Jump Statements – Continue



Jump Statements – Continue

```
class ContinueStatement
{
        public static void main(String args[])
        {
                for(int i = 1; i<=10; i++)
                 {
                        if(i%2 == 0)
                         continue;
                         System.out.print(i+ " ");
                }
        }
}
```

Output:
1 3 5 7 9

Jump Statements – return

- The java return statement is used to explicitly return from a method. It causes the program control to transfer back to the caller of the method. The return statement immediately terminates the method in which it is executed.

Jump Statements – return

```
class ReturnStatement
{
        public int getSum(int a, int b)
        {
                int c = a+b;
                return c;
        }
        public static void main(String args[])
        {
                ReturnStatement obj = new ReturnStatement();
                int sum = obj.getSum(10, 20);
                System.out.println("Sum of a and b is: " +sum);
        }
}
```

```
E:\>javac ReturnStatement.java

E:\>java ReturnStatement
Sum of a and b is: 30
```

Data Types – Java

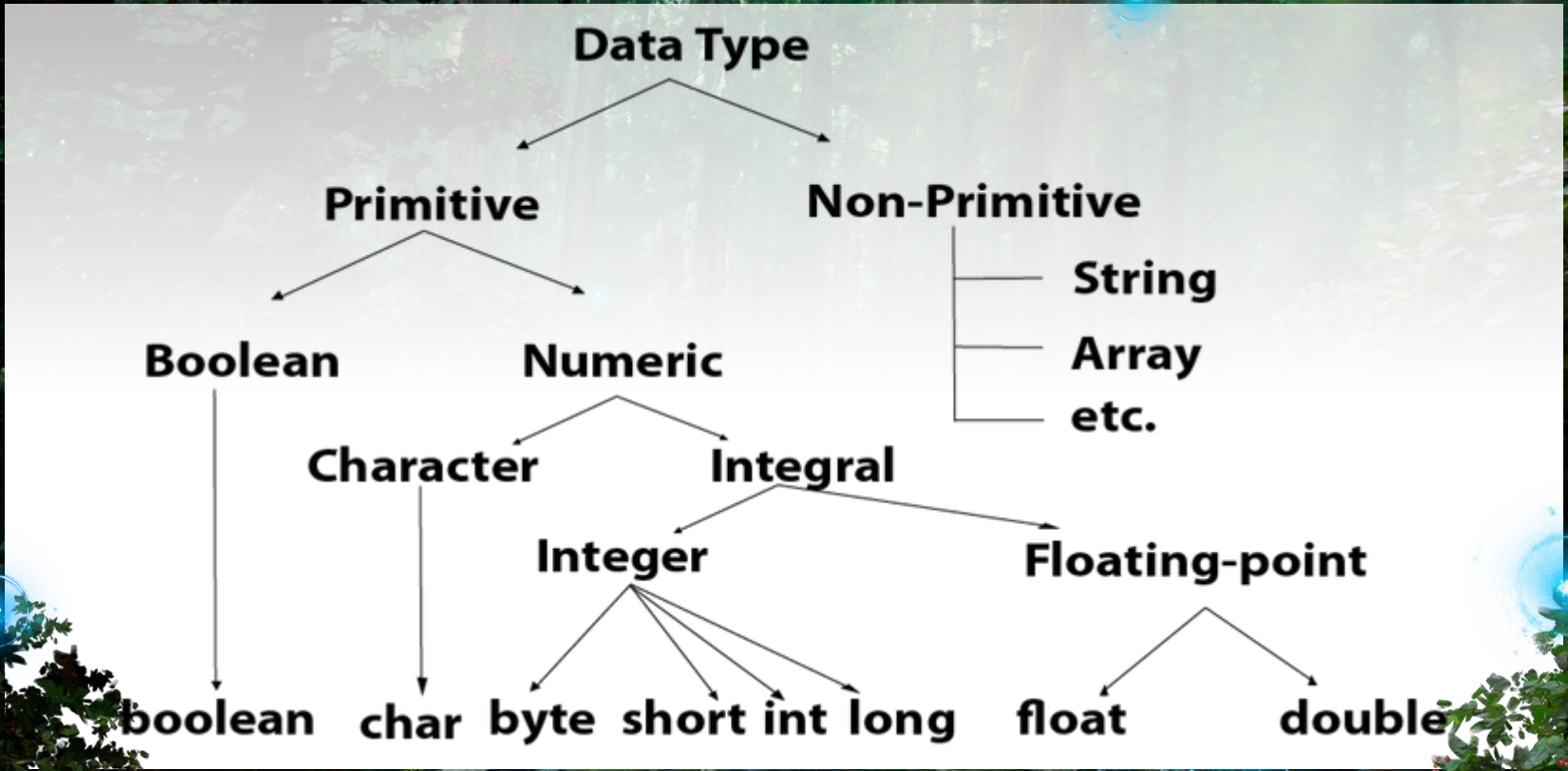
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

There are 8 types of primitive data types:

boolean data type
byte data type
char data type
short data type
int data type
long data type
float data type
double data type

Data Types – Java



Data Types – Java

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Data Types – Java

- boolean − Stores 1-bit value representing true or, false.
- byte − Stores twos compliment integer up to 8 bits.
- char − Stores a Unicode character value up to 16 bits.
- short − Stores an integer value upto 16 bits.
- int − Stores an integer value upto 32 bits.
- long − Stores an integer value upto 64 bits.
- float − Stores a floating point value upto 32bits.
- double − Stores a floating point value up to 64 bits.

Data Types – Java

Boolean Data Type

- The Boolean data type is used to store only two possible values: **true** and **false**.
- This data type is used for simple flags that track true/false conditions
- The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example Boolean one = false Boolean one = True

Byte Data Type

- The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive).
- Its minimum value is -128 and maximum value is 127. Its default value is 0.
- The byte data type is used to save memory in large arrays where the memory savings is most required. **It saves space because a byte is 4 times smaller than an integer**. It can also be used in place of "int" data type.

Example byte a = 10, byte b = -20

Data Types – Java

Short Data Type

- The short data type is a 16-bit signed two's complement integer.
- Its value-range lies between -32,768 to 32,767 (inclusive).
- Its minimum value is -32,768 and maximum value is 32,767.
- Its default value is 0.
- The short data type can also be used to save memory just like byte data type.
- A short data type is 2 times smaller than an integer.

Example short s = 10000, short r = -5000

Data Types – Java

Int Data Type

- The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive).
- Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647.
- Its default value is 0.
- The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example int a = 100000, int b = -200000

Data Types – Java

Long Data Type

- The long data type is a 64-bit two's complement integer.
- Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive).
- Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807.
- Its default value is 0.
- The long data type is used when you need a range of values more than those provided by int.

Example long a = 100000L, long b = -200000L

Data Types – Java

Float Data Type

- The float data type is a single-precision 32-bit IEEE 754 floating point.
- Its value range is unlimited.
- It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
- The float data type should never be used for precise values, such as currency.
- Its default value is 0.0F.

Example float f1 = 234.5f

Data Types – Java

Double Data Type

- The double data type is a double-precision 64-bit IEEE 754 floating point.
- Its value range is unlimited.
- The double data type is generally used for decimal values just like float.
- The double data type also should never be used for precise values, such as currency.
- Its default value is 0.0d.

Example double d1 = 12.3

Data Types – Java

Char Data Type

- The char data type is a single 16-bit Unicode character.
- Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).
- The char data type is used to store characters.

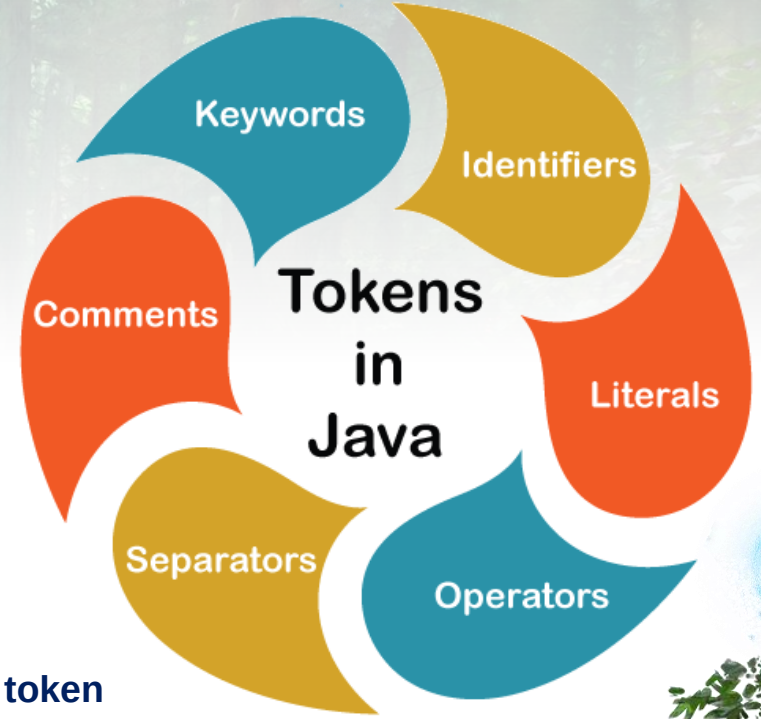
Example char letterA = 'A'

Why char uses 2 byte in java and what is \u0000 ?

- It is because java uses Unicode system not ASCII code system.
- The \u0000 is the lowest range of Unicode system.
- To get detail explanation about Unicode visit next page.

Lexical Issue or Token – Java

- Java Lexical Issues or Java Lexeme or Java Tokens
- Java Tokens are the smallest individual building block or smallest unit of a Java program; the Java compiler uses it for constructing expressions and statements.
- Java program is a collection of different types of tokens, comments, and white spaces.



White space is also considered as a token

Lexical Issues – Java

White Spaces

- Java is a free form language.
- This means that user do not need to follow any special indentation rules. In java, white spaces is a space, tab or new line.

Identifiers

- Identifiers are used for class names, method names and variable names.
- An identifier may be any descriptive sequence of uppercase and lowercase letters, numbers or the underscore and dollar sign design.

Literals – Java

- In programming literal is a notation that represents a fixed value (constant) in the source code.
- It can be categorized as an integer literal, string literal, Boolean literal, etc.
- It is defined by the programmer. **Once it has been defined cannot be changed**. Java provides five types of literals are as follows:

  - Integer
  - Floating Point
  - Character
  - String
  - Boolean

Literals – Java

Literal	Type
23	int
9.86	double
false, true	boolean
'K', '7', '-'	char
“Programming"	String
null	any reference type

Identifiers– Java

- Identifiers are used to name a variable, constant, function, class, and array.
- It usually defined by the user.
- It uses letters, underscores, or a dollar sign as the first character.
- The label is also known as a special kind of identifier that is used in the goto statement. Remember that the identifier name must be different from the reserved keywords.
- There are some rules to declare identifiers are:

1. The first letter of an identifier must be a letter, underscore or a dollar sign. It cannot start with digits but may contain digits.
2. The whitespace cannot be included in the identifier.
3. Identifiers are case sensitive.

Examples	PhoneNumber
_phonenumber	PRICE
$circumference	radius
jagged_array	a
12radius **//invalid**	a1

Keywords – Java

Keywords

- These are the pre-defined reserved words of any programming language.
- Each keyword has a special meaning.
- It is always written in lower case. Java provides the following keywords:

abstarct	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Separators – Java

Separators

- The separators in Java is also known as punctuators.
- There are nine separators in Java, are as follows:

separator <= ; | , | . | (|) | { | } | [|]

Separators – Java

- **Square Brackets []:** It is used to define array elements. A pair of square brackets represents the single-dimensional array, two pairs of square brackets represent the two-dimensional array.
- **Parentheses ():** It is used to call the functions and parsing the parameters.
- **Curly Braces {}:** The curly braces denote the starting and ending of a code block.
- **Comma (,):** It is used to separate two values, statements, and parameters.
- **Assignment Operator (=):** It is used to assign a variable and constant.
- **Semicolon (;):** It is the symbol that can be found at end of the statements. It separates the two statements.
- **Period (.):** It separates the package name form the sub-packages and class. It also separates a variable or method from a reference variable.

Separators – Java

Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a **for** statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference. (Added by JDK 8.)

Comments – Java

Comments

- Comments allow us to specify information about the program inside our Java code. Java compiler recognizes these comments as tokens but excludes it form further processing.
- The Java compiler treats comments as whitespaces.
- Java provides the following two types of comments:

Line Oriented: It begins with a pair of forwarding slashes (//).

Block-Oriented: It begins with /* and continues until it founds */.

Operators – Java

Operators

- A character or a symbol is used to perform some operation
- In programming, operators are the special symbol that tells the compiler to perform a special operation.
- Java provides different types of operators that can be classified according to the functionality they provide.
- There are eight types of operators in Java, are as follows:

  - Arithmetic Operators
  - Assignment Operators
  - Relational Operators
  - Unary Operators
  - Logical Operators
  - Ternary Operators
  - Bitwise Operators
  - Shift Operators

Operand/Operator/Expression – Java

- Variables or values which we are apply on the operator to perform some operations.
- Ex: **a%b**, where a and b are operands, **a+b5**, where **a is an operand** and **5 is an value**.
- **Operation/Expression:** Operation= operand + operator
- Ex: a+b

Operators – Java

Operator	Symbols
Arithmetic	+ , - , / , * , %
Unary	++ , - - , !
Assignment	= , += , -= , *= , /= , %= , ^=
Relational	==, != , < , >, <= , >=
Logical	&& , ||
Ternary	(Condition)?(Statement1):(Statement2);
Bitwise	& , | , ^ , ~
Shift	<< , >> , >>>

Arithmetic Operators – Java

- These operators used to perform the Arithmetic operations

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

Arithmetic Operators Example – Java

```
import java.io.*;
class Addition
{
    public static void main(String[] args)
    {
        int num1 = 10, num2 = 20;
        int sum = 0, sub=0, div=0, mul=0, modd=0;
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);
        sum = num1 + num2;
        sub = num1 - num2;
        mul = num1 * num2;
        div = num1 / num2;
        modd = num1 % num2;
       System.out.println("The Sum = " + sum + “ The Sub ="  + sub + "The Div="+ div);
       System.out.println("The Mul = " + mul + "The Mod="+ modd);
    }
}
```

```
D:\JU\2ndSem\Lab>javac Arth.java

D:\JU\2ndSem\Lab>java Arth
num1 = 10
num2 = 20
The Sum = 30 The Sub =-10The Div=0
The Mul = 200The Mod=10
```

Bitwise Operator – Java

- Bitwise operators are used to performing the manipulation of individual bits of a number.
- They can be used with any integral type (char, short, int, etc.).

Bitwise OR (|)

- This operator is a binary operator, denoted by '|'. It returns bit by bit OR of input values, i.e., if either of the bits is 1, it gives 1, else it shows 0.

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise OR Operation of 5 and 7

```
  0101
| 0111
 ________
  0111  = 7 (In decimal)
```

Bitwise Operator – Java

Bitwise AND (&)

- This operator is a binary operator, denoted by '&.' It returns bit by bit AND of input values, i.e., if both bits are 1, it gives 1, else it shows 0.

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise AND Operation of 5 and 7

```
  0101
& 0111
 ________
  0101  = 5 (In decimal)
```

Bitwise Operator – Java

Bitwise XOR (^)

- This operator is a binary operator, denoted by '^.' It returns bit by bit XOR of input values, i.e., if corresponding bits are different, it gives 1, else it shows 0.

a = 5 = 0101 (In Binary)
b = 7 = 0111 (In Binary)

Bitwise XOR Operation of 5 and 7
  0101
^ 0111
_____
  0010 = 2 (In decimal)

Bitwise Operator – Java

Bitwise Complement (~)

- This operator is a unary operator, denoted by **'~.'**
- It returns the one's complement representation of the input value, i.e., with all bits inverted, which means it makes every 0 to 1, and every 1 to 0.

a = 5 = 0101 (In Binary)

Bitwise Complement Operation of 5

~ 0101

1010 = 10 (In decimal)

Bitwise Operators Example – Java

```
class BitWiseOp {
   public static void main(String[] args)
   {
      int a = 5;
      int b = 7;

      // bitwise and 0101 & 0111=0101 = 5
      System.out.println("a&b = " + (a & b));

      // bitwise or // 0101 | 0111=0111 = 7
      System.out.println("a|b = " + (a | b));

      // bitwise xor // 0101 ^ 0111=0010 = 2
      System.out.println("a^b = " + (a ^ b));
      System.out.println("~a = " + ~a);

      }
   }
```

Output

```
a&b = 5
a|b = 7
a^b = 2
~a = -6
```

Logical Operators – Java

- Logical operators are used to performing logical **“AND”, “OR” and “NOT”** operations, i.e. the function similar to AND gate and OR gate in digital electronics.
- They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition under particular consideration.
- While using AND operator, the second condition is not evaluated if the first one is false. Whereas while using OR operator, the second condition is not evaluated if the first one is true, i.e. the AND and OR operators have a short-circuiting effect.

Logical Operators – Java

- AND Operator (&&) – if(a && b) [if true execute else don’t]
- OR Operator (||) – if(a || b) [if one of them is true execute else don’t]
- NOT Operator (!) – !(a<b) [returns false if a is smaller than b]

Logical Operators – Java

1. For AND operator

Condition 1: c > a
Condition 2: c > b

Output: **True** [Both Conditions are true]

2. For OR Operator

Condition 1: c > a
Condition 2: c > b

Output: **True** [One of the Condition if true]

3. For NOT Operator
Condition 1: c > a
Condition 2: c > b

Output: **False** [Because the result was true and NOT operator did it's opposite]

Logical Operators ‘AND’ – Java

```
class OperatorExample
{
public static void main(String args[])
{
        int a=10;
        int b=5;
        int c=20;
        System.out.println( a<b && a<c );        //false && true = false
        System.out.println( a<b & a<c );         //false & true = false
}
}
```

Output

```
false
false
```

Logical Operators ‘OR’ – Java

```
class OperatorExample
{
public static void main(String args[])
{
        int a=10;
        int b=5;
        int c=20;
        System.out.println( a>b  || a<c);     //true || true = true
        System.out.println( a>b | a<c);      //true | true = true
}
}
```

Output

```
true
true
```

Logical Operators ‘NOT’ – Java

```
class Logical
{
    public static void main(String[] args)
    {
        int a = 10, b = 1;

        System.out.println(“A Value = " + a);
        System.out.println(“B Value = " + b);
        System.out.println(“ Not of (a < b) = " + !(a < b) );
        System.out.println(“Not of (a > b) = " +  !(a > b) );
    }
}
```

Output

```
A Value = 10
B Value = 1
Not of (a < b) = true
Not of (a > b) = false
```

Assignment Operator– Java

Operator	Operation	Equivalent to
=	num = 5	num = 5
+=	num+=5	num = num+5
-=	num-=5	num = num-5
*=	num*=5	num = num*5
/=	num/=5	num = num/5
%=	num%=5	num = num%5

Assignment Operator– Java

```
class Assign
{
  public static void main(String arg[])
   {
                Scanner sr=new Scanner(System.in);
               int first,second=5;
               System.out.println("Enter the first number:");
               first=sr.nextInt();
                first+=second;  (first=first+second)
               System.out.println("Addition="+first);
               first-=second;  (first=first-second)
               System.out.println("Subtraction="+first);
               first*=second;  (first=first*second)
               System.out.println("Multiplicationd="+first);
               first/=second;  (first=first/second)
               System.out.println("Division="+first);
               first%=second;  (first=first%second)
               System.out.println("Modulus="+first);
               }
                }
```

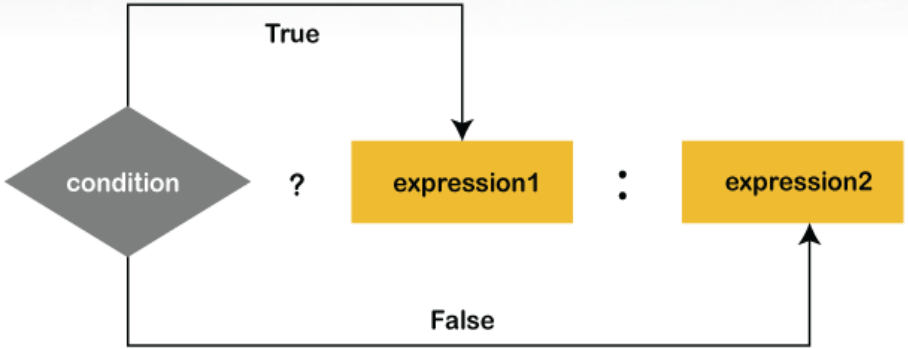
Output:

```
Enter your first number:
90
Addition=95
Subtraction=90
Multiplication=450
Division=90
Modulus=0
```

Ternary Operator – Java

- The given statement states that **if the condition returns true, expression1 gets executed**, else the **expression2 gets executed and the final result stored in a variable.**

Syntax: variable = (condition) ? expression1 : expression2



Ternary Operator – Java

Example

```
class Larg
{
        public static void main(String args[])
        {
                int x=69;
                int y=89, largest=0;
                largest = (x > y) ? x : y ;
                System.out.println("The largest numbers is:  "+largest);
        }
}
```

```
D:\JU\2ndSem\Lab>javac Larg.java

D:\JU\2ndSem\Lab>java Larg
The largest numbers is:  89
```

Relational Operator – Java

- The Relational Operators are used to check the relations between the two operands.
- Relational operators are also called comparison operators because it is used to make a comparison between the two operands.
- The result of relational operators is always a boolean value.

Operators	Operations
==	Equal to
!=	Not Equal to
>	Greater than
>=	Greater than equal to
<	Less than
<=	Less than equal to

Relational Operator – Java

```
class RelationalOperator
{
    public static void main(String[] args)
    {
        int value1 = 10;
        int value2 = 20;

        System.out.println("value1 == value2 --> " +(value1 == value2));
        System.out.println("value1 != value2 --> " +(value1 != value2));
        System.out.println("value1 > value2 --> " +(value1 > value2));
        System.out.println("value1 >= value2 --> " +(value1 >= value2));
        System.out.println("value1 < value2 --> " +(value1 < value2));
        System.out.println("value1 <= value2 --> " +(value1 <= value2));
    }
}
```

```
E:\>javac RelationalOperator.java

E:\>java RelationalOperator
value1 == value2 --> false
value1 != value2 --> true
value1 > value2 --> false
value1 >= value2 --> false
value1 < value2 --> true
value1 <= value2 --> true
```

Unary Operator – Java

- In Java, the unary operator is an operator that can be used only with an operand.
- It is used to represent the positive or negative value, increment/decrement the value by 1, and complement a Boolean value.

Unary Operators in Java

- Unary Plus
- Unary Minus
- Increment Operator
- Decrement Operator
- Logical Complement Operator

Unary Plus

Unary Minus

Increment Operator

Decrement Operator

logical compement Operator

Unary Operator – Java

Operator Name	Symbol	Description	Example	Equivalent Expression
Unary Plus	+	It is used to represent the **positive** value.	+a	a
Unary Minus	-	It is used to represent the **negative** value.	-a	-
Increment Operator	++	It **increments** the value of a variable by **1**.	++a or a++	a=a+1
Decrement Operator	--	It **decrements** the value of a variable by **1**.	--a or a--	a=a-1
Logical Complement Operator	!	It **inverts** the value of a boolean variable.	!true	-

Unary Operator – Java

```
//Increment/Decrement Operator
class Inc  {
public static void main(String f[])  {
        int p=9;r=100;
        System.out.println();
        System.out.println();
        System.out.println("Value of P::"+p);
        System.out.println("Post inc no change::"+(p++));
        System.out.println("Post Inc"+p);
        System.out.println("Pre Inc"+(++p));
        System.out.println();
        System.out.println();
        System.out.println("Value of R::"+r);
        System.out.println("Post dec no change"+(r--));
        System.out.println("Post Dec:"+r);
        System.out.println("Pre Dec:"+(--r));
        }
}
```

Output

```
Value of P:9
Post inc no change:9
Post inc:10
Pre inc:11

Value of R:100
Post dec no change:100
Post dec:99
Pre dec:98
```

Unary Operator – Java

```
//Unary Minus Operator
class UnOp
{
        public static void main(String[] args)
        {
                int x1 = 12;
                System.out.println("Before operation = " + x1);
                x1 = -x1;
                System.out.println("After operation = " + x1);
        }
}
```

Output:

```
Before operation = 12
After operation  = -12
```

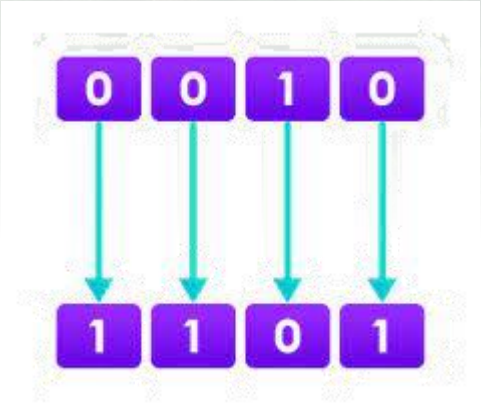
Unary Operator – Java

```
//Compliment Operator
class LogicOp
{
public static void main(String[]args)
{
      boolean Var1 = true, Var2 = false;
      Var1 = !Var1;
      System.out.println(Var1);
      Var2 = !Var2;
      System.out.println(Var2);
   }
}
```

Output:

```
false
true
```

Example Compliment



Shift Operator – Java

- The decimal representation of a number is a base-10 number system having only ten states 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.
- The Binary representation of a number is a base-2 number system having only two states 0 and 1. For example, the binary representation of 4, a base-9 decimal number system is given by 100

Binary	Decimal
1 0 1	5
↓ Left shift once	
1 0 1 0	10
↓ Left shift twice	
1 0 1 0 0 0	40

Shift Operator – Java

- In Java, shift operators are the special type of operators that work on the bits of the data.
- These operators are used to shift the bits of the numbers from left to right or right to left depending on the type of shift operator used.

There are three types of shift operators in Java:

- **Signed Left Shift Operator (<<)**
- **Signed Right Shift Operator (>>)**
- **Unsigned Right Shift Operator (>>>)**

Left Shift Operator – Java

- Left shift operator shifts the bits of the number towards left a specified number of positions. The symbol for this operator is <<. When you write x<<n, the meaning is to shift the bits of x towards left n specified positions.

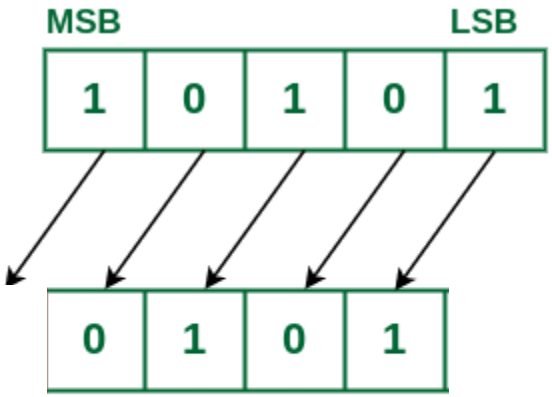
Example

- If x=10, then calculate x<<2 value.
- Shifting the value of x towards the left two positions will make the leftmost 2 bits to be lost. The value of x is 10. The binary representation of 10 is **00001010**.

The procedure to do left shift explained in here

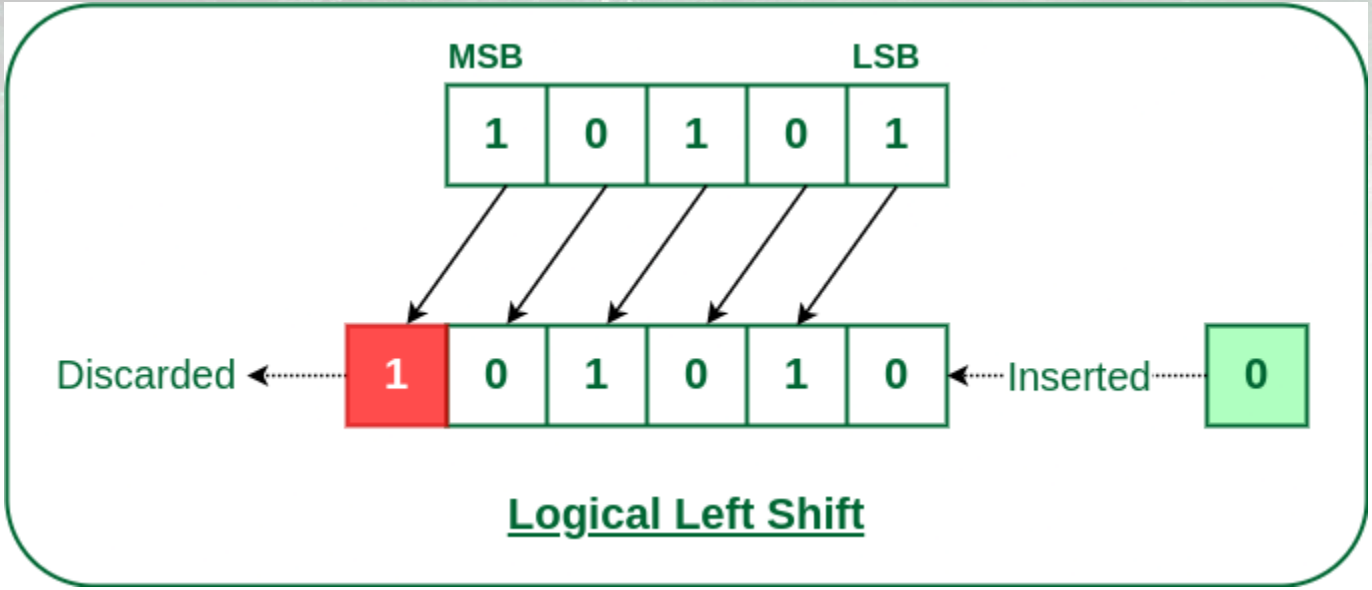
Observe the above example, after **shifting the bits to the left the binary number 00001010** (in decimal 10) becomes **00101000** (in decimal 40).

Left Shift Operator – Java



Logical Left Shift

Left Shift Operator – Java



Left Shift Operator – Java

Syntax

x << n

Here,
x: an integer
n: a non-negative integer

Example
x = 20 and n = 3

x << 3 = 160

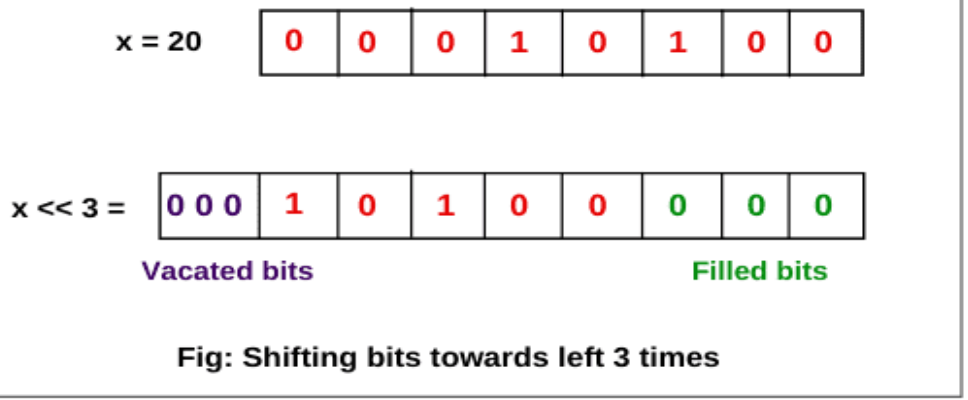


Fig: Shifting bits towards left 3 times

Left Shift Operator – Java

```
class LFTS
{
    public static void main (String[] args)
{
        int x = 5;
        int n = 1;
        int answer = x << n;
        System.out.println("Left shift " + x + " by " + n + " positions : " + answer);

    }
}
```

Output

Left shift 5 by 1 positions : 10

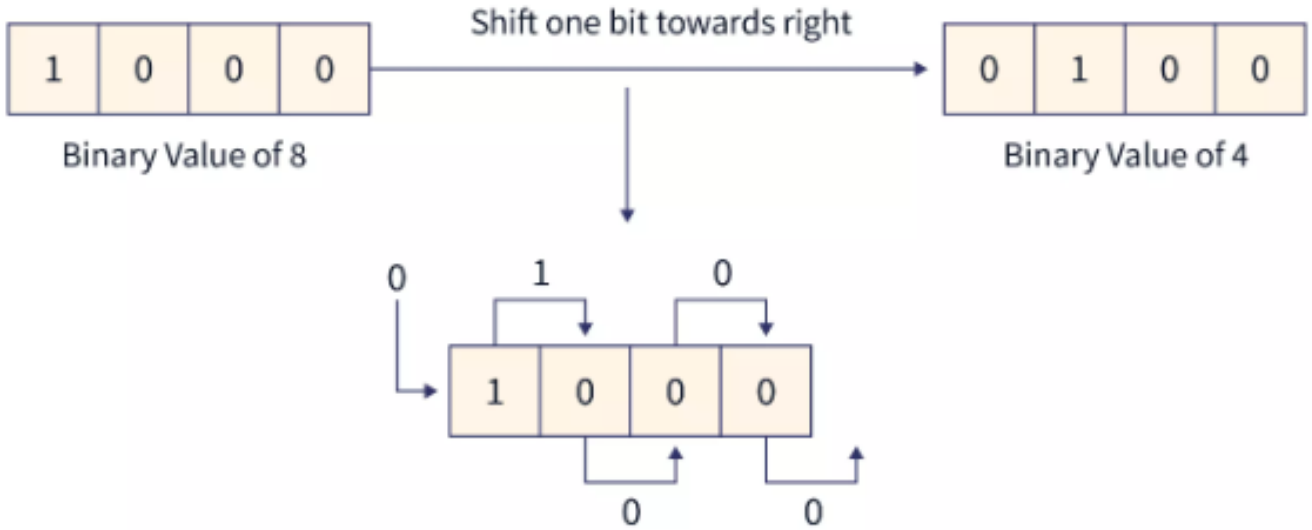
Right Shift Operator – Java

- The signed right shift operator is a special type of operator used to move the bits of the expression to the right according to the number specified after the operator.
- The Right Shift Operator shifts the bits of the number towards right a specified n number of positions.
- Right shift operator represented by the symbol >>, read as double greater than. When user write **x>>n**, the meaning is to shift the bits **x towards the right n specified positions**.

Right Shift Operator – Java

- If **x=10**, then calculate **x>>2** value.
- Shifting the value of x towards the right two positions will make the rightmost 2 bits to be lost. The value of x is 10.
- The binary representation of 10 is 00001010. The procedure to do right shift explained in the following example:
- Observe the above example, after shifting the bits to the right the binary number **00001010 (in decimal 10)** becomes **00000010 (in decimal 2).**

Right Shift Operator – Java



Right Shift Operator – Java

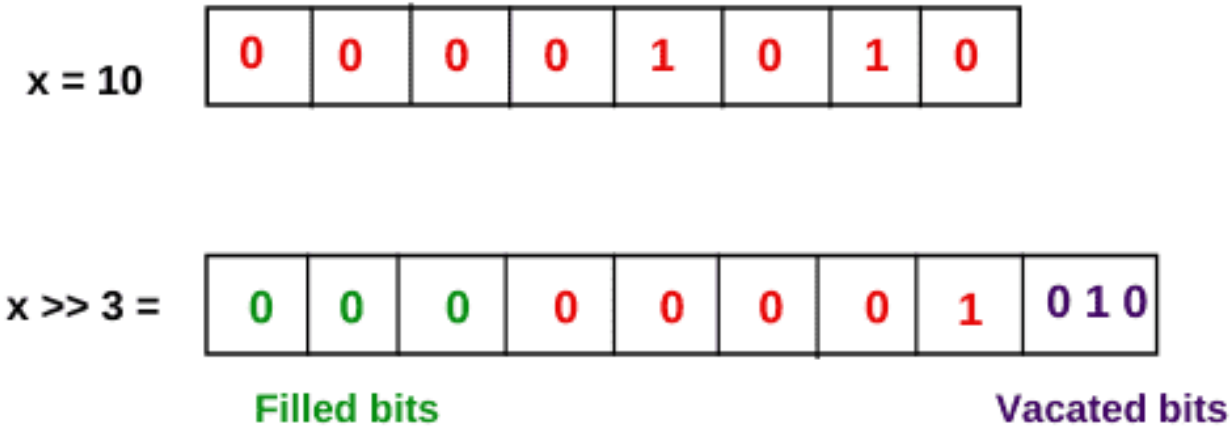


Fig: Shifting bits towards right 3 times

Unsigned Right Shift Operator – Java

- Unsigned Right Shift Operator moves the bits of the integer a given number of places to the right.
- The sign bit was filled with 0s. The Bitwise Zero Fill Right Shift Operator is represented by the symbol **>>>**.

Syntax:

left_operand >>> number

Unsigned Right Shift Operator – Java

- Right shift ">>" keeps the sign extension while shifting bit patterns, but right shift without sign doesn't keep the original sign bit intact, it fills with zero.
- **This means after using ">>>" a negative number can turned into positive number**

For non negative number:

a = 12 , b = 2

a >>> b = 3

For negative number:

a = -10 , b = 3

a >>> b = 536870910

Shift Operator – Java

```
class OperatorShifting
{
	public static void main(String args[])
	{
		byte x, y;
		x=10;
		y=-10;
		System.out.println("Bitwise Left Shift: x<<2 = "+(x<<2));
		System.out.println("Bitwise Right Shift: x>>2 = "+(x>>2));
		System.out.println("Bitwise Zero Fill Right Shift: x>>>2 = "+(x>>>2));
		System.out.println("Bitwise Zero Fill Right Shift: y>>>2 = "+(y>>>2));
	}
}
```

Output

```
Bitwise Left Shift: x<<2 = 40
Bitwise Right Shift: x>>2 = 2
Bitwise Zero Fill Right Shift: x>>>2 = 2
Bitwise Zero Fill Right Shift: y>>>2 = 1073741821
```

Operator Precedence – Java

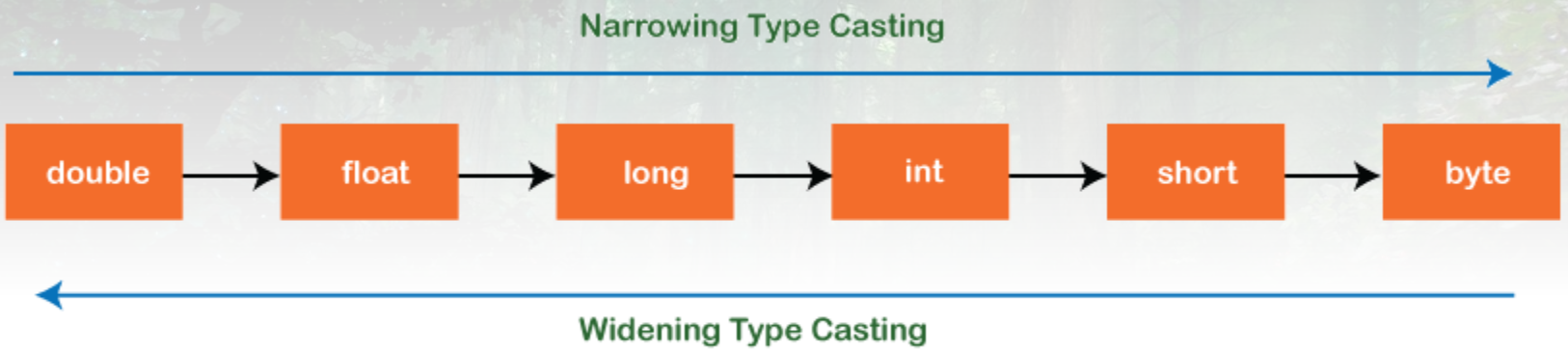
- Operators on the same line have equal precedence. When operators of equal precedence appear in the same expression, a rule must govern which is evaluated first. All binary operators except for the assignment operators are evaluated from left to right; assignment operators are evaluated right to left

Operators	Precedence
!, +, - (unary Operators)	First (Highest)
* , / , %	Second
+ , -	Third
< , <= , >=, >	Fourth
== , !=	Fifth
&&	Sixth
||	Seventh
= (assignment Operator)	Lowest

Type Conversion and Casting

- **In type casting, a data type is converted into another data type by a programmer using casting operator.**
- **Whereas in type conversion, a data type is converted into another data type by a compiler.**
- Type casting can be applied to compatible data types as well as incompatible data types

Type Casting – Java



Type Casting in Java

Type Casting – Java

- When a data type is converted into another data type by a programmer or user while writing a program code of any programming language, the mechanism is known as type casting. The programmer manually uses it to convert one data type into another.

Type Casting – Java

Syntax

Destination_datatype var= (target_datatype) variable;

(data_type) it is known as casting operator

Example

```
float b = 3.0;

int a = (int) b;
```

Two types of casting are possible in Java are as follows:

- **Implicit type casting** (also known as automatic type conversion)
- **Explicit type casting**

Type Casting Implicit – Java

```
class Conv
{
    public static void main(String[] args)
    {
        int i = 100;
        long l = i;
        float f = l;

        System.out.println("Int value " + i);
        System.out.println("Long value " + l);
        System.out.println("Float value " + f);
    }
}
```

Output

Int value 100
Long value 100
Float value 100.0

Type Casting Explicit – Java

```
class Conv {
    public static void main(String[] args)
    {
        double d = 100.04;
        long L = (long)d;
        int i = (int) L;
        System.out.println("Double value " + d);
        System.out.println("Long value " + L);
        System.out.println("Int value " + i);
    }
}
```

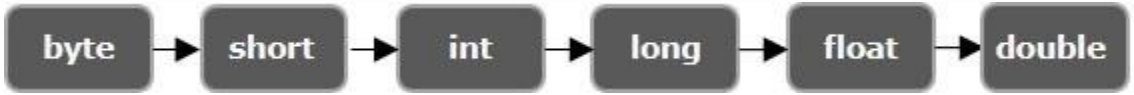
Output

Double value 100.04
Long value 100
Int value 100

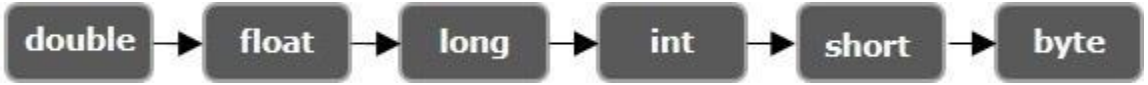
Type Conversion – Java

- Converting one primitive datatype into another is known as Type Conversion in Java.
- User can cast the primitive datatypes in two ways namely, Widening and, Narrowing.

Widening (Small size to Large size



Narrowing (Large size to Small size



Type Conversion ‘Narrowing’ – Java

```
import java.util.Scanner;
class NarrowingExample {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter an integer value: ");
    int i = sc.nextInt();
    char ch = (char) i;
    System.out.println("Character value of the given integer: "+ch);
  }
}
```

Example

Enter an integer value:
67
Character value of the given integer: C

Type Conversion ‘Widening’ – Java

```
import java.util.Scanner;
class WideningExample {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter the Character value: ");
    char c = sc.next();
    int ch = (int) c;
    System.out.println(“Integer value of the given Character: "+ch);
  }
}
```

Example

Enter the Character value:
B
Integer value of the given Character : 66

Code Block – Java

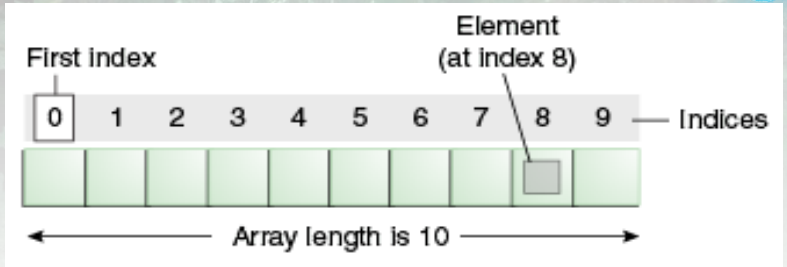
- Code block in Java is used for grouping of two or more statements.
- The code in Java always starts with curly braces and ends with close curly braces.
- A code block can be used inside a block.
- There is no restriction number of blocks and the level of nesting the block can be nested and can be included inside another Code block.
- It is generally used in loops statements (For, while loop, do-while loop).
- Class and method content are blocks too.

```java
public class WriteBlockOfCode {
        public static void main (String[] args) {
            System.out.println("This is the first statement in the block!");
            int i=100;
            System.out.println("This is the integer variable I created: " + i);
            System.out.println("This is the last statement in the block!");
        }
}
```

Arrays – Java

- Normally, an array is a collection of similar type of elements which has contiguous memory location.
- Java array is an object which contains elements of a similar data type.
- Additionally, The elements of an array are stored in a contiguous memory location.
- It is a data structure where user store similar elements.
- Users can store only a fixed set of elements in a Java array.
- Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Arrays – Java



Advantages

- **Code Optimization:** It makes the code optimized, users can retrieve or sort the data efficiently.
- **Random access:** User can get any data located at an index position.

Disadvantages

- **Size Limit:** User can store only the fixed size of elements in the array. It doesn't grow its size at runtime.

Array Types – Java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array
Syntax

```
dataType[] arr;
dataType []arr;
dataType arr[];
```

Initialization of Single Dimensional Array

```
datatype variablename=new datatype[size];
```

Ex: int x[5];
int x=new int[5];

Single Dimensional Array – Java

```
class 1D
{
    public static void main(String args[])
    {
            int a[]=new int[5];
            a[0]=10;
            a[1]=20;
            a[2]=70;
            a[3]=40;
            a[4]=50;
            for(int i=0;i<a.length;i++)
            {
              System.out.println(a[i]);
            }
    }
}
```

Output:

```
10
20
70
40
50
```

Multi Dimensional Array – Java

- Here, data is stored in row and column based index (also known as matrix form).

Syntax

```
dataType[][] arrayVar;
dataType [][]arrayVar;
dataType arrayVar[][];
dataType []arrayVar[];
```

Initialization of 2D Array

```
int[][] arr=new int[3][3]
```

```
arr[0][0]=1
arr[0][1]=2
arr[0][2]=3
arr[1][0]=4
arr[1][1]=5
arr[1][2]=6
arr[2][0]=7
arr[2][1]=8
arr[2][2]=9
```

Multi Dimensional Array – Java

```
class Testarray3
{
public static void main(String args[])
{
      int arr[][]={   {1,2,3},
                      {2,4,5},
                      {4,4,5}  };

      for(int i=0;i<3;i++)
      {
          for(int j=0;j<3;j++)
           {
                  System.out.print(arr[i][j]+" ");
           }
                 System.out.println();
       }
 }
}
```

Output:

```
1 2 3
2 4 5
4 4 5
```

For Each Array – Java

- It is used exclusively to loop through elements in an array

Syntax

```
for(dataType item : array) {
    ...
}
```

array - an array or a collection
item - each item of array/collection is assigned to this variable
dataType - the data type of the array/collection

Example for For Each

```
class DisplayArray {
  public static void main(String[] args) {
    int arr[] = {10, 20, 30, 40, 50};
        for(int i : arr)
        {
            System.out.print(i + " ");
        }
  }
}
```

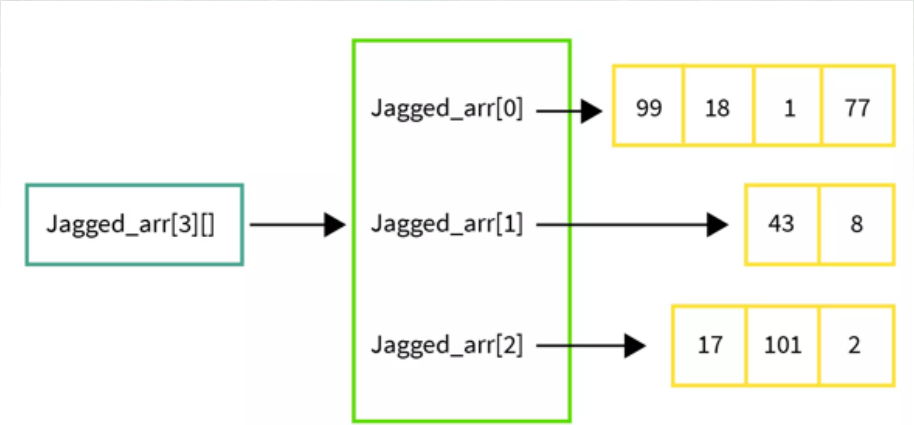
Output:-

```
10 20 30 40 50
```

Jagged Array – Java

- Creating odd number of columns in a 2D array, it is known as a jagged array.
- In other words, it is an array of arrays with different number of columns.

0 1 2
3 4 5 6
7 8



Jagged Array – Java

```
import java.util.*;
import java.io.*;
class Main {
  public static void main(String[] args) {
        int arr[][] = new int[3][];
            arr[0] = new int[]{99,100,101};
            arr[1] = new int[]{199,200};
            arr[2] = new int[]{299,300,301,302,303};
        System.out.println("Elements of 2-D Jagged Array");
          for (int i = 0; i < arr.length; i++) {
                    for (int j = 0; j < arr[i].length; j++)
                              System.out.print(arr[i][j] + " ");
                  System.out.println();
      }
  }
}
```

Output

```
Elements of 2-D Jagged Array
99 100 101
199 200
299 300 301 302 303
```

Benefits of Jagged Array – Java

- Jagged arrays are multidimensional arrays which can store multiple dimensions and can improve performance
- One of its most significant benefits is that it simplifies the process when several dimensions of data need to be stored under the same variable name.
- Additionally, it aids in memory management, which enhances the program's ability to run quickly and smoothly

Simple Programs in Array – Java

```
class DisplayArray
{
  public static void main(String[] args)
{
            int arr[] = {10, 20, 30, 40, 50};
            for(int i=0; i<arr.length; i++)
           {
                System.out.print(arr[i] + " ");
           }
  }
}
```

Output:-

```
10 20 30 40 50
```

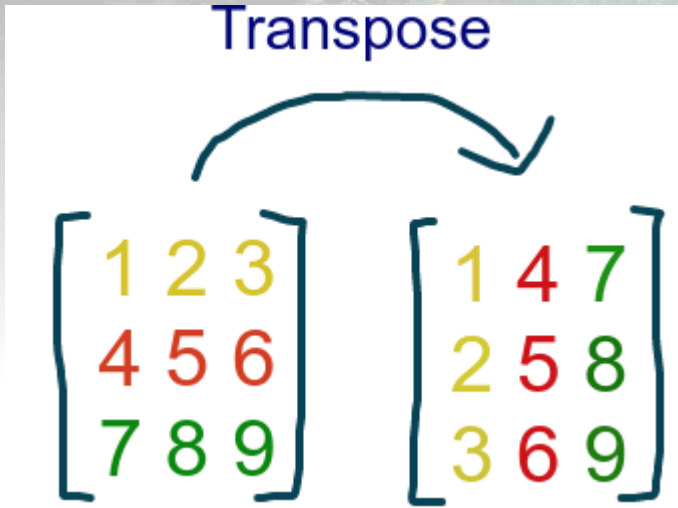
Simple Programs in Array – Java

```java
import java.util.Scanner;

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

    // create Scanner class object to read input
    Scanner scan = new Scanner(System.in);

    // declare an array variable of int type
    int arr[] = null;

    // ask the length
    System.out.print("Enter array length: ");
    int length = scan.nextInt();

    // assign size to the array
    arr = new int[length];

    // take input values
    System.out.println("Enter array elements: ");
    for(int i=0; i<arr.length; i++) {
      arr[i] = scan.nextInt();
    }

    // display the array using for loop
    System.out.println("The Entered array: ");
    for(int i=0; i<arr.length; i++) {
      System.out.print(arr[i] + " ");
    }
  }
}
```

Output:-

```
Enter array length: 5
Enter array elements:
15 25 35 45 55
The Entered array:
15 25 35 45 55
```

Array Practice– Java



These are the calculations:

3+4=7	8+0=8
4+1=5	6-9=-3

Challenge

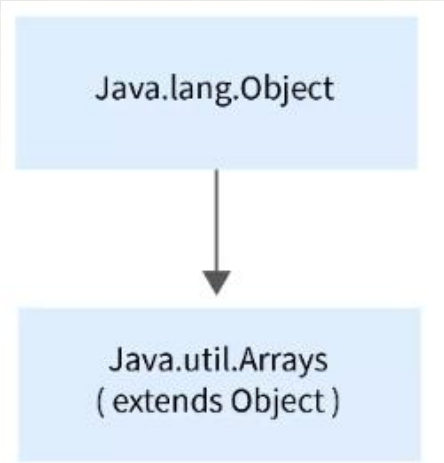
Arrays Class – Java

- This class contains various methods for manipulating arrays (such as sorting and searching). It also contains a static factory that allows arrays to be viewed as Lists.

Syntax: Class Declaration
public class Arrays extends Object

Methods in Java Array Class
The static methods of the Java Array class could be used to perform operations like:

- Filling the elements
- Sorting the elements
- Searching for the elements
- Converting the array elements to String
- Many more...



Methods – Java

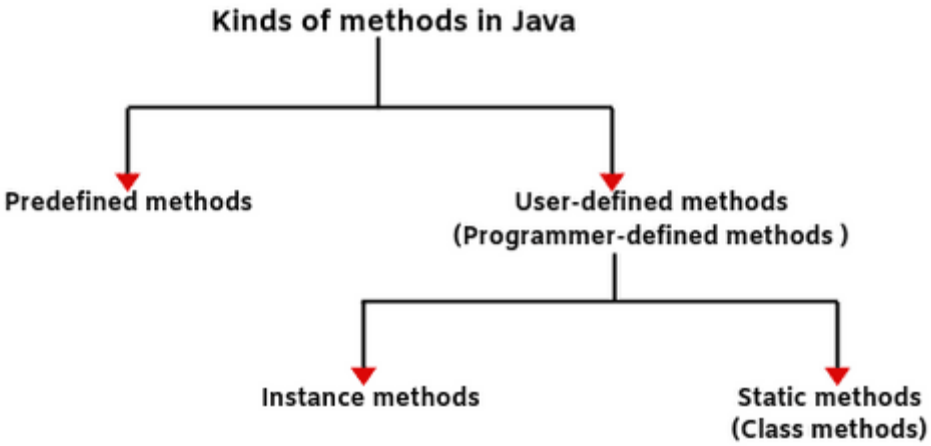


Fig: Basic kinds of methods in Java

Methods – Java

- In general, a method is a way to perform some task. Similarly, the method in Java is a collection of instructions that performs a specific task.
- It provides the reusability of code.

Categories of Built-in Methods

i) String Methods

ii) Number Methods

iii) Character Methods

iv) Array Methods etc…

String Methods – Java

compareTo() Method

- It compares two strings and supports a 3-way comparison.
- Result Criteria for 3-way comparison

If str1 = str2 then 0
If str1 > str2 then positive value
If str1 < str2 then negative value

Result Criteria for 2-way comparison

If str1 = str2 then true
If str1 (Greater than or Less than) str2 then false

String Methods – Java

Example:

```
class ex
{
        public static void main (String [] args)
        {
                String str1 = “selenium”;
                String str2 = “SELENIUM”;
                String str3 = “seleniuma”;
                String str4 = “selenium”;

                System.out.println(str1.compareTo(str2));        //Positive value
                System.out.println(str1.compareTo(str3));        //Negative value
                System.out.println(str1.compareTo(str4));        //0
        }
}
```

String Methods – Java

equals () Method
(It compares two strings and supports 2-way comparison)

Example:

```
class ex {
public static void main (String [] args){
        String str1 = “selenium”;
        String str2 = “SELENIUM”;
        String str3 = “selenium”;

        System.out.println(str1.equals(str2));//false
        System.out.println(str1.equals(str3));//true
        }
}
```

String Methods – Java

concat() Method (It concatenates two strings /Joins two strings)

```
class ex {
public static void main (String [] args){
                String str1 = “IT Companies”;
                String str2 = “List 2023”;
                System.out.println(str1.concat(str2));
                System.out.println(str1 + str2);
                }
}
```

Output

IT Companies List 2023
IT Companies List 2023

String Methods – Java

charAt() Method (Returns a character by index position)

```
class example
{
public static void main (String [] args)
{
        String str1 = “EAT MORE FOOD”;
        System.out.println(str1.charAt(1));
}
}
```

Output

A

String Methods – Java

equalsIgoreCase() Method

```
class example
{
  public static void main (String [] args)
  {
      String str1 = “Sachin”;
      String str2 = “SACHIN”;
      String str3 = “Kohli”;
      System.out.println(str1.equalsIgnoreCase(str2));
      System.out.println(str1.equalsIgnoreCase(str3));
   }
}
```

Output

True
False

String Methods – Java

toUpperCase () – Converts values to Upper case)

```
class ex
{
        public static void main (String [] args)
         {
                String str1 = “ASHWIN”;
                String str2 = “jadeja”;
                String str3 = “HARDIKpandya”;
                String str4 = “Sachin123”;

                System.out.println(str1.toUpperCase());
                System.out.println(str2.toUpperCase());
                System.out.println(str3.toUpperCase());
                System.out.println(str4.toUpperCase());
        }
    }
```

Output

```
ASHWIN
JADEJA
HARDIKPANDYA
SACHIN123
```

String Methods – Java

toLowerCase() -Converts values to Lower case

```
class example
{
public static void main (String [] args)
{
    String str1 = “ALEX”;
    String str2 = “suhan”;
    String str3 = “LAKSHan”;
    String str4 = “suhan123”;
    System.out.println(str1.toLowerCase());
    System.out.println(str2.toLowerCase());
    System.out.println(str3.toLowerCase());
    System.out.println(str4.toLowerCase());
}
```

Output

```
alex
suhan
lakshan
suhan123
```

String Methods – Java

trim() Method (Removes spaces from both sides of a String)

```
class example
{
   public static void main (String [] args)
   {
      String str1 = ”    HelloWorld   “;

      System.out.println(str1);
      System.out.println(str1.trim());
}
```

Output

HelloWorld

String Methods – Java

substring () Method (it is used to fetch the particular portion of the string)

```
class str
{
   public static void main (String [] args)
  {
   String str = “Welecome to Selenium Testing”;
   System.out.println(str.substring(12));
   System.out.println(str.substring(21));
   System.out.println(str.substring(12, 20));
   System.out.println(str.substring(9, 11));
   }
}
```

Output

Selenium Testing
Testing
Selenium
to

String Methods – Java

endsWith() -Ends with specified suffix

```
class example
{
  public static void main (String [] args)
  {
     String str = “Welcome to Selenium Testing”;
     System.out.println(str.endsWith(“Selenium Testing”));
     System.out.println(str.endsWith(“Testing”));//true
     System.out.println(str.endsWith(“Selenium”));//false
  }
}
```

Output

```
true
true
false
```

String Methods – Java

length() (returns string length)

```
class example
{
    public static void main (String [] args)
    {
        String str = “Selenium Testing”;
        String str2 = “Selenium”;
        System.out.println(str.length());
        System.out.println(str2.length());
}
```

Output

16
8

Number Methods – Java

compareTo() Method (Number, 3-way comparison)

```
class ex
{
   public static void main (String [] args)
   {
            // Integer class wraps a value of the primitive type int in an object
            //An object of type Integer contains a single field whose type is int.
        int x = 5;
        Integer a =x;
        System.out.println(a.compareTo(5));
        System.out.println(a.compareTo(6));
        System.out.println(a.compareTo(4));
}
```

Output

0
-1
1

Number Methods – Java

abs() -Returns absolute value

```
class Ab
{
public static void main (String [] args)
{
    double a =10.234;
    double b =-10.784;
    System.out.println(Math.abs(a));
    System.out.println(Math.abs(b));
}
```

Output

10.234
10.784

Number Methods – Java

round() -It rounds the value to nearest integer

```
class example
{
    public static void main (String [] args)
    {
        double a =10.234;
        double b =-10.784;
        double c =10.51;
        System.out.println(Math.round(a));
        System.out.println(Math.round(b));
        System.out.println(Math.round(c));
    }
}
```

Output

10
-11
11

Number Methods – Java

isLetter() – Checks weather the value is Alphabetic or not?

```
class Example
{
public static void main (String [] args)
{
        char a ='A';
        char b ='1';
        System.out.println(Character.isLetter(a));
        System.out.println(Character.isLetter(b));
        System.out.println(Character.isLetter('Z'));
        System.out.println(Character.isLetter('1'));
        System.out.println(Character.isLetter('*'));
}
```

Output

True
False
True
False
False

Number Methods – Java

isDigit() -Checks weather the value is Number or not?

```
class example {
public static void main (String [] args)
  {
            char a =’A’;
            char b =’1′;
            System.out.println(Character.isDigit(a));
            System.out.println(Character.isDigit(b));
            System.out.println(Character.isDigit(‘Z’));
            System.out.println(Character.isDigit(‘1’));
            System.out.println(Character.isDigit(‘*’));
  }
}
```

Output

False
True
False
True
False

Number Methods – Java

isLowerCase()-Checks weather the value is Lower case or not?

```
class example {
public static void main (String [] args){
        char a ='A';
        char b ='z';
        char c ='1';
        System.out.println(Character.isUpperCase(a));
        System.out.println(Character.isUpperCase(b));
        System.out.println(Character.isUpperCase(c));

        System.out.println(Character.isLowerCase(a));
        System.out.println(Character.isLowerCase(b));
        System.out.println(Character.isUpperCase(c));
    }
}
```

Output

```
true
false
false
false
true
false
```

Number Methods – Java

length -It returns length of the Array.

```
class Sample1
{
        public static void main (String [] args)
        {
                int [] array1 = {10, 20, 30, 40};
                System.out.println(array1.length);
        }
}
```

Output

4

Number Methods – Java

toString() -It prints an Array

```
Import java.util.Arrays;
class example
{
   public static void main (String [] args)
  {
                String [] array1 = {“Selenium”, “UFT”, “LoadRunner”, “RFT”};
                String str = Arrays.toString(array1);
                System.out.println(str);
     }
}
```

Output

[Selenium, UFT, LoadRunner, RFT]

Number Methods – Java

contains() – Checks if the Array contains a certain value or not?

```
class example
{
public static void main (String [] args)
 {
        String [] array1 = {“Selenium”, “UFT”, “LoadRunner”, “RFT”};
        boolean a = Arrays.asList(array1).contains(“UFT”);
        boolean b = Arrays.asList(array1).contains(“Java”);
        System.out.println(a);
        System.out.println(b);
  }
}
```

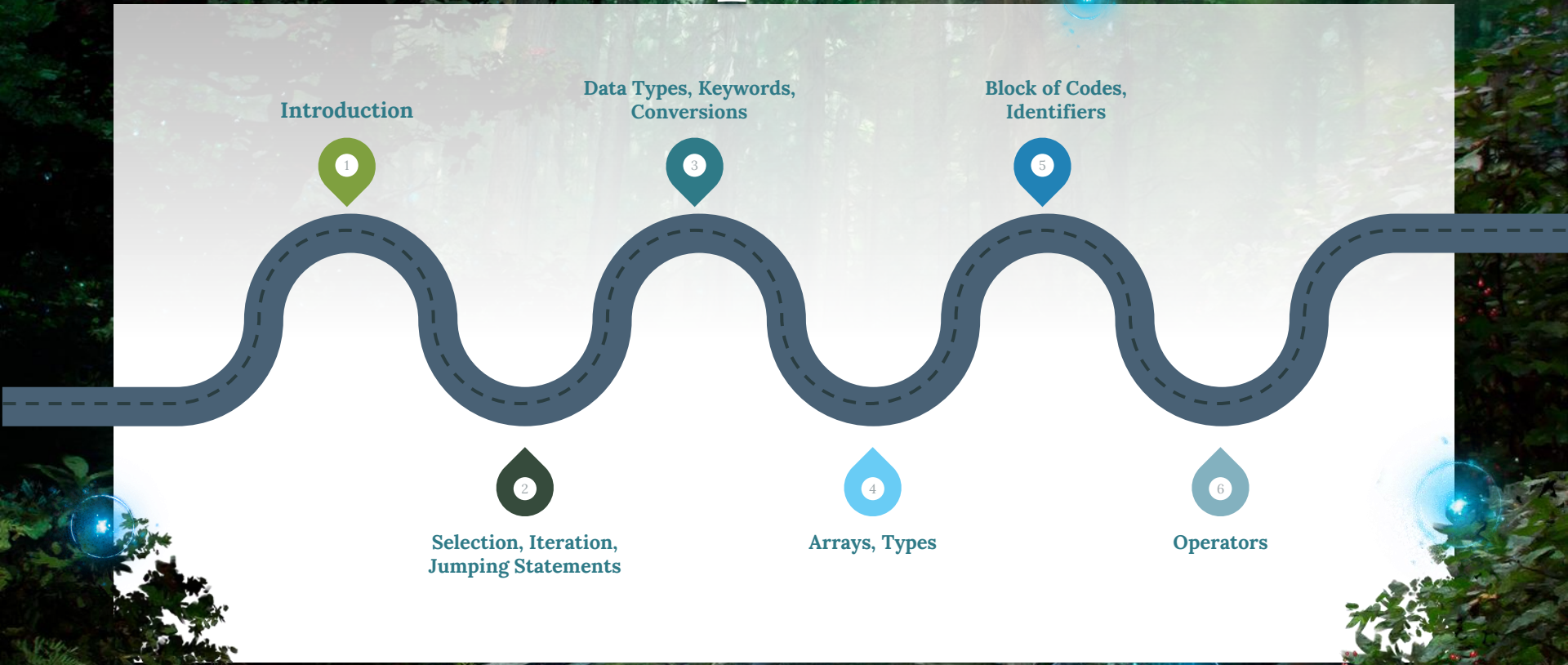
Output

True
False

References

- https://refreshjava.com/java/java-introduction
- https://www.quora.com/Is-it-possible-to-learn-Java-without-learning-C-C++
- https://www.javastudypoint.com/2019/12/java-control-statements.html
- https://stackify.com/top-java-software-errors/#:~:text=%E2%80%9CPossible%20loss%20of%20precision%E2%80%9D%20occurs,variable%20as%20a%20new%20type. Top Java Software Errors: 50 Common Java Errors and How to Avoid Them
- https://www.javatpoint.com/java-data-types
- https://courses.cs.washington.edu/courses/cse341/98au/java/jdk1.2beta4/docs/api/java/util/Arrays.html
- https://www.softwaretestinghelp.com/java-array-class-tutorial/
- https://www.scaler.com/topics/java/arrays-class-in-java/
- https://www.knowprogram.com/java/array-programs-in-java/
- https://javatpoint.com/operator-shifting

Roadmap for Unit 1



Roadmap for Java Programming

Unit I
Introduction, Looping and Operators

1

2

Unit – II
Classes and Objects, Constructors

Unit III
Inheritance, Interface, Exception Handling

3

4

Unit IV
Multthreading and Input/Output Files

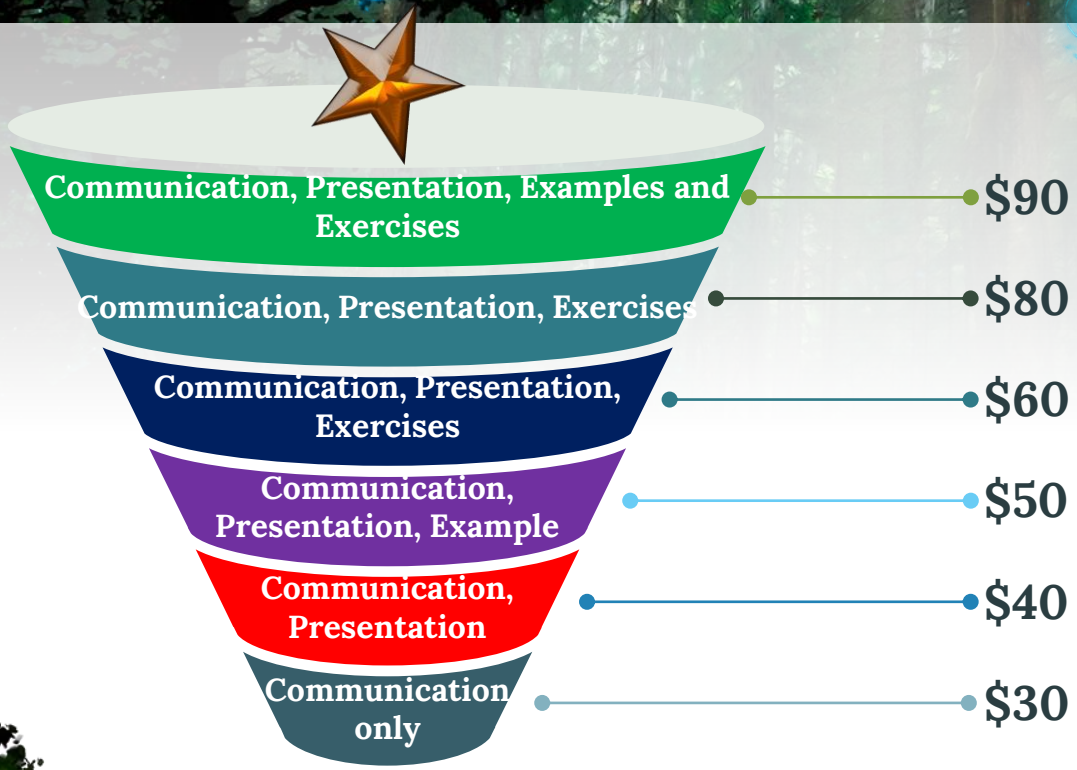
Unit V
JDBC, Swings JSP

5

6

Revision

Feedback


Communication, Presentation, Examples and Exercises
$90
Communication, Presentation, Exercises
$80
Communication, Presentation, Exercises
$60
Communication, Presentation, Example
$50
Communication, Presentation
$40
Communication only
$30

Questions

THANK
YOU!