

# Java in Hindi



[BccFalna.com](http://BccFalna.com)  
097994-55505

Kuldeep Chand

This EBook is not Just **Core Java**, but also includes some concepts of Advance Java like Basics of JDBC, **Event Driven Programming**, GUI development with AWT and Basics of Java Networking too.

In Java, all GUI development like *SWT/Swing, JavaFX* etc... are totally based on **AWT**. So, learning AWT helps very much in learning GUI Development using Java. So, in this EBook, I have covered GUI Development from and covered **AWT and Event Driven Programming** with Good Detail in Last Chapter. So that, after reading this EBook, you can start developing GUI Applications using Java easily.

Even Applets are out of market now, but I have included it frequently in this EBook to easily using and understanding GUI Development. Basics.

I have covered each Java Programming Concept with hundreds of example programs. So, it would be very easy to learn Java with this EBook.

# Java

## In Hindi



**Kuldeep Chand**

**BetaLab Computer Center  
Falna**

## **Programming Language JAVA in Hindi**

Copyright © 2011 by Kuldeep Chand

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Lead Editors: Kuldeep Chand

Distributed to the book trade worldwide by Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

e-mail [bccfalna@gmail.com](mailto:bccfalna@gmail.com)

or

visit <http://www.bccfalna.com>

For information on translations, please contact Betalab Computer Center, Behind of Vidhya Jyoti School, Falna Station Dist. Pali (Raj.) Pin 306116

Phone 97994-55505

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, the author shall not have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this book.

**This book is dedicated to those  
who really wants to be  
a  
PROFESSIONAL DEVELOPER**

**INDEX  
OF  
CONTENTS**

## Table of Contents

Java – Basics and Programming Fundamentals.....	13
Features of JAVA .....	17
Small and Simple .....	17
Object Oriented .....	17
Distributed .....	18
Compiled and Interpreted .....	18
Robust and Safe .....	19
Architecture Neutral / Platform Independent / Portable / Byte Coded.....	19
Garbage Collective .....	20
High Performance .....	20
Multithreaded and Interactive.....	20
Dynamic and Extensible .....	20
Java – Working .....	22
Java Platform.....	23
The Essentials: .....	24
Applets: .....	24
Networking: .....	24
Internationalization:.....	24
Security: .....	25
Software components: .....	25
Object serialization:.....	25
Java Database Connectivity (JDBC):.....	25
Program .....	25
Procedural Techniques and OOPS .....	27
The Object-Oriented Approach.....	29
Difference Between C++ and Java .....	31
Preprocessor.....	31
Pointers .....	32
Structure and Union .....	32
Functions .....	33
Multiple Inheritance.....	33
Strings.....	34
goto Statement.....	34
Operator Overloading .....	34
Automatic Type Casting.....	34
Variable Number of Arguments.....	35
Command Line Argument.....	35
Programming – The Basic Concept.....	36
System Software:.....	37
Application Software: .....	37
Computer Architecture .....	37
Hardware Programming .....	39
Software Programming .....	40
Language.....	40
Java Compiler (javac) .....	41
Java Interpreter (java).....	42
Structure of Java Programs.....	43
Documentation Section.....	43
Package Statements.....	43
Import Statements.....	44

Interface Statements.....	44
Main Method Class .....	44
Definition – The Applet and Application.....	44
Java - Applications.....	45
First Application in Java .....	45
Compiling Java Source File .....	46
Running Java Application .....	48
Anatomy of Java Application.....	48
Comments in Java .....	48
Java – Applet.....	51
Applet – Local and Remote.....	51
Clients and Servers.....	52
Difference – Applet and Application .....	53
Preparation – The Applet Writing .....	53
System Package – Predefined (Built-In) Library of Java Classes.....	54
Using – The System Packages .....	55
Keyword / Reserve Word .....	56
Building – The Applet Code .....	57
Applet Package – The Applet Class .....	57
OOPS and OOPL – The Definition .....	58
Problem – The Definition .....	58
Data – Value OR a Set of Values .....	59
Integer.....	59
Float.....	59
Character .....	59
Object – The Definition .....	59
Objects – Based on Problem.....	60
Abstraction – The Problem Simplifying Process.....	60
Abstract Data Type - Logical Representation of a Real World Object.....	61
Attributes – The Data Members of The Class.....	62
Behaviors – The Methods of The Class.....	62
Problem Design (OOPS) v/s Problem Implementation (OOPL) .....	63
Encapsulation – The Unitizing Process of Attributes and Behaviors.....	64
Class – A Logical Specification of Problem Related Object .....	65
Identifier Naming System .....	70
Java – Graphical User Interface and Graphics Management.....	72
Web Page – The Part of Website .....	72
HTML Tags for Web Pages .....	74
Comment Section .....	74
Head Section .....	75
Body Section.....	75
Adding Applet in HTML File.....	75
Applet Architecture – The Event Based GUI Application Program.....	76
First Applet in Java.....	82
GUI – The Event Driven Programming System .....	84
Components of an Event.....	86
Event Object .....	86
Event Source .....	87
Event Handler .....	87
Java Fundamentals – Core Concepts.....	91
Constants .....	91
Variables.....	93

Naming Constants and Variables – The Identifiers .....	93
“Java” Character set .....	95
“Java” Tokens .....	95
Keywords या Reserve Words .....	96
Identifiers .....	96
Literal .....	97
Variables .....	99
Operators .....	100
Precedence Of Operators .....	105
Data Types .....	106
Identifier (Variable / Constant) Declaration .....	107
Value Initialization .....	108
Garbage Values .....	109
Integer Data Types .....	109
Floating – Point Data Types .....	110
Boolean Data Type .....	111
Character Data Type .....	111
Variable Scope .....	118
Code Block .....	118
Type Casting .....	118
Arrays .....	121
Array Memory Allocation .....	122
Array Initialization .....	122
Strings .....	129
String Methods .....	131
StringBuffer Class .....	136
Command Line Arguments .....	138
Wrapper Classes .....	140
Control Statements .....	152
Program Control .....	153
Types Of Control Statement .....	153
Compound Statement or Block .....	154
Increment and Decrement .....	164
Loops .....	166
for Loop .....	167
Assignment Operators .....	170
Nesting of Loop .....	171
while Loop .....	176
do while Loop .....	178
Jump Statements .....	180
break Statement .....	180
continue Statement .....	182
return Statement .....	183
Drawing Graphics .....	183
Applet Canvas .....	183
Colors .....	185
Drawing Shapes .....	186
Drawing Line and Rectangles .....	186
Drawing Circles and Ellipses .....	187
Drawing Arcs .....	188
Drawing Polygons .....	188
Building Graphical User Interface .....	193



Abstract Windowing Toolkit.....	194
Components.....	194
Panel Class – The Panel Container.....	196
Push Button Control.....	196
Label Control.....	199
Rectangles and Windows.....	204
GUI Components – On The Absolute Placement.....	204
Handling Multiple-Button Events.....	206
Java OOPS – Object Oriented Programming Concept.....	212
Class and Objects.....	212
Attributes.....	216
Declaring Objects.....	224
new Operator – A Closer Look.....	225
Object Reference.....	226
Abstract Data Types.....	238
Adding Methods to Box Class.....	241
Methods Overloading.....	253
this Keyword.....	254
Automatic Garbage Collection.....	256
Finalize() Method.....	256
Methods Overloading.....	257
Constructors.....	261
Arguments Passing.....	266
Pass By Value.....	266
Pass By Reference.....	267
Access Controls.....	269
public and private Access Specifier.....	270
static Data Members and Methods.....	273
final Keyword.....	276
Nested and Inner Classes.....	276
GUI Application Of Java.....	278
Java Application – The Frame Class.....	279
Java Inheritance – Code Reusability.....	285
Reusability.....	285
Inheritance and Program Design.....	286
Composition: A “Has a” Relationship.....	286
Inheritance: A “Kind of” Relationship.....	287
Superclass and Subclass.....	289
Implementing Inheritance.....	291
Method Overriding.....	299
Constructors and Inheritance.....	302
Multilevel Hierarchy.....	311
Constructor Calling Convention.....	311
Dynamic Method Dispatch – The Run Time Polymorphism.....	312
Abstract Classes.....	317
Final Classes.....	322
Java Interfaces – Multiple Inheritances.....	326
Declaring Interfaces.....	327
Extending Interfaces.....	328
Implementing Interfaces.....	330
Java Exception – Error Handling.....	336
Compile Time Errors.....	336

Run Time Errors .....	337
Exceptions .....	338
Exception Types .....	341
<i>try</i> and <i>catch</i> Block – The Exception Handling Process .....	341
Multiple <i>catch</i> Blocks .....	343
Nested <i>try</i> Statement .....	347
The <i>throw</i> Keyword .....	347
The <i>throws</i> Keyword .....	349
The <i>finally</i> Code Block .....	349
Types of Exceptions – The Java Built – In Exceptions Classes .....	352
<i>java.lang</i> Exceptions .....	352
<i>java.io</i> Exceptions .....	354
<i>java.net</i> Exceptions .....	354
The <i>java.awt</i> Exceptions .....	354
The <i>java.util</i> Exceptions .....	355
Creating Own Exception Sub Class .....	355
Java Package – Code Reusability .....	358
Naming Conventions .....	359
Creating Packages .....	360
Java Multithreaded Programming .....	369
Java Thread Model .....	370
Thread Priorities .....	371
The Thread Class and the Runnable Interface .....	371
The Main Thread .....	371
Two Kinds of Threads .....	374
Converting a Class to a Thread .....	374
Deriving a Class From Thread .....	389
Thread Exception .....	393
Thread Scheduling – Setting Thread Priority .....	394
Establishing Thread Priority .....	395
Daemons .....	399
The ThreadGroup .....	400
Thread States – The Life Cycle of a Thread .....	402
NEWBORN State .....	403
RUNNABLE State .....	403
RUNNIG State .....	404
BLOCKED State .....	405
DEAD State .....	405
Synchronization .....	407
Deadlock .....	408
Java Networking .....	410
World Wide Web (WWW) Concepts .....	410
Distributed Programs .....	410
Protocol .....	411
IP Address .....	411
Host .....	412
Hostname .....	412
IETF (Internet Engineering Task Force) .....	413
Internet .....	413
Intranet .....	413
<i>Packet</i> .....	413
Protocol .....	413

Protocol Stack.....	414
Router .....	414
Sockets .....	414
Internet Protocols .....	414
TCP/IP Network Architecture .....	414
IPv4 And IPv6 .....	416
URL Class .....	417
Socket Class.....	417
Reliable .....	417
Ordered Stream .....	418
ServerSocket Class .....	418
DatagramSocket Class.....	418
Unreliable .....	418
Connectionless .....	419
Ports .....	419
Client/Server Technology Fundamentals .....	421
Client/Server Architecture .....	421
Client/Server Communication .....	422
Identifying a Computer.....	423
Testing A Program Without A Network .....	426
Socket Introduction.....	426
Creating A Simple Server and Client.....	428
Socket Transmission Modes .....	432
Reading From a Socket and Writing To a Socket.....	434
Working With URL .....	437
What Is a URL.....	437
Creating and Manipulating URL.....	438
Java RMI – Remote Method Invocation .....	444
RMI Applications.....	444
Advantage of Dynamic Code Loading .....	445
Remote Interfaces, Objects and Methods .....	445
Creating Distributed Application Using RMI .....	446
Design and Implement the components of Distributed Application.....	446
Compile Sources and Generate Stubs .....	447
Make Classes Network Accessible .....	447
Start The Application.....	447
Creating RMI Server .....	448
Designing A Remote Interface .....	448
RMI Technology .....	450
Java Database Management .....	452
Database Management System Software .....	452
Database Connectivity.....	453
ODBC Application Programming Interface ( ODBC API).....	453
JDBC Application Programming Interface ( JDBC API).....	454
JDBC Driver Manager.....	455
JDBC-ODBC Bridge.....	455
Installing The ODBC Driver .....	455
Connection to A Database.....	458
Querying A Database .....	460
Using The <i>Statement</i> Object.....	461
The Statement Object.....	461
The ResultSet Object.....	462

Using PreparedStatement Object .....	465
The PreparedStatement Object .....	465
Passing INPUT Parameter At Runtime .....	465
Java AWT – Abstract Windowing Toolkit .....	472
Event Driven Programming System .....	472
Components of an Event .....	473
Event Object .....	474
Event Source .....	474
Event Handler .....	475
Event Handling Mechanism – Double Approach .....	475
The JDK 1.02 Event Model .....	475
Delegation Event Handling Model .....	476
Event Classes .....	477
Event Listeners .....	479
Using The Delegation Event Model - Handling An Event .....	483
The ActionEvent Class .....	484
Handling Mouse Events .....	493
Handling Keyboard Events .....	499
Adapters .....	503
Inner Classes and Anonynouse Inner Classes for Simplifying Adapter Classes .....	509
Window Fundamentals of JAVA .....	514
Container Class .....	514
Panel Class .....	515
Window Class .....	515
Frame Class .....	515
Canvas Class .....	515
Frame Windows .....	516
Closing a Frame Window .....	517
User Interface Control Fundamentals .....	519
Adding and Removing Controls .....	520
Labels .....	520
Buttons .....	521
Check Boxes .....	522
Choice Controls .....	523
List Control .....	524
Scroll Bars .....	526
TextField Control .....	528
TextArea Control .....	530
CheckboxGroup Control .....	531
Layout Manager .....	531
Menu Bars and Menus .....	536
Dialog Boxes .....	542
FileDialog Class .....	547
Explicit Event Handling .....	549
Extending Buttons .....	550
Extending Checkbox .....	552
Fonts Handling In Java .....	554
Last but not Least. There is more .....	558

**JAVA**  
**BASICS AND PROGRAMMING**  
**FUNDAMENTALS**

## Java – Basics and Programming Fundamentals

आज हम देख सकते हैं Internet व Mobiles का कितना विस्तार हो चुका है। आज Internet इतना बढ़ चुका है कि दुनिया की जो भी जानकारी चाहिए, Internet पर उस जानकारी को प्राप्त किया जा सकता है। आज इस Internet की वजह से दुनिया बिल्कुल छोटी सी हो गई है। हम जब चाहें जिससे चाहें बात कर सकते हैं या Online Meeting कर सकते हैं। दुनिया की लगभग सारी चीजें आज Internet से जुड़ी हुई हैं। Internet पर आज हम केवल Texts ही नहीं पूरे Multimedia को देखते हैं, जिसमें Sound, Video, Animation, Graphics आदि जो कुछ भी हो सकता है, सब है।

लेकिन आज हम जिस तरह का Internet देख पा रहे हैं, कुछ समय पहले तक Internet ऐसा नहीं था। Multimedia की विभिन्न चीजों को Internet पर सम्भव बनाने में Java का बहुत बड़ा सहयोग रहा है। वास्तव में Java का विकास केवल Internet के लिए किया गया था, लेकिन आज इसका प्रयोग केवल Internet के WebPages बनाने के लिए ही नहीं होता है, बल्कि आज ये बड़े-बड़े Standalone Application Software व Distributed Application Develop करने की सबसे आसान व उपयोगी भाषा है। जितनी आसानी से हम Java का प्रयोग करके एक Distributed System Create कर सकते हैं, उतनी आसानी से किसी भी अन्य भाषा का प्रयोग करके हम Internet के लिए बड़े Software Develop नहीं कर सकते हैं।

आपने भी लोगों को ये कहते सुना होगा कि Computer Programming काफी कठिन काम है। ये हर किसी के बस की बात नहीं है। Computer Programmer बनने के लिए MCA, B-Level जैसे Degree Level Courses और हजारों रूपए के Hardware व Software की जरूरत होती है।

साथ ही वही Programmer बन सकता है जिसका दिमाग Computer की तरह काम करता हो यानी बहुत तेज हो। जो घण्टों किसी समस्या का समाधान प्राप्त करने के लिए धैर्यपूर्वक बैठ सकता हो। आदि-आदि।

एक अच्छा Programmer बनने के लिए ये सभी बातें जरूरी होती थीं लेकिन तब, जब Programmer किसी Assembly Language या Cobol, Pascal आदि जैसी किसी Language में Programming करना सीखता था। Java के साथ इससे बिल्कुल उल्टा है।

Java में Programming सीखना जितना आसान है, उतना शायद ही किसी Language को सीखना हो। इसमें बस कुछ Basic Concepts ध्यान हों, तो बहुत ही आसानी से कोई भी आवश्यकतानुसार Program बना सकता है और उसे Use कर सकता है। साथ ही वह अपने Application को Internet पर भी उतनी ही आसानी से चला सकता है जितना अपने स्वयं के Computer पर।

हम Programming को इतना Hard इसलिए मानते हैं क्योंकि ऐसा हमें अन्य Programmers ने कहा है। ये Programmers की Monopoly है ताकि उन्हें अच्छी Payment प्राप्त हो सके। यदि सभी लोग ऐसा कहने लगें, कि Programming बहुत ही सरल काम है, तो क्या Programmers को किसी Program के लिए उतने पैसे मिलेंगे जितने आज मिल रहे हैं।

शायद नहीं, इसीलिए सभी Programmers कहते हैं कि Programming सबसे कठिन काम है। हमारे देश में लोग Computer Programming को इसलिए कठिन समझते हैं, क्योंकि उन्हें उनकी भाषा में लिखे हुए Matter प्राप्त नहीं होते। दूसरी बात ये कि Computer को ठीक से तभी समझा जा सकता है, जब English पर अच्छी पकड़ हो। लेकिन ऐसा जरूरी नहीं है। Computer Programmer बनने के लिए अच्छी English उतनी जरूरी नहीं है जितनी तथ्यों को समझने व समझाने की योग्यता की जरूरत है।

Programming सीखने के लिए सबसे पहली चीज ये तय करनी होती है कि आखिर किस Language से Programming की शुरुआत की जाए। हालांकि सभी Languages में लगभग कुछ तथ्य समान ही होते हैं। जैसे Data Types, Operators, Conditional and Looping Statements आदि, लगभग सभी Languages में थोड़े बहुत अन्तर के अलावा समान ही होते हैं और उन्हें Use करने का तरीका कभी काफी हद तक सभी Languages में समान होता है।

यदि आपने “C” Language में या “C++” Language में थोड़ी बहुत Programming की है और Programming के Basic Concepts आपको Clear हैं, तो Java आपके लिए आगे बढ़ाने वाली सबसे अच्छी भाषा हो सकती है। हालांकि हर Programming Language की अपनी कुछ अलग विशेषता होती है जिसके आधार पर अलग-अलग Requirement के आधार पर अलग-अलग भाषा अधिक उपयोगी होती है। कुछ काम ऐसे भी होते हैं जो किसी Language में आसानी से Perform होते हैं और कुछ Languages में किसी भी तरह से उन कामों को नहीं किया जा सकता है।

उदाहरण के लिए यदि Fastly कोई GUI Application Software Develop करना हो, तो Microsoft Company का Visual Basic सबसे सरल Programming Language है। इसमें आज हजारों Software बन चुके हैं जिनका प्रयोग Personnel Use व Business Use दोनों स्थानों पर बहुत होता है। लेकिन Visual Basic Programs की कमी ये है कि इनकी Speed किसी अन्य Languages जैसे कि Borland C++ या Visual C++ में लिखे गए Programs की तुलना में कम होती है। इस Speed की कमी को तब महसूस किया जा सकता है, जब Program में बहुत सारे Graphics का प्रयोग किया गया हो।

जैसे कि यदि Visual Basic में Screen Saver या कोई Game Develop किया जाए तो इनकी Speed काफी कम होती है। इसलिए जो Professional Programmers होते हैं वे कभी भी Graphics Programming के लिए Visual Basic का प्रयोग नहीं करते हैं।

हालांकि Java को Visual Basic की तुलना में सीखना काफी कठिन है, लेकिन फिर भी Java को सीखना कई मायनों में काफी उपयोगी साबित होता है। Java की सबसे बड़ी विशेषता तो यही है, कि इसमें Develop किए गए Programs को हम World Wide Web पर Use कर सकते हैं। यदि आपने Internet Surfing की है तो आपने विभिन्न Websites पर कई Animations, Sounds आदि देखें व सुने होंगे। ये सभी काम Java में काफी आसानी से किए जा सकते हैं। यानी यदि आप कोई ऐसा Program बनाना चाहते हैं, जिसको Internet पर भी चलाया जा सकता है, जैसे कि Online Games, तो आपको Java की जरूरत होगी।

Java की दूसरी विशेषता ये है कि Java का Program एक विशेष तरीके से लिखा जाता है जिसमें हमें Java के सभी नियमों का पूरी तरह से पालन करना पड़ता है। यदि हम Java के किसी छोटे से नियम को भी Neglect करते हैं, तो एक छोटे से “Hello World” Program को Create करके Compile करने में भी हमें काफी परेशानियों व Errors का सामना करना पड़ सकता है।

Java को ऐसा इसलिए बनाया गया है ताकि जितनी भी Errors व परेशानियां आनी हैं, वे Program के Creation के समय ही आ जाएं, ताकि जब Program पूरी तरह से तैयार हो जाए, तब किसी प्रकार की परेशानी ना आए और Program Reliable, उपयोगी व Error Free हो। और वास्तव में जावा के Programs अन्य Languages की तुलना में काफी ज्यादा Reliable होते हैं।

कई अन्य Languages जैसे कि Visual Basic आदि में Program शुरू से अन्त तक कोई Error नहीं देता लेकिन किसी ना किसी जगह पर ऐसी Error Generate करता है, जिसका Solution करने में हमें उतना समय लग जाता है जितना उस Program को Create करने में नहीं लगता।

Java का विकास Sun Microsystems के एक Developer James Gosling ने किया था। उन्हें इसका विकास करने की जरूरत इसलिए पड़ी क्योंकि वे “C++” Language का प्रयोग करके एक Project बना रहे थे लेकिन उन्हें वह परिणाम प्राप्त नहीं हो पा रहा था जो वे चाहते थे। इसलिए उन्होंने स्वयं एक Language Develop की जिससे उनकी Requirement पूरी हो सके। इसी Language का नाम “Java” है।

Java को सीखना किसी भी अन्य Language को सीखने की तुलना में अधिक सरल है। ज्यादातर Languages एक दूसरे के लगभग समान ही हैं। इसलिए यदि एक Language में Mastery कर ली जाए तो बाकी की अन्य Languages में किसी Programmer को ज्यादा परेशानी नहीं आती है। वह आसानी से किसी भी Language में पकड़ बना लेता है। लेकिन इसके लिए जरूरी है कि उसे कम से कम एक Language में काफी जानकारी हो।

जो लोग पहले “C” या “C++” या दोनों सीख चुके हैं उन्हें Java को सीखने में कोई परेशानी नहीं आती है बल्कि वे उन लोगों की तुलना में ज्यादा जल्दी से Java को सीख लेते हैं और Java पर पकड़ बना लेते हैं, जिन्होंने “C” या “C++” नहीं सीखी है। अगर हम ऐसा कहें कि Java “C” व “C++” का मिलाजुला रूप है और Java में से उन चीजों को हटा दिया गया है, जिनको “C” व “C++” Language में सीखने में परेशानी आती थी, तो गलत नहीं होगा।

लेकिन इसका मतलब ये नहीं है कि Java को सीखने से पहले “C” व “C++” को सीखना जरूरी है। हालांकि यदि पहले “C” व “C++” सीखा जाए तो Java को समझना व सीखना सरल होता है लेकिन फिर भी हम Java से Programming सीखना शुरू कर सकते हैं। ये अपने आप में ही एक पूर्ण Language है। Java सीखने के बाद भी किसी भी अन्य Language को उतनी ही आसानी से सीखा जा सकता है जितनी आसानी से किसी और Language को सीखने के बाद Java को सीखा जाता है।

High Level Programming Languages के विकास की यदि बात करें, तो UNIX Operating System के लिए एक भाषा का विकास किया गया था, जिसका नाम “C” Language दिया गया। इस भाषा का विकास मुख्य रूप से Operating System UNIX को Develop करने के लिए किया गया था। UNIX Operating System Develop हो जाने के बाद UNIX Operating System के लिए Applications Software को Develop किया जाने लगा।

चूंकि “B” Language का विकास एक System Software को Develop करने के लिए किया गया था, इसलिए विभिन्न Programmers को इस Language में UNIX के लिए Application Software लिखने में परेशानी आती थी। इसलिए इस “B” Language को और सरल बनाया गया ताकि Programmers इस Language में Application Programs Develop कर सकें। इस Developed Language को “C” Language नाम दिया गया।

“C” Language शुरुआत में काफी उपयोगी साबित हुई लेकिन जिस तरह से हर चीज में विकास होता है, उसी तरह से Computer Technology में भी विकास हुआ। धीरे-धीरे Application Software इतने बड़े व जटिल होने लगे, कि “C” Language में Develop किए गए Programs को Manage व Maintain करना काफी कठिन हो गया। साथ ही जैसे-जैसे समय बीतता गया, Software की जटिलता भी बढ़ती गई।

इसलिए एक बार फिर Programmers को ये महसूस होने लगा कि उन्हें कुछ और अधिक सरल तरीके की जरूरत है, जिससे वे बड़े व जटिल Programs को Handle कर सकें। ये नया तरीका भी जरूरत के अनुसार Develop किया गया और इस तरीके को Object Oriented Concept कहा गया। इस Object Oriented Concept को ध्यान में रख कर Programming language “C” में



फिर विकास किया गया और इस विकास का परिणाम “C++” Programming Language के रूप में प्राप्त हुआ।

हालांकि आज Java का जिस उद्देश्य के लिए ज्यादातर प्रयोग किया जा रहा है और जावा जिस प्रकार की Programming के लिए जानी जाती है, वास्तव में Java का विकास इसके लिए नहीं किया गया था। जावा का विकास General Electronic Equipments को अधिक समझदार बनाने के लिए किया जा रहा था, ताकि विभिन्न प्रकार के Equipments को Artificial Intelligence प्रदान की जा सके। हालांकि ऐसा तो नहीं हो सका, लेकिन जावा एक Dynamic Internet Programming Language के रूप में काम आने लगी।

Java का विकास करने वाले लोग जिस Project पर काम कर रहे थे, वे उसमें “C++” Language का प्रयोग कर रहे थे, जो कि “C” Language का ही विकसित रूप है। लेकिन वे जो करना चाहते थे, वैसा “C++” के प्रयोग से नहीं कर पा रहे थे। इसलिए उन्होंने एक नई Language Develop की। इस Language को उन्होंने “C” व “C++” के आधार पर ही Develop किया है। वे Java को एक बहुत ही सरल Language बनाना चाहते थे, इसलिए उन्होंने “C” व “C++” की सभी आसान Concepts को ज्यों का त्यों उपयोग में लिया और जटिल Concepts को छोड़ दिया।

उन्होंने Java Language के Programming Syntax को भी लगभग वैसे ही उपयोग में लिया जिस तरह से “C” व “C++” में लिया जाता है। साथ ही उन्होंने कई अन्य Languages के Concepts का भी प्रयोग जावा में किया ताकि इसमें किसी भी प्रकार का Software आसानी से बनाया जा सके और Software पूरी तरह से विश्वसनीय बने।

इस तरह से Java केवल “C” व “C++” का Modified Version ही नहीं है, बल्कि कई अन्य Languages के Concepts पर आधारित एक पूर्ण Programming Language है। हालांकि इसके ज्यादातर Syntax व Coding Procedures “C” व “C++” Language के अनुसार हैं, इसलिए इसे “C++” Language का Modified Version भी कहा जा सकता है।

जैसे-जैसे जरूरत बढ़ती जाती है और जरूरत का स्वरूप बदलता जाता है, वैसे-वैसे Programming Languages को भी Develop करना जरूरी हो जाता है, ताकि वर्तमान की विभिन्न जरूरतों को पूरा किया जा सके। इसी तथ्य पर अब जावा से आगे की Language को Microsoft Company ने Develop किया है। इस Language का नाम “C#” (CSharp) है। इस Language में “C”, “C++” व Java तीनों Languages की विभिन्न विशेषताओं को Include किया गया है। Microsoft इस Language में Software Development के लिए पूरा IDE प्रदान करता है, जिसमें आज की जरूरत के अनुसार विभिन्न कामों को किया जा सकता है।

लेकिन इसका मतलब ये नहीं है कि Java अब पुरानी हो चुकी है। आज भी Java का Market में अपना एक अलग व महत्वपूर्ण स्थान है और Java को सीखे व समझे बिना, अगली Generation की Languages को समझना काफी मुश्किल है।

हालांकि जावा का विकास जिस काम के लिए किया जा रहा था, उस काम के लिए जावा उपयोगी नहीं बन पाया। लेकिन जब जावा के Developers ने देखा कि इस Language का प्रयोग Internet की Interactive Programming में काफी उपयोगी साबित हो सकता है, तब उन्होंने इस Language को Internet के लिए Develop करना शुरू किया। वे जिस Platform Independent Equipment Technology पर काम कर रहे थे, वह तकनीक Internet के लिए उपयोगी साबित हो गई।

## Features of JAVA

Java केवल एक Programming Language ही नहीं है बल्कि ये एक Platform भी है। जब Sun Microsystems ने November 1995 में Java को दुनिया से परिचित करवाया तब Company के Cofounder Bill Joy ने Java की निम्न परिभाषा दी थी कि

Java एक Small, Simple, Safe, Object-Oriented, Interpreted या Dynamically Optimized, Byte-Coded, Architecture-Neutral, Garbage-Collected, Multithreaded Programming Language है जिसमें Distributed, Dynamically Extensible Programs लिखने के लिए एक Strongly Typed Exception-Handling Mechanism है। जावा के इन्हीं गुणों को जावा के **Features** भी कहते हैं।

## Small and Simple

Java एक छोटी और सरल भाषा है जिसे आसानी से सीखा जा सकता है। जावा को इस तरह से Design किया गया है कि इसे कोई भी Programmer आसानी से सीख सके और Computer Programming के Internal Functionality को जाने बिना भी ज्यादा से ज्यादा Efficient Program Develop कर सके। यदि किसी Programmer को किसी भी Programming Language का थोड़ा भी ज्ञान है, तो वह बहुत ही आसानी से व जल्दी से Window Based Application व Internet Based Distributed Application (Applets) Develop करना सीख सकता है।

जब जावा को पहली बार Release किया गया था, तब वह काफी छोटी भाषा थी। लेकिन आज ये काफी बड़ी भाषा बन चुकी है और सभी प्रकार के Applications को Efficiently Develop करने में सक्षम है। ये Language C/C++, Simula, Ada जैसी कई अन्य Languages से प्रेरित है, लेकिन इसकी ज्यादातर Coding C++ Language के समान ही है। इसलिए किसी C/C++ Programmer को जावा सीखने में कोई कठिनाई नहीं होती है।

इस Language में C व C++ के अच्छे Features को Use कर लिया गया है जबकि इन Languages के Confusing तथा Typical Features को छोड़ कर उनके स्थान पर अधिक सरल Concepts को Include कर लिया गया है। जैसे C++ के Operator Overloading व Pointer जैसे Concepts को जावा में छोड़ दिया गया है, जबकि Multithreading जैसी Advance Technique को Add कर लिया गया है।

## Object Oriented

Java में हर चीज Object व Class के रूप में परिभाषित है, जिसे Object Oriented Programming Concept कहा जाता है। OOPS हमें Abstraction and Encapsulation, Polymorphism और Inheritance जैसे Features प्रदान करता है, जिससे हम एक समस्या को उसी तरह से Computer में Logically Organize कर सकते हैं, जिस तरह से समस्या Real World में Actually या Physically Organized रहती है। जावा में बहुत सारी जरूरी Classes पहले से ही Packages के Form में हमें प्राप्त होती है, जिन्हें बिना Rewrite किए हम ज्यों का त्यों Use कर सकते हैं।

## Distributed

Java के Programs Network पर यानी Web Pages पर भी Execute होते हैं। इसलिए इसे Distributed Language कहा जाता है। Distribution का मतलब ये होता है कि Java के Program किसी भी Platform पर Run हो सकते हैं।

हम जानते हैं कि आज कई तरह के Operating Systems उपलब्ध हैं और अलग-अलग लोग अपनी जरूरत व इच्छा के अनुसार अलग-अलग Operating Systems का प्रयोग करते हैं। कोई Windows Operating System Use करता है तो कोई Linux तो कोई MacOS या OS/2 Use करता है। ये सभी अलग-अलग Platform कहलाते हैं।

यदि हम Windows Based Computer पर कोई Program "C" या Visual Basic जैसी भाषा में Create करते हैं, तो वे Program उन सभी Computers पर आसानी से Run होते हैं जो Windows को Use करते हैं।

लेकिन यदि इन्हीं Programs को Linux या MacOS पर Execute करने की कोशिश की जाए तो ये Program उस Operating System पर Execute नहीं होते। इन Platforms के लिए Program को वापस इन्हीं Platform वाले Computers पर Compile करना पड़ता है। जबकि Java के साथ ऐसा नहीं है।

जावा में हम किसी भी Platform पर Program Create करके Compile करें, वे Program सभी अन्य Platforms पर समान रूप से Execute होते हैं। यानी Java के Programs को विभिन्न Platforms पर Distribute किया जा सकता है। इसलिए Java को Distributed Language कहा जाता है।

जावा को इस प्रकार से Design किया गया है कि हम इसमें ऐसे Applications Develop कर सकें, जो Internet पर चल सकें। इस Language में ये Ability है कि ये Data व Program दोनों को Internet पर विभिन्न Computers पर Share कर सकता है। जावा Applications Remote Objects को भी उतनी ही आसानी से Access व Open कर सकते हैं, जितनी आसानी से वे Local Computer के Objects को Open व Access करते हैं। जावा ऐसी Networking की सुविधा प्रदान करता है कि विभिन्न Remote Locations पर स्थित विभिन्न Programmers एक ही Single Project पर समान समय पर एक साथ काम कर सकते हैं।

## Compiled and Interpreted

ज्यादातर अन्य Languages के Programs या तो Compile होते हैं या फिर Interpreted होते हैं। लेकिन Java के Programs Compile भी होते हैं और Interpreted भी। Java के Programs को सबसे पहले Compile किया जाता है। Java के Program Compile होने के बाद सीधे ही Machine Language में Convert नहीं होते हैं, बल्कि ये Source Code व Machine Code के बीच की स्थिति में Convert होते हैं जिसे **Bytecodes** कहा जाता है।

इन Bytecodes को जब किसी भी Platform पर Run करना होता है तब ये Bytecodes उस Computer के Platform के अनुसार Interpreted हो कर पूरी तरह से उस Machine के अनुसार Machine Code में Convert होते हैं और उस Platform पर Execute हो सकते हैं।

## Robust and Safe

Java के Programs में Errors आने की सम्भावना अन्य Languages की तुलना में बिल्कुल कम होती है। इसलिए Java के Programs को Robust कहा जाता है। इसके Compiler में विभिन्न प्रकार से Generate होने वाली Errors को Handle करने के लिए कई Built-In तरीके Develop कर दिए गए हैं और जावा को इस तरह से Design किया गया है, कि एक बार सही तरीके से Compiled Program में कभी भी Error आने की सम्भावना नहीं रहती है। जितनी भी Errors आनी होती हैं, वे सभी Program Development व Testing के समय ही आ जाती हैं, जिन्हें Handle कर लिया जाता है।

इसमें Compile Time व Runtime दोनों स्थानों पर विभिन्न प्रकार के Errors के लिए विभिन्न Data Types की Checking होती है। विभिन्न प्रकार के Objects द्वारा ली जाने वाली Memory को ये स्वयं ही Release कर देता है, जिससे हमें इस बात की चिन्ता करने की जरूरत नहीं होती है, कि हमने सभी Unrequited Objects को Destroy करके उनकी Memory को Release किया या नहीं।

जावा में Exception Handling के लिए भी सुविधा प्रदान की गई है, जिसका प्रयोग हम Serious Errors को Trap करने व उन्हें Solve करने के लिए कर सकते हैं, जिससे हमारे Program की और सुरक्षा हो जाती है।

जब हम Internet की बात करते हैं, तब Security काफी मायना रखती है। जावा स्वयं ही विभिन्न प्रकार के Memory Management व Memory Access से सम्बंधित काम करता है, इसलिए ये कभी भी Memory व उसमें Stored Data को गलत तरीके से Access करने की छूट नहीं देता है।

इस वजह से Applet द्वारा किसी Computer में Virus आने की सम्भावना ही नहीं होती है। क्योंकि जावा में Pointers की सुविधा नहीं है जो Directly Memory को Access कर सके, इसलिए हम किसी भी Computer की Memory को Directly Access नहीं कर सकते हैं। साथ ही जावा Applets कभी किसी Client Computer के Resources को Access नहीं करते हैं, इसलिए जावा Applets कभी भी Clients के Computer या उसके Data को नुकसान नहीं पहुंचा सकते हैं।

## Architecture Neutral / Platform Independent / Portable / Byte Coded

Java के Bytecodes विभिन्न प्रकार के Processors व Operating Systems पर समान रूप से Run हो सकते हैं। इसलिए इसे Architecture Neutral or Portable कहा जाता है। जावा के Programs को केवल एक ही बार Develop करना होता है। एक बार इसे Develop करने के बाद इसे किसी भी Computer पर किसी भी Platform पर Run किया जा सकता है।

यदि Operating System, System Resources या Processor में Change किया जाता है, तब भी हमें जावा के Program में किसी प्रकार का Change करने की जरूरत नहीं होती है। यही जावा के सबसे ज्यादा Popular होने की मुख्य वजह है, जिससे हम जावा का प्रयोग Internet Programming के लिए करके World Wide Web पर Run होने वाले Applications Develop करते हैं और विभिन्न Computers को आपस में Interconnected करते हुए World Wide Web पर काम करने के लिए ऐसे Programs को Use करते हैं।

हम जावा Applet को Remote Computer से Download कर सकते हैं और फिर उसे अपने Computer पर Run कर सकते हैं। इस प्रकार की सुविधा होने से एक User को उसके घर पर ही विभिन्न प्रकार के Applications व Applets प्राप्त हो जाते हैं, जिन्हें वह Use करना चाहता है।

जावा दो तरीकों से Portable होता है: एक तो जावा Compiler Byte Codes Instructions Generate करता है, जिसे किसी भी Computer पर Implement किया जा सकता है और दूसरा जावा के Primitive या Basic Data Types Machine पर निर्भर नहीं होते हैं बल्कि जावा Platform पर निर्भर होते हैं। यानी किसी भी Compute पर जावा के सभी Data Types की Size समान होती है, चाहे हम Pentium Computer पर जावा Program को Use करें, चाहे AMD पर।

## Garbage Collective

Java एक Programmer को Memory Manage करने की सुविधा प्रदान नहीं करता है बल्कि जरूरत के अनुसार स्वयं ही Memory Management करता है। इसलिए Programmer द्वारा Memory Management के समय किसी दूसरे Data को नुकसान पहुंचाने की सम्भावना नहीं होती है। इसलिए ये Language “C” व “C++” जैसी भाषाओं की तुलना में अधिक सुरक्षित या Secure Language है।

## High Performance

Java के Program Interpreted Mode में Run होते हैं लेकिन फिर भी अन्य Interpreted Based Languages की तुलना में Java की Speed व Performance बहुत अच्छी होती है। इसलिए इसे High Performance Language कहा जाता है।

## Multithreaded and Interactive

Java ये सुविधा प्रदान करता है कि एक ही Software Program के विभिन्न भागों को एक ही समय में एक साथ Run किया जा सकता है। इसलिए इसे Multithreaded Language कहा जाता है।

उदाहरण के लिए मानलो कि हम किसी Program से Audio Sound तो सुन ही रहे हैं, साथ ही उसी Program में Scroll Bars को भी Use कर रहे हैं। किसी Window में एक तरफ कोई Movie Clip तो Play हो ही रहा है, साथ ही हम किसी अन्य Movie Clip को Open करने के लिए Open Dialog Box में किसी दुसरी Movie Clip को भी खोज रहे हैं। इस तरह से एक ही Program के विभिन्न हिस्सों का एक ही समय में एक साथ Run होना Multithreaded Concept के कारण ही सम्भव होता है।

## Dynamic and Extensible

Java में एक ही Program के कई Versions को एक साथ Maintain किया जा सकता है। इसलिए इसे Dynamic Language भी कहा जाता है। यानी जावा एक Dynamic Language है। जावा में किसी Program के लिए Required Classes जावा के Program के Run होते समय उसमें Link हो सकती है और जैसे ही उस Class का काम समाप्त होता है, वह Class

स्वयं ही Memory से Release हो जाती है। इस प्रक्रिया को Dynamic Process कहा जाता है।

जावा एक Query द्वारा ये भी Determine कर सकता है कि Program के Run Time में उससे कौनसी Class Link हो रही है। साथ ही वह Program के Run Time में भी किसी भी Dynamic Class या Dynamic Link Library से Link हो सकता है और Run Time सुविधाओं को प्राप्त कर सकता है। इस प्रक्रिया को Dynamic Linking भी कहते हैं।

जावा हमें अन्य Languages के Methods को भी जावा में Use करने की सुविधा प्रदान करता है। इन Methods को Native Methods कहते हैं और ये Program के Run Time में Dynamically Link हो कर अपना काम करते हैं। यानी हम जावा में अन्य Languages की सुविधाओं को Use करके जावा के Program की क्षमताओं को बढ़ा सकते हैं या Extend कर सकते हैं। इसी प्रक्रिया को जावा की Extensibility कहते हैं।

Java के Programs कई प्रकार के होते हैं, उनमें से कुछ निम्नानुसार हैं :

## Applications

ये ऐसे Programs होते हैं जिन्हें Execute होने के लिए किसी Browser की जरूरत नहीं होती है। ये Stand Alone होते हैं और किसी भी Computer पर Run हो सकते हैं। इन्हें Command Prompt पर Run किया जा सकता है।

## Applets

ये ऐसे Programs होते हैं जिन्हें Run होने के लिए Browser की जरूरत होती है। ये Programs Web Pages पर Run होते हैं। एक Applet Program कभी भी किसी Local Machine के Resources को Access नहीं करता है।

## Servlet

ये Programs Web Servers की Functionality को बढ़ाने के लिए लिखे जाते हैं। सामान्यतः इनका कोई GUI नहीं होता है।

## Packages

ये Java की Classes का एक Collection होता है जिसे किसी भी अन्य Java Program में आसानी से Reuse किया जा सकता है।

Object Oriented Concept Programming करने का एक असाधारण लेकिन बहुत ही Powerful तरीका है। OOP में एक Computer Program को Objects के एक ऐसे Group के रूप में परिभाषित किया जाता है जिसमें सभी Objects आपस में Interact कर सकते हैं।

यानी सभी Objects अपनी सूचनाओं का एक दूसरे के बीच Transaction कर सकते हैं। OOPS में दुनिया की हर चीज को Object माना जाता है। मानलो कि एक Worker Object Money Object को CompanyFunds Object से लेता है और अपने स्वयं के BankAccount Object में जमा करवाता है। यदि दूसरा कोई Worker Object DoublecheckFund Object को Use करता है तो Polish Object को बुलाया जाता है।

Java Program की सबसे बड़ी यदि कोई विशेषता है, तो वह ये है कि Java के Program को World Wide Web Pages पर Execute किया जा सकता है। इन Programs को **Applets** कहते हैं। Java से पहले HTML Format में ही विभिन्न Web Pages को लिखा जाता था। ये Web Pages ऐसे होते थे, जिसमें जिस Page को देखना हो उसके Hyperlink पर Click करो और दूसरा Page देखना है तो उसके Hyperlink पर Click करो और आगे से आगे बढ़ते जाओ।

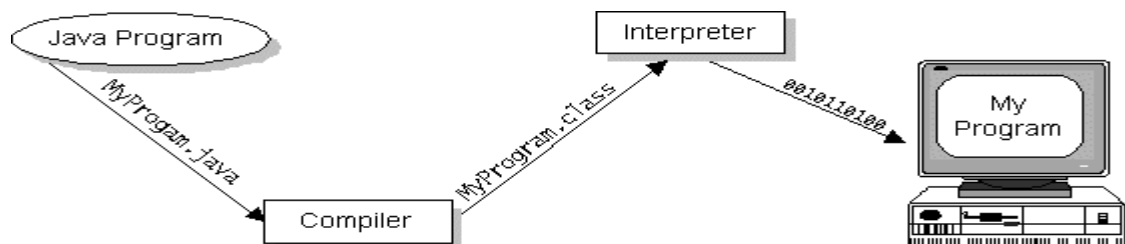
जबकि Java Applet जो कि Web Pages पर Run होते हैं, अधिक अच्छा अनुभव प्रदान करता है। इसमें User के Response के अनुसार Web Page Dynamically Update होता है। जैसे आज हम कई TV Channel पर देखते हैं जहां कोई सवाल पूछा जाता है और लोग SMS भेज कर अपना पक्ष बताते हैं। जैसे-जैसे लोग SMS भेजते रहते हैं, SMS की संख्या भी बदलती रहती है। ये काम Dynamically होता है जो कि Java के कारण ही सम्भव हुआ है।

आज User Internet पर उपलब्ध विभिन्न प्रकार के Web Pages से Java के कारण ही Interact कर सकता है। यदि इसका उदाहरण लेना चाहें, तो Share Market का सारा काम Online होता है। User जब चाहे अपने Account की Information को प्राप्त कर सकता है। यदि वह किसी Company का Share खरीदना चाहता है, तो वह Online खरीद सकता है। जैसे ही वह Share खरीदता है, उस Company के Buyers की संख्या बढ़ जाती है। इसी तरह से यदि Share को बेचा जाता है, तो Company के Sellers की संख्या बढ़ जाती है। ये जो परिवर्तन Web Pages के Data में होता है, वह Dynamic परिवर्तन कहलाता है। यानी Web Page Dynamically या Run Time में User के Interaction से Update होता है। इस प्रकार की Secure Dynamic व Online सुविधा हमें Java के कारण ही प्राप्त हो पा रही है।

हालांकि Web Based Programs की वजह से Java अधिक महत्वपूर्ण लगती है लेकिन ये एक General Purpose Language भी है जिसका प्रयोग सभी तरह के Programs को Develop करने में होता है। आज हम Mobile के जितने भी Software देखते हैं उनमें से ज्यादातर Java Based हैं। Mobile में जो Games Run होते हैं वे ज्यादातर Java में Develop किए जाते हैं।

## Java – Working

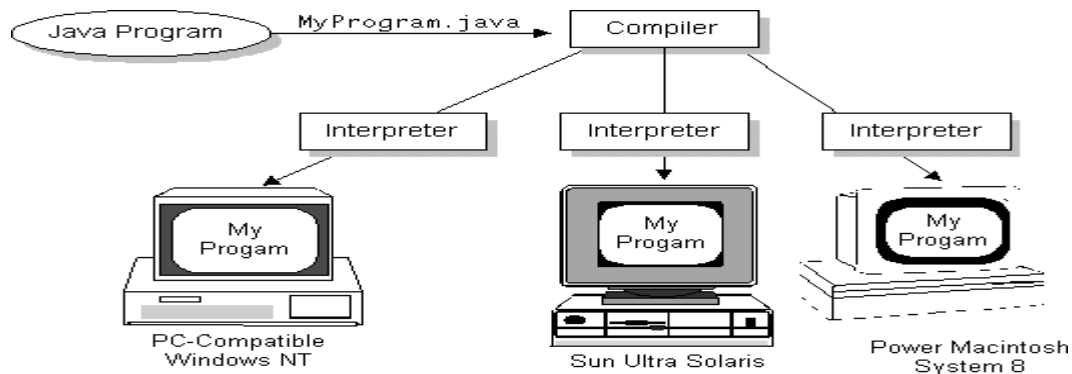
जब Java के किसी Program को Compile किया जाता है तब Java का Program पूरी तरह से Machine Language में Convert नहीं होता है बल्कि एक Intermediate Language में Convert होता है, जिसे Java Bytecodes कहा जाता है। ये Codes Platform Independent होते हैं, इसलिए इन्हें किसी भी Operating System व किसी भी Processor पर चलाया जा सकता है। Java के Program की Compilation केवल एक ही बार होती है लेकिन जितनी बार भी Java के Program को चलाया जाता है, हर बार उस Program का Interpretation होता है। इसे हम निम्न चित्र द्वारा समझ सकते हैं-



Java Bytecodes को हम Java Virtual Machine (Java VM) के लिए Machine Codes मान सकते हैं। हर Java Interpreter चाहे वह Java Development Tool हो या कोई Browser जो कि Java Applets को Run करता हो, Java Virtual Machine का ही Implementation है। Java Virtual Machine को Hardware में भी Implement किया जा सकता है, जिसका परिणाम आज के Mobile System Software हैं।

Java Bytecodes हमें ये सुविधा देते हैं कि हम Java के Program को एक बार Compile करें और कहीं भी Run करें। हम किसी Java Program को किसी भी उस Computer पर Compile कर सकते हैं जिस पर Java Compiler हो। फिर उस Java Program के Bytecodes को किसी भी उस Computer पर Run किया जा सकता है जिस पर Java VM हो।

उदाहरण के लिए एक ही Java Program Windows, OS/2 MacOS NT, Macintosh आदि विभिन्न Platform पर Execute हो सकते हैं।

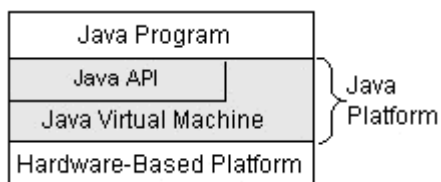


## Java Platform

Platform एक एक ऐसा Software या Hardware Environment होता है जिसमें कोई Program Run होता है। Java Platform कई अन्य Platforms से अलग है। Java Platform एक Software Platform है, जो सभी अन्य Hardware Based Platform के Top पर यानी ऊपर Run होता है। ज्यादातर अन्य Platforms Hardware व Operating System का Combination होते हैं।

Java Platform के दो Components हैं। पहला है **Java Virtual Machine (Java VM)** जिसके बारे में हम जान चुके हैं। ये Java Platform का Base या आधार है और विभिन्न Hardware Base Platform के ऊपर रहता है। दूसरा होता है Java Application Programming Interface (Java API) जिसके बारे में हम अब जानेंगे।

Java API Ready Made Software Components का एक बहुत बड़ा Collection है जो कि Programmer को GUI (Graphical User Interface) जैसी कई उपयोगी क्षमताएं प्रदान करता है। Java API को Related Components की Libraries के रूप में Group कर दिया गया है। इन विभिन्न Related Components के Group को ही **Packages** कहते हैं। एक Java Program को हम निम्न चित्रानुसार दर्शा सकते हैं:



जब एक Java Program को किसी Computer पर Execute किया जाता है तो Java Program व Hardware Based Platform के बीच Java API व Java Virtual Machine की Layer होती हैं जो Java के Program को Hardware Dependencies से अलग करती हैं।



यानी इन दोनों की वजह से Java का कोई Program किसी भी Computer के Hardware पर निर्भर नहीं होता है। एक Platform Independent Environment के रूप में Java का Program अन्य Native Codes Programs की तुलना में कुछ धीमा होता है। लेकिन फिर भी अच्छे Compilers, Java के साथ अच्छी तरह से Tune होने वाले Interpreters और Bytecodes Compilers की वजह से Java की Performance को Native Code की Performance के आस-पास लाया जा सकता है और वो भी जावा की सभी विशेषताओं के साथ।

Java Programs का सबसे अधिक जाना पहचाना यदि कोई रूप है तो वह Java Applets का है। एक Applet भी एक Java Program ही होता है लेकिन इसकी विशेषता ये है कि ये किसी Java Enabled Browser जैसे कि Internet Explorer, Google Chrome, Firefox, Safari, Opera आदि में Run होता है, स्वतंत्र रूप से ये Run नहीं हो सकता। जबकि Java Application Standalone Run हो सकते हैं।

Applets Application के समान ही होते हैं। ऐसा भी नहीं है कि Java का प्रयोग केवल Web Pages Applications लिखने के लिए ही किया जा सकता है। बल्कि Java एक Powerful Software Platform और General Purpose High Level Programming Language भी है।

Java के सबसे Common Application Programs के उदाहरण Servers हैं जो किसी Network के विभिन्न Clients को Service प्रदान करने का काम करते हैं। Web Servers, Proxy Servers, Mail Servers, Print Servers व Boot Servers Java Applications के विभिन्न उदाहरण हैं।

Servlets Applets के समान ही होते हैं, लेकिन किसी Browser में Run होने के बजाय ये Java Servers में Run होते हैं और Java Server की Configuring या Tailoring करते हैं।

एक सवाल पैदा हो सकता है कि Java API इन सभी प्रकार के Programs को किस प्रकार से Support करता है। इसका जवाब ये है कि ये इन सभी प्रकार के Programs को एक Software Components के Package के माध्यम से Support करता है जिसमें विभिन्न प्रकार की Functionalities होती हैं। Core API एक ऐसा API है जो हर Java Platform में पूरी तरह से Implemented होता है। Core API हमें निम्न Features प्रदान करता है—

## **The Essentials:**

Objects, strings, threads, numbers, input and output, data structures, system properties, date and time जैसी कई चीजों को Handle करने की सुविधा प्रदान करता है।

## **Applets:**

Java applets बनाने के लिए विभिन्न Components प्रदान करता है।

## **Networking:**

Networking की सुविधा प्राप्त करने के लिए URLs, TCP व UDP sockets तथा IP addresses प्रदान करता है।

## **Internationalization:**

ये हमें ऐसी सुविधा प्रदान करता है कि हम ऐसे Programs लिख सकते हैं जो सारी दुनिया में समान रूप से चल सकता है।

## Security:

ये हमें Low-level और high-level दोनों प्रकार की सुरक्षा प्रदान करता है। साथ ही electronic signatures, public/private key management, access control और certificates की भी सुविधा प्रदान करता है।

## Software components:

ये हमें JavaBeans जैसे Components प्रदान करता है जो किसी पहले से बने हुए Component Architecture में जैसे कि Microsoft's OLE/COM/Active-X architecture, OpenDoc और Netscape's Live Connect में Plug in हो सकता है।

## Object serialization:

ये हमें Remote Method Invocation (RMI) द्वारा दूसरे सरल उपकरणों से Communication करने की सुविधा प्रदान करता है, जिसका प्रयोग आज Mobile Technology में भी हो रहा है।

## Java Database Connectivity (JDBC):

ये हमें Relational databases से Connect होने व उन्हें Access करने की सुविधा प्रदान करता है।

Java में केवल Core API ही नहीं हैं बल्कि कुछ Standard Extensions भी हैं। ये Standard Extensions 3D, Servers, Collaboration, Telephony, Speech, Animation व कई अन्य चीजों के लिए भी APIs Define करते हैं।

## Program

Computer Program एक तरीका है जो Computer को ये बताता है कि उसे कब, क्या करना है। Computer के Boot होने से लेकर Shut Down होने तक जो भी कुछ होता है, किसी ना किसी Program की वजह से होता है। MS-Word एक Program है, Norton Antivirus एक Program है, DOS Prompt पर लिखा जाने वाला हर Command एक Program है, यहां तक कि विभिन्न प्रकार के Computer Viruses भी एक Program हैं।

आज Artificial Intelligence का एक उदाहरण Robots हैं। इन Robots को अमीर लोग अपने घरों में रखते हैं। ये Robots ऐसे होते हैं कि इन्हें जो काम करने के लिए कह दिया जाता है या किसी तरीके से बता दिया जाता है, ये Robots वे सभी काम बड़ी ही अच्छी तरह से कर लेते हैं।

जैसे यदि आप इन Robots को कहें कि जब आपके घर की Bell Ring हो तो इन्हें घर का दरवाजा खोलना है। तो ये वैसा ही करते हैं। ये Computer Program का एक साधारण सा उदाहरण है जिसमें आप किसी निर्जीव Robot को कुछ Instruction देते हैं, और वह निर्जीव Robot आपकी बात मानता है और आपके द्वारा बताया गया काम कर देता है।

इसी तरह से Computer को भी विभिन्न प्रकार के Instructions प्रदान किए जाते हैं, जिनके अनुसार Computer काम करता है। जैसे कि Microsoft Company ने Windows के Program

द्वारा Computer को ये Instruction दिया है कि यदि कोई Mouse को Move करता है, तो Monitor की Screen पर स्थित Cursor या Pointer भी उसी तरह से Move होना चाहिए। यदि कोई Start Button पर Click करता है तो Start Menu Popup होना चाहिए, आदि-आदि।

यानी Computer पर हम जो कोई Action करते हैं, उसे Response करने के लिए पहले से ही Program लिखा गया है। जब कोई Event होता है, Computer उस Event से सम्बंधित Program के अनुसार काम करने लगता है और हमें हमारा Required Result प्रदान करता है।

Computer में जो भी कुछ होता है उसे Event कहते हैं। जैसे यदि हम Mouse को Move करते हैं तो MouseMove Event Generate होता है, यदि हम Mouse से Click करते हैं तो MouseClick, Event Generate होता है। इसी तरह से यदि हम Keyboard पर कोई Key Press करते हैं तो Keypress Event Generate होता है।

ये तो **Hardware Events** के उदाहरण हैं। Computer में Software Events भी Generate होते हैं जिन्हें Response करने के लिए भी विभिन्न प्रकार के Programs लिखे गए हैं। उदाहरण के लिए किसी Window को Minimize करना, Restore करना, किसी Window को Close करना आदि **Software Events** के उदाहरण हैं। निम्न Program देखिए—

---

```
#include <stdio.h>

main()
{
    printf("Hello Gopala");
}
```

---

इस Program द्वारा हम हमारे Computer को केवल एक Message Screen पर Print करने के लिए एक Instruction प्रदान कर रहे हैं। ये Program Computer Screen पर "Hello Gopala" Print करता है।

हम किसी Computer Program में जितनी भी Coding Lines लिखते हैं, ये सभी Lines **Program Statements** कहलाती हैं। Computer उन सभी Statements को एक निश्चित क्रम में Handle करता है, ठीक उसी तरह से जिस तरह से एक रसोईया किसी विशेष प्रकार के पकवान को बनाने के लिए एक विशेष क्रम का पालन करता है।

चूंकि Computer उसी क्रम में विभिन्न Statements के अनुसार काम करता है जिस क्रम में एक Programmer किसी Program को लिखता है। इसलिए यदि कोई Program वैसा Result प्रदान नहीं करता, जैसा एक Programmer चाहता है, तो ये Computer की गलती नहीं है बल्कि उस Program की Mistake है।

ज्यादातर Program उसी तरह से लिखे जाते हैं, जिस तरह से हम कोई Letter लिखते हैं, जिसमें किसी Word Processor में हम हर Word को Type करते हैं। कुछ Programming Languages के Compilers के साथ उनके खुद के Word Processors आते हैं, जैसे कि Turbo C++ का Program Creation का पूरा IDE आता है जबकि कुछ Compilers के साथ कोई Word Processor नहीं आता।

जिन Compilers के साथ कोई Word Processor नहीं आता जिसमें Program की Coding की जा सके, तो ऐसे Program के Source Code लिखने के लिए किसी भी अन्य Word Processor

का प्रयोग किया जा सकता है। हम Java Developer Kit के सभी Components का प्रयोग किसी भी Word Processor जैसे कि Notepad या WordPad के साथ कर सकते हैं।

जब एक Program के Source Codes लिख लिए जाते हैं, तो उसके बाद उस Source File को उस Language के Extension के साथ Save करना होता है। जैसे यदि हम Notepad का प्रयोग करके "C" Language का Program लिखते हैं तो File को Save करते समय हमें File के नाम के बाद .C Extension देना होता है। उसी तरह से यदि हम Java के Program को Save करते हैं, तो हमें File के नाम के बाद .java Extension का प्रयोग करना होता है। जैसे *Program.java*, *Application.java* आदि।

हम जो Program लिखते हैं वे English के कुछ सामान्य Words होते हैं। लेकिन Computer केवल Binary Language को ही समझता है। इसलिए हमें एक ऐसे Program की जरूरत होती है जो हमारे Source Codes को Computer के समझने योग्य Machine Language में Convert कर सके।

**Interpreter** एक ऐसा Program है जो किसी भी Program की Source File के हर Statement या Code की हर Line को Computer की Machine Language में Convert करके Computer को बताता है कि उसे क्या करना है।

कुछ Languages में एक अन्य Software जिसे **Compiler** कहते हैं का प्रयोग करके Source Code File को Machine Language में Convert करता है। इन दोनों में अन्तर केवल इतना है कि Interpreter Source File के हर Line या हर Statement को Computer के समझने योग्य Binary Language में Convert करता है और यदि किसी Statement में कोई Error हो तो उस Line या Statement से आगे Interpret नहीं होता।

जबकि Compiler एक ऐसा Program होता है जो पूरे Program को एक साथ Machine Language में Convert करता है। यदि Program में कोई Error हो तो Program सभी Errors को एक साथ Display करता है और तब तक Program को Machine Language में Convert नहीं करता है जब तक कि सभी Errors को Debug ना कर दिया जाए।

जो Program Interpreted होते हैं वे Compiled Program की तुलना में धीरे चलते हैं। लेकिन Java एक ऐसी Language है जिसको Interpreter व Compiler दोनों की जरूरत होती है।

जब भी हम कोई Program लिखते हैं तो उसमें किसी ना किसी तरह की Errors हमेशा आती है। इन Errors को Computer Programming की भाषा में **Bug** कहा जाता है और इन Errors को सही करने के Process को **Debug** करना कहते हैं।

## ***Procedural Techniques and OOPS***

Pascal, C, Basic, Fortran जैसी पारम्परिक भाषाएं Procedural Languages के उदाहरण हैं, जिसमें प्रत्येक Statement Computer को कुछ करने का आदेश देता है। यानी Procedural Languages Instructions का एक समूह होता है। Procedural Languages में छोटे Programs के लिये किसी भी अन्य प्रकार के Pattern की आवश्यकता नहीं होती है। Programmer Instructions की List बनाता है और Computer उनके अनुसार काम करता है।

जब प्रोग्राम काफी बड़े व जटिल हो जाते हैं, तब Instructions की यह List काफी परेशानी पैदा करती है। इसलिये एक बड़े प्रोग्राम को छोटे-छोटे टुकड़ों में बांट दिया जाता है। इन छोटे-छोटे

टुकड़ों को Functions कहा जाता है। Functions को दूसरी अन्य भाषाओं में Subroutine, Subprogram या Procedure कहा जाता है।

एक बड़े प्रोग्राम को छोटे-छोटे Functions में विभाजित करने से पूरा Program Functions का एक समूह बन जाता है। इसे Module कहा जाता है। लेकिन ये Modules भी Procedural Programming में ही आते हैं क्योंकि सभी Functions में Statements की एक List होती है और सभी Functions मिल कर पूरा Program बनाते हैं, जिससे पूरा Program Instructions की एक बहुत बड़ी List बन जाती है।

Procedural Languages के शुरुआती दौर में इनमें ही Program Develop किए जाते थे। “C” भी एक Procedural Languages है और जब “C” भाषा का आविष्कार हुआ था तब Programmers अन्य भाषाओं को छोड़ कर “C” में ही अपने Program Develop करने लगे थे।

लेकिन समय व आवश्यकता के अनुसार जब Program बड़े व जटिल होने लगे, तब Programmers को इस भाषा में प्रोग्राम बनाने में दिक्कतें आने लगीं। उन्होंने महसूस किया कि इस भाषा में कुछ सुधार की आवश्यकता है ताकि ये भाषा सरल व लोकप्रिय बन सके। ये भाषा सरल बन सके इसके लिये इसका वास्तविक जीवन के अनुसार होना जरूरी था।

यानी हम हमारे सामान्य जीवन में जिस प्रकार से व्यवहार करते हैं, इस भाषा का भी वैसा ही होना जरूरी था ताकि Programmers इसमें अधिक सरलता व सफलता से Program बना सकें। भाषा वास्तविक जीवन के अनुसार हो, यही **Concept Object Oriented Programming** यानी **OOD** का आधार बना। “C” भाषा की इन कमियों को पहचाना गया और इसमें सुधार किया गया।

फलस्वरूप हमें “C” भाषा का एक नया संस्करण “C++” प्राप्त हुआ जो कि Object Oriented Concept पर आधारित है। आवश्यकता के अनुसार इस भाषा की कमियों को भी पहचाना गया और उसमें सुधार करने पर जो नई भाषा सामने आई वह Java थी। आइये, हम भी जानने की कोशिश करते हैं कि “C” भाषा में ऐसी कौनसी कमियां थीं, जिनमें सुधार की आवश्यकता महसूस की गई ?

Procedural Languages में काम होने का महत्व था Data का नहीं, यानी कि Keyboard से Data Input किया जाए, Data पर Processing की जाए, Errors को Check किया जाए आदि। Functions में भी इसी महत्व को जारी रखा गया। Functions कोई काम करते हैं, उसी प्रकार से जिस प्रकार से साधारण Statement करता है। Functions कोई जटिल काम भी कर सकते हैं लेकिन इनमें भी काम के होने का ही महत्व था।

पूरे Program में Data पर कोई ध्यान नहीं दिया जाता था जबकि पूरे प्रोग्राम का मूल आधार Data ही होता है। किसी Inventory के Program में इस बात का कोई ज्यादा महत्व नहीं होता है कि Data को किस प्रकार से Display किया जाता है या एक Function किस प्रकार से Corrupt Data को Check करता है, बल्कि इस बात का होता है कि Data क्या है और वह किस प्रकार से Program में काम आ रहा है। Procedural Program में Data को द्वितीय स्तर पर रखा गया था जबकि किसी भी Program का मूल आधार Data ही होता है।

उदाहरण के लिये, किसी Inventory के Program में किसी Data File को Memory में Load किया जाता है, तब ये File एक Global Variable की तरह होती है, जिसे कोई भी Function Use कर सकता है। ये Functions Data पर विभिन्न प्रकार के Operations करते हैं। यानी ये Data को Read करते हैं, Analyze करते हैं, Update करते हैं, Rearrange करते हैं, Display करते हैं और वापस Disk पर Write करते हैं। “C” में Local Variables भी होते हैं लेकिन Local Variables, महत्वपूर्ण Data के लिये इतने उपयोगी नहीं होते हैं, जो कि विभिन्न Functions द्वारा Access किए जाते हैं।

मान लें कि एक नए Programmer को Data को किसी खास तरीके से Analyze करने के लिये एक Function लिखने को कहा गया। प्रोग्राम की जटिलता से अनभिज्ञ Programmer एक ऐसा Function बनाता है, जो कि अचानक किसी महत्वपूर्ण Data को नष्ट कर देता है। ऐसा होना काफी आसान है क्योंकि कोई भी Function Data को Access कर सकता है।

इसलिये क्योंकि Procedural Language में Data Global होता है। ये कुछ ऐसा ही है जैसे कि आप अपने Personal कागजात को Telephone Directory के पास रख दें जहां कभी भी कोई भी पहुंच सकता है, उससे छेड़छाड़ कर सकता है और उसे नष्ट कर सकता है। इसी प्रकार से Procedural Languages में होता है जहां आपका Data Global होता है और कोई भी Function उसे Use करके खराब कर सकता है या नुकसान पहुंचा सकता है।

Procedural Languages की दूसरी कमी ये थी कि कई Functions एक साथ एक ही Data को Use कर रहे होते हैं, इसलिये Data को Store करने का तरीका काफी जटिल हो जाता है। समान Data को Use कर रहे सभी Functions को Modify किए बिना Data में किसी प्रकार का कोई परिवर्तन नहीं किया जा सकता है।

उदाहरण के लिये यदि आप एक नया Data Add करते हैं तो उन सभी Functions को Modify करना होगा जो कि Data को Use कर रहे हैं, ताकि ये सभी Functions Add किए गए नए Data को Use कर सकें। ये पता करना कि कौन-कौन से Function Data को Use कर रहे हैं और सभी को बिल्कुल सही तरीके से Modify करना काफी कठिन होता है।

Procedural Programs को Design करना काफी मुश्किल होता है। समस्या ये होती है कि इनका Design वास्तविक जीवन से Related नहीं होता है। जैसे कि, माना आप एक Graphics User Interface में Menus, Windows के लिये Code लिखना चाहते हैं, तो आपको ये तय करना मुश्किल होगा कि कौनसे Functions Use किए जाए? कौनसा Data Structure Use किया जाए? आदि। इनका कोई स्पष्ट उत्तर नहीं है।

Procedural Programs के साथ कई और परेशानियां हैं। उनमें से एक समस्या नए Data Type की है। Computer Languages में कई प्रकार के Built-in Data Types होते हैं, जैसे कि Integer, Float, Character आदि। मानलो कि आप Complex Numbers के साथ प्रक्रिया करना चाहते हैं या Two-dimensional Coordinates के साथ काम करना चाहते हैं या Data के साथ प्रक्रिया करना चाहते हैं। Built-in Data Type इनको आसानी से Handle नहीं कर सकते हैं।

इसलिए हमें हमारी आवश्यकतानुसार स्वयं के Data Type बनाने की जरूरत होती है। Procedural Language में स्वयं के Data Type बना कर हम उन्हें बिल्कुल Built-in Data Type की तरह Use नहीं कर सकते हैं। Procedural Language इतने उन्नत नहीं हैं। बिना अप्राकृतिक जटिल तरीकों के आप Procedural Languages में x व y दोनों Coordinates को एक ही Variable में Store करके उस पर Processing नहीं कर सकते हैं। Procedural Languages को लिखना व Maintain करना काफी मुश्किल काम होता है।

## ***The Object-Oriented Approach***

Object Oriented Language का मूलभूत विचार ये है कि जिस समस्या का समाधान Computer पर प्राप्त करना है उस समस्या के मूल Data और उस Data पर काम करने वाले Functions को Combine करके एक Unit के रूप में ले लिया जाता है। इस Unit को **Object** कहा जाता है।

एक Object के Data पर काम करने के लिये लिखे गए Operations या Functions को Java में **Methods** कहा जाता है। ये Methods किसी Object के Data को Access करने का एक मात्र माध्यम होते हैं। यदि आप किसी Object के अन्दर रखे किसी Data को Read करना चाहते हैं, तो आपको इसी Object के अन्दर Define किए गए उस Method को Use करना पड़ता है, जिसे उस Object के Data को Access करने के लिये ही परिभाषित किया गया है। यही एक Method होता है जिसकी मदद से आप उस Object के Data को Read कर सकते हैं।

आप सीधे ही Data के साथ किसी प्रकार की प्रक्रिया नहीं कर सकते हैं क्योंकि Data Hidden रहता है। इसलिये किसी प्रकार से अचानक हुए परिवर्तन से Data सुरक्षित रहता है। Data व Data को Use कर सकने वाले Functions या Operations का एक साथ एक ही Unit के रूप में होना **Encapsulation** कहलाता है।

Data का Hidden रहना यानी **Data Hiding** व **Encapsulation** Object Oriented Programming का मूल तथ्य या Key Terms है। यदि आप किसी Data को Modify करना चाहते हैं, तो आपको पता होना चाहिए कि कौनसा Method उस Data पर Required Operation करने की क्षमता रखता है। कोई भी अन्य Method उस Data को Access नहीं कर सकता है। ये Processing Program को लिखना, Debug करना व Maintain करना आसान बनाती है।

एक Java का प्रोग्राम ढेर सारे विभिन्न प्रकार के Objects का बना होता है, जो कि अपने-अपने Methods द्वारा आपस में Communication करते हैं। Java व कई अन्य OOP Languages में Member Functions को **Methods** और Data Item को **Instance Variable** कहा जाता है। किसी Object के Methods को Use करना उस Object को **Message Send** करना कहलाता है।

हम एक उदाहरण लेते हैं। माना एक बड़ा प्रीति-भोज का समारोह है जिसमें सभी मेहमान किसी Dining Table के चारों ओर बैठे हैं। जो भी खाना Table पर रखा है हम उसे Data कह सकते हैं और उस Table के चारों ओर बैठे लोगों को हम Functions या Operations मान सकते हैं।

Object के Operations हमेशा अपने Data यानी Attributes की State में परिवर्तन करते हैं। इस उदाहरण में Data (खाना) पर खाना खाने का Operation Perform किया जा रहा है। इस व्यवस्था में जब भी किसी को Table पर रखे विभिन्न प्रकार के व्यंजनों में से कुछ लेना होता है, तो वह स्वयं ही उस व्यंजन तक पहुंचता है और उसे उपयोग में ले लेता है। किसी पड़ोसी मेहमान से कोई भी व्यंजन Pass करने को नहीं कहता।

**Procedural Program** का भी यही तरीका होता है। ये तरीका तब तक बहुत ठीक है, जब तक कि खाना खाने वाले मेहमानों की संख्या सीमित हो। लेकिन यदि मेहमानों की संख्या अधिक हो तो ये तरीका ठीक नहीं कहा जा सकता है।

क्योंकि जब मेहमान अधिक होंगे तो Table भी बड़ा होगा और खाने के विभिन्न सामान पूरे Table पर काफी दूर-दूर होंगे। ऐसे में यदि कोई मेहमान किसी दूर रखे व्यंजन तक पहुंचने की कोशिश करता है, तो हो सकता है कि कोशिश करते समय उसके Shirt की Sleeves किसी दूसरे मेहमान के खाने में चली जाए या ऐसा भी हो सकता है कि कई मेहमान एक साथ किसी एक ही व्यंजन पर हाथ बढ़ाएं और व्यंजन Table पर गिर कर खराब हो जाए।

यानी यदि मेहमानों की संख्या काफी ज्यादा हो तो एक ही Table पर भोजन करना एक परेशानी वाला काम होगा। एक बड़े **Procedural Program** में भी यही होता है।

इस समस्या के समाधान के रूप में यदि कई छोटे-छोटे Tables हों और उन पर एक सीमित मात्रा में मेहमान हों और सबके पास उनका अपना भोजन हो, तो ये एक अच्छी व्यवस्था हो सकती है। इस छोटे Table पर सभी मेहमान किसी भी व्यंजन पर आसानी से पहुंच सकते हैं। यदि कोई मेहमान किसी अन्य Table पर रखे किसी व्यंजन को लेना चाहता है तो सम्भवतया वह किसी अन्य मेहमान से उस व्यंजन को लाने के लिये कह सकता है।

ये तरीका **Object Oriented Programming** का है जिसमें हरेक छोटी Table को एक **Object** कहा जा सकता है। हरेक Object में उसका स्वयं का **Data** और **Data** पर **Perform** होने वाला **Operation** या **Function** होता है। **Data** व **Operations** के बीच होने वाले विभिन्न लेन-देन अधिकतर Object के अन्दर ही होते हैं लेकिन आवश्यकतानुसार ये भी सम्भव है कि किसी अन्य Object के **Data** को भी Use किया जा सके।

चूंकि एक Object के **Data** को केवल वही Object Access कर सकता है, इसलिए यदि किसी Object A के **Data** को कोई दूसरा Object B Access करना चाहता है, तो वह Object B Object A से **Data** को Access करने के लिए कहता है। इस प्रक्रिया को **Message Passing** करना कहते हैं। Object A Object B की Request को पूरा करता है और अपने **Data** को Access करके करने के लिए उस दूसरे Object B को दे देता है।

## **Difference Between C++ and Java**

वास्तव में Java “C” व “C++” का ही Modified रूप है। चूंकि आज भी ज्यादातर Professional लोग बड़े Projects के लिए “C++” को ही चुनते हैं, इसलिए ये जानना जरूरी है कि Java में “C++” की किन विशेषताओं को लिया गया है और किन चीजों को छोड़ा गया है जो सामान्य Programmer को परेशान करती हैं।

## **Preprocessor**

“C” व “C++” में Program के Compilation को Control करने के लिए Preprocessors का प्रयोग किया जाता है। “C++” का Compiler किसी भी Source Program को Compile करने से पहले सभी Preprocessor Directives को Expand करने का काम करता है। सभी “C” व “C++” के Programmers जानते हैं कि Preprocessors का प्रयोग करने पर Program की जटिलता बढ़ जाती है। “C++” के Programmer Preprocessors का प्रयोग करके लगभग स्वयं की Language बनाना शुरू कर देते हैं।

ज्यादातर Statement के लिए व Constant मानों के लिए वे Preprocessors का प्रयोग करते हैं। इससे Program की जटिलता इतनी बढ़ जाती है कि कोई भी नया Programmer यदि उस Program को समझना चाहे तो उसे काफी परेशानी आती है। साथ ही इन Program Codes को Reuse भी नहीं किया जा सकता है।

Preprocessor Directives की एक कमी ये भी है कि इनकी Type Checking कभी भी निश्चित नहीं होती। यानी ये हमेशा एक String Format को Follow करते हैं। यदि हम **#define MAX 10** Statement लिखते हैं, तो यहां मान 10 Integer नहीं बल्कि एक String होता है।

Java में Preprocessors को हटा दिया गया है। हालांकि Java Preprocessor Directives के समान ही Functionality प्रदान करता है लेकिन अधिक Control के साथ। Java में **#define** के स्थान पर Constant Data Members का प्रयोग किया जाता है।



इसका परिणाम ये है कि Java के Codes को पढ़ना व समझना “C++” के Codes को पढ़ने व समझने की तुलना में अधिक सरल हो जाता है। साथ ही Java के Programs में Header Files का प्रयोग नहीं होता है बल्कि Java का Compiler Source Code File से सीधे ही Class Definitions बना लेता है जिसमें Class Definitions व Methods दोनों होते हैं।

## Pointers

जितने भी “C” या “C++” के Programmers हैं, वे सभी मानते हैं कि यदि Pointers को पूरी सावधानी से प्रयोग ना किया जाए तो ऐसे Errors Generate होते हैं, जिन्हें Debug करने में दिमाग का पसीना निकल जाता है। साथ ही Pointers के प्रयोग से Program हमेशा समझने में जटिल हो जाता है, हालांकि Pointers के प्रयोग से हमारा Program Directly Memory Locations को Access कर सकता है, इसलिए Program की Speed तुलना में तेज हो जाती है।

“C++” के Programmers हमेशा Dynamic Data Structure को Create व Maintain करने के लिए Pointers Arithmetic का प्रयोग करते हैं और हमेशा जटिल Bugs में फंसते हैं। “C++” Programmers का ज्यादातर समय उन Programs को Create करने में नहीं बीतता जिनमें Pointer का प्रयोग होता है, बल्कि उन Programs को Debug करने में बीतता है।

Java Pointers को Support नहीं करता है। यानी Java में Pointers जैसी कोई व्यवस्था नहीं है जो Directly Memory को Access कर सके। हालांकि Pointers के स्थान पर Java में References का बहुत प्रयोग किया जाता है जो कि Pointers के समान ही काम करते हैं लेकिन References का Arithmetic सामान्य Arithmetic जैसा ही होता है ना कि Pointer Arithmetic जैसा।

इस Process से उन सभी Errors से छुटकारा मिल जाता है जो Pointers के Mismanagement के कारण Generate होती हैं। References का प्रयोग करने से Java के Program पर्याप्त Readable व समझने योग्य होते हैं जबकि Pointers का चाहे पूरी तरह से सही प्रयोग किया जाए, लेकिन Program आसानी से समझने योग्य व Readable नहीं होता है।

“C” व “C++” के Programmers सोच सकते हैं कि वे जो काम Pointers का प्रयोग करके काफी आसानी से कुछ Data Structures को Implement कर सकते थे वे काम Java में नहीं किए जा सकेंगे। जैसे कि Dynamic Arrays Java में Create नहीं हो सकते। लेकिन ऐसा नहीं है। वास्तविकता ये है कि वे सभी काम Java में Objects व Objects के Array के प्रयोग से अधिक आसानी व Reliability के साथ किए जा सकते हैं।

Java हमें कुछ Runtime Security भी प्रदान करता है जो कोई अन्य Language Provide नहीं करती। जैसे कि यदि हम “C” या “C++” में किसी Array की Size को 10 Define किया है और हम 11<sup>th</sup> Index Number पर कोई मान Input करना चाहें तो Java हमें ऐसा नहीं करने देता जबकि “C” व “C++” में हम ऐसा करके किसी दूसरे Data को Damage कर सकते हैं।

## Structure and Union

“C” में दो तरह के (Structure and Union) और “C++” में तीन तरह के (Structure Union and Class ) Complex Data Types हैं। Java में केवल एक ही Complex Data Type है जिसे Class कहते हैं। “C” व “C++” में जितने काम इन तीनों को प्रयोग करके किए जाते हैं Java में वे सभी काम केवल एक Class से ही किया जा सकता है। जब हमें Structure या Union के Functionality की जरूरत होती है तो Java हमें इन Functionality को Class द्वारा प्राप्त करने के लिए बाध्य करता है।

हालांकि ऐसा लग सकता है कि Java में Programmers को Structure व Union के स्थान पर Class Use करने के लिए Extra काम करना पड़ेगा लेकिन ऐसा नहीं है। बल्कि Class के प्रयोग से Program की जटिलता कम हो जाती है।

Java बनाने वालों ने इसी बात को प्राथमिकता दी है कि जितना हो सके उतना Java को Simple व आसानी से समझने योग्य Language में रखा जाए, इसलिए Java में से “C” व “C++” की उन चीजों को हटा दिया गया है जो सीखने व समझने में परेशानी Create करती हैं और Language को जटिल बनाती हैं।

वैसे “C” व “C++” को उन लोगों के लिए लिखा गया था जो पहले से ही पेशेवर Programmer थे, लेकिन Java को नए Programmers को ध्यान में रख कर Develop किया गया है, जो कि पेशेवर नहीं हैं और Computer को बहुत गहराई से नहीं जानते बल्कि जिन्हें Computer का सामान्य ज्ञान ही है।

चूंकि Java में Structure व Union नहीं हैं इसलिए Java एक पूर्ण Object Oriented Programming Language है, क्योंकि इसमें जो भी काम करना होता है, उसके लिए Class Develop करनी पड़ती है। इससे Program Codes को ना केवल Reuse किया जा सकता है बल्कि Program को Maintain करना भी अन्य Languages की तुलना में काफी सरल होता है।

## Functions

“C” व “C++” में सभी Program Codes को किसी ना किसी Function में लिखा जाता है। यहां तक कि Main Program भी एक Function main() में होता है जहां आवश्यकतानुसार अन्य Functions को Call करके अपना काम पूरा किया जाता है। ये Functions Global रखे जाते हैं ताकि पूरे Program का कोई भी अन्य Function इन्हें Call कर सके और Data पर Required Processing कर सके। “C++” में भी Functions होते हैं जिनके माध्यम से Class के Data को Access किया जाता है। इन Functions को ही Method कहा जाता है।

“C++” Class के Methods Java Class के Methods के समान ही होते हैं। फिर भी चूंकि “C++” “C” जैसी Procedural Language को भी Support करता है इसलिए इसे पूरी तरह से Object Oriented Programming Language नहीं कहा जा सकता, बल्कि इसे Hybrid Language कहा जाता है। इस स्थिति में “C++” के Class के Methods व Class के बाहर के Functions दोनों का Mixture होता है, इसलिए कुछ हद तक “C++” में Confusion की स्थिति भी बनी रहती है जिससे कई बार अजीब तरह के Bugs का सामना करना पड़ता है।

Java में कोई Function नहीं होता है, इसलिए Java “C++” की तुलना में पूरी तरह से Object Oriented Programming Language है। Java Programmer को इस बात के लिए बाध्य करती है कि Programmer अपने सभी Routines को Class Methods के रूप में व्यवस्थित करके रखें। Java की इस बाध्यता के कारण Programmer अपने Codes को अधिक अच्छे तरीके से Organize करके रखता है।

## Multiple Inheritance

Multiple Inheritance “C++” की एक ऐसी व्यवस्था है जिसमें हम कई Classes से अपनी आवश्यकतानुसार कुछ-कुछ Features को लेकर एक नई Class बना सकते हैं। यानी हम एक Class को कई Parent Class से Derive कर सकते हैं। हालांकि Multiple Inheritance वास्तव

में काफी Powerful है, लेकिन कई Classes से एक Derive Class बनाने व उसे Maintain करने में Programmer को काफी परेशानी का सामना करना पड़ता है।

Java Multiple Inheritance को Support नहीं करता है। “C++” की इस Functionality को हम Java में Interfaces का प्रयोग करके प्राप्त कर सकते हैं। Java के Interfaces Object Method Description Provide करते हैं लेकिन इनमें कोई Implementation नहीं होता है।

## Strings

“C” व “C++” में Text Strings को Handle करने के लिए कोई Built – In Support नहीं है। String को Handle करने के लिए “C” व “C++” Programmers को एक Null Terminated Array का प्रयोग करना पड़ता है।

Java में Strings को एक First Class Object की तरह Implement किया जाता है यानी, Strings Java Language का मुख्य बिन्दु या Core है। Java में हम एक Object के रूप में String को Implement करते हैं, जिससे हमें कई Advantages प्राप्त होती है।

## goto Statement

“C” व “C++” में इस Statement को आवश्यकतानुसार काफी Use किया जाता है। लेकिन इसके Use करने पर जितनी तरह की परेशानियां आती हैं, उन्हें तय करना ही मुश्किल है। goto Statement का प्रयोग यदि बड़े Program में किया जाता है, तो किसी नए Programmer के लिए उस Program को समझना नामुमकिन हो जाता है। ये एक Branching Statement है, लेकिन Program में ये Statement Program Control को काफी Jump करवाता है, जिससे Program की Efficiency भी प्रभावित होती है।

Java में इस Statement को Support नहीं किया गया है। Java ने goto को एक Keyword के रूप में मान्यता दी है लेकिन इसके उपयोग को Support नहीं किया है।

## Operator Overloading

Operator Overloading को Java में Support नहीं किया गया है। Operator Overloading “C++” की एक ऐसी तकनीक है जिससे किसी भी Primary Operator को आवश्यकतानुसार दूसरा अर्थ प्रदान कर दिया जाता है। जैसे + Operator का प्रयोग दो Objects को जोड़ने के लिए किया जा सकता है। हालांकि Java में इस काम को Class में किया जाता है। लेकिन Operator Overloading तकनीक को Java में छोड़ दिया गया है। Operator Overloading Programmer के लिए एक सुविधा होती है, लेकिन इससे एक ही Operator को एक ही Program में विभिन्न अर्थ प्रदान कर दिए जाने से Program को समझने में Confusion हो जाने की सम्भावना रहती है। इसलिए इस तकनीक को Java में छोड़ दिया गया है।

## Automatic Type Casting

“C” व “C++” में Automatic Type Casting हो सकती है। यानी एक Integer प्रकार के Variable को यदि Float प्रकार का मान प्रदान करना हो, तो “C” व “C++” का Compiler Automatic Type Casting करके एक Variable के मान को दूसरे Variable के मान में Automatically Convert कर देता है। यह Automatic Type Casting कहलाती है। जबकि Java में इसे Support नहीं किया गया है।

क्योंकि जब किसी एक Data Type के Variable को दूसरे प्रकार के Data Type के Variable का मान प्रदान किया जाता है, तो Data के मान का Loss हो जाता है। जैसे यदि Integer प्रकार के Variable को किसी Float प्रकार के Variable का मान प्रदान किया जाए, तो Integer प्रकार के Variable में Float प्रकार के मान में दसमलव के बाद की संख्या Store नहीं होती है। यानी दसमलव के बाद की संख्या का Loss हो जाता है।

इसलिए Java में इस Automatic Type Casting को Support नहीं किया गया है। Java में यदि किसी Data के मान का किसी भी प्रकार की प्रक्रिया से Loss होता है तो Java Automatic Type Casting नहीं करता है। इसकी Type Casting Programmer को ही करनी पड़ती है।

## Variable Number of Arguments

“C” व “C++” में हमने देखा है कि हम printf() Function या scanf() Function में अपनी आवश्यकतानुसार एक या एक से अधिक चाहे जितने भी Arguments प्रदान कर सकते हैं। यदि हमें चार Variable के मान Output में Print करने हों तो हम printf() Function में चार Variable Argument के रूप में भेज सकते हैं और यदि केवल एक ही Variable का मान Print करना हो तो केवल एक Argument भी printf() Function में भेज सकते हैं। हालांकि ये एक अच्छा तरीका है लेकिन Java में इसे Support नहीं किया गया है। क्योंकि Compiler ये कभी Check नहीं कर पाता है कि उसे जिस Argument के मान को Print करना है वह उचित Data Type का है या नहीं।

## Command Line Argument

Java Program में जिस तरह से Command Line पर Argument Pass किए जाते हैं वे “C” व “C++” के Program में Pass किए जाने वाले Arguments की तुलना में थोड़े अलग होते हैं। “C” व “C++” में System main() Function को दो Arguments Pass करता है। पहला Argument **argc** और दूसरा Argument **argv** होता है। पहला Argument **argc** ये Specify करता है कि दूसरे Argument **argv** में कितने Arguments हैं। दूसरा Argument **argv** एक Array of Characters का Pointer होता है। इसी Argument में Command Prompt पर दिए जाने वाले Actual Arguments होते हैं।

Java में System Command Line से केवल एक ही Argument **args** Java Program में Pass करता है। ये एक Strings का Array होता है जिसमें Command Line Arguments होते हैं।

“C” व “C++” में Command Line Arguments एक Program में Pass होते हैं। इस Argument में उस Program का नाम भी होता है जिसमें Arguments को Pass करना होता है। ये नाम Command Line पर सबसे पहले First Argument के रूप में लिखा जाता है। Java में हम पहले से ही उस Program का नाम जानते होते हैं जिसमें हमें Argument भेजना होता है। ये नाम हमें उस Program के Class का नाम ही होता है, इसलिए Command Prompt पर हमें केवल Argument ही Pass करना होता है।

## Programming – The Basic Concept

Computer Programming समझने से पहले हमें ये समझना होता है कि Computer क्या काम करता है और कैसे काम करता है। कम्प्यूटर का मुख्य काम Data का Management करना होता है। हमारे आस-पास जो भी चीजें हमें दिखाई देती हैं वे सभी एक Object Oriented Programming Language के लिए Objects हैं।

हर Objects के कुछ Attributes होते हैं, जिनसे किसी Particular Object की पहचान होती है। इस Attribute में Numerical या Non-Numerical किसी ना किसी प्रकार का मान प्रदान किया जा सकता है। ये मान ही Computer का वह Data होता है, जिसे Computer Manage करता है।

यदि बिल्कुल ही सरल शब्दों में कहें तो कोई मान या मानों का समूह Computer के लिए Data होता है। ये मान दो तरह के हो सकते हैं : Numerical या Non-Numerical, Computer में हम इन्हीं Data को Store करते हैं, Process करते हैं और किसी ना किसी प्रकार की Information Generate करते हैं।

Computer केवल Electrical Signals या मशीनी भाषा को समझता है। ये मशीनी भाषा बाइनरी रूप में होती है जहां किसी Signal के होने को 1 व ना होने को 0 से प्रदर्शित किया जाता है। यदि हम हमारी किसी बात को Binary Format में Computer में Feed कर सकें, तो Computer हमारी बात को समझ सकता है।

Computer भाषा वह भाषा होती है, जिसे Computer समझ सकता है। हर Computer भाषा का एक Software होता है, जिसे Compiler या Interpreter कहते हैं। ये Software हमारी बात को Computer के समझने योग्य मशीनी भाषा या Binary Format में Convert करता है।

Computer को कोई बात समझाने के लिए उसे एक निश्चित क्रम में सूचनाएं देनी होती हैं, जिन्हें Instructions कहा जाता है। Computer इन दी गई Instructions के अनुसार काम करता है और हमारी इच्छानुसार हमें परिणाम प्रदान करता है। Instructions के इस समूह को ही Program कहा जाता है।

Computer में हर Electrical Signal या उसके समूह को Store करके रखने की सुविधा होती है। इन Electrical Signals के समूह को File कहते हैं। Computer में जो भी कुछ होता है वह File के रूप में होता है। Computer में दो तरह की File होती है। पहली वह File होती है जिसमें हम हमारे महत्वपूर्ण Data Store करके रखते हैं। इसे Data File कहा जाता है।

दूसरी File वह File होती है, जिसमें Computer के लिये वे Instructions होती हैं, जो Computer को बताती हैं कि उसे किसी Data पर किस प्रकार से Processing करके Result Generate करना है। इस दूसरी प्रकार की File को Program File कहा जाता है।

हम विभिन्न प्रकार की Computer Languages में Program Files ही Create करते हैं। जब बहुत सारी Program Files मिल कर किसी समस्या का समाधान प्राप्त करवाती हैं, तो उन Program Files के समूह को Software कहा जाता है। Computer Software मुख्यतया दो प्रकार के होते हैं:

## System Software:

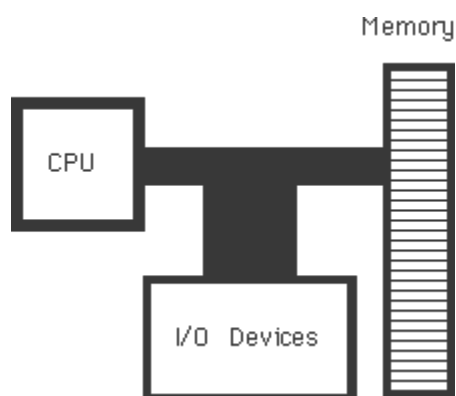
ये Software उन प्रोग्रामों का एक समूह होता है जो कम्प्यूटर की Performance को Control करता है। यानी Computer पर किस तरह से एक प्रोग्राम रन होगा और किस तरह से प्रोग्राम Output देगा। किस तरह Hard Disk पर Files Save होंगी, किस तरह पुनः प्राप्त होंगी, आदि। Windows, Unix, Linux, आदि System Software के उदाहरण हैं।

## Application Software:

ये Software प्रोग्रामरों द्वारा लिखे जाते हैं व ये Software किसी खास प्रकार की समस्या का समाधान प्राप्त करने के लिये होते हैं। जैसे Tally, MS-Office आदि Application Software के उदाहरण हैं।

## Computer Architecture

Computer से अपना मनचाहा काम करवाने के लिए, सबसे पहले हमें Computer के Architecture को समझना होगा। Computer के Architecture को समझे बिना, हम Computer Programming को ठीक से नहीं समझ सकते। Computer System के मुख्य-मुख्य तीन भाग होते हैं-



## I/O Devices

वे Devices जिनसे Computer में Data Input किया जाता है और Computer से Data Output में प्राप्त किया जाता है, I/O Devices कहलाती हैं। Keyboard एक Standard Input Device है और Monitor एक Standard Output Device है।

## Center Processing unit (CPU)

यह एक Microprocessor Chip होता है। इसे Computer का दिमाग भी कहा जाता है क्योंकि Computer में जो भी काम होता है, उन सभी कामों को या तो CPU करता है या Computer के अन्य Devices से उन कामों को करवाता है। इसका मुख्य काम विभिन्न प्रकार के Programs को Execute करना होता है। इस CPU में भी निम्न विभाग होते हैं जो अलग-अलग काम करते हैं-

## Control Unit

इस Unit का मुख्य काम सारे Computer को Control करना होता है। CPU का ये भाग Computer की आंतरिक प्रक्रियाओं का संचालन करता है। यह Input/Output क्रियाओं को Control करता है, साथ ही ALU व Memory के बीच Data के आदान-प्रदान को निर्देशित करता है।

यह Program को Execute करने के लिए Program के Instructions को Memory से प्राप्त करता है और इन Instructions को Electrical Signals में Convert करके उचित Devices तक पहुंचाता है, जिससे Data पर Processing हो सके। Control Unit ALU को बताता है कि Processing के लिए Data Memory में कहां पर स्थित हैं, Data पर क्या प्रक्रिया करनी है और Processing के बाद Data को वापस Memory में कहां पर Store करना है।

## Arithmetic Logic Unit (ALU)

CPU के इस भाग में सभी प्रकार की अंकगणितीय व तार्किक प्रक्रियाएं होती हैं। इस भाग में ऐसा Electronic Circuit होता है जो Binary Arithmetic की गणनाएं करता है। ALU Control Unit से निर्देश या मार्गदर्शन लेता है, Memory से Data प्राप्त करता है और परिणाम को या Processed Data को वापस Memory में ही Store करता है।

## Registers

Microprocessor में कुछ ऐसी Memory होती है जो थोड़े समय के लिए Data को Store कर सकती है। इन्हें Registers कहा जाता है। Control Unit के निर्देशानुसार जो भी Program Instructions व Data Memory से आते हैं वे ALU में Calculation के लिए इन्हीं Registers में Store रहते हैं। ALU में Processing के बाद वापस ये Data Memory में Store हो जाते हैं।

## Memory

Memory Computer की Working Storage या कार्यकारी मेमोरी होती है। यह Computer का सबसे महत्वपूर्ण भाग होता है। इसे RAM कहते हैं। इसी में Process होने वाले Data और Data पर Processing करने के लिखे गए Program Instructions होते हैं, जिन्हें Control Unit ALU में Processing के लिए Registers में भेजता है। Processing के बाद जो सूचनाएं या Processed Data Generate होते हैं, वे भी Memory में ही आकर Store होते हैं।

Memory में Data को संग्रह करने के लिए कई Storage Locations होती हैं। हर Storage Location एक Byte की होती है और हर Storage Location का एक पूर्णांक Number होता है, जिसे उस Memory Location का Address कहते हैं। हर Storage Location की पहचान उसके Address से होती है। 1 Byte की RAM में एक ही Character Store हो सकता है और इसमें सिर्फ एक ही Storage Location हो सकती है।

इसी तरह 1 KB की RAM में 1024 Storage Locations हो सकती हैं और इसमें 1024 अक्षर Store हो सकते हैं। जो Memory जितने Byte की होती है उसमें उतने ही Characters Store हो सकते हैं और उसमें उतनी ही Storage Locations हो सकती हैं।

जिस तरह से किसी शहर में ढेर सारे घर होते हैं और हर घर का एक Number होता है। किसी भी घर की पहचान उसके घर के Number से भी हो सकती है। उसी तरह से Memory में भी विभिन्न Storage Cell होते हैं जिनका एक Unique Number होता है। हम किसी भी Storage Cell को उसके Number से पहचान सकते हैं और Access कर सकते हैं। हर Storage Cell के इस Unique Number को उस Storage Cell का Address कहते हैं।

जिस तरह से हम किसी घर में कई तरह के सामान रखते हैं और जरूरत होने पर उस घर से उस सामान को प्राप्त करके काम में ले लेते हैं, उसी तरह से Memory में भी अलग-अलग Storage Cells में हम अपनी जरूरत के अनुसार अलग-अलग मान Store कर सकते हैं और जरूरत पड़ने पर उस Data को प्राप्त कर के काम में ले सकते हैं।

एक उदाहरण द्वारा हम Computer की कार्यप्रणाली को समझने की कोशिश करते हैं। जब हम Keyboard (Input Device) पर कोई Key Press करते हैं, तो CPU का Control Unit Active हो जाता है और Input हो रहे Character को Memory में किसी Storage Location पर Store कर देता है।

माना हम Keyboard से एक अंक 6 Input करते हैं, तो CPU का Control Unit इस मान को Memory की किसी Storage Location पर Store कर देता है। अब हम + का चिन्ह Keyboard से Press करते हैं। Control Unit उसे भी Memory की किसी Storage Cell में Store कर देता है। अब हम 4 Input करते हैं। Control Unit इसे भी Memory में Store करता है। माना कि ये तीनों मान निम्नानुसार Memory की विभिन्न Storage locations पर Save हो रही हैं –

1001	1002	1003	1004	1005	1006	1007	1008	1009	1010
6		+			4			1	0

अब यदि Input किए गए दोनों अंकों को जोड़ना हो तो Control Unit Input किए गए पहले मान को Storage Cell 1001 से उठाता है और उसे Binary Format में Convert करके CPU के Registers में भेज देता है। फिर इस अंक पर Processing करने के लिए Control Unit को Memory Location 1003 से उठाता है और CPU को इस अंक को जोड़ने की प्रक्रिया करने की Instruction देता है।

फिर Control Unit दूसरे मान 4 को Binary Format में Convert करके CPU के Register में भेज देता है और ALU को इन दोनों अंकों को जोड़ने की सूचना देता है। ALU Registers से इन Data को प्राप्त करके उन्हें जोड़ता है और जोड़े गए मान को CPU के Registers में Store कर देता है। यहां से वापस Control Unit Registers में Stored मान को Memory में Storage Cell 1009 व 1010 पर Store कर देता है। इस तरह से दो संख्याओं को जोड़ने की प्रक्रिया पूरी होती है।

Computer एक Digital Machine है। Computer तभी कोई काम कर सकता है जब उसे किसी काम को करने के लिए Program किया गया हो। Programming दो तरह की होती है। पहली Programming वह होती है जो किसी Computer को काम करने लायक अवस्था में लाने के लिए की जाती है। इस Programming को भी दो भागों में बांटा जा सकता है:

## Hardware Programming

इस Programming के अन्तर्गत Computer के Hardware यानी Computer के Motherboard पर लगाए गए विभिन्न प्रकार के Chips व Computer से जुड़े हुए अन्य विभिन्न प्रकार के Peripherals जैसे कि Keyboard, Mouse, Speaker, Monitor, Hard Disk, Floppy Disk,



CD Drive आदि को Check करने व Control करने के लिए हर Mother Board पर एक BIOS Chip लगाई जाती है। इस BIOS Chip का मुख्य काम Computer को ON करते ही विभिन्न प्रकार के Devices को Check करना होता है।

यदि Computer के साथ जुड़ी हुई कोई Device ढंग से काम नहीं कर रही है, तो BIOS User को विभिन्न प्रकार की Error Messages देता है। BIOS Chip के अन्दर ही प्रोग्राम को लिखने का काम BIOS बनाने वाली Company करती है। इसे Hard Core Programming या Firmware कहा जाता है। Hardware Programming में Chip को बनाते समय ही उसमें Programming कर दी जाती है। किसी भी Computer के Motherboard पर लगी BIOS Chip यदि खराब हो जाए, तो Computer किसी भी हालत में काम करने लायक अवस्था में नहीं आ सकता यानी Computer कभी Boot नहीं होता।

## Software Programming

Computer को काम करने लायक अवस्था में लाने के लिए जिस Software को बनाया जाता है, उसे Operating System Software कहा जाता है। BIOS Chip का काम पूरा होने के बाद Computer का पूरा Control Operating System Software के पास आ जाता है। Computer के पास BIOS से Controlling आने के बाद सबसे पहले Memory में Load होने वाला Software Operating System Software ही होता है। इसे Master Software या Operating System Software भी कहते हैं।

आज विभिन्न प्रकार के Operating System Software बन चुके हैं जैसे DOS, Windows, OS/2, WRAP, Unix, Linux आदि। इन सभी Software का मुख्य काम Computer को Boot करके User के काम करने योग्य अवस्था में लाना होता है।

दूसरी Programming वह Programming होती है, जिससे Computer हमारी बात को समझता है और हमारी इच्छानुसार काम करके हमें परिणाम प्रदान करता है। इन्हें Application Software कहा जाता है। हम किसी भी Operating System के लिए किसी भी भाषा में जब कोई Program लिखते हैं, तो वास्तव में हम Application Software ही लिख रहे होते हैं।

Application Software का मुख्य काम किसी विशेष समस्या का समाधान प्रदान करना होता है। MS-Office, Corel-Draw, PageMaker, PhotoShop आदि Application Software के उदाहरण हैं, जो हमें किसी विशेष समस्या का समाधान प्रदान करते हैं। जैसे यदि हमें Photo Editing से सम्बंधित कोई काम करना हो तो हम PhotoShop जैसे किसी Application Software को उपयोग में लेते हैं।

## Language

भाषा, दो व्यक्तियों के बीच संवाद, भावनाओं या विचारों के आदान-प्रदान का माध्यम प्रदान करती है। हम लोगों तक अपने विचार पहुंचा सकें व अन्य लोगों के विचारों का लाभ प्राप्त कर सकें, इसके लिये जरूरी है कि संवाद स्थापित करने वाले दोनों व्यक्तियों के बीच संवाद का माध्यम समान हो। यही संवाद का माध्यम भाषा कहलाती है। अलग-अलग स्थान, राज्य, देश, परिस्थितियों के अनुसार भाषा भी बदलती रहती है, लेकिन सभी भाषाओं का मकसद संदेशों या सूचनाओं का आदान प्रदान करना ही होता है।

ठीक इसी तरह कम्प्यूटर की भी अपनी कई भाषाएं हैं, जो जरूरत व उपयोग के अनुसार विकसित की गई हैं। हम जानते हैं, कि कम्प्यूटर एक इलेक्ट्रॉनिक मशीन मात्र है। ये हम सजीवों की तरह सोच विचार नहीं कर सकता है और ना ही हमारी तरह इनकी अपनी कोई भाषा है, जिससे हम इनसे सम्बंध बना कर सूचनाओं का लेन-देन कर सकें।

इसलिये कम्प्यूटर को उपयोग में लेने के लिये एक ऐसी भाषा की जरूरत होती है, जिससे हम हमारी भाषा में कम्प्यूटर को सूचनाएं दें व कम्प्यूटर उसे उसकी मशीनी भाषा में समझे और हमारी चाही गई सूचना या परिणाम को हमें हमारी भाषा में दे ताकि हम उसे हमारी भाषा में समझ सकें।

Java एक ऐसी ही भाषा है। ये एक High Level Language है। इस Language में हम सामान्य English के शब्दों में Code लिखते हैं। इन Codes को Java का एक Software Program जिसे Compiler कहते हैं, उस भाषा में Convert करता है, जिसे Computer समझ सकता है। इस Software की मदद से Compute हमारे लिखे गए Codes के अनुसार काम करता है और हमें वह परिणाम प्रदान करता है, जो हम Computer से चाहते हैं।

Java का Program लिखने के लिए एक Simple Word Processor की जरूरत होती है। Windows के साथ आने वाले Notepad को हम Java का Program लिखने के लिए Use कर सकते हैं। Java में हर Program को एक Class के रूप में लिखा जाता है। Program लिखने के बाद Source File को .Java Extension के साथ Save किया जाता है।

Java में दो तरह के Programs सबसे ज्यादा लिखे जाते हैं। जब हम Java Application लिखते हैं तो उस Application में हमेशा एक **main()** Method होता है। Java एक Case Sensitive Language है इसलिए Capital Letters में लिखा गया नाम व Small Letters में लिखा गया नाम दोनों नाम अलग-अलग होते हैं।

## Java Compiler (javac)

Java Compiler (Javac) Java Developer's Kit का एक Component है, जिसका प्रयोग Java की Source Code File को Bytecodes Executable File में Convert करने के लिए किया जाता है, ताकि वह File Java Runtime Environment (JRE) System में Run हो सके। Java की Source Code File का Extension .java होता है।

Java की Source Code File एक Standard ASCII Text File होती है। ये Java के Compiler का काम होता है कि वह Java Source Code File को Process करे और Executable Java Bytecodes Class File Create करे। Executable Bytecodes Class File का Extension .class होता है और ये अपनी Useable Form में Java Class को Represent करते हैं।

Java Compiler हमारी Source File की हर Class के लिए एक .Class File Generate करता है। तकनीकी रूप से हम एक ही Source File में एक से अधिक Class Define कर सकते हैं लेकिन Compiler एक ही Source File में Define की गई एक से अधिक Classes के लिए भी अलग-अलग .class File Create करता है।

Java Compilers इस बात के लिए जिम्मेदार होते हैं कि एक Java Source File से Java Executable Bytecodes File किस प्रकार से Generate की जाए जो कि Java Runtime System में Run हो सकें और Java Virtual Machine, जो कि Java Runtime System का

एक हिस्सा है, इस बात के लिए जिम्मेदार होता है, कि Bytecodes को किस प्रकार से Interpret किया जाए।

Java Compiler एक Command Line Tool है। इसका मतलब ये हुआ कि इन्हें DOS Prompt पर ही Execute किया जा सकता है। इनको Use करने का Syntax निम्नानुसार होता है:

```
Command Prompt > javac Options Filename
```

इस Command में Filename Argument उस Java Source Code File का नाम होता है जिसे Compile करना है। इस File में जितनी भी Classes Define की गई होती हैं, उन सभी की Bytecodes वाली .class File बनती है। ये Compiler उन सभी Classes को Bytecodes Class File में Convert कर देता है जो एक दूसरे पर Depend होती हैं।

यानी मानलो कि एक File *X.Java* है और इस File में एक दूसरी File जिसका नाम *B.Java* है, की Class को Derive किया गया है। तो इस स्थिति में हम यदि *B.Java* को Compile करते हैं तो Compiler *X.Java* की भी Compiling करेगा और दोनों फाइलों के सभी Classes की Bytecodes (.Class) File Generate करेगा।

## Java Interpreter (java)

Java Runtime Interpreter (Java) भी Java Developer's Kit का एक Component है जिसका प्रयोग Executable Java Bytecodes Classes को Run करने के लिए किया जाता है। Java Interpreter एक तरीका प्रदान करता है जिससे Java के Programs को किसी Web Browser के बिना Execute किया जा सकता है। Java Compiler जिस Class Format की Executable Bytecodes File को Generate करता है, उस Bytecodes File को Java Interpreter द्वारा Execute किया जाता है।

किसी Java Applet व Java Application Program में इतना ही अन्तर है कि Applet किसी ना किसी Java Enabled Web Browser में Run होते हैं, जबकि Application का स्वयं का Window होता है। किसी Compile किए गए Java Source File की Bytecodes (.Class) File को हम निम्नानुसार Syntax द्वारा Command Prompt पर Run कर सकते हैं—

```
Command Prompt>Java ClassName Arguments
```

यहां *ClassName* उस Class File को Specify करता है, जिसे Execute करना है। यदि ये Class File किसी Package में हो तो हमें Class File का पूरा नाम देना होता है। जैसे मानलो कि कोई Hello.class नाम की Class File India नाम के Package में है, तो हमें इस File को Run करने के लिए निम्नानुसार Command लिखना होगा:

```
Command Prompt>Java India.Hello
```

जब Java Interpreter किसी Class को Execute करता है तो वह वास्तव में उस Class के main() Method को Execute कर रहा होता है। main() Method में हम कुछ Arguments भी दे सकते हैं जिनका प्रयोग main() Method के Execution को Control करने के लिए किया जाता है।

## Structure of Java Programs

किसी जावा के Program में हम आवश्यकतानुसार एक से अधिक Classes को Define कर सकते हैं। लेकिन फिर भी हम एक Program में किसी एक Class में ही main() Method को Define कर सकते हैं। विभिन्न Classes में विभिन्न प्रकार के Data और उन Data पर Perform होने वाले Operations को Define किया जाता है।

जावा का Program बनाने के लिए हम सबसे पहले विभिन्न प्रकार की Classes बनाते हैं और फिर उन सभी Classes को Combine कर लेते हैं। एक जावा Program के विभिन्न विभागों को हम निम्नानुसार दर्शा सकते हैं :

```
Documentation Section
Package Statements
Import Statements
Interface Statements
Class Definitions
{
    Data Members
    Methods

    Main Method Class
    {
        Main Method Definitions
    }
}
```

## Documentation Section

किसी भी Program के इस Section में हमें Program से सम्बंधित Descriptions देने होते हैं। जैसे कि Author का नाम, Program Creation की Date, Program Create करने का कारण, Program की उपयोगिता आदि से सम्बंधित बातें हमें Comment के रूप में Documentation Section में लिखनी होती हैं।

हालांकि ये एक Optional Section है, फिर भी Program को Readable बनाने के लिए हमें हर Program को Well Documented करना चाहिए। यहां हमें ये बताना चाहिए कि हमने Program में किन Classes को Use किया है और किस कारण से Use किया है।

साथ ही Program के Algorithm यानी Flow को भी सारांश रूप में Describe करने की कोशिश करनी चाहिए। सामान्यतया Documentation को `/** . . . */` Format में लिखना चाहिए। इस तरह से Comment करने पर हम Documentation की एक File JDK Tools के उपयोग द्वारा Generate कर सकते हैं।

## Package Statements

Package Statement किसी भी जावा Program का पहला Statement होता है। ये Statement एक Package का नाम Declare करता है और जावा Program को बताता है कि Program में Use होने वाली Classes इन Packages में Defined की गई हैं। किसी Package को हम निम्नानुसार जावा के Source Program में Use कर सकते हैं :

```
package StudentInformation;
```

ये Statement StudentInformation नाम के Package को उस Program या Source File में Include कर लेगा, जिसमें ये Statement लिखा गया होगा।

## Import Statements

किसी Package Statement के बाद जो अगला Statement हो सकता है वह **import** Statements होता है। पहले से बनी हुई किसी Class को जावा के Source Program में Include करने के लिए हमें जावा के **import** Keyword के साथ उस Class का नाम लिखना होता है, जिसे अपने Program में Use करना होता है। ये काम हम निम्नानुसार Statement द्वारा कर सकते हैं :

```
import StudentInformation.testMarks;
```

ये Statement जावा Interpreter को testMarks नाम की Class को Source File में Include करने का Message देता है, जो कि StudentInformation नाम के Package में उपलब्ध है और इस package का एक हिस्सा है।

## Interface Statements

Interface एक Class के समान ही होता है लेकिन इसमें Methods के Declarations का एक Group होता है। ये भी एक Optional Statement है और इसका प्रयोग हम तभी करते हैं जब हमें अपने Program में Multiple Inheritance की सुविधा प्राप्त करनी होती है। इस विषय में हम आगे विस्तार से पढ़ेंगे।

## Main Method Class

जावा के हर Application Program में हमें एक Main() Method की जरूरत होती है, जिसको जावा Compiler सबसे पहले Execute करता है। ये किसी भी जावा Application Program का एक बहुत ही जरूरी हिस्सा है। किसी Simple जावा Program में केवल ये एक Method भी हो सकता है।

Main Method विभिन्न प्रकार की Classes के Objects Create करता है और उनके बीच में Communication को Establish करता है। Main Method के अन्त में पहुंचने पर Program Terminate हो जाता है और Program Control पुनः Operating System को Transfer हो जाता है।

## Definition – The Applet and Application

जावा में हम दो तरह के Programs Develop कर सकते हैं। पहले Program वे Program होते हैं जो किसी एक Computer पर Execute होते हैं और किसी समस्या का समाधान प्रदान करते हैं। ये Programs जिस Computer पर Store होते हैं उसी Computer पर Run होते हैं और उस Computer के Resources को Use करते हैं।

इस प्रकार के Programs को **Application Programs** कहते हैं। जबकि जावा में जिस दूसरे प्रकार के Program को Develop किया जा सकता है, उन Programs को Run होने के लिए जावा Platform Enabled Web Browser की जरूरत होती है। यानी ये वे Programs होते हैं जो किसी ना किसी Web Page में Run होते हैं और इन्हें उपयोग में लेने के लिए हमें Web Browser की जरूरत होती है। जावा में Develop किए जाने वाले इस दूसरे प्रकार के Programs को **Java Applets** कहते हैं।

## Java - Applications

जावा इतना सक्षम है कि हम इसमें Professional Applications Develop कर सकते हैं। हम इसमें विभिन्न प्रकार के Window Based Applications बना सकते हैं और उन्हें सफलता पूर्वक उपयोग में ले सकते हैं।

लेकिन चूंकि जावा एक Interpreted Language है, इसलिए इसके Programs की Speed "C" व "C++" Languages में Develop किए गए Applications की तुलना में कम होती है। इसलिए सामान्यतया जावा Based Applications तभी बनाए जाते हैं, जब Application Distributed हो व Network पर Execute होता हो।

क्योंकि जावा Internet के लिए एक बहुत ही अच्छी व Powerful Programming Language है, इसलिए इसमें Desktop Applications नहीं बनाए जाते हैं। क्योंकि Desktop Applications के लिए Visual Basic, "C" व "C++" Programming Languages काफी अच्छी Languages हैं, इसलिए जावा का प्रयोग सामान्यतया Web Pages के Applets बनाने के लिए किया जाता है, जिसके बारे में हम आगे विस्तार से समझेंगे।

## First Application in Java

किसी भी Language में Programming सीखने के लिए हम सबसे पहले Hello World Program से शुरुआत करते हैं। ये Program कुछ नहीं करता है, केवल Screen पर "Hello World" Message Print करता है, लेकिन इस Program के द्वारा हम Language के Program की Basic चीजों को Clear करते हैं।

### "Hello World" Application

सबसे पहले नीचे लिखे गए Program को Notepad में Type करें:

---

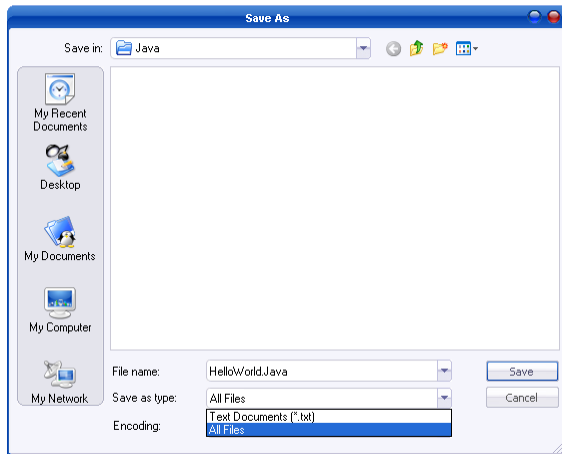
#### "Hello World" Application Program

---

```
class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");           //Display this String
    }
}
```

---

Notepad में इस Coding को Type करने के बाद File Menu से Save Option को Select करें। निम्नानुसार एक Save As Dialog Box Display होगा:



इस Dialog Box में **Save as type** Drop Down List में **All Files** को Select करें, क्योंकि यहां हमेशा Default रूप से Text Documents (\*.txt) Option Selected रहता है। अब **Save in** Drop Down List से उस Folder को Select करें, जहां पर Java की File को Save करना है और **File name** Text Box में File का वही नाम लिखें जो नाम Source File में उस Class का है जिसमें main() Method लिखा गया है।

चूंकि हमारे इस Example की Source File में **main()** Method को **HelloWorld** Class में लिखा गया है इसलिए हमारी Source File का नाम हमें HelloWorld ही देना होगा साथ ही हमारी Source File, Java की एक Program File है, इसलिए इसे **.java** Extension के साथ Save करना होगा। इस प्रकार से हमारी इस File का नाम **HelloWorld.java** होगा।

ऐसा नहीं है कि हम इस Source File का वही नाम दें जो हमने Source File में Class का नाम दिया है। हम File का नाम कुछ भी दे सकते हैं, लेकिन जब File को Compile किया जाता है, तो एक Class File बनती है। इस Class File का नाम बिल्कुल वही होता है, जो उस Source File की उस Class का नाम होता है, जिसमें main() Method लिखा गया होता है।

इसलिए किसी भी तरह के Confusion या परेशानी से बचने के लिए ठीक यही रहता है, कि हम हमेशा हमारी Source File को भी उसी नाम से Save करें, जिस नाम से हम वह Class बनाते हैं, जिसमें main() Function होता है।

## Compiling Java Source File

Source File Create करने के बाद अब हमें इस Source File को Compile करना होता है, ताकि ये File Byte Codes में Convert हो सके। Compile करने के लिए हमें Command Prompt या DOS Prompt पर जाना होता है और निम्नानुसार Command लिखने होते हैं—

```
Command Prompt \>cd\      ( Press Enter )
C:\>path=C:\jdk\bin\     ( Press Enter )
```

हम ये मान रहे हैं कि जब Java को Install किया जाता है तब Java का 7<sup>th</sup> Version C: Drive पर **jdk** नाम के Folder में Install होता है। यहां पहले Command से हम Root Directory यानी C: Drive पर आ जाते हैं और दूसरा Statement Path Statement है।

हम किसी फाईल को वहीं से Run कर सकते हैं, जहां वह File रखी हुई है। लेकिन DOS के Path Command की ये विशेषता है कि हम किसी File को वहां से Directly Run कर सकते हैं, जहां का Path हम Command Prompt पर Path Statement द्वारा लिख देते हैं।

हम जानते हैं कि Java के Bin नाम के Folder में ही Java के सभी जरूरी Tools होते हैं। इसलिए हम चाहते हैं कि हम कहीं से भी Java के Commands को Execute कर सकें और हम ऐसा तभी कर सकते हैं जब Computer को पहले ही ये बताया जाए कि उसे Command Prompt पर दिए जाने वाले विभिन्न Command को कहां पर खोजना है।

चूंकि हम चाहते हैं कि हम किसी भी स्थान पर Java का जो भी Command लिखें, वह Execute हो, तो हमें **path=C:\jdk\bin\** लिखना होता है। ये Path Set किए बिना आगे लिखे गए कोई Command नहीं चलेंगे। इसलिए सबसे पहले इसी Command को लिख कर Path Set किया गया है। हम जितनी बार भी Command Prompt पर आते हैं, हमें ये Path Set करना पड़ता है।

लेकिन यदि हमें हमेशा के लिए इस Path को Permanently Set करना हो, तो Command Prompt Open करके हम निम्न Command Fire कर सकते हैं:

```
CommandPrompt>setx PATH "c:\jdk\bin"
```

ये Command Fire करते ही हमारा Path, Windows Operating System की tmहेजतल में कक हो जाता है। परिणामस्वरूप अब हम जब भी कभी ब्वउउंदक च्त्वउचज व्वमद करते हैं, हमें हमारे च्जी को फिर सेमज करने की जरूरत नहीं होती।

Path Set करने के बाद हमें उस Folder में जाना होता है, जहां हमने हमारी Source File को Save की है। मानलो कि हमने हमारी Source File को C: Drive में Java नाम का एक Folder बना कर उसमें Save की है, तो हम निम्न Command द्वारा अपने Java Folder तक जाते हैं—

```
C:\>cd java      (Press Enter )  
C:\Java>
```

अब हमें HelloWorld.Java को निम्नानुसार Command लिख कर Compile करना होगा—

```
C:\Java>javac HelloWorld.Java      ( Press Enter )
```

यदि Source File में किसी प्रकार की Error ना हो, तो Java की File Compile हो जाएगी। जब Java की File Compile हो जाती है, तब Java की File की एक Class File बनती है, जिसका नाम वही होता है जो हमारी Source File का होता है, लेकिन उस File का Extension .class होता है। ये Class File Java की Bytecodes File होती है, जो कि Platform Independent होती है और किसी भी Java Runtime Environment में Interpret होती है।



## Running Java Application

जब Source File ठीक से बिना किसी Error के Compile हो जाती है, तब उसकी एक Class File बनती है। ये Class File ही Java Runtime Environment में Run होती है। अब HelloWorld Application को Run करने के लिए हमें Command Prompt पर निम्न Command लिखना होता है—

```
C:\Java>java HelloWorld ( Press Enter )
```

अब हमें File का Extension Use करने की जरूरत नहीं होती है। जावा का ये Interpreter सही File को स्वयं ही Run करता है और हमारे Screen पर “Hello World” Display होता है। यहां ये बात ध्यान रखने वाली होती है कि Source File का नाम हम चाहे जैसा भी दें लेकिन File को Run करते समय हमें वही नाम देना पड़ता है, जो नाम Class का होता है और ठीक उसी तरह से Class का नाम लिखना पड़ता है, जिस तरह से Source File में हमने Class का नाम लिखा है। यदि हम नाम लिखने में जरा सी भी Mistake करते हैं, तो हमें निम्नानुसार Error Message प्राप्त होता है—

```
C:\JAVA>java helloworld
Exception in thread "main" java.lang.NoClassDefFoundError: helloworld
(wrong name: HelloWorld)
```

## Anatomy of Java Application

हमने Java में एक छोटा सा Program Develop किया और उसे Compile करके Run भी कर लिया। अब हम इसकी बनावट व इसके काम करने के तरीके को समझते हैं। सबसे पहली बात तो यही है कि Java Application Standalone Java Programs होते हैं। इन्हें Java Applets की तरह किसी Java Enabled Web Browser की जरूरत नहीं होती है।

## Comments in Java

Java में Comments देने के लिए तीन तरीकों का प्रयोग किया जा सकता है। तीनों तरीकों से निम्नानुसार Comment दिया जा सकता है —

```
/* This is a Comment in Java which Java Compiler Ignores. */
```

इस /\* \*/ के बीच लिखे गए Comments को Java का Compiler पूरी तरह से Ignore कर देता है। यानी इनके बीच लिखे गए किसी भी Word को Compiler Check नहीं करता है ना ही Compile करता है। ये Comments एक Programmer अपने Program के बीच-बीच में देता रहता है, ताकि उसे उसके Program का Flow पता रहे।

```
/** Documentation */
```

इस Comment को Documentation Comment या Doc Comment कहा जाता है। जब हम हमारी किसी Class File का Documentation करना चाहते हैं तो Documentation को /\*\* . . . \*/ के बीच में लिखते हैं। इस प्रकार के Documentation को Java Developer's Kit के Javadoc Tool का प्रयोग करके एक अलग Documentation HTML File में Store किया जा सकता है। इस Documentation को अलग File में Store करने के लिए हम Command Prompt पर Javadoc Tool का निम्नानुसार उपयोग करके अपनी Class के लिए Documentation File तैयार कर सकते हैं।

```
Command Prompt> Javadoc <List of .Java FileNames>
```

```
// This is Third Comment Style
```

जब हमें केवल एक ही Line का Comment देना होता है, तब हम इस तरीके से Comment लिखते हैं। Compiler इस प्रकार के Comments को Ignore कर देता है यानी Compile नहीं करता है। एक बात हमेशा ध्यान रखें कि एक Comment के अन्दर दूसरा Comments नहीं लिखा जा सकता। यानी Comments की Nesting नहीं की जा सकती है। जैसे

```
/** Madhav is a Union Leader */ and I am his Friend*/ He is a Programmer too.*/
```

Java में किसी भी Application Program में कम से कम एक Class जरूर होती है और उस Class में एक main() Method होता है। Java का Compiler इसी main() Method से Java के Application की Compiling शुरू करता है।

### Class Statement

इस Program की पहली Code Line को देखिए:

```
class HelloWorld {
```

जब इस Program को Compile किया जाता है तब ये Code Line Compiler को ये Instruction देता है कि

```
|| "Computer, Java के इस Program का नाम HelloWorld Assign कर दे।" ||
```

जैसाकि हमने पहले भी बताया था कि एक Program के रूप में हम Computer को जितने भी Instructions देते हैं, वे सभी Statements कहलाते हैं। यहां class Statement वह तरीका है, जिससे हम हमारे Program को एक नाम देते हैं। ये Statement कई अन्य बातों को भी Determine करता है जिसके बारे में हम आगे पढ़ेंगे। class शब्द का सबसे अधिक महत्व ये है कि Java के Programs को भी Class ही कहा जाता है।

इस उदाहरण में हमारे Program का नाम HelloWorld बिल्कुल उसी तरह से लिखा गया है, जिस तरह से हमने Source File में हमारी Class का नाम लिखा है। जैसाकि हमने पहले भी बताया है कि यदि ये नाम समान नहीं होते हैं तो Compilation के समय Error Generate होता है।

### main() Statement

Program की अगली Line को देखिए

```
public static void main (String[] arguments) {
```

ये Line Computer को कहती है कि

```
|| "Program का Main Part यहां से शुरू हो रहा है।" ||
```

Java का Program कई Sections में Organized रहता है, इसलिए हमें एक ऐसे तरीके की जरूरत होती है जिससे ये पता चल सके कि Program का कौनसा हिस्सा पहले Handle होगा और

कौनसा बाद में। यहां से आगे हम जितने भी Application Programs Create करेंगे, उन सभी में main Part ही Starting Point होगा।

## Bracket Mark

हम देख सकते हैं कि हमारे Program में Brackets का भी प्रयोग हुआ है। ये Bracket किसी भी Java Program का बहुत ही महत्वपूर्ण हिस्सा हैं। Java में ये Bracket Program के विभिन्न हिस्सों को Group करने का तरीका है। Bracket की तरह ही Parenthesis का भी प्रयोग किया जाता है।

Opening Bracket व Closing Bracket के बीच जो कुछ भी लिखा जाता है, वह सबकुछ उस Group का हिस्सा होता है। Opening व Closing Bracket के बीच लिखे Statements को Block कहा जाता है।

हम हमेशा हमारे Program के विभिन्न हिस्सों की शुरुआत को दर्शाने के लिए Opening Bracket “{” का प्रयोग करते हैं और किसी विशेष हिस्से का अन्त दर्शाने के लिए Closing Bracket “}” का प्रयोग करते हैं।

एक Block के अन्दर दूसरे Block को Use किया जा सकता है, जैसाकि हमारे Program में HelloWorld Class के Block में Main() Method के Block Use किया गया है। main() Method Block के अन्दर जो भी Statement लिखे जाते हैं, वे सभी Statements Computer के लिए Instructions या Commands होते हैं जिन्हें Computer Execute करता है।

हमने main() Method में निम्नानुसार केवल एक ही Statement लिखा है—

```
System.out.println("Hello World");           //Display this String
```

ये Statement Screen पर उस Data या Information को Print कर देता है, जिसे इस Statement के Parenthesis में लिखा जाता है। जैसे यदि हम “Hello World” के स्थान पर “Java Is a Web Language” लिख दें, तो Hello World के स्थान पर यही Statement Screen पर Print होगा। जब किसी Characters के एक समूह को Double Quote के बीच लिखा जाता है, तो इस Characters के समूह को **String** कहा जाता है।

यानी “Hello World” व “Java Is A Web Language” दोनों Strings हैं, क्योंकि ये Characters का एक समूह हैं जिन्हें **Double Quote** के बीच लिखा गया है।

इस तरह से हमने Java का एक साधारण सा Program बनाया। हमें इसी तरह से जावा में विभिन्न Programs Develop करने होते हैं। हम चाहे जो भी Program बनाना चाहें, हमें निम्न Statements तो हमेशा लिखने ही होते हैं जो Compiler को बताता है कि Program की शुरुआत व अन्त कहां पर हो रहा है।

```
public class ClassName           //Name of the Class of the Java Program
{
    //Starting of Program Level Class Block
    //Other program specific statements
}
//End of the Program Level Class Block
```

Javac Compiler से हम किसी भी Java Class को Compile कर सकते हैं, लेकिन java Interpreter से वही Class Command Prompt पर Run होती है, जिसमें main() Method होता

है। इसलिए जिस Class में main() Method होता है, उस Class का नाम ही उस Class की Source File का नाम भी होना जरूरी होता है।

चूंकि जावा एक Highly Case Sensitive Language है, इसलिए जिस प्रकार का नाम हम Class का लिखते हैं, उसी तरीके का नाम हमें Source File का भी रखना होता है। साथ ही Source File का नाम हमेशा .java Extension के साथ लिखा जाता है, तभी javac Compiler उसे Compile कर सकता है। हमें हमारे Program के जितने भी Codes लिखने होते हैं, वे सभी Codes या Statements हमें Class Block के अन्दर ही लिखने होते हैं।

## Java – Applet

जावा Applet Internet Computing में Use किए जाने वाले छोटे-छोटे Programs होते हैं। इन Programs को एक Computer से दूसरे Computer में Internet पर Transport किया जा सकता है और Applet Viewer या किसी Java Enabled Web Browser द्वारा Run किया जा सकता है। किसी भी अन्य Application Program की तरह ही एक Applet भी हमारे लिए कई जरूरी काम कर सकते हैं।

उदाहरण के लिए Applet विभिन्न प्रकार के Arithmetical Operations Perform कर सकते हैं, Applet में विभिन्न प्रकार के Graphics Display हो सकते हैं जो कि Web Page पर दिखाई देते हैं, ये Applets विभिन्न प्रकार के Sounds Play कर सकते हैं और Browser में विभिन्न प्रकार के Animations Display कर सकते हैं। यानी एक Web Browser में Java Applets को Use करके हम किसी Web Site में Multimedia को Use करके Web Page को अधिक उपयोगी, Informative, सरल, Interactive व Dynamic बना सकते हैं।

जावा के कारण ही हम Internet के आज के स्वरूप को देख सकते हैं जिसमें ढेर सारा Multimedia Use किया जा सकता है। अब एक Web Page में ना केवल Simple Text व Static Image होता है, बल्कि Web Pages में हम आज विभिन्न प्रकार के Multimedia जैसे कि Sound, Animation, Videos व Graphics को भी देख सकते हैं।

जावा Applets Internet Server से Access होते हैं और Internet पर Transported होते हैं। यानी ये वे Software होते हैं जो Internet पर चलते हैं। जब User किसी Web Page के Link को Click करता है, तब उस Web Page में यदि कोई Java Applet हो, तो वह Applet Automatically Internet Server से Access हो कर User के Computer में Installed हो जाता है और Web Page के एक हिस्से की तरह Run होता है।

जब एक Applet Client के Computer पर आ जाता है, तो वह Applet User के Computer के Resources को एक सीमा में रहते हुए ही Access करता है, ताकि ये एक उचित User Interface प्रदान कर सके, विभिन्न प्रकार की जटिल Computations कर सके और Client के Computer व Data को Viruses से किसी प्रकार का खतरा ना हो।

## Applet – Local and Remote

किसी Applet को हम Web Page में दो तरीकों से Embed कर सकते हैं। पहले तरीके में हम हमारा स्वयं का Web Page बनाते हैं और उसमें Applet को Embed कर देते हैं। दूसरे तरीके में हम किसी Applet को किसी Remote Computer System से Download करते हैं और फिर उसे अपने Web Page में Embed कर लेते हैं।

एक ऐसा Applet जिसे Local Computer पर Develop किया गया हो और उसी Local System पर उस Applet को Store किया गया हो, तो इस प्रकार के Applet को **Local Applet** कहते हैं। जब एक Local Web Page किसी Local Applet को खोजता है, तब उसका Internet से Connected रहना जरूरी नहीं होता है। ऐसा Web Page सामान्यतया Local Computer की विभिन्न Directories को Search करके Applet को खोजता है और Web Page में Load कर देता है।

Remote Applet ऐसा Applet होता है जिसे किसी अन्य Developer ने Develop किया होता है और किसी ऐसे Computer System पर Store किया होता है जो कि Internet से Connected होता है। यदि हमारा Computer Internet से Connected हो तो हम उस Remote Applet को Internet द्वारा Download करके अपने Local Computer System पर Run कर सकते हैं।

किसी Remote Computer पर रखे Remote Applet को खोजने व अपने Computer System के Web Browser में Load करने के लिए हमें उस Applet के Web Address की जानकारी होना जरूरी होता है। किसी Applet के इस Web Address को **Uniform Resource Locator (URL)** कहते हैं और इस Address को हमें हमारी HTML File में APPLET Tag के CODEBASE Attribute के एक मान के रूप में Specify करना होता है। हम किसी Remote Applet को अपनी HTML File में निम्न Code Statement द्वारा Specify कर सकते हैं:

**CODEBASE = [http:// www.achyut.com/applets](http://www.achyut.com/applets)**

लेकिन जब हम Local System पर किसी Applet को Locate करना चाहते हैं, तब इस CODEBASE Tag में हमें हमारे Applet के Local Address यानी Applet का Directory Path देना होता है।

## Clients and Servers

यदि किसी User को किसी Applet की जरूरत है और वह Applet उसके Computer पर उपलब्ध नहीं है, तो वह User उस Applet को किसी Remote Computer से प्राप्त करके Use कर सकता है। उसे ये जानने की जरूरत नहीं होती है कि वह Applet किस Remote Computer पर स्थित है या किस तरह से बना है। एक तरह से देखा जाए तो एक User Internet द्वारा पूरी दुनिया के Computers से जुड़ जाता है और जिस प्रकार की सूचना चाहे उस तरह की सूचना को Internet से Applet के रूप में प्राप्त कर सकता है।

Information Technology (I.T.) की भाषा में कहें तो जिस Computer से User को सूचनाएं Applet या Web Pages के रूप में प्राप्त होती हैं, उस Remote Computer को **Server** कहते हैं और User का Local Computer जो कि Applet या Web Pages द्वारा किसी प्रकार की Information को Server से प्राप्त करना चाहता है, उसे **Client** कहते हैं।

इस तरह से एक Local Computer के **Browser** व एक Remote Computer से आने वाले **Applet** की Information के बीच में **Client/Server** की Relationship बन जाती है। इस स्थिति में **Client** वह होता है, जो Web Page के HTML Documents को अपने Local Computer पर **Download** करता है, जबकि **Server** वह होता है, जो Web Pages को Client के Computer पर **Upload** करता है।

User का Computer हमेशा Client नहीं होता है और Remote Computer हमेशा Server नहीं होता है। कब कौनसा Computer Client होगा और कब कौनसा Computer Server होगा, ये इस बात पर निर्भर होता है कि किसी समय कौनसा Computer किस तरह का काम कर रहा है।

उदाहरण के लिए जब एक User किसी Web Site का Address अपने Browser में Fill करके Internet से उस Web Page को देखना चाहता है, उस समय वह Web Page जिस Computer पर होता है, वह Remote Computer Server होता है और User का Local Computer Client होता है।

लेकिन जैसे ही Web Page User के Computer में Download हो जाता है और User किसी अन्य Web Page को Search करने के लिए किसी Search Engine में कुछ Texts Input करता है, वैसे ही User का Computer Server बन जाता है और File को Search करने वाला Program Client बन जाता है।

## ***Difference – Applet and Application***

Applications व Applets दोनों ही जावा Programs होते हैं, लेकिन दोनों ही प्रकार के Programs में कुछ अन्तर होता है। Applets कभी भी Full Featured Application Programs नहीं होते हैं। सामान्यतया इन्हें किसी छोटे से काम को पूरा करने के लिए लिखा जाता है। चूंकि इन्हें Internet पर Run करने के लिए Design किया गया है, इसलिए इनकी Design की कुछ सीमाएं हैं।

Applets में Codes के Initialization के लिए main() Method नहीं होता है। जब Applets Web Browser में Load होते हैं, तब Applet के Codes Automatically Applet Class के कुछ Methods को Call करके Applet को Start व Execute करते हैं।

Applications की तरह Applets स्वतंत्र रूप से Run नहीं हो सकते हैं। इन्हें Run होने के लिए Web Page की जरूरत होती है, जिसमें कुछ Codes लिख कर Applets को HTML Web Page में Embed किया जाता है।

Applets कभी भी Local Computer के Resources को पूरी तरह से Access नहीं कर सकते हैं। यानी Applets द्वारा हम किसी Local Computer की किसी File में Data को Write नहीं कर सकते हैं और ना ही Local Computer पर उपलब्ध किसी File के Data को Read कर सकते हैं।

Applets, Network पर उपलब्ध किसी अन्य Server से किसी प्रकार का कोई Communication नहीं कर सकते हैं।

Applets किसी Local Computer के किसी Program को Run नहीं कर सकते हैं।

## **Preparation – The Applet Writing**

जावा Applet Programs लिखने से पहले हमें इसके लिए कुछ तैयारियां कर लेनी चाहिए। सबसे पहले तो हमें ये तय करना होता है कि किसी Program में जावा Applet को कब Use करना चाहिए। इसका जवाब निम्नानुसार है:

- 1 जब हमें हमारे Web Page में कुछ Multimedia Operations को Perform करना हो व अपने Web Page को Dynamic व Interactive बनाना हो, तब हमें अपने Web Page में Java Applet को Use करना चाहिए।

उदाहरण के लिए Share–Market को ही लेते हैं। जब Trading हो रही होती है, तब हर क्षण विभिन्न Shares की कीमतों में Change होता रहता है। इन कीमतों के आधार पर विभिन्न प्रकार के Dynamic Graph बनाए जाते हैं।

इन विभिन्न प्रकार के Graphs को Computer में Web Page पर Dynamically तभी दर्शाया जा सकता है, जब हम Web Page में Multimedia को Use करें और Web Page पर Multimedia Use करने की सुविधा हमें जावा Applet द्वारा प्राप्त होती है।

- 2 जब हमें हमारे Web Page में कुछ Flash Outputs प्रदान करने होते हैं, तब भी हमें Java Applets को Use करना जरूरी हो जाता है।

जैसे मानलो कि हमारे Web Page को यदि कोई Hacker Hack करने की कोशिश करे, तो Web Page से एक विशेष प्रकार की Sound Play होनी चाहिए, और Web Page के कुछ Matters Blink होने चाहिए यदि हम हमारे Web Page में इस प्रकार की सुविधा चाहते हैं, तो हमें हमारे Web Page में Java Applet को Use करना चाहिए।

- 3 जब हम किसी Program को Develop करते हैं और उसे अन्य Users के उपयोग के लिए Internet पर उपलब्ध करवाना चाहते हैं, तब हमें जावा Applet का उपयोग करना चाहिए।

## System Package – Predefined (Built-In) Library of Java Classes

**Reusability** Object Oriented Programming System (OOPS) का एक महत्वपूर्ण Concept है। इस Concept के अन्तर्गत हम किसी काम को करने के लिए जिस Coding को एक बार लिख देते हैं, उस Coding का प्रयोग बार–बार कर सकते हैं।

जावा एक पूर्ण Object Oriented Programming Language है, इसलिए जावा में हम OOPS के इस Concept को पूरी तरह से Apply कर सकते हैं और विभिन्न प्रकार के Common Program Codes को इस तरह से लिख कर Store कर सकते हैं, जिन्हें बिना दुबारा लिखे या Modify किए ज्यों का ज्यों फिर से Use कर सकें।

जावा में इस काम को जिस Concept के आधार पर Perform किया जाता है, उसे **Package** कहते हैं। Package को हम ऐसी Classes की Library भी कह सकते हैं, जिन्हें बार–बार Use किया जा सकता है।

यदि दूसरे शब्दों में कहें तो कह सकते हैं कि Particular किसी एक काम को करने से सम्बंधित सभी Classes को यदि एक Group के रूप में परिभाषित करके Store कर दिया जाए, तो जावा में इस Classes के इस Group को **Package** कहते हैं। यानी एक Package एक ही काम से सम्बंधित विभिन्न प्रकार की Classes का एक Container होता है।

**Java API** (Application Programming Interface) Programmer को Application Development से सम्बंधित विभिन्न प्रकार के Predefined Components प्रदान करता है। इन विभिन्न प्रकार के Predefined Components (**Program Codes**) को उनके काम के आधार पर विभिन्न प्रकार के Group में विभाजित किया गया है, जिन्हें **Package** कहते हैं।

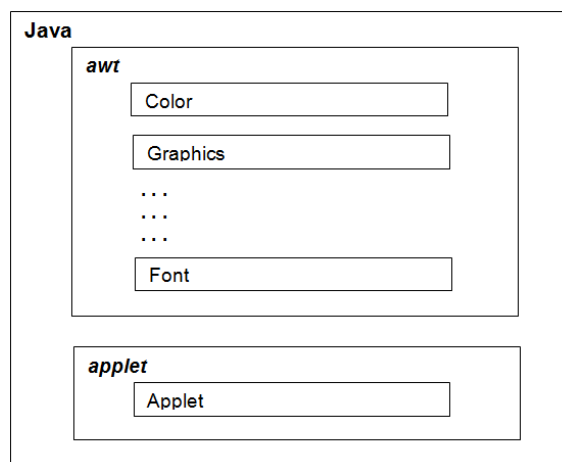
जावा में हम जो भी Program बनाते हैं, उनमें विभिन्न प्रकार की Predefined Classes को Use करते हैं, जो कि किसी ना किसी Package का हिस्सा होते हैं और ये सभी Packages Java API

के हिस्से हैं। इसलिए हम Directly या Indirectly Java API को Use करके ही अपना Program बनाते हैं।

यदि हम सारांश में कहें तो कह सकते हैं कि Java हमें कुछ Built-In या Predefined Classes प्रदान करता है, जिन्हें हम बिना किसी प्रकार का Modification किए ज्यों का त्यों अपने Program में Reuse कर सकते हैं। इन Predefined Classes को उनकी Functionality के आधार पर Group में Store किए गए हैं, जिन्हें Packages कहते हैं और ये विभिन्न प्रकार के Packages Java API के हिस्से हैं। इसलिए यदि हम किसी Java Package को अपने Program में Use करना चाहें, तो हमें उस Package को अपने Program में Use करना जरूरी होता है।

## Using – The System Packages

जावा में विभिन्न प्रकार के Packages को Hierarchical Structure में Organize किया गया है। विभिन्न प्रकार की Classes विभिन्न प्रकार के Packages में Stored रहते हैं और विभिन्न प्रकार के Packages Java API नाम के Java Platform के हिस्से हैं। इस निम्नानुसार चित्र द्वारा सरलता से समझ सकते हैं:



विभिन्न प्रकार की Required Classes को उनके Package से अपने Program में Reuse करके Access करने के दो तरीके हैं। पहला तरीका ये है कि अपने Program में जिस Class की जरूरत है उसी Class को अपने Program में Include किया जाए। केवल Required Class को ही अपने Program में Include करने के लिए हमें उस Class का पूरा Qualified नाम लिखना जरूरी होता है।

उदाहरण के लिए मान लो कि हमें हमारे Program में **awt** Package के केवल Color Class व Font Class की जरूरत है, तो केवल इन दोनों Class को अपने Program में Reuse करने के लिए हमें निम्न निम्नानुसार Statement लिखना होगा:

```
import java.awt.Color ;
import java.awt.Font ;
```

हम जानते हैं कि सभी प्रकार के Packages Java API का हिस्सा हैं, इसलिए हमें जिस किसी भी Package की Class को Use करना हो, Java के साथ उस Package के नाम को Dot Operator के साथ लिखना होगा।



हालांकि awt Package में कई Classes व Subclasses हैं, लेकिन हमें केवल Color Class के Codes व Font Class के Codes को ही अपने Source Program में Reuse करना है, इसलिए हमने Package के नाम के साथ उस Required Class के नाम को Dot Operator द्वारा Specify कर दिया है। हमारे Source Program में किसी Predefined Class के Codes को Include करने का काम **import** नाम का शब्द करता है, जो कि Java का एक Keyword या Reserve Word है।

## Keyword / Reserve Word

Keyword या Reserve Word कुछ ऐसे शब्द होते हैं, जिनका जावा Compiler के लिए विशेष अर्थ होता है। जब भी हम हमारे Program में किसी Keyword को Use करते हैं, जावा Compiler उस Keyword के लिए निर्धारित काम करने के लिए बाध्य हो जाता है।

उदाहरण के लिए यदि हम import Statement को ही लें, तो जावा Compiler को जैसे ही **import** Keyword प्राप्त होता है, वह उस Predefined Program Code को Source Program में Reuse कर लेता है, जिसका नाम इस Keyword के बाद लिखा गया होता है।

यदि हमें हमारे Program में Particular किसी एक ही Class को Use करना हो, तो उपरोक्त तरीके से उस Class को अपने Source Program में Reuse किया जा सकता है। लेकिन यदि हमें Particular किसी एक Class की नहीं बल्कि किसी Package की सभी Classes की अपने Source Program में जरूरत हो, तो हमें \* Operator का निम्नानुसार प्रयोग करना जरूरी होता है:

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

जब हम इस तरह का Statement लिखते हैं, तो इसका मतलब होता है, कि हम awt Package की सभी Predefined Classes को अपने Source Program में Reuse करने के लिए Import कर रहे हैं, जो कि स्वयं java नाम के Package में उपलब्ध है। इसी तरह से दूसरे Statement द्वारा हम applet Package की सभी Classes को अपने Program में बिना फिर से लिखे हुए Use कर रहे हैं, जो कि स्वयं java नाम के Package में उपलब्ध है।

इस तरह से हम दो तरीकों से किसी पहले से बनी Class को उसकी Class Library द्वारा Access करके अपने Program में Use कर सकते हैं। यदि हम चाहें तो बिना import Keyword का Use किए हुए भी किसी Predefined Package के किसी Class को Directly Access कर सकते हैं। इसके लिए हमें Class को Access करते ही उसका Object Create करना जरूरी होता है। यानी

```
java.awt.Button OKBtn;
```

इस तरह से हम हमारे Program में बिना import Statement का प्रयोग किए हुए भी Predefined Button Class को Use कर सकते हैं, लेकिन ये Class केवल OKBtn Object के लिए ही उपयोगी होगी। ये Button Create करके हमारे Program से फिर से Invisible हो जाएगी। इसलिए import Statement का प्रयोग करके इसे अपने Program में Use करना ठीक रहता है, अन्यथा हमें हर Class का पूरा Path ध्यान रखना जरूरी हो जाता है।

## Building – The Applet Code

Applets को कभी भी Console Mode में Run नहीं किया जा सकता है। ये हमें किसी ना किसी Browser Window में या फिर एक Special Program जिसे **Applet Viewer** कहते हैं, में ही Run होते हैं। चूंकि सभी Java Applets **Graphical** होते हैं, इसलिए इन्हें GUI Mode में ही Run किया जा सकता है। जावा में Graphical User Interface बनाने के लिए AWT (Abstract Windowing Toolkit) नाम की एक Class Library या Package है, जिसमें Applet पर Graphics Display करने से सम्बंधित विभिन्न प्रकार की Classes को परिभाषित किया गया है।

चूंकि हर Applet एक Window में Run होता है, इसलिए हर Applet के लिए हमें Window Support को प्राप्त करना जरूरी होता है और Window Creation व Management से सम्बंधित जावा की जितनी भी Classes हैं, वे सभी Classes AWT नाम के Package में उपलब्ध हैं, इसलिए हमें इस Package को अपने हर Applet Program में Include करना जरूरी होता है।

चूंकि किसी Applet को हमेंशा GUI Form में ही Display किया जा सकता है, इसलिए किसी Applet को GUI Form में Display करने के लिए जिन Functionality की जरूरत होती है, उन्हें **applet** नाम के एक Package में Predefine कर दिया गया है, जिन्हें हम बिना Extra Coding किए हुए ज्यों का त्यों Reuse कर सकते हैं।

अतः यदि हम Applet Development कर रहे हैं, तो हमें इस Package को भी अपने Source Program में Import करना जरूरी होता है ताकि हम Applet से सम्बंधित सामान्य Functionality को प्राप्त कर सकें। इस प्रकार हम समझ सकते हैं कि एक Applet Develop करने के लिए हमें **applet** Package और **awt** Package को अपने Program में Include करना जरूरी होता है और इन्हें अपनी Source File में Include करने के लिए हमें निम्न Statement लिखने होते हैं।

---

```
import java.applet.*;           // Statement 01
import java.awt.*;             // Statement 02
```

---

## Applet Package – The Applet Class

**applet** Package Java API का सबसे छोटा Package है और **Applet Class** इस Package की इकलौती Class है। **applet** Package में **Applet Class** के अलावा दूसरी कोई Class ही नहीं है। जब भी हम किसी ऐसे Web Page को Open करते हैं, जिसमें Applet Embed होता है, तो वह Applet Automatically Browser में Load हो जाता है।

चूंकि Applet हमेंशा किसी ना किसी HTML File या Web Page में Run होते हैं, जो कि किसी ना किसी Browser में Load होते हैं, इसलिए Applet के इस Environment को “**Context of the Applet**” कहा जाता है।

Applets व Applications दोनों को ही Compile करने के लिए हमें **javac** Compiler का प्रयोग करना पड़ता है, जबकि एक Console Mode Application को Execute करने के लिए हमें **java** Interpreter का प्रयोग करना पड़ता है और एक GUI Applet को Execute करने के लिए हमें **Appletviewer** का प्रयोग करना पड़ता है। किसी Applet को हम ऐसे Browser द्वारा भी देख सकते हैं, जो Java Enabled हो। **Netscape Navigator** एक **Java Enabled Browser** है, जिसमें Applets को बिना Appletviewer का प्रयोग किए देखा जा सकता है।

## **OOPS and OOP – The Definition**

हम Computer पर जो भी Program बनाते हैं वो Program किसी ना किसी Real Life Problem को Solve करने के लिए बनाते हैं और हर Real World Problem में Real World Objects होते हैं।

इस स्थिति में OOPS एक ऐसा Concept है जिसके आधार पर हम विभिन्न प्रकार की Real Life से Related Problems को Solve करने के लिए हमारी समस्या से सम्बंधित विभिन्न Real World के Physical Objects को Computer में Logically Represent कर सकते हैं और उस Real Life से सम्बंधित समस्या को Computer में Logically भी उसी तरह से Solve कर सकते हैं, जिस तरह से उस समस्या को Real World में Physically Solve करते हैं।

किसी Problem को Solve करने का वह System या तरीका जिसके आधार पर किसी Real World Problem को Solve करने के लिए उस Real World Problem से सम्बंधित Physical Objects को Computer में Logically Represent करके Problem को Computer द्वारा Solve किया जा सके, **Object Oriented Programming System (OOPS)** कहलाता है और जिस Computer Programming Language में OOPS के Concepts को ठीक तरह से Implement किया जा सकता है, उस Programming Language को **Object Oriented Programming Language (OOP)** कहा जाता है।

जावा एक ऐसी ही Object Oriented Programming Language है, जिसमें किसी Real World समस्या से सम्बंधित विभिन्न Real World Physical Objects को Computer में Logically Represent करके समस्या को उसी प्रकार से Logically Solve किया जा सकता है, जिस तरह से उस समस्या को Real World में Physically Solve किया जाता है।

## **Problem – The Definition**

दुनियां के हर काम को सम्पन्न करने के लिए हमें एक विशेष क्रम का पालन करना पड़ता है और जब हम किसी काम को ठीक तरीके से कर लेते हैं, तो परिणाम स्वरूप हमें कुछ ना कुछ उचित Output प्राप्त होता है। ठीक इसी तरीके से Computer से भी किसी काम को करवाने के लिए हमें उससे एक विशेष क्रम का पालन करवाना पड़ता है और जब Computer हमारे बताए गए किसी क्रम के अनुसार काम कर लेता है, तो हमें कुछ ना कुछ उचित परिणाम भी प्राप्त होता है। जिस किसी भी काम को करने के लिए हमें किसी विशेष क्रम का पालन करना पड़ता है, उस काम को हम एक **Problem** कह सकते हैं।

इस स्थिति में हर काम एक प्रकार की समस्या होती है क्योंकि दुनियां के किसी भी छोटे से छोटे काम को सम्पन्न करने के लिए भी हमें किसी ना किसी क्रम का पालन तो करना ही पड़ता है। एक उदाहरण से समझें तो यदि हमें पानी पीना हो तो भी हमें एक निश्चित क्रम का पालन करना पड़ता है। पानी पीने के लिए भी हमें:

- 1 सबसे पहले गिलास लेना पड़ेगा। [ **Variable Declaration** ]
- 2 गिलास में पानी लेना पड़ेगा। [ **Input** ]
- 3 फिर उस पानी को मुंह तक लाना पड़ेगा [ **Process** ]
- 4 पानी मुंह में जाएगा और तृप्ति का अहसास होगा। [ **Output** ]

यानी पानी पीने के लिए भी हमें एक निश्चित क्रम का पालन करना पड़ रहा है, इसलिए पानी पीने को भी हम एक प्रकार की समस्या के रूप में देख सकते हैं।

## **Data – Value OR a Set of Values**

मूल रूप से देखा जाए तो Computer के लिए किसी भी प्रकार का मान या मानों का समूह Data कहलाता है। Computer में हम किसी भी मान को मुख्यतः तीन प्रकार से Specify कर सकते हैं।

### **Integer**

जब हमें Computer में किसी ऐसे मान को Store करना होता है, जिसमें केवल पूर्ण संख्याएं ही होती हैं, तब उस Value को Integer प्रकार की Value कहा जाता है।

### **Float**

जब हमें Computer में किसी ऐसे मान को Store करना होता है जिसमें दसमलव वाली संख्या होती है, तब उस Value को Float प्रकार की Value कहा जाता है।

### **Character**

जब हमें Computer में केवल Characters Store करने होते हैं, तब इस प्रकार के मान को Character प्रकार का मान कहते हैं।

## **Object – The Definition**

दुनियां की हर उस चीज को **Object** कह सकते हैं, जिसे Physically देखा जा सके या Logically किसी दिखाई देने वाली चीज के Reference में महसूस किया जा सके। जैसाकि हमने कहा कि **Object** को देखा जा सकता है, इसका मतलब ये हुआ कि हर वह चीज जिसका कोई रंग, रूप व आकार हो वह **Object** है। जैसे कि **Keyboard, Employee, Client** आदि। इसी प्रकार से हर वह चीज जिसे किसी दिखाई देने वाली चीज के Reference में महसूस किया जा सके, **Object** है। जैसे **Bank Account**.

हर **Object** की कुछ विशेषताएं होती हैं, जिसकी वजह से कोई एक **Object** किसी दूसरे **Object** से अलग पहचाना जाता है। किसी भी **Object** की इन विशेषताओं को हम उस **Object** के **Attributes** या उस **Object** की **Properties** कहते हैं। किसी **Object** के **Attribute** के मान में परिवर्तन करने पर वह **Object** बदल जाता है।

उदाहरण के लिए किसी **Account Object** की एक **Property**, उसका **Account Number** हो सकता है। अब यदि किसी **Account** के **Account Number Property** का मान **Change** किया जाए, तो वह **Account** किसी दूसरे **Account Object** को Refer करने लगता है। यानी **Object** के **Attribute** के मान में यदि परिवर्तन कर दिया जाए तो **Object** बदल जाता है।

जिस तरह से हर **Object** Uniquely Identify होने के लिए अपने कुछ **Unique Attributes** का प्रयोग करता है, उसी तरह से हर **Object** किसी ना किसी तरीके से कोई ना कोई काम करता है। **Object** के इस काम करने की प्रवृत्ति को **Object** का **Behavior** कहते हैं।

## **Objects – Based on Problem**

वास्तविक जीवन (Real World) में हम किसी भी समस्या को जिस प्रकार से देखते हैं, उसी प्रकार से हम उस समस्या को Computer में भी Represent कर सकते हैं। समस्या को Computer में Logically तभी Represent किया जा सकता है, जब समस्या को ठीक प्रकार से समझा जाए और ये पता किया जाए कि समस्या के मुख्य Objects कौन-कौन से हैं।

उदाहरण के लिए मान लीजिए कि एक Company में उसके सभी Employees के Bio – Data को Manual Register से Upgrade करके Computerized करना है। अब हमें सबसे पहले इस Real World समस्या से सम्बंधित सबसे महत्वपूर्ण Object को Identify करना है।

चूंकि, विभिन्न Employees के Bio – Data को Computer पर Store करना है, इसलिए इस समस्या से सम्बंधित जो सबसे महत्वपूर्ण Object है, वह Employee ही है। Employee एक Physical Object है और इसे जब Computer में Represent किया जाएगा तो, Computer में वह Employee Logical Object कहलाएगा।

## **Abstraction – The Problem Simplifying Process**

किसी भी Real Life Problem को जब हमें Computer पर Logically Represent करना होता है, तो सबसे पहले हमें ये तय करना होता है कि समस्या से सम्बंधित वे जरूरी चीजें कौन-कौन सी हैं, जो समस्या के परिणाम को प्रभावित करती हैं। समस्या के समाधान को प्रभावित करने वाली जरूरी बातों को समस्या के समाधान को प्रभावित ना करने वाली बिना जरूरी बातों से अलग करने की प्रक्रिया को **Abstraction** कहते हैं। OOPS के इस Concept को हम पिछले उदाहरण द्वारा ही समझने की कोशिश करते हैं।

मानलो कि किसी Company के विभिन्न Employees के Bio – Data को Computer पर Store करना है। चूंकि इस काम को सम्पन्न करने के लिए एक निश्चित क्रम का पालन करना पड़ता है, इसलिए Computer के लिए ये काम एक प्रकार की समस्या है, जिसे Solve करना है।

अब चूंकि हमारी समस्या का मुख्य Object Employee है, इसलिए उसके कई Attributes हो सकते हैं जो एक Employee को दूसरे Employee से Uniquely Identify करने में मदद करते हैं। Employee की विभिन्न Properties में से कुछ निम्नानुसार हो सकते हैं:

- Employee's First Name
- Employee's Last Name
- Address
- City
- District
- State
- Date Of Birth
- Qualification
- Extra Ability
- Degrees
- Designation
- Date Of Company Joining
- Hobbies
- Contact Number

- **No. of His Brothers and Sisters**
- **His Father's Date of Birth**
- His Father's Name
- **His Friends Contact Number**

हम देख सकते हैं कि इसी तरह से एक Employee की इससे भी ज्यादा Properties हो सकती हैं, जिनके आधार पर उसे Uniquely किसी दूसरे Employee से अलग Identify किया जा सके। लेकिन इस List में हम देख सकते हैं, कि Employee के कुछ Attributes ऐसे हैं, जिनका Employee के Bio-Data यानी मुख्य समस्या के परिणाम से कोई सम्बंध नहीं है।

जैसे कि Employee के कितने भाई-बहन हैं, इस बात का Employee के Bio-Data से कोई सम्बंध नहीं है। इसी तरह से Employee के पिता के Date of Birth का Employee के Bio-Data से कोई सम्बंध नहीं है और Employee के दोस्त के Contact Number का कोई सम्बंध Employee के Bio-Data से नहीं है।

यानी ये तीन Attributes ऐसे Attributes हैं, जिनका Employee के Bio-Data से कोई सम्बंध नहीं है। इसलिए इन बिना जरूरी Data को बाकी के जरूरी Data से अलग कर देने पर हमारे सामने निम्नानुसार Attributes आते हैं:

- Employee's First Name
- Employee's Last Name
- Address
- City
- District
- State
- Date Of Birth
- Qualification
- Extra Ability
- Degrees
- Designation
- Date Of Company Joining
- Hobbies
- Contact Number
- His Father's Name

इस प्रकार से हमने समस्या (Bio-Data of Employee) के परिणाम को प्रभावित करने वाले जरूरी Attributes को समस्या से असम्बंधित बिना जरूरी Attributes से अलग किया। इस प्रक्रिया को Abstraction कहते हैं और हमें समस्या के परिणाम को प्रभावित करने वाले जो जरूरी Attributes प्राप्त हुए हैं, इन Attributes को **Abstract Attributes** कहते हैं।

## **Abstract Data Type - Logical Representation of a Real World Object**

हम देख सकते हैं कि Bio – Data Problem से सम्बंधित Attributes को प्राप्त करने के लिए हमने सबसे पहले किसी Employee के सभी Attributes पर विचार किया और फिर जरूरी Attributes को बिना जरूरी Attributes से अलग करके जरूरी Attributes को प्राप्त कर लिया।

अब इसी समस्या के Employee Object को यदि Computer में Logically Represent करना हो, तो हमें इन Abstract Attributes के आधार पर एक **Abstract Data Type** बनाना होता है। इस Abstract Data Type को Object Oriented Programming Languages में **Class** भी कहते हैं।

**Class** एक नए प्रकार का **User Defined Data Type** होता है, जो समस्या से सम्बंधित किसी Real World Object को Computer में Logically Represent करने के लिए बनाया जाता है। Class एक Specification होता है, जो किसी समस्या से सम्बंधित किसी Real World Object के विभिन्न Abstract Attributes के मानों की Computer Memory में Representation को Specify करता है।

## Attributes – The Data Members of The Class

किसी समस्या के परिणाम से सम्बंधित किसी Object के विभिन्न Attributes को जब Computer में Represent करना होता है, तब Computer में उस Object के Attributes में Store होने वाले Data के आधार पर हम ये तय करते हैं कि Object का कौनसा Attribute किस प्रकार के मान (**Integer, Character** या **Float**) द्वारा Represent हो सकता है। जो Attribute जिस प्रकार के मान द्वारा Represent हो सकता है, उस Attribute को उसी प्रकार के Data Type के साथ Declare कर दिया जाता है।

Data Type के साथ Object के Attributes को Declare करने पर Object के विभिन्न Abstract Attributes Class के **Data Members** कहलाते हैं।

इसी बात को यदि हम दूसरे शब्दों में कहें तो Abstraction से प्राप्त **Abstract Attributes** किस प्रकार के Data Store करेंगे, इस तथ्य पर निर्भर करते हुए या इस बात को ध्यान में रखते हुए, इन विभिन्न Abstract Attributes के Basic Data Type को तय किया जाता है।

जब इन Abstract Attributes को उनके Data Type के साथ Class में Specify किया जाता है, तब इन Abstract Attributes के Declaration को Class के **Data Members** कहते हैं।

## Behaviors – The Methods of The Class

कोई Object जो काम करता है, उन कामों को उस Object का Behavior कहते हैं। लेकिन जब हम किसी Object को Computer में Represent करना चाहते हैं, तब हम ये नहीं देखते हैं कि Object वास्तव में क्या-क्या कर सकता है, बल्कि हम ये देखते हैं, कि Object के वे कौन से काम हैं, जिनके द्वारा एक Object अपने किसी उस Attribute को Change करता है, जिन्हें Abstraction की प्रक्रिया द्वारा Identify किया गया है।

किसी समस्या से सम्बंधित Abstract Attributes को उस Object के जो Behaviors प्रभावित करते हैं, Object के वे Behaviors ही उस समस्या से सम्बंधित Behaviors हैं। जावा में Object की Class द्वारा इन Behaviors को Represent करने के लिए हमें Methods का प्रयोग करना होता है। Methods जावा के वे Code Segments होते हैं, जो किसी Object के किसी ना किसी Abstract Attribute के मान को किसी ना किसी प्रकार से Change करते हैं। यानी वे Operations जिन्हें सम्पन्न करने पर Object के किसी ना किसी Abstract Attribute के मान में परिवर्तन हो, **Methods** कहलाते हैं।

## **Problem Design (OOPS) v/s Problem Implementation (OOPL)**

OOPS व OOPL दोनों के आधार पर हमेशा किसी भी OOPS Concept की दो परिभाषाएं बनती हैं। एक परिभाषा केवल OOPS के Concept को Represent करने का काम करती है जबकि दूसरी परिभाषा उस पहली परिभाषा के आधार पर किसी Programming Language में OOPS के उस Concept को Implement करने से सम्बंधित होती है। जब हम OOPS के सम्बंध में कोई परिभाषा देते हैं, तो वह परिभाषा किसी Real World समस्या व उससे सम्बंधित किसी Real World **Physical Object** के सम्बंध में और किसी Software को Design करने के सम्बंध में होती है।

लेकिन जब हम OOPS की विभिन्न परिभाषाओं को किसी Programming Language में Implement करते हैं, तब OOPS का मुख्य Purpose ये होता है कि किसी Real World Object को **Best** तरीके से किसी Programming Language में **Logically** Represent किया जाए, ताकि समस्या से सम्बंधित किसी Real World Object को Computer में एक नए Data Type जिसे **Abstract Data Type** कहते हैं, के रूप में Represent किया जा सके।

किसी समस्या को Solve करने के लिए हम दो Approach Use कर सकते हैं। पहला Approach पूरी तरह से OOPS के Concepts पर आधारित होता है और Problem के Solution को Design करने से सम्बंधित होता है। यानी हम एक भी Line की Coding लिखे बिना भी किसी Software को OOPS के Concept के आधार पर Design कर सकते हैं।

अन्तर केवल इतना होगा कि इस Design का कोई प्रत्यक्ष परिणाम हमें प्राप्त नहीं होगा। जबकि OOPS के आधार पर हम जिस Modal को बनाएंगे उस Modal को यदि किसी OOPL Language में Implement कर दिया जाए तो हमें समस्या का प्रत्यक्ष परिणाम प्राप्त हो जाएगा।

इस बात को सरल तरीके से कहें तो दो संख्याओं को जोड़ने का Algorithm लिख देने से हमें दो संख्याओं का योग प्राप्त नहीं होगा बल्कि दो संख्याओं का योग करने के लिए किस प्रकार से काम करना होगा, उस काम करने की Process का पता चलेगा।

जबकि इस Algorithm के आधार पर किसी Programming Language में Computer Codes लिख दिए जाएं तो हमें दो संख्याओं की Actual जोड़ प्राप्त हो जाएगी। यानी **Algorithm** बनाना **Problem Design** करने से सम्बंधित काम है जबकि Program बनाना उस Algorithm को Implement करने से सम्बंधित काम है।

इसी तरह से **OOPS** के Concept को Apply करना **Problem Design** करने से सम्बंधित है जबकि OOPS के Concept के आधार पर बनने वाले Design को Computer Language में Implement करना Problem Solve करने से सम्बंधित काम है।

हम ऐसा भी कह सकते हैं कि जब किसी Problem को Real World में Define करना होता है, तब Problem के विभिन्न Objects को Physical रूप में Identify कर सकते हैं, लेकिन जब उसी Problem को Computer में Logical रूप में Define करना होता है, तब Problem को एक नए Data Type के रूप में Represent करना होता है, जिसे हम किसी Computer Programming Language में **Abstract Data Type** या Object की **Class** के रूप में Identify करते हैं।



## **Encapsulation – The Unitizing Process of Attributes and Behaviors**

जैसाकि हमने पहले कहा कि OOPS व OOPL के आधार पर हर OOPS Concept की दो परिभाषाएं होती हैं। यही नियम OOPS के Encapsulation Concept के सम्बंध में भी लागू होती है। जब हम OOPS के सम्बंध में देखते हैं, तो किसी Object की Internal Workings को या उसके काम करने के जटिल तरीकों को समझे बिना उसे उपयोग में लेने के लिए कुछ व्यवस्थाएं कर देना, इस प्रक्रिया को **Encapsulation** कहते हैं। इस Concept को समझने के लिए एक उदाहरण देखते हैं।

मान लो कि हमारे पास एक Computer है। अब जब हम Computer को Switch ON करते हैं, तो ये जाने बिना कि Computer किस प्रकार से ON होता है, हम Computer को उपयोग में लेते रहते हैं। इसी तरह से किसी Car के Accelerator को Press करने पर Car की Speed किस प्रकार से बढ़ती है, ये जाने बिना एक Driver Car को चलाता रहता है। इन दोनों उदाहरणों में Computer व Car दो Encapsulated Objects हैं जिनके काम करने के तरीके यानी Internal Working को जाने बिना उन्हें आसानी से उपयोग में लिया जाता है।

अब यदि हम इसी Concept को Java Programming Language के सन्दर्भ में समझें, तो परिभाषा बदल जाती है। Programming Language जावा के सम्बंध में जब हम Encapsulation को परिभाषित करना होता है, तो हम इस Concept को अग्रानुसार परिभाषित करते हैं।

समस्या से सम्बंधित Abstracted Attributes को उनके Data Type के साथ Specify करने की Process से उस Object के Data Members प्राप्त होते हैं और समस्या से सम्बंधित जिन Operations को Perform करके कोई Object अपने किसी Abstract Attribute की स्थिति या मान में परिवर्तन करता है, उस Operation को Object का Behavior कहते हैं, जिसे जावा में Implement करने के लिए जिन Codes को लिखते हैं, उन Code Block को Methods कहते हैं।

यानी किसी समस्या से सम्बंधित Object के Physical Attributes व Behaviors को Java में Implement करने पर Object के समस्या से सम्बंधित विभिन्न Abstracted Attributes को Data Members व विभिन्न Attributes को Change करने वाले Operations को Methods के रूप में Specify करते हैं।

किसी भी Object के गुण यानी उसके Attributes व उसके Behaviors दोनों किसी एक ही Physical Object के हिस्से होते हैं। इसलिए Object के **Attributes** को Represent करने वाले **Data Members** व **Behaviors** को Represent करने वाले **Methods** दोनों को एक इकाई के रूप में Specify करने की प्रक्रिया को OOPL में **Encapsulation** कहते हैं।

Encapsulation के बाद यानी Computer में किसी समस्या से सम्बंधित Object के विभिन्न Data Members (Attributes) व Methods (Behaviors) को Combine करके एक Unit के रूप में Encapsulate करने के बाद जो Specification हमें प्राप्त होता है, उस Specification को यदि एक नाम दे दिया जाए, तो वह Specification समस्या से सम्बंधित किसी Object की **Class** का Representation होता है।

जावा के लिए ये एक नए प्रकार का Data Type होता है, जो किसी Real World समस्या से सम्बंधित किसी Real World Object को Computer में Logically Represent करने में सक्षम होता है। इसे **Abstract Data Type** भी कह सकते हैं, क्योंकि ये किसी समस्या के Abstraction से प्राप्त Abstracted Attributes के आधार पर बनता है।

## **Class – A Logical Specification of Problem Related Object**

अभी तक हमने OOPS के जिन Concepts व Elements को परिभाषित किया है, उनमें से ज्यादातर परिभाषाएं भ्रमित करने वाली व समझ में ना आने वाली हैं। चलिए, हम इन परिभाषाओं को केवल जावा के सम्बंध में ही Apply करने की कोशिश करते हैं।

किसी भी Real World Problem को जब हमें Computer पर Solve करना होता है, तो सबसे पहले हमें ये जानना होता है, कि Problem से सम्बंधित वह सबसे महत्वपूर्ण Object कौनसा है, जिस पर समस्या आधारित है। Object को पहचानने का कोई निश्चित तरीका या Trick नहीं है। हर समस्या में अलग प्रकार का Object प्रभावित होता है और हर समस्या में उस अलग प्रकार के Object के अलग प्रकार के Attributes को Manage करना होता है। इसलिए किसी समस्या के सबसे महत्वपूर्ण Object का पता लगाने के लिए हमें बार-बार विभिन्न प्रकार की समस्याओं को हल करने की कोशिश करनी होती है।

कई बार किसी समस्या को Solve करते समय हम गलत Object ले लेते हैं। ऐसे में जब हम आगे बढ़ते हैं, तब समस्या Computer में पूरी तरह से Represent नहीं हो पाती है और जब समस्या पूरी तरह से Represent नहीं हो पाती है, तब हमें स्वयं ही पता चल जाता है कि समस्या से सम्बंधित जो सबसे महत्वपूर्ण Object हमने माना था, वह Object सबसे ज्यादा महत्वपूर्ण क्यों नहीं हैं। और जब हमें ये पता चल जाता है कि हमने किसी गलत Object को महत्वपूर्ण मान लिया है, तो उसी समय हमें ये भी पता चल जाता है कि समस्या से सम्बंधित सबसे ज्यादा महत्वपूर्ण Object कौनसा है।

जब बार-बार इस तरह से Practice की जाती है, तब अपने आप ही ये पता चलने लगता है कि समस्या से सम्बंधित सबसे उचित व महत्वपूर्ण Object कौनसा है। फिर भी कुछ Common नियम हैं जिनके आधार पर कुछ समस्या से सम्बंधित Objects को Directly Identify किया जा सकता है।

उदाहरण के लिए यदि हम किसी Bank Account को Manage करने के लिए Program Create कर रहे हैं, तो Bank Account से सम्बंधित कई Objects होते हैं, जो सबसे ज्यादा महत्वपूर्ण होते हैं। इसे हम निम्नानुसार ज्ञात कर सकते हैं:

### **Different Kinds of Bank Account**

{Saving A/C, Current A/C, FD A/C, RD A/C, Over Draft A/C}

### **Bank's Customers**

समस्या से सम्बंधित महत्वपूर्ण Objects पहचानने के बाद हमें उन Objects के उन Attributes को प्राप्त करना होता है, जो समस्या से सम्बंधित हों। चूंकि एक Bank Account व एक Customer दोनों के ही विभिन्न प्रकार के Attributes हो सकते हैं और ये जरूरी नहीं है कि सभी Attributes समस्या को Represent करने के लिए उपयोगी हों। इसलिए हमें किसी Object के समस्या से सम्बंधित जरूरी Attributes प्राप्त करने के लिए समस्या का Abstraction करना होगा।

किसी समस्या से सम्बंधित सबसे महत्वपूर्ण Object के समस्या से सम्बंधित सबसे महत्वपूर्ण Attributes प्राप्त करने का यानी Abstraction करने का सबसे अच्छा तरीका ये है कि हम ये पता करें कि हमारी समस्या में Object क्या Operation Perform करेगा। यानी Object काम क्या करेगा?

हम जानते हैं कि कोई Object जब भी कोई काम करता है या Operation Perform करता है, तो किसी ना किसी तरीके से वह अपने ही किसी Attribute के मान में परिवर्तन करता है। इसलिए यदि हमें ये पता चल जाए कि हमारी समस्या में हमारा Object अपने किन Attributes के मानों में परिवर्तन करेगा, तो हम उन परिवर्तित होने वाले मानों को Store करने वाले Attributes को Object

के जरूरी **Abstract Attributes** के रूप में परिभाषित कर सकते हैं, यानी **Object** के जरूरी **Data Members** का पता लगा सकते हैं।

यदि कोई **Object** कोई ऐसा **Operation Perform** करता है, जिससे **Object** के किसी भी **Attribute** के मान में कोई परिवर्तन नहीं होता है, तो समझ लेना चाहिए कि **Object** का वह **Operation** समस्या से सम्बंधित जरूरी **Operation** नहीं है और **Object** के उस **Operation** को **Neglect** कर देना चाहिए।

यदि हम सरल शब्दों में कहें तो कह सकते हैं कि किसी समस्या से सम्बंधित जरूरी **Data** का पता लगाने के लिए हमें उस समस्या से सम्बंधित सबसे महत्वपूर्ण **Object** के सम्बंध में ये पूछना चाहिए कि **Object** के क्या करने पर क्या होगा। किसी **Object** के कुछ करने पर **Object** की जिन विशेषताओं के मान बदल सकते हैं, वे विशेषताएं ही हमारी समस्या से सम्बंधित सबसे महत्वपूर्ण **Data** हैं। चलिए, इसी नियम को हम **Bank Account Object** पर **Apply** करते हैं।

**Bank Account Object** का सबसे महत्वपूर्ण काम ये है कि हम किसी **Bank Account** में अपना धन **Deposit** करवाते हैं। यानी **Bank Account Object** धन को **Deposit** करने का काम करता है। इसी तरह से हम किसी **Bank Account** में अपने **Deposited** धन को **Withdraw** करते हैं। यानी **Bank Account Object** **Bank** में जमा धन को निकालने का काम भी करता है।

इस तरह से एक **Bank Account Object** समस्या से सम्बंधित दो ही सबसे महत्वपूर्ण काम कर रहा है। हालांकि कुछ अन्य काम भी होते हैं, जिन्हें **Bank Account Object** **Perform** करता है, लेकिन हम यहां पर यही मान रहे हैं कि **Bank Object** यही दो काम करता है।

जब **Bank Account** में धन **Deposit** करते हैं, तब **Deposit Form Fill** करके रूपयों के साथ उस **Form** को भी **Bank** में जमा करवाना पड़ता है। **Deposit Form** में हमें निम्न सुचनाएं देनी होती हैं:

- Branch Name
- Branch Number
- Account Number
- Account Head Name
- Money
- Date

इसी तरह से यदि **Bank** से रूपए **Withdraw** करने हों, तो एक **Withdrawal Form Fill** करके जमा करवाना पड़ता है। उस **Withdrawal Form** में निम्न सुचनाएं देनी होती हैं:

- Branch Name
- Branch Number
- Account Number
- Account Head Name
- Money
- Date

ये सभी किसी **Bank Account** के वे **Attributes** हैं, जिन्हें **Deposit** या **Withdraw** **Operation** द्वारा **Change** किया जाता है। इसलिए ये सभी **Attributes** **Bank Account Object** के **Abstract Attributes** हैं और **Deposit** व **Withdraw** वे **Operations** हैं जो जरूरत के आधार पर इन **Abstract Attributes** को **Change** करने के लिए **Use Perform** या **Execute** होते हैं।

इस तरह से हमें Account Object के लिए दो Operations व 6 Abstract Attributes प्राप्त होते हैं। इन्हीं **Operations** व **Attributes** को हम एक **Abstract Data Type** या एक जावा **Class** के रूप में Specify करें, तो बनने वाला Description समस्या से सम्बंधित Real World Object (**Account**) को Computer में Logically Represent करता है। समस्या से सम्बंधित Account Object के विभिन्न Abstract Attributes में हम निम्न प्रकार के Data Feed कर सकते हैं:

• Branch Name	=	Jaipur
• Branch Number	=	JPR01
• Account Number	=	AC1201235
• Account Head Name	=	Achyut
• Money	=	2000.00
• Date	=	12-July-2007

जिन Attributes के मानों के साथ किसी प्रकार की Calculation हो सकती है, उन Attributes को Integer या Float प्रकार का Declare किया जाना होता है। चूंकि Bank Account का Money Attribute एक ऐसा Attribute जिसके मान के साथ किसी प्रकार की Calculation हो सकती है, साथ ही ये मान दसमलव वाला है इसलिए इस मान को Hold करने के लिए Float प्रकार के Data Type के Variable की जरूरत पड़ेगी।

जिन Attributes में Character प्रकार के मान होते हैं, उन्हें String प्रकार के Variable में Store किया जा सकता है। वे Attributes जो किसी प्रकार की Calculation में भाग नहीं लेते हैं, उन्हें भी हमें String या Character प्रकार के Variable में Store करना चाहिए।

इस नियम के आधार पर देखें तो **String** को Store करने वाले Attribute को String Keyword के साथ, Character को Store करने के लिए **char** Keyword के साथ, Integer प्रकार के मान को Hold करने वाले Attribute के साथ **int** Keyword व Float प्रकार के मान को Store करने के लिए **float** Keyword के साथ Attributes को लिखने पर हमें निम्नानुसार Format प्राप्त होता है:

- String Branch Name
- String Branch Number
- String Account Number
- String Account Head Name
- float Money
- String Date

जब हमे किसी समस्या के मुख्य Object के Abstraction से Abstract Attributes प्राप्त होते हैं, और उनमें Store किए जाने वाले मानों के आधार पर हम ये तय कर लेते हैं कि किस Attribute में किस प्रकार का मान Store होगा, और जावा के आधार पर उन Data Types को Represent करने वाले Keywords को Abstract Attributes के साथ Specify कर देते हैं, तो हमें Abstract Attributes की अब जो Description प्राप्त होती है, उसे जावा Class के **Data Members** कहते हैं।

जावा की किसी भी Class को हमेंशा किसी ना किसी Real World Object को एक नए User Defined Data Type की तरह Represent करने के लिए Develop किया जाता है और किसी भी Real World Object के कुछ Attributes के साथ उसके कुछ Behaviors भी होते हैं, इसलिए जावा की उस Account Class में विभिन्न Abstract Attributes को Represent करने वाले Data Members के साथ उन Behaviors को भी Specify करना होता है, जो समस्या से सम्बंधित Operations Perform करके इन Data Members के मानों में परिवर्तन करते हैं।

यानी Account Object को जावा में तभी पूरी तरह से Represent किया जा सकता है, जब इन Data Account के Abstract Attributes को Represent करने वाले Data Members के साथ Account Object के Behaviors यानी Deposit व Withdraw Operations को भी Specify किया जाए। सामान्यतया Abstract Attributes में परिवर्तन करने वाले Operations को जावा में **Methods** कहते हैं।

अतः जब तक हम समस्या से सम्बंधित Data Members व Methods को एक साथ एक Group के रूप में Specify नहीं करेंगे, तब तक जावा किसी Real World Object को पूरी तरह से Logically Represent नहीं कर सकेगा। इसलिए हमें Account Object के Abstract **Data Members (Attributes)** व **Methods (Behaviors)** दोनों को निम्नानुसार एक Unit के रूप में Specify करना होगा:

---

Abstract Data Type or Class

//Data Members or Abstract Attributes

String      **Branch** Name  
String      Branch Number  
String      **Account** Number  
String      Account Head Name  
**float**      Money  
String      Date

// Methods or Behaviors

Deposit  
Withdraw

---

जब हमें जावा में किसी Object के Behavior को Represent करना होता है, तब हमें Behavior यानी Method के नाम के साथ **Parenthesis** का प्रयोग करना होता है। साथ ही ये भी बताना होता है कि Method किसी प्रकार का कोई मान Return करेगा या नहीं।

चूंकि हम यहां पर Method को बिल्कुल Simple रखना चाहते हैं, इसलिए हम ये मान रहे हैं कि Method किसी भी तरह का कोई मान Return नहीं करेगा। जब हम Method से किसी तरह का कोई मान Return करवाना नहीं चाहते हैं, तब जावा Compiler को इस बात की Information देने के लिए Method के नाम के आगे **void** लिखते हैं।

जब हमें जावा की Coding के रूप में किसी Object से सम्बंधित Abstracted Attributes यानी Data Members व Behaviors यानी Methods को एक Group या एक **Single Unit** के रूप में Specify करना होता है, तब Object के विभिन्न Data Members व Methods को **Opening** व **Closing Curly Braces { }** के बीच में Specify करना होता है।

उपरोक्त दोनों नियमों को Account Class के Code Segment पर Apply करने पर हमें Account Object की Class का निम्नानुसार Specification प्राप्त होता है:

---

// Account Abstract Data Type or Class

```
class    Account  
{  
    // Data Members or Abstract Attributes  
    String    Branch Name ;
```

```
String Branch Number ;
String Account Number ;
String Account Head Name ;
float Money ;
String Date ;

// Methods or Behaviors
void Deposit()
void Withdraw()
}
```

किसी Object को Represent करने वाले Data Members व Methods को एक Unit के रूप में Specify करने की प्रक्रिया को OOPS Concept के अन्तर्गत **Encapsulation** नाम दिया गया है। यानी जब हम Account Object के Data Members व Methods को Opening व Closing Curly Braces के बीच में Specify करते हैं, तो वास्तव में हम Account Object को Encapsulate कर रहे होते हैं।

यानी हमने Account Object को उपरोक्त Code Statement द्वारा Encapsulated कर दिया है। अब इस Code Segment की शुरुआत में **class** Keyword के साथ में हम **Account** शब्द Specify कर दें, तो इसका मतलब ये है कि हमने Account Class बना दी है, जिसके Objects को जावा का Compiler पहचानता है।

जावा के सभी Code Statements के अन्त को निर्धारित करने के लिए यानी जावा के Compiler को ये बताने के लिए कि CPU को दिया जाने वाला एक Instruction पूरा हो गया है, हमें हर Statement के बाद में Semicolon का प्रयोग करना होता है।

हालांकि अभी ये Class अधूरी है, क्योंकि हमने जावा Compiler को अभी ये नहीं बताया है कि **Deposit()** व **Withdraw()** Methods Account Class के विभिन्न Data Members को किस प्रकार से Change करेंगे। जब हमें Methods को ये बताना होता है कि वे किसी Class के विभिन्न Data Members के मानों को किस प्रकार से Change करेंगे, तब हमें विभिन्न प्रकार के Java Codes लिखने पड़ते हैं। इन विभिन्न प्रकार के Java Codes को Method के Parenthesis के बाद **Opening** व **Closing Curly Braces { }** के बीच में लिखना होता है।

इन तीनों नियमों को Apply करने पर हमें Account Object की Class का निम्नानुसार Specification प्राप्त होता है, जो एक Real World Account Object को Computer में Logically Represent करता है:

---

```
// Account Abstract Data Type or Class
class Account
{
    //Data Members or Abstract Attributes
    String Branch Name ;
    String Branch Number ;
    String Account Number ;
    String Account Head Name ;
    float Money ;
    String Date ;
}
```

```
//Methods or Behaviors
void Deposit()
{
    // Java Codes for changing the states of
    // Data Members based on requirement.
}

void Withdraw()
{
    // Java Codes for changing the states of
    // Data Members based on requirement.
}
}
```

---

अभी भी इस Class के Objects Create नहीं किए जा सकते हैं। क्योंकि जावा में विभिन्न प्रकार के मानों को Store करने के लिए जिन Variables को Declare करते हैं, उन Variables के नाम देने के कुछ नियम हैं। यदि हम उन में किसी भी नियम को Neglect करते हैं, तो Java Compiler Program में Compile Time Error Generate करता है।

## Identifier Naming System

Computer में विभिन्न प्रकार के मानों को Store करने के लिए Data के अनुसार Memory में कुछ जगह Reserve की जाती है और उस Reserved जगह को पहचानने के लिए Programmer उस जगह को कोई नाम देता है। जैसे यदि Programmer किसी व्यक्ति की Age Computer में Store करना चाहते हैं, तो Age Store करने के लिए उसे Integer प्रकार का एक Identifier Create करना होगा यानी मान को Store करने के लिए Memory में कुछ जगह Reserve करनी होगी और उस Reserved Memory को एक नाम देना होगा। इस नाम से ही उस Memory Location को पहचाना जा सकता है जिस पर Programmer ने Age के मान को Store किया है। जावा में किसी Identifier को नाम देने के कुछ नियम हैं, जो निम्नानुसार हैं:

- Identifier के नाम में A to Z Capital व Small Letters का प्रयोग किसी भी प्रकार से किया जा सकता है।
- Underscore का प्रयोग किया जा सकता है।
- यदि नाम में किसी Digit का प्रयोग करना हो, तो नाम के अन्त में Digit का प्रयोग किया जा सकता है।
- नाम में किसी Operator, Special Symbol या Keyword का प्रयोग नहीं किया जा सकता है।
- नाम में Blank Space का प्रयोग नहीं किया जा सकता है, क्योंकि Blank Space एक तरह का Special Symbol है।

हमने हमारी Specification में Abstract Attributes को Represent करने वाले ज्यादातर Data Members के नाम के बीच में Whitespace का प्रयोग किया है। इसलिए जावा इन Identifiers को Accept नहीं करता है और हमें Errors देता है।

जावा का एक Naming Convention है, जिसमें जब हमें किसी Class का नाम देना होता है और नाम में दो या अधिक शब्द होते हैं तब हमें निम्नानुसार नाम देना चाहिए:

## PopulationOfIndia, AgeOfMoon, BasicSalary

इसी तरह से जब हमें विभिन्न Data Members का नाम देना होता है और नाम में एक से ज्यादा शब्द हों तो उन्हें निम्नानुसार नाम देना चाहिए:

## populationOfIndia, ageOfMoon, basicSalary

अब चूंकि हमने हमारे Description में विभिन्न Data Members के बीच Space का प्रयोग किया है, इसलिए यदि इस Class को ठीक प्रकार से Specify करना है तो हमें इस Class को निम्नानुसार Modify करना पड़ेगा।

---

```
// Account Abstract Data Type or Class
class Account
{
    //Data Members or Abstract Attributes
    String branchName ;
    String branchNumber ;
    String accountNumber ;
    String accountHeadName ;
    float money ;
    String date ;

    //Methods or Behaviors
    void Deposit()
    {
        // Java Codes for changing the states of
        // Data Members based on requirement.
    }

    void Withdraw()
    {
        // Java Codes for changing the states of
        // Data Members based on requirement.
    }
}
```

---

हालांकि हमने Deposit() व Withdraw() Method में किसी तरह के Code नहीं लिखे हैं, इसलिए इस Class के Object किसी भी तरह का कोई भी काम नहीं करेंगे। फिर भी ये Class पूरी तरह से Executable है और हम इस Class के Objects Create कर सकते हैं।

इस तरह से हम किसी समस्या से सम्बंधित किसी जरूरी Object को एक Abstract Data Type (Class) के रूप में Specify करके Object को Computer में Logically Represent कर सकते हैं। एक बार किसी समस्या के आधार पर किसी Object की Class बना देने के बाद हम उस Class के जितने चाहें उतने Objects Create कर सकते हैं।

यानी उपरोक्त Account Class के आधार पर हम जितने चाहें उतने Customers के Account को Computer में उसी तरह से Logically Create व Manage कर सकते हैं, जिस तरह से Real World में लोगों के Account को Physically Create व Manage किया जाता है।



## **Java – Graphical User Interface and Graphics Management**

Computer में GUI व Graphics Management को भी हम एक समस्या के रूप में देख सकते हैं। विभिन्न प्रकार के User Interface Components (Button, Scrollbars, Menu Bars, Status Bars, Dialog Boxes, Radio Buttons, Check Boxes, Text Fields) को Create करने व उन्हें Computer में ठीक तरह से Manage करने के लिए भी विभिन्न प्रकार की Graphics Classes को पहले से ही Develop करके जावा के Compiler के साथ हमें प्रदान किया जाता है।

इन विभिन्न प्रकार की Graphics Classes को जावा में Abstract Windowing Toolkit (**AWT**) नाम के एक Package में Store करके रखा गया है। चूंकि जावा OOPS को पूरी तरह से Support करता है, इसलिए हम इसमें **Inheritance** की सुविधा को पूरी तरह से प्राप्त कर पाते हैं।

**Inheritance** OOPS का एक ऐसा Concept है, जिसके आधार पर हम विभिन्न प्रकार की Classes बना कर उन्हें उनके Group के आधार पर विभिन्न प्रकार के Packages में Store कर सकते हैं और जरूरत होने पर उन Classes के Codes को ज्यों का त्यों प्राप्त भी कर सकते हैं और उनमें Modification करके अपनी जरूरत के अनुसार नई Classes भी बना सकते हैं, जिनमें उनकी पुरानी Classes के सभी Features मौजूद होते हैं।

इसका मतलब ये हुआ कि अपने Program को Graphics Mode में Develop करने के लिए ये जरूरी नहीं है कि हम विभिन्न प्रकार के Graphical Components को फिर से Develop करें और उनके लिए फिर से विभिन्न प्रकार की Classes Create करें, बल्कि हम AWT Package में उपलब्ध विभिन्न प्रकार की Classes को Inherit करके फिर से Reuse कर सकते हैं और अपने Program को Graphical Mode में Develop कर सकते हैं।

## **Web Page – The Part of Website**

जैसाकि हमने पहले भी कहा है कि Applet वे छोटे Programs होते हैं जो किसी Web Page को अधिक Interactive बनाने के लिए Develop किए जाते हैं। Applet को समझने से पहले हम ये समझते हैं कि आखिर Web Pages क्या हैं और इनकी जरूरत क्या है।

Web Pages ऐसे Electronic Pages हैं, जिन पर Stored Information दुनियां में कहीं भी देखी जा सकती हैं। Web Pages सूचनाओं को तेजी से सारी दुनियां में फैलाने का सबसे अच्छा माध्यम है, फिर चाहे ये सूचना किसी University का Declare किया गया Result हो या किसी Company के किसी नए उत्पाद के बारे में दिया गया विज्ञापन ।

सामान्यतया Web Pages का ज्यादातर उपयोग विभिन्न प्रकार की Companies अपने उत्पाद की Advertisement करने के लिए करती हैं, ताकि उनके उत्पाद को सारी दुनियां जान सके और उनका व्यवसाय National से बढ़कर International हो सके, यानी विदेशों में भी बढ़ सके। सामान्यतया लोग उन चीजों को ज्यादा ध्यान से देखते व सुनते हैं या ऐसी चीजों के बारे में ज्यादा गौर करते हैं, जिन्हें Represent करने के लिए विभिन्न प्रकार के Multimedia का प्रयोग किया गया हो।

साथ ही ऐसा भी होता है कि अलग-अलग लोग अलग-अलग प्रकार के परिवेश में रहते हैं और अलग-अलग भाषाओं का प्रयोग करते हैं। ऐसे में यदि किसी Information को Graphically Represent किया जाता है, तो उस Information को तुलना में ज्यादा लोग समझ पाते हैं। इसी

Concept के आधार पर ये माना गया कि यदि Web Pages भी Multimedia से युक्त हों, तो ज्यादा से ज्यादा लोग Web Pages को देखेंगे और किसी Company के उत्पाद का ज्यादा से ज्यादा लोगों में विज्ञापन हो सकेगा।

इसलिए ऐसे Web Pages बनाने की जरूरत महसूस की गई, जिसमें Multimedia यानी Sound, Video, Animation, Graphics आदि का ज्यादा से ज्यादा प्रयोग किया गया हो और किसी Web Page में Multimedia को लाने के लिए जावा Applets सबसे अच्छा साधन हैं।

इसके साथ ही आज जो व्यवसाय किए जाते हैं वे किसी एक शहर या एक देश तक सीमित नहीं होते हैं। विभिन्न प्रकार की Multinational Companies आपस में मिल कर व्यवसाय करती हैं। ऐसे में किसी Multinational Company के विभिन्न भागीदारों को कभी भी अपनी Company की Financial स्थिति जानने की जरूरत पड सकती है।

ऐसे में हर भागीदार अपनी Company की स्थिति जानने के लिए Center Office से जानकारी प्राप्त कर सके, ये व्यवस्था ठीक नहीं कही जा सकती। क्योंकि हो सकता है कि Company का कोई Partner India में रहता हो और Company का Center Office America में हो। ऐसे में India का वह व्यक्ति ये कैसे जान सकेगा कि उसकी Company की स्थिति क्या है, जिसमें उसने अपना धन Invest किया है।

इस प्रकार की स्थिति में Company से सम्बंधित सभी सूचनाओं को America के Center Office में यदि हर रोज Web Pages के रूप में Internet पर डाल दिया जाए, तो India का वह Partner भी Company की स्थिति को हर रोज किसी भी समय जान सकता है।

चूंकि Company की वर्तमान स्थिति की विभिन्न प्रकार की सूचनाओं को Analyze करके विभिन्न प्रकार के Decisions लेने होते हैं, इसलिए विभिन्न प्रकार की Information को विभिन्न प्रकार के Graphs, Charts आदि के रूप में व्यवस्थित किया जाता है, ताकि विभिन्न प्रकार के जरूरी Decisions तुरन्त लिए जा सकें।

इन विभिन्न प्रकार के Graphics व Charts आदि को विभिन्न प्रकार के User द्वारा दिए जाने वाले Data के आधार पर Online Develop हो सकें, इसके लिए ऐसे Interactive Web Pages की जरूरत होती है, जो User के साथ थोडा बहुत Interaction कर सकें और ये सुविधा केवल Applets द्वारा ही प्राप्त की जा सकती है, क्योंकि जावा Applets वे Programs होते हैं, जो Web Pages में Run होते हैं और Web Pages पूरी दुनियां में कहीं से भी Access किए जा सकते हैं।

इन Web Pages पर User द्वारा Parameter के रूप में Specify किए गए Inputs को Applets Accept कर सकते हैं और विभिन्न प्रकार से Modify होकर User की सामान्य जरूरतों को Online यानी Run Time में पूरी कर सकते हैं। सारांश ये हुआ कि Applets का प्रयोग Web Pages में करना होता है, इसलिए Applet Programming करने से पहले हमें Web Page बनाना सीखना होगा।

Web Site एक ऐसी Book की तरह होती है, जिसमें बहुत सारे ऐसे Electronic Pages होते हैं, जिनमें सूचनाओं को Present करने के लिए विभिन्न प्रकार के Multimedia का प्रयोग किया जाता है। Web Pages Develop करने के लिए हमें किसी विशेष Compiler या Software की जरूरत नहीं होती है ना ही Web Pages के Output को देखने के लिए ये जरूरी होता है कि हम Internet से Connected हों।

हम किसी Web Site के विभिन्न Pages को Notepad में Develop कर सकते हैं, और उसे Internet Explorer या ऐसे ही किसी Browser में Embed करके यानी Run करके देख सकते हैं कि ये Web Pages Internet पर Run होने पर किस प्रकार के दिखाई देंगे।

Web Pages Develop करने के लिए कुछ Basic तैयारियां करनी होती हैं। यानी हमें ये तय करना होता है कि हम Web Pages को किन लोगों के लिए Develop कर रहे हैं और उन लोगों को Web Pages द्वारा क्या जानकारी देना चाहते हैं। हम जो जानकारी देना चाहते हैं, उसके Contents को पूरी तरह से Develop करने के बाद हमें उन Contents को Web Pages के रूप में Represent करना होता है।

अपने Contents को Web Pages में Represent करने के लिए हमें HTML (Hyper Text Markup Language) Tags का प्रयोग करना होता है। HTML Tags ऐसे Tags होते हैं, जिन्हें सभी Computers Platforms समान रूप से Support करते हैं, इसलिए हम जिस Web Site को Develop करके Internet के Web Server पर Store कर देते हैं, वह Web Site विभिन्न प्रकार के Hardware व Platform को Use करने वाले सभी लोगों को समान रूप से दिखाई देती है।

HTML Pages बनाने के लिए हमें HTML के विभिन्न Tags को एक Notepad में लिखना होता है और उस Notepad की File को .HTML नाम से Save करना होता है। जब हम किसी Notepad में Create की गई File को .HTML Extension से Save करते हैं, तब वह File Internet Explorer यानी किसी Web Browser में Open होने वाली File बन जाती है। हम Windows Platform पर काम कर रहे हैं, इसलिए हम हमारी सभी Files, Internet Explorer में Open होगी।

## HTML Tags for Web Pages

HTML File बनाने के लिए हमें जो Tags लिखने होते हैं, वे Tags भी कुछ विशेष तरीके से लिखने होते हैं। एक Web Page में Contents व Tags यानी कुल दो चीजें होती हैं। Tags वे Commands होते हैं जिन्हें Web Browser समझता है और उनके आधार पर दिए गए Contents को विभिन्न प्रकार के Web Pages पर Display करता है। विभिन्न प्रकार के HTML Tags को **Opening** व **Closing Bracket** ( < > ) के बीच लिखा जाता है।

साथ ही विभिन्न प्रकार के Contents को < > व </ > के बीच लिखा जाता है। “ / ” का चिन्ह एक Tag के अन्त को दर्शाता है। एक Web Page के विभिन्न Tags व Contents को <HTML> व </HTML> Tags के बीच लिखा जाता है। किसी भी Web Page के मुख्यतः तीन भाग होते हैं:

## Comment Section

ये एक Optional Section है। हम चाहें तो इसे बिना Specify किए हुए भी Web Page Develop कर सकते हैं। Comments किसी Web Page के बारे में General जानकारी देते हैं। Comment देने के लिए <! > के बीच Statement लिखा जाता है। इस तरह के Bracket में जो Text लिखे जाते हैं, वे Browser में दिखाई नहीं देते हैं। ये Comment हमें निम्नानुसार लिखना होता है:

```
<HTML>
<! This is my first web page comment. >
</HTML>
```

हालांकि Comment Text Web Pages में Front में दिखाई नहीं देते हैं, लेकिन फिर भी ये Web Pages के साथ Client के Computer में Download तो होते ही हैं, इसलिए हमें हमारे Web

Page में कम से कम Comment लिखने चाहिए। हालांकि हम Web Page में कहीं भी और कितनी भी बार Comment लिख सकते हैं।

## Head Section

ये भी एक Optional Section है। हम चाहें तो इसे भी बिना Specify किए Web Page Develop कर सकते हैं। Head Section को <HEAD> व </HEAD> Tag द्वारा Define किया जाता है। इस Tag का प्रयोग Web Page का नाम यानी **Title** देने के लिए किया जाता है। इस Section में <TITLE> व </TITLE> के बीच में हम जो Text लिख देते हैं, वह Text Web Page के Title Bar में दिखाई देने लगता है। Web Page में दिखाई देने वाले **Title** को निम्नानुसार Tags द्वारा Define किया जा सकता है:

```
<HTML>
<! This is my first web page comment. >

<HEAD>
  <TITLE>Betalab Computer Center</TITLE>
</HEAD>

<HTML>
```

## Body Section

ये भी एक Optional Section है। Head Section के बाद में Body Section आता है, जहां हम Web Page के Contents लिखते हैं और विभिन्न प्रकार के Multimedia का प्रयोग भी यहीं पर करना होता है। उन्हें विभिन्न तरीकों से Web Page पर दर्शा सकते हैं।

## Adding Applet in HTML File

जिस तरह से हम विभिन्न प्रकार के Tags का प्रयोग करके विभिन्न प्रकार से किसी Web Page को Design करते हैं, उसी तरह से किसी Applet को HTML Page में Embed करने के लिए हमें <APPLET> </APPLET> Tag का प्रयोग करना पड़ता है।

इस Tag में हमें ये Specify करना होता है कि Web Page में Embed किए जाने वाले Applet का नाम क्या है और उस APPLET की Web Page में Size क्या होगी। यदि हमें किसी **Java** नाम के Applet को Web Page में Embed करना हो, तो हमें निम्नानुसार APPLET Tag Code लिखना होता है:

```
<HTML>
<! This is my first web page comment. >

<HEAD>
  <TITLE>Betalab Computer Center</TITLE>
</HEAD>
<BODY>
  <CENTER> Welcome to the world of Java. </CENTER>
  <BR>
```

```
<CENTER>
  <APPLET CODE = HelloJava.class WIDTH = 400 HEIGHT = 200></APPLET>
</CENTER>
</BODY>
</HTML>
```

## **Applet Architecture – The Event Based GUI Application Program**

किसी Applet को एक Web Page में किस Code द्वारा Embed किया जा सकता है और Web Page को Java Applet का प्रयोग करके किस प्रकार से अधिक Interactive बनाया जा सकता है, इस सम्बंध में थोड़ी Basic जानकारी लेने के बाद अब हम ये समझेंगे कि Applets को कैसे Develop किया जाता है। साथ ही हम ये भी जानेंगे कि Applets किस प्रकार से काम करते हैं और किस प्रकार से Define किए जाते हैं। यानी हम Applets के Architecture को समझेंगे।

Applet के सम्बंध में सबसे पहली बात ये है कि Applet एक Window-Based Event Driven Program होता है। **Event Driven Programming** के बारे में हम इसी अध्याय में आगे समझेंगे। चूंकि Applet एक Event Driven Window-Based Program होता है, इसलिए इसका Architecture किसी अन्य Normal Console-Based Program से अलग होता है।

इसलिए Applets Develop करने से पहले हमें ये जानना जरूरी है कि Applets किस प्रकार से काम करते हैं और किस प्रकार से इन्हें Design किया जाता है। Applets विभिन्न प्रकार के **Interrupts Service Routines** को Handle करने वाले Code Segments या Methods के Implementation पर आधारित होते हैं।

GUI Applications में Mouse या Keyboard से कुछ भी करना **Event** कहलाता है। यानी Mouse से Click करना या Keyboard से Key Press करना, इन दोनों स्थितियों में कुछ Events Generate होते हैं, जिनके बारे में हम आगे विस्तार से समझेंगे।

हमारा Applet इन्हीं Events का इन्तजार करता रहता है। जैसे ही कोई Event Generate होता है, AWT Package Applet द्वारा प्रदान किए गए किसी एक Event Handler Method को Call करता है और Generate होने वाले Event के सम्बंध में जानकारी दे देता है या Notify कर देता है।

एक बार ऐसा हो जाने के बाद Appropriate Action लेने और Control को वापस AWT में Pass करने की जिम्मेदारी Applet की होती है। हम किसी Applet पर जैसे ही किसी Mouse Button से Click करते हैं, एक Mouse Event Generate होता है, जिसे Applet में Define किया गया Handler Respond करता है। इसी तरह से यदि Keyboard से कोई Key Press की जाती है, तो भी एक Event Generate होता है, जिसे Applet का कोई Handler Respond करता है।

किसी Applet से सम्बंधित जितने भी Basic या Primary Initializations व Functionality हैं, उन्हें **Applet** नाम की Class में Define किया गया है। **Applet** नाम की ये Class **applet** नाम के Package में उपलब्ध है।

चूंकि जावा OOPS के Concepts को Implement करता है और एक पूर्ण Object Oriented Programming Language है, इसलिए हम इसमें Inheritance Concept को पूरी तरह से Apply कर सकते हैं और Packages में Classes के रूप में Stored विभिन्न प्रकार के पहले से Define किए गए Codes को ज्यों का त्यों Reuse कर सकते हैं।

इसलिए Applet Class को Inherit करके हम एक Applet से सम्बंधित विभिन्न Basic Initializations व Functionality को प्राप्त कर सकते हैं। चलिए, अब हम एक Applet को **Applet Class** से Inherit करके Applet Class की विभिन्न Properties व Functionality को प्राप्त करते हैं, यानी एक Applet बनाते हैं।

---

**Program: MyFirstApplet.java**

---

```
// Inheriting An Applet from Built-In Applet Class
import java.awt.*;
import java.applet.*;

public class MyFirstApplet extends Applet
{
}
```

---

इस Code को एक Notepad में लिखें "C:\\" Derive में एक JAVASOURCE नाम का Folder बना कर उसमें MyFirstApplet.java नाम से Save करें। अब Command Prompt पर जाकर निम्न Statement द्वारा उस Folder में पहुंचें, जहां पर MyFirstApplet.java को Save किया है, यानी JavaSource Folder में पहुंचें।

```
C:\>CD JAVASOURCE // Press Enter
```

अब दिखाई देने वाला Command Prompt निम्नानुसार होगा:

```
C:\JAVASOURCE>
```

अब Source File को Compile करने के लिए निम्न Command Type करें:

```
C:\JAVASOURCE>javac MyFirstApplet.java // Press Enter
```

जब बिना किसी Error के File Compile हो जाएगी, तब JAVASOURCE Folder में एक नई File बनेगी जिसका Extension **.class** होगा। यानी हमारी Source File Compile होने के बाद MyFirstApplet.class नाम की एक Class File बनेगी ये File एक Executable File होती है, जिसे किसी भी Platform में किसी भी Browser द्वारा एक Web Page में Embed करके Run किया जा सकता है।

इस Applet को **Test** करने के लिए हमें एक Web Page बनाना होगा। Web Page बनाने के लिए एक Notepad में निम्न HTML APPLET Tag लिख कर उसमें **MyFirstApplet.class** Filed को Embed कर सकते हैं।

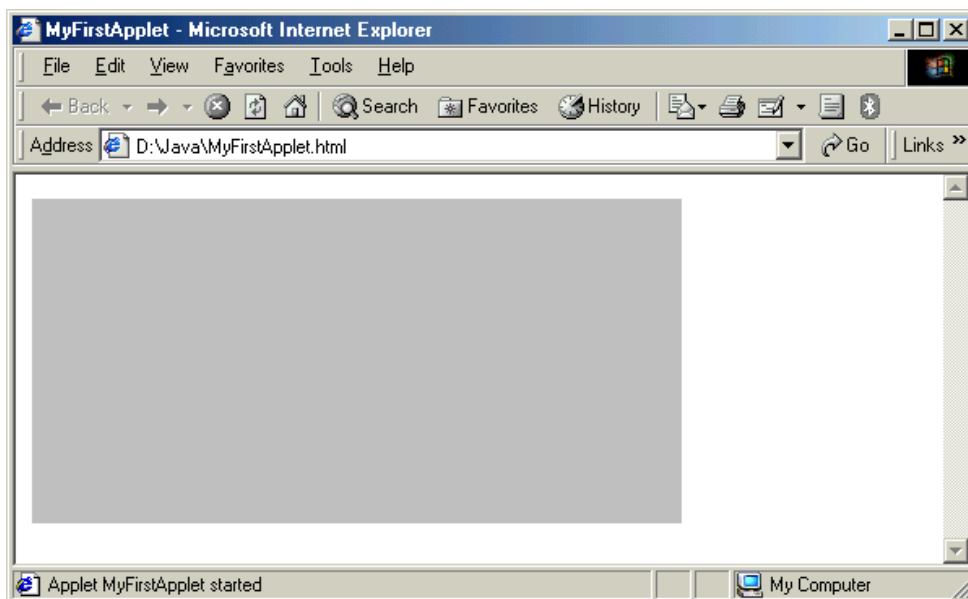
```
<! File Name : MyFirstApplet.HTML >
<! Embedding an Applet in a Web Page. >

<HTML>
<HEAD>
  <TITLE>My First Applet </TITLE>
</HEAD>
```

```
<BODY>
  <CENTER> Welcome to the world of Java. </CENTER>
  <BR>
  <CENTER>
    <APPLET CODE = MyFirstApplet.class WIDTH = 400 HEIGHT = 100></APPLET>
  </CENTER>
</BODY>
</HTML>
```

Notepad में ये HTML Tabs लिखने के बाद File को उसी Folder में Save करना होता है, जहां पर Java Applet की Class File Stored है। हम ये मान रहे हैं कि हमारी Source File व उसकी Class File दोनों ही "C:\JAVASOURCE" में Stored हैं। इसलिए इस HTML File को भी हमें यहीं पर Store करना होगा।

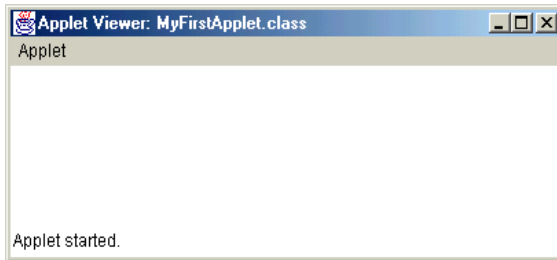
यदि हमारा Browser Java को Support करता है या Java Enabled है, तो हम Create होने वाले MyFirstApplet.HTML Web Page को Open करके Applet को Web Page में देख सकते हैं। Netscape Navigator एक Java Enabled Web Browser है, जिसमें हम किसी भी Web Page में Contained Applet को Directly देख सकते हैं। यानी यदि हम MyFirstApplet.HTML File को Open करें, तो हमें ये File निम्नानुसार दिखाई देगी:



यदि हमारा Browser Java Enabled नहीं है, तो हमें जावा के साथ आने वाले **Appletviewer** का प्रयोग करके हमें अपने Applet को Test करना होता है। Appletviewer का प्रयोग करके Applet को Test करने के लिए **MyFirstApplet.HTML** File Create करने के बाद Command Prompt पर निम्नानुसार Command लिखना होता है:

```
C:\JAVASOURCE>Appletviewer MyFirstApplet // Press Enter
```

जब ये Command लिखकर Enter Key Press करते हैं, तो हमें Output में निम्नानुसार Applet Viewer दिखाई देता है।



चलिए, अब हम इस Applet की Coding को समझने की कोशिश करते हैं। कोई भी Applet Create करने के लिए हमें जावा में पहले से Define की गई Applet Class के गुणों को Inherit करने के लिए एक Sub Class बनानी पड़ती है और पहले से बनी हुई Applet Class के सभी गुणों व Functionality को Derive करना पड़ता है। जावा किसी में Class को Inherit करने के लिए **extends** Keyword का प्रयोग करना होता है।

यानी जब हम अपनी कोई Applet Class बनाना चाहते हैं, तब हमें हमारी Applet Class को जावा की Applet Class से Inherit करना पड़ता है, ताकि हम हमारी Class में किसी Applet से सम्बंधित विभिन्न प्रकार की Properties व Functionality को ज्यों का त्यों प्राप्त कर सकें और बिना किसी प्रकार के विशेष Overhead के एक Applet Create कर सकें।

**extends** Keyword का प्रयोग इसलिए किया गया है, क्योंकि जावा में Applet से सम्बंधित जो Built-In सुविधाएं प्रदान करता है, हम उन सुविधाओं को तो ज्यों का त्यों Use कर लेते हैं, साथ ही Create होने वाली Sub Class में अपनी आवश्यकतानुसार नए Features भी Add कर सकते हैं। हमने हमारी Coding में निम्न Statement द्वारा जावा की Applet Class को Inherit किया है:

```
public class MyFirstApplet extends Applet
{
}
```

इस Statement में हमने जावा Compiler को बताया है कि हम Applet Class को Extend करके एक नई Class बना रहे हैं, जिसका नाम MyFirstApplet है और ये एक Public Class है। Public एक Access Specifier है, जो ये बताता है कि इस Class के Objects को किसी भी अन्य Class में Create किया जा सकता है। Access Specifier के बारे में हम आगे विस्तार से चर्चा करेंगे।

हम जब भी कोई Applet Extend करते हैं, तो Create होने वाले किसी भी Applet की Lifetime के मुख्य-मुख्य चार Events होते हैं और हर Event से Applet Class का एक Method Associated रहता है। किसी Applet की Life Cycle के चारों Steps को हम चार Methods द्वारा Represent कर सकते हैं:

## public void *init()* Method

जब User किसी ऐसे Web Page को Open करता है, जिसमें Applet Embed होता है, तो Web Page के Memory में Load होते ही Applet कुछ प्रारम्भिक मानों से Initialize होता है। Applet के ये प्रारम्भिक मान Applet Class के *init()* Method द्वारा Initialize होते हैं।

यदि हम Applet के Initialization के समय किन्हीं Variables को कोई मान प्रदान करना चाहें, या कुछ ऐसे काम करना चाहें, जो Applet के Initialize होते ही Execute करने हों, जैसे कि विभिन्न प्रकार के GUI Objects को Applet पर Add करना हो, तो इन कामों को करने के Codes हमें



init() Method में लिखने होते हैं। ये Method किसी Applet की Life में केवल एक ही बार Execute होता है।

यदि दुबारा init() Method को Execute करना हो, तो हमें Web Page को Close करके फिर से Open करना पड़ता है।

## **public void start() Method**

init() Method द्वारा Applet Initialize होने के बाद start() Method Call होता है। साथ ही हम जितनी भी बार Applet को Restart करते हैं, ये Method Execute होता है। Applets को उस समय Stop किया जा सकता है, जिस समय User एक Web Site से दूसरी Web Site पर Move होता है। यदि User कुछ समय बाद में फिर से उस Web Site पर आता है, तो Applet फिर से Restart होता है। इसलिए start() Method को Applet की Life Time में आवश्यकतानुसार कई बार Call किया जा सकता है।

Applet जब भी किसी Event के कारण Focus Receive करता है, ये Method Execute होता है। हम इस Method में उन Codes को लिख सकते हैं, जिन्हें Applet के Restart में Execute करवाना चाहते हैं। जैसे कि Applet में किसी Thread को Initialize करने के समय या Screen के Data को Update करते समय Applet हर बार एक Focus प्राप्त करता है जिससे हर बार Applet Restart होता है।

## **public void stop() Method**

ये Event तब Generate होता है, जब एक Web Browser किसी HTML Page को छोड़ कर किसी दूसरे Web Page पर जाता है, या Web Page पर से Focus Lost होता है। हम इस Method को तब Use कर सकते हैं जब हम Program के विभिन्न Variables को Reset करना चाहते हैं या किसी Running Thread को Stop करना चाहते हैं।

## **public void destroy() Method**

जब Applet पूरी तरह से Execute होना बन्द हो जाता है या जब User एक Page से दूसरे Page पर Move हो जाता है, तब ये Method Execute होता है। हम इस Method का प्रयोग किसी Applet द्वारा Use हो रहे किसी Resource को Free Up करने के लिए कर सकते हैं। **stop()** Method हमेशा **destroy()** Method से पहले Call होता है।

ये चारों ही Methods Automatically Call होते हैं और हम इन्हें किसी भी प्रकार से अलग से Call नहीं करते हैं। Applet में जो कुछ भी Display होता है, उसे Display करने के लिए हमें paint() Method को Use करना होता है। इस Method के बिना हम Applet में कुछ भी Display नहीं कर सकते हैं। इस Method के अलावा दो अन्य Methods **update()** व **repaint()** भी हैं, जिनका प्रयोग अलग-अलग परिस्थितियों में Applet को Paint करने के लिए करना पड़ता है।

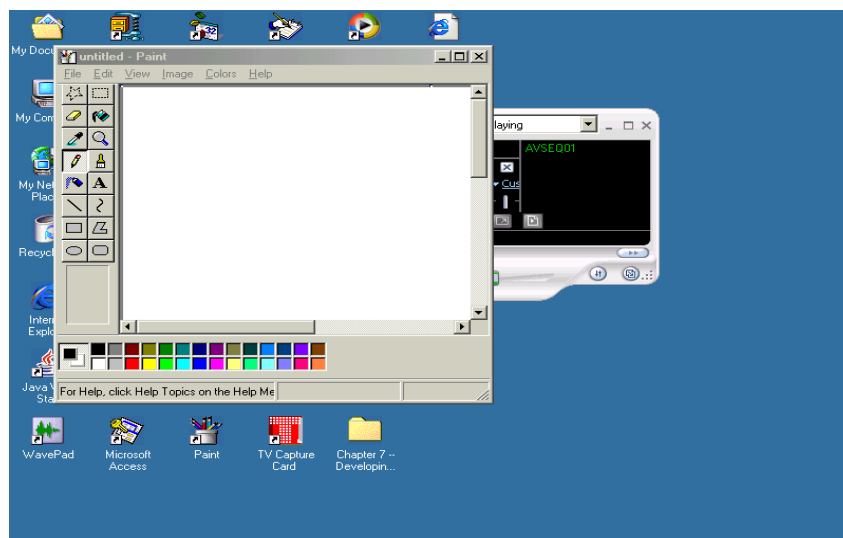
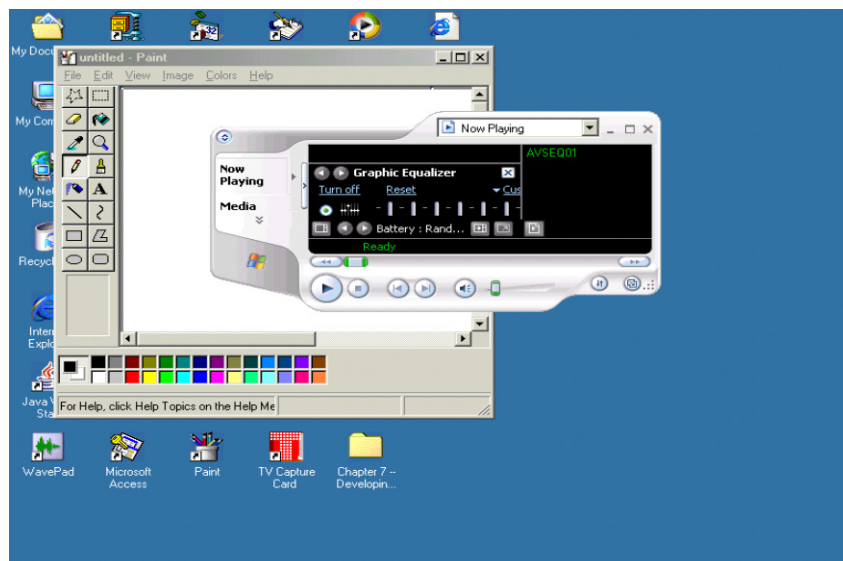
## **public void paint() Method**

जब भी हमें Applet Window को Paint करना होता है, तब हमें इस Method को Call करना पड़ता है। Applet Window को **Paint** करने का मतलब ये है कि जब भी हमें किसी Information OR Graphics को Applet में Display करना होता है, तो उस Information या Graphics को Applet

की Window में Draw कर देते हैं। Information या Graphics को Window में Display करने की प्रक्रिया को Window Paint करना कहते हैं।

कुछ स्थितियों में Paint Method स्वयं ही Automatically Call हो जाता है। जैसे जब एक Window Display हो रहा हो और उस Window पर कोई दूसरा Window Overwrite हो जाए और फिर पहले Window को Activate किया जाए, तो हमें दूसरा Window दिखाई देने से पहले Repaint होता है। इसे हम निम्न चित्र द्वारा समझ सकते हैं:

इस चित्र में MS Paint से Media Player Hide हो रहा है। अब यदि हम Media Player को Activate करें, तो Media Player फिर से Paint होता है और MS Paint Hide हो जाता है, जिसे हम निम्न चित्र में देख सकते हैं:



इसी तरह से जब हम किसी Window को Minimize या Restore करते हैं, तब भी वह Window Paint होता है। जब पहली बार Applet का Execution शुरू होता है, तब भी ये Method Call होता है।

कारण कोई भी हो, जब भी Window के Output को Redraw करना होता है, ये Method Call होता है। `paint()` Method को `repaint()` Method को Call करके Trigger किया जा सकता है।

Applet Class के `paint()` Method में Parameter के रूप में Graphics Class का एक Object Specify करना होता है। इस Parameter में उस Graphics Object की Information होती है, जिसे Applet पर Draw करना होता है।

## **public void update() Method**

ये Method `paint()` Method की तरह ही काम करता है। इस Method को Call करने पर ये Method Screen को Clear करता है और फिर से `paint()` Method को Call करके Screen को Repaint कर देता है।

## **public void repaint() Method**

जब हम Applet को Repaint करना चाहते हैं, तब हम `repaint()` Method को Call कर सकते हैं। `repaint()` Method Automatically `update()` Method को Call करके Screen को Redraw करता है।

## **First Applet in Java**

किसी Applet में Text को Display करना सबसे सरल काम है। लेकिन Applet Window Based GUI को Support करता है और Window में जो कुछ भी दिखाई देता है, वह सबकुछ Graphical Form में होता है, इसलिए जावा के Graphical Text Function को Use करके Applet में String को Draw करना पड़ता है।

जावा में String को Draw करने के लिए `drawString()` Method को Generally Use किया जाता है। ये Method `awt` Package में स्थित Graphics Class का एक हिस्सा है। Package कुछ सम्बंधित प्रकार की Classes के Collection के अलावा और कुछ नहीं होता है। निम्न Code Segment एक Applet का है, जिसमें एक String को Display करवाया गया है।

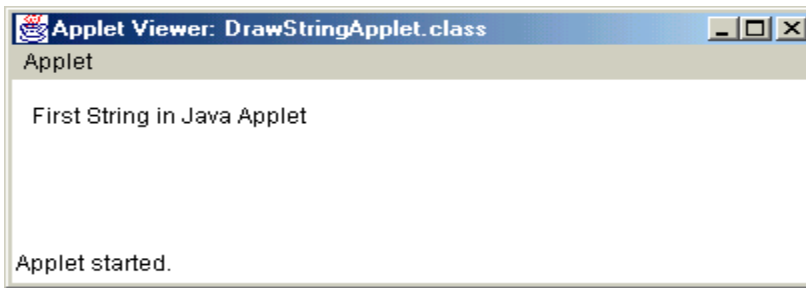
---

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

public class DrawStringApplet extends Applet
{
    public void paint(Graphics firstString)
    {
        firstString.drawString("First String in Java Applet ", 10, 25);
    }
}
```

---

**Output:**



इस Program में सबसे पहले `awt` व `applet` Package को Use करने के लिए Import किया गया है। Import Statement के कारण हम Applet Class के सभी Features को `DrawStringApplet` नाम की नई Class में Derive या Inherit करने की सुविधा प्राप्त करते हैं।

जब हम Applet Class को Derive करके उसके Features को `DrawStringApplet` Class में प्राप्त करते हैं, तो हम इसी Class में Applet Class के विभिन्न Methods को भी प्राप्त करते हैं।

`paint()` Method भी Applet Class का ही एक Method है। ये Method हमें Applet Class को Derive करने से प्राप्त होता है। इस Method को Override करके हम Applet पर String को Draw कर सकते हैं। String को Draw करने के लिए Graphics Class का `drawString()` Method Execute किया गया है।

### **public abstract void drawString(String text, int x, int y)**

इस Program में हमने Applet Class के `drawString()` Method द्वारा एक String को Applet पर `x` व `y` की Location पर Display किया है। हमें जिस String को Screen पर Display करना होता है, उसे इस Method में पहले Argument के रूप में लिखना होता है। दूसरा व तीसरा Argument एक Number होता है, जो Applet पर String की Display होने की Location को Specify करते हैं।

### **Flow of This Applet**

हम जानते हैं कि जब भी Applet की Screen को Draw करना होता है, तब Applet Class का `paint()` Method Execute होता है। जब पहली बार Applet Screen पर Appear होता है, तब `paint()` Method पहली बार Execute होता है।

इस Program में जब हम Web Page Run करते हैं और Applet Appear होता है, तब Applet के Appear होते ही जावा `paint()` Method को Call करता है और `paint()` Method `drawString()` Method को Call करता है, जो कि String को Applet पर Display कर देता है।

`paint()` Method Graphics Class का Method है। ये Method किसी Graphics Object को Applet में Display करने का काम करता है। इस Program में हम देख सकते हैं इसके Parenthesis में `firstString` नाम का एक Object Pass किया गया है।

इसका मतलब ये है कि जावा `paint()` Method में Graphics Class का एक Object Argument के रूप में भेज रहा है, जिसका नाम `firstString` है। चूंकि `firstString` Graphics Class का एक Object है। इसलिए यदि हम इस Object को Display करना चाहते हैं, तो हमें Graphics Class के ही किसी ऐसे Method को Call करना होगा, जो हमारी जरूरत को पूरा कर दे।

चूंकि हम एक String को Display करना चाहते हैं, इसलिए हमें firstString Object के लिए drawString() Method को Call करना होगा। जब हमें किसी Object के लिए किसी Method को Call करना होता है, तब हमें उस Object को Dot का प्रयोग करके Method से जोड़ते हुए Call करना पड़ता है, ताकि Call होने वाला Method उसी Object पर प्रक्रिया करे, जिसके साथ उसे Call किया गया है। इसीलिए हमने हमारे Program में निम्न Code Line लिखी है:

```
firstString.drawString("First String in Java Applet ", 10, 25);
```

ये Statement जावा Compiler को firstString Object के drawString() Method को Call करने के लिए Message देता है। drawString() Method के Parenthesis में प्रदान किया जाने वाला मान Argument कहलाता है। ये वे मान होते हैं, जिन्हें Method में Information के रूप में Pass करना होता है।

**drawString()** Method के Arguments जावा Compiler को “**First String in Java Applet**” String को Column Number 10 व Row Number 25 पर Display करने के लिए Instruction देता है। Row व Columns को Pixels में Measure किया जाता है ना कि Characters में। Pixel किसी Screen पर दिखाई देने वाला सबसे छोटा Dot होता है। Display होने वाले Text को अलग Location पर Display करना हो, तो हमें Row व Column की संख्या Change करनी होती है।

## GUI – The Event Driven Programming System

मानलो कि हम सुबह-सुबह जैसे ही Office जाने के लिए घर से बाहर निकलने वाले होते ही हैं कि तुरन्त Phone की घण्टी बजने लगती है। Phone की घण्टी का बजना एक प्रकार का **Event** है। Real Life में कई बारे ऐसे Events होते हैं, जिन्हें बाकी सभी कामों को **Suspend** करके तुरन्त **Respond** करना पड़ता है।

उदाहरण देखें तो हम चाहे जितनी भी जल्दी में हों, लेकिन अगर Phone की घण्टी बजती है, तो हमें उस Phone को Attend करना ही पड़ता है।

इसी Concept को जब हम Programming के सन्दर्भ में देखते हैं, तब किसी Program व उसके User के बीच जितनी भी Activities होती हैं, उन सभी Activities को जावा में **Events** कहा जाता है।

User Keyboard से कोई **Key Press** करता है या Mouse से Application के किसी Command Button पर **Click** करता है या Application Program के Window को **Resize** करता है, Software या Hardware का Fail होना, किसी List में से किसी Item को Select करना, Timer के Counter का Increment होना आदि विभिन्न प्रकार के Events हैं।

जब User किसी Program के साथ Interaction करता है, तब Computer एक Event Object Create करता है। ये Event Object उस Action को Represent करता है, जिसे User ने Perform किया है।

यानी मानलो कि User ने किसी Application के किसी Button पर Click किया, तो Computer एक **Object Create** करेगा और उस Object द्वारा **Click Event Represent** होगा।

फिर हम इस Click Event को एक **Event Handler Coding** द्वारा Handle कर सकते हैं और वह काम करवाते हैं, जो Click Event के Generate होने पर करवाना होता है।

ये Event Handler Code ही ये तय करता है कि किसी Event को किस प्रकार से Handle करना है। किसी Event को विभिन्न प्रकार से Drive करके Program Develop करने व उसे Handle करने की तकनीक को **Event Driven Programming** कहते हैं, जिसमें विभिन्न प्रकार की Events को Response करने वाले Code Handlers लिख कर Programming या Development किया जाता है।

जावा एक Pure Object Oriented Programming Language है, इसलिए हम इस Language में किसी भी चीज को एक Object के रूप में Represent करते हैं।

इसीलिए User द्वारा किसी Hardware को Access करने से कोई Event Generate हो या किसी Software द्वारा कोई Event Generate हो, दोनों ही स्थितियों में किसी Event को Represent करने के लिए एक Event Object Create होता है।

GUI Programming में Event Handle करना सबसे जरूरी काम होता है। GUI Program या Event Driven Programming Modal में हमारा GUI Program User के किसी भी Action का इन्तजार करता रहता है।

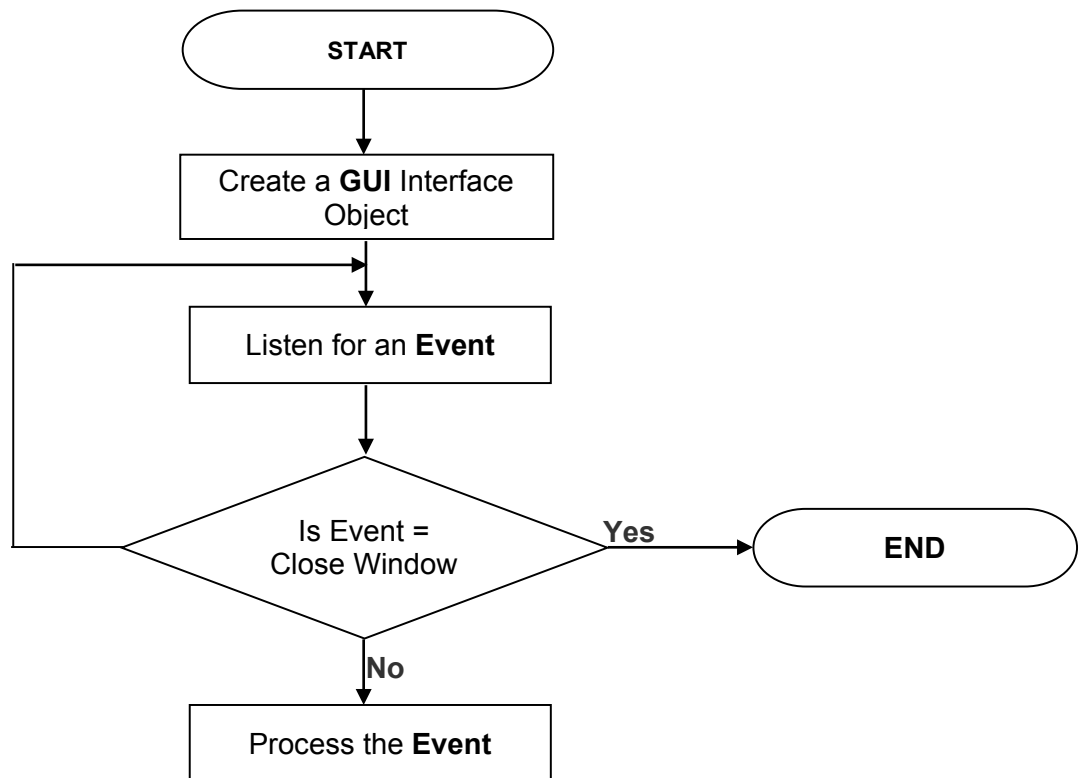
जब User उस Application के साथ किसी प्रकार का Interaction करता है, तो उस Interaction से सम्बंधित कोई ना कोई Event Generate होता है और उस Event को Handle करने के लिए कोई ना कोई Handler लिखा गया होता है, जो कि Event के Response में Execute हो जाता है।

GUI की सबसे बड़ी विशेषता ये है कि किसी GUI Application के Operations के Perform होने के क्रम को User निर्धारित करता है, यानी वह किसी Application पर दिखाई दे रहे किसी भी Button को या किसी भी अन्य Control को किसी भी समय Use कर सकता है, जबकि CUI Applications में Program के Execution का Sequence Programmer तय करता है और User को उसी प्रकार से चलना पडता है जिस तरह से Programmer ने Program को लिख कर तय किया है। Event Driven Program निम्नानुसार Execute होते हैं:

### Algorithm of GUI Application

- |   |  |   |
|---|--|---|
| 1 | START  | [Starting the Window Based GUI Application ]      |
| 2 | CREATE GUI OBJECT[Create a GUI Application Object like A Window] |   |
| 3 | LISTEN EVENT   | [Listen for an Event Occurred by the Interaction] |
| 4 | IF EVENT = CLOSE APPLICATION<br>EXIT<br>ELSE<br>PROCESS EVENT    | [If Event is for closing the Application]         |
| 5 | GOTO STEP 3  | [Go to the Step 4 again and get next Event ]      |

## Flow of GUI Application



## Components of an Event

जावा में किसी भी Event को तीन हिस्सों में बंटा हुआ माना जा सकता है:

### Event Object

जब User Keyboard के किसी Button को Press करके या Mouse के किसी Button को Click करके किसी Application से Interact करता है, तब एक Event Generate होता है। Event Driven Architecture पर आधारित Operating System इस Event व Event से Related जरूरी Data को Trap कर लेता है।

उदाहरण के लिए User किसी Application के Window पर Click करता है, तब User ने किस Button से Click किया है और किस **x, y** Location पर Click किया है, इस बात की जानकारी Data के रूप में उसी Application के उसी Object को वापस भेज देता है, जिस Object पर Click करने पर Event Generate हुआ होता है।

जावा में हर Event को एक Object द्वारा Represent किया जाता है और वह Object Generate होने वाले Event से सम्बंधित विभिन्न जानकारियों को Hold करके रखता है। Event Object के किसी Data Member के मान में परिवर्तन होने का मतलब है कि कोई Event Generate हुआ है।

या फिर हम ऐसा भी कह सकते हैं कि किसी Event Object के किसी Data Member के मान में परिवर्तन होने को ही हम Indirectly Event Generate होना कह सकते हैं। जावा में विभिन्न

प्रकार के Events को Describe व Handle करने के लिए कई Classes पहले से ही Develop करके Library के रूप में प्रदान कर दी गई हैं।

## Event Source

Event Source एक ऐसा Object है जो Event Generate करने में सक्षम होता है या Event Generate करता है। Event तब Generate होता है, जब किसी तरीके से किसी Event Object की **Internal State** यानी Object के किसी Data Member की स्थिति में परिवर्तन होता है। कोई Event Source एक से अधिक प्रकार की Events को भी Generate कर सकता है।

उदाहरण के लिए यदि हम किसी Button पर Click करते हैं, तो एक **ActionEvent** Object Generate होता है। **ActionEvent Class** के Object में इस Event के बारे में विभिन्न प्रकार की Information Stored रहती हैं।

## Event Handler

Event – Handler एक Method होता है जो Generate होने वाली Event को समझता है और उसे Process करने में सक्षम होता है। Event Handler Method एक Event प्रकार के Object को Parameter के रूप में लेता है।

ActionEvent Class में **getSource()** नाम का एक Method होता है। ये Method उस Component का Reference Return करता है, जिसने Event Generate किया है। यानी किसी Application Window पर स्थित एक Command Button को Click करने पर जो Click Event Generate होता है, उस Click Event को Generate करने वाले Command Button को **getSource()** Method Refer करता है।

---

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

public class DrawStringAppletWithEvent extends Applet
{
    TextField textBox = new TextField(50);

    public void init()
    {
        add(textBox);
    }

    public void paint(Graphics firstString)
    {
        firstString.drawString("Enter String in Text Box ", 6, 50);

        String strOfTextBox = textBox.getText();

        firstString.drawString("String in Text Box is ", 6, 100);
        firstString.drawString(strOfTextBox, 30, 120);
    }
}
```

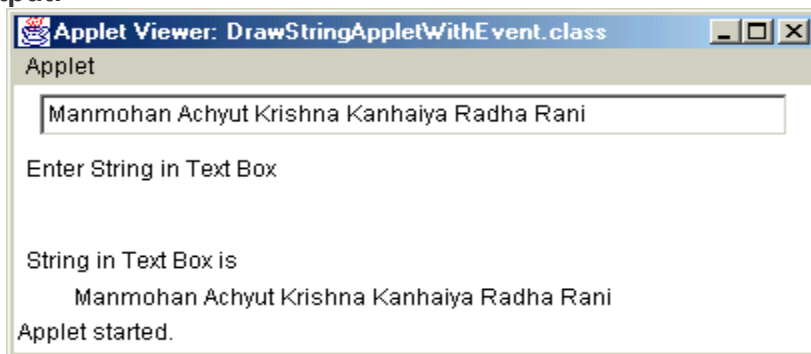


```

public boolean action(Event evnt, Object obj)
{
    repaint();
    return true;
}
}

```

Output:



इस Program में हमने Applet Class को Derive करके **DrawStringAppletWithEvent** नाम की एक Class बनाई है और इसमें Applet Class के सभी Attributes व Functionality को Inherit किया है। फिर हमने TextField Class का एक Object textBox Create किया है और उसकी Size 50 Define की है।

यानी हम Text Box में एक समय में 50 अक्षर देख सकते हैं। Create होने वाला ये Object अभी Memory में ही Create हुआ है। हम इसे तब तक Applet पर नहीं देख सकते हैं, जब तक कि हम इसे Applet पर Add ना करें। Applet पर इस textBox Object को Add करने के लिए Applet Class के **init()** Method को Override करके Create किए गए textBox Object को **Add()** Method द्वारा Applet पर Add किया गया है।

इसके बाद Applet Class के **paint()** Method को Override करके एक Message **“Enter String in Text Box”** को Applet पर Display किया है। फिर String Class का एक Object **strOfTextBox** Create किया है और TextField Class के एक **getText()** Method को Call करके textBox Object में Type किए गए Texts को इस **strOfTextBox** Object में Hold किया गया है। फिर **drawString()** Method का प्रयोग करके textBox में Type की गई String को ज्यों का त्यों Applet पर Display करवा दिया गया है।

इस Program में हमने **action** नाम के Event Method को Override किया है और इस Method में **repaint()** Method को Call किया है। इस Applet Program को Compile करके Run करने पर हमें Applet Window में एक Text Box दिखाई देता है।

इस Text Box में जो कुछ भी Type करके हम Enter Key Press करते हैं, हमें वह सब कुछ Applet पर दिखाई देने लगता है। इसका कारण ये है कि जब भी हम Text Box में कुछ लिखते हैं और Enter Key Press करते हैं, तो एक **action** Event Generate होता है। इस action Event के Generate होते ही, इस Event को Handle करने के लिए Applet में **action** नाम का एक Method लिखा गया है। इस Method में **repaint()** Method को Call किया गया है।

जैसे ही हम Text Box में कुछ Type करके Enter Key Press करते हैं, action Event Trigger होता है। ये Event Trigger होते ही, action() Method Execute होता है और Applet के paint() Method को फिर से Call करता है, लेकिन इस बार Text Box में जो Texts लिखा जाता है, वह Text Applet पर दिखाई देने लगता है।

## How to Get this Ebook in PDF Format

ये पुस्तक केवल **PDF Format Ebook** के रूप में ही Available है और आप इस पुस्तक को केवल हमारी **Official Website** (<http://www.bccfalna.com/>) से ही खरीद सकते हैं। इसलिए यदि आपको ये पुस्तक पसन्द आ रही है और आप इसे PDF Format Ebook के रूप में खरीदना चाहते हों, तो आप इस पुस्तक को Online खरीदने के लिए निम्नानुसार दिए गए **3 Simple Steps Follow** कर सकते हैं:

## Select Purchasing EBooks

सबसे पहले <http://www.bccfalna.com/how-to-pay/> Link पर Click कीजिए। जैसे ही आप इस Link पर Click करेंगे, आप हमारी Website के निम्नानुसार **Order Page** पर पहुंच जाएंगे :

<input checked="" type="checkbox"/>	<b>C Programming Language in Hindi</b>	<b>300/-</b>
<input type="checkbox"/>	<b>C++ Programming Language in Hindi</b>	<b>300/-</b>
<input type="checkbox"/>	<b>Java Programming Language in Hindi</b>	<b>300/-</b>
<input checked="" type="checkbox"/>	<b>C# Programming Language in Hindi</b>	<b>400/-</b>
<input type="checkbox"/>	<b>Data Structure and Algorithms in Hindi</b>	<b>250/-</b>
<input type="checkbox"/>	<b>Oracle 8i/9i – SQL/PLSQL in Hindi</b>	<b>300/-</b>
<input type="checkbox"/>	<b>Visual Basic 6 in Hindi</b>	<b>250/-</b>
<input type="checkbox"/>	<b>HTML5 with CSS3 in Hindi</b>	<b>350/-</b>
<input type="checkbox"/>	<b>Advance JavaScript in Hindi</b>	<b>350/-</b>
<input type="checkbox"/>	<b>jQuery in Hindi</b>	<b>350/-</b>
<input type="checkbox"/>	<b>Core PHP in Hindi</b>	<b>350/-</b>
	<b>Total</b>	<b>Rs. 700/- Only.</b>
	<b>Discounted Amount</b>	<b>Rs. 100/- Only.</b>
	<b>Total Payable Amount</b>	<b>Rs. 600/- Only.</b>

इस Page पर आपको उन पुस्तकों को Select करना है, जिन्हें आप खरीदना चाहते हैं। आप जैसे-जैसे पुस्तकें Select करते जाएंगे, आपको उनका **Total Amount**, **Discount** व **Total Payable Amount** उपरोक्त चित्रानुसार दिखाई देने लगेगा, जहां Total Payable Amount ही वह Amount है, जो आपको अपनी Selected EBooks को खरीदने के लिए **Pay** करना होगा।

पुस्तकें Select करने के बाद इसी Page पर दिखाई देने वाले **“Order Details”** Form में आपको निम्न चित्रानुसार अपना **Name**, **Email Address** व **Mobile Number** Specify करके **“Order Now”** पर Click करते हुए उपरोक्त Selected EBooks का Order Place करना होगा:

<b>Order Details</b>	
<b>Your Name</b>	<input type="text" value="Kuldeep Mishra"/>
<b>Email Address</b>	<input type="text" value="bccfalna@gmail.com"/>
<b>Mobile Number</b>	<input type="text" value="09799455505"/>
<input type="button" value="Order Now"/>	

चूंकि ये सारी पुस्तकें Physical Books नहीं बल्कि **PDF Format Ebooks** हैं। इसलिए ये पुस्तकें आपको आपके Email पर ही भेजी जाएंगी, जिन्हें आप अपने Email के माध्यम से अपने Computer पर Download करके अपने PDF Supported *Computer, Mobile, Smart Phone, Tablet PC, Net-Book, Notebook* या *Laptop* जैसी किसी भी Device के माध्यम से पढ़ सकते हैं अथवा यदि आप चाहें, तो अपने Printer द्वारा इन पुस्तकों का Hard Copy Printout निकाल सकते हैं।

इसलिए जरूरी है कि उपरोक्त “**Order Details**” Form पर आप जो **Email Address** व **Mobile Number** Specify करते हैं, वह Working और एकदम सही हो। क्योंकि किसी भी तरह की परेशानी की स्थिति में हम आपको आपके **Mobile Number** पर ही Contact करते हैं।

## Pay “Total Payable Amount”

जैसे ही आप “**Order Now**” Button पर Click करेंगे, आपको एक Email मिलेगा, जिसमें आप द्वारा Order की गई EBooks की Details होगी। Selected पुस्तकों का Order Place करने के बाद अब आपको “**Total Payable Amount**” का Payment करना होगा।

यदि आपके पास **Net-Banking** या **Mobile-Banking** की सुविधा है, तो आप Payment करने के लिए अपने Account में Login करके निम्न में से किसी भी Bank A/c में Payment Deposit कर सकते हैं:

**भारतीय स्टेट बैंक**  
**State Bank of India**  
*With you - all the way*

SBI Bank A/c no.	:	<b>31154882587</b>
Account Name	:	<b>Namita Mishra</b>
Branch Name	:	<b>Falna</b>
Address	:	<b>Near Railway Crossing, Falna Station – 306116</b>
IFSC Code	:	<b>SBIN0007868</b>

**बैंक ऑफ़ बड़ौदा**  
**Bank of Baroda**  
*India's International Bank*

BOB Bank A/c no.	:	<b>35260100003212</b>
Account Name	:	<b>Namita Sharma</b>
Branch Name	:	<b>Falna</b>
Address	:	<b>Sanderao Road, Falna, Dist. Pali (Raj.)- Pin-306116</b>
IFSC Code	:	<b>BARB0FALNAX</b>

**स्टेट बैंक ऑफ़ बीकानेर एण्ड जयपुर**  
**State Bank of Bikaner and Jaipur**  
*The Bank with a vision*

SBBJ Bank A/c no.	:	<b>61089986732</b>
Account Name	:	<b>Kuldeep Chand Mishra</b>
Branch Name	:	<b>Bali</b>
Address	:	<b>Sr. Secondary School Road, Bali- 306701</b>
IFSC Code	:	<b>SBBJ0010193</b>

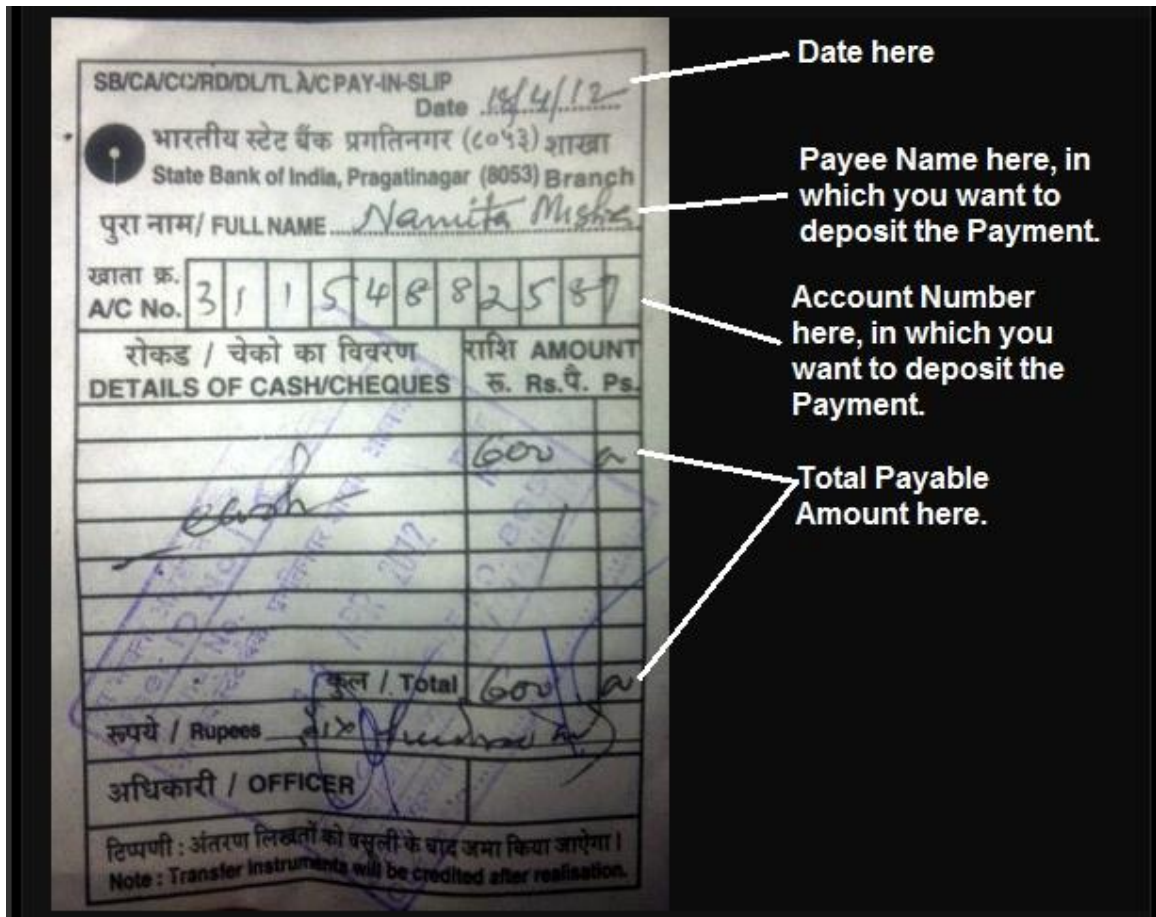
PNB Bank A/c no. : 4445000100034960  
 Account Name : Kuldeep Chand  
 Branch Name : Falna  
 Address : College Road, Falna Station – 306116  
 IFSC Code : PUNB0444500

जब आप Net-Banking के माध्यम से Payment करना चाहते हैं, तो आपको लगभग **8 से 24 घण्टे पहले** हमारे उस Account को **Beneficiary** के रूप में अपने Bank A/c से Link करना पड़ता है, जिसमें आप Payment Deposit करना चाहते हैं।

यदि आपके पास Net-Banking या Mobile-Banking की सुविधा नहीं है, तो आप हमारे किसी भी Bank A/c में **Total Payable Amount, Direct Deposit** भी कर सकते हैं।

जब आप **Direct Deposit** करना चाहते हैं, तब आपको आपके किसी भी नजदीकी Bank Branch में जाकर एक **Payment Deposit Slip Fill-Up** करना होता है, जिसमें आपको हमारे किसी भी Bank A/c की Information को Fill करना होता है, **जबकि Payment Deposit करवाने के लिए उसी Bank में आपका स्वयं का Account होना जरूरी नहीं है।**

उदाहरण के लिए यदि आप हमारे SBI Bank A/c में अपनी Selected पुस्तकों का **Total Payable Amount** Pay करने के लिए Bank में जाकर Direct Deposit करना चाहते हैं, तो आप जो **Payment Deposit Slip** Fill-Up करेंगे, वह निम्न चित्रानुसार करना होता है:



SB/CA/CC/RD/DL/TL A/C PAY-IN-SLIP  
 Date 18/4/12

भारतीय स्टेट बैंक प्रगतिनगर (८०५३) शाखा  
 State Bank of India, Pragatinagar (8053) Branch

पुरा नाम/ FULL NAME Nanita Mishra

खाता क्र. A/C No. 31154882587

रोकड / चेको का विवरण DETAILS OF CASH/CHEQUES	राशि AMOUNT रु. Rs. प. Ps.
600	
कुल / Total	600

रुपये / Rupees 600

अधिकारी / OFFICER

टिप्पणी : अंतरण लिखतों को वसूली के बाद जमा किया जाएगा।  
 Note : Transfer Instruments will be credited after realisation.

इस चित्र द्वारा आप समझ सकते हैं कि Payment, Direct Deposit करने के लिए आपको हमारे किसी Bank A/c की Information को *Payment Deposit Slip* में Specify करना होता है, इसलिए उस Bank में आपका स्वयं का Bank A/c होना जरूरी नहीं होता।

**Net-Banking, Mobile-Banking** व **Direct Deposit** के अलावा किसी अन्य माध्यम से भी आप Payment कर सकते हैं। उदाहरण के लिए कुछ Banks अपनी ATM Machine द्वारा Direct Payment Transfer करने की सुविधा Provide करते हैं।

यदि आपके Bank का ATM Machine इस तरह से Payment Transfer करने की सुविधा देता है, तो आपको Bank में जाकर Payment Deposit Slip के माध्यम से Payment करने की जरूरत नहीं होती, बल्कि आप Bank के ATM Machine से भी Directly हमारे किसी भी Bank A/c में **Total Payable Amount** Transfer कर सकते हैं। इसी तरह से यदि आप चाहें, तो हमारे किसी भी Bank A/c में Check द्वारा भी Amount Direct Deposit कर सकते हैं।

यानी आप किसी भी तरीके से हमारे किसी भी Bank A/c में *Total Payable Amount* Deposit कर सकते हैं। लेकिन हम **Money-Order, Demand-Draft** या **Check** जैसे Manual माध्यमों से Payment Accept नहीं करते, क्योंकि इस तरह का Payment Clear होने में बहुत समय लगता है। जबकि Direct Deposit या Mobile अथवा Net-Banking के माध्यम से तुरन्त Payment Transfer हो जाता है, जिससे हम आपको आपकी Purchased EBooks **20** से **30 Minute** के दरम्यान आपके Order में Specified **Email** पर Send कर देते हैं।

## Confirm the Payment

जब आप अपनी Selected पुस्तकों को खरीदने के लिए उपरोक्तानुसार किसी भी तरीके से “*Total Payable Amount*” हमारे किसी भी Bank A/c में Deposit कर देते हैं, तो Payment Deposit करते ही आपको हमें उसी Mobile Number से एक **Call/Miss Call/SMS** करना होता है, जिसे आपने Order Place करते समय “**Order Form**” में Specify किया था।

इसी Mobile Number के माध्यम से हमें पता चलता है कि आपने किन पुस्तकों के लिए Order किया है और उनका *Total Payable Amount* कितना है। साथ ही हमें ये भी पता चल जाता है कि आप द्वारा Purchase की जा रही पुस्तकें किस Email Address पर Send करनी है।

आपके *Total Payable Amount* को हम Net-Banking के माध्यम से अपने Bank A/c में Check करते हैं और यदि आपका *Total Payable Amount* हमारे किसी भी Bank A/c में Deposit हुआ होता है, तो हम आपको **30 Minute** के दरम्यान आपकी Ordered EBooks आपके Email पर Send कर देते हैं, जिसे आप अगले दिन 12AM तक Download कर सकते हैं।

यदि अभी भी आपको कोई बात ठीक से समझ में न आ रही हो या किसी भी तरह का Confusion हो, तो आप **097994-55505** पर **Call/Miss Call/SMS** कर सकते हैं। यथा सम्भव तुरन्त आपको Callback किया जाएगा और आपकी समस्या या Confusion का Best Possible Solution करने की कोशिश की जाएगी।

उम्मीद है, इस पुस्तक के Sample Chapters का Demo भी आपको पसन्द आया होगा और हमें पूरा विश्वास है कि पूरी पुस्तक आपको और भी ज्यादा पसन्द आएगी।