UNIT-1 HISTORY, INTRODUCTION AND LANGUAGE ,BASICS CLASSES AND OBJECT

PART-1

History And Features Of Java

Java Editions

JDK , JVM And JRE

JDK Tools

Compiling And Executing Basic Java Program

Java IDE (NetBeans And Eclipse)

Data Type (Integer , Float ,Character ,Boolean)

Java Tokens(keyword,literal,identifier,whitespace,separetors,comments,operators)

Java Keyword(assert, Strictfp , enum)

Typecasting

Decision Statement (If, switch)

Looping Statement (For, while, do...While)

Jumping Statement (Break, continue, Return)

Array (One Dim., Rectangular, jagged)

Command Line Argument Array

JAVA

- JAVA stands for Just Another Virtual Accelerator.

- It is a computer-based programming language that is used as a platform in itself.

- JAVA is an object-oriented, network-centric programming language for programming any mobile apps as well as software.

History of Java

- Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time.

- The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming. Later, Java technology was incorporated by Netscape.

- Java was developed by **James Gosling**, who is known as the father of Java, in **1995** at Sun Microsystems, later it occupies by oracle.

Features of Java (Java buzzwords)

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic
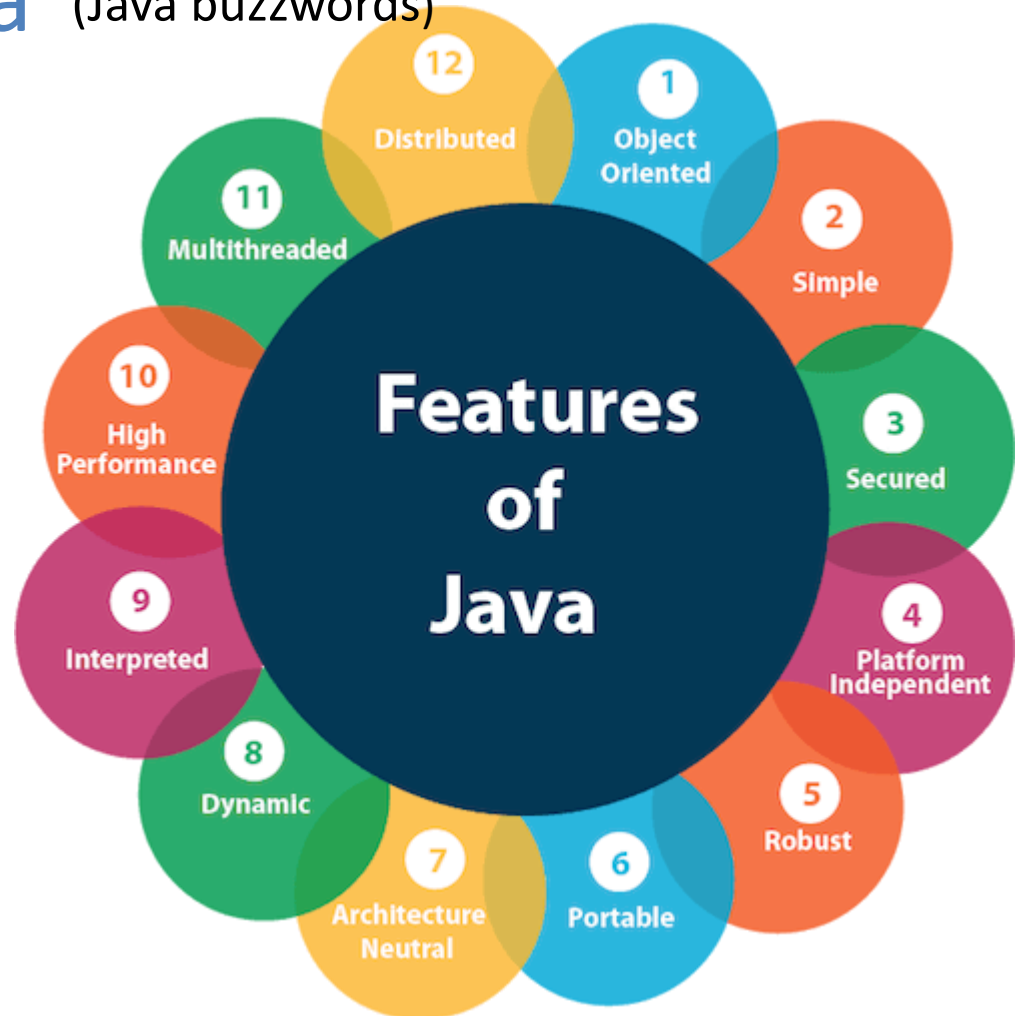
Simple

- Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).

- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.

- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

- Java is an object-oriented programming language.
- Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporate both data and behaviour.
- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
- Basic concepts of OOPs are:
1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

Platform Independent

- Java is platform independent because it is different from other languages like c,c++ etc. which are compiled into platform specific ==machines while Java is a write once, run anywhere(WORA) language.==

- A platform is the hardware or software environment in which a program runs.

- The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

  1) Runtime Environment
  2) API(Application Programming Interface)

- Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc.

- Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms.(WORA)
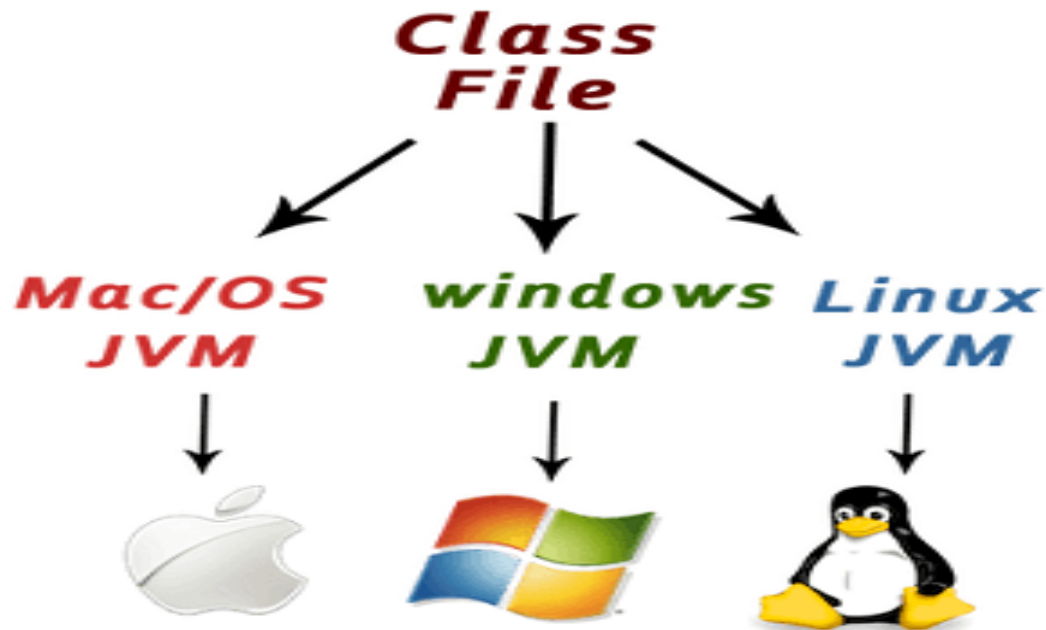
Secured

- Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

✓ <mark>No explicit pointer</mark>

✓ <mark>Java Programs run inside a virtual machine sandbox</mark>
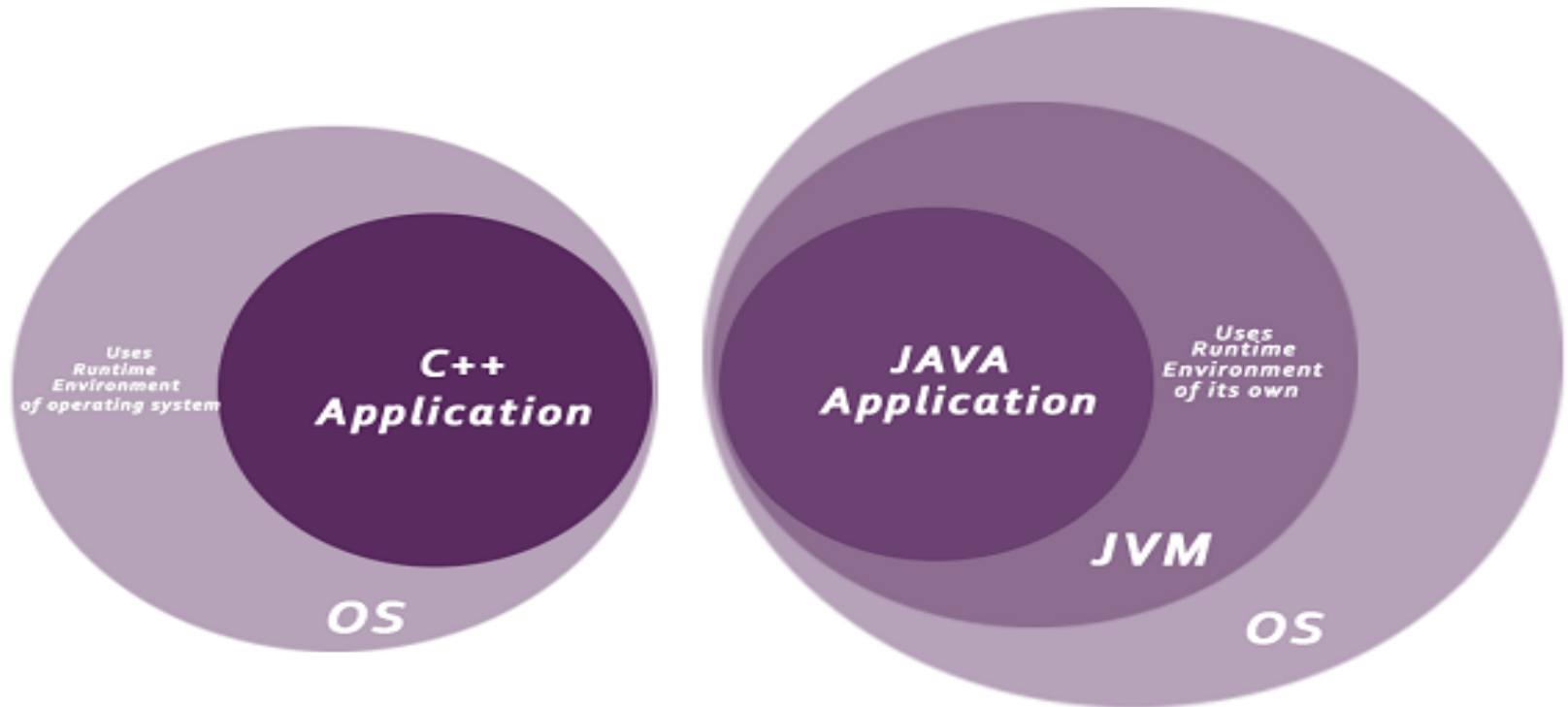
Robust

The English mining of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

❖ Architecture-neutral   ❖   Portable

- Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

- In C programming, integer data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

- Java is portable because it facilitates you to carry the Java bytecode to any platform.

- It doesn't require any implementation.

High-performance

- Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

- It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

- Java is distributed because it facilitates users to create distributed applications in Java.

- RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.
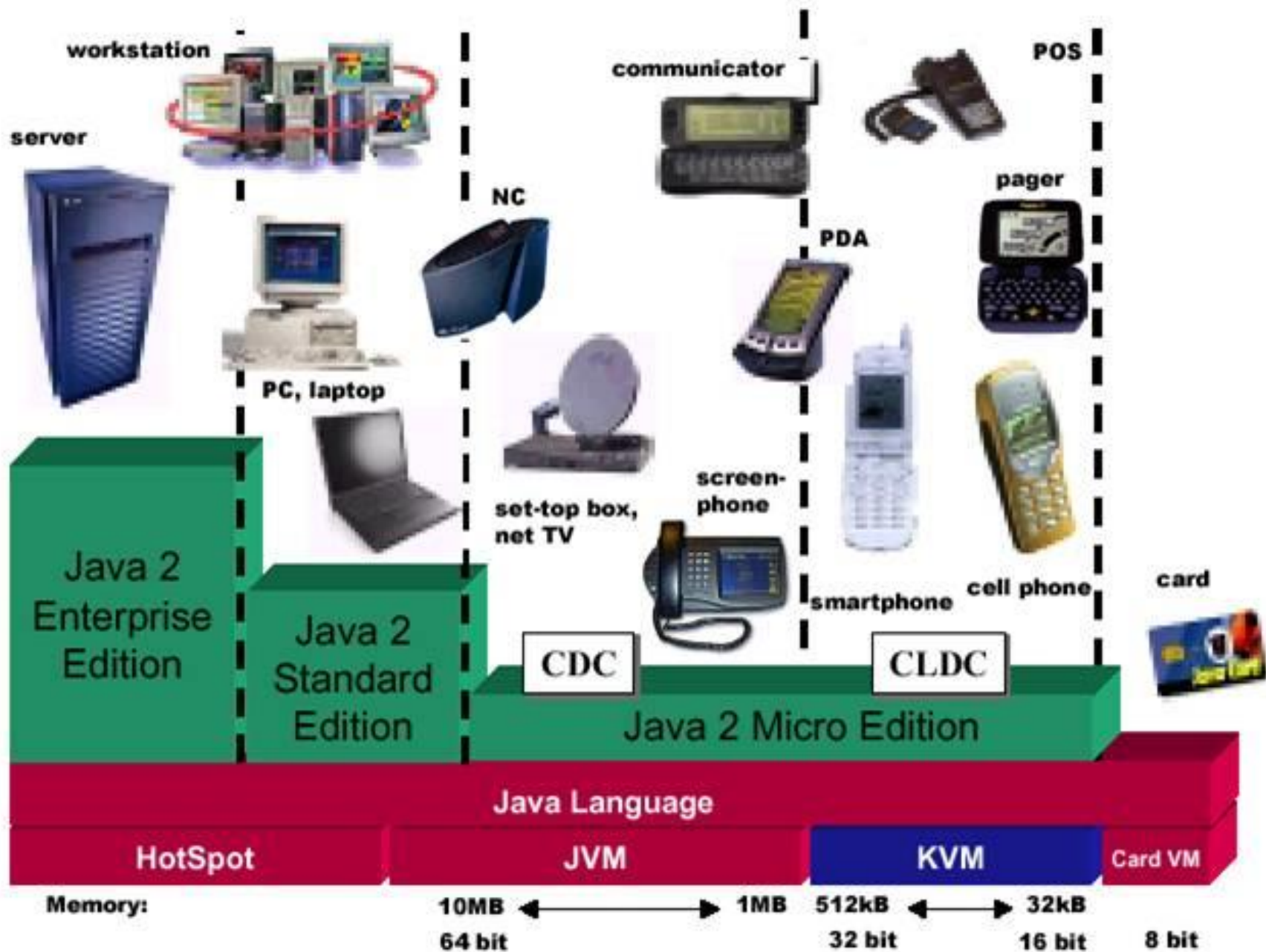
❖Multi-threaded

❖Dynamic

- A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.

- The main advantage of multi-threading is that it doesn't occupy memory for each thread.

- It shares a common memory area. Threads are important for multi-media, Web applications, etc.

- Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand.

- It also supports functions from its native languages, i.e., C and C++.

- Java supports dynamic compilation and automatic memory management (garbage collection).

Java Editions

1. **Java SE (Standard Edition):** For general-purpose use on desktop PCs, servers and similar devices.

2. **Java ME (Micro Edition):** Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. It is often used to develop applications for mobile devices, TV set-top boxes, printers and PDAs.

3. **Java EE (Enterprise Edition):** development of web services, networking, server side scripting and other various web based applications.

- Java Card
- Java FX

workstation

server

NC

communicator

POS

pager

PDA

PC, laptop

set-top box, net TV

screen-phone

smartphone

cell phone

card

Java 2 Enterprise Edition

Java 2 Standard Edition

CDC

CLDC

Java 2 Micro Edition

Java Language

HotSpot	JVM	KVM	Card VM

Memory:	10MB ⟷ 1MB	512kB ⟷ 32kB	
	64 bit	32 bit 16 bit	8 bit

J2SE(Java Platform, Standard Edition)

- Also known as Core Java, this is the most basic and standard version of Java. It's the purest form of Java, a basic foundation for all other editions.

- It consists of a wide variety of general purpose API's (like java.lang, java.util) as well as many special purpose APIs.

- J2SE is mainly used to create applications for Desktop environment.

- It consist all the basics of Java the language, variables, primitive data types, Arrays, Streams, Strings Java Database Connectivity(JDBC) and much more. This is the standard, from which all other editions came out, according to the needs of the time.

- The famous JVM of Java, the heart of Java development, was also given by this edition only. It's because of this feature, that Java has such a wide usage.

J2ME(Java Platform, Micro Edition)

- This version of Java is mainly concentrated for the applications running on embedded systems, mobiles and small devices.(which was a constraint before it's development).

- Constraints included limited processing power, battery limitation, small display etc.

- Also, the J2ME apps help in using web compression technologies, which in turn, reduce network usage, and hence cheap internet accessibility.

- J2ME uses many libraries and API's of J2SE, as well as, many of it's own.

- The basic aim of this edition was to work on mobiles, wireless devices, set top boxes etc.

- Old Nokia phones, which used Symbian OS, used this technology.

- Most of the apps, developed for the phones (prior to smartphones era), were built on J2ME platform only(the .jar apps on Nokia app store).

J2EE(Java Platform, Enterprise Edition)

- The Enterprise version of Java has a much larger usage of Java, like development of web services, networking, server side scripting and other various web based applications.

- J2EE is a community driven edition, i.e. there is a lot of continuous contributions from industry experts, Java developers and other open source organizations.

.

- J2EE uses many components of J2SE, as well as, has many new features of it's own like Servlets, JavaBeans, Java Message Services, adding a whole new functionalities to the language.

- J2EE uses HTML, CSS, JavaScript etc., so as to create web pages and web services. It's also one of the most widely accepted web development standard.

•Apart from these three versions, there was another Java version, released Java Card. This edition was targeted, to run applets smoothly and securely on smart cards and similar technology. Portability and security was its main features.

•JavaFX is another such edition of Java technology, which is now merged with J2SE 8.It is mainly used, to create rich GUI (Graphical User Interface) in Java apps.
•It replaces Swings (in J2SE), with itself as the standard GUI library.
•It is supported by both Desktop environment as well as web browsers.
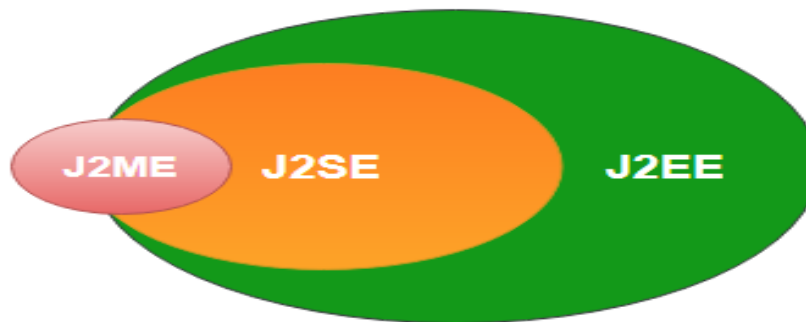
JVM

- JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist.

- It is a specification that provides a runtime environment in which Java bytecode can be executed.

- It can also run those programs which are written in other languages and compiled to Java bytecode.

- JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other.

However, Java is platform independent. There are three notions of the JVM:

          specification,
          implementation
          instance

The JVM performs the following main tasks:
- Loads code
- Verifies code
- Executes code
- Provides runtime environment

Java Source Code → Java Compiler

Java Interpreter (JVM) ← Java Bytecode (will be loaded into JVM)

Execution
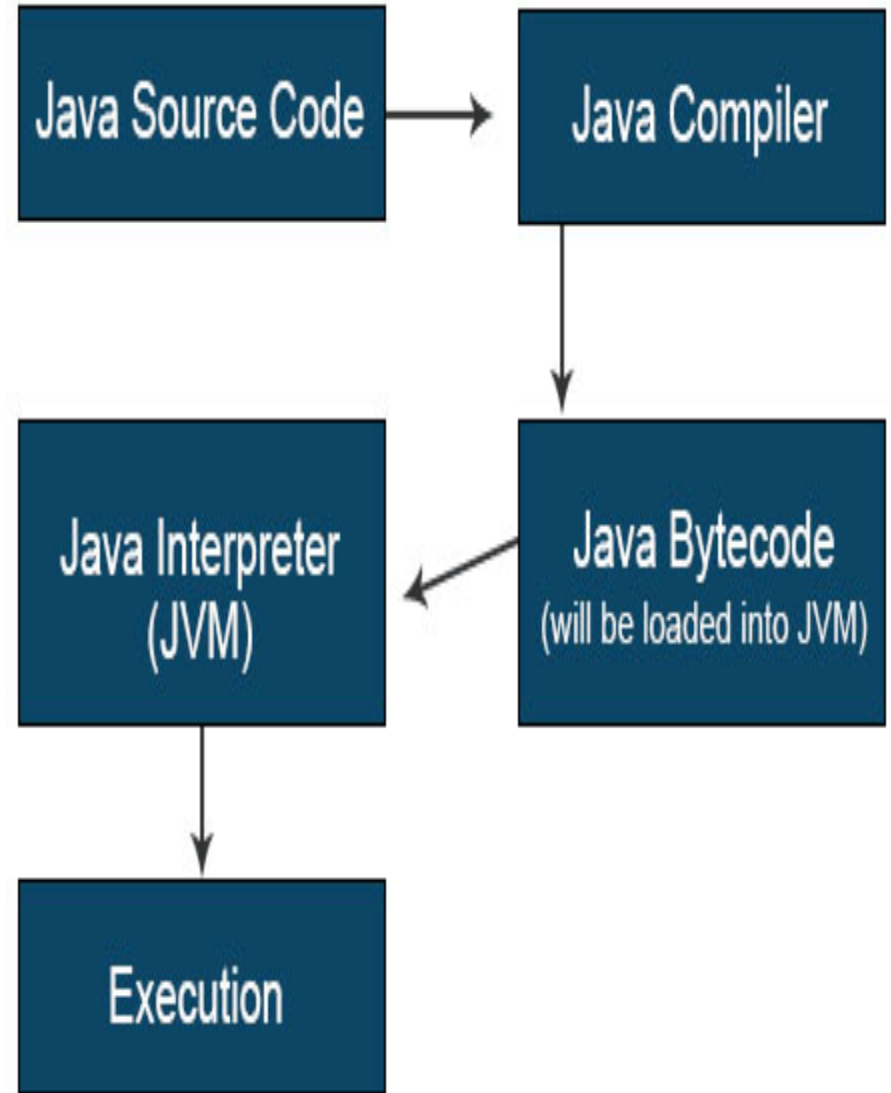
JRE

- JRE is an acronym for Java Runtime Environment. It is also written as Java RTE.

- The Java Runtime Environment is a set of software tools which are used for developing Java applications.

- It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

- The implementation of JVM is also actively released by other companies besides SunMicroSystems.

JRE

Class Loader
&
Bylecode Verification

Java Class Libraries

JVM

Java Interpreter

Threading

Garbage Collection

Etc...
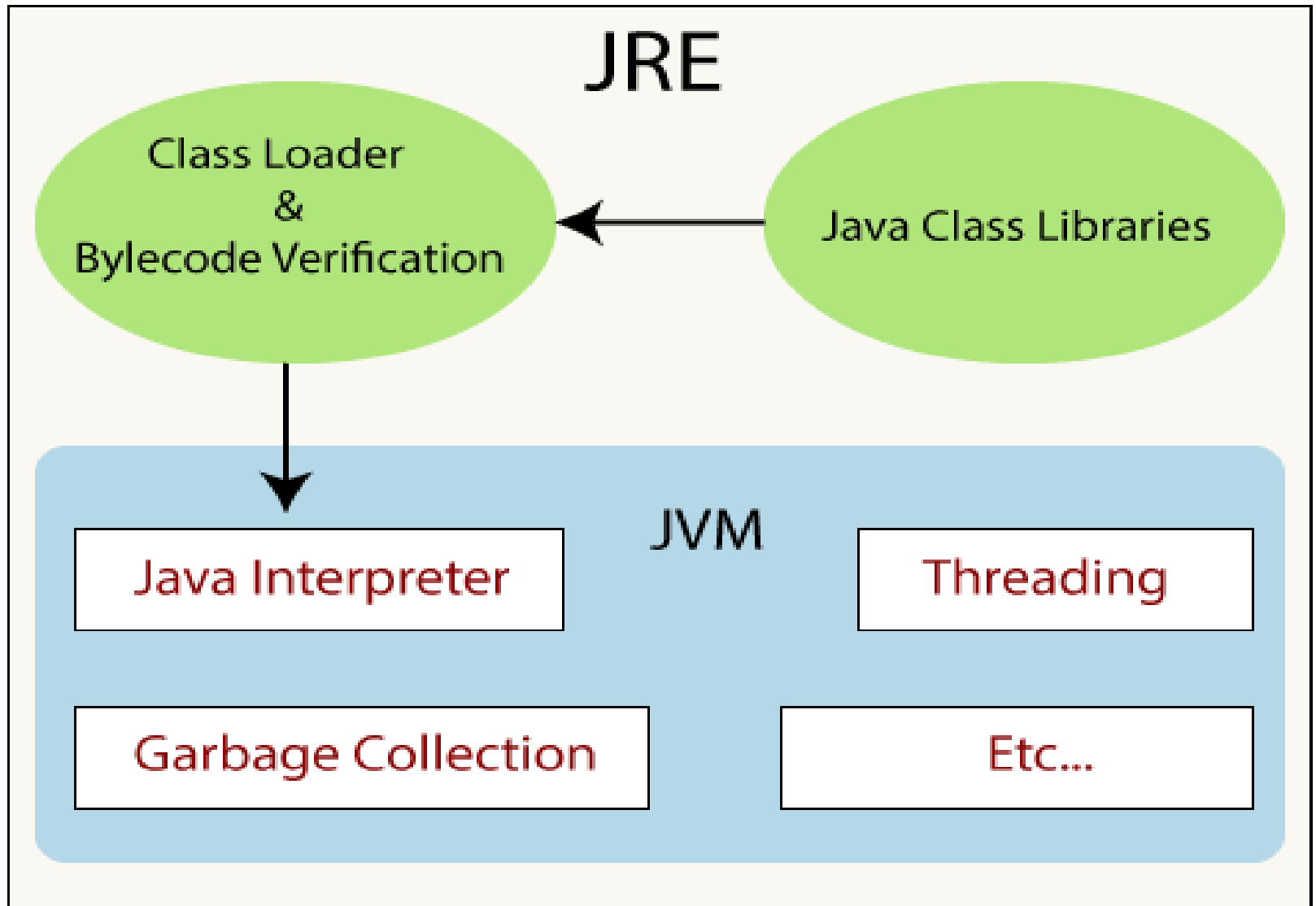
JDK

- JDK is an acronym for Java Development Kit.

- The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

- JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform(J2SE)

- Enterprise Edition Java Platform(J2EE)

- Micro Edition Java Platform(J2ME)

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

Set of libraries e.g. rt.jar etc.

JVM

Other files

Development tools e.g. javac, java etc.

JRE

JDK

JDK Tools

appletviewer	extcheck	idlj	jabswitch	jar	jarsigner	java	javac
javadoc	javafxpackager	javah	javap	java-rmi	javaw	javaws	jcmd
jconsole	jdb	jdeps	jhat	jinfo	jjs	jli.dll	jmap
jmc	jmc	jps	jrunscript	jsadebugd	jstack	jstat	jstatd
jvisualvm	keytool	kinit	klist	ktab	msvcr100.dll	native2ascii	orbd
pack200	policytool	rmic	rmid	rmiregistry	schemagen	serialver	servertool
tnameserv	unpack200	wsgen	wsimport	xjc			

AppletViewer

The appletviewer tool allows you to run applets without a web browser. It is compatible with the appletviewer tool that is supplied by Sun Microsystems, Inc. The appletviewer tool is available using the Qshell Interpreter.

Extcheck

The extcheck command checks a specified JAR file for title and version conflicts with any extensions installed in the Java SE SDK.

Javah

The javah command generates C header and source files that are needed to implement native methods. The generated header and source files are used by C programs to reference an object's instance variables from native source code.

Java

The loader for Java applications. This tool is an interpreter and can interpret the class files generated by the javac compiler.

Javac

The javac tool reads class and interface definitions, written in the Java programming language, and compiles them into bytecode class files.

javadoc

Javadoc is a tool for generating API documentation in HTML format from doc comments in source code.

jar

The jar tool combines multiple files into a single JAR archive file. jar is a general-purpose archiving and compression tool, based on ZIP and the ZLIB compression format. However, jar was designed mainly to facilitate the packaging of java applets or applications into a single archive.

idlj

The idlj compiler generates Java bindings from an IDL file. This compiler supports the CORBA Objects By Value feature.

javap

The javap tool disassembles compiled Java files and prints out a representation of the Java program. This may be helpful when the original source code is no longer available on a system.

JavaFX

JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications.

Compiling and Executing basic java program

// Java Program to Illustrate Compilation and Execution Stages

Filename: HelloWorld.java

```java
// Main class
    public class HelloWorld
     {

        // Main driver method
            public static void main(String[] args)
            {

                // Print command
                System.out.print("Hello World");
            }
     }
```

Output:

Hello World

Steps:

1. Create the program by typing it into a text editor and saving it to a file   named, say, HelloWorld.java.
2. Compile it by typing "javac HelloWorld.java" in the terminal window.
3. Execute (or run) it by typing "java HelloWorld " in the terminal window.

```java
public class HelloWorld
{

        // Main driver method
    public static void main(String[] args)
    {

        // Print command
        System.out.print("Hello World");
    }
}
```

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Admin>e:

E:\>java HelloWorld.java
Error: Could not find or load main class HelloWorld.java

E:\>javac HelloWorld.java

E:\>java HelloWorld
Hello World
E:\>
```
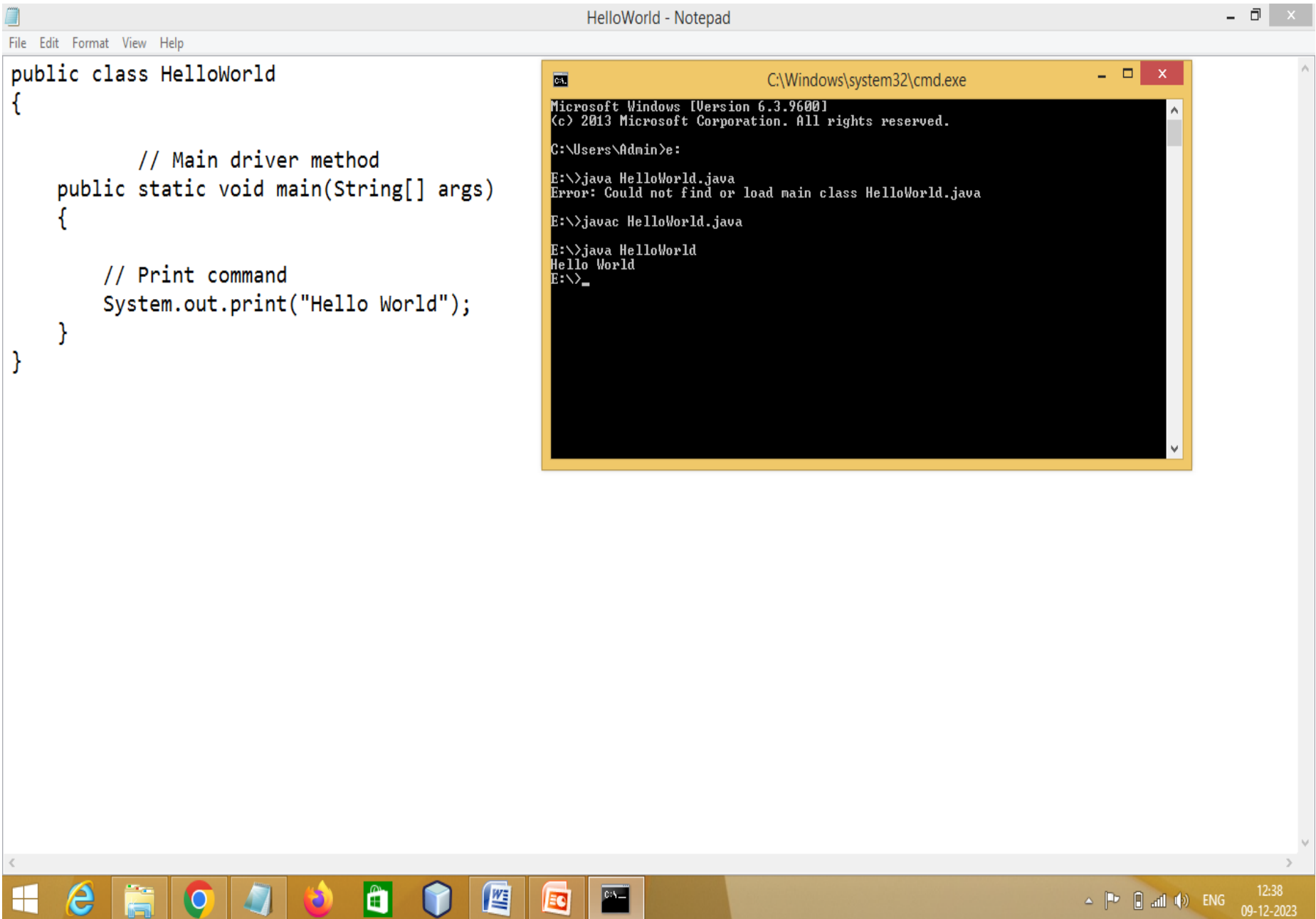
JAVA IDE

- Java editors, also known as Java IDEs (Java Integrated Development Environments), help streamline the Java programming process in many ways.

- They enable a Java developer to perform multiple tasks, such as writing, editing, and testing code, in one application – without the need to switch between separate applications.

- Popular Java IDEs for Java Development

❖ Apache NetBeans

❖ My Eclipse IDE

❖ IntelliJ IDEA

❖ Codenvy

❖ JDeveloper IDE

❖ DrJava

❖ jGRASP

Apache NetBeans

- Apache NetBeans is a free Java IDE that provides many features to help streamline the development process.

- Some of these features include Java code formatting, which lets developers change the layout of their source code to best suit their needs; code folding, which lets developers quickly collapse and expand blocks of code, and customizing keyboard shortcuts, which lets developers personalize their keyboard shortcuts and save their preferences in a dedicated user profile.

- Apache NetBeans also has smart code completion, which helps fill in missing code by automatically recommending possible code fragments.

My Eclipse IDE

•Another open-source Java IDE and one of the best IDEs used for Java EE, MyEclipse, is available for Windows, Linux, and Mac.

•The MyEclipse IDE is compatible with the Eclipse Marketplace, which means developers can search and download plugins for c/c++, javascript/typescript, php,java, HTML5,python from the Eclipse Marketplace and integrate them into the MyEclipse IDE.

•MyEclipse also features a dedicated plugin development environment, which allows developers to build new plugins from scratch and test them before integration. Another great feature of MyEclipse is the database and persistence support, with connectors for dozens of popular databases such as Spring, Maven, jQuery, and much more.

variable

- A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type.

- Variable is a name of memory location.
- There are three types of variable in java:
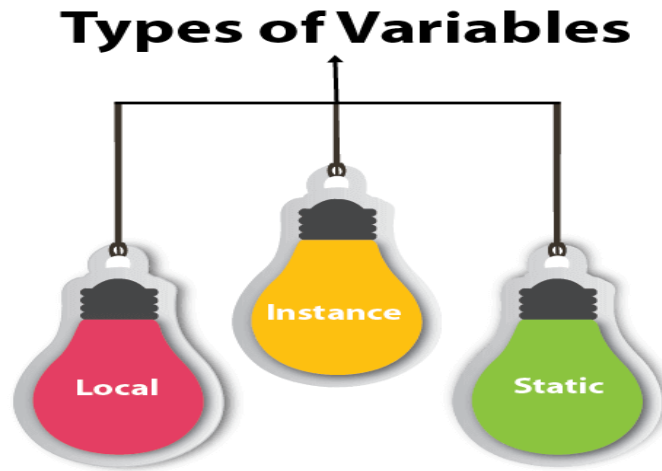
1) Local Variable

•Local variables are declared in methods, constructors, or blocks.

•Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor, or block.

•Access modifiers cannot be used for local variables.

•Local variables are visible only within the declared method, constructor, or block.

•There is no default value for local variables, so local variables should be declared and an initial value should be assigned before the first use.

•A local variable cannot be defined with "static" keyword.

Test.java

```java
public class Test
{
public void pupAge()          //pupage method
 {
     int age = 0;          //local variable
     age = age + 7;
     System.out.println("Puppy age is : " + age);
 }
 public static void main(String args[])
{
     Test test = new Test();
      test.pupAge();
 }
}
```

Output:
Puppy age is: 7

2) Instance Variable

•A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.Access modifiers can be given for instance variables.

•Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed

•The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.

•Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.

Employee.java

```java
public class Employee
{
    public String name;          // this instance variable is visible for any child class.
    private double salary;       // salary variable is visible in Employee class only.

Employee (String empName)    // The name variable is assigned in the
constructor.
{
        name = empName;
}
public void setSalary(double empSal)    // The salary variable is assigned a value.
{
        salary = empSal;
}
```

```java
public void printEmp()                    // This method prints the employee
details.
{
            System.out.println("name : " + name );
            System.out.println("salary :" + salary);
}
public static void main(String args[])
{
        Employee empOne = new  Employee("Ransika");
         empOne.setSalary(1000);
         empOne.printEmp();
 }
 }
```

Output:
name : Ransika
salary :1000.0

3) Static variable

• A variable that is declared as static is called a static variable. It cannot be local.

• You can create a single copy of the static variable and share it among all the instances of the class.

• Memory allocation for static variables happens only once when the class is loaded in the memory.

• Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.

• Static variables are created when the program starts and destroyed when the program stops

• Visibility is similar to instance variables. Default values are same as instance variables.

Employee.java

```java
public class Employee
{
    private static double salary;       // salary variable is a private static variable
    public static  String DEPARTMENT = "Development ";
                                        // DEPARTMENT is a constant

public static void main(String args[])
  {
       salary = 1000;
       System.out.println(DEPARTMENT + "average salary:" + salary);
    }
}
```

Output:
Development average salary:1000

Data Types

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

.

1.Primitive data types:

- In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.
- There are 8 types of primitive data types:

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

2. Non-Primitive data types:

- The Reference Data Types will contain a memory address of variable values because the reference types won't store the variable value directly in memory.

- They are strings, objects, arrays, etc.

Boolean Data Type

• The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

• The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

•**Example:** Boolean one = false ;

.

Int Data Type

• The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 to 2,147,483,647 (inclusive). Its default value is 0.

• The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

• **Example:** int a = 100000, int b = -200000 ;

Float Data Type

- The float data type is a single-precision 32-bit floating point. Its value range is unlimited. It is recommended to use a if you need to save memory in large arrays of floating point numbers.

- The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f ;

Char Data Type

- The char data type is a single 16-bit Unicode character with the size of 2 bytes (16 bit) . Its value-range lies between   0 to 65,535 .

- The char data type is used to store characters.

- **Example:** **char** letterA = **'A'** ;

Java Tokens

- In Java, the program contains classes and methods. Further, the methods contain the expressions and statements required to perform a specific operation.

- These statements and expressions are made up of tokens. In other words, we can say that the expression and statement is a set of tokens. The tokens are the small building blocks of a Java program that are meaningful to the java compiler. these two components contain variables, constants, and operators.

Types of Tokens

❖Keywords
❖Identifiers
❖Literals
❖Operators
❖Separators
❖Comments
❖Whitespace

Keywords:

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

01. abstract	02. Boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. thro	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

Literals

- In programming literal is a notation that represents a fixed value (constant) in the source code.

- It can be categorized as an integer literal, string literal, Boolean literal, etc. It is defined by the programmer. Once it has been defined cannot be changed. Java provides five types of literals are as follows:

- Integer ,Floating Point, Character, String, Boolean

Literal	Type
23	int
9.86	Float
false, true	boolean
'K', '7', '-'	Char
"javatpoint"	String
Null	any reference type

Separators:

- It help us defining the structure of a program. In Java, There are few characters used as separators. The most commonly used separator in java is a semicolon(;).

WhiteSpace:

- Java is a free-form language.
- This means that you do not need to follow any special indentation rule.

Symbol	Name	Purpose
( )	Parentheses	used to contain a list of parameters in method definition and invocation. also used for defining precedence in expressions in control statements and surrounding cast types
{ }	Braces	Used to define a block of code, for classes, methods and local scopes Used to contain the value of automatically initialised array
[ ]	Brackets	declares array types and also used when dereferencing array values
;	Semicolon	Terminates statements
,	Comma	Used to separates package names from sub-package and class names and also selects a field or method from an object
.	Period	separates consecutive identifiers in variable declarations also used to chains statements in the test, expression of a for loop
:	Colon	Used after labels

Identifier :

- identifiers in java are a sequence of characters to identify something in a program.

- Identifiers are used to name a variable, constant, function, class, and array.

- It usually defined by the user.

- Remember that the identifier name must be different from the reserved keywords.

- There are some rules to declare identifiers are:

❖ The first letter of an identifier must be a letter, underscore or a dollar sign. It cannot start with digits but may contain digits.

❖ The whitespace cannot be included in the identifier.

❖ Identifiers are case sensitive.

Some valid identifiers are:
- PhoneNumber ,PRICE , radius , a , a1 , phonenumber ,$circumference , jagged_array
- 12radius //invalid

Comments:

- Comments allow us to specify information about the program inside our Java code. Java compiler recognizes these comments as tokens but excludes it form further processing.

- The Java compiler treats comments as whitespaces.

- Java provides the following two types of comments:

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

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

Operators:

- <mark>Operator in Java is a symbol that is used to perform operations.</mark>

- There are many types of operators in Java which are given below:
- Unary Operator
- Arithmetic Operator
- Shift Operator
- Relational Operator
- Bitwise Operator
- Logical Operator
- Ternary Operator
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	Postfix	*expr*++ *expr*--
	Prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	Additive	+ -
Shift	Shift	<< >> >>>
Relational	Comparison	< > <= >= instanceof
	Equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	Ternary	? :
Assignment	Assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

•The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

•incrementing/decrementing a value by one

•negating an expression

•inverting the value of a Boolean

unary.java

```java
public class unary
{
public static void main(String args[])
{
    int x=10;
    System.out.println(x++); //10 (11)
    System.out.println(++x); //12
    System.out.println(x--); //12 (11)
    System.out.println(--x); //10
}
}
```

Java Arithmetic Operator

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Ari.java

```java
public class Ari
{
public static void main(String args[])
{
    int a=10;
    int b=5;
      System.out.println(a+b);//15
      System.out.println(a-b);//5
      System.out.println(a*b);//50
      System.out.println(a/b);//2
      System.out.println(a%b);//0
}
}
```

Java Ternary Operator

Java Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Ternary.java

```java
public class Ternary
{
public static void main(String args [])
{
        int  a=10;
        int b=5;
        int min=(a<b)?a:b;
        System.out.println(min);
}
}
```

Java Assignment Operator

Java assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Assign.java

```java
public class Assign
{
public static void main(String args [])
{
        int a=10;
        int b=20;
        a+=4;   //a=a+4 (a=10+4)
        b-=4;     //b=b-4 (b=20-4)
        System.out.println(a);
        System.out.println(b);
}
}
```

Logical && and Bitwise &

•The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

•The bitwise & operator always checks both conditions whether first condition is true or false.

```java
public class Andop
{
public static void main(String args[])
{
	int a=10;
	int b=5;
	int c=20;
	System.out.println(a<b&&a++<c);
	//false && true = false
	System.out.println(a);/
	/10 because second condition is not
		checked
	System.out.println(a<b&a++<c);
	//false && true = false
	System.out.println(a);
	//11 because second condition is che
		cked
}
}
```

Logical || and Bitwise |

BitOr.java

•The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

•The bitwise | operator always checks both conditions whether first condition is true or false.

```java
public class BitOr
{
public static void main(String args[])
{
    int a=10;
    int b=5;
    int c=20;
    System.out.println(a>b||a++<c);
    //true || true = true
    System.out.println(a);//10 because seco
        nd condition is not checked
    System.out.println(a>b|a++<c);
    //true | true = true
    System.out.println(a);
    //11 because second condition is checke
        d
    }
}
```

JAVA Assignment Operator

'=' Assignment operator is used to assign a value to any variable. It has right-to-left associativity.

GFG.java

```java
class GFG
{

    public static void main(String[] args)
    {

        int f = 7;
        System.out.println("f += 3: " + (f += 3));
        System.out.println("f -= 2: " + (f -= 2));
        System.out.println("f *= 4: " + (f *= 4));
        System.out.println("f /= 3: " + (f /= 3));
        System.out.println("f %= 2: " + (f %= 2));
    }
}
```

JAVA Relational Operators

•These operators are used to check for relations like equality, greater than, and less than.

•They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The general format is,

variable **relation_operator** value

CFG.java

```java
class GFG
{

    public static void main(String[] args)
    {

        int a = 10;
        int b = 3;
        int c = 5;

        System.out.println("a > b: " + (a > b));
        System.out.println("a < b: " + (a < b));
        System.out.println("a >= b: " + (a >= b));
        System.out.println("a <= b: " + (a <= b));
        System.out.println("a == c: " + (a == c));
        System.out.println("a != c: " + (a != c));
    }
}
```

JAVA Shift Operators

•These operators are used to shift the bits of a number left or right, thereby multiplying or dividing the number by two, respectively.

•They can be used when we have to multiply or divide a number by two. General format-

 number **shift_op** number_of_places_to_shift;

<<, Left shift operator: shifts the bits of the number to the left and fills 0 on voids left as a result.

>>, Right shift operator: shifts the bits of the number to the right and fills 0 on voids left as a result.

Shif.java

```java
Class shif
{
  public static void main(String[] args)
  {
     int a = 10;
   // using left shift
     System.out.println("a<<1 : " + (a << 1));


    // using right shift
     System.out.println("a>>1 : " + (a >> 1));
  }
}
```

Special Operators:

Java also supports some special type of operators like:

- Dot Operator

- InstanceOf Operator

Dot Operator (.)

- Dot operator helps to access:
    - variable/constant
    - method
    - Class
    - package
    - For Example: java.lang.System.out.println("Hello");

The instanceof Operator:

- This operator checks the existence of an object and returns a boolean value.
- It is also called a type/object comparison operator.
  - it compares if the object belongs to the specified type.

- Applying the instanceof operator with any object having a null value, returns false.

- It can also identify objects of derived and inherited classes.

```java
public class CheckInstance
{
public static void main(String[] args)
{
        CheckInstance ob1 = new CheckInstance();

        CheckInstance ob2= null;

        System.out.println("ob 1 instanceof  CheckInstance :"+(ob1 instanceof
        CheckInstance));

        System.out.println("ob2 instanceof  CheckInstance"+(ob2 instanceof
        CheckInstance));
}
}
```

Output:
ob 1 instanceof CheckInstance :true
ob 2 instanceof CheckInstance :false

Assertion:

- Assertion is a statement in java. It can be used to test your assumptions about the program.

- While executing assertion, it is believed to be true. If it fails, JVM will throw an error named AssertionError. It is mainly used for testing purpose.

- Advantage of Assertion:
- It provides an effective way to detect and correct programming errors.

- Syntax of using Assertion:
- There are two ways to use assertion.
1. assert expression;
2. assert expression1 : expression2;

```java
import java.util.Scanner;

class ass
{
 public static void main( String args[] )
    {
        Scanner scanner = new Scanner( System.in );
        System.out.print ("Enter ur age ");

        int value = scanner.nextInt();
        assert value>=18:" Not valid";
        //assert value ;

        System.out.println("value is "+value);
        }
}
```
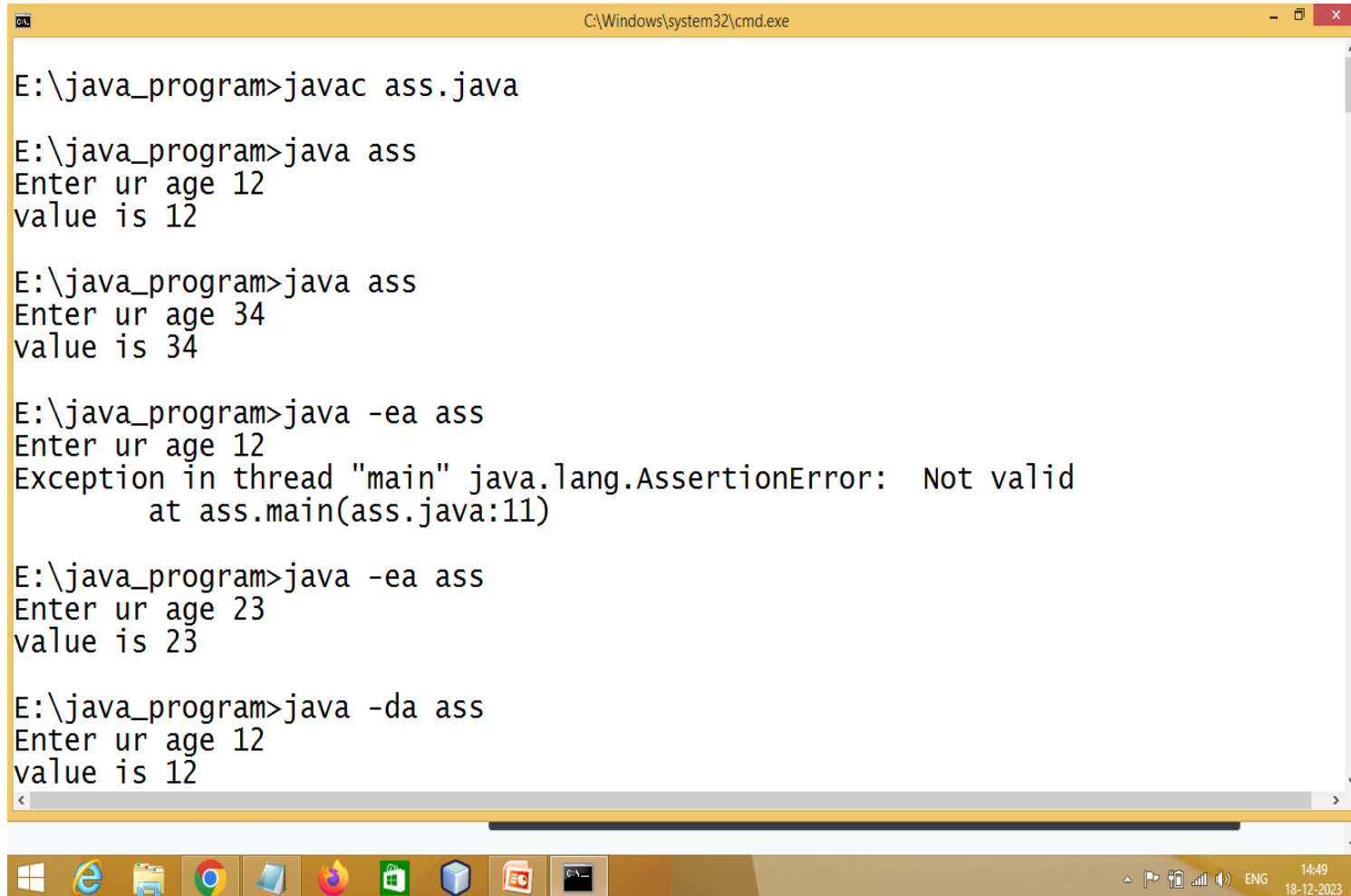
If you use assertion, It will not run simply because assertion is disabled by default. To enable the assertion, -ea or
-enableassertions switch of java must be used.

Compile it by: javac AssertionExample.java

Run it by: java -ea AssertionExample

strictfp

- It is a strict floating point that was introduced in jdk 1.2 version by following IEEE 745 standards for floating poing calculation.

- Strictfp keyword is a java modifier, that helps programmer to make floating point calculations platform independent.

- Note: the value of floating point calculation may be vary with different os.

- Because the java is plattform independent, thats why we should make our floating point value to platform independent with the help of strictfp.

```
public class Sfc
{
public static strictfp void main(String[]  args)
  {
      System.out.println(10.0/3);
  }
}
```

Output: 3.33333333335

Strictfp used for:
Class, concrete method, interface, abstract class

Strictfp not applied for:
Variable, constructor, abstract method

enum

- The enum keyword declares an enumerated (unchangeable) type.

- An enum is a special "class" that represents a group of constants ,method and private constructor (if we don't declare private,compiler internally creates private constructor).

- Enum constants are by default public, static and final that means it can not changeble (not override) .

- it can not extend other classes but it can implement interfaces.

- Because of private constructor we can't create the instance of enum using new keyword.

- Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

- 3- function: ValueOf(), values(), ordinal()

```java
class abc
{
enum Month
{
    JANUARY(1),
    FEBUARY(2),
    MARCH(3);
   int value;
   private Month(int v)
{
   value=v;
 }
}
  public static void main(String args[])
   {
    for(Month m: Month.values())
    System.out.println(m+ "  " +m.value);
    }
}
```

Java Type Casting

- Type casting is when you assign a value of one primitive data type to another type.

- The automatic conversion is done by the compiler and manual conversion performed by the programmer.

- There are two types of type casting:

I.   Widening Type Casting(Automatic)

II.   Narrowing Type Casting(Manual)

Narrowing Type Casting

| double | → | float | → | long | → | int | → | short | → | byte |

Widening Type Casting

Type Casting in Java
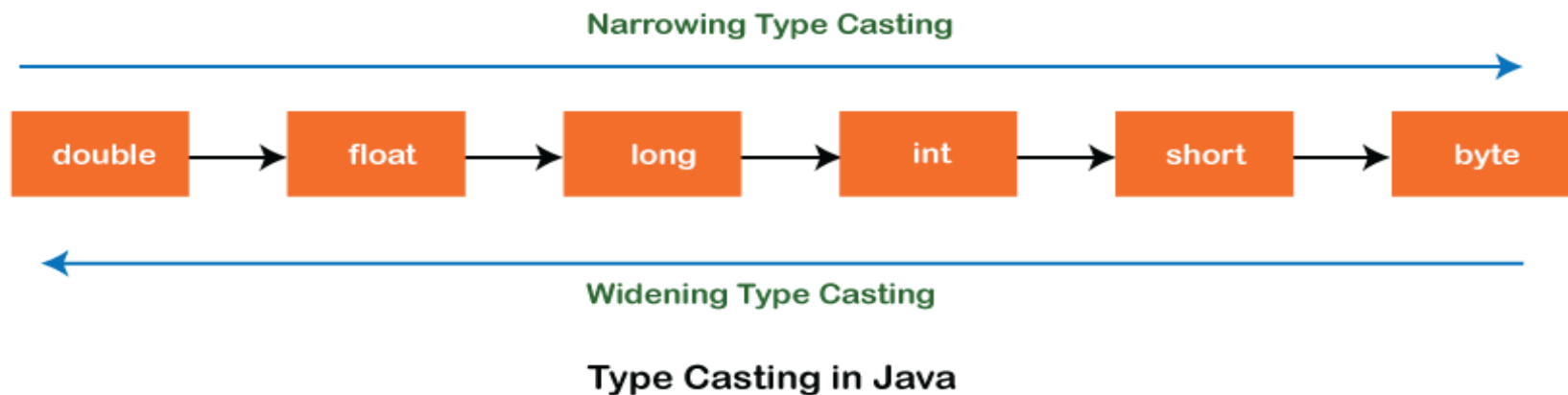
Widening Type Casting

- Converting a lower data type into a higher one is called **widening** type casting.

- It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:

- Both data types must be compatible with each other.

- The target type must be larger than the source type.

    byte -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

```java
public class Widening
{
        public static void main(String[] args) {
         int x = 9;
         long y = x;        // Automatic casting: int to long
         double z = y;     //// Automatic casting: long to double

         System.out.println(x);      // Outputs 9
         System.out.println(y);      // Outputs 9
         System.out.println(z);      // Outputs 9.0
  }
}
```

Output:
9
9
9.0

Narrowing Type Casting

- Converting a higher data type into a lower one is called **narrowing** type casting.

- It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer.

- If we do not perform casting then the compiler reports a compile-time error.

double -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

```java
public class Narrowing
{
    public static void main(String[] args) {
    double x = 89.78d;
     float y=(float)x;       // Manual casting: double to float
    int  z=(int)y;           // Manual casting: float to int
    char a=(char)z;
    short s=(short)a;
     byte b=(byte)a;         //-128 to 127 value

    System.out.println(x);
    System.out.println(y);
    System.out.println(z);
    System.out.println(a);
    System.out.println(s);
     System.out.println(b);
  }
}
```

Output: 89.78, 89.78, 89, Y, 89, 89

Decision statement:

- decision-making statements decide which statement to execute and when.

- Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java,

1.   If statement

   I.   Simple if statement

   II.   if-else statement

   III.   if-else-if ladder

   IV.   Nested if-statement

2.   switch statement

1) Simple if

- if statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not , if a certain condition is true then a block of statement is executed otherwise not.

- Syntax:

```
if(condition)
{
// Statements to execute if , condition is true
}
```

```java
import java.util.*;
 class IfDemo
 {
    public static void main(String args[])
    {

    Scanner sc=new Scanner(System.in);
    int a= sc.nextInt();

    if (a <= 15)

        System.out.println("Inside If block");
        System.out.println("i is less than 15");  //always executes as it
is outside of if block
        System.out.println("I am Not in if");
      // as if considers one statement by default again below statement
is    outside of if block
}
}
```

2) if-else:

- The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false? Here comes the else statement.

- We can use the else statement with the if statement to execute a block of code when the condition is false.

- Syntax:

```
if (condition)
{
 // Executes this block if ,condition is true
 }
 else
 {
// Executes this block if /,condition is false
 }
```

```java
import java.util.*;

class IfElseDemo
 {
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
         int i= sc.nextInt();

        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

3) nested-if

- A nested if is an if statement that is the target of another if or else. Nested if statements mean an if statement inside an if statement.

- Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

- Syntax:

```
if (condition1)
{
    // Executes when condition1 is true
 if (condition2)
{
        // Executes when condition2 is true
 }
 }
```

```
class Main
{
public static void main(String args[])
{      int i = 10;
       if (i == 10 || i<15)
  {

      if (i < 15)                        // First if statement
       System.out.println("i is smaller than 15");
       if (i < 9)                        // Nested - if statement
                                         // Will only be executed if statement above is true
          System.out.println(   "i is smaller than 12 too");
  }
else
    {
    System.out.println("i is greater than 15");
    }
   }
}
```

if-else-if ladder

- a user can decide among multiple options.
- The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that 'if' is executed, and the rest of the ladder is bypassed.
- If none of the conditions is true, then the final else statement will be executed. There can be as many as 'else if' blocks associated with one 'if' block but only one 'else' block is allowed with one 'if' block.
  - Syntax:

  if (condition)

      statement;

  else if (condition)

       statement;

      .

      .

  else

     statement;

```java
import java.util.*;
class ladderif
{
    public static void main(String args[])
{
  Scanner sc=new Scanner(System.in);
  System.out.println("enter value");
   int i = sc.nextInt();

      if (i == 10)
            System.out.println("i is 10");

      else if (i == 15)
            System.out.println("i is 15");

      else if (i == 20)
            System.out.println("i is 20");

      else
            System.out.println("i is not present");
}
}
```

switch-case

•The switch statement is a multi way branch statement.
•It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

```
Syntax:
  switch (expression)
  {
     case value1: statement1;
     break;
     case value2: statement2;
     break;

      .

      .

      case valueN: statementN;
       break;
      default: statementDefault;
  }
```

```java
import java.util.*;
 class swi {
    public static void main (String[] args)
 {

    Scanner sc=new Scanner(System.in);
     System.out.println("enter value");
     int num = sc.nextInt();
    switch(num)
    {
        case 5 :  System.out.println("It is 5");
                  break;
        case 10 : System.out.println("It is 10");
                  break;
        case 15 : System.out.println("It is 15");
                  break;
        case 20 : System.out.println("It is 20");
                  break;
        default:  System.out.println("Not present");

    }
  }
}
```

Loops in Java

For loop:

* for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

```
class GFG {
    public static void main (String[] args)
    {
        for (int i=0;i<=10;i++)
        {
            System.out.println(i);
        }
    } }
```

While loop:

•   A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.

```
class GFG
 {
   public static void main (String[] args)
 {
     int i=0;
     while (i<=10)
     {
      System.out.println(i);
      i++;
     }
   }
 }
```

do..while :

• do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of **Exit Control Loop.**

```
class GFG {
    public static void main (String[] args)
  {
    int i=0;
    do
    {
     System.out.println(i);
     i++;
    }
    while(i<=10);
  }
}
```

Jump Statements in Java

- Jump statements are used to jump the control to another part of our program depending on the conditions. Jump statements are useful to break the iterations loops when certain conditions met and jump to outside the loop.

- These statements can be used to jump directly to other statements, skip a specific statement and so on.

- Following are the jump statements in java.
- break
- continue
- return

break statement

- If break statement is encountered then execution of its immediate surrounding block will be skipped control will go to next statement immediately after that block.
- Syntax: break;

```
public class breakstat
{
    public static void main(String[] args)
    {
        for (int i = 1; i <= 5; i++)
        {
            if (i == 3 )
            {
                break;
            }
            System.out.println("The number is " + i);
        }}}
```

continue statement

- If continue statement is encountered then execution of remaining statements in its immediate surrounding block will be skipped.
- Unlike break, it skips the remaining statements not the complete block, so if the block is iterative block then control goes to the beginning of the loop.
- Syntax: continue;

```
public class contstat
{    public static void main(String[] args)
    {
        for (int i = 1; i <= 5; i++)
        {
            if (i == 3 )
            {
                continue;
            }
            System.out.println("The number is " + i);
        }}}
```

return

This is used to return value from a method. Once method returns a value, further statements after return statements will be skipped.

```java
public class Main
 {
    static void myMethod()
 {
     System.out.println("I just got executed!");
 }
    public static void main(String[] args)
 {
     myMethod();
 }  }
```

Array

- Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
- To declare an array, define the variable type with square brackets:
- Syntax: String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
- To find out how many elements an array has, use the length property:
- Ex:-   String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
                    System.out.println(cars.length);
- There are two types of array :
- One-dimensional array
- Multi-dimensional array

One-dimensional array

- The standard method includes declaring a variable and initializing it using new or with a specialized set of values using array initializer.

- data_type[] var_name = new data_type[];
- data_type var_name[] = {};
                    // curly braces encloses specialized set of values

Multi-Dimensional Array

- A multidimensional array is an array of arrays.

- Multidimensional arrays are useful when you want to store data as a tabular form, like a table with rows and columns.

- To create a two-dimensional array, add each array within its own set of curly braces.

- Example:

```
public class Main {
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    for (int i = 0; i < myNumbers.length;  ++i)    {
      for(int j = 0; j < myNumbers[i].length;  ++j)   {
        System.out.println(myNumbers[i][j]);
      }}
  }}
```

Jagged Array

- A jagged array in Java is a collection of arrays where each array may contain a varied number of elements. A two-dimensional array, in contrast, requires all rows and columns to have the same length.

- Jagged arrays are also known as "ragged arrays" or "irregular arrays". They can be created by specifying the size of each array in the declaration. For example, a jagged array with three rows can have the first row with three elements, the second with two elements, and the third with four elements.

- Syntax:

```
datatype[][] arrayName = new datatype[numRows][];
arrayName[0] = new datatype[numColumns1];
arrayName[1] = new datatype[numColumns2];
...
arrayName[numRows-1] = new datatype[numColumnsN];
```

Advantages of Jagged Array

- **Memory Efficiency:** they only allocate memory for the elements they need. In a multidimensional array with a fixed size, all the memory is allocated, even if some elements are unused.

- **Flexibility:** they can have different sizes for each row. Jagged arrays enable the representation of non-rectangular or irregular data structures.

- **Easy to Initialize:** Jagged arrays are easy to initialize because you can specify the size of each row individually.

- **Enhanced Performance:** Jagged arrays can provide enhanced performance in certain scenarios, such as when you need to perform operations on each row independently or when you need to add or remove rows dynamically.

- **More natural representation of data:** In some cases, jagged arrays may be a more natural representation of data than rectangular arrays. For example, when working with irregularly shaped data such as geographic maps, a jagged array can be a more intuitive way to represent the data.

- **Easier to manipulate:** Jagged arrays can be easier to manipulate than rectangular arrays because they allow for more direct access to individual elements. With rectangular arrays, you may need to use more complex indexing or slicing operations to access subsets of the data.

Java Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.

- The arguments passed from the console can be received in the java program and it can be used as an input.

- So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

  Example:

  class CommandLineExample{

  public static void main(String args[]){

  System.out.println("Your first argument : "+args[0]);

  } }

- compile by > javac CommandLineExample.java

- run by > java CommandLineExample hello