JAVA PROGRAMMING (314317)

By

Sangita B. Chavan

Lecturer in Comp. Engg

Govt.Polytechnic,Awasari

Unit 1:Basic Syntactical construts injava

Contents:

- Java Features and the Java Programming Environment.
- Java Tokens & Data types
- Operators & Expressions
- Defining a class
- Decision making & looping

Features of Java applications

Sun Micro Systems officially describes Java with the following features

- Compiled and Interpreted
- Platform Independent and Portable
- Object Oriented
- Robust and Secure
- Distributed

Contd..

Features of Java applications

- Simple, Small and Familiar
- Multithreaded and Interactive
- High Performance
- Dynamic and Extensible

Compiled and Interpreted

- Usually a computer language is either compiled or interpreted
- Java combines both these approaches, thus making Java a two stage system
- First Java compiler translates source code into byte code instructions
- Byte codes are not machine instructions

Contd..

Compiled and Interpreted

- In the second stage Java interpreter generates machine code

- That can be directly executed by the machine

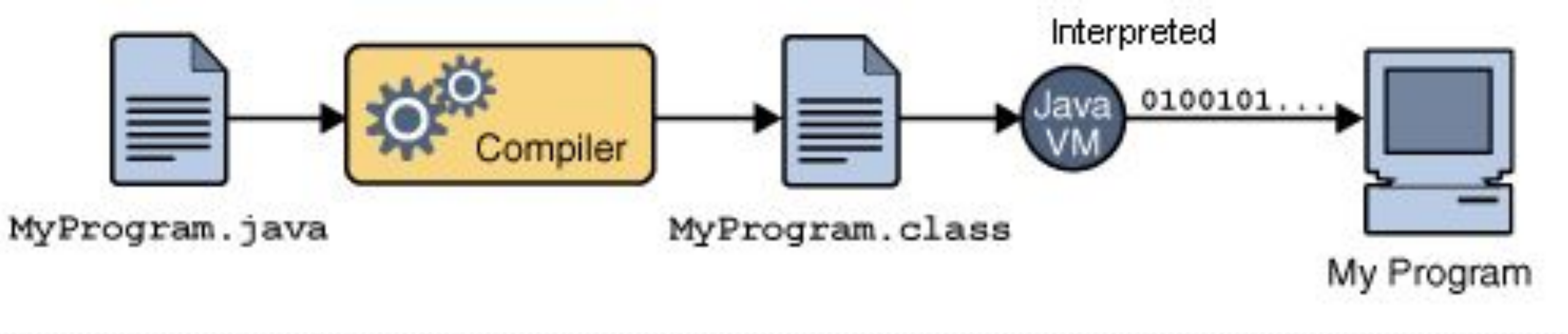


Fig. 2.1 Compiled and Interpreted

Platform Independent and Portable

- The most significant feature of Java is its portability

- Java programs can be easily moved from one computer system to another ,anywhere and at anytime

Platform Independent and Portable

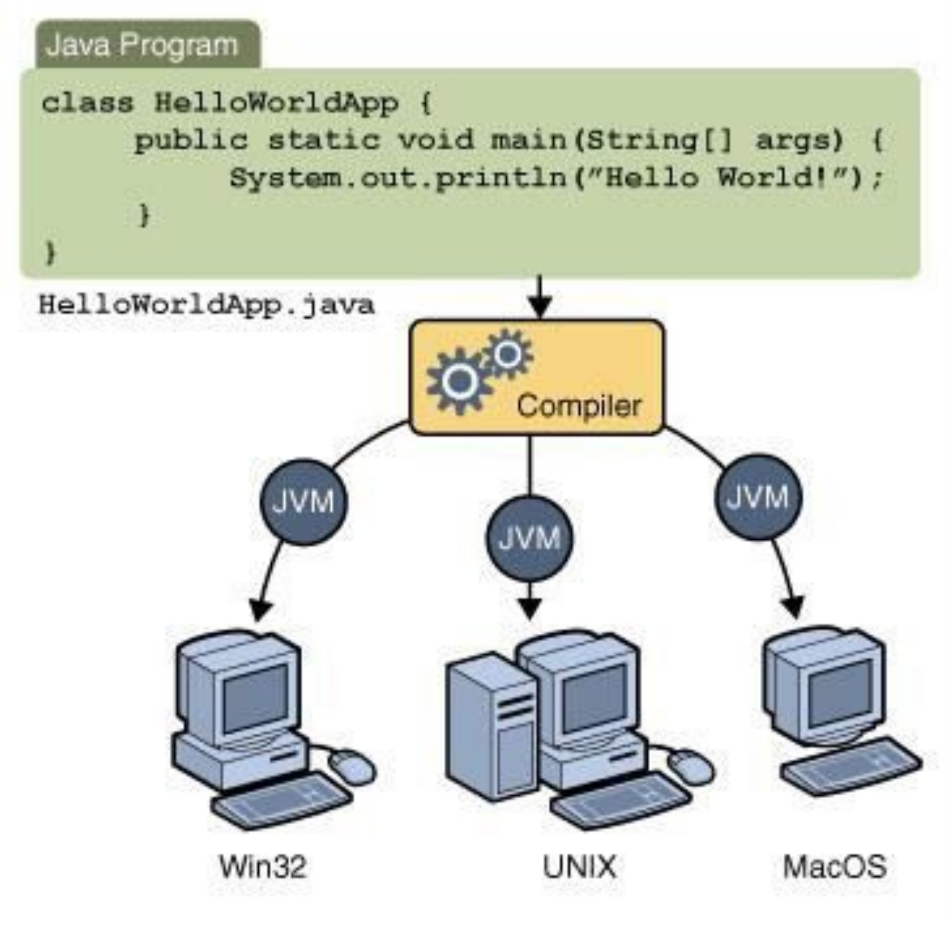
Contd..

- Changes and upgrades in
  - Operating systems
  - Processors
  - System resources

will not force any changes in Java programs

Platform Independent and Portable

Contd..



```
class HelloWorldApp {
     public static void main(String[] args) {
          System.out.println("Hello World!");
     }
}
```

Fig. 2.2 Once compiled, Java class file can be run on any platform

Object Oriented

- Java is a true object oriented language
- Almost everything in Java is an object
- All programs code and data reside within objects and classes.
- Similar to other OOP languges java also has basic OOP properties such as encapsulation,polymorphism,data abstraction ,inheritance etc.
- Java comes with an extensive set of classes arranged in packages

Robust and Secure

- Java is robust due to strong memory management, exception handling, and garbage collection.
- These features help in preventing crashes and memory leaks.
- Java provides strong security features such as no pointer usage, bytecode verification, and a security manager.
- It is widely used in banking and enterprise applications.

Distributed

- Java is designed as a distributed language for creating applications on networks.
- It has the ability to share both data and programs.
- Java supports distributed computing through technologies such as RMI (Remote Method Invocation) and web services, making it suitable for network-based applications.

Simple, Small and Familiar

- Java is small and simple language. Many features of C and C++ are not part of Java

  eg. : Java does not provide

  - Pointers
  - Preprocessors header file
  - goto statements
  - Operator overloading
  - Multiple inheritance

Contd..

Simple, Small and Familiar

- Familiarity is another feature of Java

- It is modeled on C and C++ languages

- Java is a simplified version of C++

Multithreaded and Interactive

- Multithreaded means handling multiple tasks simultaneously
- Java supports multithreaded programs
- Need not wait for the application to finish one task before beginning another
- Greatly improves the interactive performance of graphical applications

Dynamic and Extensible

- Java is a dynamic language
- Java is capable of dynamically linking new
  - Class libraries
  - Methods
  - Objects

Dynamic and Extensible

- Java is dynamic because it supports runtime class loading and dynamic memory allocation.
- New classes and libraries can be added without affecting existing programs.
- This facility enables the programmers to use the efficient available in these languages
- Native methods are dynamically linked at runtime

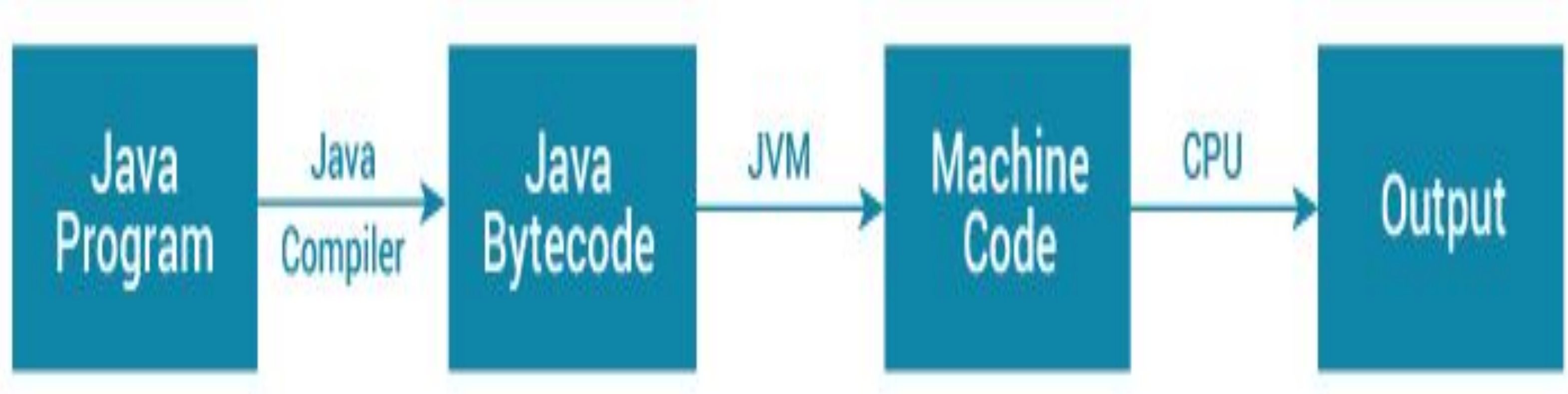
High Performance

- Java uses the Just-In-Time (JIT) compiler to convert bytecode into native machine code at runtime, which improves execution speed.

- **Conclusion**
- Java is a powerful programming language with features like simplicity, security, portability, and robustness. These features make Java one of the most popular languages in the software industry.

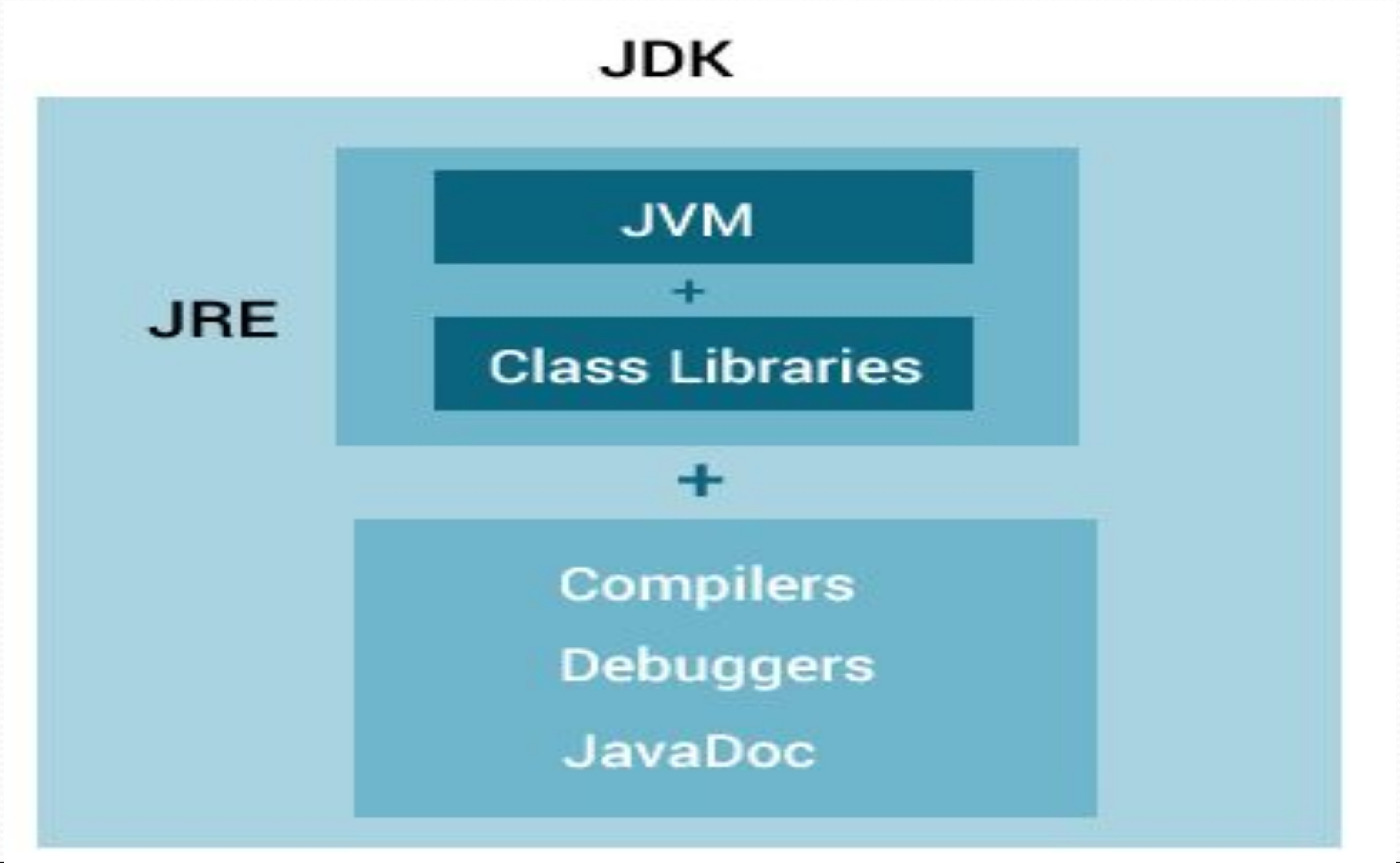
What is JVM?

- JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.
- When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).



- **What is JRE?**
- JRE (Java Runtime Environment) is a software package that provides Java class libraries, along with Java Virtual Machine (JVM), and other components to run applications written in Java programming. JRE is the superset of JVM.
- **What is JDK?**
- JDK (Java Development Kit) is a software development kit to develop applications in Java. When you download JDK, JRE is also downloaded, and don't need to download it separately. In addition to JRE, JDK also contains number of development tools (compilers, JavaDoc, Java Debugger etc).

Here's the relationship between JVM, JRE, and JDK.



JDK

JRE

JVM

+

Class Libraries

+

Compilers

Debuggers

JavaDoc

Simple java program

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

- **What's the difference between println(), print()**
- print() - prints string inside the quotes.

- println() - prints string inside the quotes similar like print() method. Then the cursor moves to the beginning of the next line.

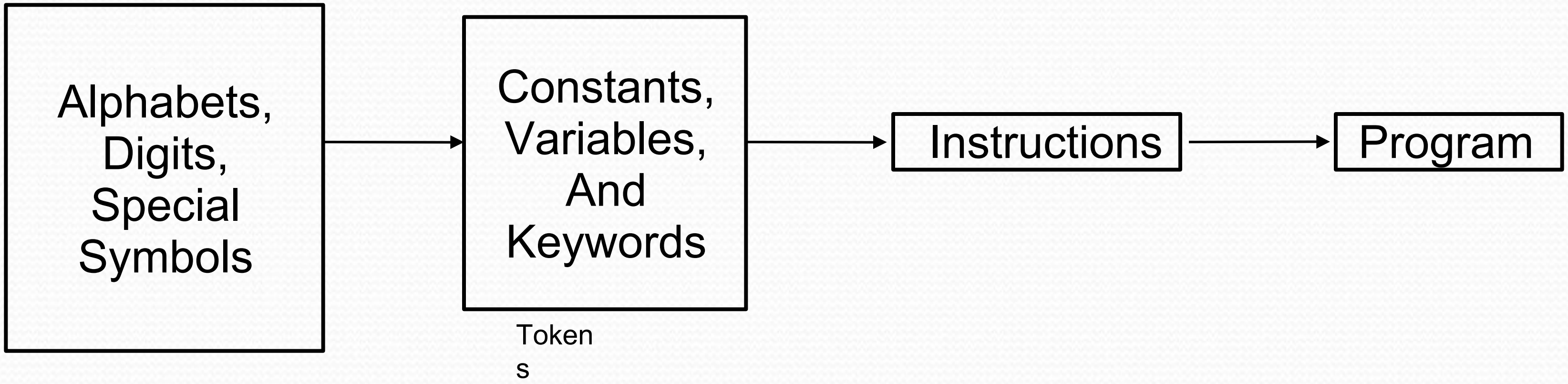
Java Tokens and Data Types

- Constants & Symbolic Constants,
- Variables
- Dynamic initialization Data types
- Array & string
- Scope of variable
- Type casting,
- Standard default values.

Java Tokens

- Identifiers
- Keywords
- Literals
- Operators
- Separators

Forming a Java Program



Alphabets, Digits, Special Symbols

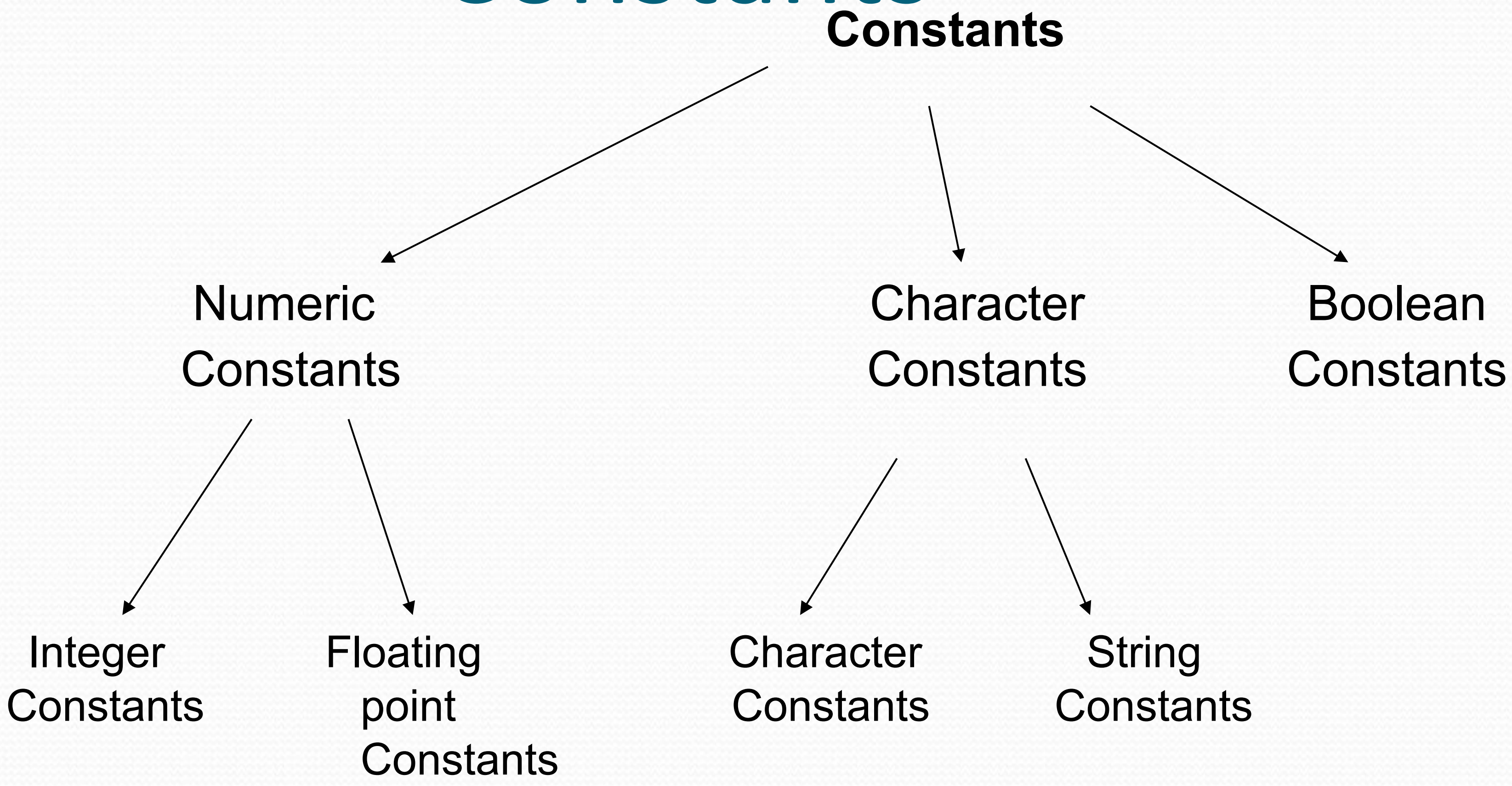
Constants, Variables, And Keywords

Token
s

Instructions

Program

Constants


Constants
Numeric Constants
Character Constants
Boolean Constants
Integer Constants
Floating point Constants
Character Constants
String Constants

Symbolic constants

if we want to define the constant in Java one type modifier is used called final. ‘final’ is the keyword used to define the symbolic constant in a Java program.

It takes the following form:

final data_type variable_name = value;

Example: *final double Pi = 3.145;*

Identifiers

- Identifiers are the programmer-designed tokens.
- They are used for naming variables, classes, methods, objects, labels, packages and interfaces in a program.
- In order to give the name for the identifiers, we have to follow following rules:

  - They can have only alphabets, digits, underscore (_) and dollar sign ($) characters.
  - They must not begin with digit. Digit can be placed anywhere except starting position.
  - Uppercase and lowercase letters are different. They can be of any length.
  - They must not be the keywords.
  - They can have any special symbol in it.

Variables

- A variable is an identifier that denotes a storage location used to store the data value.
- Unlike constants that may remain unchanged during execution of the program, a variable may take different values at different times during execution of program.
- A variable name is chosen by the programmer in meaningful way so as to reflect what it represents in the program.
- Example: `total_marks,Calci`

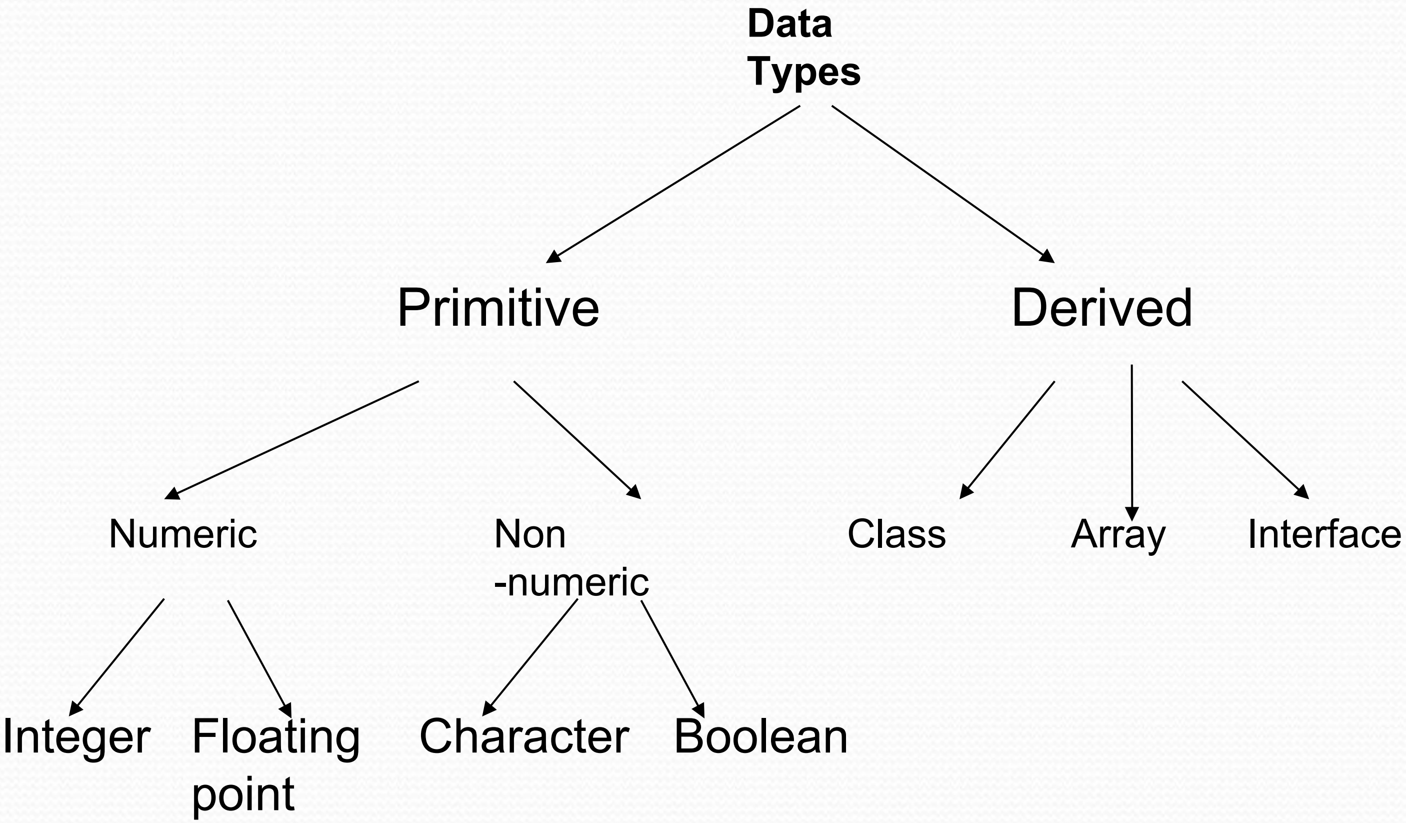
Declaration of variables:

```
Type variable1,variable2,….,variable n
```

Assign value to variable:

- `Type variable_name=value;`

Data types



Data Types

Primitive

Derived

Numeric

Non -numeric

Class

Array

Interface

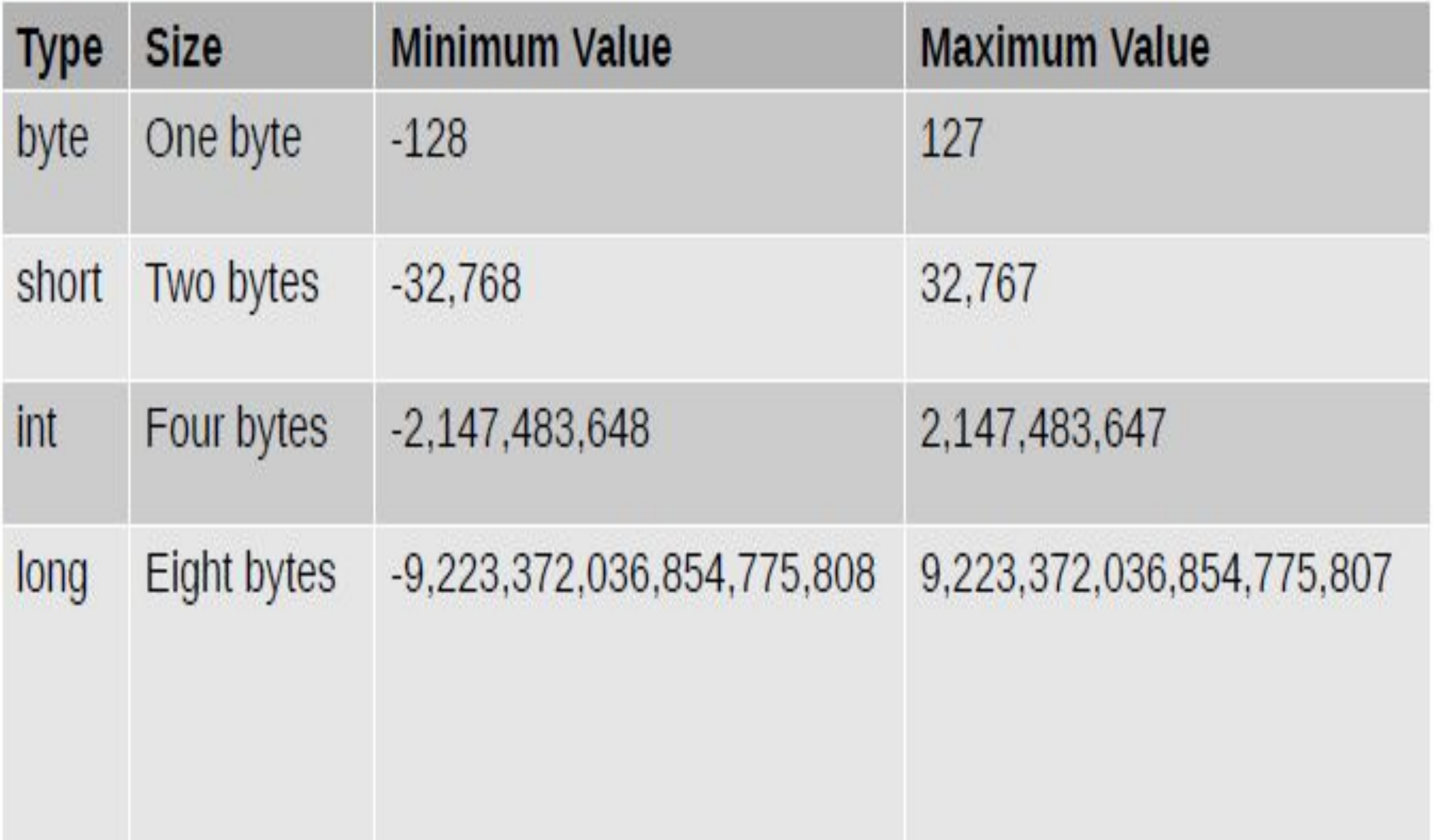
Integer

Floating point

Character

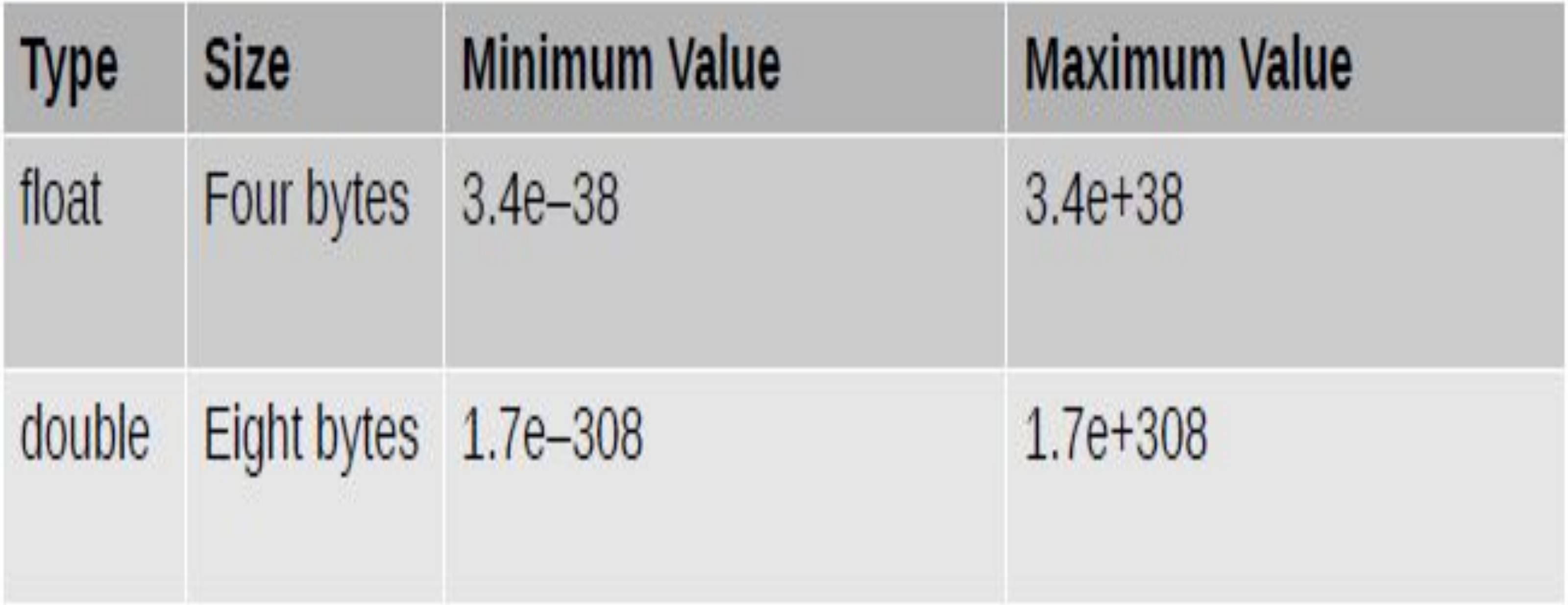
Boolean

Integer



Type	Size	Minimum Value	Maximum Value
byte	One byte	-128	127
short	Two bytes	-32,768	32,767
int	Four bytes	-2,147,483,648	2,147,483,647
long	Eight bytes	-9,223,372,036,854,775,808	9,223,372,036,854,775,807

Floating-point



Type	Size	Minimum Value	Maximum Value
float	Four bytes	3.4e–38	3.4e+38
double	Eight bytes	1.7e–308	1.7e+308

Character

- In Java, the **char** data type is used to store a **single character**.
- **Size:** 2 bytes (16 bits)
- **Range:** 0 to 65,535
- **Standard:** Unicode (supports international characters)
- **Default value:** '\u0000' (null character)
- Example: char ch = 'A';
- char digit = '5';
- char symbol = '@';
- char ch = '\u0041';

Boolean

- This is one of the most useful data types of Java.
- But, values stored inside the boolean type will only be true and false. It can not have values like 0 or 1.
- It is used in program to test a particular condition during the execution.
- Boolean is denoted by keyword boolean and uses only one bit of storage.
- All relational operators (i.e. comparison operators like <, > etc.) return these boolean values.

Strings

- The String data type is used to store a sequence of characters (text).
- String values must be surrounded by double quotes
- E.g: String greeting = "Hello World";

Scope of variables

Basically, the Java variables are categorized into three types:

- Instance variables
- Local variables
- Class variables

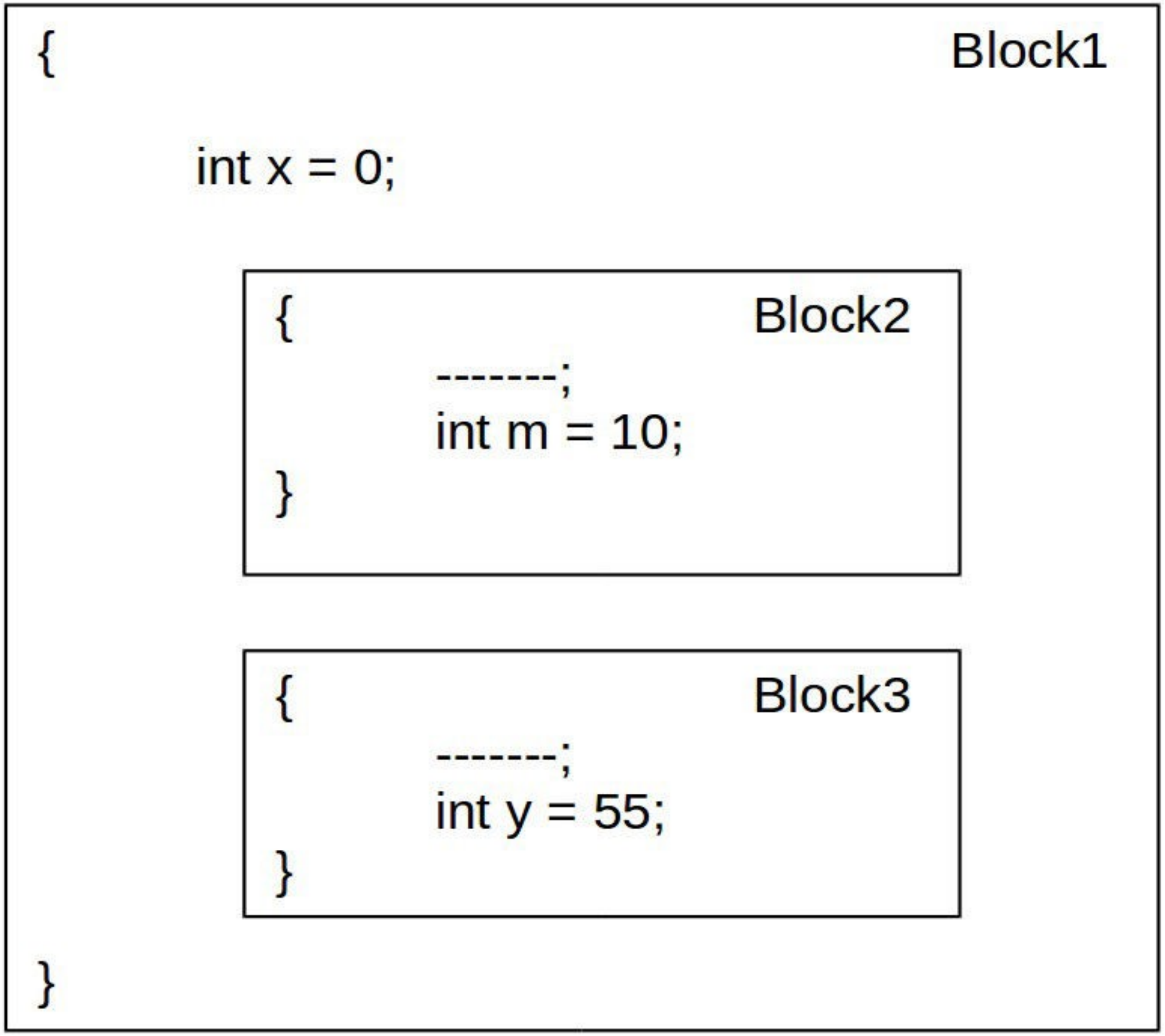
- **Instance Variables:**
- A variable which is declared inside a class and outside all the methods and blocks is an instance variable.
- General scope of an instance variable is throughout the class except in static methods.
- **Class Variables:**
- A variable which is declared inside a class ,outside all the blocks and is marked static is known as a class variable.
- General scope of an class variable is throughout the class .

- **Local Variables:**
- All other variables which are not instance and class variables are treated as local variables including the parameters in a method.
- General scope of an local variable is within the block in which it is declared.

Scope of the variables

According to the types of variables, the scopes of the Java program have following types:

- Block scope
- Class scope

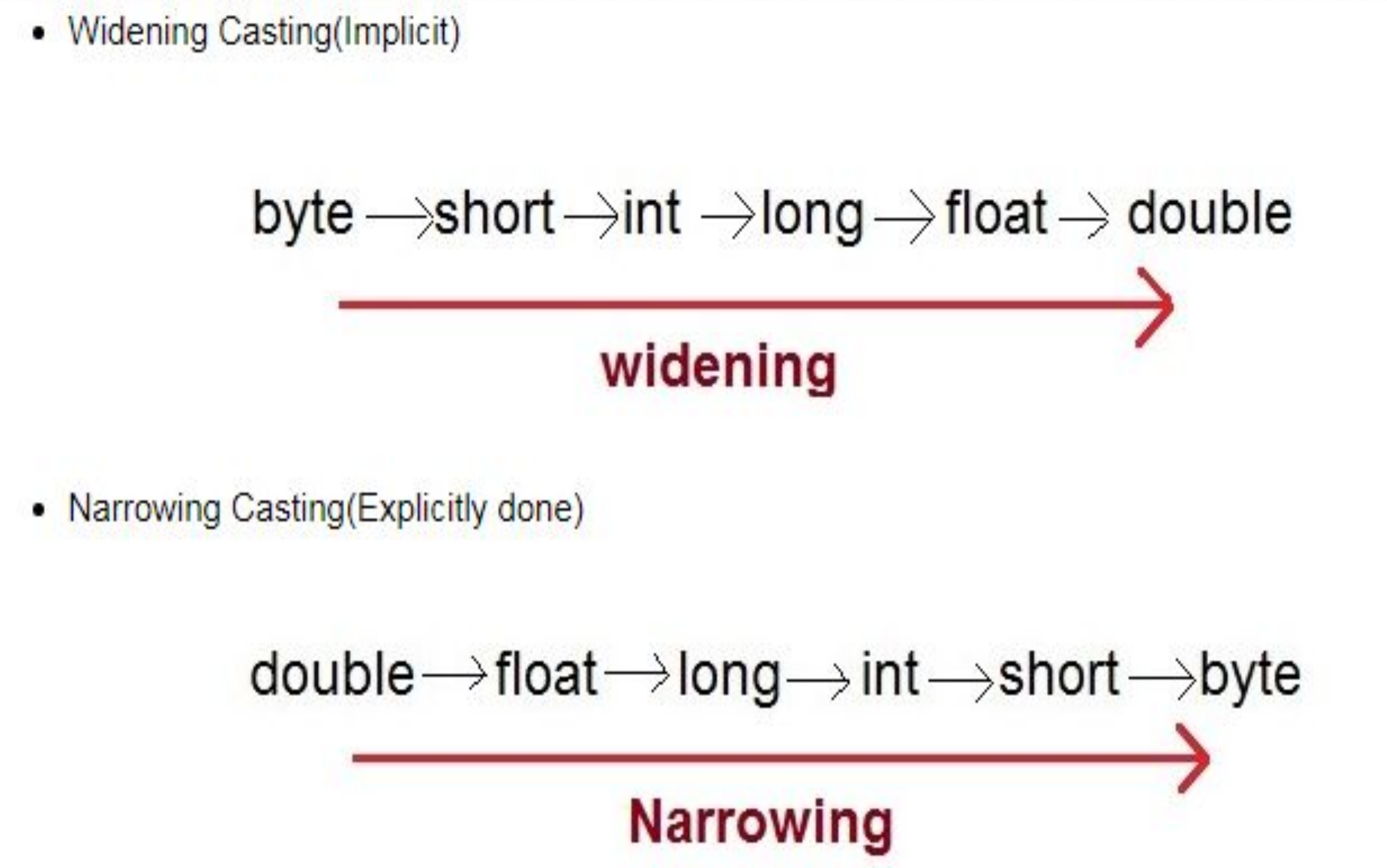


```
{                                   Block1

        int x = 0;

            {                       Block2
                    -------;
                    int m = 10;
            }

            {                       Block3
                    -------;
                    int y = 55;
            }

}
```

Type casting

- Assigning a value of one type to a variable of another type is known as **Type Casting**.
- In Java, type casting is classified into two types,
- 1. Widening Casting(Implicit)
- 2. Narrowing Casting(Explicitly done)
- In automatic type conversion always lower type is converted to higher type. This type of conversion is called as widening the conversion.
- If we want to convert higher type to lower type then type casting is necessary. For that purpose the type casting operator is used.

This type of conversion is called as narrowing the conversion. It takes the following form



- Widening Casting(Implicit)

byte → short → int → long → float → double

widening

- Narrowing Casting(Explicitly done)

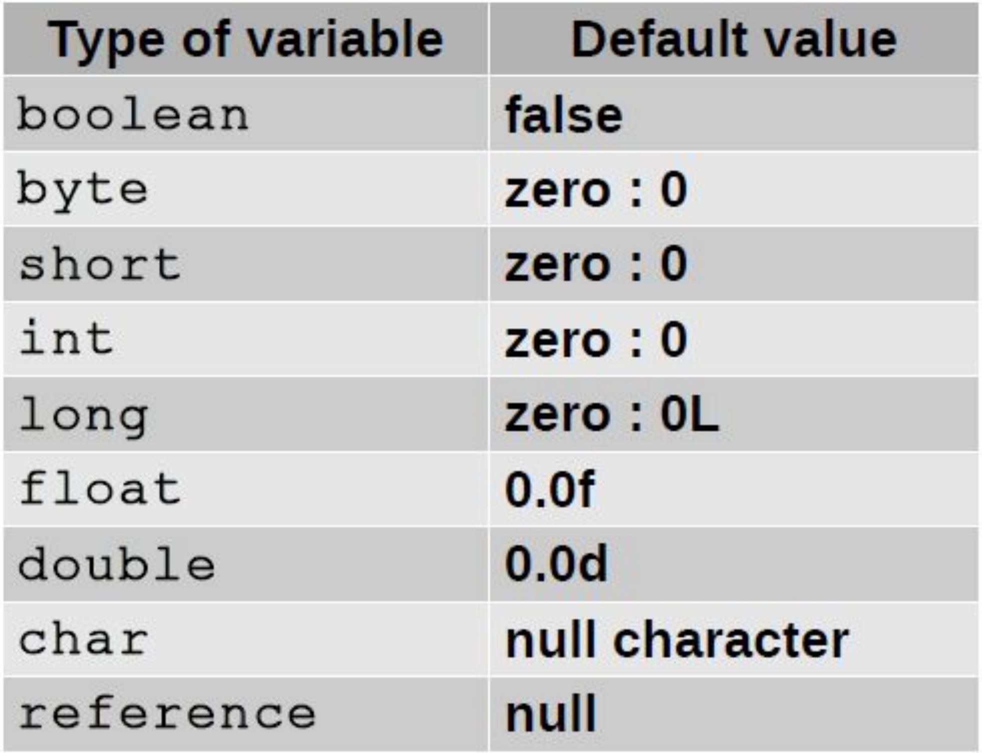
double → float → long → int → short → byte

Narrowing

Example:

```
class Casting
  {
        public              void  main(String  args[])
        static
        {     byte
              b; int val     163;
                         =
              b  =            val;
              (byte)
              System.out.println(b);
        }
  }
```

Standard default values



Type of variable	Default value
`boolean`	false
`byte`	zero : 0
`short`	zero : 0
`int`	zero : 0
`long`	zero : 0L
`float`	0.0f
`double`	0.0d
`char`	null character
`reference`	null

Operators

- Arithmetic Operators
- Assignment Operators
- Increment / Decrement Operators
- Relational Operators
- Logical Operators
- Conditional Operators
- Special Operators

Arithmetic operators

Operator	Meaning	Example
+	Addition	9 + 45
–	Subtraction	89 – 12
*	Multiplication	10 * 3
/	Division	18 / 6
%	Modulo Division	14 % 3
+	Unary Plus	+51
–	Unary Minus	–92

Relational operators

Operato	Meanin
<	Less than
>	Greater
<=	Less than or equal to
>=	Greater than or equal
==	Equal to
!=	Not equal to

Logical operators

Operator	Meaning
& &	Logical AND
| |	Logical OR
!	Logical NOT

Increment-Decrement

Like C and C++, Java is also having the increment and decrement operators' i.e.

`++ and   --`

Both of these are unary operators. The operator ++ adds 1 to the operand and – – subtracts 1 from the operand. They are only associated with the variable name, not with the constant or the expression. They can be written in following from:

```
x++ or   x—-

++x or   --x
```

- **A prefix operator first adds 1 to operand and then assigned the result to the variable on left.**
- **A postfix operator first assigns the value to the variable on left and then increments the operand.**

Conditional operator

The only ternary operator (operator having three operands) is defined in Java called as conditional operator. The character pair ? : is termed as conditional operator. This is used to construct the conditional expression of the following form:

```
condition  ? expression2  : expression3
```

Bitwise operators

Operato	Meanin
&	Bitwise
|	Bitwise
^	Bitwise
~	One’s
<<	Left
>>	Right
>>>	Right shift with zero fill

Dot operator

This is also called as dot operator (.). It is used to access the instance variable and methods of the class using objects.

For example:

```
company.salary()            //reference to method salary
company.employee            //reference to variable employee
```

Decision making and looping

- **if** statement
- **if-else** statement
- **switch-case** statement
- Conditional operator statement

if statement

```
if   (condition)
      statement;
```

or

```
if   (condition)
{
statement1;
statement2;
statement3;
}
```

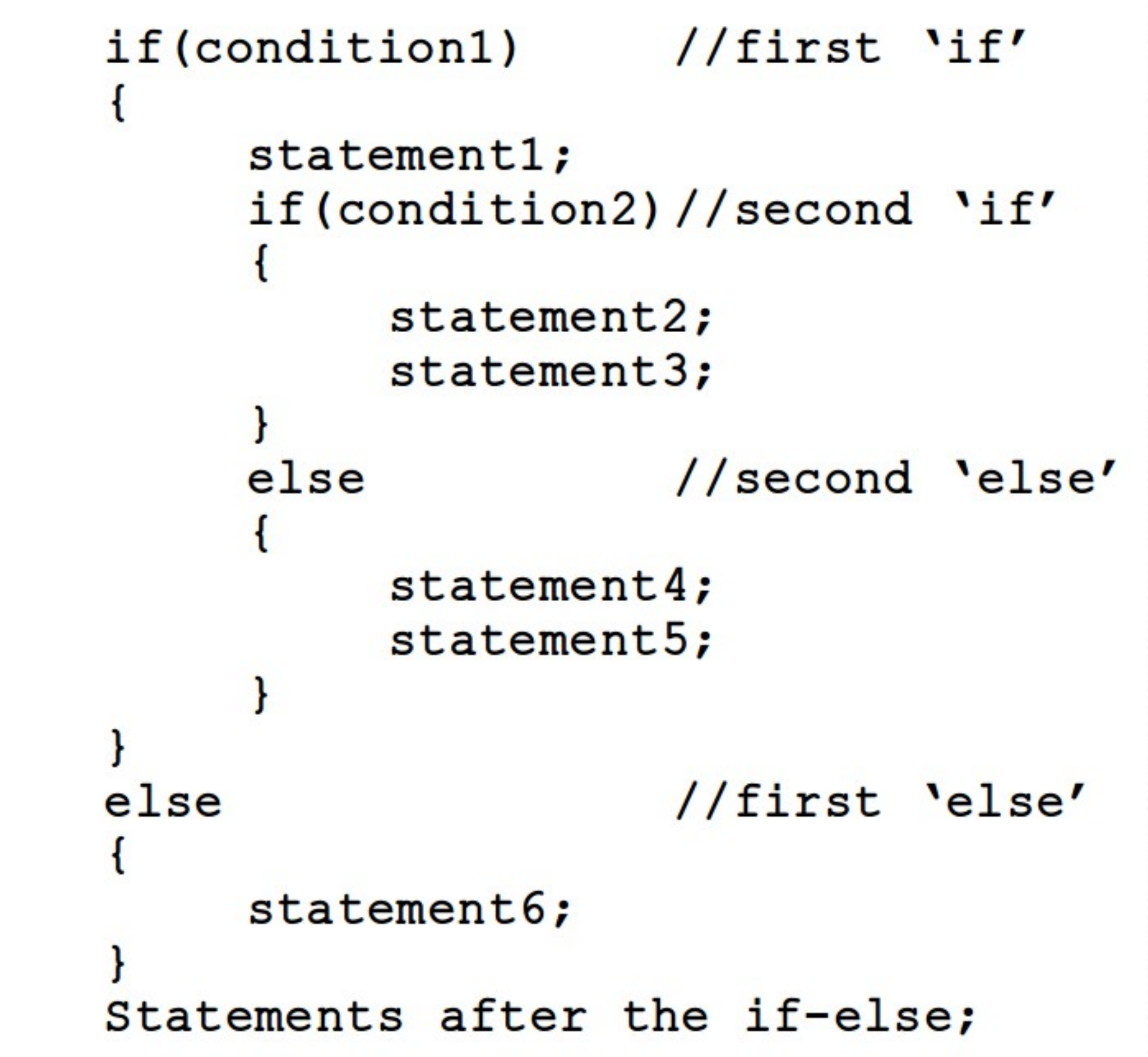
Example:

```
1.if(number < 0)
   System.out.println(“The number is negative”);

2.if(ch > ‘A’ && ch < ‘Z’)
System.out.println(“It is upper case
   letter”);

3.if(sell_price > cost_price)
   System.out.println(“You made profit”);
```

if-else statement

```
if(condition)
{
     statements for condition is true;
}
else
{
     statements for condition is false;
}
Statements after the blocks;
```

Nested if-else



```
if(condition1)     //first ‘if’
{
     statement1;
     if(condition2)//second ‘if’
     {
          statement2;
          statement3;
     }
     else          //second ‘else’
     {
          statement4;
          statement5;
     }
}
else               //first ‘else’
{
     statement6;
}
Statements after the if-else;
```

If-else ladder

```
if(condition1)
{
//code to be executed if condition1 is true
}
else if(condition2)
{
//code to be executed if condition2 is true
}
else if(condition3)
{
//code to be executed if condition3 is true
}
...
 else
{
//code to be executed if all the conditions are false
}
```

The switch statement

```
switch(variable)
{
   case  value-1:
         statements-1;
         break;
    case
         value-2:
         statements-2;
         break;
              - -  -  -  -  -
                -  -  -
              - -  -  -  -  -
                -  -  -
              default  block;
   default:
}
statement-out;
```

Example:

```
class   NumDisplay
{
public  static  voidmain(String  args[])
{
int x   =   6;
 System.out.println("x    =    "+x);
System.out.print("It  is   ");
switch(x)
{
               case1:   System.out.println("One");
                        break;
               case2:   System.out.println("Two");
                        break;
               default: System.out.println("No.   not correct");
           }
       }
  }
```

The while loop

```
Initialization;
while (condition)
{
//bodyofthe loop;
}
```

Example:

```
int i = 0;

while(i<10)

{
System.out.println(“I Love Java”);

 i++;

}
```

The do-while loop

```
do
{
    //bodyofthe    loop;
}
while(condition);
```

Example:

```
int a = 0; do

{
System.out.println(“I Love Java”); a++;

}

while(a<10);
```

The for loop

```
for(initialization ; condition ; increment/decrement)
{
    Body of the loop;
}
```

The for loop

```
for(a = 0;a < 10; a++ )
{
        System.out.println(a);
}
```

Comparison

whil	do-whil	for

x = 0;
while(x<10)
{
--------;
--------; x++;
}
	x =
	0;
	do
	{
	--------;
	--------;
	x++;
	}
	while(x<10);
		for(x=0;x<10;
		x++)
		{
		--------;
		--------;
		}

Read Input From Console

- The Scanner class is used to get user input, and it is found in the java.util package.
- To get the instance of Java Scanner which reads input from the user, we need to pass the input stream (System.in) in the constructor of Scanner class.
- For Example:
- Scanner sc = new Scanner(System.in)
  - String var = sc.nextLine()
  - int var1 = sc.nextInt()
  - float var2 = sc.nextFloat()

The Java Scanner class provides nextXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc.

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
next() OR nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Read Input From Cosole

- BufferedReader and InputStreamReader:
- The BufferedReader class of Java is used to read the stream of characters from the specified source (character-input stream). The constructor of this class accepts an InputStream object as a parameter.
- This class provides a method named **read()** and **readLine()** which reads and returns the character and next line from the source (respectively) and returns them.
- Example
  - BufferedReader br = new BufferedReader(new InputStreamReader(System.in))
  - String v1 = br.readLine()
  - int v2 = Integer.parseInt(br.readLine())
  - float v3 = Float.parseFloat(br.readLine())

Reading data from keyboard

- There are many ways to read data from the keyboard. For example:
- BufferdReader & InputStreamReader
- Console
- Scanner
- DataInputStream etc.

Java Command-Line Arguments

- The **command-line arguments** in Java allow us to pass arguments during the execution of the program.
- As the name suggests arguments are passed through the command line.
- The main() method includes an array of strings named args as its parameter.

  public static void main(String[] args) {...}
- The String array stores all the arguments passed through the command line.
- **Note**: Arguments are always stored as strings and always separated by **white-space**.
- we can pass the arguments after the class name. i.e java demo apple ball cat

Passing Numeric Command-Line Arguments

- The main() method of every Java program only accepts string arguments. Hence it is not possible to pass numeric arguments through the command line.
- Here, convert string arguments into numeric values.
- Here, the parseInt() method of the Integer class converts the string argument into an integer.
- Similarly, we can use the parseDouble() and parseFloat() method to convert the string into double and float respectively.

Class

- In Java everything is **encapsulated under classes**. Class is the core of Java language. It can be defined as **a template** that describe the behaviors and states of a particular entity.
- Class is template or blueprint of an object.
- It is logical entity.
- In Java, to declare a class **class** keyword is used. A class contain both **data and methods** that operate on that data. The data or variables defined within a class are called **instance variables** and the code that operates on this data is known as **methods**.

- Java class Syntax

```
class classname
{
    field;
        method;
}
```

Class

- Syntax of Class:

```
class  classname [extends superclassname, implements
interface name]
{
 [variable declarations]
 [method declarations]
 [method definitions]
 [main method definition]
}
```

Object

- Object is basic runtime entity.
- Physical entity.
- It can be represent: Person, table, place, bank account etc.
- Object may represent user defined data types: Vectors, stack, Linked List.
- Object can be communicate with other object: Customer and account object can communicate.

Object creation

- Object creation also called as instantiation of object.
- With the help of ‘new’ keyword object is created.
- “new” operator is used to create object of specified class and return reference to that object.

Syntax:

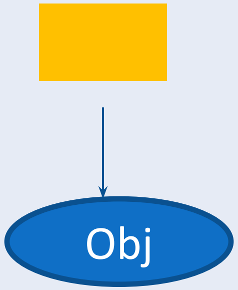
Classname Objectname = new Classname();

Object creation

- Ex. Rectangle rect; //declare object

  rect = new Rectangle(); //instantiate the object
- First statement is only declare variable to hold the object.
- Second statement actually assign the object reference to the variable.

Object creation

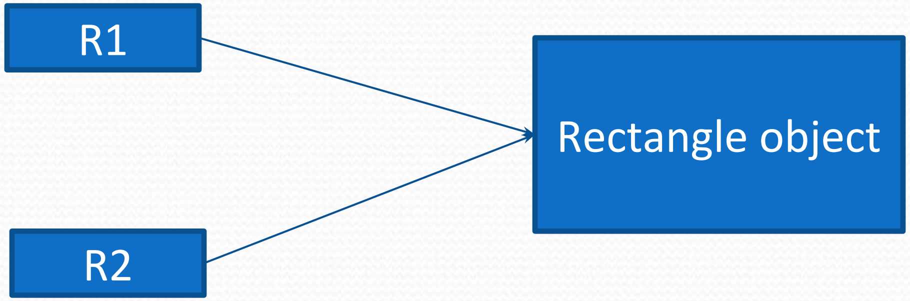
Action	Statement	Result
Declare	Rectangle rect;	rect Null
Instantiate	rect = new Rectangle()	rect 

Object creation

- Multiple object can be created for one class.
- Each object has its own copy of instance variables.
- Any changes by one object of instance variable will not affect to other object.

Object creation

- It is also possible to create 2 or more references of one object.
- Rectangle r1 = new Rectangle();
- Rectangle r2 = r1;



Accessing Instance Variables and Methods

- Instance variables and methods are accessed via created objects. To access an instance variable, following is the fully qualified path −
- /* First create an object */

Classname Objectname= new Classname();

/* Now call a variable as follows */
Objectname.variableName;

/* Now you can call a class method as follows */
Objectname.MethodName();

Constructor

- Special type of method whose name same as class name.
- Constructor gets automatically calls once object is created.
- It does not have any return type and not even void.
- Job of constructor is used to initialize internal state of object. i.e. initialize data members.

Constructor

- Two types of constructor:
  - **Default or parameter less constructor:**A constructor that has no parameter is known as default constructor. If we don’t define a constructor in a class, then compiler creates **default constructor(with no arguments)** for the class.
  - **Parameterized constructor :** A constructor that has parameters is known as parameterized constructor.
- Example: Suppose class name is Rectangle
  - Rectangle()
  - Rectangle(int height, int width)
  - Example: studentdemo.java

Constructor Overloading

- When class has more number of constructor with different number of parameters/arguments.
- Example:Constructoroverload.java

Nesting of Methods:

- When a method in java calls another method in the same class, it is called Nesting of methods.
- We can invoke that method by simply using the name of it.
- example: nestingdemo.java

Method Overloading in Java

- If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.
- There are two ways to overload the method in java
- 1.By changing number of arguments
- 2.By changing the data type

Arrays

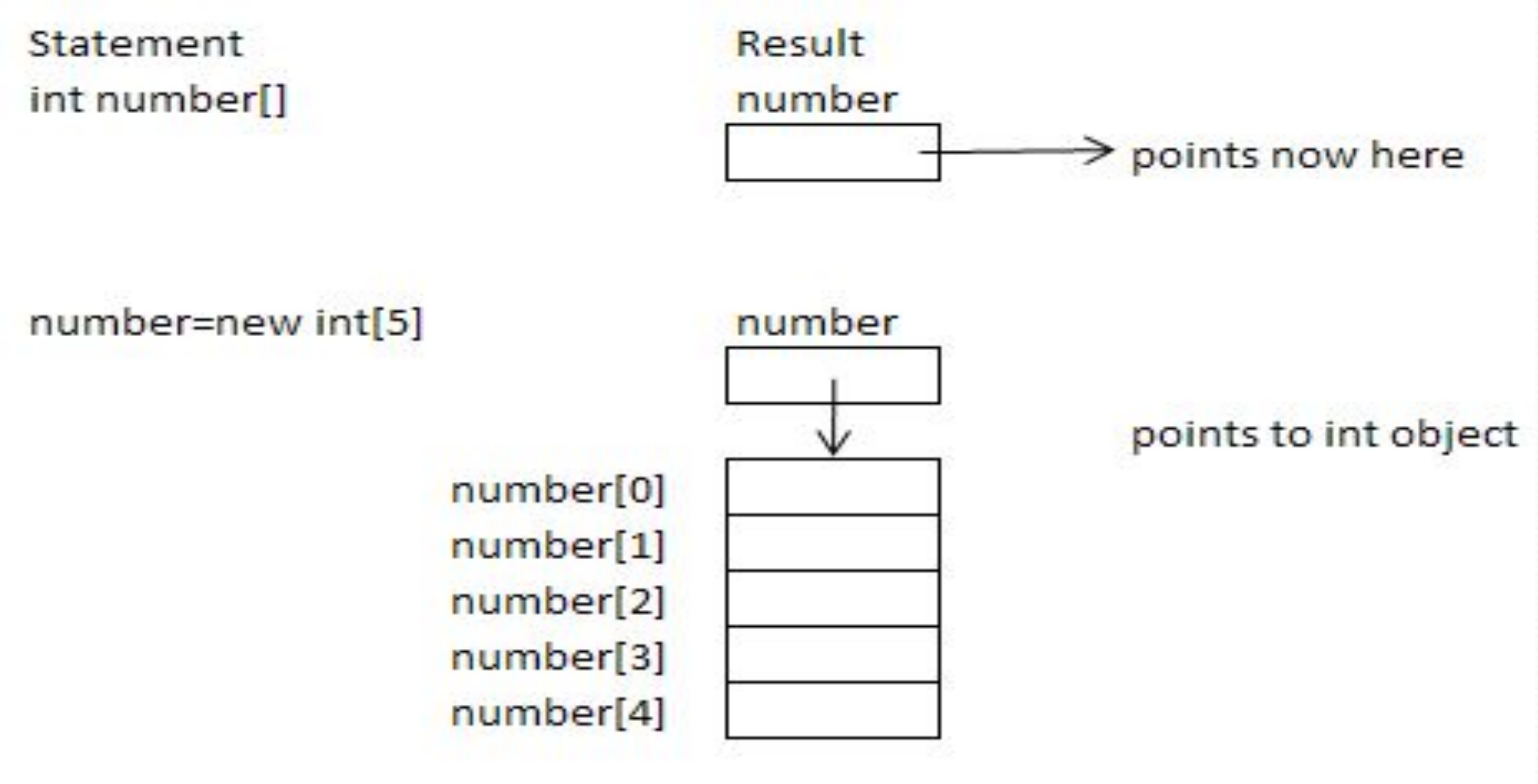
- Collection of similar type of elements.
- Types:
  - 1-D Array
  - 2-D Array

Arrays

- Creation of Array has involved three steps:
  - Declaring array
  - Creating memory location using new keyword
  - Putting values into memory locations.
- Declaration: Syntax with example
  - Form 1 : datatype arrayname[]; Ex: int a[];
  - Form 2: datatype [] arrayname; Ex int [] a;

Arrays

- Creating memory locations:
  - Arrayname = new type[size]
- Example
  - int [] number = new int[6];
  - float f [] =new float[4];
  - Student [] obj = new Student[5]



Statement

Result

int number[]

number

points now here

number=new int[5]

number

points to int object

number[0]

number[1]

number[2]

number[3]

number[4]

Arrays

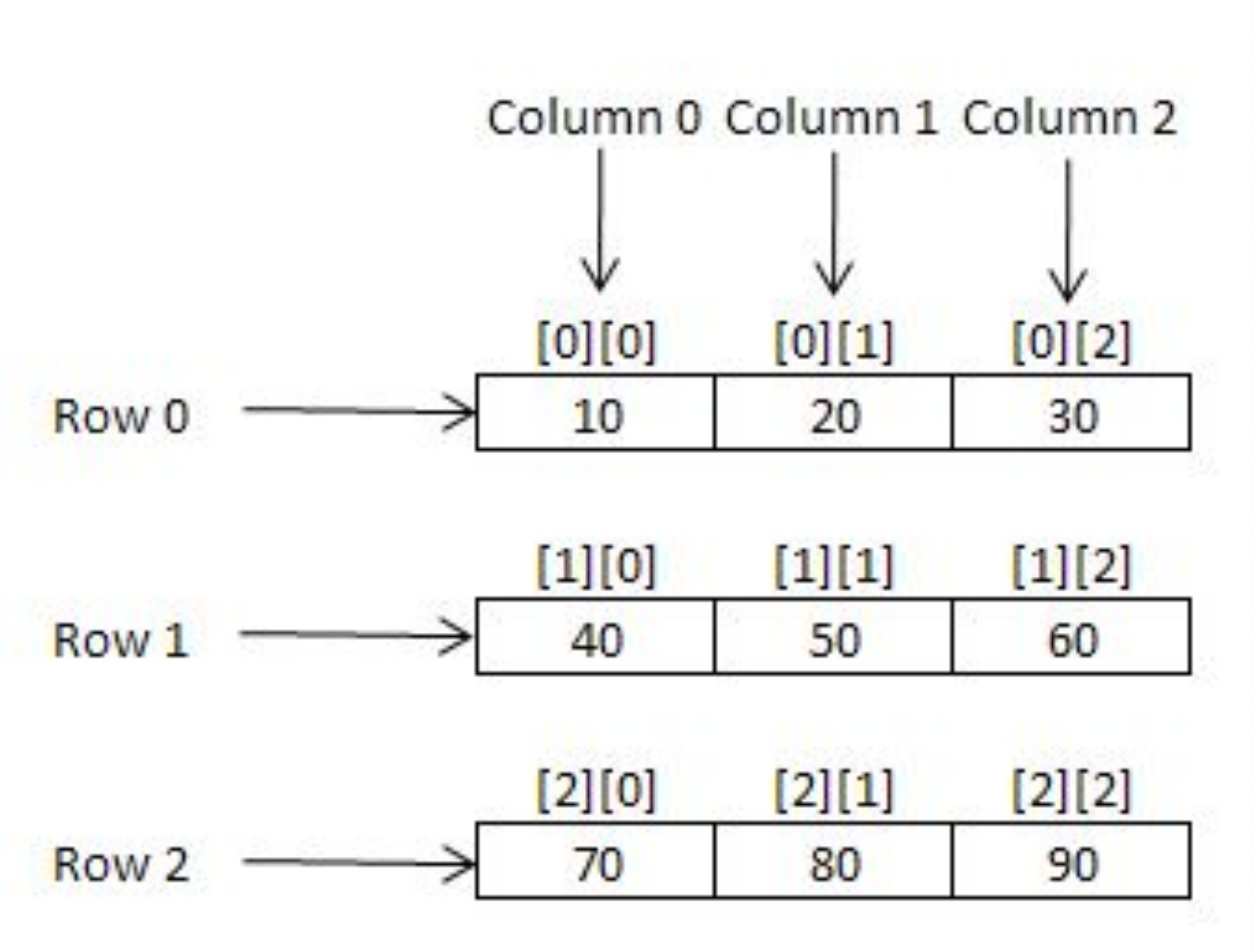
- Initialization of Array:
  - Arrayname [index] = value
  - Or int [] a ={list of values}
- **Array length:**
- In java all array store the allocated size in a variable named length.We can access the length of the array using **a.length**
- □int asize= **a.length;**

Arrays: 2D

- Declaration:
  - Form 1: type arrayname[][];
  - Form 2: type [][] arrayname;
- Example:
  - int arrayname [][]
  - Int [][] arrayname

Arrays: 2D

- Creating memory location:
  - Arrayname = new type[row][col]
  - Ex: avg = new Float[3][2]
- Initialization:
  - Arrayname[rowindex][colindex]=value



	Column 0	Column 1	Column 2
Row 0	[0][0] 10	[0][1] 20	[0][2] 30
Row 1	[1][0] 40	[1][1] 50	[1][2] 60
Row 2	[2][0] 70	[2][1] 80	[2][2] 90

Arrays: 2D

- Another way:
- Form 1:
  - int a [3][2] = {1,2,2,3,3,4}
- Form 2:
  - Int a[][] = {{1,2},{2,3},{3,4}}

Arrays: 2D

- Variable size array:
- int x [][] = new int [3][];
  - X[0] = new int[2]
  - X[1] = new int[4]
  - X[2] = new int[3]

String

- String is sequence of characters and can be implemented using two classes : “String” and “StringBuffer”.
- Both classes are present in java.lang package.

String

- String can be declared and created are:
  - String s1 = new String(“Hello”);
  - String s2 = “Hello”
- String class creates immutable string that means once created object of String class could not change.
- Reference can be changed but we can not alter assigned string.

String Array

- Syntax
- String a [] = new String [2]
- a[0] = “Hello”
- a[1] = “World”

String Methods

- **String toLowerCase()**
- **String toUpperCase()**
- **String replace(char oldchar, char newchar)**
- **String trim():** This method eliminates leading and trailing spaces.
- **Boolean equals(Object str):** This method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

String Methods

- **int length()**
- **char charAt(int)**
- **int compareTo(String str) :**This method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

if s1 > s2, it returns positive number

if s1 < s2, it returns negative number

if s1 == s2, it returns 0

- **String concat(String str)**
- **String substring(int beginIndex)**
    - **String substring(int begIndex, int endIndex)**

```
class stringexample
{
String s = “Hell”;
String s1 = “Hello”;
String s2 = “Hello”;
String s3 = “Java”;
System.out.println(s1.equals(s2));      //true
System.out.println(s.equals(s1));       //false
String s4 = new String(“Java”)
System.out.println(s1==s2);                  //true
System.out.println(s3==s4);                  //false
System.out.println(s.compareTo(s2));    //returns-1       because  s<s2
System.out.println(s1.compareTo(s2));  //returns 0       because  s1=s2
System.out.println(s2.compareTo(s));    //returns 1        because  s2>s
}
}
```

String Methods

- String toString(): Create string representation of object p
  - Ex : p.toString()

- String valueOf(p): This method converts different types of values into string.
  - Ex: float p = 98.5f
  - String str = String.valueOf(p) + “ABC”=>98.5ABC

String Methods

- indexOf()
  - Syntax: int indexOf(char ch)
  - int indexOf(char ch,int n)

StringBuffer Class

- Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed. we can modify the string in terms of length and contents.
- We can insert the characters and substring in the middle of string, or append another string.
- Syntax:
  - StringBuffer sb = new StringBuffer(“Hello world”);

StringBuffer Methods

- **setCharAt(int index,char ch)** : Modify by index character to ch.
- **append(String s2):** Append string s2 to first string at the end.
- **insert(int n, String s2):** Insert the string s2 at the nth position of the string.
- **Replace(int start,int end,String str):**It replaces the characters specified by the new string.
- **Delete(int start,int end):**It deletes the characters from the string specified by the starting and ending index.
- **setLength(int n):** Set the length of string to n
  - If n<s1.length() : s1 is truncated.
  - If n>s1.length() : Spaces are added to s1.

Vectors

- Used to create generic and dynamic array which can contain object/variable of any type and any number.
- The objects/values may not be homogeneous.
- It is found in the java.util package
- Syntax:
  - Vector v1 = new Vector();
  - Vector v2 = new Vector(5);
- **Add Elements to Vector**
- add(element) - adds an element to vectors
- add(index, element) - adds an element to the specified position

- **Constructor:**
- **Vector()**: Creates a default vector of initial capacity is 10.
- **Vector(int size):** Creates a vector whose initial capacity is specified by size.
- **Vector(int size, int incr):** Creates a vector whose initial capacity is specified by size and increment is specified by incr. It specifies the number of elements to allocate each time that a vector is resized upward.
- **Important points regarding Increment of vector capacity:** If increment is specified, Vector will expand according to it in each allocation cycle but if increment is not specified then vector's capacity get doubled in each allocation cycle.

Vector Methods

- void addElement(Object item)
- Object elementAt(int index)
- int size() & int capacity()
- boolean removeElement(Object item)
- void removeElementAt(int index)
- void removeAllElements()
- void copyInto(Object arr[])
- void insertElementAt(Object item, int index)

Wrapper Classes

- To convert the primitive data type to object and object to primitive:=□ Wrapper classes are used.
- Present in java.lang package

Wrapper Classes

Simple Type	Wrapper class
boolean	Boolean
char	Character
int	Integer
long	Long
float	Float
double	Double

Converting Primitive to Object

Constructor Calling	Conversion action
Integer intObj = new Integer(i)	Primitive Integer to integer object
Float floatObj = new Float(f)	Primitive Float to float object
Double dobj = new Double(d)	Primitive double to double object
Long lobj = new Long(l)	Primitive long to long object

Converting Object to Primitive

Method Calling	Conversion action
int I = intObj.intValue()	Integer object to integer primitive
float f = floatObj.floatValue()	Float object to float primitive

Converting Numeric String to Primitive number

Method Calling	Conversion action
int i = Integer.parseInt(str)	Convert string to integer
float f = Float.parseFloat(str)	Convert string to flaot

Converting String object to numeric object

Method Calling	Conversion action
intObj = Integer.valueOf(str)	Convert string to integer object
floatObj = Float.valueOf(str)	Convert string to float object

Java For-each Loop | Enhanced For Loop

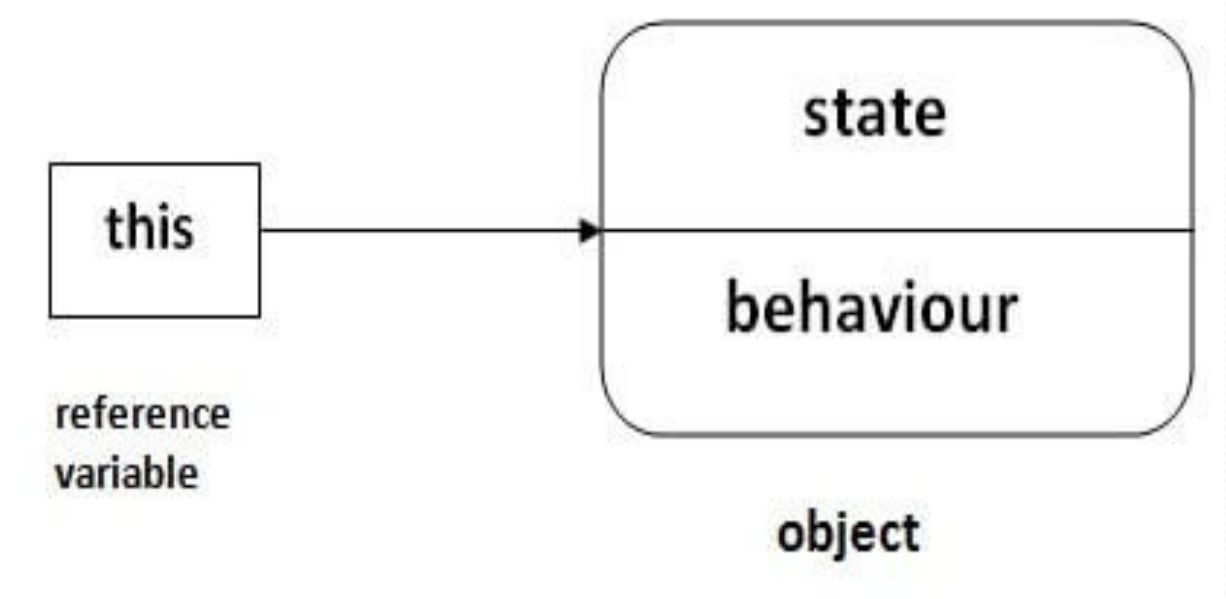
- The Java for-each loop or enhanced for loop is introduced since J2SE 5.0.
- It provides an alternative approach to traverse the array or collection in Java.
- It is mainly used to traverse the array or collection elements.
- The advantage of the for-each loop is that it eliminates the possibility of bugs and makes the code more readable.
- It is known as the for-each loop because it traverses each element one by one.

- The syntax of Java for-each loop consists of data_type with the variable followed by a colon (:), then array or collection.

```
for(data_type variable : array or collection)
{
  //body of for-each loop
}
```

this keyword in java

There can be a lot of usage of **java this keyword**.
In java, this is a **reference variable** that refers to the current object.



this keyword in java

- **Usage of java this keyword**

Here is given the 6 usage of java this keyword.

1.this can be used to refer current class instance variable.

2.this can be used to invoke current class method (implicitly)

3.this() can be used to invoke current class constructor.

4.this can be passed as an argument in the method call.

5.this can be passed as argument in the constructor call.

6.this can be used to return the current class instance from the method.

Garbage Collection

- In java garbage means “Unreferenced Object”.
- It is process of destroying unused objects.
  - In case of C □ free()
  - In case of C++ □ delete()
  - In case of Java □ Performed automatically, so its provide better memory management.
- Garbage Collector□ Part of JVM to performed this task.

Garbage Collection

- How can an object unreferenced?
  - By Nulling the reference:
    - Student s = new Student();
    - s=null;
  - By assigning reference to another
    - Student s1 = new Student();
    - Student s2 = new Student();
    - s2 = s1
  - By Anonymous Object
    - new Student()

Garbage Collection

- finalize():
  - Execute each time for garbage collection. This method can be used for cleanup processing.
  - Syntax:

```
 protected void finalize()
  {
    //finalize-code
 }
```

- If you want forcefully garbage collection:
  - gc() method is used.
  - System.gc()

Visibility Control:

- **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Visibility Control : Access Modifier

Access Levels					
Modifier	Same Class	Sub class in Same Package	Non Subclass in same package	Sub class in other Package	Non Subclass in other package
public	Y	Y	Y	Y	Y
protected	Y	Y	Y	Y	N
Default	Y	Y	Y	N	N
private	Y	N	N	N	N