CAP615
PROGRAMMING IN JAVA

Unit- 1

Created By:
Kumar Vishal
(SCA), LPU

Unit-1

Introduction to Java :

- ✓ basic java concepts,
- ✓ JDK, JRE and JVM,
- ✓ wrapper classes,
- ✓ inner and nested classes,
- ✓ working with arrays and strings,
- ✓ String, String Buffer and StringBuilder classes,
- ✓ access specifiers,
- ✓ inheritance

Introduction about the java programming development tools

What is development tools in Java?

JDK (Java Development Kit)

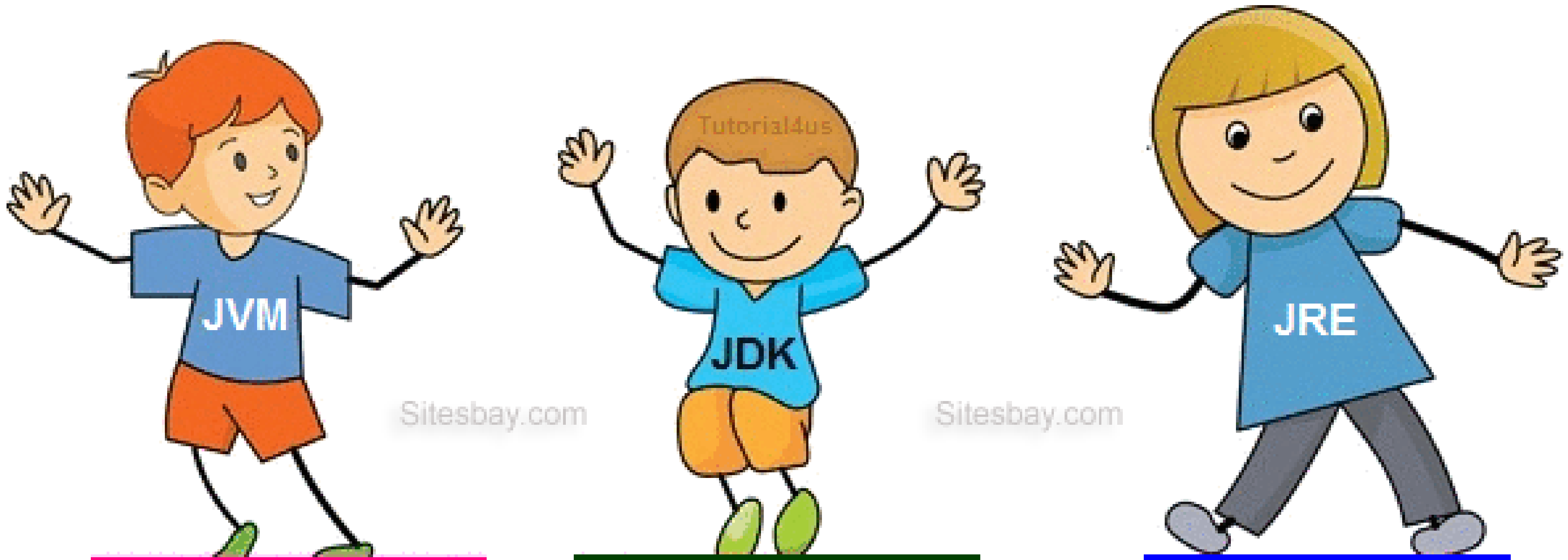
JDK

JDK is Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications.

JRE (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs.

JVM (Java Virtual Machine) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an interpreter.

Working of JVM, JDK and JRE



I provides runtime environment to execute bytecode.

I am physically exists and contains Java Tools, JVM, JRE

I contains set of libraries and tools

There are different types of Java editions are available to develop applications:

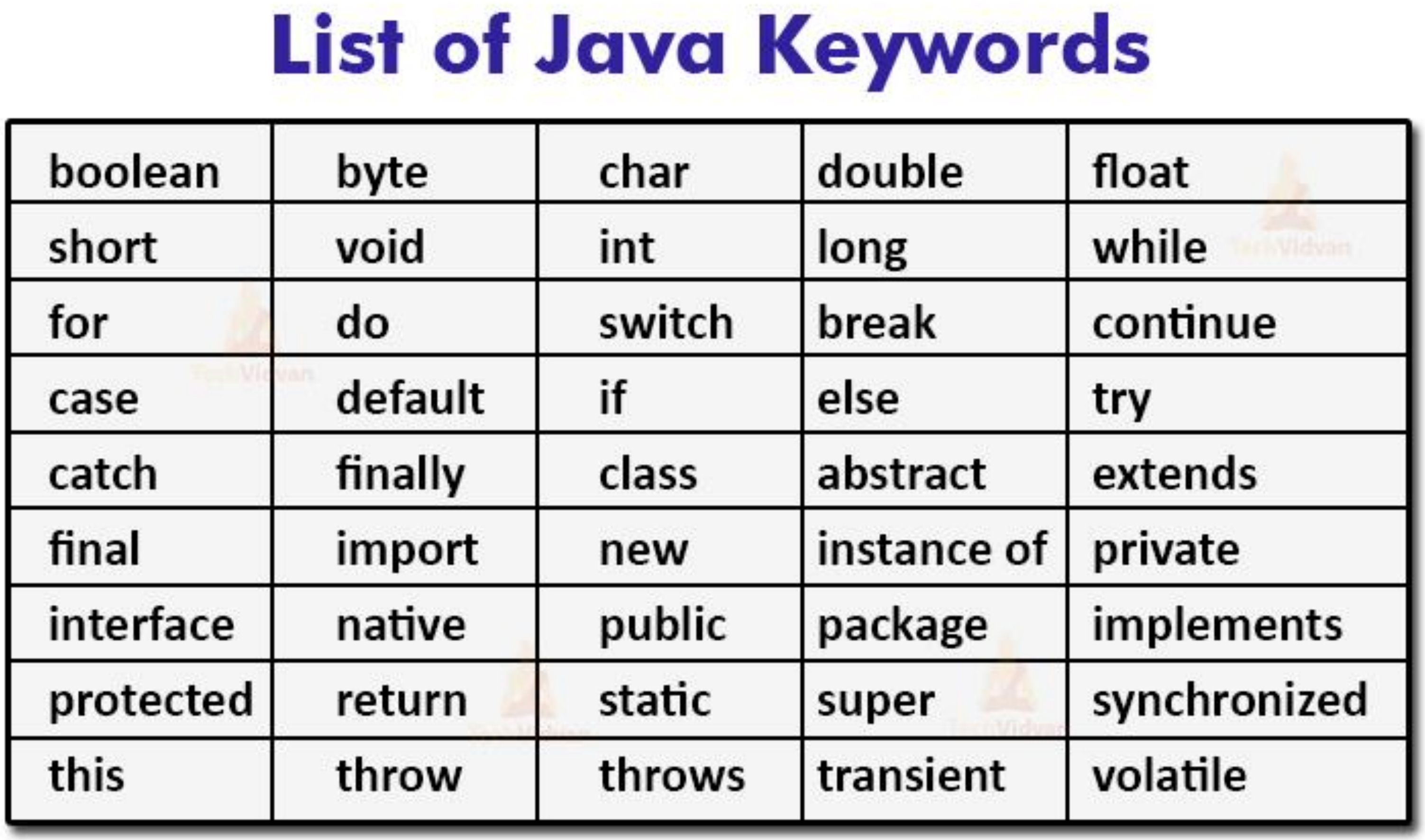
- ✓ Java Standard Edition (JSE)
- ✓ Java Enterprise Edition(JEE)
- ✓ Java Micro Edition(JME)

- Java Standard Edition (JSE)
  - JSE can be used to develop client-side standalone (independant) applications or applets.
- Java Enterprise Edition (JEE)
  - JEE can be used to develop server-side applications such as Java servlets and Java ServerPages.
- Java Micro Edition (JME).
  - JME can be used to develop applications for mobile devices such as cell phones.

Basic java concepts:

- ✓ Introduction about the java programming development tools,
- ✓ Java keywords,
- ✓ variables,
- ✓ data types,
- ✓ operators and
- ✓ control statements

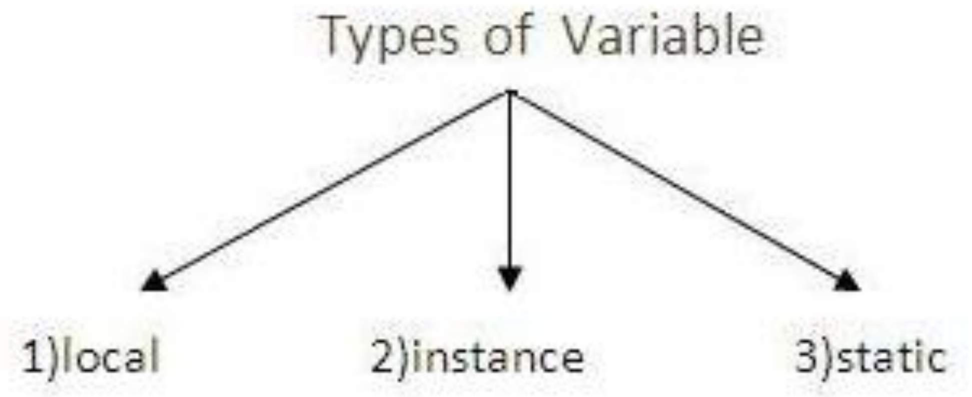
Java keywords: reserve words

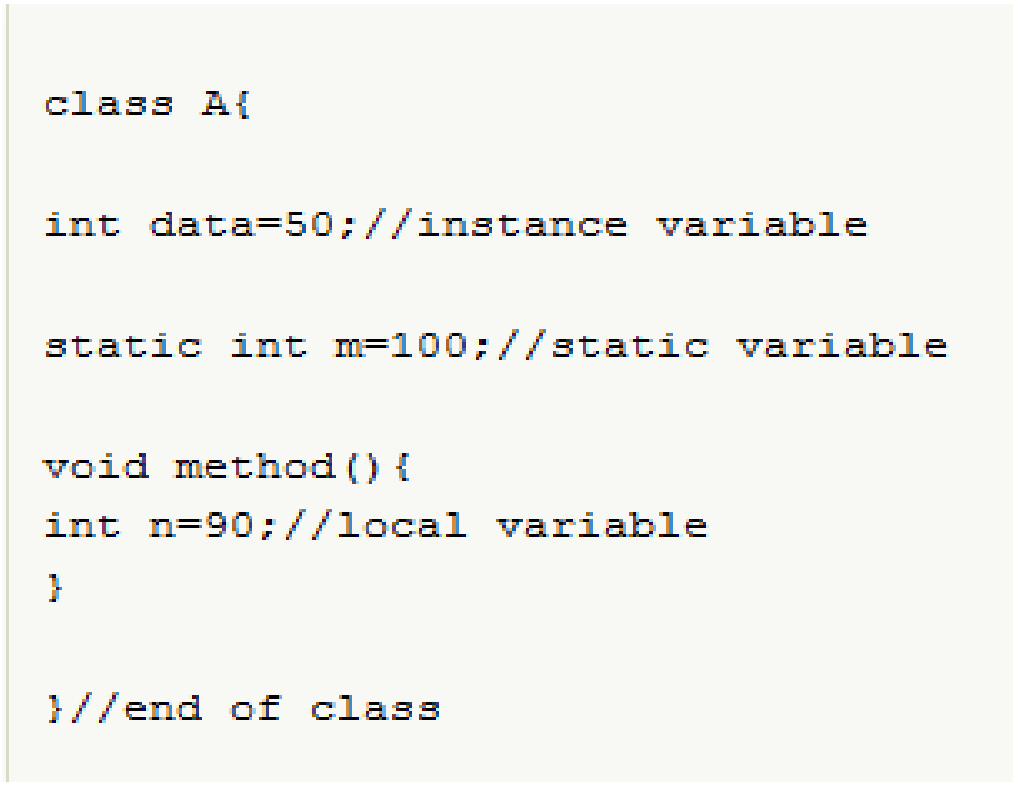


List of Java Keywords

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

Variables:





```
class A{

int data=50;//instance variable

static int m=100;//static variable

void method(){
int n=90;//local variable
}

}//end of class
```

Instance vs static

- Instance variables gets the memory at the time of object creation, each object will have the copy of instance variable, if it is incremented, it won’t reflect to other objects.
- Static variable will get the memory only once, if any object changes the value of the static variable , it will retain its value.

Data types in Java

Primitive

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

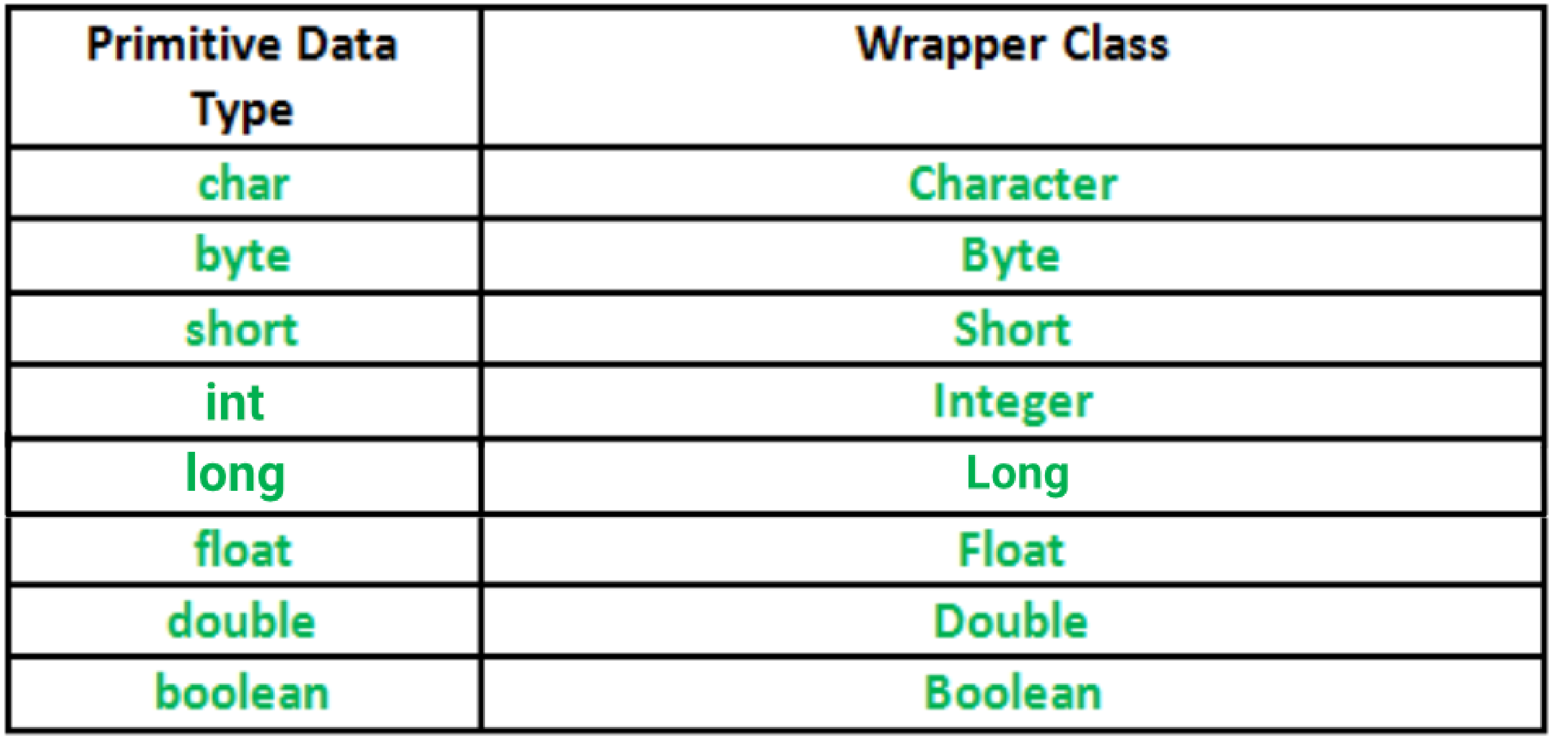
Non primitive

- String
- Array
- Class
- Interface

To find size of datatypes

```
class MainClass
{
    public static void main(String []args)
    {
        System.out.println("Size of int: " + (Integer.SIZE/8) + " bytes.");
        System.out.println("Size of long: " + (Long.SIZE/8) + " bytes.");
        System.out.println("Size of char: " + (Character.SIZE/8) + " bytes.");
        System.out.println("Size of float: " + (Float.SIZE/8) + " bytes.");
        System.out.println("Size of double: " + (Double.SIZE/8) + " bytes.");
    }
}
```

A **Wrapper class** is a class whose object wraps or contains primitive data types.



Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

When we create an object to a wrapper class, we can store primitive data types.

Example:

```
 char ch = 'a';
 Character a = ch;
```

In which case we can use:

Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.

Important :what is autoboxing and unboxing?

The wrapper class in Java provides the mechanism to convert primitive into object and object into primitive.

The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

In both which one is correct and why?

```
float  a=10.123456789
double b=10.123456789
```

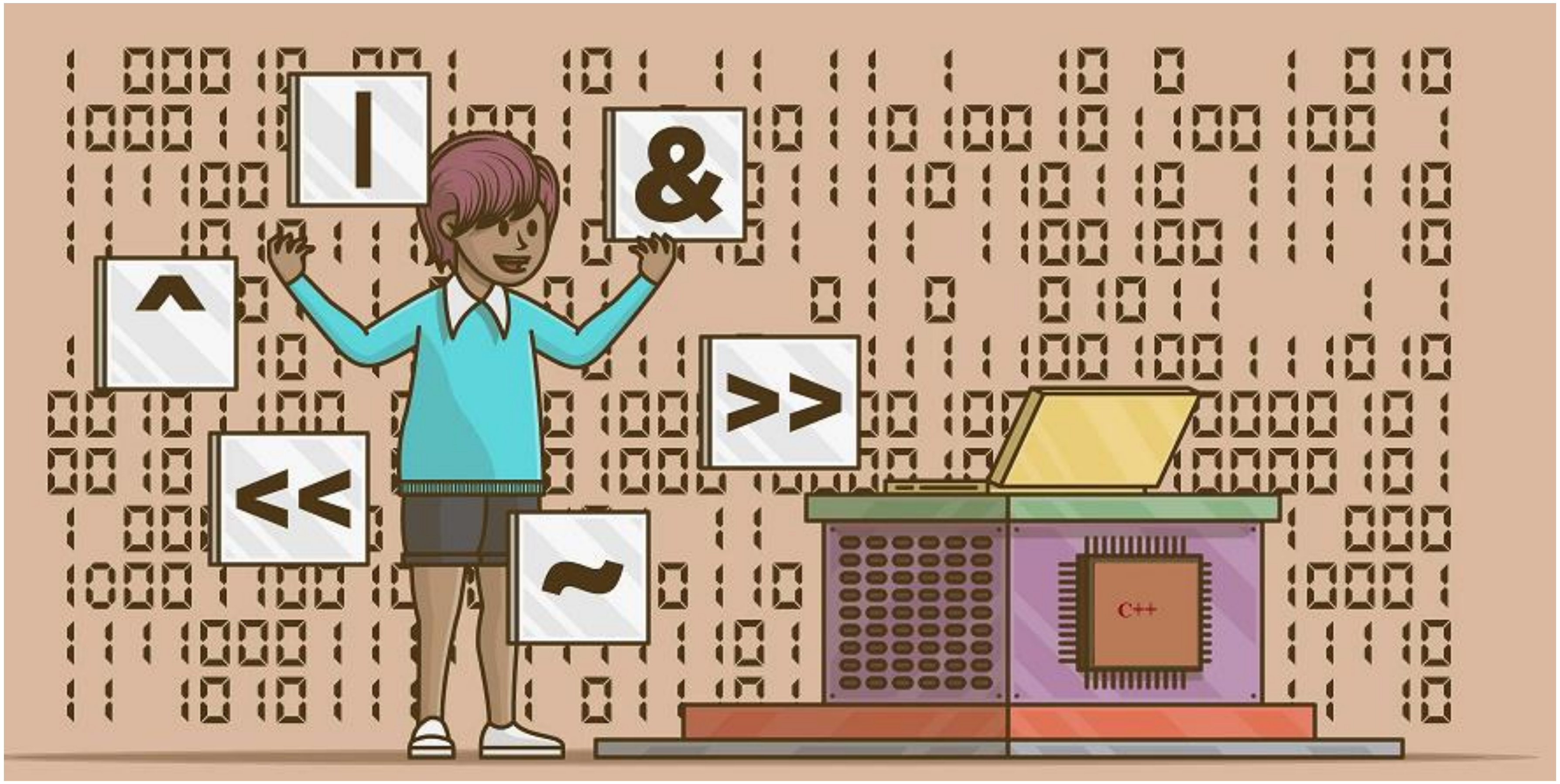
- Float can store 6 digit after decimal points
- Double can store 15 digit after decimal points
- By default decimal will store double means

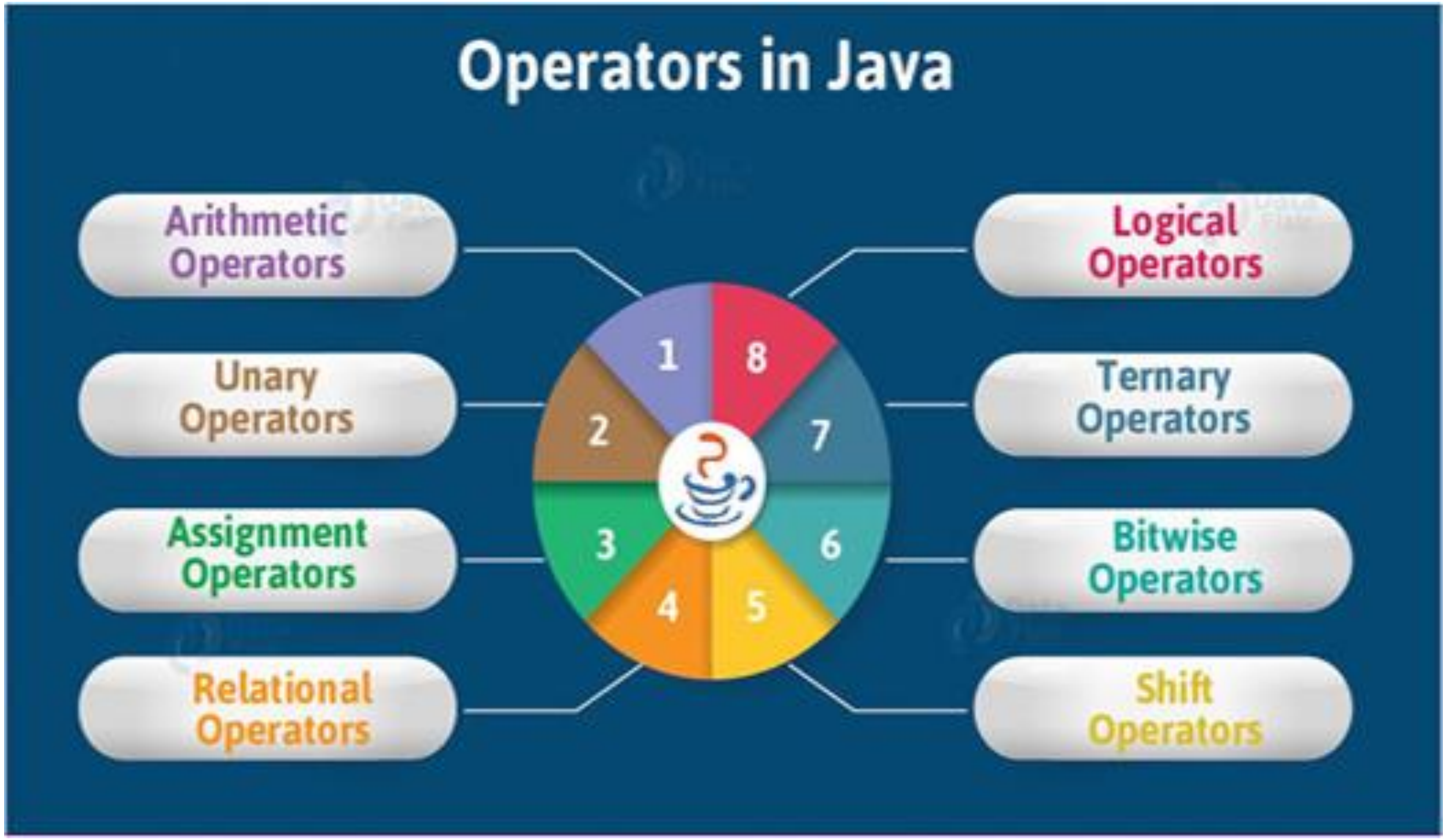
If we write Pi=3.14

It will store double datatype

```
public class Main
{
        public static void main(String[] args)
        {
                float  a=10.123456789f;
                 double b=10.123456789;
                 System.out.println(a);
                 System.out.println(b);
        }
}
```

Operators




Operators in Java
Arithmetic Operators
Unary Operators
Assignment Operators
Relational Operators
Logical Operators
Ternary Operators
Bitwise Operators
Shift Operators
1
2
3
4
5
6
7
8

Arithmetic operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Bitwise operators

Operator	Description
&	AND Operator
|	OR Operator
^	XOR Operator
~	Ones Complement Operator
<<	Left Shift Operator
>>	Right Shift Operator

What will be output?

```
class MainClass
{
    public static void main(String []args)
    {
            int a=12,b=25;
            System.out.println(a&b);
    }
}
```

A. 4
B. 6
C. 8
D. 10

AND Operator (&)

If both side bit is on result will be On

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
			1	0	0	0	8

a & b=

01100 (12)

11001   (25)

01000   (8) Ans.

OR Operator (|)

If any side bit is on result will be On

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
		1	1	1	0	1	29

a | b=

01100 (12)

11001 (25)

11101 (29) Ans.

XOR Operator (^)

If both side bit is opposite result will be On

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Steps to solve:-

- **a = 12 (find binary form:1100)**
- **b = 25 (find binary form:11001)**

How to find Binary:

64	32	16	8	4	2	1	
		0	1	1	0	0	12
		1	1	0	0	1	25
		1	0	1	0	1	21

a ^ b=

01100 (12)

11001 (25)

10101 (21) Ans.

Left Shift Operator(<<)

a=10 (1010)

a<<1

1010.0

10100(20) Ans.

a<<2

1010.00

101000(40) Ans.

Right Shift Operator(>>)

a=10 (1010)

a>>1

1010.

101(5) Ans.

a>>2

1010.

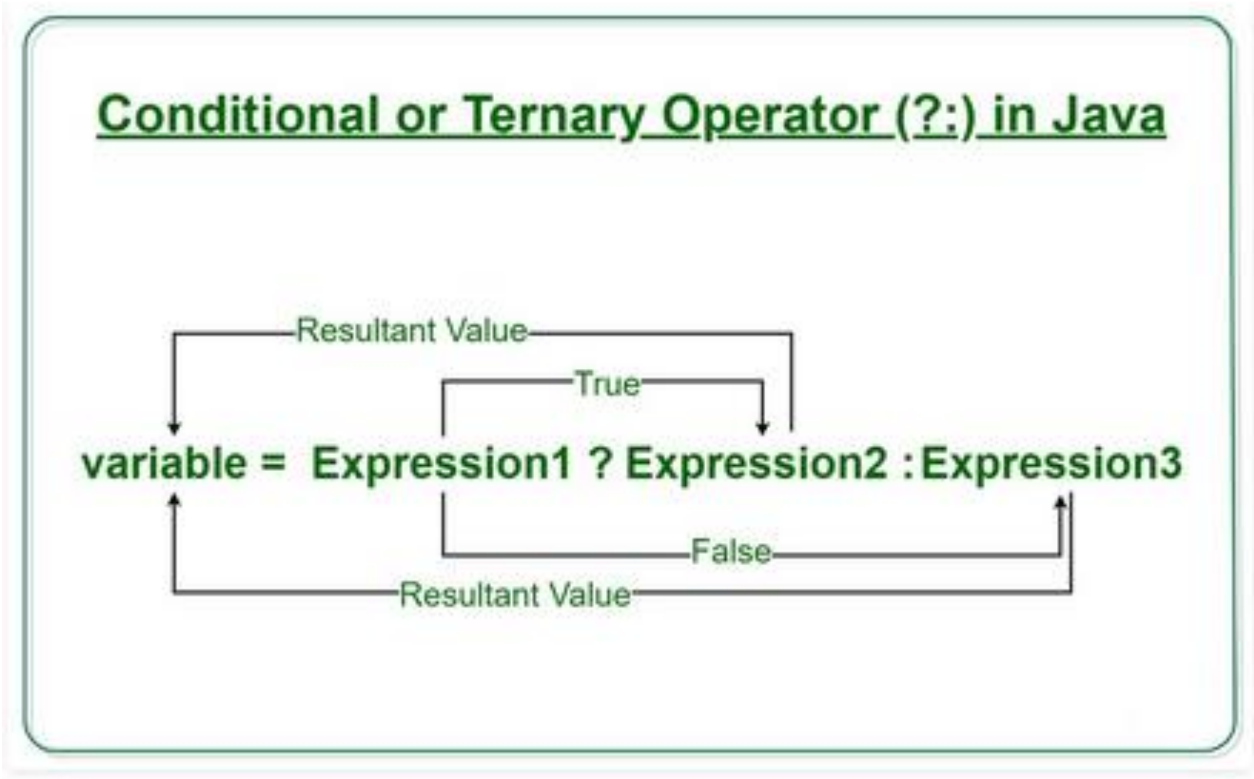
10(2) Ans.

Unary Operators in Java

Java unary operators are the types that need only one operand to perform any operation like increment, decrement, negation, etc. It consists of various arithmetic, logical and other operators that operate on a single operand.

Ternary Operator

Java ternary operator is the only conditional operator that takes three operands. It's a one-liner replacement for if-then-else statement and used a lot in Java programming.


Conditional or Ternary Operator (?:) in Java
Resultant Value
True
variable = Expression1 ? Expression2 : Expression3
False
Resultant Value

Control Statements:

- ???

if/else constructs

```
If(condition)
{
// statement execute when condition true
}
else
{
// statement execute when condition false
}
```

switch statement

To select choices/options from user it used:

Switch with choice (integer value like: 1,2,..)

Switch with choice (character value like: A,B,..)

Switch with choice (String value like: "ADD","SUB",..)

looping controls, nested loops

- While loop
- Do while loop
- For loop
- Enhanced or advanced for loop
- Nested loops means one loop inside another loop.

Enhanced For loop

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

Example:

```
int myArray[]=new int[]{11,12,13,14,15};
for(int num : myArray)
{
System.out.println(num);
}
```

Does Java support goto?

- No but keyword is there

inner and nested classes

Define a class within another class, such classes are known as *nested* classes

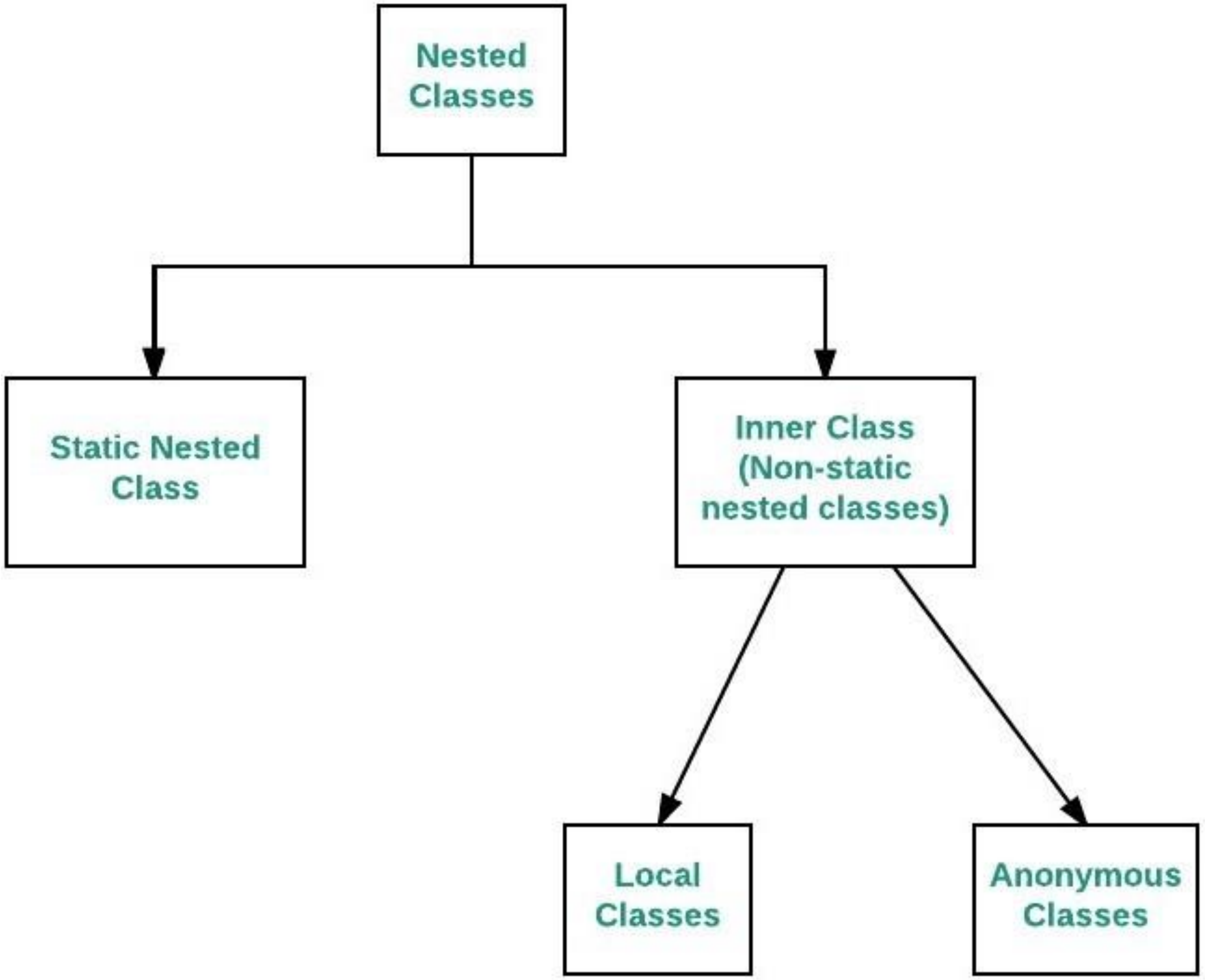
Nested classes are divided into two categories:

static nested class :

Nested classes that are declared static are called static nested classes.

inner class :

An inner class is a non-static nested class.


Nested Classes
Static Nested Class
Inner Class (Non-static nested classes)
Local Classes
Anonymous Classes

```
class Outer{

        static class inner
            {

          public void getData()
          {
             int myArray[]=new
int[]{11,12,13,14,15};
              for(int num:myArray)
              {
               System.out.println(num);
               }
          }
        }
}
```

```
public class Main
{
public static void main(String[] args)
           {
              Outer.inner in=new
Outer.inner();
              in.getData();

           }
}
```

What will be output?

```
static class Outer{

        class inner
            {
            public void getData()
           {
 int myArray[]=new int[]{11,12,13,14,15};
               for(int num:myArray)
               {
                System.out.println(num);
                }
           }
        }
}
```

```
public class Main
{
public static void main(String[] args)
          {
              Outer.inner in=new
Outer.inner();
              in.getData();
          }
}
```

Difference between Normal inner class and Static nested class

- In normal inner class, we cannot declare any static members but in the static nested class, we can declare a static member including the main method.
- Since we cannot declare the main method in the normal inner class, therefore, we cannot run inner class directly from the command prompt. But we can declare the main method and can also run the static nested class directly from the command prompt.
- A normal inner class can access both static and non-static members of the outer class directly but from the static nested class, we can access only static members.

Array

- Array is a collection of similar type of elements that have contiguous memory location.
- In java, array is an object the contains elements of similar data type.
- It is a data structure where we store similar elements. We can store only fixed elements in an array.
- Array is index based, first element of the array is stored at 0 index.

Advantage of Array

Code Optimization: It makes the code optimized, we can retrieve or sort the data easily.

Random access: We can get any data located at any index position.

Disadvantage of Array

Size Limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

Types of Array: There are two types of array.

- Single Dimensional Array
- Multidimensional Array-
  - 2D array
  - 3D array
  - Jagged array

Single Dimensional Array

- The syntax for declaring and instantiating an array:

There are two ways to declare an array,

```
type[] arrayName;
type arrayName[];
```

- How to instantiate an array

arrayName = **new** type[length];

- How to declare and instantiate an array in one statement

type[] arrayName = **new** type[length];

Examples

- **Array of integers**

```
int[] num = new int[5];
```

- **Array of Strings**

```
String[] nameList = new String[5];
nameList[0] = "Amanda Green";
nameList[1] = "Vijay Arora";
nameList[2] = "Sheila Mann";
nameList[3] = "Rohit Sharma";
nameList[4] = "Mandy Johnson";
```

Enhanced For loop

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

Example:

```
int []myArray=new int[]{11,12,13,14,15};
for(int num : myArray)
{
System.out.println(num);
}
```

Array length

- The syntax for getting the length of an array

```
arrayName.length
e.g-
int[] values = new int[10];
for (int i = 0; i < values.length; i++)
{
values[i] = i;
}
```

What will be output?

```
public class Main
{
        public static void main(String[] args)
        {
                int arr[]={010,102,17};
                for(int i=0;i<arr.length;i++)
                {
                  System.out.print(arr[i]+” “);
                }
        }
}
```

A. 10 102 17
B. 010 102 17
C. 8 102 17
D. Error

Two-dimensional arrays

The syntax for creating a rectangular array-

type[][] arrayName = **new** type[rowCount][columnCount];

- A statement that creates a 3x2 array

int[][] numbers = **new int**[3][2];

- *3x2* array and initializes it in one statement

int[][] numbers =new int[][] { { 1, 2 }, { 3, 4 }, { 5, 6 } };

```
public class Main
{
            public static void main(String[] args)
            {
            int array_variable [] = new int[10];

               for (int i = 0; i < 10; i++)
           {
           array_variable[i] = i;
           System.out.print(array_variable[i] + " ");
           i++;
         }
            }
}
```

a) 0 2 4 6 8
b) 1 3 5 7 9
c) 0 1 2 3 4 5 6 7 8 9
d) 1 2 3 4 5 6 7 8 9 10

Enhanced for loop for 2D array

```
for (int[] num: arr)
 {
        for(int data: num)
   {
          System.out.println(data);
        }
}
```

Jagged array

```
type[][] arrayName = new
   type[rowCount][];
e.g:-int num[][]=new int[4][];
num[0]=new int[1];
num[1]=new int[2];
num[2]=new int[3];
num[3]=new int[4];
```

Enhanced for loop for 2D array

```
for (int[] num: arr)
 {
       for(int data: num)
   {
         System.out.println(data);
       }
}
```

3 d array:

Syntax:

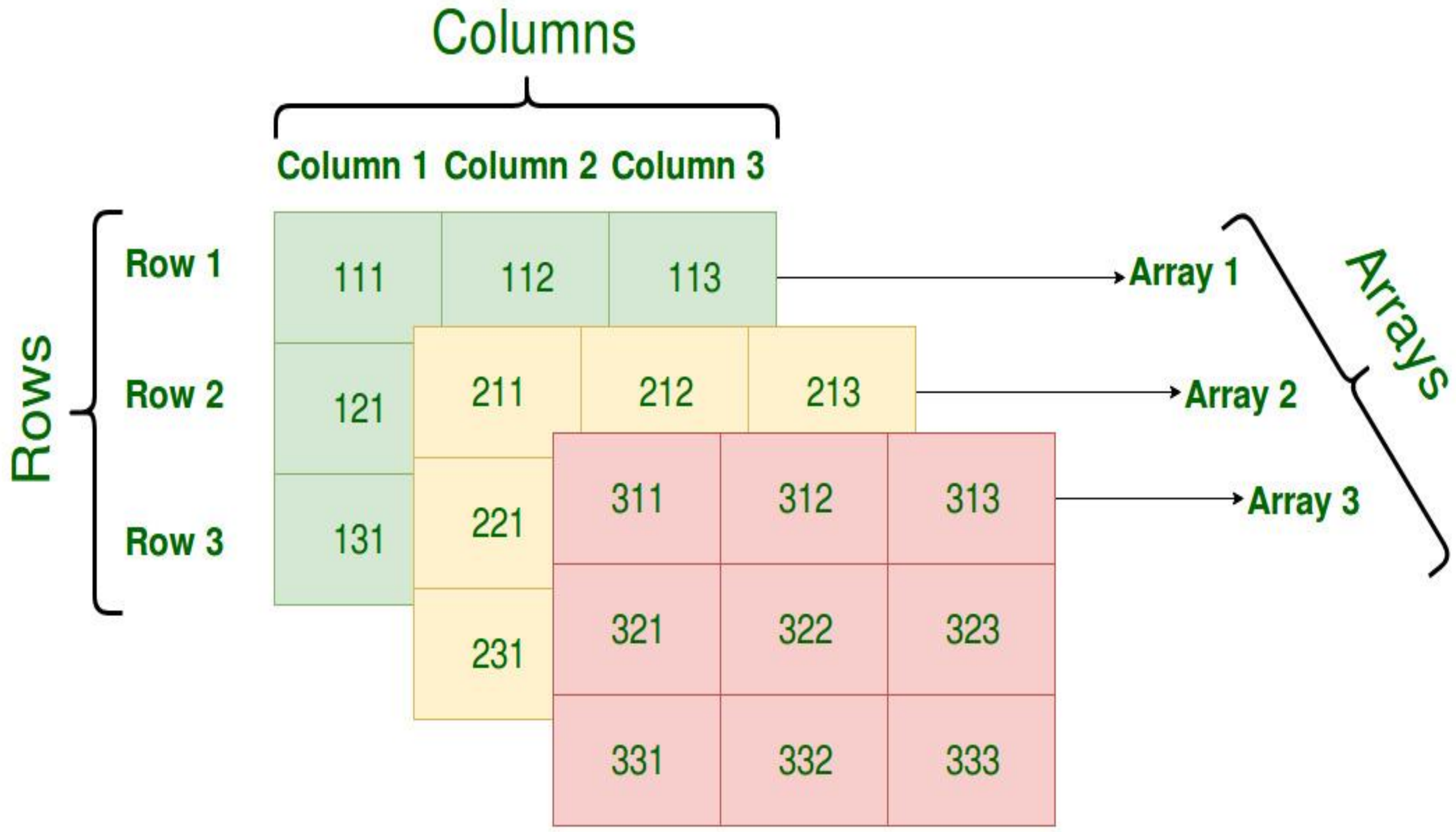
array_type[][][] array_name = new array_type[x][y][z];

Ex:

int[][][] num=new int[2][3][4];

Here, num[i][j][k] where ‘i’ is the array number, ‘j’ is the row number and ‘k’ is the column number.

int[][][] num=new int[3][3][3];



For example find exam scores obtained by three students of each department in 3 different subjects.

Electronics department:

student1 scores: 75, 87, 69

student2 scores: 90, 87, 85

student3 scores: 56, 67, 76

Computer Science department:

student1 scores: 78, 67, 75

student2 scores: 87, 98, 76

student3 scores: 67, 56, 65

Information Technology department:

student1 scores: 72, 63, 72

student2 scores: 82, 91, 71

student3 scores: 64, 56, 66

To store all these exam scores, department-wise, we will need to use three-dimensional array

```
int[ ][ ][ ] scores = new int[3][3][3];
```

4 D Array

- Array of 3 D Array

```
int [][][][] num=new int[2][2][2][2];
```

Multi dimensional array means array of arrays.

String Class

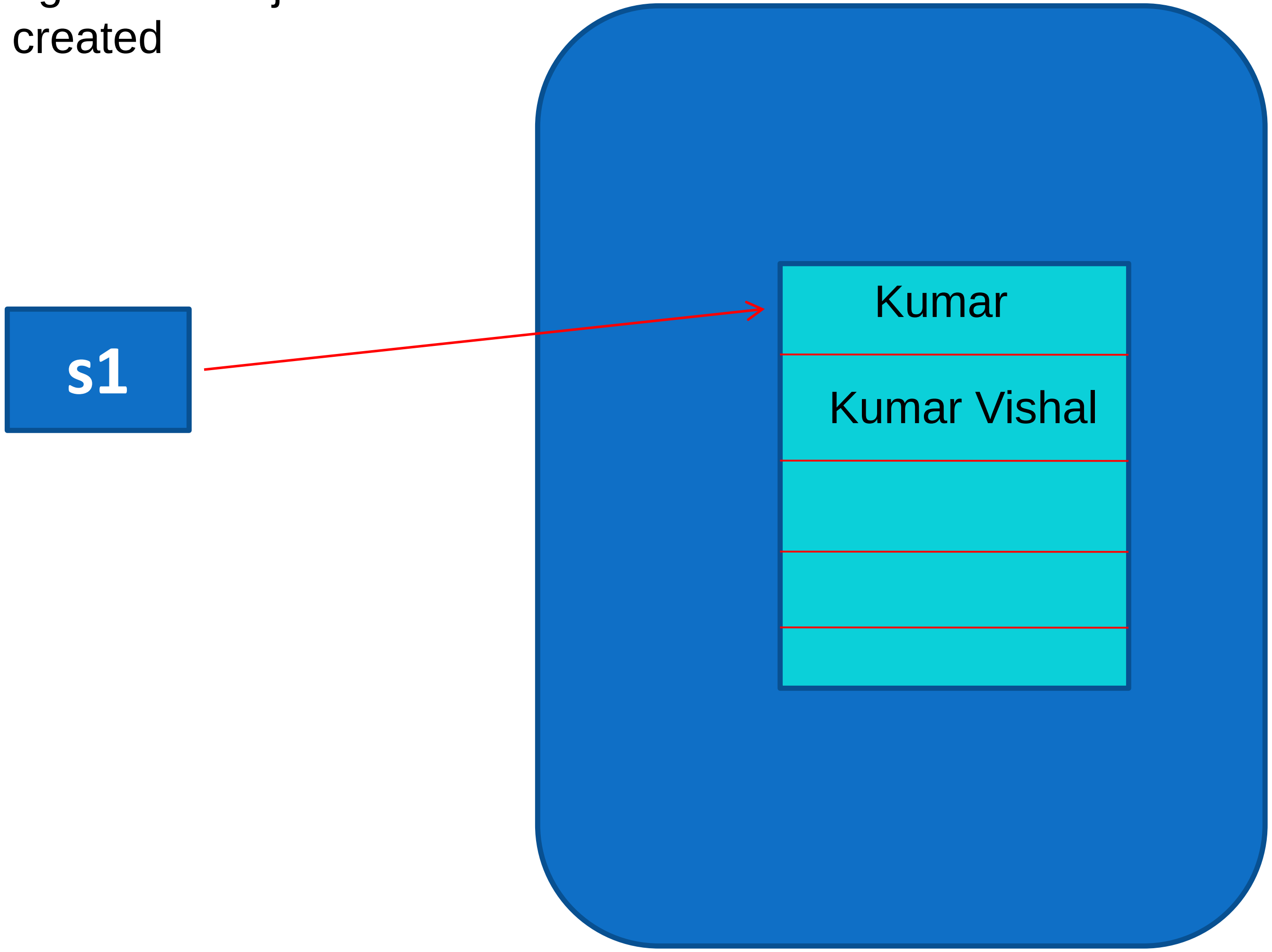
- String is a sequence of characters. But in Java, string is an object that represents a sequence of characters.
- The java.lang.String class is used to create a string object.
- In java, String objects are **immutable** which means a constant and cannot be changed once created.

string is immutable in java:

```
public static void main(String args[]){
   String s=“Kumar";
   s.concat(" Vishal");
   System.out.println(s);
 }
```

Output: Kumar

Here value not changed new object
Kumar Vishal has created



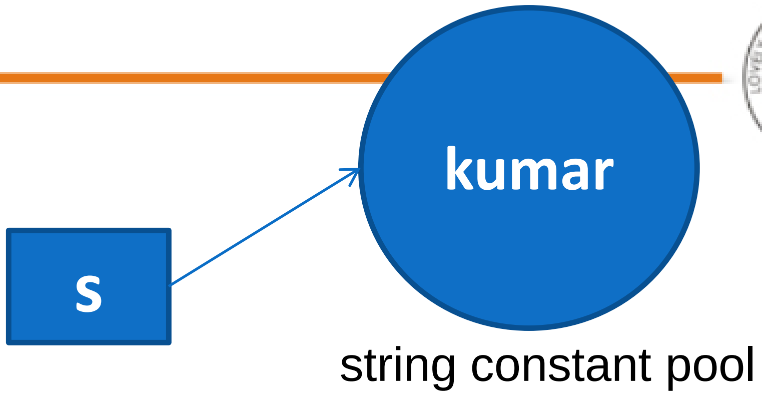
Why string objects are immutable in java?

Because java uses the concept of string literal. Suppose there are 5 reference variables,all referes to one object “kumar".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java.

There are two ways to create String object:

- By string literal
- By new keyword

string literal:



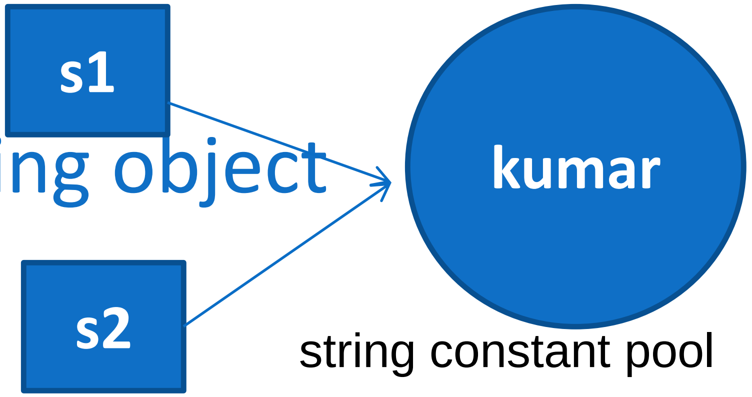
String s="kumar";

Each time you create a string literal, the JVM checks the "string pool" first. If the string already exists in the pool, a reference to the pooled object is returned. If the string doesn't exist in the pool, a new string object is created and placed in the pool.

String s1="kumar";

String s2="kumar";

//It doesn't create a new string object



By new keyword:

String s=**new** String(“kumar")

In this case, JVM will create a new string object in normal (non-pool) heap memory, and the literal “kumar" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

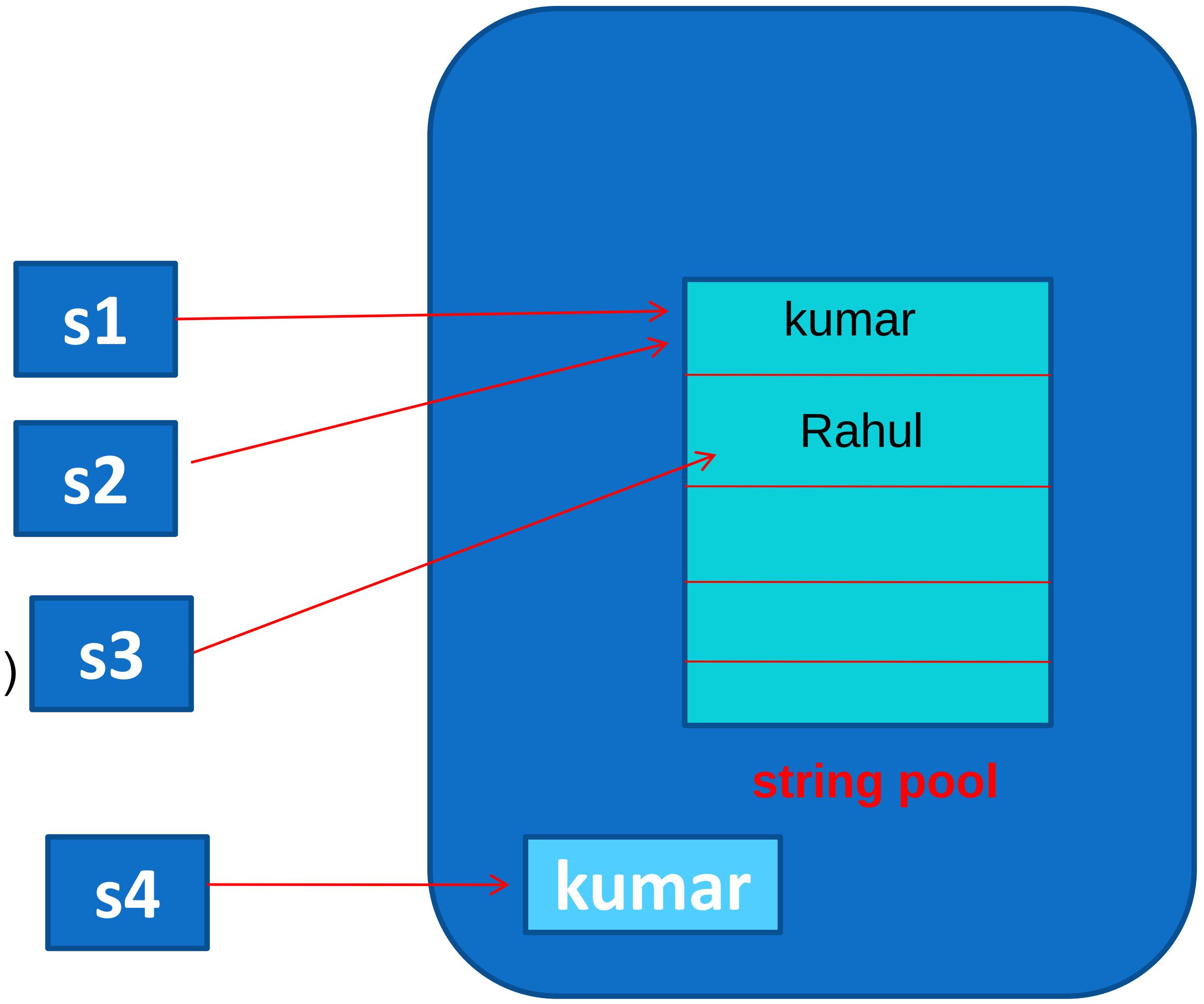
Concept:---

String s1="kumar"

String s2="kumar"

String s3= "Rahul"

String s4=new String("kumar")



s1==s2 // true
s1==s3//false
s1==s4//false

Heap

Note:- Here s1,s2,s3 and s4 are references of string object kumar, Rahul, kumar

Methods in String class:

- length(),
- charAt()
- Substring()
- concat
- indexOf()
- equals()
- compareTo()
- trim()
- replace()
- toUpperCase()
- toLowerCase();

length(), charAt()

int length(); — Returns the number of characters in the string

char charAt(i); — Returns the char at position i.

Character positions in strings are numbered starting from 0 – just like arrays.

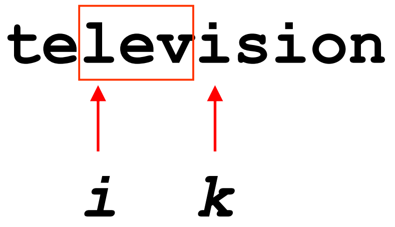
Returns:

```
”Problem".length();      →  7
”Window".charAt (2);     →  ’n'
```

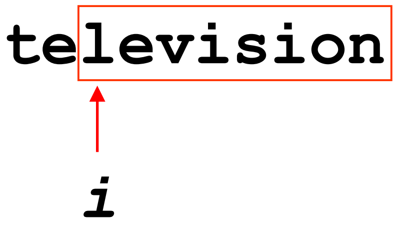
Substring()

Returns a new String by finding characters from an existing String.

- String subs = word.**substring** (i, k);
  - returns the substring of chars in positions from **i** to **k-1**



- String subs = word.**substring** (i);
  - returns the substring from the **i**-th char to the end



Returns:

```
"television".substring (2,5);   → "lev"
"immutable".substring (2);      → "mutable"
```

Concatenation()

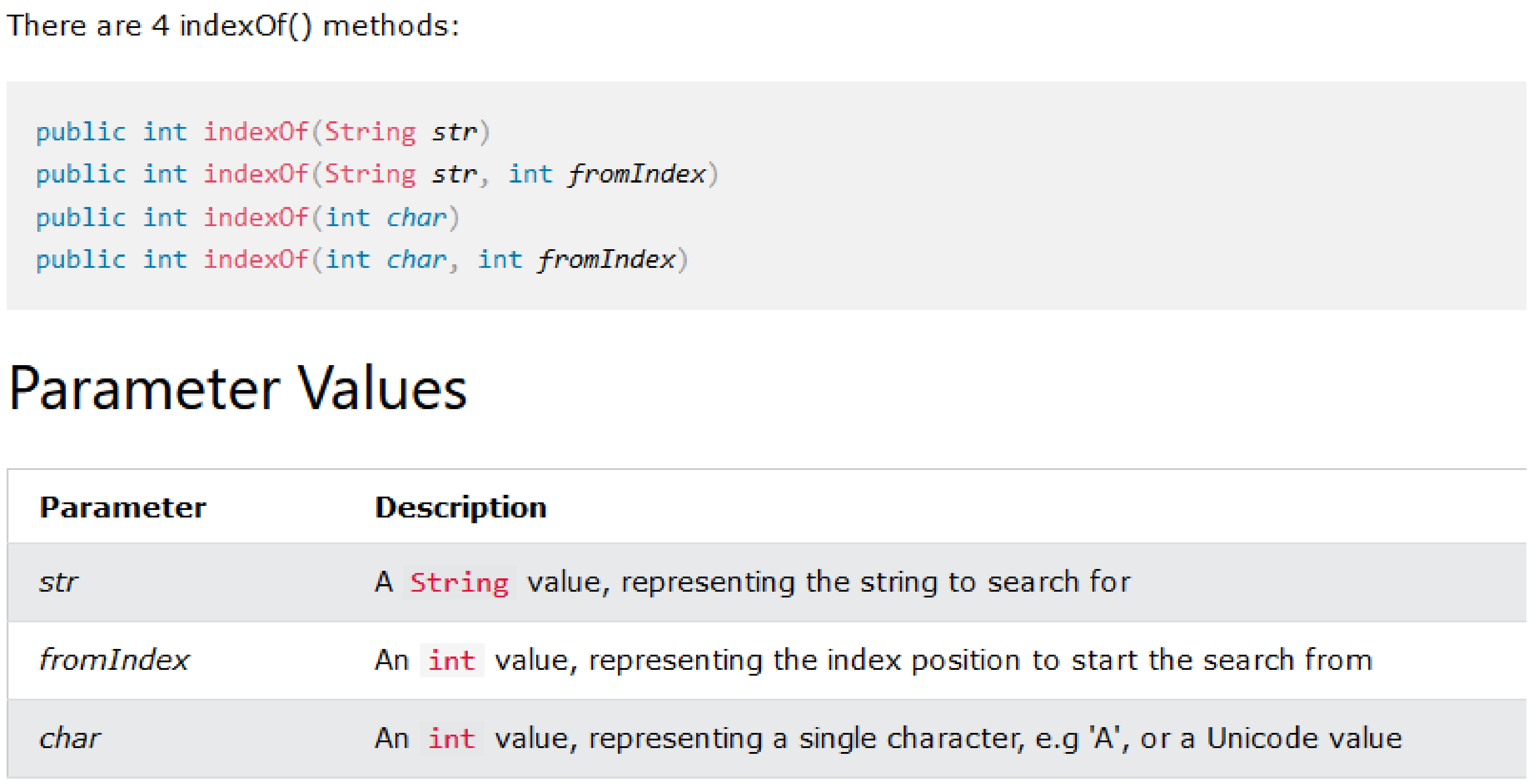
String word1 = “re”, word2 = “think”; word3 = “ing”;

int num = 2;

- String result = word1.**concat** (word2);

  //the same as word1 + word2 “rethink“

indexOf():The indexOf() method returns the position of the first occurrence of specified character(s) in a string.



There are 4 indexOf() methods:

```
public int indexOf(String str)
public int indexOf(String str, int fromIndex)
public int indexOf(int char)
public int indexOf(int char, int fromIndex)
```

Parameter Values

Parameter	Description
str	A `String` value, representing the string to search for
fromIndex	An `int` value, representing the index position to start the search from
char	An `int` value, representing a single character, e.g 'A', or a Unicode value

indexOf()

index positions

```
String name ="President George Washington";
```

	Returns:
name.indexOf ('P');	0
name.indexOf ('e');	2
name.indexOf ("George");	10
name.indexOf ('e', 3);	6
name.indexOf ("Bob");	−1

(starts searching at position 3)

(not found)

equals()/equalsIgnoreCase()

boolean b = word1.**equals**(word2);

returns **true** if the string **word1** is equal to **word2**

boolean b = word1.**equalsIgnoreCase**(word2);

returns **true** if the string **word1** matches **word2**, case-blind

```
b = “Raiders”.equals(“Raiders”);//true
b = “Raiders”.equals(“raiders”);//false
b = “Raiders”.equalsIgnoreCase(“raiders”);//true
```

Difference between == and .equals() method in Java

- equals() is a method and == is a operator
- use == operators for reference comparison (**address comparison**) and .equals() method for **content comparison**.
- In simple words, == checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects

What will be output?

```
String s1 = new String("HELLO");
String s2 = new String("HELLO");
System.out.print(s1 == s2);
System.out.print(s1.equals(s2));
```

A. true true
B. false false
C. true false
D. false true

compareTo()

int diff = word1.**compareTo**(word2);

returns the “difference” **word1 – word2**

- **if** string1 > string2, it returns positive number
- **if** string1 < string2, it returns negative number
- **if** string1 == string2, it returns 0

The **java string compareTo()** method compares the given string with current string . It returns positive number, negative number or 0.

It compares strings on the basis of Unicode value of each character in the strings.

Comparison Examples

```
//negative differences
diff = “apple”.compareTo(“berry”);//a before b
diff = “Zebra”.compareTo(“apple”);//Z before a
diff = “dig”.compareTo(“dug”);//i before u
diff = “dig”.compareTo(“digs”);//dig is shorter
```

```
//zero differences
diff = “apple”.compareTo(“apple”);//equal
diff = “dig”.compareToIgnoreCase(“DIG”);//equal
```

```
//positive differences
diff = “berry”.compareTo(“apple”);//b after a
diff = “apple”.compareTo(“Apple”);//a after A
diff = “BIT”.compareTo(“BIG”);//T after G
diff = “huge”.compareTo(“hug”);//huge is longer
```

Methods — Changing Case

String word2 = word1.**toUpperCase**();
String word3 = word1.**toLowerCase**();

returns a new string formed from **word1** by converting its characters to upper (lower) case

```
String word1 = “HeLLo“;
String word2 = word1.toUpperCase();//”HELLO”
String word3 = word1.toLowerCase();//”hello”
//word1 is still “HeLLo“
```

trim()

removing white space at both ends

does not affect whites space in the middle

Example:

```
String word1 = “ Hi Kumar “;
String word2 = word1.trim();
//word2 is “Hi Kumar” – no spaces on either end
//word1 is still “ Hi Kumar “ – with middle spaces
```

replace()

method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Syntax:

String replace(**char** oldChar, **char** newChar)

String replace(CharSequence target, CharSequence replacement)

Example:

```
String str1="hello hello hello";
 String str2="hello hello hello";
 str1=str1.replace('h', 'H');
 str2=str2.replace("hello", "hi");
 System.out.println(str1);
 System.out.println(str2);
```

replaceAll()

- The **java string replaceAll()** method returns a string replacing all the sequence of characters matching regular expression and replacement string.

Syntax:

String replaceAll(String regex, String replacement)

Example:

replace all occurrences of white spaces in a string:

```
String str = "how to do in java provides java reading materials";
String newStr = str.replaceAll(“\\s", "");
System.out.println(newStr);
```

What will be output?

```
String str = "how to do in java provides java reading materials";

String newStr = str.replaceAll("\\s", "9");
System.out.println(newStr);
```

What will be output for the following program?

```
public class MainClass
{
   public static void main(String[] args)
   {
       String s1="one";
       s1.concat("two");
       System.out.println(s1);
   }
}
```

A. one
B. two
C. onetwo
D. Compiler error

StringBuffer class

- StringBuffer is mutable means one can change the value of the object .
- The object created through StringBuffer is stored in the heap .
- each method in StringBuffer is synchronized that is StringBuffer is thread safe due to this it does not allow two threads to simultaneously access the same method . Each method can be accessed by one thread at a time .

Differences between StringBuffer and StringBuilder

StringBuffer	**StringBuilder**
StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.

methods of StringBuffer/StringBuilder class

- ***append()***
- ***capacity()***
- **ensureCapacity()**
- ***insert()***
- **reverse()**
- **replace()**
- **length()**
- **delete()**
- **deleteCharAt()**
- **substring()**

append()

- is used to append the string from one string to another string like concat.

Syntax:

StringBufferClassReference.append(any type)

Any type:-

append(char), append(boolean), append(int), append(float), append(double) etc.

capacity()

- is used to return the current capacity of buffer.
- The default capacity of the buffer is 16.
- If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.
- For example if your current capacity is 16, it will be (16*2)+2=34.

Condition1:

```
StringBuffer sb=new StringBuffer();
System.out.println("Current Capacity:"+sb.capacity());
// Current Capacity:16
```

Condition2:

```
StringBuffer sb=new StringBuffer(“hello”);
System.out.println("Current Capacity:"+sb.capacity());
// Current Capacity:21
```

ensureCapacity()

- It ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2.

Ex:

If current capacity is:70

sb.ensureCapacity(70); // no change

But

sb.ensureCapacity(71); // cahnge now 142

insert()

- It is used to inserts the string at the specified position.

Syntax:

StringBufferClassReference.insert(pos,string)

reverse()

It is used to reverses the current string

Syntax:

StringBufferClassReference.reverse()

- **replace():**

replaces the string from the specified startingIndex and endingIndex.

Syntax:

StringBufferClassReference.replace(startingIndex, endingIndex, newstring)

e.g:

```
StringBuffer sb=new StringBuffer("Hello");
sb.replace(1,3,“kumar");
System.out.println(sb);//Hkumarlo
```

- **length()** : to find the length of current string

Syntax:

StringBufferClassReference.length()

- **delete():** deletes the string from the specified startingIndex to endingIndex.

Syntax: StringBufferClassReference.delete(startingIndex,endingIndex)

e.g:

```
StringBuffer sb=new StringBuffer("Hello");
sb.delete(1,3);
System.out.println(sb);//Hlo
```

- **deleteCharAt():**

deletes the character at the index specified by *loc.*

Syntax:

```
StringBufferClassReference.deleteCharAt(int loc)
```

e.g:

```
StringBuffer sb=new StringBuffer("Hello");
sb.deleteCharAt(3);
System.out.println(sb);//Helo
```

substring()

- is used to return the substring from the specified startingIndex and endingIndex.

Syntax:

substring(int startingIndex, int endingIndex)

How to define class?

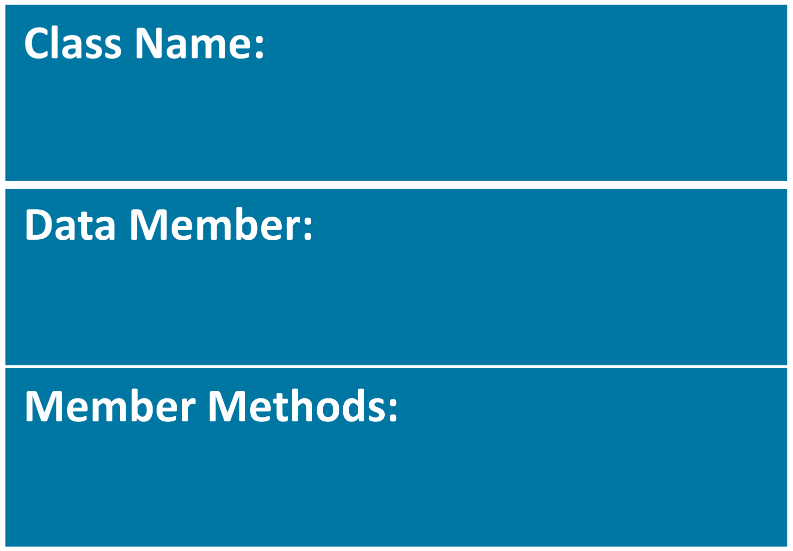
- Class is a collection of data members and member methods.
- Class is a collection of similar type of objects.

Syntax to declare a class:

```
class <class_name>
{
data_member;
member_method;
}
```

Defining scope of
The class

Class Diagram(UML)



access control in Java

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

- A **Java constructor** name must exactly match with the class name (including case).
- A Java constructor must not have a return type.
- If a class doesn't have a constructor, Java compiler automatically creates a default constructor during run-time. The default constructor initialize instance variables with default values. For example: int variable will be initialized to 0
- Constructors cannot be abstract or static or final.
- Constructor can be overloaded but can not be overridden.

Constructor

Types:

- No-Arg Constructor - a constructor that does not accept any arguments
- Default Constructor - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
- Parameterized constructor - used to specify specific values of variables in object

overview of inheritance

One class is hiring properties from another class is called **inheritance**.

Advantages: Reusability

```
Class ClassA
{


}
Class  ClassB extends ClassA
{
}
```

How to achieve encapsulation in JAVA?

Ans:

Using data member as a private and setter, getter accessor methods as a public to access these private data members.

Inheritance Types:

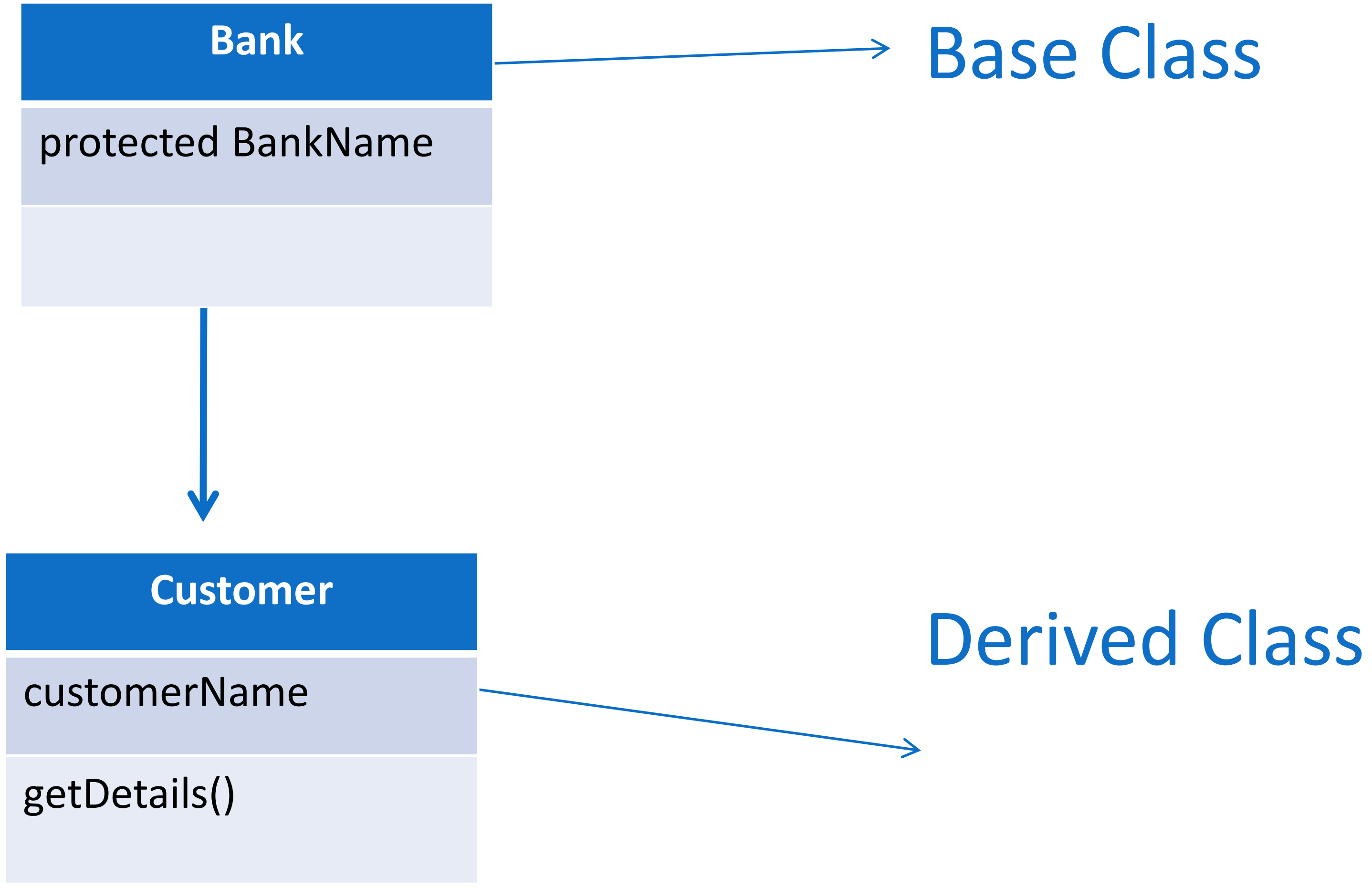
- Single
- Multilevel
- Hierarchical

Using interface it is possible:

- Hybrid
- Multiple

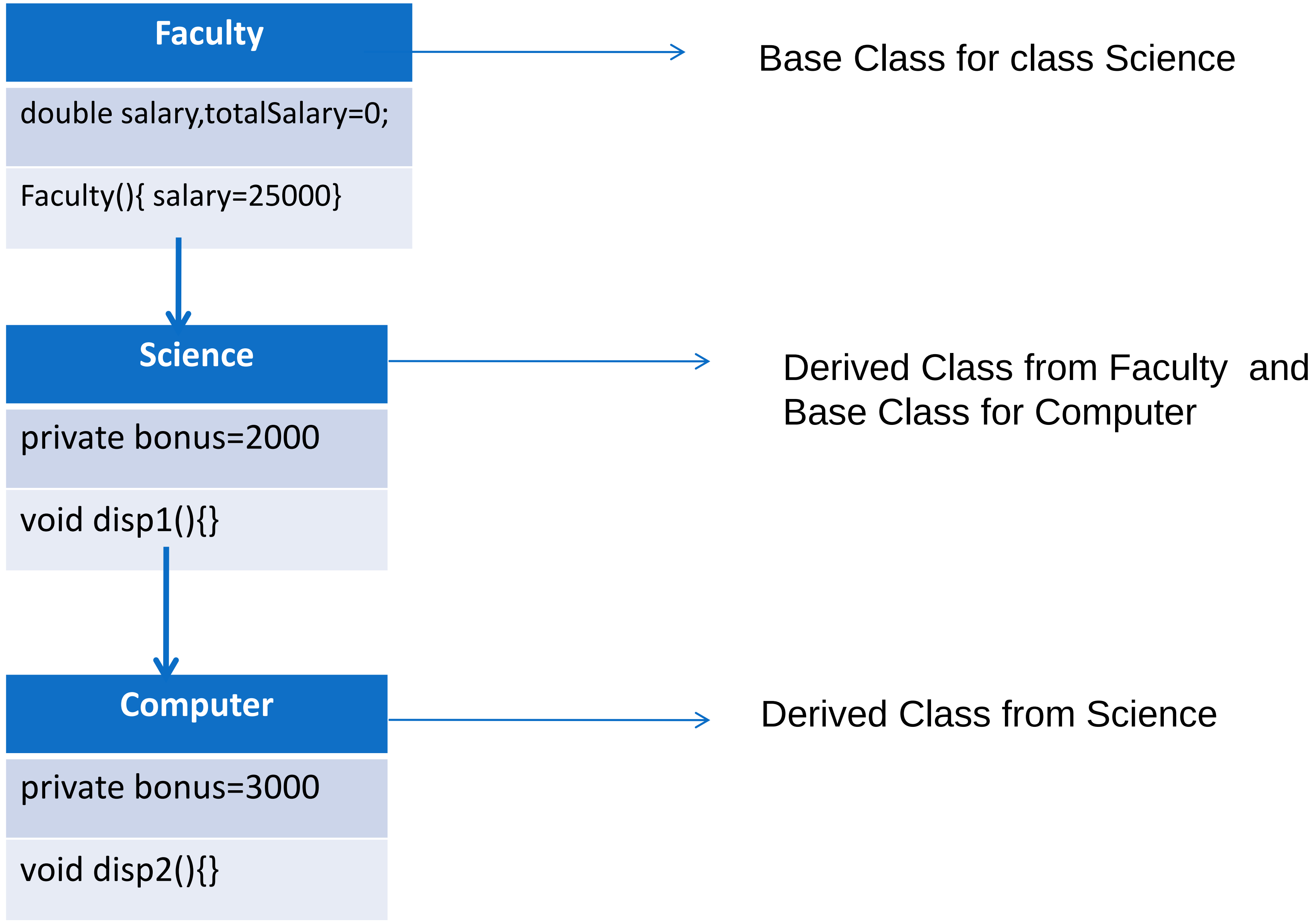
Single inheritance:

- One base class and one derived class
- One-to-one relationship



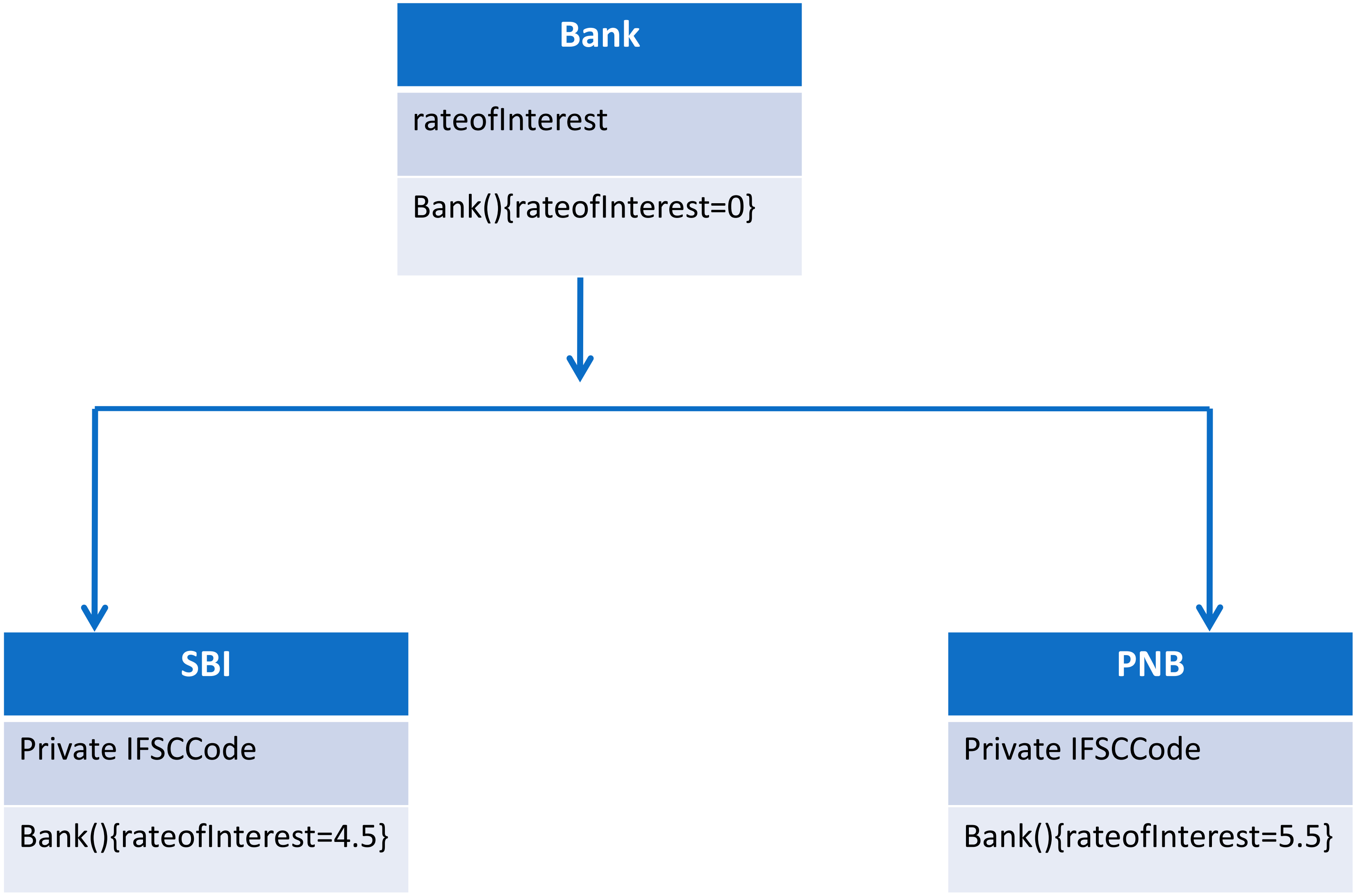
Multilevel inheritance:

- One class is going to become base class for other derived class


Faculty
double salary,totalSalary=0;
Faculty(){ salary=25000}
Base Class for class Science
Science
private bonus=2000
void disp1(){}
Derived Class from Faculty and Base Class for Computer
Computer
private bonus=3000
void disp2(){}
Derived Class from Science

Hierarchical inheritance:

- One base class and multiple derived class, one –to-many relationship



Important Notes:-

Default parent class constructor automatically called by child class but for calling parameterized constructor of base class we need to use super keyword.

Super keyword

- super keyword can be used to access the immediate parent class constructor.
- super keyword can be used to invoke immediate parent class members and methods.
- In some scenario when a derived class and base class has same data members or methods, in that case super keyword also used.

interface

- Collection of abstract methods.
- Use keyword interface
- Achieve multiple inheritance
- Use implements keyword for calling in derive class

Syntax:

```
interfcae Bank
{
 int rateOfInterest();
}
class  SBI implements Bank {
 int rateOfInterest()
{
retutn 5;
}
}
```

Abstract Class

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- A class which is declared with the **abstract** keyword is known as an abstract class in Java.
- It holds both abstract and non-abstract methods.
- It can have constructor also
- We can achieve abstraction by using either abstract class or interface.
- It can’t be instantiated
- The abstract class can also be used to provide some implementation of the interface.

Q: Difference between abstract class and interface

Ans:---????