AKTU

B. Tech II-Year

Gateway Classes

Semester -IV

CS IT & Allied Branches

BCS403-Object Oriented Programming with Java

UNIT-1 : Programming Structures in Java

Gateway Series for Engineering

- Topic Wise Entire Syllabus
- Long - Short Questions Covered
- AKTU PYQs Covered
- DPP
- Result Oriented Content

Download App

For Full Courses including Video Lectures

AKTU

B. Tech II-Year

Gateway Classes

BCS403- Object Oriented Programming with Java

Unit-1

Introduction to Programming Structures in Java

Syallbus

Introduction: Why Java, History of Java, JVM, JRE, Java Environment, Java Source File Structure, and Compilation. Fundamental,

Programming Structures in Java: Defining Classes in Java, Constructors, Methods, Access Specifies, Static Members, Final Members, Comments, Data types, Variables, Operators, Control Flow, Arrays & String.

Object Oriented Programming: Class, Object, Inheritance Super Class, Sub Class, Overriding, Overloading, Encapsulation, Polymorphism, Abstraction, Interfaces, and Abstract Class.

Packages: Defining Package, CLASSPATH Setting for Packages, Making JAR Files for Library Packages, Import and Static Import Naming Convention For Packages

Download App

For Full Courses including Video Lectures

AKTU 4th SEM

CS IT & Allied Branches

OOPs with JAVA

Enhance your Skills

All Universities

Dr. Sandeep Kumar Gupta

- ✓ Basic to Advance
- ✓ Theory Concepts with Practical Examples
- ✓ DPP (Exercise/ Assignments)
- ✓ Placement Oriented Subject

Object Oriented Programming with Java
SYLLABUS

Unit – 1

Introduction: Why Java, History of Java, JVM, JRE, Java Environment, Java Source File Structure, and Compilation. Fundamental,

Programming Structures in Java: Defining Classes in Java, Constructors, Methods, Access Specifies, Static Members, Final Members, Comments, Data types, Variables, Operators, Control Flow, Arrays & String.

Object Oriented Programming: Class, Object, Inheritance Super Class, Sub Class, Overriding, Overloading, Encapsulation, Polymorphism, Abstraction, Interfaces, and Abstract Class.

Packages: Defining Package, CLASSPATH Setting for Packages, Making JAR Files for Library Packages, Import and Static Import Naming Convention For Packages processor evolution and types, microprocessor architecture and operation of its components, addressing modes, interrupts, data transfer schemes, instruction and data flow, timer and timing diagram, Interfacing devices.

Object Oriented Programming with Java
SYLLABUS

Unit – 2

Exception Handling: The Idea behind Exception, Exceptions & Errors, Types of Exception, Control Flow in Exceptions, JVM Reaction to Exceptions, Use of try, catch, finally, throw, throws in Exception Handling, In-built and User Defined Exceptions, Checked and Un-Checked Exceptions.

Input /Output Basics: Byte Streams and Character Streams, Reading and Writing File in Java.

Multithreading: Thread, Thread Life Cycle, Creating Threads, Thread Priorities, Synchronizing Threads, Inter-thread Communication.

Object Oriented Programming with Java
SYLLABUS

Unit – 3

Java New Features: Functional Interfaces, Lambda Expression, Method References, Stream API, Default Methods, Static Method, Base64 Encode and Decode, ForEach Method, Try-with resources, Type Annotations, Repeating Annotations, Java Module System, Diamond Syntax with 08 Inner Anonymous Class, Local Variable Type Inference, Switch Expressions, Yield Keyword, Text Blocks, Records, Sealed Classes.

Object Oriented Programming with Java
SYLLABUS

Unit – 4

Java Collections Framework: Collection in Java, Collection Framework in Java, Hierarchy of Collection Framework, Iterator Interface, Collection Interface, List Interface, ArrayList, LinkedList, Vector, Stack, Queue Interface, Set Interface, HashSet, LinkedHashSet, SortedSet Interface, TreeSet, Map Interface, HashMap Class, LinkedHashMap Class, TreeMap Class, Hashtable Class, Sorting, Comparable Interface, Comparator Interface, Properties Class in Java.

Object Oriented Programming with Java
SYLLABUS

Unit – 5

Spring Framework: Spring Core Basics-Spring Dependency Injection concepts, Spring Inversion of Control, AOP, Bean Scopes- Singleton, Prototype, Request, Session, Application, Web Socket, Auto wiring, Annotations, Life Cycle Call backs, Bean Configuration styles

Spring Boot: Spring Boot Build Systems, Spring Boot Code Structure, Spring Boot Runners, Logger, BUILDING RESTFUL WEB SERVICES, Rest Controller, Request Mapping, Request Body, Path Variable, Request Parameter, GET, POST, PUT, DELETE APIs, Build Web Applications

History of Java

Java was originally developed by James Gosling at Sun Microsystems.

It was released in May 1995 as a core component of Sun's Java platform.

The language was initially called Oak after an oak tree that stood outside Gosling's office.

Later the project went by the name Green and was finally renamed Java, from Java coffee, a type of coffee from Indonesia.

Gosling designed Java with a C/C++-style syntax that system and application programmers would find familiar.

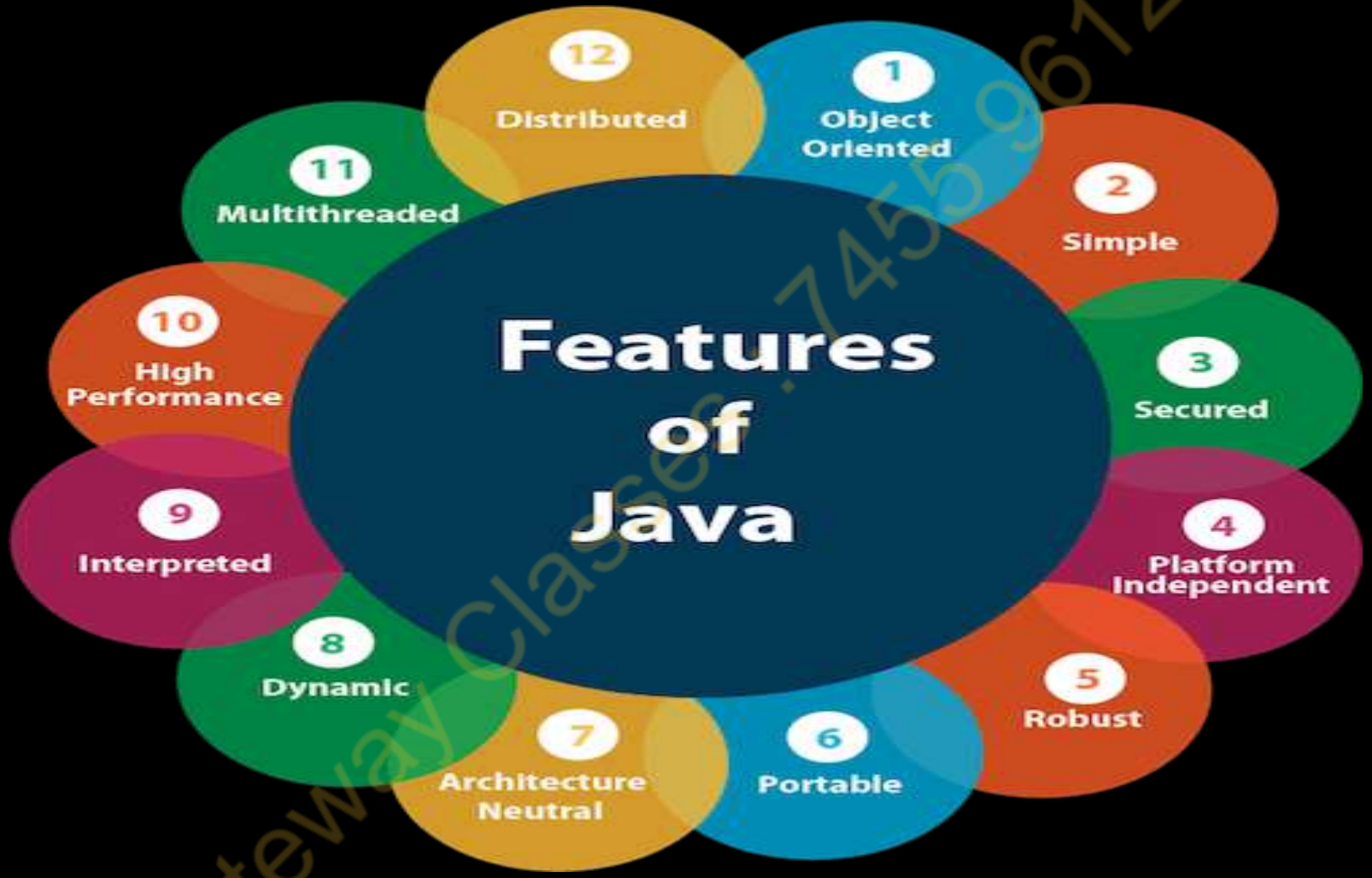
Java gained popularity shortly after its release and has been a very popular programming language since then.

Java was the third most popular programming language in 2022 according to GitHub.

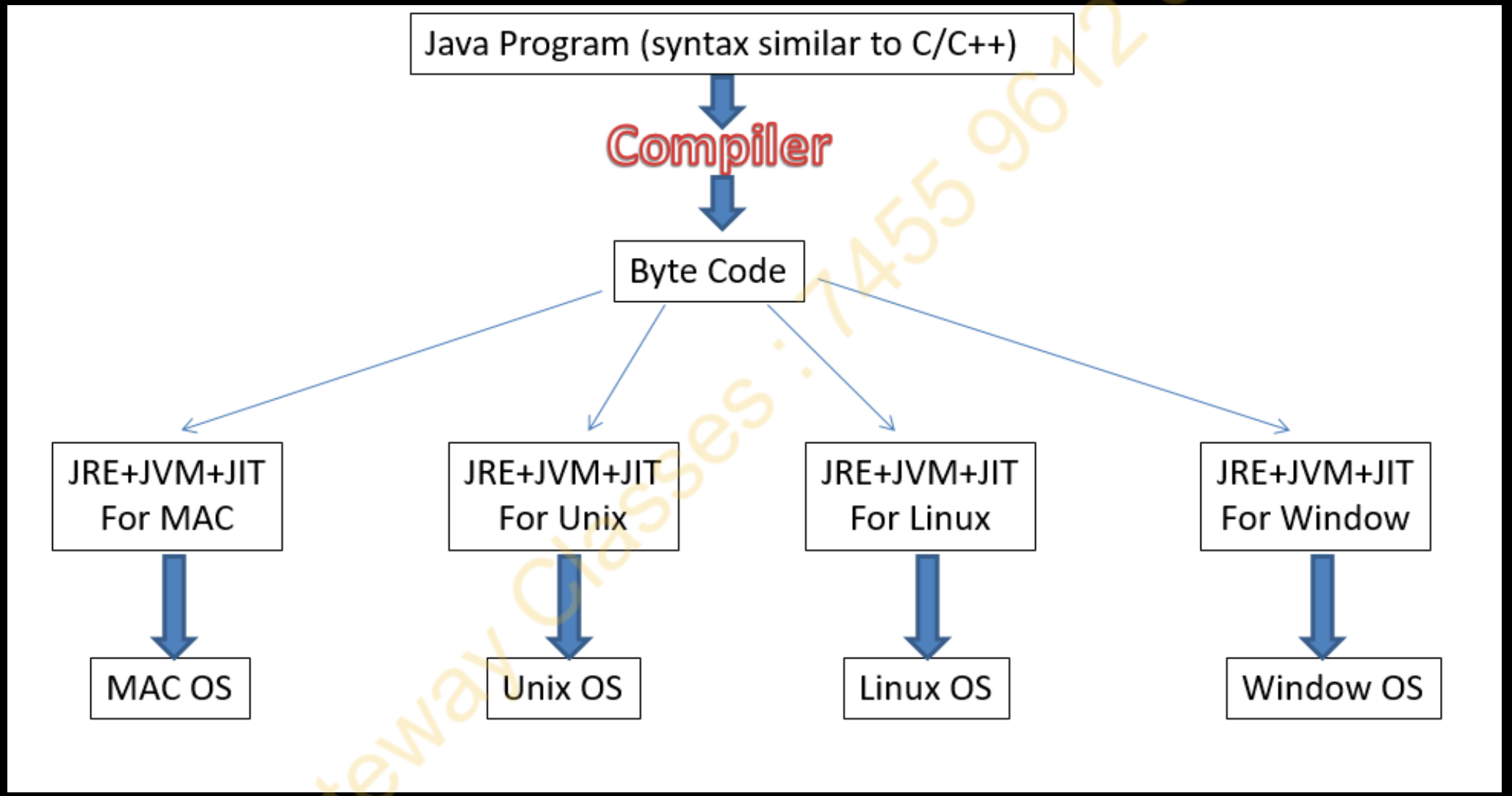
As of March 2024, Java 22 is the latest version.

Java 8, 11, 17, and 21 are previous LTS versions still officially supported.

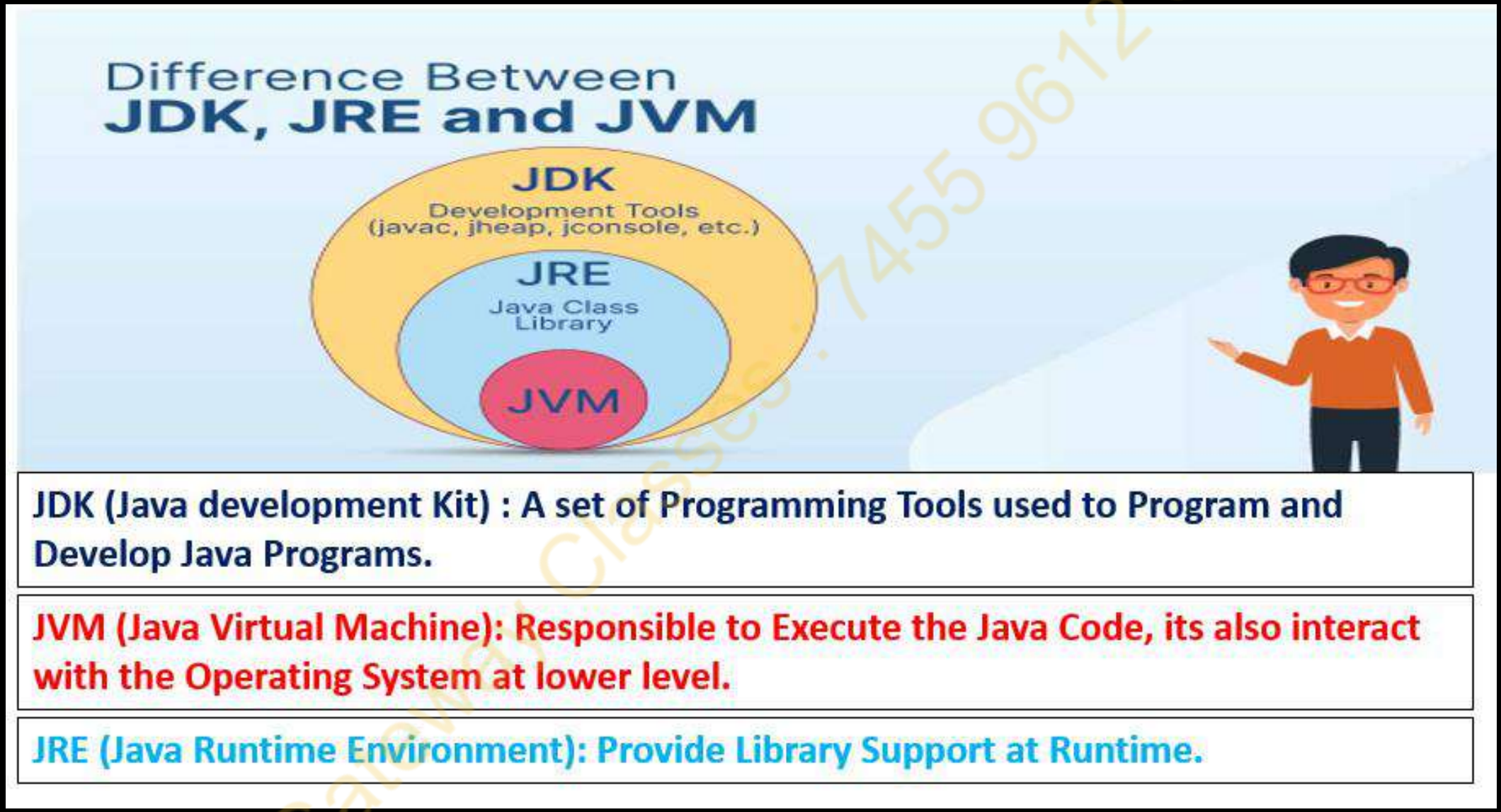
Why Java?



Environment/Architecture of Java



JDK/JVM/JRE



JDK (Java development Kit) : A set of Programming Tools used to Program and Develop Java Programs.

JVM (Java Virtual Machine): Responsible to Execute the Java Code, its also interact with the Operating System at lower level.

JRE (Java Runtime Environment): Provide Library Support at Runtime.

What we can do with Java?

Standalone Applications:

- Java standalone applications use GUI
- Components such as AWT, Swing, and JavaFX.
- These components contain buttons, list, menu, scroll panel, etc.
- It is also known as desktop alienations.

Enterprise Applications:

- An application which is distributed in nature is called an enterprise applications.

Web Applications:

- An applications that run on the server is called a web applications.
- We use JSP, Servlet, Spring, and Hibernate technologies for creating web applications.

Mobile Applications:

- Java ME is a cross-platform to develop mobile Applications which run across smartphones.
- Java is a platform for App Development in Android.

What we need to Program?

1) JDK (Java Development Kit): Just google "JDK Download"

https://www.oracle.com/java/technologies/downloads/

Linux macOS Windows

Product/file description	File size	Download
ARM64 Compressed Archive	186.57 MB	https://download.oracle.com/java/22/latest/jdk-22_linux-aarch64_bin.tar.gz (sha256)
ARM64 RPM Package	186.27 MB	https://download.oracle.com/java/22/latest/jdk-22_linux-aarch64_bin.rpm (sha256) (OL 8 GPG Key)
x64 Compressed Archive	188.29 MB	https://download.oracle.com/java/22/latest/jdk-22_linux-x64_bin.tar.gz (sha256)
x64 Debian Package	160.09 MB	https://download.oracle.com/java/22/latest/jdk-22_linux-x64_bin.deb (sha256)
x64 RPM Package	187.99 MB	https://download.oracle.com/java/22/latest/jdk-22_linux-x64_bin.rpm (sha256) (OL 8 GPG Key)

Downloading Java

Linux macOS Windows

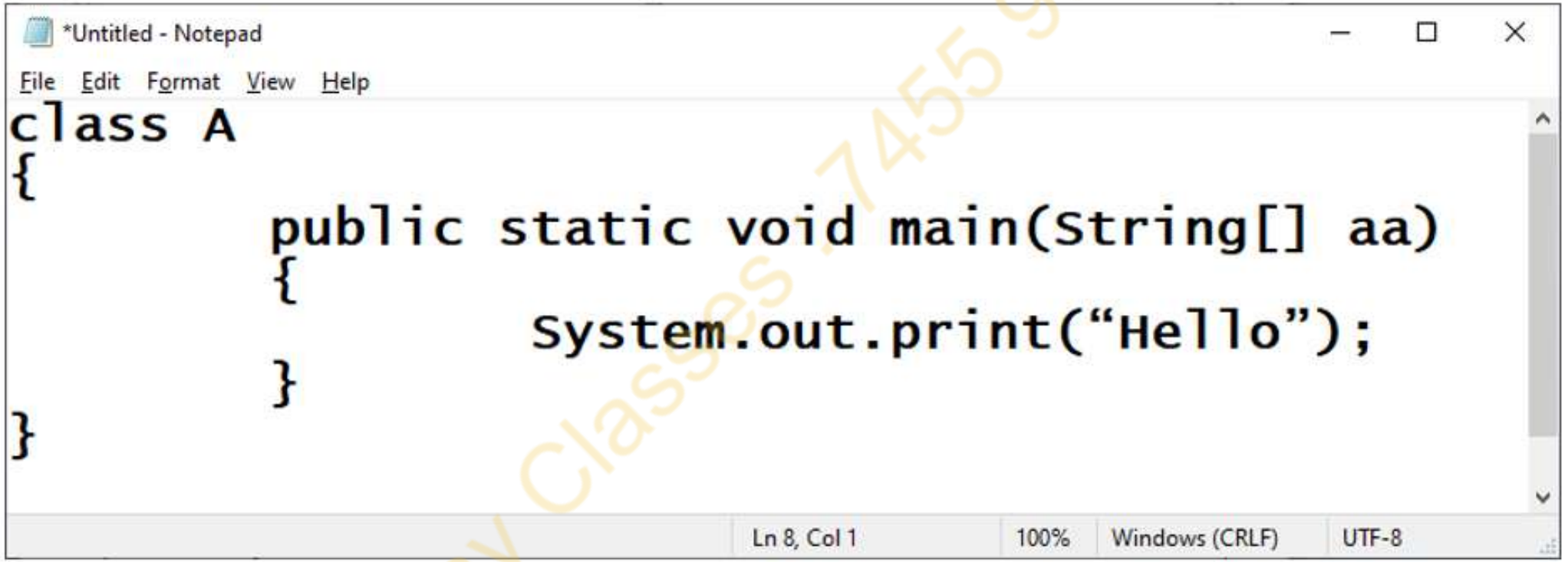
Product/file description	File size	Download
x64 Compressed Archive	185.52 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.zip (sha256)
x64 Installer	163.91 MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.exe (sha256)
x64 MSI Installer	162.07MB	https://download.oracle.com/java/22/latest/jdk-22_windows-x64_bin.msi (sha256)

Text Editor

2) A Text Editor such as

- Notepad
- TextPad
- Notepad++
- VS Code

Program Structure of Java



Practical Demonstration of

- Creating
- Compiling
- Running the Program

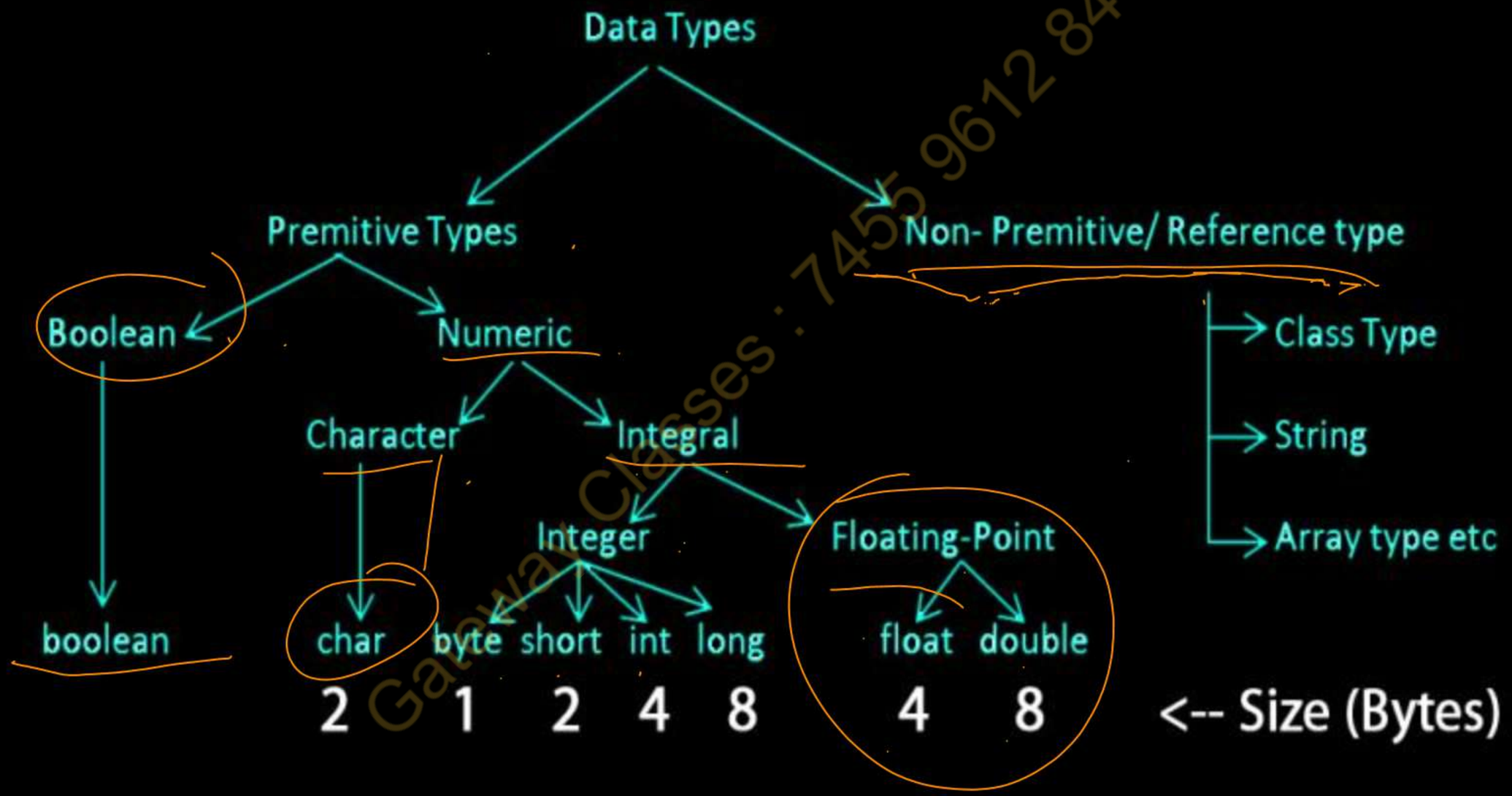
Object Oriented Programming with Java

UNIT-1
Lecture-2

What We Will Cover Today?

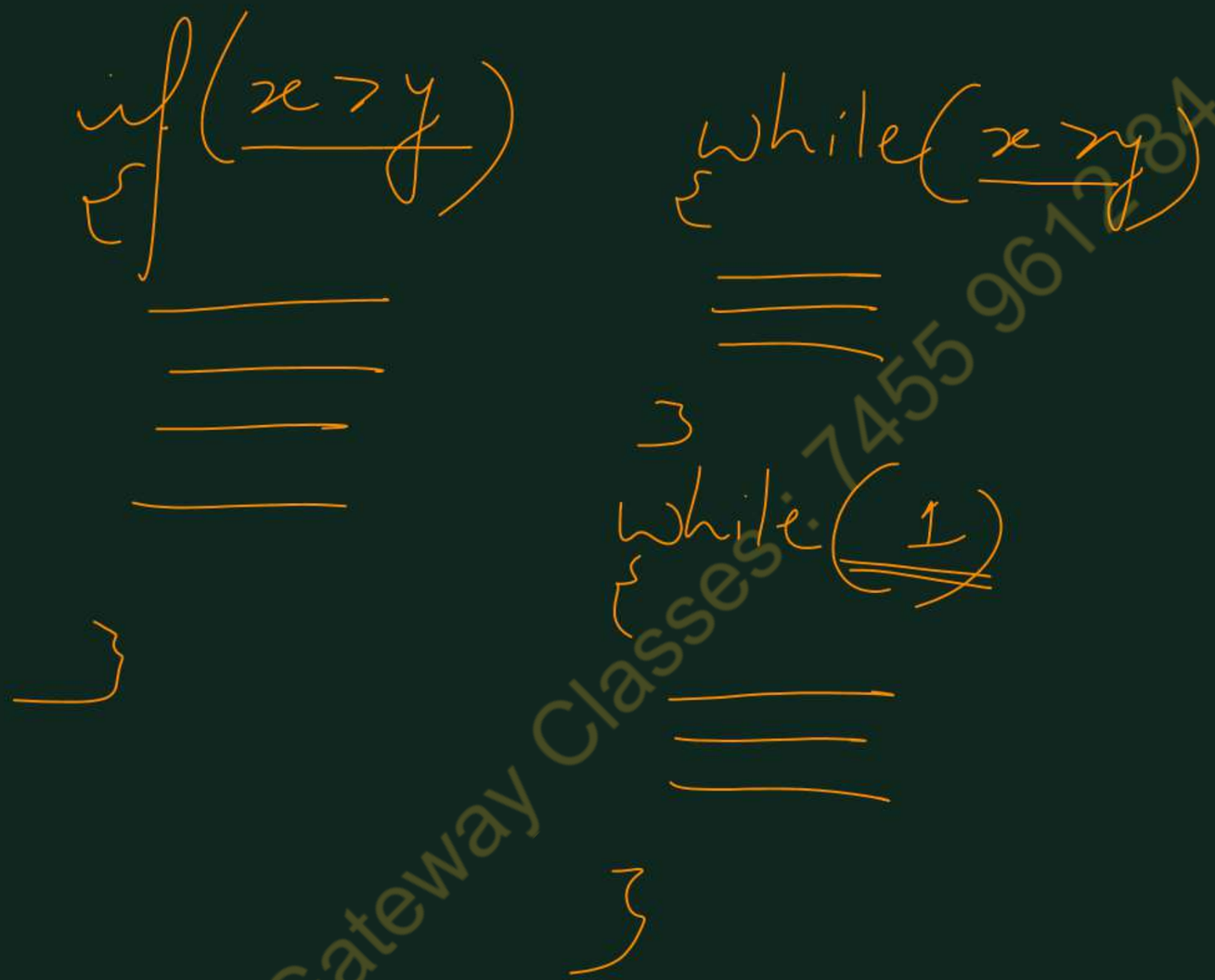
- Data Types
- Variables
- Operators
- Comments in Java

Data Types in Java



Data Types in Java

Data Type	Size (in Bytes)	Range
byte	1	-128 to +127
short	2	-32768 to +32767
int	4	-2^{31} to $2^{31}-1$
long	8	-2^{63} to $2^{63}-1$
float	4	Single Precision Floating Point Number
double	8	Double Precision Floating Point Number
char	2	Unicode Character
boolean	1 bit	only true / false



boolean x;

x = true;

x = false;

~~x = 0;~~

~~x = 1;~~

String

"Hello"

Object

String s1;

s1 = "Hello";

Object a = 5;

a = 7.5;

Data Types of Literals

By default Java use following data types for literals

25	int
25.255	double
'A'	char
"A" or "ABC"	String
45.32F or 45.32f	float
456L or 456l	long

7.9f

float ze
double ji

float x = 7.92;

byte
short
int
long
float
double

x = y;

float x = 7.92F;

double y = x; ✓ Implicit

x = y; Syntax Error

x = (float) y; Explicit Type Cast

int a = 67; 0... - 00000100 00011

int b = 300; 0 — 00100101100

byte x, y;

x = a; y = b;

x = (byte) a; 0100 0011 67

y = (byte) b; 0010 1100 44

Variables in Java

- int x;
- int x,y,z;
- double x;
- No Spaces
- First Character should be an Alphabet
- No Special Characters like +, -, , ,", $ etc
- Underscore (_) is allowed anywhere
- No Reserve Keywords
- Spaces are not allowed

Keywords in Java
(Total 50 Keywords in Current Version)

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

Operators in Java

- Arithmetic Operators
- Assignment Operator
- Unary Operators
- Relational Operators
- Bitwise and Shift Operators
- Logical Operators

Arithmetic Operators

+ Addition

e.g. 20+30 answer will be 50

- Subtraction

e.g. 100-82 answer will be 18

* Multiplication

e.g. 100*82 answer will be 8200

/ Division

e.g. 15/4 answer will be 3

e.g. 15.0/4 answer will be 3.75

e.g. 15/4.0 answer will be 3.75

e.g. 15.0/4.0 answer will be 3.75

% Remainder

e.g. 14/4 answer will be 2

Assignment Operator

=, +=, -=, *=, /=, %=

x=y;

x=x+y;	or	x+=y;
x=x-y ;	or	x-=y;
x=x*y;	or	x*=y;
x=x/y;	or	x/=y;
x=x%y;	or	x%=y;

Unary Operators

++ and --

int x=6;

x++;

x becomes 7

int y=6;

++y;

y becomes 7

int a=5;

int b;

b=++a; (Pre Increment)

b becomes 6 and a also becomes 6

int y=6;

++y;

y becomes 7

int a=5;

int b;

b=a++; (Post Increment)

b becomes 5 but a becomes 6

Relational Operators

>, >=, <, <=, ==, !=, !

e.g.

```
If(a>b)
{
   //------
}
else
{
   //------
}
```

Logical Operators

AND

OR

&& , ||

```
If(a>b && c>d)
{
//----------
}
If(a>b || c>d)
{
//--------
}
```

Bitwise and Shift Operators

~ (NOT GATE),

& (AND GATE),

| (OR GATE),

^ (XOR GATE),

>>> (Right Shift),

<< (Left shift) and

>> (Arithmetic Right Shift)

Example

int x = 65;

0 — — 00000100000 1

int y;

y = ~x; 111 — — 1011111 10

x 00011 1101

y 10101 0100

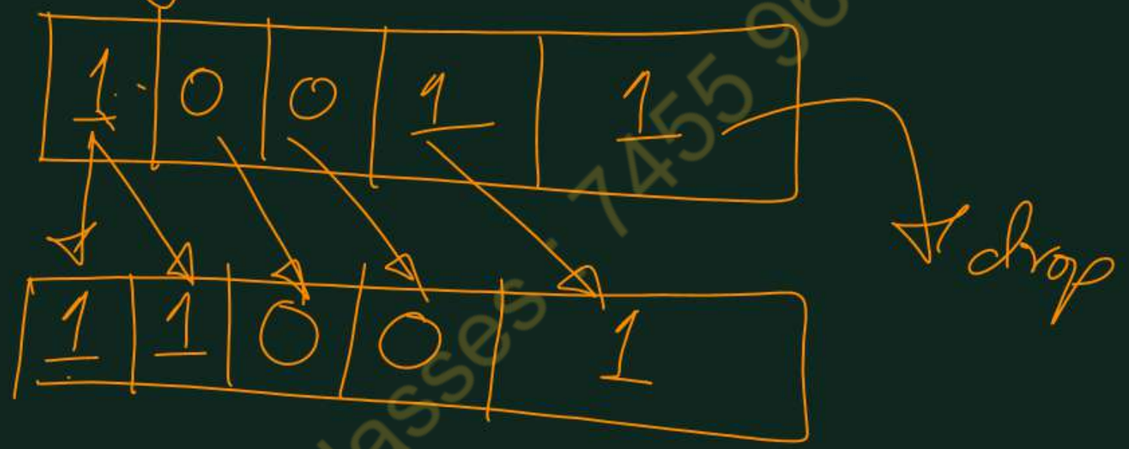
z = x & y; 00001 0100

z = x | y; 10111 1101

z = x ^ y 10110 1001

~~y = x >>> 1;~~

y = x >> 1;



Comments in Java

```
// Single Line Comments

/*
   Multiline Comments
*/

/**
   Documentation Comments
*/
```


javadoc

Object Oriented Programming with Java

UNIT-1
Lecture-3

What We Will Cover Today?

- What is Class?
- What is Object?
- Properties and Functions in Class?
- Static and Non Static Members?
- How to Write First Program in Java?

What is Class and Object ?

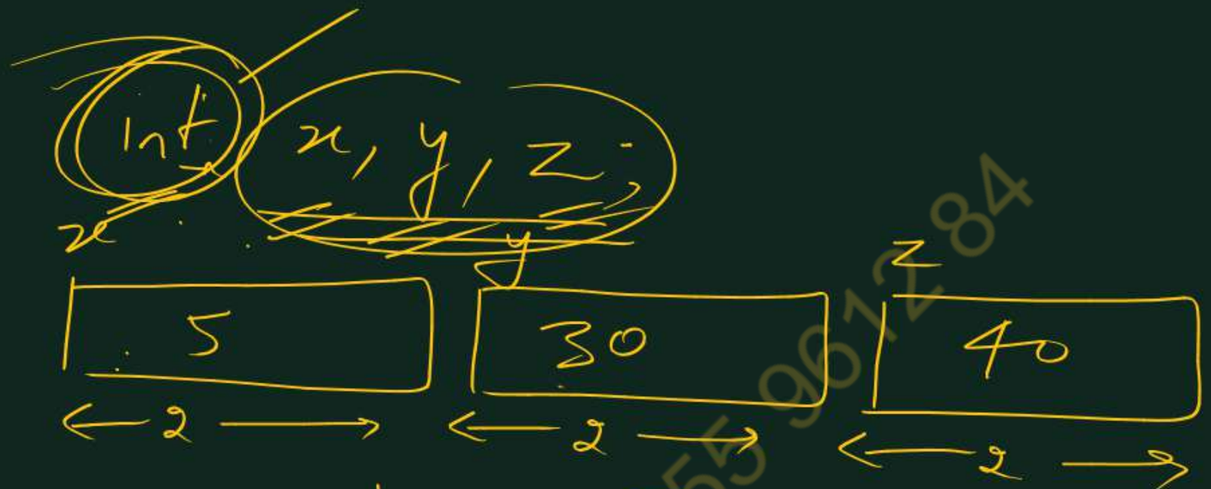
Class is a classification of Similar types of Objects. Class is virtual and Object is physical. It's a user define data type in Java similar to struct in "C" Language.

e.g.

```
class Student
{
        int rollno;
        String name;
        void study()
        {
        }
}
```

Properties

Functions



$x = 5$

$y = 30$

$z = 40$

2 bytes

+/−

−32768

+32767

Student

rollno

name

Course

~~int~~ study()

```
class Student
{
    int rollno;
    String name;
    String course;
    void study()
    {
    }
}
```

properties

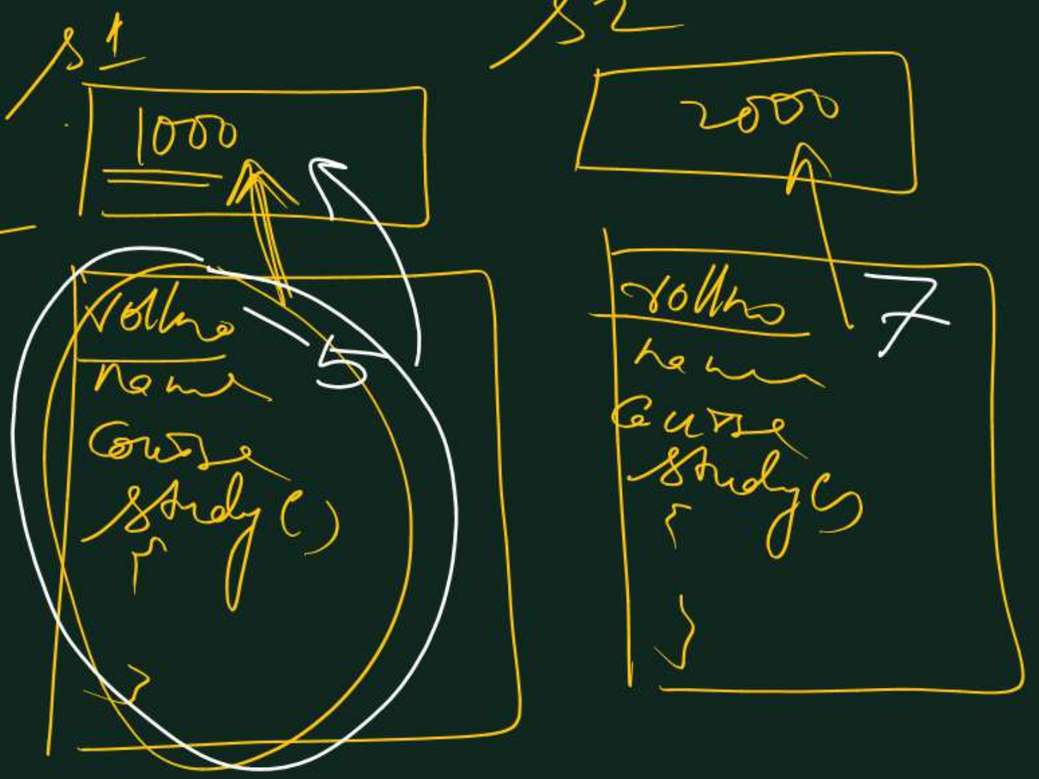
functioning

Reference Variables

```
Student s1,s2,s3;
s1=new Student();
s2=new Student();
```

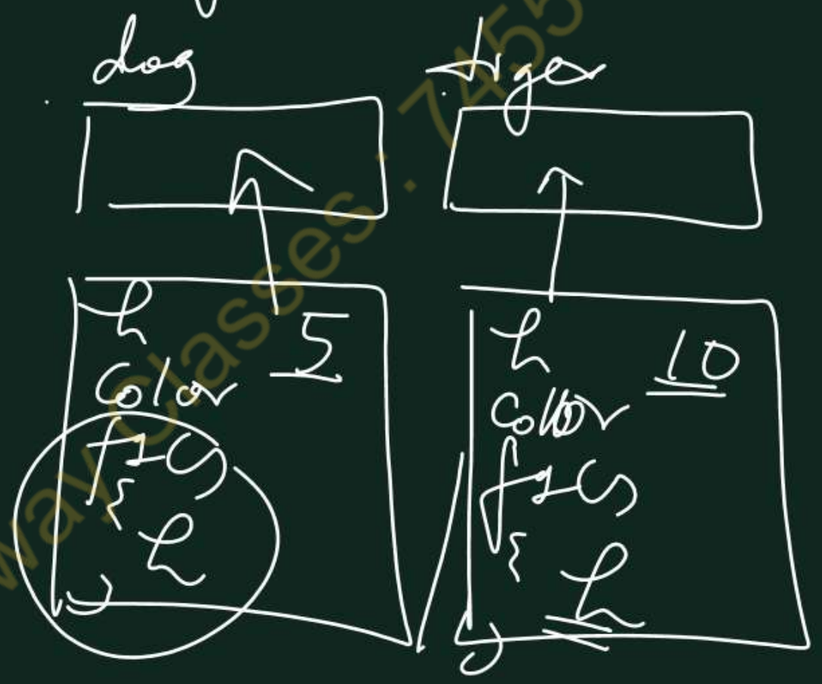
s1.rollno=5

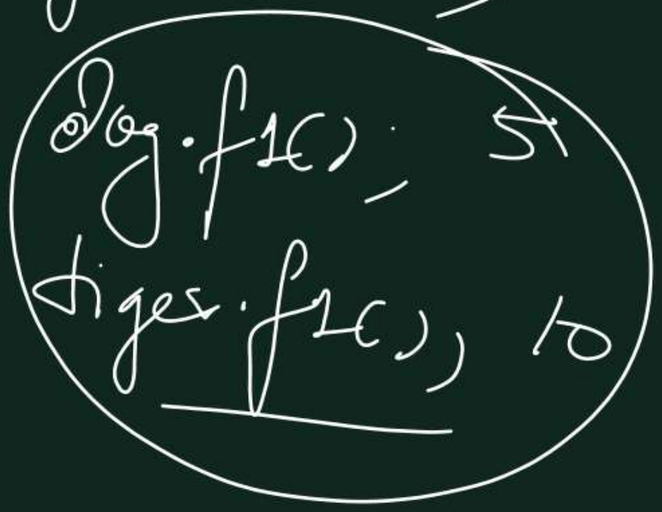
s2.rollno=7



```
class Animal
{
    int h;
    String color;
    [ void f1()
      {
        print h
      }
    ]
}
```

```
Animal dog, tiger;
dog = new Animal();
tiger = new Animal();
dog.h = 5;
tiger.h = 10;
```





Class Student
{
int rollno;
String name
static String college;
}

Student s1, s2;

```
class Student
{
    String name;
    String college;
    void f1()
    {
        name
        college
    }
    static void f2()
    {
        name
        college
    }
}
```

Student

College ABC XYZ

f2()

College

Student s1, s2;

s1 = new Student();

s2 = new Student();

Student.name = "A";

Student.college = "ABC";

Student.f1();

Student.f2(); ABC

s1.name = "Ravi";

s2.name = "Ajay";

s1.f1(); Ravi ABC

s2.f1(); Ajay ABC

s1.name = "Shyam";

s1.college = "XYZ";

s2.f2(); XYZ

A.java

javac A.java

java A

Error

```
class A
{
    public static void main(String aa[])
    {
        System.out.println("Hello");
    }
}
```

Program to Understand Print

```
class A
{
    public static void main(String aa[])
    {
        System.out.print("He\nllo Java
        1\n"); System.out.println("Hello
        Java 2");
        System.out.print("Hello Java 3");
    }
}
```

Program to Understand Print

```
class A2
{
    public static void main(String aa[])
    {
            int a=45;  System.out.println("Hello");
            System.out.println(154);
            System.out.println(52.64);
            System.out.println('A');
            System.out.println(152*10+85);
            double b=213.67;
            System.out.println(a);
            System.out.println(b);
    }
}
```

Program to Understand Print with Concatenation

```
class A3
{
    public static void main(String aa[])
    {
            int a=56;
            double b=23.68;  System.out.println("Hello");
            System.out.println(10+65);
            System.out.println("Hello"+"ABC");
            System.out.println("A="+a);
            System.out.println("Value of B is "+b);
            System.out.println("A="+a+" B="+b);
            System.out.printf("A=%d",a);
    }
}
```

Program to Understand Type Casting

```
class A4
{
    public static void main(String aa[])
    {
            float x=23.67F;
            double y;
            y=x;  x=(float)y;
            System.out.println(x);
            System.out.println(y);
    }
}
```

Program to Understand Typecasting

```
class A5
{
    public static void main(String aa[])
    {
            int a=100; int b=320;
            byte x,y;  x=(byte)a;
            y=(byte)b;
            System.out.println(x);
            System.out.println(y);
    }
}
```

Use of Suffix ‘L’

```
class A6
{
    public static void main(String aa[])
    {
        long x=3000000000L;
        System.out.println(x);
    }
}
```

Arithmetic Operators

```
class A7
{
    public static void main(String aa[])
    {
        int a=56; int b=10;
        System.out.println(a+b);
        System.out.println(a-b);
        System.out.println(a*b);
        System.out.println(a/b);
        System.out.println(a%b);
    }
}
```

Using “Scanner” to input

```
import java.util.Scanner;
class A8
{
    public static void main(String aa[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.print("Enter a  Number:");
        int a=sc.nextInt();
        System.out.print("Enter 2nd Number:");
        int b=sc.nextInt();
        int c=a+b;
        System.out.println("Sum="+c);
    }
}
```

If statement in Java

```
import java.util.Scanner;
class A9
{
	public static void main(String aa[])
	{
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter a Number:");
		int a=sc.nextInt();
		if(a%2==0)
		{
			System.out.println("Even");
		}
		else
		{
			System.out.println("Odd");
		}
	}
}
```

For Loop in Java

```
import java.util.Scanner;
class A10
{
	public static void main(String aa[])
	{
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter a Number:");
		int a=sc.nextInt();
		for(int i=1;i<=10;i++)
		{
			int b=a*i;
			System.out.println(a+"X"+i+"="+b);
		}
	}
}
```

If-else if in Java

```
import java.util.Scanner;
class A11
{
	public static void main(String aa[])
	{
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter 3 Numbers:");
		int a=sc.nextInt();
		int  b=sc.nextInt();
		int  c=sc.nextInt();
		if(a>b && a>c)
		{
			System.out.println("A is Largest");
		}
		else if(b>c)
		{
			System.out.println("B is Largest");
		}
		else
		{
			System.out.println("C is Largest");
		}
	}
}
```

Example of Class and Static

```
import java.util.Scanner;  class Student
{
    String name;
    static String college;
}
class A12
{
    public static void main(String aa[])
    {
        Student s1,s2;  s1=new Student();  s2=new
        Student();
        Student s3=new Student();  Student.college="ABC";
        s1.name="Rajat";  s2.name="Amit";  s3.name="Sumit";
        System.out.println(s1.name);
        System.out.println(s1.college);
        System.out.println(s2.name);
        System.out.println(s2.college);
        System.out.println(s3.name);
        System.out.println(s3.college);
    }
}
```

Object Oriented Programming with Java

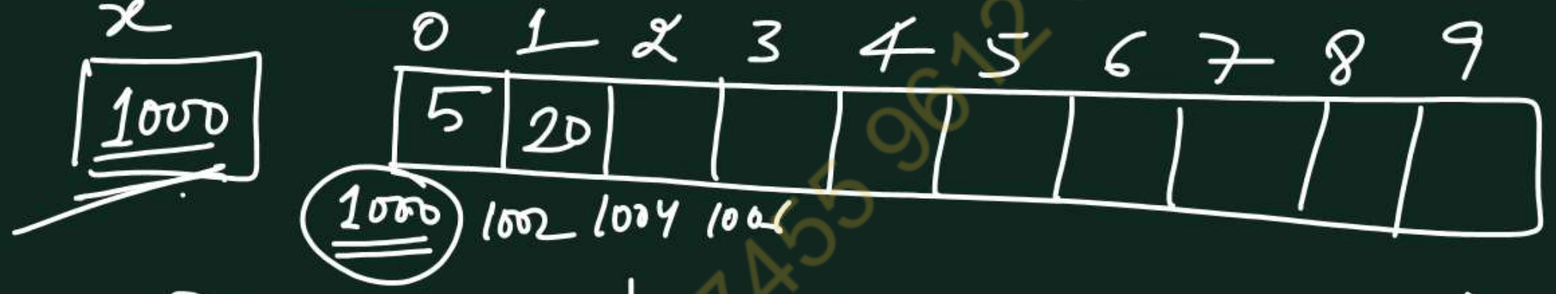
UNIT-1
Lecture-7

What we will Cover?

- What is an Array?
- Why we use Array?
- Overview/Revision of Array in C Language.
- Arrays in Java.
- Syntax and Use of 1 Dimensional Array in Java.
- Programming Examples
- Multidimensional Arrays in Java
- Programming Examples

~~int x, y, z;~~

int x[10];



x[0] = 5;

x[1] = 20;

```
printf("%d", x[1]);
printf("%u", x);
```

1000

```
int x[100];
[
for(i=0; i<100; i++)
{
    scanf("%d", &x[i]);
}
```

int x[10];

int a;

```
{ int x[];
  x = new int[10];
```


```
{ int y[] = {5, 2, 9, 4, 30, 7, 9};
{ int z[] = new int[10];
```

```
int[] x;
x = new int[10];
```

x.length → 10

```
int[] y = new int[10];
int[] z = {5, 6, 7, 20};
```

```
int[] x = new int[10];
Scanner s = new Scanner(System.in);
System.out.print("Enter 10 Values: ");
for(int i=0; i<10; i++)
{
    x[i] = s.nextInt();
}
for(int i=0; i<x.length; i++)
{
    System.out.println(x[i]);
}
```

```
int[] x = {5, 20, 3, 9, 14};
for(int item : x)
{
    System.out.println(item);
}
```

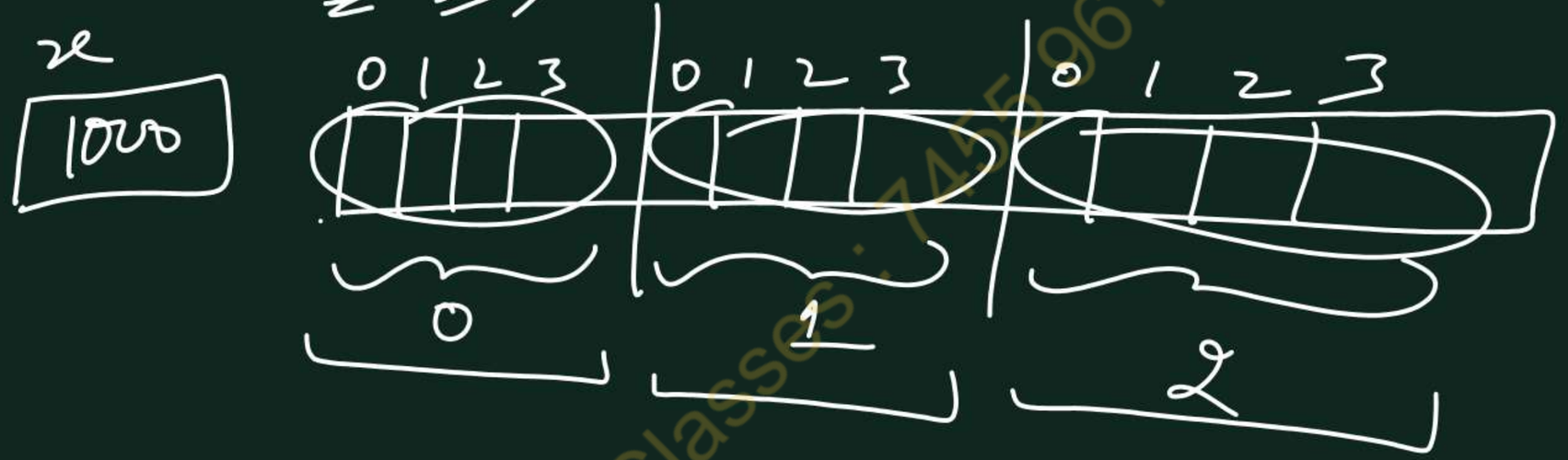
item

5
20
3
9
14

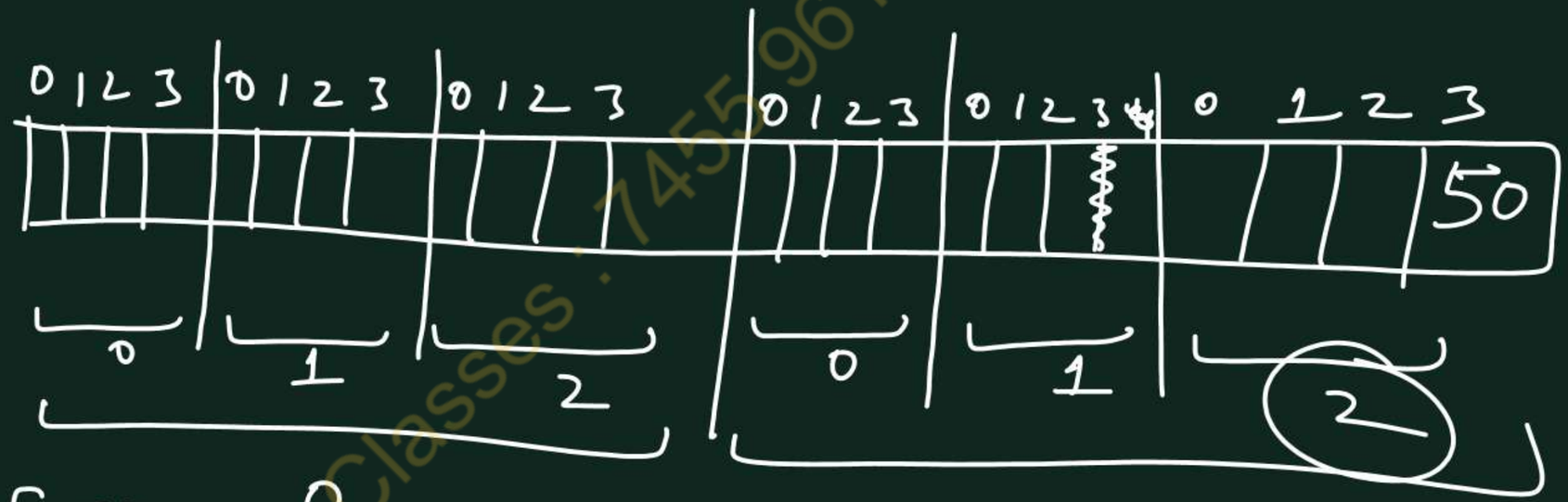
Examples

Multidimensional Arrays

int x[3][4];



int xe[2][3][4];



xe[1][2][3] = 50

1

Multidemensional in Java

Jagged Array

```
int[] a = {5, 2, 9};
int[] b = {7, 12, 9, 4, 8};
int[] c = {3, 9, 12, 4};
int[][] x = {a, b, c};
```

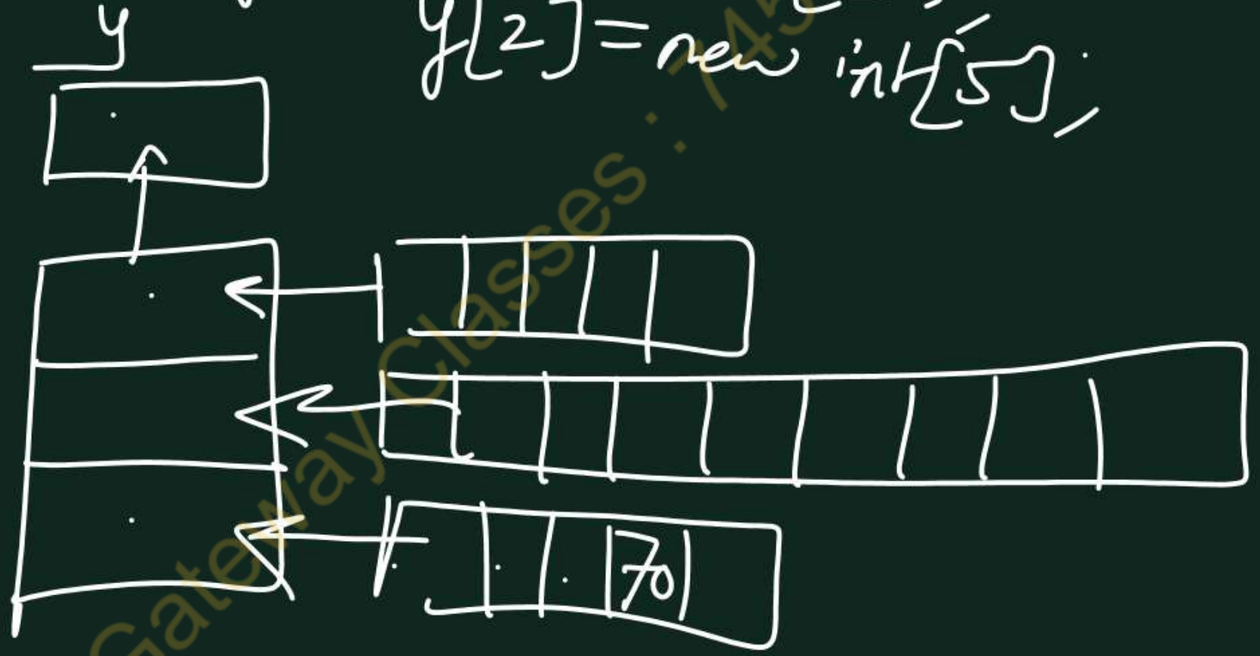
a.length 3
b.length 5
c.length 4
b[2] ? 9

x.length 3
x[0].length 3
x[1].length 5
x[2].length 4

```
for(int[] m : x)
{
    Sop("Hello");
}
```

```
int[][] y;
y = new int[3][];
y[0] = new int[5];
y[1] = new int[10];
y[2] = new int[5];
y[2][3] = 70;
```

```
int[][] z = new int[3][4];
```



Examples

```
class A16
{
	public static void main(String aa[])
	{
		int x[];
		x=new int[10];
		int y[]=new int[17];
		int z[]=new int[]{123,45,67,879,67,23,56};
		int w[]={123,45,67,879,67,23,56};
		System.out.println(x.length);
		System.out.println(y.length);
		System.out.println(z.length);
		System.out.println(w.length);
	}
}
```

```
-------------------------------------------------------------------------------------
---
class A17
{
	public static void main(String[] aa)
	{
		int[] x;
		x=new int[10];
		int[] y=new int[17];
		int[] z=new int[]{123,45,67,879,67,23,56};
		int[] w={123,45,67,879,67,23,56};
		System.out.println(x.length);
		System.out.println(y.length);
		System.out.println(z.length);
		System.out.println(w.length);
	}
}
```

```
--------------------------------------------------------------------------------
--
import java.util.Scanner;
class A18
{
	public static void main(String[] aa)
	{
		int[] x=new int[10];
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter 10 values:");
		for(int i=0;i<x.length;i++)
		{
			x[i]=sc.nextInt();
		}
		System.out.println("\nValues are:");
		for(int i=0;i<x.length;i++)
		{
			System.out.print(x[i]+"  ");
		}
		System.out.println("\nValues are:");
		for(int item:x)
		{
			System.out.print(item+"  ");
		}
	}
}
```

```
import java.util.Scanner;
class A19
{
	public static void main(String[] aa)
	{
		int[] x=new int[10];
		Scanner sc=new Scanner(System.in);
		System.out.print("Enter 10 values:");
		for(int i=0;i<x.length;i++)
		{
			x[i]=sc.nextInt();
		}
		int min=x[0];
		for(int item:x)
		{
			if(min>item)
				min=item;
		}
		System.out.println("Min:"+min);
	}
}
```

```
-----------------------------------------------------------------
import java.util.Scanner;
class A20
{
	public static void main(String[] aa)
	{
		int[] a={12,345,6,6};
		int[] b={23,45,67,213,12,46};
		int[] c={23,67,67,34,2};
		int[][] x={a,b,c};
		System.out.println(x.length);
		System.out.println(x[0].length);
		System.out.println(x[1].length);
		System.out.println(x[2].length);
		System.out.println(x[1][3]);
		System.out.println();
		for(int i=0;i<x.length;i++)
		{
			for(int j=0;j<x[i].length;j++)
			{
				System.out.print(x[i][j]+"  ");
			}
			System.out.println();
		}
		System.out.println();
		for(int[] m:x)
		{
			for(int item:m)
			{
				System.out.print(item+"  ");
			}
			System.out.println();
		}
	}
}
```

```
import java.util.Scanner;
class A21
{
	public static void main(String[] aa)
	{
		int[][] arr=new int[3][5];
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter matrix of 3x5:");
		for(int i=0;i<arr.length;i++)
		{
			for(int j=0;j<arr[i].length;j++)
			{
				arr[i][j]=sc.nextInt();
			}
		}
		System.out.println("matrix of 3x5:");
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<5;j++)
			{
				System.out.print(arr[i][j]+"   ");
			}
			System.out.println();
		}
	}
}
```

Object Oriented Programming with Java

UNIT-1
Lecture-8

String in Java

Why String is NOT a primitive data type?

String class

Methods of String Creation.

Internal Memory Structure of String Literals and String Objects.

Members of String Class

Examples

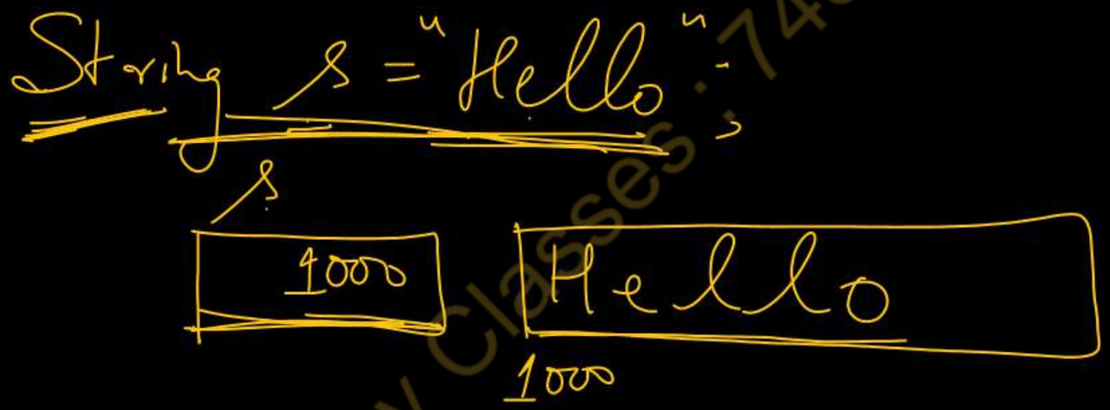
Exercise

String is NOT a primitive data type, why?

String is a collection of Characters , like character array, and all arrays are references types (we also used array to string the string in C Language), therefore we need a reference type data to String, that's why String is NOT a primitive data type in Java, it's a class.

What is String?

- All String Objects including String Literals (e.g. "Hello") are instances of String Class.
- All String Objects are immutable in Java, we can't change the string value, but we can change the string references.



String s1
String s2

s1 == s2

NEVER ==

String s1 = "Hello";

Immutable

s1 = "ABC";

int x = 20

x

20 40

x = 40

String s1 = "Hello";

String s2 = new String("Hello");

String s3 = new String("Hello");

String s4 = "Hello";

s1 == s2 F

s1 == s3 F

s1 == s2 T

s1 == "Hello" T

s2 == "Hello" F

```
class A
{
    void f1()
    {
        ========
    }
    void f2(int x)
    {
        ========
    }
    int f3()
    {
        ========
        return 10;
    }
}
```

```
void f1();
void f2(int a)
int f3()

A a = new A();
a.f1();
a.f2();  X
f2(70);
int x = a.f3();
```

Library of String-2

- String substring(int start)
- String substring(int start,int end)
- String toLowerCase()
- String toUpperCase()
- int indexOf(String s)
- int indexOf(String s,int start)
- int lastIndexOf(String s)
- int lastIndexOf(String s,int start)
- String replace(String oldstr,String newstr)

int x = s.indexOf("Hello",15)

String s1="Hello";

String s2="Example of Str

String s3=s2.substring(2);

String s4=s2.substring(3,5);

int x=s2.indexOf("a");

ample of Str

am

String s = "_____";

String s1 = s.replace("Ab", "xeyz");

String toLowerCase()

String toUpperCase()

int length()

boolean equals(String s)

boolean equalsIgnoreCase(String s)

String s1 = _____

String s2 = _____

boolean b = s1.equals(s2);

if(b==true)
Same
else
Different

if(b)
{
Same
}
else
{
Different
}

Library of String-3

- String[] split(String delim)
- String trim()
- char[] toCharArray()
- byte[] getBytes()

```
String s1 = "   Hello Java   ";
String s2 = s1.trim();          // "Hello Java"
String a[] = s1.split("a");
char[] ch = s1.toCharArray();
byte[] b = s1.getBytes();
```

Object Oriented Programming with Java

UNIT-1
Lecture-9

What We Will Cover?

OOPs Properties in Java

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction
5. Modularity or Packaging
6. Association
7. Aggregation

Benefits of OOPs

- Provides modularity and enhances program for easier troubleshooting as objects exist independently
- Real world programming
- Code reusability through inheritance
- OOPs concepts designed with lesser limitations for larger programs are hard to write.
- Enhances easier and clear modular structure for programs.
- Flexibility through polymorphism
- Effective problem solving and better software quality
- Lower maintenance cost & resilience to change and hides information

…Benefits of OOPs

- Flexibility of upgrading systems of any size
- Improvise the creation of Graphical User Interface (GUI) applications
- Faster goals achieved for programmers
- Greater programmer productivity & data become active
- Faster, accurate, better written applications compared to a procedural program
- Simple to develop and implement
- Much easier communication with the message passing capability

OOPs Concepts

Encapsulation

```
class Student
{
    String name;
    void study()
    {
    }
}
```

```
Student s=new Student();
s.name
s.course
s.study()
s.f1();
s.f2();
```

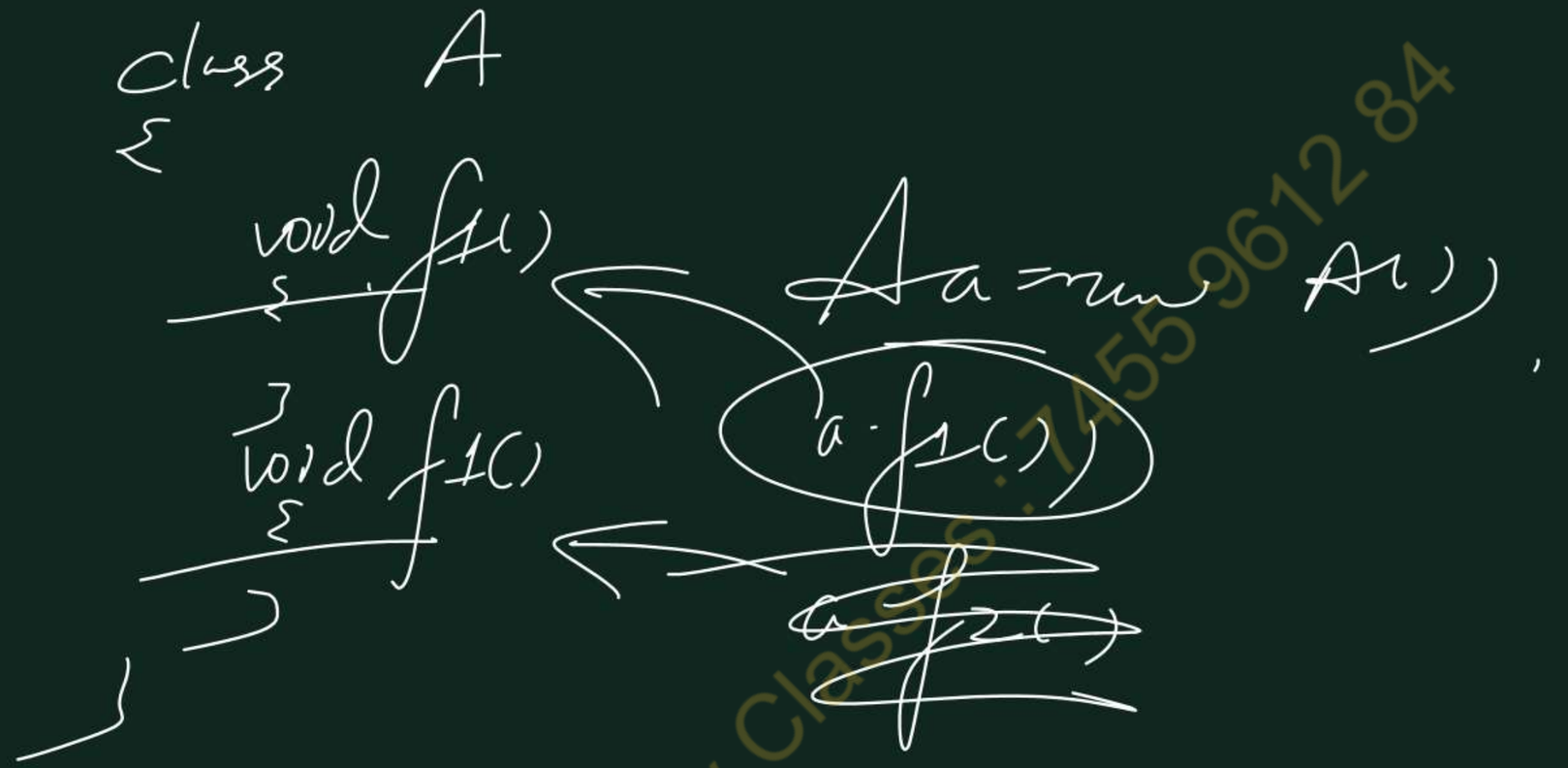
OOPs Concepts

```
class Student
{
    String name;
    String course;

    void study()
    {
    }

    void study()
    {
    }
}
```

```
Student s = new Student();
s.study();
```



Function Overloading

No. of Parameters
OR
Type of Parameters

BEST MATCH

```
class A
{
  void fun()
  {
  }
  void f1(int x)
  {
  }
  void f1(double x)
  {
  }
}
```

A a=new A();

a.f1(2);

a.f1(7.9);

a.f1(7.82L);

a.f1(7.85);

```
class Box
{
    int x, y, z;
    void volume()
    {
        int v = x * y * z;
        Sop(v);
    }
    void f1()
    {
        x = 2
        y = 3
        z = 4
    }
}
```

```
Box b1 = new Box();
b1.x = 2;
b1.y = 3;
b1.z = 4;
b1.volume();
```

24

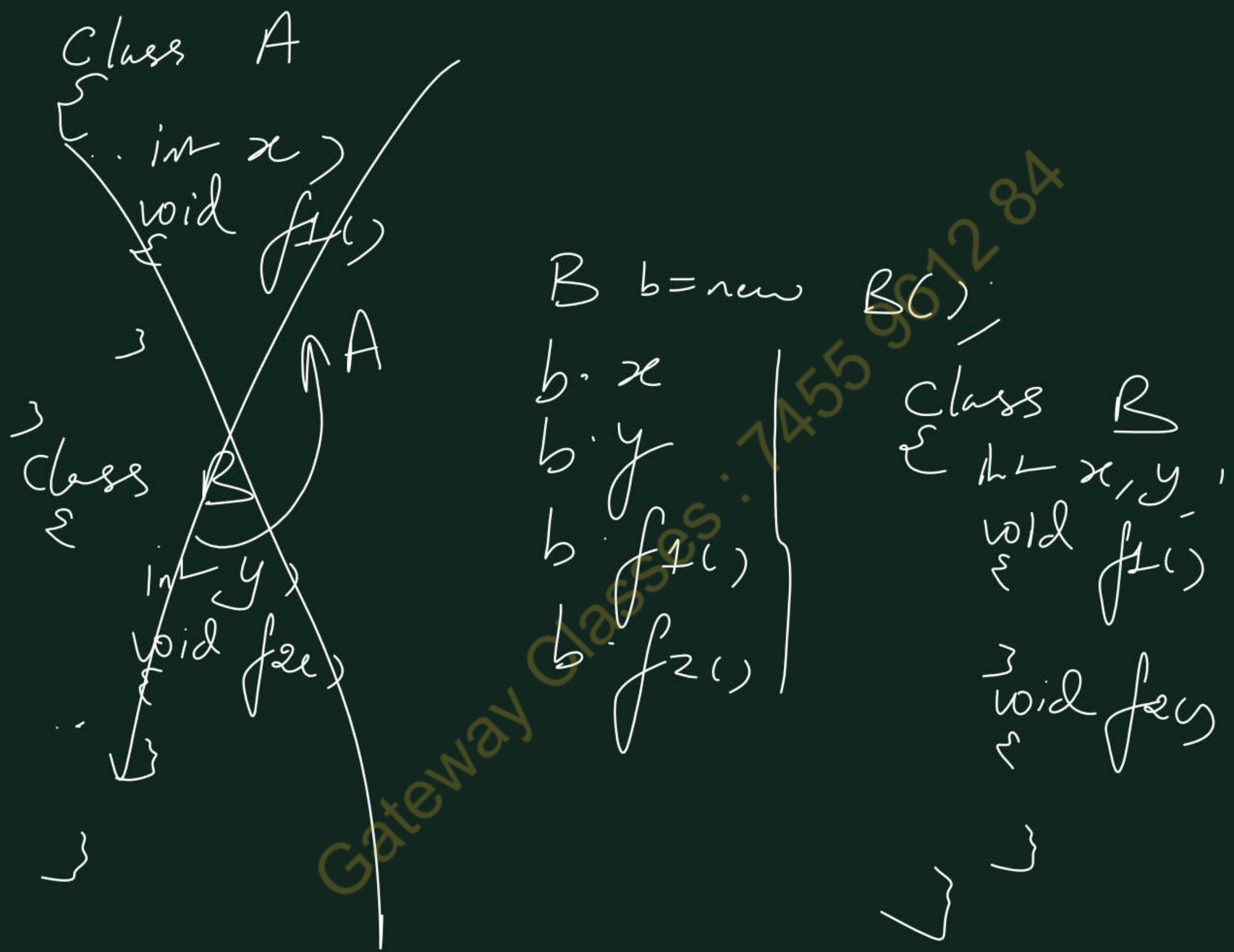
```
class Box
{
    int x, y, z;
    void volume()
    {
        int v = x*y*z;
        System.out.println(v);
    }
    Box()
    {
        x = 2;
        y = 3;
        z = 4;
    }
    Box(int a, int b, int c)
    {
        x = a;
        y = b;
        z = c;
    }
}
```

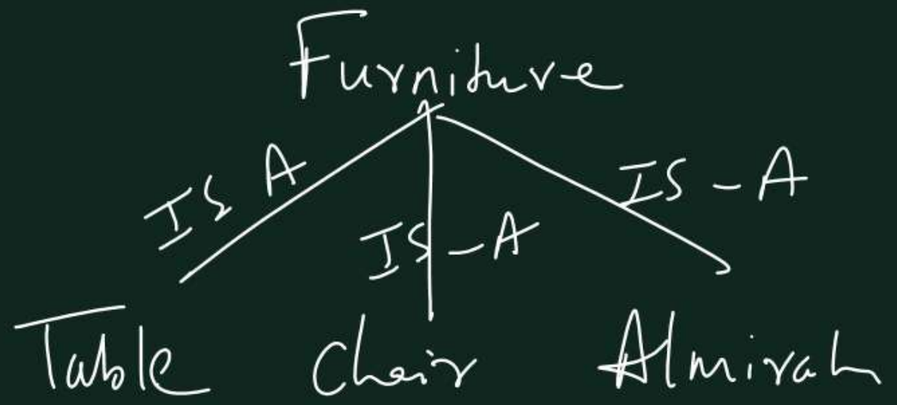
```
Box b1 = new Box();
b1.volume();
24
Box b2 = new Box(2,3,9);
b2.volume();
54
```

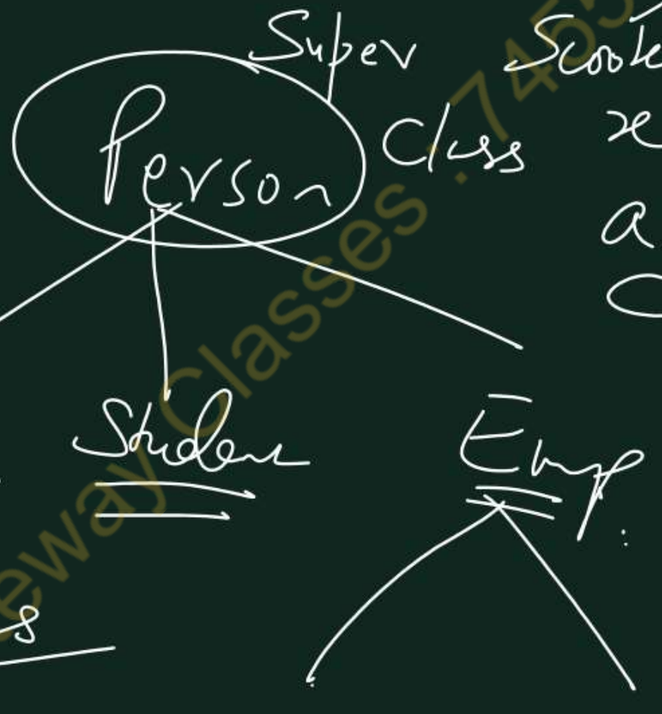
```
class Emp
{
	int eid;
	String name;
	String department;
	//Function Overloading
	void f1()
	{
		System.out.println("Zero");
	}
	void f1(int a)
	{
		System.out.println("1 Integer");
	}
	void f1(double b)
	{
		System.out.println("1 Double");
	}
	void f1(float b)
	{
		System.out.println("1 Float");
	}
}
class A25
{
	public static void main(String[] ar)
	{
		byte x1=43;
		short x2=34;
		int x3=45;
		long x4=56;
		float x5=23;
		double x6=23;
		String x7="ABC";
		boolean x8=true;
		char x9='A';
		Emp e=new Emp();
		e.f1();
		e.f1(x1);
		e.f1(x2);
		e.f1(x3);
		e.f1(x4);
		e.f1(x5);
		e.f1(x6);
		//e.f1(x7);
		//e.f1(x8);
		e.f1(x9);
	}
}
```

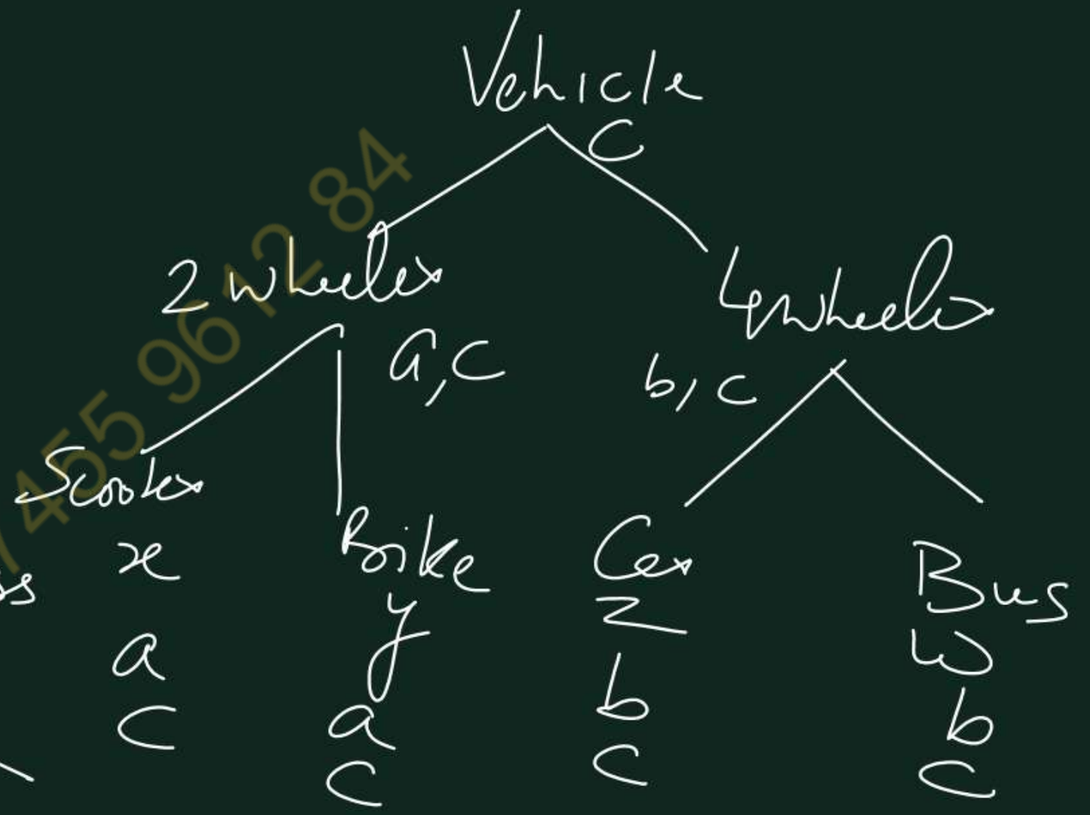
```
//Example of Constructor

class Emp
{
	int eid;
	String name;
	String department;
	Emp(String nm)
	{
		name=nm;
	}
	Emp(int id,String nm)
	{
		eid=id;
		name=nm;
	}
}
class A26
{
	public static void main(String[] ar)
	{
		Emp e1=new Emp("Ravi");
		System.out.println(e1.eid);
		System.out.println(e1.name);
		Emp e2=new Emp(45,"Amit");
		System.out.println(e2.eid);
		System.out.println(e2.name);
	}
}
```



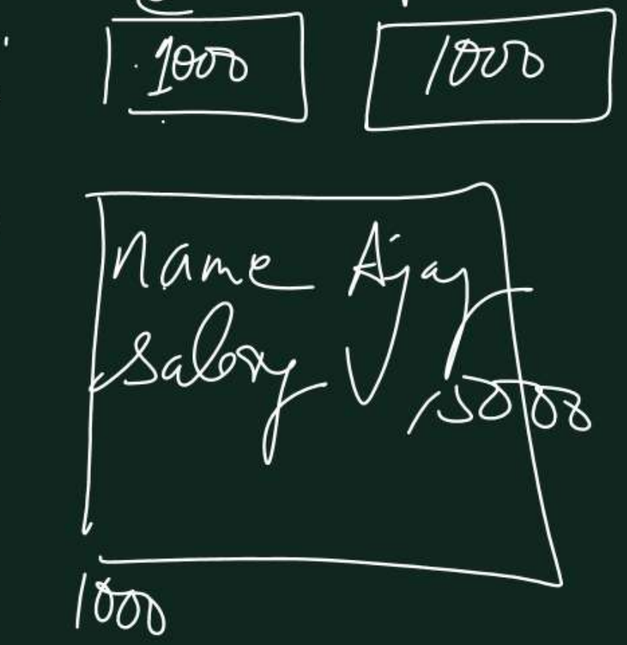


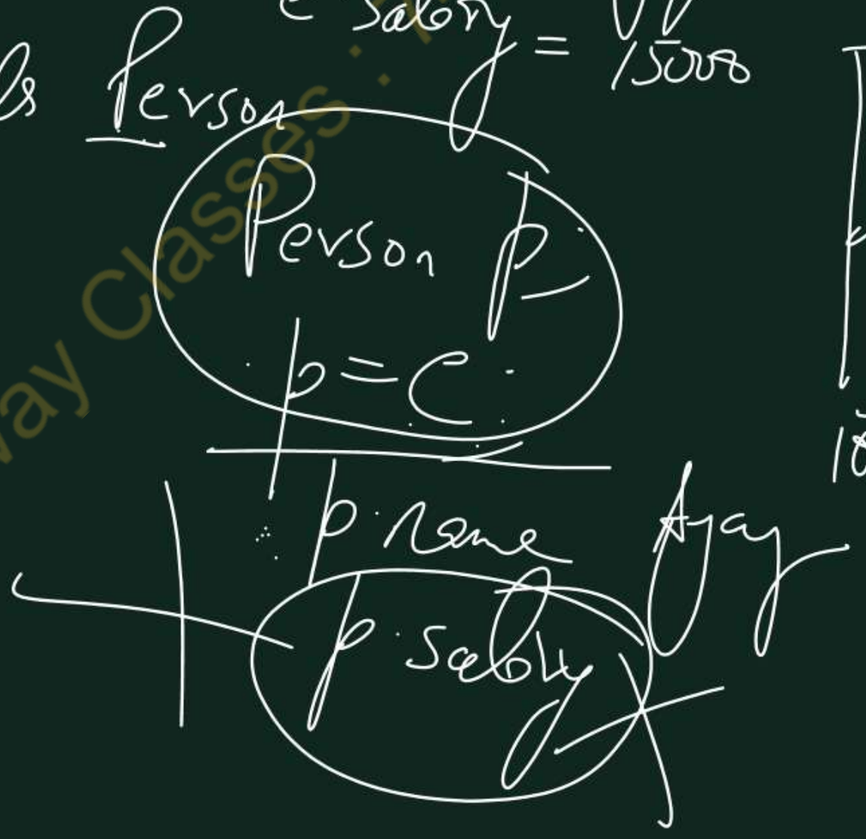




```
Class Person
{
   String name;
}
Class Student extends Person
{
   String course;
}
Class Emp extends Person
{
   int salary;
}
```

```
Emp e;
e = new Emp();
e.name = "Ajay";
e.salary = 15000;
```



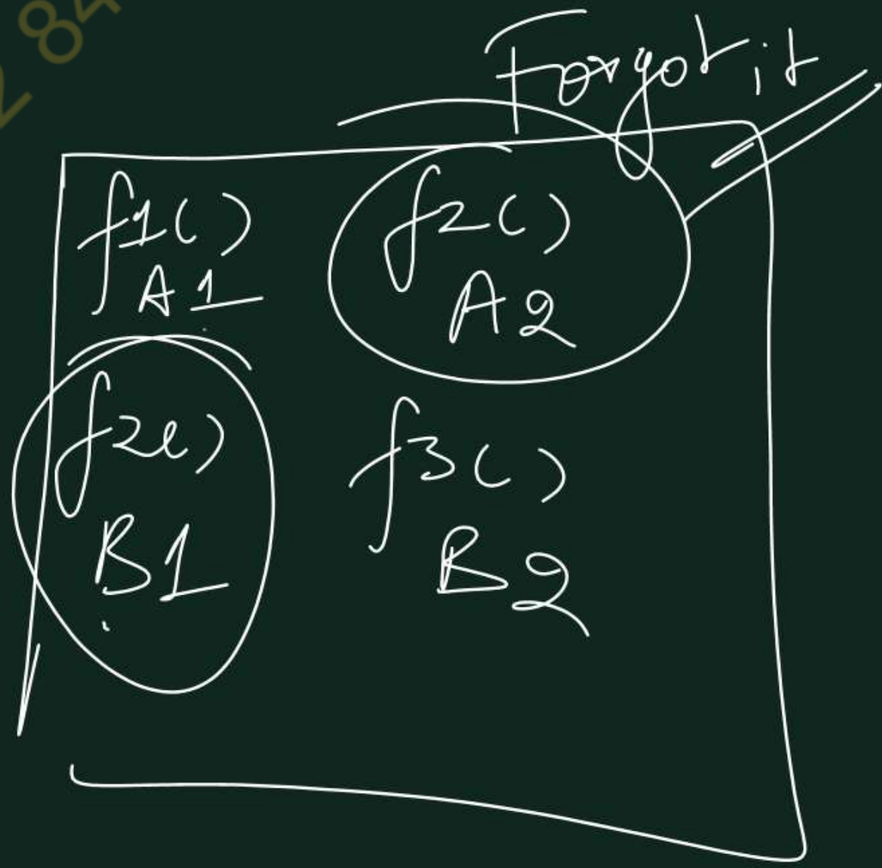


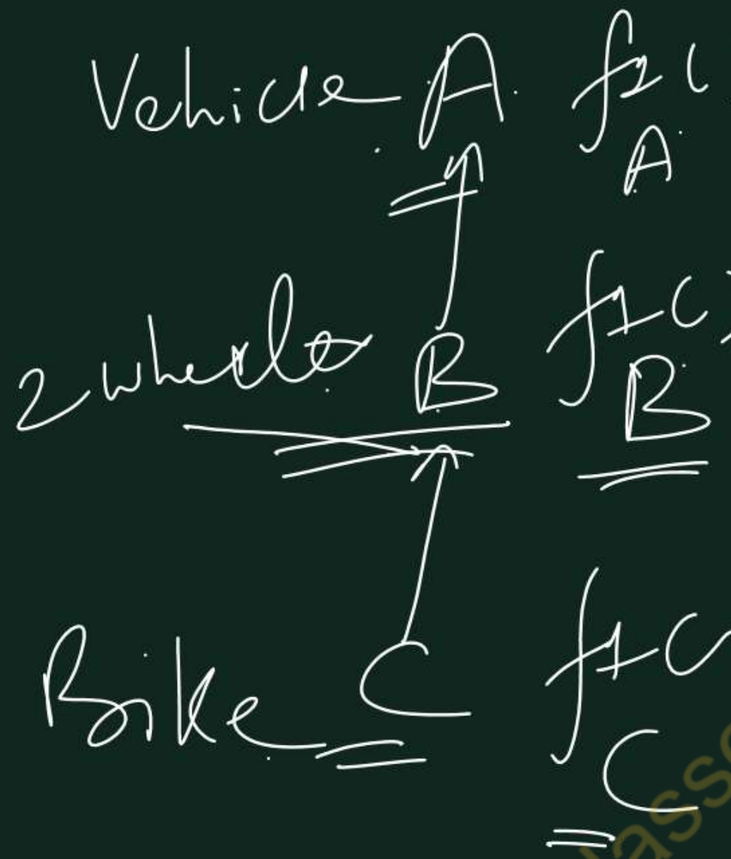
OVERRIDING

```
class A
{
  void f1()
  {
    A1
  }
  void f2()
  {
    A2
  }
}
class B extends A
{
  void f2()
  {
    B1
  }
  void f3()
  {
    B2
  }
}
```

Overriding

```
B b=new B();
b.f1(); A1
b.f2(); B1
b.f3(); B2
A a=b;
a.f1(); A1
a.f2(); B1
a.f3();
```





```
// Memory Allocated to Super then Sub
class Person
{
	Person()
	{
		System.out.println("Person");
	}
}
class Student extends Person
{
	Student()
	{
			System.out.println("Student");
	}
}
class A27
{
	public static void main(String[] ar)
	{
		Student s=new Student();
	}
}
```

```
// Generalization
class Person
{
	String name;
}
class Student extends Person
{
	String course;
}
class Emp extends Person
{
	int salary;
}
class A28
{
	public static void main(String[] ar)
	{
		Student s1=new Student();
		s1.name="Abhay";
		s1.course="B.Tech";
		System.out.println(s1.name);
		System.out.println(s1.course);
		System.out.println();
		Person p=s1;
		System.out.println(p.name);
		//System.out.println(p.course);
	}
}
```

```
//Function Overriding
class Person
{
	void f1()
	{
		System.out.println("Person F1");
	}
	void f2()
	{
		System.out.println("Person F2");
	}
}
class Student extends Person
{
	void f3()
	{
		System.out.println("Student F3");
	}
	void f2() // function overriding
	{
		System.out.println("Student F2");
	}
}
class A29
{
	public static void main(String[] ar)
	{
		Student s=new Student();
		s.f1();
		s.f2();
		s.f3();
		Person p=s;
		p.f1();
		p.f2();
		//p.f3();
	}
```

```
}
```

class A
{
void f1()
{
}
void f2()
{
}
}

abstract class B
{
void f1()
{
}
abstract void f2();
}

interface C
{
void f1();
void f2();
void f3();
}

A a = new A();

B b = new B();

C c = new C();

class D implements C
{
}

```
abstract class A
{
	abstract void f1();
	void f2()
	{
		System.out.println("F2 in A");
	}
}
interface B
{
	void f1();
	void f2();
}
class C
{
	void f1()
	{
	}
	void f2()
	{
	}
}
class D implements B
{
	public void f2()
	{
	}
}
class A30
{
	public static void main(String[] ar)
	{

	}
}
```

Object Oriented Programming with Java

UNIT-1
Lecture-10

What We Will Cover?

OOPs Properties in Java

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction
5. Multiple Inheritance in Java
6. Use of “super” Keyword
7. Use of “final” keyword

Shape s1, s2, s3;

s1 = new Circle();

s2 = new Rectangle();

s3 = new Triangle();

s1.area(); Circle

s2.area(); Rectangle

s3.area(); Triangle

Shape

void area()
{
}

Circle

void area()
{
}

Circle
}

Rectangle

void area()
{
}

Rectangle
}

Triangle

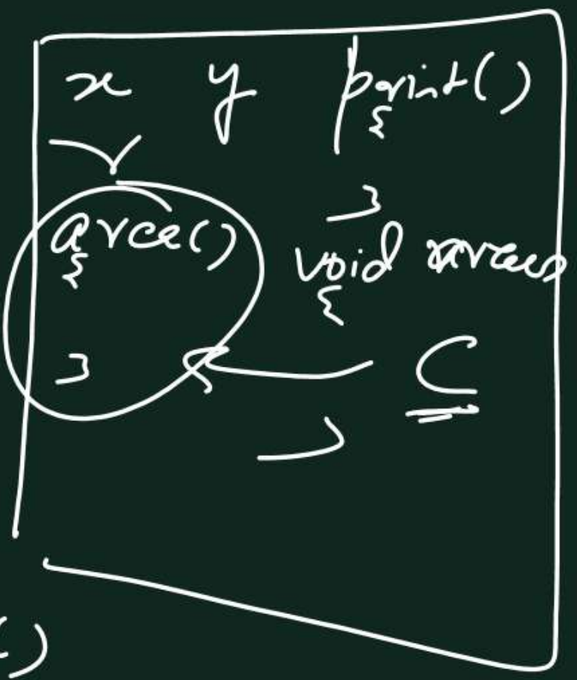
void area()
{
}

Triangle
}

int x, y;

void print()
{
S.O.P(x)
S.O.P(y)
}

s1



Shape s1, s2, s3;
s1 = new Circle();
s2 = new Triangle();
s3 = new Rectangle();

s1.area();
s1.print();

~~abstract~~ Shape
interface

int x, y;
abstract void print();
abstract void area();

Circle
void area()
{
C
}
void print()
{
x
y
}

Triangle
void area()
{
T
}
void print()
{
x
y
}

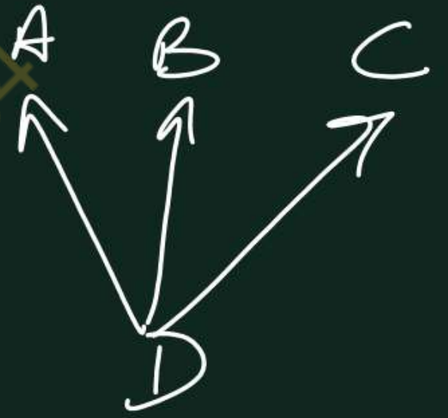
Rectangle
void area()
{
R
}
void print()
{
x
y
}

Ellipse

Multiple Inheritance

Simple/Single

Multilevel

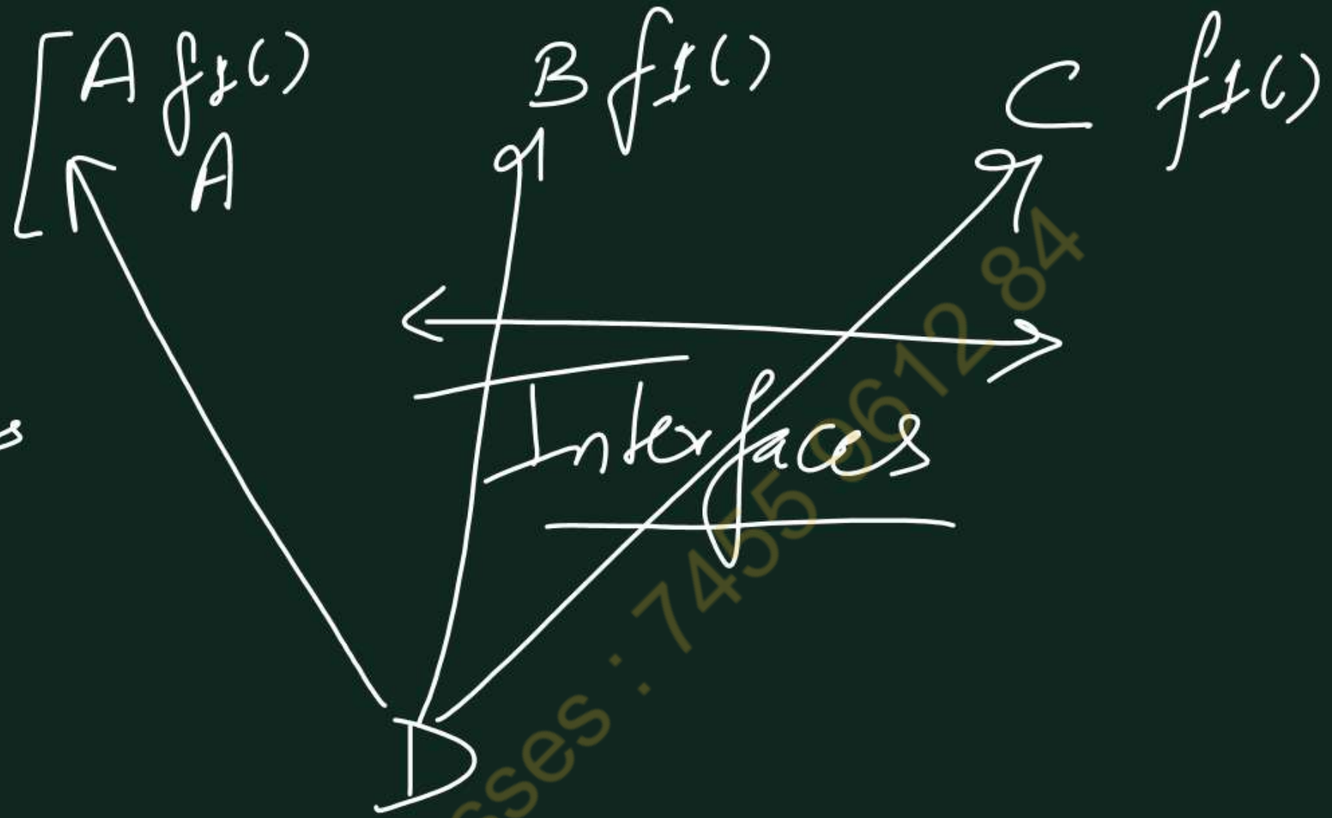


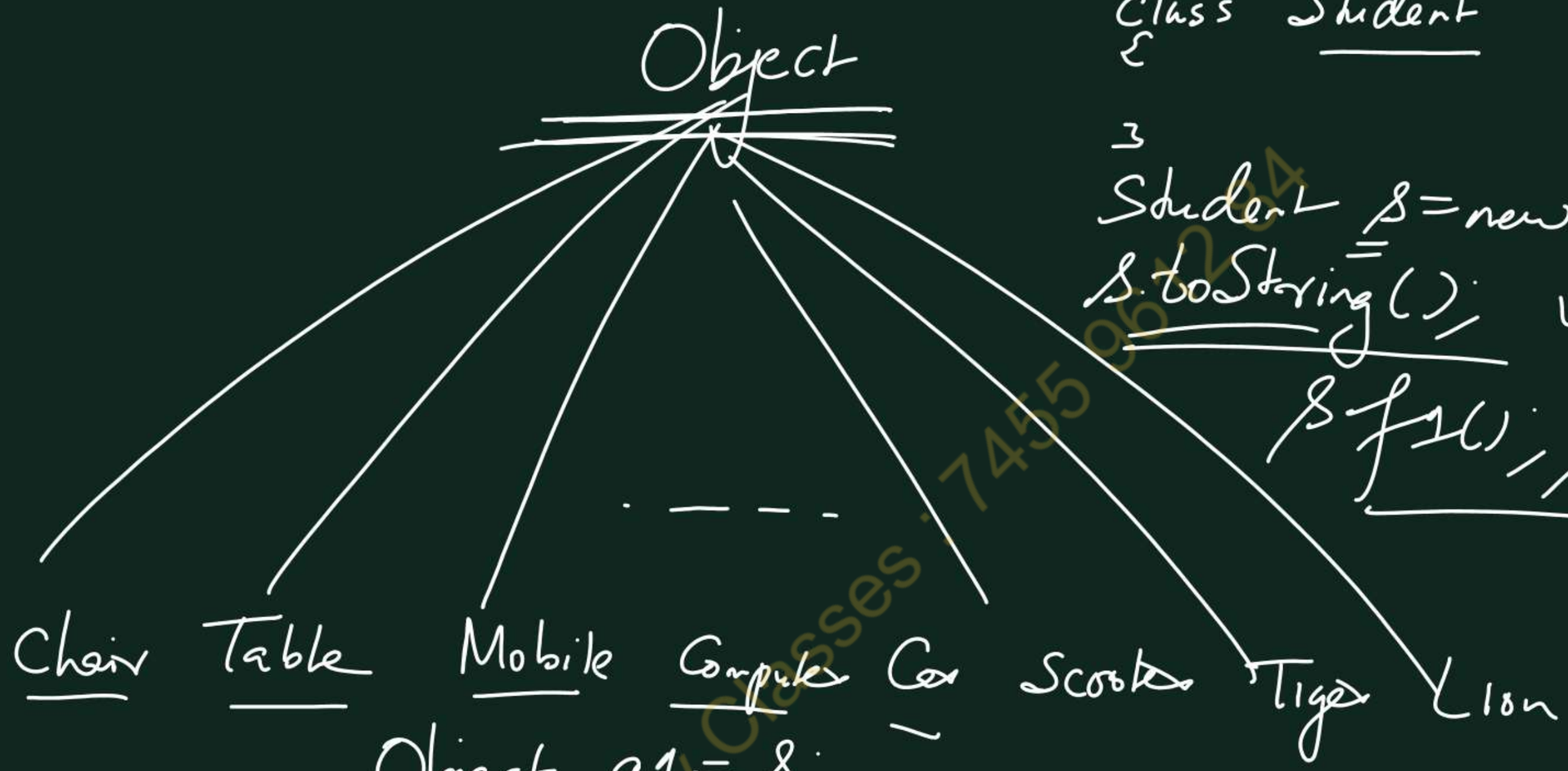
Multiple

Class

Abstract class

Interface






```
class Student
{
}
Student S = new Student();
S.toString();
S.fun();  X
```

```
Object a1 = S;
       a2 = 72
       a3 = 7.9
       a4 = "Hello";
```

OOPs Concepts

1) final — To Declare Const
2) — To Declare Individual Class (To Prevent Inheritance)
3) — To Prevent Overriding

OOPs Concepts

final int x=20;

x=30; ✗

final int y;

y=30; ✓

y=50; ✗

final class A

class B extends A

Error

class A
{
void f1()
}

final void f2()

class B extends A
{
void f1()
}

f2() Error

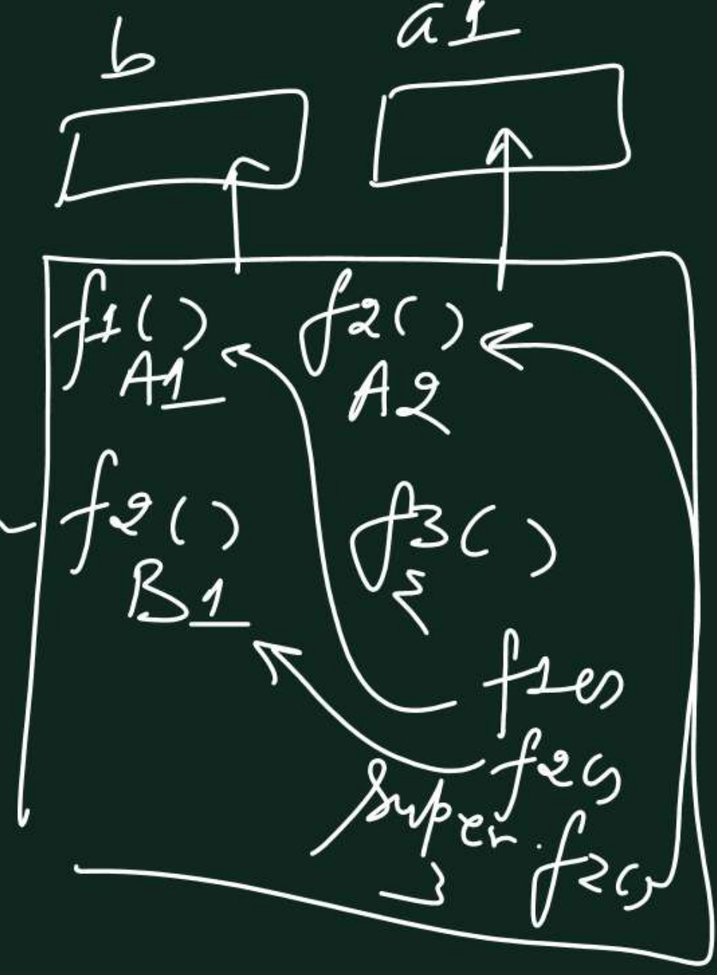
}

```
class A
{
    void f1()
    {
        A1
    }
    void f2()
    {
        A2
    }
}
class B extends A
{
    void f2()
    {
        B1
    }
    void f3()
    {
        f1();
        f2();
        super.f2();
    }
}
```

```
A a=new A();
a.f1()  A1
a.f2()  A2
```

```
B b=new B();
b.f1()  A1
b.f2()  B1
b.f3()  A1
        B1
        A2
```

```
A a1=b;
a1.f1()  A1
a.f2();  B1
a.f3();  ✗
```



```
class A
{
    A(int x)
    {
        A2
    }
    A(String s)
    {
        A3
    }
}

class B extends A
{
    B()
    {
        super(73);
        B1
    }
    B(double a)
    {
        super(50);
        B2
    }
    B(String ss, int x)
    {
        super(ss);
        B3
    }
}
```

B b1 = new B();
A2
B1

B b2 = new B(4.92)
~~A2~~
B2

B b3 = new B("AB", 4);
~~A3~~
B3

```
// Polymorphism
class Shape
{
	int x;
	int y;
	void print()
	{
		System.out.println(x);
		System.out.println(y);
	}
	void area()
	{
	}
}
class Circle extends Shape
{
	int r;
	void area()
	{
		System.out.println("Circle");
	}
}
class Triangle extends Shape
{
	void area()
	{
		System.out.println("Triangle");
	}
}
class Ellipse extends Shape
{
	int r1,r2;
	void area()
	{
		System.out.println("Ellipse");
	}
}
class A31
{
	public static void main(String[] ar)
	{
		Shape s1,s2,s3;
		s1=new Circle();
		s2=new Ellipse();
```

```
		s3=new Triangle();
		s1.area();
		s2.area();
		s3.area();
	}
}
```

```
// Polymorphism with Abstraction
abstract class Shape
{
	int x;
	int y;
	void print()
	{
		System.out.println(x);
		System.out.println(y);
	}
	abstract void area();
}
class Circle extends Shape
{
	int r;
	void area()
	{
		System.out.println("Circle");
	}
}
class Triangle extends Shape
{
	void area()
	{
		System.out.println("Triangle");
	}
}
class Ellipse extends Shape
{
	int r1,r2;
	void area()
	{
		System.out.println("Ellipse");
	}
}
class Rectangle extends Shape
{
```

```
}
class A32
{
	public static void main(String[] ar)
	{
		Shape s1,s2,s3;
		s1=new Circle();
		s2=new Ellipse();
		s3=new Triangle();
		s1.area();
		s2.area();
		s3.area();
		Shape s=new Shape();
	}
}
```

```
// Polymorphism with Abstraction using Interface

interface Shape
{
	void print();
	abstract void area();
}
class Circle implements Shape
{
	int r;
	public void print()
	{
	}
	public void area()
	{
		System.out.println("Circle");
	}
}
class Triangle implements Shape
{
	public void print()
	{
	}
	public void area()
	{
		System.out.println("Triangle");
	}
}
```

```
class Ellipse implements Shape
{
	int r1,r2;
	public void print()
	{
	}
	public void area()
	{
		System.out.println("Ellipse");
	}
}
class Rectangle implements Shape
{
}
class A33
{
	public static void main(String[] ar)
	{
		Shape s1,s2,s3;
		s1=new Circle();
		s2=new Ellipse();
		s3=new Triangle();
		s1.area();
		s2.area();
		s3.area();
		Shape s=new Shape();
	}
}
```

```
//Use of "final" keyword

final class A
{
}
//class B extends A
//{
//}
class C
{
	void f1()
	{
	}
	final void f2()
	{
```

```
	}
}
class D extends C
{
	void f1()
	{
	}
	//void f2()
	//{
	//}
}
class A34
{
	public static void main(String[] ar)
	{
		final int x=23;
		//x=789;
	}
}
---------------------------------------------------------------------------
//use of "super" keyword with constructor
class A
{
	//A()
	//{
	//	System.out.println("A1");
	//}
	A(int x)
	{
		System.out.println("A2");
	}
	A(String s)
	{
		System.out.println("A3");
	}
}
class B extends A
{
	B()
	{
		super(100);
		System.out.println("B1");
	}
	B(double a)
```

```
	{
		super(20);
		System.out.println("B2");
	}
	B(String s,int a)
	{
		super(s);
		System.out.println("B3");
	}
}
class A35
{
	public static void main(String[] ar)
	{
		B b1=new B();
		System.out.println();
		B b2=new B(23.567);
		System.out.println();
		B b3=new B("ABC",55);
	}
}
------------------------------------------------------------
// Multiple Inheritance using Interfaces
class A
{
}
interface B
{
}
interface C
{
}
class D extends A implements B,C
{
}
class A35
{
	public static void main(String[] ar)
	{
	}
}
```

Object Oriented Programming with Java

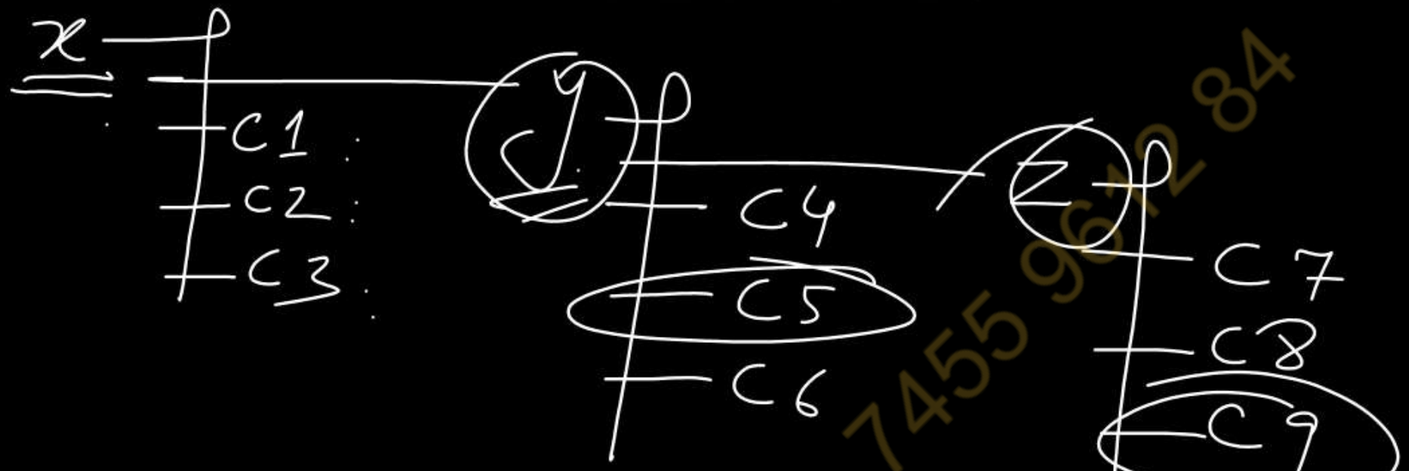
UNIT-1
Lecture-11

What We Will Cover?

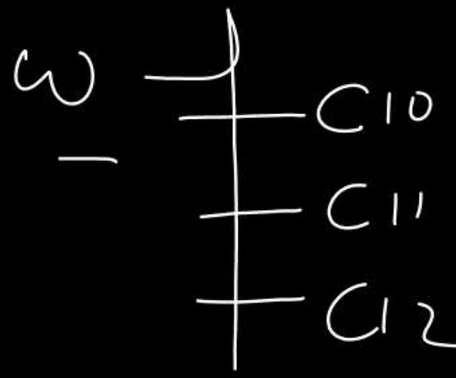
OOPs Properties in Java

1. Packages
2. Access Modifiers
3. Access Specifiers
4. Java's Package Structure
5. Import Statement
6. Static Import in Java
7. CLASSPATH, creating jar file etc.

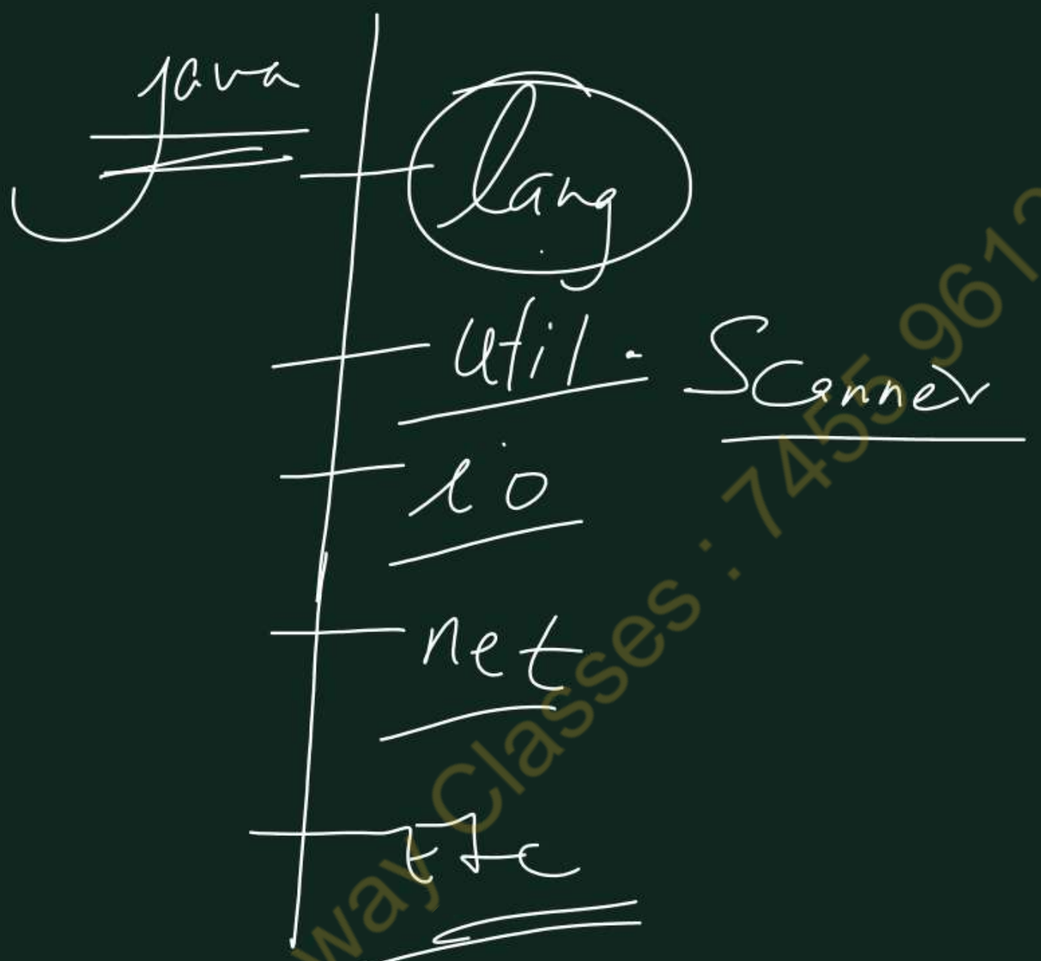
OOPs Concepts



x.y.C5



x.y.z.C9



```
package p1;
public class A1
{
    private    x
               y
    protected  z
    public     w
}

class A2 extends A1
{
    x  ✗
    y  ✓
    z  ✓
    w  ✓
}

class A3
{
    void f3()
    {
        A1 a = new A1();
        a.x  ✗
        a.y  ✓
        a.z  ✓
        a.w  ✓
    }
}
```

```
import p1.A1;
import p1.*;

package p2;
class B1 extends p1.A1
{
    x  ✗
    y  ✗
    z  ✓
    w  ✓
}

class B2
{
    void f2()
    {
        p1.A1 a = new p1.A1();
        A1 b = new A1();
        b.x  ✗
        b.y  ✗
        b.z  ✓
        b.w  ✓
    }
}
```

static

final

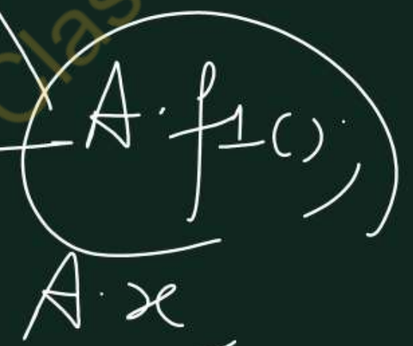
native

synr...

Static Import

```
package p1;
public class A
{
public static void f1()
{
public static int x=10;
```

```
Static Import p1.A;
Class B
{
psvm()
{
f1();
sop(x);
}
}
```



Jar files

Jar

Thank You

AKTU

B. Tech II-Year

Gateway Classes

Full Courses Available in App

AKTU B.Tech I- Year : All Branches

AKTU B.Tech II- Year

Branches : 1. CS IT & Allied

2. EC & Allied

3. ME & Allied

4. EE & Allied

Downlod App Now

Download App

Full Courses

- V. Lectures
- Pdf Notes
- AKTU PYQs
- DPP