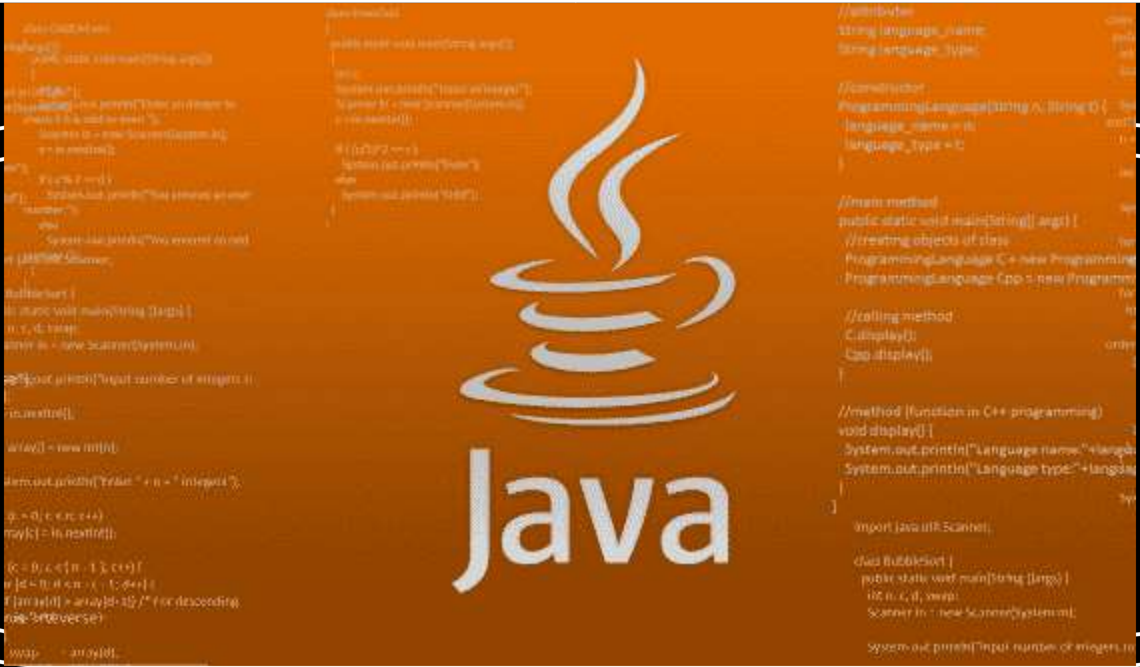
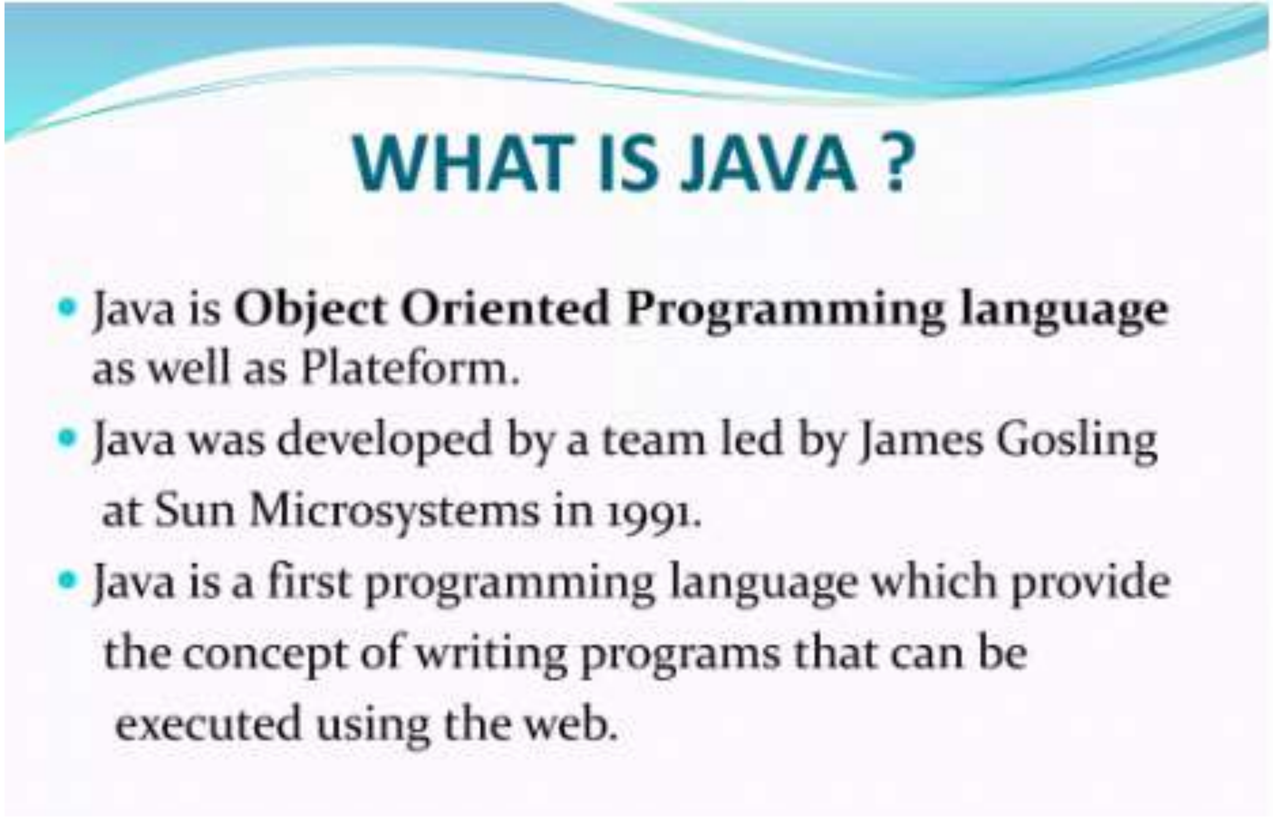
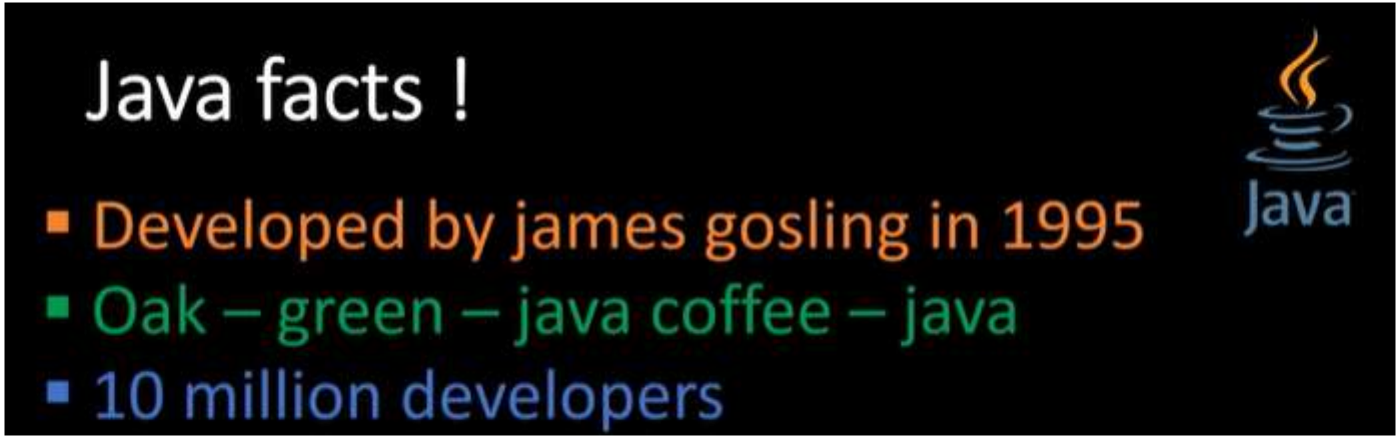


Let’s dive deep into the plethora of useful **Java Basics**



WHAT IS JAVA ?

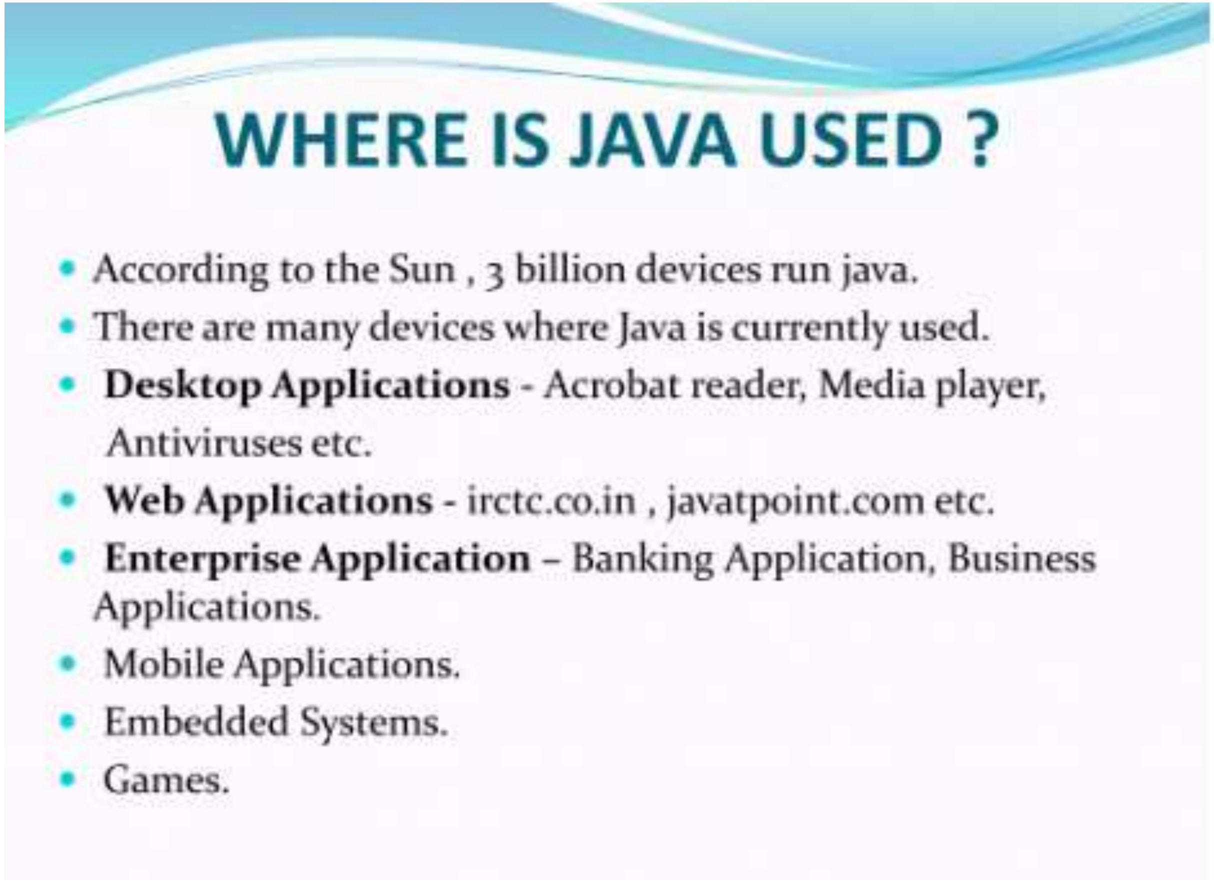
- Java is **Object Oriented Programming language** as well as Plateform.
- Java was developed by a team led by James Gosling at Sun Microsystems in 1991.
- Java is a first programming language which provide the concept of writing programs that can be executed using the web.


Java facts !

- Developed by james gosling in 1995
- Oak – green – java coffee – java
- 10 million developers

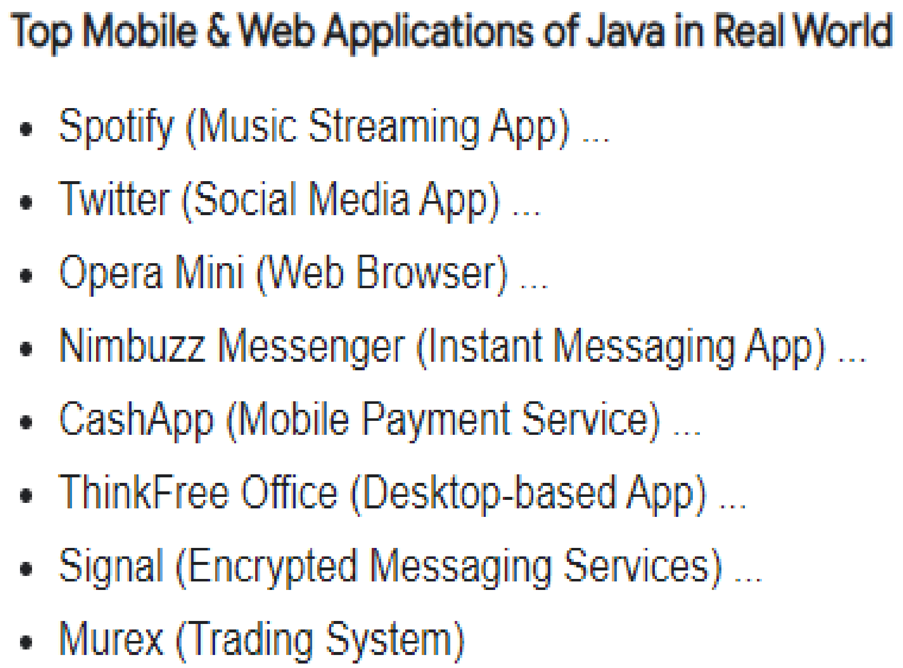
Java and Android

Java practically runs on 1billion plus smartphones today because Google's Android operating system uses Java APIs.



WHERE IS JAVA USED ?

- According to the Sun , 3 billion devices run java.
- There are many devices where Java is currently used.
- **Desktop Applications** - Acrobat reader, Media player, Antiviruses etc.
- **Web Applications** - irctc.co.in , javatpoint.com etc.
- **Enterprise Application** – Banking Application, Business Applications.
- Mobile Applications.
- Embedded Systems.
- Games.



Top Mobile & Web Applications of Java in Real World

- Spotify (Music Streaming App) ...
- Twitter (Social Media App) ...
- Opera Mini (Web Browser) ...
- Nimbuzz Messenger (Instant Messaging App) ...
- CashApp (Mobile Payment Service) ...
- ThinkFree Office (Desktop-based App) ...
- Signal (Encrypted Messaging Services) ...
- Murex (Trading System)

NASA World Wind

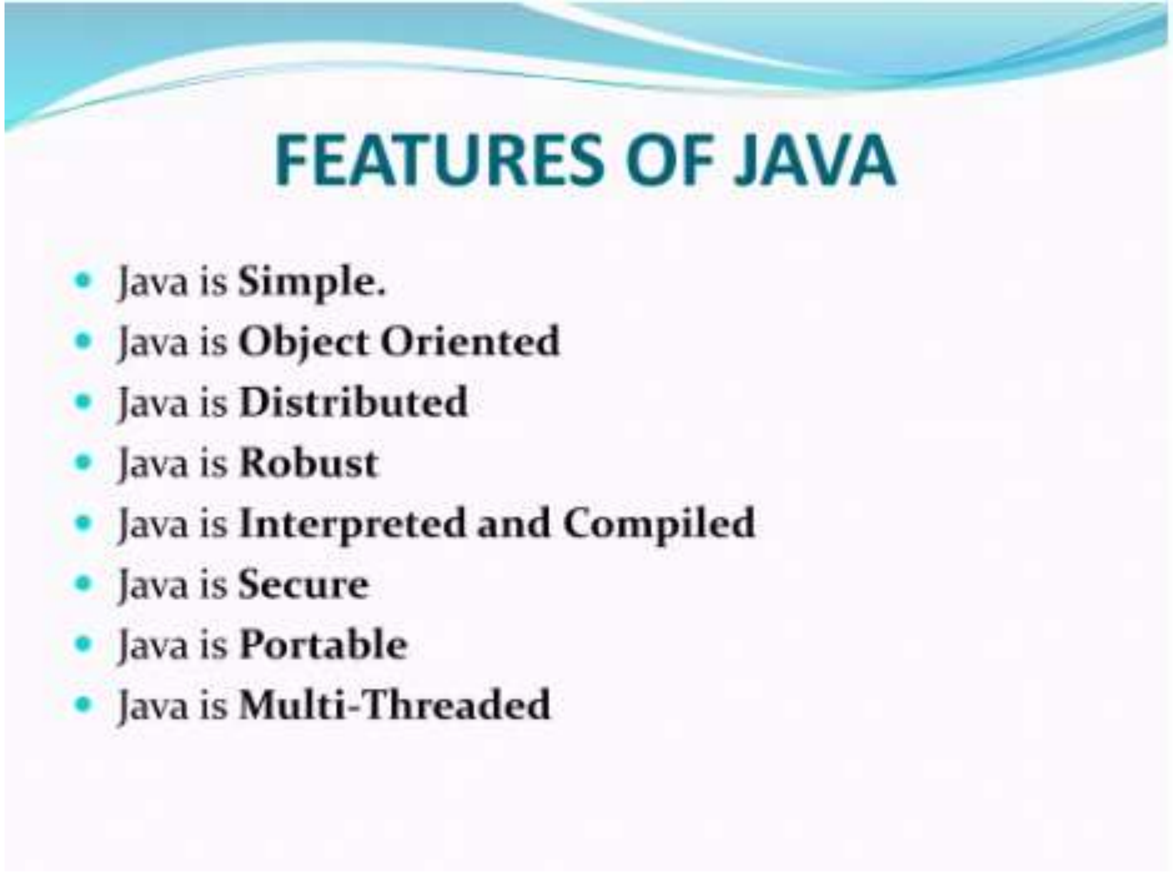
Google & Android OS

NETFLIX (JAVA & Python)

LinkedIn

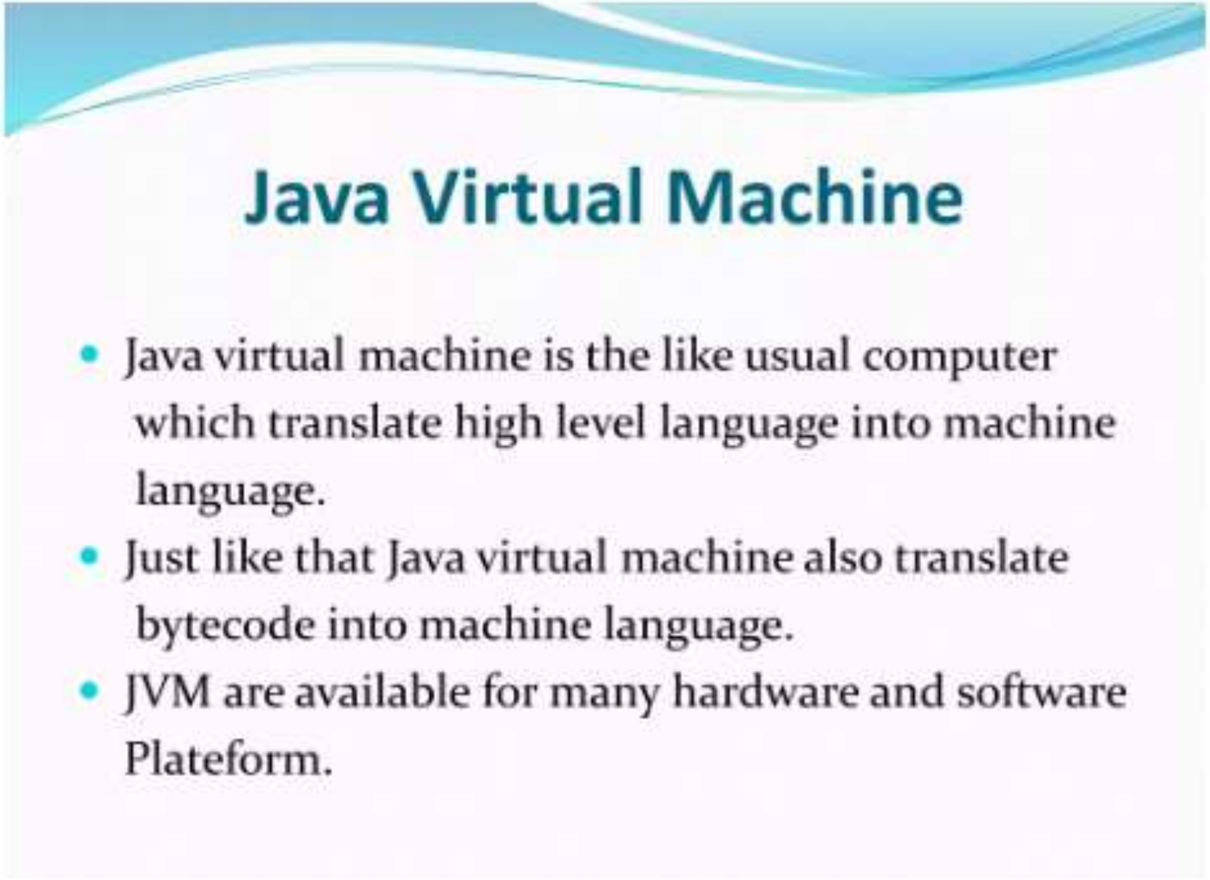
Uber

Amazon



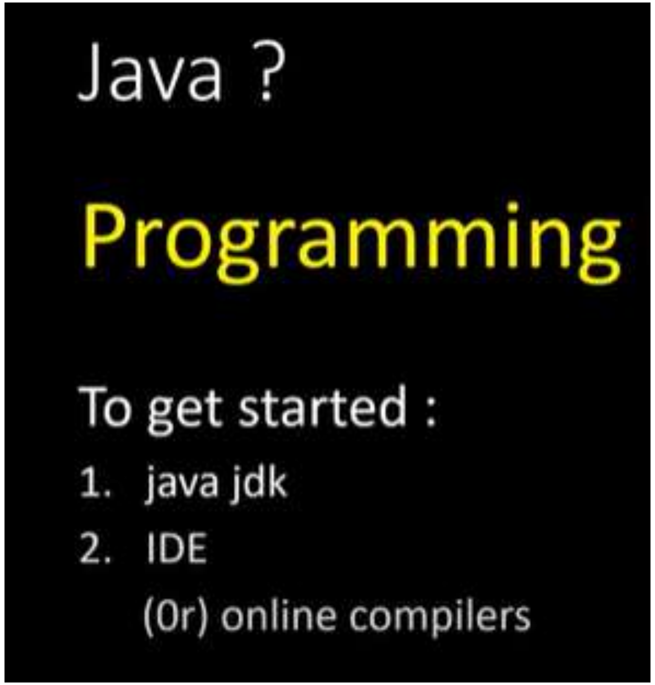
FEATURES OF JAVA

- Java is **Simple.**
- Java is **Object Oriented**
- Java is **Distributed**
- Java is **Robust**
- Java is **Interpreted and Compiled**
- Java is **Secure**
- Java is **Portable**
- Java is **Multi-Threaded**



Java Virtual Machine

- Java virtual machine is the like usual computer which translate high level language into machine language.
- Just like that Java virtual machine also translate bytecode into machine language.
- JVM are available for many hardware and software Plateform.

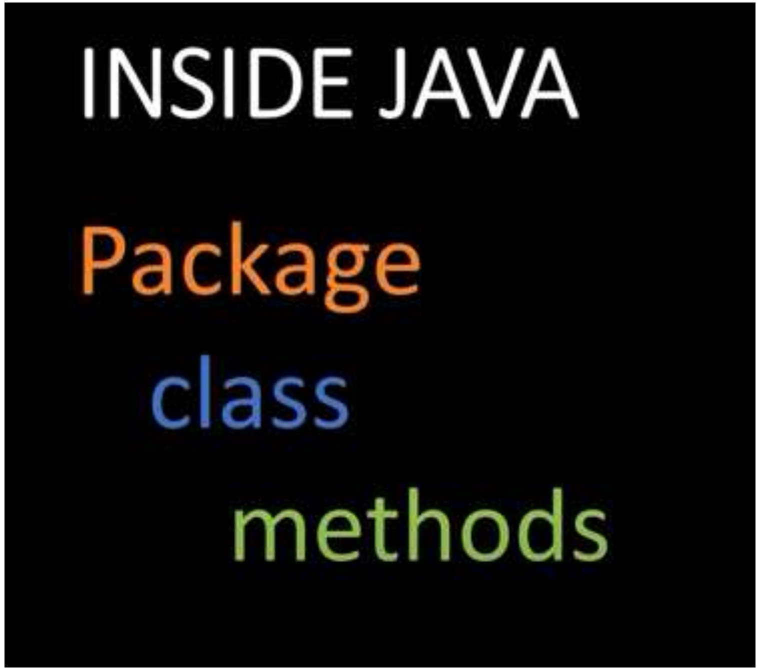

Java ?

Programming

To get started :

1. java jdk
2. IDE

   (0r) online compilers


INSIDE JAVA

Package

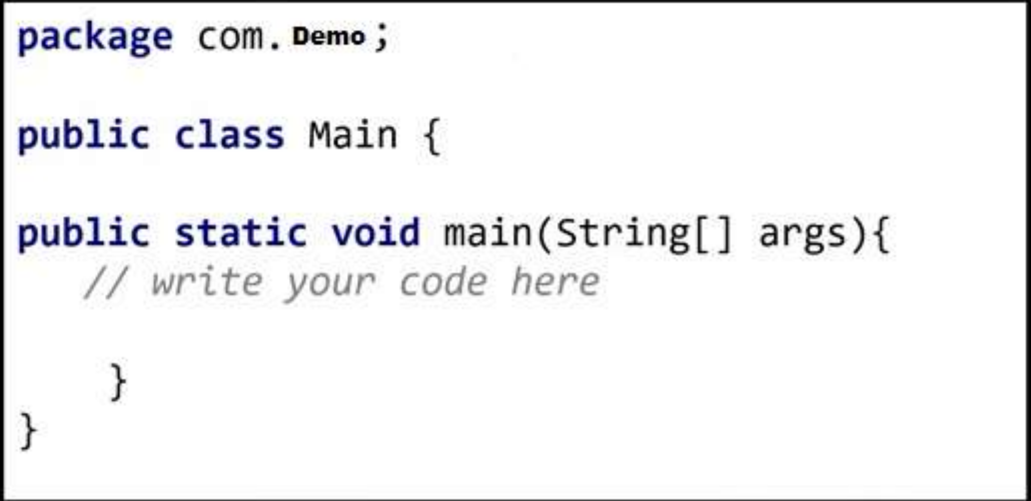
class

methods

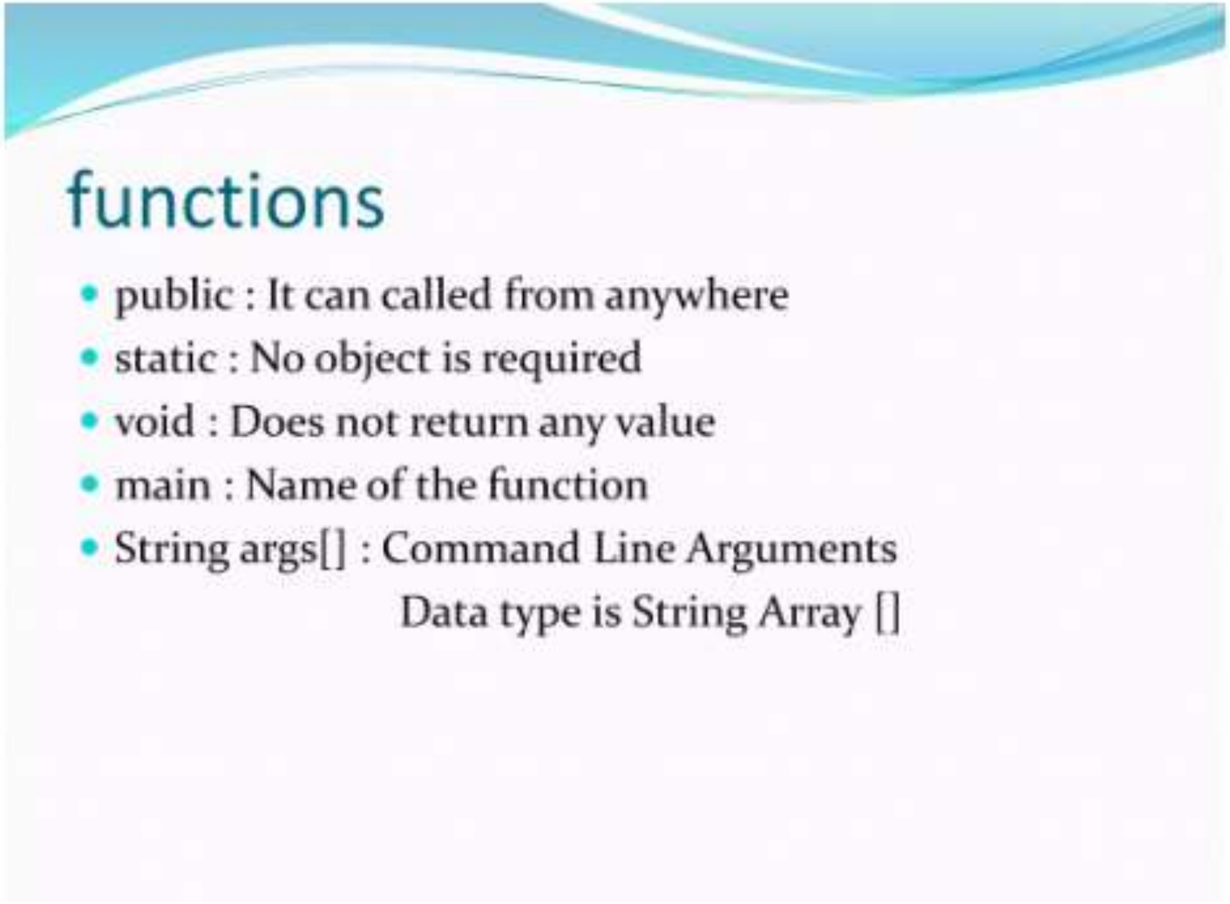
JAVA PROGRAM

.java file

Structure ?



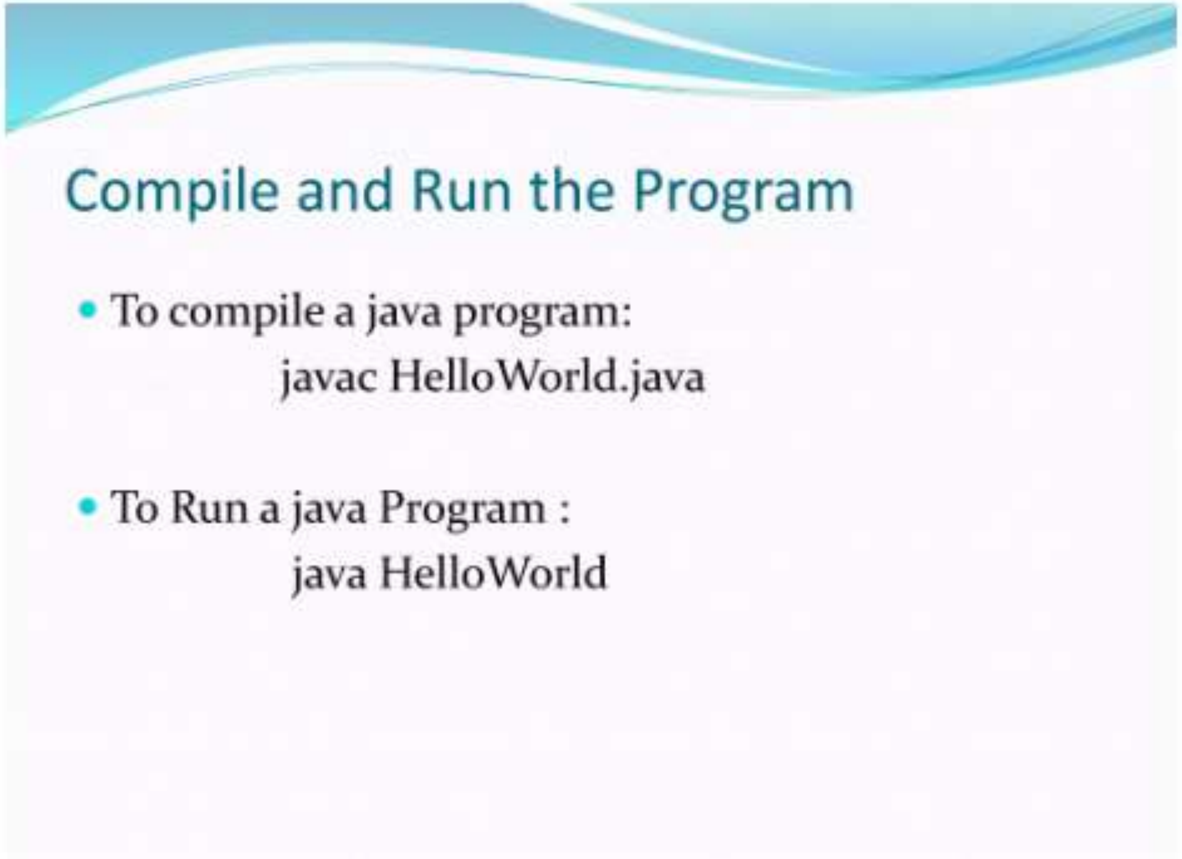
```
package com.Demo;

public class Main {

public static void main(String[] args){
   // write your code here

    }
}
```



functions

- public : It can called from anywhere
- static : No object is required
- void : Does not return any value
- main : Name of the function
- String args[] : Command Line Arguments

  Data type is String Array []



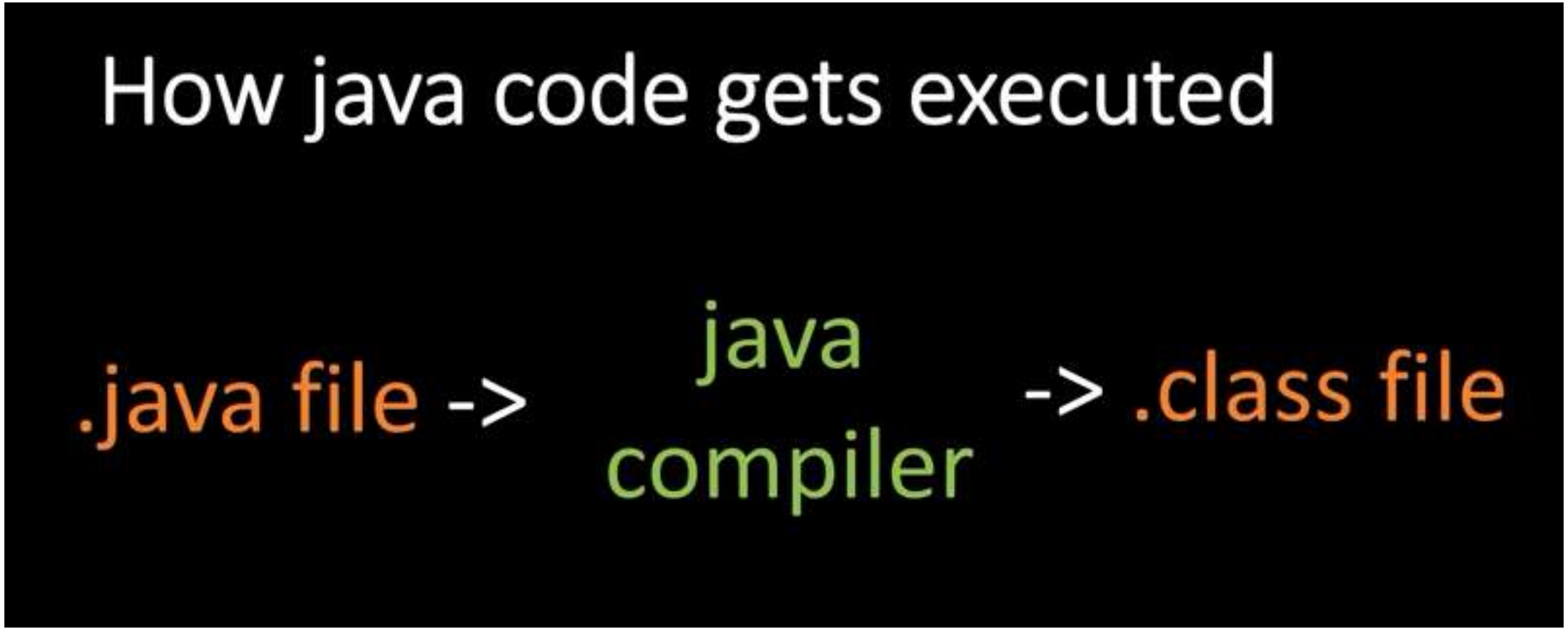
Compile and Run the Program

- To compile a java program:

```
javac HelloWorld.java
```

- To Run a java Program :

```
java HelloWorld
```



How java code gets executed

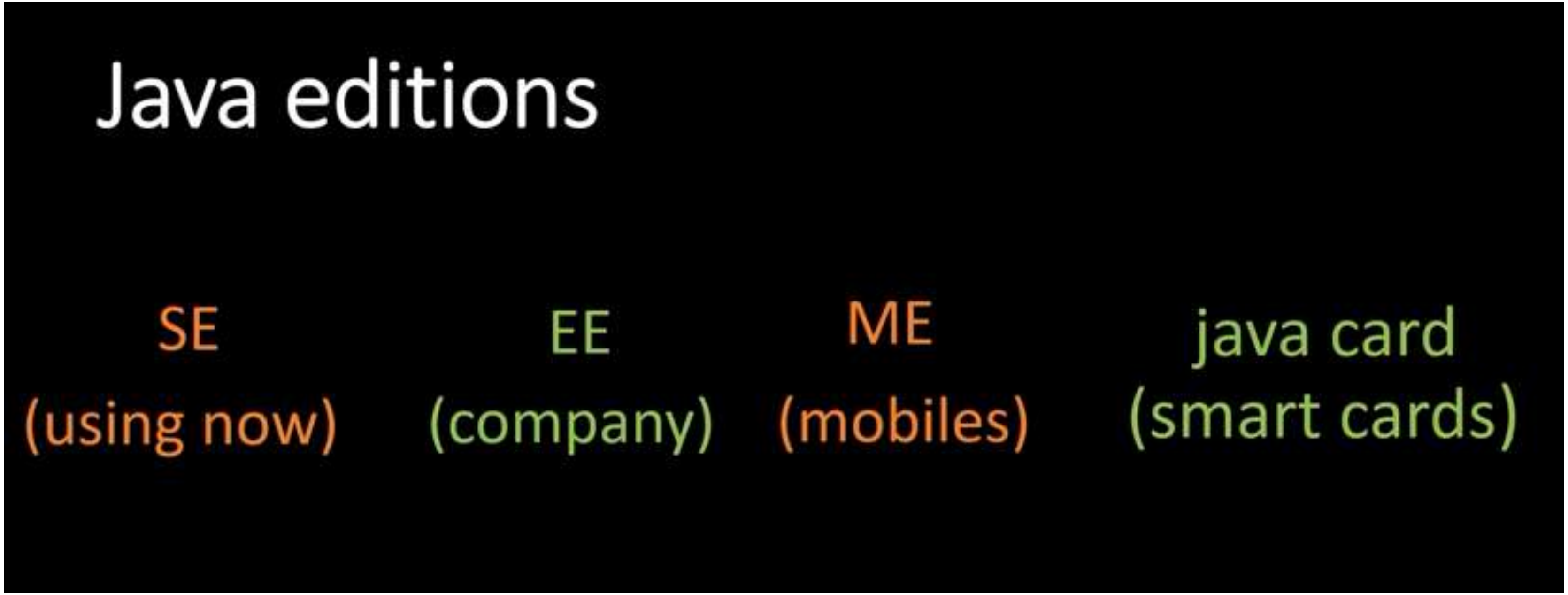
.java file -> java compiler -> .class file

Actually the fact is Java platform independent then but JVM (Java Virtual Machine) is platform dependent.
There are two tools which use for compiling and running Java programs

javac -It is a compiler which converts Java source code to byte code that is a .class file. This byte code is standard for all platforms, machines or operating systems.

Java – It is an interpreter. This interprets the .class file based on a particular platform and executes them.

Jvm -Java virtual machine comes into play. Jvm for windows will be different from Jvm for Solaris or Linux. But all the Jvm take the same byte code and executes them in that platform.



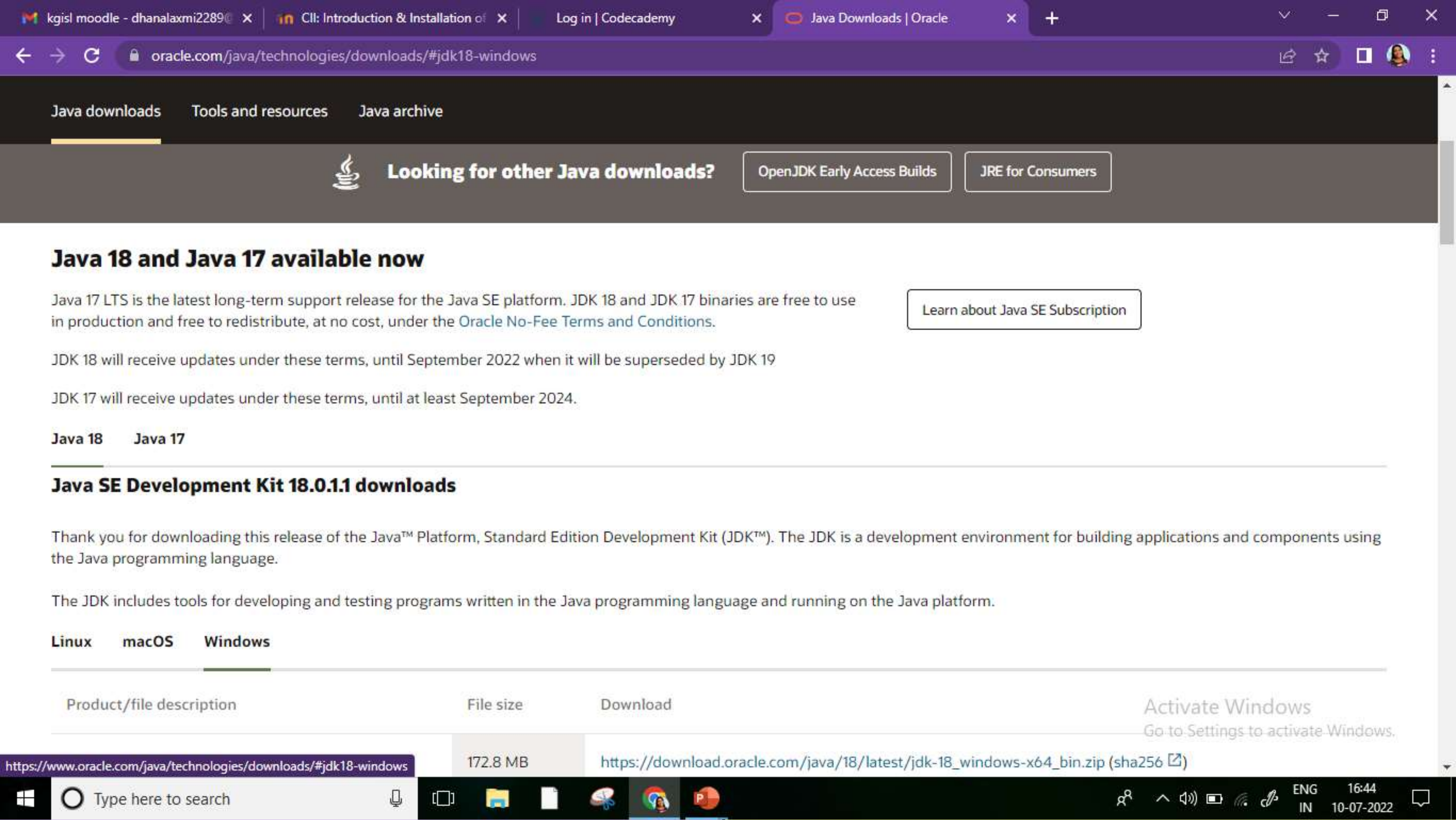
Java editions

SE (using now)

EE (company)

ME (mobiles)

java card (smart cards)


kgisl moodle - dhanalaxmi2289
CII: Introduction & Installation of
Log in | Codecademy
Java Downloads | Oracle
oracle.com/java/technologies/downloads/#jdk18-windows
Java downloads
Tools and resources
Java archive
Looking for other Java downloads?
OpenJDK Early Access Builds
JRE for Consumers
Learn about Java SE Subscription
Activate Windows
Go to Settings to activate Windows.
https://www.oracle.com/java/technologies/downloads/#jdk18-windows
Type here to search
ENG
IN
16:44
10-07-2022

Java 18 and Java 17 available now

Java 17 LTS is the latest long-term support release for the Java SE platform. JDK 18 and JDK 17 binaries are free to use in production and free to redistribute, at no cost, under the Oracle No-Fee Terms and Conditions.

JDK 18 will receive updates under these terms, until September 2022 when it will be superseded by JDK 19

JDK 17 will receive updates under these terms, until at least September 2024.

Java 18 Java 17

Java SE Development Kit 18.0.1.1 downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and components using the Java programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

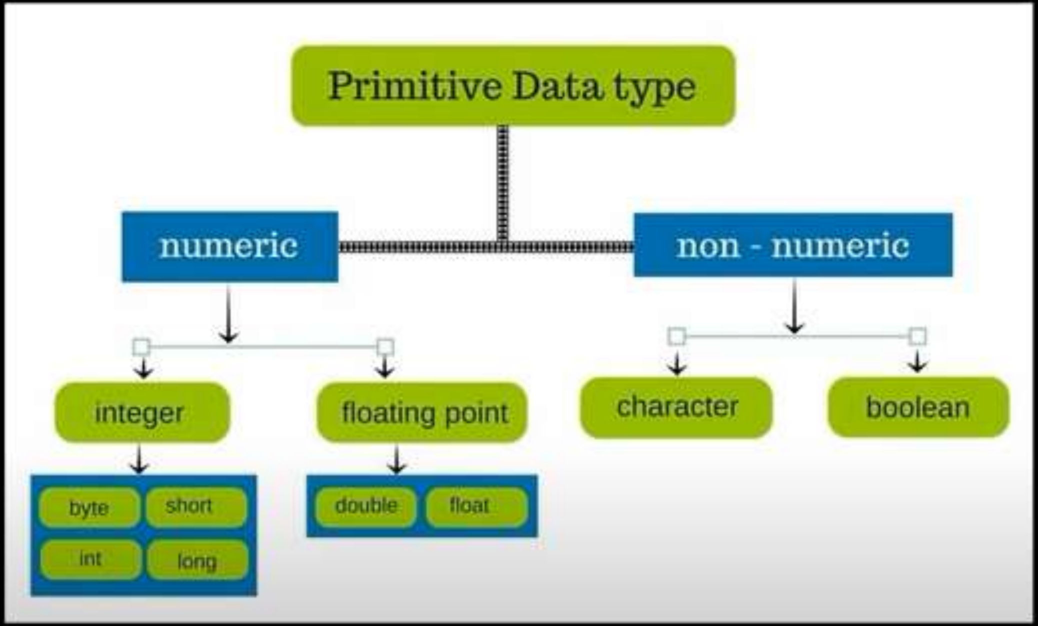
Linux macOS Windows

Product/file description	File size	Download
	172.8 MB	https://download.oracle.com/java/18/latest/jdk-18_windows-x64_bin.zip (sha256)

Data types

Two categories :

- Primitive
- Reference



Data Type	Size	Description
`byte`	1 byte	Stores whole numbers from -128 to 127
`short`	2 bytes	Stores whole numbers from -32,768 to 32,767
`int`	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
`double`	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
`boolean`	1 bit	Stores true or false values
`char`	2 bytes	Stores a single character/letter or ASCII values

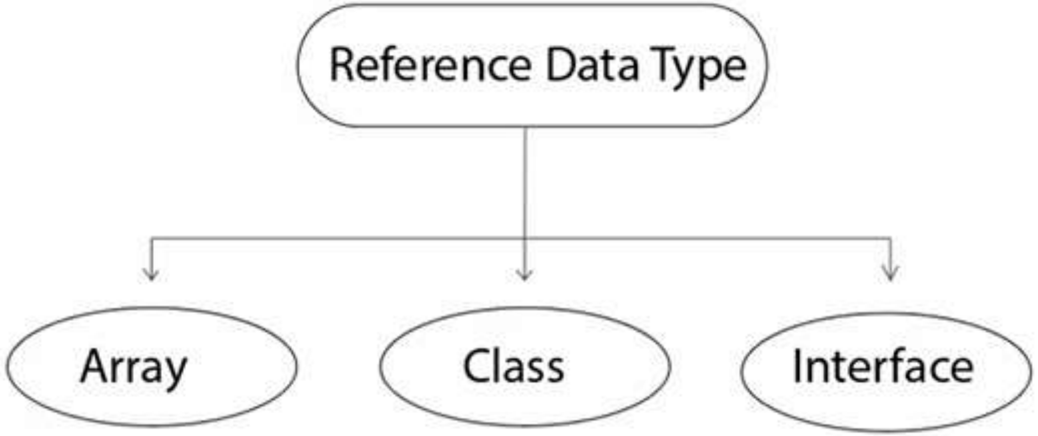
Non-Primitive Data Types

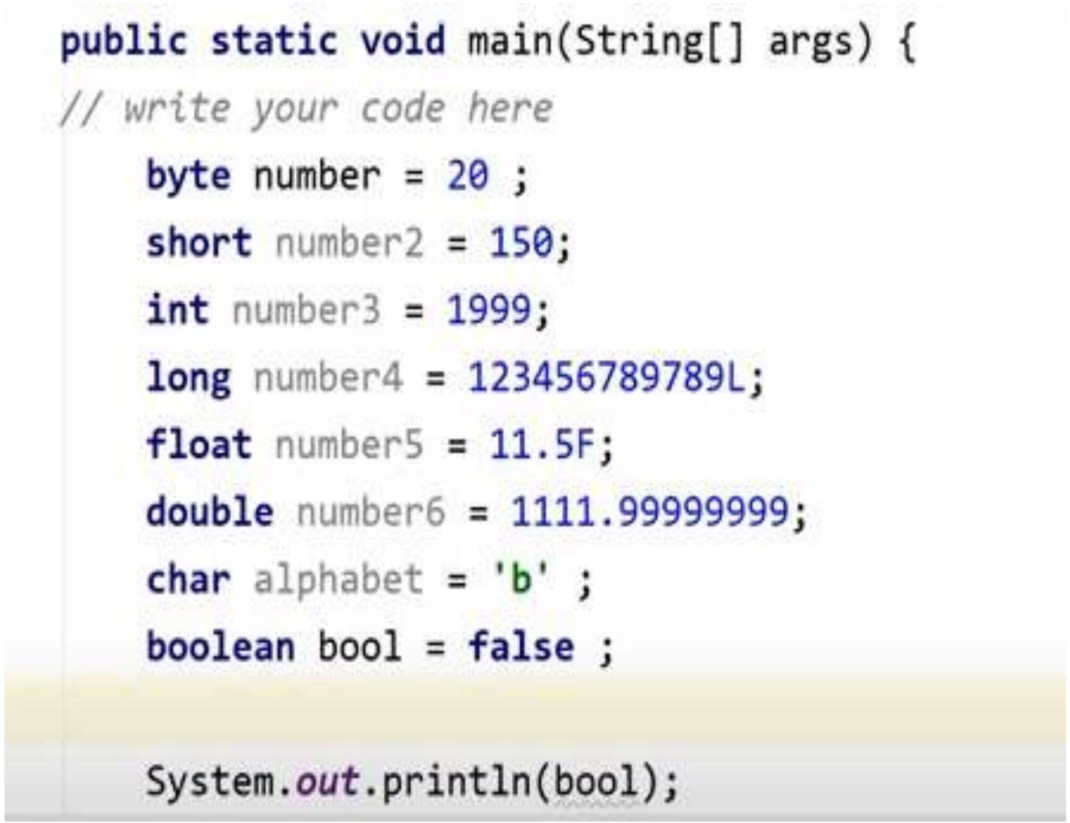
Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **primitive** and **non-primitive** data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for `String`).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be `null`.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc. You will learn more about these in a later chapter.


Reference Data Type
Array
Class
Interface



```java
public static void main(String[] args) {
// write your code here
    byte number = 20 ;
    short number2 = 150;
    int number3 = 1999;
    long number4 = 123456789789L;
    float number5 = 11.5F;
    double number6 = 1111.99999999;
    char alphabet = 'b' ;
    boolean bool = false ;

    System.out.println(bool);
```

Primitive Data Types - float

The below program declares a float variable called **price** and assigns a value of **20.12** to it.
The **f in the end** indicates it is a float data type.

```
public class Hello {

    public static void main(String[] args) {
        float price=20.12f;
        System.out.println(price);
    }
}
```

Primitive Data Types - double

The below program declares a double variable called **hike** and assigns a value of **120.12** to it.

```
public class Hello {


    public static void main(String[] args) {
        double hike=120.12;
        System.out.println(hike);
    }
}
```

By default any floating point value is considered as double but you can also specify d in the end to indicate that the data type is double.

Hence **double hike=120.12;** and **double hike=120.12d;** are same.

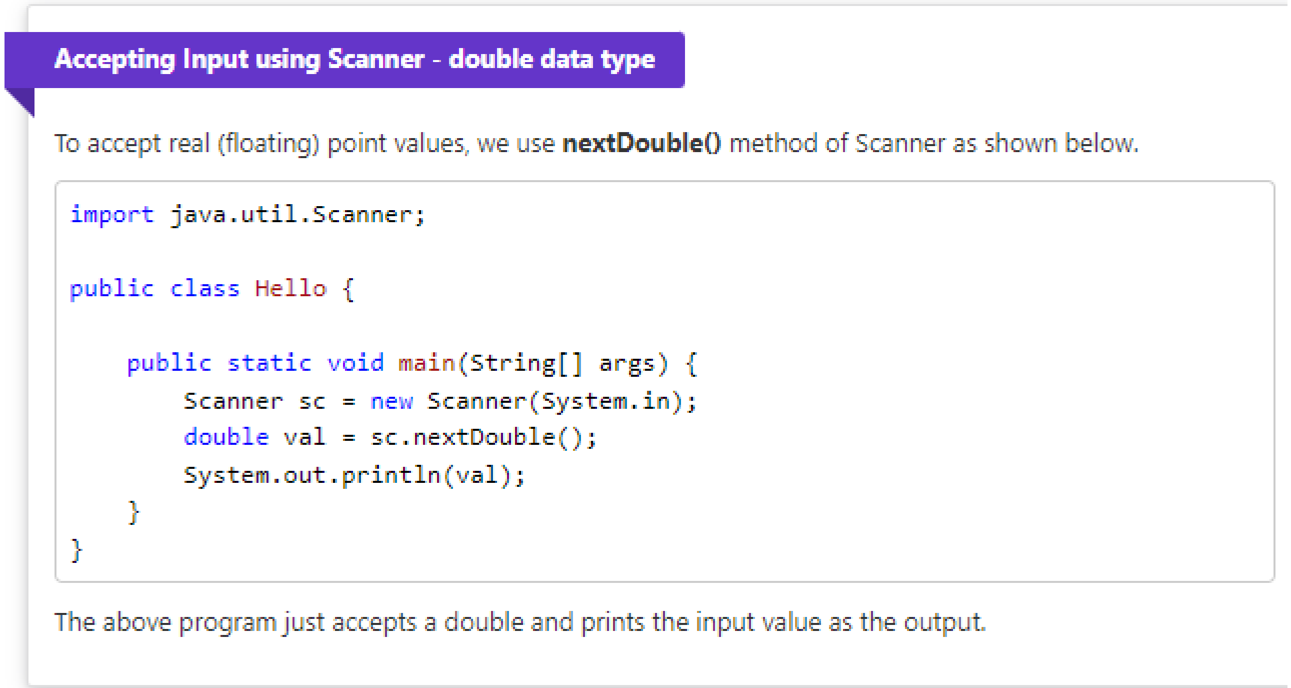
Accepting Input using Scanner - int data type

Till now in our examples we hardcoded values (also called literals). But in practice, the program must accept input from a source.

Scanner is an important class whose instances are useful for breaking down formatted input into tokens and translating individual tokens according to their data type.

As an example the below program reads an int data type, adds 100 to it and prints the new value as the output.

```
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int x = sc.nextInt();
        System.out.println(x + 100);
    }
}
```



Accepting Input using Scanner - double data type

To accept real (floating) point values, we use **nextDouble()** method of Scanner as shown below.

```java
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double val = sc.nextDouble();
        System.out.println(val);
    }
}
```

The above program just accepts a double and prints the input value as the output.

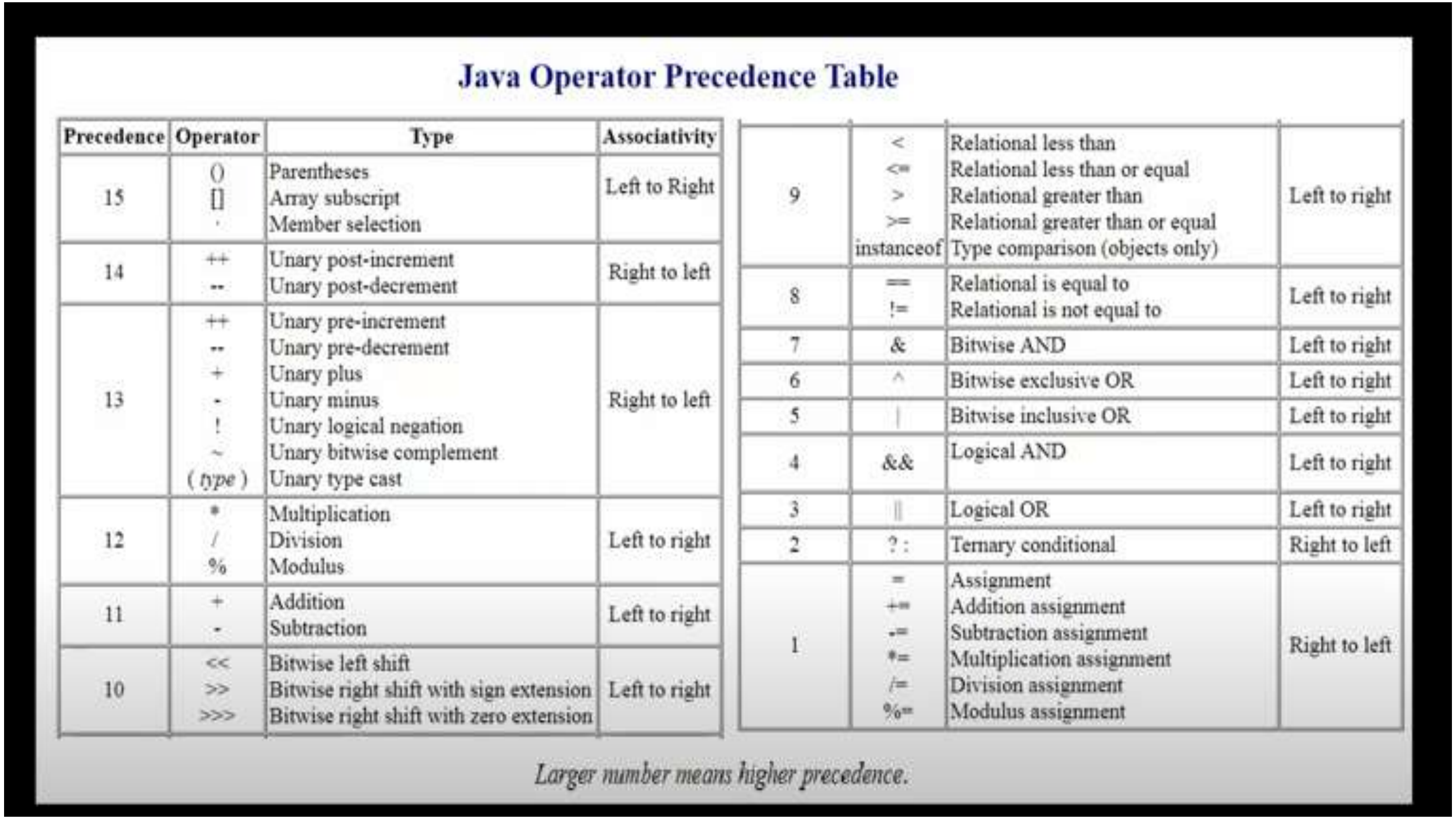
Formatting output - decimal places

When printing data types like double and float as output, we can format the values upto certain decimal places.

As an example, the below program rounds up the output value upto 2 decimal places.

```java
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double price = sc.nextDouble();
        System.out.format("%.2f", price);
    }
}
```

If the input to the above program is 12.2566, the output is 12.26 (rounded upto 2 decimal places).
If the input to the above program is 5.1249, the output is 5.12 (rounded upto 2 decimal places).



Java Operator Precedence Table

Precedence	Operator	Type	Associativity
15	() [] ·	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Larger number means higher precedence.

Arithmetic Operators

+ Additive operator (also used for String concatenation)
- Subtraction operator
*** Multiplication operator**
/ Division operator
% Remainder operator

The below program prints the product of two int values x and y (passed as input) using the multiplication operator *.

```
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int x=sc.nextInt();
        int y=sc.nextInt();
        System.out.println(x*y);
    }
}
```

Remainder operator

The remainder operator % is used to find the remainder when a number is divided by another number.

The below program prints the remainder when x is divided by y.

```java
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int x=sc.nextInt();
        int y=sc.nextInt();
        System.out.println(x%y);
    }
}
```

Mixing arithmetic operators

Several arithmetic operators can be used in a single statement.

As an example the below program calculates the sum of the values of two times x and one-third of y. Then it subtracts 2 from the resulting value.

```java
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int x=sc.nextInt();
        int y=sc.nextInt();
        int result = 2*x + y/3 -2;
        System.out.println(result);

    }
}
```

Equality Operator

We use two equal symbols to check for equality of primitive variables like int.
As an example, the program below prints true if first number is equal to second number.

```java
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int firstNum = sc.nextInt();
        int secondNum = sc.nextInt();

        boolean equal = (firstNum == secondNum);
        System.out.println(equal);
    }
}
```

Note: Remember that a single = is an assignment operator

Unary Operators

+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

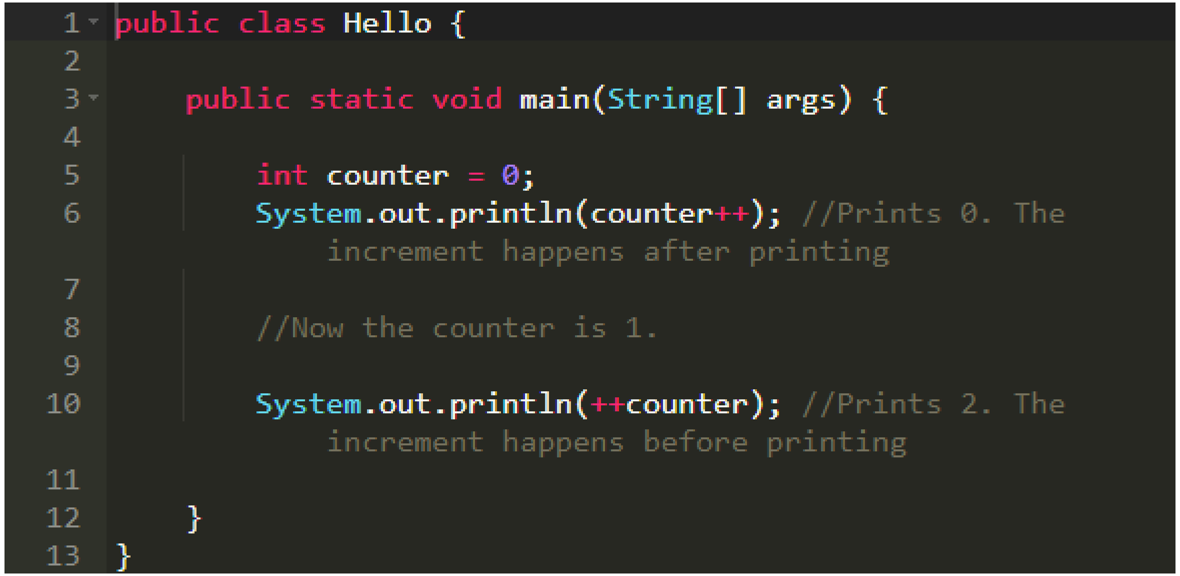
The increment/decrement operators can be applied before (prefix) or after (postfix) the operand. The code result++; and ++result; will both end in result being incremented by one. The only difference is that the prefix version (++result) evaluates to the incremented value, whereas the postfix version (result++) evaluates to the original value.

The below program illustrates the way prefix and postfix operators work.

```java
public class Hello {

    public static void main(String[] args) {
        int i = 3;
        i++;
        // prints 4
        System.out.println(i);
        ++i;
        // prints 5
        System.out.println(i);
        // prints 6
        System.out.println(++i);
        // prints 6
        System.out.println(i++);
        // prints 7
        System.out.println(i);

    }
}
```

Unary Increment Operator

Unary Increment operator is used to **increase the value by 1.**
It can be used before or after a variable as exhibited in the below program.



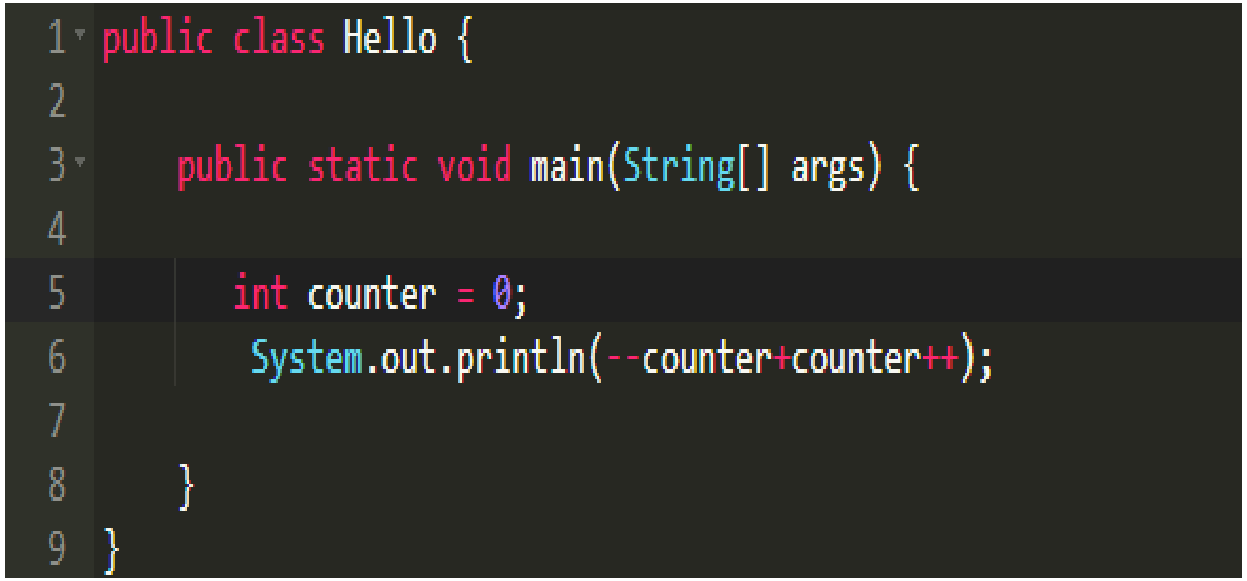
```java
public class Hello {

    public static void main(String[] args) {

        int counter = 0;
        System.out.println(counter++); //Prints 0. The increment happens after printing

        //Now the counter is 1.

        System.out.println(++counter); //Prints 2. The increment happens before printing

    }
}
```

Unary Decrement Operator

Unary Decrement operator is used to decrease the value by 1.
It can be used before or after a variable as exhibited in the below program.

```
public class Hello {

    public static void main(String[] args) {

        int counter = 5;
        System.out.println(--counter); //Prints 4
        System.out.println(counter--); //Prints 4
        System.out.println(counter); //Prints 3

    }
}
```

Eg:



```java
public class Hello {

    public static void main(String[] args) {

        int counter = 0;
         System.out.println(--counter+counter++);

    }
}
```

if else

if else condition can be used for either or situation.
As an example, the program checks if the number passed as input is odd or even.

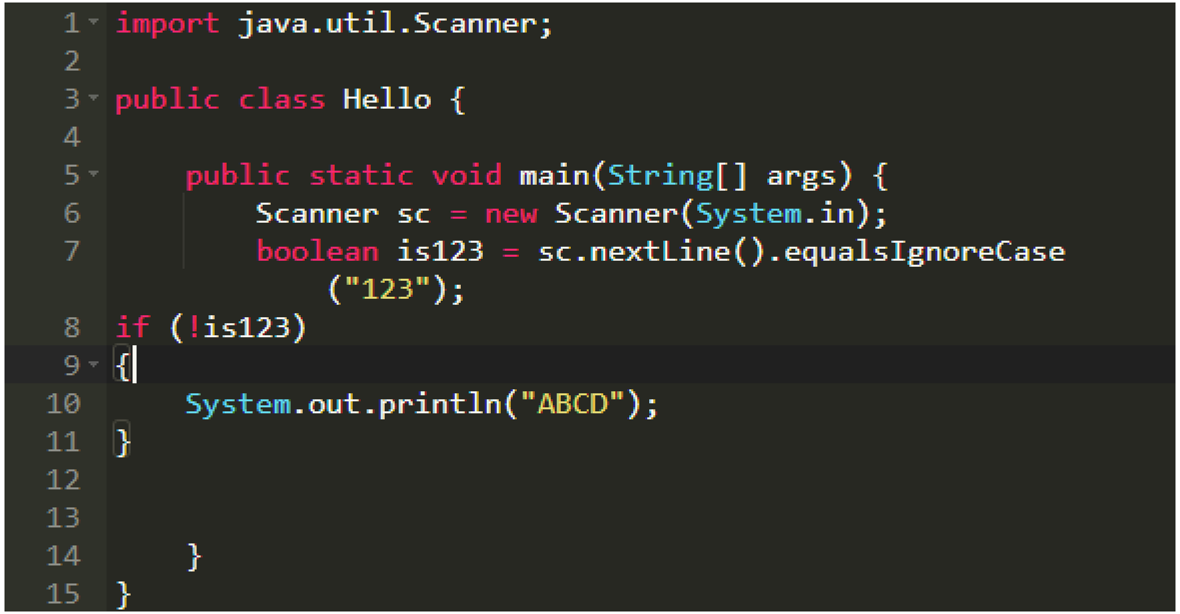
```
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        int remainder = number % 2;

        if (remainder == 0) {
            System.out.println("even");
        } else {
            System.out.println("odd");
        }
    }
}
```

Negation Operator

Negation operator (also called **Logical Complement** operator) is used to **reverse the boolean value** (true or false).

The below program prints "small" if the input number is less than 100 using the negation operator.

```java
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int number = sc.nextInt();
        boolean greaterThanEqualToHundred = number >= 100;

        //NEGATION operator being used
        if (!greaterThanEqualToHundred) {
            System.out.println("small");
        }


    }
}
```



```
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        boolean is123 = sc.nextLine().equalsIgnoreCase
            ("123");
if (!is123)
{
    System.out.println("ABCD");
}


    }
}
```

Logical Operator - AND

To check for AND condition, we use **&&** operator.

The below program prints yes if the input number is divisible by both 4 and 5. Else it prints no.

```
import java.util.Scanner;
public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        if(number%5 == 0 && number%4 == 0){
            System.out.println("yes");
        }else{
            System.out.println("no");
        }
    }
}
```

Logical Operator - OR

To check for OR condition, we use **|| operator**.

The below program prints yes if the input number is divisible by 5 or 4 or 7. Else it prints no.

```java
import java.util.Scanner;
public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        if(number%4==0 || number%5==0 || number%7==0){
            System.out.println("yes");
        }else
        {
            System.out.println("no");
        }

    }
}
```

ternary operator

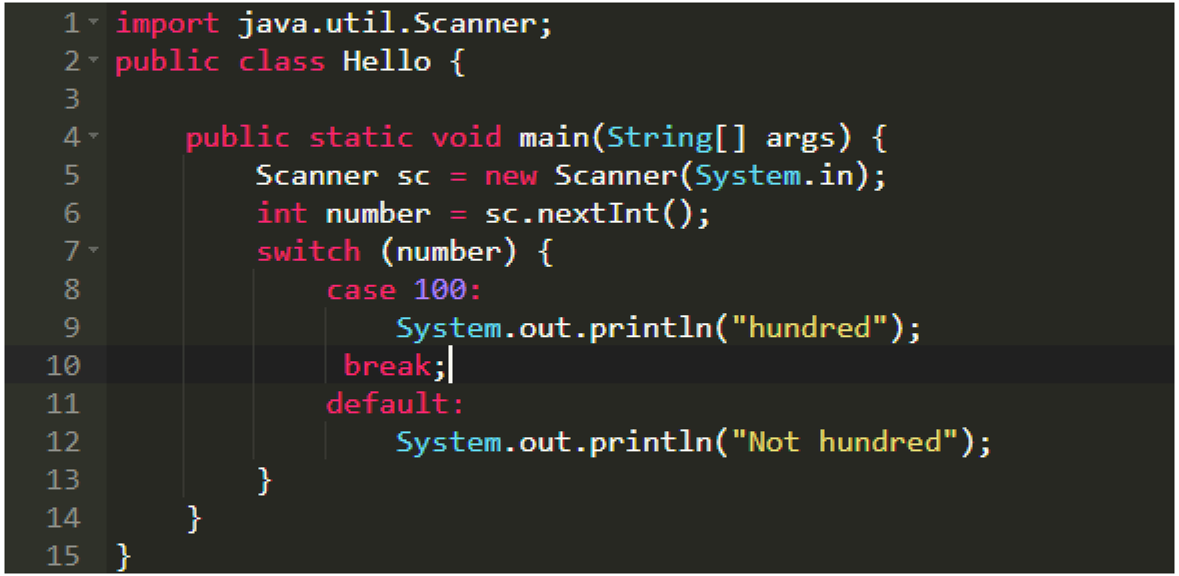
ternary operator can be used instead of simple conditions involving if else.
As an example, the program checks if the number passed as input is positive.

```java
import java.util.Scanner;

public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        String result = number > 0 ? "positive":"notpositive";
        System.out.println(result);

    }
}
```

Switch statement

Switch statement is used to take one of the action based on the input value.
The below program checks if the input number is 1,2,3 and prints "one" or "two" or "three" accordingly.
Else it prints "Not one two three"

```
import java.util.Scanner;
public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        switch(number){
            case 1:
                System.out.println("one");
                break;
            case 2:
                System.out.println("two");
                break;
            case 3:
                System.out.println("three");
                break;
            default:
                System.out.println("Not one two three");
         }
     }
}
```



```java
import java.util.Scanner;
public class Hello {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int number = sc.nextInt();
        switch (number) {
            case 100:
                System.out.println("hundred");
             break;
            default:
                System.out.println("Not hundred");
        }
    }
}
```

Iterating using for and while loops

for and while loops can be used for iteration.
As an example, the program below prints from 1 to 5 using for loop.

```
public class Hello {

    public static void main(String[] args) {
      for(int counter=1;counter<=5;counter++){
          System.out.print(counter+" ");
      }
    }
}
```

The same can be accomplished using while loop as below.

```
public class Hello {

    public static void main(String[] args) {
      int counter=1;
      while(counter<=5){
          System.out.print(counter+" ");
          counter++;
      }
    }
}
```

do while statement

Between while and do-while statements, the only difference is that in do-while after executing the statements atleast once, the condition to loop is checked.
Hence, the **statements within the do block are always executed at least once**.
The below program will print "5" as the code within do block will be executed atleast once. (Even though the check counter > 100 is not true).

```java
public class Hello {

    public static void main(String[] args) {
        int counter = 5;
        do {
            System.out.println(counter);
        } while (counter > 100);
    }
}
```

The program prints just the input number if the number is even.
If the input number is odd, it prints the input number and the next number (which is even).

```
public class Hello {

    public static void main(String[] args) {
        int number = Integer.parseInt(args[0]);
        do {
            System.out.println(number);
        } while (number++ % 2 != 0);
    }
}
```

Please remember that the post decrement operator ++ increases the value of number only after that specific line is executed.

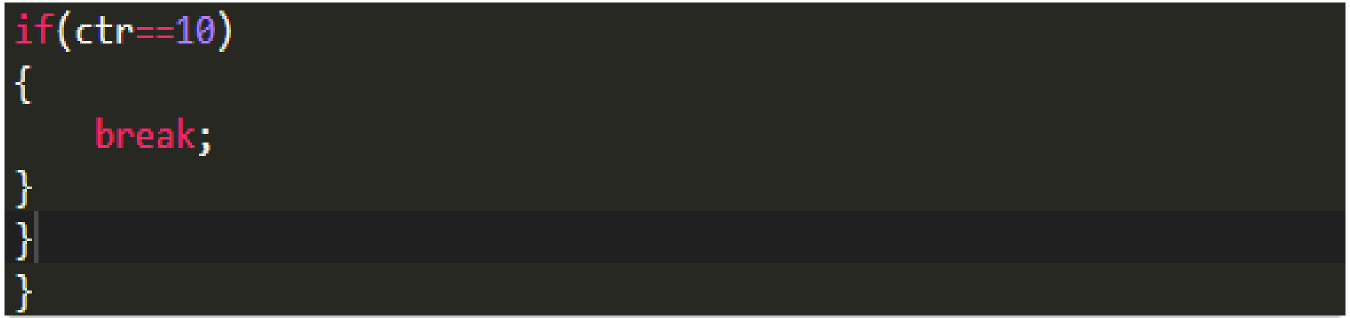
break statement

break statement is used to terminate a for, while, or do-while loop.

In the below program, break is used to terminate the for loop execution when the ctr has reached 5. Thus the program prints only from 1 to 5.

```
public class Hello {

    public static void main(String[] args) {
        for(int ctr=1;ctr<10;ctr++){
            System.out.println(ctr);

            if(ctr==5){
                break;
            }
        }//eof for loop
    }//eof main method
}//eof class
```

```
public class Hello {


    public static void main(String[] args) {
        for(int ctr=2;ctr<=20;ctr+=2){
            System.out.println(ctr);
```



```
if(ctr==10)
{
    break;
}
}
}
```

```
        }//eof for loop
    }//eof main method
}//eof class
```

break statement - nested loops

break statement terminates only the immediate enclosing loop.
In the program below, we have nested for loops.
The break statement **terminates only the for loop with bigcounter.**
Hence the output of the program is

```
1
1000
2000
2
1000
2000
```

```java
public class Hello {

    public static void main(String[] args) {
        for (int smallcounter = 1; smallcounter <= 2; smallcounter++) {
            System.out.println(smallcounter);

            for (int bigcounter = 1000; bigcounter <= 5000; bigcounter += 1000) {
                System.out.println(bigcounter);
                if(bigcounter == 2000){
                    break;
                }
            }

        }//eof for loop
    }//eof main method
}//eof class
```

continue statement

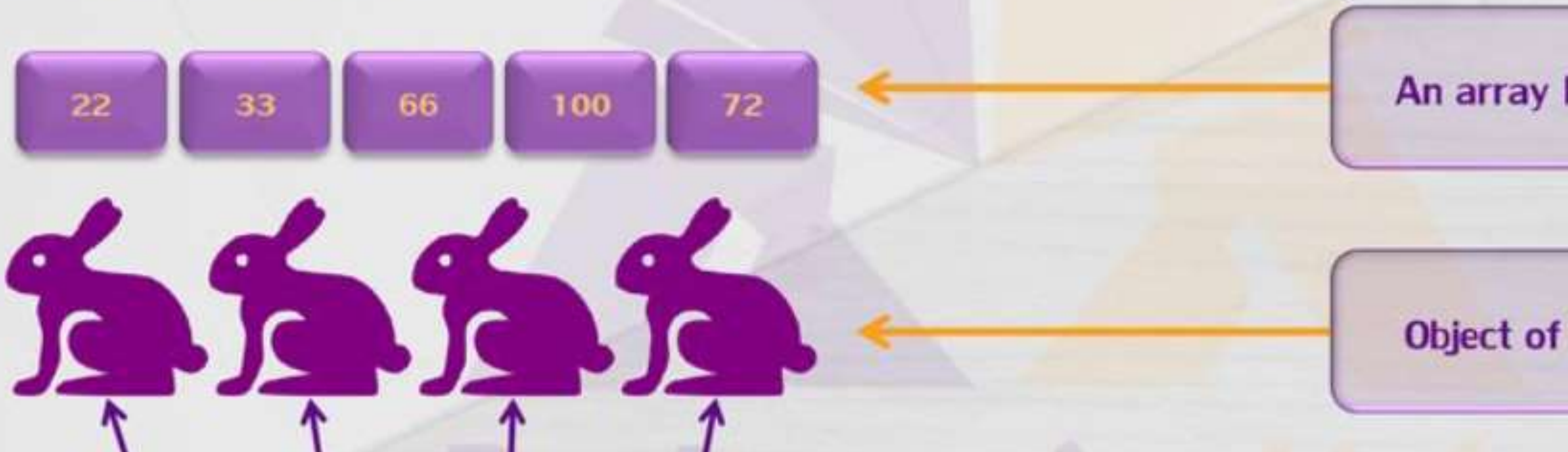
The continue statement **skips** the remaining statements in the current iteration of a for, while, or do-while loop.
In the below program we use continue statement to print only the even numbers from 1 to 10.

```
public class Hello {

    public static void main(String[] args) {
       for(int counter=1;counter<=10;counter++){
           if(counter%2 == 1){
               //DO NOT PRINT ODD NUMBERS
               continue;
           }

           System.out.println(counter);
       }
    }
}
```

Arrays in Java

Arrays in Java

- An array is collection of elements of similar type
- Array elements are always stored in consecutive memory blocks
- Arrays could be of primitive data types or reference type
- Arrays in Java are also objects



Arrays in Java

- Reference variables are used in Java to store the references of objects created by the operator – n
- Any one of the following syntax can be used to create a reference to an int array

```
int x[ ];
int [ ] x;
```

x

null

- Th n be us rring to any int array

```
// Declart a reference to an int array
int [ ] x;
// Create a new int array and make x refer to it
```

Arrays in Java

- The following statement also creates a new int array and assigns its reference to x

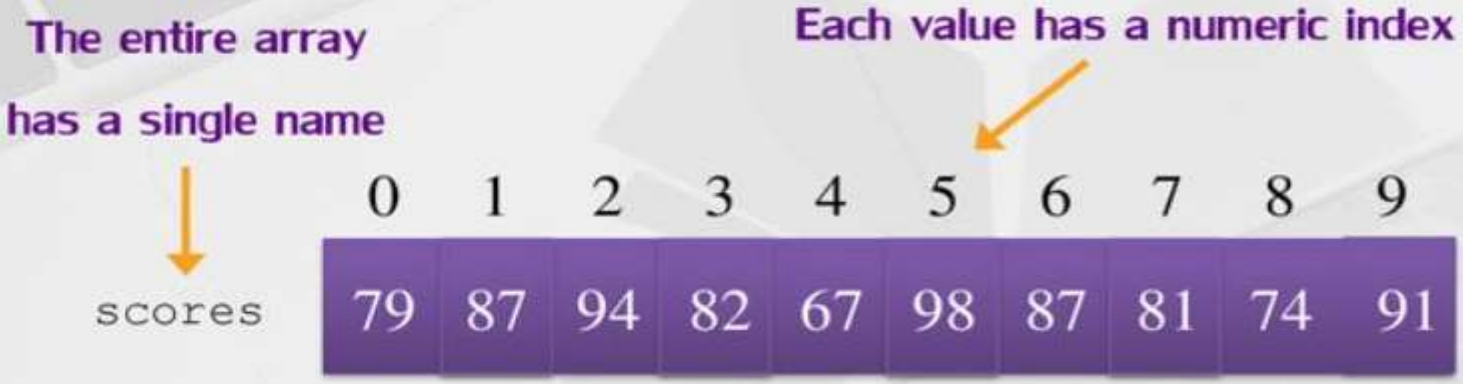
```
int [ ] x = new int [5];
```

- In simple terms, references can be seen as names of an array

Array Object

An array is a groups like typed variables that is referred to a by a common type name. A specific element in an array is accessed by its index. Array offers a convenient meaning of grouping same information.

Arrays in Java



The entire array has a single name
Each value has a numeric index
scores
0 1 2 3 4 5 6 7 8 9
79 87 94 82 67 98 87 81 74 91

An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays in Java

- A particular value in an array is referenced

  using the array name followed by the index in brackets

- For example, the expression

  - scores[2]

  refers to the value 94 (the 3rd value in the array)

- That expression represents a place to store a single integer and can be used wherever an variable can be used.

Declaring Arrays

The scores array could be declared as follows:

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

The type of the variable scores is int[] (an array of integers)

Note that the array type does not specify its size, but each object of that type has a sp
size

The reference variable scores is set to a new array object that can hold 10 integers

An array is an object, therefore all the values are initialized to default ones (here 0)

Array Example

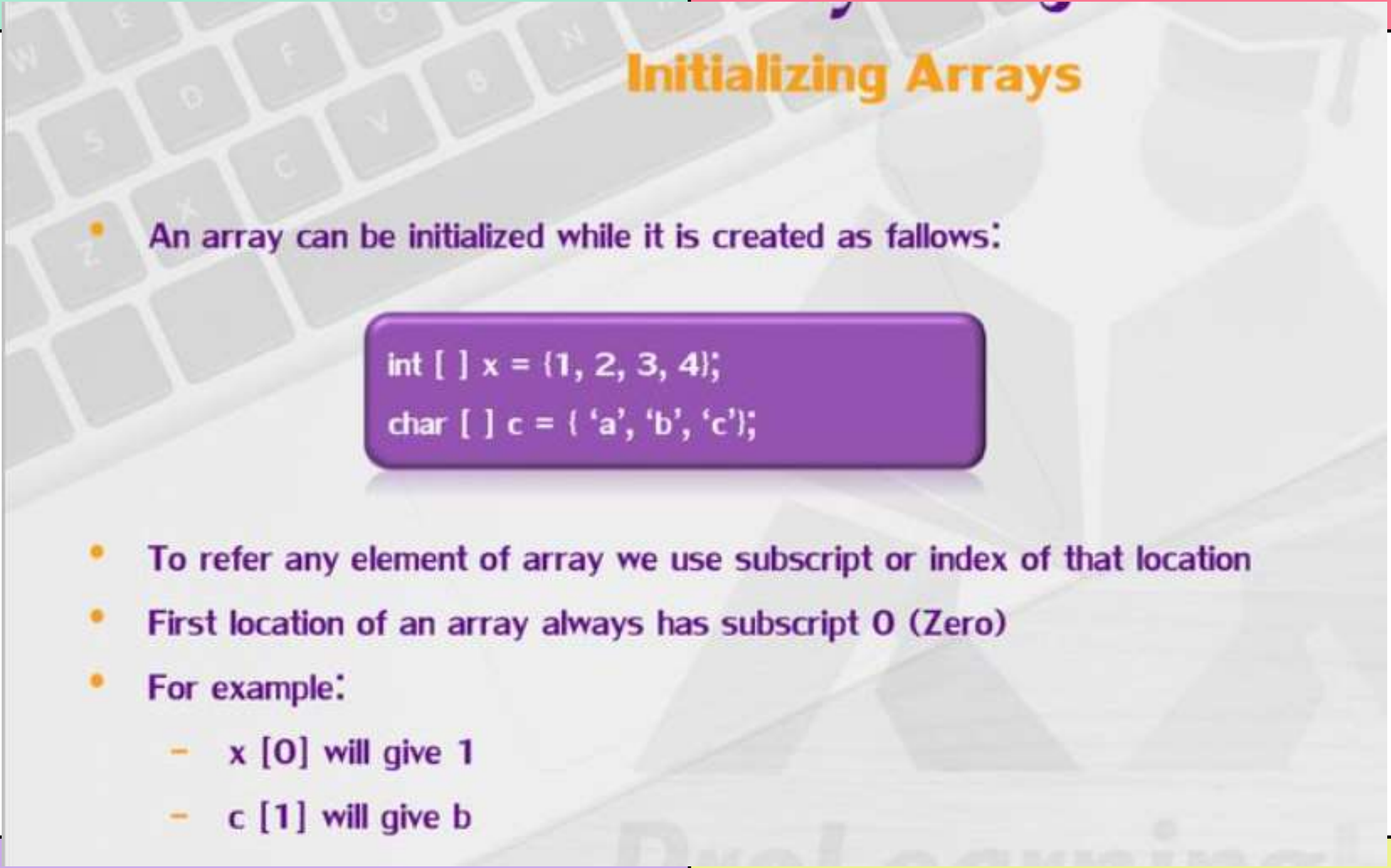
An array element can be assigned a value, printed, or used in a calculation:

```
scores[2] = 89;

scores[first] = scores[first] + 2;

mean = (scores[0] + scores[1])/2;

System.out.println ("Top = " + scores[5]);
```

Array Example

Another examples of array declarations :

```
float[] prices = new
float[500];boolean[] flags;
flags = new boolean[20];
char[] codes = new char[1750];
```



Initializing Arrays

- An array can be initialized while it is created as fallows:

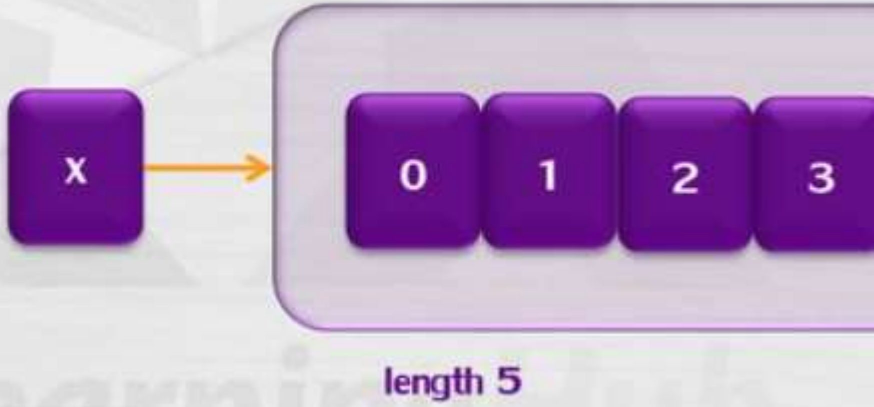
```
int [ ] x = {1, 2, 3, 4};
char [ ] c = { ‘a’, ‘b’, ‘c’};
```

- To refer any element of array we use subscript or index of that location
- First location of an array always has subscript 0 (Zero)
- For example:
  - x [0] will give 1
  - c [1] will give b

Length of an Array

- Unlike C. Java checks the boundary of an array while accessing an element in it
- Programmer is not allowed to exceed its boundary
- And so, setting a for loop as follows is very common:

```
for (int i = 0; i < x.length; ++i) {
        x [i] = 5;
}
```



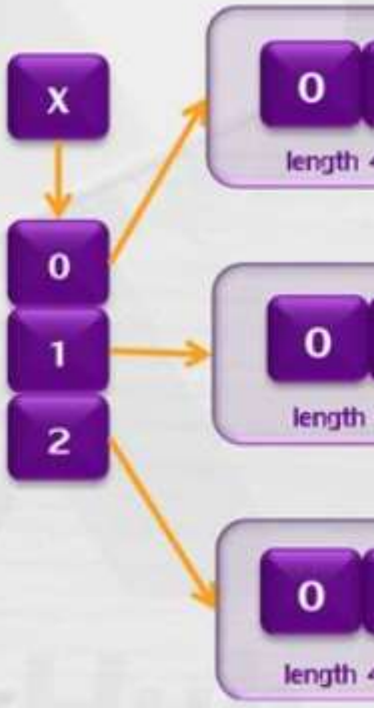
This works for any size array

Array Example

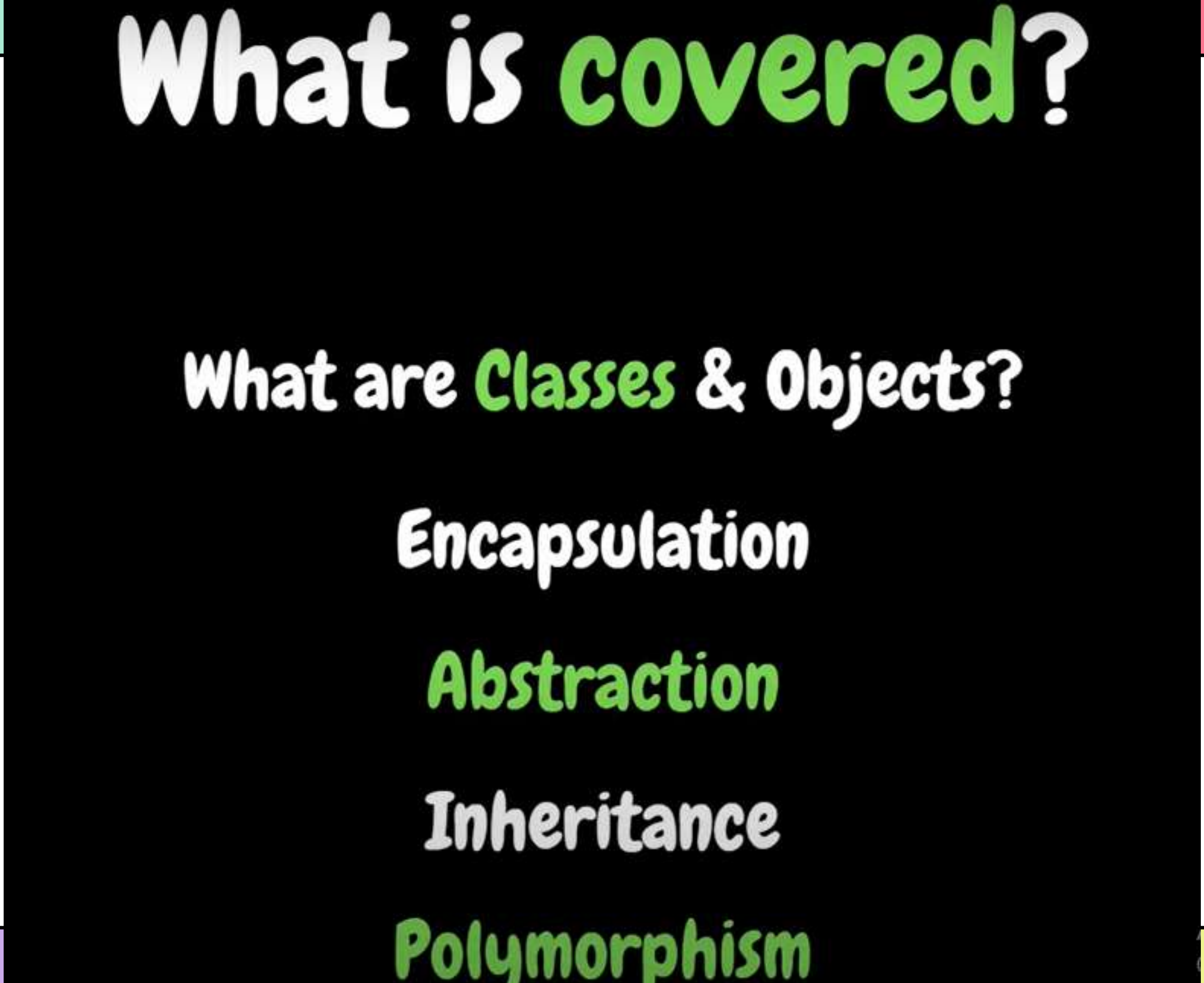
```
public class ArrayDemo {
    public static void main(String[ ] args) {
        int x[ ] = new int [5];
        // loop to assign the values to array
        for(int i = 0; i < x.length; ++i){
            x[i] =1+2;
        }
        // loop to print the values of array
        for(int i = 0; i < x.length; ++i){
            System.out.println(x[i]);
        }
    }
}
```

Multidimensional Arrays

```
int [ ][ ] x;
//x is a reference to an array of int arrays
x = new int[3][4];
//Create 3 new int arrays, each having 4 elements
//x[0] refers to the first int array, x[l] to the second and so on
//x[0][0] is the first element of the first array
//x.length will be 3
//x[0].length, x[1].length and x[2].length will be 4
```



Introduction to Object Oriented Programming



What is covered?

What are Classes & Objects?

Encapsulation

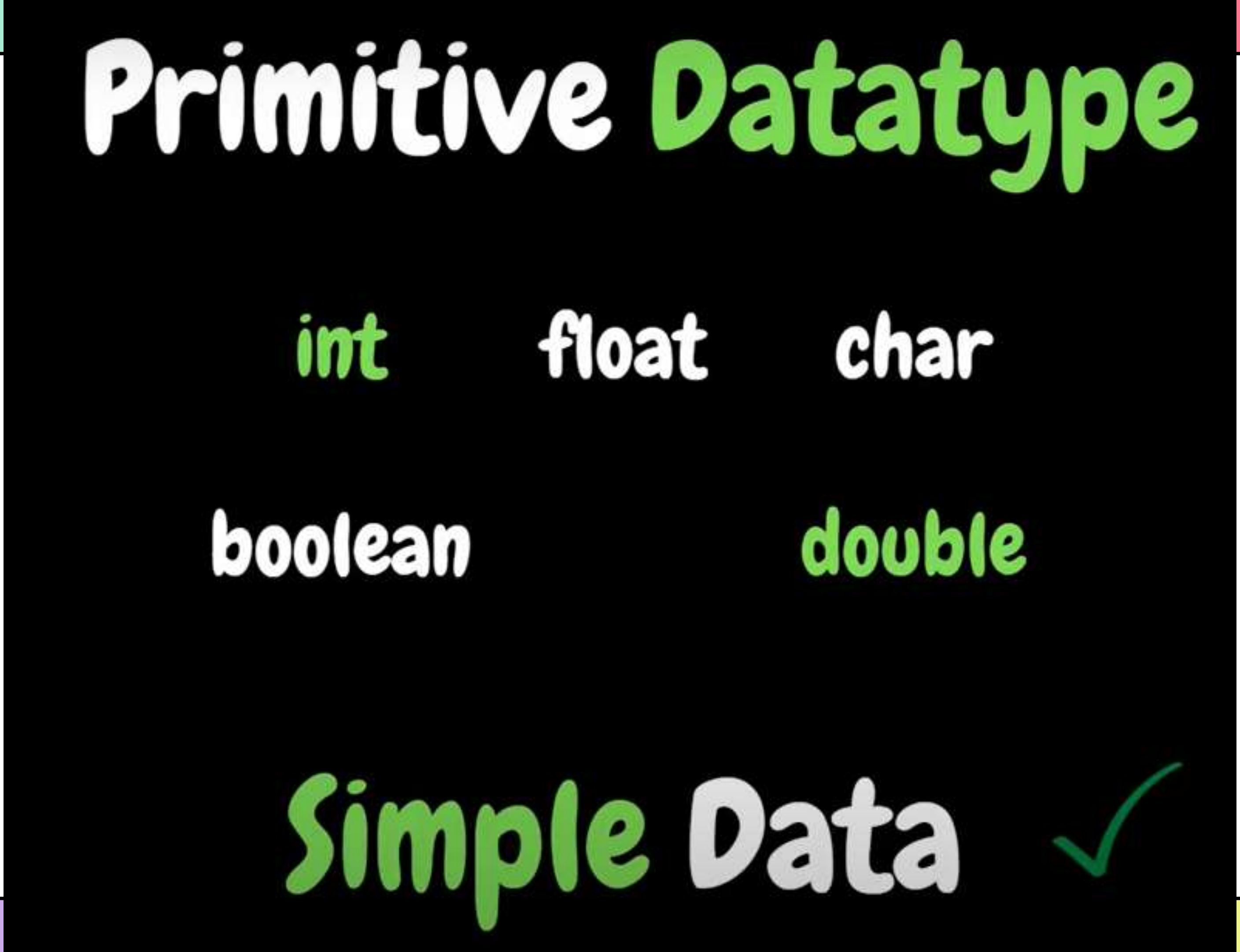
Abstraction

Inheritance

Polymorphism

What is OOP?

$$\begin{tikzcd} \text{OOP} \arrow[r] & \text{Object} \arrow[r] & \text{Primitive Datatype} \end{tikzcd}$$

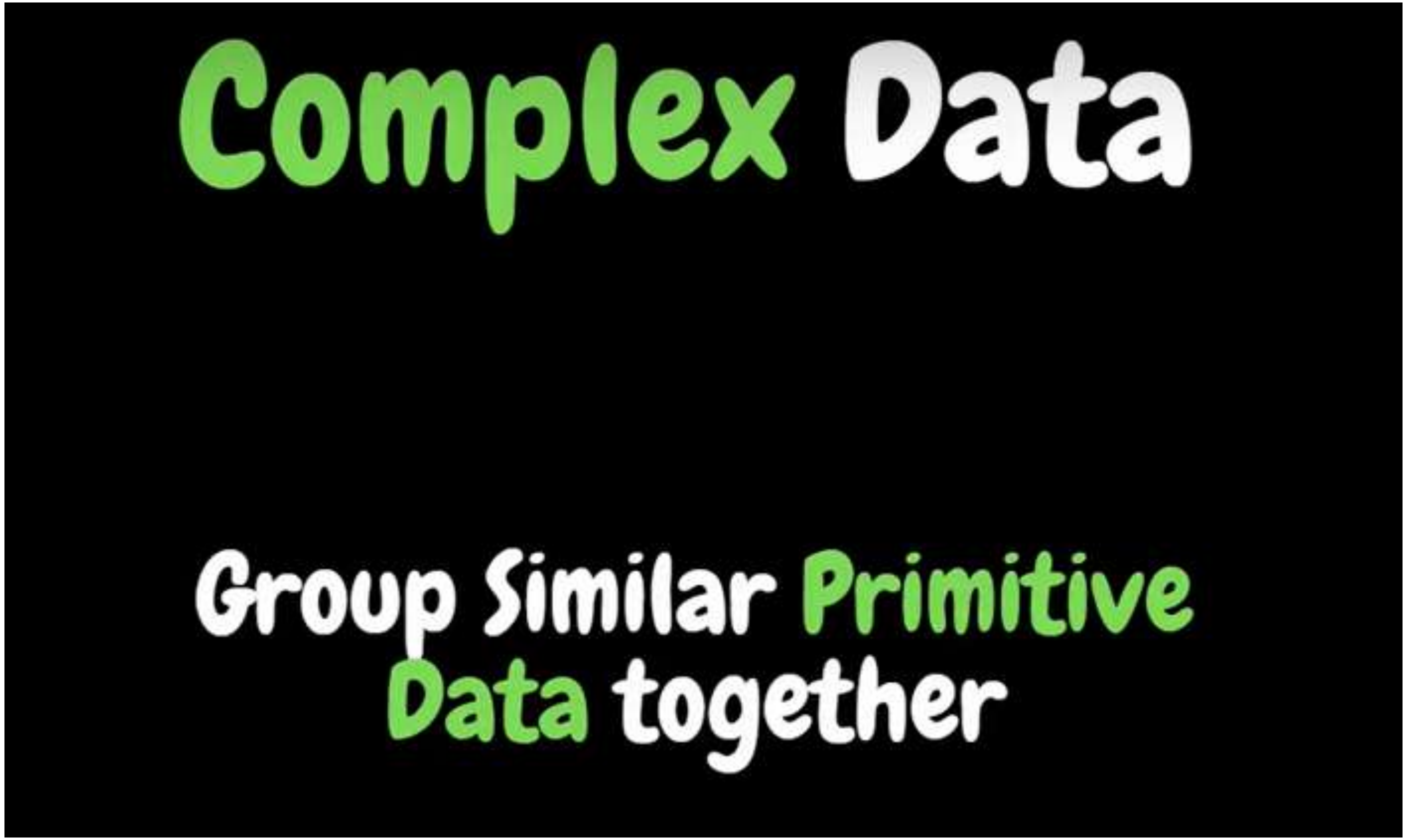


Primitive Datatype

int float char

boolean double

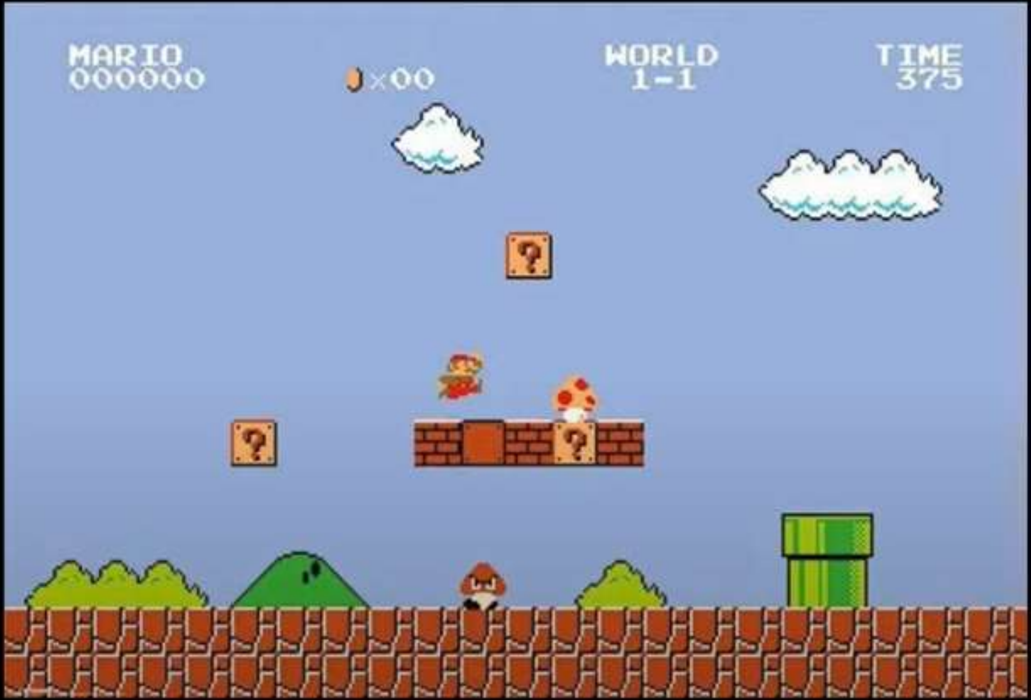
Simple Data ✓



Complex Data

Group Similar Primitive Data together

Why we need OOP? – Example



Player

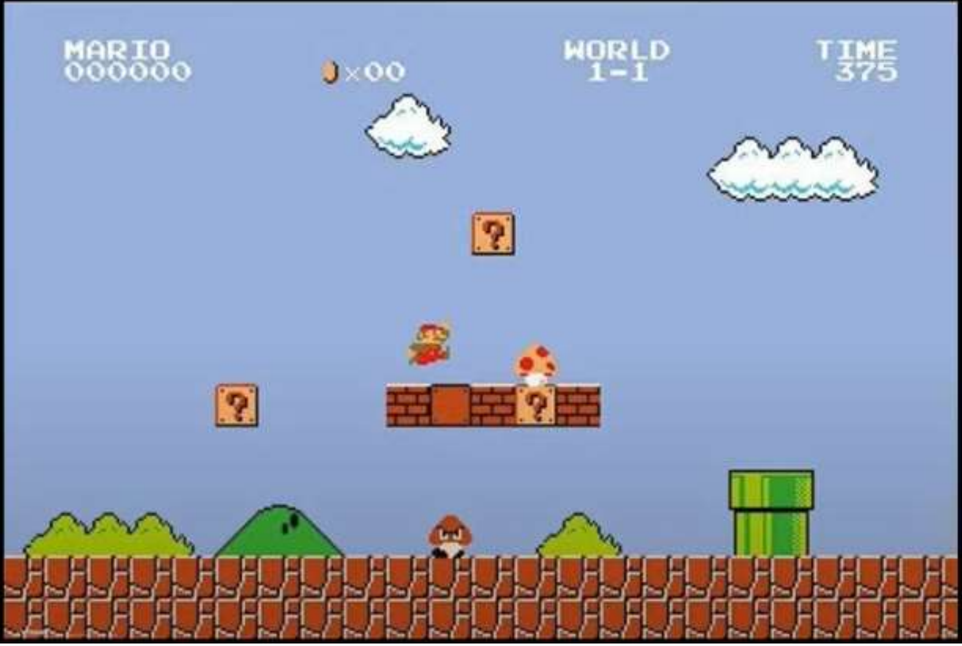
Ground

Enemy

Bricks

Pipes

Why we need OOP? – Example

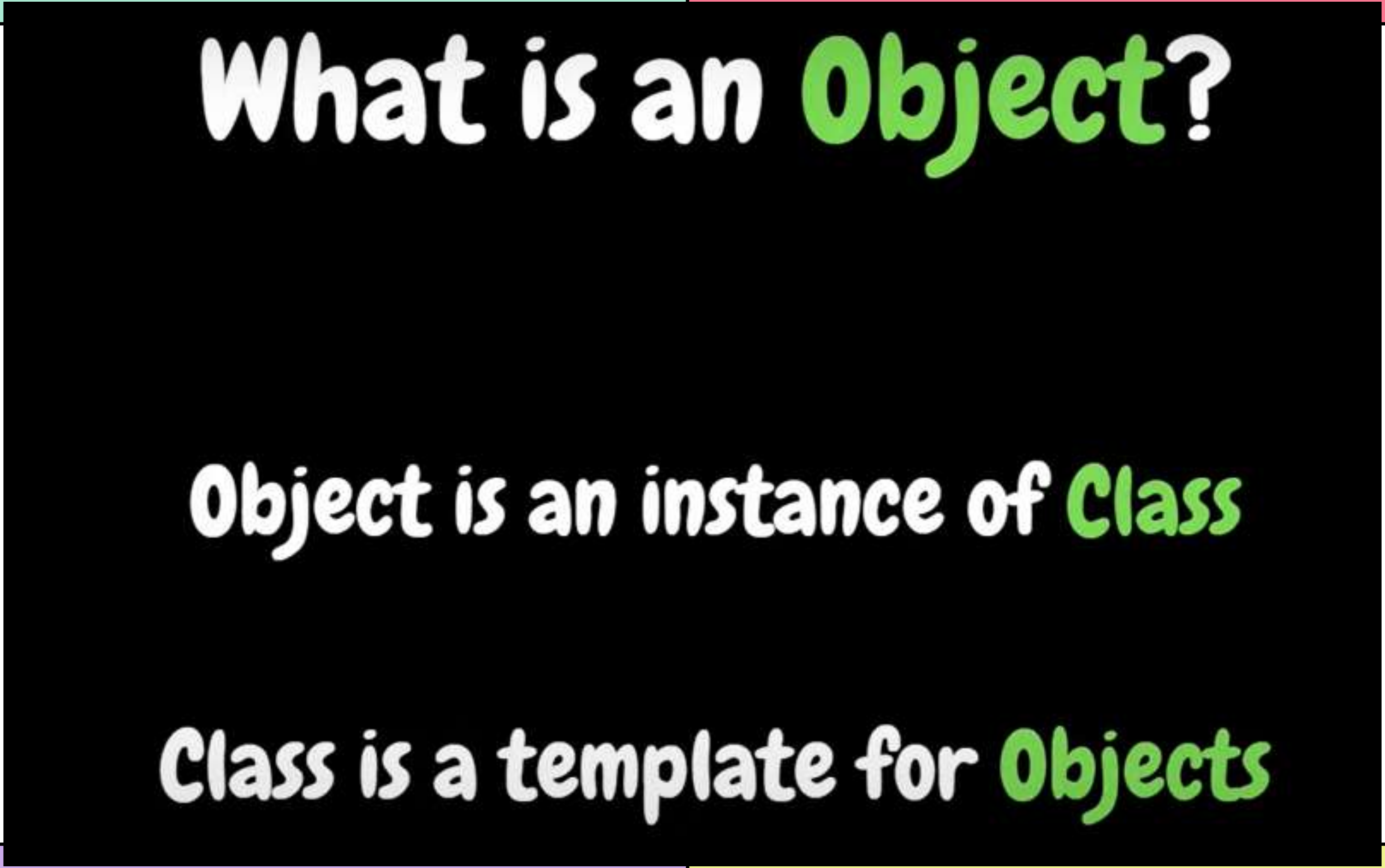


Player

Position

Size

Health



What is an Object?

Object is an instance of Class

Class is a template for Objects

What is an Object?

Class	Object
Home Blueprint	Actual Home

Class & Object – Mario Example

Enemy

Class

Size

Position

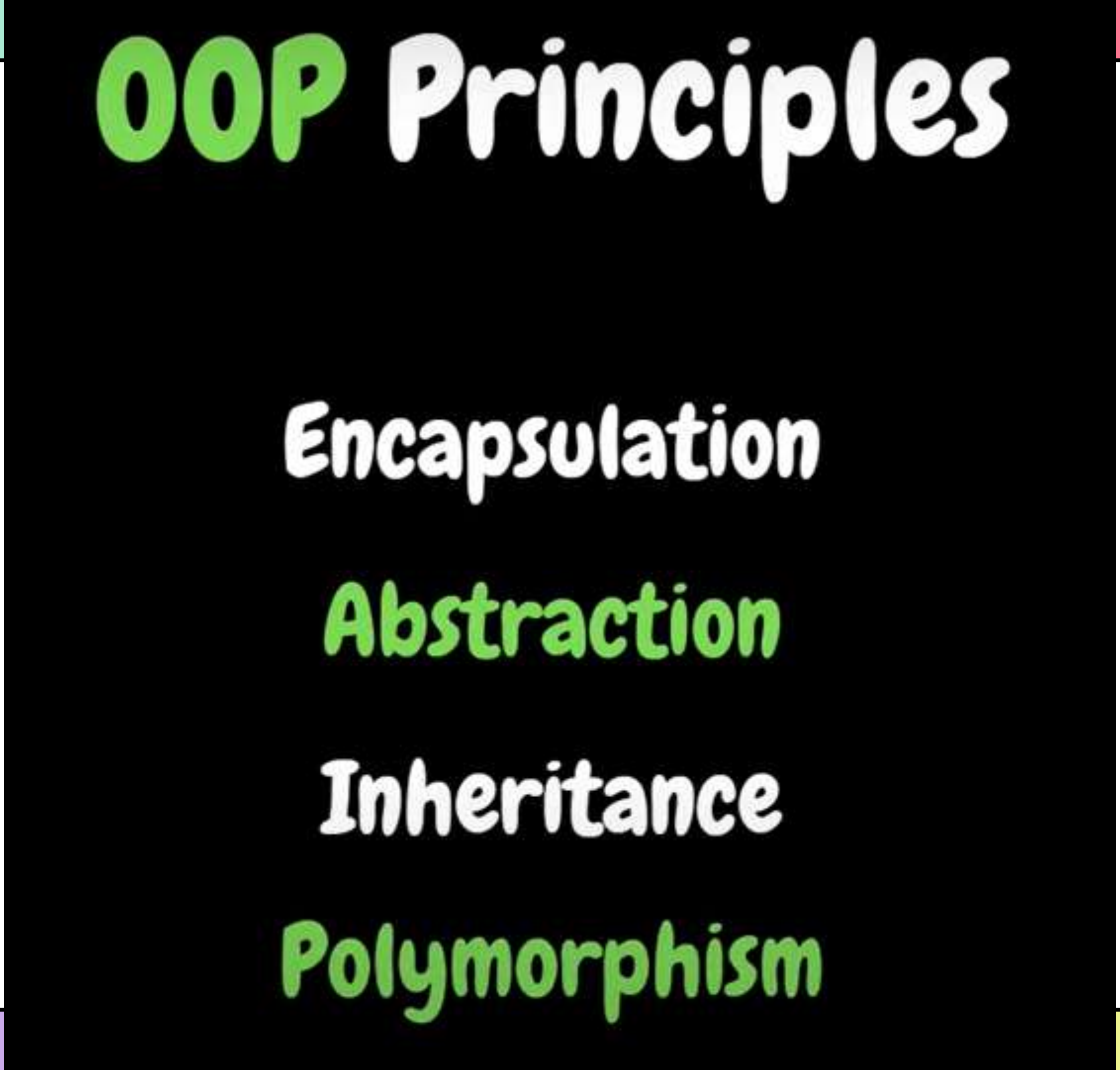
move()

Objects

Enemy 1

Enemy 2

Enemy 3



OOP Principles

Encapsulation

Abstraction

Inheritance

Polymorphism

What is Encapsulation?

Encapsulation is grouping data with methods in a class

Hiding data. Prevent direct access from outside

What is Encapsulation?

Access through methods

Getter

Setter

Retrieve Information

Modify Information

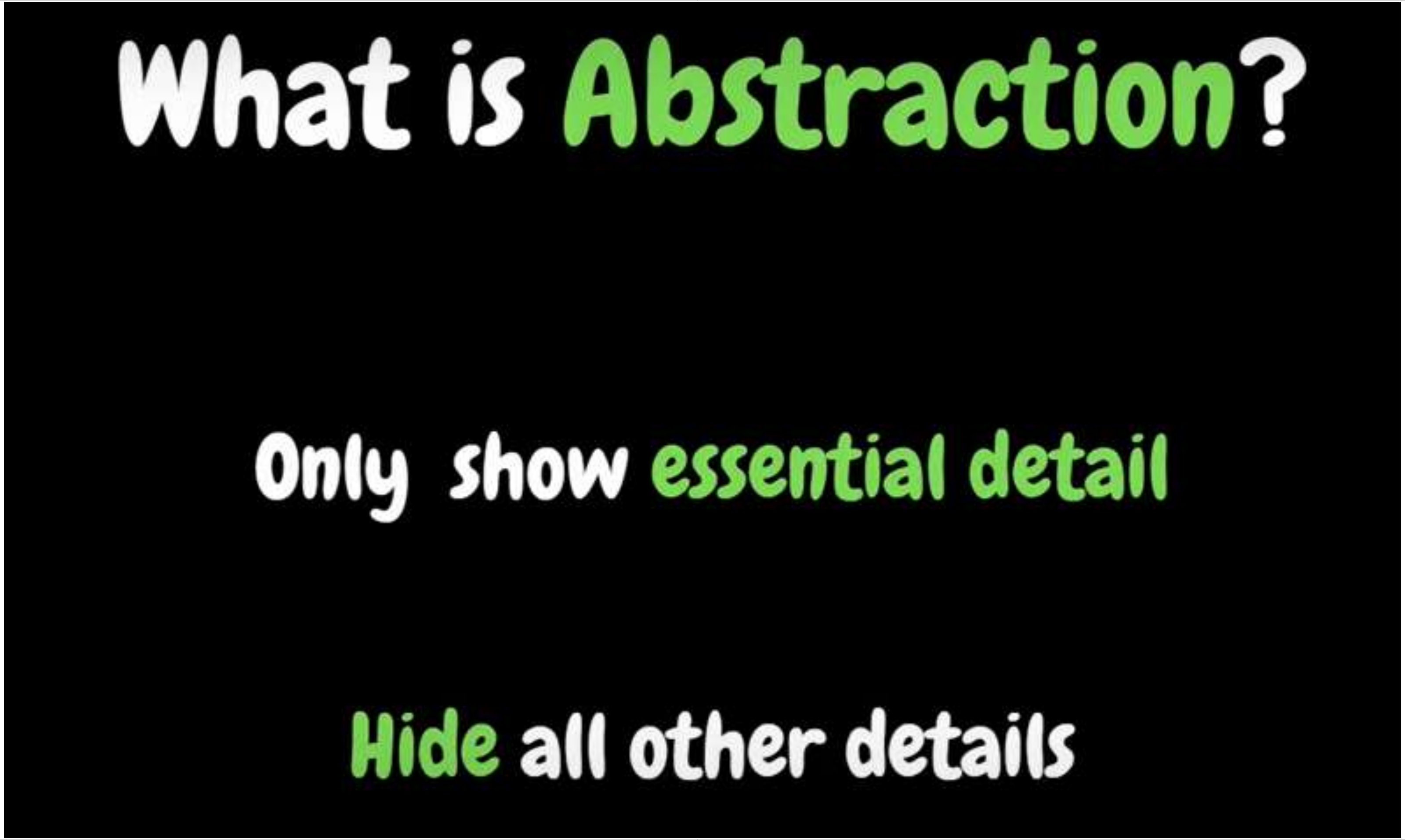
class Enemy

class Player

class Game

class Items

class Background



What is Abstraction?

Only show essential detail

Hide all other details

What is Abstraction? – Laptop Exampl

Only shows essential detail

Keyboard	Trackpad
Monitor	USB

Hides all other detail

RAM	Hard disk
Battery	CPU

How is Abstraction done?

Interface

Shown outside class

Implementation

Hidden from outside

Keyboard

Mouse

Monitor

Class

CPU

Hard disk

RAM

Abstraction Example – Mario

class Enemy

Interface

Implementation

getPosition()

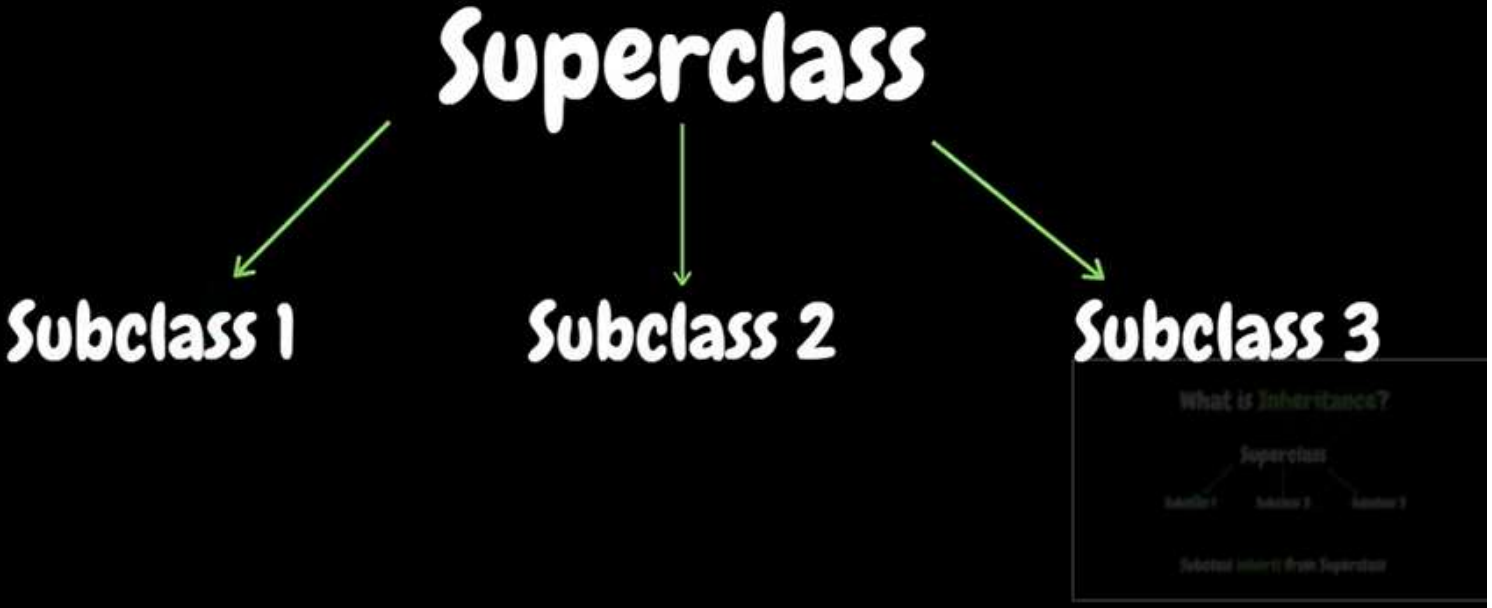
What is Inheritance?

Allows classes to use data & method from another class

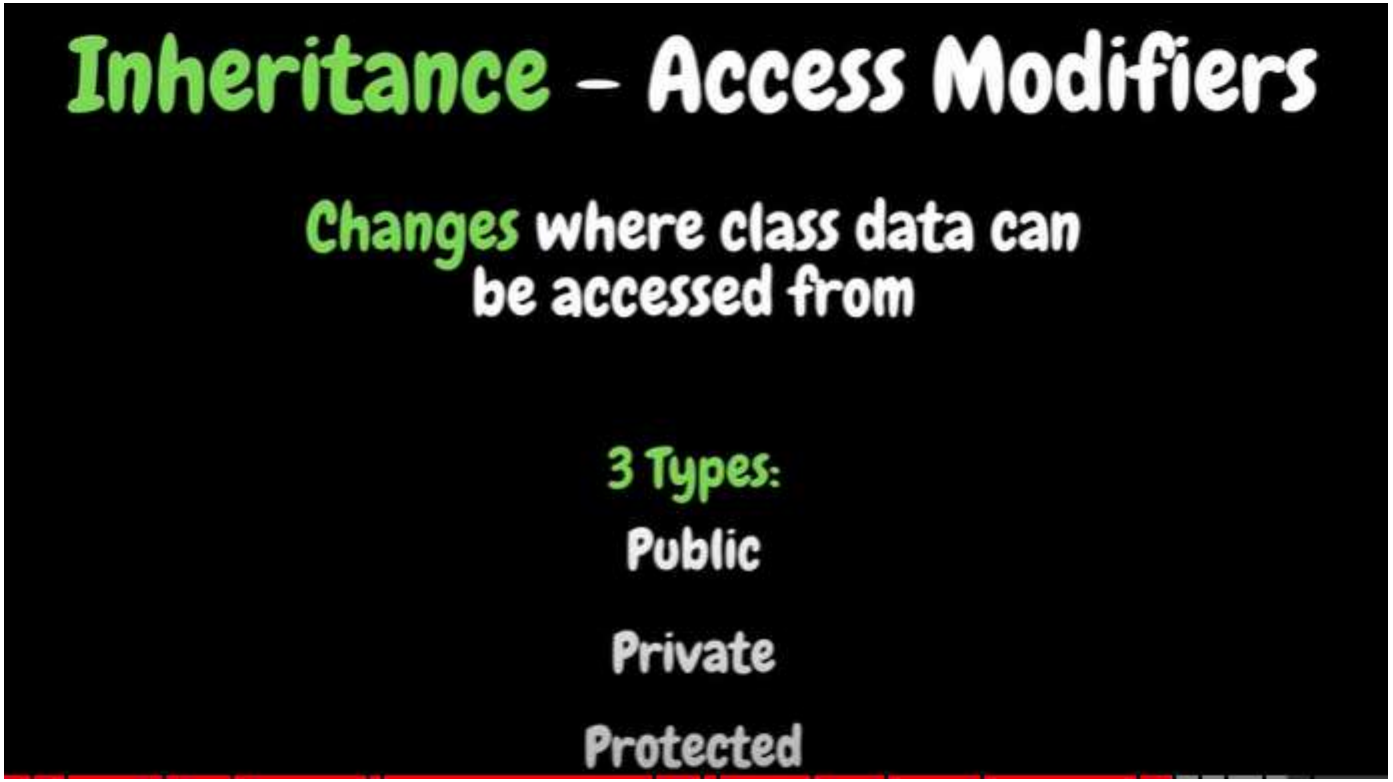
What is Inheritance?

$$\begin{tikzcd}
 & \text{class Enemy} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{class Monkey} & \text{class Octopus} & \text{class Dragon}
\end{tikzcd}$$

What is Inheritance?



Subclass inherit from Superclass



Inheritance – Access Modifiers

Changes where class data can be accessed from

3 Types:

Public

Private

Protected