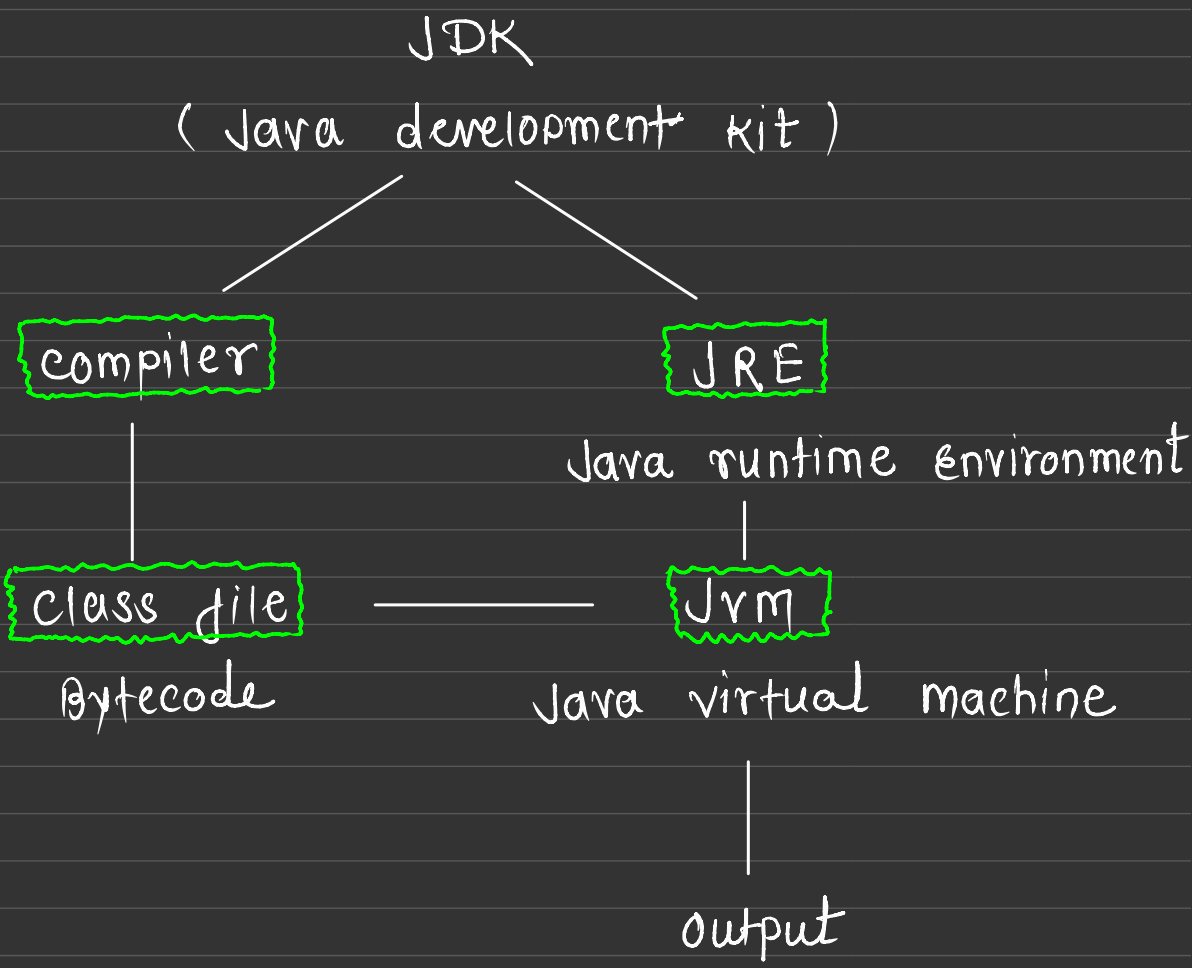
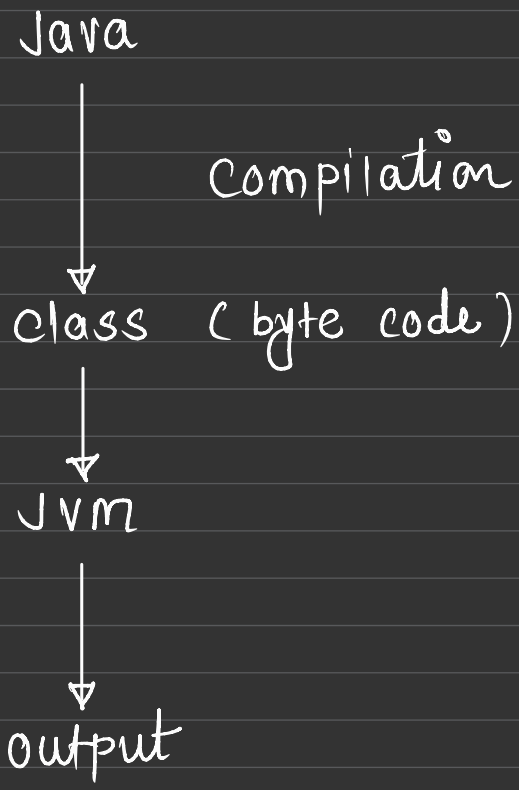

Sundeep
Saradhi

Java programming





Compilation → Javac filename.Java

Execution → Java filename

set classpath

① Command prompt → cmd

set path = "c:\program files\java\jdk1.8\bin"

once this path is set we can run our program from anywhere in the computer / system.

② system properties

↳ Environmental variables

↳ User variables

↳ New

↳ variable name

Path

variable value

location

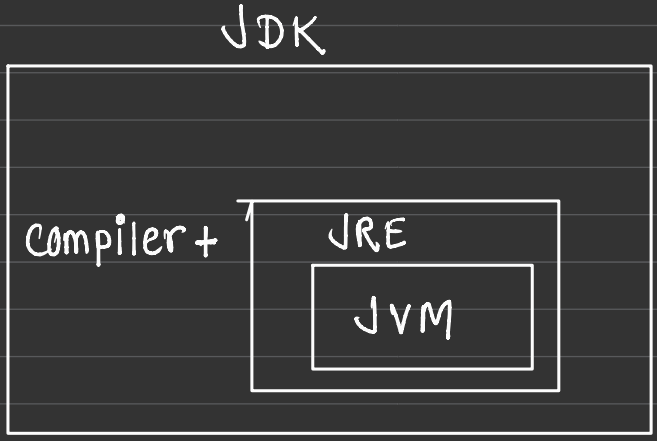
Revision

JDK

JRE

JVM

Byte code



Java Buzzwords

① platform independent (can run on any environment)

→ output will be same

② Java follows object oriented programming

- Abstraction
- Encapsulation
- Inheritance
- polymorphism

③ simple

- No concept of pointers
- Explicit memory allocation
- Structures
- operator overloading

④ security

- secure for internet application
- Raise out of Boundary exception in arrays
- Strongly typed language

⑤ Robust

↳ Early checking of errors

↳ Garbage collector

↳ Exception handling

⑥ portable

↳ Java program written on one environment can be executed in another environ-ment.

⑦ Multithreading

↳ Concurrent execution of several parts of same program at the same time.

↳ Improve CPU utilization

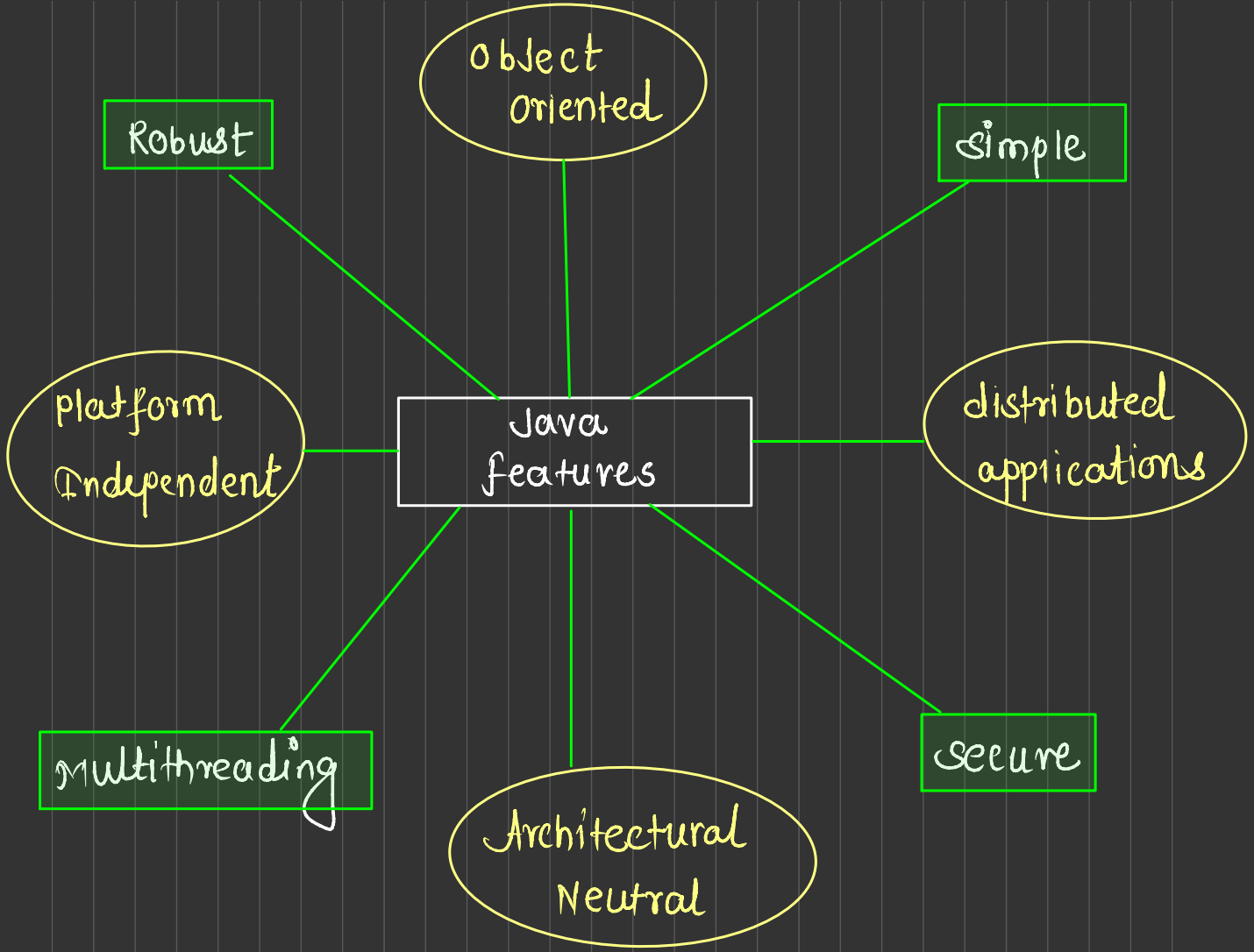
⑧ Distributed Applications

↳ software that runs on multiple computers connected to a network at same time.

* RMI (Remote method Invocation)
* EJB (Enterprise Java Beans)

⑨ Architectural Neutral

↳ Irrespective of Architecture the memory allocated to the variables will not vary.


Object Oriented
Robust
Simple
Platform Independent
Java Features
distributed applications
Multithreading
Architectural Neutral
Secure

OOP's Concepts

- object
- class
- Abstraction
- Encapsulation
- Inheritance
- polymorphism

① object

↳ Real world Entity

↳ properties

↳ Task

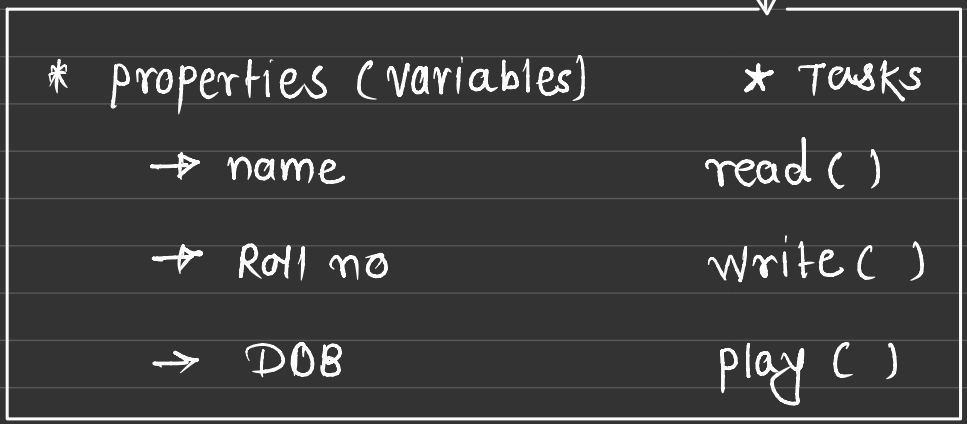
↳ objects are instance of a class

② Class

↳ Class is blueprint to make objects.

↳ We can make n number of objects of a class.

eg: Student (class) → Nitin (object)
Student (class) → Rahul (object)



Classes can not communicate with each other, if required to communicate objects should be created

Communication means - Invoking the methods of one class to another class

③ Abstraction

↳ Showing only Essential parts and hiding implementation details

Ex:- Android Application

- .apk
- .exe

④ Encapsulation

↳ Binding variables & methods under single Entity.

Input and output

output

system.out.print();

↳ everything will be printed in a single line

ex:- system.out.print("welcome")

system.out.print("Java")

system.out.println();

↳ It will print output on multiple lines

when we need access specifiers

%.d → Int

%.f → float

%.d → double

when we use printf

system.out.printf

Input

* Scanner - class → Keyboard

* Buffered Reader - class
  - ↳ InputStreamReader
  - ↳ file reader
  - ↳ keyboard & file (existing)

* Scanner
  - ↳ Import java.util.*;

* BufferReader
  - ↳ Import java.io.*;

Scanner

Import java.util.*;

next() → Read String → sc.next();

nextInt() → Read Integer → sc.nextInt();

nextFloat() → Read Float → sc.nextFloat();

nextDouble() → Read double → sc.nextDouble

Creation of object

Classname objectname = new constructor();

Scanner sc = new Scanner(System.in);

Input & output

```
String name;
int a;
float b;
double c;

Scanner io = new Scanner(System.in)

    name = io.next();
    a = io.nextInt();
    b = io.nextfloat();
    c = io.nextDouble();
```

↑ Methods

BUFFERED READER

* Read input
  - Keyboard
  - File

Read charecters from charecter input streams

Input Stream Reader
- ↳ Reads bytes & decoded to charecter set system.in

FileReader → Read data from files

↳ To Exception

BUFFERED READER
↓
To package

Buffer

System.in

BUFFERED READER

↳ Reads Charecters / strings

* read() → Reads single charecter
* readLine() → Reads multiple charecter String

Integer.parseInt( )

Float.parseInt( )

```
InputStreamReader ir = new
                       InputStreamReader
                       (System.in)

int a = Integer.parseInt(br.readLine());
int a = Integer.parseFloat(br.readLine());
String str = br.readLine();
```

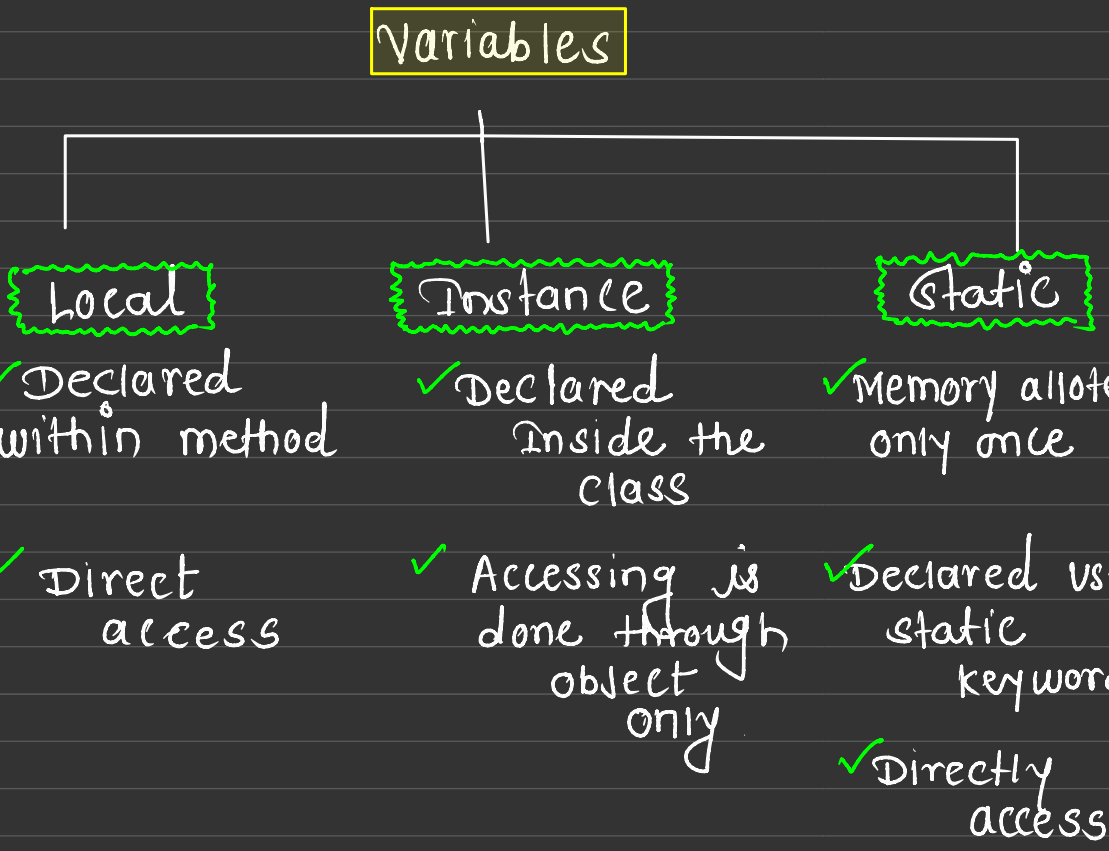
BUFFERED READER

```
FileReader fr = new FileReader(path);

BufferedReader br = new BufferedReader(fr)

String str = br.readline()

System.out.println(str)
```

command line arguments

```
class Add
{
    public static void main (String Args[])
    {
        int a, b, c ;
        a = Integer. parseInt (args[0]) ;
        b = Integer. parseInt (args[1]) ;
        c = a+b
        System.out. println (c);
    }
}
```

Types of variables



```
Class variable
{
    static int c = 30 ; Static variable
    int a = 10 ; Instance variable
  public static void main (String args[])
  {
      int b = 20 ;  Local variable

      system.out.println (c) ; → 30
      system.out.println (b) ; → 20
      system.out.println (a) ; → Error

  }
}
```

```
Class variable
{
    Static int c = 30 ;  Static variable
    int a = 10 ;  Instance variable
  public Static void main (String args[])
  {
      int b = 20 ;  Local variable

      system.out.println (c) ; → 30
      system.out.println (b) ; → 20
    Variable obj = new Variable (system.in);
                    ↳ object creation
      system.out.println (obj.a) ; → 10

  }
}
```

Constructor

Object Creation

```
Classname objectname = new constructor();
```

- `new` → Create an instance (object)
- `constructor()` → Initialize the object

Rules :—

1. constructor name must be equal to class name
2. constructor doesn't have any return type

Constructor

```
Class Student
{   String name;
    int rollno;
    public static void main (String args[])
    {   Student s1 = new Student( );
                     ↳ constructor
        System.out.println (name);  ✗
        System.out.println (rollno);  ✗
    }                  ↳ whats wrong here ?
}
```

Constructor

```
Class Student
{   String name;
    int rollno;
    public static void main (String args[])
    {   Student s1 = new Student( );
        System.out.println (s1.name);
        System.out.println (s1.rollno);
    }
}
```

declared but not initialized

↳ constructor

↳ null

↳ null

Class Student

parameterized ~~Constructor~~ Constructor → default

{ String name;
int rollno; } Declared but not initialized

Student( ) ← default
{ name = "ABC";
rollno = "123";
}

public static void main (String args[])
{ Student s1 = new Student( );
↳ constructor

System.out.println (s1.name);
↳ ABC
System.out.println (s1.rollno);
↳ 123

}
}

```
Class Student
{   String name;
    int rollno;
```

Declared but not initialized

```
    Student()
    {   name = "ABC";
        rollno = "123";  }

    Student (String str, int n) {
        name = str;
    }   rollno = n;

    public static void main (String args[])
    {   Student s1 = new Student( );
```

default constructor

```
        Student s2 = new Student ("DEF", 456);
```

parameterized

```
        System.out.println (s1.name);
        System.out.println (s1.rollno);
        System.out.println (s2.name);
        System.out.println (s2.rollno);

    }
}
```

Main Method

return type

```
public static void main (String args[])
```

- public → Access Modifier
- static → direct Access without object
- main → name of function
- String args[] → for command line argument

* we dont have to create object to access main function thats why we write static

STATIC BLOCK, METHOD, VARIABLE

3:17

↳ object is real world entity

↳ class is not real world entity (Blueprint)

↳ Every class has variables and methods.

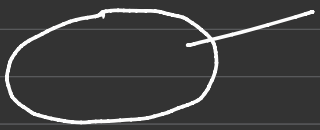
Variables — Static

Single class ← (1) Directly access

Multiple classes ← (2) Access with the help of class name.

Static block → 1st static block will be executed

```
static
{
}
```



executed by default without any explicit call.

```
class StaticDemo
{
    static int a = 10;
    static void display()
    {
        System.out.println("STATIC METHOD");
    }
    static { }
    {
        System.out.println("STATIC BLOCK");
    }
    public static void main(String a[])
    {
        System.out.println(a);
        display();
    }
}
```

← no explicit call for static block

↑ direct access no need of creating object

```
class StaticDemo
{
    static int a = 10;
    static void display()
    {
     System.out.println("STATIC METHOD");
    }
    static
    {
     System.out.println("STATIC BLOCK");
    }
     public static void main(string a[])
     {
     StaticDemo obj = new StaticDemo();

        System.out.println(obj.a)
        obj.display();
}
}
```

```
Class StaticDemo
{
    Static int a = 10;
    Static void display()
    {
      System.out.println("STATIC METHOD");
    }
    Static
    {
      System.out.println("STATIC BLOCK");
    }
    }
class main {
    public Static void main(string a[])
    {   System.out.println(staticDemo.a);
        staticDemo.display(); } }
```

TWO Classes

3:20