Date. 3/3/22
Page No.

Java Programming.

1991 (Sun Microsystem)

Called Oak by James Crosslin one of the founder of Java.

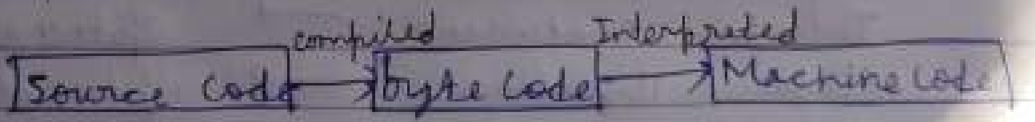
- Simple. portable. Reliable

First application language World wide.

- Java is an object oriented ~~pro~~programming language developed by sun-microsystems of USA in 1991.

- Originally Called Oak by one of the inventor of Java.

Java- Purely Object oriented.
How Java Works?
Java is compiled into the byte code & then it is interpreted to machine code.



JDK (Java Development Kit)
Collection of tools used for developing & running java programms.

JRE (Java Runtime Environment)
Helps in ~~too~~ executing programs developed in Java.

main method()

Entry point to an application.

Static

without making object ~~class~~ run class function.

Naming Conventions

- For class we use Pascal Convention
- For function we use cameKaseConvention

add two number

(Pascal) → (camel)

AddTwoNumbers — addTwoNumbers

For class — for function

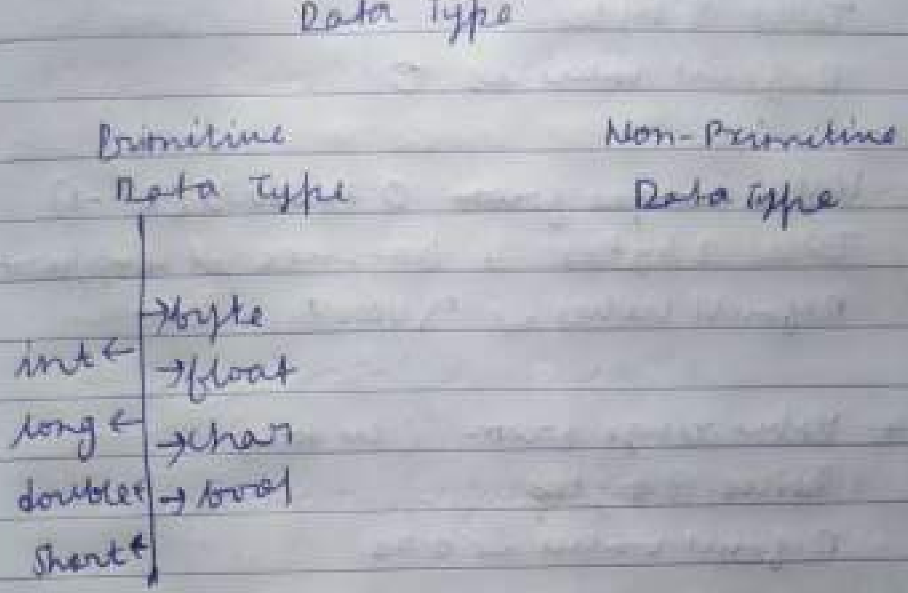
Anatomy of a Java Program

Documentation section → Suggested
Package Statement → Optional
Import Statement → Optional
Interface Statement → Optional
Class Definitions → Optional

Main Method Class
{ → Essential
Main Method Definition
}

Date
Page No.

Data types in Java



Java is statically typed

~~Java is~~ variables must be cleared before use
There are 8 primitive data types supported by Java

1) **byte** - Value ranges from -128 to 127

2) Short - Value ranges from $-(2^{16})/2 + (2)^{16}/2 - 1$
- Takes 2 bytes
- Default value is 0.

3) int - • Value ranges from $-(2^{32})/2$ to $(2)^{32}/2 - 1$
- Takes 4 bytes
- Default value is 0.

4) Float - value ranges from
- Takes 4 bytes
- Default value is 0.0f

5 long → Value ranges from $-(2^{64})/2$ to $(2^{64})/2 - 1$
Takes 8 bytes
Default value is 0

6 char → Value ranges from 0 to 65535 ($2^{16}-1$)
Takes 2 bytes → because it support unicode
Default value is '\u000d

7 double → Value range from (see docs)
Takes 8 bytes
Default value is 0.0d

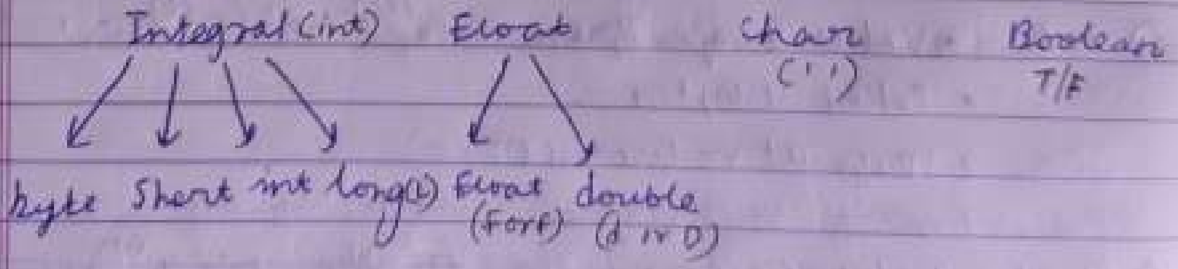
8) boolean → Value can be true or false.
Size depends on JVM
Default value is false.

4/3/22

Literals in Java

→ How to choose data type for our variable

Primitive Data types



A constant value which can be assigned to the variable is called as a literal.

101 → Integer literal
101F → Float literal
"Knockcat" → String "

Keywords

Words which are reserved & used by the java compiler
They cannot be used as an Identifier.
eg static, public etc.

JRE (Java Runtime Environment)
↓
Layer over OS

Feature of Java

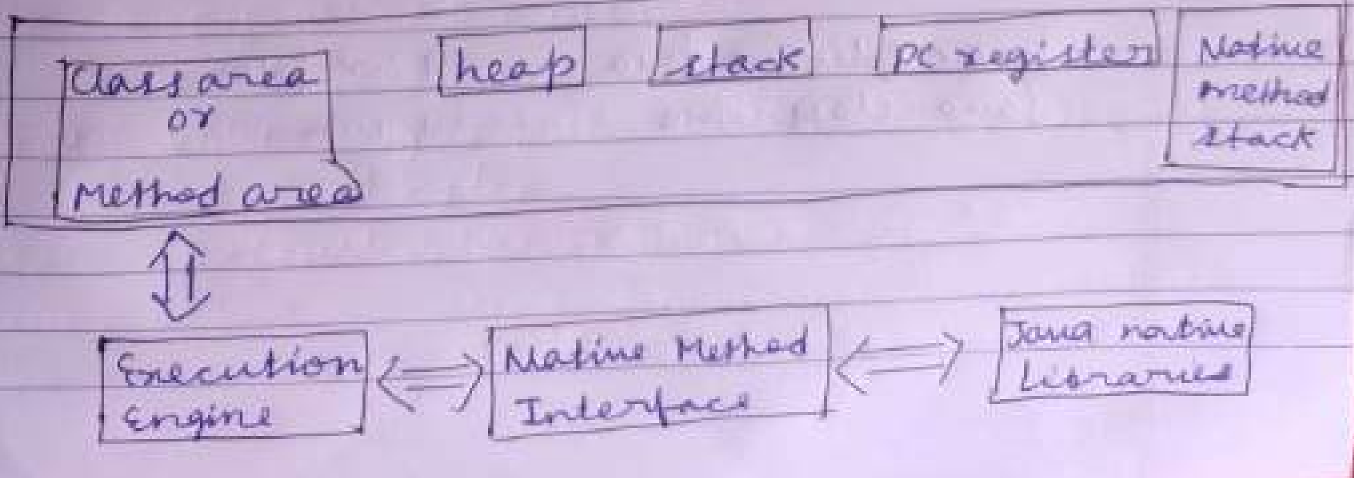
Simple, Portable, Platform Itdependent, Secure, OOP, Robust, Architecture neutral, Interpreted, high performance, Multithread, Distributed, Dynamic, Multilinged,

JVM / JDK / JRE

.java extension
javac compiler

.class class file / byte code / virtual machine code

Class Loader



① Bootstrap class loader loads rt.jar (contains packages we used)

② Extension class loader loads Java home/JRE/lib/ext jars

③ System/Application class loader loads classes from class path.

Execution Engine
Virtual Processor
Interpreter
JIT (Just in time compiler

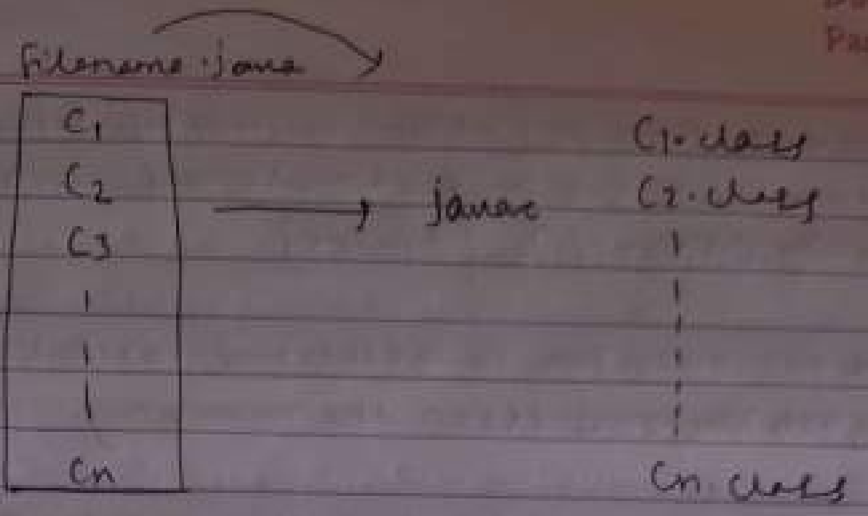
4/3/22

array of string objects

```
Class Xyz
{
   public static void main (String[] args)
   {
      System.out.println("Hello");
   }
}
```

If class is public than file name is same as classname.

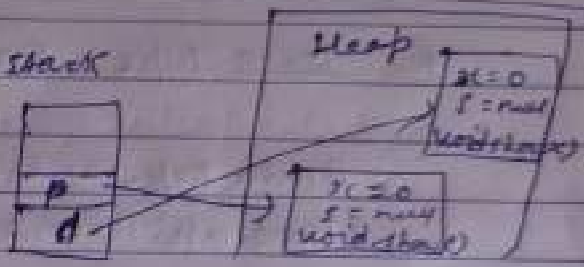
Javac filename.java ⟶ compile
Java classname ⟶ run



A file can have almost only 1 public class

5/3/22

Creating an object



```
class Xyz
{
  int x;
  string s;
  void show()
  { Sytemoutpritln(x); System.out.println(s);
  }

  public static void main (String[]args)
  {
      Xyz  d =  new Xyz();
      Xyz  p= new Xyz();
      d.show();
      p.x = 10;
      p.s = "xyz";
      p.show();
}
```

p,d are references.

default value of x = 0
s = null

We can terminate the class as ; (as we do in C++)
but here it is totally optional.

Anonymous Object (object without refference is known as)
one time use, we don't make reference

new Xyz().show(); [Syntax]

Garbage collector is a thread which will find the type of objects & clean the memory.

* We can create multiple object in one line

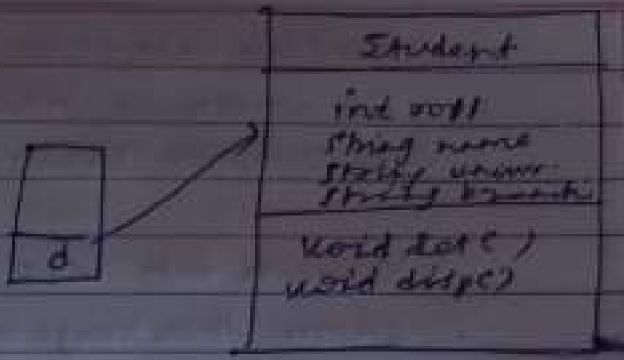
Xyz d = new Xyz(), p= new Xyz();

* Data Type - Size & Default value

Data Type	Default value	Default Size
boolean	false	1 bit
char	'\u0000'	(unicode property) 2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Unicode Property - Support non-ascii Characters also.

Student Class



```
class Student
{
  int roll
  String name, universityname,
       branch;

  void set( )
  {
      roll = 10 ;
      name = "Vishal";
      Universityname = "GEHU";
      branch = "CSE";
  }
void disp ()
  { System.Out.println(rool, name, universityname ,branch)};
 public static void main(string[] args)
  { Student d = new Student);
       d.set();
       d.show();
  }

}
```

Taking User Input

Command line arguments
Scanner class
I/O

Reading data from Keyboard

In order to read data from the Keyboard, java has a scanner class.

Scanner ~~dat~~ class has a lot of methods to read the data from the Keyboard.

```
    Scanner S = new Sanner(System.in);
                        ↳ Read from keyboard
   int a = S.nextInt();
                  ↳ method to read From the Keyboard
              Integer in this case.
```

```
  //Sum two numbers (User input)
import java.util.Scanner
    public class Main
     {
       public static void main(Strings[] args)
       {
          System.out.println("Enter 2 numbers");
          Scanner sc = new Scanner(System.in);

          System.out.println("Enter value for a");
          int a = sc.nextInt();

          System.out.println("Enter value for b");
          int b= sc.nextInt();

     int Sum = a + b;

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

next() method read only one word.

we should use nextline();

User Input

1) String[] arg will contain command line Arguments.
   → in form of string.
   → can be typecasted into required datatypes
   → using parse methods of wrapper classes of desired data type.

   for eg → Integer.parseInt(arg[0]); //wrapper class of integer

2) Scanner class
   take input in form of tokens → {breaks when find whitespace}
   java.util package - will be loaded by bootstrap loader.
   java.util.scanner
   nextLine() → This function's delimeter is new line/Enter
   next() → delimeter → space (String)
   nextInt() → nextFloat()
   import java.util.* → all classes loaded
   {not sub packages}
   input through Keyboard
   Scanner sc = new Scanner (System.in)

3) Object class (predefined or user defined)
   java.lang.object {predefined class by default all classes directly or indirectly are child of this class

→ Object class consist of 11 proper/standard methods.
   - These methods are in object class
   - These methods came in our class automatically as all are child of this class.

Date 12/3/22
Page No.

Saturday

Constructors : called at the time of object creation.
- Used to Initialize fields
- By default
- No return type .

constructor can have access specifier.

```
class Xyz
{
   int x, y;
    Xyz()
    {
       x = 5;
       y = 6;
    }
   Xyz (int m, int n)
   {
       x = m;
       y = n;
  }

  public static void main( String[] args)
  {
     Xyz  d = new Xyz();
     Xyz  p = new Xyz(8, 10);
     Xyz  d1 = new Xyz(5);  // Error
```

* Static variable

```
class Xyz
{
  static int x;
    Xyz()
    {
     x++;
    }

  public static void main(String[] args)
  {
      Xyz d= new Xyz();
      Xyz p= new Xyz();
      Xyz d1 = new Xyz();
      System.out.println(x);
  }
```

#

```
a = 10;
System.out.println (~a);      // -(variable)-1
```

this is reference variable in Java which referes to current object.

```
class Xyz
{ int x,y;
  Xyz(int x, int y)
  { this.x = x; this.y = y; }        //ambiguity if we do
  void show()                                x = x ; y = y.
  { System.out.println(x + " "+y); }    compiler will give
 public static void main(String[] args)      0 0 as answer
 { xyz d = new Xyz(5, 6);
   d.show(); .
 }
```

Date
Page No

```
Class Xyz
{
int x, y;
void set (int x, int y)
{
   this.x = x ;
   this.y = y ;
   show();        //this.show();
}

void show ()
{
   System.out.println(x + " " + y);
}

Public static void main (String[] args)
{
    Xyz d=new Xyz();
    d.set(5, 6);
}
}
```

Constructor chaining

~~[illegible]~~ for connecting constructor, & reusing of construc

```
class Xyz
{ int x, y;
  Xyz()
  {                         → must be First line of construct
    this(5, 6);
    System.out.println("Hello");
  }
  Xyz (int x, int y)
  { System.out.println("Hi...");}
```

Variables in Java

① local Variables → Stack Area

② Instance variable → Memory when Object is created

③ Static variable (accessible through object)

→ location in class Area, along with byte code.
(accessible without object,
shared variables)

Non-static variables

cannot be used in static method

Java control statements

① if else statements

② while, For, Do-while

③ For each loop (for forward traversal)

```
eg- String S[] = {"a", "b", "c"};
    For(String SS : S) System.out.println(S);
```

④ Switch case( )

↓

$\text{int/char/byte/short/}\underbrace{\text{String/Enum}}_{\text{Java7}}$

```
  public static void main (String[] args)
  {
      Xyz d=new Xyz();
  }
}
```

* This line must be the first statement in the constructor.

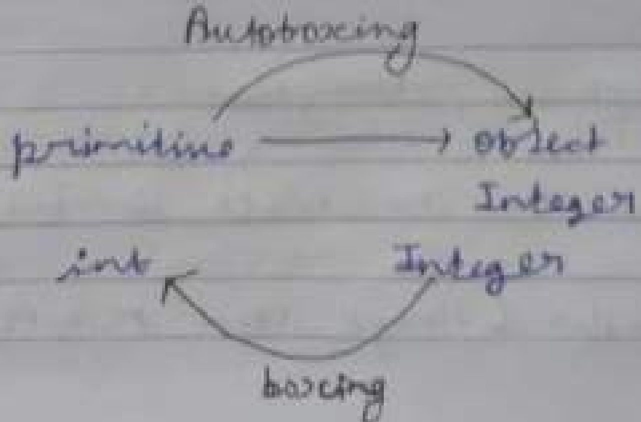
#chained constructor reuse the constructor.

22/3/22

Primitive into object & object into primitive.

java.lang package

primitive Type	wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double



Where we use Wrapper class?

1) Socket programming.
when we send string data in a flow from one computer to other.
We cannot send data in primitive form therefore we have to use wrapper class.

Autoboxing

```
Class Auto
 {
    Public Static void main (String args[])
    {
        int x=5;
        Integer Xbox = x;   //Autoboxing
        Integers yobj = 25;
        int y= yobj;      //unboxing
    }
```

Strings in Java   in java,
Strings are object that represent sequence of ~~obj~~ characters

Strings in Java
java.lang

String, Stringbuffer, StringBuilder

* Strings are immutable in java.

* StringBuffer & StringBuilder mutable

- In java we can create string objects in 2ways.

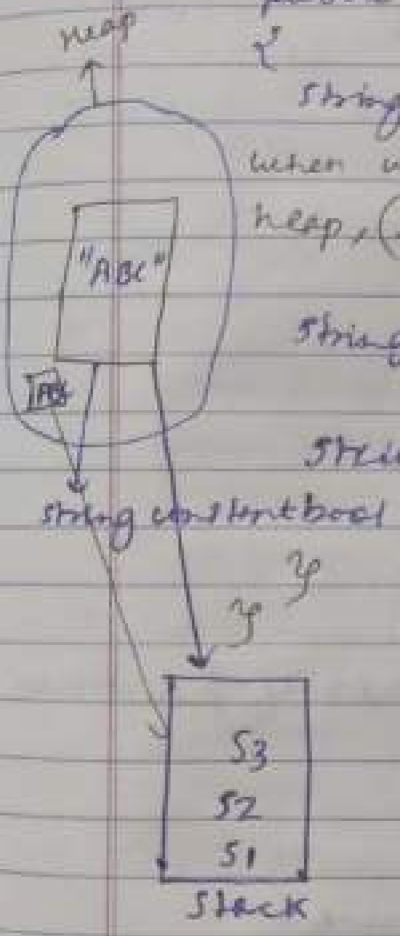
String
(1) String Literals
(2) By new Keyword

```
Class {
  public static void main ( String args [])
  {
    String s1 = "ABC"; // Literal (refer to same if already exist)
```

when we use literal, then there is one more area in heap, (String constant bool) especially for string objects.

```
    String s2 = "ABC";

    String s3 = new String(" ABC"); // form on heap
                                        refered by s3
  }
}
```

Fresh new memory location create on heap



It will check ABC literal is on string constant bool or not if it exist then same literal is refer by others.

(Thats why strings are mutable, because if we change string then it got changed by all who refer it.)

s2 refering to same literal

String s1 = "ABC"; // literal

* System.out.println(s1.length()); ~~n = arr[].length;~~ // length

* System.out.println(s1.charAt(0)); // at index 0

String s2 = "ABC";

* System.out.println(s1.equals(s2)); // check value exact match gives true otherwise false.

* s.concate( ) // will not change the string as strings are immutable.

// compare object
String s = "ABC";
String s1 = "XYZ";
String s2 = "ABC";
System.out.println (s == s2); // true
System.out.println (s == s1); // False

* equalsIgnorecase (); // ignore cases.

* ~~lastIndexOf()~~ indexof() // tells the index of particular character.
// default value is -1

* After 5th Index tell the occurence of character
~~System.out.println('o', 5)~~ System.out.println (s.indexof('o', 5));

* lastIndexof () // tells index from last.

System.out.println(s.lastIndexof ('c'));
System.out.println(s.lastIndex of ('c', 7);

* System.out.println(to

* trim() //remove starting & ending spaces
  //not works in string literal as they are immutable.

* toLowerCase()
* toUpperCase()

* // Java String Substring
  System.out.println (s.substring(2)); //from 2nd index
  System.out.println (s.substring(2, 4)); // from 2 to before 4

* //isEmpty()
  String str = "";
  System.out.println(str.isEmpty()); //true

26/3/22

* StringBuffer
  Java StringBuffer class is used to create mutable (modifiable) string objects.
  The StringBuffer class in Java is the same as String class except it is mutable i.e it can be changed.

Note : Java StringBuffer class is thread-safe.
i.e multiple threads cannot access it simultaneously.
So it is safe & will result in an order.

// methods are synchronized
// Thread safety
StringBuilder : If thread safety is not required & we want mutability, go with StringBuilder.

*

```
class Append {
public static void main (String[] args)
{
   StringBuffer sb = new StringBuffer("Hello");
   sb.append("Java"); //now orignal string is changed
   System.out.println(sb); //prints Hello Java
}
}
```

sb ⟶ Hello Java

* insert method

append at particular index

```
StringBuffer sb = new StringBuffer("Hello");
sb.insert(1,"Java"); //now orignal string is changed
Sys.tem.out.println(sb); // print HJavaHello
```

* replace method

```
StringBuffer sb = new StringBuffer("Hello");
sb.replace(1,3,"Java");      // replace from 1 to 3
System.out.println(sb); //print HJavalo
```

* delete method

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

* reverse method

```
StringBuffer sb = new StringBuffer("Hello");
sb.reverse();
System.out.println(sb); //print olleH
```

Java StringBuilder Class

Java StringBuilder class is ~~used to create~~ same as String Buffer class except it is non-synchronised.

All functions are also available in String Builder

In Java array is an object of a dynamically generated class.
Java array inherits the object class.
We can store primitive values or objects in an array.

Syntax Array declaration

```
dataType[] arr;
dataType []arr;
dataType arr[];

int a[] = new int[5]; // declaration and instantiation
a[0] = 10; // initialisation

// run time
// array is a child of object class.

// traversing an array
  for(int ~~int~~ i=0; i<a.length; ++i)
  System.out.println(a[i]);

  int a[] = {33, 3, 4, 5}; // declaration, instantiation,
                               initialisation

  a.length()
```

```
// For each loop
   for (int i : arr)     // Forward direction traversing
// ArrayIndexOutOfBoundException // class in Java          29/3/22
```

~~Inheritance~~

Multidimensional

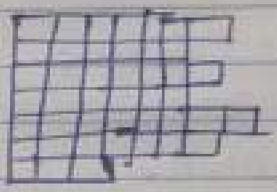
dataType [][] array, dataType [][]array, datatype array[][]

* datatype []array[]   // declaration

declaration Instantiation Initialization

```
int arr[][] = {{1,2,3},{2,4,5},{4,4,5}};
```

Jagged Array



```
int arr[][]= new int [3][];
arr[0]= new int[3];
arr[1]= new int[4];
arr[2]= new int[2];

int cnt = 0
      for(int i=0 ; i<arr.length; ++i)
          for (int j=0 ; j<arr[i].length; ++j)
                arr[i][j] = ++cnt;
```

```
1 2
4 5 6
8
```

// Every class has a signature

```
int intArray[] = new int[3];
byte byteArray[] = new byte[3];
short shortArray[] = new short[3];

//Array of String.
 String[] strArray = new String[3];
```

* System.out.println(intArray.getclass()); //give signature of the array

intArray.getclass().getsuperclass()

class java.lang.object

signature class [I //int array

class [B //byte array

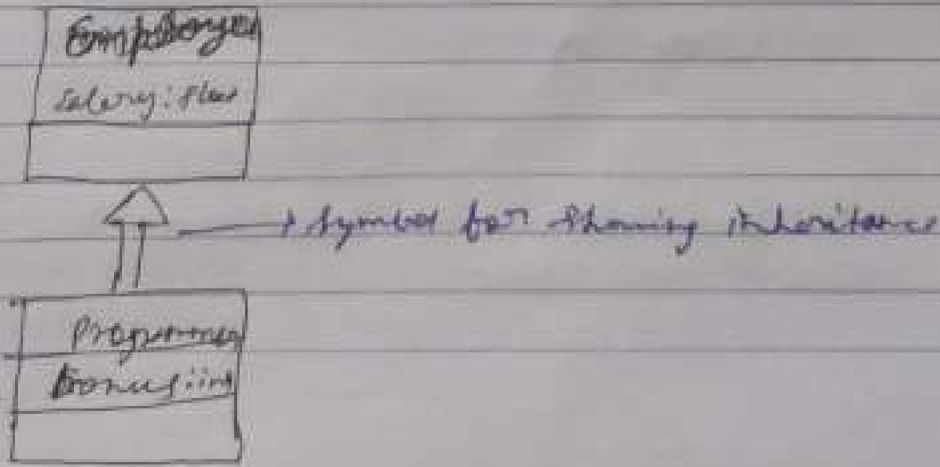
class [S //short "

Inheritance

for reusability either make (1) object of class or we use (2) Inheritance.

Multiple Inheritance not allowed in Java.

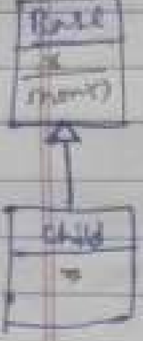
~~[illegible]~~



Date. 1/4/22
Page No.

Multiple inheritance in Java

~~path Child : Base~~



```
class Base
{
  int x = 10;
  void show()
  {
    System.out.printlen(x);
  }
}

class Child extends Base
{
  public static void main(String[] args)
  {
    Child d = new Child();
    d.x = 15;
    d.show();
  }

void show()   // method overriding
{
 System.out.printlen(x+2);
}
```

#Multilevel

```
// Multilevel
 class Base
 { int x=10;
   void show()
   { System.out.println(x);
   }
 }
 class Child extends Base
 { int y;
 }

 class NewChild extends Child
 {
 }
```

// Multiple inheritance is not allowed in java
↓
error in compile time.

```
 class A
 {  void show()}
    {      }
 }
class B
{   void show ()
    {      }
}
Class C extends A, B        // not allowed
{  public static void main (String args[])
   {   C d = new C ();
}  }     d.show();
}
```

Final Keyword

Can be used for 3 purpos

1. Final Variable
2. Final method
3. Final class

① → Final variable

```
Final int n =10;   // the value is constant throughout program
                   // value cannot be changed.
```

② → Final method

// Final method cannot be overidded or changed in child class.

```
Class A
{
    final void show() { }    // cannot be changed
    void display() { }
}

Class B extends A
{
    void show ()   // compile error  // cannot be overidded
                                        in child.
    void display ()
}
```

We use final method if we do not want to override that method in child class?

(3) f) Final class

If a class is final, it cannot be extends or inherited.
∴ If we want to reuse it we have to use it with the help of object.

```
Final class A
{

}
Class B extends A   //not allowed
{
}
```

* Final method is inherited but we cannot changed.

blank or unitialized final variable //a final variable which is not initialized at the declaration.

```
Class A
{
  Final int x;          //can only initialised using
                          constructor if not initialised
A ()
{ x = 10;
}
public static void main (String[] args)
{
  A d=new A();
}
}
```

Static block execute before main & used for initializing static variables.

Date 2/10/22
Page No.

final static blank variable

```
final static int x;    // can only initialize with the help
of static block.
```

```
Static
{
  x = 10;
}
```

// Final variable can be passed as argument but cannot be changed.

```
class Xyz
{
 void show(final int x)
 {
    x = x+2;   //error
 }
 public static void main(String[]args)
 {
    Xyz d= new Xyz();
     d.show(8);
}
```

passing final var
as argument

// Polymorphism <u>one name different task</u>
println("") // function has // methods

```
class Xyz
{
  void show()
  { ~~p~~System.out.println("Hi");   //Hi }

 int show(int x, ~~int y~~)
 {
   System.out.println(x);
 }

float show (int x, int y)
{
   System.out.println(x + " " + y);
}

 public static void main(String[] args)
 {
   Xyz d = new Xyz();
  ~~Systo~~ d.show();          //Hi
        d.show(2);          //2
        d.show(3, 8);    // 3  8
        d.show(3, "xyz"); // no matching func found
 }

float show(int x, int y)  // Not allowed
int show(int x, int y)    // Not allowed
```

depend on three factor
① Sequence of parameter, // float, int | int float
② no of parameters
③ types of parameters.

Polymorphism is of 2 types
① Compile time
② Runtime by DMD (Dynamic Method dispatch)
③ Compile time polymorphism is achieved by method overloading
④ Runtime polymorphism is acheived by method overriding. //late binding
↑resolved by JVM

```
Class P
{
  void red ()
  {
    System.out.println("red");
  }
  void blue ()
  {
    System.out.println("blue");
  }
}
class C extends P
{
  void red ()
  {
    System.out.println("~~child's red~~ black");
  }
void green ()
{ System.out.println("child's green");
}
```

Child object refered by parent reference

Date

Page No.

Compiler will perform the & that will be resolved by JVM

```
public static void main( String args[])
{
  P d = new P();
  d.red(); // red
  d = new C(); // child object refer by parent(UPCASTING)
  d.red(); //black
  d.green(); //Error
}
}
```

Compile time error

// ~~runtime~~ overrided method are resolve ~~by~~ at runtime by JVM

5/4/22

Super Keyword in Java - -

The super keyword in java is a referrence variable which is used to refer immediate parent class object.

Usage of Java Super Keyword

1. Super can be used to refer immediate parent class instance variable.
2. Super can be used to invoke immediate parent class method.
3. Super () can be used to invoke immediate parent class constructor.

①

```
class A
{int x=10; }
class B extends A
{ int x= 25;
  void show(){ System.out.println(super.x); //10
               System.out.println(x); // 25 }
  Public static void main(String[] args)
  { B d = new B();
    d.show();
}
```

Date
Page No.

```
② class A
{ void display()
    {
      System.out.println("hi");
    }
}
class B extends A
{
   void show()
   {
      display(); // super.display(); //at time of compilation
     System.out.println("Bye---");
   }

}

③ class A
{ A()
    { System.out.println("Hi"); } }
    class B extends A {
      B() { super(); // implicitly
      System.out.println("Hello---");
    }
  public static void main(String args[])
  {
     B d=new B();
  }

}
```

/calls constructor using super.

Super must be first line

Date.
Page No.

Abstraction in Java

0 to 100% abstraction ——— --) by abstract (abstract keyword) class 100% abstraction ---) by interfaces.

9/12/21

abstract class -

cannot be instantiated (object cannot be created).
we can use these class by inheriting them.

```
abstract class xyz
{          ——) can keep abstract method inside them
}

        abstract void show ();
```

abstract method can only kept inside abstract class.

```
class   xyz          X
{
abstract void show();
}
```

```
abstract class xyz   ✓
 {
  abstract void show()
 }
```

* Abstract class can also keep normal methods, variables, constructors, main

* abstract method must be define in child class.

```
abstract class A
{ void show
}
abstract class P extents A
{ }
```

) not give error even though we have not defined show but P its also a abstract class.

```
abstrat Class Xyz
    {
        abstract void show();          //abstraction
        abstract void put();
    ~~y~~   void hello();              // not abstraction
    }                                  (partial abstraction by abstract
                                        class)
    class Pqr extends Xyz
    {
        void show () { }
        void put () { }
    }
```

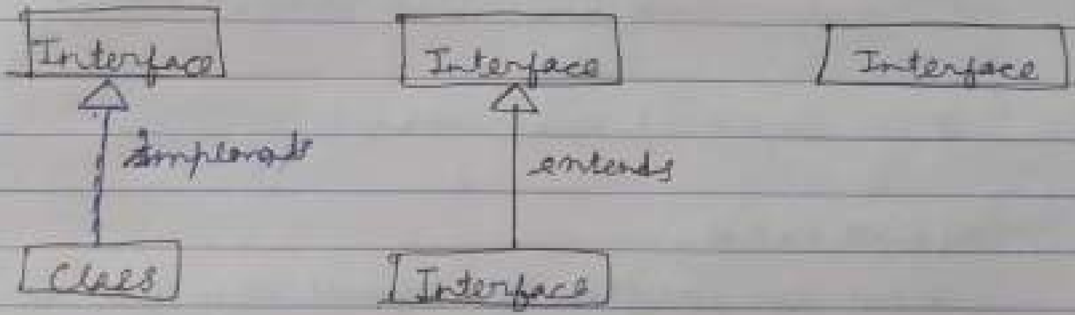
Interface in Java 100% abstraction

All methods will be abstract.

Interfaces object cannot be created.

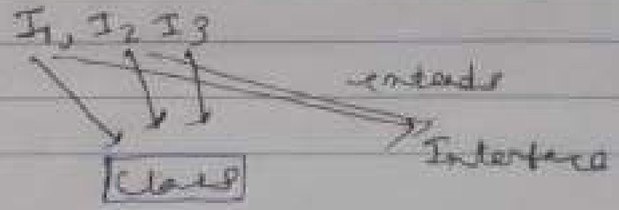
```
Interface Xyz
{
  // final variables ------ public static final   by default     // also have default
  // abstract methods                                            method & private
  // From java 8 default methods with body                       method
  // From java 9 private methods " "

  int n = 5 ; // public static final int n = 5; //by defau

  //abstract methods
  void show (); // public abstract void show ();

  // interfaces are implement in class.
}
```

```
interface B
{
  void show();        //public abstract void show();
  void put();         //  "     "       "   put();
}
class C implements B
{
   void show() //Error → //signature mismatch it
   {                        should be public
   }
public void put()            ∴ we have to write public
  {
  }

}
```



* We can implement as many as interface in one class.

*



Date 12/4/22
Page No.

Tuesday

Using class & interface in another class

```
interface I1
{

}
interface I2
{

}

class C
{

}

class D extends C implements I1, I2
{

}
```

```
// default method
   // we can give body of default specifier in interface
   // cannot override in child // we cannot override ~~but it will be~~
                          // compile error          ~~not override~~
 interface I1
 { default void show ()
      {
         System.Out.println("knowkat");
      }
 }
```

```
class C implements I1
{
    public static void main (String[] args)
    {
        C d=new C();
        d.show(); // Khoulcat
    }
}
```

* // Static methods cannot be overided in child. but if we write that method in child class it will be hide.

```
                    // now writing static method in
interface I1          Java is allowed.
{
    static void show ()
    {
        System.out.println("Hi");
    }
}

class C implements I1
{
    public static void main (String[] args) {
    I1.show (); // calling by interface name
    }           // can also called using child object.
}
```

```
// private method

interface I1
{
    private void show()                          // Java 9
     {     // private method cannot override, cannot calle
        System.out.println("hi");                  by child.
     }
    default void put()                           // Java 8
     {
        show();
     }

}

Class C implements I1
{
    public static void main(String[] args)
    {
       ~~I1 obj~~   C d = new C();
    ~~}~~               d.put();
    }
}
```

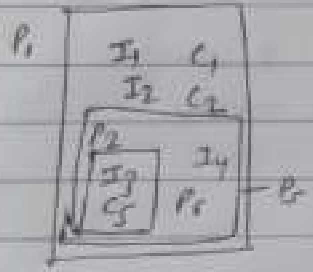
tagged interfaces or marker interface

interface I
{

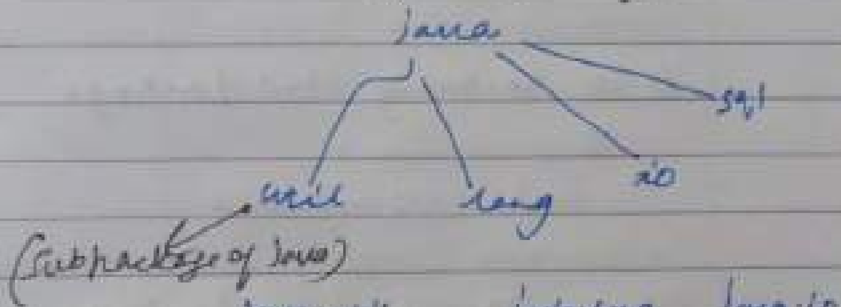
}

Serializable interface   //predefined interfaces
Remote

* Package // like headerfiles in C, C++.



Predefined packages



java.util , java.lang java.io
java.sql

import p1.*;   // access P1
import means all members except subpackages.

import p1.p2.*;   // For accessing P2 , and not P1 & P3

to
white color

Date 16/4/22
Page No.

// package name start from small case just a convention

// a folder will create at same directory with name mypl
// javac -d . myclass.java
delimiter → destination

// current working directory

```
            d:\> javac -d . myclass.java
```

```
package mypl ;

public class myclass
{
    public static void main(String[] args)
    {
        System.out.println("Hi ");
    }

}
```

// For running that package

```
    java mypl.myclass
```

// 3 ways to import a package

① import package .*;   // import mypl.*;
                          import javaawt.*;

import java.util.scanner ←

② import package.classname;   // import mypl.myclass;

③ fully qualified name

Date. 19/4/22
Page No.

```
package p1;
public class MyClass
{
    public void show()
    {
        System.out.println("Hi");
    }
}
```

① // How to import package

```
import p1.*;
class MyC2
{
    public static void main(String[] args)
    {
        MyC1 d = new MyC1();
        d.show();
    }
}
```

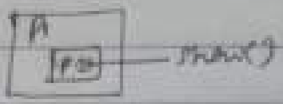
// Rename or Delete MyC1.Java

② import p1.MyC1; //import package.classname
// that problem will not occur, compiled successfully

③ Fully qualified name-

```
p1.MyC1 d = new p1.MyC1();
d.show();
```

// creating Subpackage



```
package p1.p2;
public class MyC3
{
void show()
    {  S.O.P("hi...");  }
}
```

// This will create p2 under p1

* For accessing

```
① import p1.p2.*;
② import p1.p2.MyC3;
③ ~~MyC3 o~~  p1.p2.MyC3 obj = new p1.p2.MyC3();
```

Interfaces can also be created in Packages
// on compiling Interface will also there

```
package P1;
 interface I1 {
 }
interface I2 {
}
class Knock
{
}
```

// 2 interfaces & 1 class will be present in P1

Access Modifier

1 private | highly restricted
2 default |
3 protected |
4 public ↓ low restricted

// Outer class cannot be private, protected
// Inner class can be private or protected on nesting

Variables/methods can be private
↓
not accessible outside
either through inheritance
or by object

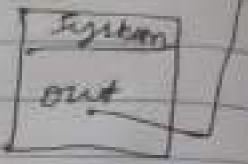
Not allowed

Access modifier	within class	within package	outside package by sub class only	outside package
private	Y	N	N	N
default	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

// Static import

```
import static java.lang.System.*;
class xyz
{
  public static void main(String[] args)
  {
     out.println("hello"); // no need of writing System
  }
}
```

reference of printstream, defined in System class



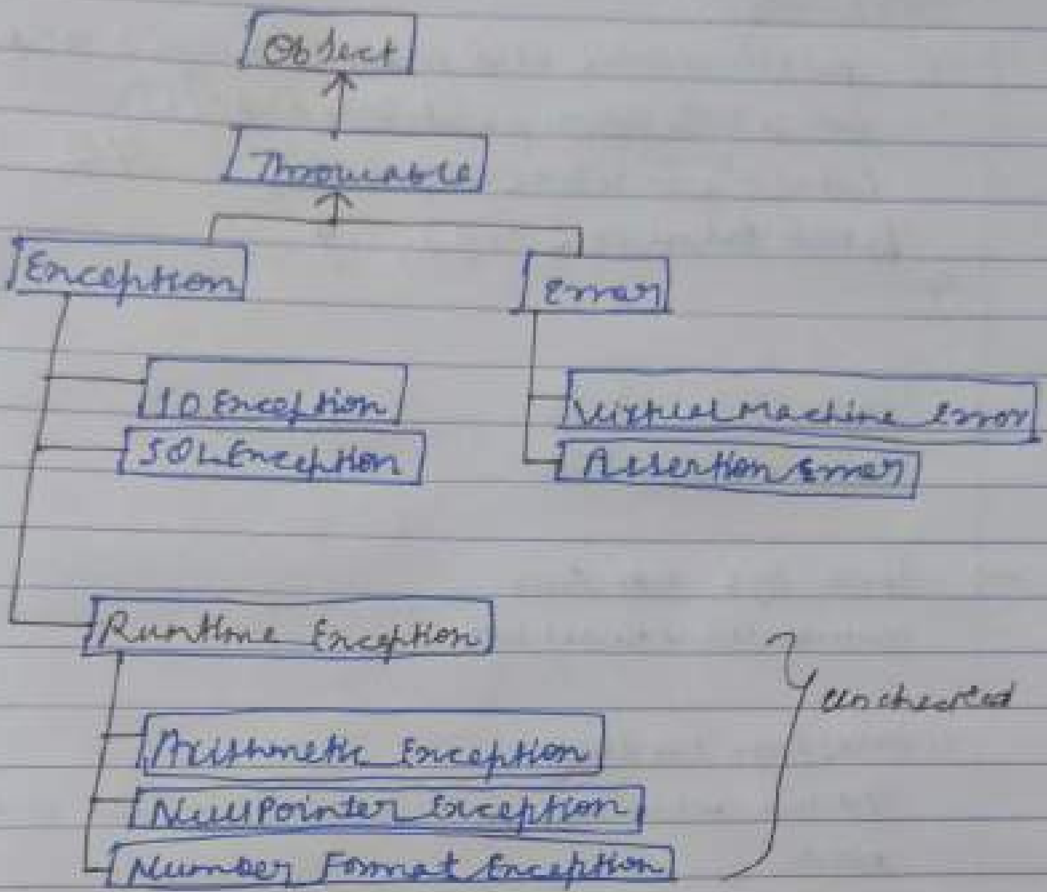
Exception Handling

// prevent program from abnormally termination at run time.

try catch
try finally
throws
throw

Exception types
Checked Exception
Unchecked Exception
Error

Hierarchy of Exception class



// There are no spaces in these classes :)

Compiler enforced Exception
compiler role

Unchecked

```
Class Xyz
{  public static void main(String[] args) {
   int x = Integer.parseInt(args[0]);
   int y = Integer.parseInt(args[1]);     //0
  System.out.println(x/y); }
}
```

→ java Xyz 5 0
ArithmeticException

→ java Xyz ten five
NumberFormatException

//Exception handling
The line which we felt error, impose it in try block

```
try
{
}
catch (Reference of ExceptionClass) //n no. of catches are
{ // handler (Exception Handler)                    possible
}
catch (Reference of ExceptionClass)
{
}
      //n no of catches are possible with one try block

catch (Exception [illegible] e)
{
  // if any exception is not handled, this will handle
}  everything (mother of all exception)
```

→ Every exception has corresponding handler
if not than there is default handler for all

```
catch (Exception e)      //always come at end.
{

}
```