OOPS

Static Keyword :-

→ Method

- Static Method is a class method.
- need not create an object to call static method
- It is used to manipulate static variables

```
class Student
{
    int id;
    String name;
    static String college = "UPES"

    static void change()
    {
        college = "IIT";
    }

    Student (int i, String n)
    {
        id = i ;
        name = n;
    }

    void display ()
    {
        S.O.P (id + " " + name + " " + college);
    }
```

college
class area

Date: _____ Page: _____

```
psvm ( string args [])
{
Student. change();

Student s1 = new Student (1, "Avi");
Student s2 = new Student (2, "Rahul");

s1. display ();
s2. display ();

}
}
```

O/P 1 Avi IIT
2 Rahul IIT.

Q. Cube of no. using static method

```
class Cube
{
    int a;
    static int Code (int a)
    {
        return(a * a * a);
    }
psvm(s args[])
    {
    int r = Cube.Code(3);
    sysem.out.pr"(r);
```

Date: Page:

Note: The static method cannot use non-static data member or call non-static method directly (need to create an obj before calling)

Static Block

- It is used to initialize static data member
- It is executed before main method at the time of class loading

```
class A
{
  static {
      S.O.p. ("static block invoked");
    }
  psvm (String args[])
    {
      S.O.p ("Main method");
    }
}
```

```
static {
      S.O.p (" — ")
      System.exit (0);
    }
```

Run in JDK 1.5

OOPS

30/8

Date: Page:

This Keyword

(i) This Keyword can be used to refer current class instance variable
(Removing Ambiguity

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

 Student(String name) int rollno)
    {
       this. name = name
       this. rollno = rollno.
    }

 void display()
    {
    sop(rollno + "  " + name);
    }

 psmv(String args[])
    { Student s = new Student(1, "Avi");
      s.display
    }
}
```

Date: Page:

(2) this() can be used to invoke current class constructor
- First call in constructor.
- constructor chaining

```
class Student {

    String name;
    int rollno;

    Student ( )
    {
      sop ("Defaut invoked");
    }

  Student (String name, int rollno){

      this();
      this.name = name;
      this.rollno = rollno;
  }
    void display ( )
    {
       sop (                    )
    }
  psmv (          )
  {

  }
}
```

Outpt Defaut invoked
1. Avi

this() constructor call should be the first statement otherwise it would be a compile time error.

(3) this() keyword can be used to invoke current class method implicitly.

(4) this() keyword can be passed on as an argument in the method.

```
class A
{
    String name;
    int id;

    m(A obj)
    {
        sop(    );
    }

    n()
    {
        m(this);
    }

    psvm(String args[])
    {
        A obj = new A();
        obj.n();
    }
```

AWT
⇓
Swings
⇓
event handling

J.S.R

Date : / /
Page :

11/sept/2022

OOPs

Inheritance.

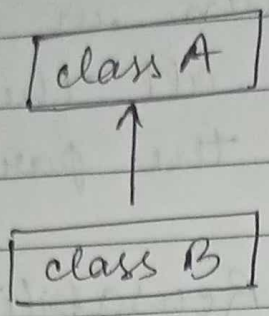
It is a mechanism in which one object acquires all the properties & behavior of the parent object.

Purpose → code Reusability
Method Overriding

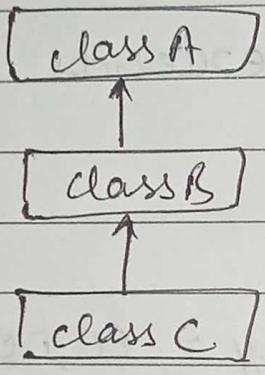
```
class Subclass extends Superclass

class Employee
  {
    int salary = 60,000;
  }
class Programmer extends Employee
  {
   int bonus = 20000

     psvm(String args[]) {

     Programmer p = new Programmer();

     SOP(P.salary + " " + p.bonus);
    }
  }
```

Types of Inheritance

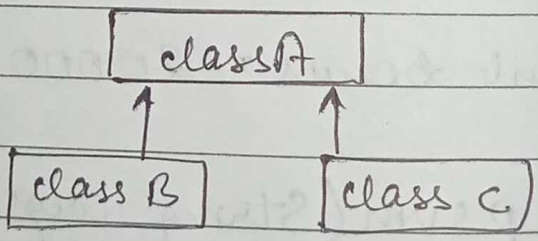
Single Level



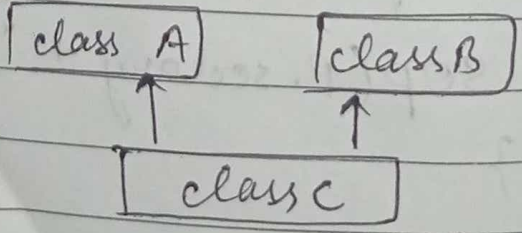
Multilevel



Hierarchical



Multiple



Date : / /
Page :

```
class A
  {
  void display { s.o.p("class A"); }
  }

class B extends A
  {void print { s.o.p ("class B"); }
  }

class C extends B
  { void real
      { s.o.p ("class C"); }
  }

psmv ( string args[]) {
      C.obj = new C ( );

        obj.real ( );
        obj.print ( );
        obj.display ( );
  }
```

```
class A
{
  A()
  {
    sop("I am A");
  }
}
class B extends A
{
  B()
  {
    sop("I am B");
  }
}
class C extends B
{
  C()
  {
    sop("I am C");
  }
}
class Test
{
  psmv(String args[])
  {
    C obj = new C()
  }
}
```

5/sept

Super Keyword

(1) super() keyword is used to refer immediate ~~parent~~ parent class instance variable.

```
class Vehicle
{
  int speed = 40;
  Vehicle()
  {
    sop("In Vehicle class");
  }
}

class Bike extends Vehicle
{
  int speed = 60;
  Bike()
  {
    sop("In Bike class");
  }

psvm (string args[]).
{
  Bike obj = new Bike();
  s.o.p.(obj.speed);
}
}
```

o/p => 60

```
void display()
  {
    sop(super.speed)
  }

psmv(        )
{
Bike obj= new Bike();
    obj.display();
  }
```

O/P – 40

② super is used to invoke parent class constructor.

```
class A
  {
   A()
   {
    sop("In class A");
   }
}

class B extends A
{
  B()
  {
   Sop("In class B");
  }
```

```
psmv (String args[])
{
  B obj = new B()
}
}
```

o/p - In class A
In class B

```
class A
{
  A (int a, int b)
  {
    this.a = a;
    this.b = b;
  }
}

class B extends A
{
  int c;
  B (int c)
  super (a, b);
  this.c = c;
}
```

Date : / /
Page :

(3) It is used to invoke immediate parent class method.

```
class A
{
  void show()
  {
    s.o.p("I am A");
  }
}

class B extends A
{
  void show()
  {
    sop("I am B");
  }
  void call()
  {
    super.show();
  }
psvm(  )
{
  B obj = new B()
  obj.call()
```

Date : / /
Page :

final Keyword

1) final keyword can be used with variable (const.)

2) method (Method can not be overriden)

3) class (class cannot be inherited

```
class Bike
{
  final int speed = 100;

  void run()
  {
    speed = 200;
  }

  psmv(String args[])
  {
    Bike obj = new Bike()
    sop(obj.speed);
        obj.run();
  }
```

Compile Time Error

Date : / /
Page :

```
class Vehicle
  {
final void show ( )
    {
      sop ("I am A");
    }

  }
class Bike extends vehicle

    void show ( )
      {
        sop("I am B");
      }
psvm ( string args[ ])
} Bike obj = new Bike()
    obj. show ()
  }
```

compile Time Error

Date : / /
Page : / /

final class'

final

1) Are final methods inherited.
2) what is Blank final variable
3) static Blank final variable

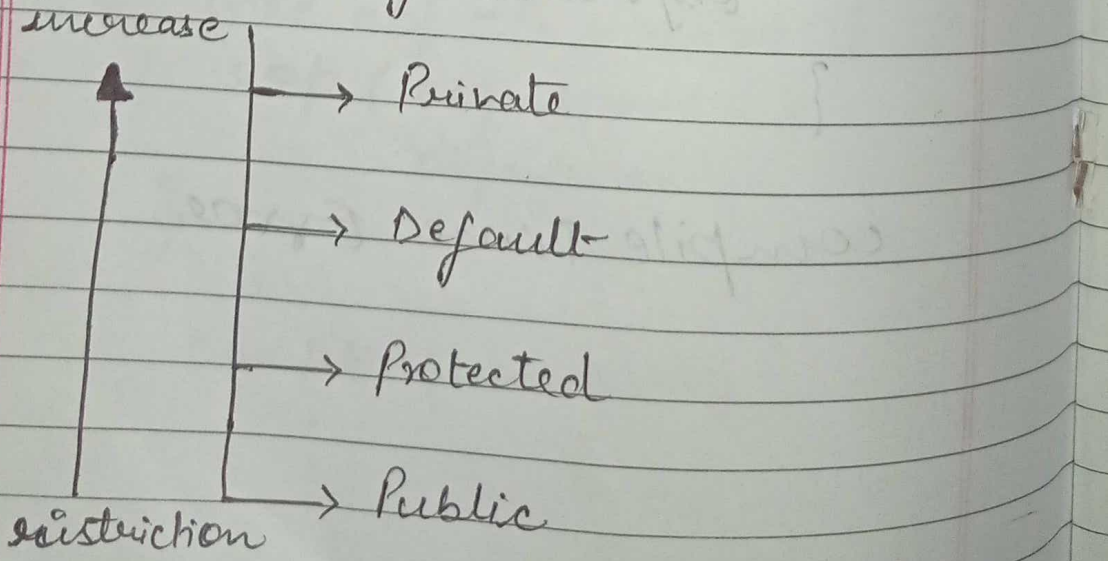
1) Yes final method is inherited but cannot override.

2) It is a final variable that is not initialized during declaration

3) A final static variable that is declared but not given a value or not initialized is a

6/9/2022

Access Modifier



Date : / /
Page :

Private

```
class A
{
private int speed = 40;
private void message()
  {
    s.o.p("In class A" + speed)
  }

class B
  {
    psvm ( )
    {
      A obj = new A()
      obj.message();  // "
      s.o.p(obj.speed); // "
    }
  }
```

* classes can never be private or protected. except Nested classes constructors & methods can be private.

Date : / /
Page :

Default

Default can be excess with in a package

Package is a collection of similar type of classes & interfaces & subpackages

package pack; → user defined Package

```
class A
{
default void message()
  {
    s.o.p("In A");
  }

package mypack;

import pack.*;

class B
{
  psvm( )
  {
    A obj = new A()
    obj.message()
  }
}
```

C.T.Error

Date : / /
Page :

Protected

```
package pack;

public class A
  {
    protected void message()
      {
        S.O.P("In A");
      }
  }

package mypack;

import pack.*;

class B extends A
  {
    psvm(    )
      {
        A obj = new A();
        obj.message();
      }
  }
```

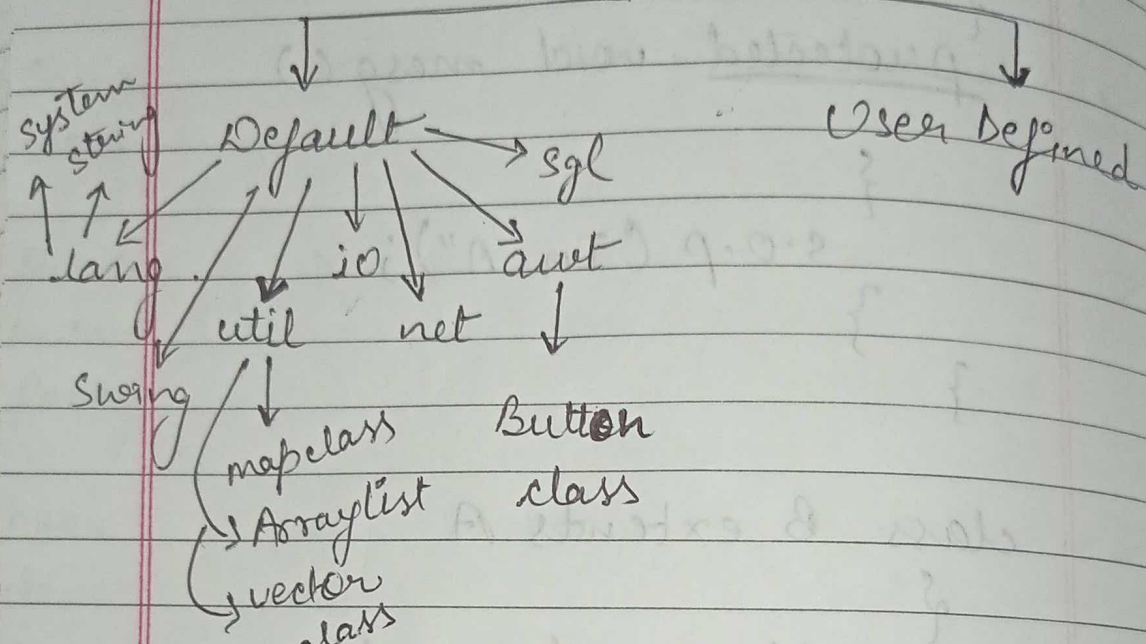
Date : / /
Page :

	Private	Default	Protected	Public
Same class	✓	✓	✓	✓
Subclass in same package	X	✓	✓	✓
Other classes in same package	X	✓	✓	✓
Subclass in other package	X	X	✓	✓
Non Sub class in other package	X	X	X	✓

```
class A
{
  protected void mesg()
    {
      s.o.p("In A");
    }
}

class B extends A
  {
    public void mesg()
        {
          sop("In B");
        }
    psvm(  )
      {
        B obj = new B();
        obj.mesg()
      }
  }
```

8/09/2022

Packages



having collision

Access protection

Date : / /
Page :

```
package pack;

import java.lang.*;
import mypack.*;
class A
  {
     psvm (      )
       {    B obj = new B();
       {    obj.message();
         S.O.P ("I am A");
       }
  }
```

```
A.java   ← current directory      // save
javac -d . A.java                 // compile
java pack.A                       // Interpret
```

```
package mypack;

public class B {
  public void message()
        {
          SOP("I am B")
        }
  }
```

If you skip to import then use
=> fully qualified

```
package pack;
class A
{ psvm(  )
  { mypack.B.obj = new.
    obj.message();        mypack.B();
  }
}
```

If you import the package all the classes & interface of the package will be imported excluding the classes & interfaces of sub packages.
Hence, you need to import the sub package as well.

javac classpath c:\class pack.A

enterpeize, webpage, mobile, standalone

```
import java.lang.*;

class A
  {
    privati void message()
    {
      s.o.p ("In A");
    }

  class B extends A
    {
      public static void main (String args[])
```

× × ×

```
package pack;

public class A

  private void mesg1()
    {
    }
  public void mesg2()
    {
    }
  void mesg3()
    {
    }
```

```
protected void msg4()
    {
    }

package mypack1;

import pack.*;

{
  psvm(  )
   {
     A obj = new A();
     obj.msg1(); // error
     obj.msg2(); // run
     obj.msg3(); // error
     obj.msg4(); // run
```

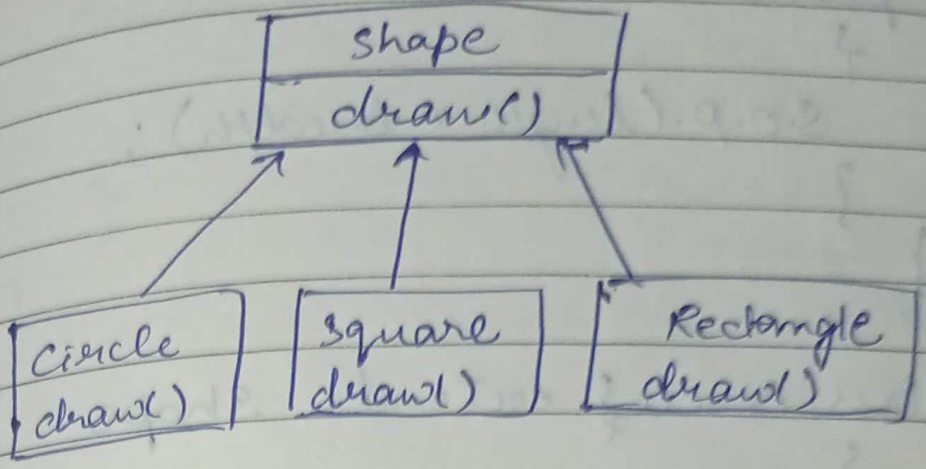
2)

12/sep

Polymorphism

many ← poly morphism → form

1) It means the ability by which a single variable of given type can be used to reference objects of different types & to automatically the method that is specific type of the object the variable references



2) The main benefit of polymorphism is that it reduces the complexity by allowing the same name to perform different task.

```
class Shape
{
  public void area()
  {
    S.O.P.("Area is being calculated");
  }
}

class Rectangle extends Shape()
  {
    private double length, breadth;

    Rectangle(double l, double b){
    {
        length = l;
    } breadth = b;
```

```
    public void area()
    {
      S.O.P.(length * breadth);
    }
}

class Circle extends Shape
  {
  private double radius;

    Circle(double r)
      {
        radius = r;
      }
    public void area()
      {
        S.O.P(Math * radius * radius);
      }
  }

class Test
{
 psmv(S a[])
 {
  Shape s; // Reference variable of
                    Super class
Rectangle reg = new Rectangle(10,5);

          s = reg;
          s.area();
```

1)

2)

Factory Method
↳ Inbuilt method

3)

4)

Date: / /
Page:

```
Circle c = new Circle(5);
S=c;
S.area();
}
}
```

Dynamic method Dispatcher
Dynamic Binding

behavior

The polymorphic can only be achieve if

1) The method call for sub call object must be through a super class reference variable

2) The method must be declared in super class & defined in sub class

3) The method in superclass & subclass must have same name & parameter list with a same of parameters where corresponding parameters must be have the same type.
(i.e. override)

4) The method return type must either be the same in the super class & subclass or must be a co-variant return type.

Date : / /
Page :

* Return types are said to be covariant if the return type of method in the derived class is the sub class of the return type of base class

5) The method access specifier must be no more restricted in the subclass than in the super class.

Abstract classes

```
abstract class shape
    {
abstract public void area();
abstrad public void circumference();
    }
```

abstract class ka object nahi bna skte

abstract class ko final nahi bna skte

Object nahi create kar skte

abstract class mei data members, methods & constructor bhi ho skta hai

ager sub class sare method override kar rhi hai super class ke to uss class ka name concrete ho jaega

Date : / /
Page :

However agr sare nahi kare override atleast one kar rhi toh, abstract- name hoga subclass ka.

We should have atleast one abstract method in abstract class

constructor aswellas static method cannot be abstract.

```
Rectangle reg = new Rectangle(10,5);
        S = r1
        s.area();
        s.circumference
```

```
public void circumferene()
    {
        s.o.p(2*(length+breadth));
    }
```

Date : / /
Page :

13/sep

Interfaces

always contain abstract method and data members also
It provides 100% abstraction

```
interface Shape
    { int i ; // public static final
public  <- [ abstract void area();
             abstract void circumference();
    }

class Rectangle implements shape
{ private double length, breadth;

Rectangle (double l, double b)
    {
      length = l;
      breadth = b;
    }
public void area()
  {
     S.O.P("Ar. of R =" + length * breadth);
  }
pubic void cimumference()
  {
   Sop("Perimt of Rectangle" + 2*(length + breadth));
  }
```

```
class Circle implements Shape
{ private double radius;
  circle (double r)
  {
    radius = r;
  }
  public void area ()
    {
      s.o.p (Math.PI * radius * radius);
    }
  public void circumference ()
    {
      s.o.p (Math.PI * radius * 2);
    }
}

class Test
  {
    psvm ( )
      {
        Shape [ ] s = new Shape [2];
        s[0] = new Rectangle (2, 5);
```

Reference variable

```
        s[i] = new Circle (5);

    for(int i=0; i<s.length; i++)
      {  s.o.p(s[i].area());
         sop(s[i].circumference());
      }
    }
}
```

1) Interfaces are easier to with an inheritance because you don't have to worry any implementation details in interface.

2) It facilitates multiple inheritance which is not possible with classes

3) It allows object of unrelated classes to be processed polymorphically.

(4) ✓ class A implements C, D (Possible)

✗ class A extends C, D (Not possible)

✓ class A extends B implements C (possible)

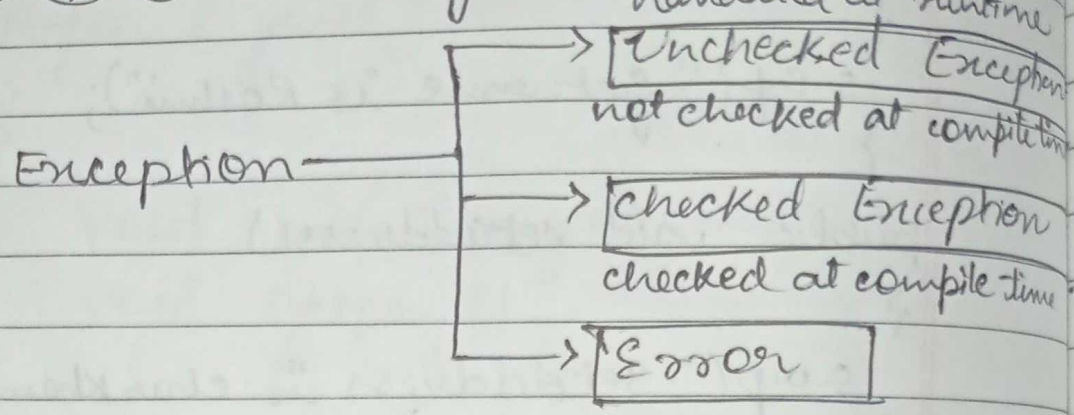
Date: / /
Page:

```
interface LengthConversionUnits
{   double FEET_TO_INCHES = 12.0;
    double FEET_TO_METER = 0.3048;
    double KM_TO_MILES = 0.62137;
    double KM_TO_NAUTMILES = 0.53996;
}

interface LengthConversion extends LengthConversionUnits
{
    double feet to Meters (double f);
    double feet to meter (double f);
    double km to Miles (double f);
    double kmTO Nautmiles (double f);
}

class Conversion implements LengthConversion
{
    conversion {
    public double feet to inches (double f)
        {
            return f * FEET_TO_INCHES;
        }
```

```
public double feet to Meter(double f)
{
  return f * FEET_TO_METER;
}

public double km to Miles (double f);
{
  return f * KM_TO_MILES;
}

puble double km to Nautmiles (double f);
{
  return f * KM_TO_NAUTMILES;
}

class conversionDemo
{
  psmv(  )
  {
    conversion cl = new Conversion( );

      cl.feet to Inches (12);
      cl.feet to Meter (2);
      cl. km to miles (5);
      cl. km to nautmiles (8);
```

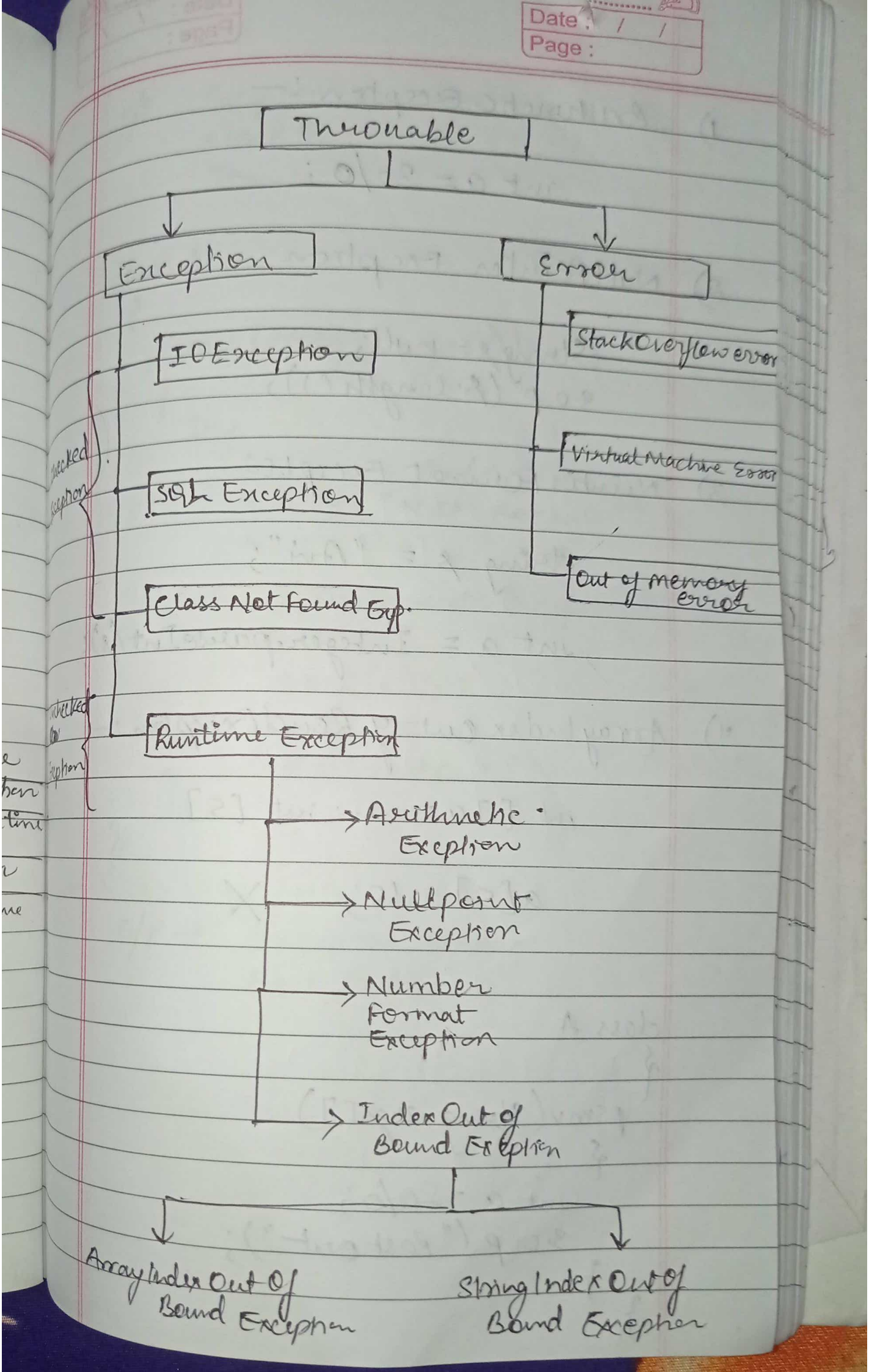
15/sep

Exception Handling



Keyword →

try, catch, finally, throw, throws


Throwable
Exception
Error
IOException
SQL Exception
Class Not Found Exp.
Runtime Exception
StackOverflow error
Virtual Machine Error
Out of memory error
Arithmetic Exception
Nullpoint Exception
Number Format Exception
Index Out of Bound Exception
Array Index Out Of Bound Exception
String Index Out Of Bound Exception

Date : / /
Page :

1) Arithmetic Exception :-

```
int a = 30/0;
```

2) NullPointer Exception

```
String s = null;
s.o.p (s.length());
```

3) Number Format Exception

```
String s = "Avi";
int a = Integer.parseInt(s);
```

4) ArrayIndex out of BoundException :-

```
int [] a = new int [5]
a[5] = 10;   X
```

```
class A
{
  psmv(String args[])
  {
    int a = 20/0;
    s.o.p ("Rest out");
  }
}
```

Date : / /
Page :

o/p Exception Message
stack trace
Terminate

```
class A
{
  psvm (String args[])
  {
    try
    { int a = 20/0;
      s.o.p ("Not Execute") X
    }
    catch (ArithmeticException e)
    {
      s.o.p(e);
    }
    s.o.p ("Rest Out");
  }
}
```

object of exception class

o/p → Exception Message
Rest Out

Page :

A try can be followed with multiple catch block but at the time only one catch will arise

```
class A
{
  psmV(String args[])
  {
    try
      {
        int a[] = new int[5];
        a[5] = 30/0;
      }
  }
  catch(Arithmetic Exception e)
  {
    s.o.p(e);
  }

  catch(ArrayIndexOutOfBound Exception)
  {
    s.o.p(e);
  }
  catch(Exception e)
  {
    s.o.p(e);
  }
  s.o.p("Rest Out");
  }
}
```

mon
19/09

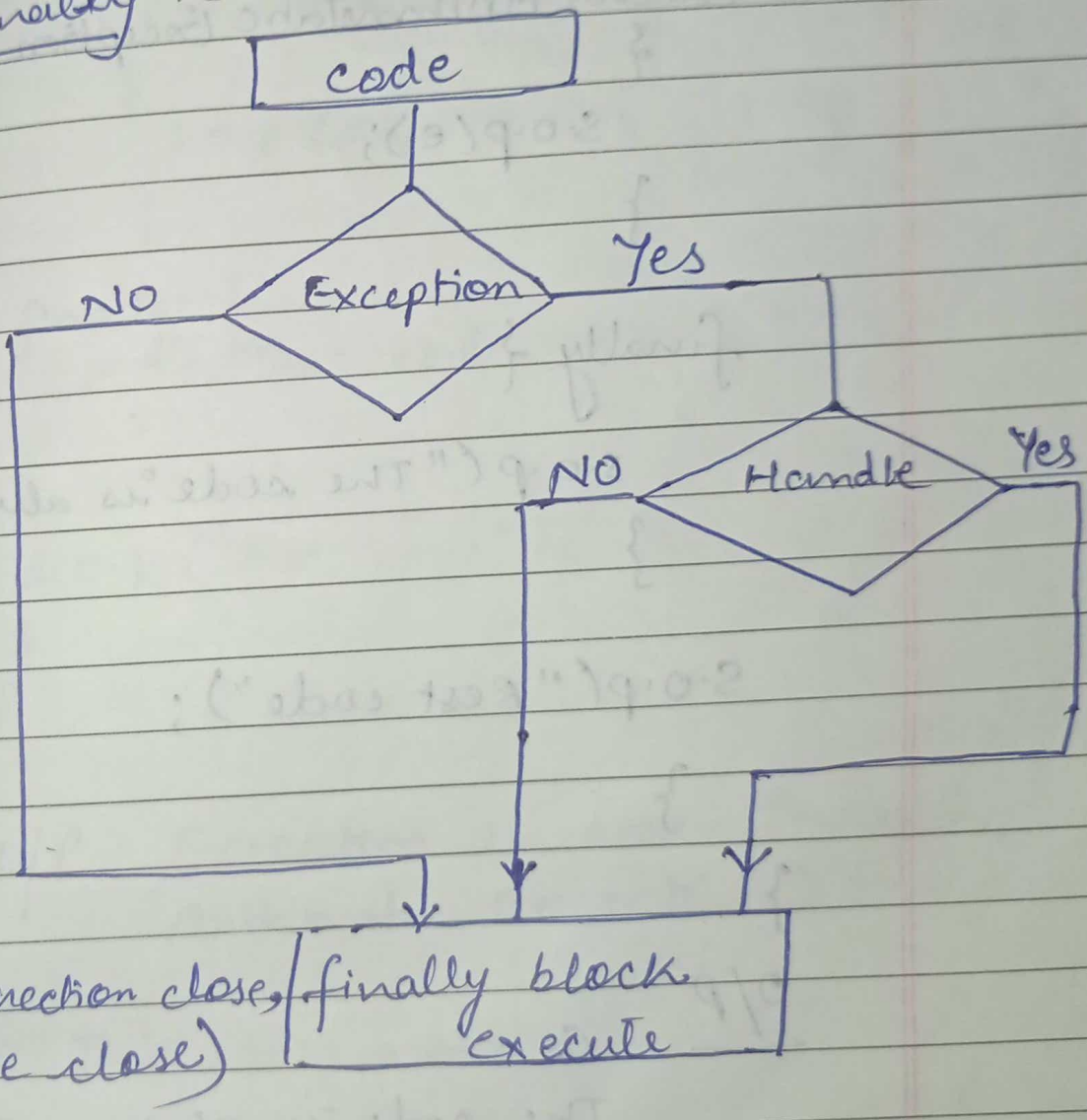
Date : / /
Page :

qf Exception in Main Thread
Arithmetic Exception : / zero

Rest Out.

Flow of [redacted] catch is specialized to Generalize.
Wrna C.T. Error aa jaegi

finally is block



// (connection close, file close)

Date : / /
Page :

* For Each try block there can be zero or more catch blocks but only one finally block

The finally will not be executed if the progrm exits either by calling system.exit or by causing fatal error that causes the process to abot.

throw :-

→ It is used to throw an exception explicitly

throw new ArithmaticException("new Exception");

```
class Treat
{
    psvm(String args[])
    {
        try{
            check(18);
        }
        catch(Exception e)
        {
            s.o.p(e);
        }
    }
}
```

```
class Treat
{
  static void check(int a)
  {
    if (a<18)
    {
      throw new ArithmaticExcepn
                ("Under age");
    }
    else
    {
      s.op("Person can vote");
    }
  }
```

```
import java.io.*;

class Demo
{
public static void meth1() throws FileNotFound
                                  Excep
                                  -tion
{
FileReader f = new FileReader("D:\\
                Sraycta.\\ txt");
BufferedReader bf = new BufferedReader(f);
throw new FileNotFoundException();
}
```

Date : / /
Page :

20/09
Tue.

```
psvm (String args[]);
 {
  try
   {
     Meth1();
   }
  catch (Exception e)
   {
    s.o.p(e);
   }
 } //main
} //class
```

checked exception ko throw karne ke liye try-catch lagana mandatory hai

throws method ke sath likhte hai

throw
creating an object of exception class

throws
for declaration
used with method

Date : / /
Page :

Customised Exception / User Defined Exception

```
class CustomException extends Except
                                -ion
{
  CustomException (String str)
    {
      super(str);
    }
}

class Demo
{
  psvm ( String args[]):
  {
    try
      {
        throw new CustomException
              ("CustomException
                    occured");
      }

    catch (CustomException ce)
      {
        s.o.p (" Exception caught ");
        s.o.p (ce. getMessage());
      }
  }
}
```

o/p ⇒ Exception caught
Custom Exception Occured.

Execution Propogation Unchecked Exception

Date : / /
Page :

```
class TestException
{

  void p()
  {
    int a = 35/0;
  }

  void n()
  {
    p();
  }

  void m()
  {
    try{
        n();
       }
    catch (ArithmaticException e)
       {
        s.o.p(e);
       }
  psvm (String args[])
  {
    TestException obj = new TestException();
        obj.m();
    s.o.p ("Program Exe");
  }
}
```

O/P ⇒ Exception in p()
/ zero
Program Executed

By default unchecked exception are forwarded in the falling chain

- Checked Exception cannot be propogate on their own

```
import java.io.*;

class TestException
{
 void p() throws IOException
  {
  throw new IOException("Device");
  }

 void n() throws IOException
```

Date: / /
Page:

Q Which exception should be Declared

A Checked Exception should be declared.

throw	throws
used to throw an exception explicitly in the code, inside the function or block of code.	Used in the method signature to declare an exception which might be thrown by the function while the execution of code.
~~throw can declare~~ is followed by instance	is followed by class name.
is used with in the method	used with the method signature
we are allowed to throw only one exception at a time	we can declare multiple exceptions using throws

22/9

Keyword block method

Date : / /
Page :

↳ code of clearance

system.exit ⇒ finally not execute

checked exception can not be throw,

throws is used only in the declar of exception it cannot handle it.

finalize use for garbage collection

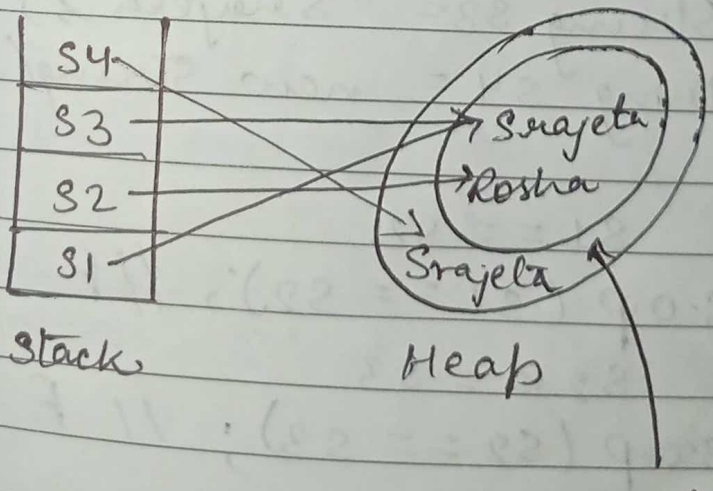
It is the method in java used to perform cleanup processing just before object is garbage collected.

It is used with the objects.

String

Method-1

String s1= "Srajeta"; // Literal Method
String s2 = "Rosha";
String s3 = "Srajeta";



Method-2 (new keyword)

String s4 = new String ("Srujela");

Comparing String

1) Equals Method [equals()]
2) == operator // reference
3) compareTo() method

1) String s1 = "Srujela";
   String s2 = "srujela";

Return Boolean variable

equals method)

→ check content of string

s.o.p(s1.equal(s2)); // false
s.o.p(s1.equalsIgnoreCase(s2)); // True

2) String s3 = "Srujela";
   String s4 = new String("srujela");

s1 == s2
s.o.p (s1 == s2); // F
s2 == s3
s.o.p (s2 == s3); // F
s3 == s4
s.o.p. (s3 == s4); // F
s.o.p (s1 == s3); // T

3)

26/9/2

a)

b)

Date : / /
Page :

3) return +ve, 0, -ve value

+ve → s1 > s2

0 → both equal

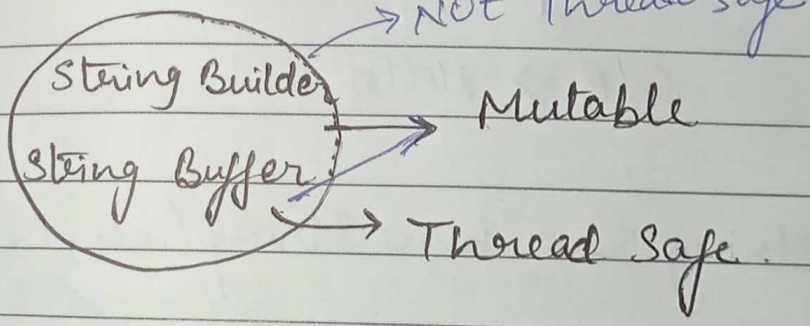
-ve → s1 < s2

s.o.p (s1.compareTo(s2)); +ve -ve

s.o.p (s1.compareTo(s3)); 0

s.o.p (s2.compareTo(s1)); +ve

ascii value of capital letter ⇒ less than small letter.



16/9/22

Strings

a) implicit method (literal method)

b) new keyword

Immutable

→ Memory management

→ Security

→ Thread safety

Date : / /
Page :

String comparison :-

* String concatination

Use of constructor

(I) String s1 = new String();
String s2 = new String (s1);

(III) String s1 = new String ("Srajda");

(II) char [] ch = {'H', 'e', 'l', 'l', 'o'};

String s1 = new String (ch);
s.o.p(s1);

O/P ⇒ Hello

String s2 = new String (ch, 1, 4);

start index (inclusive) — last index (exclusive)

s.o.p(s2);

O/P ⇒ ell.

byte [] b = {"32", "28", "40"}

String s1 = new String (b);
s.o.p(s1);

Date : / /
Page :

String Concatination

```
class Test
{
    int id;
    String name;

Test(int i, String n)
{
  id = i;
  name = n;
}

void display()
{
s.o.p(id + " " + name);
}
```

① + operator

```
String s = "Srajeta" + "Gupta";
s.o.p(s);
```

⇒ String Builder

(new StringBuilder()).append("Srajeta").
append("Gupta").toString()

compiler convert string to the object of String Builder class

Date : / /
Page :

(2) String s = 50+30+"Srayela"+40+60
s.o.p(s);

O/P // 80 Srayela 4060

(3) String s1 = "Srayela";
String s2 = "Gupta";
s1 = s1.concat(s2);
s.o.p(s1);

O/P — Srayela Gupta

27/09

(4) Format specifiers

String s1 = 'Hi';
String s2 = "Java";

String s3 = String.format("%s %s", s1, s2);

s.o.p(s3);

(5) String s = String.join(" ", s1, s2);
s.o.p(s);

Java 8 & above

(6)

(7) class

Date : / /
Page :

StringJoiner class

Java 8 ↑

```
class Test
{
  psvm(String args[])
  {
    StringJoiner s = new StringJoiner(",");
    s.add("Hi");
    s.add("Java");
    s.o.p(s)
```

13/09

toString() method :-

```
class Student
{
  int rollno;
  String name;
  String city;

Student(int r, String n, String c)
  {
    rollno = r;
    name = n;
    city = c;
  }
psvm()
  {
    Student sl = new
```

```
public String toString()
    {
      return rollno + " " + name +
                    " " + city;
    }

psvm ( )
  {
   Student s1 = new Studen(1, "Sreeja", "Jamm");
  }  s.o.p(s1);
}
```

o/p . Student@ deefc---

o/p 1 , Sreeja Jamm

To represent any object as string
toString

Printing any object internally invokes toString() method

SubString

```
String s1 = "Hello Java";
   s.o.p (s1.substring(4));
```

0 1 2 3 4 5 6 7 8 9

// - Java

```
s.o.p (s1.substring(2, 7));
```

// llo_J

Date : / /
Page :

```
sop(s1.toUpperCase()); // HELLO JAVA
sop(s1.toLowerCase()); // hello java
```

* String s2 = " Jammu ";
  s.o.p (s2.trim()); // Jammu

```
s.o.p(s1.startsWith("He")); // True
s.o.p(s1.endwith("v")); //false
s.o.p(s1.charAt(3)); // l
s.o.p(s1.length()); // 10
```

* intern() method → A pool of strings is inially empty is maintain privately by class string.
when intern() method is invoke, if the pool already contain string equal to this string method object as determine by the string method then the string of pool is return

```
String s = new String("SG.");
s3 = s.intern();
s.o.p(s3);
// S.G.
```