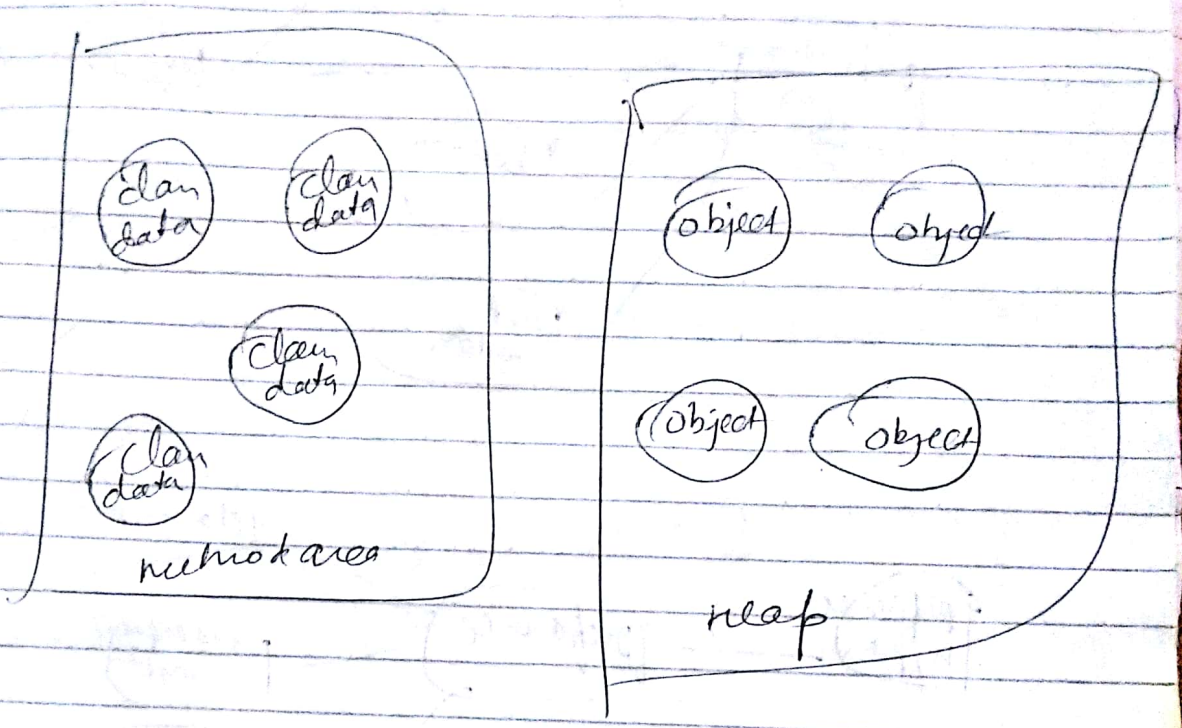
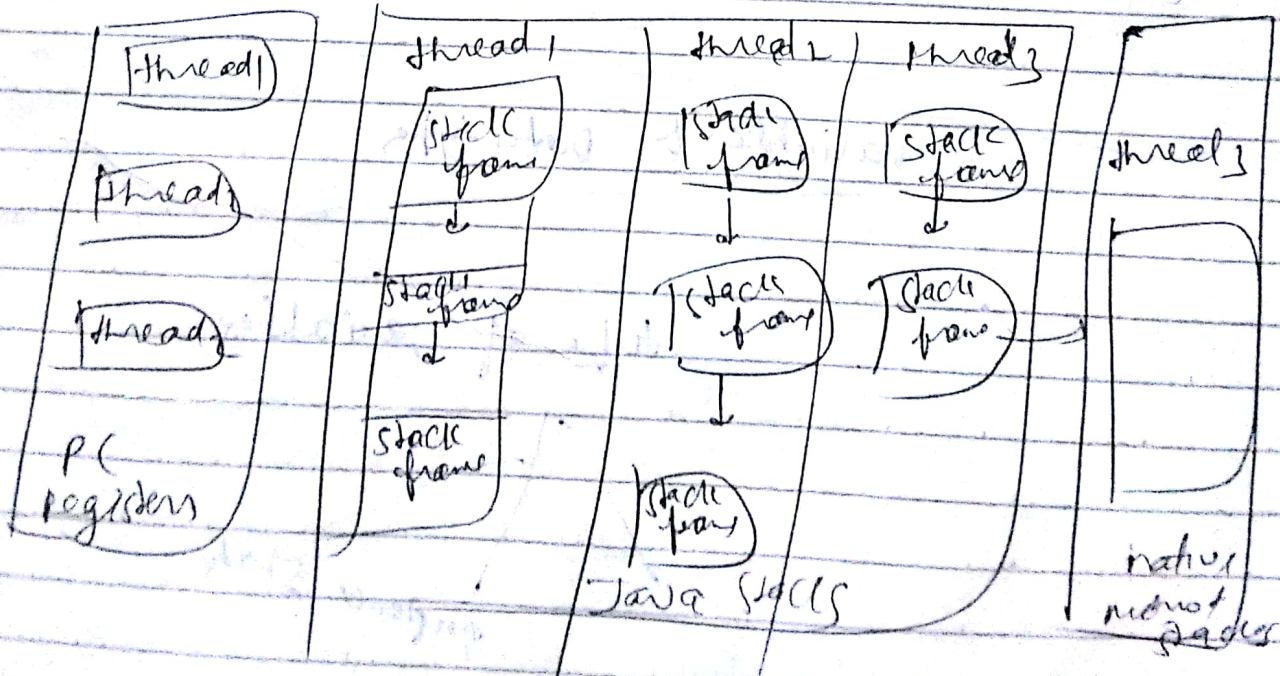
Organization of JVM

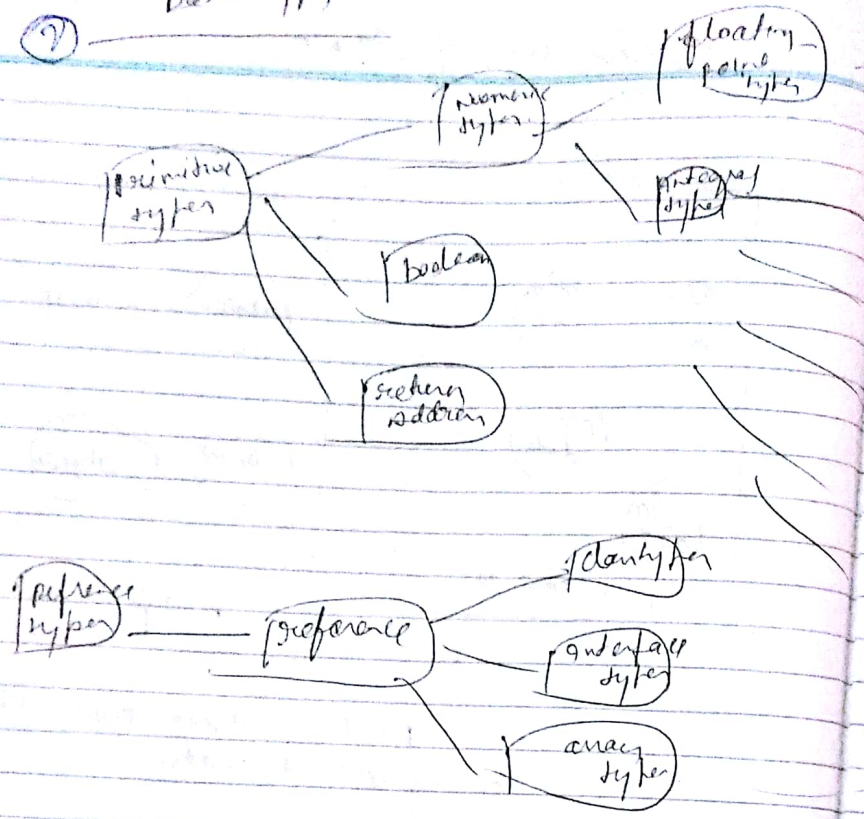


Runtime data area shared among all threads.



Runtime data areas exclusive to each thread

Data types



class A {

int data = 50;

static int m=100

void method()

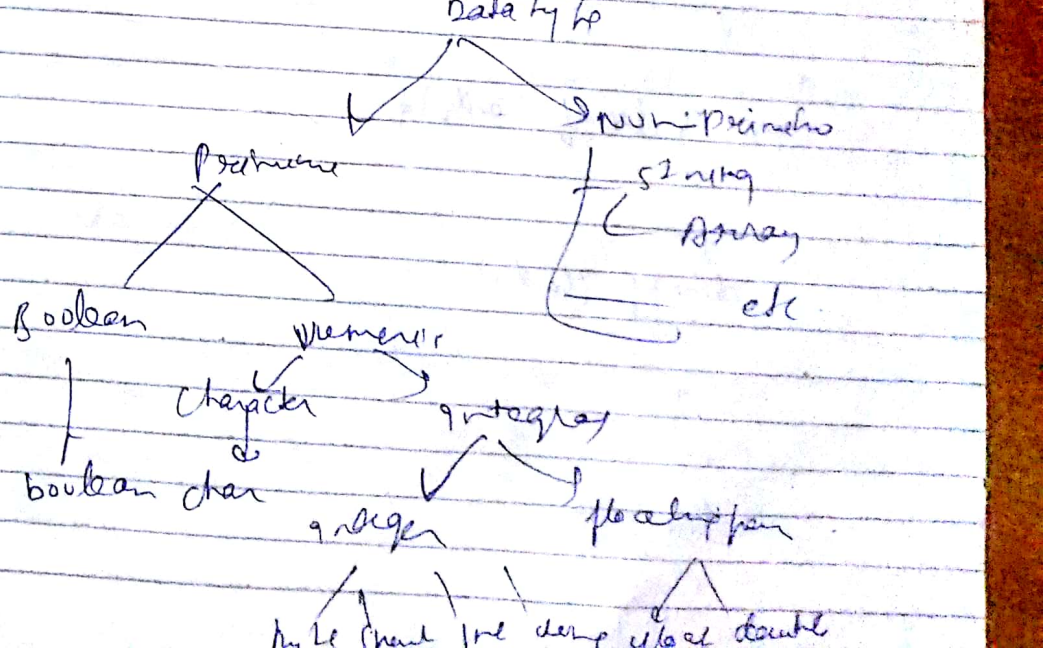
{

int n.90 local variable

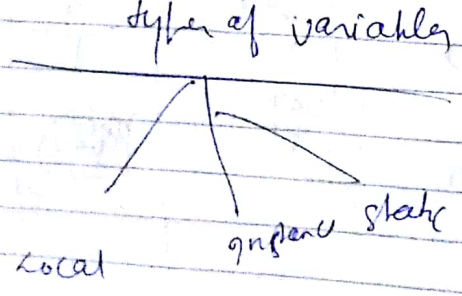
}

} end of class

Data types of in Java



variables & Data types



(4)

Control flow

```
if (condition) {
    // code
}

if (condition) {
    // code
} else {
    // code
}
```

Java if-else if ladder

```
if (condition1)
{

} else if (cond2) {

}
else if (cond3) {

}

else {

}
```

Switch case (5)

```
switch ( exp ) {
  case value1:
    // code
  break;
  case value2:
    // code
  break;
  ...
  default:
    code to be executed if all cases are not matched
}
```

for loop

```
for (initialization, condition, incr/decr) {
} // code to be executed
```

```
public class forexample{
psvm(String[] args){
for(int i=1;i<=10;i++)
{
SOP(i);
}
}
}
```

Java for-each loop (we use to traverse array)

```
for (Type var : array){
// code
}
```

example

```
public class forarrayloop{
psvm(String[] args)
{
int arr[]={12,23,04};
for (int i:arr){
SOP(i)
```

while loop

```
while (condition){
// code
}
```

Java do-while loop

```
do {
// code
} while (condition);
```

Java Break -

```
public class Breakex{
psvm(String[] args){
for(int i=1;i<=10;i++)
{
if(i==5)
break;
}
SOP(i);
}
}
```

oops concept

object
class
inheritance
polymorphism

Abstraction using abstract class & interface
Encapsulation

Java Datatype

byte → default value → 0
1 byte

short → default value → 0
taken 2 bytes

int → 4 bytes
default value → 0
range min → -2,147,483,648
max 2,147,483,647

long
8 bytes

float
32 bit → 4 bytes (single precision)
default value → 0.0f
syntax float f1 = 234.5f

double → 8 bytes
(double precision) default value → 0.0d

boolean - 1 bit either true or false

char - 2 bytes

char letterA = 'A'

Java literals

WRAPPER CLASSES

mechanism to convert primitive into object and object into primitive

Primitive	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

```
class A
{
    psvm (String args[])
    {
        byte grade = 2;
        int marks = 50;
        float price = 8.6f;
        double rate = 50.5;
                // wrapping Datatypes into class
        Byte g1 = new Byte (grade);
        Integer m1 = new Integer(marks);
        Float f1 = new Float (price);
        Double d1 = new Double( rate);
        SOP( "Byte object g1 "+g1);
        SOP ("Integer object "+m1);
        SOP ("float ---- "+f1);
        SOP ("double ---- "+d1);

        //Unwrapping

        byte bv = g1.byteValue();
```



```
        int iv = m1.intValue();
        float fv = f1.floatValue();
        double dv = g1.doubleValue();
        // print the values from above
        SOP (bv);
        SOP(iv);
        SOP(fv);
        SOP(dv);
    }
}
```

Creating objects of the wrapper class

```
Integer intObj = new Integer(34);
Integer intObj = new Integer("34");
```

Retrieving the value wrapped by a wrapper class object

```
int x = intObj.intValue();
```

Conversion between Data types

```
int x = Integer.parseInt("134");
double y = Double.parseDouble("134.7");
String s1 = String.valueOf('9');        s1 = "9"
String s2 = String.valueOf(true);       s2 = "true"
```

Array :- array is used to store the group of values.

```
int [] a;
int a[];
a = new int[7];
int[] a = new int[5];
```



```
public class Array
{
  psvm (String arg[])
  {
    int [] a = new int[7];
    a[2] = 3;
    a[5] = 7;
    sop("value of a[5] before passing as to changeElement is "+a[5]);
    changeElement(a[5]);
    sop("value of a[5] after passing it to method changeElement is "+a[5]);
  }
```

Processing array using loops

```
int [] a = new int[7];
for(int i=0; i<a.length; i++)
{
  a[i] = (i+1);
}
```

```
for (int i=0; i<a.length; i++)
{
    SOP("element at index "+i+" is "+a[i]);
}
```

Multidimensional array :-

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

for (int i=0; i< a.length ; i++)
{
    for (int j=0; j<a[i].length ; j++)
        SOP(a[i][j]);
    SOP();
}
```

Arithmetic operators

```
+
-
*
/
%
```


Conditional operators

```
==   equal to
!=   Not equal
>    greater than
<    less than
>=   greater than or equal to
<=
```

Logical operators

```
&&   short circuit AND
||   short circuit OR
```

Ternary operator or ? operator or conditional operator

```
x = (a<b) ? 50 : 60;
```

(a<b) → true, 50 → false

The Not operator

It is also called bitwise complement the unary Not operator will inverts all of the bits of its operand.

The AND operator

The OR operator

XOR (exclusive OR) operator

XOR produces a 0 bit if both operands are same otherwise 1 bit. Similarly for boolean operands it will result in false if both operands are same (either false or both true) the result will be true.

```
public class bitwise_logical_demo
{
    PSVM(String args[])
    {
        int a=65 → 1000001
        int b=33 → 0100001
        SOP("a&b="+(a&b));
        SOP("a|b="+(a|b));
        SOP("a^b="+(a^b));
        SOP("~a="+~a);
```


bitwise shift operators

>> Signed Right shift

int a=8 → 1000 (representation)

SOP(a>>1); // this will print 4 (0100)

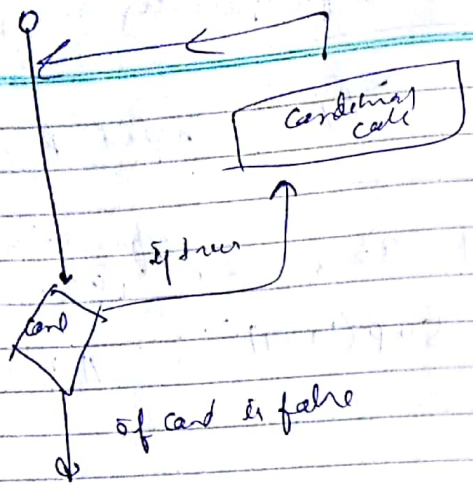
<< (signed left shift)

int a=8 → 1000

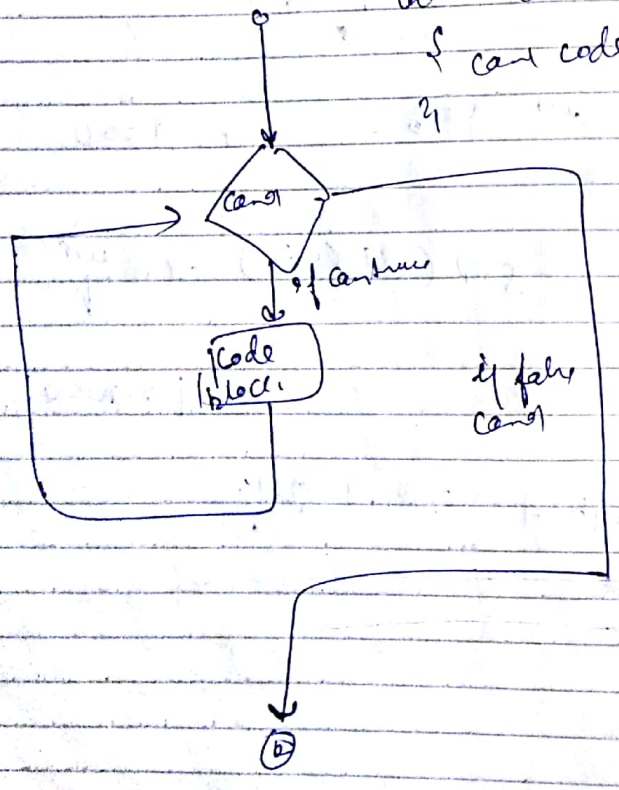
SOP(a<<2) → will print 32

→ 100000

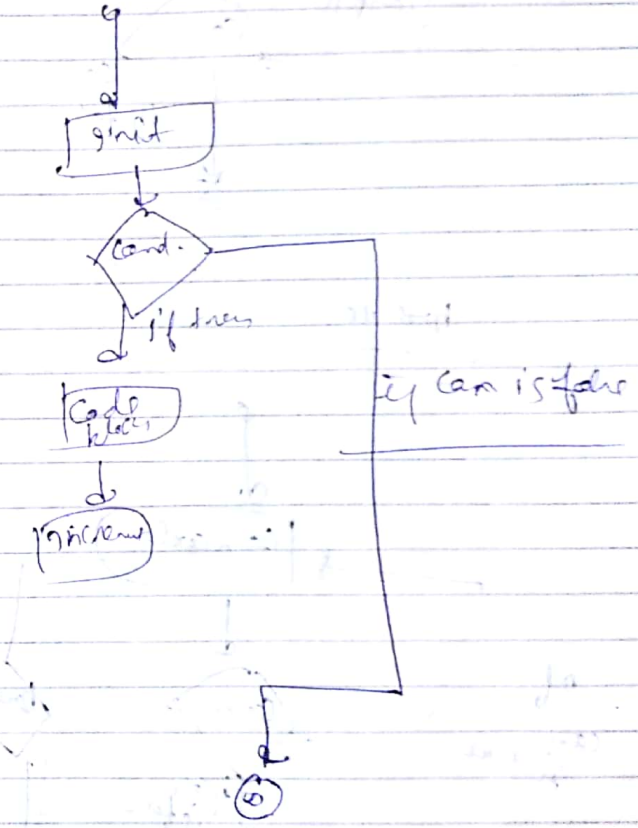
>>> unsigned right shift



while (cond)
{ cond code;
}

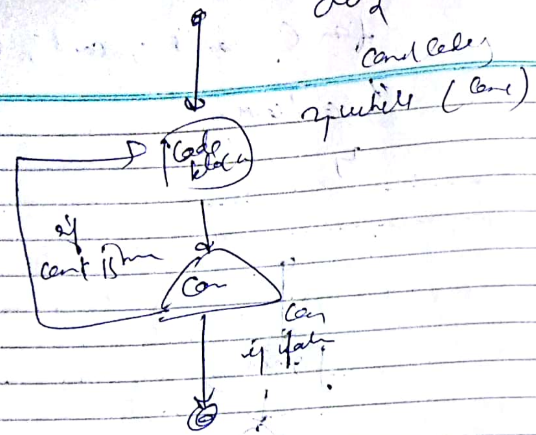


for (init ; cond ; incr)
{ cond code;
}

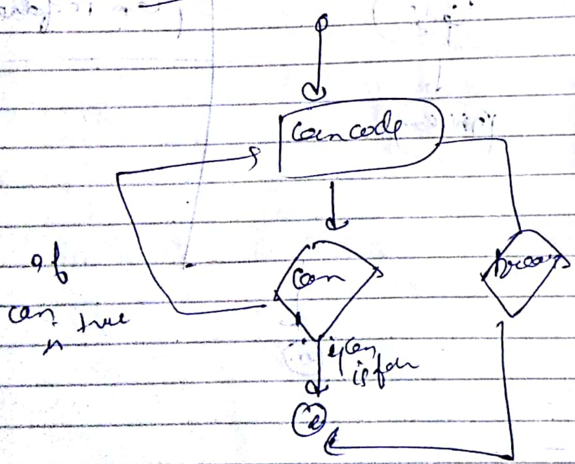


DO while

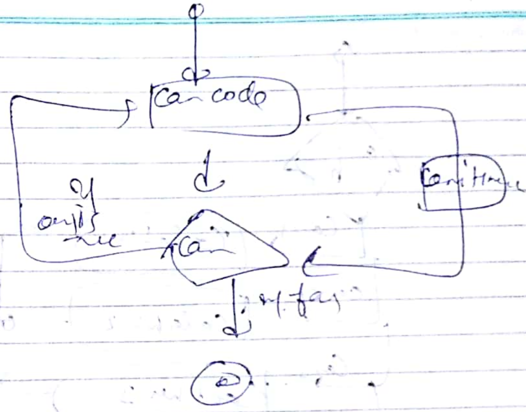
do while (con)

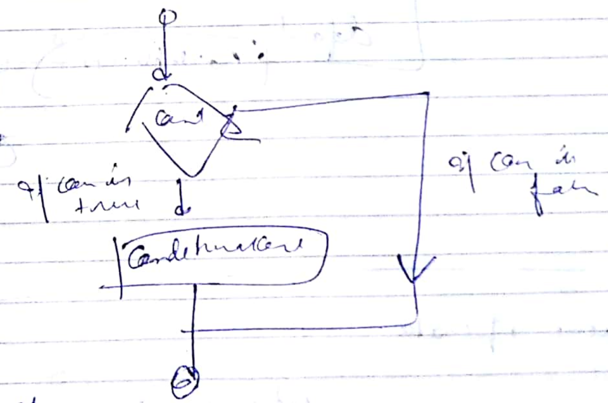


break

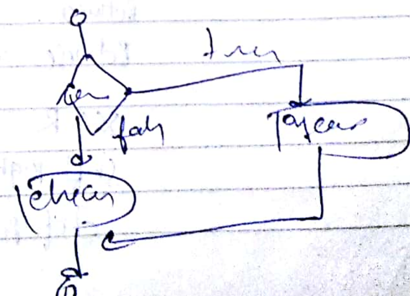


continue

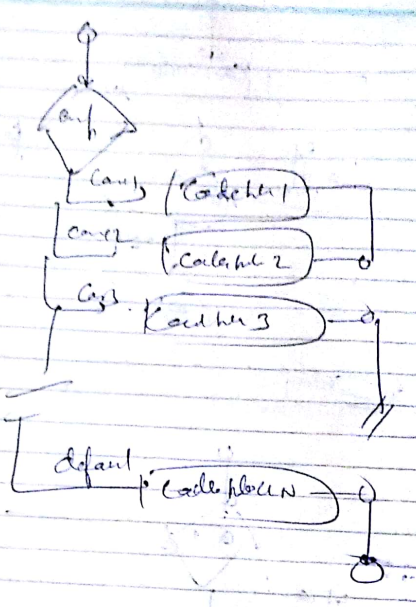




if else



Switch



Bitwise operator

~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	XOR
>>	Shift Right
>>>	Shift Right zero fill

<< Shift left

&= Bitwise and assignment

|= Bitwise or assignment

Bitwise NOT

00101010 to
becomes
11010101

The bitwise AND

00101010
00001111

00001010

bitwise OR

00101010
00001111

00101111

Bitwise XOR

00101010
00001111

00100101 XOR

Left Shift

a<<2

a = 64 → 01000000

10000 0000 → 256

Right Shift

a = 35 = 00100011 (2 bits are lost)

>>2

00001000 → 8

Unsigned Right Shift

11111111 11111111 11111111 11111111 → -1

>>>24

00000000 00000000 00000000 11111111

Boolean logical operators :

Operator	Result
&	Logical AND
|	Logical OR
^	Logical XOR
||	Short-circuit OR
&&	Short-circuit AND
!	Logical unary NOT
&=	AND Assignment
|=	OR Assignment
^=	XOR Assignment
==	equal to
!=	Not equal to
?:	Ternary if-then-else

A	B	A|B	A&B	A^B	!A
false	false	false	false	false	true
True	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

Classes
Control flow
Arrays
Logical operators

StringBuffer & String class (String is immutable & StringBuffer is mutable)

```
public class Test
{
    psvm(String args[])
    {
        StringBuffer sbuffer = new StringBuffer("Hello");
        sbuffer.append("StringBuffer");
        SOP(sbuffer);
    }
}
```

StringBuffer methods

(1) public StringBuffer append(String s)

(2) public StringBuffer reverse()

```
public class test {
    psvm StringBuffer sbuff = new StringBuffer("abcdefg");
```



```
        sb.delete(3,7)
        SOP(sb)
    }
}
```

String length

```
public class Test
{
    psvm(String args[])
    {
        String s1 = "Hello World";
        int len = s1.length();
        SOP(len)
    }
}
```

for concatenation

s1.concat(s2);

charAt()

String s1 = "Strings are immutable";
char result = s1.charAt(8);
SOP(result);

(30) String : compareTo(Object o)

public int compareTo(Object o)

equals()

public boolean equals(Object anyObject)

getBytes(String s1)

public byte[] getBytes()

public class test {

psvm(String args[])

{

String s1 = new String

("welcome
to tutorial
point");

byte[] str = s1.getBytes();

SOP(str)

}

}

(31) String indexOf()

public int indexOf(int ch)

Replace

public String replace(char oldCh,
char newCh)

String substring()

public String substring(int beginIndex)

or

public String substring(int beginIndex,
int endIndex)

lowercase

public String toLowerCase()

Java Arrays

datatype[] arrayrefvar

or

datatype arrayrefvar[];

ex:

Creating array

arrayrefvar = new datatype[size];

double[] mytest = new double[10];

Test Array

```
public class testarray {
    public static void main(String args[])
    {
        double mytest[] = {1.9, 1.2, 3.5, 4.3};

        // print array
        for(int i=0; i<mytest.length; i++)
        {
            System.out.println(mytest[i] + " ");
        }

        // summing all elements
        double total = 0;
        for(int i=0; i<mytest.length; i++)
            total += mytest[i];
```

```
public class testarray{
    PSVM(String args[]){
        double[] myline = {1.9, 2.9, 3.4, 3.5};
        // print all array elements
        for(double element : myline)
            SOP(element);
    }
}
```

Java method

(creating method)

```
public static int methodname(int a, int b)
{
    // body
}
```

```
public class examplearray{
    PSVM(String args[])
    {
        int a = 4
        int b = 12
        int c = add(a, b)
        SOP(c);
    }
    public static int add(int m, int n)
    {
        int total
        total = m + n;
        return total;
    }
}
```

(36) passing parameters by value

```
public class swap5
{
    psvm(String args[])
    {
        int a = 10;
        int b = 45;
        sop("Before swapping" + a + b);
        swapfun(a,b);
        sop(a,b);
    }
    public static void swapfun(int a, int b)
    {
        int c = a;
        a = b;
        b = c;
        sop("after swapping" + a + b);
    }
}
```

(37) Method overloading

Advantage of method overloading

The constructor :

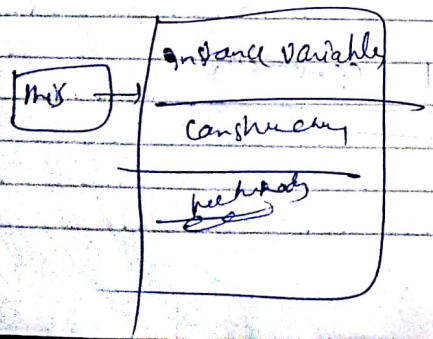
```
class myclass
{
    int x;
    // following is my constructor
    myclass()
    {
        x = 10;
    }
}
public class classcons
{
    psvm(String args[])
    {
        myclass t1 = new myclass();
        myclass t2 = new myclass();
        sop(t1.x + " " + t2.x);
    }
}
```

Parameterized constructor :

(39)

```
class myclass
{
    int x;
    myclass(int i)
    {
        x = i;
    }
}
public class classes
{
    psvm(String[] args)
    {
        myclass t1 = new myclass(10);
        myclass t2 = new myclass(20);
        sop(t1.x + ". " + t2.x);
    }
}
```

this keyword



(40)

```
class student
{
    int age;
    student(int age)
    {
        this.age = age;
    }
}
```

this keyword is used to

(1) Differentiate the instance variable from local variable if they have same names within a constructor or a method

(2) Call one type of constructor (Parametrized or default) from other in a class.

```
class student
{
    int age
    student()
    {
        this(20);
    }
    student(int age)
    {
        this.age = age;
    }
}
```

(41) The finalize method

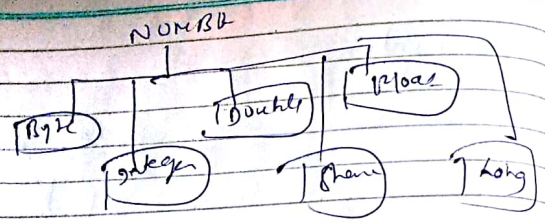
```
protected void finalize()
{
    // finalization code here
}
```

Scanner class : reading from any input.

```
import java.util.*;
public class ScannerTest
{
    public static void main(String args[])
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("enter your rollno");
        int rollno = sc.nextInt();
        System.out.println("enter your name");
        String name = sc.next();
        System.out.println("enter your fee");
        System.out.println(_____);
    }
    sc.close();
}
```

(42)

WRAPPER Class



Number methods

1) XX xValue()
2) compareTo()
3) equals()
4) valueof
5) toString()
6) parseInt

```
public class Test
{
    PSVM(String args[])
    {
        Integer X=5;
        // Returns byte primitive data byte
        SOP(X.byteValue());
        // returns double primitive
        SOP(X.doubleValue())
```



```
        SOP(X.longValue())
    }
}
```

compareTo()

```
public class Test
{
    PSVM(String args[])
    {
        Integer X=5;
        SOP(X.compareTo(3));
        SOP(X.compareTo(5));
        SOP(X.compareTo(7));
    }
}
```

1
0
-1

equals()

```
public class Test
{
    PSVM(String args[])
    {
        Integer X=5;
        Integer Y=10;
```

```
integer z=5;
    String a=5;
        SOP (X.equals(y));
            SOP( X.equals (z));
                SOP ( x.equals(a));
    }
}
```

valueOf()

```
Public class test{
    PSVM( String args[])
    {
        Integer x= Integer.valueOf(9);
        Double c= Double.valueOf(5);
        float = Float.valueOf("80");
```

toString() method

47

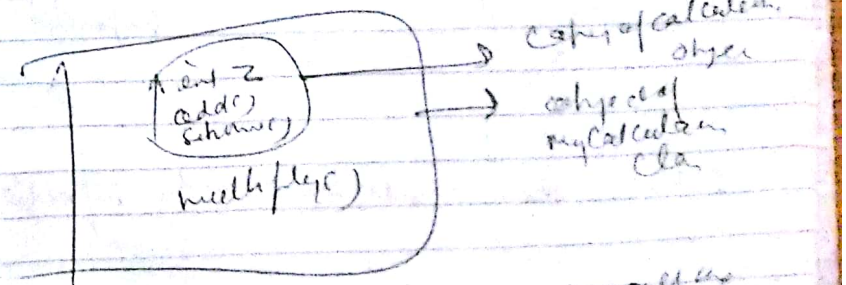
Java → inheritance

extends keyword

```
class super{
    ---
    ---
}
class sub extends super{
    ---
    ---
}
```

example

```
class calculation
{
    int z;
    public void add(int x, int y)
    {
        z=x+y;
        SOP(z);
    }
    public void subtract(int x, int y)
    {
        z=x-y;
        SOP(z);
    }
}
public class mycalculation extends calculation
{
    public void multiply(int x, int y)
    {
        z=x*y;
        SOP(z);
    }
    psvm(String args[])
    {
        int a=20, y=10;
        mycalculation demo = new mycalculation();
        demo.add(a,y);
        demo.subtract(a,y);
        demo.multiply(a,y);
    }
}
```



note → super class can not access methods of subclass.

4) The super keyword

(1) It is used to differentiate the members of superclass from the members of subclass, if they have the same names.

(2) It is used to invoke the superclass constructor from subclass.

Sample code

```
class super-class
{
    int num=20;
    public void display()
    {
        SOP("This is the display method of superclass");
    }
}
public class subclass extends super-class
{
    int num=10;
    public void display()
    {
        SOP("This is the display method of subclass");
    }
    public void my-method()
    {
        subclass sub = new subclass();
        sub.display();
        super.display();
        SOP("The value of variable of subclass is "+sub.num);
        SOP("The value of variable of superclass is "+super.num);
    }
    PSVM(String args[])
    {
        subclass obj = new subclass();
        obj.my-method();
    }
}
```

(2) Sample code

51

invoking superclass constructor

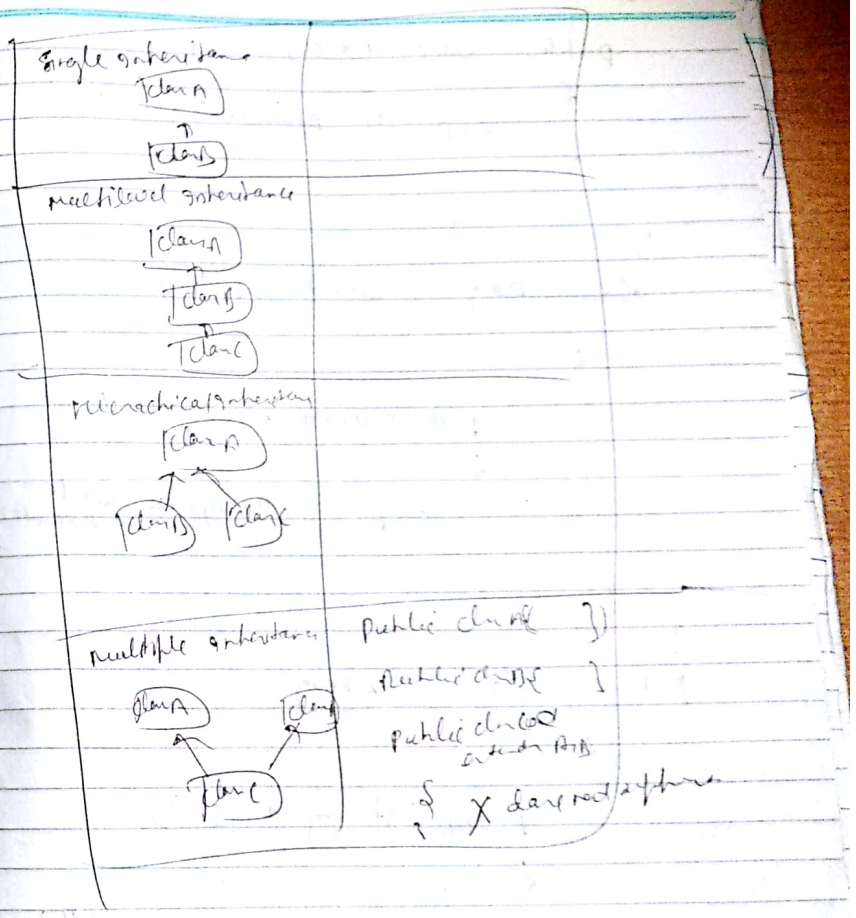
```
class superclass{
    int age;
    superclass (int age){
        this.age = age;
    }
    public void getAge()
    {
        SOP("The value of the
        variable name age
        in superclass is " + age);
    }
}

public class subclass extends
                    superclass{

    subclass (int age){
        super(age);
    }
    psvm (String args[])
    {
        subclass sub = new
    } } sub.getAge(); subclass(21);
```

52

Types of inheritance



Overriding

53

```
class Animal{
    public void move()
    {
        SOP("Animals can move");
    }
}
class Dog extends Animal
{
    public void move()
    {
        SOP("Dogs can walk and run");
    }
}
public class TestDog
{
    PSVM(String args[])
    {
        Animal a = new Animal();
        Animal b = new Dog();
        // Animal reference but Dog object
        a.move();
        b.move();
    }
}
```

54 using super keyword

```
class Animal{
    public void move()
    {
        SOP("Animals can move");
    }
}
class Dog extends Animal
{
    public void move()
    {
        super.move(); // invokes the superclass method
        SOP("Dogs can walk and run");
    }
}
public class TestDog
{
    PSVM(String args[])
    {
        Animal b = new Dog();
        b.move();
    }
}
```

Java polymorphism

```
public class employee
{
    private String name;
    private String address;
    private int number;

    public employee (String name,
                     String address,
                     int number)
    {
        SOP ("constructing an
              employee");
        this.name = name;
        this.address = address;
        this.number = number;
    }
    public void mailCheck()
    {
        SOP ("mailing a
              check to" +
              this.name
              + " " + this.address);
    }
```

```
    public String toString()
    {
        return name + " " + address
               + " " +
               number;
    }
    public String getName()
    {
        return name;
    }
    public String getAddress()
    {
        return address;
    }
    public void setAddress (String newAddress)
    {
        address = newAddress;
    }
    public int getNumber()
    {
        return number;
    }
}
```

```
Public class salary extends Employee
{
    private double salary;
    Public salary(String name,
                  String address,
                  int number, double
                  salary)
    {
        super(name, address, number);
        setSalary(salary);
    }
    Public void mailCheck()
    {
        SOP("Within mailCheck
             of salary
             class");
        SOP("Mailing check to"
            + getName() + "
            with salary" + salary);
    }
```



```
Public double getSalary()
{
    return salary;
}
Public void setSalary(double
                      newSalary)
{
    if(newSalary >= 0.0)
    {
        salary = newSalary;
    }
}
Public double computePay()
{
    SOP("Computing salary Pay
         for " + getName());
    return salary/52.0;
}
}
```

```
Public class VirtualDemo
{
    PSVM(String[] args)
    {
        salary s = newSalary("mod
```

Abstract class

```
abstract class A {
    abstract void callme();
    void callmetoo()
    {
        sop("this is a concrete method");
    }
}
class B extends A
{
    void callme()
    {
        sop("B's implementation of callme");
    }
```



```
}
class AbstractDemo {
    A a = new A();   // not possible
    B b = new B();
    b.callme();
    b.callmetoo();
}
```

Cannot create an object of abstract class

61

Abstract Class

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction in java

(1) Abstract class (0 to 100%)

(2) interface (100%)

```
abstract class A{}

abstract void printStatus();
        +
abstract class Bike{
    abstract void run();
}

class Honda extends Bike{
    void run()
    { SOP("running safely");}
psvm(String args[])
{
    Bike obj = new Honda();
    obj.run()
```

62

The interface in java is a mechanism to achieve fully abstraction

```
interface printable{
    int min=5;
    void print();
}
```

↓ printable.java

(compiler)

↓

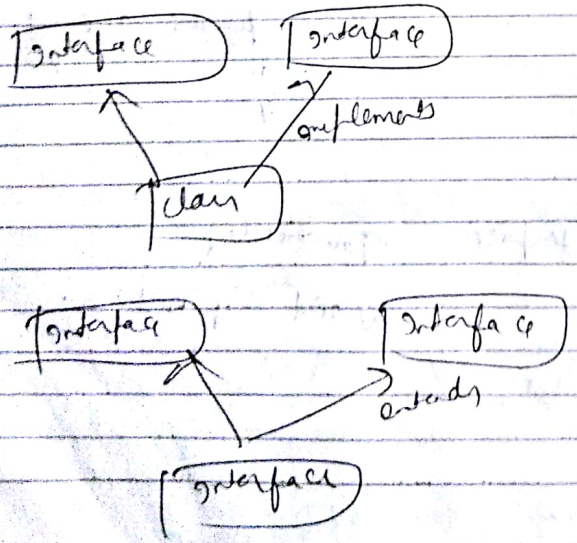
```
interface printable
{
    public static final int min=5;
    public abstract void print();
}

interface printable{
    void print();
}
```

63

```
class A6 implements printable{
    public void print()
    {
        SOP("hello");
    }
    psvm(String args[])
    {
        A6 obj = new A6();
        obj.print();
    }
}
```

multiple inheritance in java by interface



64

```
interface Printable{
    void print();
}
interface Showable{
    void show();
}
class A7 implements Printable,
                    Showable{
    public void print(){SOP("hello");}
    public void show(){SOP("welcome");}
    psvm(String args[])
    {
        A7 obj = new A7();
        obj.print();
        obj.show();
    }
}
```

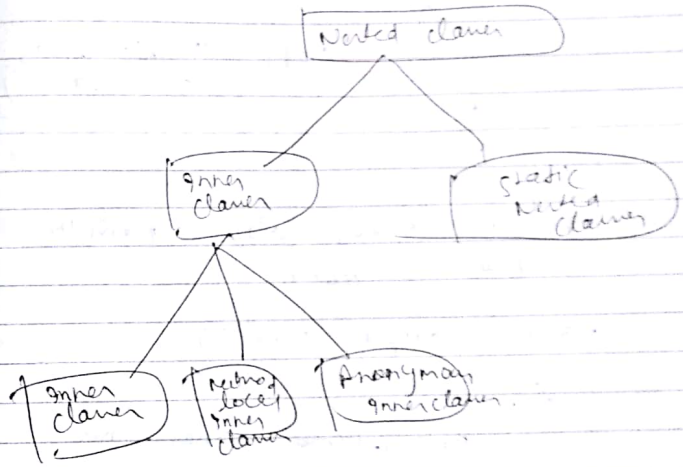
Abstract class	interface
1) It can have abstract and non abstract method	It can have only abstract method
2) It doesn't support multiple inheritance	It supports multiple inheritance
3) It can have final, non-final, static and non static variables	Interface can only have final & static variables
4) Abstract class can have static main & constructor	It can't provide
5) abstract keyword	interface keyword

Java packages

collection of classes & interfaces

access modifiers

Nested class



There are three types of innerclasses

1) innerclass
2) method-local inner classes
3) Anonymous innerclass

67

```
class Outer_Demo
{
    int num;
    // inner class
    private class Inner_Demo
    {
        public void print()
        {
            SOP("This is my inner class");
        }
    }
    // Accessing inner class from the method within
    void display_inner()
    {
        Inner_Demo inner = new Inner_Demo();
        inner.print();
    }
}
Public class My_Class {
    PSVM(String args[])
    {
```

68

```
// Instantiating the outer class
Outer_Demo outer = new Outer_Demo();
// Accessing the display_inner()
method
outer.display_inner();
}
}
```

Accessing the private members

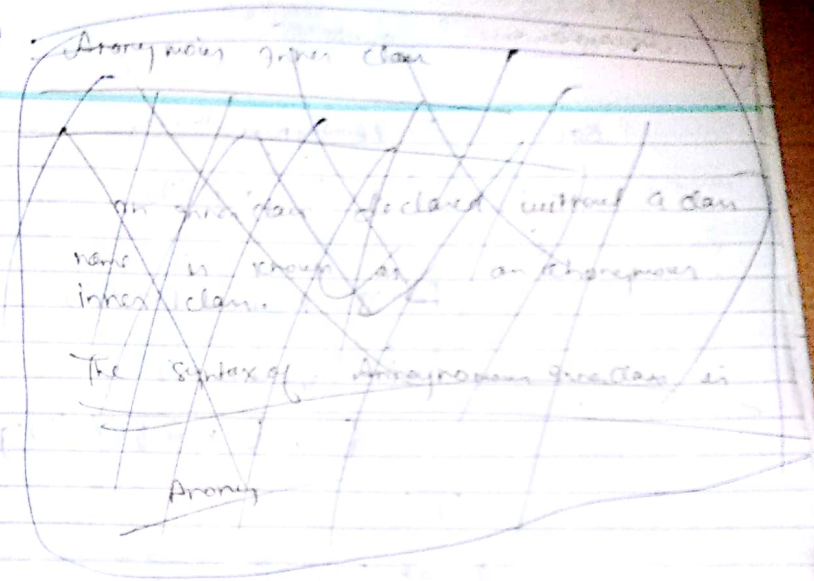
```
class Outer_Demo {
    // Private variable of the outer class
    private int num=175;
    // inner class
    public class Inner_Demo
    {
        public int getnum()
        {
            SOP("This is the getnum method of My inner class");
            return num;
        }
    }
}
```

```
public class My_Conf
PSVM(String args[])
{
// instantiating the
        outer class.
outer_Demo outer = new
                outer_Demo();
//instantiating the inner_
        class

outer_Demo.inner_Demo =
        outer.
        new Inner_Demo();
SOP( inner.getNum());
}
}
PSVM (String args[])
{
OuterClass outer = new
                OuterClass();
outer.my_method();
}
}
```

70

Anonymous inner class



A class that have no name is known as anonymous inner class in java. It should be used if you have to override method of class or interface.
It can be created by two ways

(1) class (may be abstract or concrete)

(2) Interface

example

```
abstract class Person
{
    abstract void eat();
}
```

71

```
class TestAnonymousInner{
    psvm(String args[])
    {
        Person p=new Person()
        {
            void eat()
            {SOP("eat fruits");}
        };
        p.eat();
    }
}
```

another example using interface

```
interface Eatable
{
    void eat();
}
class TestAnonymousInner1
{
    psvm(String args[])
    {
```

72

```
Eatable e = new Eatable()
{
    public void eat()
    { SOP("eat fruits");
    }
};
e.eat();
}
}
```

method - local inner class

```
public class outerclass
//instance method of the outer class
void my_method()
{
    int num = 23;
// method - local innerclass
class MethodInner_Demo{
    public void Print()
    { SOP("This is method innerclass "+num);
    }
} // end of innerclass
// calling the inner class
MethodInner_Demo inner = new MethodInner_Demo()
inner.Print()
}
psvm(String args[])
{
outerclass outer = new outerclass();
outer.my_method();
}
}
```

73

Exception handling

An exception is a problem that arises during the execution of a program.

two types of exception

1) Checked exception
2) unchecked exception

A checked exception is an exception that occurs at the compile time, these are called compile time exception. These exception cannot simply be ignored at the time of compilation, the programmer should take care of these exceptions.

```
import java.io.File;
import java.io.FileReader;
public class FileDemo
{
 psvm(String args[])
 {
  File file = new File("E://file.txt");
```

74

```
FileReader fr = new FileReader(file);
 }
}
```

- Error will come

unreported exception
FileNotFound exception
must be caught or declared to be thrown.

FileReader fr = new FileReader(file)

because read() and close() of FileReader throws IOException, we have to catch them.

Unchecked exception: It is an exception that occurs at the time of execution. These are called as Runtime exceptions.

Runtime exceptions are ignored at the time of compilation.

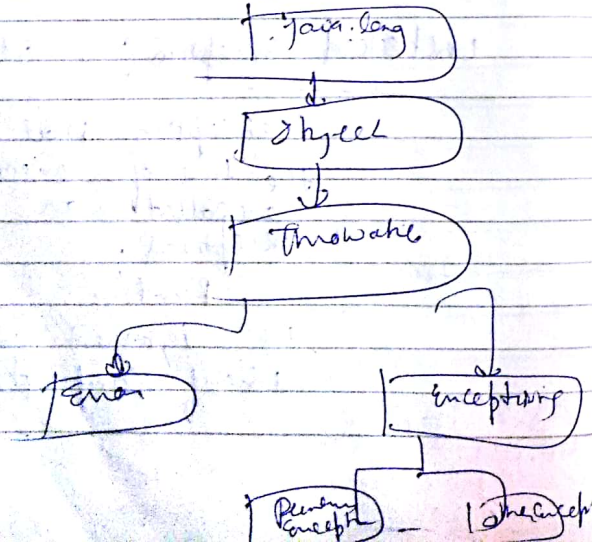
75

for eg:- ArrayIndexOutOfBounds

```
public class UncheckedDemo
{
    public static void main(String args[])
    {
        int num[] = {1,2,3};
        System.out.println(num[5]);
    }
}
```

ArrayIndexOutOfBounds exception will come

Exception Hierarchy



76

Built in exception

ArithmeticException
ArrayIndexOutOfBoundsException
NullPointerException
NumberFormatException

Checked exceptions

1) class NotPresentException
2) InterruptedException
3) IllegalArgumentException

Exception methods()

1) public String getMessage()
2) Public String toString()
returns the name of the class concatenated with the result of getMessage()
3) public void printStackTrace()

Catching exceptions

```
try
{
    // Protected code
}
catch(ExceptionName e1)
{
    // Catch block
}
```

77 for eg

```
import java.io.*;
public class exceptionf
{
    psvm (String args[])
    {
        try {
            int a[]= new int[2];
            SOP ("Access element at three" + a[3]);
        } catch (ArrayIndexOutOfBoundsException e)
        {
            SOP ("exception thrown" + e);
        }
        SOP ("out of the block");
    }
}
```

multiple catch blocks

```
try
{
}
catch (exceptiontype1 e1)
{ // Catch block
}
```

78

```
catch (exception type2 e2)
{
// Catch block
}
catch (exception type3 e3)
{
}
```

example

```
try
{
    file = new FileInputStream(f1.read);
    n = (byte) file.read();
} catch (IOException e)
{
    e.printStackTrace();
    return e;
} catch ( FileNotFoundException e1)
{
    e1.printStackTrace();
    return -1;
}
```

The Throws / throw keyword

If a method does not handle a checked exception the method must declare it using the throws keyword.

Example

```
import java.io.*
public class classname
{
    public void deposit(double amount)
        throws RemoteException
    {
        // method implementation
        throw new RemoteException();
    }
}
```

for eg.

```
public void withdraw(double amount)
        throws
RemoteException,
InsufficientFundsException.
```


The Finally block

```
try
{
    // Protected code
} catch(ExceptionType1 e1)
  {
    // Catch block
  } catch(ExceptionType2 e2)
  {
    // Catch block
  }
finally
{
    // the finally block always execute
}
```

Example

```
public class exception
{
PSVM( String args[])
{
    int a[] = new int [2];
try
{
    SOP ("access element
                three :"
                +a[3]);
} catch (ArrayIndexOutOfBoundsException
                e)
{
    SOP( "exception thrown :" +e);
}
finally{
    a[0]=6;
    SOP ("first element is" +
                a[0]);
    SOP( "the finally
        block executed");
}
}
}
```

82

User defined exceptions

If you are creating your own exception then it is known as user defined exception.

for eg.

```
class InvalidAgeException extends Exception
{
    InvalidAgeException(String s)
    {   super(s);
    }
}

class TestCustomException
{
  static void validate (int age)
                throws InvalidAgeException
  {
    if (age<18)
        throw new InvalidAgeException
            ("not valid");
    else
        SOP( "welcome to vote");
  }
```

83 PSVm(String args[])

{

try

{ validate(13);

} Catch (Exception m)

{ SOP ("Exception occured" + m);

SOP ("rest of the code");

}

}

Throw keyword

throw exception.

eg

Public class TestThrow {

static void validate (int age)

{ if (age<18)

throw new ArithmeticException ("not valid");

else SOP ("welcome to vote");

}

84 PSVm(String args[])

{

validate(13);

SOP ("rest of the code");

}

}

85

Java Applet Basics

An applet is a java program that runs on a web browser.

difference between applet and standalone Java application.

(1) An applet is a java class that extends the java.applet.Applet class.

(2) Applet are designed to embedded within an HTML page.

(3) A JVM is required to view an applet.

(4) Applet have strict security rules that are enforced by the web browser.

(5) A main() method is not invoked on an applet, and an applet class will not define main()

Lifecycle of an Applet

init → initialization will be done inside this method.

start → This method is automatically called after the browser calls init method.

stop → This method is automatically called when the user moves off the page on which the applet sits.

destroy → This method is only called when the browser shuts down normally.

paint → invoked immediately after the start() method and anytime the applet needs to repaint itself in the browser. This method paint is actually inherited from java.awt.

Helloworld Applet

```
import java.applet.*;
import java.awt.*;
public class Helloworldapplet
extends Applet
```

```
{
    public void paint(Graphics g)
    {
        g.drawString("Hello world", 25, 50);
    }
}
```

How to run applet file

- by html file
- by appletviewer tool

```
/*
<applet code="HelloworldApplet.class"
        width="300"
        height="300">
</applet>
*/
```

```
C:\> javac HelloworldApplet.java
C:\> appletviewer HelloworldApplet.java
```


example of Graphics in applet

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet {

    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g.drawString("welcome", 50, 50);
        g.drawLine(20, 30, 20, 300);
        g.drawRect(70, 100, 30, 30);
        g.fillRect(170, 100, 30, 30);
        g.drawOval(70, 200, 30, 30);
    }
}
```

myapplet.html

```
<html>
<body>
<applet code="GraphicsDemo.class"
        width="300"
        height="300">
</applet>
</body>
</html>
```

89 Parameter in Applet

Applet class provides getParameter method

Public String getParameter (String parameterName)

```
import java.applet.Applet;
import java.awt.Graphics;

Public class UseParam extends Applet{

    public void paint(Graphics g)
    {
        String str;
        str = getParameter("msg");
        g.drawString(str, 50, 50);
    }
}
```

90 myApplet.html

```
<html>
<body>
<applet code = "UseParam.class"
width = "300" height = "300">
<param name = "msg" value =
"welcome to applet">
</applet>
</body>
</html>
```