JAVA

1. What is the difference bet" C & JAVA.
2. Briefly explain about different features of object oriented Language.

3. JAVA & C

- The major difference is that JAVA is an object oriented language and C is a procedural language.
- Java doesn't include the C unique statements keywords sizeof & a typedef
- Java doesn't contain the datatypes struct & union.
- Java doesn't define the types modifiers keywords, auto, extern, register, signed & unsigned.
- Java doesn't support an explicit pointer type.
- Java doesn't have a preprocessor and :: , we can't used #define, #include & #ifdefstatements.
- Java requires that the functions with no arguments must be declared with empty parenthesis & not with void keywords as done in C.
- Java adds new operators such as instance of & >>>.
- Java adds many features for object oriented programming.

(2). Different features of object oriented Programming Language:-

1) Class

→ It is an abstract concept & simply a template (doesn't occupy memory)
→ A class contains data & methods to manipulate data.
→ class is simply a blueprint of a structural representation of attributes & actions.

2) Object

→ An object is the instance of class
→ It is a bundle of variables and related methods.
→ objects from the basic unit of object orientation with behaviour and identity.

3) Method →

→ class defines both attributes and behaviour.
→ Variable defines attributes and method represents behaviour
→ It is an entity through which an object interacts with external environment and whatever the object does can only be done through methods.

4) Message Passing:

It is the process by which the objects communicate with each other. It is obvious that communication can only be done through the methods.

5) Abstraction :→

- Abstraction means hiding the details. A programmer hides the complexity of the program from end user.
- End users have only methods or functions to interact with the data without knowing how manipulation is done.

6. Encapsulation →

- It means binding the code and data together.
- Code implies the methods or function that manipulate data. This technic protects the data from outside interference and behaves as protective wrapper.
- A user can interact with the data through the methods that are available in the specific object.

7. Inheritance →

- Inheritance is the mechanism to organize and structure software program. It is a technique by which an object acquires the features of another object along with its own feature.
- The object from which the property is acquired is known as parent object.
- The object which acquires the property is known as child object.

. Polymorphism :→

- It means one name forms. It also means one method performing many tasks
- This is a technique through which appropriate method invocation is performed by appropriate type specification.

6/8/19

Difference bet^n procedural language & object oriented language

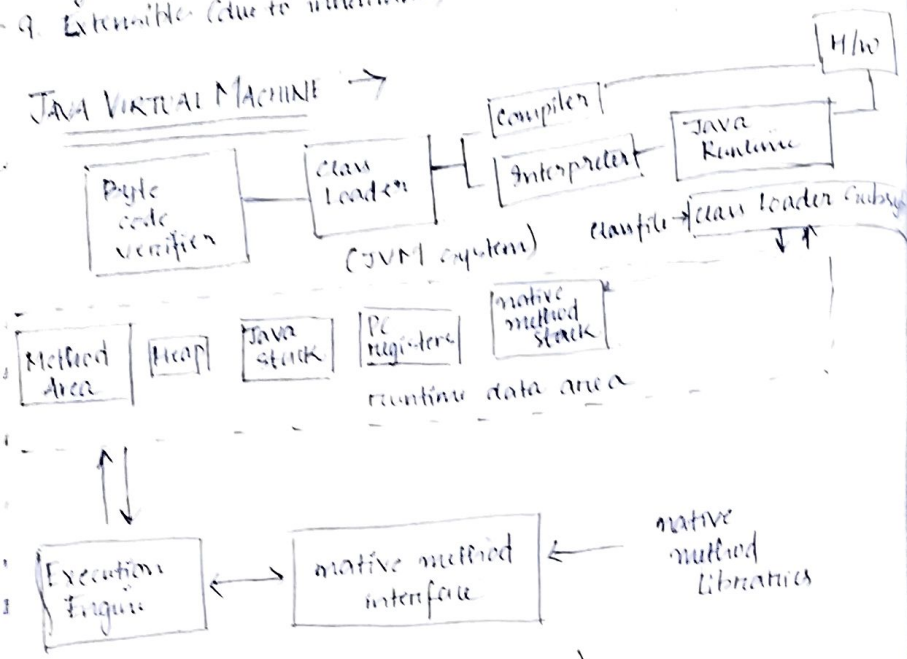
PROCEDURAL	Object oriented
• Focus is on algorithm	• Focus is on data.
• Programs are collection of functions.	• Program are collection of object.
• Global variables are shared by all functions.	• No concept of Global variables.
• Data moved from function to function freely.	• Functions that operate on data must belong to some class.
• Data is not hidden	• Data is hidden
• Topdown approach	• Bottom up approach
• Require less memory	• Require more memory
• No acces specifier	• 4 types of acess specifier are Public, private, protected & default.
• less ~~secu~~ secure.	• More secure.
• No concept of polymorphism.	• Polymorphism concept ~~lict~~ like method overLoading & method overriding are present.
• Explicit use of pointer is allowed.	• Programmer is not allowed to use pointer.
• Structure union concept in procedural language.	• structure, union concept absent.
• Preprocessor directive present.	• Packages are used in place of preprocessor directive.

2. Features of Java :-

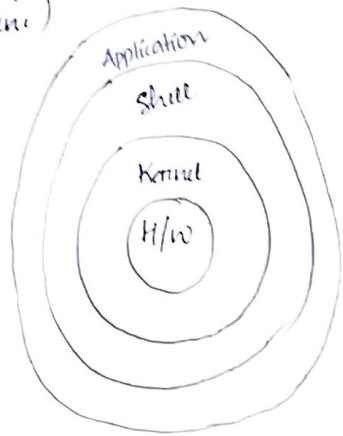
1. Compile and Interpreted.
2. Platform independent and portable (class file is portable)
3. Robust and Secure.
4. Distributed
object oriented

6 - Multithreaded.
7 - High performance
8 - Dynamic
9 - Extensible (due to inheritance)

JAVA VIRTUAL MACHINE →



(Java virtual Machine)



Kernel } operating system
Shell }

[10 marks]

Q) Briefly explain about JAVA VIRTUAL MACHINE [2 marks]

Q) What is JRE (JAVA Runtime environment)

A) JRE contains JVM and other class library required for program execution. The task carried in JRE are

i) loading of class file
ii) Verification of byte code.
iii) Interpretation of byte code.

Q. What is JIT? (Just in time compiler)

JIT compiler directly convert the byte code to machine code in place of interpreter. This process make program execution fast

Garbage collection in JAVA :→

JVM automatically reallocates memory from the variables which are not in use for that it uses the method.

System.gc()
Runtime.gc()

Q) How Garbage collection is done in JAVA [2 marks]
OR
which method are used for memory management.

Construction of JAVA program :→

Ex :-

```
import java.lang.*;
class X
{
public static void main (String a[])
 {
  System.out.println("CET");
   }
  }
```

System class println - method out - object.

for static method there is no need to create object whereas for non static method it is compulsion.

String is an class. (Every class should begin with capital letter)

Different types of Variables :—

3 types of variable :-

i) Local variable
ii) Instance "
iii) Static "

How to give input to program?
There are 3 ways we used to feed input.

- Command line Argument
- Scanner class (Java.util.*);
  ie, Java util package
- Readline() (java.io.*)

```
Public static void main (string a[])
{ int a = Integer.parseInt(a[0]);
  int b = Integer.parseInt(a[1]);
  int c = a+b;
  System.out.println("sum=" +c);
}
```

java CX.java
javaX 5 6

```
import java.util.*;
Scanner ob = new Scanner(System.in);
int a = ob.nextInt();
float b = ob.nextFloat();
double c = ob.nextDouble();
int a = ob.next();
```

memory allocation
InputStreamReader ob = new InputstreamReader(System.in);
BufferReader ob1 = new BufferedReader(ob).

String S = ob1.readline();

When readline method are used exception is thrown

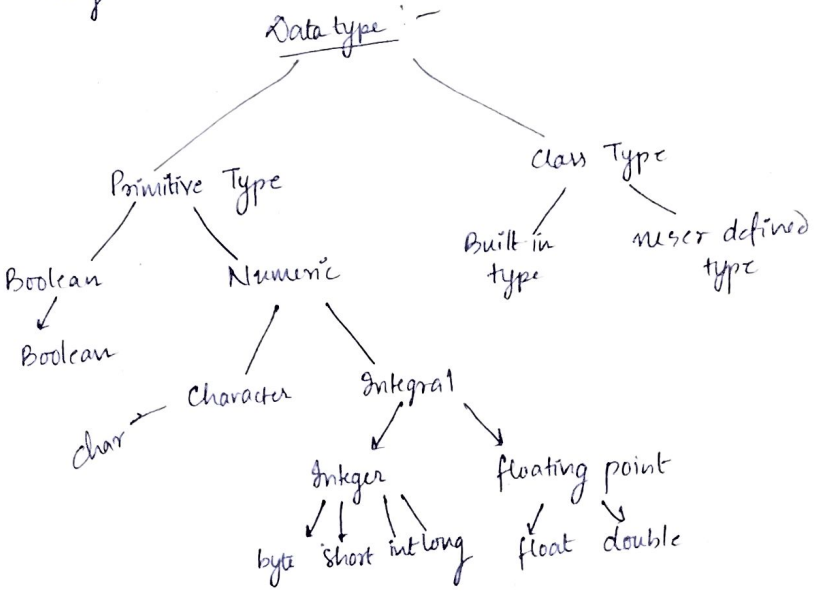
→ Exception is a situation if not solved causes problem in program execution
Exception and error are different.

void X() throws Exception.

Datatypes:—

Primitive ~~datatyp~~ datatypes

int
byte
short
char
float
double
long



Write a note on datatype? (5)

for Boolean default value false(0)
size 1 bit

for char size 2 byte
default value '\u000'

For byte size 1 byte
default value 0

For short size 2 byte
default value 0

For int size 4 byte
default value = 0

For long size 8 byte
default value = 0 L

For float size 4 byte
default value = 0.0f

For double size 8 byte
default value = 0.0d

For byte range -2^7 to $2^7 - 1$
" short range -2^{15} to $2^{15} - 1$
" int range -2^{31} to $2^{31} - 1$
" long range -2^{63} to $2^{63} - 1$

For float range
It is a simple precision 32 bit IEEE 754 floating format

For double
It is a double precision 64 bit IEEE 754 floating point.

TYPE CASTING

Implicit
Explicit
wrapper class

Q) What is typecasting ?

Q)¹ Write a program to display your personal information.

```
class A
{
Public static void main (String arg[ ])
{
System.out.println ("~~Regd no = 18AHOE204~~
Name = Nleha Parveen");
System.out.println ("Regno = 18AHOE204");
}
}
```

Q2) WAP that accepts integer from user and determine whether it is even or odd and display appropriate statements

```
import java.util.*;
class X
{
Public static void main (String arg[ ])
{
Scanner ob = new Scanner (System.in);
int a = ob.nextInt ();
if(a % 2 = = 0)
System.out.println (" It is an even number");
else
System.out.println (" It is an odd number");
}
}
```

Q3) WAP that adds all the even number present in a given range

```
import java.util.*;
class ABC
{
Public static void main (String arg[ ])
{
Scanner ob = new Scanner (System.in);
int a = ob.nextInt ();
int b = ob.nextInt ();
if (a % 2 = = 0)
while (a < b) {
System.out.println (a);
int a = a+2;
}
}
```

```
else
{
while (a<=b)
{
int a = a+1;
System.out.println(a);
}
}
}
}
```

Q) Give two examples of separators used in Java.

Ans→ ; - semicolon
— - space.

Q) Using 2 class concept, WAP to find factorial of a no.

```
import java.util.*;
class X
{
Scanner ob=new Scanner(System.in);
int n = ob.nextInt();
}
class Y
{
Public static void main (string args[]);
{
X object = new ~~object~~; X();
int f=1;
for (int i=1; i<= object.n; i+1)
{
f = f * i;
}
System.out.println("The factorial is "+f);
}
}
```

[5 marks]

20/9/19

Q) Write a note on operators used in JAVA.

0 → +ve
1 → -ve

→ operator preference [2 mks]
→ Unary operator highest priority.

Q). Find largest of 3 numbers using ternary operator.

Ans→
```
import java.util.*;
class A
{
Public static void main (String args[]);
{
Scanner obj = new Scanner (System.in);
int a = ob.nextInt();
int b = ob.nextInt();
int c = ob.nextInt();
int max = (a>b)?((a>c)?a:c):((b>c)?b:c);
System.out.println(max + " is the highest of all");
}
}
```

Q) WAP, to check whether a year is leap year or not.

var >> no-of-bits-to-be-shifted. (Sign is preserved)
var >>> no-of-bits to be shifted. (Sign is not preserved).

Ans →
```
import java.util.*;
class A
{
Public static void main(String args[])
{
Scanner ob = new Scanner (System.in);
```

```
int year = ob.nextInt();
if (year % 4 == 0)
 {
 if (year % 100 == 0 && 400 == 0)
 {
 System.out.println(year + " is a leap year");
 }
 else
 System.out.println(year + " is not a leap year");
 }
else
System.out.println("It is not a leap year");
}
}
```

Q) WAP that calculates ~~compound~~ simple interest on a principal.

```
import java.util.*;
class Math_Interest
{
Public static void main (String arg[]){
{
int p = Integer.parseInt(arg[0]);
int r = Integer.parseInt(arg[1]);
int t = Integer.parseInt(arg[2]);
float i = (p*r*t)/100;
System.out.println(Interest + "the simple intrest");
}
}
```

Q) WAP that find sum of digits of a number.

```
import java.util.*;
Class AS
Public static void main (String arg[]){
int n1 = Integer.parseInt(arg[0]);
int sum = 0; int r;
while (n1 != 0)
{
int r = n1 % 10;
int sum = sum + r;
n1 = n1/10;
}
System.out.println("The sum is" + sum);
}
}
```

OR

```
import java.util.*;
class S{
public static void main (String arg[]){
Scanner ob = new Scanner(System.in);
int n1 = ob.nextInt();
int sum = 0;
int r;
while (n1 != 0)
{
r = n1 % 10;
s = sum + r;
n1 = n1/10;
}
System.out.println("The sum is" + sum);
}
}
```

(7)
```
import java.util.*;
class Z
{
public static void main(String arg[]){
Scanner ob = new Scanner (System.in);
int p = ob.nextInt();
int r = ob.nextInt();
int t = ob.nextInt();
float i = (p*r*t)/100;
System.out.println(i + " is the simple intrest");
}
}
```

26/8/19

```
Switch (expn./var)
{
case val1:
    statement-1;
    break;

case val2:
    statement-2;
    break;
~~case val3:~~
    .
    .
    .
default:
    statement-n;
}
```

Q) Input an integer in the range of 1 to 7. Then check which is the day in the week to printed.

do-while

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

Q) 2 Find the sum of all odd integer present between 1 to n using do while loop.

Q). The difference bet" break and continue. [2 marks]
" " " while and do while.

Q) 3 Find the sum of all odd indexed digit of a number use continue statement.

Constructor :—

→ constructor is used for object intialize.
(Some memory space is allocated)

⇒ The syntax of constructor is

```
classname()
{
  //body
}
```

⇒ There is no returntype for constructor.

⇒ The parameter list maybe / may not be empty.

⇒ The parameter is present, parameterized constructor.

⇒ Not present, Non-parameterized constructor.

⇒ The access specifier can be of default & public but protected and private are not allowed.

Access specifier

i) Defaut (more secure)
ii) Private → (it is not visible outside class, so the object outside class cannot access)
iii) Protected
iv) Public
(after private)

Protected — Parent class ⇒ variable are visible to only immediate child class
↓
child class ⇒ flow is restricted to one level only
↓
ch class

Default (Then)

Visible to all the members of all classes i.e, present in the class.

→ visibility limited to one package.

Public (least secure)

It is visible outside the class.

⇒ Constructor cannot be used abstract final static synchronized

Final - value cannot be changed.

Static - All the static types memory is allocated at the time of compilation.

Synchronized - 2 entities both are working on same object.
→ Basically used in with methods

- In our program, we haven't define any constructor then JVM called default constructor at runtime.
- If you have declared parameterized constructor in your program but try to access default constructor error will occur.
- Copy of object can be created called as object cloning

Q4. WAP that checks whether a number is prime or not use constructor.

Q5 WAP that finds ~~error of~~ area of triangle, rectangle circle using the concept of constructor overloading.

Q). Diff. bet" constructor and method

Q1 import java.util.*;
class M {
public static void main (String args[]) {
Scanner ob = new Scanner (System.in);
int num = ob.nextInt();
switch (num)
{
case 1:
System.out.println ("Monday");
break;
case 2:
System.out.println ("Tuesday");
break;
case 3:
System.out.println ("Wednesday");
break;
case 4:
System.out.println ("Thursday");
break;
case 5:
System.out.println ("Friday");
break;
case 6:
System.out.println ("Saturday");
break;
case 7:
System.out.println ("Sunday");
break;
default:
System.out.println ("Invalid");
}
}

Q2. Constructor Overloading —

Inside one class we can write multiple constructor with different types of parameters

X()
{
body
}

X(int a, int b)
{
body
}

No of parameter types of

(2). import java.util.*;
class O {
public static void main (string arg[]) {
Scanner ob = new Scanner (System.in);
int n = ob.nextInt ();
int s=0;
if (n%2==0)
{
n = n-1;
}
do
{
s = s+n;
n = n-2;
}
while (n>=1);
System.out.println ("The sum is" +s);
}
}

Structure

Class X
{
// constructor
// define the logic
// constructor ends
p.v.s.m (string a[])
{
// create object pass value to
// constructor
}
}

3. class Y
{
public static void main (string args[])
{
Scanner ob = new Scanner (System.in);
int n = ob.nextInt();
int b[] = new Int[5];
int a, s=0;
int i=0, j=4;
while (n>0)
{
a = n%10;
b[i] = a;
i++
n = n/10;
}
for(i=0; i<5; i++)
{
x[j] = b[i];
j--;
}
for(j=0; j<5; j++) {
System.out.println (x[j]);
}
for(j=0; j<5; j++){
if (j%2==0) {
continue;
}
s = s + x[j];
}
System.out.println ("The sum is" +s);
}
}

4. import java.util.*;
class Prime {
int i, K = 1;
Prime (int n)
{
for (i=2; i<= n/2; i++)
{
if (n%2 == 0)
{
K=0;
break;
}
else {
K=1;
}
}
if (K==0)
System.out.println (n + " is not a prime number");
else
System.out.println (n + " is a prime number");
}
public static void main (String arg[]) {
Scanner ob= new Scanner (System.in);
Prime p = new Prime(x);
}
}

(5) import java.util.*;
class Area {
int l, w;
float b, h;
float r;
Area (int l, int w) {
int area = l * w;
System.out.println (area);
}
Area (float r) {
double area = 3.14 * r * r;
System.out.println (area);
}
Area (float b, float h)
{
double area = (double) 0.5 * b * h;
System.out.println (area);
}
public static void main (String args[])
{
Scanner ob = new Scanner (System.in);
int x = ob.nextInt();
int y = ob.nextInt();
float a = ob.nextFloat();
float b = ob.nextFloat();
float c = ob.nextFloat();
Area a1 = new Area (x, y);
Area a2 = new Area (c);
Area a3 = new Area (a, b);
}
}

RULES FOR STATIC MODIFIER →

- Static variables are declared outside the method and constructor
- If the static variables is not initialised, it takes the default value of datatype
- There is only one copy of static variable that exists for a particular class.
- Static variables can be accessed within non-static method and constructor directly
- Static variable can be accessed through class name and also using object name directly within a static method
- Static methods can be called without creating the object of corresponding class

8). WAP that contains static methods other than main method then call that method inside main without creating any object.

```
import java.util.*;
class Y
{
int i, k=1;
Prime (int n)

class Area Y
{
int r; static
double float r; double a;
Static void static area (float r) { double r) {
    a = 3.14 * r * r;
    System.out.println(a);
}
Public static void main (String arg[])
{
Y. area (3.5);
}
}
```

Method Overloading

WAP that explains the use of the concept of method overloading.

Ans →

```
import java.util.*;
class Area { Demo{
int i,b,s;
float r;
void area (int a, int b){
double a = i*b;
System.out.println(a);
}
void area (int s){
double a = s*s;
System.out.println(a);
}
void area (int r){
double a = 3.14 * r * r;
System.out.println(a);
}
public static void main (String args[]){
System.out.println("enter the dimension of rectangle, Square, circle");
Area Scanner
Area d = new Area();
d.area (14.5);
d.area (3, 2);
d.area (4);
}
}
```

9). Passing object to method and returning object

⇒ Passing object to method as parameter and the method returns object as output.

Q)1 WAP to find the larger among 2 numbers using the concept of method accepting object & returning object.

```
import java.util.*;
class X
{
int a;
int b; res;
}
class Y
{
X ob = new X(); Scanner sc = new Scanner(System.in);
    //assign the values ob.a and ob.b;
~~X res = larger(ob);~~ ob.a = sc.nextInt(); ob.b = sc.nextInt();
  X res = larger(ob);
  System.out.println(res.c)
}
X larger(X ob1)
{
 X ob2 = new X();
```

Class

```
import java.util.*;
 class X
{
int a,b,res;
public static void main(String a[])
{
 X ob = new X();
 Scanner sc = new Scanner(System.in);
 ob.a = sc.nextInt();
 ob.b = sc.nextInt();
 ob.res = larger(ob);
 System.out.println("Larger = " + ob.res);
 }
 X larger(X ob1)
 {
 X m = new X();
 if (ob1.a > ob1.b)
  m.res = ob1.a
 else
  m.res = ob1.b
  return m;
  }
 }
}
```

29/8/19.

Q)2 Find the fibonacci series upto n terms using only passing by object method.

```
import java.util.*;
 Class X
{
int a,b,c,n;
public static void main(strig args[]){
  X ob = new X();
 Scanner sc = new Scanner(System.in).
 ob.n = sc.nextInt();
```

```
= Series( Fibonacci(ob);
    }
    X Fibonacci(X ob)
    {
      X m = new X()
    int a=0, b=1, s;
    for(i=0; i<ob.1.m; i++)
    {
    s=a+b;
      a=b;
      b=s;
    System.out.println(s);
      }
    }
```

Q3. WAP that finds factorial of a number. Use the following concepts

i) recursion

ii) Method accepting as argument and returning object to the calling method.

i)
```
import java.util.*;
  class X {
    int a;
Public static void main (String args[]) {
Scanner ob = new Scanner(System.in);
int n = ob.nextInt();
  int fact = factorial(n);
System.out.println("The factorial is "+ fact);
}
int static factorial (int n){
if (n==0)
{
return 1;
else
return n * factorial (n-1);
}
}
```

ii)
```
import java.util.*;
class X {
 int n; res;
public static void main (String args[]) {
  X ob = new X();
Scanner sc = new Scanner (System.in);
  ob.n = sc.nextInt();
  ob.res = Factorial(ob);
  System.out.println
```

// Factorial of a no. using

i) recursion

ii) Method accepting object and returning object to calling method

```
i) public class main
{
int n, fact;
public static void main (String a[])
{
Main ob = new Main();
ob.n = 5;
ob.fact = 1;
Main ob1 = ob.factorial (ob);
System.out.println (ob1.fact);
}
Main factorial (Main ob2) {
if (ob2.n == 0 || ob2.n == 1)
return ob2;
else
{ ob2.fact = ob2.fact * ob2.n;
ob2.n = ob2.n - 1;
return factorial (ob2);
}
}
}
```

(10) Use of static and non-static block

Syntax for static block:

```
Static
{
//body
}
```

Syntax for non static block

```
{
//body
}
```

=> Whether static or non-static all the content should be written outside method.

=> Before object creation, the body of static block is executed.

For accessing non-static block, object ~~are~~ creation is required. [default object]

=> Syntax - exit (int).

1. WAP that shows the use of static block and non-static block.

```
class X
{
static
{
System.out.println("Inside a static block");
}
{
System.out.println("This is executed due to the object");
```

```
}
public static void main (String a[])
{
X ob = new X();
}
}
```

ARRAY :⇒

Syntax

```
datatype arrayname = new datatype [size];
                    // declaration if elements are of
                       primitive datatypes.
```

Array contain object

Syntax

```
Classtype arrayname = new classtype [size];
                    // declaration if elements
                       are objects.
```

Q) 2. Implement the linear search algorithm using array.

```
import java.util.*;
class Array {
int A[] = new int[5]; int Key; int[] A = new int[10];
                              int key;
A[0] = 10;
public static void main (String a[])
{
Scanner ob = new Scanner (System.in);
for (int i=0 ; i<10; i++)
{
int A[i] = ob.nextInt();
}
System.out.println ("Enter the array elements");
int key = ob.nextInt();
for (int i=0; i<10; i++)
{
for (int j = i+1; j<10; j++)
{
if (A[i] = A[j])
{
flag = 0;
}
else
flag = 1;
}
}
if (flag == 0)
system.out.println ("the element is found");
else
system.out.println ("The element is not found");
}
}
```

(3) Rewrite the above program array of object concept

```
Class X
{
int n;
public static void main()
{
X[] A = new X[5];
Scanner ob = new Scanner(System.in)
for(int i=0; i<5; i++)
{
X A[i] = new ~~X[5]~~, X();
A[i].n = ~~5~~ ob.nextInt();
}
int key = ob.nextInt();
for(int i=0; i<5; i++)
{
if(key == A[i].n)
{
flag=0;
}
if(flag==0)
System.out.println("FOUND");
else
System.out.println("NOT FOUND");
}
}
```

Q(2)

```
import java.util.*;                    //Linear search
class Linear
{
public static void main(String a[])
{
int[] A = new int[10];
int item;
Scanner SC = new Scanner(System.in);
item = SC.nextInt();
for(int i=0; i<10; i++)
    A[i] = SC.nextInt();
int res = linearsearch(A, item);
   if(res != -1)
System.out.println(Item + "found at index" + res);
Static int linearsearch(int[] A, int item)
{
for(int i=0; i<10; i++)
{
if A[i] == item
    return(i);
}
return(-1);
}
```

Multidimensional Array

datatype[][] arrayname = new datatype[sz1][sz2]

Q). WAP to perform multiplication of 2 matrices.

PACKAGE

default → java.lang.*

2 types of package :-

① built in ② User defined type

```
Package pkgname; // Syntax for package creation
Class 1
{
// body
}
class 2
{
// body
}
```

We are saving the above program
class name should be the name of file.
foldername is same as package name.

```
Package Y;
Public class X
{
// body void display(){
                    //
                    }
}
import Y.*;
class Z
{
public static void main(String args[])
{
X ob= new X();
ob.display;
```

Y – X.java
e – X.java

* Documents – Y – X.java
Z.java

The file and package must be under same folder.
CLASSPATH.

WAP that creates a package which contain a class. The class contains variable, methods and constructors. Design the program for the calculation of gross salary of an employee (Basic, P.A, HR).

```
package Y;
{
public class X
{
x static double total;
    double basic, HR, PA;
    X (double b, double h, double p){
     basic=b;
     HR=h;
     PA=p;
    }
    public double totalsalary(){
double total= basic + HR * basic + PA * basic;
     return total;
     }
    }
    import java.util.*;
    import Y.*;
     class Z
     {
```

```
public static void main (String args[])
{
Scanner sc = new Scanner (System.in);
double b = sc.nextDouble();
double p = sc.nextDouble();
double h = sc.nextDouble();

X ob = new X(b,p,h);
double total = ob.totalSalary();
System.out.println( total + " is the gross salary");
}
}
```

13/9/19

STRING CLASS

* It is a final class that means it can't be inherited and you can't create child class.
* The objects of string class are immutable (they can't be changed).

New objects are created when you attempt to change.

String s1 = "CET"
↑ ref (stack)
object (heap)

String s1 = "CE"

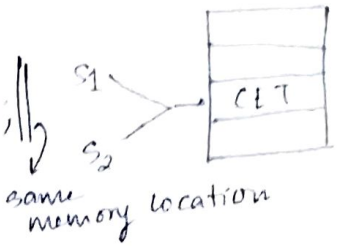
* Each reference in string has a hashcode value that identifies object.

ob.hashcode()
↓
It should be of string type.

* The constructors of string class:

String()
String (char ch[])
String (char[] c)
int start, int length → Array

String s1 = "CET";
String s2 = new String("CET");



same memory location

String s1 = "CET";
String s2 = "ET"; → different memory location.

Methods of STRING Class

(1) length()
String s1 = "CET";

(2) s1.length();
char charAt(int index)

(3) boolean equals ignore case (String)

(4) boolean equals (String)
s1.equal(s2)

If s1.equal(s2) → same memory
If s1 == s2 → different memory

(5) int compare To (String)

(6) int index of (char ch)

(7) String concat (String)

(8) string tolower case()

(9) string to upper case()

(10) char[] to char Array()

(11) string trim()

Q1. WAP to check whether a string is palindrome or not.

Q2. WAP to reverse the given string.

```
import java.util.*;
import java.lang.*;
class MYSTRING
{
public static void main (String args[]){

int length = 5;
String c1 = new string ("MADAM");
char [ ]c = c1.toCharArray();
for (int i=length-1; i>=0; i--)
 {
System.out.println (c[i]);
 }
}
}
```

String Buffer class

- It is similar to string class but the prime difference is object are mutable.
- It belongs to Java.lang package.
- class is of public & final type.

Constructors

Default - StringBuffer( )
StringBuffer(String)
StringBuffer (int capacity)

Methods

```
int length();
int capacity();
void setLength (int);
char charAt (int);
void setcharAt (int index, char ch);
string Buffer append (object);
string Buffer append (string);
string Buffer append (char);
void getchar (int start, int end, char c[], int index);
string Buffer delete (int index, int length);
string Buffer replace (int index, int length, string s);
String Buffer insert (int index, string s);
String Buffer reverse();
string to string( );
```

Q). WAP that replaces every odd indexed character of a string with a star (*). (Use string buffer class)

```
import java.util.*;
import java.lang.*;
class MYSTRINGBUFFER

{
public static void main (string args[])

{
int length = 5;
StringBuffer c1 = new StringBuffer ("MADAM");
string s = c1.toString();
char() c = s.toCharArray();
for (int i=0; i<5; i++) {
 if (i%2!=0) {
  c[i] = '*';
 }
}
for (int i=0; i<5; i++) {
 system.out.println (c[i]);
  }
 }
}
```

String tokenizer class

=> It facilitates to convert a string into a set of of tokens with the help of delimiter.

Different constructor of the class

ex :- string Tokenizer ("CET");

string Tokenizer (string)
string Tokenizer (string, string)

string Tokenizer (string, string, boolean)

If boolean is true, both strings delimeter will get.

=> If the 3rd value is true, the delimeter is returns. If it is false then it is same as second one.

Methods of String Tokenizer class

=> int countTokens()
=> boolean hasMoreTokens()

"C b/E b/Tb"

String → class

nextToken() → Throws NoSuchElement exception

String nextToken(string newdelimiter)
String nextElement()

STRM

Q) WAP that shows the use of string tokenizer class, constructor and methods.

```
package - java.lang (default package)
import java.lang.*;
class Man
{
public static void main (string arg[])
{
StringTokenizer s1 = new string Tokenizer ("CET");
StringTokenizer s2 = new string Tokenizer("CET COLLEGE");
string s3 = s2.nextToken(); sopl(s3);
int n = s2.countToken();
sopl(n);
}
}
```

Wrapper class :-

=) It facilitates conversion of primitive type data to object type & vice versa.

=) It belongs to java.lang packages.

Integer
Float
Double
Boolean
long
Charader
Short
Byte

Constructors and Methods for classes :-

constructors
Boolean (boolean)
Boolean (string)

Methods

parse Boolean (string)
boolean Value ( ) ← return boolean object
to string (boolean)
Value of (string)
Value of (boolean)

Byte class

constructors

Byte (byte)
Byte (string)

Methods

parse Byte (string)
Value of (string)
Value of (Byte)
to string (byte)

string → byte
string → Byte
byte → Byte
byte → string

Short class

constructor

③ Example of (conversion of p... of

byte b = 1;
Byte ob = new Byte (b);
System.out.println (ob);

Dt-17/9/19

(II) Example of conversion of classtype to primitive type.

Boolean b1 = new Boolean ("yes");
boolean b = b1;
System.out.println (b);

What is autoboxing and unboxing ? [2marks]

WAP that explain the concept of autoboxing and unboxing

Exception handling :- ⇒

Unwanted situation that occurs during program execution and disturbs the normal flow of execution.
Basically, exception arises, where our code is unable to handle some situation.
Ex - we know that some number is divided by 0, result is infinity but our system doesn't know.

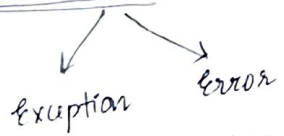
⇒ The error is also a ~~creatio~~ situation that causes problem in program execution. The problem is in java runtime environment.

⇒ exceptions can be handled by JVM (except^n handling inbuilt in JVM) as well as the programmer.

* There are 2 types of except^n. -
⇒ (i) checked exception – compile time exception
(ii) Unchecked exception – Runtime exception.

Q). Differentiate bet^n checked and unchecked except^n.

=> Throwable class → parent class → object class is super class



Object is the parent class of throwable class.

=> Compile time exception is handled by compiler. These are the exception which are caused by unexpected condition outside the control of code.

Ex (i) Class Not Found Exception.
(ii) NoSuchMethod Exception
(iii) NoSuchField Exception
(iv) CloneNotSupported Exception.
(v) IOException.

The checked exceptions are recoverable.

```
P.V.S.M (String args[]) throws IOException
  {
    // body
  }
```

Unchecked Exception :—

These are the exception that are handled by runtime. This extends RuntimeExceptionClass.

This is generated when there is error in code.

~~eg. NULL POINTER~~

eg. NullPointerException.

ArithmeticException
NumberFormatException.
ArrayIndexOutofBoundException.
StringIndexOutofBoundException.

1 ERROR :=>

Different types of error :—

1) StackOverFlowError
2) VirtualMachineError
3) OutofMemoryException

Exception Handling Mechanism :—

```
try
{
 // body
}
Catch (Exception ob)   (Exception → parent class)
 {
 // body
 }
```

=> Each try block can be followed by multiple catch block but the argument of catch block should be different.

```
Catch (NumberFormatException ob1)
 { //body
 }
```

Q). What is the difference betn final, finally, finalize().
- final → keyword making value constant
- finally → keyword in exceptn handling mechanism
- finalize() → garbage collection

```
finally
{
//body
}
```

→ Finally gives message about a particular situation

basing on which the programmer can modify the code.
→ Multiple catch can be used but one finally.

Throw Throws

// Unboxing and Autoboxing.

```
import java.io.*;
 class Auto{
public static void main (String args[]){
byte b=1;
Byte ob= new Byte(b);
System.out.println("value of ob" + ob);
Boolean c1 = new Boolean ("YES");
boolean c = c1;
System.out.println(c1);
}
}
```

value of ob1
false

Q). import java.util.*;
public class CHI
{
public static void main (String args[]){
StringTokenizer st1 = new StringTokenizer ("hello Readers, welcome to cet", " ");
while (st1.hasMoreTokens())
System.out.println(st1.nextToken());
StringTokenizer st2 = new StringTokenizer ("JAVA:CODE: String", ":");
while (st2.hasMoreTokens())
System.out.println(st2.nextToken());
StringTokenizer st3 = new StringTokenizer ("JAVA CODE STRING", ":", true);
while(st3.hasMoreTokens())
System.out.println(st3.nextToken());
}
}

```
package Y;
public class X
{
double basic, HR, PA;
public X(double b, double h, double p){
basic=b;
HR=h;
PA=p;
}
public double totalsalary(){
double total = basic + HR * basic + PA * basic;
return total;
}
}
```

folder
→ Y
→ X
→ class

```
import java.util.*;
import Y.*;
class Z
{
public static void main(String args[])
{
Scanner Sc = new Scanner(System.in);
double b = Sc.nextDouble();
double p = Sc.nextDouble();
double h = Sc.nextDouble();
X ob = new X(b,p,h);
double total = ob.totalsalary();
System.out.println(total + "is the gross salary");
}
}
```

① WAP that uses one try block 2 catch block and one finally block

```
import java.util.*;
class X
{
public static void main (String args[])
{ try{
int a[]= new int[5];
int a[10]= 5;
}
catch ( ArithmeticException e) ~~{ e.getMessage(); e)~~
{
System.out.println (e.getMessage())
}
catch ( ArrayIndexOutOfBoundException e) ~~e.getMessage()~~
{
System.out.println (e.getMessage());
}
finally
{
System.out.println ("Rest of the code is executed");
}
}
}
```

② WAP that shows the use of
1) NosuchMethodException
2) ClassNotFoundException
3) NumberFormatException
4) StringIndexOutOfBoundException
5) IllegalThreadStateException
6) ArrayStoreException

② Ans →

```
import java.util.*;
class ExceptionHandling{
public static void main (String args[])
{
try{
try{ ExceptionHandling ob= new ExceptionHandling();
        ob.display(); }
catch (NosuchMethodException e) {
System.out.println( e.getMessage()); }
try{ A obj= new A();
  catch ( classNotFoundException e) {
System.out.println( e.getMessage()); }
try{
  String s= "xyz";
    int i= Integer.parseInt (s); }
catch ( NumberFormatException e) {
System.out.println( e.getMessage()); }
try {
      String s1= "CAT";
        int l= s1.length;
        for (int i=0; i<=l; i++)
      {
      char ch= s1.charAt(i);
      }}
catch (StringIndexOutBoundException e){
System.out.println ( e.getMessage());
}
try {
    int [ ] a= new int [5];
      a[1]= 2;
}
```

```
catch(ArrayStoreException e){
System.out.println(e.getMessage());
}

}
catch(Exception e){
System.out.println(e.getMessage());
}
}
}
```

29/09/19

Userdefined exception:-

Exception is the class that is to be extended.

```
class X extends Exception
{
```

1] Create the userdefined exception using exception class and then throw the exception using exceptional handling keywords.

Q). Define an exception called NoMatchException that is thrown when an entered string is not equal to India.

A). WAP that throws this exception.

```
import java.io.*;
class J extends Exception
{
J(String msg)
{super(msg);} → constructor of the superclass is called.
}
class T
{
public static void main(String a[])
{
try
{
System.out.println("Enter a string");
InputStreamReader ob=new InputStreamReader(System.in);
BufferedReader obj =new BufferedReader(ob);
String s= obj.readline()
if(s!="India")
{throw new J("String Mismatch");}
}catch(Job2){
System.out.println("Exception occurred");
}}}
```

In place of J write NoMatchException

Q). What is the difference bet^n super and super().

Here, the superclass is the exception class

Q). Design an userdefine exception which is thrown when it is tried to print the value of class variable which is not converted to instance vasiable.

```
import java.util.*;
import java.io.*;

class No
```

```
public class Main1 {
public static void main (string args[]) {
    int a[] = {20,20,40};
     int n1 = 15, n2 = 10;
     int result = 10;

try {
  result = n1/n2;
  System.out.println("The result is "+ result);
     for (int i = 5; i >= 0; i--) {
System.out.println("the value of array is "+a[i]);
  }
  } catch (ArrayIndexOutofBoundException e) {
       S.O.P("array is out of bounds" +e);
    } catch (ArithmaticExuption e) {
        S.O.P("cant divide by zero"+e);
    }
  finally {
   S.O.P("rest of the code is executed");
    }
  }
}
```

1) NoSuchMethodException.

2. classNotFoundException.

```
class Rapoo {
}
class torture {
} class MyClass {
public static void main (String [] args)
    { object o = class.forName (args[0]).newInstance();
      System.out.println ("Class created for" + o.getClass().
                                                  getName());
    }
}
```

3. NumberFormatException.

```
class NumberFormat_Demo{
public static void main(String args[])
    {
     try {
        int num = Integer.parseInt ("Kill");
        System.out.println(num);
        }
        catch (NumberFormatException e){
          System.out.println ("Number format exception");
        }
    }
}
```

4) class StringIndexOutofBound_Demo{
public static void main (String args[])

```
    try{
        String a = "This is like chipping";
         char c = a.charAt (24);
         System.out.println (c);
    }
    catch(s:                    e ) {
        S.O.P("                  ");
    }
}
}
```

4). ArrayStoreException

```
public class GFG {
    public static void main (String args[])
    {
     try {
        Object a[] = new Double [2];
           a[0] = 4;
     }
    catch (ArrayStoreException e ) {
     System.out.println("ArrayStoreException found:" +e
}
}
```

```
import java.io.IOException;
class MyException extends Exception{
    MyException (String a)
    {super(a); }

  public class Main
{
public static void main (String args[]){
  try{
      String a= args[0];
      if ( a.equals ("Indians")){
System.out.println(" you have enter "+a);
    }
    else
    {
      throw new MyException (" Not Indians");
      }

    } catch(MyException e){
    System.out.println(" you have entered "+e.getMessage();
      }

    }

}
```

30/09/19

Inheritance in Java

=> The objective is code reusability.

=> In the process of inheritance, parent class property is inherited to child class.
=> Parent class is also known as super class.
=> Child class " " " desired class
=> Multiple inherited property possible in a single class.
=> Private variables can access through public methods.

Types of Inheritance

i) single Inheritance
ii) Multilevel
iii) Hierarchical
iv) Hybrid
v) Multiple Inheritance is absent in Java.

Why Multiple Inheritance is absent in Java?
In Java, we can create child classes from parent class in the properties of child class(or) if the properties of one or more than one child class is inherited then their later class (or) derived class and this create ambiguity.
is a possibility of having classes in same name in
In order to avoid this problem multiple inheritance is absent in java. But the concept can be used by using interface.

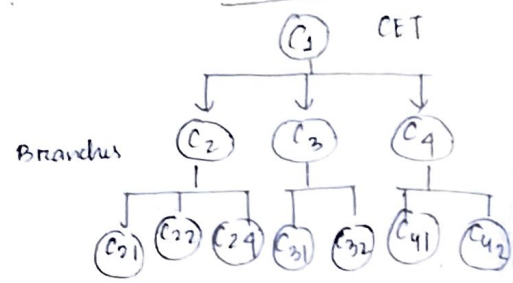
Syntax:-

```
class
{
  // body
}
class child extends parent
 { //body
}
```

→ extends → Keyword
Purpose for creation of child class from parent class.

Multinomial

Hierarchial



Construction and Inheritance

```
class X
{
 X()
 {
 //body
 }}
class Y extends
{
 Y()
 {  super(); // super() can be with or without argument.
 //body
 }
}
```

```
public class Z
{
public static void main(String[])
 {
 Yobj = new Y();
 }
 }
```

Super → Method overload/overloading generally not perform in inheritance but method overriding is possible.

super() should be the first line of the construction body.

→ Explicit call (super) is made to execute properties of super class.

In class, in same name variable exists in child class & parent class.

obj.a = 5; class Y

super.a = 5;

class Y has 2 'a'. But the local variable has highest precedence.

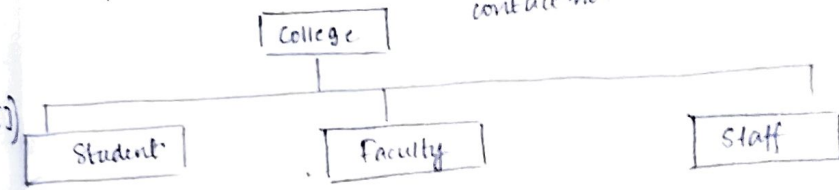
Q). What is the difference between this() and super().

this.a → correspond to current object (object initialisation).

this() → It points to the constructor.

WAP for the following :-

College Name :-
name
Address
contact no.



1. Name
2. Registration number :- Student Regd.no
   Faculty Regd.no
3. Department — Student
   \ Faculty

   Pay scale — Faculty
   \ staff

Input()
display().

```
import java.util.*;
class College
{
College()
{
String CollegeName = "CET";
String Address = "PBSR";
long int Contact no. = "9437522xxxx";
System.out.println("CET" + CollegeName + "\n");
S.O.P(Address + "\n");
S.O.P(contact no. + "\n");
}}
class Student extends College {
```

```
Super();
Student()
  {
int Student Redg.no = "1501106XXX";
char Department = "IT";
String S.o.p(Redg.no + " ");
       S.o.p("Department + " ");
       }
       }
   Class Faculty extends college
     {
   Super();
   Faculty()
  {
int pay scale = "5457";
char Department = "IT";
String S.o.p( payscale + " ");
       S.o.p( Department + " ");
       }
       }
      Class Staff extends college
        {
        super();
        Staff()
        {
          int payscale = "7948";
          S.op.( payscale + " ");
          }
          }
    public class
      {
      public static void main (String[] args)
      {
      Student s = new Student();
      Faculty f = new Faculty();
      Staff st = new staff();
      }
      }
```

27/09/19

Method Overriding

When both child and parent class contain methods to methods with same signature then after inheritance properties of parent class to child class

Child class has two methods with same signature

If the method signature is different then using the child class object parent class method can be called but the case is different when the signature is same. In this case the child class method overrides parent class method. As a result child class method is executed. In order to execute parent class inheritance method we may follow the foll. steps —

[i] Create parent class object to call parent class method.

[ii] You call the parent class method using super keyword inside child class method

[iii] All this happened at the time of compilation.

The soln to method overriding is dynamic method dispatch

A/q to this method at runtime it is decided that which method will be executed whether parent class or child class method.

If you want to execute parent class, just create the reference of child class and assign parent class constructor.

Reference is not a object. It stores address.

Signature same but method body different
When both the methods are executed
Dynamic method dispatch is also known as Runtime polymorphism
ie Method overriding

Method Overriding

→ Compile time polymorphism
i.e, method overloading

```
import java.util.*;
class A
{
int i,j;
A(int a, int b)
{ i=a ; j=b; }
void show( )
{
System.out.println(" i and j " +i+ " " +j); }
}
class B extends A
{
 int k;
B(int a, int b, int c)
 { super(a,b);
    k=c;
 }
void show( )
 {
  System.out.println("K= "+K); }
  }

Class Main
{
 public static void main (String args[ ])
 {
 int a,b,c;
 Scanner sc = new Scanner (System.in);
    a = sc.nextInt();
    b = sc.nextInt();
    c = sc.nextInt();
B ob = new B(a,b,c)
  ob.show( )
}
}
```

```
A ref;
ref = new A(a,b);
ref.show( );
ref = new B(a,b,c);
ref.show( );
}
}
```

A obj = obj is a reference of A.
obj = ob (is possible)
obj is pointed to child class.
A ob2 = new A( )
B ob1 = ob2

ob=ob2 | Due to inheritance property it will be accessible

ob.show( ).

Q). WAP that contains one parent class and one child class. There is a variable name A which is present in both parent class and child class print the value of 'a' in both class.

(super class
super keyword