16/1/23

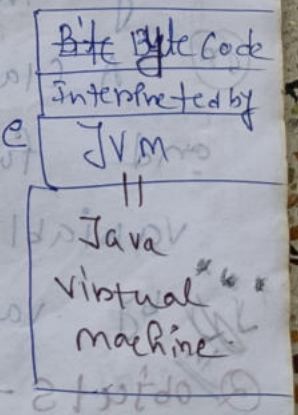
Object oriented Programming

- class is a concept (Not taking any space)
  structure is a collection of different data types
  function is associated with structure.
  class is also a kind of structure.
- Objects are the real world entites.
  (It takes memory space)
- Abstraction - we are hiding the unnecessary details. This is a concept. Those details which are not needed to us/computer are removed by abstraction.

17/1/23

- Features of Java (write one time & run many times)

1) Compile & interpreated -
2) Platform independent & Portable
3) Oject oriented
4) Robust & secure
5) Distributed
6) multithreaded



After writing code byte code generated
Robust = It is not susceptable of any kind of problems.
Threading = execution of Programm/Process
In Java threading is supported, not inc.

• Applet = Small programmes. (Eg- Dianosor game)
Applets are downloded from the servers.
they are stored in browser.

① Class - In object-oriented programming,
a class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behaviors (member functions or methods).
The user-defined objects are created using the class keyword.

② A class is a collection of variables and functions working with these variables. variables are defined by var and functions by function. [In Java function = method]

② Objects - object is an instance of a class. It is a real word entity. It is a physical entity.

Class is a blue print from which objects are created.

③ Abstraction - It is the process of hiding the internal details of an application from the outer world.
[Abstraction is used to describe things in simple terms. It's used to create a boundary between the application and the client programs.]

④ Features of Java -

i) Simple - It is very easy to learn, and its syntax is simple clean and easy to understand. According to sun microsystem, Java language is a simple programming language.

ii) object-oriented - It is an object oriented programming language. Everything in Java is an object. object oriented means we organize our software as a combination of different types of objects that incorporate both data and behaviours.

iii) secured - Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because - • No explicit pointer
• Java prog. run inside a

Virtual machine sandbox.
- classloader
- Bytecode verifier
- security manager

iv) Robust - the eng. meaning of Robust is strong. Java is robust because - ① It uses strong memory management.

② There is a lack of pointers that avoids security problems.

③ It provides automatic garbage collection which runs on the Java virtual machine (JVM) to get rid of objects which are not being used by a Java application anymore.

④ There are exception handling and the type checking mechanism in Java.

vi) Architecture - neutral - Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

vii) portable - It is portable as it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

viii) High-performance - It is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.

ix) Distributed - It is distributed as it facilitates users to create distributed applications in Java.

x) multi-threaded - A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.

xi) Dynamic - Java is a dynamic language. It supports the dynamic loading of classes.

⑤ *** Applet - It is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantages - ① secured
② It works at client side so less response time.
③ It can be executed by browsers running under many platforms including Linux, windows, mac Os etc.
Disadvantages - ① plugin is required at client browser to execute applet.

30/1/23

① difference between Java & c.
② why Pointers isn't use in Java.

Union
float int

Java Prog

```
class test
{
    public static void main(String args[])
    {
        System.out.println("Hiah");
    }
}
```

Keyword → class, static
Test = class
public = Access specified
↓
this method can be accessed by anywhere.

Structure is A and we are calling data & the function is A.D. D is a member of A

Package - package is similar to folder.
If it is private the method will be accessable by the methods in between the class. only

- class variable = same with static variable.
- Command line argument (concept of c)
- String = Object
- Test & String both are class so we are writing 'T' and 'S' in Caps.
- The method naming should be - calculateArea()
- args[]

IDE = Eclipse
For now = notepad

VS Code -
Javac = Test.java
Java Test

Class file = test.class
It will run at any platform
When we are passing some arguments then
Java test (arguments)

1/2/23

① run & compile -
cd Foldername
↓
Javac filename.java
↓
Javac Test

input Java Classname
ip ip ip

- Difference between Java & C
OOP & procedure oriented

Java	C
1) It is an object oriented language.	1) It is Procedure oriented language.
2) It is interpreted language.	2) It is a Complied language.
3) High level language.	3) low-level language
4) It doesn't support Pointers.	4) It supports Pointers.
5) It support inheritance.	5) It doesn't support inheritance
6) top down approach works here.	6) Bottom-top approach works here.
7) 32 keywords	7) 57 keywords
8) Code executed directly Here there is,	8) code executed by Jvm.
9) multithreading.	9) multithreading. Here there is no
10) Platform independent & secure	10) It isn't Platform independent and secure.

- why Pointers don't use in Java.
Java don't use pointers because using Pointer the memory area can be directly accessed which is a security issue, Pointers need so memory spaces at the runtime, to reduce the usage of memory spaces Java does not support Pointers. & increase the security

*[Pointer is a variable that holds the address of another variable. So by using pointer we can easily get the address of any element in memory directly.]

1) what is Package in Jave - A Java package is a collection of related classes and interfaces providing access protection and namespace management. it is a mechanism to encapsulate a group of classes, sub packages and interfaces.

Public = access specifier.

class = keyword

static = It is a keyword, it means that the particular member belongs to a type itself, rather than to an instance of the type.

String arg[] = command line arguments

String array

System = Predefined class

Out = output stream

Println = method to display the string

Access modifiers/ specifiers — They are keywords in object-oriented languages that set the accessibility of classes, methods, and other members. Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components. In C++ there are only three access modifiers. In Java - there are 4 access modifiers and they are —

//keyword - Java keywords are also known as reserved words. keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

Ex - abstract, boolean, break, byte, case, catch, char, class, continue, double, do, else, enum, default, extends, final, finally, float, for, if, implements, import, instance of, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, try, void, volatile, while

//Static = The static keyword in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class. If we use static keyword we don't have to declare object.

Predefined class - Pre-defined class name can be used as a class name.

output stream = It accepts output bytes and sends them to some sink. It is an abstract class is the superclass of all classes representing an output stream of bytes.

Package, Private, Protected, ...

i) Private - The access level of a private modifier is only within the class. It can't be accessed from outside the class.

ii) Default - The access level of a default modifier is only within the package. It can't be accessed from outside the package.

iii) Protected - The access level of a protected modifier is within the package and outside the package through the child class. It can't be accessed from outside the package, if we don't make the child class.

output stream = it accepts output bytes

int, interface, long, native, new, null

i) Public - The access level of a Public modifier is everywhere. It can be accessed from within class, outside the class, within the package & outside the package.

ii) Protected - The access level of a protected modifier is within the package and outside the package but through the child class. If you want to access it outside the package, it

iii) Protected - The access level of a protected modifier is within the package and outside the package but through the child class. If you want to access it outside the package, it

(**)

int, it accepts output bytes

Method - A method is a block of code which only runs when it is called. You can pass data, known as parameters, into a method. They are also known as functions.

Class variables - Sometimes, we want to have variables that are common to all objects. This accomplished with the static modifier. Fields that have the static modifier in their declaration are called static fields or class variables or static variables. They are associated with the class, rather than with any object.

Binding

1) Static binding - When type of the object is determined at compiled time, it is known as static binding.

2) Dynamic binding - When type of the object is determined at run-time, it is known as dynamic binding.

13/2/23

- Token = main, void, public
  cannot be ~~some~~ shorted
- Identifiers - func name, variable name etc. (method)
- Keywords = Break, if, else, continue.
- Datatypes - int, char, ~~void~~ etc
- Constants in java → float → Numeric Constant (Ex. 10)
  → ~~Numeric~~ character Constant
- Size of datatypes - (Done)

Types of Variable → Instant variables (this are objects)
→ ~~Class~~ The instances are different.

Diff

2) Class variables - They are the part of the class. If we chang the class variable then it will be reflected to all the variables of the class.

3) Local variables - They are the variable that are defined inside the method.

- Type casting - In java, type casting is a method or process that converts a datatype into another data type in both ways manually and automatically.
- Conditional operator - The conditional operator is also known as the ternary operator. This operator consists of 3 operands and

is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable.

- **Dot operator** The dot operator (.) is used to access the instance variables and methods of class objects. Ex - Personal.age() // Reference to the variable age
  Personal.Salary() // Reference to the method Salary()

It is also used to access classes and sub packages from a package.

- **How to create an object** (two classes are there)

How to declare a class

```
Class Rectangle
{
    int length;
    int breadth;
    void getData(int x, int y)
    {
        length = x;
        breadth = y;
    }
    int rectArea()
    {
        int area = length * breadth;
        return(area);
    }
}
```

x, y are local variables

How to make an object

```
Class Prog
{
    public static void main(String args[])
    {
        Rectangle R1 = new Rectangle();
        R1.getData(5,10);
        int area = R1.rectArea();
        S.o.p("Area of R1 is" + area);
    }
}
```

Any method inside the class can use any data members of that class

R1 is a reference, it is similar to the pointer.

new Rectangle() = we are making an object of Rectangle R1

after S.o.p we can also write,

```
Rectangle R2
R2 = new Rectangle();
R2.getData(7,3);
```

// Constructor = special method. It is used to initialize the data members of the object. It's name is as same

as class.

```
class Rectangle
{
    int length;
    int breadth;

Rectangle(int x, int y)
    Constructor
    {
        length = x
        breadth = y
    }
int rectArea()
{
    int area = length * breadth;
    return (area);
}
}
```

```
class Prog
{
public static void main(String args[])
{
Rectangle R1 = new Rectangle(5, 10);
[No need to write R1.getdata(5,10); if you write it]
int area = R1.RectArea();
S.O.P("Area of R1 is "+ Area);

}
}
```

If you don't pass any parameters in the Rectangle (constructor) then Java will take it as '0'.

Convention :-

21/2/23

CW

- Constructor is used to initialize the object.

Advantage - we don't have to write so much code, much easier to use.

- Overloading - It is a concept where we make multiple definations of a method (like polymorphism).

```
class Room
{
  int length;
  int breadth;
  Room(int a, int b)
  {
    length = a;
    breadth = b;
  }
  Room(int a)
  {
    length = breadth = a;
  }
  int area()
  {
    return (length * breadth);
  }
}
```

```
class T
{
  public static void main(String args[])
  {
    Room R1 = new Room(5,6);
    Room R2 = new Room(5);
    S.O.P("Area of R1 is" + R1.area());
    S.O.P("Area of R2 is" + R2.area());
  }
}
```

- Polymorphism - the ability to take more than one form. The property of characteristics of OOPs.

'+' is stating a polymorphism.

- Encapsulation - Data methods + methods

- Purely object oriented Program? why it's called purely.

- Ans > Pure object oriented language or complete O.O.L. are fully Object O.L. which supports or have features which treats everything inside prog as objects. It doesn't support Primitive data type (like int, char, float, bool, etc.)

There are 7 qualities to be satisfied for a Programming language to be Pure O.O. They are:

1) Encapsulation / Data hiding
2) Inheritance
3) polymorphism
4) Abstraction
5) All Predefined types are objects
6) All user defined types are objects
7) All operations performed on objects must be only through methods exposed at the object.

Ex - Smalltalk

They are called Pure because there is no diff. between values which are objects and values which are Primitive types.

Actual & formal argument in java:

Actual argument / Parameter - The actual value that is passed into the method by a caller.

Formal argument / Parameter - It is a single identifier, the name, and the actual parameter the value (of the same type as the identifier)

(Identifier = unique names)

27/2/23

Ex-1

```
class Add
{
    static int summation(int a, int b)
    {
        return (a+b);
    }
}
class Addition
{
    P.S.V.M(String args[])
    {
        int x = Add.summation(5,6);
        S.O.P("sum is", x);
    }
}
```

(static method)

- a, b is local variable of method static int summation.
- x is local variable of method P.S.V.M (String args[]).
- static / Class variable is shared by all the object. It is declared under Class Add

```
{
    static ... (int x, y)
    {
    }
}
```

(int a, int b)

x, y = Class / static variable

int add (int a, int b) → static

formal argument = while the implimentation of the method the callers can change the actual parameter i.e. the variable name (note: callers can't change the data type). If the callers do so, then the argument will be called formal argument, it is actually an identifire.

static

method b P.S.V.m

- If we create object Add A1 = new Add();
  Add A2 = new Add(). Then A1 & A2 both one having x & y. by both x & y value be different in case of A1 & A2 object.

- ```
  Class Add
  {
      static int x;
      - - -
  }
  ```

this x is dependent on class. It will be written like Add.x . Add is class. So static variable = class variable.

In this code static is a class int summation() is dependent on the class. Here we can call this method by Add.summation(5,6); we dont have to create object. If it will not be a class dependent on we have to create the object.

(2)

```
Class Add
{
    static int summation(int a, int b)
    { return(a+b);
    }
    static void Print(int a)
    {
        S.O.P("sum is" + a);
    }
}
Class Addition
{
    P.S.V.M(String args[])
    {
        int x = Add.summation(5,6);
        Add.print(x)
    }
}
```

This a static method we don't have to create object. If static will not be declared then we have to create object.

[To shorten the prog. we define static method].

↳ Use of static method —

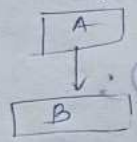
(1)
```

20/3/23

Class

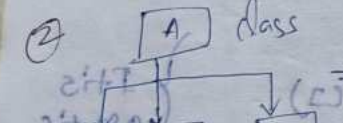
Inheritance: It is a concept of OOPs. We will inherite the properties of some class in one class. If we inherite from a single class then it is single inheritance.

① A class ⟹ single inheritance



B is inherited from A.

② A class ⟹ Hierarchical Inheritance

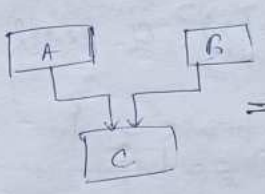


B has a property then C & D can have same properties.

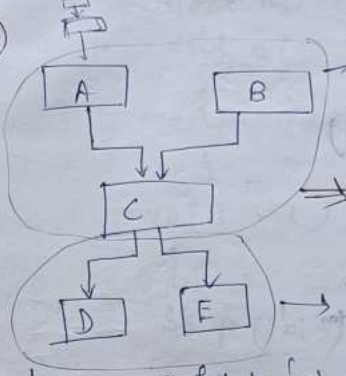
③ A → B → C ⟹ multi level Inheritance

B has some prop. of A & C has some prop. of B.

④ A, B → C ⟹ multiple Inheritance



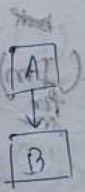
⑤ A, B → C; C → D, E ⟹ Hybrid Inheritance ⟹ Hierarchical



It's a hybrid of 2 or 3 kinds of Inheritance.

Ex of Inheritance

```
class A                    // Parent / super / base class
{
    int x;
    float y;
    void m()
    {
    }
}
class B extends A          // child / sub / derived class
{
    int a;
    float b;
}
```



Class B will get all of the properties of A. ie, B has a, b and method and also x, y & the method (the class of A). From B we can call x.

```
void h()
{
}
}

class T
{
  psvm (String args[])
  {
    B B1 = new B();
    B1.a = 5;
    B1.x = 7;
    B1.m();
  }
}
```

// method overloading

```
class A
{
  int x;
  float y;
  void m()
  {
    S.o.p("Hello");
  }
  void m(int a)
  {
  }
}
```

this has to be create in this class not B

```
class B extends A
{
  int a;
  float b;
  void m()
  {
  }
  void m()
  {
    S.o.p("Hi");
  }
}
```

method overwriting

```
Class T
{
  psvm(String args[])
  {
    B B1 = new B();
    B1.a = 5;
    B1.x = 7;
    B1.m();   → this will print "Hi", as the previous method void m() of A is overwritten.
    B1.m(5);
    A A1 = new A();
    A1.m();
    A1.m(6);  → this will print 'Hello'
  }
}
```

Distinguish between parent & sub class method overwriting

distinguish between arguments & method overloading. (Imp for viva)

Use = We can achieve code reuse by inheritance & can add more features in it.

27/3/23

String object = String is a obj in java

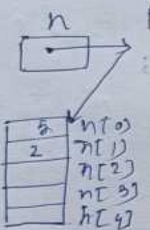
~~String first;~~

Imp

Array –

int n[ ]; →

n = new int[5]; →



n[0] = 5
n[1] = 2

int num[ ] = {2, 5, 6, 7} →

x = num.length ⇒ To find the length of array

Sop (x)

→ o/p = 4

String (object of java) [In Java there is an object class, all the classes made by programmer are the child of object class]

String first;

first = new String("Aliah");

String last;

last = new String ("University"); [function of object class, it consists of many methods, it's mainly used for getting access over all the classes]

String s = first + last;

String array, Lexicographical

String methods i) s1.toLowerCase()

first.toLowerCase → ALIAH

ii) s1.compareTo(s2), it will be 0 if s1 = s2, it will be 1 if s1 ≠ s2 → to sort the strings we have to use it.

s1 = Agra, s2 = Aliah

s1 < s2

iii) s1.equalIgnoreCase()

~~List~~ → X

Wrapper class – This is used for methods so that, we can convert the ~~primitive~~ data types objects into other types

boolean, int etc has corresponding wrapper class (Integer, Boolean)

Package = It is inbuild of user defined.

→ ParseInt = Convert string type to primitive type

Integer.ParseInt ⇒ wrapper class

Long.ParseLong [str]

String ⟶ object of that type / wrapper class

we use, Integer.valueOf(str)

primitive data type ⟶ String

toString — from primitive to string

Object type

Integer n = new Integer(5)

wrapper class

n = 5 will come out as object

(primitive, String, Object)

parseInt method ⟶ fast type casting

Scanner = faster

~~Interface~~

Abstract class - A single method is abstract the whole class is abstract class. there is no object in abstract class. It should be overriding.

```
abstract class Shape
{   void getData()
    {
        abstract int calc();   <- abstract method
    }                          <- don't have any definition
}
```

uses - We can make a abstract definition of abs. class and we can use it in inheritate class. If we can't make any object of abstract class also.

in the 0 class ⇒ Abstract class (sl have we only add inside this definition has to be written in child class (overriding))

Interface - It is a concept we use in java. It is a class. In interface every method is abstract. (Abstract class has atleast one abstract method) it has no object. It should always inheritate

Difference between interface & abstract class

Abstract class	Interface
1) Abstract class can have abstract and non-abstract methods.	1) Interface can have only abstract methods.
2) It doesn't support multiple inheritance.	2) Interface supports multiple inheritance.
3) It can have final, non-final, static & non-static variables.	3) It has only static & final variables.
4) It can provide the implementation of interface	4) It can't provide the implementation of abstract class

5) The abstract keyword is used to declean abstract class.	5) The interface keyword is used to decleare interface.
6) It can extend another java class & implement multiple java interfaces.	6) It can extend another java interface only.
7) It can be extended using keyword "extends"	7) It can be implemented using keyword "implements"
8) A java abstract class can have class members like private, protected etc.	8) members of a java interface are public by default.
9) Ex - Public abstract class shape{ Public abstract void draw(); }	9) Ex - Public interface Drawable{ void draw(); }

Final variable - It is constant we cannot change it.

Interface

```
interface I
{
    ... all methods are abstract here (we are not needed to write abstract, it is by default abstract)
}
```

Static final - All the data members are static & final. It is a keyword that is used to represent the class members. It is basically used with methods & variables to indicate that it is a part of the class not the object.

Ex-①

```
interface Area
{
    final static pi = 3.14;
    float Compute(int x, int y);
}
class Rect implements Area
{
    float compute(int x, int y)
    {
        return (x*y);
    }
}
```

as we are extending from interface

```
class circ implements Area
{
    float Compute(int x, int y)
    {
        return (pi*x*y);
    }
}
```

→ access Specifier / visibility Control

this things are also applicable for methods

Private	
Private Protected	
friendly (nothing is written = default)	
Protected	
Public	

i)

```
class A
{
    Private int x;
    void Print()
    {
        S.o.p(x);
    }
}
```

x can be accessed inside the class only

If we create an Object of this class we can't access x.

ii)

```
class A
{
    Private Protected int x;
    void print()
    { S.o.p(x); }
```

}

If we create a object of this class then we can access it. If there is a class but it isn't sub class of the class, but it is present in package, so we can't access it.
(It can be accessed from sub classes of the package, and also inside the the same class.)

iii) class A
{
int A;
void Print()
{ SOP(x);
}}

Accessed by:
From inside the class from the sub-classes of the same package, other class in same package

iv) Class A
{
Protected int x;
void Print()
{
SOP(x);
}
}

Accessed by -
Sub Class in same package
. . . other package
inside the class

v) Can be accessed by everything.
class A
{
public int x;
void Print()
{ S.O.P(x);
}
}

- Java tokens - A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:
1) keywords 2) Identifiers 3) constants 4) special symbols 5) operators

1) keyword - keywords are pre-defined or reserved words in a programming language. Each keyword is meant to perform a specific function in a program. Since keywords are referred names because by doing so, we are for a

compiler, they can't be used as variable names because by doing so we are trying to assign a new meaning to the keyword which isn't allowed. The keywords in Java are – abstract, int, for, final, byte etc.

2) Identifiers – They are used as the general terminology for naming of variables, functions and arrays. These are user-defined names consisting of an arbitrarily long sequence of letter or, the underscore (_) as a first character. Identifier names must differ in spelling & case from any keywords. You cannot use keywords as identifiers; they are reserved for special use. Once declared, you can use the identifier in later program statements to refer to the associated value. A special kind of identifier, called a statement label, can be used in goto statements. Ex- myVariable, x, i etc (valid), myVariable123geeks (not valid).

3) Constants / Literals – Constants are also like normal variables. But, the only difference is, their values can't be modified by the program once they are defined. Constants refer to fixed values. They are also called as literals. Constants may belong to any of the data type.

Syntax – final data_type variable_name;

4) Special symbols – The following special symbols are used in Java having some special meaning and thus, can't be used for some other purpose.

[ ] , ( ) , { } , ; , * , =

5) Operators – Java provides many types of operators which can be used according to the need. They can classified based on the functionality they provide. Some of the types are-

a) Arithmetic operators – These operators involve the mathematical operations that can be used to perform various simple or advanced arithmetic operations on the primitive data types referred

to as the operands. These operators consist of various unary & binary operators that can be applied on a single or two operands. Let's look at the various operators that java has to provide under the arithmetic operators.

Operators	Result
+	Addition of two numbers
-	Subtraction
*	multiplication
/	Division
%	Divides two numbers and returns the remainder

b) Unary operators - Java unary operators are the types that need only one operand to perform any operation like increment, decrement, negation etc. It consists of various arithmetic, logical & others operators that operate on a single operand. Let's look at the various unary operators in detail and see how they operate.

Unary minus(-) → this operator can be used to convert a positive value to a negative one. [Syntax = -(operand)]

'NOT' operator(!) → this is used to convert true to false and vice versa. Basically, it reverses the logical state of an operand. [Syntax = !(operand)]

Increment (++) → It is used to increment the value of an integer. It can be used in two separate ways:

Post-increment operator [syntax = num++]

Pre-increment operator [syntax = ++num]

[Illustration; num = 5
num = 5
after that num = 6]

[Illustration; num = 5
++num = 6]

Decrement (--) → It is used to decrement the value of an integer. It can be used into 2 separate ways-

Post-decrement operator [syntax = num--]

[Illustration; num = 5
num-- = 5]

Pre-decrement operator [syntax = --num]

[Illustration; num = 5
--num = 4]

Bitwise complement (~) → this unary operator returns the 1's complement representation of the input value or operand, i.e. with all bits inverted, which means it makes every 0 to 1, and every 1 to 0. [syntax = ~(operand)]

[Illustration, a = 5 [0101]
result = ~5
~0101 = 1010]

c) Assignment operators → These operators are used to assign values to a variable. The left side operand of the assignment operator is a variable, and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type of the operand on the left side. Otherwise, the compiler will raise an error. This means that the assignment operators have right to left associativity, i.e. the value given on the right hand side of the operator is assigned to the variable on the left. Therefore, the r.h.s value must be declared before using it or should be a constant. The general format of the assignment operator is, variable operator value;

Simple Assignment operator → It is used with the '=' sign where the left side consists of the operand & the right side consists of a value.

Compound Assignment operator → The compound operator is used where +, -, * and / is used along with the = operator.

1. (=) operator → It is used to assign the value on the right to the variable on the left.

2. (+=) operator → This operator is a compound of '+' & '=' operators. It operates by adding the current value of the variable on the left to the value on the right & then assigning the result to the operand on the left.

3. (-=) operator → It is used for subtracting the variable's value on the right from the current value of the

variable on the left and then assigning the result to the operand on the left.

4. (*=) operator → This operator multiplies

5. (/= operator) → this operator is a compound of '/' and '=' operators. It operates by dividing the current value of the variable on the left by the value on the right and then assigning the quotient to the operand on the left.

6. (%= operator) the remainder to the operand on the left.

d) Relational operator They are a bunch of binary operators used to check for relations between two operands including equality, greater than, less than etc. they return a boolean result after the comparison and are extensively used in looping statements as well as conditional if-else statements and so on. the general format of representing relational operators is:

variable1 relation_operator variable2

Relational operators in java -

'Equal to' operator (==) → This operator is used to check whether the 2 given operands are equal or not. the operator returns true if the operand at the left-hand side is equal to the right side.

'not equal to' (!=) → This operator is used to check whether the two given operands are equal or not. It functions opposite to that of the Equal-to-operator. It returns true if the operand at the left-hand side is not equal to the right.

'Greater than' operator (>) This checks whether the 1st operand is greater than the 2nd operand or not. The operator returns true when the operand at the left-hand side is greater than the right-hand side.

'less than' operator (<) - this checks whether the 1st operand is less than the 2nd operand or not. The operator returns true when the operand at the left-hand side is less than the right-hand side. It functions opposite to that of the greater-than operator.

'greater than or equal to' (>=) - this checks whether the 1st operand is greater than or equal to the 2nd operand or not. The operator returns true when the operand at the left-hand side is greater than or equal to the right-hand side.

Less than or equal to (<=) - This checks whether the 1st operand is less than or equal to the 2nd operand or not. The operator returns true when the operand at the left-hand side is less than or equal to the right-hand side.

e) Logical operators - these are used to perform logical "AND", "OR" and "NOT" operations, i.e. the function similar to AND gate & OR gate in digital electronics. They are used to combine two or more conditions/constraints or to complement the evaluation of the original condition under particular consideration. One thing to keep in mind is, while using AND operators, if the 2nd condition is not evaluated if the 1st one is false. whereas while using OR operators, if the 2nd condition is not evaluated if the 1st one is true, i.e the AND and OR operators have a short-circuiting effect. Used extensively to test for several conditions for making a decision.

1) AND operator (&&) - if (a && b) [if both are true execute else don't]

2) OR operator (||) - if (a||b) [if one of them is true or both are true execute else don't]

3) NOT operator (!) - !(a<b) [returns false if a is smaller than b]

f) ternary operator - Java ternary operator is the only conditional operator that takes three operands. It's a one-liner replacement for the if-then-else statement & is used a lot in java programming. we can use the ternary operator in place of if else conditions or even switch conditions using nested ternary operators. Although it follows the same algorithm as of if-else statement, the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.

[Syntax -
variable = Exp1 ? Exp2 : Exp3]

g) Bitwise operators - These are used to perform the manipulation of individual bits of a number. They can be used with any integral type (char, short, int etc). They are used when performing update and query operations of the Binary indexed trees.

Now let's look at each one of the bitwise operators in java.

1) Bitwise OR(|) - This operator is a binary operator, denoted by '|'. It returns bit by bit OR of i/p values, i.e. if either of bits is 1, it gives 1 else it shows 0.

2) Bitwise AND(&) - This operator is a binary operator denoted by '&'. It returns bit by bit AND of i/p values. i.e., if both bits are 1, it gives 1, else it shows 0.

3) Bitwise XOR(^) - This operator is a binary operator denoted by '^'. It returns bit by bit XOR of i/p values, i.e. if corresponding bits are different, it gives 1, else it shows 0.

4) Bitwise Complement(~) - This operator is a unary operator, denoted by '~'. It returns the one's Complement representation of the i/p value, i.e. with all bits inverted, which means it makes every 0 to 1, and every 1 to 0.

6) Bit-shift operators (shift operators)

They are used to shift the bits of a number left or right thereby multiplying or dividing the number by two, respectively. They can be used when we have to multiply or divide a number by two.

syntax -

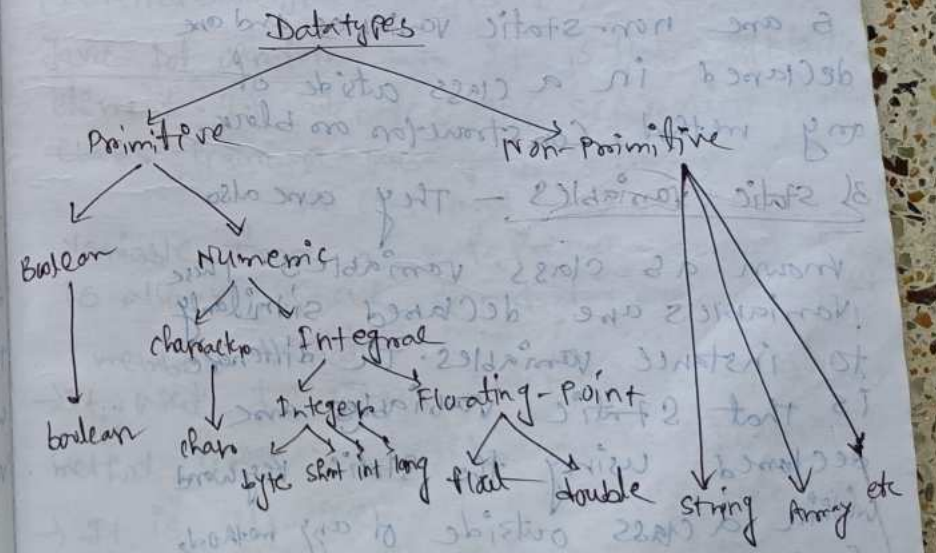
```
number shift_of number_of_places_to_shift;
```

They are of 4 types -

1) signed right shift op. (>>)
2) unsigned " " (>>>)
3) Left shift operator (<<)
4) Unsigned left shift operator (<<<)
   - moves all bites by a given no. of bits to right
   - Same as signed right shift, but the vacant leftmost position is filled with 0 instead of the sign bit
   - moves all bits by a given no. of bits to left
   - Same as signed left shift, but the vacant left most position is filled with 0 instead of the sign bit.

Data type - [Java is statically typed and also a strongly typed language because in java, each type of data] // Data types in java are of different sizes and values that can be stored in the variable [that is made as per convenience and circumstances to covers up all test cases.] // Java has two 2 categories in which data types are segregated.

1) **Primitive Data type** - such as boolean, char, int, short etc.
2) **non-Primitive Data type or Object Data type** - such as string, Array, etc //



Variables Java variables are the data containers that save the data values during java program execution. Every variable is assigned to a data type that designates the type and quantity of value it can hold. A variable is memory location name for the data.

types of variable in java -

1) **Local variables** - A variable defined within a block or method or constructor is called a local variable

2) **Instance variable** - Instance variable is a non-static variables and are declared in a class outside of any method, constructors or block.

3) **Static variables** - They are also known as class variables. These variables are declared similarly to instance variables. The difference is that static variables are declared using the static keyword within a class outside of any method, constructors or block.

Type casting - This means to change one state into another state and is done by the programmer using the cast operator. This is done during the program design time by the programmers. Typecasting also refers to Narrow conversion. Because in many cases, we have to cast large datatype values into smaller datatype values according to the requirement of the operations. [We can also convert large datatype values into smaller datatype values that's why type casting called Narrow Casting.]

[Syntax = () is cast operator]

[Required Datatype = (target Type) variable]

Java Dot operator It is just a syntactic element. It denotes the separation of method from a package, separation of a the class and variable from a reference variable. It is also known as separator or period or member operator.

→ It is used to separate a variable or method from a reference variable.

→ It is also used to access classes and subpackages from a package.

→ It is also used to access the member

of a package or a class.

Constructors - In java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor memory for the object is allocated in the memory.
It is a special type of method which is used to initialize the object.
Every time an object is created using the new() keyword, at least one constructor is called.
It calls a default constructor if there is no constructor available in the class. In such case, java compiler provides a default constructor by default.

method overloading - If a class has multiple methods having same name but different in parameters, it is known as method overloading. If we have to perform only one operation having same name of the methods then we can take help of method overloading.

It increases the readability of the program.

Polymorphism - the word polymorphism means having many forms. In simple words we can define polymorphism as the ability of a message to be displayed in more than one form. The word 'poly' means many and 'morphs' means forms, so it means many forms.

Encapsulation - It is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines. The Java Bean class is the example of a fully encapsulated class. We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setters & getters methods to set & get the data in it.

static keyword - The static keyword in java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as class variable)
2. method (also known as a class method)
3. Block
4. Nested class

Inheritance - It is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPS.

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance is also known as a Parent-child relationship.

terms used in inheritance

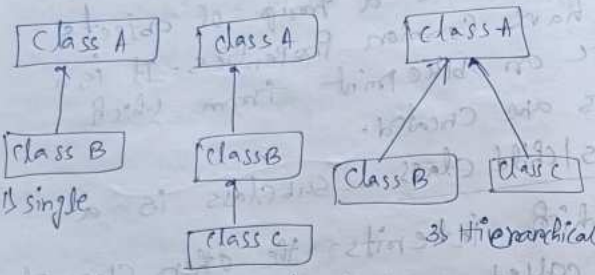
class - A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.

Sub class/child class - subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.

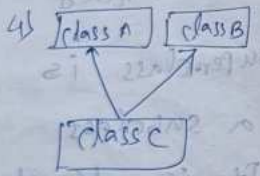
super class/Parent class - superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

Reusability - As the name specifies, reusability is a mechanism which facilitates you to reuse the field and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Types of Inheritance in Java –



1) single
2) multilevel
3) Hierarchical



4) multiple

1) **Single Inheritance** – when a class inherits another class, it is known as a single inheritance. Ex – Dog class inherits the Animal class, so there is the single inheritance.

2) **multilevel Inheritance** – when there is a chain of inheritance, it is known as multilevel inheritance.
Ex – Baby Dog class inherits the Dog class that again inherits the Animal class, so there is multilevel inheritance.

3) **Hierarchical Inheritance** – When 2 or more class inherits a single class, it is known as hierarchical inheritance. Ex = Dog & cat classes inherits the Animal class, so there is hierarchical inheritance.

4) **multiple Inheritance (through Interfaces)**

Here, one class can have more than one superclass and inherits features from all parent classes. [Note = java doesn't support multiple inheritances with class. In java we can achieve multiple inheritances only through interfaces.]

Method overriding –
If subclass has the same method as declared in the parent (superclass), it is known as method overriding. In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent classes it is known as method overriding.

2nd Array - normally, an array is a collection of similar type of elements which has contiguous memory location. Java array is an object which contains elements of a similar data type. Additionally, the elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a java array."

2nd String - In java string is basically an object that represents sequence of char values. An array of characters works same as java String.

String methods → compareTo(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

String array - An array is an essential and most used data structure in java. A string array is an array of a fixed number of string values. (Include definition & detail about string & array). The string array works similarly to other data types of array.

2nd Wrapper Class - The wrapper class in java provides the mechanism to convert primitive into object and object into primitive. Auto boxing & unboxing feature convert primitives into objects and objects into primitives automatically.

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

Abstract class - A class which is declared with the abstract keyword is known as an abstract class in java. It can have abstract and non-abstract methods (method with the body).

→ An abstract class must be declared with an abstract keyword.

→ It can have abstract and non-abstract methods.

→ It can't be instantiated.

→ It can have constructors & static methods also.

→ It can have final methods which will force the subclass not to change the body of the method.

Interface - An interface in java is a blueprint of a class. It is declared by using interface keyword. It has static constants & abstract methods. The interface in java is a mechanism to achieve abstraction.

→ It can't be instantiated just like the abstract class.

→ We can have default and static methods in an interface.

→ It represents the IS-A relationship

→ We can have private methods in an interface.

Final variable - In java we can use final keyword with variables, methods, and classes. When final keyword is used with a variable of primitive data types such as int, float, etc. the value of the variable can't be changed. (3) when it is used with methods & classes, the methods & classes can't be changed also.

10/4/23

→ Interface from one interface to another interface, then we have to use extends. From a single interface we can implement 2 classes.

Classes → multiple inheritance isn't allowed in java (but in interface is allowed)

Java packages - It is kind of folder in which there are related classes.

Import java.IO
Import java.util
Import java.lang

Import java.util.*;

If we want to import everything we have to import *, if we want to import a single class we have to import IO (But basically we need for not to write additional things)

convention
Package name . Classname . methodname

Imp (multithreading)

ms dos don't support multitasking (in 1995).

context switching

It switches contents/works by using one core, they fastly (multitasking)

2 Core . 2 processors

Process. A prog. which is executing in CPU.

If we kill parent process the child process will die. (All the processes have some common resources)

Light weight process = Threads.

The thread is a light weight process.

How to create thread - By creating a thread class. We will use an inbuild class, thread is a class in java. We can write the whole code in run.

2) By converting a class into a thread (by runable interface, inside it we have a run methode, we can overwrite it)

How to extnd a thread class

1st approach

```
class A extends Thread        (Thread class = Inbuild class)
{                             (Run method is public)
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            S.O.P("From Thread A: i="+i);
        }
        S.O.P("Exit from A");
    }
}
class B extends Thread
{
    public void run()
    {
        for(int j=1; j<=5; j++)
        {
            S.O.P("From thread B: j="+j);
        }
        S.O.P("Exit from B");
    }
}
```

```
Class Threadprog
{
    psvm(String args[])
    {
        // object A & B (start of)
        newA().start(); // start() - Already defined in thread Class it will start a thread
        newB().start();
    }
    // not added any constructor for default construction
}
```

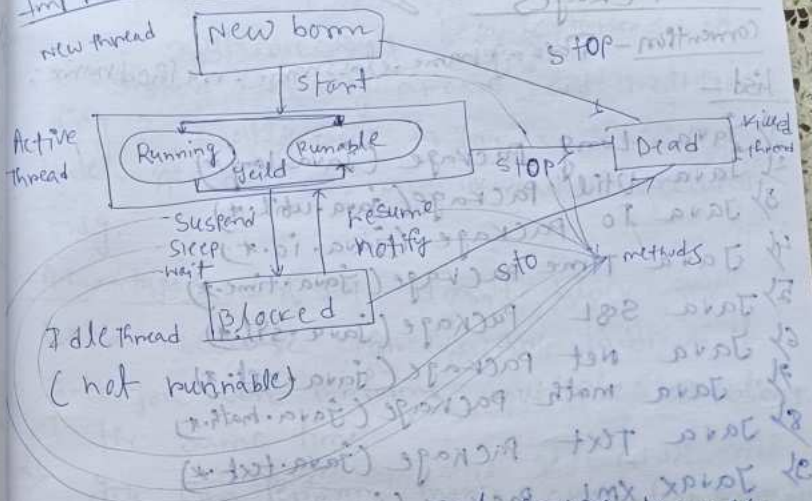
O/P →

From thread A: i = 1
From thread A: i = 2
From B: j = 1
B: j = 2
B: j = 3
A: i = 3
B: j = 4
B j = 5
Exit from B
A: i = 4
A: i = 5
Exit from A

If we run it again it will give a different O/P (there is a bare chance to get same o/p) we got O/P in running state.

Imp for exam

Life cycle of thread



Resume or notify to get blocked state to active thread.

To get active thread we have to use stop

Runable: To get a active thread it will go to blocked 1st.

Java Packages

Convention - Packagename.classname.methodname

List -
1) Java Lang package (java.lang.*)
2) Java Util Package (java.util.*)
3) Java Io package (java.io.*)
4) Java Time package (java.time.*)
5) Java SQL Package (java.sql.*)
6) Java Net Package (java.net.*)
7) Java math package (java.math.*)
8) Java Text Package (java.text.*)
9) Javax XML Package (javax.xml.*)
10) Javax Swing Package (javax.swing.*)

Multithreading

[multitasking - It is the ability of CPU to perform multiple tasks simultaneously (at a time). There will be continuous context switching of CPU between the tasks which is rapid & hence results in better Performance.]

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub process [Thread definition] the smallest unit of processing. multiprocessing and multithreading both are used to achieve multitasking. However we use multithreading more than multiprocessing because threads use a shared memory area. Java multithreading is mostly used in games, animation etc.

Note: At a time one thread is executed only.

Advantages - 1) It doesn't block the user because threads are independent and you can perform multiple operations at the same time.

2) You can perform many operations together so it saves time.

3) Threads are independent, so it doesn't affect other threads if exception occurs in a single thread.

Process - It provides control of native processes started by ProcessBuilder [A prog in execution that is executing on CPU]

How to create a thread

There are 2 ways to create a thread:
1) By extending Thread class
2) By implementing Runnable interface.

Thread class - Thread class provides constructors and methods to create

end perform operations on a thread. thread class extends object class and implements Runnable interface. Some commonly used constructors of thread class:

→ Thread()
→ Thread (String name)
→ Thread (Runnable r)
→ Thread (Runnable r, String name)

Runnable Interface

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method name run().

public void run(): is used to perform actions for a thread.

CW

Exception

26/4/23

This are detected at compilation time

Runtime errors, compile type errors (Syntax error, Symmentic errors (i=i, instead of i=1))

(missing ; missing brackets, missing } etc)

This are only detected at runtime (no way to predict)

y, y=0, then it will show error.

Array out of bound exception

a[-1], but array is started from 0, so it will show an error

Exceptions are also runtime errors.

Compile time errors aren't the exception.

Exception is a object in java & it is the subclass of the object. Exception also have some childClass. (subclass)

1) Arithmetic exception (Divide by '0')
2) Array index out of bound exception
3) File not found exception (the file that isn't present)
4) IO exception (Not able to read from a file or any kind of i/p devices)
5) Number Convert exception
6) out of memory exception
7) Stack overflow exception
8) String Index out of Bound exception

Exception is the Superclass of all the subclass. If you don't know the exception type then you can write it.

Exception → Checked Exception
→ Unchecked exception

Checked exception - Customised Error we can handle it.

Unchecked exception - we have not handle it, jvm will handle it by its own.
↳ will stop the execution.

```
try *
{
  Statement 1;
}
catch(Exception e)
{
  Statement 2;
}
```

```
try
{
  x = a/(b-c);
}
catch(ArithmeticException e)
{
  S.o.p("Divide by zero!!");
  b = b+c;
}
```

↳ It is a class, catch is like a function & 'e' is like a argument
↳ Here in place of e we can write anything ;)

```
try
{
  b = a/(b-c);
  a[8] = 8 ; a[i] = 85
}
catch(ArrayIndexOutOfBoundException e)
{
  S.o.p("array index out of bound");
}
catch(ArithmeticException e)
{
  S.o.p("Divide by zero");
}
```

By doing that this part will be skipped

At last we can add finally.

```
finally
{
}
```

↳ this block will always execute

It is used for, some works that has to be done behind the completion of execution. (like, some file has to be closed etc)

Exception Handeling

It is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained. (Runtime errors such as ClassNotFoundException, IOException etc)

Exception – In java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

Types of Java Exceptions

There are mainly 2 types of exceptions: Checked and unchecked. An error is considered as the unchecked exception. However, according to oracle, there are 3 types of exceptions namely:

1) **Checked Exception** – The classes that directly inherit the Throwable class except RuntimeException and Error are known as checked exceptions. For example, IOException, SQLException etc. Checked exceptions are checked at compile-time.

2) **Unchecked Exception** – The classes that inherit the RuntimeException are known as unchecked exceptions. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time, but they are checked at runtime.

3) **Error** – It is irrecoverable. Some examples of errors are OutOfMemoryError, VirtualMachineError, AssertionError etc.

Runtime Error – Runtime errors occur or we can say, are detected during the execution of the program. Sometimes these are discovered when the user enters an invalid data or data which is not relevant. Runtime errors occurs when a program doesn't contain any syntax errors but asks the computer to do something that the computer is unable to reliably do. During compilation, the compiler has no technique to detect these kinds of errors. It is the JVM that detects it while the program is running. To handle the errors during the run time we can put our error code inside the try block & catch the error inside the catch block. Ex – Divide by 0,

Compile time Errors - Compile Time Errors are those errors which prevent the code from running because of an incorrect syntax such as a missing semicolon at the end of a statement or a missing brackets, class not found etc. These errors are detected by the java compiler & an error message is displayed on the screen while compiling. Compile time Errors are sometimes also referred to as syntax errors. These kind of errors are easy to spot & rectify because the java compiler finds them for you. The compiler will tell you which piece of code in the program got in trouble & its best guess as to what you did wrong. Usually the compiler indicates the exact line where the error is, or sometimes the line just before it, however, if the problem is with incorrectly nested braces the actual error may be at the beginning of the block." In effect, Syntax errors represent grammatical errors in the use of the programming language. Ex - misspelled variable name, method, logical error etc, syntax error.

Built-in Exceptions -

① ArithmeticException - It is thrown when an exceptional condition has occured in an arithmetic operation.

② ArrayIndexOutofBoundsException - It is thrown to indicate that an array has been accessed with an illegal index. the index is either negative or greater than or equal to the size of the array.

③ ClassNotFoundException - This Exception is raised when we try to access class whose definition is not found.

④ FileNotFoundException - this Exception is raised when a file is not accessible or doesn't open.

⑤ IOException - It is thrown when an input-output operation failed or interrupted

⑥ InterruptedException - It is thrown when a thread is waiting, sleeping, or

doing some processing, & it is interrupted.

(7) NoSuchFieldException – It is thrown when a class doesn't contain the field (or variable) specified

(8) NoSuchMethodException – It is thrown when we are accessing a method that isn't found.

(9) NullPointerException – This exception is raised when referring to the members of a null object. Null represents nothing.

(10) NumberFormatException – This exception is raised when a method could not convert a string into a numeric format.

(11) RuntimeException – This represents an exception that occurs during runtime.

(12) StringIndexOutOfBoundsException – It is thrown by String class methods to indicate that the index is either negative or greater than the size of the string.

(13) IllegalArgumentException – This will throw the error or error statement when the method receives an argument which is not accurately fit to the given relation or condition. It comes under the unchecked exception.

(14) IllegalStateException – This exception will throw an error or error message when the method is not accessed for the particular or operation in the application. It comes under the unchecked exception.

Userdefined Exceptions

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, the user can also create exceptions which are called 'user defined Exceptions'.

Try & catch Exc

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try & catch keywords come in pairs:

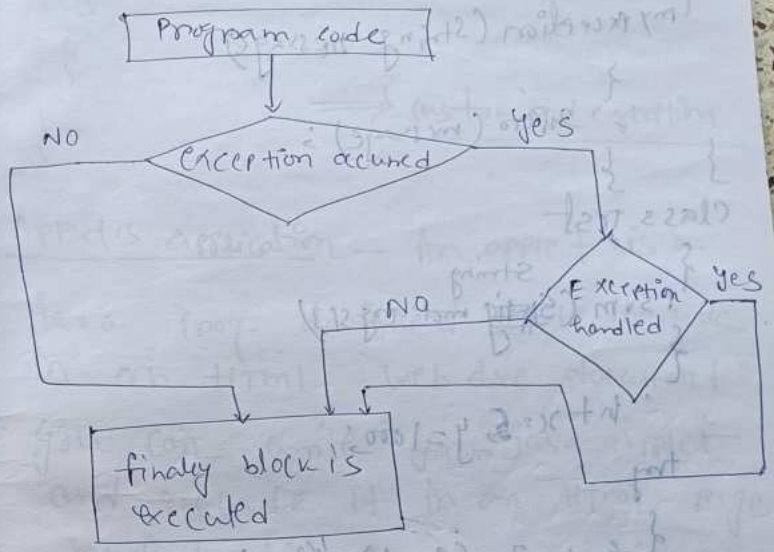
try {
// block of code to try
}
catch (Exception e) {
// Block of code to handle errors
}

Throw keyword - The Java throw keyword is used to throw an exception explicitly. We specify the exception object which is to be thrown. The exception has some message with it that provides the error description. These exceptions may be related to user I/Ps, servers etc.

Finaly block - Java finaly block is used to execute important code such as closing the connection etc.

Java finaly block is always executed whether an exception is handled or not. Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

the finaly block follows the try catch block.



3/5/23

① Throwable class

[Throwable class extends throwable]

super(a,b) = It calls the constructor of the parent class.

this operator = It is a reference variable that is the reference of the object that is called. It will go to a methods, it will go misably like

getData(int a)
This.a = 5;

```
class
myException extends Throwable
constructors
{myException(String message)
    {
     super(message);
    }
}
class Test
{
psvm(String args[])
{
  int x=5, y=1000;
try
{float z=(float)x/(float)y;
if (z<0.01)
{
throw new myException("To small no.");
}
}
Catch (myException e)
{
SOP("caught myException");
SOP(e.getMessage());
finally
{
  S.O.P("Inside finally");
}
}
}
```

creating an object of my exception

This will refer inside the throwable class

⟹ customised exception

Applet's application — An applet is a java prog. designed to be include in an HTML web doc document. You can write your java applet and include it in an HTML page. much in the same way an image is included.

(4)

Prev yp qn Sdh 2022

Gp-A

1) Imp Differentiate Compile time Polymorphism & Runtime Polymorphism.

Compile time Polymorphism	Runtime Polymorphism
1) In Compile time polymorphism, call is resolved by the Compiler.	1) In Run time Polymorphism, call is not resolved by Compiler
2) It is also known as static binding, Early binding and overloading as well.	2) overriding is named as Runtime polymorphism.
3) It provides faster execution.	3) It provides slow execution.
4) It is less flexible.	4) It is more flexible.
5) It is used to increase the readability of the prog.	5) It is used to provide the specific implementation of the method that is already provided by its super class.
6) It is performed within class.	

6) It occurs in 2 classes that have IS-A (inheritance) relationship.

7) Here parameters must be different.

8) Distinguish between arguments

#) Here parameters must be same.

&) It is distinguish between parent & subclass.

2) Elaborate how a new thread can be created in Java with proper code snippets.

Done (1st Part)

```
class A extends Thread
{
  public void run()
  {
    for(int i=1; i<=5; i++)
    {
      S.o.p("From thread A: i= "+i);
    }
    S.o.p("Exit from A");
  }
}
class B extends Thread
{
  public void run()
  {
    for(int j=1; j<=5; j++)
    {
      S.o.p("From Thread B: j= "+j);
    }
    S.o.p("Exit from the B");
  }
}
class C extends Thread
{
  public void run()
  {
    for(int k=1; k<=5; k++)
    {
      S.o.p("From thread C: k= "+k);
    }
    S.o.p("Exit from C");
  }
}
```

The o/p -

```
From Thread A: i = 1
From Thread B: j = 2
From Thread C: k = 1
-
-
Exit from A
-
-
Exit from B
Exit from C
```

3) what is the difference between a class, abstract class & interfaces?
Done

Class	Abstract class	Interface
1) A class can have both ~~es~~ abstract class as well as concrete class	1) It can have abstract & non abstract methods.	1) It can have only abstract methods
2) multiple inheritance doesn't support	2) ---- doesn't support	2) - supports these
3) can have final, non final, static, non static variables supported.	3) ----	3) only static and final variables

Rest of contrasts = Done
[class = Abstract class = diff = Done]

4) Describe different ~~types~~ kinds of Inheritance with a diagram. Done

5) what are the different types of variables in Java? Done

Diagram

```
{ static int y = 5;
  int x = 0;
  {
    int n = 5;
  }
  {
    int m = 10;
  }
}
```

x is instance variable
y is class/static variable
n, m → local variables

6) ~~Im~~ ~~Imp~~ How to create your own package in java? ~~Package~~ = Done

1) First we should choose a name for the package we are going to create and include. The package command in the 1st line in the java programme Source code.

2) Further inclusion of classes, interfaces annotation types, etc that is required in the package can be made in the package. For ex, the below single statem-

entire creates a package named as called "FirstPackage".
Syntax:- to declare the name of the package to be created.
The package statement simply defines in which package the classes defined belong.

Package FirstPackage;

Implementation - To create a class inside a package -

1) First declare the package name as the first statement of our Program.
2) Then we can include a class as a part of the package.

(7) Describe dynamic method dispatch.
It is known as runtime polymorphism or method overriding. Rest of Part - Done

CP-B

8) a) Describe all the features of an object oriented programming.
The objs oriented programming language supports all the features of normal Programming languages. In additional it supports some imp. concepts and terminology which has made it popular among programming methodology.
The imp features of OOP are -

(1) Class - An abstract matter of real world entity or concept - or thing. Like Car, human being etc.
(2) Object - An instance (which is basically representation the (class) shows class's attributes (variables).
(3) method - Private or Public functions associated with a class, to access the class (obj)
(4) Inheritance - Ability to get other class features. creating new class from Parent classes. As Chr class is inherited from vehicle class.
(5) Polymorphism - Acquiring more than one form (Ex- method overloading, virtual

functions). Abstraction

⑥ Data Handeling:- Hide complexity & gives necessary details.

⑦ Encapsulation - Hide complex part of the prog. Hiding data (attributes) of a class from others by declaring ; Private, public etc.

⑧ Overloading - A function is overloaded when same name is given to different functions.

⑨ Reusability - one class can be used again and again.

⑩ Abstraction - This is a concept by which we only access the necessary attributes of that class without knowing much about the others attributes or methods of that class.

9) Describe method overloading & method overriding. Done

Ex method overloading ex -

```
class Room
{
    float length;
    float breadth;
    Room (float x, float y).        // constructor1
    { length = x;
      breadth = y;
    }
Room( float x)                      // Constructor2
  {
    length = breadth = x;
  }
int area()
  { return( length * breadth);
```

}
}

Here we have used 2 constructors of same name 'Room()'.

```
Room room1 = new Room(25.0, 15.0);
Room room2 = new Room(20.0);
```

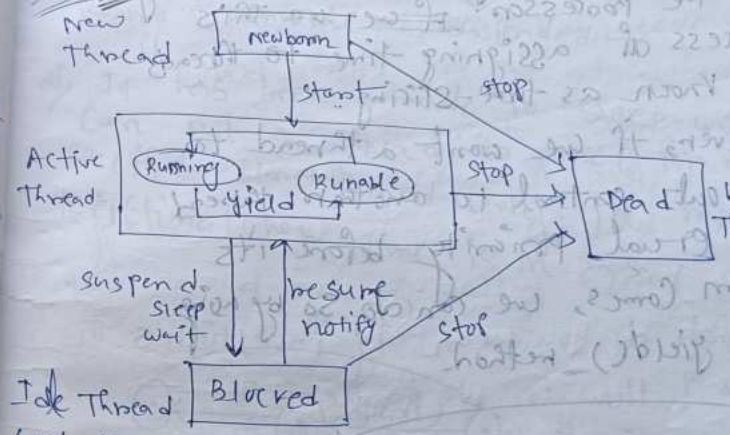
Here room1 obj is representing the rectangle & room2 obj is representing the square, by the same constructor name 'Room()'. This is method overloading.

```
class super
{
  ---
  ---
  void display()
  {
    s.o.p("---");
  }
}
class sub extends super
{
  ---
  ---
  void display();
  {
    s.o.p("---");
    s.o.p("---");
  }
}
```

display() is overridden.

the method

9) Draw & describe the state transition diagram of a thread in java.



State transition diagram of a thread

newborn state - when we create a thread obj, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state, we can do only one of the following things with it:

→ schedule it for running using start() method.
→ Kill it using stop() method.

Runnable state

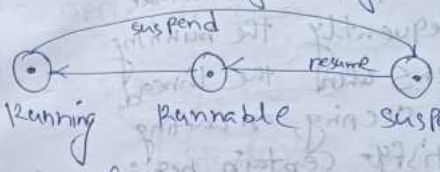
The runnable state means that the thread is ready for execution and is waiting for the availability of the processor. If we have this process of assigning time to threads is known as time-slicing.

However, if we want a thread to give out control to another thread of equal priority before its turn comes, we can do so by using the yield() method.

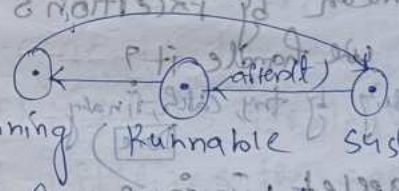
Running State

Running means that the processor has given its time to the thread for its execution. The thread runs until it relinquishes control on its own or it is preempted by a higher priority thread. A running thread may relinquish its control in one of the following situations.

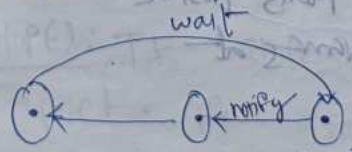
1) It has been suspended using suspend() method. A suspended thread can be revived by using the resume() method.



2) It has been made to sleep. We can put a thread to sleep for a specific time period using the method sleep(time) where time is in ms.



3) It has been told to wait until some event occurs. This is done using the wait() method. The thread can be scheduled to run again using notify() method.



Blocked state - A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state. This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements.

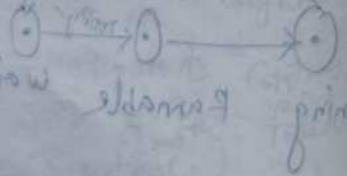
Dead state - Every thread has a life cycle. A running thread ends its life when it has completed executing its run() method. It is a natural death.

(10) What do you mean by Exceptions in java? How do we handle it?

Done (Exception handling by try, catch, finally - Done)

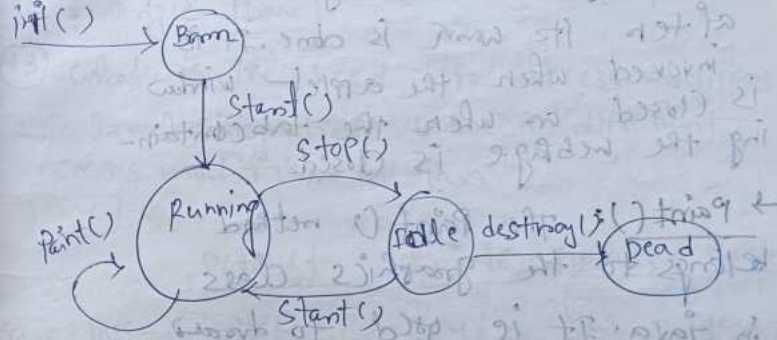
(11) ~~Not in syllabus~~ What is an applet in java?

Applet is a special type of programme that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.



Not in syllabus

Draw and describe the life cycle of an applet in java.



There are 5 methods of an applet life cycle & they are

→ **init()**: It is the 1st method that initializes the applet.

→ **start()**: It contains the actual code of the applet & starts the applet. It is invoked immediately after the init() method.

→ **stop()**: It stops the execution of the applet. The stop() method is invoked whenever the applet is stopped, minimized

moving from one tab to another in the browsing the stop() method is invoked.

→ destroy(): It destroys the applet after its work is done. It is invoked when the applet window is closed or when the tab containing the webpage is closed.

→ Paint(): the Paint() method belongs to the graphics class in java. It is used to draw shapes like circle, square etc.

sequence of method execution when an applet is executed:

1) init()
2) start()
3) Paint()

(12) write a java program to create a class shape. using constructor overloading initialize values of shape objects named circle, square, Rectangle & triangle. Done

(13) what is the need for access modifiers in java? Differentiate between different access modifiers in java.

Done

	Default	Private	Protected	Public
Same class	yes	yes	yes	yes
Same package subclass	yes	NO	yes	yes
Same package non-subclass	yes	NO	yes	yes
Different package sub class	yes	NO	yes	yes
Different package non-subclass	NO	NO	NO	yes

Prv Jan 2021

Sn-A

1) Briefly explain how java platform is independent?

Java is platform-independent because it uses a virtual machine. The java programming language & all APIs are compiled into bytecodes. Bytecodes are effectively platform-independent. The virtual machine takes care of the differences between the bytecodes for the different platforms. The run time requirements for java are therefore very small. The java virtual machine takes care of all hardware-related issues so that no code has to be compiled for different hardware. Done

2) what is abstract class? Give example

Done. An example of an abstract class in the JDK is AbstractMap, which is a part of collections Framework.

```
public abstract class Main {
public abstract void Draw();
}
```

3) what are tokens? Done

4) What are the access modifiers? Done

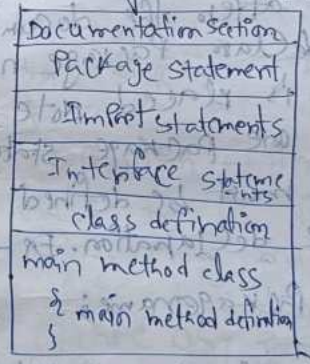
5) What is dynamic or run time polymorphism? Done

Sn-B

1) Explain different parts of java program with an appropriate example.

Different parts of java prog

Structure of java prog - Java is an object oriented programming language, platform independent, & secure programming language that makes it popular. Using the java programming language, we can develop a wide variety of applications.



← Structure of java Prog.

A typical structure of a java prog contains the following elements:

→ Documentation section
→ Package declaration
→ Import Statements
→ Interface section
→ Class Definition
→ class variables & variables (constants)
→ main method class
→ methods & behaviours

i) Documentation section – It is an important section but optional for java prog. It includes basic info about a java prog. It improves readability of the prog. Ex – use of comments (multiline, single line & documentation comments).

→ single line comments = // statement
→ multiline comments = /* statement */
→ Documentation comments = /** statement */

ii) Package declaration – It is also optional. It is placed just after the documentation section. Here we declare package name in which the class is placed. Note that there can be only one package statement in a java prog. It must be defined before any class & interface declaration. Ex –

Package ~~java demo~~ packagename;
keyword

iii) Import statements – The package contains many predefined classes & interfaces. If we want to use any class of particular package, we need to import that class. The import statement represents the class stored in the other package. We use import keyword to import the class.

import java.util.*; // It import all the classes

iv) Interface section – It is optional section. We can create interface in this section if required. We use interface keyword to create an interface. It contains only constants & method declaration. We can use interface by using implements keyword. An interface can also be used with others interface by using the extends keyword. Ex –

```
interface Car
{
  void start();
  void stop();
}
```

v) class Defination – In this section, we define the class. It is vital part of a java prog. A java prog may contain more than one class defination. Without class we can't create any java prog. We use class keyword to define the class. It is a vital part of java prog. Ex –

```
class student // class defination
{
}
```

vi) class variables & constants – In this section, we define variable & constants that are to be used later in the prog. They are defined just after class defination. They stores

values of the parameters. Ex-

```
class student
{
    String name;
    int id;
    double percentage;
}
```

viii) main method class - In this section, we define main() method. It is essential for all Java Progs, as the execution of Java Prog starts from the main() method. It must be inside the class. Ex-

```
psvm(String args[])
{
}
```

viii) methods & Behavior - In this section, we define the funtionality of the Prog. by using methods. The methods are the set of instructions that we want to perform.

Ex-

```
Public class Demo
{
    psvm(S. a)
    {
        void display()
        {
            S.o.P("Hello");
        }
    }
}
```

2) Explain the significance of Public, Protected & Private access specifiers in inheritance.

methods declared public in a superclass also must be public in all subclass.

protected - must either to be protected or public - they can't be private.

private aren't inherited at all, so there is no rule for them.

Access specifiers - Done

types of access specifiers - Done

3) Explain the process of defining and creating package with suitable examples.

Done

4) What is interface? what are the similarities between interfaces & classes?

Done

They are similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface. Along with abstract methods, an interface may also contain constants, default methods, static methods, static methods & nested types.

5) what is array? How do you declare the array in java? Give examples.

[Done] to create an array — but we have to give the data type (generally int) then the name [ ] after that we have to ...

```
int intArray[ ]; // declaring array
intArray = new int[20]; // allocating memory space
```

(OR)

```
int[ ] intArray = new int[20];
```

→ to create array (1D) we use this syntax =

datatype [ ] = array name;

(2D)

datatype [ ] [ ] = array name;

6) write the difference between primitive & non-primitive data types in java.

Primitive data type	Non-Primitive datatype
1) They are predefined by language it self	1) They need to be defined by us.
2) They are stored in stack.	2) They are stored in stack but the original object is in the heap.
3) Here the type size depends on the type of data structure.	3) Here, the size isn't fix.
4) It can't be NULL	4) It can consist a NULL value.
5) They the primitive ds can be used to call the methods.	5) The non-primitive ds can't be used to call methods.
6) The computer knows them from previously.	6) we have to make the computer know of them.

Gr-C

1) what is inheritance? write a prog. to implement single level inheritance assuming suitable data.

[Done]

2) Explain any 5 object oriented features supported by java with examples.

1) Inheritance - [Done]. 2) Method overloading - [Done] 3) method overriding - [Done], 4) classes - [Done] 5) objects - [Done], Ex - [Done]

3) Differentiate between procedural programming & OOP language. write a java prog. to initialize & display different types of ints, and floating point variables.

Procedural prog.

1) It is divided into different blocks of code.
2) It follows top-down approach
3) Less secure
4) Doesn't support overloading.
5) Abstraction is not possible here.
6) All data members are public here.
7) Data modification is the function requires more effort
8) Useful for small problems.

Ex = C, FORTRAN etc

OOPs

1) It is divided into the different objects of classes
2) Follows bottom up approach.
3) Highly secure
4) Support overloading
5) Abstraction is possible here.
6) Data members can be Public, Private, Protected, here default
7) Data modification is the function requires less effort.
8) Useful for complex problems

Ex = Java, C++ etc

```
public class main1
{
    psvm(sa)
    {
        short temp = -200;
        int range = -4250000;
        float number = -42.3f;
        double b = 123.435555;
        long C = 2000000000000000;
        s.o.p (" short:" + temp);
        s.o.p (" int:" + range);
        s.o.p (" long:" + C);
        s.o.p (" float:" + number);
        s.o.p (" double:" + b);
    }
}
```

o/p = short: -200
int: -4250000
long: 2000000000000000
float: -42.3
double: 123.435555

4] Explain how java is robust & interactive. write a java prog to sum only first 5 elements of array using for each looping

Done

```
Public class sum-n {
 Psvm (Sa)
 {
 int [] arr = new int [] {1,2,3,4,5};
int sum=0;
for (int i=0 ; i<arr.length; i++)
 {
 sum=sum+arr[i];
 }
 S.O.P("result: " + sum);
 }
}
```

O/P = result :15

5] Define Thread & explain 2 ways of creation of thread. Explain the life cycle of Thread. Done

~~Thread~~

OOPs sugg. 2023 by Sir (Soln)

1] This keyword - 'this' keyword refers to the current object in a method or constructor

The most common use of the this keyword is to eliminate the confusion between class attributes & parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

this can be used for -

→ Invove current class constructor
→ Invoke current class method
→ Return the class obj
→ Pass an argument in the method call
→ Pass an argument in the constructor call

Ex- Public class Main {

```
int x;
Public Main(int x) {
  this.x = x;
}
P S V M (Sa[]) {
main myobj = new main (5);
S.O.P ("value of x=" + myobj.x);
```

}
}

② Static variable, method, block & nested class example - (Actual Qn= about static keyword)

Ex. of static variable & method -

```
class Test
{
    p sv m (S a[])
    {
        Public static int i; // Static variable
    }
}
Class Test static
{
    p sv m (S a[]) // It is also a static method
    {
        Test obj = new Test();
        Test obj1 = new Test();
    }
}
```

Ex of Static block -

```
Public class multipleStaticBlocks{
{
    SOP ("Instance block-1");
}
{
    SOP ("Instance block-2");
}
```

```
Static {
    S.O.P ("static block-1"); // static block
}
Static {
    S.O.P ("static block-2"); // static block
}
Psvm (S a[])
    new multipleStaticBlocks ();
}
}
```

Ex of static nested class

```
class TestOuter1 {
    static int data = 30;
    static class Inner{
    void msg(){ S.O.P ("data is" + data);}
    }
Psvm (S a[]) {
TestOuter1.Inner obj = new TestOuter1.Inner();
    obj.msg();
}
}
```

(3) working of start() & run() in Thread: (or) use of start() & run() in thread.

In java multithreading, start() & run() are 2 most important methods working of them -

1) New thread Creation - when prog calls the start() method, a new thread is created & then run() method is executed. But if we directly call the run() method, no new thread will be created & run() method will be executed as normal method call.

(4) Diff. of start() & run() in thread.

start()	run()
1) Creates a new thread & the run() method is executed on the newly created thread.	1) No new thread is created & the run() method is executed on the calling thread itself.
2) Can't be invoked more than one time otherwise throws Java.lang.IllegalStateException.	2) multiple invocation is possible.
3) Defined in Java.lang.Thread class.	3) Defined in Java.lang.Runnable interface & must be overriden in the implementing class.

(5) super keyword - this keyword in java is a reference variable that is used to refer to parent class objs. An understanding of Inheritance & Polymorphism is needed in order to understand the super keyword. It is majorly used in -

→ Use of super with variables
→ methods
→ Constructors

(6) StringBuffer - Java StringBuffer class is used to create mutable (modifiable) string objs. The StringBuffer class in Java is the same as string class except it is mutable, i.e it can be changed. Note - Java StringBuffer class

is thread-safe i.e. multiple threads can't access it simultaneously. So, it is safe & wise result in an order.

(7) Diff. of string & StringBuffer.

String	StringBuffer
1) The String class is immutable.	1) It is mutable.
2) String class uses string constant pool.	2) StringBuffer uses heap memory.
3) It is slower while performing concatenation operation.	3) It is faster while performing concatenation operation.
4) It overrides the equals() method of Obj class. So you can compare the contents of 2 strings by equals() method.	4) It doesn't override the equals() method of Obj class.
5) It is slow & consumes more memory during concatenation of too many strings.	5) It is fast & consumes less memory during the concatenation of too many strings.

Rest of 7ms = [pone]

The prog of multithreading, done in class = [Dmg]

---

1) What is oop? ***

As the name suggests, object oriented programming or oops refers to langs that use objects in Programming. oops aims to implement real-world entities like inheritance, hiding (abstraction) etc in programming. The main aim of oop is to bind the data & the functions that operate on them together (so that no other part of the code can access this data except that func).