25/9/24

JAVA

⇒ WORA (Write Once run anywhere)

→ Platform independent — JVM, JRE, JDK. (→ platform dependent (takes byte code → sends to OS))

→ code → byte code → execute.

⇒ JVM executes the byte code → platform independent.

* Java devely by Sun Microsystems in 1991 named as "Oak".
* In 1995 it is renamed as Java.
* Java as both complation and interpreter. [not 100% OOP]
* Class - basic framework / common template / blueprint of object.
* Object - physical existence of class. (real world entity.)
* Variables in class mentions the state of class
* Methods of the class mentions the behaviour of the class.
* Limitations of structures in C,
  1. No initialization
  2. No method
  3. No security.
* Syntax to create object of class: classname objname = new classname();
* Keyword new → allocates the memory.
* Memory will never allocated while creating class.
* In java String is wrapper class
* Primitive datatypes: char, byte, short, int, double, float, long, boolean.
* Class name is given in Pascal case. Eg: StudentDetailsClass{ }
* Method name is of camel case - Eg: conSing()
* Java filename should be name of class we main exsists. It should be public class.
* To access object use dot operator. Eg: S1.name = or with getter/setter.

* All objects default value is NULL.
* Java as no garbage value, throws error if not initialized (Local scope)
* Char datatype size in java is 2 bytes.
* Compliers default constructor is "Default constructure" Eg: Student();
* boolean default value "false".

Expt.No. 26/09/2024

Page No.
Date

Constructor:

* Constructor name should be same as class name.
* As no return type, because it is not returning anything only it sets the value.
* It is invoked when the object is created.

Type:
T1: * Default constructor - no arguments in parantheses
Eg: For greetings, assigning static values.
T2: * Parameterized constructor -
* Constructor overloading - have same constructor name, have different no. of arguments.

Constructor chaining:

Constructor created inside another constructor.
If a default con inside the parameter cons, when we call the object of the parameter default is also invoked.

```
Eg: Student (int rollno)
    {   this.()
        this.rollno = rollno;
    }
    Student (int rollno, String name)
    {   this(rollno);
        this.name = name;   }
    Student (int rollno, String name, int age)
    {   this (rollno, name);
        this.age = age;
    }
```

* Reference - mentions the object location.

* If Student S6 = S5; ⇒ same object (memory in heap) is alloted to both references.

* Here if S5 is altered it also affect S6.

* If it should not affect. We can do,

(i) Student S6 = new Student(S5.name, S5.roll-no);

(ii) Student (Student S5) ⟶ Copy constructure.
{ this(name, roll-no); }
Student S6 = new Student(S5);

Variable - name given to particular memory location.

Types: (i) Local (ii) Instance (iii) Static / class variable

(i) Local - variable declared inside the method.

(ii) Instance - Variable has scope in its class fully. (object binding)
↳ will vary from object to object.

(iii) Static - binds with class, - variable is static variable.
↳ doesn't change for every object. (Remains same).
↳ the single memory is used even it's called/used many times. (Shared memory)
↳ Memory is created at the class loading time, i.e. before creating objects.
↳ Instead of this is instace, here we use "class_name.variable". Eg: Department.name = name;

Global scope

* Instance, Static variable if value is not assigned, then default value is used.

Static function

* A static member can be accessed inside another static member without the object.

Expt.No.....

Page No.....
Date .....

Static function:

function:	Static	Non-static	Non-static	Static
	↓ (no obj)	↓ (obj not need)	↓ (no obj need)	↓ (obj need)
Ivar:	static	static	non-static	Non-static

Task 3:

class student → Int rollno, age ; String name, deptName ; Static int clsTotal = 0, Static String collegeName = "SECE" ; static displayTotall() { } display details() { }

Requirements: (main function)

* Constructor (default, parameterized), chaining, copy constructor, method call for both static & non-static.

STATIC BLOCK:

* Executes before the execution of main block.

→ Execution precedence:

* Static block, Static method, Static variable
*

—

Static block:

* initialize defaultly to '0'. (variables)
* Used to initialize, update static variable.

Initializer block:

* Without any method name and return type. Eg: { }.
* Executes before the execution of initializer block.
* Executes in object creation time before the default constructor execution.
* Used when we have change the value of instance or non-static variable.
* In array .length is property and for string length() is a function.

METHOD OVERLOADING:

```
Eg: static int findLength (int arr[])
    { return arr.length ; }
    static int findLength (String s)
    { return s.length() ; }
    static int findLength (String s , String r)
    { return s.length() + r.length() ; }
```

Real time eg1: search of a product in e-commerse platform even with product id or with product name.

Real time eg2: Leetcode search.

* Method overloading as typecasting concept.
  ↓
  converts the datatype to another datatype (larger than that)
* It also applicable for constructor loading.
* this() should be the first line of the constructor.

Expt.No.....

Page No.....
Date .....

INHERITANCE :

* Parent ← Child (Child to parent - extends).

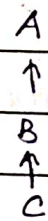
TYPE :

(i) Simple inheritance :

⇒ one child extracts all methods of one parent.

⇒ When the object is created for child, it can also access the functions in parent.

(ii) Multilevel inheritance :



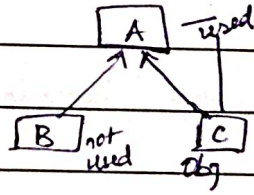
⇒ A child have parent, grandparent as level by level.

⇒ Chain of inheritance. [another class can be extended from the inherited class].

⇒ Eg : Iphone and its updating versions.

(iii) Hierarchical inheritance :

⇒ Two or more childrens can be inherited from a parent class



(iv) Multiple inheritance :

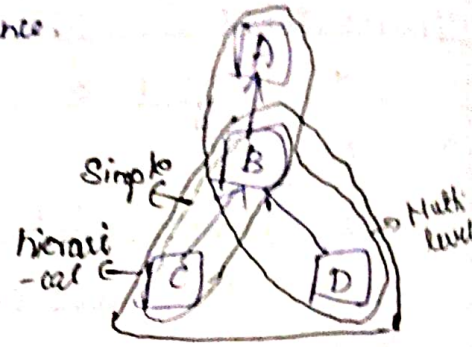
⇒ Two or more parents for a single child.

⇒ Not possible for java, because the child class may get confuse from which parent class it should extends. Compiler gets confused.

⇒ It causes ambiguity error - issues that are not defined clearly. The diff results in several of programs supports our observations.

(v) Hybrid inheritance :

→ More than one type of inheritance in the same code is called hybrid inheritance.



OVERRIDING :

SUPER :

FOR-EACH Loop :

* Iterates with the element without using index :
* Syntax : for(int i : arr) // arr[] = {1,2,3,4,5}
{ System.out.println(arr[i]); }

Solve leetcode : 66 , 1281 , 2011 , 88 , 58 , 771 , 1108.

28th Jan ARRAYS :

Arrays array or may not be continuous memory allocation. Because array is an object which is stored in heap memory.

30/8/

Arrays.copyOf() - copies the elements of arrays upto the needed length.

Arrays.copyOfRange() - copies element fr within the gvn range.

Arrays.mismatch(arr1, arr2) - compares 2 arrays and returns the index from where the mismatch occurs.
- if no mismatch returns -1.

Arrays.parallelSort() - uses parallel sort - merge sort algorithm. takes less time. Used when array size is large.

Arrays.compare(a1, a2) - compares two arrays. equal - 0.
a1 > a2 = 1, a2 > a1 = -1.

obj_name.clone(a1) = clone is not a static method. where the obj_name is the array_name. return type - object
- creates new memory and assigns the copied values, the updation in arr2 does not effect arr1.

PACKAGES:
⇒ collection of many classes files
⇒ Eg: util, lang, io etc
⇒ can also be user defined.

ACCESS MODIFIERS:
⇒ Restricts the usage visibility of the code.
⇒ Public, Protected, Default, Private. (low-to-high restriction)

Private - access/visible only within the class
Default - used/visible anywhere within the same package. Also known as package private.
Protected - used in same package and also in different package by extending its class.

Expt.No.....

Page No.....
Date .....

Public - used in same & different package by creating objects directly without extending it.

METHOD OVERRIDDING:

isa - is the concept of inheritance.

hasa - is the concept of composition.

→ When the same function in parent class as to be used in child class with same argument but with same / different functionality is called method overridding.

→ Annotation - @Override

Upcasting - When there is a common feature in parent and child class, using the object of the parent class we can print the methods overidden by the child class.

Eg: class Parent { walk() talk(); }

class Child { walk() talk() run() sleep() }

main: Parent P = new Child();

P can access only the method overriden.

FINAL:

* Keyword in java. It is used for enhanced protection.
* These values cannot be changed.
* final variable :
  - ↳ 4 types → final instance, final local, final static, final reference.
  - ↳ final int A = 10 ; // since 10 is constant (can't change the variable name is in capital.
  - ↳ all maths operations can be performed but one updation or reassignment can be done.

Local final : ↳ The value of final is initialized only once.
↳ static does not change memory location, final cannot change the value.

Instance final
- ↳ Eg: Roll_no in Student class which cannot be changed.
- ↳ It should be assign a value when declaring itself else using constructor or initializer block.
- ↳ It different every instance but the value assigned to that instance cannot be changed.
- ↳ Eg: S1.ROLLNO = 100 ;
  S2.ROLLNO = 101 ;

Final static :
- ↳ It doesn't change both memory and value.
- ↳ Eg: static final String COLLEGE_NAME = "SECE" ;
  Eg: Pin code of any particular area.
- ↳ Initialized at the time of declaration otherwise can be initialized with static block.

```
static { COLLEGE_NAME = "SECE" ; }
```

Reference final ↳ static Stu = new Stu() ; S1.name = "ABC" ; S2.name = "XYZ" ;
Here only address of obj is unchangable, the attributes are normal so they can be changed.

Expt.No.....

Page No.....
Date .....

↳ Eg: final int arr[] = {1,2,3,4,5}; arr[3] = 100; is possible

↳ Here the final is only for reference, so that the address cannot change, but the value in the address can be changed.

PlusOne Prblm in Leetcode : I/P = [1,9,9] - O/P = [2,0,0]

```
public static int[] addOne (int arr[])
{   for( int i = arr.length - 1 ; i >= 0 ; i--)
    {   if (arr[i] < 9)
        {   arr[i] += 1 ; return arr; }
    } arr[i] = 0 ;
    int result[] = new int [arr.length+1] ;
    result[0] = 1 ; // other indexes are assigned to default '0'.
} return result ;
```

Final method :

* cannot be overriden. prefixed with keyword "final"
* Eg: CalculateFunction used to calculate interest rate in bank.
  Eg : CGPA calculation.
* Final String in a class can be extended in child class and used / using super keyword.

Final class :

* The class cannot be inherited (extended).
* It does not have child class but it may be a final class
* It can be accessed with its own object

Private constructor:

* We cannot create object. Because private is not visible in other classes.
* It can be accessed by calling the constructor inside another constructor.
* Private constructor can have object only within that class.

POLYMORPHISM:

* Poly - many, morphs - things
* Eg1: method overloading (same name with diff arg length).
  Eg2: method overriding (same name but in different forms).
* 2 types → compile time, runtime polymorphism.
* Compile time - Decides which as to run in the compile time itself. Also known as static polymorphism. Eg: Overloading method.
* Runtime - Decides which as to run at the runtime only. Also known as dynamic polymorphism. Eg: method overriding.

↳ Downcasting - assigning the parent methods to the child's reference using typecasting. (classcast exp error)

- The field which is upcasted can also be downcasted, otherwise it can't be downcasted.

instanceOf() - method in java.lang.
- returns boolean
- Used to identify whether the Object is the instance of that class or not.

↳ downcasting is not safe to use because instanceOf should be known.

Expt.No..03/10/2024

Page No.....
Date .....

ABSTRACTION :

* Displaying only the needed things for the user and hiding the functions of how it works.
* This can be achieved by 3 methods → abstract class, encapsulation and interface.

(i) Using Encapsulation :

↳ Use the private method into another method.

(ii) Using abstract class :

↳ Is used when particular percentage of function alone as to be abstracted.

↳ abstract method does not have body, instance cannot be created.

↳ It can be used only by inheritance. So the object as to be created for its child class.

↳ Eg: Security in phone.

↳ abstract method is used when some function does not have common difinition, it as only the templelate using that the function can be altered as per the need.

* Final cannot be Inherited but abstract must be inherited.

↳ Abstract method is used only in abstract class.

↳ Using abstract class we can achieve 0-100% of abstraction

* Encapsulation is implement level process, abstraction is designing level process.

* Upcasting can be done in abstraction class, because it cannot create only object but it can have reference.

* In abstract class there should be atleast 1 abstract method, but there is no condition that all methods should be abstract.

* Constructor can also be created, it is invoked by creating the object for its child class.

VARIABLE LENGTH ARGUMENT: (Not oops concept).

```
Code: public static int sum(int... arr)   // ... will take argument
      { int total = 0;                        in dynamic size.
        for (int i : arr)
        { total += i; }
        return total;
      }
```

* It can be of any datatype.

(i) Combining all the strings in the array seperated by space.

```
Code: String join (String... arr) // to seperate with space ' '.(c=' ')
      { String s = "";
        for (String i : arr)
        { s += s+i; }   s = s+i+c;
        return s;
      }
```

Expt.No.....

Page No.....
Date .....

(iii) INTERFACE :

* Prefixed with keyword "interface".
* It is 100% abstraction
* All methods are public abstract, everything here is public, all variable are final.
* A class can be extended using "implements" keyword.
* Interface can achieve multiple inheritance.
* Any method implements in child class it should be "public" access modifier.
* Object cannot be created directly of Interface class
* Upcasting can be done.
* Only declaration is done. It is defined it its child class.
* @ Reference can be created for interface
* static method in interface can be accessed without creating object. and body can also included.
* All methods in parent class as to be defined in child class compulsary, because it is abstract.
* Eg: Calculator
* default ('keyword') method [not accer specifier] is used to write body in interface without static method. (after java 8th version).
* If multiple defaul method is used in multiple parent class it will create ambiguity because the OS get confuse while executing child class.
* If the same is done for normal class it does not show error. Because it will assign the same definition to the multiple methods declared in multiple parent class.
* Constructor, protected, private cannot be used.

* When a ~~object~~ reference is assigned NULL in constructor it does not show compile time error, it throws null pointer reference error in runtime.

* If it is a static method, it can be accessed.

STRING :

String.format("format", ...args) -

4/10 Obj_name.repeat(n4) - repeats the string n times

obj_name.toUpperCase() -

obj_name.toLowerCase() -

(S1 == S2) - false (when memory same)
- true (with different memory)

S1.equals(S2) -

S1.equalsIgnoreCase(S2) -

S1.compareTo(S2) - equal (0), (S1 > S2 - (+ve)), (S2 > S1 - (-ve))

S1.compareToIgnoreCase(S2) -

Expt.No.....

Page No.....
Date .....

startsWith('the') - returns true / false

startsWith(' ', pos) -

endsWith('ate') - returns true / false according to original string and suffix given

S1.substring(begin index) -

S1.substring(begin idex, end index) -

S1.trim() - removes the spaces before, after the string.

S1.indent(n) - gives n spaces

S1.split(delimeter) - seperates with that delimeter and returns as array. Eg: "entertainment".split('e') - [, nt, rtainm, nt]

S1.split(delimeter, limit) - seperates with only the limit no. of parts.

S1.concat(S2) - add S2 with S1 [does not store anywhere]

(+) operator

System.out.println(S1+(1+2)) // O/P: One String3

System.out.println(2+3+S1+true+null) // O/P: 5Stringtruenull

↦ Everything after one string value is considered as string.

S1.length() - returns length of the string

S1.replace(Oldchar, new-char) - replace old with new value.

S1.contains(substring)) - checks whether the substring is in main string or not.

join - static method, joins with the delimeter.

Eg: String.join("@", "hello", "world", "bye");

isBlank() - returns true when the string is empty or it as only white spaces.

isEmpty() - returns true only it is empty, returns false even it as white spaces.

S1.getBytes() - converts string array to bytes array

S1.toCharArray() - return as array for all the char in string.

Sl.getChars(Begin, end, char_array, arrays_begin_index)

.intern() - checks whether the given string is already in string pool, if it as it just assigns the reference to that automatically.

System.gc() - garbage collector - clears the extra memory created unwantedly by the program.

STRING BUILDER AND STRING BUFFER:

* Classes in java.lang
* These are used to store mutable string.
* string buffer is thread safe, builder is not.
* thread safe - accessing of a same with multiple person are carried on safely (sequentially).
* string builder follows multi threading (parallel processes).
* string buffer - more memory, takes more time.
* In all java code java.lang is imported defaultly.
* Constructors can also be used. (default, parameterized, capacity)
* Default capacity of all constructor is 16 if we need more than that we can expand it with the formula (old * 2 + 2)
* ensureCapacity method is used to give the capacity of the constructor explicitly.
* Sb.getClass().getName() - returns name of the class of obj sb.
* Sb.getPackage() - returns the package where the obj sb is.
* Sb.getClass().getPackage().getName() -

METHODS:

(i) obj-name.insert(index, element) - inserts the value in that index

(ii) obj-name.setCharAt(index, element) - changes that value in that index (Overwrite)

Expt.No.....

Page No.....
Date .....

(iii) obj_name.reverse(delete (start, end) - deletes the string in that range.

(iv) obj_name.deleteCharAt(index) - deletes the char at that index.

(v) obj_name.reverse() - reverse the characters in the string.

(vi) length() -

(vii) capacity() -

(viii) append() -

To convert string to string buffer - string = new StringBuffer();

To convert string Buffer to string - toString()

1) Swap 2 strings without using 3rd variable.

Code:
```
str1 = str1 + str2;
str2 = str1.substring(0, str1.length() - str2.length());
str1 = str1.substring(str2.length());
```