10/03/2021 10→22/03

Programming using Java

Programming constructs:-

1. sequential - line by line set of stmts
b. Selectional construct - Decision making stmts
c. Looping & Iteration stmts.

Java OO concepts:-	Relationship
1. class & object	1. Inheritance
2. Encapsulation	2. Aggregation
3. Abstraction	3. Association
4. Inheritance (connecting objects)	
5. Polymorphism	

Enterprise Application - Hosted on Internet

Non functional Requirement - Techinal Architects

" " - Develop using technology by system engineers

→ Availability, security, reliability

ex:-
```
class welcome {
    Public static void main (String[] args){
        System.out.Println ("Hello");
          ↓       ↓
        Class   Method
    }
}
```

Compiler → Compiles for whole program

Interpreter → Step by step compilation

→ Java is both

→ Source code ⇒ Complied ⇒ byte code

byte code ⇒ Interpreter reads ⇒ Machine code line by line

Source code ⇒ welcome.java

Compiler ⇒ javac welcome.java

byte code ⇒ welcome.class

Interpreter ⇒ java welcome (or) java welcome.class

$\boxed{\text{syso + ctrl + spacebar} \Rightarrow \text{system.out.println}}$

JVM → interpreter

→ Virtual machine - system software

<u>Byte code verifier</u>:-

→ JVM puts code to byte code verifier

→ checks changing memory address, violation of access right on objects.

<u>Jit compiler</u> ⇒ Compiler optimization

→ when reapeated method calls, it help to convert byte code to native code

→ Same code cannot be translated again, it just copys the machine code.

→ Fast execution of program.

⊗ Architecture of Java → JVM structure

Robust – strong data type checking

eg:- int num1;
    num1=4;
    - Memory management

Keywords:-
→ reserved word
→ have their own meaning

Variables:-
→ Memory location
→ Camel casing notation (start with lower)
→ internal word should be uppercase
eg:- totalAmount

identifier → name to a variable

Data Types:-
→ Primitive
→ Non-primitive

Char : 1 byte in C
2 byte in Java
default value - \u
Unicode Representati

Internationalisation
→ Country's lang

Operators:-
→ Unary
→ Binary

Logical operators – && || – short circuit operators
&& - if the first expression is false, will not execute

|| - if " " " is true

Bitwise :-

OR :-

→ if either of bit 1, ⇒ 1

AND :-

→ if both are 1 ⇒ 1

XOR :- (^)

(1, 0), (0, 1) ⇒ true

Bitwise Complement (~) :-

0 ⇒ 1, 1 ⇒ 0

⊗ 2's complement :-

→ Find lsb of 1	eg:- 100100
	step1:- xxx<u>1</u>00
→ Flip left	step2:- 011100 //

⊗ shift Operators :-

→ left shift

→ Right shift

The left-shift and right-shift by 1 are equivalent to the product of first term and second term

a = 5 → 0000 0101 << 2 ⇒ 0001 0100 → 20

a << 2 :

5 * (2 ^ 2)

↳ left swift

5 * (4) = 20

32 16 8 4 2 1

1 0 1 0 0

$1 << 3 = 1 * pow(2, 3)$

$1 >> 3 = 1 / pow(2, 3)$

Right shift :-

a = 5

5 >> 2 :-

5 / (2^2) ⇒ 5/4 ⇒ 1

0000 0101 ⇒ 0000 0001

↳ >> 2

signed Right shift => >>
unsigned Right shift => >>>

>> preserve sign bit , >>> not preserve sign bit
both (+ & -)

eg: a=10
↳ 0000 1010   1 4 2 1
↳ a>>>1 0000 0101
↳ 5 //

a=10
= 10 / (2*1)
= 10/2 = 5 //

Selection Control Statement: ⓧ → Hand-on

→ if, else

——— × ———

11/03/2021   **Iteration Control Structure**

```
while (<condition>){
      <statement>;
}
```
=> Entry Controlled

Input in Java:-
→ io package - 4 lines - to accept one data
→ Scanner class - java.util.Scanner

```
do{
   <st>;
 } while (<condition>);
```
➡ Display a statement
Then checks condition
=> Menu driven

```
for (<initial>; <condition>; <+/->){
   <statement>;
}
```

Nested- Loop:-

→ for-while
→ for-for
→ while-for
→ do-while-for
→ do-while-while
→ do-while-do-while
→ for-do-while

Break:- Search Scenario - key value - break

Break / continue ⇒ cannot be used without loop.

only be used in any loop

```
eg:-  for ( )
      {
         if(condition)
            { break }
      }
```

break / continue cannot be used only in if...

Continue:-

↓objective → for (while / do-while) it skips

→ in For loop

→ It is used only in loop.

Object Oriented Programming

Structured Programming

→ Mainly functionality

→ does not constract data

(used only for few customers)

↓ draback

OOP => Functionality & also contract data
=> can be used for multiple input data
=> deals with real world
=> organizing & maintainance is easy.
=> helps to break code
=> smaller stable subsystem can be created
=> Integrate & reuse classes.

Class:-
↳ state (attribute / variables)
↳ Functionally (behaviour/ method)
=> It is a blue print / template
=> It gives a structure

Object → allocates memory for all attributes
→ is an instance of class
→ real world entity.

$\boxed{\text{Class have many objects}}$

*. In C only modularity is there but not OOP.
=> class is a abstract / user defined data type.

Access Specifiers - hide / expose the member of a class
Public - access anywhere, violating encapsulating
Private - data hiding, access it only inside a class
Private - used for data members
public - used for method

Private method → cannot invoke through objects

→ " by public method

$\boxed{\text{Creating object for a class using new}}$

new ⇒ dynamic memory allocation

↳ stores value in Heap memory

Book myBook = new Book();

↓

stack memory

$\boxed{\text{How objects are there}}$ ⇒ $\textcircled{x}$ objective

Methods :-

Pass by value ⇒ both actual / formal separate memory loc.

" " refference ⇒ " " " has same memory loc.

Constructor :-

→ Special Method - name same as class name

→ Only Access Specifier (no return or void)

→ are automatically invoked when obj are created

Types :- → Default ⟶ Implicit default

↳ Explicit "

→ Parameterized

→ To initialize the instance variable

This keyword:-

→ access instance variable

→ if instance variable & formal arguments variable name same

→ Implicit refference to current object.

→ this () – to invoke another constructor
 – must be the first stmt.

12|3|2021 **Memory Management** ⊗ → objective

when an object is no longer referred by any reference variable, then those objects are eligible for garbage collection.

$\boxed{\text{How many obj \& reference ? } \otimes}$

Encapsulation:-

⇒ Set methods are used to set the private variable.

Abstraction:-

⇒ hide internal details.

$\boxed{\text{Class Diagram} \Rightarrow \otimes}$ ⇒ Assesment

□ → Red → Private
O → Green → public

Array:-

→ Collection of homogeneous data elements.

→ Fixed size.

→ one array which contains multiple data.

→ can access by index.

→ Elements are stored in contiguous memory location.

```
int a[] = {1,2,3};
int a[] = new int [5];

for (i=0; i<a.length; i++)
```

length → built in property.

String:-

⊗ equalsIgnoreCase ⇒ equals without case sensitive

↳ Hands-on

15/03/2021

Debugging

Debugging: To understand the flow of Program execution

∴ No syntax errors.

Code Analysis: → Analysis source code without execution

→ To improve code quality

→ Coding standards

PMD - Programmer's Mistake detector - Only for Java

Sonarqube - Code analysis - Agile Deops - Multiple languages

[TTD - Test Driven Development - Techinal Best Practices

- use Junit

1, Write code
2, Test case
3. Unit Test

In TTD

-> Write Test case - Test input
-> Run " " - Fail
-> Write code
-> Run Test case - Pass] Not imp

Hands-on

(x) Static :-

-> static varaible -> class variables

-> Memory will be once made and that be used for other variable

=> static variable
=> " block
=> method

-> static variable is access by class name.

| instant method & static method | (x) objective => difference

| Auto generation of Ids in static | (x) Hands-on

Get Counter () => return no. of Customer = (x) Hands-o

Relationship :-

→ Aggregation ⎤
→ Association ⎥ ⊗ Hands-on
Hands-on ⊗ → Inheritance ⎦

Association :-

→ Loosely coupled relation - no dependent
→ Independ on each other

——x——————————x——

16/03/2021

⊗ **Aggregation :-** (———◇)
Hand-on

```
class Customer{
        Private Address address; //Aggregation
    }

class Address {
    }
```

⊗ objective

In-heritance :- (———◆) (———△)

- Use (extends)
- child class does not inherit non-public method
- " " " " " constructor

this ()	Super()
same class	Inheritance

this. → To resolve instance variable hiding

this() → to invoke constructor

Polymorphism :-

one method name → multiple action

→ Static → Method overloading /Compile time

→ Dynamic → Run time / Method overriding

Static :-

⇒ same method name, differnt operation, no. of parameter

⇒ we don't consider return type.

⇒

Method Overloading ⊗ objective

———— x ————

DAY-6

17/03/2021

Compile time polymorphism - Early Binding

Run time polymorphism - dynamic - Late Binding

eg:- Method overriding

⇒ Both Return type & Parameter should be same.

⇒ If Base is public child also should be public

⇒ Private is Base is not accessible.

super. var → To invoke base class variable (same name)

super() → to invoke contructor

super. method() → to " method

objective ⊗ static poly	dynamic poly
→ Same class	→ two different class (Inheritance)
→ No return type	Return type

Object class :-

→ equals() - checks value & memory location

→ hashcode()

→ tostring()

object obj :-

obj = (String) new String("John");

——— × ———

18/03/2021 DAY - 7

Abstract :- - In compelete

⊗ Hands-on - It can have multiple abstract

- Non abstract method – concrete class

- Abstract class should be inheritance.

- Abstract method should be overriding.

Final:- variable

→ It should be in Caps. (Code ethics)

→ Can also be static

→ Final class is cannot inherited.

⊗ Hands-on

Interface:- _ methods & data members

methods – by default → public & abstract

attributes / data members → by default –/ public static final

variables are static in interface b'cos:-

Interface cannot be instantiated – cannot create objects

– can create reference for interface

1, without objects we can access – static

2. All classes using interface – public

3. cannot modify the values – final

⊗ Packages:-

objective

⊗ Exception

DAY-8

19/03/2021

Unit Testing Not $\otimes$ for Hands-on /obj

Junit - Automate Test cases

assert() , assert Equals() => Methods

@Test Suite -> calls all test cases at shot

Regular Expression:- $\otimes$ objective

Customer|? → Quantifiers
↓
Meta characters

Map- deal with large data
(key, value) => key must be unique

$\boxed{\text{Inteface, class or Method can be generic}}$

Array List => objective

Test Case & Test Methods $\otimes$ objective

22/3/2021 DAY-9

List is an interface

Array	LinkedList
* fixed size	no fixed size (dynamic)
* same data of collection	Need not be contigous
* insert & delete is difficult	Adjusting pointers
* wastage of memory	No memory wastage
* Search operation (Based on index)	Traversal from beginning (from head node)

Empty List returns $\boxed{-1}$ LinkedList ⊛ Objective

Set :-

→ Set remove duplicate value

Hash set → Output is unordered

→ Set is interface

$\boxed{\text{LinkedHash set \& Tree set}}$ ⊛ objective

- Tree set $\boxed{\text{compareTo()}}$ uses. ⇒ detecting duplico

$\boxed{\text{null = 32 ASCII value}}$ ⇒ comes first

$\boxed{\text{("null") is different as null}}$

For user defined List equals & Hashcode should be defined.

Map -> used for large data
-> used for searching through key

key must be unique, value can be same

null value can be added to Map

Queue :-

Queue -> insert at rear delete at front

Dequeue -> insert & delete at both

-> Queue is a ordered list

Hands-on

-> Relationships
-> static -Id generation

———— X ————