- Recognise known patterns and built upon them

Day5 -

12/09/2022

- In java 1 & 0 will not work on boolean type

- True = ∞ loop , false = unreachable

Keep

Module 1

- Section 1 - Basics of Java programming
- Section 2 - O.S.
- Exam = 60 marks - 20m - mcq , 40m - Lab [20 Sept]
  O.S + Lab [28 Sept]

Sec1 - Basics of Java Programming

- Java is general purpose programming language
- Dev by Sun microsystem, U.S., 1991 (James Gosling)
- Green team : Green project
- Java name based on Java island
- OOPs →
- Java is Programming language + Platform

- Java as a platform - It can be H/w / s/w

↳ Platform helps to run programs (HW + SW)

↳ It provides 1) JRE - Java Runtime Env

2) API - Appln Programming Interface

- JDK - 1st version - 1996, Jan

* - Applications of Java

(1) ↳ Desktop/standalone application : VLC, quickheal

(2) ↳ web application - Database is stored on 1 system i.e. on server & accessed by client

Eg: Google, youtube

(3) ↳ Enterprise Application - Company based Eg: SBI

↳ Distributed application - multiple users of Enterprise

Java helps here

↳ used for load Balancing

↳ security challenges

(4) ↳ Mobile Application (J2ME edition)

Eg: Games

- Java Platform / Edn

1. J2SE - Standard Edn : students

↳ standard desktop

2. J2EE : Enterprise Editions
   - Heavy duty server system - multiple-tier
   - component based progr. - Advanced java

3. J2ME - Micro-Edition
   - Small & memory constrained devices
   - mobile applications

4. JavaFX
   - used to develop rich internet applications.
   - make use of API

Java versions

- JDK 1.0 : 1996
- JDK 1.4 : 2002 - Merlin
- JDK 1.5 : 2004 - Tiger
- JDK 1.8 : 2014 - SE

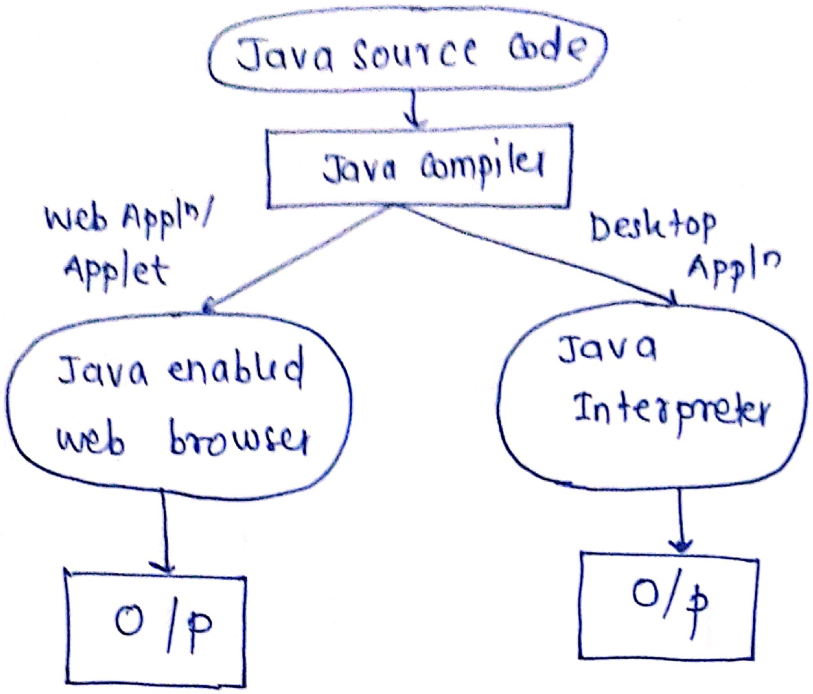
Int.Q. → Features

[Q. C++ & Java utilities & diff (lambda, Annotation)

C++	Java
1) Platform dependent (O.S)	1) Platform independent (O.S)
2) mainly focused on system programming	2) various application can be designed
3) Multiple inheritance	3) Dont have multiple inheritance
4) goto statement	4) don't support goto
5) Pointer are used	5) Pointer are not used
6) operator overloading	6) Don't have operator overloading
7) Compiler	7) compiler & Interpreter

H.W.Q.
↓

Way of using java program



- main() : initial point of java program

↳ public static void main (String args[])

- public → Access modifier
- static → (method modifier)
- void → Return type
- String args[] → Parameter Data type

(public - so main method can be called anywhere

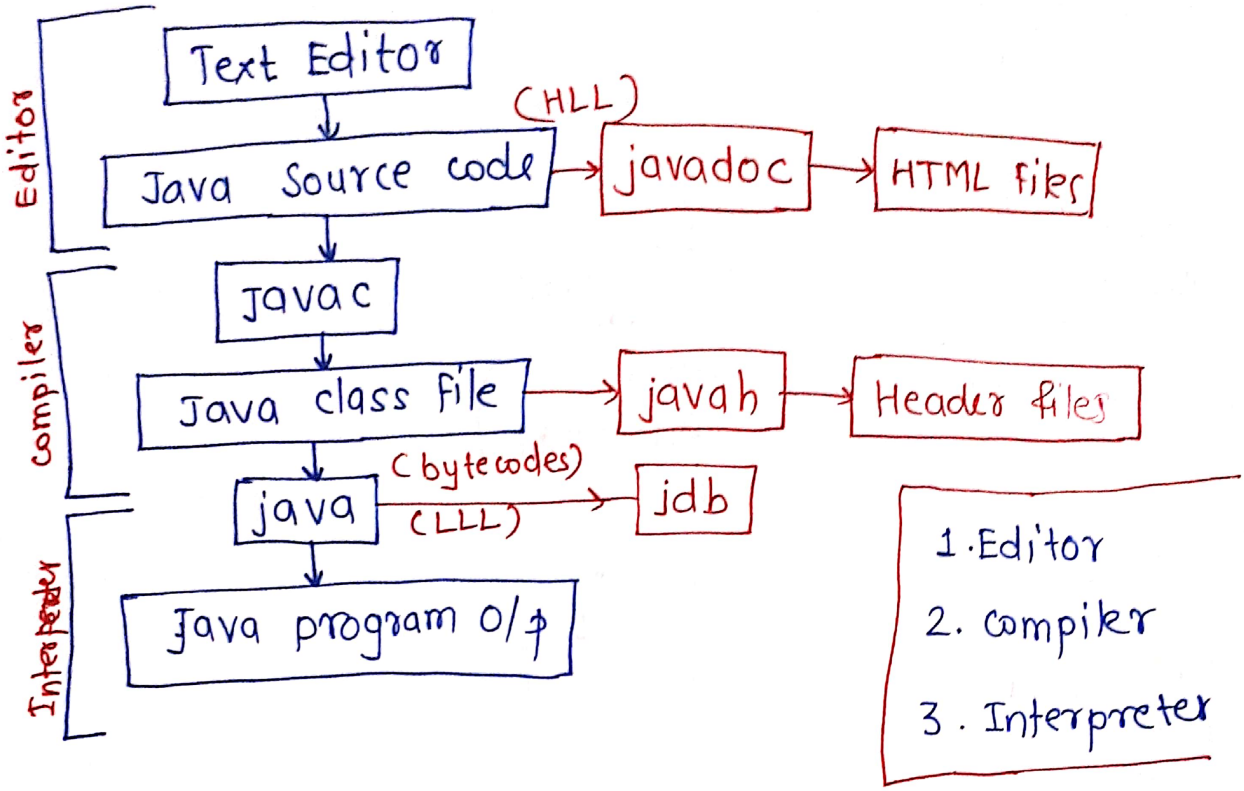
to call ↓

↳ JVM (Java virtual m/c)

↳ System.out.println ("Hi")

- System → Java class name → in lang. package
- out → Reference variable → print class & io package
- println → method name

Process of building & running Java applicatn Programs



2. compiler

* JIT compiler (just in time) *

Int Q

↳ to improve
↳ to reduce the amount of time needed for compilation

– will translate HLL → LLL into bytecode

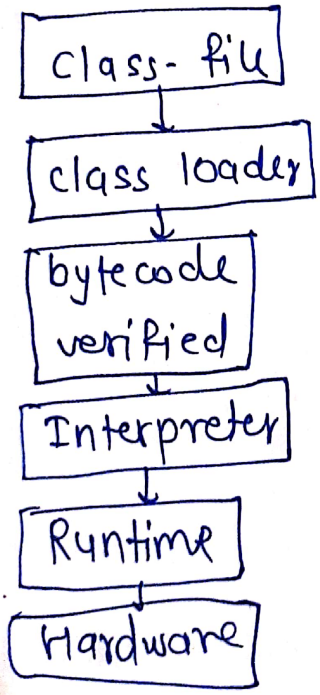
3. Interpreter

– LLL in bytecode converted to machine language

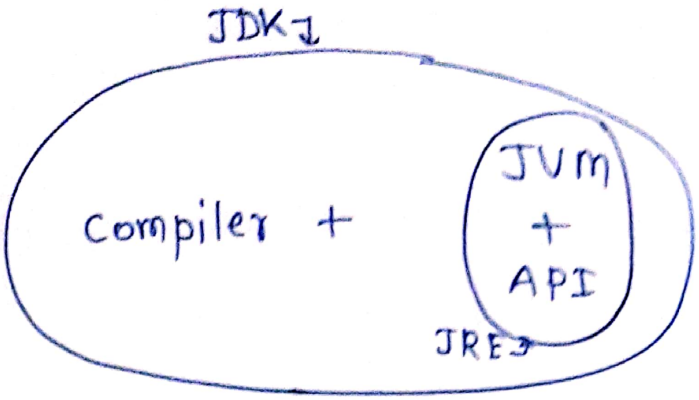
– JVM - Place where HLL → LLL → ML (Hybrid LL)
(java virtual m/c)

JVM = HLL $\xrightarrow[\text{byte codes}]{}$ LLL $\longrightarrow$ HLL

- Bytecode improves running time ← (Q. Int) *
  - 1) Generated after compilation
  - 2) Generated in .class file
  - 3) Platform independent
  - 4) Browser will load our bytecode much faster
  - 5) Source code is not revealed to users (Data hiding)
  - 6) Security checks performed by Interpreter
  - 7) Screens out malicious code



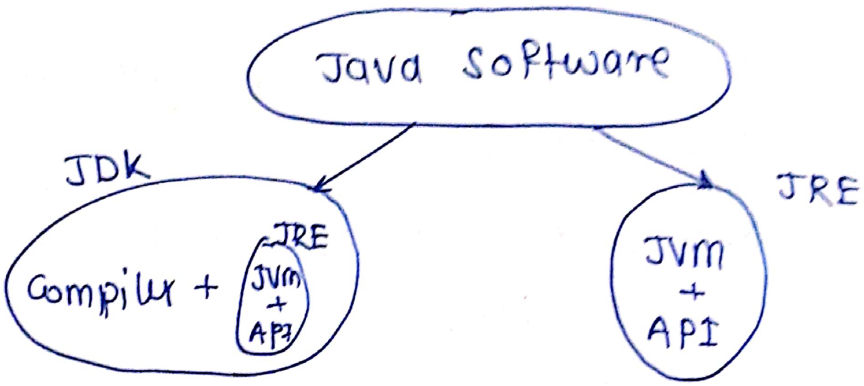
- In one class file multiple programs can be executed



JRE - Runtime env
JDK
JVM - virtual m/c

*
Q.Int

- User will only require JRE



- JVM - Abstract machine
  - Runtime env
  - h/w & s/w required
  - follows steps - Load code
    Verifies code
    Execute code
    - Runtime env

* JDK tool
  - javac - compiler ; javadoc - HTML
  - java - interpreter ; javah - header file
  - jdb - java debugger ; appletviewer

- API - Application programming Interface

↳ Inbuilt packages - 1. Language packages
2. Utility packages
3. Input/output packages
4. Networking
5. AWT (Abstract window toolkit)

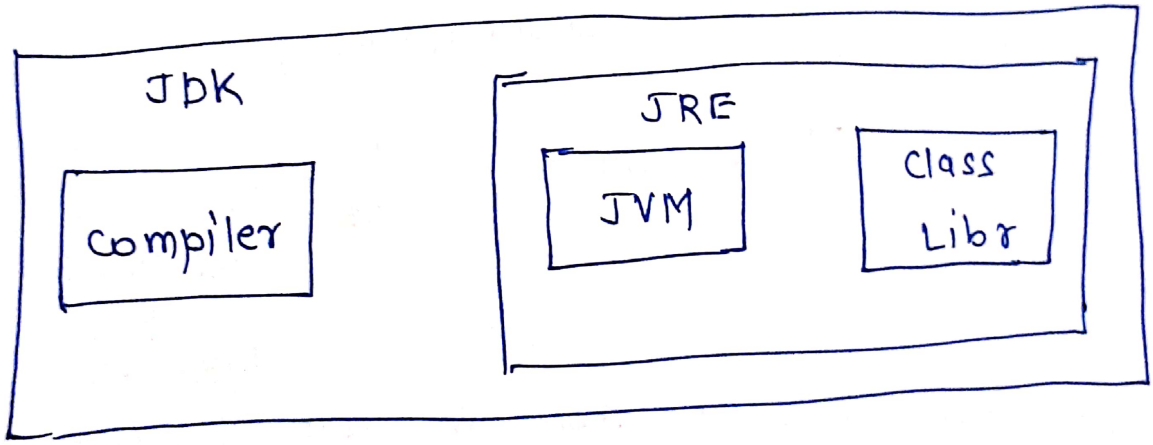
They provide readymade classes & functions

↳ Every package have different classes & functions

(Ref: Java complete guide)

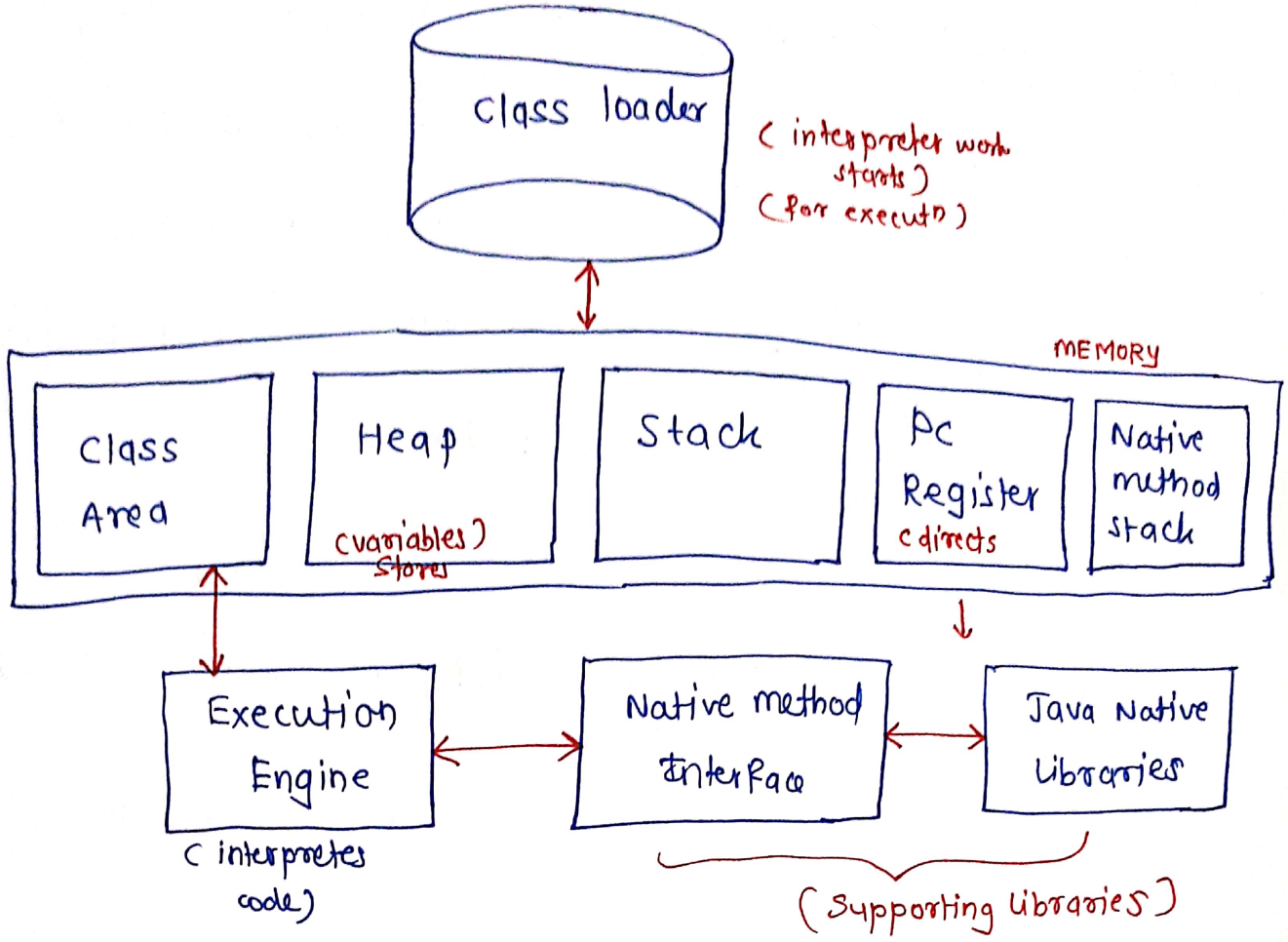
Day 6

13/09/22



- Class loader

↳ To load it into memory



* [Why Java is platform Independent ? Q. Int Que

↳ Source code → java compiler → Bytecode → Any OS
.java .class
(Universality of Bytecode)

↳ In other languages there are diff interpreter acc to diff platforms but in java universal interpreter. (JRE will be supporting to every OS)

↳ In java bytecodes has universal interpreter functionality.

↳ It is portable

Features of Java *

Int Q

- Simple
- object oriented
- Platform independant
- secure language
- Dynamic
- Architectural neutral
- Interpreted language
- multithreaded —u—
- Distributed language

1) Simple – It does not have any complex feature
– X Eg: Operator overloading, pointer, multiply inheri etc

2) Object Oriented – combination of diff. types of object
– object - Info about data & behaviour
– 4 pillars of OOPs Int Q *
i. Abstraction - Data hiding + encapsulatn
ii. Encapsulatn – Binding data members & fn to a class
?iii. inheritance – Transfer of object/class in addn of their existing properties to other class
Eg: camera & camera of phone
iv. Polymorphism -

3) Secure – Dont have pointers

4) Robust - Strong memory mngmnt
– Automatic garbage collection

5) Architectural neutral

- no implement dependant features

6) Interpreted language

- JRE, JVM fn

7) Multithreated language

- Thread → Task is one activity - 1 action
- Parallel thread - multithreading

8) Distributed language

- RMI (Remote method Invocation)
- EJB (Enterprise Java Beans)
- CORBA

Hw. Q.1. Study JVM architecture

Q.2. Understand the r/s betn JDK, JRE & JVM

Q.3. Study the Featurs of OOPs / Java.

Q.4. study & prepare real life examply of oops pillars.

Java tokens

- Token is small units.
- Java program - collection of tokens

Java tokens

1. Reserved words
   - keywords - lowercase
   - having sp. meaning

[keywords are always in small case

2. Identifiers
   - Tokens
   - names for classes, methods, variable, package
   - char. set : A-Z, a-z, 0-9,

3. Literals
   - constant values
   - Integers, float, character, string, boolean

→ int x = 23; // decimal values
int y = 0x3ef // hexadecimal
int z = 0771 // octal

4. Operators
   - - , + , =

5. Seperators
   - ( ) { } [ ] , . &

Variables

int x; // defined variable , int x = 10

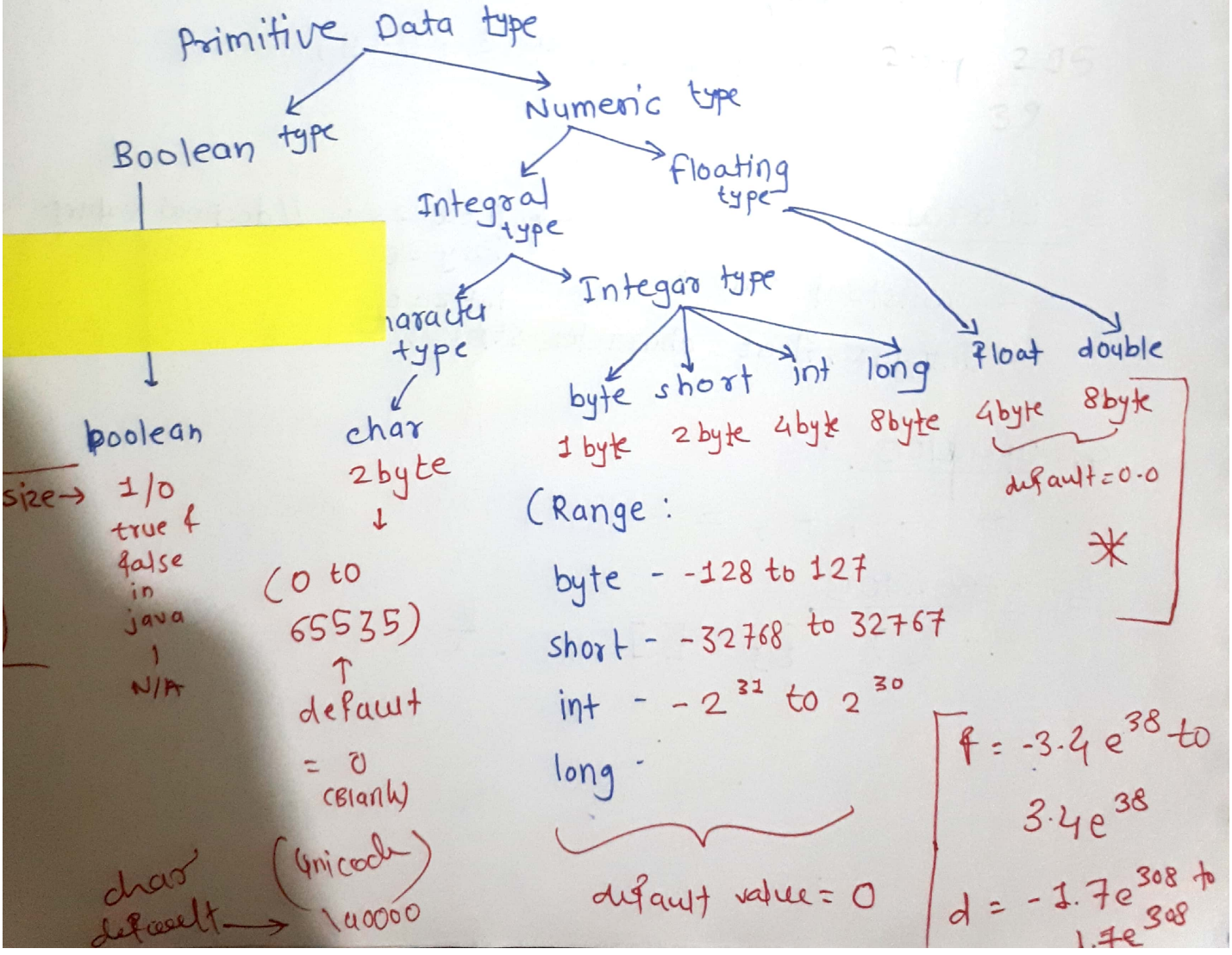
x = 10; // Initialised —||— , double f = 0.333;

char c = 'a' ; String s = "abcd"

Data types

1. Primitive Data type
   - int, float, double, byte, char

2. Non-primitive data type
   - class, array, interface



- Wrapper classes

HW. Q. chart -

Name of data type	size	range	default value	wrapper class

Day 7

15/09/22

- In long data type `long g = ~~~ L`

↑ this L need to be written

- single float - 5 values after decimal

- double float - multiple values are printed (upto 14)

- Typecasting
  - Data type casting converts 1 data type to another data type
  - only possible in primitive data type

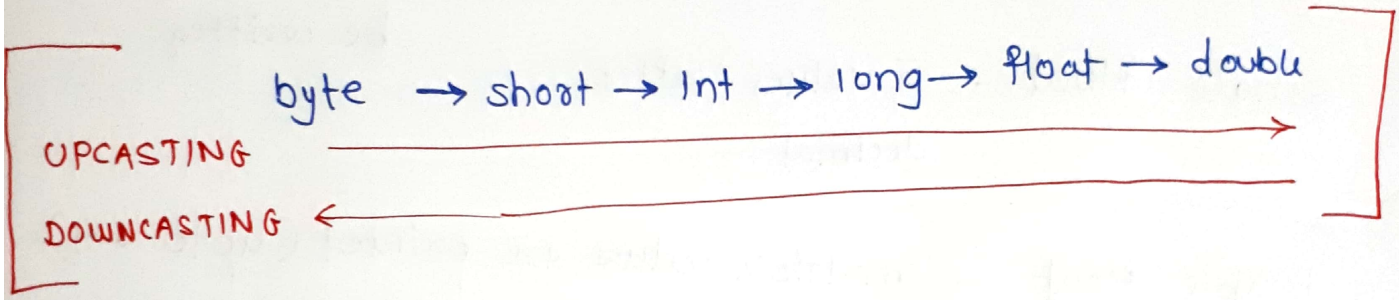
Two types (1) Upcasting - small capacity data type to larger capacity data type

Ex:

```
double dl ; // 8 bytes
int i = 100 ; // 4 bytes
dl = i ; dl ⇒ int
SOP(dl) // 100.0
```

② Downcasting - Bigger size data type into smaller size data type

Eg:
```
double dl = 100.0 ; // 8 bytes
int i ;           // 4 byte
i = dl ;          // int = double
Sop(dl)           // 100.0
Sop(i)            // 100
```
↖ we may loose some values



- In upcasting - No need to mention datatype
- In downcasting - need to mention datatype
  - if size doesnot fits it does upcasting automatically & tries.

Ex:
```
float i ;
double dl = 100.98456 ;
i = (float) dl ; // downcasting
S.O.pln (dl) ;
S.O. pln (i) ;
```

char Datatype

- ASCII values varied as per countries
- So Unicode system – '\u0000' ← default value

Lowest value - \u0000
Highest value - \uFFFF

- char is primitive & String is non-primitive
- ' ' for single character, " " for string
- Array is supposed to starts from 0. args[0]

Command line arguments

↳ Arguments that are passed in cmd prompt

Ex:
```
class {
    Pu          args[]){
        S.O.pln ("Name = " + args[0]);
        ————(" Last name = " + args[1]);
```

(providing i/p in cmd line)

Wrapper class - converts 1 data type 2 another data type

Ex: typecasting using wrapper class

```
String S1 = args[0];
String S2 = args[1];
{ int i = Integer.parseInt(S1);
  int j = ————————(S2); } any class can be used
  int sum = i + j
```

Hw.

Q. Study diff. types of ~~unary~~ operators

Unitary - ++i , i++

Arithmetic - + , - * , / , %

Logical - && , || , !

Assignment - = , += , -= , *= , /= , %=

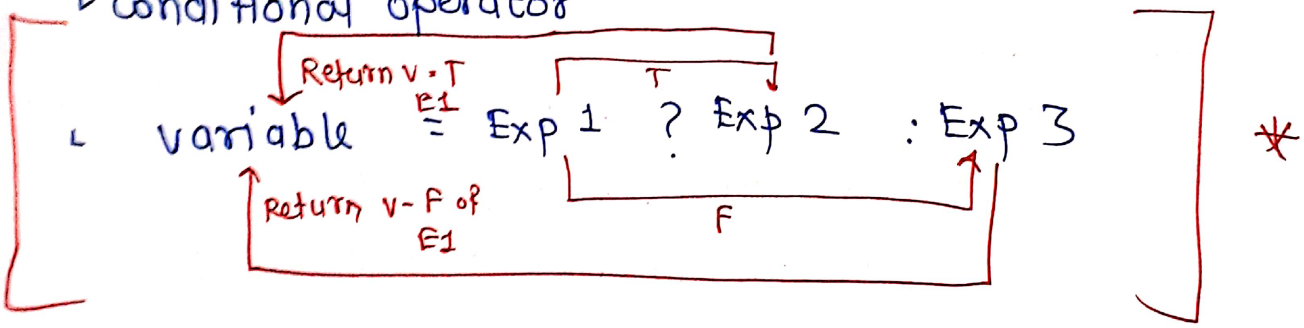
Relational - < > <= >= == !=

Bitwise - << >> , <<< , >>>

Ternary : ?:

Ternary operator

↳ conditional operator



*

variable = Exp 1 ? Exp 2 : Exp 3

```
if (Exp 1)
 {
    variable = Exp2
 }
else
 {
    variable = Exp 3
 }
```

Program →

```
class {
publi {
    int n1 = 10, n2 = 20;
    int max = (n1 > n2) ? (n1) ; (n2) ;
    S.O.pln ("max = " + max);
```

This expression can vary

Q. odd / Even

based on this →

```
String s = (i % 2 == 0) ? "Even" ; "odd" ]
    S.O.pln (s)
```

Q. Int

↳ single line can be used

↳ optimisation

§ optimisation

Day 8

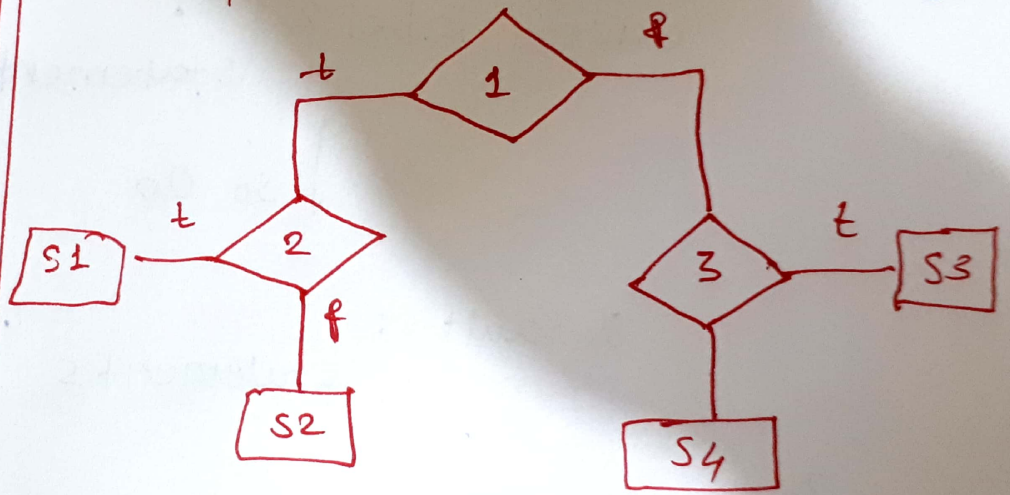
16/09/2022

- Switch case

[refer - Reference book, g4g

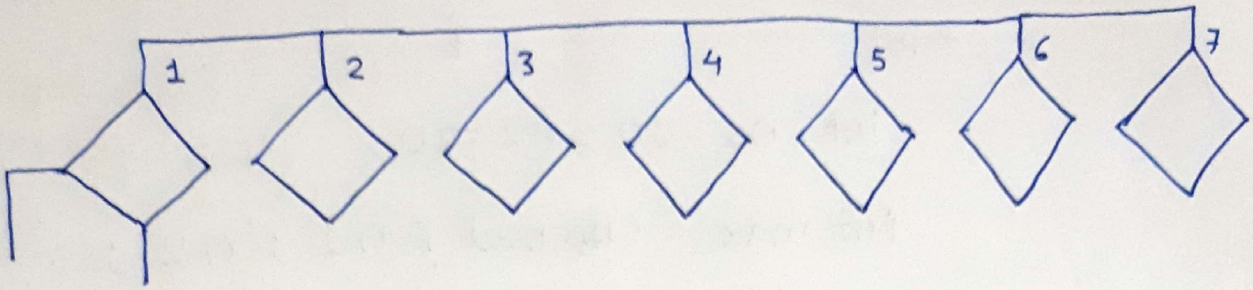
Nesting example

```
if (cond)
{   // True
    if (cond)
    {
S1      // True
    }
    else
    {
S2      // false
    }
}
else (cond)
{   // false
S3  if { // true }
S4  else { true }
}
```



- multiple nesting conditions can be generated

- Ex: Switch case example - 7 days & different conditions



- switch case method reduces complexity & ehnances readability & optimisation of ~~the~~ a program.
(Ps: Depends on the need of a program)

- Only 'if' part is possible but only 'else' part can not be executed.

Syntax:

```
switch (x) {
        case label 1 :
                statements ;

        case label 2 :
                statements ;

                ↓ so on

        default : statements ;
}
```

Label namewise datatype shall be written here
Eg: (char | byte | short | int)
↳ Expression ✓

← Else part

- Any primitive data type can be i/p here (except : string)

Ex:

```
class
Public
int day = 3 ;          // This can be user defined with scanner ph
String DayName ;
Switch (day) {
    case 1 :
        DayName = "Today is monday." ;
        break ;
    case 2 :
        DayName = "Tue" ;
        break ;
    case 3 :
        DayName = "wed" ;
        break ;
    .
    .
    .
    case 7 :
    default :
        DayName = "Invalid Day" ;
}
S.O.Pln ("Day is " + DayName);
}
}
```

Breaking statement after every case *

H.W.

Q.1. write same program with SOP statements

Q.2. write a program to identify vowels

Q.3. Work with enum week dataset

Q.4. Input with Int, char, string

1 — 7

m — Ss

monday — satur Sur

> 85 : Dis

60 to 85 - First

50-60 - 2nd

40-50 - 3rd

Class & object

Class - Group of objects, which have common practices

Object - It is real world entity

Data members - common characters of class

Task/Activity - Functionality / method

Ex: Football player

Data members - Height, Experience, Position, jersey no, club, Age

Method - Running speed, contribution in game, strike rate

Class is → Template / Blueprint → class : default / Public (Access Specifier)

→ Logical entity

Object - It is runtime entity

Q. Int

└ It has 3 characteristics

1. state 2. Behaviour 3. Identity

- An object is an instance of class ? why

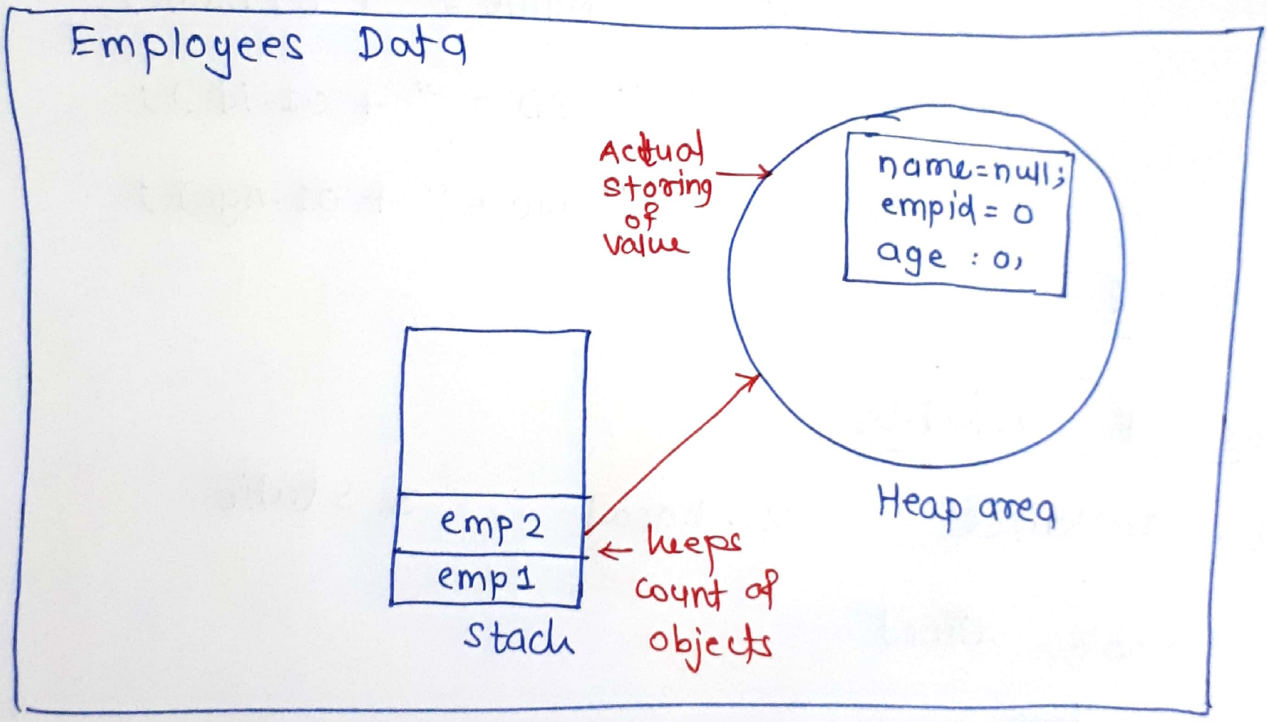
Heap memory

└ Area of memory where values are stored

Stack

└ when we create object it is stored in stack

Ex:



HW.

①. WAP for building

Eg:

```
Class obj1 {

    String name;
    int id;
    int age;

    public ____ {
        // reference obj created
        Obj1 o1 = new Obj1();

        o1.name = "Harshal";
        o1.id = 012;
        o1.age = 26;

        S.o.pln ("Name = " + o1.name);
        ____ ("ID = " + o1.id);
        ____ ("Age = " + o1.Age);

    }
```

Instance variable

Local variables are here

Here new obj2 can be added

Assigned value to instance variable

Types of variable

1. Instance
2. Local
3. Static

↓

variables defined in Class

```
        default :
            DayName = "Invalid Day" ;
        }
        S.O.Pln ("Day is " + DayName);
    }
}
```

H.W.

Q.1. write same program with SOP statements

Q.2. write a program to identify vowels

Q.3. Work with enum week dataset

Q.4. Input with Int, char, string ← 1 — 7
m — Ss
Monday — saturday, Sunday

Q. Result evaluation

> 85 : Dis

60 to 85 - First

50-60 - 2nd

40-50 - 3rd

g:

```
Class obj1 {
    String name ;
    int id ;
    int age ;

    public ——— {
        // reference obj created
        Obj1 O1 = new Obj1 ();

        O1.name = "Harshal" ;
        O1.id   = 012 ;
        O1.age  = 26 ;

        S.o.pln ("Name = " + O1.name);
        ——— ("ID = " + O1.id );
        ——— ("Age = " + O1.Age) ;

    }
```

Instance variable

Local variables are here

Here new obj2 can be added

Assigned value to instance variable

es of variable

1. Instance ↓ variables defined in Class
2. Local
3. Static

Day 9

17/09/22

Types of variables

1. Instance variable
   - ↳ Variables defined in class
   - ↳ when Instance is created memory is allocated to variables
   - ↳ Reference object is needed to access diff. instance variables
   - ↳

Access modifiers:
① Public ② Pvt,
③ Protected ④ Default

2. Local variable
   - ↳ scope of variable is within methods we are using (in constructors & block as well)

Ex:

```
int calculate()
{
    int i, j;
    i = 10; j = 20;
}
```

} scope is within this method only → so Local variables

3. Static variable

↳ with declaratn of static, variable becomes static

Ex:

```
Class Employee {
    String Name ;
    int id ;
    static int Deptcode = 111 ;
}
```

Static variable → static → final

- static variable can only be defined / Assigned / initialised once.
- final keyword can be added

Methods

- AKA member function
- Syntax:

```
returntype methodname (parameter list)
{
    // scope of method
}
```

returntype: variable type of return (int, char ... string)
methodname: task/oper?
parameter list: Data values

- method declaration - `void sayHello ( ) ;` ← calling method
- method defination -

```
Void sayHello ( ) ← Defining method
{
    // scope
}
```

- method is place where we mention about op^n / task we are going to perform.

Ex: return data type — Task (camelcase)

```
double calculateDisc()
```

← Naming convn ex.

Ex: (modifier)* Return type — method name — Parameter

```
public String getName (String s)
{
    String name = "CDAC Mumbai";
    name = name + " " + s;
    return name;
}
```

Types of methods

1. Inbuit method → Predefined methods

2. User defined method
   - a. Instance methods / non-static methods
   - b. Static methods

Ex: Predefined methods

```
public class Predef---- {
public s v m {
  s.o.pln ("square root = " + math.sqrt(25));
}
}
```

class — method

↑ Readymade available function in math Library

Ex. Instance method

```
public class Instance {

    public s v m

    void sayHello ()
    {
        s.o.pln ("Hello CDAC !");
    }

    p. s. v. m () {

        Instance method

        Instance i1 = new Instance ();

        i1 sayHello ();
    }
}
```

method defin^n

obj. created

method declaratn / call

Repeatatn of code is dealt by instance method here

Ex: Instance method 2

```
public class Instance 2 {

    void sayHello (string s)
    {
        s.o.pln (s);
    }

    p. s. v. m {

    Instance i2 = new Instance 2 ();

    i1.sayHello ("GM");
    }
```

Parameter

method call

Argument

```
public class Instance3 {

  public static void main (string args[]) {

   Instance3  i1 = new Instance3 ();

    String S1 = i1.sayHello ( "Gm");

     s.o.pln (s1);
     s.o.pln (i1.sayHello ( "Gm"));

  }

   String sayHello (string s)

    {  string str;
       str = s;
    }  return str;
```

Call by value

- class Test1{

```
    int addition (int x, int y)
    {  int sum = x + y;
       return sum;
    }

   p.s.v.m    {
     Test1 t1 = new Test1();
     int add = t1.addn (5,5)  ← Call by value / pass by value
        s.o.pln (add);
    }
}
```

Ex :

```
class Test 2 {

    Static int calc (int l)
    {
        return l x l x l ;
    }

    Public static void ——

        calculate (7) ;
        S.O.pln ( " volume of cube = " + calculate (7));

    }
}
```

using static

Eliminating object

- In static method, add static keyword to the method
  - w/o object we can give call to the method
- Can not use non static data member / call non-static methods
- this & super can not be used in static content

Instance method
- Instance methods
- Data member
- static data mem.
- Static methods

Static method
- static data mem
- static methods

Ex:

```
Public class Test5 {

    Static int x = 20;
    Static int change()
    {
        int x = 10;
        return x;
    }
    Public stat v m {
        s.o.pln(x);
        int a = change();
        s.o.pln(a);
        s.o.pln(x);
    }
}
```

O/P

20
10
20

Q.1. Diff planets - class - planet, variable -
object - m, V, E, m, S, U, T, N, P

Q.2 Bank Appln - withrawal, Balance, check, min 1
Bal

Private - Declared class

Default - only for same package

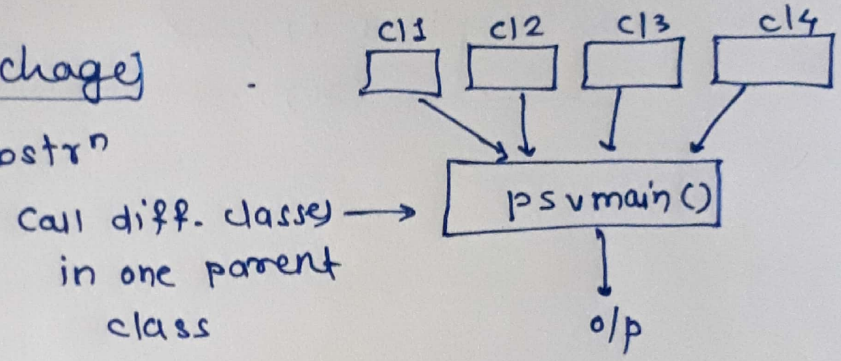
Protected - ——v—— & sub-classes

Day 10 — 18/09/22

Topic : Constructors

- User defined package

↳ useful for abstn



- method overloading

↳ names of method are same

2 types (1) Number of Parameters

```
Class operation1 {
    private int x;
    p.s. int sum (int i, int j)
    { return i+j;
    }
    p.s. int sum (int i, int j, int k)
    { return i+j+k; }
}
```

method overloading

```
    }
class operation2 {
    p.s.v.m ( ) {
        s.o.pln ( operation1.sum(11,22,33));
    }
}
```

② Data type

- compiler check data type of child classes
- selection of class based on data type mentioned in parent class
- In mixed data type → Error

Q.H.W. - W.p. for 4 classes (int + ~~char~~ + float + double + s) & print. (with overloading method)

Ans: 4 classes with diff. data types
main class - print (↑)

Constructors

- Name of constructor should be same as Class name
- It has no return type
- Every progr. must have default constructor
- Purpose of c. is to initialise the value
- multiple constructors can be written → constructor overloading

- Default value can be initialised in default constructor. (refer ss : 11:10) 19/9

- Parameterised constructor

  ↳

- when obj is created constructors gets a call if it gets involed

- No need to call constructor specifically in program.

(ss : 11:33) 19/9

Ex:

- Parameterised constructor

```
student (int s1, string s2, int s3)
{
    id = s1;
    name = s2;
    marks = s3;
}
```

- If we write constructor with parameters as

Student S1 = new student (555, "HD", 91);

then parameterised constructor will be called.

- If we dont pass values then default will be printed.

- this keyword → To refer the particular instance from particular constructor

H.W.

Q. WAP for cricketer details. Add some instance variable & apply the concepts of constructor overloading as follows

- Default constructor - 1
- Parameterised constructor - 2/3
- Apply constructor overloading

Q. WAP for Diet · protein content.

- class - object
- constructors
- Data hiding
- methods
- overloading

Arrays

↳ Array is an indexed collection of similar data elements that has fixed size

- Array is an object
- Defining array - st.1 → Declare array variable
  st.2 → Create an array

Syntax:

st.1 {
<data type> <arrayname> [] ;
or
<data type> [] <arrayname> ;

st.2 { <arrayname> = new <data type> [size] ;

Combining both,

<data type> <arrayname> [] = new <data type> [size] ;

Ex:

int a1[] = new int[5] ;

int* ↑ creation of group

```
for ( int i = 0 ; i<=4 ; i++)        (a1.length can also be used)
{
    s.o.pln ( "Index " + i + " = " + a1[i]);
}
```

Ex : 2 :

```
int [] a1 = {10,20,30,40,50};

for loop

a1[4] = 100;          ← Individual Index value can be modified
SOP ( a1[4]);
```

Ex : 3 : For each loop → Used to display

```
int [] a1 = {10,20,30,40,50};

for (int x : a1 )     ← Enter array name
{                       x =
    System.out.pln(x);
}
```

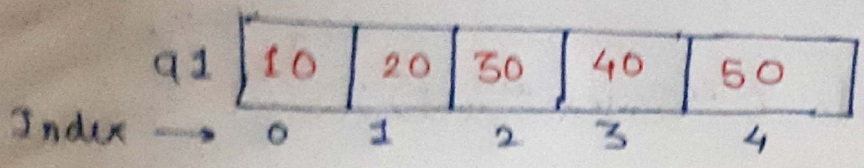
for each loop

```
int a1[] = new int[5];
```

↑ will be stored in main method stack as a reference

↑ Elements are stored in heap. Hashcode created at runtime in heap.

Structure →



a1[0] = 10
a1[1] = 20
a1[2] = 30
a1[3] = 40
a1[4] = 50
} values

- Index by default starts from 0
  ↳ Used to access the values

Size
int : 4 bytes
size = 5

ml = 20 bytes

- Array over flow - Array Index Out of Bound Exceptn Error

- Ex: int a1 = {0 2, 1 4, 2 6, 3 8, 4 10}; ← Array Initialiser
  S.O.p (a1.length) ← to trace the length
  — " — (a1[2]); = 6

- 1D array - It can be Horizontal & vertical

- multiple arrays can be declared in single line

Ex:
```
int a1[], b1[], c1[];
int a1[], b, c ;
int [] a1, b, c;
```
} Array declaratn

Array creatn

```
a1 = new int[5];
a1[0] = 10;
a1[1] = 20;
a1[2] = 30;
a1[3] = 40;
a1[4] = 50;
```

Ex: Using Scanner

```
import
class
public s v m

Scanner sc = new Scanner(System.in);
int a1[]= new int[5];
                          /a1.length
  for (int i=0 ; i<5 ; i++)
  {
     s.o.p ("Enter Array Element "+i);
      a1[i] = sc.nextInt() ;
  }

  for (int x:a1)
  {
       s.o.pln(x);
  }
```

Ex: Character Array

```
chart a1[] = {'a', 'u', 'e', 'j', 't'}
{ int//char a1 = new char[5];

   for (int i=0 ; i<5 ; i++)
//{   s.o.p ("Array");
//} // a1[i] = sc.next();

for (int x:ar)

{  s o p (x);
}
```

- Refer code in laptop

Array char.java

- char a1[] = { } ← initialisation

char a1[] = new char[] { } ← specific memory allocatn is done.

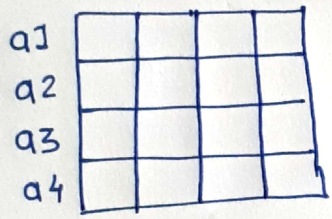
↳ while we initialize, size is automatically considered

2D Array

- int a1[][] = new int [5][5];

<data type><array name>[][] = new <d type>[row][col];

Ex:



← when 1D arrays are combined it form 2D array

Row → No. of arrays in the 2D array

Col → No of elements present in each array

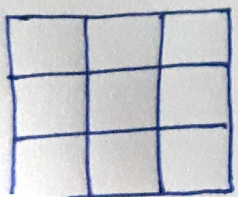
Ref: SS @ 14:08 19/09

Ex:

```
int a[] = {1,2,3};
int b[] = {1,2,3};
int c[] = {1,2,3};

int a1[][] = new int [3][];
a1[0] = a ;   // i.e. 1,2,3
a1[1] = b ;   // i.e. 1,2,3
a1[2] = c ;   // i.e. 1,2,3
```

Defining row is imp



Ref : ~~java~~ Array 2D.jav (MOD 1)

```
for (int i = 0 ; i < a1.length ; i++)
    {
        for (int j = 0 ; j < a2.length ; j++)
        {
            S.O.p (" "+ a1[i][j]);
        }
        S.O.pln ()
    }
```

- Jagged array - Diff. number of elements in diff arrays

- Using for each loop for 2D array

```
for (int ~~[]~~ y[] : a1)
    { for (int x[] : a2)
        { SOP
```

?

- CP module

- JVM, JRE
- class, obj
- Java basics
- constructors
- overloading
- OOPs (4)
- coding pillars