29th April, 2020

Operators and simple if

jvm → converts byte code to machine code.

java consists of packages. & packages consists of classes.
ArrayList → import java.util.*
classes consists of variables and methods.

```
class Person {
    int age;
    void eat(){
    }
}
```

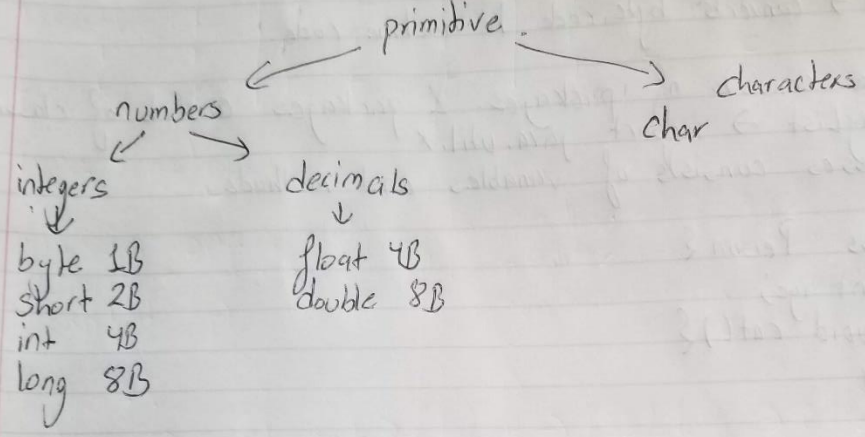
If we want to access the variable (age) or method (eat()) we need to create object (Memory) for the class (Person) if we want to access the class, we need to import the package.

Data types : Classification of information.
(information)

two types :- 1) Primitive
2) Non Primitive

~~Primitive~~

Data Types



Operators & Assignments:

1) increment / decrement:

pre-increment: ++y
post-increment: y++

pre_decrement: --y
post-decrement: y--

Example:

int n=4, y;
y = ++n; n=5 & y=5
y = n++; n=5 & y=4

int n=4;
y = --n; n=3 & y=3
y = n--; n=3 & y=4

```
class Operators {
  public static void main (String args[]){
    int n=4, y;
    y = ++n;
    System.out.println ("n = " + n + " and y= " +y);
  }
}
```

Arithmetic Operators:
(+, -, *, /, %)

```
byte + byte = int
byte + short = int
int + long = long
long + float = float
double + char = double
char + char = int
```

example:
'A' + 'A' /130
↓
ASCII - A(65)
a(97)

Relational Operators:
(<, >, >=, <=)

Equality Operators:
==, !=

```
int a = 20;
a/2 ==10  // both are equal (true)
a != 10   // both are not equal (true)
```

Bitwise Operators:
(&, |, ^)

```
T&T = T
F|F = F
T|F = T
```

4 = 100
5 = 101
4 & 5 = 100 = 4
4 | 5 = 101 = 5
4 ^ 5 = 001 = 1

And			OR			^ (XOR)		
T	T	T	T	T	T	T	T	F
T	F	F	T	F	T	T	F	T
F	T	F	F	T	T	F	T	T
F	F	F	F	F	F	F	F	F

Type casting:

Converting one datatype to another data type
1) Implicit → done by the compiler
2) Explicit → " " " programmer

Implicit:
(1B) byte → (2B) short → (4B) int → (8B) long → (4B) float → (8B) double.

Assignment Operators:
There are 3 types of assignment operators.
1) simple assignment operators. ex: int a = 10
2) chained assignment operators.
   a = b = c = d = 20
3) Compound assignment operators.
   += ; a += 30
   -= ; a -= 2
   %=
   &=
   |=
   ^=

Control statements:

1) Simple if
2) if else
3) nested if else
4) if else ladder

Simple if:

syntax:

```
if (condition)
{
  st-1;
}
```

if else:-

syntax:

```
if (condition)
{
  st-1;
}
else
{
  st-2;
}
```

for method we need object
class " " package.

Output in Java:

println()

object is required to access method.

java.io.*
java.lang.*
java.util.*

java.io.Print

javap -> java package.

javap java.lang.System.

Input in Java:

java.io.InputStream
class InputStream{

}

java.lang.System. ⊖ imported default by JVM.
class System{
static InputStream in;
}

in → method area

System.in (It contains the input object in String format
Scanner: It ~~can do~~ converts the String format object to required format

java.util.Scanner
class Scanner{
int nextInt();
}

```
java.util.Scanner
class Scanner{
    int nextInt();
}

Scanner sc = new Scanner(System.in);
int a;
a = sc.nextInt();
}
```

polymorphism:

poly → many
morphism → forms / states / properties

1) method overloading
2) method overriding

1) method overloading:
method can be written same with ~~same~~ different type of parameter
different no. of parameters or different order.
(in same class)

eg:-
```
int sum(int n, int y)
int sum(int n, int y, int z)
int sum(float n, float y, int z)
```

is also known as compile time
polymorphism as the compiler d
which method needs to be inv
based on the parameters

Java
There are 6 standards in oops
1) Class
2) Object
3) Encapsulation
4) Polymorphism
5) Inheritance
6) Abstraction

Object :-
~~every whatever is~~
exist
properties (
action

eg:- Mango
Apple
It has properties

class :-
doesn't exist

eg :- fruit
It individual doesn't have

Class :- Collection of object

rahul → object (rahul exists physically)
Person → class (Person doesn't ; it's a category)

rahul properties (age, height, weight, color)
action (eat, talk, walk, sleep)

```
class Person{
    int age;
    float height;
    void eat();
}
```

```
Person rahul;
rahul.age=22
```

Rules:-
ClassName :- The first letter should be capital (eg:- (NScanner)
method name :- ex: get()
getName()

variable name:-
1) starts with (A-Z || a-z || _ or $)
After ^ → (0-9)
eg:- 123total = 56; // wrong
eg:- total123 = 56; // correct

2) case sensitive (eg:- int NUMBER, number);
3) Reserved keywords can't be used.
ex. for, while, if, do, break, ... etc
4) Spaces are not allowed between the name

```
class Student{

}

class Person{

}

class Employee{

}

class Test{
    Public static void main (String args[]){
        We can create object for student / Employee / Person
    }
}
```

Static → doesn't depends on object/memory ; depends on cla

In Java we have two kinds of memory:-
Object memory → programmer is responsible
Static memory → predefined memory which can be accessed using the class name.

Method:
a set of instruction written to perform a particular task.
Every method has two properties.

1) return type
2) parameters

```
return_type method (par1, ..., parN){
}
```

example:-

```
sum(){
  int a=10, b=20, c;
  c = a+b
  return c;
}
```

Based on these two properties the methods are classified four types:

1) method without return type & without parameters.
   ex: void sum()
2) method without return type with parameters.
   ex: void sum(int x, int y)
3) method with return type; without parameters.
   ex: int sum()
4) method with return type & with parameters.
   ex int sum(int x, int y)

```
    1
   12
  123
 1234
12345
```

```
for (int i=1 ; i<=5 ; i++) {
  for (int n=5 ; n>=i ; n--) {
    System.O.P (" ");
  }
  for (int j=1 ; j<=i ; j++) {
    S.O.P (j);
  }
  S.O.PLN ( );
}
```

if

Arrays :-
Single variable can store only one value. Therefore arrays are used to do it.

Array : Collection of homogenous elements.
Collection of similar elements.

declare : (1) datatype ar[];
(2) datatype[] ar;
(3) datatype []ar;

int a []= new int[5];

ex :-
int n[];
int[] n;
int []n;

We need for loops to access arrays elements.

a[i][j]

Single Dimension Array

```
import java.util.Scanner;
Public class Operators{
    public static
        int a[];  (size)        } Declaration
        a = new int[5];
Scanner sc=new Scanner (System.in);
    for(i=0; i<5; i++){
        ~~for(int~~ S.O.Pln("Enter array elements");
        a = sc.nextInt();
    }

    for(int i=0; i<5; i++){
        S.O.Pln(~~a[i]);~~ ("a[" +i+ "] = " + a[i]);
    }
```

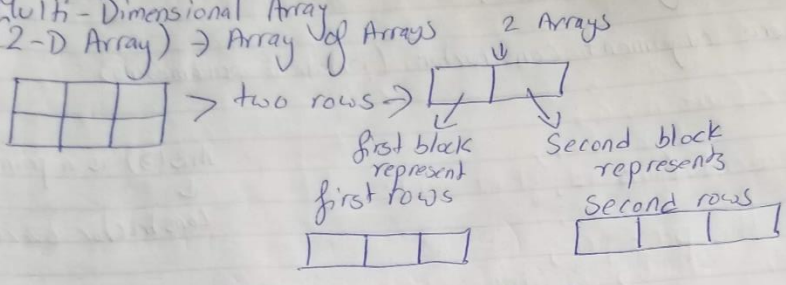
i	a[i]
0	1
1	2
2	3
3	4
4	5

- Two dimensional array: a[][]
  rows | column

Assignment:-
Write two dimensional array same as above example.

Practice

Multi-Dimensional Array (2-D Array) → Array of Arrays



Declare: `int[][] a;`

```
int  [][]a;
int[] []a;
int a[][];
int[] a[];
```

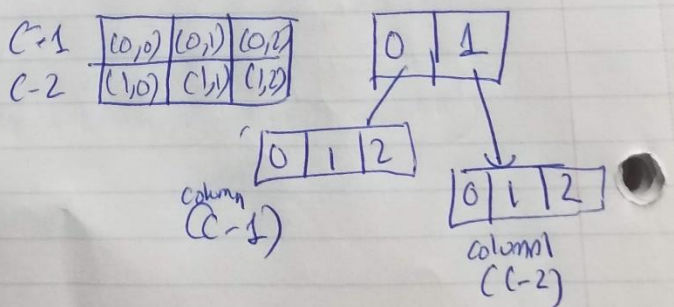
IMP

```
int[] a, b;     a → 1D
                b → 1D

int a[], b;     a → 1D
                b → just a variable (INT).
```

```
int[][] a, b;     both are 2D array.
int[] a[], b;     a → 2D array
                  b → 1D array
int[] a[]; b[];   a → 2D array
                  b → 2D array
int[][] a, []b;   a → 2D array
                  b → compile error
```

```
int[][] a;
a = new int[2][3]
```



for each loop:

```
for (datatype variable : array/collection) {
for (int n : a) {
    S.O.Pln(n);
}
```

⇒ 1 2 3 4 5

* String args[] → string array

```
public ... (String n[]){
    for (i : n)
        String S.O.P (i);
```

Print

java Class Hello

Hello

java class Hello world

Hello

world

java class "Hello world"

Hello world

array :-

```
int n
array n[] = new int[n][n];
```

Constructor:
A constructor is a method that is used to initialized the properties of object when it is created
classname obj = new constructor();

It takes part in object creation & it initialise the variables

properties:
1) Constructor name is same as classname.
2) Constructor has only one property i.e parameter.
3) Constructor can't have any return type i.e not even void.
4) Constructor gets executed when object is created i.e once per object creation.

```
class Student{
    int id;
    float per;
    Student(){                    // constructor
        id = 1023;
        per = 76.45f;
    }
    void details(){               // methods
        S.O.Pln("id=" +id);
        S.O.Pln("per=" +per);
    }
}
class Demo{
    p.s.v.m(String args[]){
        Student rahul = new Student();   // constructor executed
        rahul.details();          // 1023
                                  // 76.45f
```

we can put parameters too.

SQL: (Structured Query Language)

oracle → database
mysql
mongodb
sql server

SQL

DDL (Data Definition language)
create - we can create rows & columns
alter
↳ Modify → columns
↳ add → add new column to existing table
↳ rename
↳ drop

DML (Data Manipulation language)
insert
↳ sing rows
↳ Multiple row

Update
↳ entire column
↳ particular row
↳ multiple row

delete
↳ entire table (entire column is not possible)
↳ single row
↳ multiple row

select
↳ All rows
↳ particular column
↳ single row & multi row.

Rules of SQL:

1) Dublicate table names are not allowed.
2) Dublicate columns in a table are not allowed.
3) Every commands or query ends with a semicolon.
4) The character values can be given in single quotes itself.
5) SQL is case insensitive i.e ~~that~~ what erer case we write it will take upper case.

Datatypes: classification of information.

numbers:

- number (size);
- It can store without decimal values.
  ex: id number (10)
  The size limit is 38. (size indicates no. of digits)
- number (precision, scale);
  • store decimal values

ex:
percentage number (5, 3) → Total numbe
76234 → 76.234 → from last number it take 3 numb

- characters:
  char (size);
  It can store only characters.
  ex:- name char (30)
  size is limited to 2000.

Inheritance

parent / base / super

int a;
a*a

Square

int b;
a+b;

Sum extends Square

child / derived / sub

Three types :-

Single level
Multi level
hierarichal

Single level

package

```
import java.util.*;
class Square{
  int a;
  void input(){
    Scanner sc = new
        Scanner(System.in
    S.O.Pln("Enter a value:");
    a = sc.nextInt();
  }
  void sqr(){
    S.O.Pln("Square = "+(
  }
}
```

```
class Sum extends Square{
  int b;
  void accept(){
    Scanner sc. = ....;
    S.O.Pln("Enter");
    b = s.nextInt();
  }
  void add(){
    S.O.Pln("Sum = "+(a+b));
  }
}
```

```
class InheritanceDemo {
  public static
  Sum n = new Sum();
  n.input();      n.accept();
  n.sqr();        n.add();
```

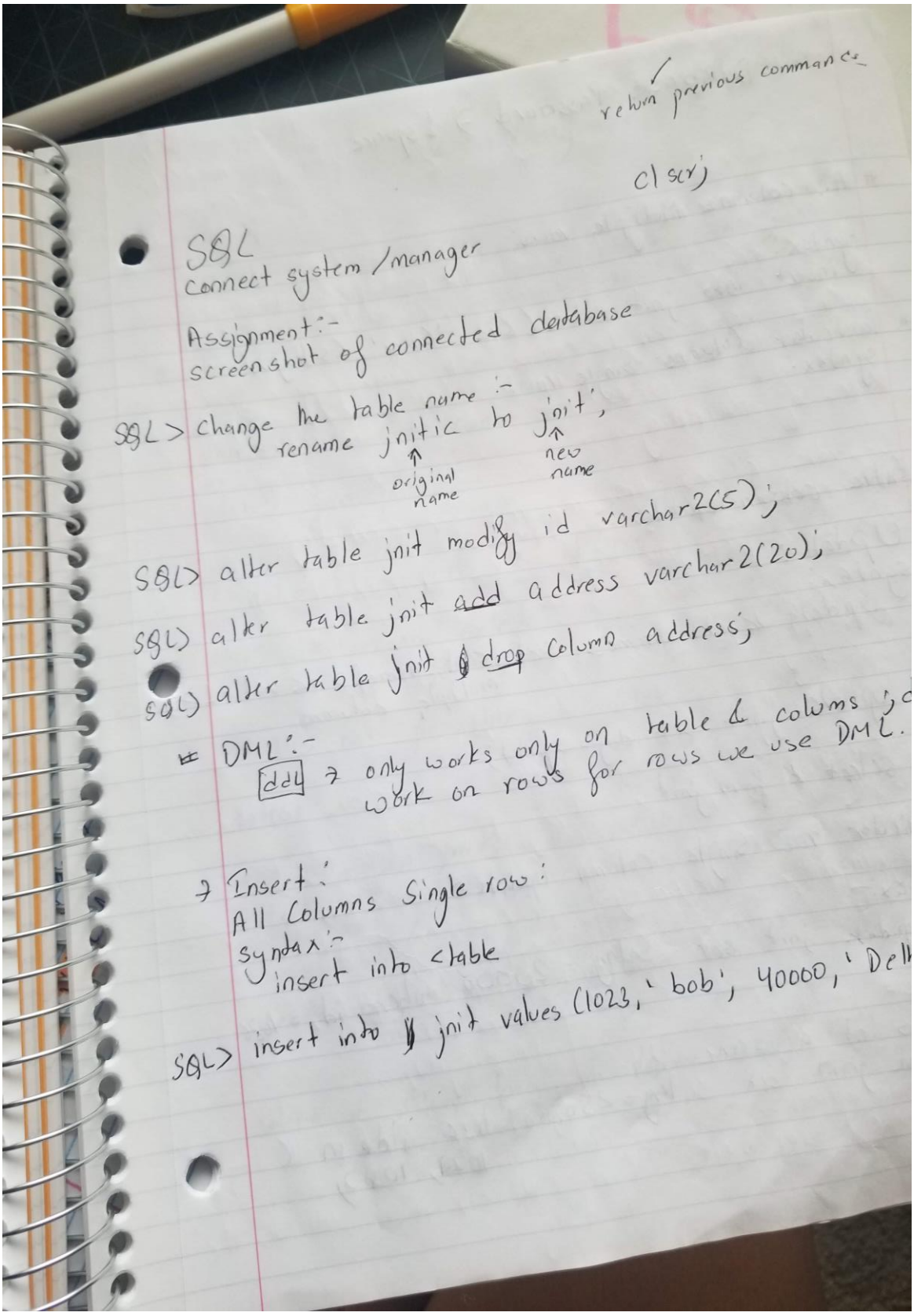
<extends>
we can access
from super class
(Parent class

object → new keyword
static → already defined memory.

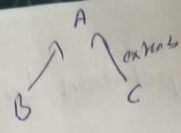
```
class Parent{
    int age =54;
}

class Child extends Parent{
    int age =22;
    void show(){
        S.O.P("Child age is "+age);
        S.O.P("Parent age is "+ age);
                                  → super.age;
    }
}
Class Test{
    P. s. v. m(S. args[]){
        Child x = new Child();
        x. show();
    }
}

final int age=22;
Child(){
    age =34;
    S.O.P("Child age is "+age);
```


return previous commands
cl scr;
SQL
connect system/manager
Assignment:-
screenshot of connected database
SQL> change the table name :-
rename jnitic to jnit;
original name
new name
SQL> alter table jnit modify id varchar2(5);
SQL> alter table jnit add address varchar2(20);
SQL> alter table jnit drop column address;
DML:-
ddl → only works only on table & colums ;
work on rows for rows we use DML.
→ Insert:
All Columns Single row:
syntax:-
insert into <table
SQL> insert into jnit values (1023, 'bob', 40000, 'Dell

→ It happens in inheritance.
run time polymorphism



parent class

```
A a = new B();
a = new C();
a.compute();
```

Point at C

Method overriding

~~class Car~~
~~void cost()~~

method name can be same with same type of parameters or same order of parameters or same no. of parameters.

Rules of overriding:

1) Seperate classes are required for overriding.
2) Inheritance class is mandatory.
3) Seperate objects are required for overriding

Class Casting:-
converting one class object to another class.

- Upcasting
- Downcasting

- Upcasting :- sub class obj is converted into super class obj i.e upcasting
  Super class reference storing the sub class object

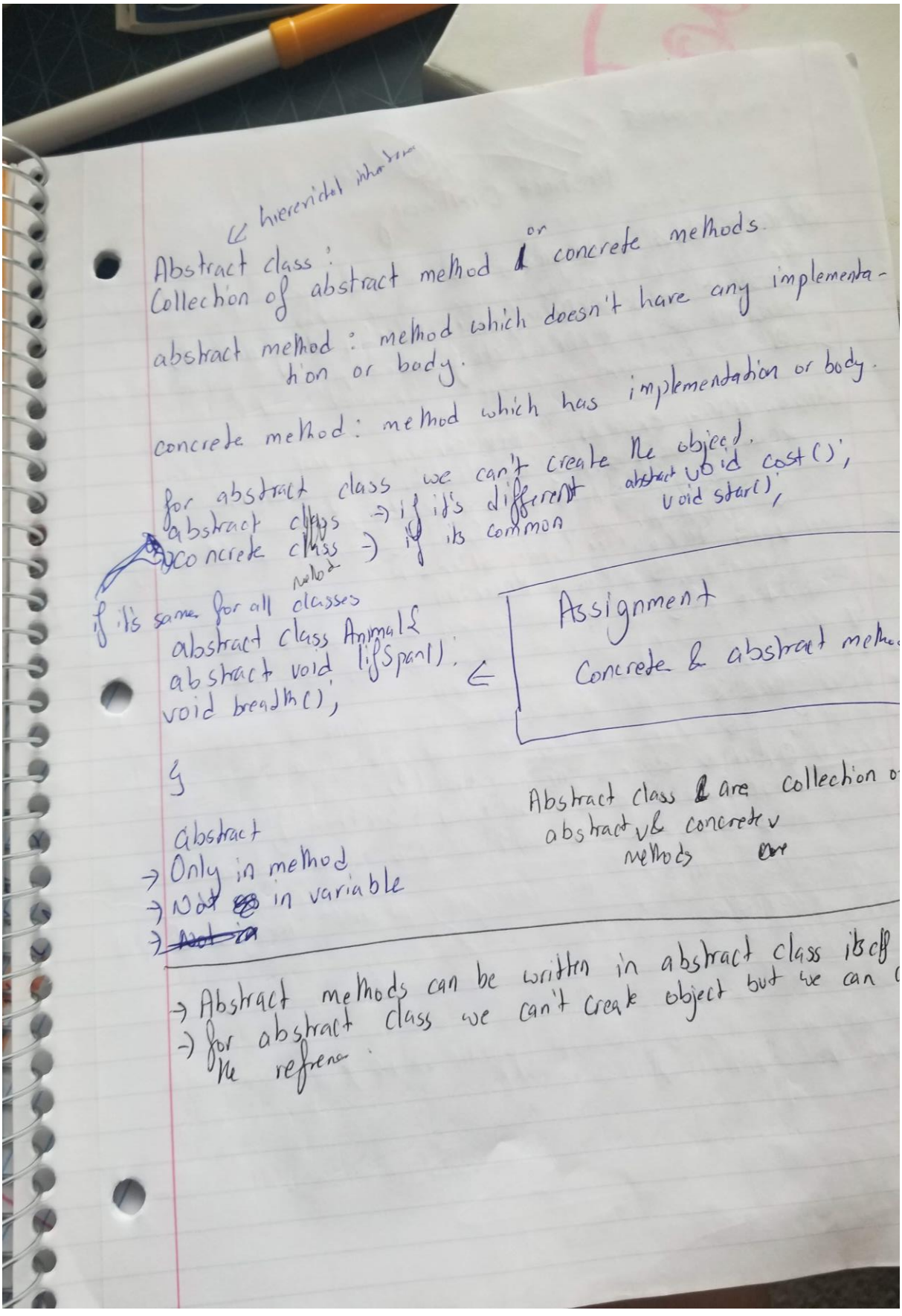
```
Car n = new Car();
n.cost();
n.milage();
n = new Audi();
n.cost();
n.milage();
```

Assignment Overriding

movieid movieName

new memory is created.
n is now pointing to Audi object

jvm: responsible for allocating memory, it will call a method gc (Garbage collector) which deallocates the memory.



Hierarchical inheritance

Abstract class:
Collection of abstract method & concrete methods.

abstract method: method which doesn't have any implementation or body.

concrete method: method which has implementation or body.

for abstract class we can't create the object.
abstract class → if it's different
concrete class → if its common

if it's same for all classes
abstract class Animal{
abstract void lifeSpan();
void breath();
}

abstract void cost();
void start();

Assignment
Concrete & abstract meth

Abstract class & are collection o
abstract & concrete
methods

Abstract
→ Only in method
→ Not in variable
→ ~~Not in~~

→ Abstract methods can be written in abstract class itself
→ for abstract class we can't create object but we can
the refere

final → constant

method name must be same with different type of parameters or order of parameters or number of parameters.

method name must be same with same type of parameters or order of parameters or number of parameters.

method: set of instruction that perform a task.
instance variables → heap area.
static → method are, depends on classname.
The variable whose value is common to all the obj.

Interface:

- A abstract class which contains all the abstract methods instead of using abstract class we can use inheritance.
- It is pure abstract, not concrete.
- Interface all the methods are by default abstract & public.
- A class must implementation an interface, a class cannot extend an interface
- Interface supports multiple inheritance
- If a class implements a interface then all the abstract method must be overrided.
- If we are overriding the interface abstract methods then all the overrided

---

In interface, it's by default (public static) so we don't have to mention it.

class casting → converting one class object into another class object

upcasting → sub class object is converted into super class object i.e upcasting

abstract class:-
collection of abstract methods & concrete methods

abstract method: The method which doesn't have any implementation.

concrete method: The method which has implementation or body.

```
class A {
  void sum(){}
}
```

```
class B{
  void sum(){};
}
```

```
class C extends A,B{
  p.s.v.m(String Args[]){
    C n = new C;
    n.sum();
  }}
```

Compiler gets confuse which sum to take (A or B). That confusion leads to ambiguity.

When both the super classes has same name, the derived class cannot differentiate which method class it has to call/execute.

Interface supports multiple inheritance
class & abstract class doesn't supports multiple inheritance but inheritance support multi

Exception:
In a normal flow of execution, if an abnormal situation occurs then jvm will terminate the program by throwing the exception class object.
(run time error)

Exceptional Handling

```
class ExceptionDemo{
  public static void main (String args[]){
    int a=10, b=2;
```

For exception handling;

```
try {
        → block which has risky code;
}
catch (ExceptionClassname reference){
        msg related to exception.
}
```

We have two predefined classes.

1 Object:
Object class is the super class for all the classes.

Sub class

Throwable ← super class

- Exception
  - RuntimeException
    - ArithmeticException
    - NullPointerException
    - ArrayIndex (AIOBE)
    - ClassCastException
  - IOException
  - EOFException
- Error → sub class

javap java.lang.ArithmeticException...

Syntax error:
if we miss any (;) or ({}) in the code it will give syntax error.

Logical error:
a=10; b=5
if (a>b)
S.O.P ("b is max");

Runtime Exception:-
< exception definition >

Exception is classified into two types:-

1) Un-checked
Runtime Exception.

2) checked.
IOException. & Interruption, Servelet, SQL.

Exceptional handling
↳ means to skip the instruction with the exception & the remaining part of the program correctly.
↳ to stop the inappropriate termination of program.

single catch

1.
```
try {
    riskyCode;
}
catch (ExceptionClass ref){
    msg;
}
```

multiple catch

```
try {
    riskyCode;
}
catch (      )
{
}
catch (      )
{
}
```

try with multiple catch

3)
```
try{
    riskyCode;
}
Catch (
```

Finally → is a block where the program will execute, even if it is wrong or not.

- try-catch
- try with multiple catch block
- nested try catch
- try-catch { try-catch }

throw → keyword used to generate the user defined exception

throws → handle exception.
↳ handle the unchecked exception only

```
class UserException{
    public static void main(String args[]){
    Scanner sc = new Scanner(System.in);

try {
    int age = sc.nextInt();
    if (age>=18)
    System.out.println("Eligible.");
    else
    throw new RuntimeException("Not Eligible");
}
catch (Exception e){
    e.printStackTrace();
}
```

OR

```
class p.s.v.m(String args[]) throw Exception{
    int age = sc.next
}
```

autoboxing → Convert it as an object.
compiler.

Generic

Generic:- (unknown type) class Example<E>{
- screenshot void show(E x){
S.O.P ln(x);
}
}.

Array:-
collection of similar data types.
drawbacks:
1: fixed size.
2: can't store disimilar data.

That is why:-

Collection:-
A group of individual objects.
note: with the help of genenics & wrapper class.

Set: It is a group of individual objects.
1. It doesn't allow dublicate values.
2. It allows null value.
3. It allows heterogenous elements.

HashSet:- It follows random order.
LinkedHashSet:- It follows insertion order
TreeSet: It follows ascending order.
→ It doesn't allow heterogenous elements. ~~has~~ nes TreeSet<String>();
TreeSet<String> ts=
ts.add("f");
ts.add("g");
→ cannot repeat.