DAY1

Date: / /

OOPS WITH JAVA 8

JDK - Java Devpm Kit) includes) does not includes
JRE - Java Runtime Envm. ↙

→ 1996 - JDK 1.0
- Write Once Run anywhere
C. - Compiled version was platform dependent
Java - .java compile → .class → runs on any platform
└ platform independent

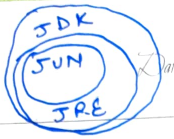
→ 1997 - JDK 1.1
→ Inner class , Java to DB connectivity. (JDBC)
→ Remote method invocation (RMI)
→ Reflection → to fetch metadata of class.

98 → JDK 1.2
- Collection Framework → Array resizable
- Hotspot JVM (used to run .class files)
→ J2SE 1.3
-

2009 → J2SE 1.4
- Regu. Exp.

2004 → J2SE 1.5
- logging API → to write the user activities.
- Generic

MATRIKAS



Date / /

- specifying type of collection

Enum → constant

Var-args →

- for each loop. → iterates array or collection

2006 → J2SE 1.6

→ Enhancement in JDBC API

2010 → oracle → sun microsystems

2011 → J2SE 1.7

- Try catch enhancements
- Better readability
- Introduced (_) in numbers.
- switch (strings)

2014 → J2SE 1.8

- lamba exp.
- license for commercial use.

* Javac

MATRIKAS

Date / /

Syntax.

```
class HelloWorld {
    Public static void main (String[]args){
        System.out.println("HelloWorld!");
    }
}
```

for comment //

for multiline /* ... */

→

```
class B {
    public static void main (String[] args){
        System.out.print("Hello World - Bx!!!");
    }
}
```

Int

★ Data Types.

1. Primitive DT
   - Integer
     - byte - 8 bit - 1 byte (-128 to 127)
     - short - 16 (32768 to 7)
     - int - 32 (-2^{31})
     - long - 64 (-2^{63})
   - float
     - float - 32 single precision
     - double - 64 double precision
   - character { char - 16 0 to 65535
   - boolean { boolean < true / false
     JVM specific

2. Reference DT.
   - Objects
   - Strings
   - Array.

no 0 for false
1 for true
in Java.

MATRIKAS

10 << 2 → 10 * 2^2
10 >> 2 → 10 / 2^2

1 byte → 8 bit long.

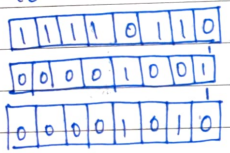
← binary.

b = 10;

2	10	
2	5	0
2	2	1
2	1	0
	0	1

for negative 2's compliment.

-10



is converting back to 10
2's

byte size

eg.

-128 to 127.

→ short size

-32268 to 32267

→ Char → It is the only unsigned DT in java hold +ve only.

c = 'a'; Assign ASCII values to char.

range → 0 to 65535 , 2^{16}.

- In terminal → It takes only ASCII value i.e if value is 128 or more then it will give (?).

→ float

d = 10.23 d or D ;

f = 10.23 f or F ;

MATRIKAS

byte b = 100;
↓ ↓ ↓
DT var literals

* By default float values are double so specify f at right side value.

eg. float f = 100.23 f; if only 100.23 double

- literal to variable
  - literal should be in range of variable type

for to var (first size is checked). target container.
- Compare both sides LHS should be eq. or more than RHS size.

* Type Casting.

eg. byte b = (byte) i;
- To forcefully type cast the size.
- But still will take only till -128 to 127.
- But there can be loss of precision

2	132	0
2	66	0
2	33	1
2	16	0
2	8	0
2	4	0
2	2	0
	1	0

10000100
1
01111011
1
10000100
2^7 ... 2^2
= 12

MATRIKAS

Date : / /

eg. s.o.pln ((int)(char)(byte)-2);

-2

-2 -1 0 65534 65535

-2

-1

→ rotates after 0.

130. is 3 position after 127.

* → -128 -127 -126 127

Then rotation on left.

* Narrowing conversion

converting bigger container to smaller

RHS → 32 → 8 → LHS

int byte.

- Widening conversion

smaller converts into bigger

RHS → 8 → 32 → LHS

eg. byte b1 = (int) 'c'; possible

char b2 = 'c';

byte b3 = (int) 'c'; not possible.

→ post increment

int j = i++ takes i value in j first then increment

j = ++i first increment then value is incremented & value of i

DAY 2

Date : / /

* short circuit operator

|| or or if first condition is true then breaks there

&& and and if first condition is false then it breaks there

* first character of class name & var name

: a-z A-Z or _ or $

After first

* If Else Syntax with example.

```
class Demo {                    → public static void main (String[] args){
    int a = 20;
    int b = 20;
    if (a < b) {
        System.out.println ("A is small");
    } else if (a == b) {
        System.out.println ("Equal");
    } else {
        System.out.println ("B is small");
    }
}
```

Concat in print statement

→ System.out.println (a+" "+b+" = "+(a+b));

Date / /

~~B No. to~~

* Binary no. to int

```
int p = 0B10;
System.out.println(p);
```

O/p → 2.

→ `int p = 10; // decimal`
`int q = 0b1010; // binary`
`int x = 0xa; // hexadecimal`
`int s = 012; // octal`

* Loops

eg. `class LoopDemo {`

1 table

```
public static void main(String[] args){
// for (initialization , condition , increment/decre)
   executes 1 time
int var=2
for (int i=1; i<=10; i++) {
        System.out.println(var+"*"+i+"="+(var*i));
    }
}
}
```

for 20 tables

```
for (int i=1, i<=20, i++) {
    for (int j, j<=10, j++) {
        System.out.println(i+"*"+j+"="+(i*j));
    }
}
```


Date / /

While loop.

1 table

```
int x = 1;
while (x<=10){
    System.out.println(var+"*"+x+"="+(var*i));
    x++;
}
```

for 2 tables

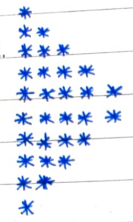
```
int p = 1;
int q = 1;
while (p<=20){
    while int q=1;
    while (q<=10){
        System.out.println(p+"*"+q+"="+(p*q));
        q++;
    }
    p++
}
```

Pattern

```
*
**
***
****
*****
```

```
class Pattern
    for (int i=1; i<=5; i++){
        for (int j=1; j<=i; j++){
            System.out.print("*");
        }
        System.out.print("\n");
    }
}
```

Date / /

2.


```
for (int i=1; i<=10; i++){
    for (int j=1; j<=i; j++){
        System.out.print("*");
    }
    System.out.print(" ");
}
for (int i=5; i>=5; i--){
    for (int j=1; j<=i; j++){
        System.out.print("*");
    }
    System.out.print(" ");
}
```

* Switch statement
  - → default is not necessary.
  - → switch statement executes one statement from multiple conditions.
  - → Case value must be of switch expression type only.
  - → Value must be literal or constant & doesn't allow variables.
  - → Must be unique
  - → If break not present it follow through the case statements.

Date / /

* Methods
  - Collection of instructions that perform specific task.
  - To achieve reusability of code.

eg.

```
class Method Demo {
    static void add (int a, int b) {
        System.out.println(a+"+"+b+"="+(a+b));
    }
    static void sub (int a, int b) {
        System.out.println(a+"-"+b+"="+(a-b));
    }
    static int mul (int a, int b) {
        int result = a* b;
        return result;
    }
    public static void main(String[] args) {
        int verylongvariablename = 10;
        add(10, 20);
        sub(10,20);
        int res = mul(10,20);
        System.out.println("Mul Result = "+
        System.out.println(verylongvariablename);
    }
}
```

DAY 3

Date: / /

* Static Variable / class variable (memory utilization)
  - → If variables are not loaded in memory then we can't use (access) the variables. (Inside method)
  - → Local variables always stay on stack (Part of main method)
  - → Block variables are declared
    - ↳ needed to be initialized.
  - → static var. are initialized by default value (0)
  - → Static var. ~~are variables~~ are declared inside class outside method with static keyword.

eg. Class StaticDemo {
    static int = a

↳ Part of class.

↳ static var. are loaded only once when class is loaded.
  - → Only one copy exists & shared with multiple methods.

→ 1> compile 2) run → JVM 3) JVM loads in memory.
  4) static var are loaded in memory
  5) Stack memory comes & main ~~was~~ is loaded.

→ Once modified the values will be carry forwarded after that.

→ Static → There is no need of creating an object to invoke the static method. It's saves memory.

Date: / /

* Static Block.
  - → Used to initialize static variable.
  - → staticB. is declared inside class outside the method.
  - → eg. file staticDemo3.
  - → It will always execute even if static variable is not declared or given.
  - → Invokes only once.

* Final Variables.
  - → Final variable does not change in any condition.
  - → Also known as compile constants. → static const → local const.
  - → Does not give default value
  - → Can be used as constant is switch case.
  - → final int a = 20 : a contains binary representation of a. i.e 20.

DAY 4

* Instance Variable / Instance Method / Init Block
  - → Instance var are declared inside class outside method without static keywords & also instance method are declared without static keyword.
  - → Part of Object
  - → Has default values.

MATRIKAS

→ Run flow, class loaded.
static box.
→ static variable will be declared
→ To access inst. var. we have to make object of the class in which you have declared the var.

→ Object syntax
- Always created with new keyword.

In main method

InstanceVarDemo ivd1 = new InstanceVarDemo();
class name ref. var. refer var. new obj. class name

or A a1 = new A();
dt ref. var.
reference (stack) object → heap

In stack
~~A1~~
ivd1

heap memory
A ()
InstanceVarDemo()
x=0 y=0
a=10 b=20

- To access Inst. var
→ (ivd1.a) / (ivd1.b)
or (a1.x) / (a1.y).
- To modify
- ivd1.a = 100; or ivd1.b = 200;
a1.x = 10; a1.y = 20;
It will change value of a,b,x,y in heap memory.

Date / /

→ Contains of ivd1 → binary representation of how to reach heap memory to access that object
stack to heap. → path.

→ Multiple objects can be created in same class
→ to give maximum data or create multiple values of any org. or project or anything.
→ Created object will create obj. in heap memory.
→ Objects can have multiple values or copies.
↳ inst. var. values copied in various objects
→ Modifies only that inst. var. which is assigned to that object.

→ One ref. multiple objects
last inst. var. will have latest values & rest will be unreachable & garbage collectors comes into action.

- Init Block. (Runs first).
→ Syntax.
```
{
    System.out.println("Init block invoked");
    a = 1000;
    b = 2000;
}
```
→ Runs multiple times when object is made.
→ Have fixed value.

* constructor can not be abstract, static, final & synchronized. Date: / /

→ Under class bottside method.
→ Multiple init blocks can be cleared & runs as per order it has been created.

* Instance Method.
→ Syntax
```
void m2 () {
    System.out.println ("m2");
}
```
→ To access or run method object needs to be created.
→ To access ins method
↳ ~~Instance First Class~~
```
InstanceMethodDemo imd = new InstanceMethodDemo();
imd.m2 ();
```
→ From other class also to access make object by the class name & access the variables. or methods.
- Always invoked thr. objects

* Constructor
→ Constructor name & class name must be same
→ Don't have return types
→ Used to initialize inst. variable. (dynamically)
→ Runs after init block if init is present.
→ Multiple constructors can be created as per need.

→ cmd - javap class name to fetch metadata of class

Date: / /

→ Syntax.
eg.
```
{
    Employee (int eid, String name, double sal, String gen) {
        employeeid = eid;
        employeeName = name;
        salary = sal;
        gender = gen;
    }
}
```
To call or pass values.
```
{
→ Employee e1 = new Employee (1, "A", 10000.0, "Female");
    :
To print
System.out.println (e1.employeeId + " " + e1.EmployeeName + " " + e1.salary + " " + e1.gender)
    :
```

* final instance variable
- Initialize on same line or either in init block or constructor.

constructor.
→ By default a constructor is declared in the class by java. with zero arguments. by class name.

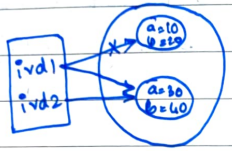
Date: / /

DAY 5

~~Assigning~~ ref. variables.

- Objects can change values too
  ivd 1 = ivd 2
  - copying contents of ivd 2 in ivd 1.
  - changing values of ivd 1 will also be same for ivd 2.

in heap memory.



* This keyword

- Every instance block has local ref. var. named this. Ref. to currently invoked object.
- Works in background

eg.

```
void print () {
    System.out.println( a + " " + b );
}
```

(a → this.a, b → this.b)

→ Only available in instance method / constructor / init

- Points to the object through which it is called. / currently invoked.

→ Used during name conflict.

Date: / /

* Object can hold three values

X x = _____ ;

→ object of the child class of class X.
  X x = new Y();

→ null

→ new X();

Int

class load time

→ When we write java classname we are submitting it to JVM and at class load time static variables will get memory.

eg.

```
class RandomDemo
{
    static int a;            // static primitive var.
    static X x;              // static instance var.
    int a;                   // Instance var.
    X xInstance;             // Ref. Instance var.
}
```

(1) class var.

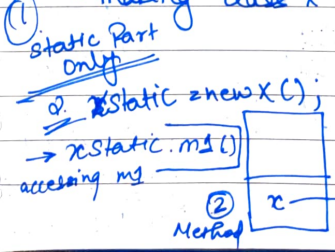
→ making class X object & assigning it to x

(1) Static Part Only

2. xstatic = new X();

→ xstatic.m1();

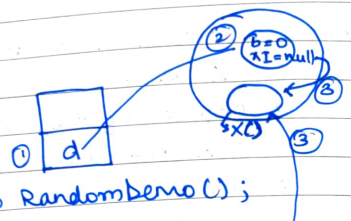
accessing m1



Date : / /

② Instance Part Only.

① int b;
X xInstance;

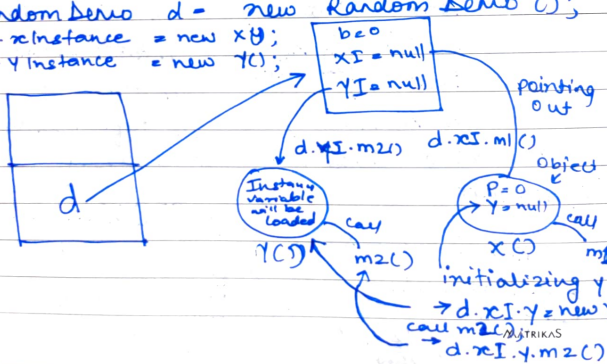


② RandomDemo d = new RandomDemo();
→ accessing values.
d.b = 100;

③ d.xInstance = new X(); makes object of X class.

d.xInstance.m1(). to call method of class X.

③ eg. int b;
X xInstance;
Y yInstance;
After invoking class
RandomDemo d = new RandomDemo();
d.xInstance = new X();
d.yInstance = new Y();



DAY-6

if import statement present of Pack2.A. Then to access class A of same package we have to write
pack1.A A a1 = new pack1.A();

- final object
eg. final B b = new B();
- Object of b cannot be modified.
- Path is fixed.
- The contents of that objects can be modified.

★ **Arrays.** → collection of similar type of elements.
- arrays are objects in java. elements.
- 1D DT [] arr = new DT [ ];
ref var.
- Arrays have default value.
eg. int → 0 , String → null
char → ' ' , bool → false.

- 2D int [][] arr2d = new int [5][5];
rows columns
5×5 elements
→ default → null.

→ Every element in object itself is an array.
↳ having row+1 objects.
→ Calling method thr. null element will give run time error. It will compile but after run file it will give null Pointer exception error.

Immutable Object → An object whose values will not be changed

* for each loop
  - → Only used to iteration purpose.
  - → Can not initialize values.
  - → It goes to 1st element & copy that element to another variable to print that element content.

★ Packages Import & Access Modifies

→ In Package name → no spaces allowed.

→ If classes in same package no need to import anything.

- Access Modifiers
  - Private
  - Protected
  - Public
  - } written
  - Default → Default

→ Written at class level → before class
Method level → before method
Variable level → before var.
↳ (static, only instance var)

→ If import statement is written then it will give priority to import statement.

→ If not it will search in same package.

→ At class level only public access modifier is allowed.
↳ Public class name { }

→ Access Modifiers → Defines the visibility of variables & methods.
- → public. → anywhere accessed
- Private. → within class accessed

Accessing (classes in one java file)	same class	same package	Outside packg
Public	✓	✓	✓
Private	✓	×	×
Protected	✓	✓	✓ * Only thr. inheritance
Default	✓	✓	×

- Most Restricted
- → Private → Default → Protected → Public.

★ Accessing in diff. packages eg. Pack 1 & Pack 2
→ import Pack1.A → to access contents of A in C class in pack 2.

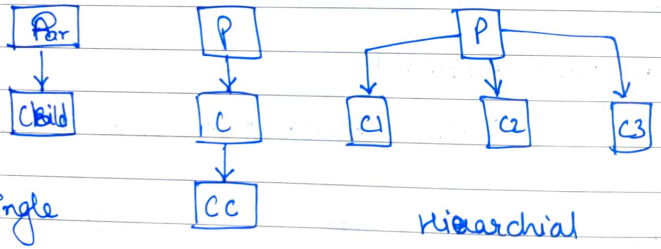
→ To use constructor of class in diff. package then make sure it is visible. i.e Public class.

Date / /

DAY 7

* Encapsulation
→ process of wrapping code & data together into a single unit.
→ Protects the data (instance var).
→ Get → To read the value of Inst. var. (& Return)
Set → To write / update the value of Inst. var.
→ No. of variables = No. of get-set pairs. (read, set) else / Get or Set.

* Inheritance
→ Java does not support multiple inheritance using classes.



Single
Hierarchial

Multilevel
→ class child extends class parent.
→ P ↑ C → class C extends class P

Date / /

→ Interface ↑ class C implements → class implements Interface

→ Interface → class not possible.

eg.
```
class Parent {
    int a = 10;
    int b = 20;
    void () {
    }
class child extends Parent {
}

class SingleINDemo {
    child c = new child ();
    System.out.print (c.a);
```

→ Only Instance var | Methods are inherited.
→ Assigning object of child class to the ref. of parent class is called upcasting.
eg. Parent p2 = new child (); (Parent class, child class)

1. Compiler always checks ref. type & not object type.
2. JVM always checks object type (RHS) & not ref. type. & invokes.
3. Which overridden Method to call is decided by object type.

→ overridden → run time polymorphism (many forms).

* Overridding
→ Parent class method overridden by child class
→ Method name must be same
Return type " " "
No. of args " " "
Type & order " " " (Data Type).
Access modifier can be same or less restrictive than parent.
→ @Override → compiler checks for overriding rules.

* Overloading
→ Same class of P-C class.
→ - Method name must be same
- Args canbe same or diff. if same then
• one of them arg. must be diff.
- Return type can be same or diff. (if same, not enough)
- Access modifier " " " " "

MATRIKAS

Date / /

overloading → compile time polymorphism.

DAY 8

→ Method overloading is not possible by changing just the return type as when we pass the args to method then the methods there will be ambiguity.

MATRIKAS

Date: / /

Final Class

- Child class cannot inherit from final parent class.
- Child class cannot be created. (compile time error)

Final Method

- Final methods cannot be overridden.

- Protected variable: can be accessed outside package only thr. Inheritance i.e child will only access by its variable & inside the inherited class.

- If you want to access protected var of parent class of pack 1 to inherited class of pack 2 then you can access. if child class or any other class not inherit but in same package can also access the protected var. thr. making method in child class and accessing in non inherited class. (main method is in non inherited class).

- Protected variable cannot be accessed thr. parent object in child class.

Date: / /

Int

- Instance variable always resolved on ref. type (JVM run time).
  - i.e it will print variable of ref. var. type class.
- Instance method always resolved on object type.

Constructor chaining

- By default first line of every constructor is to call super class constructor with zero args.

- If parent class, then super(); of that class will be object class by default.
- Object class belong to java.lang.object.

- For object to be made, then constructor of object class will be loaded. & Be that in chain of various class constructors.

Methods to complete the chain

- If no constructor matches then make constructor of class E(); with zero args. to complete the chain.
- Make the constructor with 1 or more args of child class & match with parent class (making constructor with 1 or more args).

class ← this is concrete class.
abstract class ← this is abstract class.

→ To call constructor within same class then instead of super() use this(); with similar no. of arguments.

→ this() & super() cannot be used together.

(Abstraction → process of hiding the implementation details & showing only functionality to the user)

★ Abstract Class.

→ Parent —don't know prop.→ Child

Child —know/inherit Prop.→ Parent.

→ Abtract methods are methods which are declared but not implemented. (cannot answer the prop. of child)

→ If method() is abstract then
↳ The class must also be declared abstract class.

(T&C)

• Contract b/w Concrete class & Abstract class

rules → concrete class must provide implementation of all abstract methods of parent class (unimplemented)

Case → If child class does not fulfill the rule then make that child class abstract too.

→ if partial method fulfill the rule

- Interface don't have constructors

Date / /

then override that abstract method with the result.

→ After that the further child class will provide implementation to the remaining unimplemented abstract method.

→ Implementation means → override that abstract method.

• Abstract class Rules

→ Object of abstract class cannot be created. (RHS)

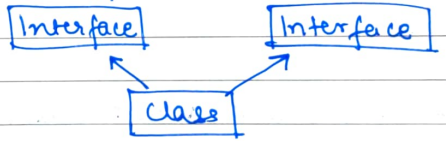
→ Ref. can be created (LHS)

★ Interface.

(→ A blueprint of class, used to achieve abstraction. or description of actions that an object can do).

→ Interface —extends→ Interface; Class —implements→ Interface; Class —extends→ Class

→ Java supports multiple inheritance using Interfaces



★★ → Class implements multiple interfaces.
Class extends only one class.

→ Class in which methods does nothing then consider that class as interface

→ While overriding abstract method public needs to be written before method.

→ Syntax:
interface class <class Name>{
}

→ interface has abstract method only
↳ if empty then only abstract method can be created. - by default public abstract is provided

→ interfaces are by default abstract. but if one of the method is implemented then abstract is written before class.

→ Abstract can have non abstract method.

→ By default interface methods are public and abstract.
But abstract needs to be written public in order to make that class or method public.

Rules
→ Child class implements Parent class.
→ All overridding properties
→ Concrete class must implement all unimplement Abstract methods of parent interfaces.
→ If child class are abstract class then public before abstract methods must be written.

→ If there is no parent class of child class then parent of child class will be object class.

→ Object cannot be created of Interfaces.

→ If abstract method is in class then no need to provide public because it is defined in class.

DAY 9

- Extends & implements can be written together.
eg. Extends 1st then implements 2nd.

* Garbage Collection
→ It is a background thread with lowest priority of 1.
→ Priority levels 1 → 10
→ In control of JVM, user can only req. to JVM to invoke GC.
(1) → If user requests then the obj. not in use if will invoke GC thread.
(2) → You can release objects for GC to free memory. (GC works on its own).
* Methods to release objects or making them eligible for Garbage collection.

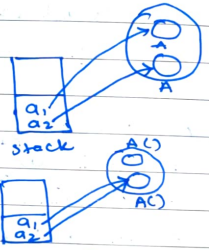
MATRIKAS

Stack → LIFO Last In First Out
FILO First In Last Out

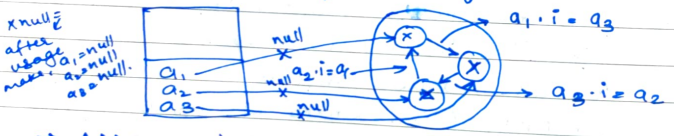
Date / /

1. Nulling a Ref.
eg. A a = new A();
After using a make that ref. = null
i.e a = null;
then the link will not be there & it will be unreachable & GC will remove that object & free memory.

2. Reassigning a ref. variable
eg. A a_1 = new A();
A a_2 = new A();
:
After usage
$a_1 = a_2$;



3. Isolating a ref. / Island of Isolation.



→ After using a_1, a_2, a_3 make them null and the objects referring each other will be eligible for GC.

MATRIKAS

Date / /

* How to request JVM to run GC:
1. System.gc();
2. Runtime is a singleton class.
→ Runtime rt = Runtime.getRuntime();
rt.gc();

→ After collecting Garbage, the GC will invoke finalize method to release resources held by that object. & before freeing memory
→ finalize() { → method of object class.
release resources;
}
We never release resources in finalize method.

* Wrapper Classes (To make collections we need primitive to be converted to wrapper class).
→ ① Primitive —Boxing (convert)→ Wrapper class
② P. ←Unboxing— W C

ways int a = 40;
①(a) Integer i = new Integer(a);
(b) value of (static method) - accepts only string args.

MATRIKAS

a=20; Auto boxing
Integer j = a;

compiler will write Integer.valueOf(a) internally.

eg. Integer i5 = Integer.valueOf("200");
Double d = Double.valueOf("20.3");

② Unboxing → converting wrapper to primitive.

eg. int a = i3.intValue();
byte b = i3.byteValue();
ref. var (It will convert i3 value to primitive)

// converting String to primitive
// parseXXX
eg. int p = Integer.parseInt("100");
double q = Double.parseDouble("10.2");

→ Wrapper classes objects are Immutable
→ value of obj. will never change.

→ For each and every primitive data type there is one class associated to it.

Primitive	Class
byte, short	Byte, Short
int, long	Integer, Long
float	Float
double	Double
char	Character
boolean	Boolean

MATRIKAS

Date: / /

1st way to convert or boxing methods

• Boxing. int a = 20;
eg. Integer i = new Integer(a);
double d = 10.2;
Double d1 = new Double(d);

2nd
Integer i = new Integer("10");
→ in string only integer values should be passed else number format exception. error.
→ It will compile but error at runtime.

3rd static method. valueOf (accepts only string value)
Integer.valueOf("200");

* Pool
→ Boxing type
eg. Integer i1 = 100;

→ Made by Java for every DT except float & double

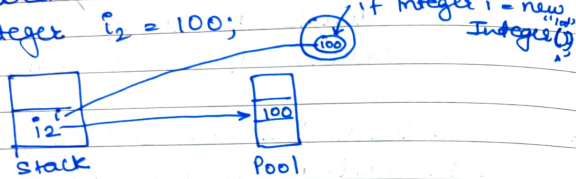
→ Byte	-128	127
Short	-128	127
Integer	-128	127
Long	-128	127

MATRIKAS

Date / /

→ If value within range then object will be created in ST Pool.

Integer i2 = 100;



→ if out of range then object will refer to heap memory.

DAY 10

Exception → event which disrupts normal flow & abruptly terminates at line of probable exception.

* Exception Handling

Parent of all classes

Object

↕

Throwable

Runtime error → Error

Error → Stack Overflow (stack), Out of Memory (objects heap)

→ we can write error handler code but that will be of no use.

Exception → Checked Exception → IO Exceptions → compile time exception

Exception → Unchecked Exception → RuntimeException → Null pointer Exception → Runtime exception

→ Stack Trace → Errors on which line or cause of error after runtime of that file.

↳ process of errors.

→ Checked Ex - Are those exception who directly or indirectly inherit Exception class.

→ Unchecked Ex - Are those exception who directly or indirectly inherit runtime exception class.

↳ compiler will not give any warning during compilation.

* Handling Exception

→ At line of Exception

eg.

```
try {
    Integer i = new Integer();
} catch (NumberFormatException e) {
    System.out.print("Enter nos. only");
} finally {

}
System.out.println("Reached here");
}
}
```

(At this line jvm internally with throw keyword and raise a new object with that exception)

can be same class or parent class

It will not raise exception & run the code & print accordingly

→ For one try only one catch will execute

MATRIKAS

Date: / /

Rule
→ Wider exception to be written at the last.
eg. Parent ref. = new ⊟ Exception();
↳ i.e most specific to be written above wider exception.

→ Finally block will always execute. even if there is no exception.
↳ Release resources → guaranteed
↳ At end after try & catch block.

→ If more than one exception in try block the it will raise first exception & go to catch block directly.

* Exception Propogation of Unchecked Exp.

→ Exception thrown at the line having exception probability & if it is not caught there then it drops down to the previous method, and again drops if not caught again.

→ By default unchecked exception are forwarded in calling chain (propogated).

* → Compiler never throws exception it forces you to do something about that exception.

DAY 11

* Checked Exception
→ Compiler will force to do something abt that exception.
→ To handle 2 Method
1. Try Catch (Handle by writting)
2. Propogate Exception. (Not able to handle)
↳ using throws compiler will compile but JVM will throw exception & program will crash

To read file
Ques import java.io.*
or
import java.io.File;
FileReader;
FileNotFoundException;

→ Compiler will raise unreported exception, ↳ type of checked exc.
of FileNotFoundException which might come or thrown. & Handle that exception
* → must be caught or declared to be thrown.
• throws → written at method level to propogate an exception.

2.
public static void main(String[] args) throws FileNotFoundException {

}

DAY 12

* looks for file in same path or directory you are in.

eg. To load .class file

```
public static void main(String[] args){

    try {
        Class.forName("A");
    }catch(ClassNotFoundException e){
        print statement
    }finally{
        print statement.
    }
```

To read A.class file →

* Checked Exception Propogation

→ By default, checked exceptions are not forwarded in calling chain (propagated) They are needed to be thrown manually.

→ If in a method you don't want to write try catch then throw that exception using throws keyword at method level i.e void m1() throws exception name {
}

** → Exception can be handled in any method in call stack either in main() method, p() method, n() method or m() method.

Date / /

→ checked exception → catch block can be written only when there is exception related to checked exception & compiler forces you to do so. otherwise the program will not compile.
↳ throws can be written elsewise. Even if there is no exception related to checked exception.

→ If you are not going to write try & catch then propagate by using throws keyword at method level

→ But at any method () you wrote try & catch then don't write throws keyword in any further method.

* Rethrowing an Exception

→ Rethrow for Unchecked class

```
throw e;
```

then you also have to write throws at method level to propogate it again.

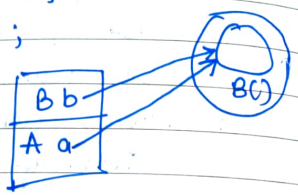
Object of child class refering to reference of parent class, is upcasting

*Down casting

Only upcasted variables can be downcasted (Inheritence).

eg. class B extends A {}

A a = new B();



Forcefull convert upcasted variable to child class var.

B b = (B) a;

eg. class B extends A { }
class C extends B { }

possiblity

A a = new C (); * a ka ref. var b ke ref. main convert kar rahe hai.
B b = (B) a;
C c = (C) a;

IMP eg **

Class Cast Exception at runtime
unchecked Exception

A a3 = new A();
B b3 = (B) a3;

Compiler will not give any error but JVM will throw exception.

DAY 12

* Inner Classes

→ It can access all the members (data members & methods) of the outer class, including private.

→ It is a class that is declared within a class or interface.

→ Regular IC → Instance Inner class
Method local Ic → Refer to local variable
Anonymous IC
Static / Nested Ic

* Method Local Ic., Rules :-

→ Inner class can be public, protected, default, private.
→ local var. can only be final.
→ Cannot apply access modifier to method local Ic.
→ First declare class then create object.
→ Only final is allowed to Method local IC.
→ Method local IC can access var already declared in method outside MLIC.
→ Can Access local var. but cannot modify values of that var. inside MLIC.

* Static / Nested IC

→ To access static class var or methods no need to create Object of Outer Class.

* Anonymous I C.
→ class with no name, Java provides name.
→ Used to create child class w/o any name
→ Also be called as M C I C but w/o name
→ Anonymous IC creates concrete child class.
→ Abstract Anonymous IC cannot be created
↳ create only concrete child class.

• Lamba expression works only on functional interfaces
→ funtional means interface with only one abstract method.
→ focuses only on method.

Syntax.

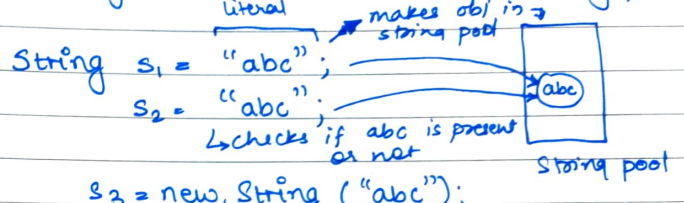
eg. Runnable r4 = () -> {
System.out.println("r4");
};
// create child class
// create object of child class & assign to r4.

DAY 13

javap lang.String To see in-b, Methods Java has.

* String class has 13 constructors.

* → String objects are immutable
String literals are saved in String pool.



→ value of abc will never change, if you reassign that variable then it will make new object with that content but that content which was already there will not change.
→ GC will not work for Pool.
→ Pool - used for reusing object having similar value.

→ == - checks binary representation of path
↳ equals - checks content of the object.

→ compare to → gives difference

* → String Pool :- It is a seperate place in the heap memory where the values of all the string defined in program.
On assignment of value to a string variable value will be stored in heap in string constant pool.

→ The string constant pool is small cache that resides within the heap.

Date / /

* Collection

→ Framework that provides an architecture to store & manipulate groups of objects.

Types:

Ordered: which maintains the insertion order

eg: 10 , 20 , 30 , 40

Un ordered: does not provide any order i.e elements cannot be accessed using indexing.

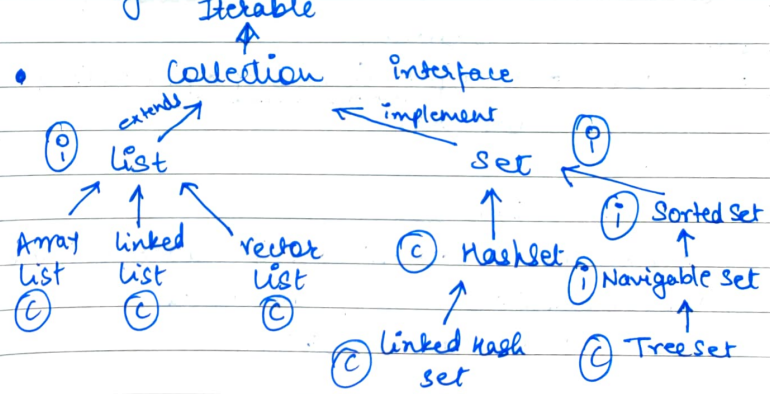
eg: 20 , 30 , 10 , 40

Sorted: At time of insertion sorts the data.

eg: i/p 5 3 2 1 6 o/p 1 2 3 5 6

Un sorted: The data inserted is in random manner not sorted.

eg:



MATRIKAS

MATRIKAS

Date: / /

<key, value>

Map

Hashmap → Linked Hashmap

Sorted map → Navigable map → Tree Map

* List is the only collection on which we can work on the basis of index
  - Ordered collection.
  - ~~Fixed~~ Size already defined.
- Set/map → maintains uniqueness

When to use

- Array list → Fixed size, Ordered, Faster retrieval, Less Random insertion
- Linked list → More Random insertions/deletions Ordered.
- Vector → multiple threads → synchronised but one at a time Ordered.

- Set → Uniqueness
  - Hash set → Ordered doesn't matter, uniqueness
  - Linked Hash Set → Ordered, uniqueness.
  - Tree Set → uniqueness, Sorted.

Array list default size 10 is more than float

$= \frac{\text{odd} \times 3}{2} + 1$

Date: / /

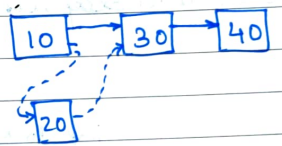
1. Array list → during insertion the list size is increased & the values after that index is shifted & then the value is inserted.

eg.

10	20	30	40	50
0	1	2	3	4

stack - Last in First out

2. Linked list → Internally it maintains nodes. having next pointer



for removing

deleting

* Collections Class → Static utility class.
  - → Import java.util.Collections;

To sort List ~~Array~~ collection.

* Backed Collections
  - → Works as reflection

1st collection	2nd collection
Element Added	reflected
Reflected	Element Added
Removed	Reflected.

Object class equals() does == internally.

→ Tree Set methods
→ ts.headSet(); makes collection of < than argument declared
ts.tailSet(); makes collection of >= arg.
ts.subSet(); makes collection of range.

* Linked HashSet, HashSet can have null value.
→ Tree Set can have null but at runtime it will give NullPointerException

Day 14

→ To maintain uniqueness, equals & hashcode (imp to maintain uniqueness) & toString method should be written.
↓
To give meaningful message after runtime

→ If no toString method in Employee class then Object class's toString() will be called & ~~hashcode~~ hexadecimal value will be displayed.

Collection → java.util. package.

* Collections.sort → used to sort two objects
1. Comparable, Comparator. → used for multiple sorting criterias.
↓
functional interface.
used (compareTo())
(one criteria only).

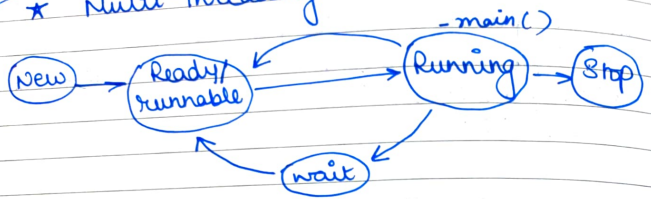
Queue → First in First out.
↳ Priority Queue
↳ Array Deque faster than Array List & Stack & has no capacity restriction

Multithreading - process of executing multiple threads simultaneously.

Date / /

DAY 15 thread - sub process.

★ Multi Threading



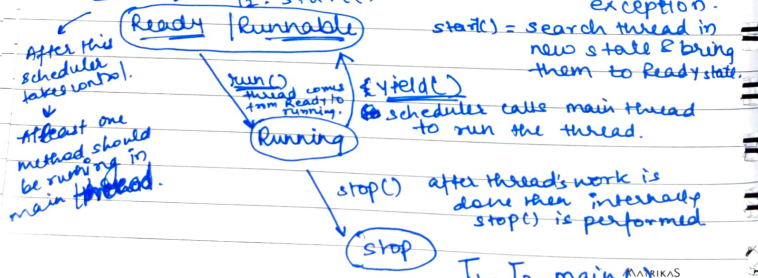
→ Instance method are run by main method.

→ Main thread parent of methods.
main method run by → main thread
eg. main() {
main thread.

★ Thread life cycle.

→ eg. T_1 & T_2 in new state

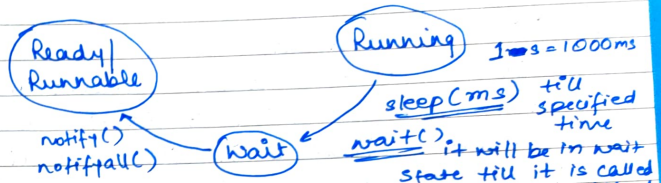
Start() → T_1.start() → do only once otherwise exception.
T_2.start()



start() = search thread in new state & bring them to Ready state.

T_1, T_2, main

Date / /



→ User access → start(), wait(), sleep()
notify(), notifyall().

→ Thread → extend Thread
└→ implements Runnable.

main	T_1	T_2
→ executes main method	common task → 1 class	
→ All other methods which main invokes	different Task	
	Two classes.	
	Two objects.	

★ T_2.start()
T_2.start()
T_2.start()
→ It will give exception (Illegal thread state exception) if not present in new state.

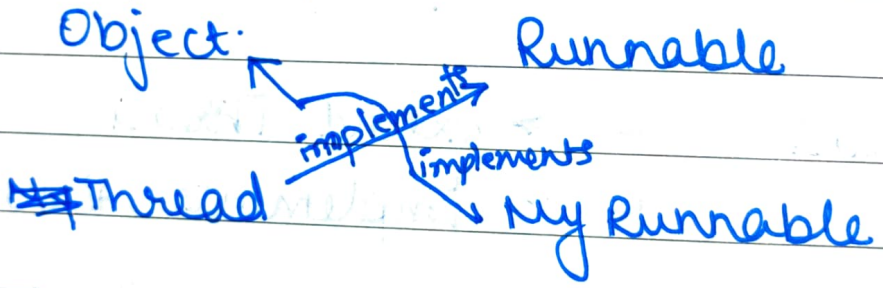
★ Controlling Schedular. by giving it no optn.

→ sleep() method is used. It sends the main method to wait for certain time & gives chance to other methods.

By default threads priority = 5 Date: / /

→ join () method.
- joins two methods, main and other methods, main will wait till other methods finishes its job & main is invoked again & other method is called.

* Runnable



→ My runnable does not have thread as parent.
→ To make a thread explicitly.

eg. MyRunnable r1 = new Runnable();
making thread explicitly.
Thread t1 = new Thread (r1); binds thread to r1
Thread t2 = new Tread (r1);