DATE 01 02 2022

Core Java

Section 1	Section 2	Section 3
Basics	OOPs concepts	JAVA libraries
(1½ months)	(1 months)	(15 days)

Syllabus:

Section 1

- Introduction of JAVA
- JDK, JVM, JRE
- Variables
- Methods
- Scanner Class
- Referance variable
- Object
- JAVA Operators
- JVM memory architecture
- Blocks
- Types of variables
- constructor
- Strings
- Arrays
- Control statements
- Call by value
- Call by references

Introduction of JAVA

DATE 01 02 2022

Q. What is JAVA?

→ - JAVA is general purpose object oriented programming language.
- Genral purpose means we can develop various types of applications by JAVA.
- In other words JAVA is general purpose means it is not domain specific.
- We can develop following types of applications by using JAVA.

e.g. Desktop Application, Web Application, Android Applications.

Q. What are the editions of JAVA language.

→ JAVA programming language distributed in 3 editions:

1) JSE (JAVA Standard Edition)
2) J2EE (JAVA Enterprise Edition)
3) JME (JAVA Micro Edition)

1) JSE (JAVA standard Edition):
- JSE is also known as base edition or 'core Java'.
- Desktop applications can be develop by using core Java.

2) J2EE (JAVA Enterprise Edition):
- J2EE is also known as advanced edition or advance JAVA.
- Web Applications can be develop by using advance JAVA.

3) JME (JAVA Micro Edition):
- JME is also known as mobile edition or android edition.
- Mobile Applications can be develop by using micro edition.

DATE 02 02 2022

Q. explain the application areas of JAVA language?

- Since java is programming language we can develop multiple types of application by using JAVA.
- Following are the application area of JAVA programming language:

1) Desktop Applications:
   - Desktop applications are also known as standlone applications.
   - The application which operates without internet is desktop applications.
   - Core JAVA can be used to develop desktop applications

2) Enterprise Applications:
   - All enterprise applications are business applications
   - We can develop business applications by using core JAVA and Advance JAVA.

3) Android Applications:
   - It is possible to develop android applications by using JAVA.
   - JAVA is useful in the developement of gaming applications.

4) Automation
   - Automation is the technology which can be use in testing.
   - Most of the automation tools are develop by using JAVA language
     e.g. Selenium, Cucumber

5) Big Data Analytics:
   - JAVA language plays an important role in big data technologies.
   - Most of the bigdata tools uses JAVA language

DATE

6) DevOps:
- DevOps stands for development and operations.
- Devops tools are use in application deployment.
- We can use JAVA as a base language for all the devOps tools.

Q. What are the features JAVA?

1) Simple:
- JAVA is simple programming language
- Because we can easily maintain JAVA applications.
- JAVA is independent on other programming language

2) Object Oriented:
- JAVA is object oriented programming language.
- We can develop different applications by using object oriented concepts.
  (i) Inheritance
  (ii) Polymorphism
  (iii) Encapsulation
  (iv) Abstraction

3) Portable:
- JAVA language is a portable programming language.
- Portable means we can port JAVA applications from one enviornment to another enviornment.
- In other words portable means we can change development enviorment for JAVA applications.

4) Platform Independent:
- JAVA is platform indepedent programming language.
- Platform independent means we can execute JAVA application by using any OS platform.

5) Robust:
- JAVA is robust programming language because of its dynamic memory management

Q. explain the development process of java application?
- Since java is compiler based programming language we have to follows 3 steps to develop JAVA Application.
- following are the pheses involved in application development:
  1) coding
  2) compilation
  3) Execution.

1) Coding:
- The process of writing JAVA statements into one file and saving respective file with '.JAVA' extension is known as coding.
- The file which contain the source code is known as source file.

2) Compilation:
- JAVA compailer is responsible for compilation process
- following are the responsibilities of compiler:
  i) check syntax and rules of programming language.
  ii) If syntax is correct then convert english statements into byte code.
  iii) If syntax is incorrect then complailer will throw error will know as 'Compile time error'

3) Execution:
- JVM is software component responsible for execution process.
- After compilation byte code will be stored into .class file which is understandable language for JVM.
- JVM will execute .class file with the help of JIT & JRE.

DATE

Filename.java	Filename.class	JDK
Coding	Compilation 1) check syntax 2) conversion into byte code	Execution JRE JVM + JIT

Q Explain the IT environments in details.

- Every IT organization will have 3 different environment
  1) Development Environment
  2) Testing Environment
  3) Production Environment

1) Development Environment:

- All developers are responsible for coding, compilation and execution.
- We have to install JDK software in development environment

2) Testing Environment:

- Software testers are responsible to test the features developed by application developer.
- In testing environment we can do 2 types of testing:
  i) MANUAL TESTING
  ii) Automation
- Manual testers are responsible only for execution.
- Automation testers are responsible for coding, compilation and execution

3) Production Environment:

- After sucessful testing software will inter into production Environment.
- In this environment application will be tested by client employees which is also known as UAT (User Acceptance Testing)

JDK, JVM, JRE

DATE

Q. What is the use of JDK software

- JDK software is installable software for JAVA programming language.
- Which means we can develop JAVA applications by using JDK software
- JDK Software is having different versions.

JDK 1.0

→ JDK 8.0 (LTS)

JDK 17.0

Q. How to configure JDK software?

- After installation its mandatory to configure JDK software because JDK is an external software.
- Following are steps for configuration:

1) C:→ program files → java → JDK → bin
   ↑ copy path
2) This PC → properties → Adv. system setting → env. variables
3) Create 'path' variable & paste copied path.

Q Explain the components of JDK?

- JDK acts as a container for multiple software components.

1) **Compiler:**
   - Compailer is responsible to convert source code into byte code.

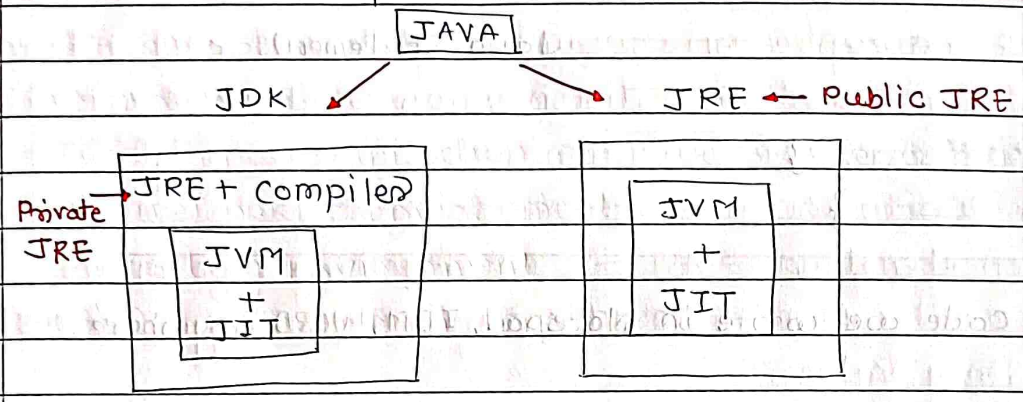
2] **JVM (JAVA Virtal Machine)**
   - JVM is responsible for the execution of byte code
   - JVM acts as an interpreter for JAVA programming language

3] **JIT (Just In Time compiler)**
   - JIT is responsible to convert byte code into machine code

4] JRE (JAVA Runtime Environment):

- JRE provides execution support for JVM.
- It means JVM will execute class file with the help of JRE.
- Without JRE execution is not possible.

Components of JDK



Q. What is the difference between Private JRE and Public JRE.

- JRE present inside the JDK folder is known as Private JRE.
- JRE present outside the JDK folder is known as Public JRE.
- Private JRE is useful for developer (development environment)
- Public JRE is useful for tester (testing environment).

Q. Explain JAVA program stucture

- Since JAVA is object oriented programming language we have to represent everything in terms of class and object.
- To develop JAVA application we have to follows 3 steps:
  1) Coding
  2) Compilation
  3) Execution

DATE

1) **Coding:**

First.JAVA

```
class first
{
    Public static void main (string[] args)
    {
        ——— Java statements
    }
}
```

- Code we write in Notepad, IDM, WORD anywhere.

2) **Compilation:**

JAVAC first.JAVA

3) **Execution:**

JAVA first

eg(i)

```
Class first
{
    Public static void main (string [] args)
    {
        System.out.println ("Welcome to JSE");
    }
}
```

e.g.(ii) File name : Demo.java

```
class Second
{
    Public static void main (string [] args)
    {
        system.out.println ("core java");
    }
}
```

DATE 07 02 2022

- For compilation
  javac Demo.java

- For execution
  java second

Q. It is mandatory to provide file name same as class
- It is not mandatory to provide file name same as class name.
- If file name and class name is different then we have to use class name at the time of execution.
- According to standards of JAVA language file name should be same as class name.

e.g. (3)

```
class Master
{
    Public static void main (string[] args)
    {
        System.out.println ("CORE JAVA");
        System.out.println (25);
        System.out.println ('J');
        System.out.println (true);
    }
}
```

e.g. (iv)

```
class Central
{
    Public static void main (string [] arg)
    {
        System.out.println ("Welcome to \'Jspiders\'")
    }
}
```

For output: welcome to 'Jspiders'

DATE

Q. What is the use of '+' operator?

- It is a binary operation which can be use to perform addition and concatenation.
- If the expression contains only numeric input then addition will takes place.
- If the expression contains string input then concatenation will takes place.
- If the expression contains only character input then JVM will perform addition based on their 'ASCII values'.

Q. What is concatenation:

- Concatenation is the process of joining two inputs.
- concatenation is possible only if expression contains string values.

e.g.

```
Class OperatorDemo
{
    Public static void main (string [] args)
    {
        System.out.println (40+20);                  60
        System.out.println (10.25+30);               40.25
        System.out.println (15.35+25.35);            40.70
        System.out.println (25+"JAVA");              25JAVA
        System.out.println (5+10+"SQL");             15SQL
        System.out.println ("JSE"+12+25+36);         JSE122536
        System.out.println ("JSE"+(12+25+36));       JSE63
        System.out.println ('J'+'A');                139
        System.out.println ('A'+20);                 85
        System.out.println ('C'+"JAVA");             CJAVA
        System.out.println (25+true);  → error
        System.out.println ("25"+true);              25true
        System.out.println ("25"+'10');              2510
    }
}
```

classmate

PAGE

DATE

① 'J' + "SQL" + 'A' + 20 + (10+5)

→ JSQLA2015

② "JAVA" + 5 + ('A' + 10) + 20 + "SQL"

→ JAVA57520SQL

③ 20 + "JAVA" + 10 + "Manual" + 30 + "Aptitude" + 'A' + 'c' + 's'

→ 20JAVA10MANUAL30APTITUDEACS

④ true + 10 + "SQL" + 5 + 10

→ error

⑤ "JAVA" + true + 5 + (2 + false);

→ error

⑥ "A" + 'B' + 20 + (10+5) + true

→ AB2015true

Q. What is Datatype?

- Datatypes canbe used to represent type of information.
- In java datatypes are classified in two chategories:
  1) Primitive
  2) Non-Primitive

Datatypes

- Primitive (keywords)
  - → int - 4bytes
  - → float - 4bytes
  - → double - 8byste
  - → char - 2 bytes
  - → long - 8 bytes
  - → Short - 2 byte
  - → boolean - 1 bit → true / false
  - → byte - 8 bits
- Non-Primitive (Classes)
  - → string

Variables

DATE 08 02 2022

Q. What is Variable?

- Variable is small piece of information.
- We can create variables to store different types of program data.
- Variable is a temporary storage which means we can store the information for short time.
- Variables plays an important role to transfer the data frontend to database.
- Every programming language allows us the create variables.

Q. How to create variables?

- We have to follow three steps in order to create variables:

1) Declaration

- This is a process to creating new variable by using apropriate datatype.

double Price;

2) Initialization

- This is a process of assining initial value to the already declare variable.

price = 25.25.

3) Re-Initialization (optional)

- This is the process of providing a new value to the already intialized variable

Price = 35.25

Price = 45.25

DATE

①
```
Class Product
{
    public static void main(String[] args)
    {
        // declaration
        String productName;
        double productPrice;

        // initialization
        productName = "LAPTOP";
        product Price = 25000.25;

        // print info
        system.out.println(productName);
        System.out.println(productPrice);
    }
}
```

②
```
Class VariableDemo1
{
    public static void main(String[] args)
    {
        // one-step initialization
        String s = "JAVA";

        // re-initilization
        s = "SQL";

        System.out.println(s);
    }
}
```

ANS = SQL

DATE

Q. Problem statement

5-Products (prices) — First make a sum of all the prices — Apply 10% discounts

```
class ProductDemo
{
    public static main void (String [] args)
    {
        // one-step initialization
        double price1 = 25
        doble price2 = 100
        double price3 = 120
        double price4 = 40
        double price5 = 20

        double sum = price1 + price2 + price3 + 40 + 20

        double amount = sum - sum * 0.1;

        System.out.println("TOTAL :" + amount);
        Not mandatory
    }
}
```

Q. Problem statement

quantity =

Price =

total = ?

Apply 5% discount

new amount

↓

Apply 3% GST

Classroom Code: nk37gy2
nk37gy2

DATE

```
class ProductDemo1
{
    public static void main (String [] args)
    {
        // one-step initialization
        int quantity = 2
        double price = 100

        double sum = 100 + 100;
                Total  Price  Price

        double amount = sum - sum * 0.05;
                discount

        double GST = amount + amount * 0.03;

        System.out.println (sum); ("SUM:"+sum)
        System.out.println (amount); ("AMOUNT:"+amount)
        System.out.println (GST); ("TOTAL:"+GST)
    }
}
```

Assignment 1

Q. Write a program to store employee name, basic salary and pf amount. calculate net employee salary. After calculation increment basic salary by 15% and finally display employee salary with and without incrementation.

```
class Employee
{
    public static void main(String [] args)
    {
        // decleration
            String employeeName;
            double basicSalary;
            double pfAmount;
```

```
// initialization
    employeeName = "Prathmesh";
    basicSalary = 25000;
    pfAmount = 2450;

double netsalary = basicSalary + pfAmount;

double incrementsalary = basicSalary + basicSalary X 0.15;

System.out.println("SALARY:" + netsalary);
System.out.println("INCREMENT SAL:" incrementsalary);
}
}
```

Q. Problem Statements:

- Subject of 5 subjects - sum - percentage

```
Class VariableDemo4
{
    public static void main(String [] args)
    {
        double m1 = 68, m2 = 75, m3 = 80, m4 = 60, m5 = 72;

        double sum = m1 + m2 + m3 + m4 + m5;

        double percentage = sum/500 * 100;

        System.out.println("TOTAL SUM:" + sum);

        System.out.println("PERCENTAGE:" + percentage);
    }
}
```

DATE

Q. Problem Statement:

- store product mrp, gst percentage, transport cost
- Apply 10% dealership commission
- display final cost to user

```
Class Product1
{
    public static void main (string[] args)
    {
        double productCost = 750;
        double Gstpercentage = ~~productcost +~~ 8;
        double Transport cost = 60;

        double gstAmout = productcost + product cost x 0.03;

        double total = gstAmount + Transport cost;

        double final = total + total x 10.1;

        System.out.println ("Cost To Dealer:" + total);
        system.out.println ("Cost to User:" + final);
    }
}
```

Q. Re-initialization:

```
class VariableDemo6
{
    public static void main (string[] args)
    {
        int x1 = 25, x2 = 85, x3 = 35;

        System.out.println (x1 + "\t" + x2 + "\t" + x3);
```

→ space tab

```
        x3 = x2; // x3 = 85
        x1 = x3; // x1 = 85
        x2 = x1; // x2 = 85

        system.out.println(x1 + "\t" + x2 + "\t" + x3);
    }
}
```

Q. J A V A
74 65 86 65

```
class VariableDemo7
{
    public static void main(String[] args)
    {
        char ch1 = 'J', ch2 = 'A', ch3 = 'V', ch4 = 'A';
        int no1 = ch1, no2 = ch2, no3 = ch3, no4 = ch4;

        System.out.println(ch1+"\t"+ch2+"\t"+ch3+"\t"+ch4);
        System.out.println(no1+"\t"+no2+"\t"+no3+"\t"+no4);
    }
}
```

Q. Explain diff between keywords and identifier?

Keywords:

- All the keywords are pre-defined words.
- Since the keywords are provided by JAVA language all keywords are reserved in nature.
- Reserved means we cannot use keyword as a variable name, method name or class name.
- It's mandatory to declare all the keywords by using lower case latters.
- Developer cannot modify the keywords.
  e.g. class, public, static, void

METHODS

DATE 10 02 2022

Identifier:

- Identifiers are the user defined words.
- Variable name, method name and class name will be consider as identifier
- Following are the rules to decleare valid identifier
  i) Every identifier should contain ALFA or ALFA Numeric values.
  ii) Identifier should start with either ALPHABATE or '_'

e.g. int 123 par; (invalid)
int par123; (Valid)
int p23; (Valid)

Q. What is Method?

- Method is a collection of java statements.
- Method is used to perform operations on program data.
- Without having method we cannot execute java program.
- In JAVA methods are classified into 2 types:
  1) Internal Methods
  2) External Methods

1) Internal Methods:

- Internal methods are the predefined methods.
- JAVA language has provided multiple internal methods to perform operations on data.
- JVM is responsible for the execution of Internal method
- developer cannot modify structure of internal methods.

e.g. (optional for external methods)

public static void main (string [] args)

Acess modifier | non-acess modifier | Return type | name | parameter

2) External Methods:

- Methods created by programer are known as an external method.
- External methods are also known as user defined or custom methods.

DATE

- Developer is responsible for execution of external methods

e.g.

```
Class Demo1
{   //Internal Method
    public static void main (String [] args)
    {
        System.out.println ("Internal Method");
        run();
        run();
    }
    //External method
    static void run ()
    {
        System.out.println ("External Method");
    }
}
```

Q. What is the output of the program if class does not contain main method?

→ - If JAVA class does not contain main method then program will compailed sucessfully but at a time of execution JVM will show run-time error.

- It is manadatory to decleare main method in order to execute external method.

Q. What is Method chaining?

- The process of calling one external method from another external method and that external method from main method it known as method chaining.

- It is possible to call one external method from another external method at least one external method need to be call from main method.

e.g.

DATE

```
Class Demo2
{
    static void test()
    {
        System.out.println("TEST METHOD");
    }
    Static void display()
    { System.out.println("Display Method");
      test();
    }
    static void ~~display~~ execute()
    {
        System.out.println("EXECUTE METHOD");
        Display();
    }
    public static void main(String[] args)
    {
        System.out.println("MAIN METHOD");
        execute();
        System.out.println("=======");
        execute();
    }
}
```

Q. Explain input parameters?

- Input parameters are the variables declared inside the method signature.
- We can use input parameters to pass run time data to the method.
- It is possible to decleare multiple input parameters inside the singal method.
- In JAVA methods further classified into two types
  1) Parameterise Method.
  2) Non - Parameterise Method

DATE

- The method which accepts input parameters is known as parameterised method.
- If method is parameterised we have to provide same no. of arguments at the time method call.
- The method which does not accepts input parameters is known as non-parameterised method or zero-parameterised method.

e.g.

```
Class Demo3
{ //externalt parameterized method.
  Static void multiply (double a, double b)
  {                      ↳ Parameters (variables)
    System.out.println (a*b);
  }
  public static void main (String[] args)
  {
    multiply (10.2, 20.2);  → arguments (values)
    multiply (25.3, 26.4);
    multiply (60, 36.52);
  }
}
```

Assignment 2

Q. Problem Statements:

i) Add 4 values
ii) Multiply 3 values
iii) Sub 2 values
iv) Div 2 values

Write a program to perform basic arithmetic operations by using external methods with parameters. Create 4 methods and make call for all the methods from main method.

```
class Method1
{
  Static void add (double A, double B, double C, double D)
   {
     System.out.println(A+B+C+D);
   }
  Static void multiply (double A, double B, double C)
   {
     System.out.println(A x B x C);
   }
  Static void sub (double A, double B)
   {
     System.out.println(A-B);
   }
  Static void div (double A, double B)
   {
     system.out.println(A/B);
   }
  public static void main(String [] args)
   {
     add (24.3, 45.6, 85, 89.8); = 194.7
     multiply (5.2, 2.3. 5.9); = 70.564
     sub (1130.3, 230); = 900.3
     div (3500, 65); = 53.846153
   }
}
```

```
class Demo4
{
  public static void main (String [] args)
   {
     area of Circle (2.35);
     area of Circle (9.25);
   }
```

DATE

```
static void areaofCircle(double radius)
{
    double result = 3.14 x radius x radius; [2πR]
    system.out.println("AREA:"+result);
}
}
```

Q. Problem statements

Gross = basicsalary + HRA + incentives

net salary = basic + HRA + i - pf - tax

```
class Demo5
{
    public static void main(String[] args)
    {
        double basicSalary =
        double HRA =
        double incentives =
        double pf =
        double tax =
    }
}
    static void ~~main~~ calculateNetsalary(bs, HRA, in, pf, tax)
    {
        ~~System~~ double result = bs + HRA + in - pf - tax;
        system.out.println("Netsalary:"+result);
    }
    static void calculate grosssalary(bs, HRA, in)
    { double Total = bs + HRA + in
        system.out.println("Gross:"+Total)
    }
    public static void main(String[] args)
    {
        calculate Netsalary(15, 20, 25, 30, 35);
        calculate Net salary(15, 20, 25);
    }
}
```

classmate

PAGE

Q. What is the use of RETURN type.

- Every method java must have a written/Return type.
- Return type represents the type of information which is going to be share by method.
- In JAVA we can have two types of methods
  1) Void Method.
  2) Non-Void Method.

1) VOID Method:

- If method is having return type as a void it means we can not transfer the data from respective
- In other words method is a void which means return statement is absent.

2) NON-VOID

- It is possible to transfer the data from one method to another method by using return statement.
- If method is having return statement then written type should not be void.
- In case methods returns the value then we have to use datatype of respective value as a Return Type.
- We cann't decleare multiple return statements inside the singal method because every method is having only one Return Type.
- Following syntax is used return statement inside method.

Syntax:

```
datatype method name()
{
    ____________
    ____________
    ____________
    ____________
    ____________
    JAVA Statements

    return value;
}
```

e.g.

```
class Demo5
{
    public static void main (String[] args)
    {
        int output = addition();
        System.out.println("ADDITION:"+output);
    }

    Static int addition()
    {
        int result = 25+65;
        return result;
    }
}
```

Q. Why main method is having return type as void?

- Main method is the internal method is responsible for execution.
- Execution starts with main method and ends with method so it is not possible to a return value from main method.
- Because of all these resons main method having return type as void.

- **Method Declerations:**

1. Internal + void + parameterize
2. Internal + void + non-parameterize
3. Internal + non-void + parameterize
4. Internal + non-void + non-parameterize
5. External + void + para
6. External + void + non-para
7. External + non-void + para
8. External + non-void + non-para
9. Next Line method

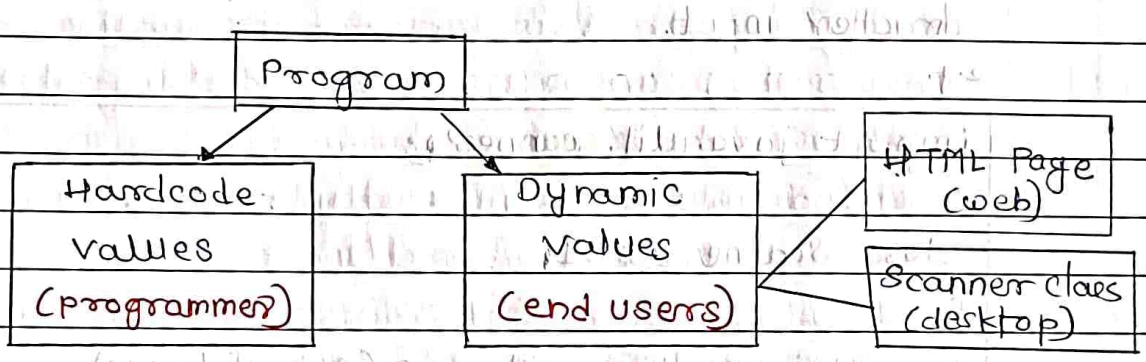
What is diff. between hardcore and dynamic value.

1) Hardcode Value:

- Values provided by programmer in a java program are known as hardcode values.
- Hardcode values are the constant values.

2) Dynamic Values:

- Values provided by end user are known as dynamic values.
- Dynamic values are the non-constant values.
- To accept input from user we have different options available.
  i) HTML page for web applications
  ii) Scanner class for desktop applications



How to accept input by using Scanner Class?

- Scanner class is an internal class declared inside the java.util package.
- We can accept different types of dynamic values by using scanner class.
- To accept input from user we have to follow 3 steps:

① Import the scanner class from java.util package.

```
Import java.util.scanner;
```

② After import apperation we have activate scanner class by creating an object.

```
Scanner Scan = new Scanner(System.in);
```

Ref. variable — Object — Accept i/p.

DATE

(B) Scanner class has provided several methods to accept input from users.

method name: next() → string input
next int() → integer input
next float() → float input
next double() → double input
next long() → long input
next short() → short input
next Boolan() → Boolan input
next byte() → Byte input
next

- Scanner class has not provided any method to accept character input.

e.g.

```
import java.util.scanner;

Class ScannerDemo1
{
    public static void main (string[] args)
    {
        Scanner scan = new scanner (system.in);
        system.out.println("ENTER NAME");
        String name = scan.next();

        System.out.println("Welcome"+name);
    }
}
```

Q. Enter first no ____ (Statements Que)
Enter Second no ____
Enter third no ____
Total: ____ Avg: ____

DATE

```
import java.util.Scanner;

Class ~~Demo~~ ScannerDemo2
{
    public static void main(String[] args)
    {
        Scanner scan = new scanner (System.in);
        System.out.println("ENTER FIRST NUMBER");
        double no1 = scan.nextdouble();

        System.out.println ("ENTER SECOND NUMBER");
        double no2 = scan.nextdouble ();

        System.out.println ("ENTER THIRD NUMBER");
        double no3 = scan.nextdouble();

        double Total = no1 + no2 + no3;
        system.out.println ("TOTAL:" + Total);

        double Avg = Total/3
        System.out.println(
    }
}
```

```
import java.util.scanner;

class ScannerDemo3
{
    public static void main (String [] args)
    {
        Scanner sc1 = new scanner(system.in);
        System.out.println ("ENTER QTY");
        int qty = sc1.next INT();
```

main method (data) → external method (parameters)

external (return) statement → main

DATE 15 02 2022

```
System.out.println ("ENTER PRICE");
double price = sc1.nextDouble();

calculateBill (qty, price);
}
static void calculateBill (int qty, double price)
{
double total = qty x price;
System.out.println ("Total AMOUNT:" + total);
}
}
```

Q. What is use of nextLine method?

- nextLine method is declare inside the scanner class
- This method can be use to accept string value which contains white spaces.
- It is possible to accept multiple words from a single line with the help of nextLine method.

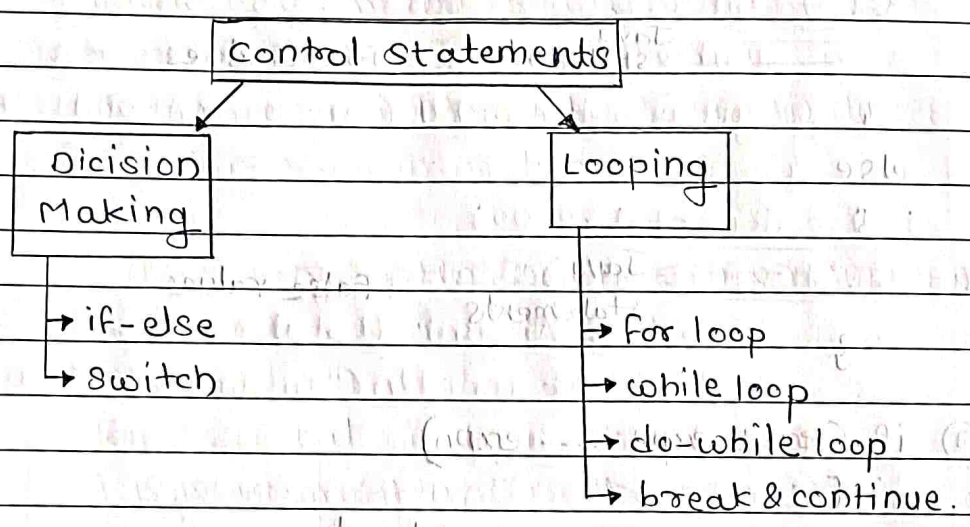
```
import java.util.Scanner;
class ScannerDemo4
{
public static void main (string [] args)
{
Scanner sc1 = new Scanner (System.in);

System.out.println ("Enter Subject Name");
String subject = sc1.nextLine(); (core JAVA)
                                  (O/P)

System.out.println ("SUBJECT:" + subject);
}
}
```

Control Statements

DATE

Q. What is the use of control statements?

- Control statements are mainly use to control the flow of program execution.
- Every programming language uses control statements to control the flow of execution.
- JAVA control statement classified into 2 types.
  1) Dicision making statements
  2) Looping statements.



Q. Explain the use of dicision making statements.

- Dicision making statements are use to apply conditions or restriction on program data.
- In java we can use two statements in order to apply condition and restriction.
  i) if-else statement
  ii) Switch statement
- It is possible to apply various conditions by using if-else or switch statement.

Q. Explain working of if-else statement?

- if-else is one of the dicision making statements which can be used to decide the flow of program execution.
- Following are the possible combinations of if-else

DATE

Combinations:

1) if (exp1, exp2....., expn)
{
Java statements
}

2) if (exp1, ... expn)
{
Java statements — True values
}
else
{
Java statements — False values
}

3) if (exp1, exp2 ..... expn)
{
Java statements
}
else if (exp1..exp)
{
Java statements
}

4)
if (exp1, exp2 ..... expn)
{
if (exp1, exp2 ..... expn)
{
Java statements
}
}

DATE

e.g.①
```
import.java.util.scanner;

class ControlDemo1
{
    public static void main(String[] args)
    {
        scanner sc1 = new scanner(System.in);

        System.out.println("ENTER NUMBER");
        int no = sc1.nextInt();
        {
        if (no%2==0)
          {
            system.out.println("NO IS EVEN");
          }
        else
          {
            system.out.println("NO IS ODD");
          }
    }
}
```

e.g.②
```
import.java.util.scanner;

class ControlDemo2
{
    public static void main(String [] args)
    {
        scanner ~~out.println~~ sc1 = new scanner (system.in);
        system.out.println ("ENTER QTY");
        double qty = sc1.nextDouble();

        System.out.println ("ENTER PRICE");
        double price = sc1.nextDouble();
```

DATE

```
    Calculate (Qty, price);
  }
  Static void calculate (double qty, double price)
  {
    double total = qty x price
    System.out.println ("TOTAL:" + total);

    if (total < 2000)
    {
      double discount amount = (total x 0.05);
      System.out.println ("Discount AMOUNT:" + discount amount)
    }
    else if (total > 2000)
    {
      double disAmount = (total x 0.1);
      system.out.println ("DisAmount:" + dis amount);
    }
  }
}
```

Q. Assignment

Problem Statements:

NUMBER1

NUMBER2

(i) Number1 is greater than no2

(ii) Number2 is greater than no1

(iii) Number2 is equal to Number2

DATE

```
import java.util.Scanner;

class ControlDemo1
{
    public static void main(String[] args)
    {
        Scanner sc1 = new Scanner(System.in);
        System.out.println("NUMBER 1");
        double no1 = sc1.nextDouble();

        System.out.println("NUMBER 2");
        double no2 = sc1.nextDouble();

        if (no1 > no2)
        {
            System.out.println("Number1 is greater than Number2");
        }
        if (no1 < no2)
        {
            System.out.println("Number2 is greater than Number1");
        }
        if (no1 == no2)
        {
            System.out.println("Number1 is equal to Number2");
        }
    }
}
```

If we write (no1 = no2)

Error: double cannot converted to boolen.

DATE 16 02 2022

Q. cmd :

Select employee type
1) HR (+20%)
2) Manager (+30%)
1/2 (Accept choice)
Enter Salary ____
salary ____
Incremental ____

```
import.java.util.Scanner;

class controlDemo2
{
    public static void main(Strings[]args)
    {
    Scanner sc1=new scanner(System.in);

      System.out.println("Select Employee Type");
      int type= sc1.nextInt();
      System.out.println("1:HR\n 2:MANAGER);

      System.out.println(" Enter Salary");
      double salary = sc1.nextDouble();

    if (type==1)
{
    getHRsalary(salary);
}
else if (type==2)
{
    getManagersalary(salary);
}
else
{
    System.out.println("Invalid Salary"or Type")
}
```

classmate PAGE

DATE

```
}
static void getHRsalary (double salary)
{
    double updated salary = salary + salary x 0.2;

    System.out.println("SALARY:" + salary);
    system.out.println("INCREMENT:" + updated salary);
}
static void getManagerSalary (double salary)
{
    double updated salary = salary + salary x 0.4;

    System.out.println("SALARY:" + salary);
    system.out.println("INCREMENT:" + Updated salary);
}
}
```

Q. cmd

```
select mode of operation
1) Area of square
2) Area of triangle

1 / 2 / 3 Accept choice

Enter side (if 1)

Enter base & height (if 2)

Result (Area)
```

DATE

```
import.java.util.Scanner;
class ControlDemo3
{
  public static void main(String[] args)
  {
    Scanner sc1 = new Scanner(System.in);

    System.out.println("Select mode of operation");
    System.out.println(1: Area of square \n
                        2: Area of triangle);
    int type = sc1.nextInt();
```

DATE

```
import java.util.Scanner;
class ControlDemo3
{
  public static void main(String[] args)
  {
    Scanner sc1 = new Scanner(System.in);

    System.out.println("ENTER NUMBER");
    int no= sc1.nextInt();

    if (no>0)
    {
      if (no%3==0)
      {
        System.out.println("NUMBER IS POSITIVE AS WELL
                                AS DIVISIBLE BY 3");
      }
      else
      {
        System.out.println("NUMBER IS POSITIVE AND
                                NOT DIVISIBLE By 3");
      }
    }
    else
    {
      if (no%7==0)
      {
        System.out.println("NUMBER IS NEGATIVE AS
                                WELLAS DIVISIBLE BY 7");
      }
      else
      {
        System.out.println("NUMBER IS NEGATIVE AND
                                NOT DIVISIBLE BY 7");
```

DATE 17 02 2022

Q. Explain industry naming standards.

- Industry naming standards are nothing best practices which we have to follow when writing source code.

1) Every class name or interface name should start with capital letter and subsequent word also start with capital letter

e.g. class Demo
class StudentInfo

2) Every variable name or method name should start with small case latter and subsequent word capital letter.

e.g. int id
int studentId

Q. What is Business Logic Class?

- It is possible to decleare multiple classes inside the .JAVA file
- If the java file contains more than one classes then only one class have main method.
- The class which contains main method is known as an executable class / Main class.
- The class which does not contain main method will be considered as Business Logic Class or functional Class.
- It is possible to create multiple business logic classes inside the singal Java file.
- File name should be same as class name which contains main method.

Q. Explain members of the class.

- Anything decleare inside the class body will be considerd as member of class.
- A class can have two types of members
  1) Data Members
  2) Function Members.

DATE

- Data members are the nothing but variables declare inside the class.
- Function members are nothing but methods declare inside the class.
- Data and function members can be static or non-static.

Members
- Data Member
  - Static
  - Non-Static
- Function Member
  - Static
  - Non-Static

Q. Explain static members of class.
- The members which are declare by using static keywords are known as static member.
- Static means common accesible resource.
- Static members will have single copy into memory.
- To access startic members from another class we have to provide class name as referance.

Q. Explain Non-Static Members.
- The members which are declare without using static keyword are known as Non-static
- Non-Static means dynamic resource.
- Non-static members will have multiple copies into memory.
- To access Non-Static Members from another class we have to provide object reference.

DATE

```
// business logic class
class Demo
{
  static int K=20;

  static void test()
   {
     System.out.println("TEST METHOD");
   }
}
// executable class
class MainApp1
{
  public static void main (String [] args)
   {
     System.out.println(Demo.K);
     Demo.test();
   }
}
```

```
// business logic class
class Sample
{
  // non-static data members
  int p=20;

  // non-static function members
  void test()
   {
     System.out.println("TEST METHOD");
   }
}
class MainApp2
{
  public static void main (String [] args)
```

DATE

```
{
    System.out.println(new Sample().p);
    new sample().test();
}
}
```

eg.③

```
import java.util.Scanner;
// business logic class
Class Circle
{
    Static double pi = 3.14;
    // Area
    Void area(double radius)
    {
        double result = pi x radius x radius;
        System.out.println("AREA:"+result);
    }
    // Circumferance
    Void circumference(double radius)
    {
        double result = 2x pi x radius;
        System.out.println("CIRCUMFERANCE:"+result);
    }
}
// executable class
class MainApp3
{
    public static void main(String [] args)
    {
        Scanner sc = new Scanner(System.in)
        System.out.println("Enter Radius");
        double rad = sc.nextDouble();
        System.out.println("SELECT MODE:");
        System.out.println("1. Area");
```

```
    system.out.println("2:CIRCUMFERANCE");
    int type = sc1.nextINt();

  if (type==1)
  {
    new circle().area(rad);
  }
else if (type==2)
  {
    new circle().circumferance(rad);
  }
else
  {
    system.out.println("Invalid Type");
  }
}
}
```

eg.④ statements

Enter current reading

Enter previous reading

Select customer:

① Residential (unit=4.75)

② Commercial (unit=5.75)

Result:

```
import.java.util.Scanner;
// business logic class
class Electricity Bill
{
  static void unit1 = 4.75
  static void unit2 = 5.75
```

DATE

```
void getResidentialBill(double current, double previous)
 {
   double total = current - previous;
   double amount = total x unit1;
   System.out.println("Total Amount:"+amount);
 }

void getCommercialBill(double current, double previous)
 {
   double total = current - previous;
   double amount = total x unit2
   System.out.println("Total Amount:"+amount);
 }
}
// executable class
Class MainApp4
 {
   public static void main(String [] args)
   {
     Scanner sc1 = new Scanner(System.in);
     System.out.println("Enter current reading");
     double current = sc1.nextDouble();
     System.out.println("Enter Previous reading");
     double previous = sc1.nextDouble();
     System.out.println("Enter Customer Type");
     System.out.println("1: Residential Customer");
     System.out.println("2: Commercial Customer");
     double type = sc1.nextDouble();

 if (type==1)
  {
    new ElectricityBill().getResidetialBill(current, previous);
  }
```

```
else if
  {
    new ElectricityBill().getCommercialBill(current, previous)
  }
else
  {
    System.out.println("Invalid Type");
  }
 }
}
```

Q. Difference between static and Non-static members.

Static Members	Non-static members
1) static members are common accesible resource.	1) Non-static members are dynamic resources.
2) static members are having single copy into JVM memory	2) Non-static members are having multiple copies into JVM Memory.
3) These members are the permenent members of class.	3) Non-static members are having multiple copies into JVM memory
4) Static variables are also known as class variable.	4) These variables are also known as instance variables.

DATE 22 02 2022

Q. Explain accesibility rules.

Rules for static context:

1) We access static members inside the static context.
2) If they belongs to same class then direct access posible
3) To acess static members from different class we have to provide class name as referance.
4) Its mandatory to create an object of the class to access non-static members inside the static context.

Rules for Non-static context:

- We can access non-static members inside the non-stat context.
- Direct Access is possible if they belongs to same class.
- We have to create an object of respective class to acess non-static members from another classes.
- It is possible to access static members from non-static context. If they belongs to same class then direct access is possible else we have to provide class name as referance.

1)
```
class Demo
{
  int K=110;
  void test ()
  {
    System.out.println (K);
  }
}
```
⇒ Valid

2)
```
class sample
{
  double d = 10.50;
  static void display ()
```

DATE

```
{
  s.o.p(d);
 }
}
```

⇒ Invalid

3) class central

```
{
 static int k=10;
 void display()
 {
  s.o.p(k);
 }
}
class master
{
 void test()
 {
  central display();
 }
}
```

⇒ Invalid

4) class sample

```
{
 static  a = false;
 double b = 10.50;
}
class demo
{
 valid run(c)
 {
  s.o.p (sample.b);
 }
```

Referance Variables

DATE

```
void run
{
  s.o.p(sample.a);
 }
}
```

⇒ Invalid

5) Class Master
```
{
  int j=50;
 }
Class Central
 {
   void test()
   {
     s.o.p(new.master().j);
   }
 }
Class Demo
  {
    public static void main(string[]args)
    {
      central.test();
    }
  }
```

⇒ Invalid

Q. What is referance variable.

→ - Referance variable is a class type variable.
- It means the variable which is decleared by using class name will be considered as referance variable.
- The default value for referance variable is always null.
- By using referance variable we can store the add of an object.

DATE

- It is possible to acess multiple non-static members by using referance variable.
- If we print referance variable then add of an object will get printed.
- One object can have multiple references.

e.g.

```
// Business logic class
Class Demo
{
   int K = 20;
   int j = 40;
}
// executable class
class MainApp's
{
   public static void main (String [] args)
   {
      Demo d1 = new Demo ();   // First copy
      System.out.println (d1.k);
      System.out.println (d1.j);

      Demo.d2 = new Demo();  // 2nd copy
      System:out.println(d2.k);
      System.out.println (d2.j);

      System.out.println(d1);
      System.out.println(d2);
   }
}
```

object variable

JVM memory

1st copy	2nd copy
int k = 20	int k = 20
int j = 40	int j = 40
Demo@123abc (address)	Demo@456abc (address)

e.g.② //business logic class

```
class Employee
{
    int employeeID;
    String employeeName;

    Void acceptDetails (int id, string name)
    {
        employeeID = id;
        employeeName = name;
    }
    void displaydetails ():
    {
        System.out.println ("Employee ID:" + employeeID);
        System.out.println ("Employee Name:" + employeeName);
    }
}
Class MainApp6
{
    public static void main (String [] args)
    {
        Employee e1 = new Employee ();
        e1. acceptdetails (101, "DINESH");
        e1. display details ();

        Employee e2 = new Employee ();
        e2. acceptdetails (102, "ASHWIN");
        e2. display details ();

        System.out.println (e1);
        System.out.println (e2);
    }
}
```

DATE

e1	e2
emp1 = 101	empID = 102
empName = DINESH	empName = "ASHWIN
@copy1	@copy2

e.g.③

```
// business logic class
class Facebook
{
    void loginApp()
    {
        System.out.println("Logged in through Application");
    }
    void loginwebsite()
    {
        System.out.println("Logged in through webpage");
    }
}
// Executable class
Class MainApp7
{
    public static void main(String[] args)
    {
        Facebook f1 = new Facebook();
        f1.loginwebsite();

        Facebook f2 = f1;
        f2.loginapp();

        System.out.println(f1);
        System.out.println(f2);
    }
}
```

} address will be same

Facebook
loginApp()
loginWeb()
@copy

classmate

PAGE

Object

DATE

Q. What is an object?

- An entity having its own state and behaviour will be considered as an object.
- State represents characteristics of an object where as behaviour represents functionality of an object.
- In case of JAVA object is a copy of non-static members.
- Non-static data member represents state of an object where as non-static function member represents behaviour of a object.
- Class is a container for object state and behaviour.
- In java object is also known as instance of the class.

Object
- State → Characteristics
- Behaviour → Functionality

eg①
```
//object
class Laptop
{
  //state
  String company="LENOVO";
  double price=50000.25;
  int ram size=8;
  boolean isGraphic Enabled=true;
  //behaviour
  void runOS()
  {
    System.out.println("Running windows OS");
  }
  void displaypicture()
  {
    System.out.println("Display picture");
  }
```

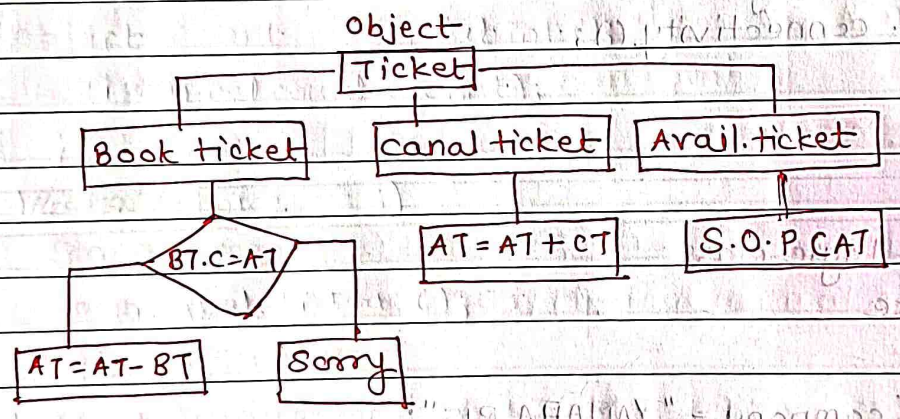
```
void connectivity ()
{
  System.out.println ("connect different devices");
 }
}
// Executable class.
Class MainApp1
{
  public static void main(String [] args)
  {
    Laptop l = new Laptop ()
    System.out.println ("COMPANY" + l.company);
    System.out.println ("PRICE" + l.price);
    System.out.println ("RAM" + l.ramsize + "GB");
    System.out.println ("IS GRAPHICS Enable" + is graphic
                                                enabled);

    l.runOS ();
    l.displayPicture ();
    l.connectivity ();
  }
}
```

eg.② // business logic class

```
Class Bike
{
  String company = "YAMAHA R15";
  double price = 214000.1;
  int fuel tank = 12

void fueleconomy ()
{
  System.out.println (" 40 km per litre")
}
```

```
void maxspeed()
{
    System.out.println("130 km per hour")
}
// Executable class
Class MainApp2
{
    public static void main(String [] args)
    {
        Bike b1 = new Bike();
        System.out.println("COMPANY:"+company);
        System.out.println("Price:"+price);

        b1.fueleconomy()
        b1.maxspeed()
    }
}
```

Q.



cmd.

Select mode

1. Book ticket
2. canal ticket
3. Aval. ticket

I/II/III

Enter no. of ticket

Enter no. of ticket

Apply count

calculate bill

DATE 24 02 2022

```
import java.util.scanner
class Ticket
{
    static int costperticket = 100;
    static int availableticket = 60;
  void booktickets (int countticket);
   {
      if (book count <=availableticket);
       {
        availableTickets = availableTickets - bookcount;
        int total = book count * cost perticket;
        System.out.println("Total Amt:"+total);
        system.out.println("Tickets Have Been book sucessfully");
       }
      else
       {
        system.out.println("sorry tickets are no available");
       }
    }
  void cancelTicket (int cancelcount)
    {
      availableTickets = availableTickets + cancelcount;
      System.out.println("ticket have been cancel sucessfully");
    }
  void checkAvailability
    {
      System.out.println("AVAILABLE TICKETS");
    }
}

class MainApp2
{
  public static void main (string [] args)
   {
```

DATE

```
Scanner sc1 = new Scanner(System.in);
Ticket t1 = new Ticket();
System.out.println("Select mode of Ticket");
System.out.println("1: Book Ticket");
System.out.println("2: Cancel Ticket");
System.out.println("3: Available check");

int choice = sc1.nextInt();

if (choice==1)
{
System.out.println("Enter no.of tickets");
int count = sc1.nextInt();
t1.bookTickets(count);
}
else if (choice==2)
{
System.out.println("Enter no. of tickets");
int count = sc1.nextInt();
t1.cancelCount(count);
}
else if (choice==3)
{
t1.availeTickets();
}
else
{
System.out.println("Invalid Choice");
}
}
}
```

Operators

DATE 25 02 2022

Q. Explain JAVA operators.

- Operators are used to perform operations on program data.
- In JAVA operators are classified:
  1) Arithmetic
  2) Assignment
  3) Relational
  4) Unary
  5) Logical
  6) Bitwise
  7) Ternary

Q. Explain Arithmetic operators

Arithmetic operators:

- Arithmetic operators can be use to perform basic arithmetic operations on program data.

Operator	Usage	Example
+	Addition & concatenation	c = a+b
-	Subtraction	c = a-b
*	Multiplication	c = a*b
/	Division	c = a/b
%	Modulo (get reminder)	c = a%b

Q. Explain Assigment operators

Assigment Operators:

- Assignment operators can be use assign value to specific variable.

Operator	Usage	Example
=	Assignment	a = 10
+=	Addition & Assignment	a += b (a = a+b)
-=	Substraction & Assignment	a -= b (a = a-b)
*=	Multiplication & Assignment	a *= b (a = a×b)

/=	Division and Assignment	a/=b (a=a/b)
%	Modulo and Assignment	a%=b (a=a%b)

Q. Explain Relational operators:

Rational operators:

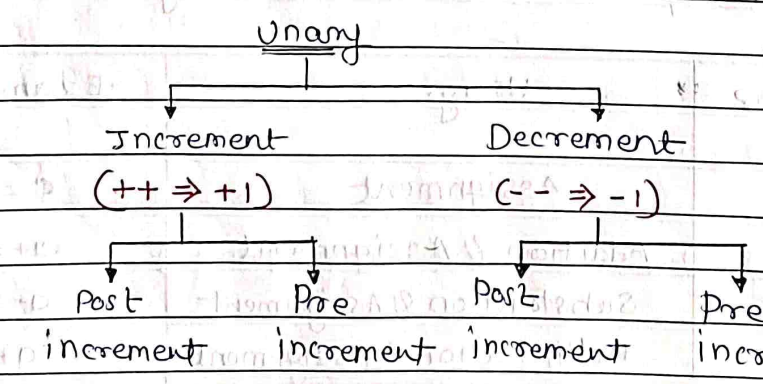
- Relational operators are also known as comparision op
- We can compare two or more inputs by using relational operators.
- Output of relational operation is a boolean value.

operator	Usage	Example.
==	check for equality	a==10
!=	Negation of equality	a!=b
<	less than	a<b
>	greater than	a>b
<=	less than equal	a<=b
>=	greather than equal	a>=b

Q. Explain Unary Operators.

Unary Operators:

- Unary operators are mainly use to perform operations on singal variables.
- Unary operations can be done by:
  1) Increment
  2) Decrement



DATE

Post Increment:

- Post increment means first use current value inside the expression and then increment original value by +1.

Pre Increment:

- It means first increase the original value by +1 and then use inside the expression.

Post Decrement:

- It means first use the current value inside the expression and decrease the original value by -1.

Pre decrement:

- It means first decrease the original value by -1 and then use inside the expression.

e.g.

1) int x=10;
   int y;

   Normal addition

   y = x++ ⊕ x ⊕ ++x; (10+11+12)

   S.O.P (x); 12

   S.O.P (y); 33

2) int a=2;
   int b=4;

   b = a++ ⊕ ++b ⊕ ++a ⊕ a-- ⊕ a; 3+5+4+3=15

   2+5+4+4+3;

   S.O.P (a); 3

   S.O.P (b); 18

DATE

3]

```
int p=4; 5
int q=6;

q = p++ + q++ + p + q + --p;      5+7+5+7+4=28
                                  4+6+5+7+4=26
S.O.P(p); 4
S.O.P(q); 26
```

4]

```
int d=5;
int e=10;

e = ++d + ++e + --d + ++d + d;

SOP(d); 6
S.O.P(q); 34
```

5]

```
int p=10; 11
int q=2;

q = ++p + p++ + --p + q + p      10+11+12+2+11 = 46
if (++p>12)
  {
    p++;
    q--;
  }
else
  { p--; 11
    q---; 45
  }
```

Q. Explain logical operators.

5] Logical Operators:

- Logical operators can be use to perform logical operations on program data.
- We can use logical operations along with relational operation/operators.

logical and logical operator

1) &&

IN1	IN2	O/P
T	T	T
T	F	F
F	X	F

2) ||

IN1	IN2	O/P
T	X	T
F	T	T
F	F	F

3) logical not (!)

- This operator always returns negation (opposite) of original output.
- We can use logical not operator along with other logical operators

e.g. i) int x = 12;
int y = 36;
int z = 24;

s.o.p (x > y && y < z) → False

s.o.p (z > x || y > x) → True

DATE

(ii)

int a = 10;

int b = 20;

S.O.P ((++a > --b)) → True

S.O.P (a++ > 10 && b++ < 20) → False

(iii) int p = 14;

int q = 6;

5 + 6 + 14 + 13 = 48

q = ++p + q + p-- + p; 15 + 6 + 15 + 14

if ((++p > 15 && --q < 25))

13 > 15 F F

{

p++; 14

q--; 50

}

S.O.P (p) → 15

S.O.P (q) → 50

(iv) int a = 20;

int b = 15;

19 + 16 + 20 + 20

b = --a + ++b + ++a + a;

if ((++b < 50 || --a < 20) && (--b > 40 || ++a > 20))

(15 < 50 T 19 T

{

a++; → 20

b--; → 74

}

(v) int x = 12 15 14 13 14

int y = 24 64 65 64 63

12 + 24 + 13 + 14 = 63

y = x++ + y++ + x++ + x;

if ((++y < x++ && ++x < y++) || (x-- < y++ && --y > x--))

62 F T 13 < 6 T 61 > 12

{

x++; 19 → 14

y--; 64 → 63

}

classmate

PAGE

DATE

Q. Explain Bitwise operators:

Bitwise Operators:

- Bitwise operation are used to perform bitwise operations on program data.
- We can perform operations on individual bits by bitwise operators
- bitwise operation are possible only if input contains integer values.
  e.g. int, long, short, byte
- To perform bitwise operation every defined value will be converted into binary format.

1) Bitwise &:

IN1	IN2	O/P
T(1)	T(1)	T(1)
T(1)	F(0)	F(0)
F(0)	T/F	F(0)

2) Bitwise |:

IN1	IN2	O/P
T(1)	T/F	T(1)
T(1)	F(0)	T(1)
F(0)	F(0)	F(0)

e.g. (i) int a = 18;
int b = 14;
int c = a & b
= 18 & 14
= 2

	32	16	8	4	2	1
18 →	0	1	0	0	1	0
14 →	0	0	1	1	1	0
	0	0	0	0	1	0

DATE

(ii) int a = 35

int b = 29

int c = a|b → 63

int d = a&b → 1

(c=)

	32	16	8	4	2	1
35 ⇒	1	0	0	0	1	1
29 ⇒	0	1	1	1	0	1
	1	1	1	1	1	1

= 32+16+8+4+2+1 = 63

(d=)

32	16	8	4	2	1
1	0	0	0	1	1
0	1	1	1	0	1
0	0	0	0	0	1

= 1

(iii) int p = 19;

int q = 42;

int r = p|q; → 59

int s = p&q; → 2

int t = r&s; → 2

(p|q)

	32	16	8	4	2	1
p →	0	1	0	0	1	1
q →	1	0	1	0	1	0
R →	1	1	1	0	1	1

→ 32+16+8+2+1 = 59

(p&q)

	32	16	8	4	2	1
p →	0	1	0	0	1	1
q →	1	0	1	0	1	0
s →	0	0	0	0	1	0

→ 2

(R&S)

	32	16	8	4	2	1
R →	1	1	1	0	1	1
S →	0	0	0	0	1	0
	0	0	0	0	1	0

→ 2

(iv)
```
int x = 9;
int y = 12;
if (++x > 9 && --y < 12)
  {
    x++
  }
  else;
  {
    y--;
  }
```

3) << Left shift operator:

- Left shift operator can be use to shift binary bits to the left side by specifying no. of
- shifting the bit by one position which means x2.

4) >> Right shift Operator:

- Right shift operator can be use to shift binary bits to the right side by specified.
- In case of right shift operator end point is fixed which is 1. It means after shifting if bits crosses first position then output will be 0 (zero).

e.g.

(1) int x = 15;
int y = x << 2;
= 60

shortcut
1st → 15 x 2 = 30
2nd → 30 x 2 = 60

	32	16	8	4	2	1	
org	0	0	1	1	1	1	
1st shift →	0	1	1	1	1	0	
2nd shift →	1	1	1	1	0	0	

32 + 16 + 8 + 4 = 60

DATE

e.g.

(i) int x = 15;

int y = x>>2

= 3

	32	16	8	4	2	1
Org →	0	0	1	1	1	1
1st →	0	0	0	1	1	1
2nd →	0	0	0	0	1	1

(ii)

int x = 22

int y = x<<5

= 704

1st → 22 × 2 = 44
2nd → 44 × 2 = 88
⋮
5th → 352 × 2 = 704

(iii) int p = 34;

int q = p>>4;

	32	16	8	4	2	1
	1	0	0	0	1	0
1st →	0	1	0	0	0	1
2nd →	0	0	1	0	0	0
3rd →	0	0	0	1	0	0
4th →	0	0	0	0	1	0
	0	0	0	0	1	0 → 2

Q. Explain Ternary operator.

Ternary Operator:

- Ternary operator act as a decision making statements.
- We can apply conditions/restriction on program data by using ternary operator.
- Following is the syntax:

```
Variable = (condition1 ....., n)? "exp1" : "exp2";
```

(condition1 ....., n) → if; "exp1" → P; "exp2" → else

Q. write a program to check no. is positive or negative without using if else statement.

```
importjava.util.Scanner;
class MainApp1
{
  public static void main (String [] args)
  {
    Scanner sc1 = new Scanner (System.in);
    System.out.println ("ENTER NUMBER");
    int no = sc1.nextINt ();

    String result = (no>0)? "POSITIVE": "NEGATIVE";

    System.out.println ("NO IS"+result);
  }
}
```

Q. Select color

1) RED
2) GREEN
3) BLUE

choice=:

Msg

Without using if-else.

DATE

```
import java.util.Scanner;
class Demo
{
  public static void main(String [] args)
  {
    Scanner sc1 = new Scanner(System.in)
    System.out.println("Enter Color");
    System.out.println("1: RED"); \n
    System.out.println("2:Green");
    System.out.println("3: BLUE");
    System.out.println("Select Type:");
    int type = sc1.nextInt();

    String color = (choice==1)?"RED":((choice==2)?"Green":
                                                  BLUE");
                                  :(choice==3)?"BLUE":"Invalid

    System.out.println("Selected color:"+color);
  }
}
```

Q. Cmd

Accept Number

check whether number is divisible by 5, 3 or both and display msg. also not divisible by both.

```
import java.util.Scanner;
class Demo1
{ public static void main(String [] args)
  {
    Scanner sc1 = new Scanner(System.in);
    System.out.println("Enter Number:");
    double No = sc1.nextDouble();
```

Types of Variables

DATE

```
sting result = (no/5 == 0 && no/3 == 0)? "Both divisible by 5&3"
                : ((no/3 == 0)? "Divisible by 3"
                : ((no/5 == 0)? "Divisible by 5"
                : Not divisible by Both));

system.out.println("No IS" + Result);
}
}
```

Q. Explain Types of variables.

Types of variables

- In java variables are classified into 3 times.

1) class variable
2) Instance variable
3) local

Class variable	Instance variable	local variable.
- variables decleare inside the class but outside the method are known as class variables.	- variables decleare inside The class but outside the method without using static keywords are known as instant variable	- variables decleare inside the method block are known as local variable.
- All class variables are knowns as static variable.	- All instance variables are known as non-static variable.	- lotale variable can be static or non-static depends on method type.
- All class variables will be store into static pool area.	- All instance variables will be stored into heap area.	- All local variable will be stored into stack.

DATE

- We can access class variables anywhere inside same class as well as outside class by using classname referance.	- we can acess instance variables anywhere inside same class as well as outside class by using object reff.	- It is not possible to acess local variable outside the methods.
- class variables will have default values	- Instance variables have default values.	- It's mandatory to initalized local variables sience they don't have default value

JVM Memory Architecture

DATE 03 03 2022

Q. Explain JVM memory Architecture.

- Every software having its own architecture.
- Java programming language has also having this own memory architecture which known as JVM memory Architecture
- JVM Memory Architecture divide in following section:
  i) Class loader subsystem
  ii) Run time memory
  iii) execution engine

1) Class loader subsystem:

- class loader subsystem is responsipal to conduct following applications
  i) Loading
  ii) Linking
  iii) Initialization

1) Loading:

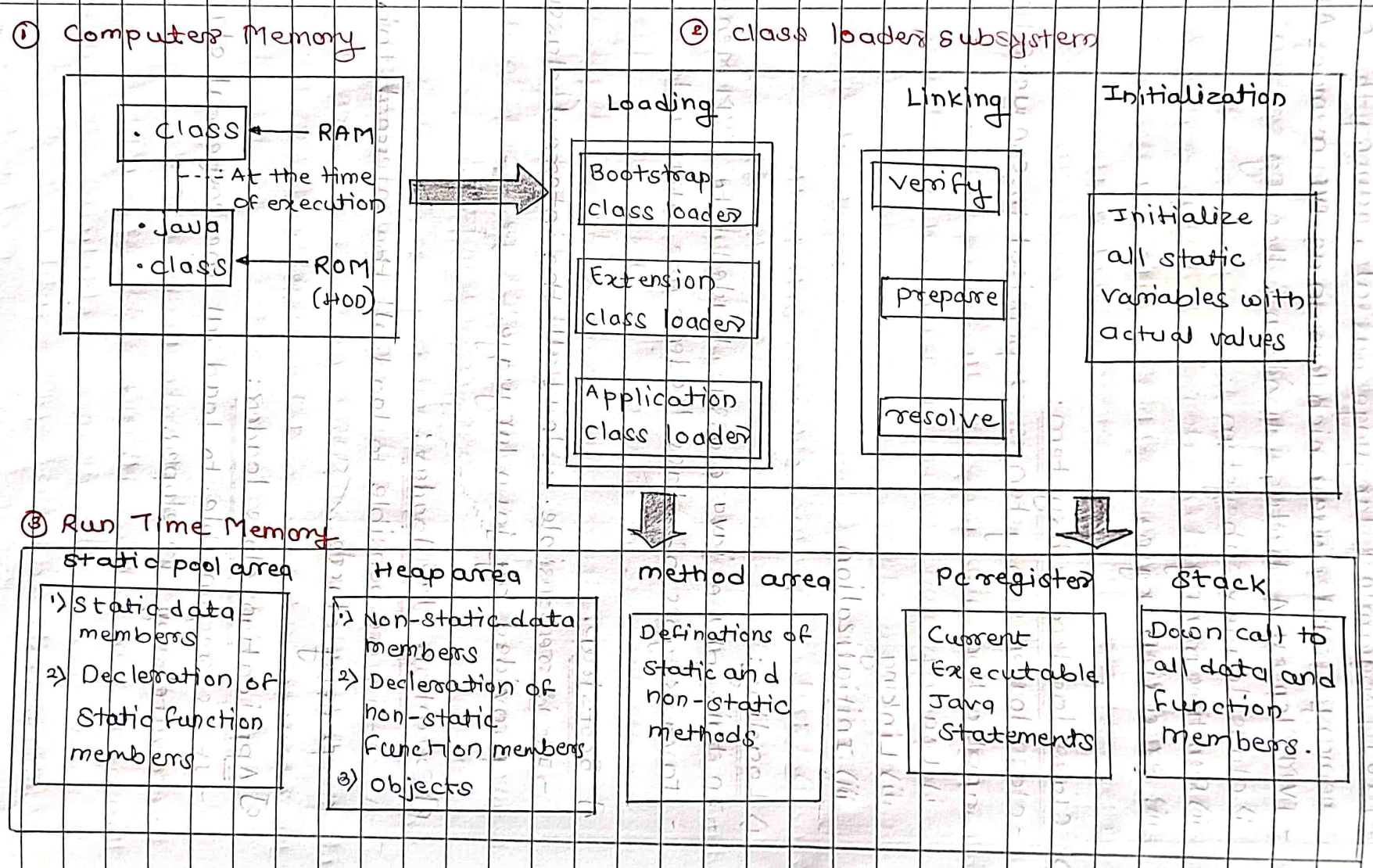
- In this phase java class is loaded into JVM Memory.
- Java 3 types of class loaders are available.

A) Bootstrap:
- It is responsible to load all the internal classes provided by java language.

B) Extention class loader:
- It is responsible to load all the ~~internal~~ third party classes.

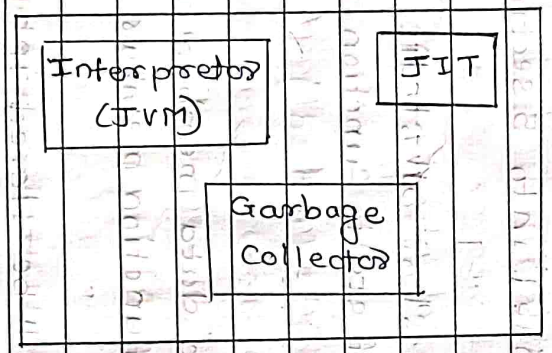
C) Application class loader:
- It is responsible to load all the external classes created by developer.

JVM Memory Architecture



classmate
PAGE
DATE

④ Execution Engine



DATE

2) Linking:

In this phase following operations will be performed

i) Verify:

- Here Byte cond will be varified by JVM.
- If byte code logically incorrect then JVM will throw runtime error.

ii) Prepare:

- Here all static variables will be inishialized with default values.

iii) Resolve:

- In this phase all internal essues related with byte code will be resolved.

3) Initialization:

- Here all static variable will be initialize with default values.
- If actual value is absent then default value will be stored into memory

2] Run Time Memory:

- Run time memory is a temporary storage.
- All data members and function members will be stored into run time memory.
- Runtime memory further divided into 5 sections:

1] Static pool Area:

- This area is responsible to store all static data members and decleration of static function members.

2] Heap Area:

- In this area all non-static data membar and decleration of non-static function membar will be available.
- All user define objects will be stored into Heap Area.

DATE

3] Method Area:
- It acts as container for definetions of static and nonstatic members.

4] Pc register:
- It contains current executable JAVA statements.

5] stack:
- stack is responsible for the execution of all data & function members.
- stack work on first in last out principal.

3] Execution Engine:
- Execution engine is responsible to finish the execution process of java programming.
- Following components are available inside the execution engine.

1] Interpretor:
- It is responsible for interpretation of byte code.
- In JAVA JVM itself acts as interpretor.

2] JIT (Just in time):
- JIT is responsible to convert bytecode into machine code.

3] Garbage Collector:
- It is responsible to realocated memory allocated by JVM.
- Garbag commection can be done implicity as well as explecity.

DATE 04 03 2022

Q. Explain class loading.

- The process of loading members of the class into JVM Memory is known as class loading.
- Class loading can be done implicity as well as explecity.
- If java file contains only main class then implicit class loading will takes place.
- If java file contains more than one classes then class loading must be done explicit
- Class Members will be loaded into JVM memory in the fallowsequence.

1) static data members } Automatically loaded by JVM
2) static function members
3) Non-static data members } object creation is mandatory to load non-static members.
4) Non-static function members.

Program for implicit class loading

```
Class MemoryDemo1
{
    static int k=10;
    double d=25.25;

    Static void test()
    {
        System.out.println("TEST Method");
    }

    void display()
    { System.out.println("Display Method");
    }

    Public static void main(strings[] arg)
    {
        System.out.println("Main Method Started);
        System.out.println("K.value is "+k);
```

```
test();
MemoryDemo1 m1 = new MemoryDemo1();

System.out.println("D value is"+m1.d);
m1.display();

System.out.println("Main Method Ended");
}
}
```

Step1:

MemoryDemo1.java
↓
MemoryDemo1.class (ROM)

Step2:

Started execution
MemoryDemo1.class
(RAM)

Step3:

class loader subsystem

(I) Loading :- Bootstrap class loader
↳ system.class, string.class
- Application class loader
↳ memoryDemo1.class

(II) Linking
→ Byte code verified
→ prepare → static int k=10
→ resolve

(III) Initialization
static int k=10

DATE

Step 4: Data will be stored into run time memory

Run Time Memory — LIFO / FILO

① Static pool area ④ Heap area ② Method Area ③ Stack

```
static int k=10;          new memory d();          {
static void test();       double d=25.25;          //test method
public static void main();  void display();        }
                          (only after object creation)  {
                                                   //main
                                                   }
                                                   {
                                                   //display
                                                   }
```

1) m1.display
2) m1.d
3) test();
4) k
5) main()

```
//business class
class student
{
    //static ----> one time loading
    static string batchcode = "1CCAII";

    //non-static ----> multiple time loading
    void markAttendance (stringname)
    {
        system.out.println ("Batch Code:" + batchcode);
        system.out.println ("Attendance marked
        sucesfully for:" +name);
    }
}

class MemoryDemo2
{
    public static void main (string [] args)
    {
        //explicit class loading.
        student s1 = new student ();
        s1. markAttandance ("DINESH");
        s1. markAttandance ("SURESH");
    }
}
```

Explain differance between static loadi

Static loading	Dynamic loading.
- The process of loading static members into JVM memory is known as static loading.	- The process of loading non-static members into JVM memory is known as dynamic memory.
- Static will be done automatically which means JVM will automatically loads static members into memory	- Non-static members must be loaded only through object creation which means automatic loading is not possible.
- Static loading will takes place only once in program life cycle.	- Dynamic loading can be done multiple times in a program life cycle.
- Static loading is the permenant loading	- Dynamic loading is the temporary loading.

Blocks

DATE 07 03 2022

Q. What is block?

Blocks:

- Block is a collection of executable java statements.
- Blocks can be use to perform special operations in java program.
  e.g. database connection, server connection, file connection
- Blocks are mainly classified into two types
  1) Static block.
  2) Non-static block.

Q. What is the use of static block?

- The block which is declared by using static keyword is known as static block.
- Static block can be use to develop business logic which remain common for all users & will be executed only once in application life cycle.
- It is possible to declare multiple static blocks inside singal java class.
- If java class constains static block and main method then static block will be executed before main method.
- Static block will be execute at the time of class loading where as main method will be executed after the class loading.
- Following is a syntax to use static block:

```
static
{
    ①business logic
    ②Common for all users
    ③Executed only once.
}
```

e.g

```
class MainApp1
{
    public static void main(String [] args)
    {
        System.out.println("Main Method");
    }
    static
    {
        System.out.println("Static Block 1");
    }
    static
    {
        System.out.println("Static Block 2);
    }
}
```

Output:

Static Block 1

Static Block 2

Main Method

Program to execute static block of business logic class.

```
class central
{
    Static j=10;

    Static
    {
        system.out.println("Static block of central");
    }
}
```

DATE

```
class MainApp2
{
    Static
    {
        System.out.println("Satic block of man App");
    }
    public static void main(String [] args)
    {
        System.out.println("Main Method");
        System.out.println("J value is" + central.j);
```

Note:
If there is no value to call form business logic then to main class. then we call all business logic class.
e.g. central c1 = new central();

e.g.②

```
class Master
{
    Static int k = 25; //a

    Static
    {
        k = 30; //b
    }
    Static
    {
        k = 35; //c (Reinitialization)
    }
    Static
    {
        int k = 45; //local variable
        System.out.println("k value is" + k);
}
```

DATE

```
class MainApp3
{
  public static void main (String [] args)
  {
    System.out.println ("K value is" +Master.K);
  }
}
```

3. What is the use of non-static block.

- The block which is declare without using static keyword.
- Non-static block can be use to develop business logic which remains common for all objects and will be executed multiple times.
- Non-static block will be execute at the time of object creation.
- If java class contains static and non-static block then highest priority given for execution of static block.

e.g.

```
class MainApp4
{
  Static
  {
    System.out.println ("STATIC BLOCK");
  }
  ——— Non-static block
  {
    System.out.println ("Non-Static Block");
  }
  public static void main (String [] args)
  {
    System.out.println ("Main Method");
    MainApp4 m1 = new MainApp4 ();  → object call
    MainApp4 m2 = new MainApp4 ();    needed in non-
  }                                   static block.
}
```

DATE

e.g. ②

```
class FacebookAccount
{
  {
    System.out.println("Account created Sucessfully");
    Createprofile ();
  }
  void createprofile ()
  {
    System.out.println(" Profile created Sucessfully");
  }
}
class MainApp5
{
  public static void main(String [] args)
  {
    FacebookAccount F1 = new FacebookAccount();
    FacebookAccount F2 = new FacebookAccount();
  }
}
```

Q. Explain loading and execution flow of java class if it contain static and non-static block.

- At time of loading highest priority given to static members.
- When static members are getting loaded into memory parallerly static blocks will be executed by JVM.
- At a time of object creation non-static members will be loaded into memory parallerly non-static block will be executed by JVM.

Execution flow:

Static loading:
1) static data members
2) Static function members

① Static block executed by JVM (only once)

Dynamic loading:
3) non-static data members → At the time of object creation
4) non-static function members → non-static block will be executed by JVM (multiple times

②

Q. Differance between static and non-static block?

Static Block	Non-Static block
1) Block which is declearred by using static keyword is known as static block.	1) Block which is declearred without using static keyword is known as non-static block.
2) Static block will be executed by JVM at time of class loading.	2) This will be executed by JVM at time of object creation.
3) Static block will be executed only once in application lifecycle.	3) This block will be executed multiple times and depends on object creation.
4) This can be used to initialize static variables.	4) This block can be used to initialize static as well as non-static variables.

DATE

Explain differance between Methods and blocks.

Methods	Blocks
1) Methods are used to perform regular operations in JAVA program.	1) Blocks can be use to perform Special operations in JAVA program.
2) Developer is responsible to make a call methods.	2) Blocks will be automatically executed by JVM.
3) We can provide input parameters to methods.	3) We can'nt provide input parameters to blocks.
4) It is possible to return a value from methods.	4) It is not possible to return value from blocks.
5) Methods are having identity (name).	5) Blocks doesn't have identity (name).

Constructor

DATE

Q. What is constructor?

Constructor:

- Constructor is special member of the java class which can be use to initialize non-static data class.
- Constructor name should be as class name.
- We cann't use return type for a constructor because it is special member.
- Every JAVA class must have a constructor either defined while compiler or else program.
- If compailer defines the constructor then it is non as default constructor.
- compiler will defined the constructor only if user defined constructor is not available.
- Compailar can defined only '0' paramitenized constructor.
- If programmer defines the constructor then it is known as used defined constructor.
- Programmer can define '0' (zero) paramiterized as well as paramitenized method.
- Constructor will be executed at the time of object creation.

Q. What are the responsibilites of new operator

- New operator is resposible of to allocate memory for non-static membar
- After memory allocation new operator will load all non-static member into memory.
- After loading non-static members finally it called the constructor of respective class.

DATE

* Default constructor:

- What is use of javap
  - JAVAP act as diassemble
  - javap command can be use to show detail information of java class after sucessful compilation.

eg.
```
class Master
{
    Master(),
    {
        Default class / Default constructor.
    }
}
class MainApp6
{
    public static void main(String [] args)
    {
        Master m1 = new master();
        System.out.println (m1);
    }
}
```

→ javap Master } To show constructor
javap MainApp6

* User defined Constructor:

eg.①
```
class Product
{
    int productId;
    String nameProduct;

    product (int ID, String name)
    {
        productId = id;
        productname = name;
    }
```

DATE

```
void display ()
 {
   system.out.println (productId);
   System.out.println (productName);
 }
}
class MainApp7
 {
   public static void main (String [] args)
    {
      product p1 = new product (101, " TV");
      p1.display ();
    }
 }
```

e.g ②

```
class Student
{
   int rollNo;
   double percentage;

   // Userdefined Constructor
   Student (int no, double marks)
   {
      rollNo = no;
      percentage = marks;
   }

   void display ()
    {
      system.out.println (rollNo + "\t\t" + percentage);
    }
}
```

```
class MainApp
{
    public static void main(String[] args)
    {
        Student s1 = new Student (25, 78.25);
        s1.display();
        Student s2 = new Student (50, 83.70);
        s2.display();
    }
}
```

Program to demostate execution flow of java class if it contain non-static block as well as constructor.

```
class Demo
{
    Demo()
    {
        System.out.println ("zero Param Constructor");
    }
    //non-static block
    {
        System.out.println ("Non-static block");
    }
    Static ()
    {
        System.out.println ("Static block");
    }
}
class MainApp8
{
    public static void main(String[] args)
    {
        Demo d1 = new Demo()
    }
}
```

Use if we need to provide parameters

DATE

o/p (execution class)

Static block

Non-static block

Zero Param Constructor

Q. What is constructor overloading

Constructor Overloading:

- The process of creating multiple constructors of same class with different parameter is known as constructor overloading.
- At a time of constructor overloading parameters must be different either while datatype or length.
- Constructor overloading can be use to create different object of same class.

```
class Employee
{
    int empId;
    String empName;
    double empSalary;
    int yearofexp;

    Employee (int id, String name, double salary, int exp)
    {
        empID = id;
        empName = name;
        empSalary = salary;
        yearofexp = exp;
    }
    Employee (int id, string name, double salary)
    {
        empID = id;
        empName = name;
        empSalary = Salary;
    }
```

```
void display()
{
    System.out.println(empID+"\t"+empName+"\t"
                        +empSalary+"\t"+year of exp);
}
}

Class MainApp9
{
    public static void main(String[] args)
    {
        Employee E1 = new Employee(101," Mangesh",5000,3);
        E1.display();

        Employee E2= new Employee(101," Sagar",10000);
        E2.display();
    }
}
```

year of exp
display '0'
default value.

Q. What is the use of this keyword?

- this is a special keyword which can be use to differentiate local variable and instance variable.
- This keyword always points to current class instance (Objec
- It means address of an object and this keyword will be same.
- We can use this keyword anywhere inside constructor or non-static method.

DATE

```
class Master
{
    int j;
    Master (int j)
    {
        this.j = j;
    }
    void display ()
    {
        System.out.println ("J value is" +j);
    }
}
class MainApp10
{
    public static void main (String [] args)
    {
        Master M1 = new Master (25);
        M1.display ();
    }
}
```

local variable

instance variable

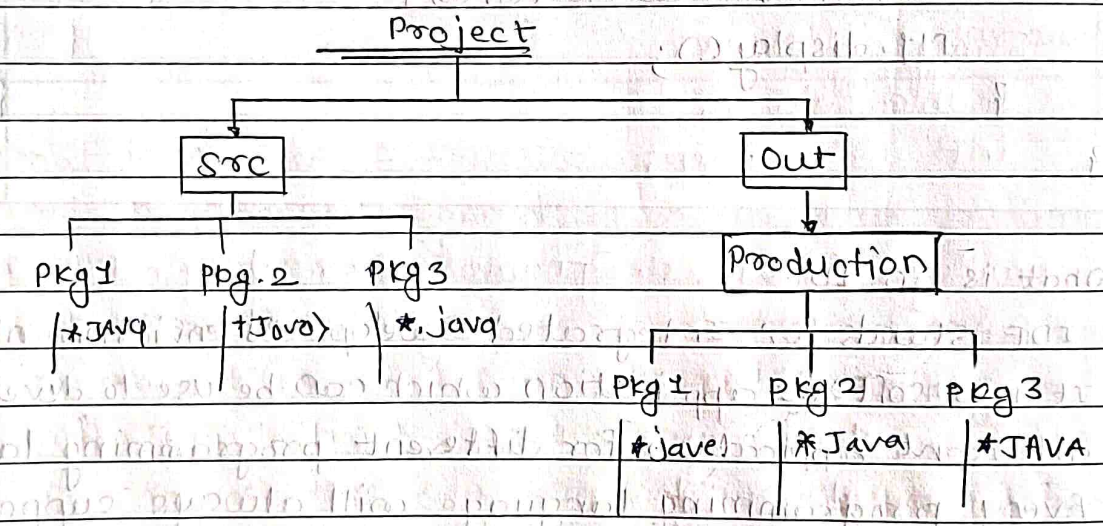
Q. What is an IDE.

- IDE stands for Integrated Development enviornment.
- It is a software application which can be use to develop different application for different programming language.
- Every programming language will always support software
- It acts as a container for different container or different components.
  1) editor
  2) debugger
  3) VCS (git, JVN)
  4) Automation tools.
  5) Deployment tools
  6) Devops.

- For JAVA we can use full IDE software.
1) eclipse
2) netbeams
3) IntelliJIDEA
4) Jcreator
5) VS code.

- By using IDE we can develop by different application, web, Android, hybrid etc.
- The using IDEis we can do rapid development.

Q Explain the product structure ~~in java~~ in IntelliJIDEA?
- ~~Every IDE is having its object structure~~
- Every IDE is having its own project structor to develop java applications.
- Following is the project structure from intelligent JIDEA software.



Q. How to create new project in intelliJ IDEA?
- We have to follow several steps to create a new project.
1) Go to file option & select new project from list
2) After selecting new project search java option and configure JDK

Control Statements
[looping]

DATE 10 03 2022

3) Inter project name & location and
4) Create new package inside the source folder to store java classes.

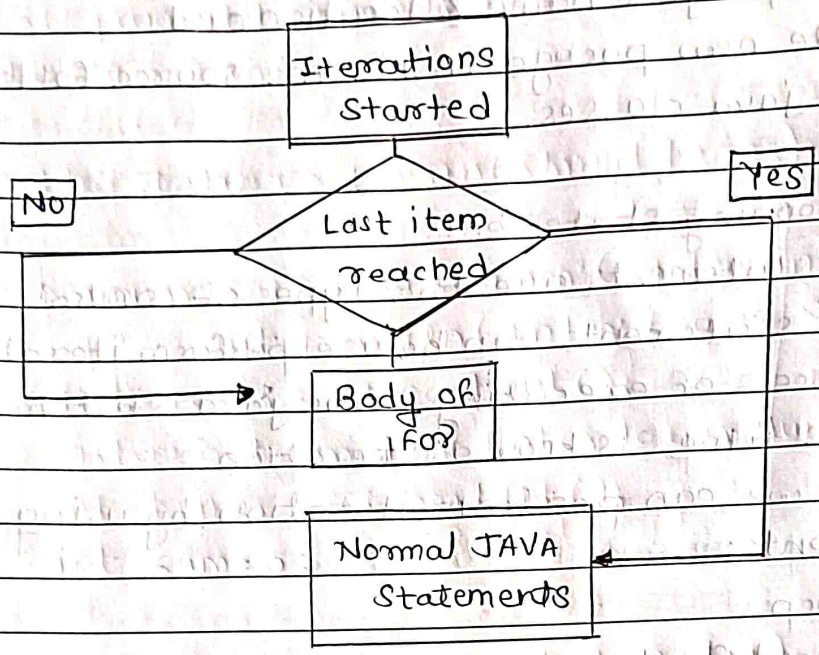
Explain looping statements.

- Looping statements are the types of control statement.
- Looping statements can be use perform iteration.
- The process of executing same java statements multiple times is know as iteration.
- In java we can perform iteration by using following statements:
  1) For loop
  2) Enhanced for loop
  3) While loop
  4) Do-while loop
  5) break and continue.

1) For loop:

- For loop is one of the looping statement.
- It can be use to perform specific number of iteration (finite iterations)
- In case of for loop we have to specify count of iteration.
- following is sytax for loop:

```
For loop
    └──→ finite iterations

for (initialization ; condition ; ++|--)
     (start)          (end)
{
    ____________
    ____________ JAVA statements
}
```



eg.
```
public class Demo1 {
    public static void main (Strings [] array) {
        for (int a = 1, a < 5 = ; a++) {
            system.out.println (a);
        }
    }
}
```

Q.

cmd

Enter lower limit

Enter upper limit

No. is divisible by 4.

DATE

```
import java.util.Scanner;
public class Demo2
{
    public static void main (String [] args
    {
        Scanner sc1 = new Scanner (System.in);
        System.out.println (" Enter lower limit");
        double No1= sc1.nextDouble ();
        System.out.println (" Enter upper limit");
        double No2 = sc1.nextDouble ();

        for (double a = lower; a <= upper; a++)
        {
            if (a%4 == 0)
            {
                System.out.println (a);
            }
        }
    }
}
```

No1 No2

Q.

Enter lower limit

Enter Upper limit

Sum from Lower to Upper

e.g. 10+11+12+13+14+15

```
import java.util.scanner;
public class Demo3
{
    public static void main(string[] args)
    {
        Scanner sc1 = new Scanner(System.in);
        System.out.println("Enter Lower Number:");
        int Lower = sc1.nextInt();
        System.out.println("Enter lower Number:");
        int higher = sc1.nextInt();
        int sum = 0;

        for(int a = lower ; a<=higher ; a++)
        {
            sum = sum+a;
        }
        System.out.println("SUM:"+sum);
    }
}
```

Q. Substraction /odd numbers

```
import java.util.scanner;
public class Demo 4
{
    public static void main(string[] args)
    {
        scanner sc1 = new scanner(System[] args);
        System.out.println("Enter Upper Number:");
        int Upper = sc1.nextInt();
        System.out.println("Enter lower Number:");
        int lower = sc1.nextInt();

        ~~int sum = 0~~
```

DATE

```
for (int a= Upper; a>=lower; a--)
  {
    if (a%2 !=0)
      {
        System.out.println (a);
      }
  }
}
}
```

Q. cmd

Enter total no. of users

(5 - users)

Enter Name

_____

Welcome _____ } as per no of users.

Welcome _____

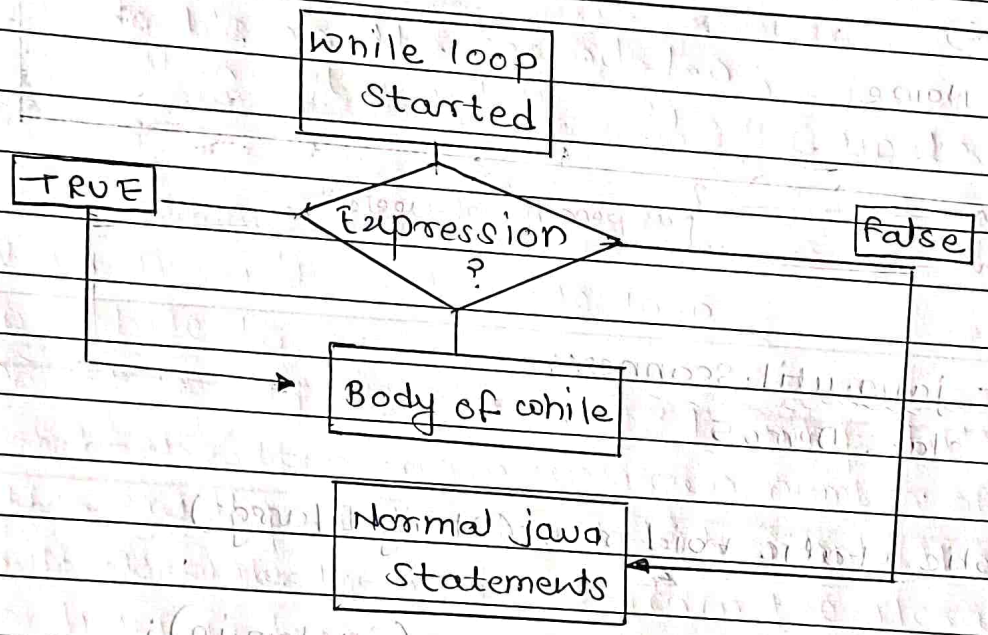
```
import java.util.scanner;
public class Demo5
{
  public static void main (String [] args)
  {
    Scanner sc1 = new scanner (System.in);
    System.out.println ("Enter total no. of users:");
    double count = sc1.nextDouble();

    for (double a=1; a<=count; a++)
    {
      System.out.println ("Enter Name");
      String NAME = sc1.next();
      System.out.println ("Welcome"+name);
    }
  }
}
```

Explain the working of while loop.

2) While loop:

- While loop is one of the looping statements it can be use to perform unknown no. of iteration.
- It case of while loop we don't have to specify end point
- Following is syntax for while loop:

```
while (expression /conditions)
  {
    ____________
    ____________ java statements
  }
```



e.g.

```
public class WhileDemo2 {
  public static void main (String [] args);
  {
    Scan sc1 = new scanner( System.in);
    int no=0;
    while (no>=0)
    {
      System.out.println
      no= sc1.nextInt ();
      System.out.println (no);
    }
  }
}
```

DATE

Q. CMD

select mode of operation

1: convert min to sec.

2: convert sec to min.

3: Exit

Accept choice

RRfection operation

1:

2:

3:

```
import .java.util.Scanner
 {
   public static void main (string [] args)
   {
     Scanner sc1 = new scanner (System.in);
     boolen status = true;
     while (status)
      {
        System.out.println("select mode of operation);
        System.out.println("1: Min to sec")
        System.out.println("2: sec to min);
        System.out.println("3: Exit")

        int (choice == 1)
         {
           System.out.println("Enter Minutes");
           double min = sc1.nextdouble();
           sec to min (sec)
         }
       int (choice == 2)
         {
           System.out.println("Enter seconds");
           double sec = sc1.nextdouble();
           sec to min (sec);
         }
```

```
else
 {
   status = False;
 }
}
}
static void mintosec (double minutes)
 {
   double result = mimutes * 60;
   System.out.println ("SECONDS":"+result);
 }
Static void sectomin (double seconds)
 {
   double result = seconds / 60;
 } Sytem.out.println ("Minutes:"+result);
}
```

Q. Explain the use of do while loop

3] **Do-while loop:**

- Do-while loop is one of the looping statements which can be use to perform unknown number of iterations.
- It is exactly similar to while loop only the difference is that while loop will perform the iteration only if condition is true. whereas do-while loop will check the condition after performing iter
- While loop is known as entry control loop and do-while loop is known as do-while loop.
- Following is a syntax:

```
do
{
    Java statements
}
while (exp);
```

e.g.

```
public class WhileDemo3
{
  public static void main (String[] args)
  {
    int a = 2;
    // entry control
    While (a>2)
    {
      a--;
    }
    System.out.println ("A: " +a); //2
    // exit control
    int b = 2;
    do
    {
      b--;
    } while (b>2);
    System.out.println ("B:" +b); //1
  }
}
```

Q. Explain differance between break and continue.

Break and Continue

- Break statement can be used to exprecitly stop iterations whereas continue statement can be used to specify specific iteration.
- We can use break and conitnue statement anywhere inside the for loop, while loop, do-while loop, if else or which statements.

```
public class BreakDemo1
{
  public static void main(String [] args)
  { //break statement
    for (int a=1; a<=10; a++)
    {
      if(a==5)
      {
        break;
      }
    System.out.println(a); //1 2 3 4
    }
  //continue statement
  for (int a=1; a<=10; a++)
    {
      if(a==5)
      {
        continue;
      }
    System.out.println(a); //1,2,3,4, 6,7,8,9
    }
  }
}
```

Switch case statement:

- Switch statement is one of the disision making statement
- It can be use to apply condition or restriction on Java statements.
- Switch statement will execute specific state of java statements based on given choice
- Following is syntax to use switch case statement

DATE

```
Switch (exp)
{
   case 1:
            break;
   case 2:
            break;
   default:
}
```

e.g.

```
import java.util.scanner;
public clas DemoSwitch
{
   public static void main(String [] arg)
   {
      scanner sc1 = new Scanner (System.in)
      system.out
      system .out
      System. out. println ("DELHI");
      String s = sc1.next()

Swith (choice)
  {
     case "PUNE":
        system. out. println ("Selected Pune city");
        break;
     case "Mumbai;
```

DATE

```
case "DELHI":
System.out.println("Invalid City");
Break;

default:
System.out.println("Invalid city");
break;
}
}
}
```

Pattern logical

pattern no 1.

```
      c1  c2  c3
l1    *
l2    *   *
l3    *   *   *
```

```
public class pattern1
{
  public static void main(String [arg)
  {
    int lines=3
    int star=1
    //rows
    for(int r=1;r<= lines ; r++)
    {
      //column
      for(int c=1 ; c <=star ; c++)
      {
        System.out.println("*");
      }

    System.out.println();
    Star++;
  }
}
}
```

classmate

PAGE

DATE

pattern no.2

```
public class pattern2
{
  public static void main(String [ ]args)
  {
      int lines = 4;
      int star = 4;

    for (int r=1; r<= lines; r++)
     {
        for (int c=1; c<=star; c++)
         {
           System.out.print("*");
         }
    System.out.println();
    star--;
     }
  }
}
```

output → * * * *
* * *
* *
*

pattern no.3

Q.3. pattern

A
B C
D E F

DATE

```
public class pattern3
{
  public static void main (String [] args)
  {
      int lines = 3;
      int char = 1;
      char ch = 'A';

    for (int r = 1; r <= lines; r++)
    {
      for (int c = 1; c <= char; c++)
      {
        system.out.println (ch);
        ch++;
      }
    System.out.println ();
    char ++
  }
}
}
```

Pattern no. 4

Q.

```
1 2 3 4
1 2 3
1 2
1
```

DATE

```
public class pattern4
{
  public static void main (String [] args)
  {
     int lines = 4;
     int star = 4;
     ~~int no = 1;~~

     for (int r = 1; r<= lines; r++)
     {
        int no = 1;
        for (int c=1; c<= star; c++)
        {
          System.out.print (no);
          no++;
        }
       system.out.println ();
       star--;
     }
  }
}
```

Pattern no.5

Q.

```
A
B   B
C   C   C
D   D   D   D
```

DATE

```
public class pattern5
{
  public static void main (string [] args)
  {
    int lines = 4;
    int star = 1;
    char ch= 'A';

    for (int r=1 ; r<= lines ; r++)
    {
      ~~char ch = 'A';~~
      ~~System.out.println (ch)~~
      for (int c=1 ; c <= lines ; ~~c++~~)
      {
        system.out.print~~ln~~(ch);
      }
    system.out.println();
    star++;
    ch++;
  }
 }
}
```

Pattern no. 6

Q.

```
4 4 4 4
3 3 3
2 2
1
```

DATE

```
public class pattern6
{
    public static void main (string [] args)
    {
        int lines = 4;
        int star = 1;
        int no = 1;

        for (int r = 1 ; r <= lines ; r++)
        {
            for (int c = 1 ; c <= lines ; c++)
            {
                System.out.print (no);
            }
            system.out.println ();
            star--;
            no--;
        }
    }
}
```

Pattern no.7

```
___ ___ *
___ *   *
*   *   *
```

```
public class pattern7
{
    public static void main (String [] args)
    {
        int lines = 3;
        int star = 1;
        int spaces = 2;

        for (int r=1; r<= lines, r++)
        {
            for (int s=1; s<=spaces; s++)
            {
                system.out.print("__");
            }
            for (int c=1; c<= star; c++)
            {
                System.out.print("*");
            }
            System.out.println();
            Star++;
            Spaces--;
        }
    }
}
```

Q.

```
__ __ *
__ * *
* * *
__ * *
__ __ *
```

DATE

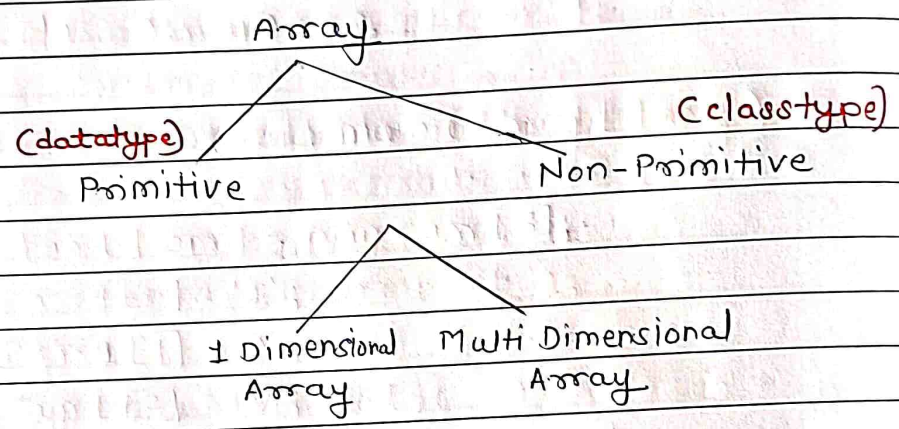
Array

DATE 15 03 2022

Q. What is Array?

Array:

- It is one of the core data structures.
- Array can be used to store multiple element as a single unit.
- By using array we can make a group of same type elements. (homogeneous data).
- In java arrays are mainly classified:
  1) Primitive Array
  2) Non-Primitive Array



Q. Explain how to create One-Dimensional Array.

One-Dimensional Array:

- One-Dimensional Array can be used to store data in terms of rows and indexe's.
- Following are the steps to create array:

1) Decleration:

We have to decleare an array by using appropriate datatype.

e.g. `int [] arr1;`

2) Size allocation:

After decleration its mandatory to allocate size for array variable.

e.g. `arr1 = new int[5];`

DATE

3] Initialization:

After size allocation we can initialize array variable by using appropriate index number.

e.g.

arr[0] = 10

arr[1] = 20

structure:

10	20	X	X	X
0	1	2	3	4

- Program 1

```
public class ArrayDemo1
{
  public static void main(String [] args)
  {
    // decleration
    int [] arr1;
    // size allocation
    arr1 = new int [5];
    // initialization
    arr1[0] = 100;
    arr1[1] = 200;
    arr1[2] = 300;
    arr1[3] = 400;
    arr1[4] = 500;
    // acess data
    // regular for loop
    for (int a=0; a<=4; a++)
    {
      System.out.println(arr1[a]);
    }
    // enhanced for loop
    for (int s: arr1)  ---> only increment
    {
      System.out.println(s);
    }
  }
}
```

DATE

Q. What is the difference between regular for loop and enhanced for loop.

- In case of regular for loop we have to provide specific start as well as end point where as enhanced for loop accepts only two parameters that is iterator name and collection name.
- Regular for loop can travels the array forward as well as backward direction where as enhanced for loop is capable to travel array only in forward direction

• Program No.2

```
public class ArrayDemo2
{
    public static void main(String [] args)
    {
        char [] arr1 = new char [3];
        arr1 [0] = 'J';
        arr1 [1] = 'A';
        arr1 [2] = 'V';
        arr1 [3] = 'A';

        for (int a=0; a<4; a++)
        {
            System.out.println (arr1 [a]);
        }
    }
}
```

O/p:
Exception in thread main
ArrayIndexOutOfBoundsException

Q. What is Array index out of Bounds Exception?

- This is a type of run time exception thrown by JVM
- If developer is trying to store element at invalid position inside the array then JVM will throw this exception.

• Program No.3

Q write a program to create dynamic array by accepting total no. of courses and names of courses from end users.

Enter total no. of courses

_____

Enter course names

_____

_____

_____

DATE 16 03 2022

- Program no. 4 → Dynamic Array

```
import java.util.scanner;
public class ArrayDemo4
{
    public static void main (String [] args)
    {
        Scanner sc1 = new scanner (System.in);
        System.out.println ("Enter Total no. of courses");
        int count = sc1.nextInt ();
        // Decleration
        String [] courses;
        // Size allocation
        Courses = new string [count];
        // Initialization
        System.out.println ("Enter Names of courses:");
        for (int a=0; a<count; a++)
        {
            courses [a] = sc1.next ();
        }
        System.out.println ("Selected courses");
        for (int a=0; a<count; a++)
        {
            System.out.println (courses [a]);
        }
    }
}
```

Q. Write a program to accept total no. of subjects and marks of respective no. of subject from end user. calculate total marks and average marks.

cmd

```
Enter total no. of subjects
_____
Enter marks for each sub
_____
_____
_____
Total marks:
Average:
```

DATE

~~Program No. 5 →~~

```
import java.util.Scanner;
public class ArrayDemo5
{
    public static void main (String [] args)
    {
        Scanner sc1 = new Scanner (System.in);
        System.out.println ("Enter total no. of subjects");
        int count = sc1.nextInt ();
        double [] marks = new double [count];
        System.out.println ("Enter Marks");
        // Accept Marks of multiple subjects
        for (int a=0; a<count; a++)
        {
            marks [a] = sc1.nextDouble ();
        }
        // calculate total marks
        double sum = 0.0
        for (int a=0; a<count; a++)
        {
            sum = sum + marks [a];
        }
        System.out.println ("Total Marks:" +sum);
        System.out.println ("Avg Marks:" +(sum/count));
    }
}
```

cmd

Enter total no of elements

Enter values

O/p

$10^3 = 1000$

$20^3 = ---$

DATE

Program no. 6

```
import java.util.scanner;
public class ArrayDemo6
{
    public static void main(String [] args)
    {
        scanner sc1 = new scanner (System.in);
        System.out.println (" Enter total no. of elements");
        int count = sc1.nextInt();
        double [] element = new double [count];
        System.out.println (" Enter values:");
        for (int a=0; a<count; a++)
        {
            element [a] = sc1.nextDouble();
        }
        for (int a=0; a<count; a++)
        {
            double cube = element [a] * element [a] * element[a];
            System.out.println ("cube:" + cube);
        }
    }
}
```

Q. Write a program to accept no. from user and check wheather the given no exist or not inside the array and display appropriate msg.

Decleration

```
int [] arr1;
```

Size allocation;

```
arr1 = new int(4)
arr[0] = ____
```

DATE

Program no. 7

```
import java.util.scanner;
public class ArrayDemo7
{
    public static void main (String [] args)
    {
        scanner sc1 = new scanner (system.in);
        System.out.println("Enter No");
        int no = sc1.nextINT ();

        double [] arr1 = new double [4];
          arr1 [0] = 10
          arr1 [1] = 40
          arr1 [2] = 70
          arr1 [3] = 80

        boolean found = false;
        for (int a=0; a<4 ; a++)
        {
          if (arr1 [a] = = no)
          {
            System.out.println(" Number Exist");
            found = true;
          }
        }
        if (found == false)
        {
          system.out.println ("No does not exist");
        }
    }
}
```

DATE

Q. How to create Array literal

- Array literal is the approach to create array variable
- In case of array literal we don't have to provide array size in advance which means we can create array without providing initial capacity.

Syntax:

```
int [] arr1 = { no1, no2, no3, no4 ..... non };
```

Program no. 8

```
public class ArrayDemo8
{
  public static void main (String [] args)
  {
    double [] data = { 10.25, 70.25, 30.25, 60.75, 80.75 };

    for (int a=0; a< data.length; a++)
    {
      System.out.println (data [a]);
    }
  }
}
```

Program no. 9

```
public class ArrayDemo9
{
  public static void main (String [] args)
  {
    int [] data1 = { 20, 60, 70 };

    int [] data2 = { 40, 30, 90 };

    int [] result = new int [3];
```

```
for (int a=0; a< data1.length; a++)
 {
   result [a] = data1 [a] + data2 [a];
   System.out.println(result [a]);
 }
}
}
```

Q.

```
① String [] productNames= { "TV"; "Mobile", 'Laptop', 'AC'};
② double [] productprices= { 12000, 15000, 45000, 50000};
③ int [] stock = { 10, 50, 20, 30}
```

cmd

select mode of operation
1) Purchase product
2) Search product
3) Exit

1:

1) TV
2) Laptop
3) Mobile
4) AC

:

Enter qty = 10

DATE

Program No.10

— Business logic class —

```
public class StoreManager
{
  Static string [] products = {"TV", "MOBILE", "Laptop", "AC"};
  static double[] prices = {10000, 15000, 45000, 50000};
  static int [] stock = {10, 50, 20, 30};

  void purchaseProduct (int choice, int qty)
  {
    boolen found = false;
    for (int a=0; a< products.length; a++)
    {
      if (choice == a && qty < stock [a])
      {
        double total = qty * prices[a];
        System.out.println("TOTAL AMOUNT:"+total);
        stock [a] = qty;
        found = true;
      }
    }
    if (found == false)
    {
      System.out.println("Product not found or out of
                                stock")
    }
  }

  void searchProduct (int choice){
    boolean found = false;
    for (int a=0; a < products.length; a++){
      if (a == choice){
        System.out.println ("PRICE:"+ prices [a]);
        System.out.println ("STOCK:" + stock [a]);
        found= true;
      }
    }
```

DATE

```
if (found==false){
    System.out.println("Product Not Found");
}
}
```

Executable class

```
import java.util.Scanner;
public class ArrayDemo10{
    public static void main(String[] args){
    Scanner sc1 = new Scanner(System.in);
    StoreManager s1 = new StoreManager();
    boolean status = true;
    while (status){
        System.out.println("Select Mode");
        System.out.println("1: Purchase Product");
        System.out.println("2: Search Product");
        System.out.println("3: Exit");
        int mode = sc1.nextInt();
        if (mode==1){
            System.out.println("0: TV");
            System.out.println("1: MOBILE");
            System.out.println("2: LAPTOP");
            System.out.println("3: AC");
            int choice = sc1.nextInt();
            System.out.println("ENTER Qty");
            int qty = sc1.nextInt();
            s1.purchaseProduct(choice,qty);
        } else if (mode==2){
            System.out.println("0: TV");
            System.out.println("1: MOBILE");
            System.out.println("2: LAPTOP");
            System.out.println("3: AC");
            int choice = sc1.nextInt();
            s1.searchProduct(choice);
        } else {
            status = false;
        }
    }
}
```

Array 2D

DATE 21 03 2022

Q. Explain 2D Array?

- 2D Array is type of multidimensional array.
- 2D array can be used to store data in terms of rows and colums.
- We have to follow following steps:

① Decleration

int [] [] arr1

② Size allocation

arr1 = new int [row] [cols]

③ Initialization

arr1 [row] [col] = value1

Program 11

```
public class ArrayDemo11
{
    public static void main(String [] args)
    {
        int [] [] arr1;
        arr1 = new int [2] [2];
        arr1 [0] [0] = 100;
        arr1 [0] [1] = 200;
        arr1 [1] [0] = 300;
        arr1 [1] [1] = 400;
    // rows
        for (int a=0; a<arr1.length; a++)
        {
            //colums
            for (int b=0; b<arr1.length; b++)
            {
                System.out.print (arr1 [a] [b] + "\t");
            }
            System.out.println ();
        }
    }
}
```

DATE

Q. Create array for 5 rows, 2 colums of student and Stream.

Student	STREAM

program 12

```
public class ArrayDemo12
{
  public static void main(string [] args)
  {
      String [][] student = new string [5][2];
      student [0] [0] = "Prathmesh";
      student [0] [1] = "Mechanical";
      Student [1] [0] = "Tushar"
      Student [1] [1] = " Mechanical"
      Student [2] [0] = "Mahaling"
      Student [2] [1] = "civil"
      Student [3] [0] = "Harshal"
      Student [3] [1] = "Computer"
      Student [4] [0] = " Aakash"
      Student [4] [1] = "Electrical"
  //rows System.out.println ("NAME \t STREAM");
  for (~~string~~ int a=0; a< student.length; a++)
  {
   //columns
    for (~~string~~ int b=0; b< ~~student.length~~ student[a].length; b++)
    {
      System.out.println (student[a] [b] + "\t");
    }
    system. out. println ();
  }
}
}
```

DATE

Q. cmd

```
Enter total no. of floors:
_____
Enter total no. of offices:
_____
Enter office no:
_____
_____
_____
_____
```

O/P

```
floor No.1
office no. 1, — , 5
```

```
import.java.util.scanner;
class ArrayDemo13
{
  public static void main (String [] args)
  {
    Scanner sc1 = new scanner (System.in);
    System.out.println ("Enter total no. of floors");
    int floors = sc1.nextInt ();
    System.out.println (" Enter total no. of offices
                          on each floor");
    int offices = sc1.nextInt ();
  //create 2D array
    int [] [] building = new int [floors] [offices];
    System.out.println ("Enter" + (floors * offices) +
                          " office no");
  //floors
    for (int r=0; r< floors; r++)
    {
      System.out.println ("Enter office nos for" + (r+1) "f
      //offices
      for (int c=0; c< offices; c++)
      {
        building [r] [c] = sc1.nextInt (); //office no.
      }
    }
```

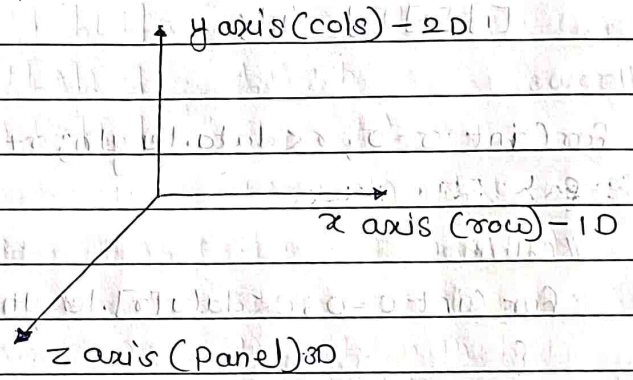
classmate

PAGE

```
System.out.println("==========");
// floors
for (int r=0; r< floors; r++)
  {
    System.out.println ("Floor No: "+ (r+1));
    // offices.
    for (int c=0; c< offices; c++)
      {
        System.out.println("office no: "+building[r][c]+"\t");
      }
    System.out.println ();
  }
}
}
```

Explain 3D array

- 3D array is a type of multi-Dimensional array.
- 3D array can be use to store data in terms of rows, colums and panels.



- Following are the steps:

① Decleration:

```
int [] [] arr1;
```

② Size allocation:

```
arr1 = new int [row] [colums] [panels];
```

③ Initialization:

```
arr1 [r][c] [P] = value;
```

DATE

```
public class ArrayDemo14
{
  public static void main (String [] args)
  {
    // Decleration
      char [] [] [] data;
    // size allocation
      data = new char [2] [2] [2];
    // initialization
    // first row
     data [0] [0] [0] = 'A';
     data [0] [0] [1] = 'B';
          [0] [1] [0] = 'C';
          [0] [1] [1] = 'D';
    // second row
     data [1] [0] [0] = 'E';
          [1] [0] [1] = 'F';
          [1] [1] [0] = 'G';
          [1] [1] [1] = 'H';
    //rows
     for (int r=0; r<data.length; r++)
      {
       // Colums
        for (int c=0; c<data [r].length; c++)
         {
          //panels
          for (int p=0; p<data [r] [c].length; p++)
           {
            System.out.println (data [r] [c] [p] + " /t");
           }
         }
       system.out.println ()
      }
  }
```

classmate

PAGE

Strings

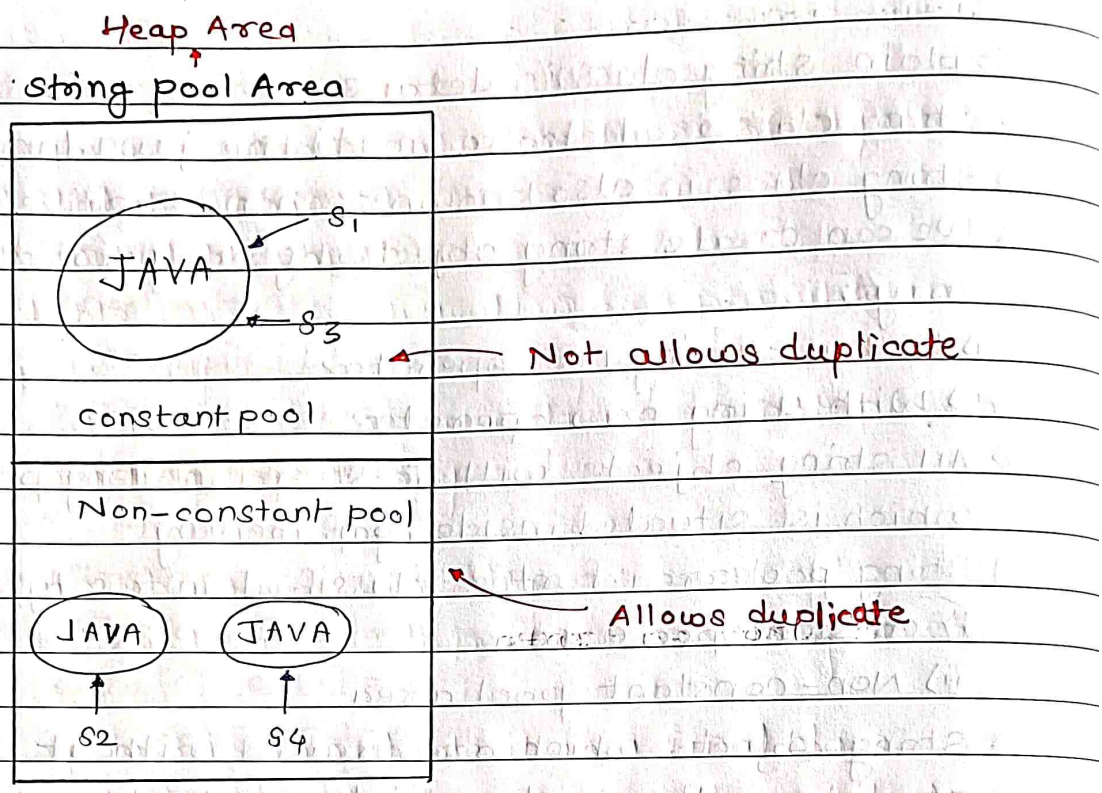
DATE 22 03 2022

Q. Explain string class in details

1) String class is decleare inside the java.lang packagee.
2) String class is also known as non-primitive datatype.
3) We can create string objects by using two different ways:
   i) Without using new operator
   ii) With using new operator
4) All string objects will be stored in string pool area which is situated inside heap memory.
5) String pool area further classified in two types:
   i) constant pool area.
   ii) Non-constant pool area.
6) String objects which are created without using new operator will get stored into constant pool area.
7) String object which are created with using new operator will get stored into non-constant pool area.
8) Constant pool area does not allow to store duplicate objects were as non-constant pool area allows to store duplicate objects.

• Program 1

```
public class StringDemo1
{
    public static void main(String[] args)
    {
        String S1 = "JAVA";
        String S2 = new String(original: "JAVA");
        String S3 = "JAVA";
        String S4 = new String(original: "JAVA");
    }
}
```



Q. How to compare string objects?

- We can compare string objects by using three different way:
  i) == (equal to)
  ii) equals();
  iii) equalsIgnorCase method;

1) == operator:
- This is a basic comparison operator.
- This operator always compares adderes of an objects.
- If two object are having same address then o/p will be true else it will be false.

2) equals():
- This method is declare inside the object class.
- This method is responsible to compare the contain of two string object.
- This method is case-sensitive.

DATE

3) equalsIgnoreCase():

- This method is decleared inside the string class.
- This method is also use for the comparision of content available inside the address.
- This method is case-insencitive.

• Program 2

```
public class StringDemo2
{
  public static void main(String [] args)
  {
    //constant pool area
    String S1 = "PUNE";
    String S2 = "Mumbai";
    String S3 = "BANGLORE";
    String S4 = "PUNE";

    //Non-constant pool area
    String str1 = new String(original: "PUNE");
    String str2 = new String(original: "MOMBAI");
    String str3 = new String(original: "Bangalore");

    System.out.println(S1 == S3);
    System.out.println(S1 == S4);
    System.out.println(S1.equals(str1));
    System.out.println(S4 == str1);
    System.out.println(S3.equals(str3));
    System.out.println(S2.equalsIgnoreCase(str2));
  }
}
```

DATE 24 03 2022

Q. What is differance between mutable and immutable object.

- In java we can create two types of objects:
  i) Mutable object
  ii) Immutable object
- Mutable means modification are possible in existing object copy.
- Immutable means modifications are not possible in existing object copy.

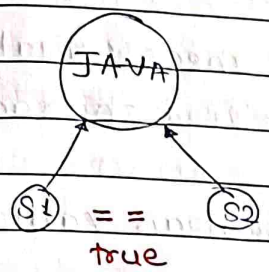
Q. Explain string Immutability.

- Immutability is the important property of string class.
- String class is immutable it is a final class.
- Immutable doesn't mean we cann't reinitialize the value.
- But after reinitialization insted of doing of modificat in existing copy one new copy will be generated.

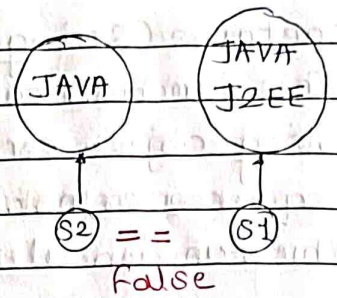
Q. Write a program to prove that string objects are immutable in nature.

```
public class StringDemo3
{
    public static void main (String [] args)
    {
        String S1 = "JAVA";
        String S2 = "JAVA";

        System.out.println (S1 == S2);

        S1 = S1 + "J2EE";

        System.out.println (S1 == S2);
    }
}
```

Before Re-initialization



After re-initialization



Explain the functions of string class.

- string class have provided several inbuilt functions to to perform operation on string objects.

1] Length ()

- This function can be use to calculate a length of string value.
- Length function always written total no. of charachters present inside the string.

2] CharAt (int index)

- This function accepts the index number as input parameters and ~~written~~ returns the character present at specified index.

3] indexOf (char ch)

- This function always accepts the character input and returns the index no. of specified character.
- If the character is having more than one occurances then this function will written index no. of first occurance.

4] lastIndexOf (char ch)

- This function can be use to display last index of specified character if it is having multiple occurances.

DATE

5] contains (String charseq):
- This function can be used to check weather the string object contains the specified character sequance or not.
- This function alway written boolean values.

e.g.
s.contains ("Ja") → true/false

6] startsWith (String charseq)
- This function can be used to check weater the string values is starting with specified character sequence or not.
- Written boolean o/p.

7] endsWith (String charseq)
- This function can be used to check weather string values is endding with specified character sequence or not.
- Written boolean o/p.

8] substring ():
- This function can be use to extract specified no. of characters from given string values.
- In other words substring means small part of original string.

e.g.
substring (int start index)
OR
substring (int: systart index, int end index)

9] toUpperCase (): convert all to uppercase

10] toLowerCase (): convert all to lowercase

DATE

Program No. 4

```
public class StringDemo4
{
    String s = "JAVA Developer";
    System.out.println (s.length());
    System.out.println (s.charAt(7));
    System.out.println (s.indexOf('a'));
    System.out.println (s.lastIndexOf (ch: 'a'));
    System.out.println (s.contains ("eve"));
    System.out.println (s.startsWith ("jav"));
    System.out.println (s.endsWith ("per"));
    System.out.println (s.substring (5));
    System.out.println (s.substring (0, 4));
    System.out.println (s.toUpperCase());
    System.out.println (s.toLowerCase());
}
public static void main (String[] args)
}
```

Output:

```
14
V
1
3
True
False
True
Developer
Java
JAVA DEVELOPER
java developer
```

11] toCharArray ()
- This function can be use to convert string value in the form of character array.
- It means this function will return written array of type characters.

Q. write a program to accept string value from end user and print all the character in reverse order as well as display original string?

```
import java.util.scanner;
public class StringDemo5
{
  public static void main (String [] args)
  {
    Scanner sc1 = new scanner (system.in);
    System.out.println ("Enter String");
    String str = sc1.next ();
    System.out.println ("ORIGINAL:" + str);
    // convert string into char array
    char [] cArray = str.toCharArray ();
    System.out.println ("REVERSE:");
    for (int a = cArray.length - 1; a >= 0; a--)
    {
      System.out.print (cArray [a]);
    }
  }
}
```

12] Split (string exp)
- Split function can be use to divide a string value into multiple substring.
- This function accepts input parameter of type string which represents condition or expression.
- Split function always written array of type string.

DATE

- If the specified character sequence is not available inside the string value then split function will written original string without any modification.

Program no.6

```
public class Demo6
{
  public static void main(String[] args)
  {
    String s="JAVA J2EE SPRING HIBERNATE";
    System.out.println(s);
    String[] courses = s.split(" ");
    for (int a=0; a<courses.length; a++)
    {
      System.out.println(courses[a]);
    }
  }
}
```

13] isEmply[]:

- This function can be use to check weather stream is empty or not.
- This fuction always writtens boolean-typ of with values.
- If string is empty then output will be true else it will be false.

14] trim():

- This function can be use to remove white space from string values.
- White space will be remove either from leading or training position.
- After removing white space this function will ~~relud~~ be written new string value.

DATE

- This function is mainly use in string validation.

Program no. 7

```
import java.util.scanner;
public class StringDemo7
 {
   public static void main (String [] args)
   {
     Scanner sc1 = new scanner (System.in);
     System.out.println ("Enter Name of country:");
     String s = sc1.nextline();
     String d = s.trim();
     boolean b = d.isEmpty();

     if (b == false)
     {
       System.out.println ("country name is: " + d);
     }
     else
     {
       System.out.println ("please enter valid county");
     }
   }
 }
```

(15) replace ()

- This function can be use to replace single characters or multiple characters ~~from~~ with new values from given string.
- Syntax:

replace (char need to be replaced , char new char)

DATE

replace (String oldsubstring, String newsubstring)

Program no. 8

```
public class StringDemo 8
{
  public static void main (String [] args)
  {
    String s = "Qspiders";
    System.out.println ("ORIGINAL:" +s);

    String p = s.replace ('Q' , 'J');   // Old char, New char
    System.out.println ("Latest:" +p);

    String q = s.replace ("Qsp" , "Jp");
    System.out.println ("Latest 1:" +q);
  }
}
```

Q.

Enter Name	→ cheak wheathe name is empty or not
Enter email	→ Email should be contain
Enter Password	→ Password must contain 6 char
Confirm Password	→ Password and confirm password must be same.

```
import java.util.scanner;
class stringDemo 9
{
  public static void main (String [] args)
  {
    Scanner sc = new scanner (System.in);
```

DATE

```
System.out.println("Enter Name");
String name = sc.nextLine();
string user.name.trim();
if (user. is Empty())
  {
    System.out.println("Enter Email");
```

classmate

PAGE

Call by value:

- The process of calling a method by providing appropriate value is known as call by value.
- If method accepts input parameters of type primitive then call by value will takes place.
- We can call a method by providing multiple values

Call by Referance:

- The process of calling a method by providing object referance is known as call by referance.
- If method accepts non-primitive input parameters then call by referance will takes place.
- It is possible to pass referances of multiple objects to the method.

- **Program 1**

Business logic class (1)

```
public class Employee
 {
     int empID = 150;
 }
```

Business logic class (2)

```
public class compony
 {
    void sheduleInterview (Employee e)
     {
        if (e.empID == 150)
         {
            System.out.println("Interview sheduled
                                Sucefully");
         }
        else {
            System.out.println("Invalid ref");
        }
 }
```

DATE

```
public class MainApp1
{
    public static void main (String [] arg)
    {
        company c1 = new company ();
        //Employee e1 = new Employee ();
        c1. SheduleInterview (new employee ());
    }
}
```

- Program 2

Business logic class - 01

```
public class Shop
{
    void getshop Info ()
    {
        System.out.println ("Shope type is electronics");
        Salesman s1 = new salesman ();
        s1. sellproduct (new product ());
    }
}
```

Business logic calass - 02

```
Class Salesman
{
    void sellproduct (product p)
    {
        p.get ProductInfo ();
        s
    }
}
```

DATE

Business logic class - 03

```
class Product
{
  void getProductInfo()
  {
    System.out.println("Product name is Laptop");
    System.out.println("product price is 50000");
  }
}
```

Executable class

```
class MainApp2
{
  public static void main(String[] args)
  {
    Shop S1 = new Shop();
    S1.getShopInfo();
  }
}
```

- Program No. 3

Business logic class:

```
public class Student
{
  void getStudentInfo()
  {
    System.out.println("Student ID is 105");
    System.out.println("student stream is CSE");
  }
}
```

DATE

Business logic class②

```
Class Institute
{
    Student getAdmission ()
    {
        Student s1=new student();
        return s1;
    }
}
```

Executable class

```
public class MainApp3
{
    pubic static void main (String [] args)
    {
        Institute i = new Institute ();
        //method chaining
        /* Student s=i.getAdmission();
         s.getstudentInfo();*/
        i.getAdmission().getStudentInfo();
    }
}
```

DATE 04 04 2022

Syllabus:

- Introduction to OOPs
- Object Composition
- Inheritance
- Types of inheritance
- Constructor call
- Constructor chaining
- Method Overloading
- Method Overriding
- Type casting
- Polymorphism
- Generalization
- Abstract class
- Interface
- Access Modifiers
- Encapsulation
- Singleton Design Pattern
- Abstraction
- Coupling

DATE

Q. What is object oriented programming.

- object oriented is a programming style which can be use to develop different applications.
- If language is object oriented then we have to represent everything in terms of classes and object.
- Following are the principals of object oriented program.
  1) Inheritance
  2) Polymorphism
  3) Encapsulation
  4) Abstraction

Explain types of object relationship.

- In JAVA one object is related with another object by using two type of relationship
  1) HAS-A Retionship
  2) IS-A Relationship.

Object A $\xrightarrow[\text{IS-A}]{\text{HAS-A}}$ Object B

HASA - object Composition
IS-A - Inheritance

Object composition:

- Object composition is also known as HAS-A relatiopship. or object Association,
- If obe object holds the property of another object then it is known as object composition.
- In case of composition are have to create object or dependancy class inside the other class which represent mainentity.

DATE

- If we are creating object of one business class inside the other business class then it can be static or non-static.
- If it is a static then we have to use class name is referance and non-static are have to provide object.

• Program 01

```
// main Entity.
public class Bike
  { //object composition.
    static Engine e1 = new Engine ();
    void drive ()
     {
      System.out.println (" Driving a Bike");
     }
  }

// dependancy class
class Engine
{
  void start ()
   {
    System.out.println (" Engine Started");
   }
  void stop ()
   {
    System.out.println (" Engine Stopped");
   }
}
```

DATE

```
public class MainApp1
{
  public static void main (String [] args)
   {
     Bike bi = new Bike ();
     Bike.et.start ();
     bi.drive ();
     Bike.et.stop ();
   }
}
```

Program 02

```
//Main Entity
public class Generator
{
  static Electricity et = new Electricity ();
  void work()
   {
     System.out.println ("Generator is Running");
   }
}

//dependancy class
class Electricity
{
  void start ()
   {
     System.out.println (" Supply of electricity started");
   }
  void stop()
   {
     System.out.println (" Supply of electricity stopped");
   }
}
```

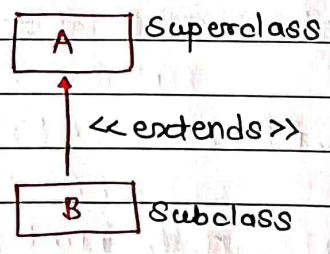
```
//Executable class
Class MainApp2
{
    public static void main (String [] args)
    {
        Generator g1 = new Generator ();
        Generator. e1. Start ();
        g1. ~~drive~~ work ();
        Generator. e1. stop();
    }
}
```

Q. Define Types of Inheritance

- In JAVA inheritance classified into 4 types
  1) Single Level
  2) Multi level
  3) Multiple
  4) Hierarchical

1) Single level Inheritance:

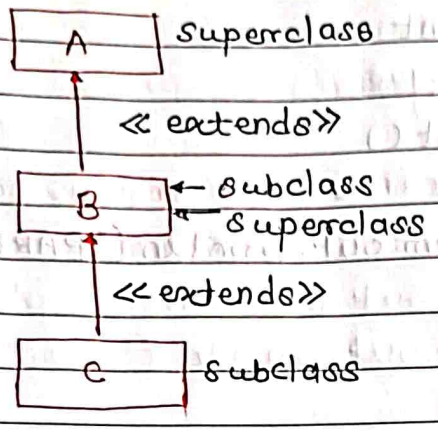
- If one subclass acquires the properties of only one superclass then it is known as single level inheritance



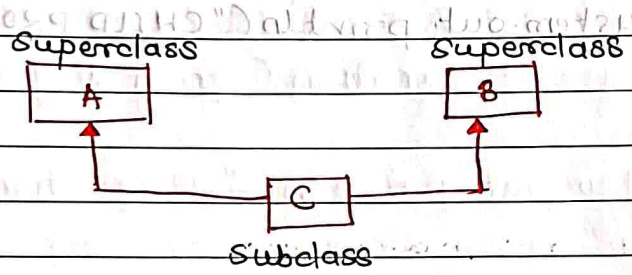
2) Multi level inheritance

- If one subclass extends the properties of one superclass and that superclass extends properties of its superclass then it is known as multi-level inheritance.

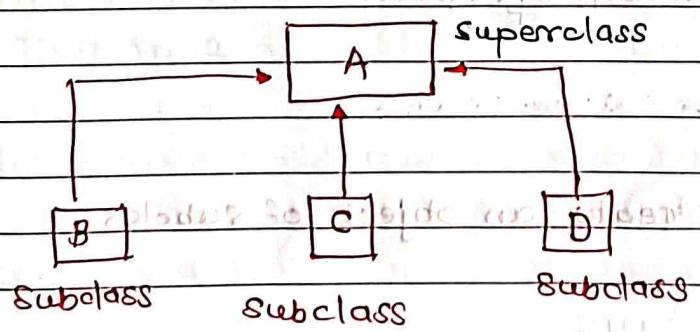
- Multilevel inheritance is also known as inheritance chain.



3) Multiple inheritance
- If one subclass is trying to access properties of more than one super classes then it will be considered as a multiple inheritance.



4) Hierarchical inheritance
- If one superclass is having more than one subclasses then it is known as heirarchical inheritance.
- In other words heirarchical inheritance mean more than one subclasses accessing properties of only one superclass.



DATE

* Example for single level inheritance:

```
// Superclass
class Parent
{
  void test()
  {
    System.out.println("PARENT PROPERTY");
  }
}

// Subclass
public class Child extends Parent
{
  void display()
  {
    System.out.println("CHILD PROPERTY");
  }
}

public class MainApp1
{
  public
  {
    // object of subclass
    Child c1 = new Child();
    c1.test();
    c1.display();
  }
}
```

Always create an object of subclass.

DATE

* program for multilevel inheritance.

```
//superclass
public class Instagram V1
{
  void makpost()
  {
    System.out.println("POST PHOTO OR VIDEO");
  }
}

//subclass as well as superclass
class Instagram V2 extends Instagram V1
{
  void stories()
  {
    System.out.println("CREATE STORIES");
  }
}

//subclass
class Instagram V3 extends Instagram V2
{
  void reels()
  {
    System.out.println("Making Reels");
  }
}

class MainApp2
{
  public static void main(String[] args)
  {
    InstagramV3 V = new InstagramV3();
    V.makepost;
    V.stories;
    V.reels;
  }
}
```

DATE

* Implematation for hierachical inheritance.

```
// Superclass
// General Information.
class Employee
{
  void basicInfo (int empID, double empSal)
  {
    System.out.println("Employe ID is" +empID;
    System.out.println(" Employee Salary is" +empSal);
  }
}

// Subclass
// Specific Information
class Permanent Employee extends Employee
{
  void getDesiganation (String desg)
  {
    System.out.println("Designation is" + desg);
  }
}

// subclass    // specific Information
class ClientSideEmployee extends Employee
{
  void get contract details (int months)
  {
    System.out.println("Contract Duration is"+months
  }
}

class MainApp 3
{
  public static
  {
```

```
PermanentEmployee P1 = new PermanatEmployee();
P1.basicInfo (empID: 101, empSal = 25000);
P1.getDesignation (desg: "HR");

System.out.println ("=========");

ClientSlideEmployee E1 = new ClientSlideEmployee();
C1.basicInfo (empID: 102, empSal: 30000);
C1.getContractDetails (months: 15);
}
}
```

* **Inheritance:**

- Inheritance is also known as IS-A relationship.
- If one class acquires the properties of another class then it will be considered as inheritance.
- The class which acquires properties of another class is also known as subclass.
- The class from which properties are going to inheritance is known as super class.
- Inheritance is possible by using extends keyword.
- Inheritance is applicable only if two objects belongs to same category

O/P

```
Employee ID is: 101
Employee Sal is: 25000
Designation is HR
=========
Employee ID is: 102
Employee Sal is: 30000
~~Desi~~ contract details: 15 monts.
```

DATE 06 04 2022

Q. Explain Constructor call?

* Constructor call

1) The process in which subclass constructor invokes the constructor of superclass is known as constructor call.
2) If two classes are having IS-A relationship the constructor call is mandatory.
3) To make a constructor call we have to use super-statement.
4) Superstatement must be the 1st statement of constructor body.
5) It is not possible to declare multiple super statements inside the constructor.
6) Constructor call can be done implicitly as well as explicitly.

Q. Explain Implicitly constructor call?

- If compiler calls the constructor of superclass by providing super statment inside the subclass body then it is known as implicit constructor call.
- Implicit constructor call is possible only if superclass constructor is zero parameterized.

Q. Explain Explicit constructor call?

- If developer calls the constructor of super class by providing super statement inside the subclass constructor body then it is known as explicit constructor call.
- If superclass constructor is parameterized then constructor call must be done explicitly.

DATE

- Example for Implicit Constructor Call

```
//Superclass
class Master
{
  master()
  {
    System.out.println("superclass constructor");
  }
}
//Subclass
class central extends master
{
  central() // Implicit subclass/constructor
  {
    system.out.println("subclass constructor");
  }
}

class MainApp1
{
  public Static Void Main(String[] args)
  {
    central c1 = new. central();
  }
}
```

- Program for explicit constructor

```
//Superclass
class Master{
  Master(int a)
  {
    System.out.println(a);
    System.out.println("superclass constructor");
  }
}
```

DATE

```
//Subclass
class Central extends Master
{
    Central() // explicitly constructor call
    {
        Super(20)
        System.out.println("subclass Constructor");
    }
}

class MainApp2
{
  public static void main(String[] args)
  {
    central c1 = new central();
  }
}
```

O/P

a: 20

Superclass constructor

Subclass constructor

Q. Why Java doesn't support multiple Inheritance?

- Multiple inheritance means one subclass is trying to access properties of more than one superclass.
- Java doesn't supports multiple inheritance because of following reasons.

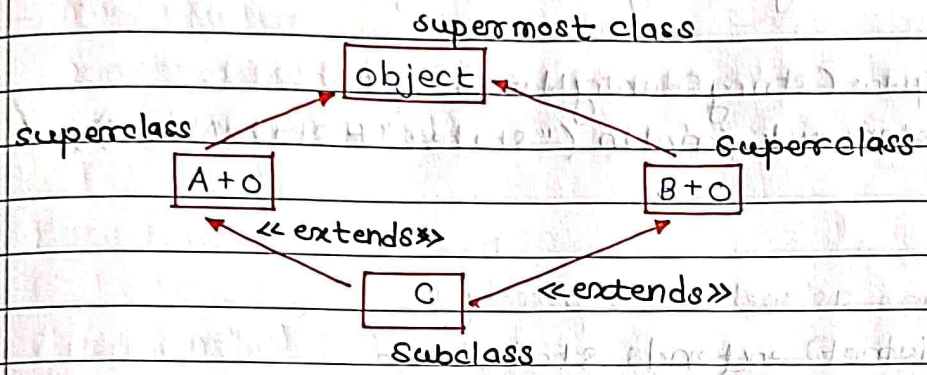
1) Subclass is extending the properties of more than one superclasses then we have to call constructors of the respective super class.

2) But it is not possible to use multiple super statements inside the constructor body.

3) Every superclass in java internally extends properties of an object class which creates

diamond ambiguty problems.

- Due to this reasons we can not achieve multiple inheritance in java.



• Diamond Ambiguty Problem •

Q. What is Constructor chaining?

- Subclass constructor calls the constructor of superclass that superclass constructor call the constructor of its superclass this process is known as constructor chaining.
- Constructor chaining is possible only of two class is having IS-A relationship.
- In java everyclass must have a superclass either defined by compiler or developer.
- If programmer has not defined a superclass then compiler will decleare object as a superclass.
- It means every java class is having properties of object class because of this reason java is object oriented programming language.

```
//Superclass
class country
{
  country (string countryName)
  {
    system.out.println "(country = + countryName);
  }
}
```

DATE

```
class State extends Country
{
  State (String CountryName, String StateName)
  {
    super (String CountryName)
    System.out.println ("State : " + StateName");
  }
}
//Subclass as well as superclass
class District extends State
{
  District (String countryName, String StateName, String DistrictName
  {
    super (String CountryName, String StateName);
    System.out.println (" District : " + districtName);
  }
}

class MainApp1
{
  public static void main (String [] args) {
  District d1 = new District ("India", "Maharashtra",
                                "Pune");
  }
}
```

O/P

India

Maharashtra

Pune

DATE

Q. What is the use of this statement?

- This statement can be use to call another constructor of same class.
- This statement should be the first statement of constructor body which means multiple this statements are not allowed.

```
class Run
{
  Run (int a)                    ①
  {
    System.out.println ("A:"+a);
  }
  Run (String s)                 ②
  {
    this (int a: 25);
    System.out.println ("S:"+s);
  }
  Run (char c)                   ③
  {
    this (String:"JAVA");
    System.out.println ("C:"+c);
  }
}

class Execute extends Run
{
  Execute ()
  {
    Super (c:'J');
    System.out.println ("Subclass Constructor");
  }
}
```

```
class MainApp
{
  public static void main(String [] args)
  {
    Executable ex = new Executable();
  }
}
```

O/P:

A : 25

S : JAVA

C : J

Q. What is the use of super keyword?

- super keyword can be used to access data and function members of superclass.
- It is possible to differentiate superclass and subclass members if they have same identity.

```
class Vehicle
{
  int speed = 340;
  double price = 500000;
}

class Bike extends Vehicle
{
  int speed = 120;
  double price = 50000;
  void display()
  {
    System.out.println("Speed:"+speed);
    System.out.println("price:"+price);
    System.out.println("Speed:"+super.speed");
```

```
        System.out.println("price:" + super.price);
    }
}

class MainApp2
{
    public static void main(String[] args)
    {
        Bike b1 = new Bike();
        b1.display();
    }
}
```

Q. Write the differance between super(), super keyword, this(), this keyword.

super()	super keyword	this()	this keyword
1) super statement can be used to call the constructor of superclass.	1) super keyword is used to access variables and methods of superclass	1) this can be used to call another constructor of same class.	1) this keyword used to differentiate instant and local variable.
2) super() must be declared inside the constructor body.	2) We can declare super key anywere inside the class.	2) this() must be declared inside the constructor body	2) this keyword can be declare inside the constructor as well as Non-Static methods.
3) super() must be the 1st statement of constructor body.	3) its not mandatory to declare super keyword as 1st statement.	3) this() should be the 1st statement of constructor body.	3) It not mandatory to declare this keyword as 1st statements.

4) Multiple super statement are not allowed inside the constructor.	4) We can use super keyword multiple time.	4) Multiple this() are not allowed inside the constructor.	4) coe can use this keyword multiple time

Q. What is method Overloading?

Method Overloading:

- The process of creating multiple method with the same name but differant arguments is known as method overloading.
- In case of method overloading parameters must be different either by length or datatype.
- Method overloading can be used to achieve same functionality with differante parameters.
- It is possible to overload static as well as non-stat methods.
- subclass can overload method from super class.
- Method overloading can be used to achive compile time polymorphism.

• Program No.1

```
class Facebook
{
  string email = " abc@gmail.com";
  string password = "123456";
  int contact = 7077000;

  void login (String userEmail, string userpassward)
  {
    if (userEmail.equals (email) && userPassward.equals(pass
    {
      system.out.println ("login Sucessfully");
}
```

```
else {
        system.out.println("Invalid details");
    }
}

void login(int usercontact, String userPassword)
{
  if (usercontact == contact && userpassword.equals(password))
   {
     System.out.println("Login Sucessfully");
   }
  else
   {
     System.out.println("Invalid details");
   }
}

import.java.util.Scanner;
class MainApp1
{
  public static void main(String[] args)
  {
    Scanner sc1 = new Scanner(System.in)
    Facebook f1 = new Facebook();
    System.out.println("Select Login Method");
    System.out.println("1: Login by Email" \n "2: Login by
                                contact");
    int choice = sc1.nextInt();
    if(choice == 1)
    {
      System.out.println("Enter Email");
      String email = sc1.next();
      System.out.println("Enter Password");
      String pass = sc1.next();
```

```
        f1.login (email, pass);
      }
    else if (choice == 2)
      {
        System.out.println ("Enter ~~Email~~ contact");
        int contact = sc1.nextInt();
        System.out.println ("Enter Passward");
        String pass = sc1.next();
        f1.login (contact, pass);
      }
    else
      {
        System.out.println ("Invalid Choice");
      }
  }
```

- Program No.2

```
class calculator
{
  static void add (int a, int b)
    {
      System.out.println (a+b);
    }
  static void add (double c, double d)
    {
      System.out.println (c+d);
    }
  static void add (int p, int q, int r)
    {
      System.out.println (p+q+r);
    }
}
```

DATE

```
class MainApp2
{
  public static void main (String [] args)
  {
    calculator.add (10,20);
    calculator.add (10.25, 20.45);
    Calculator.add (80, 50,100);
  }
}
```

O/P → 30 30.70 230

- Program No.3

```
class Vehicle {
  void searchVehicle (int rcNo) {
  System.out.println ("searching vehicle details for: "+rcNo);
  }
}

class Bike extends Vehicle {
  void searchVehicle (String model) {
  System.out.println ("searching details for:" +model);
  }
}

class MainApp3
{
  public static void main (String [] args)
  {
    Bike b1 = new Bike ();
    b1. searchVehicle (12345);
    b1. searchVehicle ("HERO");
  }
}
```

DATE

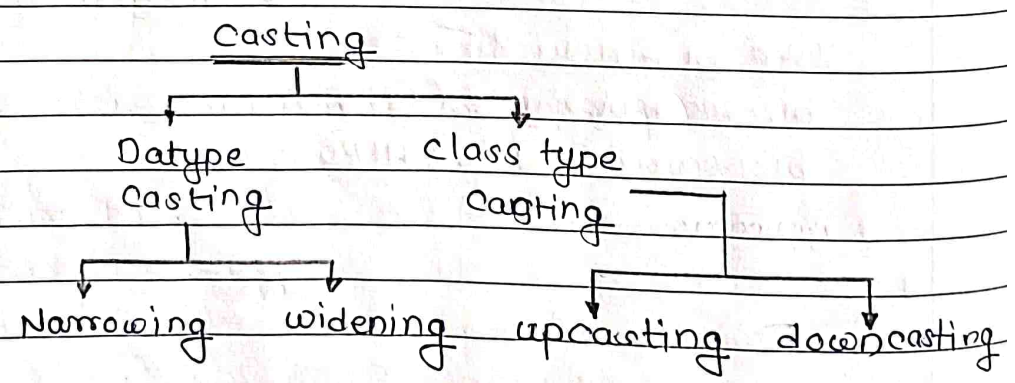
DATE

Q. Explain differance between overloading method and method overriding.

Overloading	Overriding
1) The process of creating multiple methods with same name but diff. parameters is known as method overloading	1) The process of inheriting method from superclass and changing its implementation in sub class is known as method overriding.
2) we can overload static as well as non-static method	2) We can override only non-static method.
3) Overloading can be use to achive same functionality with different parameter	3) overriding can be use to achieve different implentation with a same parameters
4) Method overloading is possible without inheritance	4) Method overriding is not possible without inheritance.
5) Overloading can be use to achieve compile type polymorpism	5) overriding can be use to achieve ~~front time~~ run time polymorphism.

Q Explain Casting process in JAVA.

- The process of converting one type of information into another type is known as a casting.
- Casting can be done by useing 2 ways:
  1) Datatype Casting
  2) Class type Casting



DATE

Q. What is datatype casting?

Datatype Casting:

- Datatype casting is also known as primitive casting.
- The process of converting one primitive value into other primitive datatype is known as datatype casting.
- Datatype casting further classified into 2 types:
  1) Narrowing
  2) Widening

1) Narrowing:

- The process of converting higher type of information into its lower type is known as.
- Narrowing must be done explicitly (By developer).
- It is also known as explicit or forceful casting.

2] Widening:

- The process of converting lowertype of information into its higher type is known as a widening.
- Widening can be done implicitly as well as explicitly.
- It is also known as implicit or automatic casting.

Note:

It is not possible to perform casting operation if input type is boolean

Program 1

```
class CastingDemo1
{
  public static void main (String [] args)
  {
    //matching Info
    int k=20;
    double p=32.25;
    System.out.println (k+"\t\t"+p);
```

```
// narrowing
    int j = (int) p;
    System.out.println (j); //82
// widening
    double d = k;
    System.out.println (d); //20.0
  }
}
```

Program 2

```
public class CastingDemo2
{
  public static void main (String[] args)
  {
    float f1 = 25.368 f;
      // narrowing
      int i1 = (int) f1;
      System.out.println (f1+" \t\t" + i1);

      double d1 = 12.35967897 6
      // narrowing
      float f2 = (float) d1;
      System.out.println (d1+" \t\t" + f2);
  }
}
```

Program 3

```
class CastingDemo3
{
  public static void main (String[] args)
  {
      char ch1 = 'J';
      char ch2 = 'k';
```

DATE

```
//widening
int x1 = ch1;
int x2 = ch2;
System.out.println(x1+"\t\t"+x2);   out 74   75

int x3 = 65;
//narrowing
Char ch3 = (char)x3;
System.out.println(ch3);   out → 'A'
}
}
```

Program 4

```
Class CastingDemo4
{
public static void main(String[] args)
{
//narrowing
short s1 = (short) 789658;
System.out.println(s1); [3226] (Garbage Value)
}
}
```

Q. What is Garbage Value?

Garbage Value:

- This is random value generated by JVM at the time of casting op
- If developer is trying to cast one type of information into another type then JVM will generate garbage value if conversion is not valid

Q. Is it possible to convert int value into short datatype.

- The process of converting int value into short datatype will be considered as narrowing and must be done

ex

- But it is not recommanded to perform such type of conversion because user will never get genine output.
- Following is approach to convert one integer value into other integer datatype

long - 8
int - 4
short - 2 → widening (recommonded)
byte - 1

long - 8
int - 4
short - 2 → narrowing (not recommonded)
byte - 1

Q. What is class type casting?

- Classtype Casting:
  - The process of casting properties of one class into another class is known as class
  - To perform classtype casting we have to follow rules:
    i) At the time of class type casting two classes should have IS-A relationship.
    ii) The object which has to be casted should have the properties of target class.
  - class type casting further divided into 2 types
    i) Upcasting
    ii) Downcasting

i) Upcasting:
- The process of casting subclass into superclass is known as a upcasting.
- In other words upcasting means object of subclass but referance of superclass.

DATE

- Upcasting can be done implicitely as well as explicitly.
- After upcasting we can not access the property of subclass which means it is possible to access only general properties

• Program 1:

```
//Superclass
class Demo
{
  void test ()
  {
    System.out.println ("TEST METHOD");
  }
}

//Subclass
class Sample extends Demo
{
  void display ()
  {
    System.out.println("DISPLAY METHOD")
  }
}

class MainApp1
{
  public static void main (String [] args)
  {
    Sample s1 = new Sample ();
    s1.test ();
    s1.display();
    //Upcasting
    Demo d1 = new Sample ();
    d1.test ();
  }
}
```

DATE

- Program 2 :

```
// superclass
class Machine
{
  void getType
  {
    system.out.println ("DISPLAY Machine Type");
  }
}

// subclass
class laptop extends machine
{
  //general info  @Override
  void gettype ()
  {
    system.out.println ("Machine type is laptop");
  }
  //specific Information
  void displayprocessorInfo()
  {
    system.out.println ("Machine processor is Intel")
  }
}

// subclass
class AC extends machine
{

  void getType ()
  {
    system.out.println ("Machine type is AC");
  }
```

DATE

```
void displaycapacity()
 {
   System.out.println("Capacity of AC is 3 ton);
 }
}
//acess general and specific properties
class MainApp2
{
 public static void main (String [] args);
 {laptop l1 = new Laptop();
  l1.getType(); //General
  l1. displayprocessorInfo(); //specific

  AC a1 = new AC();
  a1.get type(); //General
  a1. displayCapacity(); //specific
 }
}

//acess only general properties
class MainApp3
{
 public static void main (String [] args)
 {
    Machine M1;

    m1 = new Laptop(); // Upcasting
    m1.gettype();

    m1 = new AC(); //upcasting
    m1.get type();
 }
}
```

DATE

- Program 3

```
class Bike
{
  void getType
  {
    System.out.println("Display bike type");
  }
}

class petrolbike extends bike
{
  void getType()
  {
    system.out.println(" Bike type is Petrol");
  }
}

class Electricbike extends bike
{
  void gettype()
  {
    System.out.println("Bike type is electric");
  }
}

class MainApp4
{
  public static void main(String[]args)
  {
    Bike bt;

    bt = new petrolbike();
    bt.gettype();
    bt = new electricbike();
    bt = gettype();
  }
}
```

DATE 18 04 2022

2) Downcasting:

Q. write a program to demonstate downcasting operation.

```
//super class
class Car
{
  void gettype ()
  {
   System.out.println("Display car type?
  }
}
//subclass
class Electricat extends car
{

  void get type()
  {
   System.out.println (car type is electric)
  }
  void get battery Info ()
  {

  }
}

class Mainclass App4
{
  public static void main (string [] arg)
  {
   Electrical car  c1 = new electric car ();
   c1.get type ();
   c1.get Battery Info ();
  upcasting
   car c2 = new Electriccar ();
   c2.get type ();
```

DATE

```
// downcasting
Electric car e1 =(Electric car) new car();
}
```

Q What is class cast exception

- It is type of subtype exception thrown by JVM.
- It is also known as unchect exception

Q When it occurs

- If developer is ~~typen~~ trying to perform downcasting without upcasting then ~~throw~~ JVM will throw this exception.

Q Why compilar ~~not~~ able through class cast exception.

- Compilar check only syntax and rules of language hence will not get only compile time error or exception

Q. How to avoid class cast except

- To avoid class cast exception we have to use instance of operator.

Q. Program to avoid Classcast Expection.

```
class MainApp5
{
public static void main (String [] args)
 {
    car c1= new car();
    if (c1 instence of Electric car);
    {
{   Electriccar e1=(Electriccar) new car();
    }
else{
        system.out.println("Pro
    }
```

classmate

PAGE

DATE

Q. What is Downcasting:

- The process of casting superclass properties into subclass is known as downcasting.
- In other words downcasting means referance of subclass but object of superclass.
- Downcasting must be done explicitily.
- Downcasting is possible only on abcasted object.

Program:

```
//superclass
class Student
{
  void getStream():
  {
    System.out.println("Get student stream");
  }
}

// subclass
public class ComputerStudent extends Student
{ @override
  void getstream()
  {
    System.out.println(Student Stream is computer);
  }

// most specific
void getComputerSubjects()
  {

  }

class MainApp6
{
  public
    {
```

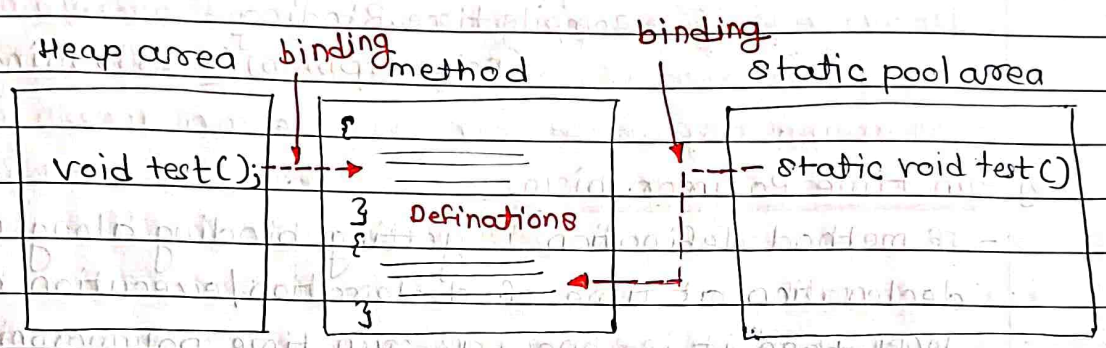
```
//Upcasting
Student s1 = new computer student();
s1.getstream();
//Downcasting
ComputerStudent c1 = (computerstudent) s1;
c1.getstream();
c1.getcomputersubjects();
}
}
```

Upcasting	Downcasting
1) Process of casting subclass info into superclass is known as upcasting.	1) Process of casting superclass info into subclass is known as downcasting
2) Upcasting is the process creating object of subclass with referance of superclass	2) Downcasting is the process creating object of superclass with a referance of subclass.
3) Upcasting means casting most specific properties to most general properties.	3) Downcasting means casting most general properties into most specific properties
4) Upcasting can be done implicitly as well as explicitely	4) Downcasting must be done explicitely.
5) Upcasting is possible without downcasting.	5) Downcasting not possible without downcasting
6) After upcasting we can acess only general properties.	6) After downcasting we can acess general as well as specific properties.

DATE

Q. what is binding?

- The process of integrity method decleraton and defination is known as
- In other words binding means attaching method decleration along with its defination.
- Binding can be done at the time of compilation or execution.



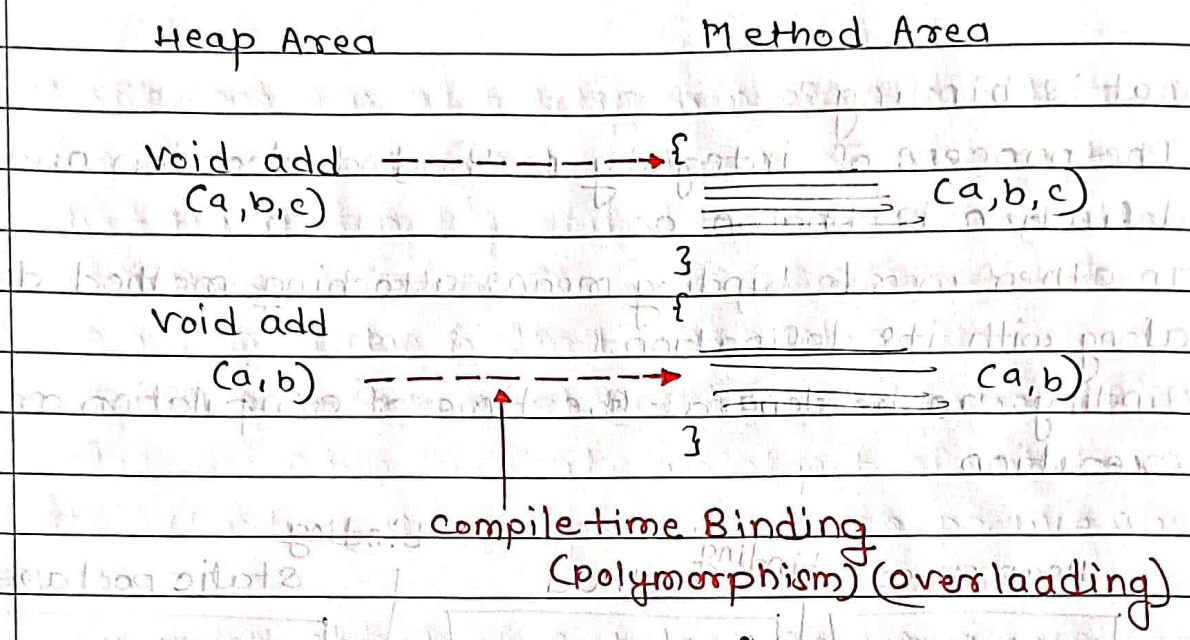
Q. What is polymorphism?

- **Polymorphism:**
- An object showing different behaviour at different stages of its lifecycle then it is known as polymorphism.
- It is one of the OOPs principle.
- Polymorphism classified into two types:
  1) Compile time polymorphism
  2) Run time polymorphism

1) **Compile time polymorphism:**

- If method defination is getting bindes along with its decleration at the time of compillation by compiler then it is known as compile time polymorphism.
- compile time polymorphism is also knon as static binding or early binding.
- Static binding cannot be rebainded. (rebinded)

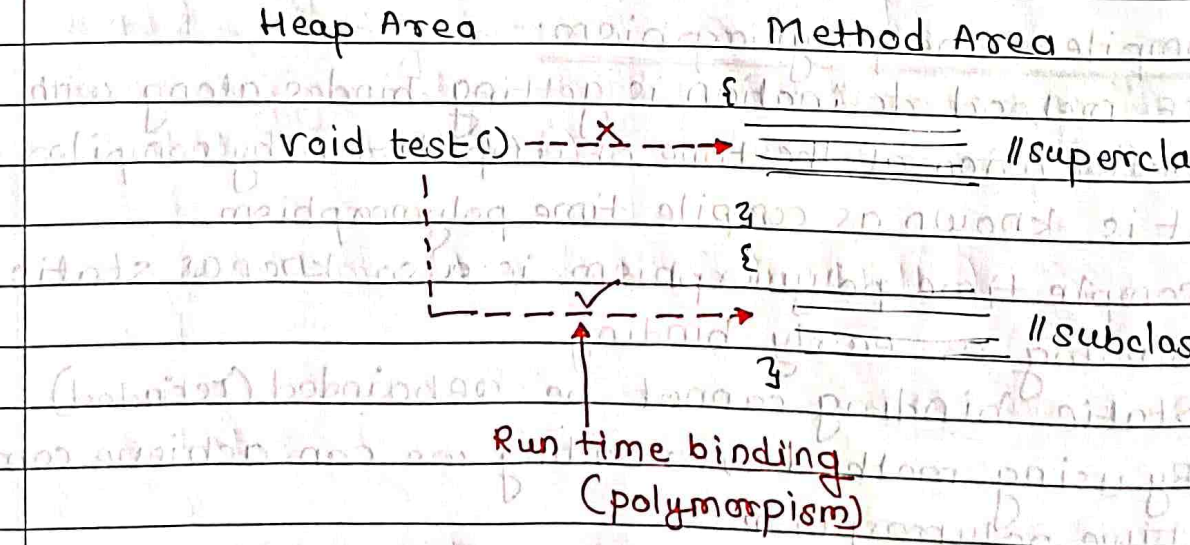
By using method overloading we can achieve compile time polymorphism.



2] <u>Run time Polymorphism.</u>

- If method defination is getting binding along with its decleration at time of decleration/execution by the JVM then it is know as run time polymorphism.
- It is also known as dynamic binding or late binding.
- Dynamic binding can be rebinded.
- To achieve run time polymorphism we can use following concepts
  1) Inheritance
  2) Method overriding
  3) Upcasting.

- Runtime polymorphism can be use to achieve generalization



DATE 20 04 2022

Q. What is generalization?

- Generalization is the process of extracting common properties from all the subclasses and storing into one superclass.
- We can achieve generalization through run time polymorphism.

Q. explain abstract class.

• Abstract Class:

- The method which having decleration as well as defination is known as concrete method.
- The class which contains concrete method which consider as concrete class.
- The method which is having only decleration is known as abstract method.
- Abstract method must be decleare inside the abstract class.
- Abstract class can have only abstract method, only concrete method and abstract as well as concrete method.
- We cannot create object of abstract class because new operator does not work with abstract keyword.
- Subclass can extends the properties of superclass which is abstract in nature. In such a case subclass is responsible to provide an implimentation for all the abstract method if not then subclass will also become abstract class.

Q. Explain structure of abstract class.

- We can decleare following members inside the abstract class:

abstract clas _____

//static non-static variables

// non-~~static~~ static abstract and concrete methods.

// Static concrete metod.

// static and nonstic block.

// constructor.

DATE

Q. How to acess non-static members of abstract class.

- Since we cannot create object of abstract class directly it is not possible to acess non-static members.
  e.g. non-static method, constructor
- In order to access non-static properties of abstract class we have to create an object of subclass wich is extending the properties of respective abstract class.
- By using object of subclass we can access non-stic variable methods and constructor of abstract class.

• Program 1.

```
// Structure of abstract class
public abstract class Demo
 {
    static int K=10;
    int j=20;

    abstract void test()

    void display()
     {
        System.out.println("Display Method")
     }

    static void run()
     {
        System.out.println("Run Meth")
     }

    static
     {
        System.out.println("static block");
     }
```

classmate

PAGE

```
{
  System.out.println("Non-static block");
}
Demo ()
  {
  System.out.println("constructor");
  }
}

class MainApp1
{
  public static void main(String[]args)
  {
    System.out.println("K :"+Demo.K);
    Demo.run ();
  }
```

- Program 2:

```
public abstract class Employee
{
  abstract void calculateSalary (double salary);
}

public class HR extends Employee
{@Override
  void calculateSalary (double salary)
  { //10000
    double netsalary = salary+10000;
    System.out.println("NET SALARY:"+netsalary);
  }
}
```

```
class manager extends employee
{
  void calculateSalary (double salary)
  {
    double netsalary = salary + 20000;
    System.out.println ("Net salary" + netsalary);
  }
}

class MainApp2
{
  public static void main (String [] args)
  {
    Employee e1;
    e1 = new HR();
    e1.CalculateSalary(15000);

    e1 = new manager();
    e1.calculateSalary(20000);
  }
}
```

- Program 3

```
// Superclass
Public abstract class Instagram_v1
{
  void PostPhoto()
  {
    System.out.println("Posting Photos");
  }
  abstract void reels();
  abstract void stories();
}
```

DATE

```
//Subclass
public abstract class Instagram_V2 extends Instagram_V1
{
  void reels()
  {
    System.out.println("uploading reels");
  }
}
//subclass
Public class Instagram_V3 extends Instagram_V2
{
  void stories()
  {
    system.out.println("Upload stories");
  }
}

Public class MainApp3
{
  public static void main(String [] args)
  {
    Instagram_V3 V = new Instagram_V3();
     V.postPhoto();
      V.reels();
      V.Stories();
  }
}
```

DATE

Q. What is interface?

Interface:

→ 1) Interface is java type defination block which can be used to store abstract methods.

2) Concrete methods are not allowed inside the interface

3) By default all variables declared inside interface are public, static and final.

4) A java class can implements properties of an interface

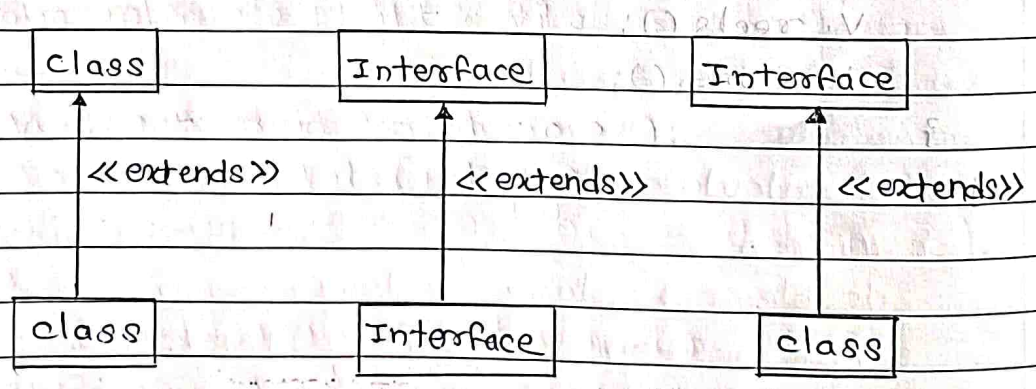
5) It is not possible to create object of interface because it contains only abstract methods.

6) One interface can extends the properties of another interface

7) Interface is used to acheive 100% abstraction

8) following is structure of interface:

```
interface _____
{
  // non-static abstract members
  // static method concrete
  // public static final variables
}
```



DATE

Q. Explain real time use of interface?

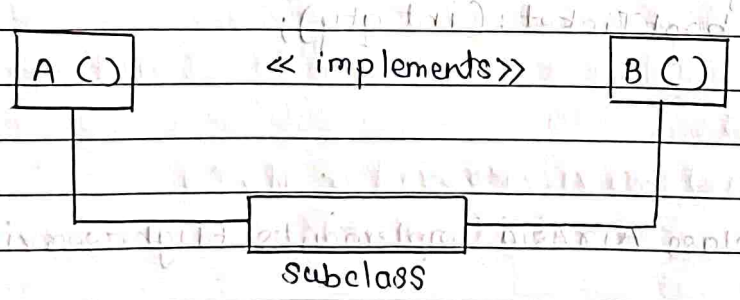
→ - Interface can be used for:

1) API creation
2) Multiple inheritance achievement
3) To achieve loose coupling.

- API acts as a middleware between 2 applications or layers.
- It is possible to acheive, multiple inheritance by using interface because interface doesn't have properties of object class.
- Loose coupling means are object is independant of another objet, By using interface we can create loosely coupied objects.
- Tight coupling means one object is purely dependant on another object.

Q. How to acheive multiple inheritance by using interface?

→ - By defaut java does not support multiple inheritance because it creates diamond ambiguity problem.
- One class cannot extends properties of more than one superclasses but same class can implements properties of more than one interfaces.
- To acheive multiple inheritance we have to create common implementation class for multiple interfaces.



- **Structure of interface:**

```
public interface Demo
 {
    int p = 20;
    void test ();
    Static void display ()
    {
      System.out.println ("Display Method");
    }
 }

Public class MainApp
 {
   public static void main (String [] args)
    {
      System.out.println ("p:" + Demo.p);
      Demo.display ();
    }
 }
```

- **Program No. 1**

```
Public interface FlightService //Super interface
 {
   void getServiceProviderInfo ();
   void bookTickets (int qty);
 }

//Subclass
Public class AirAsia implements FlightService
 {
   public void getServiceProviderInfo()
    {
      system.out.println ("service provider name is
                                Air Asia");
    }
```

DATE

```
@override
public void bookticketS (int qty)
{
  double total = qty * 2500;
  System.out.println("Total AMT:"+total)
}
}

//subclass
public class Indigo implements FlightService
{ @override
  public void getServiceProviderInfo()
  {
    System.out.println("Service Provider name is
                        Indigo");
  }
  @override
  public void booktickets(int qty)
  {
    double total = qty * 2700;
    System.out.println("Total Amt:"+total)
  }
}
//subclass
public class AirIndia implements flightservice
{ @override
  public void getServiceProviderInfo()
  {
    System.out.println("Service provider name is
                        AirIndia");
  }
  @override
  public void booktickets(int qty)
  {
    double total = qty * 2400;
    System.out.println("Total Amt:"+total)
  }
}
```

DATE

```
import java.util.Scanner;
class MainApp1
{
    public static void main (String[] args)
    {
        Scanner sc1 = new Scanner (System.in);
        System.out.println("select service provider");
        System.out.println("1: AIR ~~INDIA~~ Asia");
        System.out.println("2: Indigo");
        System.out.println("3: Air India");
        int choice = sc1.nextInt();
        System.out.println("Enter no. of tickets");
        int count = sc1.nextInt();
        Flight services fs = null;
        if (choice == 1)
        {
            fs = new AirAsia (); // upcasting
        }
        else if (choice == 2)
        {
            fs = new Indigo(); // Upcasting
        }
        else if (choice == 3)
        {
            fs = new Air India; // Upcasting
        }
        else
        {
            System.out.println("Invalid choice");
        }
        fs.getserviceProvider info ()
        fs.bookTickets (count);
    }
}
```

DATE

Q. write a program to achieve multiple inheritance by using interfaces.

```
@functional Interface //JDK 1.8+
public interface AmazonMusic
{
  void playmusic();
}

Public interface Primevideo
{
  void playMovie();
}

//multiple Inheritance
public class PrimeAccount implements AmazonMusic,
Primevideo
{ @override
  public void playmusic()
  {
    System.out.println("Playing Music");
  }
  @override
  public void playmovie()
  {
    System.out.println("Playing Movie");
  }
}

Class MainApp2
{
  public static void main(String[]args)
  {
    PrimeAccount p1= new PrimeAccount();
    P1.playmusic();
    P1.PlayMovie();
}
```

DATE

Q. What is Functional Interface?

- The interface which contain only one abstract meth is known as
- Functional interface was introduce in JDK 1.8+
- We can create Functional interface by using @functional interface annotation
- Annotation provides additional information about class or interface.

• Program No. 3

```
public class frontend //superclass
{
    void designUI ()
    {
        System.out.println ("Designing UI by using
                                HTML, CSS, JS");
    }
}
@functional interface
public interface Backend
{
    void developServerProgram (String language);
}
@functional interface
public interface DataBase
{
    void designDatabase (String vendor);
}
// multiple interface
public class Application extends FrontEnd Implements
BackEnd, Database
{ @override
    public void developServerProgram (String language)
```

DATE

```
{
    System.out.println("Developing Server Program by using" + language);
}
@override
public void designDatabase(String vendor)
{
    System.out.println("Designing database by using!"+vendor)
}
}

class MainApp3
{
    public static void main(String[] args)
    {
        Application a1 = new Application();
        a1.designUI();
        a1.developServerProgram(language:"JAVA");
        a1.designDatabase(vendor:"MySQL");
    }
}
```

DATE 25 04 2022

Q Explain Access Modifiers?

Access Modifiers:

- Access Modifiers are used to apply visibility restriction on program data for members of class
- Following modifiers are used to apply visibility restrictions:
  1) Private
  2) Package load (default)
  3) Protected
  4) Public

1) Private

- If members are having private access then we cannot access those members outside the class.
- Which means private members are visible only inside the same class.

2)

- It is also known as default modifier.
- If members are having package level access then we cannot use this members outside the packag

3) Protected

- If members are protected in nature then we can access those members anywhere inside the same package as well as outside package only through inheritance.

4)

- If members are having public access then we can use those members anywhere inside the same project as well as outside the project.

DATE

Q. Explain Non-Access Modifiers in JAVA?

- Non-Access Modifiers:
  - Non-access modifiers cannot change the visibility of members but it is possible to change implimentation
  - following are the non-access modifier:
    1) Static
    2) ~~Non-s~~ abstract
    3) final

- Program No.1

```
Package Modifiers1;  ——→ 1st package

public class Master
{
  private int k=10;
  int j=20;  ———— (Package)
  protected int s=30;
  public int d=40;
}

Public class MainApp1
{
  public static void main (String [] args)
  {
    Master m1 = new Master ();
    System.out.println ("J:"+m1.j);
    System.out.println ("S:"+m1.s);
    System.out.println ("D:"+m1.d);
  }
}
```

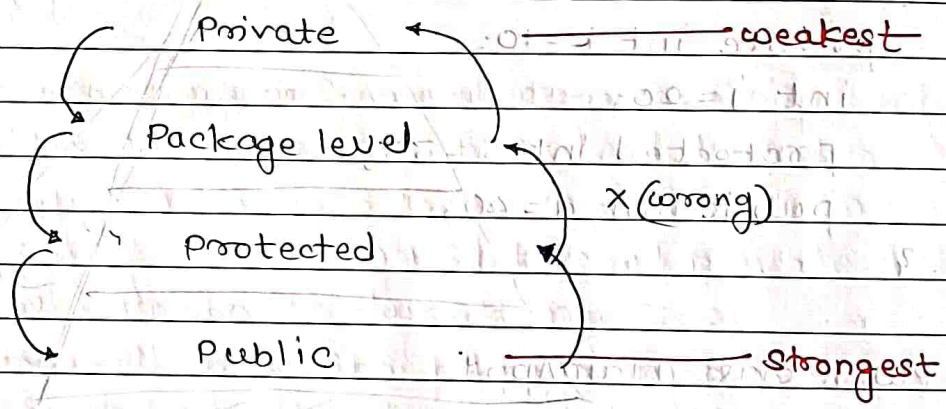
O/P —

20

30

40

```
Package Modifiers2;  ——— 2nd package

import modifiers1.Master;  ——— because of 2nd package we need to import
class MainApp2 extends Master
{
    public static void main(String [] args)
    {
        MainApp2 m1 = new MainApp2 ();  — object creaction of subclass
        System.out.println ("S:" +m1.s);
        System.out.println ("D:" +m1.d);
    }
}
```



Explain the rules for using access modifiers.

Section 1:

- If one class is declare inside another class then it is known as inner class.
- Inner class can have any of 4 access modifiers but outer class must be public or package level.
- All members of class are having package level access by default where as members of interface are public in nature.

DATE

Section 2:

- It is possible to declare multiple classes inside the single JAVA file. In a such a case only one class have public access and rest of the classes will be having package level access.
- File name should be same as a class name which is having public access.
- If all classes are having package level access then we can provide any class name as a file name.

Section 3:

- Constructor can have any of 4 acess modifier.
- Default constructor will be having same access as that of class.
- If constructor is private then object creation is not possible.

DATE 28 04 2022

Q. What is Encapsulation?

- Encapsulation is one of the OOPs principle.
- The process of binding members of the class along with class body and protecting them using access modifier is known as encapsulation.
- In other words encapsulation means binding data members and function members as a single unit.
- Encapsulation can be use to acheive data hiding.
- Encapsulation can be done by using following concepts:
  1) JAVA bean class
  2) Inner class

1) JAVA Bean Class:

- The class which contains private datamembers and public getters, setters is known as JAVA bean class.
- Getters method can be used to read private data member outside the class.
- Setters methods are used to modify private data members outside the class.
- JAVA bean class is also known as DTO class. DTO stands for Data Transfer Object.
- By using JAVA Bean Class we can acheive encapsulati

• Program No.1

```
// Java bean class or DTO class
public class Product
{
  private int productID;
  private string product Name;
  // Read
  public int getProduct ID()
  { Return productId;
  }
```

DATE

```
// write
  public void setProductID (int ProductID)
  {
    this. productID = product Id;
  }
// Read
  public String ~~set~~ ProductName ()
  {
    return ProductName;
  }
// Write
  public void setProductName (String ProductName)
  {
    this. productName = productName;
  }
}

class MainApp1
{
  public static void main(String [] args)
  {
    product p1 = new Product ()
    //read and write productID
    p1. setProductID (36);
    int id = p1. getProductID ();
    //read and write productName
    p1. setProductName ("LAPTOP");
    String name = p1. getProductName ();

    System. out. println ("Product ID" + id);
    System. out. println ("Product Name" + name);
  }
}
```

DATE

- Program No.7

```
// Outer class
class Laptop
{
  String company = "HP";
  // Inner class
  class RAM
  {
    int ramSize = 8;
    void displayRamInfo()
    {
      System.out.println("Ram Size is "+ramsize+"GB");
      System.out.println("Ram type is DDR4");
    }
  }
  // inner class
  class HardDisk
  {
    int hardDiskMemory = 500;
    void displayHardDiskInfo()
    {
      System.out.println("Hard Disk Memory"+
                          hard diskMemory +"GB")
      System.out.println(Hard disk type is SSD)
    }
  }

class MainApp2
{
  public static void main (String [] args)
  {
    Laptop l = new Laptop();
    System.out.println("company name is"+l.company);
```

```
Laptop.Ram r1 = l.new RAM();
r1.displayRamInfo();

Laptop.HardDisk h1.l.new HardDisk();
h1.displayHardDiskInfo();
}
}
```

2) Inner Class

- Inner class is also known a nested class.
- If one class is decleare inside the athother class it is know as.
- Inner class have its own state and behaviour.
- It possible to create multiple inner classes

- Inner class can be static or not static.
- To access the property of inner class we have to provide
- We can acheive encapsulation by using inner class.

DATE 29 04 2022

Q. What is Design Pattern?

Design Pattern:

- Design Pattern is a logical solution for commonly occured problems.
- Every Programming language is having predefined design patterns.
- In Java language we can use following design patterns:
  1) Singleton design Pattern
  2) Factory design Pattern
  3) MVC design pattern (Model View Controller)
  4) Observe design pattern
  5) Abstract factory pattern.

1) Singleton Design Pattern

- The process of creating only one object throughout the application life cycle is known as singleton design pattern.
- In order to create singleton object we have to declare static method.
- The class which contains singleton object is known as a singleton class.
- In case of singleton design pattern duplicate objectes are not allow.
- Singleton design pattern can be use to achieve encapsulation.

• Program No. 1

```
public class GoogleAccount
{
    Static int count = 0;
    Static GoogleAccount g1;
    private GoogleAccount ()
    {
        Count ++;
    }
```

DATE

```
//Singlton Method
static googleAccount createAccount ():
{
  if (count ==0)
  {
    g1 = new GoogleAccount ();
    System.out.println ("ACCOUNT Created");
  }
  else
  {
    System.out.println ("Account Already Created");
  }
  return g1;
}
void accessGmail ()
{
  System.out.println ("Acess Gmail");
}
void access Drive ()
{
  System.out.println ("Access Drive");
}

class MainApp5
{
public static void main (String [] args)
{
  GoogleAccount acc1 = GoogleAccount.createAccount();
  acc1.accessDrive ();
  GoogleAccount acc2 = GoogleAccount.createAccount();
  acc1.accessGmail ();
  System.out.println (acc1);
  System.out.println (acc2);
}
}
```

DATE

Q. What is Abstraction?

- Abstraction:

- Abstraction is one of the OOPS principle.
- The process of hiding implimentation of an object from its utilization layer is known as
- In other words abstraction means hide object functionality and provide user interface to access the features.
- To achieve abstraction we have to follow three steps:
  1) Generalise all the properties of an object into an interface.
  2) Provide implementation for object features by using classes.
  3) Access object properties by using referance of interface through utilization layer.
- The advantage of abstraction is loose coupling.
- After abstraction utilization layer should independant of implementation layer which means we can modify one layer without affecting other layer

Business logic — Presentation logic

Implementation layer ⟷ Utilization layer → HTML Web
→ Mainclass (Desktop)

- Program No.1

```
Package Abstraction;
public ~~class~~ interface Switch   // Step 1 Generalize properties of an object into an interface
{
    void switchon ();
    void switchoff ();
}
```

DATE

```
// implementation layer
// STEP-2 provide implementation for object features by
  using class
public class Ledlight implements Switch
{ @override
  public void switchOn()
  {
    System.out.println("Led Light switch On");
  }
  @override
  public void switchOff()
  {
    System.out.println("Led Light Switch Off");
  }
}

public class SolarLight implements Switch
{ @override
  public void switchOn()
  {
    System.out.println("Solar Light switch On");
  }
  @override
  public void switchOff()
  {
    System.out.println("Solar Light Switch Off");
  }
}

// Utilization layer
public class MainApp1
{
  public static void main(String[] args)
  {
    switch s1;
```

```
// loose Coupling
S1 = new LedLight ();  //upcasting
S1.switchOn();
S1.switchOff();

S1 = new SolarLight ();  //upcasting
S1.switchON();
S1.switchoff();
```

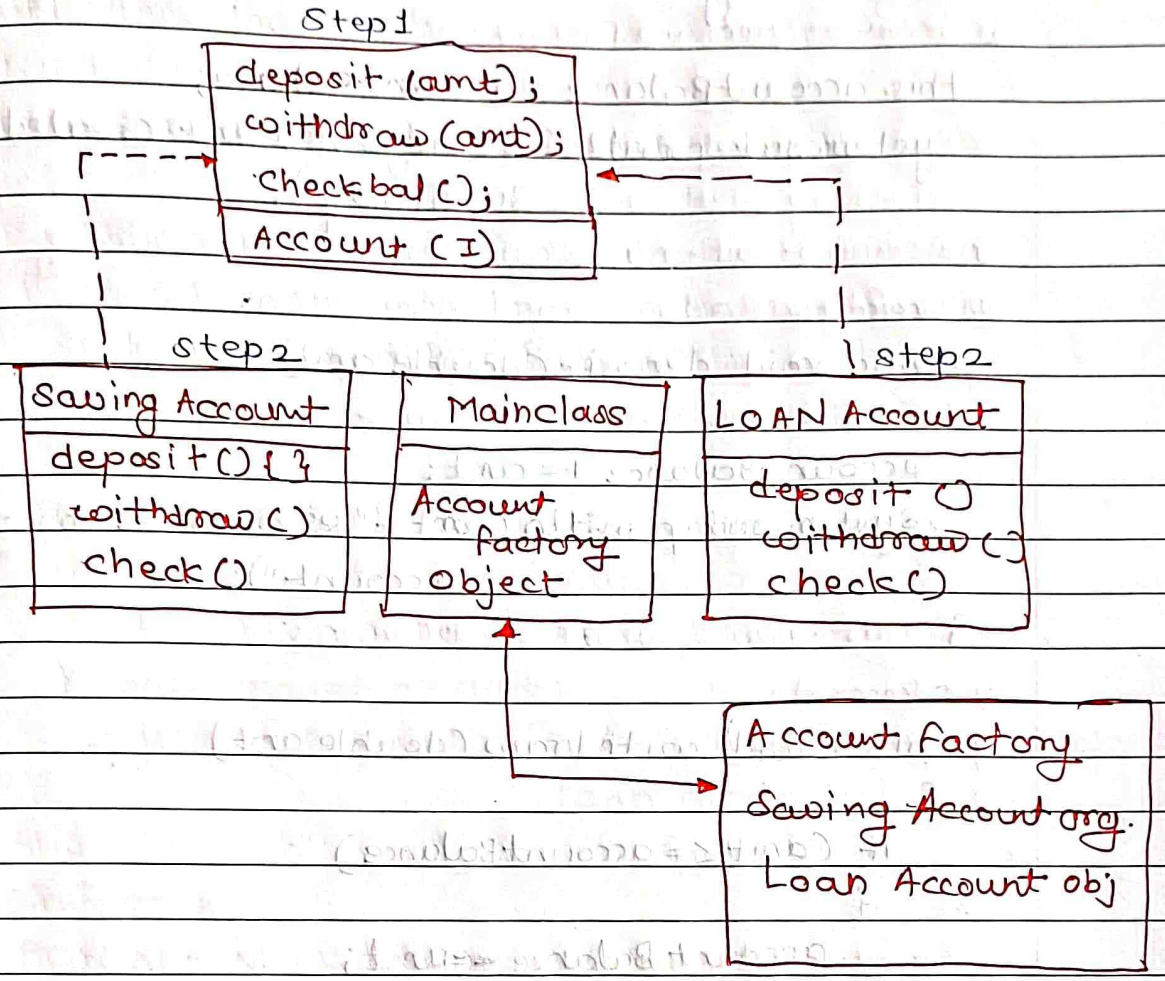
What is Factory Design Pattern?

- The process of creating objects of implementation classes without exposing it to the utilization layer is known as factory
- The method which contains object of implementation classes is known as factory method or helper method.
- The class which contains factory method is known as a factory class or helper class
- By using factory design pattern we can acheive advanced abstraction.
- Structure:

Implementation → Factory Class → Utilization

Business logic class	create objects of implet^n classes	Presentation logic factory class object

Q.




```
Package abstraction;
//STEP-1
public interface Account
{
    void deposit (double amt);
    void withdraw (double amt);
    void checkBalance ();
}

STEP-2
Public class SavingAcount implements Account
{
    double accountBalance;
```

DATE

```
public SavingsAccount (double accountBalance)
{
  this.accountBalance = accountBalance;
  System.out.println("Savings Account created
                        sucessfully");
}
@override
public void deposit (double amt)
{
  accountBalance += amt;
  System.out.println (amt + "Rs credited to your
                        account");
}
@override
public void withdraw (double amt)
{
  if (amt <= accountBalance)
  {
    accountBalance -= amt;
    System.out.println (amt + " Rs debited from
                        your account");
  }
  else
  {
    system.out.println ("Insufficient Account
                        balance");
  }
}
@override
public void checkBalance()
{
  system.out.println ("Active account Balance
                        :" + accountBalance);
}
}
```

DATE

```
publi class loanAccount implements Account
{
  double loanAmount;

  public class LoanAccount (double LoanAccount)
  {
    this. loanAmount = LoanAmount;
    System.out.println ("Loan Account created
                        sucessfully");
  }
  @ override
  public void deposit (double amt)
  {
    Loan Amount -= amt;
    System.out.println (amt+" RS deducted from your
                        loan account");
  }
@ override
  public void withdraw (double amt)
  {
    Loan Amount += amt;
    System.out.println (amt+" RS creadited to your
                        loan account");
  }
@ override
  public ~~the~~ void checkBalance ()
  {
    System.out.println ("Active loan Amt:" +LoanAmount);
  }
}
```

DATE

```
// Factory class
public class AccountFactory
{
    Account createAccount (int type, double amt)
    { //Factory Method
      Account a1 = null;
      if (type == 1)
      {
        a1 = new SavingsAccount (amt);
      }
      else if (type == 2)
      {
        a1 = new LoanAccount (amt);
      }
      return a1;
    }
}

import java.util.Scanner;
public class MainApp2
{
  public static void main (String [] args)
  {
    AccountFactory factory = new AccountFactory();
    Scanner sc1 = new Scanner (System.in);
    System.out.println ("select Account Type");
    System.out.println ("1: SAVINGS \t\t 2: LOAN");
    int type = sc1.nextInt ();
    System.out.println ("Enter Amount");
    double amt = sc1.nextDouble ();
    Account acc = factory.createAccount (type, amt);

    boolean status = true;
    while (status)
    {
```

DATE

```
System.out.println("Select Mode of Operation");
System.out.println("1: Deposit \t 2: Withdraw \t 3: Check
                    Balance \t 4: Exit")
int choice = sc1.nextInt();
if (choice == 1)
{
  System.out.println("Enter Amount to be deposited");
  double depositAmt = sc1.nextDouble();
  acc.deposit(depositAmt);
}
else if (choice == 2)
{
  System.out.println("Enter Amount to be withdrawn");
  double withdrawAmt = sc1.nextDouble();
  acc.withdraw(withdrawAmt);
}
else if (choice == 3)
{
  acc.checkBalance();
}
else
{
  status = false;
}
}
}
}
```

DATE

DATE

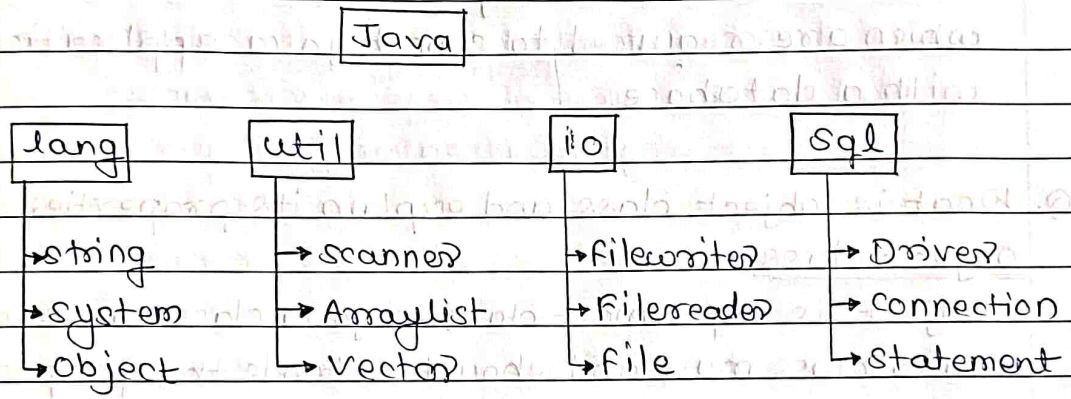
DATE

classmate

PAGE

What is JAVA API or JAVA liabraries.

- Java API or liabraries are collection of multiple classes
- In order to develop Java Applications we have to use some inbuild classes and interfaces.
- In JAVA liabraries multiple packages are available.
- Every package contains several ~~packageces~~ classes and interfaces
- Following are the core packages of JAVA programming languages.



1) lang Package:

- This is the basic package of JAVA liabraries.
- It contains classes and interfaces which are required to develop simple java application.
- We can use any class or interface from this package without providing input statement.

2) util Package:

- This package contains the utility classes and collection framework classes.
- To use the properties of util package we have to provide input statement

3] IO (Input/output):
- This package acts as a container for a classes and interfaces which are required to perform file opertions.
- We can perform basic file operations like creating file, writing file, reading file etc.

4] SQL Package:
- This package is mainly use to perform database relational operations.
- It contains the multiple classes and interfaces which are required to connect java application with a database.

Q. What is object class and explain its properties
Object class:
- Object is supermost class in java class hierarchy.
- Java class must and should extends the properties of object class.
- Every java class will be having properties of an object class hence java is object oriented programming language.
- Following of the important methods of an object class:
1) toString()
2) hashCode()
3) equals()
4) getClass()
5) clone()
6) finalize()
7) wait(), notify(), notifyAll()

DATE

1] toString():

i] to string method declare inside the object class.
ii] This method can be used for string representation of an object.
iii) It means after overriding too string method we can display customizing output instead of object address.
iv] This method will be automatically executed by JVM if developer is trying to print referance variable.
v] Programmer can override toString method inside the user defined class.

Program No.1

```
package objectclass;
//business logic class
public class Bike extends Object
{
  string company;
  double price;

  public Bike(String company, double price)
  {
      this.company = company;
      this.price = price;
  }
 @override
  public String toString()
  {
    return company+"\t\t"+ price;
  }
}

public class MainApp1
{
  public static void main(String [] args)
  {
```

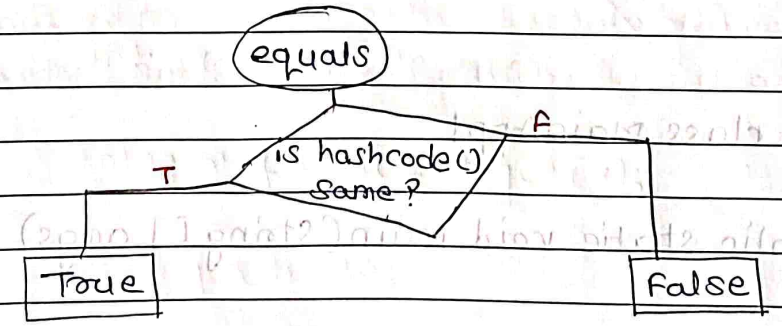
```
    Bike b1 = new Bike (company:"HONDA", price:45000.25
    System.out.println (b1);
    Bike b2 = new Bike (company:"Bajaj", price:65000.25
    System.out.println (b2);
  }
}
```

2] **hashCode ():**

- hashCode method declare inside the object class
- This is a special purpose method which always writtens hashcode of an object.
- hashCode is nothing but an int value generated by JVM at the time of object creation.
- Every object in java is having unique hashCode.
- If two objets are having same content then there adderess will be different but hashCode will be remain same.
- We can use hashCode for unique identification of an object.

3] **equals ():**

- equals method is also declare inside the object class
- we can use equals method for comparison of two object.
- Equals methods internally checks the hashCode of an object.
- If two objects are having same hashCode then output will be true else will be falls.



DATE

Program No. 2

```
public class HashCodeDemo
{
  public static void main (String [] args)
  {
    String s1 = new String (original: "JVM");
    String s2 = new String (original: "JRE");
    String s3 = new String (original: "JVM");

    System.out.println (s1 == s2); // false
    System.out.println (s1 == s3); // false

    System.out.println (s1.hashCode ()); //73857
    System.out.println (s2.hashCode()); //78725
    System.out.println (s3.hashCode ()); //73857

    System.out.println (s1.hashCode () == s2.hashCode()); F
    System.out.println (s1.hashCode() == s3.hashCode()); T
  }
}
```

4] getClass ():

- getClass method is declare inside the object class.
- This method can be use to display fully qualified class name of current object.
- The class name which is declared along with package names is known as fully qualified classes.

DATE

Q. What is Wrapper Class?

- **Wrapper Class:**
  - Wrapper calasses are also known as non-primitive datatype.
  - All wrappler classes are decleared inside the java.lang package.
  - Every primitive datatype is having seperate wrapper class. It means 8 wrapper classes are provided by Java language.
  - We can us wrapper class to represents single/simple information in form of object.
  - following are the list

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

  - All wrapper classes extends the properties of an object class.

Q. Explain how to store information into wrapper class

→ - In order to store information we have to create an object of respective wrapper class.
- Syntax:

① int x = 20; — Primitive

Integer y = new Integer (x); ← Non-Primitive

ii Integer b = 30;

Q. What is differance between Autoboxing and Autounboxing

- The process of converting primitive information into non-primitive format is known as Autoboxing
- In other words Autoboxing is the process of representing simple value in form of object

e.g.

int a = 10 — Primitive

Integer ref = a; — Autoboxing

Autounboxing:

- Autounboxing is the process converting non-primitive information into primitive format.
- In other words autounboxing means representing object information as a simple value.

e.g.

Integer i = new Integer (20); — Non-Primitive

int a = i — Autounboxing

Q. What the use of Parsing technique?

- Parsing:

- Parsing technique is specifically designed for the convertion of string value
- We can pass string value into primitive datatype with the help of following methods:

1) Integer.parseInt()
2) Float.parseFloat()
3) Double.parseDouble()
4) Long.parseLong()
5) Short.parseShort()
6) Boolean.parseBoolean()
7) Byte.parseByte()

- To convert any primitive value into string format we have to use toString() method.

DATE

e.g

1) Integer.toString (int a)
2) Double.toString (double a)
3) Float.toString (float a)
4) Char.toString (char b)
5) Long.toString (Long a)
6) Short.toString (short b)
7) Boolean.toString (boolean c)
8) Byte.toString (byte b)

- Program No. 1

```
public class ParsingDemo
{
  public static void main (Args [] String)
  {
    String s1 = "50";
    String s2 = "30";
    System.out.println (s1+s2); —— 5030
    //String to Integer
    int a = Integer.parseInt (s1);
    int b = Integer.parseInt (s2);
    System.out.println (a+b); —— 80

    double d1 = 38.53;
    double d2 = 14.48;
    System.out.println (d1+d2); —— 50.63
    //double to String
    String str1 = Double.toString (d1);
    String str2 = Double.toString (d2);
    System.out.println (str1+str2); —— 38.5314.48
  }
}
```

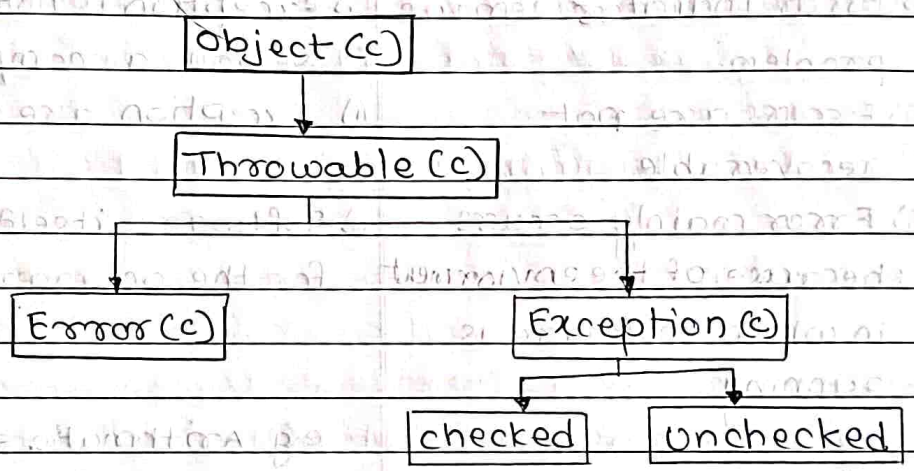
DATE

Q. What is need of exception handling?

- **Need of exception handling:**

- Since Java is sychronous programming language execution of next statement completely depends on previous statement.
- It means if there are some issues with first line then remaining lines of codes will not be executed.
- With the help of exception handling we can fix the issues present inside the java program.

- **Exception hierarchy of JAVA language:**



- In Java language throwable is the super class ino exception handling mechanism.
- Throwable class is having 2 subclasses
  1) Error 2) Exception
- Expection class is also having 2 subclasses.
  1) Checked Exception
  2) Unchecked Exception

- **Error:**

- Error indicates a series problem caused due to lack of system resourses.

eg. stackoverflow error, out of memory Error

- **Exception:**
  - Exception is an abnormal situation or unwanted event ocurs during the execution program and disturbs normal flow of execution.
  - There are two type of exceptions:
    1) Compile time exception. (check exception)
    2) Run time exception (uncheck exception)

Q. State differance between Error and Exception.

Error	Exception
i) Error indicates serious problem.	i) Exception is not that must serious as compare to error
ii) Errors are not recoverable	ii) Exception are recoverable.
iii) Error mainly occurs because of the enviorment in which software is running.	iii) Software itself responsible for the occurance of exceptions
e.g. Out of memory error, Runtime exception	e.g. Arithmatic exception, Numberformat exception

Q What is exception handling?

- **Exception handling:**
  - Exception handling is mechanism provided by Java language to handle different types of exceptions.
  - This mechanisms provides different blocks and key words which can be used to handle exceptions.
  - Following are the important components of exception handling mechanism:
    1) Try block
    2) catch block

3) Finally block
4) throw keyword
5) throws keyword

1] **try block:**
- try block acts as a container for abnormal java statements.
- In other words the code which may cause an exception must be declare inside the try block.

2] **catch block:**
- catch block must be declare along with try block.
- catch block is responsible to handle the exception.

Q. What is differance between checked and unchecked exception?

Checked Exception	Unchecked Exception
1) Checked exception also known as compile time exception.	1) Unchecked exception also known as run time exception.
2) compiler provides indication to developer if exception is going to occur.	2) JVM directly throws the exception without any warning.
3) If java application is connected with external resources or application then compiler will provide indication to the developer about exception.	3) In case of unchecked exception external resources are not involved.
4) It is mandatory to handle checked exception.	4) It is not mandatory to handle unchecked exception.
5) Checked exceptions are having highest priority. e.g. SQL exception, FileNot found exception	5) Unchecked exceptions are having lowest priority. e.g. NumberFormat Exception, Arithmatic Exception.

DATE

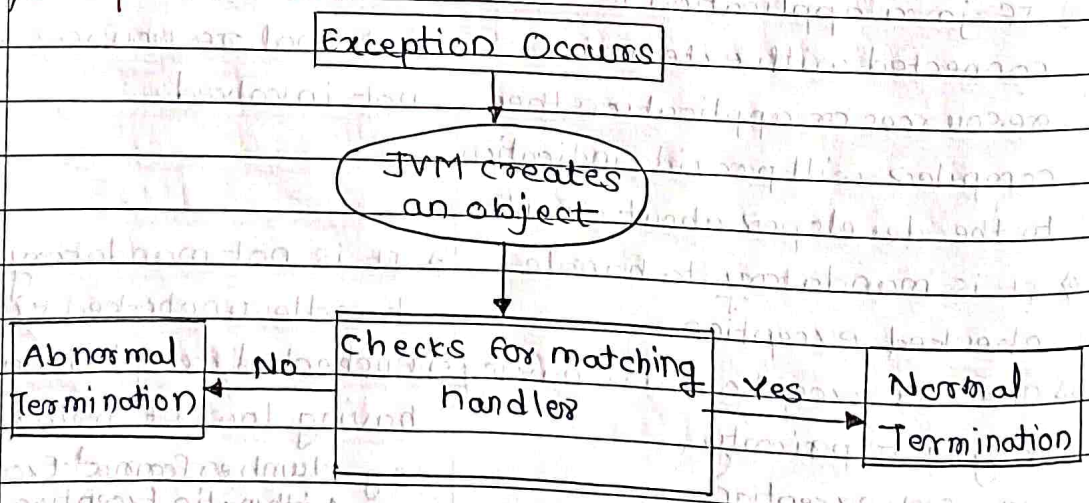
- Program No.1

```
public class ExceptionDemo1
{
  public static void main (String [] args)
  {
    System.out.println ("program Started");
    String Str = "15abc";          //error
  try {
      int a = Integer.parseInt (Str); //new NumberFormatEx
      System.out.println (a);
    }
  catch (NumberFormatException p)
    {
      System.out.println (p)
    }
  System.out.println ("Program Ended")
  }
}
```

O/p → Program Started
NumberFormatException ( )
program ended.

Q. Explain internal working of try and catch block?

→ 1) Exception Flow:



DATE

- If exception occurs in java program then jvm is responsible to create an object of respective exception class.
- Try block will throw exception object towards catch block.
- catch block is responsible to handle exception.
- In order to handle the exception we have to provide matching handler inside catch block.
- If catch block is having matching handler then exception will be handled and normal termination takes place.
- If catch block is not having matching handler then abnormal termination takes place.

Q. What is NumberFormatException?

- It is a class declared inside java.lang package NumberFormatException is a type of unchecked or run time exception.
- If developer is trying to convert genuine string value into number format (int, float, double etc.) then JVM will throw NumberFormatException.

• Program No. 2

Write a program to handle an exception by using generic handler.

```
package exceptionhandling;

public class ExceptionDemo2
{
  public static void main (String [] args)
  {
    String str = null;

    try {                                    //object created internally
      System.out.println (Str.toLowerCase ());//new nullpointerException
    }
```

classmate PAGE

DATE

```
catch (Exception e)          // General handler
{
  // Exception e = new NullPointerException ----- upcasting
  System.out.println(e)
}
System.out.println("program ended");
}
}
```

Q. What is NullPointerException?

- It is class declared inside java.lang package.
- NullpointerException is a type of unchecked exception.
- If developer is trying to perform operations on null value then JVM will threw NullPointerException.

Q. How to use multiple catch block along with single try block?

- We can use multiple catch blocks in order to handle multiple exceptions.
- One try block can have multiple catch blocks.
- At time of declaration multiple catch blocks subclass catch block must be declared before superclass catch block.
- Following is a syntax to use multiple catch block:

```
try
{
}
catch (Specific handler)
{
}
catch (Generic handler)
{
}
```

DATE 09 05 2022

- Program No.3

```
import java.util.Scanner;
public class ExceptionDemo3
{
 public static void main (String [] args)
 {
  Scanner sc1 = new scanner (System.in);
  System.out.println ("Enter first number");
 // user will enter alphabates, 0 etc so;
  try
  {
   int no1 = sc1.nextInt ();
   System.out.println ("Enter Second Number");
   int no2 = sc1.nextInt ();
   int result = no1 / no2;
   System.out.println (result);
  }
  catch (ArithmeticException e)
  {
   system.out.println (e);
  }
  catch (InputmismatchException e)
  {
   System.out.println (e);
  }
 }
}
```

Q] What is InputMismatch Exception?

- It is type of unchecked exception decleared inside java.util package.
- This exception mainly occurs at time of accepting input from end user.
- If you are using scanner class then we have to handle input mismatchException.

Q. Explain Arithmetic Exception?

- It is type of unchecked exception declared inside java.lang package.
- If developer is trying to perform invalid arithmetic operations then JVM will throw this exception.

Q. Why to use nested try catch block?

- If one try catch block is declared inside another try catch block then it is known as nested try catch block.
- We can use nested try catch blocks to handle multiple exceptions in single java class.
- It is possible to declare multiple nested try catch blocks inside single outer try catch block.

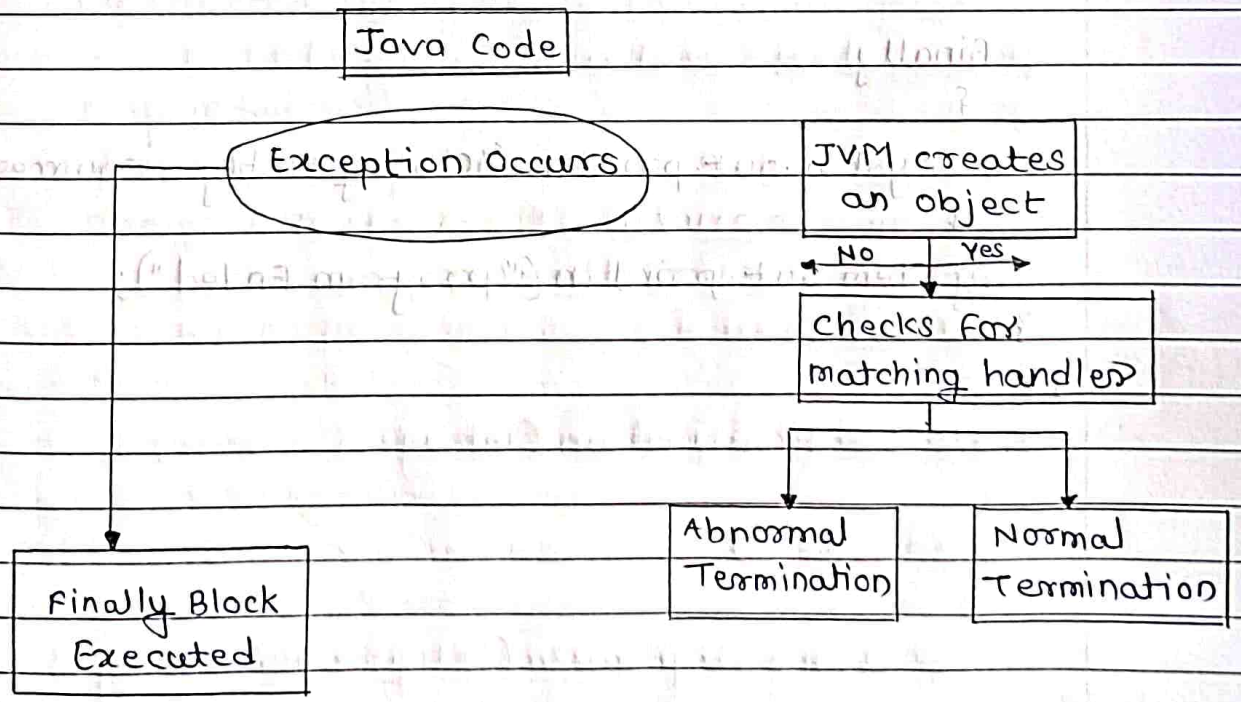
• program No.4

```
public class ExceptionDemo4
{
  public static void main (String [] args)
  {
    String s = null;
    int [] arr = new int [3]
    // outer try catch block
    try {
    // nested try catch block
      try{
          System.out.println(s.length());
        }
      catch (NullPointerException e)
        {
          System.out.println (e);
        }
    arr[3] = 40;
    System.out.println (arr[3]);
  }
```

```
catch (ArrayIndexOutOfBoundsException e)
{
  System.out.println(e);
 }
}
}
```

Q. What is use of finally block?

- This block is one of the exception handling component.
- This block is mainly used to close costly resources.
- The resourses which are involved in interapplication communication are known as costly resources.
- Finally block will be executed irrespective of exception scenarios.
- This block must be declared after try or catch block.
- It is not possible to declare multiple finally blocks along with try or catch block.
- Following is the execution flow of java program if it contains finally block.



DATE

- program No.5

```
package Exceptionhandling;
public class ExceptionDemo6
{
 public static void main (String [] args)
 {
  System.out.println (" program started");
  String str = "Java";

 try
 {
   System.out.println (" Try Started");
   System.out.println (str.length ());
   // System.exit (0);
   System.out.println (" Try ended");
 }
 // catch (Arithmetic Exception e)
  {
   System.out.println (e);
  }

 Finally
  {
   System.out.println (" Closing costly resources");
  }
  System.out.println (" program Ended ");
 }
}
```

DATE

case 1:

Exception occured as well as handled.

try started
NullpointerException
closing costly resources
program ended.

Case 2:

Exception occured but didn't handled.

Program started
try started
closing costly resources
java.lang.NullPointerException

case3:

Exception did not occured.

program started
try started
4
try ended
closing costly resources
program ended.

Q. What use of throw keyword?

- Throw Keyword:
  - Throw keyword is one of the exception handling components.
  - Throw keyword is mainly used to explicitly throw an exception.
  - In other words by using throw keyword we can create an abnormal situation inside the Java program.
  - Following is the syntax:

```
throw objectofexception class;
```

  - We can not use multiple throw keywords inside same block or method.

- Program No. 6

```
import java.util.Scanner;
public class ExceptionDemo1
{
  public static void main (String [] args)
  {
    Scanner sc1 = new scanner (System.in);
    System.out.println ("Enter age");
    int age = sc1.nextInt();
    validation (age);
  }
static void validation (int age)
  {
    if (age >= 18)
    {
      System.out.println ("Eligible for voting");
    }
```

```
else
  {
    //Explicit exception object
    throw new IllegalArgumentException("Age Invalid");
  }
}
}
```

we can provide any exception

Message gets displayed

Q. What is IllegalArgumentException?

- It is a type of unchecked exception decleared inside java.lang package.
- Developer can create an object of this exception class by using throw keyword.
- If end user has provided illegal values for input parameters then we can throw this exception.

Q. What is custom Exception?

- Custom exception is also known as user Defined exception.
- Developers can create their own exception classes in order to handle custom exception scenarios.
- To create custom exception we have to use following syntax:

- For creating unchecked exception:

```
class Demo extends RunTimeException
  {

  }
```

- For creating checked exception:

```
class Demo extends exception
  {

  }
```

DATE

- Program No. 7

```
Package Exceptionhandling;

//Subclass custom unchecked exception
public class InvalidEmailException extends RuntimeExcep
{
  public InvalidEmailException(String message)
  {
    Super (message);
  }
}

Package Exceptionhandling;
import java.util.scanner;
public class ExceptionDemo2
{
  public static void main (String [] args)
  {
    Scanner sc = new scanner (System.in);
    System.out.println ("Enter Email ID");
    String email = sc.next();
    validation (email);
  }

static void validation (String email)
  {
   if (email.cantains ("@") && email.contains (".."))
   {
    system.out.println ("Email id is valid");
   }
   else
   {
    //throwing custom exception
```

```
throw new InvalidEmailException("Enter valid email ID");
}
}
}
```

Q. Write a program to create a text file through java application.

```
public class ExceptionDemo3
{
  public ~~class~~ static void main(String[] args)
  {
    String str = "Java is object oriented language";
    FileWriter fw = null;  → Default

    try
    {
      fw = new FileWriter("E:\\ICCA II data\\message.txt")   location
Default {
      fw.write(str);
      fw.close();
      System.out.println("File created sucessfully");
    }
    catch (~~File~~ IOException e)
    {

      System.out.println(e);
    }
  }
}
```

Q. What is IOException?

IO Exception:

- This exception is a type of checked exception decleared inside java.IO package.
- This exception mainly occurs at the time of file handling operations.

e.g.

creating file, reading data from existing file and updating existing file.

- since it is type of checked exception it is mandatory to handle IOException by using try catch block.
- If you are not handling this exception then compile will throw syntax error.

- Exception Propagation:
  - An exception thrown from top of the stack will dropdown to call stack this process is know as exception propagation.
  - In other word exception propagation means exception object is moving from one method to another method.
  - This process will be continued unless and until reaches to main method or else getting handle by developer.
  - Unchecked exception are automatatically propagated by JVM whereas developer is responsible to propagate checked exception.

DATE

```
public class ExceptionDemo1
{
  static void test()
  {
    System.out.println("Test Started");
    int c = 10/0;
    System.out.println("Test Ended");
  }
  static void display()
  {
    System.out.println("Display started");
    test();
    System.out.println("Display ended");
  }
  static void info()
  {
    System.out.println("Info Started");
      try {
          display();
        }
      catch (ArithmaticException e)
      {
        System.out.println(e);
      }
    System.out.println("Info Ended");
  }
  public static void main(String[] args)
  {
    System.out.println("Main Started");
    info();
    System.out.println("Main Ended");
  }
}
```

O/P → main Started
info
display
Main
Arithmatic exception — handle
info ended
main ended

Stack (top)
test ()
display ()
info ()
main ()
(bottom)

handled exception

- **Throws Keyword:**
  - Throws Keyword is one of the exception handling component.
  - Throws keyword is mainly use to declare an exceptio
  - We can use throws keywords for propagation of checked exception.
  - It is possible to declare multiple exceptions using throws keyword.

- **Program No.8**

```
import java.io.IOException;
public class ExceptionDemo1
{
    static void test() throws IOException
    {
        throw new IOException();
    }
```

DATE

```
static void display () throws IO Exception
{
  system.out.println ("Display started");
  test ();
  System.out.println (" Display Ended");
}
public static void main (String [] args) throws IO Exception
{
  system.out.println ("main started");
  display ();
  System.out.println ("Main Ended");
}
}
```

Explain differance between throws and throw keywords?

Throw	Throws
1) Throw keyword can be use to create an exception object	1) Throws keywords can be use to decleare an exception
2) Along with throw keyword we have to provide object of exception class.	2) Along with throws keyword we have to provide exception class name.
3) It is not possible to throw multiple exceptions by using single throw keyword.	3) It is possible to decleare multiple exception by using throws keywords.
4) Throw keyword must be decleared inside method or block	4) We have use throws keyword along with method signiture
5) By using throw keyword we can propagate only unchecked exceptions	5) By using throws keyword we can propagate checked exception.

DATE 12 05 2022

Q. What is differance between collection, collections and collection framework?

- Collection is an interface decleared inside the java.util package.
- Collections is a class decleared inside the java.util package.
- Collection framework is a combination of classes and interfaces.

Q. Explain the drawbacks of array.

- By using array variable we can store only same type of data that is homogeneous information.
- Array variable is a fixed width or static width it means run time we can not change capacity of array.
- Array manupulation is time consuming process

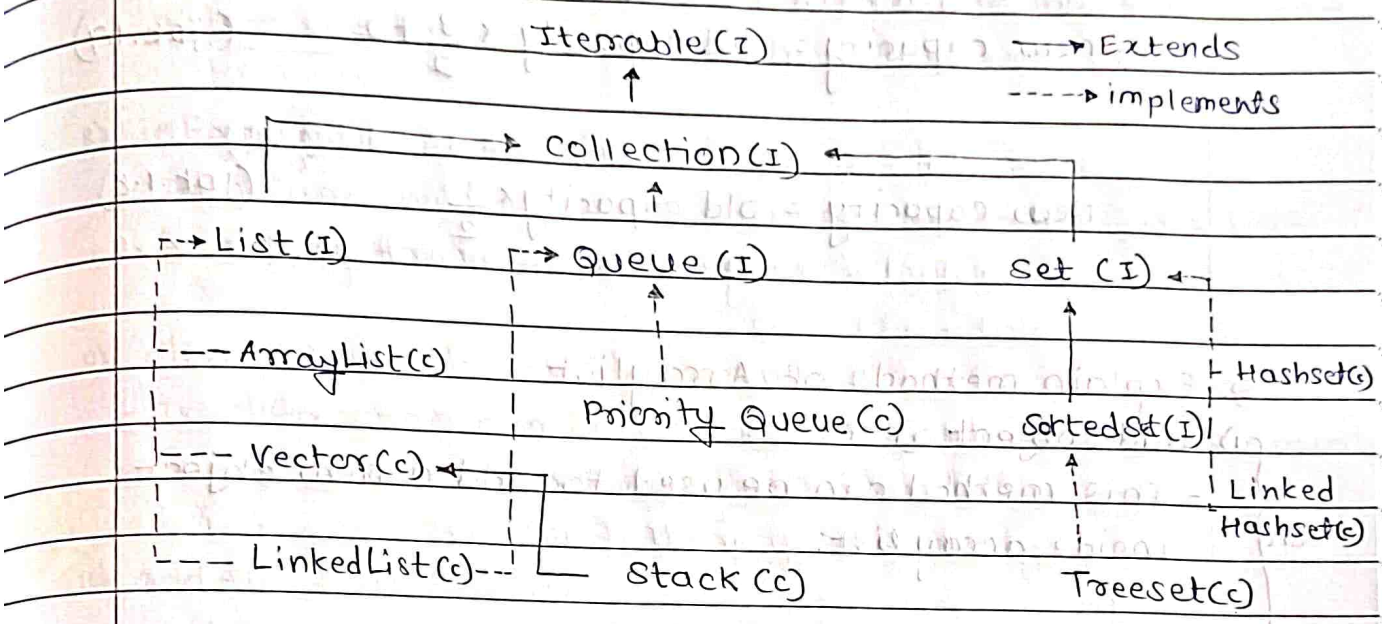
Q. What is Collection?

- An entity which represents group of multiple objects is known as collection.
- Collection can store different types of element as a single entity.
- Collection framework helps storing and processing data efficiently.

Q. What are the advantages/benifits of collection framework.

- Collection framework is resizeable in nature that means dynamic structure.
- It increases code reuseability.
- It reduces development effort.
- We can easily implement data structure features with rediment classes and interfaces.
- It reduces the efforts of code maintenance.

Q. Explain hierarchy of collection framework?



Q. Explain properties of List interface

- List interface is decleared inside the java.util package
- List interface can be use to store data in seq. manner.
- We can store null and duplicate value inside the list
- Following are the implementation classes of list interface.
  1) ArrayList
  2) Vector
  3) LinkedList

1) ArrayList Class:

- Arraylist is an implementation class of list interface.
- Arraylist class is decleare inside the java.util package
- Arraylist internally uses dynamic array datastructure.
- Since it is an implementation class of list interface data will be store in a sequenceal manner.
- Arraylist allows to store null as well as duplicate
- Default capacity of arraylist is 10 elements and it will grow at runtime with the help of following

DATE

Formula:

$$\text{new capacity} = \text{old capacity} \times \frac{3}{2} + 2 \quad \text{—— (JDK.1.7)}$$

$$\text{New capacity} = \text{old capacity} \times \frac{3}{2} \quad \text{—— (JDK.1.8)}$$

Q. Explain methods of Arraylist.

Insert

1) add (object o)
- This method can be used to add a new object inside arraylist.

2) add (int index, object o)
- This method can be use add new object at perticular index number.

Delete

3) remove (object o)
- This method can be use to delete object from arraylist.

4) remove (int index)
- This method can be used to remove an object from the specified index number.

Update

5) Set (int index, Object o)
- This method can be used to modify existing object available inside the arraylist.

Select

6) get (int index)
- This method can be use to fetch the data from arraylist based on specific index number.

nderOf (object o)

This function can be use to display indexOf specified object.

isEmpty()

- This function can be use to check weather the list is empty or not.

clear()

- This function can be use to delete all elements from the list without affecting its structure.

) addAll (collection c)

- This function can be use to add all the elements from one ~~analyst~~ arraylist into another ~~analyst~~ arraylist.

```
import java.util.ArrayList →//Non-Generic Arraylist
public class ArrayListDemo
{
  public static void main (String [] args)
  {
    Arraylist data = new ArrayList();
    data.add (25);
    data.add ("JAVA");
    data.add ('G');
    data.add (25.36);
    data.add (true);
    data.add (25);
    data.add (null);

    System.out.println (data);
  }
}
```

DATE

- Program No. 2

```
import java.util.ArrayList;
public class ArrayListDemo2
{
  public static void main (args [] string)
  {
    ArrayList <Integer> data = new ArrayList<>();
     data.add (10)
     data.add (20)
     data.add (30)

    System.out.println (data);
  }
}
```

Q. What is differance between Generic and Non-Generic ArrayList?

- Generic Arraylist allows to store only same type of information whereas Non-Generic arraylist allows to store same type or different type of information.
- While creating Generic arraylist we have to specify type of information whereas non-generic arraylist does not requires any type of information.

- Program No. 3

```
//Generic ArrayList
import java.util.ArrayList;
public class ArrayListDemo3
{
  public static void main (String [] args)
  {
    ArrayList <String> cities = new Arraylist <> ();
```

DATE

```
cities.add ("PUNE");
cities.add ("Mumbai");
cities.add ("Delhi");
//Retrieving data from arrayList
//option 1
System.out.println (cities)
// option2 for loop
for (int a=0; a< cities.size(); a++) {
    System.out.println (cities.get (a)); }
// option3 Enhanced for loop
for (String s: cities)
 {
    System.out.println (s);
 }
}
}
```

- Program No. 4

```
import java.util.ArrayList;
public class ArrayListDemo4
{
public static void main (String[] args)
 {
  ArrayList<Double> percentage = new ArrayList<>();
    percentage.add (75.83);
    percentage.add (60.84);
    percentage.add (87.39);

  //Option 4 Iterator interface
    Iterator<Double> itr = percentage.iterator();
    while (itr.hasNext())
    {
      if (itr.next ()==75.63)
```

```
        {
          itr.remove();
          System.out.println("Removed sucessfully");
        }
      }
      System.out.println(percentage);
    }
  }
```

Q. What is the differance between for loop and iterator interface.?

- for loop as well as iterator interface are used for list traversing.
- If we are using for loop then at a time we can perform one operation which means concurrent modifications not allow.
- If we are using iterator interface then concurrent modifications are possible.

Q. What is concurrent Modification Exception.?

- It is a type of unchecked exception declared inside the java.lang package.
- If developer is trying to perform concurrent operations by using for loop then JVM will throw ConcurrentModificationException

Program No.5

```
import
public class ArrayListDemo5
{
  public static void main (String [] args)
  {
    ArrayList <character> data = new ArrayList<>();
```

DATE

```
data.add('H')
data.add('K')
data.add('F')
data.add(index:1, element:'T');
data.add('E')
data.set(3,'P')
data.remove(index:2);
data.add('W')

System.out.println(data.get(3));
system.out.println("=========");
System.out.println(data);
System.out.println(data.size());
}
}
```

① H

② H K

③ H K F

④ H T K F

⑤ H T K F E

⑥ H T K P E

⑦ H T P E

⑧ H T P E W

DATE 14 05 2022

* Sorting data.

• Program No. 6

```
import java.util.ArrayList;
import java.util.collections;
public class Demo6 {
  public static void main(String [] args)
  {
      ArrayList <Integer> data = new ArrayList <>();
      data.add (12);
      data.add (5);
      data.add (36);
      data.add (25);
      data.add (65);

      System.out.println (data)
      // Sort arraylist in asending order
      Collections.sort (data);
      System.out.println (data);
      // Sort ArrayList in desending order
      collections.sort (data, Collections.reverseOrder());
      System.out.println (data);
  }
}
```

* Merging Operation

(P-7)

```
import.java.util.ArrayList;
public class ArrayListDemo7
{
  public static void main (String [] args)
  {
      Arraylist <String> data1 = new ArrayList <>();
      data1.add ("REACT");
      data1.add ("ANGULER");
      data1.add ("VUE");
```

```
Arraylist <String> data2= new Arraylist <>();
data2.add ("SPRING")
data2.add ("HIBERNATE");
data2.add ("SPRING BOOT");
// Add data1 into data2
data2.addAll(data1)
System.out.println (data2)
}
}
```

- Program No.8

```
// Business Logic Class
public class Product
{
  int productId;
  String productName;
  double productPrice;

  public Product(int productId, String productName,
                 double product price)
  {
    this.productId= productId;
    this.product Name= productName;
    this.product Price= productPrice;
  }

  public string toString()
  {
    return productId + "\t" + productName + "\t"
           + productPrice;
  }
}
```

DATE

```
// Main Class or Executable class
public class ArrayListDemo8
{
  public static void main(String[] args)
  {
    Product p1 = new Product (101, "Mobile", 15000);
    Product p2 = new Product (102, "Laptop", 50000.5);
    Product p3 = new Product (103, "Projector", 65000);
    // list of custom objects
    ArrayList<~~String~~ Product> data = new ArrayList<> ();
    data.add (p1);
    data.add (p2);
    data.add (p3);
    // Traverse ArrayList
    for (Product p:data)
    {
      //object ———— For Non-generic program
      System.out.println (p);
    }
  }
}
```

* DAO/DTO pattern:

DTO class Book.java	DAO class BookOper.java	mainClass	cmd
bookID bookName bookPrice	add() display() delete()		1. ADD NEW Book (ID, Name, Price) 2. DISPLAY Book 3. DELETE Book 4. Exit

DATE

```
// DTO class
public class Book
{
  int bookId;
  String bookName;
  double bookPrice;

  public Book (int bookId, String bookName, doublebookPrice)
  {
    this.bookId = bookId;
    this.bookName = bookName;
    this.bookPrice = bookPrice;
  }
@override
  public String toString
  {
    return bookId+"\t"+bookName+"\t"+bookPrice;
  }
}

// DAO class
public class BookOperations
{
  // create an arraylist to store book objects
  ArrayList<Book> bookList = new ArrayList<>();

  public void addBook (int id, String name, double price)
  {
    Book b1= new Book (id, name, price);
    bookList.add (b1);
    System.out.println ("Book Sucessfully added");
    System.out.println ("=========");
  }
```

DATE

```
public void displayBooks()
{
    System.out.println("ID\t Name \t PRICE");
    System.out.println("==========");
    for (Book b: bookList)
    {
        System.out.println(b);
        System.out.println("----------");
    }
    System.out.println("==========");
}

public void deleteBook(int bookId)
{
    Iterator<Book> itr = bookList.iterator();
    boolean found = false;
    while (itr.hasNext()).
    {
        Book b = itr.next();
        if (b.bookId == bookId)
        {
            itr.remove();
            System.out.println("Book Deleted");
            found = true;
        }
    }
    if (!found)
    {
        System.out.println("Book Not Found");
    }
    System.out.println("==========");
}
```

DATE

MainClass

```
import java.util.Scanner;
public class ArrayListDemo9.
{
  public static void main(String[] args)
  {
    Scanner sc1 = new Scanner(System.in);
    BookOperations b1 = new BookOperations();
    boolean status = true;
    while (status)
    {
      System.out.println("Select Mode of operation");
                         ("1. Add New Book");
                         ("2. Display All books");
                         ("3. Delete Book");
                         ("4. Exit");
      int choice = sc1.nextInt();

      switch (choice)
      {
        case 1:
        System.out.println("Enter Book Id");
        int id = sc1.nextInt();
        System.out.println("Enter Book Name");
        String name = sc1.next();
                         ("Enter Book Price");
        double price = sc1.nextDouble();
        b1.addBook(id, name, price);
        break;

        case 2:
        b1.displayBooks();
        break;
```

```
case3:
    System.out.println("Enter BookId");
    int bookId = sc1.nextInt();
    b1.deleteBook(bookId);
    break;

default:
    Status: false;
  }
 }
}
}
```

2] vector class:

- Vector is implementation class of List introduce
- Vector class is declared inside the java.util package.
- vector is a legacy class ~~with~~ because it is avai from first version of java.
- vector class internally uses dynamic array to store data.
- vector is similar to arraylist only the differen is that arraylist is a asynchronus and vector is synchronus.
- Default capacity of vector is 10 elements and capacity will be increased by using following formula:

  new capacity = old capacity x 2

DATE

Program 1:

```
import java.util.Vector;
public class Demo1
{
  public static void main (String [] args)
  {
    Vector <Integer> data = new ~~data~~ Vector <>();
    data.addElement (36);
    data.addElement (42);
    data.addelement (45);

    System.out.println (data);
  }
}
```

What is stack:

- Stack is a serveral class of vector which is declared inside the java.util package.
- Stack can be used to process the data in first in ~~last~~ last out.
- Following are the important method:
- This method can be use Av

- By default every new object will be inserted at top of the pyees. push
- This method can be used to remove an object for ~~pop~~ pop of stack.

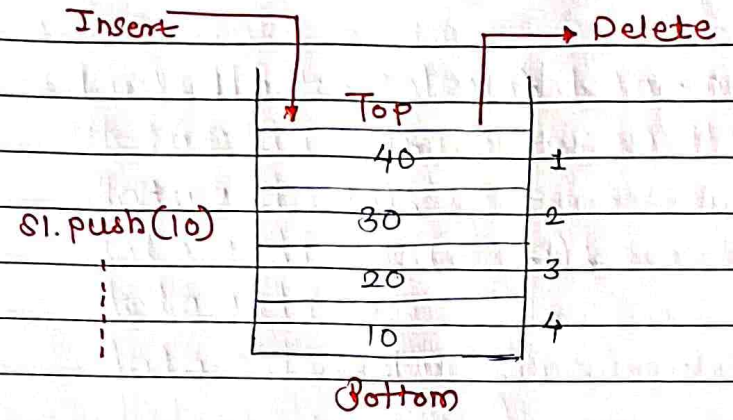
- This method can be used to search the projection of specified object is empty.
- This method will check weather the stack is empty or not and writtens boolean value

DATE

classmate

PAGE

v) peek():
This method always contten elements the from top of stack



- Program No. 2

```
public class StackDemo1
{
  public static void main (String [] args);
  {
    Stack<Double> data = new Stack<>();
    data.push (item: 36.25); // Bottom
    data.push( 12.25);
    data.push( 45.25);// Top

    System.out.println (data);
    System.out.println (data.search (o: 36.25));
    data.pop();
    System.out.println (data.peek());
    System.out.println (data.isEmpty ());
  }
}
```

DATE

a) Linked List:

- Linked list is an implementation class of List interval
- Linked list internally uses double linked list datastructure.
- Linked list store the data is sequenceial manner and

Explain differance between Arraylist & Linked list.

1) Arraylist implements list	1) Linked List implements
2) Arraylist internally uses dynamic array datastructure.	2) Linked list internally use double linked list datastructure.
3) Arraylist stores the data in terms of ind	3) Linked list store the data in terms of nodes.
4) Arraylist having defautt capacity of 10 element.	4) Linked list doesn't have default capacity.
5) Arraylist is recommended for DQL operation.	5) Linked list geaters recommended for DML operations.
6) In arraylist shifting will happen after performing operation.	6) Big shifting will not happen because data is available in nodes.

- Program No. 1

```
public class LinkedListDemo1
{
  public static void main(String [] args)
  {
    LinkedList <Integer> data= new LinkedList<>();
    data.add (25);
    data.add (35);
    data.addFirst (e: 10);
    data.add (55);
    data.removeFirst();

    System.out.println (data);
  }
}
```

O/P → (25, 35, 55)

Explain properties of set interface

Set interface:

- Set interface is type of collection interface
- Set interface store the data in random order.
- Set interface allows to store only unique objects
- Set interface will having three implementation classes:
  1) HashSet
  2) LinkedHashSet
  3) TreeSet

- Set interface internally uses Hash Table data structure to store the data.
- Default capacity of all the implementation classes is 16 element with load factor 0.75. It means after utilization of 75% memory

capacity will be double

new capacity = old capacity × 2

Hash Table

Key	value
10	Present
20	Present
30	Present

HashSet	LinkedHashSet	TreeSet
HashSet is pure implimentation of set interface	LinkedHashSet impliments properties of List and set interface.	Tree set is implimentation of sorted state interface
Hashset store the data in random order	LinkedHashSet stores the data in sequencial manner.	Treeset store the data in sorted manner (ascending order)
Hashset doesnot allow to store duplicates object.	LinkedHashset does not allow to store duplicate values.	Treeset does not allow to store duplicate values.

DATE

- Program no. 1

```
import java.util.Hashset;
public class HashSet
{
  public static void main (String [] args)
  {
    Hashset <String> data = new Hashset <> ();
    data.add ("JAVA");
    data.add ("C");
    data.add ("Python");
    data.add ("JAVASCRIPT");
    data.add ("REACT");...
    data.add ("C");..... (Unique value not allow)

    System.out.println (data);
  }
}
```

O/P

(JAVA, JAVASCRIPT, C, PYTHON, REACT)

DATE

DATE

Business Logic class

```
public class ShoppingCart
{
  HashSet<String> products = new Hashset<>();
  LinkedHashSet<String> cartItems = new
                                LinkedHashSet<>();
  {
    products.add ("AC");
    products.add ("Laptop");
    products.add ("Mobile");
    products.add ("Headphone");
  }

public void displayProducts()
{
  for (String s: products)
  {
    System.out.println(s);
  }
  System.out.println ("==========");
}

public void addToCart (String Name)
{
  if (products.contains(name))
    {
      cartItems.add(name);
      System.out.println("Product Added Sucessfully");
    }
  else
    {
      System.out.println (" product Not found");
    }
  System.out.println ("==========");
}
```

DATE

```
public void displayCartItems()
{
  for (String s: cartItems)
  {
    System.out.println(s);
  }
  system.out.println("==========");
}

public void removeCartItem(String Product)
{
  if (cartItems.contains(product))
  {
    cartItems.remove(product);
    System.out.println("Product Removed");
  }
  else
  {
    System.out.println("First ADD to cart");
  }
  system.out.println("==========");
}
```

Executable class

```
import java.util.Scanner;
public class HashDemo2
{
  public static void main(String[] args)
  {
    Scanner sc1 = new Scanner(System.in);
    Shopping cart s1 = new ShoppingCart();
    boolean status = true;
    while (status)
    {
      system.out.println("1: VIEW Products");
```

```
System.out.println(" 2: ADD to cart");
                   ("3: View Cart Items");
                   ("4: Remove cart items");
                   ("5: Exit)
int choice = sc1.nextInt();
Switch (choice)
 {
  case 1:
     s1.displayProducts();
     break;
  case 2:
     System.out.println("Enter products name");
     String name = sc1.next();
     s1.addToCart(name);
     break;
  case 3:
     s1.displayCartItems();
     break;
  case 4:
     System.out.println("Enter product name");
     String product = sc1.next();
     s1.removeCartItem(product);
     break;
  case 5:
     Status = false;
     break;
  default:
     System.out.println("Invalid Choice");
 }
}
}
}
```

Q Explain the properties of Queue

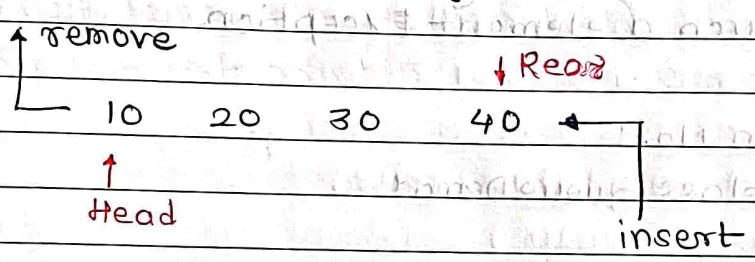
* Queue: (I):

- Queue interface is a type of collection interface.
- Queue interface can be use to perform the operation in FIFO (First in First out) Manner.
- queue interface is having two implimentation classes:

① Linked List

② priority

- In case of queue interface new element will be inserted at the end of queue and element will be remove from front of key.



* Methods of queue interface:

1) Offer (object o)

- This method can be used to add new element inside the Queue.

2) peek ():

- This method can be used to display head of the queue.
- If queue is empty this method will return null value.

3) Element ()

- This method is also used to display head of the queue.

- If queue is empty then this method will throw nosuchaelement exception.

4] Poll ():
- This method can be used to remove head of the queue.
- If queue is empty this method will written null value.

5] remove ():
- This method is also used to remove head of the queue
- If queue is empty then this method with show no such a element Exception.

- Program No. 1

```
public class QueueDemo1
{
  public static void main (String [] args)
  {
    Queue <Integer> data = new LinkedList<>();
    data.offer (e: 25)
    data.offer (e: 35)
    data.offer (e: 45)
    data.offer (e: 55)

    System.out.println (data);
    System.out.println ("HEAD:" + data.peek());
    ~~system~~ data.poll();
    system.out.println ("HEAD:"
    data.poll();
    System.out.println ("HEAD:"
    data.poll();
```

```
system.out.println("Head:" + data.peek());
data.poll();
system.out.println("HEAD:" + data.peek());
}
}
```

Q. Write differance between LinkedList and priorityQueue.

- Both are implimentation classes of queue interface only the differance is that Linkedlist retrive the data in sequencial manner where as priorityQueue retrive the data in ascending order (smallest element will be removed first).

```
public class QueueDemo2
{
  public static void main(String[] args)
  {
    Queue<Integer> data = new PriorityQueue<>();
    data.offer(e:65)
    data.offer(e:75)
    data.offer(e:20)
    data.offer(e:85)

    System.out.println(data);
    System.out.println("HEAD:" + data.peek());
    data.poll();
    System.out.println("HEAD:" + data.peek());
  }
}
```

DATE

Q. cmd

1: Generate Token
2: Process Token
3: Exit
1
Tokan no is _____

```
import Scanner.java.util;
Public class QueueDemo3
{
  public static void main (String [] args);
  {
    static Queue<Integer> tokens = new LinkedList<>();
    Static int = 201;
    scanner sc1 = neo scanner (System.in);
    boolean status = true;
    while (status)
    {
      System.out.println ("1:Generate Token");
      sy
      sy
      int choice = sc1.nextInt ();
      if(choice == 1)
        {
          generateTokens();
        }
      else if (choice == 2)
        {
          processTokens();
        }
      else
        {
          status = false;
        }
  } }
```

classmate

PAGE

DATE

```
static void processTokens(){
    System.out.println("Processing Token No:"+
                                    tokens.peek());
    tokens.poll();
    System.out.println("ORDER Processed sucefully");
}

static void generateTokens()
{ tokens.offer(tokenNo);
    System.out.println("Your Token no is:"+ tokenNo);
    tokenNo++;
}

}
```

DATE

What are the properties of MAP.

- Map (I) is not a part of collection interface but it is a part of collection framwork.
- Map interface is declearred inside the java.util packg
- We can use Map interface store the data in term key-value pair.
- Map interface internally used HashTable data structure to store the data.
- Default capacity of Map is 16 element with load factor 75%.
- Map interface is having 3 implementation classes.
  1) HashMap
  2) Linked HashMap
  3) Tree Map.

9. Differance betwee.....

HashMap	LinkedHashMap	TreeMap
1) HashMap is pure implimentation of interface.	1) LinkedHashMap is having properties of map as well as interface.	1) TreeMap is implimentation of sorted map.
2) HashMap stores the data in random order without duplicate entrys.	2) LinkedHashMap store data in a sequence manner without duplicate entrys	2) TreeMap stores the data acessending order without duplicate entrys.

DATE 27 05 2022

Q Explain methods of Map (I) interface:

1) Size ()
- This method can be used to display total number of available entries inside the map.

2) ContainsKey (object key)
- This method can be used to check weather the map contains specified key or not.

3) ContainsValue ():
- This method is used to check weather the map contains specified value or not.

4) get (object key)
- This method writtens the value of specified key.

5) Put (Object key, object value)
- This method can used to add new entry inside the map.

6) Remove (object key):
- This method can be used to remove a value along with specified key.

7) clear ():
- This method is used to delete all the entries from HashMap or anymap.

8) getkeySet ():
- This method will return/written set of all the keys.

9) Values ():
- This method will written all the values from map

classmate

10) entrySet():
- This method will written set of all the entries that is key-value pair.

* Program No. 1

```
public class HashMapDemo1
{
  public static void main (String [] args
  {
    HashMap<Integer, String> data = new HashMap ());
    data.put (17, "RED");
    data.put (20, "Green");
    data.put (35, " BLUE");
    data.put (39, " Voilet");

    System.out.println (data);
  }
}
```

* Program No. 2

```
public class LinkedHashMapDemo2
{
  public static void main (String [] args)
  {
    LinkedHashMap <Integer, String] data = new LinkedHashMap<>(
    data.put (411038, "PUNE");
    data.put (635891, " Banglore");
    data.put (411036, " Delhi);
    data.put (356542, " Chennai");
    data.put (744103, " pune");

    Set <map.Entry <Integer, String>> entry = data.
                                entrySet ();
```

```
System.out.println("PINOED" \t\t CJOY");
System.out.println("==========");
For (Map.Entry<Integer, String> e: entry)
{
    System.out.println(e.getKey()+"\t\t"+e.getValue())
}
}
}
```

Q.

Cmd

1. ADD NEW ENTRY
2. GET RESULT
3. Exit

choice

Enter Subject Name

_____

Enter Subject Marks

_____

Subject Marks

Total: Percentage:

Result:

<35 Fail

>=35 <=50 → Second class

>50 <60 → Higher Second

>60 <70 → First

>=70 → Dist

DATE

- Business Class:

```
public class ResultSimulator
{
    HashMap<String, Double> data = new HashMap<>();

    public void add subject (String name, double marks)
    {
        data.put (name, marks);
        System.out.println("Subject Added Sucessfully")
    }

    public void displayResult
    {
        Set<Map.Entry<String, Double>> entry = data.entrySet(
        double sum=0.0;
        For (Map.Entry<String, Double> e: entry)
        {
            System.out.println (e.getKey() +"\t\t"
                                    +e.get value());
            sum+ =e.get Value();
        }
        system.out.println ("=====  =======");
        double percent = sum/data.size ();

        System.out.println ("TOTAL:"+sum+" \t\t
                                    PERCENT: "+percent);
        if (percent<35.0)
        {
            System.out.println ("RESULT: FAIL");
        }
        else if (percent>= 35.0 && percent<= 50.0)
        {
            System.out.println("RESULT": Second class");
```

classmate

PAGE

DATE

```
else if (percent >=50 && percent <60)
 {
  system.out.println ("Result: Higher second class");
 }
else if (percent >=60 && percent <70)
 {
  system.out.println ("Result: First class");
 }
else
 {
  System.out.println ("Result: Destinction");
 }
}
}
```

- public class MainApp ——— (Executable class)

```
{
 public static void main (String [] args)
 {
  Scanner sc1 = new scanner (System.in);
  ResultSimulator r1 = new ResultSimulator();
  boolean status = true;
  while (status)
  {
   System.out.println ("Select Mode of operation");
   System.out.println ("1: Add new Subject");
   System.out.println ("2: Get Result");
   System.out.println ("3: Exit");
   int choice = sc1.next();
   switch (choice)
   {
    system.out.println("Enter subject name");
    string name = sc1.next();
```

DATE

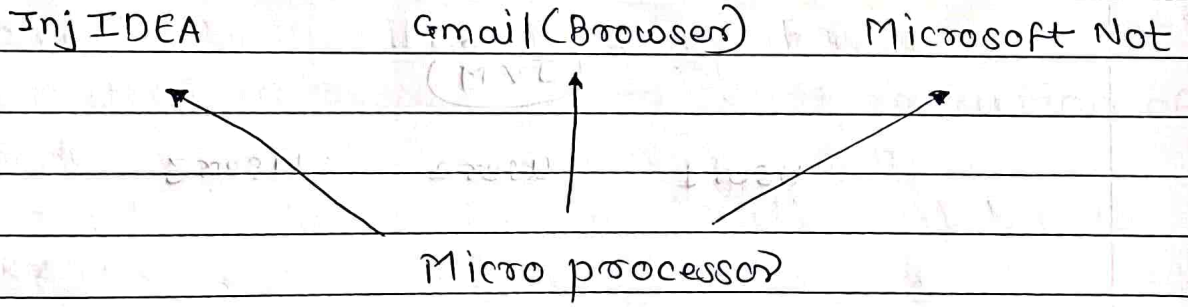
```
System.out.println("Enter subject marks");
double marks = sc1.nextDouble();
r1.add subject (name, marks);
break;
case 2:
    r1.displayResult();
    break;
default:
    status = false;
    break;
}
}
}
}
```

What is multi-Tasking:

- If the microprocessor ~~include~~ executing multiple task simuteneously or concurrently than is known as multitasking
- Multitasking can be achieved by using two ways:
  1. Multiprocessing (Process base multitasking)
  2. ~~Process B~~ Multithreading (thread base multitasking)

Multiprocessing:

- Exicuting multiple processes simultaneously by using one or more ~~imro~~ microprocessor is known as is known as multiprocessing
- Multiprocessing will be done at OS level.
- It means developer is not responsible for multi processing it will by operating system.
- Communication cost is very high between two processes

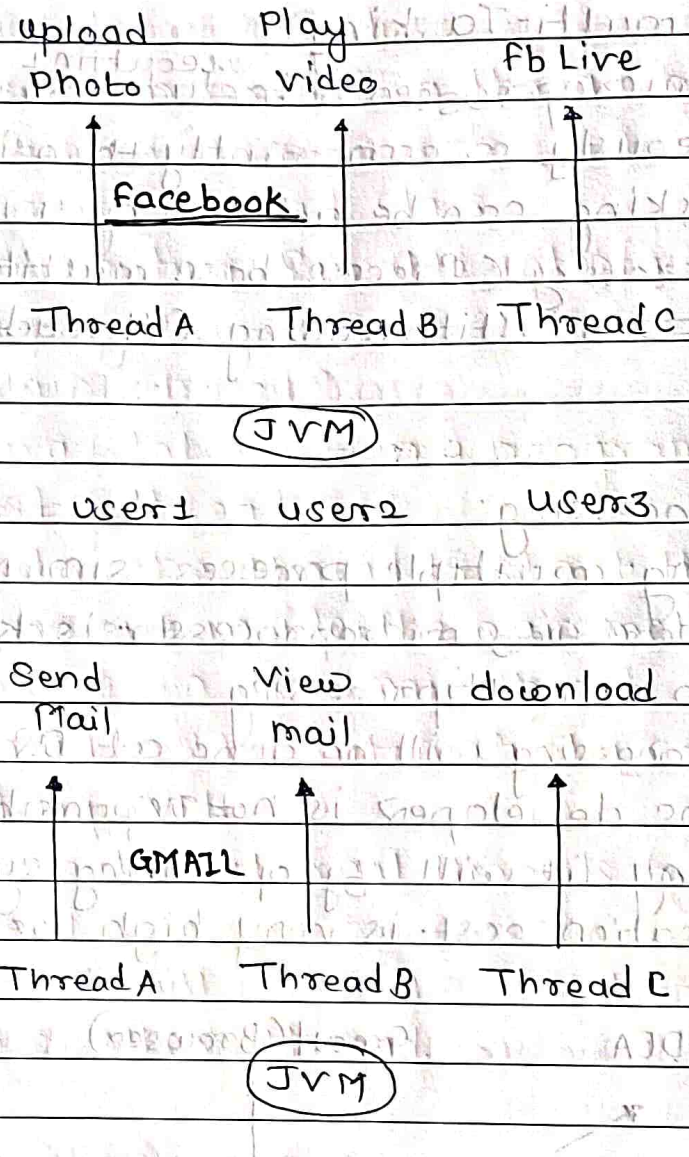


Multithreading:

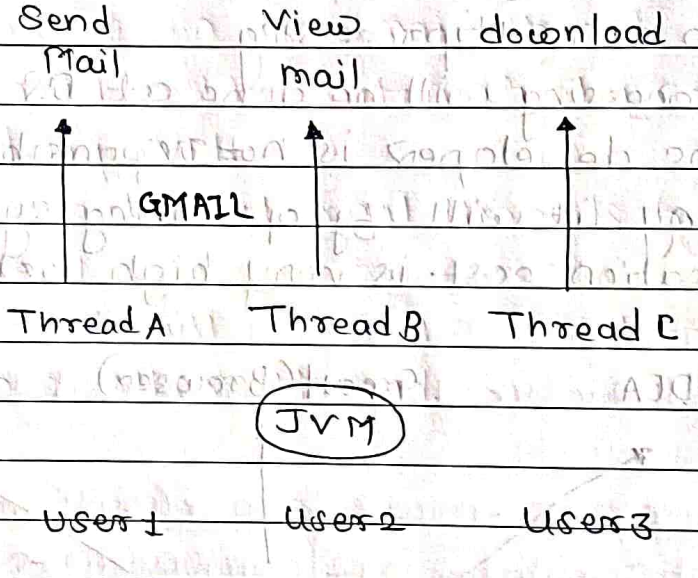
- Executing multiple threads or subclass/task simultaneously is know as multithreading.
- Multithreading can be done at programming level.
- Developer can create multiple threads to execute single process.
- Communication cost is very low between two threads.

classmate

eg ①



eg ②

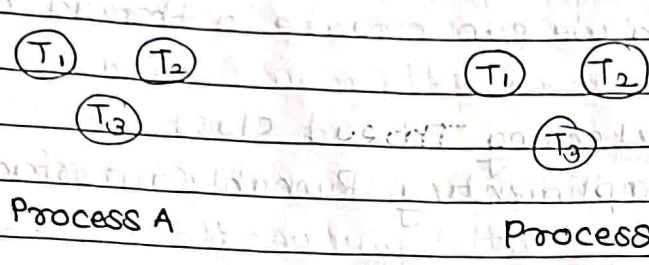


Q. What is Thread?

- Thread is light weight subprocess that can run simultaneously with other parts of the process.
- Threads are independant of each other it means if exception occur in one thread then it will not affect another thread.
- One thread can easily connect with other thread.

DATE

OS



Q. Explain Thread Class?

- Thread is the inbield class decleare inside the java.lang package.
- In java multithreading is completly depends on thread class.
- Thread class has provided several methods and constructor to support multithreading.
- Following are the important methods of thread class:

1] Start():
- This method is responsible to start execution of thread.

2] Run():
- This method perform specific action on thread object.

3] Sleep (long milliseconds):
- This method temperary stops execution of thread

4] Join():
- This method ~~along~~ allows one thread to complete execution

5] Stop():
- This method permenently stops the execution of thread

PAGE

classmate

DATE

Q. How to create a thread?

- In java we can create a thread by using 2 ways:
  1) By extending Thread class
  2) By implimenting Runnable interface.

Q. What is Main Thread?

- Every Java statement in a program will be executed by thread.
- JVM creates the thread to start the execution of code present inside the main method this thread is known as a main thread.
- Main thraid will be automatically created and executed by JVM.
- We can create external threads by using main thread.
- Main thread will finish the execution after execution of other thread.

DATE 31 05 2022

* Program No. 1

• Creating a thread by extending thread class

```
// Subclass
public class Demo extends Thread
{
  @ override
  public void run()
  {
    System.out.println("Thread is Created");
  }
}

public class MainApp1
{
  public static void main(String[] args)
  {
    Demo d1 = new Demo(); // Thread 1 created
    Demo d2 = new Demo(); // Thread 2 created

    d1.start(); // Thread 1 started
    d2.start(); // Thread 2 started
  }
}
```

* Program no. 2

• To create a thread by implimenting Runnable Interface

```
public class Master implements Runnable
{
  @ override
  public void run()
  {
    System.out.println("Thread Created");
  }
}
```


classmate

DATE

```
public class MainApp2
{
  public static void main (String [] args)
  {
    Master m1 = new master (); // normal object
    Master m2 = new Master (); // normal object

    Thread t1 = new Thread (m1); //converted to thread
    Thread t2 = new Thread (m2); //converted to thread

    t1. start ();
    t2. start ();
  }
}
```

* Program No.3 (Conquerant Programming)

```
public class central extends Thread
{
  @ Override
  public void run()
  {
    Char ch='A';

    for (int a=1; a<=3; a++)
    {
      System.out.println (ch);
      try {
          Thread.sleep (mills: 1000);
        }
      catch (InterruptedException e)
        {
          e. printStackTrace();
        }
      ch++
    }
  }
}
```

DATE

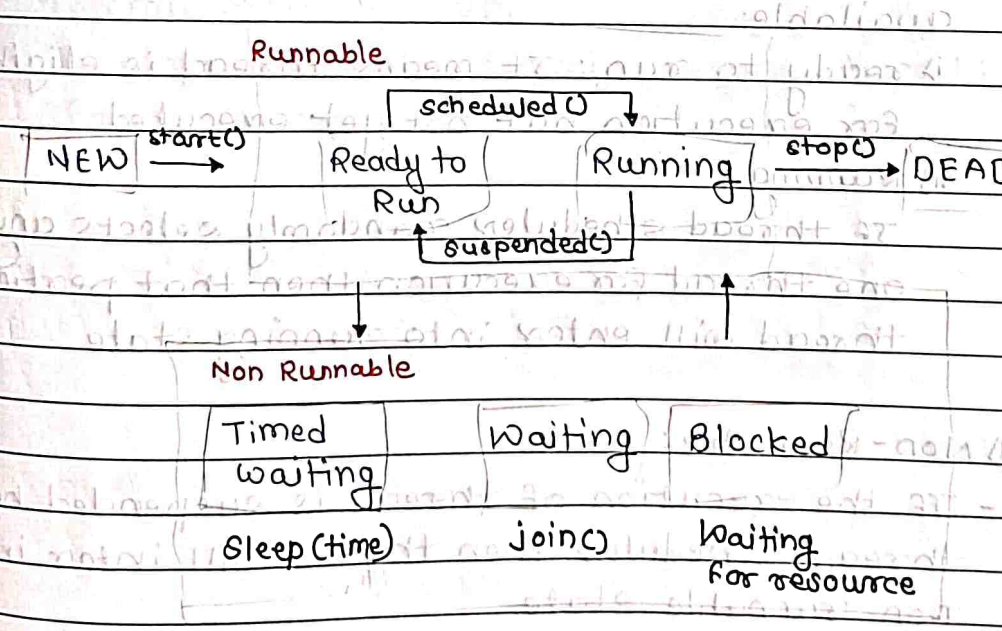
Conquerant / Asyncronise Programming.

```
public class MainApp3
{
  public static void main (String[] args)
  {
    Central c1 = new Central ();// thread 1
    Central c2 = new central ();// thread 2
    Central c3 = new Central ();// thread 3

    c1.start ();// thread 1 started
    c2.start ();
    c3.start ();
  }
}
```

O/P

```
A   B   C
A   B   C
A   B   C
    1sec
```

Q. Explain Thread Life Cycle:



DATE

Thread Life Cycle:

- It represents execution of thread object.
- In thread life cycle different phases are available.

1) New
2) Runnable
3) Non-Runnable
4) Terminate

1) New

- When developer creates a new thread then it will be available inside the new state
- In other words every thread object will start the execution from new state.

2) Runnable

- If developer calls start method then thread will enter into the runnable state.
- In runnable state two subphases are available:
  - i) ready to run: It means thread is eligible for execution but not yet executed.
  - ii) Running:
    If thread scheduler randomly selects any one thread for execution then that perticular thread will enter into running state.

3) Non-Runnable:

- If the execution of thread is suspended by thread scheduler then thread will inter into non-runnable state.
- In non-runnable state 3 supphases are present:

DATE

i) Timed waiting:
- If developer calls the sleep() method then thread will inter into time waiting phase.
- Time waiting means thread is suspended for specificified no. of seconds.

ii) Only waiting:
- If developer calls 'join' method then thread will inter into the waiting phase.
- In this phase thread will be suspended for unknown number of seconds.

iii) Blocked:
- If the resource for which thread is waiting which is already acquire by another thread then the perticular thread is blocked.
- One's the thread get blocked chances of execution was very less.

4) Terminate:
- After sucessful execution thread will be terminated by JVM.
- Developer can also terminate the thread by using stop() method.

classmate