CORE JAVA

Introduction to Java:

LanguageIntroduction to Java:

Language

=>Alphabets

=>Grammer(Syntax)

=>Construction Rules

Note:

=>Every language will have its own Alphabets, Grammer and Constructions

Parts:

a.Core java

b.AdvJava

a.Corejava:

=>CoreJava provides the following Programming components or Java Alphabets:

1.Variables

2.Methods

3.Blocks

4.Constructors

5.Classes

6.Interfaces

7.AbstractClasses

=>The following concepts are available from Corejava:

1.Object Oriented Programming Concept.

2.Exception Handling process

3.Multi-Threading process

4.Java Collection Framework(JCF)

5.IO Streams and Files

6.Networking

GUI(Graphical User Interface)

1.AWT(Abstract Window Toolkit)

2.Swing

3.Applet

4.javaFx

=>CoreJava provides the following Object Oriented features:

1.Class

2.Object

3.Abstraction

4.Encapsulation

5.PolyMorphism

6.Inheritance

Note:

=>Using CoreJava Components and Concepts we can develop StandAlone applications.

faq:

define StandAlone applications?

=>The applications which are installed in one computer and performs actions in the same computer are known as StandAlone applications.

According to developer,

No HTML input

No Server Environment

No DataBase Connection

---

b.AdvJava:

=>AdvJava provides the following technologies used in Constructing WebApplications:

1.JDBC

2.Servlet

3.JSP

=>JDBC stands for 'Java DataBase Connectivity' and which is used to establish communication b/w JavaProgram and DataBase product.

=>'Servlet' means 'Server program' and which is used to accept the request from User/client.

=>JSP stands for 'Java Server page' and which is response from WebApplication.

---

faq:

wt is the diff b/w

(i)Language

(ii)Technology

(iii)Framework

(i)Language:

=>Language Provides programming components used in Construction.

Exp:

CoreJava

(ii)Technology:

=>Technology means applying the knowledge to the realtime world for construction.

Exp:

AdvJava

(iii)Framework:

=>Framework means the Structure ready constructed and available for application development.

Exp:

Hibernate

Spring

==========

Dt: 6/9/2021

**imp*

Define Program?(Syllabus)

=>Program is a set-of-Instructions.

define Programming?

=>The process used in constructing programs is known as Programming process.

define programmer?

=>The person who writes programs is known as Programmer.

Note:

=>The programs are saved with language extention.

Exp:

Test.c

Test.cpp

Test.java

=>After writing and Saving the program,the program will have the following two stages:

1.Compilation Stage

2.Execution Stage

1.Compilation Stage:

=>The process of checking the program constructed within the rules of language or not,is known as Compilation process.

Note:

=>After Compilation process,

=>c and c++ programs generate Objective code

=>Java Programs generate Byte Code

2.Execution Stage:

=>The process of running the compiled code and checking the required output is generated or not,is known as Execution process.

Note:

=>In c and c++ programs the Objective Code is converted into Executable code and generate result.

=>In Java programs the Byte code is loaded onto JVM(Java Virtual Machine) for execution.

Dt : 7/9/2021

faq:

wt is the diff b/w

(i)Objective Code

(ii)Byte Code

(i)Objective Code:

=>The compiled code generated from c/c++ programs is known as Objedctive Code.

=>while Objective code generation,OperatingSystem(PlatForm) is participated,because of this reason Objective code is PlatForm dependent code.

Dis-Advantage:

=>The objective code which is generated from one PlatForm cannot be executed on other PlatForms.

Note:

=>The c and c++ languages which are generating Objective code are PlatForm dependent languages.

(ii)Byte Code:

=>The Compiled code generated from java programs is known as Byte Code.

=>while Byte code generation operatingSystem(PlatForm) is NotParticipated,because of this reason ByteCode is PlatForm independent code.

Advantage:

=>The ByteCode generated from one PlatForm can be executed on all the PlatForms,based on JVM.

Note:

=>The JavaLanguage which is generating ByteCode is PlatForm independent language.

==========

Diagram:

==========

define High Level Language?

=>The Language programs constructed from user understandable formats,is known as Hight Level

Language.

Exp:

c,c++,Java

define Low Level language?

=>The Language program formats which are not understandable by the user,is known as Low

Level language.

Exp:

Machine language

define Translator?

=>Translator is used to convert one language format into another language format,which means

HLL format to LLL format and LLL format to HLL.

=>These Translators are categorized into two types:

(i)Compilers

(ii)Interpreters

(i)Compilers - Compiler Translates total program at-a-time

(ii)Interpreters - Interpreter Translates the program line-by-line

==========

Dt :8/9/2021

Java Versions:

1995 - Java Alpha

1996 - JDK 1.0

1997 - JDK 1.1

1998 - JDK 1.2

2000 - JDK 1.3

2002 - JDK 1.4

2004 - Java5(Tiger java)

=>JDK 1.5

=>JRE 1.5

2006 - Java6

=>JDK 1.6

=>JRE 1.6

2011 - Java7

=>JDK 1.7

=>JRE 1.7

2014 - Java8

=>JDK 1.8

=>JRE 1.8

2017 - Java9

=>JDK 1.9

=>JRE 1.9

2018 - Java10,Java11

2019 - Java12,Java13

2020 - Java14,Java15

2021 - Java16

----

faq:

wt is the diff b/w

(i)JDK

(ii)JRE

(i)JDK:

=>JDK stands for 'Java Development Kit' and which provides JavaCompiler,JavaLibrary and JVM.

JavaCompiler - is used to compile the Source code and generate the ByteCode.

JavaLib - provides PreDefined components used in application development

JVM - is used to execute JavaByteCode.

**imp*

define JavaLib?

=>JavaLib is represented using the word 'java'.

=>JavaLib is collection of 'packages'

=>packages are collection of 'Classes and Interfaces'

=>Classes and Interfaces are collection of 'Variables and methods'

=>The following are some important packages from JavaLib:

CoreJava:

java.lang - Language package

java.io - Input/Output package

java.util - Utility package

java.net - Networking package

GUI:

java.awt - Abstract Window Toolkit package

javax.swing - swing programming package

java.applet - Applet programming package

AdvJava:

java.sql - DataBase Connection package

javax.servlet - Servlet programming package

javax.servlet.jsp - JSP programming package

(ii)JRE:

=>JRE stands for 'Java Runtime Environment' and which contains only JavaLib and JVM.

==========

Dt : 9/9/2021

Installing Java s/w and Setting path:

step-1 : Find the System environment(System type) where the JDK going to be installed.

step-2 : Download JDK(JDK16) from Oracle WebSite

step-3 : Install JDK

Note:

=>After installation process we can find one folder with name 'java' in 'ProgramFiles'.

C:\Program Files\Java

step-4 : Set path in 'Environment' variables

RightClick on MyComputer->Properties->Advanced System Settings->Environment Variables,

click 'new' from System Variables

Variable name : path

Variable value : C:\Program Files\Java\jdk-16.0.2\bin;

step-5 : Click 'ok' for three times,the JavaPath is setted.

Note:

=>Open CommandPrompt and check the following commands are working or not

javac - Used for Compilation process

java - Used for Execution process

==========

dt : 11/9/2021

faq:

define Environment Variables?

=>The variables which are controlled and managed by the operatingSystem are known as 'Environment variables'

=>Environment variables are categorized into two types:

(a)User variables

(b)System variables

(a)User variables:

=>The information in User variables can be used by only individual user.

(b)System variables:

=>The information in System variables can be used by all the users of Computer System.

Note:

=>In above installation process,the s/w information is kept in System variables

faq:

wt is the advantage of setting JavaPath in Environment variables?

=>when we set a JavaPath in Environment variables,then the JavaProgram can be compiled and executed from any location of Computer System.

Diagram:

Dt : 17/9/2021

****imp***

define 'Class'?

=>'Class' is a 'Structured Layout' and which is used to generate Objects.

=>Class in Java is a collection of Variables,Methods(functions) and main().

Variables - to hold data

Methods(functions) - to perform actions

main() - is the starting point of program execution.

=>we use 'class' keyword to construct classes in Java.

Structure of class in Java:

```
class Class_name
{
  //variables
  //methods
  //main()
}
```

Note:

=>In JavaLang main() method is having the following Standard format

"public static void main(String args[])"

Example program:

wap to display the msg as "Welcome to Java Programming"?

```
class Display
{
  public static void main(String args[])
  {
    System.out.print("Welcome to Java Programming");
  }
}
```

Writing,Saving,Compiling and Executing Java Program:

step-1 : Create one folder(destination folder) in any drive

E:\Demo123

step-2 : Open EditPlus(any text Editor) and type the program

Open EditPlus->Browse and select destination folder(Demo123)->click on 'File'->new->Jave and type the program

step-3 : Save the program in destination folder

syntax:

Class_name.java

Exp:

Display.java

Click on 'File'->Save->name the file(Display.java) and click 'save'

Note:

=>Open CommandPrompt and perform,Compilation and Execution process.

To open CommandPrompt->GoTo destination folder->type 'cmd' in address bar and press 'Enter'.

step-4 : Compile the program as follows

syntax:

javac Class_name.java

Exp:

javac Display.java

step-5 : Execute the program as follows

syntax:

java Class_name

Exp:

java Display

==========

Example program-2:

wap to add two numbers and display the result?

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

```
            int a=12,b=13;
            int c = a+b;
            System.out.println("The value of a="+a);
            System.out.println("The value of b="+b);
            System.out.println("The sum="+c);
        }
}
```

o/p:

The value of a=12

The value of b=13

The sum=25

==========

dt : 18/9/2021

**imp*

DataTypes in Java:

=>The types of data which we are expecting as input to Java programs are known as 'DataTypes in Java'.

=>DataTypes in Java are categorized into two types:

1.Primitive DataTypes

2.NonPrimitive DataTypes

1.Primitive DataTypes:

=>'Single valued data formats' are known as Primitive datatypes.

=>Primitive DataTypes are categorized into the following:

(a)Integer DataTypes

(b)Float DataTypes

(c)Character DataType

(d)Boolean DataType

(a)Integer DataTypes:

=>The numeric data which is represented without decimal point,is known as Integer datatype.

Types:

(i)byte - 1 byte(8 bits)

(ii)short - 2 bytes

(iii)int - 4 bytes

(iv)long - 8 bytes

Note:

=>byte and short datatypes are used for Stream data,which means used for Multi-Media data.

(Audio,Video,Image,Animation and Text)

=>int datatype is used in normal programming.

=>long data type is used to hold very large values like PhoneNo,cardNo,...

=>when we want to assign long value,we must use 'l' or 'L' in the RHS of declaration.

Ex:

```
long l = 9898981234L;
```

(b)Float DataTypes:

=>The numeric data with decimal point representation is known as Float datatype.

Types:

(i)float - 4 bytes

(ii)double - 8 bytes

Note:

=>float datatype is used in normal programming.

=>when we want to assign float value,we must use 'f' or 'F' in the RHS of declaration.

=>double datatype is used to hold very large scietific calculated values.

(c)Character DataType:

=>The 'single valued' character represented in single quotes is known as 'Character data type'

Ex:

'k','i','r',...

Types :

char - 2 bytes

(d)Boolean DataType:

=>The datatype which is represented in the form of true or false,is known as Boolean datatype.

Types:

boolean - 1 bit

Ex program : DataTypes.java

```
class DataTypes
{
        public static void main(String[] args)
        {
                byte b = 127;
                short s = 32767;
                int i = 456782;
                long l = 9898981234L;
                float f = 12.34F;
                double d = 12345.678;
                char ch = 'A';
                boolean bl = true;
                System.out.println("byte value:"+b);
                System.out.println("short value:"+s);
                System.out.println("int value:"+i);
                System.out.println("long value:"+l);
                System.out.println("float value:"+f);
                System.out.println("double value:"+d);
    System.out.println("char value:"+ch);
```

```
        System.out.println("boolean value:"+bl);
            }
}
```

o/p:

byte value:127

short value:32767

int value:456782

long value:9898981234

float value:12.34

double value:12345.678

char value:A

boolean value:true

byte - 8 bits	short - 16 bits
$2^8 = 256$	$2^{16} = 65536$
256/2 = 128	65536/2 = 32768
range : -128 to +127	range : -32768 to + 32767

==========

**imp*

2.NonPrimitive DataTypes:

=>'Group Valued data formats' are known as NonPrimitive datatypes or Referential datatypes

=>NonPrimitive datatypes are categorized into the following:

(a)Class

(b)Interface

(c)Array

(d)Enum

==========

Assignment1:

wap to evaluate the following:

*(i) z = x+y/(a*b);*

*(ii) c = (a*b)/(x*y)+(p*q);*

==========

Dt: 21/9/2021

**imp*

Object-Oriented Programming:

=>The process of constructing programs using the class-object concept is known as Object Oriented programming.

=>In Object Oriented programming we control the following NonPrimitive datatypes or referentia datatypes:

1.Class

2.Interface

3.Array

4.Enun

**imp*

1. Class:

=>class is a 'structured layout' in java and which generates objects.

=>class is a collection of variables and methods.

=>class is loaded onto method_area of JVM.

=>classes in Java are categorized into two types:

(a)User-defined classes

(b)Built-in classes

(a)User-defined classes:

=>The classes which are defined by the programmer are known as User-defined classes.

(b)Built-in classes:

=>The classes which are available from JavaLib are known as Built-in classes or PreDefined Classes

---

**imp*

Define Object?

=>Object is a memory related to a class holding the members of classes.

=>we use 'new' keyword in java to create objects

=>syntax of object creation using 'new' keyword:

Class_name obj_name = new Class_name();

Note:

=>In the process of constructing applications, we use SubClass and MainClass

SubClass – Class which is declared without the main() method.

MainClass – Class which is declared with main() method

Ex program:

Wap to display user details using Class-Object Concept?

```
Class User
{
String name,mailId;
Void getUserDetails()
{
System.out.println("name:"+name);
System.out.println("MailId:"+mailId);
}
}
Class UserMainClass
```

```
{
Public static void main(String[] args)
{
User u = new User();
u.name="Raj";
u.mailId="raj@gmail.com";
u.getUserDetails();
}
}
```

Dt: 22/9/2021

wap to display Employee details using Class-Object Concept?

EmployeeData

=>empId,empName,empDesg,empBSal

=>void getEmployeeData()

EmployeeAddress

=>hNo,sName,city,pinCode

=>void getEmployeeAddress()

EmployeeContact

=>phoneNo,mailId

=>void getEmployeeContact()

EmployeeMainClass

=>public static void main(String[] args)

==========

Assignment1:

wap to display Student details using the Class-object concept?

StudentData

=>rollNo,stuName,branch

=>void getStudentData()

StudentAddress

=>hNo,sName,city,pinCode

=>void getStudentAddress()

StudentContact

=>phoneNo,mailId

=>void getStudentContact()

StudentMainClass

=>public static void main(String[] args)

Assignment2:

wap to display Customer details using the Class-Object concept?

CustomerData

=>accNo,custName,balance,accType

=>void getCustomerData()

CustomerAddress

=>hNo,sName,city,pinCode

=>void getCustomerAddress()

CustomerContact

=>phoneNo,mailId

=>void getCustomerContact()

CustomerMainClass

=>public static void main(String[] args)

==========

**imp*

'Scanner' class:

=>'Scanner' class is a Built-in class from JavaLib available from 'util' package.

=>'Scanner' class will provide the following Built-in methods which are used in reading the data into Java programs:

1.nextByte() - to read byte data

2.nextShort() - to read short data

3.nextInt() - to read int data

4.nextLong() - to read long data

5.nextFloat() - to read float data

6.nextDouble() - to read double data

7.nextBoolean() - to read boolean data

8.nextLine() - to read String data

Dt: 23/9/2021

Ex program: DemoInput.java

o/p:

Enter the name:

Raj

Enter PhoneNo:

9898981234

Enter the percentage:

67.78

Name: Raj

PhoneNo:9898981234

Per:67.78

==========

Ex program: wap to read and display Product details using class-object concept?

o/p:

Enter the ProdCode:

A111

Enter the ProdName:

CDR

Enter the ProdPrice:

456.78

Enter the ProdQty:

12

=====Product details=====

ProdCode:A111

ProdName:CDR

ProdPrice:456.78

ProdQty:12

==========

Assignment1:

wap to read and display User details using the Class-object concept?

UserDetails

=>uName,pWord,fName,lName,addr,mailId,phoneNo

=>void getUserDetails()

UserMainClass2.java

=>public static void main(String args[])

Assignment2:

Update BookDetails program by reading and displaying

==========

Dt: 24/9/2021

Note:

=>The data will be skipped when we read String data, after reading numeric data like byte,short,int,long,float and double.

=>This can be overcomed using the following Built-in parse methods:

```
byte b = Byte.parseByte(s.nextLine());
short s1 = Short.parseShort(s.nextLine());
```

```
int i = Integer.parseInt(s.nextLine());
long l = Long.parseLong(s.nextLine());
float f = Float.parseFloat(s.nextLine());
double d = Double.parseDouble(s.nextLine());
```

Note:

=>Byte,Short,Integer,Long,Float and Double are built-in WrapperClasses.

Ex program : DemoInput2.java

```
import java.util.Scanner;
class DemoInput2 //MainClass
{
        public static void main(String[] args)
        {
    Scanner s = new Scanner(System.in);
            System.out.println("Enter the name:");
            String name = s.nextLine();
            System.out.println("Enter the PhoneNo:");
            long phoneNo = Long.parseLong(s.nextLine());
            System.out.println("Enter the MailId:");
            String mailId = s.nextLine();
            System.out.println("Name:"+name);
            System.out.println("PhoneNo:"+phoneNo);
            System.out.println("MailId:"+mailId);
        }
}
```

o/p:

```
Enter the name:
Raj
Enter the PhoneNo:
```

9898981234

Enter the MailId:

raj@gmail.com

Name: Raj

PhoneNo:9898981234

MailId:raj@gmail.com

Diagram:

==========

Ex program : Update EmployeeMainClass.java

```
import java.util.Scanner;

class EmployeeData

{

        String empId,empName,empDesg;

        int empBSal;

   void getEmployeeData()
```

```
        {
                System.out.println("====EmployeeData====");
                System.out.println("EmpId:"+empId);
    System.out.println("EmpName:"+empName);
    System.out.println("EmpDesg:"+empDesg);
    System.out.println("EmpBSal:"+empBSal);
        }
}
class EmployeeAddress
{
        String hNo,sName,city;
        int pinCode;
  void getEmployeeAddress()
        {
                System.out.println("=====EmployeeAddress=====");
    System.out.println("HNo:"+hNo);
    System.out.println("SName:"+sName);
    System.out.println("City:"+city);
    System.out.println("PinCode:"+pinCode);
        }
}
class EmployeeContact
{
        long phoneNo;
        String mailId;
  void getEmployeeContact()
        {
                System.out.println("=====EmployeeContact=====");
                System.out.println("PhoneNo:"+phoneNo);
    System.out.println("MailId:"+mailId);
```

```
        }
}
class EmployeeMainClass //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                EmployeeData ed = new EmployeeData();
                EmployeeAddress ea = new EmployeeAddress();
                EmployeeContact ec = new EmployeeContact();
    System.out.println("Enter the EmpId:");
                ed.empId = s.nextLine();
                System.out.println("Enter the EmpName:");
                ed.empName = s.nextLine();
                System.out.println("Enter the EmpDesg:");
                ed.empDesg = s.nextLine();
                System.out.println("Enter the EmpBSal:");
                ed.empBSal = Integer.parseInt(s.nextLine());
    System.out.println("Enter the HNo:");
                ea.hNo = s.nextLine();
                System.out.println("Enter the StreetName:");
                ea.sName = s.nextLine();
                System.out.println("Enter the City:");
                ea.city = s.nextLine();
                System.out.println("Enter the PinCode:");
                ea.pinCode = Integer.parseInt(s.nextLine());
    System.out.println("Enter the PhoneNo:");
                ec.phoneNo = Long.parseLong(s.nextLine());
                System.out.println("Enter the MailId:");
                ec.mailId = s.nextLine();
```

```
                ed.getEmployeeData();
                ea.getEmployeeAddress();
                ec.getEmployeeContact();

        }
}
```

Assignment1:

Update Student details program by reading and displaying the details?

Assignment2:

Update Customer details program by reading and displaying the details?

==========

****imp***

Operators in Java:

=>Operator is a special symbol or keyword used to perform operations.

=>The following are some important operators used in Java:

1. ***Arithmetic Operators***
2. ***Relational Operators***
3. ***Logical Operators***
4. ***Increment/Decrement Operators***

1. ***Arithmetic Operators:***

=>The operators which perform basic operations are known as Arithmetic operators.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplicatioon
/	Division
%	ModDivision

2. ***Relational Operators:***

=>The operators which perform comparisons and generate the boolean results.

Operator	Meaning
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Is equal to
!=	Not Equal to

3. ***Logical Operators:***

=>The operators which are used to compare two comparisons and generate boolean result.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical AND(&&):

A	B	A&&B
T	T	T
F	T	F
T	F	F
F	F	F

Logical OR(||):

A	B	A||B

T T T

F T T

T F T

F F F

Logical NOT(!):

A !A

T F

F T

4. Increment/Decrement Operators:

=>Increment operator is used to increment the value by 1 and decrement operator is used to decrement the value by 1.

Operator Meaning

++ Increment

-- Decrement

==========

=>Alphabets

=>Grammer(Syntax)

=>Construction Rules

Note:

=>Every language will have its own Alphabets, Grammer and Constructions

Parts:

a.Core java

b.AdvJava

a.Corejava:

=>CoreJava provides the following Programming components or Java Alphabets:

1.Variables

2.Methods

3.Blocks

4.Constructors

5.Classes

6.Interfaces

7.AbstractClasses

=>The following concepts are available from Corejava:

1.Object Oriented Programming Concept.

2.Exception Handling process

3.Multi-Threading process

4.Java Collection Framework(JCF)

5.IO Streams and Files

6.Networking

GUI(Graphical User Interface)

1.AWT(Abstract Window Toolkit)

2.Swing

3.Applet

4.javaFx

=>CoreJava provides the following Object Oriented features:

1.Class

2.Object

3.Abstraction

4.Encapsulation

5.PolyMorphism

6.Inheritance

Note:

=>Using CoreJava Components and Concepts we can develop StandAlone applications.

faq:

define StandAlone applications?

=>The applications which are installed in one computer and performs actions in the same computer are known as StandAlone applications.

According to developer,

No HTML input

No Server Environment

No DataBase Connection

----

b.AdvJava:

=>AdvJava provides the following technologies used in Constructing WebApplications:

1.JDBC

2.Servlet

3.JSP

=>JDBC stands for 'Java DataBase Connectivity' and which is used to establish communication b/w JavaProgram and DataBase product.

=>'Servlet' means 'Server program' and which is used to accept the request from User/client.

=>JSP stands for 'Java Server page' and which is response from WebApplication.

---

faq:

wt is the diff b/w

(i)Language

(ii)Technology

(iii)Framework

(i)Language:

=>Language Provides programming components used in Construction.

Exp:

CoreJava

(ii)Technology:

=>Technology means applying the knowledge to the realtime world for construction.

Exp:

AdvJava

(iii)Framework:

=>Framework means the Structure ready constructed and available for

application development.

Exp:

Hibernate

Spring

==========

Dt: 6/9/2021

**imp*

Define Program?(Syllabus)

=>Program is a set-of-Instructions.

define Programming?

=>The process used in constructing programs is known as Programming process.

define programmer?

=>The person who writes programs is known as Programmer.

Note:

=>The programs are saved with language extention.

Exp:

Test.c

Test.cpp

Test.java

=>After writing and Saving the program,the program will have the following two stages:

1.Compilation Stage

2.Execution Stage

1.Compilation Stage:

=>The process of checking the program constructed within the rules of

language or not,is known as Compilation process.

Note:

=>After Compilation process,

=>c and c++ programs generate Objective code

=>Java Programs generate Byte Code

2.Execution Stage:

=>The process of running the compiled code and checking the required output is generated or not,is known as Execution process.

Note:

=>In c and c++ programs the Objective Code is converted into Executable code and generate result.

=>In Java programs the Byte code is loaded onto JVM(Java Virtual Machine) for execution.

Dt : 7/9/2021

faq:

wt is the diff b/w

(i)Objective Code

(ii)Byte Code

(i)Objective Code:

=>The compiled code generated from c/c++ programs is known as Objedctive Code.

=>while Objective code generation,OperatingSystem(PlatForm) is participated,because of this reason Objective code is PlatForm dependent code.

Dis-Advantage:

=>The objective code which is generated from one PlatForm cannot be executed on other PlatForms.

Note:

=>The c and c++ languages which are generating Objective code are PlatForm dependent languages.

(ii)Byte Code:

=>The Compiled code generated from java programs is known as Byte Code.

=>while Byte code generation operatingSystem(PlatForm) is NotParticipated,because of this reason ByteCode is PlatForm independent code.

Advantage:

=>The ByteCode generated from one PlatForm can be executed on all the PlatForms,based on JVM.

Note:

=>The JavaLanguage which is generating ByteCode is PlatForm independent language.

==========

Diagram:

==========

define High Level Language?

=>The Language programs constructed from user understandable formats,is known as Hight Level

Language.

Exp:

c,c++,Java

define Low Level language?

=>The Language program formats which are not understandable by the user,is known as Low

Level language.

Exp:

Machine language

define Translator?

=>Translator is used to convert one language format into another language format,which means

HLL format to LLL format and LLL format to HLL.

=>These Translators are categorized into two types:

(i)Compilers

(ii)Interpreters

(i)Compilers - Compiler Translates total program at-a-time

(ii)Interpreters - Interpreter Translates the program line-by-line

==========

Dt :8/9/2021

Java Versions:

1995 - Java Alpha

1996 - JDK 1.0

1997 - JDK 1.1

1998 - JDK 1.2

2000 - JDK 1.3

2002 - JDK 1.4

2004 - Java5(Tiger java)

=>JDK 1.5

=>JRE 1.5

2006 - Java6

=>JDK 1.6

=>JRE 1.6

2011 - Java7

=>JDK 1.7

=>JRE 1.7

2014 - Java8

=>JDK 1.8

=>JRE 1.8

2017 - Java9

=>JDK 1.9

=>JRE 1.9

2018 - Java10,Java11

2019 - Java12,Java13

2020 - Java14,Java15

2021 - Java16

---

faq:

wt is the diff b/w

(i)JDK

(ii)JRE

(i)JDK:

=>JDK stands for 'Java Development Kit' and which provides JavaCompiler,JavaLibrary and JVM.

JavaCompiler - is used to compile the Source code and generate the ByteCode.

JavaLib - provides PreDefined components used in application development

JVM - is used to execute JavaByteCode.

**imp*

define JavaLib?

=>JavaLib is represented using the word 'java'.

=>JavaLib is collection of 'packages'

=>packages are collection of 'Classes and Interfaces'

=>Classes and Interfaces are collection of 'Variables and methods'

=>The following are some important packages from JavaLib:

CoreJava:

java.lang - Language package

java.io - Input/Output package

java.util - Utility package

java.net - Networking package

GUI:

java.awt - Abstract Window Toolkit package

javax.swing - swing programming package

java.applet - Applet programming package

AdvJava:

java.sql - DataBase Connection package

javax.servlet - Servlet programming package

javax.servlet.jsp - JSP programming package

(ii)JRE:

=>JRE stands for 'Java Runtime Environment' and which contains only JavaLib and JVM.

==========

Dt : 9/9/2021

Installing Java s/w and Setting path:

step-1 : Find the System environment(System type) where the JDK going to be installed.

step-2 : Download JDK(JDK16) from Oracle WebSite

step-3 : Install JDK

Note:

=>After installation process we can find one folder with name 'java' in 'ProgramFiles'.

C:\Program Files\Java

step-4 : Set path in 'Environment' variables

RightClick on MyComputer->Properties->Advanced System Settings->Environment Variables,

click 'new' from System Variables

Variable name : path

Variable value : C:\Program Files\Java\jdk-16.0.2\bin;

step-5 : Click 'ok' for three times,the JavaPath is setted.

Note:

=>Open CommandPrompt and check the following commands are working or not

javac - Used for Compilation process

java - Used for Execution process

==========

dt : 11/9/2021

faq:

define Environment Variables?

=>The variables which are controlled and managed by the operatingSystem are known as 'Environment variables'

=>Environment variables are categorized into two types:

(a)User variables

(b)System variables

(a)User variables:

=>The information in User variables can be used by only individual user.

(b)System variables:

=>The information in System variables can be used by all the users of Computer System.

Note:

=>In above installation process,the s/w information is kept in System variables

faq:

wt is the advantage of setting JavaPath in Environment variables?

=>when we set a JavaPath in Environment variables,then the JavaProgram can be compiled and executed from any location of Computer System.

Diagram:

Dt : 17/9/2021

****imp***

define 'Class'?

=>'Class' is a 'Structured Layout' and which is used to generate Objects.

=>Class in Java is a collection of Variables,Methods(functions) and main().

Variables - to hold data

Methods(functions) - to perform actions

main() - is the starting point of program execution.

=>we use 'class' keyword to construct classes in Java.

Structure of class in Java:

```
class Class_name
{
  //variables
  //methods
  //main()
}
```

Note:

=>In JavaLang main() method is having the following Standard format

"public static void main(String args[])"

Example program:

wap to display the msg as "Welcome to Java Programming"?

```
class Display
{
  public static void main(String args[])
  {
    System.out.print("Welcome to Java Programming");
  }
}
```

Writing,Saving,Compiling and Executing Java Program:

step-1 : Create one folder(destination folder) in any drive

E:\Demo123

step-2 : Open EditPlus(any text Editor) and type the program

Open EditPlus->Browse and select destination folder(Demo123)->click on 'File'->new->Jave and type the program

step-3 : Save the program in destination folder

syntax:

Class_name.java

Exp:

Display.java

Click on 'File'->Save->name the file(Display.java) and click 'save'

Note:

=>Open CommandPrompt and perform,Compilation and Execution process.

To open CommandPrompt->GoTo destination folder->type 'cmd' in address bar and press 'Enter'.

step-4 : Compile the program as follows

syntax:

javac Class_name.java

Exp:

javac Display.java

step-5 : Execute the program as follows

syntax:

java Class_name

Exp:

java Display

==========

Example program-2:

wap to add two numbers and display the result?

```
class Addition
{
        public static void main(String[] args)
        {
                int a=12,b=13;
                int c = a+b;
                System.out.println("The value of a="+a);
                System.out.println("The value of b="+b);
                System.out.println("The sum="+c);
        }
}
```

o/p:

The value of a=12

The value of b=13

The sum=25

==========

dt : 18/9/2021

**imp*

DataTypes in Java:

=>The types of data which we are expecting as input to Java programs are known as 'DataTypes in Java'.

=>DataTypes in Java are categorized into two types:

1.Primitive DataTypes

2.NonPrimitive DataTypes

1.Primitive DataTypes:

=>'Single valued data formats' are known as Primitive datatypes.

=>Primitive DataTypes are categorized into the following:

(a)Integer DataTypes

(b)Float DataTypes

(c)Character DataType

(d)Boolean DataType

(a)Integer DataTypes:

=>The numeric data which is represented without decimal point,is known as Integer datatype.

Types:

(i)byte - 1 byte(8 bits)

(ii)short - 2 bytes

(iii)int - 4 bytes

(iv)long - 8 bytes

Note:

=>byte and short datatypes are used for Stream data,which means used for Multi-Media data.

(Audio,Video,Image,Animation and Text)

=>int datatype is used in normal programming.

=>long data type is used to hold very large values like PhoneNo,cardNo,...

=>when we want to assign long value,we must use 'l' or 'L' in the RHS of declaration.

Ex:

```
long l = 9898981234L;
```

(b)Float DataTypes:

=>The numeric data with decimal point representation is known as Float datatype.

Types:

(i)float - 4 bytes

(ii)double - 8 bytes

Note:

=>float datatype is used in normal programming.

=>when we want to assign float value,we must use 'f' or 'F' in the RHS of declaration.

=>double datatype is used to hold very large scietific calculated values.

(c)Character DataType:

=>The 'single valued' character represented in single quotes is known as 'Character data type'

Ex:

'k','i','r',...

Types :

char - 2 bytes

(d)Boolean DataType:

=>The datatype which is represented in the form of true or false,is known as Boolean datatype.

Types:

boolean - 1 bit

Ex program : DataTypes.java

```
class DataTypes
{
        public static void main(String[] args)
        {
                byte b = 127;
                short s = 32767;
                int i = 456782;
```

```
                long l = 9898981234L;
                float f = 12.34F;
                double d = 12345.678;
                char ch = 'A';
                boolean bl = true;
                System.out.println("byte value:"+b);
                System.out.println("short value:"+s);
                System.out.println("int value:"+i);
                System.out.println("long value:"+l);
                System.out.println("float value:"+f);
                System.out.println("double value:"+d);
    System.out.println("char value:"+ch);
    System.out.println("boolean value:"+bl);
        }
}
```

o/p:

byte value:127

short value:32767

int value:456782

long value:9898981234

float value:12.34

double value:12345.678

char value:A

boolean value:true

byte - 8 bits

$2^8 = 256$

256/2 = 128

range : -128 to +127

short - 16 bits

$2^{16} = 65536$

65536/2 = 32768

range : -32768 to + 32767

==========

*imp

2.NonPrimitive DataTypes:

=>'Group Valued data formats' are known as NonPrimitive datatypes or Referential datatypes

=>NonPrimitive datatypes are categorized into the following:

(a)Class

(b)Interface

(c)Array

(d)Enum

==========

Assignment1:

wap to evaluate the following:

(i) z = x+y/(a*b);

(ii) c = (a*b)/(x*y)+(p*q);

==========

Dt: 21/9/2021

**imp*

Object-Oriented Programming:

=>The process of constructing programs using the class-object concept is known as Object Oriented programming.

=>In Object Oriented programming we control the following NonPrimitive datatypes or referentia datatypes:

1.Class

2.Interface

3.Array

4.Enun

**imp*

1. Class:

=>class is a 'structured layout' in java and which generates objects.

=>class is a collection of variables and methods.

=>class is loaded onto method_area of JVM.

=>classes in Java are categorized into two types:

(a)User-defined classes

(b)Built-in classes

(a)User-defined classes:

=>The classes which are defined by the programmer are known as User-defined classes.

(b)Built-in classes:

=>The classes which are available from JavaLib are known as Built-in classes or PreDefined Classes

---

**imp*

Define Object?

=>Object is a memory related to a class holding the members of classes.

=>we use 'new' keyword in java to create objects

=>syntax of object creation using 'new' keyword:

Class_name obj_name = new Class_name();

Note:

=>In the process of constructing applications, we use SubClass and MainClass

SubClass – Class which is declared without the main() method.

MainClass – Class which is declared with main() method

Ex program:

Wap to display user details using Class-Object Concept?

```
Class User
{
String name,mailId;
Void getUserDetails()
{
System.out.println("name:"+name);
System.out.println("MailId:"+mailId);
}
}
Class UserMainClass
{
Public static void main(String[] args)
{
User u = new User();
u.name="Raj";
u.mailId="raj@gmail.com";
u.getUserDetails();
}
}
```

Dt: 22/9/2021

wap to display Employee details using Class-Object Concept?

EmployeeData

=>empId,empName,empDesg,empBSal

=>void getEmployeeData()

EmployeeAddress

=>hNo,sName,city,pinCode

=>void getEmployeeAddress()

EmployeeContact

=>phoneNo,mailId

=>void getEmployeeContact()

EmployeeMainClass

=>public static void main(String[] args)

==========

Assignment1:

wap to display Student details using the Class-object concept?

StudentData

=>rollNo,stuName,branch

=>void getStudentData()

StudentAddress

=>hNo,sName,city,pinCode

=>void getStudentAddress()

StudentContact

=>phoneNo,mailId

=>void getStudentContact()

StudentMainClass

=>public static void main(String[] args)

Assignment2:

wap to display Customer details using the Class-Object concept?

CustomerData

=>accNo,custName,balance,accType

=>void getCustomerData()

CustomerAddress

=>hNo,sName,city,pinCode

=>void getCustomerAddress()

CustomerContact

=>phoneNo,mailId

=>void getCustomerContact()

CustomerMainClass

=>public static void main(String[] args)

==========

**imp*

'Scanner' class:

=>'Scanner' class is a Built-in class from JavaLib available from 'util' package.

=>'Scanner' class will provide the following Built-in methods which are used in reading the data into Java programs:

1.nextByte() - to read byte data

2.nextShort() - to read short data

3.nextInt() - to read int data

4.nextLong() - to read long data

5.nextFloat() - to read float data

6.nextDouble() - to read double data

7.nextBoolean() - to read boolean data

8.nextLine() - to read String data

Dt: 23/9/2021

Ex program: DemoInput.java

o/p:

Enter the name:

Raj

Enter PhoneNo:

9898981234

Enter the percentage:

67.78

Name: Raj

PhoneNo:9898981234

Per:67.78

==========

Ex program: wap to read and display Product details using class-object concept?

o/p:

Enter the ProdCode:

A111

Enter the ProdName:

CDR

Enter the ProdPrice:

456.78

Enter the ProdQty:

12

=====Product details=====

ProdCode:A111

ProdName:CDR

ProdPrice:456.78

ProdQty:12

==========

Assignment1:

wap to read and display User details using the Class-object concept?

UserDetails

=>uName,pWord,fName,lName,addr,mailId,phoneNo

=>void getUserDetails()

UserMainClass2.java

=>public static void main(String args[])

Assignment2:

Update BookDetails program by reading and displaying

==========

Dt: 24/9/2021

Note:

=>The data will be skipped when we read String data, after reading numeric data like byte,short,int,long,float and double.

=>This can be overcomed using the following Built-in parse methods:

```
byte b = Byte.parseByte(s.nextLine());
short s1 = Short.parseShort(s.nextLine());
int i = Integer.parseInt(s.nextLine());
long l = Long.parseLong(s.nextLine());
float f = Float.parseFloat(s.nextLine());
double d = Double.parseDouble(s.nextLine());
```

Note:

=>Byte,Short,Integer,Long,Float and Double are built-in WrapperClasses.

Ex program : DemoInput2.java

```
import java.util.Scanner;
class DemoInput2 //MainClass
```

```
{
        public static void main(String[] args)
        {
    Scanner s = new Scanner(System.in);
          System.out.println("Enter the name:");
          String name = s.nextLine();
          System.out.println("Enter the PhoneNo:");
          long phoneNo = Long.parseLong(s.nextLine());
          System.out.println("Enter the MailId:");
          String mailId = s.nextLine();
          System.out.println("Name:"+name);
          System.out.println("PhoneNo:"+phoneNo);
          System.out.println("MailId:"+mailId);
        }
}
```

o/p:

Enter the name:

Raj

Enter the PhoneNo:

9898981234

Enter the MailId:

raj@gmail.com

Name: Raj

PhoneNo:9898981234

MailId:raj@gmail.com

Diagram:

==========

Ex program : Update EmployeeMainClass.java

```
import java.util.Scanner;

class EmployeeData

{

        String empId,empName,empDesg;

        int empBSal;

   void getEmployeeData()

        {

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

                System.out.println("EmpId:"+empId);

     System.out.println("EmpName:"+empName);

     System.out.println("EmpDesg:"+empDesg);

     System.out.println("EmpBSal:"+empBSal);

        }

}
```

```
class EmployeeAddress
{
        String hNo,sName,city;
        int pinCode;
   void getEmployeeAddress()
        {
                System.out.println("=====EmployeeAddress=====");
     System.out.println("HNo:"+hNo);
     System.out.println("SName:"+sName);
     System.out.println("City:"+city);
     System.out.println("PinCode:"+pinCode);
        }
}
class EmployeeContact
{
        long phoneNo;
        String mailId;
   void getEmployeeContact()
        {
                System.out.println("=====EmployeeContact=====");
                System.out.println("PhoneNo:"+phoneNo);
     System.out.println("MailId:"+mailId);
        }
}
class EmployeeMainClass //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                EmployeeData ed = new EmployeeData();
```

```
                EmployeeAddress ea = new EmployeeAddress();
                EmployeeContact ec = new EmployeeContact();
        System.out.println("Enter the EmpId:");
                ed.empId = s.nextLine();
                System.out.println("Enter the EmpName:");
                ed.empName = s.nextLine();
                System.out.println("Enter the EmpDesg:");
                ed.empDesg = s.nextLine();
                System.out.println("Enter the EmpBSal:");
                ed.empBSal = Integer.parseInt(s.nextLine());
        System.out.println("Enter the HNo:");
                ea.hNo = s.nextLine();
                System.out.println("Enter the StreetName:");
                ea.sName = s.nextLine();
                System.out.println("Enter the City:");
                ea.city = s.nextLine();
                System.out.println("Enter the PinCode:");
                ea.pinCode = Integer.parseInt(s.nextLine());
        System.out.println("Enter the PhoneNo:");
                ec.phoneNo = Long.parseLong(s.nextLine());
                System.out.println("Enter the MailId:");
                ec.mailId = s.nextLine();

                ed.getEmployeeData();
                ea.getEmployeeAddress();
                ec.getEmployeeContact();

            }
}
```

Assignment1:

Update Student details program by reading and displaying the details?

Assignment2:

Update Customer details program by reading and displaying the details?

==========

**imp*

Operators in Java:

=>Operator is a special symbol or keyword used to perform operations.

=>The following are some important operators used in Java:

1. *Arithmetic Operators*
2. *Relational Operators*
3. *Logical Operators*
4. *Increment/Decrement Operators*

1. *Arithmetic Operators:*

=>The operators which perform basic operations are known as Arithmetic operators.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplicatioon
/	Division
%	ModDivision

2. *Relational Operators:*

=>The operators which perform comparisons and generate the boolean results.

Operator	Meaning
>	Greater than
>=	Greater than or equal
<	Less than

<= Less than or equal

== Is equal to

!= Not Equal to

3. Logical Operators:

=>The operators which are used to compare two comparisons and generate boolean result.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Logical AND(&&):

A	B	A&&B
T	T	T
F	T	F
T	F	F
F	F	F

Logical OR(||):

A	B	A||B
T	T	T
F	T	T
T	F	T
F	F	F

Logical NOT(!):

A	!A
T	F

F T

4. Increment/Decrement Operators:

=>Increment operator is used to increment the value by 1 and decrement operator is used to decrement the value by 1.

Operator	Meaning
++	Increment
--	Decrement

==========

Dt: 25/9/2021

**imp*

Control Structures in Java:

=>The part of the program which is controlled by the programmer is known as Control Structure.

=>Control Structures are categorized into three types:

1.Selection Statements

2.Iterative Statements

3.Branching Statements

1.Selection Statements:

=>The statements which are used to select one part of the program for execution based on condition are known as Selection Statements or Conditional Statements.

Types:

(a)simple if

(b)if-else

(c)Nested if

(d)Ladder if-else

(e)switch-case

2.Iterative Statements:

=>The statements which are used to execute some lines of the program repeatedly on some condition are

known as Iterative statements or Repeatitive statements or Looping Structures.

Types:

(a)while loop

(b)do-while loop

(c)for loop

3.Branching Statements:

=>The statements which are used to transfer the excution control from one location to another location

are known as Branching Statements or Transfer Statements.

Types:

(a)break

(b)continue

(c)exit

(d)return

Note:

=>'goto' statement is not available in Java.

==========

Exp program:

wap to read employee bSal and calculate totSal?

totSal = bSal+HRA+DA;

HRA = 93% of bSal

DA = 63% of bSal

Note:

=>bSal of an employee must be min 5000/-

```
import java.util.Scanner;
```

```
class EmployeeSalary //SubClass
{
        int bSal;
        float totSal;
        void calculate()
        {
                totSal = bSal+(0.93F*bSal)+(0.63F*bSal);
        }
        void getEmployeeSalary()
        {
                System.out.println("=====EmployeeSalary=====");
                System.out.println("BSal:"+bSal);
                System.out.println("TotSal:"+totSal);
        }
}
class EMainClass //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                EmployeeSalary es = new EmployeeSalary();
                System.out.println("Enter the bSal:");
                es.bSal = s.nextInt();
                if(es.bSal>=5000)
                {
                        es.calculate();
                        es.getEmployeeSalary();
                }//end of if
                else
                {
```

```
                    System.out.println("Invalid bSal...");
                }
            }
}
```

o/p:

Enter the bSal:

12000

=====EmployeeSalary=====

BSal:12000

TotSal:30720.0

==========

Ex program:

wap to read six sub marks and calculate the following:

=>totMarks

=>percentage

=>result

```
import java.util.Scanner;
class StudentResult //SubClass
{
	int s1,s2,s3,s4,s5,s6,totMarks;
	float per;
	String result;
	void calculate()
	{
		totMarks = s1+s2+s3+s4+s5+s6;
		per = (float)totMarks/6;//TypeCasting
		if(per>=70 && per<=100)
		{
			result="Destinction";
		}
		else if(per>=60 && per<70)
		{
			result="FirstClass";
		}
		else if(per>=50 && per<60)
		{
			result="SecondClass";
		}
		else if (per>=35 && per<50)
		{
			result="ThirdClass";
		}
		else{
			result="Fail";
```

```
                }
        }
        void getStudentResult()
        {
                System.out.println("====StudentResult====");
                System.out.println("Total Marks:"+totMarks);
    System.out.println("Percentage:"+per);
    System.out.println("Result:"+result);
        }
}
class SMainClass //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                StudentResult sr = new StudentResult();
                System.out.println("Enter the marks of Sub1:");
    sr.s1 = s.nextInt();
                System.out.println("Enter the marks of Sub2:");
    sr.s2 = s.nextInt();
                System.out.println("Enter the marks of Sub3:");
    sr.s3 = s.nextInt();
                System.out.println("Enter the marks of Sub4:");
    sr.s4 = s.nextInt();
                System.out.println("Enter the marks of Sub5:");
    sr.s5 = s.nextInt();
                System.out.println("Enter the marks of Sub6:");
    sr.s6 = s.nextInt();
                sr.calculate();
                sr.getStudentResult();
```

```
        }
}
```

==========

Assignment:

Update above program as follows:

(i)If Sub marks are entered in b/w 0 to 100 then calculate totMarks,per and result,if not display Invalid marks.

(ii)If any Sub marks entered in b/w 0 to 34 then display the result as 'Fail'.

==========

Dt : 27/9/2021

Assignment: Solution

Updated SMainClass.java

```
import java.util.Scanner;
class StudentResult //SubClass
{
        int s1,s2,s3,s4,s5,s6,totMarks,p=0;
        float per;
        String result;
        void calculate()
        {
                totMarks = s1+s2+s3+s4+s5+s6;
                per = (float)totMarks/6;//TypeCasting
                if(p==1)
                {
        result="Fail";
                }
                else if(per>=70 && per<=100)
```

```
                {
                        result="Destinction";
                }
                else if(per>=60 && per<70)
                {
                        result="FirstClass";
                }
                else if(per>=50 && per<60)
                {
                        result="SecondClass";
                }
                else if (per>=35 && per<50)
                {
                        result="ThirdClass";
                }
                else{
                        result="Fail";
                }
        }
        void getStudentResult()
        {
                System.out.println("====StudentResult====");
                System.out.println("Total Marks:"+totMarks);
    System.out.println("Percentage:"+per);
    System.out.println("Result:"+result);
        }
}
class SMainClass //MainClass
{
        public static void main(String[] args)
```

```
        {
                Scanner s = new Scanner(System.in);
                StudentResult sr = new StudentResult();
                System.out.println("Enter the marks of Sub1:");
    sr.s1 = s.nextInt();
                System.out.println("Enter the marks of Sub2:");
    sr.s2 = s.nextInt();
                System.out.println("Enter the marks of Sub3:");
    sr.s3 = s.nextInt();
                System.out.println("Enter the marks of Sub4:");
    sr.s4 = s.nextInt();
                System.out.println("Enter the marks of Sub5:");
    sr.s5 = s.nextInt();
                System.out.println("Enter the marks of Sub6:");
    sr.s6 = s.nextInt();
                if((sr.s1>=0 && sr.s1<=100)&&(sr.s2>=0 && sr.s2<=100)&&(sr.s3>=0 &&
sr.s3<=100)
                        &&(sr.s4>=0 && sr.s4<=100)&&(sr.s5>=0 && sr.s5<=100)&&(sr.s6>=0 &&
sr.s6<=100))
                {
                        if(sr.s1<=34 || sr.s2<=34 || sr.s3<=34 || sr.s4<=34 || sr.s5<=34 ||
sr.s6<=34)
                        {
                                sr.p=1;
                        }
      sr.calculate();
                  sr.getStudentResult();
                }//end of if
                else
                {
                        System.out.println("Invalid SubMarks...");
```

```
                    }

            }
}
```

==========

**imp*

Variables in Java:

=>Variables are the data holders in the program.

=>Variables in Java are categorized into two types.

1.Primitive DataType variables

2.NonPrimitive DataType variables(Reference Variables)

1.Primitive DataType variables:

=>The variables which are declared with Primitive datatypes like byte,short,int,long,float,double, char and boolean are known as Primitive datatype variables.

Note:

=>Primitive datatype variables will hold values.

2.NonPrimitive DataType variables(Reference Variables):

=>The variables which are declared with NonPrimitive datatypes like Class,Interface,Array and Enum are known as NonPrimitive DataType variables or Reference variables.

Note:

=>NonPrimitive datatype variables will hold Object references.

=>Based on 'static' keyword the variables are categorized into two types:

1.Static Variables

2.NonStatic Variables

1.Static Variables:

=>The variables which are declared with 'static' keyword are known as static variables or Class Variables.

=>These static variables will get the memory within the class while class_loading.

=>These Static variables can be accessed with Class_name.

2.NonStatic Variables:

=>The variables which are declared without static keyword are known as NonStatic variables.

=>These NonStatic variables are categorized into two types:

(a)Instance Variables

(b)Local Variables

(a)Instance Variables:

=>The NonStatic variables which are declared outside the methods are known as Instance variables.

=>These Instance variables will get the memory within the object while object creation.

=>These Instance variables can be accessed with object_name

(b)Local Variables:

=>The NonStatic variables which are declared inside the methods are known as Local variables.

=>Local variables will get the memory within the method or Method Frame,while method execution.

Dt : 28/9/2021

**imp*

Methods in Java:

=>Methods are the actions performed to generate results.

=>Methods in Java are categorized into two types:

1.static methods

2.NonStatic methods or Instance methods

1.static methods:

=>The methods which are declared with static keyword are known as static method.

=>These static methods will get the memory within the class while class loading.

=>These static methods are accessed with Class_names.

Structure of static methods:

static return_type method_name(para_list)

{

//method_body

}

Rule:

=>Static methods can access only static variables directly,but cannot access Instance variables.

=>static methods are categorized into two types:

(i)Built-In static methods

(ii)User defined Static methods

(i)Built-In static methods:

=>The static methods which are available from JavaLib are known as Built-in Static methods or PreDefined Static methods.

(ii)User defined Static methods:

=>The static methods which are defined by the programmer are known as User defined static methods.

2.NonStatic methods or Instance methods:

=>The methods which are declared without static keyword are known as NonStatic methods or Instance methods.

=>These NonStatic methods will get the memory within the object while object creation.

=>These NonStatic methods are accessed with Object_name.

Structure of Instance method:

return_type method_name(para_list)

{

```
 //method_body
}
```

Rule:

=>Instance methods can access both static and Instance variables directly.

=>These Instance methods are categorized into two types:

(i)Built-In Instance methods

(ii)User defined Instance methods

(i)Built-In Instance methods:

=>The Instance methods which are available from JavaLib are known Built-In instance methods or

PreDefined Instance methods.

(ii)User defined Instance methods:

=>The Instance methods which are defined by the programmer are known as User defined Instance

methods.

==========

Note:

=>Static variables and Instance variables will get default values,but Local variables will not get

default values,in this proces we have to assign values to Local variables.

==========

ex program:DemoVariables1.java

```
import java.util.Scanner;
class Display //SubClass
{
        static int a;//Class Variable
        int b;//Instance Variable
        void dis1()//NonStatic
```

```
        {
                int c=10;//Local Variable
                System.out.println("======Instance method dis1()====");
                System.out.println("The value a:"+a);
                System.out.println("The value b:"+b);
                System.out.println("The value c:"+c);
        }
        static void dis2()
        {
                int d=20;//Local variable
                System.out.println("=====static method dis2()====");
                System.out.println("The value a:"+a);
                //System.out.println("The value b:"+b);//Compilation_Error
                System.out.println("The value d:"+d);
        }
}
class DemoVariables1 //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                System.out.println("Enter the value of a:");
                Display.a = s.nextInt();
                System.out.println("The value a:"+Display.a);

                Display d = new Display();
                System.out.println("Enter the value of b:");
                d.b = s.nextInt();
                System.out.println("The value b:"+d.b);

                Display.dis2();//Static method_call
```

```
                d.dis1();//Instance method_call
        }
}
```

o/p:

Enter the value of a:

12

The value a:12

Enter the value of b:

13

The value b:13

=====static method dis2()====

The value a:12

The value d:20

======Instance method dis1()====

The value a:12

The value b:13

The value c:10

***imp**

JVM Architecture:(JVM Internals)

=>JVM Stands for Java Virtual Machine and which is used to execute Java ByteCode.

=>The s/w component which internally having the behaviour like machine is known as Virtual Machine.

=>This JVM internally having the following three partitions:

1.Class Loader SubSystem

2.Runtime DataArea

3.Execution Engine

Dt : 29/9/2021

1.Class Loader SubSystem:

=>Class Loader SubSystem will load the class file(ByteCode) on to JVM,in this process it uses the following:

(a)Loader

(b)Linker

(c)Initiate

(a)Loader:

=>The loader will load the required files into Current running program.

(b)Linker:

=>Linker will link the loaded files into Current running program.

(c)Initiate:

=>Initiate Component will perform initialization process,which means specify the Execution control

to start the execution.

2.Runtime DataArea:

=>Runtime Data Area internally divided into the following blocks:

(a)Method Area

(b)Heap Area

(c)Java Stack Area

(d)PC Register Area

(e)Native Method Area

(a)Method Area:

=>The Memory block where the class is loaded,is known as Method Area.

=>while class loading static members will get the memory within the class.

=>In this process main() method will get the memory within the MainClass and this main() method

automatically copied onto JavaStackArea to start the execution process.

(b)Heap Area:

=>The memory block where the objects are created is known as HeapArea.

(c)Java Stack Area:

=>The memory block where the methods are executed is known as Java Stack Area.

=>main() method is the first method copied onto JavaStackArea and this main() method will call

remaining methods for execution.

Diagram :

---

faq:

define Method Frame?

=>The partition of Java Stack Area where the method is copied for execution is known as Method Frame.

=>After Method Execution completed,the method frame will be destroyed automatically.

Note:

=>MainClass is loaded onto Method Area first and the remaining SubClasses are loaded based on their

requirement in execution process.

---

faq:

scope of Variables:

=>Static variables are accessed by both static methods and Instance methods.

=>Instance variables are accessed by only Instance methods of same object.

=>Local variables are accessed only inside the method.

---

faq:

Life-Time of Variables:

=>Static Variables are available until class is available on Method Area.

=>Instance Variables are available until Object is available om Heap Area.

=>Local variables are available until the method execution.

Dt : 30/9/2021

**imp*

define parameters?

=>parameters are the variables which are used to transfer the data from one method to another method.

=>Based on parameters the methods are categorized into two types:

1.Methods without parameters

2.Methods with parameters

1.Methods without parameters:

=>The methods which are declared without parameters are known as 0-parameter methods or methods

without parameters.

2.Methods with parameters:

=>The methods which are declared with parameters are known as parameterized methods or methods with

parameters.

----

faq:

define return_type?

=>return_type specify the methods will return the value after method execution or not.

=>Based on return_type the methods are categorized into two types:

1.Non-Return_type methods

2.Return_type methods

1.Non-Return_type methods:

=>The methods will not return any value after method execution are known as Non-Return_type methods.

Note:

=>The methods which are declared with 'void' are known as Non-Return_type methods.

(void methods are Non-Return_type methods)

2.Return_type methods:

=>The methods which return the value after method execution are known as Return_type methods.

Note:

=>we use 'return' statement to return the value after method execution.

=>The returned value will come back to the method_call.

=>The methods which are declared without 'void' are known as Return_type methods.

(Non-void methods are return_type methods)

Ex program:

wap to read two int values and perform arithmetic operation based on User choice?

```
import java.util.Scanner;

class Addition

{

        int add(int x,int y)

        {

                return x+y;

        }

}

class Subtraction

{

        int sub(int x,int y)

        {

                return x-y;
```

```
        }
}
class Multiplication
{
        int mul(int x,int y)
        {
                return x*y;
        }
}
class Division
{
   float div(int x,int y)
        {
                return (float)x/y;
        }
}
class ModDivision
{
        int modDiv(int x,int y)
        {
                return x%y;
        }
}
class DemoMethods1 //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                System.out.println("Enter the int value1:");
                int v1 = s.nextInt();
```

```
System.out.println("Enter the int value2:");
            int v2 = s.nextInt();
            System.out.println("=====Choice====");
            System.out.println("1.add\n2.sub\n3.mul\n4.div\n5.modDiv");
            System.out.println("Enter your Choice:");
            int choice = s.nextInt();
switch(choice)
            {
                    case 1:
    Addition ad = new Addition();
                      System.out.println("Sum:"+ad.add(v1,v2));
                            break;
                    case 2:
                            Subtraction sb = new Subtraction();
                      System.out.println("Sub:"+sb.sub(v1,v2));
                            break;
                    case 3:
                            Multiplication ml = new Multiplication();
                      System.out.println("Mul:"+ml.mul(v1,v2));
                            break;
                    case 4:
                            Division dv = new Division();
                      System.out.println("Div:"+dv.div(v1,v2));
                            break;
                    case 5:
                            ModDivision md = new ModDivision();
                      System.out.println("ModDiv:"+md.modDiv(v1,v2));
                            break;
  default:
                            System.out.println("Invalid Choice....");
```

```
                }//end of switch
        }
}
```

Execution flow of above program:

ClassFiles:

Addition.class

Subtraction.class

Multiplication.class

Division.class

ModDivision.class

DemoMethods1.class(MainClass)

Note:

=>v1,v2,x and y are known as parameters,becuase they are used to transfer the data from one method to

another method.

=>v1 and v2 are known as actual parameters,because they hold original input data.

=>x and y are known as formal parameters,because they hold intermediate data for calculations.

=>we can have same parameter names in actual parameters and formal parameters.

---

define switch-case statement?

=>switch-case is used when we want to select one from multiple available options or cases.

syntax:

```
switch(value)
{
  case 1: statements;
  break;
  case 2: statements;
  break;
  .
  .
  case n: statements;
  break;
  default: statements;
}
```

Execution Behaviour:

=>The switch-value is compared with available options or cases,if switch-value is matched with available option then statements are executed.

=>The break statement stops the execution of switch-case.

=>If the switch-value is not matched with any available options then default is executed.

Assignment:

wap to verify the StruBranch and then read rollNo?

(i)Read stuBranch

=>Verify the branch is in "CSE" or "EEE" or "ECE",if not display "invalid branch".

(ii)If the branch is verified then read stuRollNo.

o/P

branch :

rollNo :

==========

Dt : 1/10/2021

Updated Student Details program:

program : DemoMethods2.java

```
import java.util.Scanner;
class CheckBranch //SubClass
{
	boolean k=false;
	boolean verify(String br)
	{
		switch(br)
		{
			case "CSE":k=true;
				break;
			case "ECE":k=true;
				break;
			case "EEE":k=true;
				break;
		}//end of switch
		return k;
	}
}
class StudentResult //SubClass
{
	String result;
```

```
float per;
void calculate(int totMarks,boolean p)
{
        per = (float)totMarks/6;
        if(p)
        {
                result="Fail";
        }
        else if(per>=70 && per<=100)
        {
                result="destinction";
        }
        else if(per>=60 && per<70)
        {
                result="firstClass";
        }
        else if(per>=50 && per<60)
        {
                result="secondClass";
        }
        else if(per>=35 && per<50)
        {
                result="ThirdClass";
        }
        else{
                result="Fail";
        }
}
void getStudentResult()
{
```

```
                System.out.println("Per:"+per);
                System.out.println("Result:"+result);
        }
}
class DemoMethods2 //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                System.out.println("Enter the Student Name:");
                String name = s.nextLine();
                System.out.println("Enter the Student Branch:");
                String branch = s.nextLine().toUpperCase();
                CheckBranch cb = new CheckBranch();
    boolean z=cb.verify(branch);
                if(z)
                {
                        System.out.println("Enter the Student rollNo..");
                        String rollNo = s.nextLine();
                        if(rollNo.length()==10)//Nested if or Inner if
                        {
                                System.out.println("====Enter 6 Sub marks===");
                                int i=1,totMarks=0;
                                boolean p=false;
                                while(i<=6)
                                {
                                        System.out.println("Enter the marks of sub"+i);
                                        int sub = s.nextInt();
                                        i++;
                                        if(sub<0 || sub>100)
```

```
                    {
                        System.out.println("Invalid marks...");
                        i--;
                        continue;//skip the below lines within the loop
                    }
                    if(sub>=0 && sub<=34)
                    {
                        p=true;
                    }
                    totMarks = totMarks+sub;
                }//end of loop
                System.out.println("stu Name:"+name);
                System.out.println("Stu Branch:"+branch);
            System.out.println("Stu RollNo:"+rollNo);
                System.out.println("TotMarks:"+totMarks);
                StudentResult sr = new StudentResult();
                sr.calculate(totMarks,p);
                sr.getStudentResult();
            }//end of if
            else
            {
                System.out.println("Invalid rollNo...");
            }
        }//end of if
        else
        {
            System.out.println("Invalid branch...");
        }
    }
}
```

==========

faq:

define while loop?

=> In while looping Structure the condition is checked first,if the condition is true then the loop block is executed.This process is repeated until the contition is false.

syntax:

```
while(contition)
{
  //loop_block
}
```

FlowChart:

faq:

define do-while loop?

=>In do-while loop the condition is checked after loop_block execution.This process is repeated until the condition is false.

syntax:

```
do
{
 //loop_block
}
while(condition);
```

FlowChart:

---

faq:

define for loop?

=>'for' loop is more simple in representation when compared to while and do-while loops,becuase

Initialization,Condition and Incre/Decre declared in the same line.

syntax:

```
for(Initialization;Condition;Incre/Decre)
{
  //Loop_block
}
```

FlowChart:

Execution behaviour of 'for' loop:

step-1 : Initialization process(Initialize the loop variable)

step-2 : Check the condition

step-3 : If the condition is true,then the loop_block is executed.

step-4 : perform Incre/Decre after Loop_block execution

=>repeat the steps 2,3 and 4 until the condition is false.

==========

Dt : 2/10/2021

Execution flow of above program:(Student Details)

ClassFiles:

CheckBranch.class

StudentResult.class

DemoMethods2.class(MainClass)

---

define continue statement?

=>'continue' statement is used to skip the lines from the Iteration.

=>'continue' statement must be declared inside the condition.

=>The lines declared after the continue statement will be skipped from the Iteration.

---

define toUpperCase() method?

=>toUpperCase() method is used to convert the given String into UpperCase.

define length() method?

=>length() method is used to find the length of String.

Assignment:

wap to perform Bank Transaction process?

Diagram:

Dt : 4/9/2021

Assignment:(Solution)

```
import java.util.Scanner;
class CheckPinNo //SubClass
{
        boolean k=false;
        boolean verify(int pinNo)
        {
                switch(pinNo)
                {
                        case 1111:k=true;
                                break;
                        case 2222:k=true;
                                break;
                        case 3333:k=true;
                                break;
                }//end switch
                return k;
        }
```

```
}
class WithDraw //SubClass
{
        void wDraw(int amt)
        {
                System.out.println("Amt withDrawn:"+amt);
                System.out.println("Balance Amt:"+(2000-amt));
                System.out.println("Transaction completed...");
        }
}
class Deposit //SubClass
{
        void deposit(int amt)
        {
                System.out.println("Amt Deposited:"+amt);
                System.out.println("Balance Amt:"+(2000+amt));
                System.out.println("Transaction completed...");
        }
}
class DemoMethods3 //MainClass
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                int count=0;
                xyz:
                while(true){
                  System.out.println("Enther the pinNo:");
                  int pinNo = s.nextInt();
                  CheckPinNo cpn = new CheckPinNo();
```

```
boolean z = cpn.verify(pinNo);
if(z)
{
        System.out.println("====Choice====");
        System.out.println("1.WithDraw\n2.Deposit");
        System.out.println("Enter the Choice:");
        int choice = s.nextInt();
        switch(choice)
        {
                case 1:
                        System.out.println("Enter the amt:");
                  int a1 = s.nextInt();
                        if(a1>0 && a1%100==0)
                  {
                                if(a1<=2000)
                                {
                                        WithDraw wd = new WithDraw();
                                        wd.wDraw(a1);
                                }//end of if
                                else
                                {
                                         System.out.println("InSufficient fund...");
                                }
                  }//end of if
                         else
                  {
                                System.out.println("Invalid amt...");
                  }
                        break xyz;
                case 2:
```

```
                    System.out.println("Enter the amt:");
                int a2 = s.nextInt();
                    if(a2>0 && a2%100==0)
                {
                        Deposit dp = new Deposit();
                        dp.deposit(a2);
                }//end of if
                    else
                {
                        System.out.println("Invalid amt...");
                }
                    break xyz;
            default:
                    System.out.println("Invalid choice....");
            break xyz;
        }//end switch
    }//end of if
    else
    {
        System.out.println("Invalid PinNo...");
        count++;
    }
    if(count==3)
        {
        System.out.println("Sorry ! Transaction Blocked....");
        break xyz;//steop the loop
        }
  }//end of while
}
}
```

==========

Note:

=>we can declare a label for while loop and this while loop can be controlled based on label.

=>'break' is used to stop the iterative statements.(while and for)

==========

Note:

=>In realtime do-while loop is less used when compared to while loop,because do-while loop will execute the loop-block once for the false condition and waste the execution time and degrades the performance of an application.

==========

Note:

=>In realtime for loop is used in 'Strings' and 'Arrays',because these concepts will use index values.

**imp*

define String?

=>The sequenced collection of characters which are represented in double quotes is known as String.

ex:

"nit","java",...

=>The characters in the String are organized based on index values.

```
import java.util.Scanner;

class DString1 //MainClass

{

        public static void main(String[] args)

        {

                Scanner s = new Scanner(System.in);

                System.out.println("Enter the String:");
```

```
            String str = s.nextLine();
            System.out.println("str:"+str);
            int len = str.length();
            System.out.println("length of str:"+len);
            System.out.println("====Reverse of String===");
            for(int i=len-1;i>=0;i--)
            {
                System.out.print(str.charAt(i));
            }//end of loop
        }
}
```

o/p:

Enter the String:

java language programming

str:java language programming

length of str:25

====Reverse of String===

gnimmargorp egaugnal avaj

Assignment:

wap to read a String and check the String is palindrome String or not?

Dt : 5/10/2021

Exp program:

wap to read a String and display the count of Vowels from the given String?

```
import java.util.Scanner;
class DString2
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                System.out.println("Enter the String:");
                String str = s.nextLine();
                int len = str.length();
```

```
System.out.println("str:"+str);
System.out.println("length of str:"+len);
int count=0;
for(int i=0;i<=len-1;i++)
{
        char ch = str.charAt(i);//retrive char by index value
        switch(ch)
        {
                case 'a':
                case 'A':count++;
                break;
                case 'e':
case 'E':count++;
                break;
                case 'i':
case 'I':count++;
                break;
                case 'o':
case 'O':count++;
                break;
                case 'u':
case 'U':count++;
break;
        }//end of switch
}//end of loop
System.out.println("Number of Vowels:"+count);
}
}
```

o/p:

Enter the String:

java language programming

str:java language programming

length of str:25

Number of Vowels:9

----

faq:

define charAt() method?

=>charAt() is a built-in method from String class and which is used to retrieve character from the String based on index value.

Method Signature:

public char charAt(int);

syntax:

char ch = str.charAt(index);

---

faq:

define Array?

=>Array is a Sequenced collection of elements of same data type.

(or)

=>Array is a Sequenced collection of Similer datatype elements.

syntax:

```
data_type arr_var[] = new data_type[size];
```

ex program:

```
import java.util.Scanner;
class DArray1
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
    System.out.println("Enter the Size of Array:");
                int size = s.nextInt();
                int a[] = new int[size];//Array to hold int values
                System.out.println("Enter "+size+" elements:");
                for(int i=0;i<size;i++)
                {
                        a[i]=s.nextInt();

                }//end of loop
                int len = a.length;//finding the length of array
                System.out.println("length of array:"+len);
                System.out.println("====Display Array using Old for loop====");
                for(int i=0;i<=len-1;i++)
```

```
                {
                        System.out.print(a[i]+" ");
                }
    System.out.println("\n====Display Array in reverse using Old for loop====");
                for(int i=len-1;i>=0;i--)
                {
                        System.out.print(a[i]+" ");
                }
     System.out.println("\n====Display using Extended for(Java5)=====");
            for(int k : a)
                {
                  System.out.print(k+" ");
                }//end of loop
          }
}
```

o/p:

Enter the Size of Array:

10

Enter 10 elements:

1

2

3

4

5

6

7

8

9

10

length of array:10

====Display Array using Old for loop====

1 2 3 4 5 6 7 8 9 10

====Display Array in reverse using Old for loop====

10 9 8 7 6 5 4 3 2 1

====Display using Extended for(Java5)=====

1 2 3 4 5 6 7 8 9 10

----

diagram:

Assignent1:

Update above program with double data.(read abd display double data)

Assignment2:

Updtae above program with String data.(read and display String data)

----

faq:

wt is the diff b/w

(i)length()

(ii)length

=>length() method is used to find the length of Strings.

=>'length' keyword is used to find the length of Arrays.

----

faq:

define 'Extended for' ?

=>'Extended for' introduced by Java5(2004) version and which is used to retrieve elements based on

Container_name.

=>'Entended for' is a auto retrieval loop and no initialization,no condition and no incre/Decre.

syntax:

```
for(data_type var : Container_name)
{
  //loop_block;
}
```

Summary od Control Structures in Java:

1.Selection Statements:

(a)simple if

=>if block without else block

(b)if-else

=>if and else blocks

(c)Nested if

=>declaring if inside the if

(d)Ladder if-else

=>More number of if-else blocks

(e)switch-case

=>selecting one from Multiple available options or cases.

2.Iterative Statements:

(a)while loop

=>check the condition and then execute the loop-block,repeat the process until the condition is

false

(b)do-while loop

=>execute loop-block and then check the condition,repeat the process until the condition is

false

(c)for loop

=>In initialization,condition and Incre/decre declared in the same line

3.Branching Statements:

(a)break

=>break is used to stop the switch-cases and Iterative statements

(b)continue

=>skip the lines from the iteration

(c)exit

=>exit is used to stop the program.(Terminating program)

(d)return

=>return will transfer the value from one method to another after method execution.

==========

==========

Dt : 6/10/2021

(d)PC Register Area:

=>Program Counter(PC) Registers will record the status of method executions in JavaStackArea.

=>Every method which is executing in JavaStackArea will have its own Program Counter Register.

=>All these Program Counter Registers are opened in a separate memory block known as PC-Register Area.

(e)Native Method Area:

=>The methods which are declared with 'native' keyword in JavaLib are known as Native methods.

=>These native methods internally having c or c++ code.

=>when these methods are used in Java Program,then they are separated and loaded on to separate memory

block known as Native method Area

=>These Native methods are executed using JNI(Java Native method Interface)

=>while execution JNI uses Native method Libraries.

3.Execution Engine:

=>ExecutionEngine is an Executor of JVM and which starts the execution process with main() method

available in JavaStackArea.

=>This ExecutionEngine internally having two translators:

(i)Interpreter

(ii)JIT(Just-In-Time) Compiler

(i)Interpreter:

=>Interpreter will start the execution process and executes normal Instructions,while execution

if interpreter finds Stream data(Multi-Media data) then transfers the control to JIT-Compiler.

(ii)JIT(Just-In-Time) Compiler:

=>JIT-Compiler will execute Stream Instructions,which means Multi-Media instructions or

Bulk-Instructions.

---

faq:

why Java uses Interpreter in Execution process?

=>when we have Interpreter in execution process then we can accept the request in the middle of

execution process.

=>which is preferable in Network Applications or Internet Applications

==========

*imp

Blocks in Java:

=>The set-of-statements which are declared within the flower brackets({}) and executed automatically

is known as 'block'.

=>Blocks in Java are categorized into two types:

1.Static block

2.NonStatic blocks(Instance block)

1.Static block:

=>The block which is declared with 'static' keyword is known as 'static' block

syntax:

```
static
{
 //set-of-statements;
}
```

Execution behaviour of 'static' block:

=>'static' block is executed automatically while class loading.

=>'static' block executes only once,because the class is loaded onto MethodArea only once.

2.NonStatic blocks(Instance block):

=>The blocks which are declared without 'static' keyword are known as NonStatic blocks or Instance

blocks.

syntax:

```
{
 //set-of-statements;
}
```

Execution behaviour of 'Instance block':

=>Instance blocks are executed automatically while object creation.

=>Instance blocks are executed for all the multiple object creations.

Dt : 7/10/2021

ex program : DBlocks1.java

```
class Test1 //SubClass
{
        int a=10;//Instance variable
        static int b=20;//static variable
        static
        {
                System.out.println("SubClass Static block");
                //System.out.println("The value a:"+a);//Compilation_Error
                System.out.println("The value b:"+b);
        }
}
class DBlocks1
{
        public static void main(String[] args)
        {
                Test1 ob1 = new Test1();//Object1
                Test1 ob2 = new Test1();//Object2
        }
}
```

o/p:

```
SubClass Static block
The value b:20
```

Note:

=>Static blocks can access only static variables,but cannot access Instance variables.

----

ex program : DBlocks2.java

```
class Test2 //SubClass
{
        int a=10;//Instance variable
        static int b=20;//Static varibale

        {
                System.out.println("SubClass Instance block");
                System.out.println("The value a:"+a);
                System.out.println("The value b:"+b);
        }
}
class DBlocks2 //MainClass
{
        public static void main(String[] args)
        {
                Test2 ob1 = new Test2();//Object1
                Test2 ob2 = new Test2();//Object2
        }
}
```

o/p:

```
SubClass Instance block
The value a:10
The value b:20
SubClass Instance block
The value a:10
```

The value b:20

Note:

=>Instance blocks can access both Instance variables and Static variables.

---

faq:

wt is the diff b/w

(i)Methods

(ii)Blocks

=>Methods are executed on Method_call,but Blocks are executed automatically withot Block_call.

Note:

=>In realtime Instance blocks are less used when compared to Static blocks.

=>In realtime static blocks are used in DAO(Data Access Object) layer of MVC(Model View Controller)

to hold DB Connection.

==========

**imp*

Constructors in Java:

=>Constructor is a method having the same name of the class and executed while object creation,

because the constructor call is available in Object creation syntax attached with 'new' keyword.

Note:

=>we must not use return_type for constructor,because the constructor will have Class_return_tytpe.

Constructor Structure:

```
Class_name(para_list)
{
  //method_body
}
```

=>Based on parameters the Constructors are categorized into two types:

(1)Constructors without parameters

(2)Constructors with parameters

(1)Constructors without parameters:

=>The constructors which are declared without parameters are known as 0-parameter constructors or

Constructors without parameters.

ex program: DemoCon1.java

```
class CTest //SubClass
{
        CTest()
        {
                System.out.println("Constructor CTest()");
        }
        void dis()
        {
                System.out.println("Instance method dis()");
        }
}
class DemoCon1 //MainClass
{
        public static void main(String[] args)
        {
                CTest ob = new CTest();//Con_Call
                ob.dis();//method_Call
        }
}
```

o/p:

Constructor CTest()

Instance method dis()

Execution flow of above program:

ClassFiles:

CTest.class

DemoCon1.class(MainClass)

----

faq:

wt is the diff b/w

(i)Constructor

(ii)Instance method

=>Constructor is executed while object creation,but Instance method is executed after object creatioin.

=>while Object creation the constructor is executed only once,but the instance method can be called for

execution any number of times.

faq:

wt is the diff b/w

(i)Instance block

(ii)Constructor

=>Both Components are executed while object creation,but Instance block is having highest priority in

execution than Constructor,because blocks has highest priority in execution than methods.

faq:

wt is the diff b/w

(i)static block

(ii)Constructor

=>Both components are executed only once,but Static block is executed while class loading and

Constructor is executed while object creation.

ex program:DemoCon2.java

```
class CTest2 //SubClass
{
        {
                System.out.println("Instance block..");
        }

        CTest2()
        {
                System.out.println("Constructor..");
        }

        static
        {
                System.out.println("Static block..");
        }
}
class DemoCon2 //MainClass
```

```
{
        public static void main(String[] args)
        {
                CTest2 ob = new CTest2();//Con_call
        }
}
```

o/p:

Static block..

Instance block..

Constructor..

Dt : 8/10/2021

**imp*

(2)Constructors with parameters:

=>The Constructors which are declared with parameters are known as parameterized constructors or

Constructors with parameters.

ex program:

```
import java.util.Scanner;
class User //SubClass
{
        String userName,mailId;//Instance variables memory in Object
        User(String a,String b)//a and b Local variables memory in Method Frame
        {
                userName = a;
                mailId = b;
        }
        //Loading data from Local variables to Instance variables
        public void getUser()
```

```
        {
                System.out.println("====User Details====");
                System.out.println("UserName:"+userName);
                System.out.println("MailId:"+mailId);
        }
}
class DemoCon3
{
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                System.out.println("Enter the UserName:");
                String uN = s.nextLine();
    System.out.println("Enter the MailId:");
                String mId = s.nextLine();
      //uN and mId are Local variables memory in Method Frame
    User u = new User(uN,mId);//Con_call
                u.getUser();//Method_Call
        }
}
```

o/p:

```
Enter the UserName:
nit.v
Enter the MailId:
mzu672
====User Details====
UserName:nit.v
MailId:mzu672
```

Execution flow of above program:

ClassFiles:

User.class

DemoCon3.class(MainClass)

---

faq:

wt is the advantage of Constructor?

=>Constructor is used to initialize instance variables while object creation and which saves the execution time and generates High Performance.

faq:

define 'this' keyword?

=>'this' is built-in(pre-defined) keyword holding the reference of object from where the current method or constructor is executing.

faq:

In wt situation 'this' keyword is used?

=>when we have same names in Local variables and Instance variables,then we use 'this' keyword to load the data from local variables to Instance variables.

faq:

define default Constructor?

=>The Constructor without parameters added by the compiler at compilation stage is known as

default constructor.

faq:

In wt situation default constructor is added?

=>The compiler will add default constructor,when it finds class declared without any constructors.

----

faq:

define static Constructor?

=>There is no concept of static constructor in java,because constructor means executed while object

creation attached with new keyword.(Constructor cannot be Class level component)(Compilation Error)

**imp*

packages in Java:

=>packages are collection of Classes and Interfaces.

=>Packages are categorized into two types:

1.Built-In packages(Pre-Defined packages)

2.User defined packages

1.Built-In packages(Pre-Defined packages):

=>The packages which are available from javaLib are known as Built-In packages or Pre-Defined packages.

=>The following are some important packages from JavaLib:

CoreJava:

java.lang - Language package(default package)

java.io - Input/Output package

java.util - Utility package

java.net - Networking package

GUI:

java.awt - Abstract Window Toolkit package

javax.swing - swing programming package

java.applet - Applet programming package

AdvJava:

java.sql - DataBase Connection package

javax.servlet - Servlet programming package

javax.servlet.jsp - JSP programming package

**imp*

2.User defined packages:

=>The packages which are defined by the programmer are known as User defined packages.

=>we use 'package' keyword to declare User defined packages.

syntax:

package package_name;

Dt : 9/10/2021

Example:

```
package p1;
public class Salary
{
  public void calculate(int bSal)
```

```
  {
    float totSal = bSal+(0.93F*bSal)+(0.63F*bSal);
    System.out.println("The TotSal:"+totSal);
  }
}
```

step-1 : save the program in ProjectFolder as

Salary.java

step-2 : Compile the program as follows

javac -d . Salary.java

Note:

(-d .) indicates JavaCompiler to create package and load the class file as member of the package

```
package p2;
import java.util.Scanner;
import p1.Salary;
public class EMainClass
{
  public static void main(String[] args)
  {
    Scanner s = new Scanner(System.in);
    System.out.println("Enter the BSal:");
    int bSal = s.nextInt();
    Salary sl = new Salary();
    sl.calculate(bSal);
  }
}
```

step-1 : Save the program in ProjectFolder as

EMainClass.java

step-2 : Compile the program as

javac -d . EMainClass.java

step-3 : Execute the program as follows

java p2.EMainClass

---

o/p:

E:\Demo123\DPackages>javac -d . Salary.java

E:\Demo123\DPackages>javac -d . EMainClass.java

E:\Demo123\DPackages>java p2.EMainClass

Enter the BSal:

12000

The TotSal:30720.0

Diagram:

**imp*

Creating Java Project using IDE(Eclipse):

(IDE - Integrated Development Environment)

step-1 : Download and Install IDE Eclipse

step-2 : Open IDE Eclipse,while opening name the WorkSpace(Folder) and click 'launch'

step-3 : Create Java Project

Click on File->new->Project->Java->select 'Java Project' and click 'next'->name the project and click 'finish'

step-4 : Create packages

RightClick on 'src'->new->package,name the package and click 'finish'

step-5 : Create Classes

RightClick on package->new->Class,name the class and click 'finish'

Note:

=>To increase Editor font,click on Window->Preferences->General->Appearance->Colors and Fonts->

Java->Java Editor Text Font->...

step-6 : Execute the program

Open MainClass and Click 'Run'->'Run'

==========

Dt : 11/10/2021

faq:

define 'import' statement?

=>'import' statement is used to make one class available to an other class,in this process the Classes are in different packages.

=>Importing process in Java can be done in three ways:

(i)using 'import package_name.Class_name/Interface_name;'

(ii)using 'import package_name.; '*

(iii)Using 'Fully Qualified Names'.

(i)using 'import package_name.Class_name/Interface_name;':

=>In this importing process we specify the Class_name/Interface_name to be available to current running class.

=>which is also known as 'Explicit importing process'.

ex:

```
import java.util.Scanner;
import p1.Salary;
```

(ii)using 'import package_name.; ':*

=>In this importing process all the Classes and interfaces from the package are available to Current running class.

=>which is also known as 'Implicit importing process'.

ex:

```
import java.util.*;
import p1.*;
```

(iii)Using 'Fully Qualified Names':

=>The process of declaring Classes and Interfaces with package_names part of programming code are known as 'Fully Qualified Names'.

ex:

```
java.util.Scanner s = new java.util.Scanner(System.in);
p1.Salary sl = new p1.Salary();
```

---

faq:

define 'static' import?

=>The process of declaring 'import' statement with 'static' keyword is known as 'static' import.

syntax:

```
import static package_name.Class_name/Interface_name.*;
```

Note:

=>when we use 'static import' then all the static members of Classes or Interfaces are available

to Current running program,in this process the static members can be accessed directly without Class_name or Interface_name.

faq:

define sqrt() method?

=>sqrt() is a Built-in static method from 'java.lang.Math' class and which is used to find the sqrt of given nuber.

Method Signature of sqrt() method:

public static double sqrt(double);

syntax:

double z = Math.sqrt(val);

ex program:

```
package pack;
import java.util.Scanner;
import static java.lang.Math.*;
public class DStatic {
        public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    System.out.println("Enter the value:");
    double val = s.nextDouble();
    double z = sqrt(val);
    System.out.println("Sqrt of "+val+" is "+z);
   s.close();
        }

}
```

---

faq:

define Access Modifiers in Java?

=>Access Modifiers in Java specify the scope of programming components in Project.

=>The following are some important access modifiers in Java:

(a)public

(b)private

(c)protected

(d)default

(a)public:

=>public programming components are accessed within the ProjectFolder.

(b)private:

=>private programming components are accessed only within the Class.

(c)protected:

=>protected programming components are accessed within the package.

Note:

=>These protected programming components can also be accessed by the ChildClass declared outside the

package in Inheritance process.

(d)default:

=>default programming components are accessed only inside the package.

Note:

=>The programming components which are declared without any access modifiers in Class are known

as default programming Components.

Diagram:

==========

Assignment1:

Convert 'DemoMethods1.java' into packages concept

JavaProjectName : ArithmeticOperations

packages,

test : Addition,Subtraction,Multiplication,Division.ModDivision

maccess : DemoMethods1

Assignment2:

Convert 'DemoMethods2.java' into packages concept

JavaProjectName : StudentDetails

packages,

test : CheckBranch,StudentResult

maccess : DemoMethods2

Assignment3:

Convert 'DemoMethods3.java' intp packages concept

JavaProjectName : BankTransaction

packages,

test : CheckPinNo,WithDraw,Deposit

maccess : DemoMethods3

==========

Dt : 12/10/2021

**imp*

RelationShip b/w classes:

=>The process of establishing communication b/w classes is known as 'RelationShip b/w Classes'.

=>RelationShip b/w classes are categorized into the following:

1.References

2.Inheritance

3.InnerClasses

1.References(Working with NonPrimitive DataType variables):

=>The process of declaring NonPrimitive datatype variable part of class and this NonPrimitive datatype variable will hold the Object reference of another class.

ex program:

Address.java

```
package test;
public class Address {
    public String hNo,sName,city;
    public int pinCode;
    public void getAddress()
    {

   System.out.println("====AddressDetails
====");
       System.out.println("HNo:"+hNo);
       System.out.println("SName:"+sName);
       System.out.println("City:"+city);

   System.out.println("PinCode:"+pinCode)
;
    }
}
```

Employee.java

```
package test;
public class Employee {
     public String empId,empName,empDesg;
     public Address ad = new
Address();//Object reference of Address
class
     public void getEmployee()
     {

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

System.out.println("EmpId:"+empId);
```

```
System.out.println("EmpName:"+empName);

System.out.println("EmpDesg:"+empDesg);
      }
}
```

Read.java

```
package test;
import java.util.Scanner;
public class Read {
      public void read(Scanner s,Employee
e)
      {
         System.out.println("Enter the
EmpId:");
         e.empId = s.nextLine();
         System.out.println("Enter the
EmpName:");
         e.empName = s.nextLine();
         System.out.println("Enter the
EmpDesg:");
         e.empDesg = s.nextLine();
         System.out.println("Enter the
HNo:");
         e.ad.hNo = s.nextLine();
         System.out.println("Enter the
SName:");
         e.ad.sName = s.nextLine();
         System.out.println("Enter the
City:");
         e.ad.city = s.nextLine();
         System.out.println("Enter the
PinCode:");
```

```
            e.ad.pinCode = s.nextInt();
        }
}
```

Display.java

```
package test;
public class Display {
    public void dis(Employee e) {
        e.getEmployee();
        e.ad.getAddress();
    }
}
```

DemoRef1.java

```
package maccess;

import java.util.Scanner;

import test.*;

public class DemoRef1 {

        public static void main(String[] args) {

    Scanner s = new Scanner(System.in);

    Employee e = new Employee();

    Read r = new Read();

    r.read(s, e);//Reference variables as parameters

    Display d = new Display();

    d.dis(e);//Reference as parameter

    s.close();

        }

}
```

o/p:

Enter the EmpId:

A121

Enter the EmpName:

Raj

Enter the EmpDesg:

SE

Enter the HNo:

12-34/h

Enter the SName:

SRNagar

Enter the City:

Hyd

Enter the PinCode:

12345

====EmployeeDetails====

EmpId:A121

EmpName:Raj

EmpDesg:SE

====AddressDetails====

HNo:12-34/h

SName:SRNagar

City:Hyd

PinCode:12345

Dt : 13/10/2021

Execution above program:

ClassFiles:

Address.class

Employee.class

Read.class

Display.class

DemoRef1.class(MainClass)

Note:

=>In the above diagram Employee class object is holding the reference of Address class object.

=>The methods of Employee class can access all the members of Address class using reference.

=>Address class object is created while Employee class object creation,in this process Address Class object is 'dependent object' on Employee Class object.

Assignment:

Construct Student details program using references Concept.

Diagram:

Dt: 18/10/2021

Assignment:(Solution)

Construct Student details program using references Concept.

Address.java

```
package test;
public class Address {
    public String hNo,sName,city;
    public int pinCode;
    public void getAddress()
    {

    System.out.println("====AddressDetails
====");
        System.out.println("HNo:"+hNo);
        System.out.println("SName:"+sName);
        System.out.println("City:"+city);

    System.out.println("PinCode:"+pinCode)
;
    }
}
```

StudentResult.java

```
package test;
public class StudentResult {
    public String result;
    public float per;
    public void calculate(int
totMarks,boolean p)
    {
        per = (float)totMarks/6;
        if(p)
        {
            result="Fail";
        }
        else if(per>=70 && per<=100)
        {
            result="destinction";
        }
        else if(per>=60 && per<70)
        {
            result="firstClass";
        }
        else if(per>=50 && per<60)
        {
            result="secondClass";
        }
        else if(per>=35 && per<50)
        {
            result="ThirdClass";
        }
        else{
            result="Fail";
        }
    }
    public void getStudentResult()
```

```
    {

    System.out.println("====StudentResult=
===");
        System.out.println("Per:"+per);

    System.out.println("Result:"+result);
    }

}
```

CheckBranch.java

```
package test;
public class CheckBranch {
    public boolean k=false;
    public boolean verify(String br)
    {
        switch(br)
        {
            case "CSE":k=true;
                break;
            case "ECE":k=true;
                break;
            case "EEE":k=true;
                break;
        }//end of switch
        return k;
    }
}
```

Student.java

```
package test;
public class Student {
    public String name,branch,rollNo;
```

```
    public Address ad = new Address();
    public CheckBranch cb = new
CheckBranch();
    public StudentResult sr = new
StudentResult();
    public void getStudent(){

   System.out.println("====StudentDetails
====");
       System.out.println("Name:"+name);

   System.out.println("Branch:"+branch);

   System.out.println("RollNo:"+rollNo);
    }
}
```

Read.java

```
package test;
import java.util.Scanner;
public class Read {
   public void read(Scanner s,Student
stu)
   {
      System.out.println("Enter the
StuName:");
      stu.name = s.nextLine();
      System.out.println("Enter the
Branch:");
      stu.branch =
s.nextLine().toUpperCase();
      boolean k =
stu.cb.verify(stu.branch);
      if(k)
```

```
        {
            System.out.println("Enter the
RollNo:");
            stu.rollNo = s.nextLine();
            if(stu.rollNo.length()==10)
            {
                System.out.println("Enter the
HNo:");
                stu.ad.hNo = s.nextLine();
                System.out.println("Enter the
StreetName:");
                stu.ad.sName = s.nextLine();
                System.out.println("Enter the
City:");
                stu.ad.city = s.nextLine();
                System.out.println("Enter the
PinCode:");
                stu.ad.pinCode =
Integer.parseInt(s.nextLine());
                System.out.println("====Enter
6 Subject Marks====");
                int totM=0,i=1;
                boolean p=false;
                while(i<=6) {
                    System.out.println("Enter
the Sub"+i);
                    int sub =
Integer.parseInt(s.nextLine());
                    i++;
                    if(sub<0 || sub>100)
                    {

        System.out.println("Invalid
marks...");
```

```
                    i--;
                    continue;
                }
                if(sub>=0 && sub<=34)
                {
                    p=true;
                }
               totM=totM+sub;
            }//end of loop
           stu.sr.calculate(totM, p);
        }//end of if
        else
        {
            System.out.println("Invalid
rollNo..");
            System.exit(0);
        }

    }//end of if
    else
    {
        System.out.println("Invalid
branch...");
        System.exit(0);//Terminating
program
    }
  }
}
```

Display.java

```
package test;
public class Display {
    public void dis(Student stu) {
       stu.getStudent();
```

```
        stu.ad.getAddress();
        stu.sr.getStudentResult();
    }
}
```

DemoRef2.java(MainClass)

```
package maccess;
import java.util.Scanner;
import test.*;
public class DemoRef2 {
    public static void main(String[] args)
{
        Scanner s = new Scanner(System.in);
         Student stu = new Student();
         Read r = new Read();
         r.read(s, stu);
         Display d = new Display();
         d.dis(stu);
         s.close();
    }
}
```

Diagram:

Note:

=>In the above program 'Student' class object is holding the references of following objects:

(i)'Address' Class Object reference

(ii)'StudentResult' class Object reference

(iii)'CheckBranch' class Object reference

Dt : 19/10/2021

imp

Multiple Objects holding reference of Same Object:

=>we can have Same object reference in Multiple objects,in this process the methods of Multiple objects can access the members of same object using reference.

Ex:

Product.java

```
package test;
public class Product {
   public String pCode,pName;
   public float pPrice;
   public int pQty;
   public void getProduct()
   {
       System.out.println("====ProductDeta
ils====");
       System.out.println("PCode:"+pCode);
       System.out.println("PName:"+pName);
       System.out.println("PPrice:"+pPrice
);
       System.out.println("PQty:"+pQty);
   }
}
```

Read.java

```
package test;
import java.util.Scanner;
public class Read {
    public Product p;//Instance reference
variable
    public Scanner s;//Instance reference
variable
```

```
                //Loosly Coupled reference
concept
    public Read(Scanner s,Product
p)//Constructor to initialize Instance
variables
    {
        this.s=s;
        this.p=p;
    }
     public void read()
     {
        System.out.println("Enter the
ProdCode:");
        p.pCode=s.nextLine();
        System.out.println("Enter the
ProdName:");
        p.pName=s.nextLine();
        System.out.println("Enter the
ProdPrice:");
        p.pPrice=s.nextFloat();
        System.out.println("Enter the
ProdQty:");
        p.pQty=s.nextInt();
     }
}
```

Display.java

```
package test;
public class Display {
     public Product p;//Instance
reference variable
     public Display(Product
p)//Constructor to initialize Instance
variable
```

```
        {
            this.p=p;
        }
        public void dis()
        {
            p.getProduct();
        }
}
```

DemoRef3.java(MainClass)

```
package maccess;

import java.util.*;

import test.*;

public class DemoRef3 {

        public static void main(String[] args) {

                Scanner s = new Scanner(System.in);//Scanner class object

    Product p = new Product();//Product class object

    Read r = new Read(s,p);//Con_Call

    r.read();

    Display d = new Display(p);//Con_call

    d.dis();

        }

}
```

Diagram:

Note:

=>In the above program objects of 'Read and Display' classes holding the reference of 'Product' object.

****imp***

Types of references:

=>References in Java are categorized into two Types:

(i)Tightly Coupled reference

(ii)Loosly Coupled reference

(i)Tightly Coupled reference:

=>The process of linking dependent objects is known as Tightly Coupled reference.

Ex:

DemoRef1.java and DemoRef2.java

Note:

=>when we create one object,automatically another object is created

known as Dependent objects.

=>In DemoRef1.java,Address Class object is created while creating object for Employee class.In this process Address class object is dependent object on Employee Class.

(ii)Loosly Coupled reference:

=>The process of linking Independent objects is known as Loosly Coupled reference.

Ex:

DemoRef3.java

Note:

=>when we create one object,other objects are not created are known as Independent objects.

=>In DemoRef3.java,Product class object is created independently without depending on object creation of Read and Display classes.

Note:

=>References Concept is also known as 'HAS-A' relation,because object 'HAS-A' reference of another object.

==========

**imp*

Inheritance in Java:

=>The process of linking classes with 'extends' keyword is known as Inheritance.

Diagram:

B ob = new B();

syntax:

```
class A
{
 //members
}
class B extends A
{
 //members
}
```

=>In Inheritance process we always create object for ChildClass.

B ob = new B();

=>In Inheritance process the members of ParentClass are available to ChildClass through 'extends' keyword.

Dt : 20/10/2021

Case-1 : NonStatic members from the PClass/SuperClass

Case-2 : Constructors from the PClass/SuperClass

Case-3 : Static members from the PClass/SuperClass

Case-1 : NonStatic members from the PClass/SuperClass

=>In Inheritance process one object is created and the object is holding all the NonStatic members of PClass and all the Non Static members of CClass.

ex program:

A.java

```
package test;
public class A {
    public int a;
    public void m1() {

   System.out.println("====PClass====");
      System.out.println("The value
a:"+a);
    }

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

B.java

```
package test;
public class B extends A{
     public int b;
     public void m2() {
```

```
System.out.println("====CClass====");
        System.out.println("The value
b:"+b);
      }

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

DInheritance1.java(MainClass)

```
package maccess;
import test.*;
public class DInheritance1 {
    public static void main(String[] args)
{
        B ob = new B();
        ob.a = 12;
        ob.b = 13;
        ob.m1();
        ob.m2();
    }
}
```

o/p:

PClass Instance block...

CClass Instance block

====PClass====

The value a:12

====CClass====

The value b:13

Execution flow of above program:

ClassFiles:

A.class

B.class

DInheritance1.class

Note:

=>In Inheritance process the PClass is loaded onto MethodArea first and then CClass is loaded.

=>In Inheritance process while Object creation PClass members will get the memory first and then CClass members will get the memory.

=>In Inheritance process CClass methods can access the members of PClass but,the methods of PClass cannot access the members of CClass.

Case-2 : Constructors from the PClass/SuperClass

(i)O-parameter Constructor from the PClass/SuperClass

=>when we have 0-parameter constructor from the PClass,then the Compiler at compilation stage will add 'super()' to the CClass constructor

and which is PClass Con_call.

Diagram:

ex program:

PClass.java

```
package test;
public class PClass {
    public PClass()
    {
       System.out.println("PClass
Constructor..");
    }
}
```

CClass.java

```
package test;
public class CClass extends PClass{
     public CClass() {
        System.out.println("CClass
Constructor...");
     }
```

```
}
```

DInheritance2.java(MainClass)

```
package maccess;
import test.*;
public class DInheritance2 {
    public static void main(String[] args)
{
        CClass ob = new CClass();//CClass
Con_Call
    }
}
```

o/p:

PClass Constructor..

CClass Constructor...

(ii)Parameterized Constructor from the PClass/SuperClass

=>when we have Parameterized constructor from the PClass,then we must

add 'super()' to the CClass constructor to call PClass constructor.

Diagram:

ex program:

PClass.class

```
package test;
public class PClass {
    public PClass(int x)
    {

   System.out.println("====PClass===");
       System.out.println("The value
x:"+x);
    }
}
```

CClass.class

```java
package test;
public class CClass extends PClass{
    public CClass(int y)
    {
        super(y);//PClass_Con_Call
    }
}
```

DInheritance3.java(MainClass)

```java
package maccess;
import test.*;
public class Dinheritance3 {
    public static void main(String[] args)
{
        CClass ob = new
CClass(123);//CClass Con_call
    }
}
```

o/p:

====PClass===

The value x:123

Note:

=>In Inheritance process the PClass Constructor is called by CClass

Constructor using 'super()'.

Case-3 : Static members from the PClass/SuperClass

=>In Inheritance process the static members of PClass are available to

CClass through 'extends' keyword and can be accessed with CClass_name.

ex program:

PClass.java

```
package test;
public class PClass {
   public static int a;
   public static void m1() {
       System.out.println("===PClass===");
       System.out.println("The value
a:"+a);
   }
   static
   {
       System.out.println("PClass Static
block");
   }
}
```

CClass.java

```
package test;
public class CClass extends PClass{
    public static int b;
    public static void m2() {
       System.out.println("===CClass===");
       System.out.println("The value
b:"+b);
    }
    static
    {
       System.out.println("CClass Static
block..");
    }
}
```

DInheritance4.java(MainClass)

```
package maccess;
import test.*;
```

```
public class DInheritance4 {
    public static void main(String[] args)
{
        CClass ob = new CClass();//Empty
Object
        CClass.a=12;
        CClass.b=13;
        CClass.m1();
        CClass.m2();
    }
}
```

o/p:

PClass Static block

CClass Static block..

===PClass===

The value a:12

===CClass===

The value b:13

Dt : 21/10/2021

Digram:

---

faq:

can we access static members of Class using Object reference?

=>Yes,we can access static members of class using Object reference,

because the object reference also generated from the Class.

(object reference also belongs to class)

faq:

define Empty Object?

=>The object with zero members is known as 'Empty object'.

faq:

In wt situation Empty Object is created?

=>when we create object for the class which is holding only static members

then Empty object is created.

---

faq:

define Method Overriding process?

=>The method with same method signature in PClass and CClass,then the PClass method is replaced by CClass method while object creation is known as Method Overriding process.

=>Same method Signature means,

Same return_type

Same method_name

Same para_list

Same para_type

Case-1 : Instance method Overriding process

Case-2 : Constructor Overriding process

Case-3 : Static method Overriding process

Case-1 : Instance method Overriding process:

=>The Same Instance method signature in PClass and CClass then PClass Instance method is replaced by CClass Instance method while Object Creation,is known as Instance method Overriding process.

ex program:

PClass.java

```
package test;
public class PClass {
    public void m(int x) {
        System.out.println("====PClass m()===");
        System.out.println("The value x:"+x);
    }
}
```

CClass.java

```
package test;
```

```
public class CClass extends PClass{
    public void m(int x) {
        System.out.println("====CClass
m()===");
        System.out.println("The value
x:"+x);
     }
}
```

DInheritance5.java(MainClass)

```
package maccess;
import test.*;
public class DInheritance5 {
    public static void main(String[] args)
{
        CClass ob = new CClass();
        ob.m(123);
    }
}
```

o/p:

====CClass m()===

The value x:123

Diagram:

Case-2 : Constructor Overriding process

=>There is no concept of Constructor Overriding process,because we cannot have same Constructor names in PClass and CClass.

Case-3 : Static method Overriding process

=>There is no concept of Static method Overriding process,because the Static methods are binded to classes and available within the classes.

---

faq:

define Method Hiding process?

=>The process in which the execution control cannot reach the method for execution is known as Method Hiding process.

faq:

In wt situation Method Hiding process is raised?

=>when we have same Static method signature in PClass and CClass then the execution control will execute the method from CClass,but the execution control cannot reach PClass method,known as Method Hiding process.

Dt : 22/10/2021

ex program:

PClass.java

```
package test;
public class PClass {
    public static void m(int x) {
        System.out.println("====PClass
m()====");
        System.out.println("The value
x:"+x);
    }
}
```

CClass.java

```
package test;
public class CClass extends PClass{
      /*public static void m(int x) {
         System.out.println("====CClass
m()====");
         System.out.println("The value
x:"+x);
      }*/
}
```

DInheritance6.java(MainClass)

```
package maccess;
import test.*;
public class DInheritance6 {
   public static void main(String[] args)
{
```

```
            CClass.m(123);
        }
}
```

Digram:

Conclusion:

(i)Method Overriding - Same Instance method Signatures in PClass and CClass

then the PClass method is replaced by CClass method.

(ii)Method Hiding - Same Static method Signatures in PClass and CClass

then PClass method is hided by CClass method while execution.

----

faq:

define Method Overloading process?

=>More than one method with same method_name but differentiated by their

para_list or para_type is known as Method Overloading process.

case-1 : Instance method Overloading process

Case-2 : Constructor Overloading process

case-3 : Static method Overloading process

case-1 : Instance method Overloading process:

=>More than one Instance method with same method_name but differentiated by their para_list or para_type is known Instance method Overloading process.

faq:

wt is the diff b/w

(i)super

(ii)this

=>'super' keyword is used to access the variables and methods from the PClass or SuperClass.

=>'this' keyword is used to access the variables and methods from the same class.

ex program:

PClass.java

```
package test;
public class PClass {
    public void m(float x)
    {
        System.out.println("x:"+x);
    }
}
```

CClass.java

```
package test;
public class CClass extends PClass{
     public void m(int a,int b,int
c,float x)
      {
        this.m(a,b,x);//Calling the method
from same class
        System.out.println("c:"+c);
      }
```

```
      public void m(int a,int b,float x)
      {
           this.m(a,x);//Calling the
method from Same class
          System.out.println("b:"+b);
      }
      public void m(int a,float x)
      {
         super.m(x);//PClass method call
         System.out.println("a:"+a);
      }
}
```

DInheritance7.java(MainClass)

```
package maccess;
import test.*;
public class DInheritance7 {
    public static void main(String[] args)
{
        CClass ob = new CClass();
        ob.m(1,2,3,12.34F);

    }
}
```

o/p:

x:12.34

a:1

b:2

c:3

----

faq:

define Method Chaining process?

=>The process of calling one method from another method using 'this' or

'super' keywords is known as Method Chaining process.

ex:

above program

diagram:

Case-2 : Constructor Overloading process

=>More than one constructor differentiated by their para_list or para_type is known as Constructor Overloading process.

faq:

wt is the diff b/w

(i)super()

(ii)this()

=>'super()' is used to access the constructors from the PClass or SuperClass.

=>'this()' is used to access the constructors from the Same class

ex program:

PClass.java

```
package test;
public class PClass {
      public PClass(int x) {
         System.out.println("x:"+x);
      }
}
```

CClass.java

```
package test;
public class CClass extends PClass{
    public CClass(int x,int y,int z) {
       this(x,y);//Con_call from the same
class
       System.out.println("z:"+z);
    }
    public CClass(int x,int y) {
       super(x);//Con_call from the PClass
       System.out.println("y:"+y);
    }
}
```

DInheritance8.java(mainClass)

```
package maccess;
import test.*;
public class DInheritance8 {
```

```
    public static void main(String[] args)
{
        CClass ob = new
CClass(11,12,13);//Con_call
    }
}
```

o/p:

x:11

y:12

z:13

----

faq:

define Constructor Chaining process?

=>The process of calling one constructor from another constructor using

'super()' or 'this()' is known as Constructor Chainingf process.

ex:

above program

----

case-3 : Static method Overloading process

=>More than one static method with same method_name,but differentiated by

their para_list or para_type is known as Static method Overloading process.

faq:

can we use 'super' and 'this' keywords to access static members?

=>Yes,we can access static members using 'super' and 'this' keywords,but

these keywords must be used in NonStatic methods because 'super' and 'this'

are NonStatic reference variables.

PClass.java

```
package test;
```

```
public class PClass {
   public static void m(int x) {
       System.out.println("====3rd
m()====");
       System.out.println("x:"+x);
   }
}
```

CClass.java

```
package test;
public class CClass extends PClass{
   public static void m(int x,int y,int
z) {
           System.out.println("====1st
m()====");
           System.out.println("x:"+x);
           System.out.println("y:"+y);
           System.out.println("z:"+z);

       }
   public static void m(int x,int y) {
        System.out.println("====2nd
m()====");
           System.out.println("x:"+x);
           System.out.println("y:"+y);
       }
   public void dis(int x,int y,int z) {
       this.m(x, y, z);
       this.m(x, y);
       super.m(x);
   }
}
```

DInheritance9.java(MainClass)

```
package maccess;
import test.*;
public class DInheritance9 {
    public static void main(String[] args)
{
     CClass ob = new CClass();
     ob.dis(12,13,14);
    }
}
```

o/p:

====1st m()====

x:12

y:13

z:14

====2nd m()====

x:12

y:13

====3rd m()====

x:12

==========

Dt : 23/10/2021

faq:

can we perform Overloading process for standard main() method?

=>Yes,we can perform Overloading process for standard main() method

because main() method also comes under static method category.

faq:

can we pass parameters to the standard main() method?

=>Yes,we can pass parameters to the standard main() method while

execution command,because the standard main() method call is available in

execution command.

syntax:

java Class_name para1 para2 ...

Ex:

java DInheritance10 123 345 456

Note:

=>The parameters which are passed to standard main() method are available in String Array(args) as follows:

diagram:

ex:

```
package maccess;

public class DInheritance10 {

        public static void main(String[] args) {

                DInheritance10.main(12.34F);//method_call

    System.out.println("===Standard main()====");
```

```
                for(int i=0;i<args.length;i++)
                {
                        System.out.println(args[i]);
                }
  }
        public static void main(float k) {
    System.out.println("===Overloaded main()====");
    System.out.println("The value k:"+k);
  }
}
```

o/p:

```
E:\Demo123>java DInheritance10 java A 123 12.34
===Overloaded main()====
The value k:12.34
===Standard main()====
java
A
123
12.34
```

---

faq:

define Command Line Argument program?

=>The program in which we pass parameters to the standard main() method is known as Command Line argument program.

Arguments means Values

Parameters means Variables

Types of Inheritance:

=>Inheritances are categorized into the following:

1.Single Inheritance

2.Multiple Inheritance

3.Multi-Level Inheritance

4.Hierarchal Inheritance

5.Hybrid Inheritance

Diagrams:

=>In Realtime,these inheritances are categorized into two types:

(a)Single Inheritance

(b)Multiple Inheritance

(a)Single Inheritance:

=>The process of extracting the features from one class at a time is known as Single Inheritance.

ex:

above program

(b)Multiple Inheritance:

=>The process of extracting the features from more than one class at a time is known as Multiple Inheritance.

diagram:

Note:

=>Multiple Inheritance process using classes concept is not available in java,because which generate replication of programming components and raises ambiguity.The ambiguity state applications will generate wrong results.

=>In Java,Multiple Inheritance process can be acheived using Interfaces.

Dt : 25/10/2021

**imp*

Interfaces in Java:

=>Interface is a Collection of variables and abstract methods upto Java7 version.

faq:

define abstract methods?

=>The methods which are declared without method_body are known as abstract methods.

structure of abstract method:

return_type method_name(para_list);

faq:

define Concrete methods?

=>The methods which are declared with method_body are known as Concrete methods

structure of concrete method:

```
return_type method_name(para_list)
{
  //method_body
}
```

Coding Rules of Interface:

Rule-1 : we use 'interface' keyword to declare interfaces.

structure of Interface:

```
interface Interface_name
{
  //Interface_body
}
```

Rule-2 : The programming components which are declared inside the interface are automatically 'public'.

Note:

=>The programming components which are declared inside the class without any access modifier are automatically 'default'.

Rule-3 : The interface can be declared with both Primitive DataType variables and NonPrimitive DataType variables.

Rule-4 : The variables which are declared inside the interface are

automatically static and final variables.

Note:

=>static variables in interface will get the memory within the interface and can be accessed with Interface_name.

=>final variables must be initialized with values,once initialized cannot be modified.

Rule-5 : The methods which are declared inside the interface are automatically NonStatic abstract methods.

(There is no concept of static abstract methods in Java)

Rule-6 : Interfaces cannot be instantiated because the interfaces are abstract components.

Rule-7 : These Interfaces are implemented to classes usining implements keyword and the classes are known as implementation classes.

Rule-8 : These implementation classes must construct the bodies for all the abstract methods of Interface.

Ex program:

IComparable.java

```
package test;
public interface IComparable
{
      public abstract int compare(int x,int y);
}
```

Greater.java

```
package test;
public class Greater implements IComparable{
       public int compare(int x,int y)
       {
         if(x>y) return x;
         else return y;
       }
}
```

Smaller.java

```java
package test;
public class Smaller implements IComparable{
   public int compare(int x,int y)
   {
         if(x<y) return x;
         else return y;
   }
}
```

DInterface1.java

```java
package maccess;

import test.*;

import java.util.*;

public class DInterface1 {

        public static void main(String[] args) {

    //IComparable ob = new IComparable();//Compilation_Error

                Scanner s = new Scanner(System.in);

                System.out.println("Enter the value1:");

                int v1 = s.nextInt();

                System.out.println("Enter the value2:");

                int v2 = s.nextInt();

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

                System.out.println("1.Greater\n2.Smaller");

                System.out.println("Enter the Choice:");

                int choice = s.nextInt();

                switch(choice)

                {

                case 1://Greater

                        Greater gt = new Greater();

                        System.out.println("Greater Value:"+gt.compare(v1,v2));

                        break;

                case 2://Smaller

                        Smaller sm = new Smaller();

                        System.out.println("Smaller Value:"+sm.compare(v1,v2));
```

```
                    break;
            default:
                    System.out.println("Invalid choice...");
            }//end of switch
            s.close();
        }
}
```

Dt : 26/10/2021

Diagram:

ClassFiles:

IComparable.class

Greater.class

Smaller.class

DInterface1.class(MainClass)

Assignment1:

Construct the following model to demonstrate Interface concept.

Diagram:

Assignment2:

Convert Bank Transaction process into Interface Concept as follows:

Diagram:

Rule-9 : There is no concept of declaring blocks and Conmstructors in Interfaces.

Rule-10 : Interface can have features of another interface through 'extends'

keyword.

Diagram:

Ex program:

ITest1.java

```
package test;
public interface ITest1 {
      public abstract void m1(int x);
}
```

ITest2.java

```
package test;
public interface ITest2 extends ITest1{
    public abstract void m2(int y);
}
```

IClass.java

```
package test;
public class IClass implements ITest2{
    public void m1(int x) {
      System.out.println("x:"+x);
    }
    public void m2(int y) {
      System.out.println("y:"+y);
```

```
    }
}
```

DInterface4.java(MainClass)

```
package maccess;
import test.*;
public class DInterface4 {
      public static void main(String[] args) {
       IClass ob = new IClass();
       ob.m1(123);
       ob.m2(234);
      }
}
```

Dt : 27/10/2021

Assignment1:(Solution)

Construct the following model to demonstrate Interface concept.

IArithmetic.java

```
package test;
public interface IArithmetic {
     public abstract float process(int x,int y);
}
```

Addition.java

```
package test;
public class Addition implements IArithmetic{
      public float process(int x,int y)
      {
            return x+y;
      }
}
```

Subtraction.java

```
package test;
public class Subtraction implements IArithmetic{
      public float process(int x,int y)
      {
            return x-y;
      }
}
```

Multiplication.java

```
package test;
public class Multiplication implements IArithmetic{
      public float process(int x,int y)
      {
```

```
            return x*y;
        }
}
```

Division.java

```
package test;
public class Division implements IArithmetic{
      public float process(int x,int y)
      {
      return (float)x/y;
     }
}
```

ModDivision.java

```
package test;
public class ModDivision implements IArithmetic{

      public float process(int x,int y)
      {
            return x%y;
      }
}
```

DInterface2.java(MainClass)

```
package maccess;

import java.util.Scanner;

import test.*;

public class DInterface2 {

        public static void main(String[] args)

        {

                Scanner s = new Scanner(System.in);

                System.out.println("Enter the int value1:");

                int v1 = s.nextInt();

     System.out.println("Enter the int value2:");

                int v2 = s.nextInt();

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

                System.out.println("1.add\n2.sub\n3.mul\n4.div\n5.modDiv");

                System.out.println("Enter your Choice:");

                int choice = s.nextInt();
```

```
    switch(choice)
                {
                        case 1:
        Addition ad = new Addition();
                          System.out.println("Sum:"+ad.process(v1,v2));
                                break;
                        case 2:
                                Subtraction sb = new Subtraction();
                          System.out.println("Sub:"+sb.process(v1,v2));
                                break;
                        case 3:
                                Multiplication ml = new Multiplication();
                          System.out.println("Mul:"+ml.process(v1,v2));
                                break;
                        case 4:
                                Division dv = new Division();
                          System.out.println("Div:"+dv.process(v1,v2));
                                break;
                        case 5:
                                ModDivision md = new ModDivision();
                          System.out.println("ModDiv:"+md.process(v1,v2));
                                break;
      default:
                                System.out.println("Invalid Choice....");
                }//end of switch
    s.close();
        }

}
```

Assignment2:(Solution)

Convert Bank Transaction process into Interface Concept as follows:

Balance.java

```
package test;
public class Balance {
     public double bal=2000;
     public void getBalance() {
       System.out.println("Balance Amt:"+bal);
     }
}
```

CheckPinNo.java

```
package test;
public class CheckPinNo {
      public boolean k=false;
      public boolean verify(int pinNo)
      {
            switch(pinNo)
            {
                  case 1111:k=true;
                        break;
                  case 2222:k=true;
                        break;
                  case 3333:k=true;
                        break;
            }//end switch
            return k;
      }
}
```

Transaction.java

```
package test;
public interface Transaction {
    public static final Balance b = new Balance();
    public abstract void process(int amt);
}
```

WithDraw.java

```
package test;
public class WithDraw implements Transaction{
      public void process(int amt)
      {
            if(amt<=b.bal) {
            System.out.println("Amt withDrawn:"+amt);
            b.bal=b.bal-amt;
            b.getBalance();
            System.out.println("Transaction completed...");
            }else {
                  System.out.println("Insufficient fund...");
```

```
            }
        }
}
```

Deposit.java

```
package test;
public class Deposit implements Transaction{
      public void process(int amt)
      {
            System.out.println("Amt Deposited:"+amt);
            b.bal=b.bal+amt;
            b.getBalance();
            System.out.println("Transaction completed...");
      }
}
```

DInterface3.java(MainClass)

```
package maccess;

import java.util.Scanner;

import test.*;

public class DInterface3 {

        public static void main(String[] args)

        {

                Scanner s = new Scanner(System.in);

                int count=0;

                xyz:

                while(true){

                  System.out.println("Enther the pinNo:");

                  int pinNo = s.nextInt();

                  CheckPinNo cpn = new CheckPinNo();

                  boolean z = cpn.verify(pinNo);

                  if(z)

                  {

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

                          System.out.println("1.WithDraw\n2.Deposit");

                          System.out.println("Enter the Choice:");

                          int choice = s.nextInt();
```

```
switch(choice)
{
        case 1:
                System.out.println("Enter the amt:");
          int a1 = s.nextInt();
                if(a1>0 && a1%100==0)
           {
                        WithDraw wd = new WithDraw();
                        wd.process(a1);
           }//end of if
                 else
           {
                    System.out.println("Invalid amt...");
           }
                break xyz;
        case 2:
                System.out.println("Enter the amt:");
          int a2 = s.nextInt();
                if(a2>0 && a2%100==0)
          {
                    Deposit dp = new Deposit();
                    dp.process(a2);
          }//end of if
                else
          {
                    System.out.println("Invalid amt...");
          }
                break xyz;
        default:
                System.out.println("Invalid choice....");
```

```
                    break xyz;
                }//end switch
            }//end of if
            else
            {
                System.out.println("Invalid PinNo...");
                count++;
            }
            if(count==3)
                {
                System.out.println("Sorry ! Transaction Blocked....");
                break xyz;//stop the loop
                }
        }//end of while
        s.close();
    }
}
```

==========

*imp

Multiple Inheritance Models using Interfaces:

Model-1 : Extracting the features from more than one interface into a Class.

Diagram:

Ex program:

ITest1.java

```
package test;
public interface ITest1 {
     public abstract void m1(int x);
}
```

ITest2.java

```
package test;
public interface ITest2 {
   public abstract void m2(int y);
}
```

IClass.java

```
package test;
public class IClass implements ITest1,ITest2{
      public void m1(int x) {
        System.out.println("x:"+x);
      }
      public void m2(int y) {
        System.out.println("y:"+y);
      }
}
```

MultipleInheritance1.java(MainClass)

```
package maccess;
import test.*;
public class MultipleInheritance1 {
      public static void main(String[] args) {
        IClass ob = new IClass();
        ob.m1(123);
        ob.m2(234);
      }
}
```

Model-2 : Extracting the features from one class and any number of interfaces into a class

Diagram:

Ex program:

PClass.java

```
package test;
public class PClass {
   public void m1(int x) {
         System.out.println("x:"+x);
   }
}
```

ITest.java

```
package test;
public interface ITest {
    public abstract void m2(int y);
}
```

CClass.java

```
package test;
public class CClass extends PClass implements ITest{
     public void m2(int y) {
       System.out.println("y:"+y);
     }
}
```

MultipleInheritance2.java(MainClass)

```
package maccess;
import test.*;
public class MultipleInheritance2 {
      public static void main(String[] args) {
       CClass ob = new CClass();
       ob.m1(123);
       ob.m2(234);
      }
}
```

Model-3 : Extracting the features from more than one interface into a Interface.

Diagram:

ITest1.java

```
package test;
public interface ITest1 {
   public abstract void m1(int x);
}
```

ITest2.java

```
package test;
public interface ITest2 {
    public abstract void m2(int y);
}
```

ITest3.java

```
package test;
public interface ITest3 extends ITest1,ITest2{
    public abstract void m3(int z);
}
```

IClass.java

```
package test;
public class IClass implements ITest3{
   public void m1(int x) {
         System.out.println("x:"+x);
   }
   public void m2(int y) {
         System.out.println("y:"+y);
   }
   public void m3(int z) {
         System.out.println("z:"+z);
   }
}
```

MultipleInheritance3.java(MainClass)

```
package maccess;
import test.*;
public class MultipleInheritance3 {
      public static void main(String[] args) {
        IClass ob = new IClass();
        ob.m1(123);
        ob.m2(124);
        ob.m3(234);
      }
}
```

==========

Dt : 28/10/2021

*imp

Abstract Class in Java:

=>The class which is declared with abstract keyword is known as abstract class.

=>Abstract class in Java can hold Variables,Concrete methods,abstract methods,Blocks,Constructors and features.

=>we must use 'abstract' kmeyword to declare abstract methods in abstract classes.

=>we cannot instantiate abstract classes,because abstract classes in Java are abstract components

=>These abstract classes are extended to classes using 'extends' keyword and these extention classes must construct body for abstract methods.

Diagram:

Ex program:

AClass.java

```
package test;
public abstract class AClass {
    public void m1(int x) {
        System.out.println("x:"+x);
    }
    public abstract void m2(int y);
}
```

EClass.java

```
package test;
```

```java
public class EClass extends AClass{
    public void m2(int y) {
       System.out.println("y:"+y);
    }
}
```

AbstractClass1.java(MainClass)

```java
package maccess;
import test.*;
public class AbstractClass1 {
   public static void main(String[] args)
{
         EClass ob = new EClass();
         ob.m1(123);
         ob.m2(124);
   }
}
```

Models:

Models:

Comparision diagram:

---

faq:

wt is the diff b/w

(i)Normal class

(ii)Abstract class

=>Normal classes can hold only concrete methods,but abstract classes can hold both abstract methods and Concrete methods.

=>Normal classes can be instantiated,but abstract classes cannot be instantiated.

==========

*imp

InnerClasses in Java:

=>The process of declaring the class inside the class is known as InnerClass or NestedClass.

=>InnerClasses in Java are categorized into three types:

1.Member InnerClasses

2.Local InnerClasses

3.Anonymous InnerClasses

1.Member InnerClasses:

=>The InnerClasses which are declared as members of class are known as

Member InnerClasses.

=>These Member InnerClasses are categorized into two types:

(a)Static Member InnerClasses

(b)NonStatic Member InnerClasses(Instance Member InnerClasses)

(a)Static Member InnerClasses:

=>The Member InnerClasses which are declared with static keyword are known as Static Member InnerClasses.

Coding Rules:

=>Static member InnerClasses can be declared with both static and NonStatic members.

=>The methods of Static member InnerClasses can access static members of OuterClass directly,but cannot access Instance variables.

syntax of creating object for Static member InnerClasses:

OuterClass_name.InnerClass_name ob = new OuterClass_name.InnerClass_name();

Ex:

SubClass1.SubClass2 ob2 = new SubClass1.SubClass2();

(b)NonStatic Member InnerClasses(Instance Member InnerClasses

=>The Member InnerClass which is declared without static keyword is known as NonStatic member InnerClass or Instance member InnerClass.

Coding rules:

=>NonStatic member InnerClasses can be declared with both static and NonStatic members.

=>Static methods of NonStatic member InnerClasses can access static members of OuterClass directly,but cannot access Instance members.(Java16)

=>NonStatic methods of NonStatic member InnerClasses can access all the members of OuterClass directly.

syntax of creating object for NonStatic member InnerClasses:

OuterClass_name.InnerClass_name ob =

OuterClassObject_name.new InnerClass_name();

Ex:

SubClass1.SubClass3 ob3 = ob1.new SubClass3();

Dt : 30/10/2021

2.Local InnerClasses:

=>The InnerClasses which are declared within the methods of OuterClass are known as Local InnerClasses.

=>These Local InnerClasses can be only NonStatic member InnerClasses.

Coding rule:

=>Local InnerClass declaration,Object creation and method access must be declared inside the methods of OuterClass.

Ex program:

SubClass1.java

```
package test;
public class SubClass1 {
   public int a=10;
   public static int b=20;
   public void m1() {
       System.out.println("===OuterClass
method m1()====");
       System.out.println("a:"+a);
       System.out.println("b:"+b);
       class SubClass4{
           public static void m4() {
               System.out.println("===Local
InnerClass static method m4()====");
```

```
//System.out.println("a:"+a);//Compilatio
n_Error
                System.out.println("b:"+b);

            }
            public void m44() {
                System.out.println("===Local
InnerClass Nonstatic method m44()====");

System.out.println("a:"+a);//Compilation_
Error
                System.out.println("b:"+b);

            }
        }//Local InnerClass
        SubClass4 ob4 = new SubClass4();
        SubClass4.m4();
        ob4.m44();
    }//OuterClass method
    public static class SubClass2{
        public static void m2() {
          System.out.println("===Static
member InnerClass static method
m2()====");

//System.out.println("a:"+a);//Compilatio
n_Error
          System.out.println("b:"+b);
        }
        public void m22() {
            System.out.println("===Static
member InnerClass Nonstatic method
m22()====");
```

```
//System.out.println("a:"+a);//Compilatio
n_Error
            System.out.println("b:"+b);
        }
    }//Static member InnerClass
    public class SubClass3{
        public static void m3() {
            System.out.println("===NonStatic
member InnerClass static method
m3()====");

//System.out.println("a:"+a);//Compilatio
n_Error
            System.out.println("b:"+b);
        }
        public void m33() {
            System.out.println("===NonStatic
member InnerClass Nonstatic method
m33()====");

System.out.println("a:"+a);//Compilation_
Error
            System.out.println("b:"+b);
        }
    }//NonStatic member InnerClass
}//OuterClass
```

InnerClass1.java(MainClass)

```
package maccess;
import test.*;
public class InnerClass1 {
    public static void main(String[] args)
{
```

```
        SubClass1 ob1 = new
SubClass1();//OuterClass object
        ob1.m1();
        SubClass1.SubClass2 ob2 = new
SubClass1.SubClass2();//Static Member
InnerClass Object
        SubClass1.SubClass2.m2();
        ob2.m22();
        SubClass1.SubClass3 ob3 = ob1.new
SubClass3();
                                        //N
onStatic member InnerClass object
        SubClass1.SubClass3.m3();
        ob3.m33();
    }
}
```

o/p:

===OuterClass method m1()====

a:10

b:20

===Local InnerClass static method m4()====

b:20

===Local InnerClass Nonstatic method m44()====

a:10

b:20

===Static member InnerClass static method m2()====

b:20

===Static member InnerClass Nonstatic method m22()====

b:20

===NonStatic member InnerClass static method m3()====

b:20

===NonStatic member InnerClass Nonstatic method m33()====

a:10

b:20

Diagram:

---

faq:

InnerClass within the Interface:

=>we can also declare InnerClasses within the interface and which are automatically static member InnerClasses.

faq:

InnerClass within the AbstractClass:

=>we can also declare InnerClass within the abstract class and which can

be static or NonStatic member InnerClasses.

Ex program:

ITest.java

```
package test;
public interface ITest {
   public static class SubClass2{
       public void m2(int x) {
           System.out.println("x:"+x);
       }
   }//InnerClass
}//OuterInterface
```

AClass.java

```
package test;
public abstract class AClass {
    public static class SubClass3{
       public void m3(int y) {
           System.out.println("y:"+y);
       }
    }//InnerClass
}//OuterAbstractClass
```

InnerClass2.java(MainClass)

```
package maccess;
import test.*;
public class InnerClass2 {
    public static void main(String[] args)
{
        ITest.SubClass2 ob2 = new
ITest.SubClass2();
        ob2.m2(123);
        AClass.SubClass3 ob3 = new
AClass.SubClass3();
        ob3.m3(234);
    }
}
```

---

faq:

define Generalization process?

=>In Generalization process one object is created and the object will

hold all the members of PClass and only Overriding members from the CClass

=>This Generalization process is used to restrict the CClass to have only

Overriding members.

=>we use the following syntax to acheive generalization process

PClass ob = new CClass();

Diagram:

Note:

=>This Generalization process can also be acheived using interfaces,by

the following syntax:

Interface_name ob = new Impl_Class_name();

Diagram:

Dt:1/11/2021

Ex_Application-1:(Generalization process using Classes)

PClass.java

```
package test;
public class PClass {
   public void m1(int x) {
         System.out.println("===PClass m1()====");
         System.out.println("x:"+x);
   }
   public void m2(int y) {
         System.out.println("====PClass m2()====");
         System.out.println("y:"+y);
   }
}
```

CClass.java

```
package test;
public class CClass extends PClass{
    public void m1(int x) {
      System.out.println("===CClass m1()====");
      System.out.println("x:"+x);
    }
    public void m3(int z) {
      System.out.println("====CClass m3()====");
      System.out.println("z:"+z);
    }
}
```

Generalization1.java(MainClass)

```
package maccess;
import test.*;
public class Generalization1 {
   public static void main(String[] args) {
         PClass ob = new CClass();
         ob.m1(123);
         ob.m2(234);
         //ob.m3(124);//Compilation_Error
   }
}
```

Ex_Application-2:(Generalization process using Interfaces)

ITest.java

```
package test;
public interface ITest {
   public abstract void m1(int x);
}
```

IClass.java

```
package test;
public class IClass implements ITest{
   public void m1(int x) {
         System.out.println("====IClass m1()===");
         System.out.println("x:"+x);
   }
   public void m2(int y) {
         System.out.println("===IClass m2()====");
         System.out.println("y:"+y);
   }
}
```

Generalization2.java(MainClass)

```
package maccess;
import test.*;
public class Generalization2 {
      public static void main(String[] args) {
        ITest ob = new IClass();
        ob.m1(123);
        //ob.m2(234);//Compilation_Error
      }
}
```

**imp*

3.Anonymous InnerClasses:

=>The InnerClasses which are declared without name are known as Anonymous InnerClasses.

=>These Anonymous InnerClasses are categorized into the following:

(a)Anonymous InnerClass as Class extention

(b)Anonymous InnerClass as Implementation Class

(a)Anonymous InnerClass as Class extention:

=>The process of declaring the CClass without name is known as Anonymous InnerClass as Class Extention.

syntax:

class PClass

{

//m1(x)

//m2(y)

}

PClass ob = new PClass()

{

//m1(x)

//m3(z)

};

Note:

=>Anonymous InnerClass as Class extention model is a modification of 'Generalization process using classes'.

Ex_Application:

PClass.java

```
package test;
public class PClass {
   public void m1(int x) {
         System.out.println("===PClass m1()====");
         System.out.println("x:"+x);
   }
   public void m2(int y) {
         System.out.println("====PClass m2()====");
         System.out.println("y:"+y);
   }
```

```
}
```

InnerClass1.java(MainClass)

```
package maccess;
import test.*;
public class InnerClass1 {
   public static void main(String[] args) {
         PClass ob = new PClass()
         {
               public void m1(int x) {
                 System.out.println("===CClass m1()====");
                 System.out.println("x:"+x);
               }
               public void m3(int z) {
                 System.out.println("====CClass m3()====");
                 System.out.println("z:"+z);
               }
            };
         ob.m1(123);
         ob.m2(234);
         //ob.m3(124);//Compilation_Error
   }
}
```

(b)Anonymous InnerClass as Implementation Class:

=>The process of declaring the implementation class without name is known as Anonymous InnerClass as Implementation Class.

syntax:

```
interface ITest
{
 //m1(int x);
}
ITest ob = new ITest()
{
 //m1(int x)
 //m2(int y)
};
```

Note:

=>Anonymous InnerClass as Implementation class model is a modification of 'Generalization process using Interfaces'.

Ex_Application:

ITest.java

```
package test;
public interface ITest {
   public abstract void m1(int x);
}
```

InnerClass2.java(MainClass)

```
package maccess;
import test.*;
public class InnerClass2 {
      public static void main(String[] args) {
        ITest ob = new ITest()
        {
          public void m1(int x) {
                System.out.println("====IClass m1()===");
                System.out.println("x:"+x);
          }
          public void m2(int y) {
                System.out.println("===IClass m2()====");
                System.out.println("y:"+y);
          }
      };
        ob.m1(123);
```

```
        //ob.m2(234);//Compilation_Error
      }
}
```

---

*imp

LambdaExpressions:(Java8)

=>The process of declaring the method without method_name is known as LambdaExpression,and which is also known as Anonymous method.

structure of LambdaExpression:

```
(para_list)->
{
 //method_body
}
```

Note:

=>The interface abstract method signature will hold LambdaExpression,in this process we call LambdaExpression for execution using Interface_method_name.

syntax:

```
interface ITest
{
 public abstract void m1(int x);
}

ITest ob = (int x)->
   {
     System.out.println(x);
   };
```

Note:

=>Anonymous InnerClass as Implementation class model is modified as LambdaExpression in Java8 version.

Ex_Application:

ITest.java

```
package test;
public interface ITest {
   public abstract void m1(int x);
}
```

LambdaExpression1.java(MainClass)

```
package maccess;
import test.*;
public class LambdaExpression1 {
      public static void main(String[] args) {
        ITest ob = (int x)->
         {
                System.out.println("====IClass m1()===");
                System.out.println("x:"+x);
          };
        ob.m1(123);//Call_lambdaExpression using
Interface_method_name
      }
}
```

--------Dt : 2/11/2021

Execution flow of LambdaExpression:

Advantage of LambdaExpression:

=>when we use LambdaExpression then separate class files are not generated,in

this process loading time of execution process is saved and generates

HighPerformance.

Coding Rule:

=>The interface which is providing abstract method signature to hold LambdaExpression must be declared with only one abstract method,known as Functional Interface.

----

faq:

wt is the diff b/w

(i)Normal Interface

(ii)Functional Interface

=>Normal Interface can hold any number of abstract methods,but Functional must hold only one abstract method.

Ex_Application:

IComparable.java

```
package test;
public interface IComparable
{
    public abstract int compare(int x,int y);
}
```

LambdaExpression2.java

```
package maccess;
import test.*;
import java.util.*;
public class LambdaExpression2 {
        public static void main(String[] args) {
                Scanner s = new Scanner(System.in);
```

```
System.out.println("Enter the value1:");
int v1 = s.nextInt();
System.out.println("Enter the value2:");
int v2 = s.nextInt();
System.out.println("====Choice====");
System.out.println("1.Greater\n2.Smaller");
System.out.println("Enter the Choice:");
int choice = s.nextInt();
switch(choice)
{
case 1://Greater
        IComparable gt = (int x,int y)->
            {
                if(x>y) return x;
                else return y;
            };
        System.out.println("Greater Value:"+gt.compare(v1,v2));
        break;
case 2://Smaller
        IComparable sm = (int x,int y)->
                {
                        if(x<y) return x;
                        else return y;
                };
        System.out.println("Smaller Value:"+sm.compare(v1,v2));
        break;
default:
        System.out.println("Invalid choice...");
}//end of switch
s.close();
```

```
        }
}
```

==========

Ex_Application:

IArithmetic.java

```
package test;
public interface IArithmetic {
    public abstract float process(int x,int y);
}
```

LambdaExpression3.java

```
package maccess;
import java.util.Scanner;
import test.*;
public class LambdaExpression3 {
        public static void main(String[] args)
        {
                Scanner s = new Scanner(System.in);
                System.out.println("Enter the int value1:");
                int v1 = s.nextInt();
    System.out.println("Enter the int value2:");
                int v2 = s.nextInt();
                System.out.println("=====Choice====");
                System.out.println("1.add\n2.sub\n3.mul\n4.div\n5.modDiv");
                System.out.println("Enter your Choice:");
                int choice = s.nextInt();
    switch(choice)
                {
                        case 1:
```

```
IArithmetic ad = (int x,int y)-> x+y;
                    System.out.println("Sum:"+ad.process(v1,v2));
                        break;
                case 2:
                        IArithmetic sb = (int x,int y)-> x-y;
                    System.out.println("Sub:"+sb.process(v1,v2));
                        break;
                case 3:
                        IArithmetic ml = (int x,int y)-> x*y;
                    System.out.println("Mul:"+ml.process(v1,v2));
                        break;
                case 4:
                        IArithmetic dv = (int x,int y)-> (float)x/y;
                    System.out.println("Div:"+dv.process(v1,v2));
                        break;
                case 5:
                        IArithmetic md = (int x,int y)-> x%y;
                    System.out.println("ModDiv:"+md.process(v1,v2));
                        break;
        default:
                        System.out.println("Invalid Choice....");
            }//end of switch
    s.close();
        }

}
```

==========

Assignment:

Update Bank Transaction process(Interface_App3) using LambdaExpressions,which means WithDraw and Deposit classes must be in the LambdaExpressions.

==========

Dt : 2/11/2021

Assignment:(Solution)

Update Bank Transaction process(Interface_App3) using LambdaExpression

Balance.java

```
package test;
public class Balance {
      public double bal=2000;
      public void getBalance() {
        System.out.println("Balance Amt:"+bal);
      }
}
```

CheckPinNo.java

```
package test;
public class CheckPinNo {
      public boolean k=false;
      public boolean verify(int pinNo)
      {
            switch(pinNo)
            {
                  case 1111:k=true;
                        break;
                  case 2222:k=true;
                        break;
                  case 3333:k=true;
                        break;
            }//end switch
            return k;
      }
}
```

Transaction.java

```
package test;
public interface Transaction {
    public static final Balance b = new Balance();
    public abstract void process(int amt);
}
```

LambdaExpression4.java

```
package maccess;

import java.util.Scanner;

import test.*;

public class LambdaExpression4 {
```

```
public static void main(String[] args)
{
        Scanner s = new Scanner(System.in);
        int count=0;
        xyz:
        while(true){
         System.out.println("Enther the pinNo:");
         int pinNo = s.nextInt();
         CheckPinNo cpn = new CheckPinNo();
         boolean z = cpn.verify(pinNo);
         if(z)
         {
                System.out.println("====Choice====");
                System.out.println("1.WithDraw\n2.Deposit");
                System.out.println("Enter the Choice:");
                int choice = s.nextInt();
                switch(choice)
                {
                        case 1:
                                System.out.println("Enter the amt:");
                          int a1 = s.nextInt();
                                if(a1>0 && a1%100==0)
                           {
                                                Transaction wd = (int amt)->
                                                        {

if(amt<=Transaction.b.bal) {

System.out.println("Amt withDrawn:"+amt);

Transaction.b.bal=Transaction.b.bal-amt;
```

```
Transaction.b.getBalance();

System.out.println("Transaction completed...");
                                                                    }else {

System.out.println("Insufficient fund...");
                                                                    }
                                                                 };
                                                        wd.process(a1);
                                        }//end of if
                                            else
                                        {
                                                    System.out.println("Invalid amt...");
                                        }
                                            break xyz;
                                    case 2:
                                            System.out.println("Enter the amt:");
                                      int a2 = s.nextInt();
                                            if(a2>0 && a2%100==0)
                                      {
                                                    Transaction dp = (int amt)->
                                                                    {
                                                                          System.out.println("Amt
Deposited:"+amt);

Transaction.b.bal=Transaction.b.bal+amt;
                                                                          Transaction.b.getBalance();

System.out.println("Transaction completed...");
                                                                    };
                                                    dp.process(a2);
```

```
                    }//end of if
                        else
                    {
                            System.out.println("Invalid amt...");
                    }
                        break xyz;
                default:
                        System.out.println("Invalid choice....");
                break xyz;
            }//end switch
        }//end of if
        else
        {
            System.out.println("Invalid PinNo...");
            count++;
        }
        if(count==3)
            {
            System.out.println("Sorry ! Transaction Blocked....");
            break xyz;//stop the loop
            }
    }//end of while
    s.close();
    }
}
```

==========

*imp

Method References in Java:(Java8)

=>The Process in which the abstract method signature of Functional interface is attached with the method_body from a class,where the class is not related to

interface is known as Method reference.

=>These Method references are categorized into the following:

(a)Reference to Constructor

(b)Reference to Instance method

(c)Reference to Static method

(a)Reference to Constructor:

=>The process in which abstract method of Functional interface is holding the body of Constructor is known as 'Reference to Constructor'.

syntax:

Func_Interface ob = Class_name::new;

Ex:

ITest ob1 = Display :: new;

(b)Reference to Instance method:

=>The process in which abstract method of Functional interface is holding the body of Instance method is known as 'Reference to Instance method'.

syntax:

Func_Interface ob = obj_name :: method_name;

Ex:

ITest ob2 = d :: m1;

(c)Reference to Static method:

=>The process in which abstract method of Functional interface is holding the body of Static method is known as 'Reference to Static method'.

syntax:

Func_Interface ob = Class_name :: method_name;

Ex:

ITest ob3 = Display :: m2;

Ex_Application:

ITest.java

```
package test;
public interface ITest {
    public abstract void dis(int k);
}
```

Display.java

```
package test;
public class Display {
    public Display(int x) {
      System.out.println("====Constructor_body====");
      System.out.println("The value x:"+x);
    }
    public void m1(int y) {
      System.out.println("====Instance Method_Body====");
      System.out.println("The value y:"+y);
    }
    public static void m2(int z) {
      System.out.println("====Static Method_Body====");
      System.out.println("The value z:"+z);
    }
}
```

MethodReferences.java(MainClass)

```
package maccess;
import test.*;
public class MethodReferences {
      public static void main(String[] args) {
        ITest ob1 = Display :: new;//Reference to Constructor
        ob1.dis(123);//Constructor_body is executed

        Display d = new Display(111);//Con_Call
        ITest ob2 = d :: m1;//Reference to Instance method
        ob2.dis(124);//Instance method_body is executed

        ITest ob3 = Display :: m2;//Reference to Static method
        ob3.dis(125);//Static method_body is executed
      }
}
```

Diagram:

**imp*

Concrete methods in Interface:

=>From Java8 version onwards the interface can be declared with Concrete methods.

=>Concrete methods in interface are categorized into the following:

(a)static concrete methods

(b)default concrete methods

(c)private concrete methods

(a)static concrete methods:

=>From Java8 version onwards the interfaces can be declared with static concrete methods.

=>These static concrete methods will get the memory within the interface while interface loading and can be accessed with Interface_name.

Coding Rule:

=>Static Concrete methods of Interfaces are not available to implementation classes,which means Static concrete methods cannot be accessed with Impl_Class_name.

Ex:

ITest.java

```
package test;
public interface ITest {
    public abstract void m1(int x);
    public static void m2(int y) {
      System.out.println("====Static concrete method m2()===");
      System.out.println("The value y:"+y);
    }
}
```

Demo1.java(MainClass)

```
package maccess;
import test.*;
public class Demo1 {
      public static void main(String[] args) {
            ITest ob = (int x)->{
                  System.out.println("====m1()====");
                  System.out.println("The value x:"+x);
            };
            ob.m1(123);//Abstract_method_call
          ITest.m2(124);//Static method_call
          //ob.m2(125);//Compilataion_Error
      }
}
```

Dt : 5/11/2021

(b)default concrete methods:

=>From Java8 version onwards the interfaces can be declared with default concrete methods.

=>These default concrete methods must be declared with 'default' keyword.

Coding rules:

=>These default concrete methods are available to implementation classes and can be accessed with implementation class object.

Ex:

ITest.java

```
package test;
public interface ITest {
      public default void m1(int x) {
        System.out.println("===Default Concrete m1()===");
```

```
        System.out.println("The value x:"+x);
      }
      public abstract void m2(int y);
}
```

Demo2.java(MainClass)

```
package maccess;
import test.*;
public class Demo2 {
      public static void main(String[] args) {
        ITest ob = (int y)->
        {
          System.out.println("====m2()====");
          System.out.println("The value y:"+y);
        };
        ob.m1(123);//default_method_call
        ob.m2(124);//abstarct_method_call
      }
}
```

(c)private concrete methods:

=>From Java9(2017) version onwards the interface can be

declared with private concrete methods

=>These private Concrete methods are categorized into two types:

(i)static private concrete methods

(ii)NonStatic private concrete methods

Coding rule:

=>Private concrete methods are accessed only inside the interface,

which means private concrete methods are accessed by the

NonPrivate concrete methods of Same interface.

Ex_Program:

ITest.java

```
package test;
public interface ITest {
      private void m1(int x) {
        System.out.println("===m1()===");
        System.out.println("The value x:"+x);
      }
      static private void m2(int y) {
        System.out.println("===m2()===");
```

```
        System.out.println("The value y:"+y);
      }
      default void dis(int x,int y) {
        this.m1(x);
        ITest.m2(y);
      }
}
```

Demo3.java(MainClass)

```
package maccess;
import test.*;
public class Demo3 {
      public static void main(String[] args) {
         ITest ob = new ITest() {};
                  //Anonymous Class_body with 0-members
         ob.dis(123,124);
      }
}
```

==========

****imp***

InnerInterfaces in Java:

(i)InnerInterface within the class:

=>we can declare InnerInterfaces within the class and which can be static or NonStatic member InnerInterfaces.

(ii)InnerInterface within the Interface:

=>we can declare InnerInterfaces within the interface and which are automatically static member InnerInterfaces.

(iii)InnerInterface within the AbstractClass:

=>we can declare InnerInterfaces within the abstractClass and

which can be static or NonStatic member InnerInterfaces.

Ex_Program:

SubClass.java

```
package test;
public class SubClass {
    public static interface ITes1{
      public abstract void m1(int x);
    }//InnerInterface
}//OuterClass
```

ITest.java

```
package test;
public interface ITest {
    public static interface ITest2{
      public abstract void m2(int y);
    }//InnerInterface
}//OuterInterface
```

AClass.java

```
package test;
public abstract class AClass {
   public static interface ITest3{
         public abstract void m3(int z);
   }//InnerInterface
}//OuterAbstractClass
```

InnerInterface.java(MainClass)

```
package maccess;
import test.*;
public class InnerInterface {
      public static void main(String[] args) {
        SubClass.ITes1 ob1 = (int x)->{
          System.out.println("===m1()===");
          System.out.println("x:"+x);
        };
        ob1.m1(123);

        ITest.ITest2 ob2 = (int y)->{
       System.out.println("===m2()===");
       System.out.println("y:"+y);
        };
        ob2.m2(124);

        AClass.ITest3 ob3 = (int z)->{
       System.out.println("===m3()===");
       System.out.println("z:"+z);
        };
```

```
        ob3.m3(125);
    }
}
```

imp

InnerAbstractClasses:

(i)InnerAbstarctClass within the Class:

=>we can declare InnerAbstractClasses within the Class and which can be static or NonStatic member InnerAbstractClasses.

(ii)InnerAbstractClass within the Interface:

=>we can declare InnerAbstractClasses within the interface and which are automatically Static member InnerAbstractClasses.

(iii)InnerAbstractClass within the AbstractClass:

=>we can declare InnerAbstractClasses within the AbstractClasses and which can be static or NonStatc member InnerAbstractClasses.

Ex_program:(Assignment)

Dt : 8/11/2021

Ex_program:(Assignment)

SubClass.java

```
package test;
public class SubClass {
   public abstract static class AClass1{
      public abstract void m1(int x);
   }//InnerAbstractClass
}//OuterClass
```

ITest.java

```
package test;
public interface ITest {
    public abstract static class AClass2{
         public abstract void m2(int y);
      }//InnerAbstractClass
}//OuterInterface
```

AClass.java

```
package test;
public abstract class AClass {
    public abstract static class AClass3{
         public abstract void m3(int z);
      }//InnerAbstractClass
}//OuterAbstractClass
```

InnerAbstractClass.java(MainClass)

```
package maccess;
import test.*;
public class InnerAbstractClass {
   public static void main(String[] args)
{
       SubClass.AClass1 ob1 = new
SubClass.AClass1() {
      public void m1(int x) {
          System.out.println("x:"+x);
```

```
        }
      };
      ob1.m1(123);

      ITest.AClass2 ob2 = new
ITest.AClass2() {
        public void m2(int y) {
            System.out.println("y:"+y);
        }
      };
      ob2.m2(124);

      AClass.AClass3 ob3 = new
AClass.AClass3() {
        public void m3(int z) {
             System.out.println("z:"+z);

        }
     };
     ob3.m3(125);
    }
}
```

==========

Summary of Programming Components:(Java Alphabets)

1.Variables

(a)Primitive DataType Variables

(i)Static

(ii)NonStatic

=>Instance

=>Local

(b)NonPrimitive DataType Variables

(i)Static

(ii)NonStatic

=>Instance

=>Local

2.Methods

(a)Static methods

(b)NonStatic methods(Instance Methods)

3.Blocks

(a)Static blocks

(b)NonStatic blocks(Instance blocks)

4.Constructors

=>NonStatic Constructor

5.Classes

(a)Static(Static classes are only InnerClasses)

(b)NonStatic

6.Interfaces

(a)Sattic(Static Interfaces are only InnerInterfaces)

(b)NonStatic

7.AbstractClasses

(a)Static(Static AbstractClasses are only InnerAbstractClasses)

(b)NonStatic

Comparision Diagram:

==========

faq:

wt is the diff b/w

(i)Abstract Class

(ii)Interface

=>Abstract Classes can hold blocks and Constructors,but Interfaces cannot hold blocks and constructors

Note:

=>AbstractClasses and Interfaces are abstract components in Java and which cannot be instantiated.

==========

*imp

Exception Handling Process in Java:

define Exception?

=>The disturbance occured from the application is known as Exception.

define Exception Handling Process?

=>The process used to handle the exception is known as Exception handling process.

=>we use the following blocks in handling exception:

1.try

2.catch

3.finally

1.try:

=>'try' block will hold the statements which are going to raise the exception.

syntax:

```
try
{
 //statements
}
```

Note:

=>when exception raised in try block,then one object is created to hold exception details and the object_reference is thrown onto catch block. (The Object is created for Exception_type_class)

2.catch:

=>The catch block will hold Object reference thrown from the try block and the required msg is generated from catch block.

syntax:

```
catch(Exception_Type_Class obj_ref)
{
 //msg
}
```

3.finally:

=>finally block is part of exception handling process but executed independently without depending on exception.

syntax:

```
finally
{
  //Statements
}
```

Diagram:

Note:

=>In realtime finally block will hold resource closing operations like IO Close,File Close,DB Close,N/W Close,...

==========

faq:

define 'Throwable'?

=>'Throwable' class is from 'java.lang' package and which is

root of Exception Handling Process.

=>This 'Throwable' class is extended into the following two SubClasses:

1.Error

2.Exception

Hierarchy of 'Throwable' class:

1.Error:

=>The disturbance which is occured from the environment is known as 'error'.

=>'java.lang.Error' is the PClass of all the errors occured from the environment.

=>There is no separate process to handle errors.

2.Exception:

=>'java.lang.Exception' is the PClass of all the exceptions

raised from the application.

=>All the exceptions raised from the application are categorized into two types:

(a)UnChecked Exceptions

(b)Checked Exceptions

(a)UnChecked Exceptions:

=>The exceptions which are not identified by the compiler at compilation stage will be raised at execution stage are known as UnChecked Exceptions or Runtime Exceptions.

(b)Checked Exceptions:

=>The exceptions which are identified by the compiler and raised at compilation stage are known as Checked Exceptions or CompileTime exceptions

---

Dt : 9/11/2021

Types of UnChecked Exceptions:

=>UnChecked Exceptions are categorized into two types:

(i)Built-In UnChecked Exceptions

(ii)User defined UnChecked Exceptions

(i)Built-In UnChecked Exceptions:

=>The UnChecked exceptions which are available from JavaLib are known as Built-In UnChecked Exceptions or

Pre-Defined Exceptions.

Ex:

java.lang.ArithmeticException

java.lang.NumberFormatException

java.util.InputMismatchException

...

Ex_Program:(DException1.java)

```
package maccess;
import java.util.*;
public class DException1 {
      public static void main(String[] args) {
              Scanner s = new Scanner(System.in);
              try
              {
              System.out.println("Enter the value:");
              int v = s.nextInt();
              int k = 44/v;//Exception when v=0
                   //Object Created for ArithmeticException
              System.out.println("The value k:"+k);
              }//end of try
              catch(ArithmeticException ob)
              {
                   System.out.println("Enter only Non-Zero value");
                   System.out.println("Details:"+ob.getMessage());
                   //ob.printStackTrace();
              }
              finally
              {
              s.close();
              }
      }
}
```

o/p:

Enter the value:

0

Enter only Non-Zero value

Details:/ by zero

Diagram:

Note:

=>The lines which are declared after Exception_line will be skipped from the execution process,because the control is transferred to the catch block directly.

faq:

define getMessage() method?

=>getMessage() method is from java.lang.Throwable class and which is used to display Exception_details from the object.

Method Signature:

public java.lang.String getMessage();

syntax:

String msg = obj.getMessage();

faq:

define printStackTrace() method?

=>printStackTrace() method is also from java.lang.Throwable class and which is used to display the complete details of exception.

=>Complete details means it displays Exception_Type_Class name,details,method_name and Line_no.

Method Signature:

public void printStackTrace();

syntax:

obj.printStackTrace();

****imp***

Handling Multiple Exceptions

(i)we can handle Multiple exceptions by declaring multiple

catch blocks

Ex_program:(DException11.java)

```
package maccess;
import java.util.*;
public class DException11 {
      public static void main(String[] args) {
              Scanner s = new Scanner(System.in);
              try
              {
              System.out.println("Enter the value:");
              int v = s.nextInt();
              //Exception for Non-Integer input
              //Object created for InputMismatchException
              int k = 44/v;//Exception when v=0
                   //Object Created for ArithmeticException
              System.out.println("The value k:"+k);
              }//end of try
              catch(InputMismatchException ime)
              {
                   System.out.println("Enter only Integer Value");
                   System.out.println("Details:"+ime.getMessage());
              }
              catch(ArithmeticException ob)
              {
                   System.out.println("Enter only Non-Zero value");
                   System.out.println("Details:"+ob.getMessage());
                   //ob.printStackTrace();
              }
              finally
              {
              s.close();
              }
      }
}
```

(ii)From Java7 version onwards we use single catch block to

hold multiple exceptions

syntax:

catch(Exception1 | Exception2 | ... ob)

{

//msg

}

Ex_program:(DException111.java)

```
package maccess;
import java.util.*;
public class DException111 {
      public static void main(String[] args) {
             Scanner s = new Scanner(System.in);
             try
             {
             System.out.println("Enter the value:");
             int v = s.nextInt();
             //Exception for Non-Integer input
             //Object created for InputMismatchException
             int k = 44/v;//Exception when v=0
                  //Object Created for ArithmeticException
             System.out.println("The value k:"+k);
             }//end of try
             catch(InputMismatchException |
                       ArithmeticException ob)//Java7
             {
                  System.out.println
                  ("Enter only Non-Zeo Integer Value");
                  System.out.println("Details:"+ob.getMessage());
             }
             finally
             {
             s.close();
             }
      }
}
```

==========

**imp*

(ii)User defined UnChecked Exceptions:

=>The UnChecked Exceptions which are defined and raised

by the programmer are known as User defined UnChecked Exceptions.

=>we use the following steps to define,raise and handle

User defined UnChecked Exceptions:

step-1 : The user defined class must be extended from 'java.lang.Exception' class.

step-2 : The class must be declared with try-catch blocks

step-3 : define the condition to raise the exception

step-4 : If the condition is true then raise the exception,

which means create object for User defined class from where

the exception is raised

step-5 : use 'throw' keyword to throw the Object_reference onto catch block.

step-6 : Generate the required msg from Catch block.

Note:

=>Pass Exception details as parameter while creating object

for User defined class.

Ex_Program:(DException2.java)

```
package maccess;
import java.util.*;
public class DException2 extends Exception{
      public DException2(String msg) {
            super(msg);
      }
      public static void main(String[] args) {
            Scanner s = new Scanner(System.in);
          try
          {
            System.out.println("Enter the User Age:");
            int age = s.nextInt();
            if(age<18)//Exception
            {
              DException2 ob =
               new DException2("Not-Eligible for Voting");
              throw ob;
            }
            System.out.println("Eligible for Voting...");
            System.out.println("Age:"+age);
          }//end of try
          catch(DException2 ob)
          {
            System.out.println("Details:"+ob.getMessage());
          }
          finally
          {
             s.close();
          }
      }
}
```

======Dt : 12/11/2021

Execution flow of above program:

==========

**imp*

faq:

define 'throw' keyword?

=>In the process of handling user defined exception,we use 'throw' keyword to throw the object reference onto catch block.

==========

Types of Checked Exceptions:

=>Checked Exceptions are categorized into two types:

(i)Built-In checked exceptions

(ii)User defined Checked exceptions

(i)Built-In checked exceptions:

=>The Checked exceptions which are available from JavaLib are known as Built-In checked exceptions.

Ex:

java.lang.InterruptedException

java.lang.ClassNotFoundException

java.io.IOException

java.sql.SQLException

...

faq:

define sleep() method?

=>sleep() method is used to stop the program execution temporarly on some time.

=>This sleep() method is available from 'java.lang.Thread' class

Method Signature of sleep():

public static native void sleep(long) throws java.lang.InterruptedException;

Diagram:

syntax:

Thread.sleep(time);

Ex_Program:

```
package maccess;
public class DException3 {
      public static void main(String[] args) {
         for(int i=1;i<=10;i++) {
            System.out.println("Value:"+i);
```

```
            try {
            Thread.sleep(5000);//Method_call
            }catch(InterruptedException ie)
            {ie.printStackTrace();}
          }//end of loop
        }
}
```

==========

Dt : 13/11/2021

**imp*

(ii)User defined Checked exceptions:

=>The Checked exceptions which are defined by the programmer are known as User defined Checked exceptions.

=>we use the following two steps to raise User defined checked exceptions:

step-1 : add 'throws' keyword to the method Signature and ignore the exception in current method and raise at method_call

step-2 : use 'throw' keyword in catch block and perform re-throwing process. In this re-throwing process the object reference is moved to the catch block of method_call.

Ex_Application:

BranchCheck.java

```
package test;
public class BranchCheck extends Exception
{
       public BranchCheck() {}
         public BranchCheck(String msg) {
               super(msg);
         }
         public void verify(String br)throws BranchCheck
         {
            try {
                  switch(br) {
                  case "ECE":
                        break;
                  case "EEE":
                        break;
                  case "CSE":
                        break;
```

```
                    default: //raise exception
                          BranchCheck bc = new BranchCheck
                          ("Invalid branch");
                          throw bc;
                    }//end of switch
              }catch(BranchCheck bc)  {
                    throw bc;//re-throwing process
              }
          }
}
```

DException4.java(MainClass)

```
package maccess;

import java.util.*;

import test.BranchCheck;

public class DException4 {

          public static void main(String[] args) {

        try(Scanner s = new Scanner(System.in);){

          try {

          System.out.println("Enter the Stu_Branch:");

          String branch = s.nextLine().toUpperCase();

          BranchCheck bc1 = new BranchCheck();//Con_call

          bc1.verify(branch);//Method_Call

          System.out.println("Valid Branch...");

          System.out.println("Branch:"+branch);

          }//end of try

          catch(BranchCheck  bc)

          {

           System.out.println(bc.getMessage());

          }

        }//try-with-resource

          }

}
```

---

faq:

define 'Exception re-throwing process'?

=>The process of declaring 'throw' keyword in catch block and throwing the object reference is known as 'Exception re-throwing process'.

faq:

define 'Exception Propagation'?

=>In Exception re-throwing process the exception is moved from one method to another method is known as 'Exception Propagation'.

---

faq:

define try-with-resource statement?

=>try-with-resource statement introduced by Java7(2011) version and which is used to close the resources automatically,which means finally block is not needed.

syntax:

```
try(resource1;resource2;...)
{
 //statements
}
```

Ex:

```
try(Scanner s = new Scanner(System.in);)
{
 //statements
}
```

Note:

=>when we use try-with-resource statement then 'catch' block is optional block.

---

define 'Enhanced try-with-resource' statement?

=>'Enhanced try-with-resource' statement introduced by Java9(2017) version and in which the resources are declared outside the try and resource reference variables are declared with try.

syntax:

```
resource1;resource2;...
try(res1_var;res2_var;...)
{
 //statements
}
```

Ex:

```
Scanner s = new Scanner(System.in);
try(s;)
{
 //statements
}
package maccess;
import java.util.*;
import test.BranchCheck;
public class DException44 {
            public static void main(String[] args) {
                        Scanner s = new Scanner(System.in);
          try(s;){
            try {
            System.out.println("Enter the Stu_Branch:");
            String branch = s.nextLine().toUpperCase();
            BranchCheck bc1 = new BranchCheck();//Con_call
            bc1.verify(branch);//Method_Call
            System.out.println("Valid Branch...");
```

```
        System.out.println("Branch:"+branch);
        }//end of try
        catch(BranchCheck  bc)
        {
         System.out.println(bc.getMessage());
        }
       }//try-with-resource
         }
}
```

---

faq:

define 'java.lang.NullPointerException'?

=>'java.lang.NullPointerException' is raised when we use NonPrimitive datatype variables assigned with null values.

Ex_Code:

```
package maccess;
public class DException5 {
   public static String name;
          public static void main(String[] args) {
     System.out.println("name:"+name);
     int len = name.length();//NullPointerException is raised
     System.out.println("Length:"+len);
          }
}
```

==========

Assignment:

Update Bank Transaction Model with Exception Handling process

==========

Dt : 15/11/2021

Assignment:(Solution)

Update Bank Transaction Model with Exception Handling process

Balance.java

```
package test;
public class Balance {
    public double bal=2000;
    public void getBalance() {
      System.out.println("Balance Amt:"+bal);
    }
}
```

Transaction.java

```
package test;
public interface Transaction {
    public Balance b = new Balance();
    public abstract void process(int amt)throws WithDraw;
}
```

WithDraw.java

```
package test;
@SuppressWarnings("serial")
public class WithDraw extends Exception implements Transaction{
        public WithDraw() {}
        public WithDraw(String msg) {
              super(msg);
        }
        @Override
       public void process(int amt) throws WithDraw
       {
        try {
              if(amt>b.bal)//Exception
              {
                   WithDraw wd = new WithDraw
                              ("Insufficient fund");
                   throw wd;
              }
              System.out.println("Amt withDrawn:"+amt);
              b.bal = b.bal-amt;
              b.getBalance();
              System.out.println("Transaction completed...");
        }//end of try
        catch(WithDraw wd)
        {
              throw wd;//re-throwing
        }
       }
}
```

Deposit.java

```java
package test;
public class Deposit implements Transaction{
   @Override
     public void process(int amt)
   {
        System.out.println("Amt deposited:"+amt);
        b.bal=b.bal+amt;
        b.getBalance();
        System.out.println("Transaction completed...");
   }
}
```

CheckPinNo.java

```java
package test;
@SuppressWarnings("serial")
public class CheckPinNo extends Exception{
      public CheckPinNo() {}
    public CheckPinNo(String msg) {
      super(msg);
    }
    public void verify(int pinNo)throws CheckPinNo
    {
      try {
            switch(pinNo)
            {
            case 1111:
                  break;
            case 2222:
                  break;
            case 3333:
                  break;
            default: //raise exception
                  CheckPinNo cpn1 =
                          new CheckPinNo("Invalid PinNo");
                  throw cpn1;
            }//end of switch
      }//end of try
      catch(CheckPinNo cpn1)
      {
            throw cpn1;//re-throwing process
      }
    }
}
```

DException6.java(MainClass)

```java
package maccess;

import test.*;

import java.util.*;

@SuppressWarnings("serial")

public class DException6 extends Exception{
```

```
    public DException6(String msg) {
            super(msg);
    }
    public static void main(String[] args) {
Scanner s = new Scanner(System.in);
try(s;){
    int count=0;
    xyz:
    while(true) {
            try {
                    System.out.println("Enter the pinNo:");
                    int pinNo = s.nextInt();
                    CheckPinNo cpn2 = new CheckPinNo();
                    cpn2.verify(pinNo);//method_Call
                    System.out.println("===Choice===");
                    System.out.println("1.WithDraw\n2.Deposit");
                    System.out.println("Enter the Choice:");
                    int choice = s.nextInt();
                    switch(choice)
                    {
                    case 1:
                            System.out.println("Enter the amt:");
                            int a1 = s.nextInt();
                            if(!(a1>0 && a1%100==0))//Exception
                            {
                                    DException6 ob =
                                    new DException6("Invalid amt");
                                    throw ob;
                            }
                            WithDraw wd = new WithDraw();
```

```
                wd.process(a1);//Method_call
                break xyz;
        case 2:
                System.out.println("Enter the amt:");
                int a2 = s.nextInt();
                if(!(a2>0 && a2%100==0))//Exception
                {
                        DException6 ob =
                        new DException6("Invalid amt");
                        throw ob;
                }
                Deposit dp = new Deposit();
                dp.process(a2);//Method_call
                break xyz;
        default:
                System.out.println("Invalid Choice..");
                break xyz;
        }//end of switch
}//end of try
catch(CheckPinNo | DException6 | WithDraw ob)
{
  System.out.println(ob.getMessage());
  if(ob.getMessage().equals("Invalid PinNo"))
  {
        count++;
  if(count==3)
  {
        System.out.println("Transaction blocked");
        break xyz;
  }
```

```
                    continue;
                    }//End of if
                    break xyz;
                  }
          }//end of loop
      }//end of try-with-resource
          }
}
```

Assignment-1:

Update above application Using Anonymous InnerClasses with Exception Handling process.

Assignment-2:

Update above application Using LambdaExpressions with Exception Handling process.

==========

faq:

define Annotation?

=>The tag based information which is added to the programming components like Interface,Class,method and Variable is known as Annotation.

=>These Annotations are represented using '@' symbol

=>These Annotations will provide information to compiler at compilation stage.

Ex:

```
@SuppressWarnings
@Override
```

@SuppressWarnings:

=>@SuppressWarnings will provide information to close the raised warning.

@Override

=>This @Override annotation will specify the compiler to check the method is Overrided method or not.

==========

**imp*

Multi-Threading in Java:

define Application?

=>set-of-programs collected together to perform defined action is known as application.

define Process?

=>According to Java Programmer,the Application under execution is a process.

define Task?

=>The part of process is known as Task.

Note:

=>According to Java programmer,each program(Servlet) in Application is a Task

Dt : 16/11/2021

define Milti-Tasking ?

=>Executing Multiple tasks simultaneously is known as Multi-Tasking. (Simultaneously means at-a-time but not parallel)

define Thread?

=>The part of task is known as Thread.

Note:

=>In the process of executing multiple Tasks only some part of task is executed known as Thread.

define Multi-Threading?

=>Executing Multiple threads Simultaneously is known as Multi-Threading.

Diagram:

**imp*

Create and Executing Thread:

=>In the process of creating thread the class must be implemented from 'java.lang.Runnable' interface.

Diagram:

=>we use the following steps to construct thread:

step-1 : The user defined class must be implemented from 'java.lang.Runnable' interface

step-2 : The user defined class must construct the body for run() method and which hold the required logic.

step-3 : Create object for User defined class.

step-4 : Create object for Built-In class 'java.lang.Thread' and while object creation pass object reference of User defined class as parameter.

step-5 : Execute run() method using start() method

Ex_Application:

Display1.java

```
package test;
public class Display1 implements Runnable{
    @Override
     public void run() {
     for(int i=1;i<=5;i++) {
           System.out.println("Task-One : "+i);
           try {
           Thread.sleep(4000);
           }catch(InterruptedException ie) {ie.printStackTrace();}
     }
    }
}
```

Display2.java

```
package test;
public class Display2 implements Runnable{
    @Override
     public void run() {
       for(int i=101;i<=105;i++) {
             System.out.println("Task-Two : "+i);
             try {
             Thread.sleep(4000);
             }catch(InterruptedException ie) {ie.printStackTrace();}
       }
      }
}
```

DThread1.java(MainClass)

```
package maccess;
import test.*;
public class DThread1 {
     public static void main(String[] args) {
       Display1 ob1 = new Display1();
       Display2 ob2 = new Display2();

       Thread t1 = new Thread(ob1);
       Thread t2 = new Thread(ob2);

       t1.start();
       t2.start();
     }
}
```

o/P:

Task-One : 1

Task-Two : 101

Task-One : 2

Task-Two : 102

Task-One : 3

Task-Two : 103

Task-Two : 104

Task-One : 4

Task-One : 5

Task-Two : 105

Diagram:

****imp***

Anonymous InnerClasses in Multi-Threading:

=>The process of declaring User defined classes without name is known as

'Anonymous InnerClasses in Multi-Threading'.

Ex_Application:

DThraed2.java(MainClass)

```
package maccess;
public class DThread2 {
      public static void main(String[] args) {
         Thread t1 = new Thread(new Runnable()
        {
           @Override
```

```
            public void run() {
            for(int i=1;i<=5;i++) {
                  System.out.println("Task-One : "+i);
                  try {
                  Thread.sleep(4000);
                  }catch(InterruptedException ie)
{ie.printStackTrace();}
            }
          }
      });
        Thread t2 = new Thread(new Runnable()
        {
            @Override
            public void run() {
                    for(int i=101;i<=105;i++) {
                          System.out.println("Task-Two : "+i);
                          try {
                          Thread.sleep(4000);
                          }catch(InterruptedException ie)
{ie.printStackTrace();}
                    }
              }
        });

        t1.start();
        t2.start();
      }
}
```

---

*imp

LambdaExpressions in Multi-Thraeding:(Java8)

=>The process of declaring run() method without name is known as

LambdaExpressions in Multi-Threading.

Ex_Application:

DThread3.java(MainClass)

```
package maccess;
public class DThread3 {
      public static void main(String[] args) {
         Thread t1 = new Thread(()-> {
            for(int i=1;i<=5;i++) {
                  System.out.println("Task-One : "+i);
                  try {
                  Thread.sleep(4000);
                  }catch(InterruptedException ie)
{ie.printStackTrace();}
            }
          });
```

```
        Thread t2 = new Thread(()-> {
                    for(int i=101;i<=105;i++) {
                          System.out.println("Task-Two : "+i);
                          try {
                          Thread.sleep(4000);
                          }catch(InterruptedException ie)
{ie.printStackTrace();}
                    }
              });

        t1.start();
        t2.start();
      }
}
```

Dt : 17/11/2021

****imp***

Thread Synchronization:

=>The process of ordering the threads for execution is known as Thread Synchronization process.

=>Thread Synchronization process canbe done in two ways:

1.Mutual Exclusion process

2.Thread Communication Process

1.Mutual Exclusion process :

=>The process of locking the programming resources like Class or Object or method,and ordering the threads for execution is known as Mutual Exclusion Process.

=>Mutual Exclusion process canbe done in three ways:

(a)synchronized block

(b)synchronized method

(c)static synchronization

(a)synchronized block:

=>The set-of-statements declared with synchronized keyword is known as synchronized block.

=>This synchronized block is used to apply the lock on the object and

which is known as Object locking process.

syntax:

synchronized(ref_var)

{

//statements

}

Ex_Application:

Printer.java

```
package test;
public class Printer {
      public void print(int n,String user) {
        for(int i=1;i<=n;i++) {
              System.out.println(i+" print for "+user);
              try {
              Thread.sleep(4000);
              }catch(Exception e) {e.printStackTrace();}
        }//end of loop
      }
}
```

User1.java

```
package test;
public class User1 implements Runnable{
      public Printer p;
      public User1(Printer p) {
            this.p=p;
      }
      @Override
      public void run() {
        synchronized(p)
        {
      p.print(5,Thread.currentThread().getName());
        }//end of lock
      }
}
```

User2.java

```
package test;
public class User2 implements Runnable{
      public Printer p;
      public User2(Printer p) {
            this.p=p;
      }
      @Override
```

```
      public void run() {
        synchronized(p)
        {
        p.print(5,Thread.currentThread().getName());
        }//end of lock
      }
}
```

DThread4.java(MainClass)

```
package maccess;
import test.*;
public class DThread4 {
      public static void main(String[] args) {
            Printer p = new Printer();

            User1 u1 = new User1(p);
            User2 u2 = new User2(p);

            Thread t1 = new Thread(u1);
            Thread t2 = new Thread(u2);

            t1.setName("Ram");
            t2.setName("Alex");

            t1.start();
            t2.start();
      }
}
```

o/p:

1 print for Ram

2 print for Ram

3 print for Ram

4 print for Ram

5 print for Ram

1 print for Alex

2 print for Alex

3 print for Alex

4 print for Alex

5 print for Alex

Diagram:

Limitation of Object Locking process:

=>In Object Locking process all the instance methods within the object will be under the lock and the methods are available to one user at-a-time.

Note:

=>This limitation can be Overcomed using 'synchronized method process'.

Dt : 18/11/2021

(b)synchronized method:

=>The process of declaring instance method with synchronized keyword is known as synchronized method.

=>In this process the lock is applied on individual instance method and the method is available to one user at a time.

Ex_Application:

Printer.java

```
package test;
public class Printer {
      public synchronized void print(int n,String user) {
        for(int i=1;i<=n;i++) {
              System.out.println(i+" print for "+user);
              try {
              Thread.sleep(4000);
              }catch(Exception e) {e.printStackTrace();}
        }//end of loop
      }
}
```

User1.java

```
package test;
public class User1 implements Runnable{
      public Printer p;
      public User1(Printer p) {
            this.p=p;
      }
      @Override
      public void run() {
      p.print(5,Thread.currentThread().getName());
      }
}
```

User2.java

```java
package test;
public class User2 implements Runnable{
      public Printer p;
      public User2(Printer p) {
            this.p=p;
      }
      @Override
      public void run() {
        p.print(5,Thread.currentThread().getName());
      }
}
```

DThread5.java(MainClass)

```java
package maccess;
import test.*;
public class DThread5 {
      public static void main(String[] args) {
            Printer p = new Printer();

            User1 u1 = new User1(p);
            User2 u2 = new User2(p);

            Thread t1 = new Thread(u1);
            Thread t2 = new Thread(u2);

            t1.setName("Ram");
            t2.setName("Alex");

            t1.start();
            t2.start();
      }
}
```

(c)static synchronization:

=>The process of declaring static method with synchronized keyword is known as static Synchronization,which means the lock is applied on the class and also known as Class Locking process.

Ex_Application:

Printer.java

```
package test;
public class Printer {
   public synchronized static void print(int n,String user)
   {
       for(int i=1;i<=n;i++) {
             System.out.println(i+" print for "+user);
             try {
             Thread.sleep(4000);
             }catch(Exception e) {e.printStackTrace();}
       }//end of loop
     }
}
```

User1.java

```
package test;
public class User1 implements Runnable{
      @Override
      public void run() {
       Printer.print(5,Thread.currentThread().getName());
      }
}
```

User2.java

```java
package test;
public class User2 implements Runnable{
      @Override
      public void run() {
        Printer.print(5,Thread.currentThread().getName());
      }
}
```

DThread6.java(MainClass)

```java
package maccess;
import test.*;
public class DThread6 {
      public static void main(String[] args) {
            User1 u1 = new User1();
            User2 u2 = new User2();

            Thread t1 = new Thread(u1);
            Thread t2 = new Thread(u2);

            t1.setName("Ram");
            t2.setName("Alex");

            t1.start();
            t2.start();
      }
}
```

2.Thread Communication Process:

=>The process of establishing communication b/w threads using the following methods from 'java.lang.Object' class is known as Thread Communication process.

(i)wait()

(ii)notify()

(iii)notifyAll()

(i)wait():

=>wait() method will make the thread to wait until it receives msg in the form of 'notify()' or 'notifyAll()'.

(ii)notify():

=>notify() method will unlock the resource after execution and send msg

to the next waiting thread.

(iii)notifyAll():

=>notifyAll() method also unlock the resource after execution and send msg

to all the multiple waiting threads.

Ex_Application : Demonstrating Producer-Consumer Problem

Producer.java

```
package test;
public class Producer implements Runnable{
      public StringBuffer sb=null;
      public Producer()
      {
            sb = new StringBuffer();
      }
   @Override
   public void run()
   {
         synchronized(sb)
         {
               try {
                     for(int i=1;i<=10;i++)  {
                           sb.append(i+":");
                           System.out.println("Producer
Appending...");
                           Thread.sleep(4000);
                     }//end of loop
                     sb.notify();//send message to Waiting thread
               }catch(Exception e) {e.printStackTrace();}
         }//end of lock
   }
}
```

Consumer.java

```
package test;
public class Consumer implements Runnable{
      public Producer prod=null;
      public Consumer(Producer prod)
      {
            this.prod=prod;
      }
   @Override
   public void run()
   {
         synchronized(prod.sb)
         {
               try {
                     System.out.println("Consumer Blocked...");
```

```
                    prod.sb.wait();//Blocking the Consumer
                    System.out.println("===Display using
Consumer===");
                    System.out.println(prod.sb);
                }catch(Exception e) {e.printStackTrace();}
            }//end of lock
    }
}
```

DThrea7.java(MainClass)

```
package maccess;
import test.*;
public class DThread7 {
      public static void main(String[] args) {
        Producer p = new Producer();//Con_Call
        Consumer c = new Consumer(p);//Con_Call

        Thread t1 = new Thread(p);
        Thread t2 = new Thread(c);

        t2.start();
        t1.start();
      }

}
```

o/p:

Consumer Blocked...

Producer Appending...

Producer Appending...

Producer Appending...

Producer Appending...

Producer Appending...

Producer Appending...

Producer Appending...

Producer Appending...

Producer Appending...

Producer Appending...

===Display using Consumer===

1:2:3:4:5:6:7:8:9:10:

Dt : 19/11/2021

Diagram:

Note:

=>From the above diagram Consumer is blocked until it receives msg from the Producer.

----

faq:

define Thread Life-Cycle?

=>Thread Life-Cycle demonstrates different states of thread from Thread Creation to Thread Completion and Thread Creation to Thread termination.

=>The following are the states of Thread:

1.Thread Creation(New Thread)

2.Ready-to-run

3.Running

(i)Thread Completion

(ii)Blocked State

=>DeadLock

Blocked State:

=>The temporary blockage of thread is known as Blocked State,and which is also known as 'Live Lock'.

=>In any one of the following event is raised then the thread is under LiveLock or Blocked State:

(i)wait()

(ii)sleep()

(iii)Blocked on I/O

(iv)Blocked to Join

(v)Blocked to Lock

(i)wait():

=>wait() method will block the thread until it receives msg in the form of notify() or notifyAll().

(ii)sleep():

=>sleep() method blocks the thread on some timer.

(iii)Blocked on I/O:

=>Incomplete IO operation is known as Blocked on I/O

(iv)Blocked to Join:

=>In thread dependencies one thread will be blocked to join the result of another thread is known as Blocked to Join.

(v)Blocked to Lock:

=>Incomplete Locking process is known as Blocked to Lock.

Thread Scheduler:

=>Thread Scheduler is an internal Built-In algorithm organizing threads from ready-to-run state to running State based on the following algorithms:

(a)Time Slicing Algorithm

(b)Priority Based Algorithm

(a)Time Slicing Algorithm:

=>In Time Slicing Algorithm all the multiple threads are executed based on time slice.

Note:

=>Time Slicing algorithm is the default algorithm used by Thread Scheduler.

(b)Priority Based Algorithm:

=>In Priority Based Algorithm the threads are executed based on thread pririties.

Note:

=>we use setPriority() method to set the priority for threads.

Method Signature:

public final void setPriority(int);

syntax:

t1.setPriority(8);

=>Thread Prorities must be taken in b/w 1 to 10

1 - Least Priority

10 - Highest Priority

5 - Default Priority/Normal Priority

=>we use getPriority() method to display the priority of threads.

Method Signature:

public final int getPriority();

syntax:

int p = t1.getPriority();

Application of thread:

1.Threads are used in Gaming Applications

2.Thraeds are used in Server Development

3.Threads are used in Server Application Development

Dt : 20/11/2021(Saturday)

**imp*

Arrays in Java:

=>Array is a Sequenced collection of elements of same data type

=>The elements in Array are organized based on index values.

=>Array in Java is a Sequenced collection of Similer objects,which means Objects of same class.

=>Arrays in Java are categorized into two types:

1.Single Dimensional Arrays

2.Multi Dimensional Arrays

1.Single Dimensional Arrays:

=>The Arrays which are represented with one dimension are known as Single-D Arrays or 1-D Arrays.

syntax:

Class_name arr_var[] = new new Class_name[size];

Exp_Program:

wap to read and display multiple book details?

BookData.java

```
package test;
public class BookData {
    public String bCode,bName,bAuthor;
    public float bPrice;
    public int bQty;
    public BookData(String bCode,String bName,
           String bAuthor,float bPrice,int bQty) {
      this.bCode=bCode;
      this.bName=bName;
      this.bAuthor=bAuthor;
      this.bPrice=bPrice;
      this.bQty=bQty;
    }
    public String toString() {
      return bCode+"\t"+bName+"\t"+bAuthor+"\t"+
           bPrice+"\t"+bQty;
    }
}
```

DArray1.java

```
package maccess;

import java.util.*;

import test.*;

public class DArray1 {

        public static void main(String[] args) {

    Scanner s = new Scanner(System.in);

    try(s;){

        try {
```

```
        System.out.println("Enter the number of Books:");
        int n = Integer.parseInt(s.nextLine());
        BookData bd[] = new BookData[n];
                //Array holding n BookData objects
        System.out.println("**Enter "+n+" BookDetails**");
          for(int i=0;i<n;i++)
          {
                System.out.println("Enter BookCode:");
                String bCode = s.nextLine();
                System.out.println("Enter BookName:");
                String bName = s.nextLine();
                System.out.println("Enter BookAuthor:");
                String bAuthor = s.nextLine();
                System.out.println("Enter BookPrice:");
                float bPrice = Float.parseFloat(s.nextLine());
                System.out.println("Enter BookQty:");
                int bQty = Integer.parseInt(s.nextLine());
                bd[i]=new BookData(bCode,bName,bAuthor,bPrice,bQty);
          }//end of loop
          System.out.println("====Display BookDetails===");
          for(int i=0;i<n;i++)
          {
                System.out.println(bd[i].toString());
          }
        }catch(Exception e) {
                System.out.println(e.getMessage());
        }
      }//end of try-with-resource
        }
}
```

Diagram:

Ex_Program : DArray2.java

```
package maccess;
public class DArray2 {
      public static void main(String[] args) {
         Integer a[] = new Integer[3];
                        //Array holding 3 Integer Objects
         String str[] = new String[3];
                        //Array holding 3 String Objects
         System.out.println("====Integer Objects====");
         a[0] = new Integer(12);
         a[1] = new Integer(23);
         a[2] = new Integer(21);
         for(int i=0;i<3;i++)
         {
            System.out.println(a[i].toString());
         }//end of loop
         System.out.println("====String Objects====");
         str[0] = new String("java");
         str[1] = new String("nit");
         str[2] = new String("hyd");
         for(int i=0;i<3;i++)
         {
            System.out.println(str[i].toString());
         }//end of loop
```

```
        }
}
```

---

faq:

define 'Object' Array?

=>The array which is declared with 'java.lang.Object' class is known as Object Array.

Note:

=>'java.lang.Object' class is the PClass of all the classes used in the application,because of this reason Object Array can hold Dis-Similer objects which means objects of different classes.

syntax:

Object o[] = new Object[3];

Ex_Program : DArray3.java

```java
package maccess;
import test.*;
public class DArray3 {
      public static void main(String[] args) {
         Object o[] = new Object[3];
                      //Array to hold DisSimiler objects
         o[0] = new Integer(111);
         o[1] = new String("NIT");
         o[2] = new Product("A123","KB",123.45F,12);
         System.out.println("===Display from Object Array===");
         for(int i=0;i<3;i++)
         {
            System.out.println(o[i].toString());
         }//end of loop
      }
}
```

o/p:

===Display from Object Array===

111

NIT

A123 KB 123.45 12

(b)Multi Dimensional Arrays:

=>The Arrays which are declared with multiple dimensions are known as

Multi Dimensional Arrays.

Note:

=>In realtime Multi-D Arrays are less used when compared to Single-D Arrays

Ex of Multi-D Arrays:

2-D Arrays

3-D Arrays

4-D Arrays

...

=>2-D Arrays are used to construct 'Jagged Arrays'

synatx of 2-D Arrays:

Class_name arr_var[][] = new Class_name[rows][cols];

faq:

define Jagged Array?

=>The Array which hold Array Objects is known as Jagged Array.

Ex_program:DArray4.java

```
package maccess;
public class DArray4 {
      public static void main(String[] args) {
         Integer a1[] = {1,2,3};
         Integer a2[] = {11,12,13,14};
         Integer a3[] = {22,23,24,25,26};
         Integer a[][] = {a1,a2,a3};//Array holding Array objects
         System.out.println("====Jagged Array====");
         for(int i=0;i<a.length;i++) {
            System.out.print("Array-"+(i+1)+":");
            for(int j=0;j<a[i].length;j++) {
                  System.out.print(a[i][j]+" ");
            }//end of InnerLoop
            System.out.println();
         }//end of OuterLoop
      }
}
```

o/p:

====Jagged Array====

Array-1:1 2 3

Array-2:11 12 13 14

Array-3:22 23 24 25 26

Dt : 21/11/2021(Sunday)

Diagram:

Dis-Advantage of Arrays:

=>Array Size once defined cannot be modified at runtime or execution time,because of this reason Arrays are not preferable in realtime to hold dynamic data or runtime data.

Note:

=>This Dis-Advantage can be Overcomed using 'Collection<E>'.

****imp***

Java Collection Framework(JCF):

define 'Collection<E>'?

=>'Collection<E>' is an interface from java.util package and which is root of JCF.

=>This 'Collection<E>' interface is extended to the following Sub-Interfaces:

1.Set<E>

2.List<E>

3.Queue<E>

Hierarchy of Collection<E>:

define Framework?

=>The Structure which is ready constructed and available for application development is known as Framework.

Complete Structure of Collection<E>:

define Generic Programming Components?

=>The programming components which are ready to accept any type of data at runtime are known as Generic Programming Components.

=>The following are Generic Programming Components:

(a)Generic Types

(b)Generic methods

(c)Generic Classes

(d)Generic Interfaces

(a)Generic Types:

=>The types which are ready to accept any type of data at runtime are known as Generic Types.

List:

T - Type

E - Element

K - Key

V - Value

(b)Generic methods:

=>The methods which are ready to accept any type of data as parameters are known as Generic methods

syntax:

<T> return_type method_name(T)

{

* //method_body*

}

(c)Generic Classes:

* =>The Objects of Generic Classes can hold objects of any type.*

syntax:

class Class_name<T>

{

* //class_body*

}

(d)Generic Interfaces:

* =>Generic Interfaces are implemented to GenericClasses.*

syntax:

interface Interface_name<E>

{

* //Interface_body*

}

==========

1.Set<E>:

* =>Set<E> organizes elements without index values and cannot hold duplicate elements.*

* =>Set<E> is implemented into the following:*

*   (a)HashSet<E>*

*   (b)LinkedHashSet<E>*

*   (c)TreeSet<E>*

=>HashSet<E> organizes elements without any order.(No order in the elements)

=>LinkedHashSet<E> organizes elements in insertion order.

=>TreeSet<E> organizes elements in Ascending order automatically.

Ex_Program : DSet1.java

```
package maccess;
import java.util.*;
public class DSet1 {
      public static void main(String[] args) {
          Scanner s = new Scanner(System.in);
          try(s;){
             try {
                  Set<Integer> st=null;
                     //Set holding Unlimited Integer objects
                  System.out.println("===Choice===");

      System.out.println("1.HashSet\n2.LinkedHashSet\n3.TreeSet");
                  System.out.println("Enter the Choice:");
                  int choice = s.nextInt();
                  switch(choice)
                  {
                  case 1:
                        st = new HashSet<Integer>();
                        System.out.println("HashSet Object Created");
                        break;
                  case 2:
                        st = new LinkedHashSet<Integer>();
                        System.out.println("LinkedHashSet Object
Created");
                        break;
                  case 3:
                        st = new TreeSet<Integer>();
                        System.out.println("TreeSet Object Created");
                        break;
                  default:
                        System.out.println("Invalid Choice...");
                        System.exit(0);
                  }//end of choice
                  System.out.println("Enter the number of Integer
Objects:");
                  int n = s.nextInt();
                  System.out.println("Enter "+n+" Integer Objects");
                  for(int i=1;i<=n;i++)
                  {
                        st.add(new Integer(s.nextInt()));
                  }//end of loop
                  System.out.println("====Display from Set
object====");
                  System.out.println(st.toString());
             }catch(Exception e) {e.printStackTrace();}
          }//end of try-with-resource
      }

}
```

o/p:

===Choice===

1.HashSet

2.LinkedHashSet

3.TreeSet

Enter the Choice:

3

TreeSet Object Created

Enter the number of Integer Objects:

5

Enter 5 Integer Objects

10

11

9

23

12

====Display from Set object====

[9, 10, 11, 12, 23]

**imp*

2.List<E>:

=>List<E> organizes elements based on index values and can hold duplicate elements.

=>List<E> is implemented into the following:

(a)ArrayList<E>

(b)LinkedList<E>

(c)Vector<E>

=>ArrayList<E> organizes elements in Sequence and which is Non-Synchronized class.

=>LinkedList<E> organizes elements in Non-Sequence and which is also Non-Synchronized class.

=>Vector<E> organizes elements in Sequence and which Synchronized class.

Ex_Program : DList1.java

```
package maccess;
import java.util.*;
public class DList1 {
      public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        try(s;){
          try {
                List<Integer> l=null;
                    //List holding Integer Objects
                System.out.println("====Choice===");

          System.out.println("1.ArrayList\n2.LinkedList\n3.Vector");
                System.out.println("Enter the Choice:");
                int choice = s.nextInt();
                switch(choice)
                {
                case 1:
                      System.out.println("ArrayList object Created");
                      l = new ArrayList<Integer>();
                      break;
                case 2:
                      System.out.println("LinkedList object Created");
                      l = new LinkedList<Integer>();
                      break;
                case 3:
                      System.out.println("Vector object Created");
                      l = new Vector<Integer>();
                      break;
                default:
                      System.out.println("Invalid Choice...");
                      System.exit(0);
                }//end of switch
                System.out.println("Enter the number of Integer
Objects");
                int n = s.nextInt();
                System.out.println("Enter "+n+" Integer Objects");
                for(int i=1;i<=n;i++)
                {
                      l.add(new Integer(s.nextInt()));
                }//end of loop
                System.out.println("===Display from List Object====");
                System.out.println(l.toString());
          }catch(Exception e) {e.printStackTrace();}
        }//end of try-with-resource
      }
}
```

o/p:

====Choice===

1.ArrayList

2.LinkedList

3.Vector

Enter the Choice:

3

Vector object Created

Enter the number of Integer Objects

3

Enter 3 Integer Objects

12

34

12

===Display from List Object====

[12, 34, 12]

==========

Dt : 22/11/2021

Limitation of ArrayList<E>:

=>when we perform add() operation on ArrayList<E> then the elements moved

Backword direction,and when we perform remove() operation then elements moved

forward direction,in this process if we perform more number of add() and remove() operations then the performance of an application is degraded becuase which waste the execution time in moving the elements in forward and backward.

Note:

=>In realtime ArrayList<E> is not preferable in the applications where we have more number of add() and remove() operations .

=>This Dis-Advantage of ArrayList<E> can be overcomed using LinkedList<E>

al.add(1,500);

al.remove(1);

```
package maccess;
import java.util.*;
public class DList2 {
```

```
        public static void main(String[] args) {
         ArrayList<Integer> al = new ArrayList<Integer>();
             //ArrayList object holding UnLimited Integer objects
         al.add(new Integer(12));
         al.add(new Integer(11));
         al.add(new Integer(13));
         System.out.println(al.toString());
         al.add(1,new Integer(500));
         System.out.println(al.toString());
         al.remove(1);
         System.out.println(al.toString());

        }
}
```

LinkedList<E>:

=>In LinkedList<E> the elements are organized in Non-sequence and the elements are available in 'nodes'.

=>This LinkedList<E> node is divided into the following parts:

(i)Previous Node Address

(ii)Data

(iii)Next Node Address

Note:

=>In realtime LinkedList<E> is preferable in applications where we have more number of add() and remove() operations.

Summary:

ArrayList<E> - is used in User Logins

LinkedList<E> - is used in Admin Logins

Vector<E>:

=>Vector<E> is Synchronized and thread-safe class.

Some important methods of Vector<E>:

public synchronized E elementAt(int);

public synchronized E firstElement();

public synchronized E lastElement();

public synchronized void setElementAt(E,int);

public synchronized void removeElementAt(int);

public synchronized void insertElementAt(E,int);

public synchronized void addElement(E);

public synchronized boolean removeElement(java.lang.Object);

public synchronized void removeAllElements();

Ex_program:

```
package maccess;
import java.util.*;
public class DList3 {
      public static void main(String[] args) {
          Vector<Integer> v = new Vector<Integer>();
                                //Vector holding integer objects
          v.addElement(new Integer(12));
          v.addElement(new Integer(11));
          v.addElement(new Integer(13));
          System.out.println(v.toString());
          System.out.println("FirstElement:"+v.firstElement());
          System.out.println("LastElement:"+v.lastElement());
          System.out.println("length of Vector:"+v.size());
          System.out.println("Ele at index 1:"+v.elementAt(1));
          v.setElementAt(new Integer(500),1);
          System.out.println(v.toString());

      }
}
```

Note:

=>In realtime Vector is used in Connection Pooling process and also used

in Multi-Threading applications.

Dt: 23/11/2021

faq:

define Stack<E>?

=>Stack<E> is a ChildClass of Vector<E> and which organizes elements based

on the algorithm First-In-Last-Out or Last-In-First-Out.

=>The following are the methods form Stack<E>:

public E push(E);

=>This method used to add element to the Stack<E>.

public synchronized E pop();

=>This method is used to delete element from the top-of-Stack.

public synchronized E peek();

=>This method is used to display the element from top-of-stack.

public boolean empty();

=>This method is used to check the Stack<E> empty or not.

public synchronized int search(java.lang.Object);

=>This method is used to search the element in Stack and display the position of an element.

Ex_program : DStack.java

```
package maccess;
import java.util.*;
public class DStack {
      public static void main(String[] args) {
        Stack<Integer> st = new Stack<Integer>();
        st.push(new Integer(12));
        st.push(new Integer(13));
        st.push(new Integer(14));
        st.push(new Integer(15));
        System.out.println(st.toString());
        System.out.println("peek element:"+st.peek());
        int pos = st.search(new Integer(13));
        System.out.println("position of Object 13:"+pos);
        st.pop();
        System.out.println(st.toString());
      }
}
```

o/p:

[12, 13, 14, 15]

peek element:15

position of Object 13:3

[12, 13, 14]

Diagram:

Note:

=>search() method searches the element from top-of-stack to bottom-of-stack and display the position of an element.

==========

Note:

=>Stack<E> and Queue<E> are used part of Algorithmic design in realtime.

==========

3.Queue<E>:

=>Queue<E> organizes elements based on the algorithm First-In-First-Out or List-In-Last-Out.

=>'PriorityQueue<E>' is the implementation of Queue<E> and which organizes

elements based on elements Priority.

Note:

=>In realtime PriorityQueue<E> is used to hold multiple threads running on Priority.

Ex_Program:

```
package maccess;
import java.util.*;
public class DemoQueue {
      public static void main(String[] args) {
         PriorityQueue<Integer> pq=new PriorityQueue<Integer>();
         pq.add(new Integer(500));
         pq.add(new Integer(100));
         pq.add(new Integer(50));
         pq.add(new Integer(3000));
         System.out.println(pq.toString());
      }
}
```

==========

faq:

define Deque<E>?

=>Deque<E> means double-ended-queue and which organizes elements on both ends,which means we can add elements on both eneds and we can delete elements on both eneds.

=>The following are the implementations of Deque<E>:

(i)ArrayDeque<E>

=>Elements are in sequence

(ii)LinkedList<E>

=>Elements are in Non-Sequence

Ex_Program:

```
package maccess;
import java.util.*;
public class DemoDeque {
      public static void main(String[] args) {
        ArrayDeque<Integer> ad = new ArrayDeque<Integer>();
        ad.add(new Integer(12));
        ad.add(new Integer(13));
```

```
        ad.add(new Integer(14));
        ad.add(new Integer(15));
        System.out.println(ad.toString());
        ad.addFirst(new Integer(11));
        ad.addLast(new Integer(16));
        System.out.println(ad.toString());
        ad.removeFirst();
        ad.removeLast();
        System.out.println(ad.toString());

    }
}
```

o/p:

[12, 13, 14, 15]

[11, 12, 13, 14, 15, 16]

[12, 13, 14, 15]

==========

PCODE	PNAME	PPRICE	PQTY
A121	Mouse	234.56	12
A122	KB	200.45	10
A123	FDD	187.45	9
A124	CDR	123.45	13

Limitation of Collection<E>:

=>when we use Collection<E> to hold DB-Table data,it cannot differentiate Primary-Key and Non-Primary_Key values.

Note:

=>The Limitation of Collection<E> can be overcomed using Map<K,V>

Define Map<K,V>?

=>Map<K,V> is an interface from java.util package and which organizes elements in the form of Key-Value pairs.

K-Key(Primary_Key)

V-Values(Non-Primary_Key values)

=>The following are the implementation classes of Map<K,V>:

(a)HashMap<K,V>

(b)LinkedHashMap<K,V>

(c)TreeMap<K,V>

(d)Hashtable<K,V>

=>Hashtable<K,V> is synchronized class and remaining classes are Non-Synchronized classes.

=>HashMap<K,V> and Hashtable<K,V> organizes elements with out any order.

=>LinkedHashMap<K,V> organizes elements in insertion order.

=>TreeMap<K,V> organizes elements automatically in ascending order based on Key(Primary_key).

Primary-Key

Non-Primary-Key values

PCODE	PNAME	PPRICE	PQTY
A121	Mouse	234.56	12
A122	KB	200.45	10
A123	FDD	187.45	9
A124	CDR	123.45	13

Dt : 24/11/2021

Hierarchy of Map<K,V>:

Ex_program:Demonstrating Map<K,V> objects

Note:

=>In the process of mapping DB-Table data on to Map<K,V> objects,we construct User defined class having variables equal to the Non-Primary Key values.

Class : ProductValues

```
package maccess;
public class ProductValues {
    public String pName;
    public float pPrice;
    public int pQty;
    public ProductValues(String pName,float pPrice,int pQty) {
      this.pName=pName;
      this.pPrice=pPrice;
      this.pQty=pQty;
    }
    public String toString() {
      return pName+"\t"+pPrice+"\t"+pQty;
    }
}
```

MainClass : DemoMap.java

```
package maccess;
import java.util.*;
import java.util.Map.Entry;
public class DemoMap {
        public static void main(String[] args) {
                Scanner s = new Scanner(System.in);
                try(s;){
        Map<String,ProductValues> m=null;
        System.out.println("====Choice===");
        System.out.println
        ("1.HashMap\n2.LinkedHashMap\n3.TreeMap\n4.Hashtable");
        System.out.println("Enter the Choice:");
        int choice = s.nextInt();
        switch(choice)
        {
        case 1:
            m = new HashMap<String,ProductValues>();
            break;
        case 2:
            m = new LinkedHashMap<String,ProductValues>();
            break;
        case 3:
            m = new TreeMap<String,ProductValues>();
            break;
        case 4:
            m = new Hashtable<String,ProductValues>();
            break;
        default:
            System.out.println("Invalid Choice...");
            System.exit(0);
```

```
        }//end of switch
        //Adding the data to Map object
        m.put(new String("A124"),
                    new ProductValues("CDR",123.45F,13));
        m.put(new String("A122"),
                    new ProductValues("KB",200.45F,10));
        m.put(new String("A121"),
                    new ProductValues("Mou",234.56F,12));
        m.put(new String("A123"),
                    new ProductValues("FDD",187.45F,9));
        System.out.println("===Display from Map====");
        for( Entry<String, ProductValues> e : m.entrySet())
        {
          System.out.println(e.getKey()+"\t"+e.getValue());
        }
                }//end of try-with-resource


          }
}
```

o/p:

```
====Choice===
1.HashMap
2.LinkedHashMap
3.TreeMap
4.Hashtable
Enter the Choice:
3
===Display from Map====
A121    Mou     234.56  12
A122    KB      200.45  10
```

A123 FDD 187.45 9

A124 CDR 123.45 13

==========

faq:

define Entry<K,V>?

=>Entry<K,V> is an inner interface of Map<K,V> and which is used to retrieve Key-value pairs from the Map<K,V> object.

=>The following are some important methods of Entry<K,V>:

public abstract K getKey();

=>This method is used to retrieve key data.

public abstract V getValue();

=>This method is used to retrieve Value data

==========

Dt : 25/11/2021

define Enum<E>?

=> Enum<E> is an abstract class from java.lang packages.

=>we use 'enum' keyword to declare Enum<E>.(creating implementation of Enum)

syntax:

enum Enum_name

{

//elements

//variables

//methods

}

=>we use values() method to retrieve data from 'Enum<E>' objects.

Ex_program:

Cars.java

```
package maccess;
public enum Cars {
      Figo(900),Alto(200),Dezire(600);//Elements
      public int price;
      private Cars(int price){
            this.price=price;
      }
      public int getPrice() {
            return price;
      }
}
```

DemoEnum.java

```
package maccess;
public class DemoEnum {
      public static void main(String[] args) {
         System.out.println("===Display data from Enum<E>===");
         for( Cars c : Cars.values())
         {
           System.out.println
           (c.toString()+" Costs "+c.getPrice()+" thousand
Dollars");
         }//end of loop
      }
}
```

o/p:

===Display data from Enum<E>===

Figo Costs 900 thousand Dollars

Alto Costs 200 thousand Dollars

Dezire Costs 600 thousand Dollars

Diagram:

Note:

=>Each element of enum is generated as separate object internal to the Enum<E>.

=>The constructor within the Enum must be only private constructor.

=>Enum is less used when compared to Classes and Interfaces.

==========

**imp*

dedfine Abstraction?

=>The process of hiding the implementations from the EndUser/User is known as Abstraction process.

Note:

=>In Java,abstraction process can be acheived using Interfaces and abstract

classes

=>The Applications which are constructed using Interfaces and Abstract Classes are 'Abstraction process Applications'.

**imp*

define Encapsulation?

=>The process of binding all the programming components into a single unit class is known as Encapsulation process.

Note:

=>Class is a collection of Variables,methods,blocks,Constructors,features, InnerClasses,InnerInterfaces,InnerAbstractClass and Exception Handling Components.(try,catch,finally,throw and throws)

=>Interface is a Collection of Variables,abstract methods,Concrete methods, features,InnerClasses,InnerInterfaces,InnerAbstractClasses and Exception Handling Components.(try,catch,finally,throw and throws)

Comparision Diagram:

==========
=

*imp

PolyMorphism in Java:

=>The process in which the same programming component having many forms is known as PolyMorphism

Poly - Many

Morphism - Forms

=>The PolyMorphism is categorized into two types:

1.Dynamic PolyMorphism

2.Static PolyMorphism

1.Dynamic PolyMorphism:

=>The many forms at execution stage is known as Dynamic PolyMorphism or Runtime PolyMorphism.

Exp:

Method Overriding process.

2.Static PolyMorphism:

=>The many forms at compilation stage is known as Static PolyMorphism or Compile time PolyMorphism

Exp:

Method Overloading process.

==========

Dt : 26/11/2021

faq:

How Method Overriding process comes under Dynamic PolyMorphism?

=>Through Method Overriding process we can have more than one form at execution stage without changing Method_Signature,because of this reason Method Overriding process comes under Dynamic PolyMorphism or Runtime PolyMorphism.

Diagram:

Note:

=>process() method of Transaction interface having the following two forms at runtime:

=>WithDraw_Logic

=>Deposit_Logic

faq:

How Method OverLoading process comes under Static PolyMorphism?

=>Through Method Overloading process we can have more than one form by changing para_list or para_type,and these forms are identified and checked by the compiler at compilation stage,because of this reason Method OverLoading process comes under Static PolyMorphism or Compiletime PolyMorphism.

add(int,int)

add(int,int,int)

add(int,float)

Ex_Program:

Addition.java

```
package test;
public class Addition {
      public void add(int x,int y) {
       System.out.println("sum1:"+(x+y));
      }
      public void add(int x,int y,int z) {
       System.out.println("sum1:"+(x+y+z));
      }
      public void add(int x,float y) {
       System.out.println("sum1:"+(x+y));
      }
}
```

DemoPoly1.java(MainClass)

```
package maccess;
import test.*;
public class DemoPoly1 {
      public static void main(String[] args) {
        Addition ad = new Addition();
```

```
        ad.add(1,2);
        ad.add(1,2,3);
        ad.add(1,2.3F);
      }
}
```

=>The Compiler at compilation stage will control the following keywords:

1.static

2.private

3.final

1.static:

=>The following are the static programming components:

(a)static variables

(b)static methods

(c)static blocks

(d)static Classes

(e)static interfaces

(f)static AbstractClasses

Note:

=>There is no concept of static constructor in Java.

*imp

2.private:

=>The following are the private programming components:

(a)private variables

(b)private methods

(c)private Constructors

(d)private classes

Note:

=>There is no concept of private Blocks,private Interfaces and private

abstract classes.

(a)private variables:

=>The variables which are declared with private keyword are known as private variables.

Rule:

=>Private variables are accessed only inside the class,which means accessed by the methods of Same class.

Note:

=>In realtime private variables are used part of Java Bean classes and POJO Classes.(POJO - Plain Old Java Object)

(b)private methods:

=>The methods which are declared with 'private' keyword are known as private methods.

Rule:

=>These private methods are accessed by the Non-Private methods of same class.

Ex_Program:

PDisplay.java

```
package test;
public class PDisplay {
    private int a=10;
    private void m()
    {
     System.out.println("===Private method m()===");
     System.out.println("The value a:"+a);
    }
    public void dis()
    {
     this.m();//Private_method_call
    }
```

}

DemoPoly2.java

```
package maccess;
import test.PDisplay;
public class DemoPoly2 {
     public static void main(String[] args) {
        PDisplay ob = new PDisplay();
        //ob.m();//Compilation_Error
        ob.dis();
     }
}
```

o/p:

===Private method m()===

The value a:10

---

*imp

(c)private Constructors:

=>The Constructors which are declared with private keyword are known as private Constructors.

Rule:

=>Private Constructor is executed when the object is created inside the same class where private constructor is declared.

Note:

=>Using the behaviour of Private constructor we can construct 'SingleTon class'.

faq:

define SingleTon class?

=>The Class which generates only one object and the object is created inside the same class,is known as 'SingleTon class'.

Dt : 27/11/2021

define 'SingleTon class design pattern'?

=>The process of Constructing 'SingleTon Class' and using in the application development is known as 'SingleTon class design pattern'.

=>The following components are used in constructing 'SingleTon class design pattern':

(a)private static reference variable

(b)private Constructor

(c)static method

(a)private static reference variable

=>private static reference variable will hold the reference of object created inside the class.

(b)private Constructor:

=>private constructor is executed when the object is created inside the same class and which restrict the object creation from externally.

(c)static method:

=>we use static method to access the object reference outside the class.

Ex_Program:

CTest.java

```
package test;
public class CTest {
     private CTest() {}
    private static CTest ob = null;
    static
    {
     ob = new CTest();//Con_call
    }
    public static CTest getRef()
    {
```

```
      return ob;
     }
     public void dis(int k) {
      System.out.println("====Instance method dis()====");
      System.out.println("The value k:"+k);
     }
}
```

DemoPoly3.java(MainClass)

```
package maccess;
import test.CTest;
public class DemoPoly3 {
     public static void main(String[] args) {
          CTest ob = CTest.getRef();//Accessing the Object
reference
         ob.dis(123);
     }

}
```

o/p:

====Instance method dis()====

The value k:123

Note:

=>This 'SingleTon class design pattern' is used part of DAO(Data Access

Object) Layer in MVC(Model View Controller) to hold DB-Connection code.

(d)private classes:

=>The Classes which are declared with private keyword are known as

Private classes.

Note:

=>These private classes can be declared as only InnerClasses,which means

OuterClasses cannot be private.

****imp***

3.final:

=>The following are the final programming components:

(a)final variables

(b)final methods

(c)final classes

Note:

=>There is no concept of final blocks,final constructors,final interfaces and final abstract classes.

(a)final variables:

=>The variables which are declared with final keyword in classes are known as final variables.

Rule:

=>These final variables must be initialized with values and once initialized cannot be modified.

Note:

=>final variables in classes canbe initialized using constructor.

(b)final methods:

=>The methods which are declared with final keyword are known as final methods.

Rule:

=>final methods cannot be overrided or cannot be replaced.

(There is no Overriding process for final methods)

(c)final classes:

=>The classes which are declared with final keyword are known as final classes.

Rule:

=>final classes cannot be extended,which means there is no inheritance process for final methods.

Note:

=>In realtime,we use final programming components to construct 'Immutable Classes'.

faq:

define 'Immutable Classes'?

=>The class which is constructed with the following rules is known as Immutable class.

Rule-1 : The class must be final class.

Rule-2 : The variables in class must be private and final variables.

Rule-3 : The class must be declared with only 'Getter methods'.

Rule-4 : These 'Getter methods' must be final methods.

define 'Getter methods'?

=>The methods which are used to get the data from the objects are known as 'Getter methods'.

define 'Setter methods'?

=>The methods which are used to set the data to the objects are known as 'Setter methods'.

Note:

=>These Immutable classes will generate 'Immutable objects'.

faq:

define 'Immutable objects'?

=>The objects once created cannot be modified are known as 'Immutable objects'.(Secured Objects)

Ex_Program:

User.java

```
package test;
  //Immutable Class
public final class User {
     private final String userName,passWord;
     public User(String userName,String passWord) {
      this.userName=userName;
      this.passWord=passWord;
     }
     public final String getUserName() {
          return userName;
     }
     public final String getPassWord() {
          return passWord;
     }
}
```

DemoPoly4.java(MainClass)

```
package maccess;
import test.*;
public class DemoPoly4 {
     public static void main(String[] args) {
       User u = new User("nit.v","mzu672");//Immutable object
       System.out.println("UserName:"+u.getUserName());
       System.out.println("PassWord:"+u.getPassWord());
     }
}
```

Note:

=>In realtime Immutable Objects(Secured Objects) are used in Banking Domain applications to record transaction details.

Note:

=>Based on Security the objects in Java are categorized into two types:

(i)Mutable Objects

(ii)Immutable Objects

(i)Mutable Objects:

=>The Objects once created can be modified are known as Mutable objects.

(ii)Immutable Objects:

=>The Objects once created cannot be modified are known as Immutable Objects.

==========

Dt : 29/11/2021

**imp*

strings in Java:

=>The Sequenced Collection of characters which are represented in double quotes is known as string.

Ex:

"nit","hyd","java",...

=>we use the following classes from java.lang package to create string objects:

1.String class

2.StringBuffer class

3.StringBuilder class

1.String class:

=>'String' class is from java.lang package and which generate 'Immutable Objects'.(Once created cannot be modified)

=>'java.lang.String' class is having 15 Constructors.(Java16 version)

=>we use the following two syntaxes to create objects for 'java.lang.String' class:

syntax-1 : Using 'String literal process'

String s1 = "java";

syntax-2 : Using 'new operator process'

String s2 = new String("program");

Ex_Program : DemoString1.java

```
package maccess;
public class DemoString1 {
     public static void main(String[] args) {
        String s1 = "java";//syntax-1
        String s2 = new String("program");//syntax-2
        String s3 = s1.concat(s2);//Concatenation process
                                  //Instance Factory method.
        int len1 = s1.length();
        int len2 = s2.length();
        int len3 = s3.length();
        System.out.println("s1:"+s1.toString());
        System.out.println("length of s1:"+len1);
        System.out.println("s2:"+s2);
        System.out.println("length of s2:"+len2);
        System.out.println("s3:"+s3);
        System.out.println("length of s3:"+len3);
     }
}
```

o/p:

s1:java

length of s1:4

s2:program

length of s2:7

s3:javaprogram

length of s3:11

----

faq:

define String Constant pool?

=>The separate partition of Heap_Area where String objects are created is known as String Constant pool.

(i)when we use 'String literal process' then the execution control will check the 'String constant pool' is any object having same data.

=>If Object is not available then new object is created.

=>If Object is available then use the reference of existing object without creating new object.

(ii)In 'new operator process' the object is created directly in Heap_Area and the object is binded with the reference of object created in String Constant pool.

faq:

define String Concatenation process?

=>The process of combining multiple Strings into a Single String is known as String Concatenation process.

Note:

=>In String Concatenation process,separate object is created to hold concatenated Strings.

faq:

define Factory methods?

=>The methods which hide the object creation process from user are known as Factory methods.

=>factory methods are categorized into two types:

(i)Static factory methods.

(ii)Instance factory methods.

Note:

=>The characters in Strings are organized based on index values,but Strings in Java are not Arrays.

---

faq:

define toString() method?

=>toString() is a Built-in method available in String Classes,WrapperClasses, Collection classes,Map Classes and Enum to display the data available from the objects.

Method Signature of toString():

public java.lang.String toString();

syntax:

String var = obj.toString();

Note:

=>This toString() method is a Auto-Executable method and which is executed automatically when we display Object reference.

Dt : 30/11/2021

faq:

define String Comparision process?

=>The process of comparing two Strings is known as String Comparision process.

=>This String Comparision process can be done in three ways:

(i)using 'equals()' method

(ii)Using 'compareTo()' method

(iii)Using 'is equal to'(==) operator

(i)using 'equals()' method:

=>equals() method will compare two strings and generate boolean result.

Method Signature:

public boolean equals(java.lang.Object);

public boolean equalsIgnoreCase(java.lang.String);

syntax:

boolean k = s1.equals(s2);

Note:

=>In realtime equals() method is used in authentication process.

(ii)Using 'compareTo()' method:

=>compareTo() method also used to compare two Strings and generate int result.

Method Signature:

public int compareTo(java.lang.String);

public int compareToIgnoreCase(java.lang.String);

syntax:

int z = s1.compareTo(s2);

if z==0 ,then Strings are equal

if z>0 , then s1>s2

if z<0 , then s1<s2

Note:

=>In realtime compareTo() method is used in Sorting process.

Ex_Program:

```
package maccess;
import java.util.*;
public class DemoString2 {
     public static void main(String[] args) {
        try(Scanner s = new Scanner(System.in);){
         System.out.println("Enter the String1:");
         String s1 = s.nextLine().trim();
         System.out.println("Enter the String1:");
         String s2 = s.nextLine().trim();
         System.out.println("====equals()====");
         boolean k = s1.equalsIgnoreCase(s2);
         if(k) {
              System.out.println("Strings are equal...");
         }else {
              System.out.println("Strings are Non-Equal...");
         }
         System.out.println("====compareTo()====");
         int z = s1.compareToIgnoreCase(s2);
         if(z==0) {
              System.out.println("Strings are equal...");
         }else if(z>0) {
              System.out.println("S1 is greater than s2...");
         }else {
              System.out.println("S1 is less than s2...");
         }
        }//end of try
     }
}
```

o/p:

Enter the String1:

java

Enter the String1:

program

====equals()====

Strings are Non-Equal...

====compareTo()====

S1 is less than s2...

Note:

=>'trim()' method is used to remove the spaces before and after the Strings

=>'IgnoreCase' means conside the content for caparision without considering the case.

(iii)Using 'is equal to'(==) operator:

=>'is equal to'(==) operator will compare the object references,which means it will not compare the contents.

Ex_Program:

```
package maccess;
public class DemoString3 {
     public static void main(String[] args) {
        String s1 = new String("java");
        String s2 = new String("java");
        String s3 = "java";
        String s4 = "java";
        System.out.println("===new operator process===");
        if(s1==s2) {
         System.out.println("Strings are Equal...");
        }else {
         System.out.println("Strings are Not-Equal..");
        }
        System.out.println("====String Literal process====");
        if(s3==s4) {
         System.out.println("Strings are equal...");
        }else {
         System.out.println("Strings are Not-equal...");
        }
```

```
        }
}
```

o/p:

===new operator process===

Strings are Not-Equal..

====String Literal process====

Strings are equal...

Note:

=>In realtime 'is equal to'(==) operator is not preferable to compare

Non-Primitive DataType variables,because which generate Wrong results.

==========

Ex_Program : DemoForamts.java

```
package maccess;
public class DemoFormats {
     public static void main(String[] args) {
        System.out.println("===UpperCase Letters===");
        for(int i=65;i<=90;i++)
```

```
        {
          System.out.print((char)i+" ");
        }//end of loop
        System.out.println("\n===LowerCase Letters===");
        for(int i=97;i<=122;i++)
        {
          System.out.print((char)i+" ");
        }//end of loop
        System.out.println("\n===Numbers===");
        for(int i=48;i<=57;i++)
        {
          System.out.print((char)i+" ");
        }//end of loop
      }
}
```

o/p:

===UpperCase Letters===

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

===LowerCase Letters===

a b c d e f g h i j k l m n o p q r s t u v w x y z

===Numbers===

0 1 2 3 4 5 6 7 8 9

==========

Assignment-1:

wap to read a String and display the sum of numbers from the given String?

(without using Built-in methods)

I/P : java17 by 2021

O/P : sum = 1+7+2+0+2+1 = 13

Assignment-2:

wap to read a String and display Vowels,Consonents,numbers and special

Symbols separatly?

i/p : java_17 by */ 2021

O/P:

Vowels : aa

Consonents : jvby

numbers : 172021

*Spl symbols : _ */*

==========

Dt : 1/12/2021

**imp*

IO Streams and Files:

define Stream?

=>The Continuous flow of data is known as Stream.

=>Streams in Java are categorized into two types:

1.Byte Stream(Binary Stream)

2.Character Stream

1.Byte Stream(Binary Stream):

=>The continuous flow of data in the form of 8-bit is known as Byte Stream or Binary Stream.

Note:

=>Byte Stream supports all the multi-media data formats like Text,Audio, Video,Image and Animation.

2.Character Stream:

=>The Continuous flow of data in the form of 16-bit is known as Character Stream(Text Stream).

Note:

=>Character Stream is preferable for Text data,but not preferable for Audio,Video,Image and Animation.

---

faq:

define Input Stream?

=>The Stream into Java Program is known as Input Stream.

faq:

define Output Stream?

=>The Stream out of Java Program is known as Output Stream.

---

faq:

define File Storage?

=>The smallest permanent storage of computer system which is controlled and managed by the Operating System is known as File Storage.

Note:

=>In the process of establishing communication b/w JavaProgram and File Storage,the JavaProgram must be constructed using classes and interfaces available from java.io package.

Diagram:

**imp*

Classes related to Byte Stream or Binary Stream:

1.DataInputStream

2.DataOutputStream

3.FileInputStream

4.FileOutputStream

1.DataInputStream:

=>DataInputStream class is from java.io package and which is used to read Binary Stream into Java Program.

syntax:

DataInputStream dis = new DataInputStream(source);

2.DataOutputStream:

=>DataOutputStream class is from java.io package and which is used to send Binary Stream out of Java Program.

syntax:

DataOutputStream dos = new DataOutputStream(destination);

3.FileInputStream:

=>FileInputStream class is from java.io package and which is used to find the file and opens the file to read binary Stream.

Syntax:

FileInputStream fis = new FileInputStream("fPath/fName");

4.FileOutputStream:

=>FileOutputStream class is from java.io package and which is used to create new file and opens the file to write Binary Stream.

syntax:

FileOutputStream fos = new FileOutputStream("fPath/fName");

Ex_Program : Transfer the file from one location to another location.

```
package maccess;
import java.io.*;
import java.util.*;
public class DFile1 {
        public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      try(s;){
          try {
          System.out.println("Enter fPath&fName(Source):");
          File f1 = new File(s.nextLine());
          FileInputStream fis = new FileInputStream(f1);
          System.out.println("Enter the fPath&fName(Destination)");
          File f2 = new File(s.nextLine());
          FileOutputStream fos = new FileOutputStream(f2);
```

```
            int k;
            while((k=fis.read())!=-1)
            {
                    fos.write(k);
            }//end of loop
            fos.close();
            fis.close();
            }catch(Exception e) {e.printStackTrace();}
        }//end of try-with-resource
          }
}
```

o/p:

Enter fPath&fName(Source):

D:\images\DD.jpg

Enter the fPath&fName(Destination)

E:\DemoFiles\TT.jpg

Diagram:

***imp**

Classes related to Character Stream:

1.BufferedReader

2.FileReader

3.FileWriter

1.BufferedReader:

=>BufferedReader class is from java.io package and which is used to read character Stream into Java Program.

syntax:

BufferedReader br=new BufferedReader(new InputStreamReader(source));

2.FileReader:

=>FileReader class is from java.io package and which is used to find the file and opens the file to read Character Stream.

syntax:

FileReader fr = new FileReader("fPath/fName");

3.FileWriter:

=>FileWriter class is from java.io package and which is used to create new file and opens the file to write Character Stream.

syntax:

FileWriter fw = new FileWriter("fPath/fName");

Ex_Program :

wap to read character Stream from Console and write into a File?

```
package maccess;
import java.io.*;
public class DFile2 {
     public static void main(String[] args) {
       try {
        BufferedReader br=new BufferedReader
                  (new InputStreamReader(System.in));
        File f = new File("E:\\DemoFiles\\Text.txt");
        FileWriter fw = new FileWriter(f);
        System.out.println("Enter the Data:(@ at end)");
        char ch;
        while((ch=(char)br.read())!='@')
        {
             fw.write(ch);
        }//end of loop
        System.out.println("Data stored in file
Successfully");
        fw.close();
        System.out.println("===Display from File===");
        FileReader fr =
                       new
FileReader("E:\\DemoFiles\\Text.txt");
        int k;
        while((k=fr.read())!=-1)
        {
            System.out.print((char)k);
        }//end of loop
        fr.close();
        br.close();
       }catch(Exception e) {e.printStackTrace();}
     }
}
```

o/p:

Enter the Data:(@ at end)

Thread

file

java

nit

hyd

task

@

Data stored in file Successfully

===Display from File===

Thread

file

java

nit

hyd

task

----

faq:

define 'File' class?

=>'File' class is from java.io package and which is used to find the file properties like File_length,File_exist or not,File_path,...

syntax:

File f = new File("fPath/fName");

==========

faq:

define Serialization process?

=>The process of converting Object State into Binary Stream is known as

Serialization process.

=>This serialization process is performed using writeObject() method available from 'java.io.ObjectOutputStream' class.

Method Signature of writeObject():

public final void writeObject(java.lang.Object)throws java.io.IOException;

syntax:

```
ObjectOutputStream ois = new ObjectOutputStream(fos);
oos.writeObject(object_name);
```

Ex_program:

TransLog.java

```
package maccess;
import java.util.Date;
import java.io.*;
@SuppressWarnings("serial")
public class TransLog implements Serializable{
    public long bAccNo;
    public float amt;
    public Date d;
    public TransLog(long bAccNo,float amt,Date d) {
            this.bAccNo=bAccNo;
            this.amt=amt;
            this.d=d;
    }
    public String toString() {
            return "BAccNo:"+bAccNo+"\nAmt:"+amt+
                            "\nDate-Time:"+d;
    }
```

```
}
```

DemoSerialization.java(MainClass)

```
package maccess;
import java.io.*;
import java.util.*;
public class DemoSerialization {
        public static void main(String[] args) {
    try {
            Scanner s = new Scanner(System.in);
            System.out.println("Enter the Beneficiery AccNo:");
            long bAccNo = s.nextLong();
            System.out.println("Enter the amt to be Transferred:");
            float amt = s.nextFloat();
            TransLog ob1 = new TransLog(bAccNo,amt,new Date());
            FileOutputStream fos =
                        new FileOutputStream("E:\\DemoFiles\\Obj.txt");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(ob1);//Serialization process
            System.out.println("Object Stored Successfully...");
            oos.close();
            fos.close();
            s.close();
    }catch(Exception e) {e.printStackTrace();}
        }

}
```

o/p:

Enter the Beneficiery AccNo:

313131

Enter the amt to be Transferred:

12000

Object Stored Successfully...

----

faq:

define De-Serialization process?

=>The process of converting binary Stream into Object State is known as De-Serialization process.

=>we use readObject() method from 'java.io.ObjectInputStream' class to perform De-Serialization process.

Method Signature of readObject():

public final java.lang.Object readObject() throws java.io.IOException,

java.lang.ClassNotFoundException;

syntax:

ObjectInputStream ois = new ObjectInputStream(fis);

Object ob = ois.readObject();

Ex_Program:

DemoDeSerialization.java(MainClass)

```
package maccess;
import java.io.*;
public class DemoDeSerialization {
     public static void main(String[] args) {
        try {
          FileInputStream fis =
                    new
FileInputStream("E:\\DemoFiles\\Obj.txt");
          ObjectInputStream ois = new ObjectInputStream(fis);
          TransLog ob2 = (TransLog)ois.readObject();
          System.out.println("===TransactionDetails===");
          System.out.println(ob2.toString());
          ois.close();
        }catch(Exception e) {e.printStackTrace();}
     }
```

}

o/p:

===TransactionDetails===

BAccNo:313131

Amt:12000.0

Date-Time:Wed Dec 01 16:50:34 IST 2021

==========

Diagram:

==========

Note:

=>To perform Serialization and De-Serialization the User defined class must be implemented from 'java.io.Serializable' interface.

=>This 'Serializable' interface is Empty interface because which donot

have any members.

=>This 'Serializable' interface is also known as 'Markable' interface or 'Tagging interface'.

==========

**imp*

Networking in Java:

define Computer Network?

=>The inter connection of Autonomous computers is known as Computer Network.

define N/w Protocol?

=>The Protocols used in N/W to establish communication b/w Computers are known as N/W Protocols.

=>These N/W Protocols are categorized into two types:

(a)Connection Oriented Protocols

(b)Connection Less Protocols

(a)Connection Oriented Protocols:

=>In Connection Oriented Protocols Senders will get Ack from receivers.

Ex:

TCP/IP -

(b)Connection Less Protocols:

=>In Connection Less Protocols Senders will not get any Ack from the receivers.

Ex:

UDP -

define IP Address?

=>The Unique identification number which is used to identify the computer in the N/w is known as IP Address.

=>The following are used to establish communication b/w Computers in the N/W:

1.Socket programs

2.RPC -

3.RMI -

4.CORBA -

5.WebService

==========

Dt : 2/12/2021

2.StringBuffer class:

=>StringBuffer class is from java.lang package and which generate 'Mutable Object'.(Mutable objects means can be modified)

=>The following are the constructors from StringBuffer class:

```
public java.lang.StringBuffer();
public java.lang.StringBuffer(int);
public java.lang.StringBuffer(java.lang.String);
public java.lang.StringBuffer(java.lang.CharSequence);
```

Case-1 : Using 'StringBuffer()' constructor

syntax:

```
StringBuffer sb = new StringBuffer();
```

=>In this syntax the StringBuffer object is created with the capacity 16 and the capacity increases dynamically at runtime by doubling the capacity and adding 2.

16==>(16+16+2)==>34=>(34+34+2)==>70...

=>we can use insert() and reverse() methods on StringBuffer objects and

which cannot be used on 'java.lang.String' objects.

Ex_Program : SBuffer1.java

```
package maccess;
public class SBuffer1 {
      public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();//Con_call
        System.out.println("default capacity:"+sb.capacity());
        System.out.println("length:"+sb.length());
        sb.append("java");
        System.out.println("sb:"+sb.toString());
        System.out.println("capacity:"+sb.capacity());
        System.out.println("length:"+sb.length());
        sb.append("program");
        System.out.println("sb:"+sb.toString());
        System.out.println("capacity:"+sb.capacity());
        System.out.println("length:"+sb.length());
        sb.insert(4,"language");
        System.out.println("sb:"+sb.toString());
        System.out.println("capacity:"+sb.capacity());
        System.out.println("length:"+sb.length());
        System.out.println("===reverse====");
        System.out.println(sb.reverse());
      }
}
```

o/p:

default capacity:16

length:0

sb:java

capacity:16

length:4

sb:javaprogram

capacity:16

length:11

sb:javalanguageprogram

capacity:34

length:19

Case-2 : Using 'StringBuffer(int)' constructor

syntax:

StringBuffer sb = new StringBuffer(4);

=>In this Syntax,the StringBuffer object is created with the capacity based on programmer choice.

=>In this process we can pass the capacity as parameter while object creation.

Ex_Program : Sbuffer2.java

```
package maccess;
public class SBuffer2 {
      public static void main(String[] args) {
         StringBuffer sb = new StringBuffer(4);//Con_Call
         System.out.println("default capacity:"+sb.capacity());
         System.out.println("length:"+sb.length());
         sb.append("hyd");
         System.out.println("sb:"+sb.toString());
         System.out.println("capacity:"+sb.capacity());
         System.out.println("length:"+sb.length());
         sb.append("nit");
         System.out.println("sb:"+sb.toString());
         System.out.println("capacity:"+sb.capacity());
         System.out.println("length:"+sb.length());
      }
}
```

o/p:

default capacity:4

length:0

sb:hyd

capacity:4

length:3

sb:hydnit

capacity:10

length:6

Case-3 : Using 'StringBuffer(java.lang.String)' constructor

syntax:

StringBuffer sb = new StringBuffer("nit");

=>In this syntax,the StringBuffer object is created with the capacity

equal to the sum of "16 + length of String passed as parameter".

Ex_Program : SBuffer3.java

```
package maccess;
public class SBuffer3 {
     public static void main(String[] args) {
        StringBuffer sb = new
StringBuffer("nithyd");//Con_call
        System.out.println("default capacity:"+sb.capacity());
        System.out.println("length:"+sb.length());
        System.out.println("sb:"+sb.toString());
     }
}
```

o/p:

default capacity:22

length:6

sb:nithyd

Case-4 : Using 'StringBuffer(java.lang.CharSequence)' constructor

syntax:

StringBuffer sb1 = new StringBuffer("java");

StringBuffer sb2 = new StringBuffer(sb1);

=>This syntax is used to establish communication b/w two StringBuffer

objects.

=>The data which is binded to buffer1 through constructor(while object

creation) is available to buffer2.

=>The data which is added to buffer1 using append() method is not available

to buffer2.

diagram:

Ex_Program : SBuffer4.java

```
package maccess;
public class SBuffer4 {
     public static void main(String[] args) {
          StringBuffer sb1 = new StringBuffer("java");
          StringBuffer sb2 = new StringBuffer(sb1);
          System.out.println("sb2:"+sb2.toString());
          System.out.println
          ("default capacity of sb2:"+sb2.capacity());
          System.out.println("length of sb2:"+sb2.length());
          sb1.append("nit");
          System.out.println("sb2:"+sb2.toString());
          System.out.println
          ("capacity of sb2:"+sb2.capacity());
          System.out.println("length of sb2:"+sb2.length());

     }
}
```

o/p:

sb2:java

default capacity of sb2:20

length of sb2:4

sb2:java

capacity of sb2:20

length of sb2:4

Note:

=>This StringBuffer class is synchronized class.

faq:

define Synchronized class?

=>The class which is declared with synchronized methods is known as Synchronized class.

Note:

=>when StringBuffer class is synchronized then it is Thread_safe class and used in Multi-Threading applications.

3.StringBuilder class :

=>StringBuilder class also from java.lang package and which generate "Mutable objects".

=>The following are the constructors from StringBuilder class:

public java.lang.StringBuilder();

public java.lang.StringBuilder(int);

public java.lang.StringBuilder(java.lang.String);

public java.lang.StringBufilde(java.lang.CharSequence);

Note:

=>StringBuilder class is Non-Synchronized class and Non-Thread_safe class,and which is used in Non-Threading applications.

==========

*imp

WrapperClasses in Java:

=>WrapperClasses are used to make Primitive DataTypes available in the form of Objects.

=>Every Primitive DataType will have its own WrapperClass and there are 8 WrapperClasses available from 'java.lang' package.

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

Dt : 3/12/2021

faq:

define Boxing process?

=>The process of Binding Primitive DataTypes in to WrapperClass objects is known as Boxing process.

=>This Boxing process is performed using Constructors.

=>The following are the list of Constructors from WrapperClasses:

WrapperClass	Constructors
Byte	byte,String
Short	short,String
Integer	int,String

Long	*long,String*
Float	*float,double,String*
Double	*double,String*
Character	*char*
Boolean	*boolean,String*

Ex_Program : DWrapperClass1.java

```
package maccess;
public class DWrapperClass1 {
     public static void main(String[] args) {
            //Boxing process
           byte b = 123;
           short s = 234;
           Byte ob1 = new Byte(b);
           Byte ob2 = new Byte("126");
           Short ob3 = new Short(s);
           Short ob4 = new Short("345");
           Integer ob5 = new Integer(234);
           Integer ob6 = new Integer("123");
           Long ob7 = new Long(123456);
           Long ob8 = new Long("345678");
           Float ob9 = new Float(12.34F);
           Float ob10 = new Float(234567.89);
           Float ob11 = new Float("67.89");
           Double ob12 = new Double(123456.78);
           Double ob13 = new Double("3412.45");
           Character ob14 = new Character('A');
           Boolean ob15 = new Boolean(true);
           Boolean ob16 = new Boolean("false");
           System.out.println("====Display from Objects===");
           System.out.println("ob1:"+ob1.toString());
           System.out.println("ob2:"+ob2);
           System.out.println("ob3:"+ob3);
           System.out.println("ob4:"+ob4);
           System.out.println("ob5:"+ob5);
           System.out.println("ob6:"+ob6);
           System.out.println("ob7:"+ob7);
           System.out.println("ob8:"+ob8);
           System.out.println("ob9:"+ob9);
           System.out.println("ob10:"+ob10);
           System.out.println("ob11:"+ob11);
           System.out.println("ob12:"+ob12);
           System.out.println("ob13:"+ob13);
           System.out.println("ob14:"+ob14);
           System.out.println("ob15:"+ob15);
           System.out.println("ob16:"+ob16);
```

```
        }
}
```

o/p:

====Display from Objects===

ob1:123

ob2:126

ob3:234

ob4:345

ob5:234

ob6:123

ob7:123456

ob8:345678

ob9:12.34

ob10:234567.89

ob11:67.89

ob12:123456.78

ob13:3412.45

ob14:A

ob15:true

ob16:false

Note:

=>Boxing process using constructor is deprecated from Java9 version.

----

faq:

define AutoBoxing process?

=>The Boxing process which is performed automatically is known as AutoBoxing process.

Note:

=>AutoBoxing process means assigning Primitive DataType values to

Non-Primitive DataType variables

Ex_Program : DWrapperClass2.java

```
package maccess;
public class DWrapperClass2 {
     public static void main(String[] args) {
           //AutoBoxing process
          byte b = 123;
          short s = 234;
          Byte ob1 = b;
          Short ob2 = s;
          Integer ob3 = 234;
          Long ob4 = 123456L;
          Float ob5 = 12.34F;
          Double ob6 = 123456.78;
          Character ob7 = 'A';
          Boolean ob8 = true;

          System.out.println("====Display from Objects===");
          System.out.println("ob1:"+ob1.toString());
          System.out.println("ob2:"+ob2);
          System.out.println("ob3:"+ob3);
          System.out.println("ob4:"+ob4);
          System.out.println("ob5:"+ob5);
          System.out.println("ob6:"+ob6);
          System.out.println("ob7:"+ob7);
          System.out.println("ob8:"+ob8);
     }
}
```

o/p:

====Display from Objects===

ob1:123

ob2:234

ob3:234

ob4:123456

ob5:12.34

ob6:123456.78

ob7:A

ob8:true

----

faq:

define UnBoxing process?

=>The process of taking primitive datatype values outof WrapperClass objects is known as UnBoxing process.

=>To perform UnBoxing process we use the following methods:

public byte byteValue();

public short shortValue();

public int intValue();

public long longValue();

public float floatValue();

public double doubleValue();

public char charValue();

public boolean booleanValue();

Ex_Program : DWrapperClass3.java

```
package maccess;
public class DWrapperClass3 {
     public static void main(String[] args) {
            //Boxing process
           byte b = 123;
           short s = 234;
           Byte ob1 = new Byte(b);
           Short ob2 = new Short(s);
           Integer ob3 = new Integer(234);
           Long ob4 = new Long(123456);
           Float ob5 = new Float(12.34F);
           Double ob6 = new Double(123456.78);
           Character ob7 = new Character('A');
           Boolean ob8 = new Boolean(true);

              //UnBoxing process
           byte b1 = ob1.byteValue();
```

```
            short s1 = ob2.shortValue();
            int i = ob3.intValue();
            long l = ob4.longValue();
            float f = ob5.floatValue();
            double d = ob6.doubleValue();
            char ch = ob7.charValue();
            boolean b2 = ob8.booleanValue();

            System.out.println("====Display from Objects===");
            System.out.println("b1:"+b1);
            System.out.println("s1:"+s1);
            System.out.println("i:"+i);
            System.out.println("l:"+l);
            System.out.println("f:"+f);
            System.out.println("d:"+d);
            System.out.println("ch:"+ch);
            System.out.println("b2:"+b2);
        }
}
```

o/p:

====Display from Objects===

b1:123

s1:234

i:234

l:123456

f:12.34

d:123456.78

ch:A

b2:true

----

faq:

define AutoUnBoxing process?

=>The UnBoxing process which is performed automatically is known as

AutoUnBoxing process.

Note:

=>AutoUnBoxing process means assigning Non-Primitive datatype variables

to Primitive DataType variables

Ex_Program : DWrapperClass4.java

```
package maccess;
public class DWrapperClass4 {
     public static void main(String[] args) {
           //AutoBoxing process
          byte b = 123;
          short s = 234;
          Byte ob1 = b;
          Short ob2 = s;
          Integer ob3 = 234;
          Long ob4 = 123456L;
          Float ob5 = 12.34F;
          Double ob6 = 123456.78;
          Character ob7 = 'A';
          Boolean ob8 = true;

           //AutoUnBoxing process
          byte b1 = ob1;
          short s1 = ob2;
          int i = ob3;
          long l = ob4;
          float f = ob5;
          double d = ob6;
          char ch = ob7;
          boolean b2 = ob8;

          System.out.println("====Display from Objects===");
          System.out.println("b1:"+b1);
          System.out.println("s1:"+s1);
          System.out.println("i:"+i);
          System.out.println("l:"+l);
          System.out.println("f:"+f);
          System.out.println("d:"+d);
          System.out.println("ch:"+ch);
          System.out.println("b2:"+b2);
     }
}
```

o/p:

====Display from Objects===

b1:123

s1:234

i:234

l:123456

f:12.34

d:123456.78

ch:A

b2:true

==========

faq:

why we have to make Primitive DataTypes available in the form of Objects?

=>The Frameworks like JCF(Java Collection Framework) and Hibernate accepts data in the form of objects,because of this reason we have to make Primitive DataTypes available in the form of Objects.

==========

define 'Object' class?

=>'Object' class is from java.lang package and which is the ParentClass of all the classes used in the application

=>The following are the methods from java.lang.Object class:

1.hashCode()

2.toString()

3.clone()

4.equals()

5.wait()

6.notify()

7.notifyAll()

8.getClass()

9.finalize()

1.hashCode():

=>The unique numeric number which is generated while object creation is

known as hashCode.

=>we use hashCode() method to display the hashCodes.

method Signature:

public final int hashCode();

syntax:

int hc = obj.hashCode();

Note:

=>we display the hashCode and check the object is created or not.

2.toString():

=>toString() method is used to display the data from the objects.

**imp*

3.clone():

=>The process of creating the duplicate copy of an object is known as Cloning process.

=>we use clone() method to perform cloning process.

Method Signature:

protected native java.lang.Object clone()

throws java.lang.CloneNotSupportedException;

syntax:

Object ob = obj.clone();

faq:

wt is the diff b/w

(i)Original Object

(ii)Cloned Object

=>The Object created from the class directly is known as Original object and the Object created from the Object is known as Duplicate object or

Cloned Object.

Note:

=>Cloning process can be performed in two ways:

1.Shallow Cloning process

2.Deep Cloning process

1.Shallow Cloning process:

=>In Shallow Cloning process only OuterObjects are cloned,which means InnerObjects or Reffered objects are not cloned.

2.Deep Cloning process:

=>In Deep cloning process all the objects are cloned,which means both OuterObjects and InnerObjects.

Dt : 4/12/2021

4.equals():

=>equals() method will compare two objects and generate boolean result.

5.wait()

6.notify()

7.notifyAll()

=>These there methods are used to establish communication b/w threads in Multi-Threading application.

8.getClass():

=>This getClass() method is used to display the class name of an object.

9.finalize():

=>finalize() method is used to check the objects are eligible for garbage

collection process or not.

==========

faq:

define Garbage Collection Process?

=>The process of finding anonymous objects or UnReffered objects and destroying them is known as Garbage Collection Process.

=>This Garbage Collection process is automated process part of ExecutionEngine.

=>ExecutionEngine will use 'gc()' method to perform Garbage Collection process.

=>This gc() method is available from 'java.lang.System' class.

Method Signature:

public static void gc();

syntax:

System.gc();

=>This gc() method internally calls finalize() method to check the object is Anonymous or not,then it destroys.

==========

GUI(Graphical User Interface) programming:

define StandAlone Application?

=>The applications which are installed in one computer and performs actions in the same computer are known as StandAlone applications or Windows Applications or DeskTop applications.

=>Based on User interaction StandAlone applications are categorized into two types:

1.CUI Applications

2.GUI Applications

1.CUI Applications:

=>The applications in which the user interacts through console (commandPrompt) are known as CUI Applications.

(CUI - Console User Interface)

Ex:

above programs

2.GUI Applications:

=>The applications in which the user interacts through GUI components are known as GUI Applications.(GUI-Graphical User Interface)

=>we use the following to construct GUI Components:

(a)AWT(Abstract Window Toolkit)

(b)Swing

(c)javaFx

(a)AWT(Abstract Window Toolkit):

=>AWT stands for 'Abstract Window Toolkit' and which is used design GUI Components.

=>The Classes and Interfaces related to AWT are available from 'java.awt' package.

=>In the process of constructing AWT programs the User defined classes must be extended from 'java.awt.Frame' class.

Diagram:

Ex_Program:

```
package test;
import java.awt.*;
import java.awt.event.*;
@SuppressWarnings("serial")
public class FirstApp extends Frame
{
public void paint(Graphics g)
        {
        g.setColor(Color.red);
        g.fillRect(300,100,200,200);
```

```
        g.setColor(Color.blue);
        g.fillRect(40,40,200,200);
        g.setColor(Color.green);
        g.fillOval(350,160,80,80);
        }
public static void main(String args[])
{
FirstApp f=new FirstApp();
f.setTitle("FirstApp");
f.setSize(800,500);
f.setVisible(true);
f.setBackground(Color.yellow);
f.addWindowListener(new MyClass());
//close the window
}
}
class MyClass extends WindowAdapter
{
public void windowClosing(WindowEvent e)
{
System.exit(0);
}
}
```

Note:

=>AWT Components are PlatForm dependent components and will not Support MVC(Model View Controller).

(b)Swing:

=>'swing' is also used to design GUI Components and which are introduced to overcome the dis-advantages of AWT.

=>The classes and interfaces related to 'swing' are available from 'javax.swing' package.

=>swing components are PlatForm Independent and supports MVC.

=>In the process of constructing swing applications the user defined class must be extended from 'javax.swing.JFrame'.

Diagram:

Ex_Program:SwingApp.java

```
package test;
import java.awt.*;
```

```
import java.awt.event.*;
import javax.swing.*;//package
@SuppressWarnings("serial")
public class  SwingApp extends JFrame
        implements ActionListener
{
            String str,str1,str2;
   JLabel lb;
            JLabel lb1;
            JLabel lb2;
            JLabel lb3;
            JLabel lb4;
            @SuppressWarnings("rawtypes")
            JComboBox jc;
            JTextField t1;
            JTextField t2;
            JTextField t3;
            JButton b1;
            JButton b2;
            @SuppressWarnings({ "unchecked", "rawtypes" })
            SwingApp()// constructor
            {
   Container c = this.getContentPane();
   String str[] = {"Ece","Cse","Eee"};
   jc = new JComboBox(str);
            c.setLayout(null);
   c.setBackground(Color.yellow);
   Font f1 = new Font
```

```
                    ("Dialog",Font.BOLD,30);
lb=new JLabel("ADDITION");
lb.setFont(f1);
lb.setBounds(505, 50, 500, 50);
lb.setForeground(Color.blue);
lb4=new JLabel("Branch : ");
lb4.setFont(f1);
lb4.setBounds(500, 100, 500, 50);
lb4.setForeground(Color.red);
jc.setFont(f1);
jc.setBounds(700, 100, 100, 50);
jc.setForeground(Color.red);
Font f = new Font
                    ("Dialog",Font.BOLD,20);
lb1=new JLabel("Enter the val1");
lb1.setFont(f);
lb1.setBounds(50, 100, 500, 50);
lb1.setForeground(Color.red);
t1=new JTextField(50);
t1.setBounds(200, 100, 100, 50);
lb2=new JLabel("Enter the val2");
lb2.setFont(f);
lb2.setBounds(50, 200, 500, 50);
lb2.setForeground(Color.red);
t2=new JTextField(50);
t2.setBounds(200, 200, 100, 50);
lb3=new JLabel("Result");
lb3.setFont(f);
```

```
lb3.setBounds(50, 300, 500, 50);
lb3.setForeground(Color.red);
t3=new JTextField(50);
t3.setBounds(200, 300, 100, 50);
b1=new JButton("Add");
b1.setBounds(100, 400, 100, 50);
b2=new JButton("Clear");
b2.setBounds(250, 400, 100, 50);
 c.add(lb);
        c.add(lb1);
        c.add(t1);
        c.add(lb4);
        c.add(jc);
 c.add(lb2);
 c.add(t2);
 c.add(lb3);
 c.add(t3);
        c.add(b1);
        c.add(b2);
        b1.addActionListener(this);
        b2.addActionListener(this);
        }
        public static void main(String[] args)
        {
                SwingApp obj = new SwingApp();
                obj.setTitle("SwingFrame");
                obj.setSize(800,600);
                obj.setVisible(true);
```

```
    obj.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);//close the window
        }
public void actionPerformed(ActionEvent ae)
        {
                str=ae.getActionCommand();
                if(str.equals("Add"))
                {
                        str1=t1.getText();
                        str2=t2.getText();
        int a = Integer.parseInt(str1);
        int b = Integer.parseInt(str2);
      Test t = new Test();
            try
            {
            t.check(a, b);//method call
            }
            catch(Test e)
            {
             JOptionPane.showMessageDialog
            (this,"Invalid Input");
            }
        int c = a+b;
        t3.setText(""+c);
                }
                else
                {
                        t1.setText("");
```

```
                t2.setText("");
                t3.setText("");
            }
        }
}
@SuppressWarnings("serial")
class Test extends Exception
{
        void check(int a,int b)
        throws Test
        {
            try
            {
                if(a==0 || b==0)
            {
            throw new Test();
              }
            }//end of try
            catch(Test t)
            {
                throw t;//re-throw
            }
        }
}
```

Ex_Program : StudentApp.java

```
package test;

import java.awt.*;

import java.awt.event.*;

import javax.swing.*;

import java.util.*;

@SuppressWarnings("serial")

public class StudentApp

extends JFrame implements ActionListener

{

        String str1,str2=null,str3,str4;

        JLabel lb1;

        JLabel lb2;

        JLabel lb3;
```

```
JLabel lb4;
JLabel lb5;
JLabel lb6;
JLabel lb7;
JLabel lb8;
@SuppressWarnings("rawtypes")
JComboBox jc;
JTextField t1;
JTextField t2;
JTextField t3;
JTextField t4;
JTextField t5;
JTextField t6;
JButton b1;
JButton b2;

@SuppressWarnings({ "unchecked", "rawtypes" })
StudentApp() //constructor
{
        Container c=this.getContentPane();
        String str1[]=
        {"ECE","CSE","EEE","MECH","CIVIL"};
        jc= new JComboBox(str1);
        c.setLayout(null);
        c.setBackground(Color.yellow);
Font f1=new Font("dialog",Font.BOLD,30);
        lb1=new JLabel("Student Data");
        lb1.setFont(f1);
```

```
        lb1.setBounds(450,50,500,50);
        lb1.setForeground(Color.red);
Font f=new Font("dialog",Font.BOLD,20);
lb3= new JLabel("BRANCH");
        lb3.setFont(f);
        lb3.setBounds(450,100,500,50);
        lb3.setForeground(Color.red);
                jc.setFont(f);
                jc.setBounds(550,100,150,50);
                jc.setForeground(Color.GREEN);
        lb2=new JLabel("NAME");
        lb2.setFont(f);
        lb2.setBounds(50,100,500,50);
        lb2.setForeground(Color.red);
                        t1=new JTextField(50);
                t1.setBounds(200,100,200,50);
                lb4=new JLabel("RNO");
        lb4.setFont(f);
        lb4.setBounds(50,180,500,50);
        lb4.setForeground(Color.red);
                t2=new JTextField(50);
                t2.setBounds(200,180,200,50);
        lb5=new JLabel("6 SUB MARKS");
        lb5.setFont(f);
        lb5.setBounds(50,260,500,50);
        lb5.setForeground(Color.red);
                t3=new JTextField(50);
                t3.setBounds(200,260,300,50);
```

```
lb6=new JLabel("TOTAL");
lb6.setFont(f);
lb6.setBounds(50,340,500,50);
lb6.setForeground(Color.red);
        t4=new JTextField(50);
        t4.setBounds(200,340,150,50);
lb7=new JLabel("PERCENTAGE");
lb7.setFont(f);
lb7.setBounds(450,340,500,50);
lb7.setForeground(Color.red);
        t5=new JTextField(50);
        t5.setBounds(600,340,150,50);
        lb8=new JLabel("RESULT");
lb8.setFont(f);
lb8.setBounds(50,420,500,50);
lb8.setForeground(Color.red);
        t6=new JTextField(50);
        t6.setBounds(200,420,150,50);
        b1=new JButton("Calculate");
b1.setBounds(300,500,100,50);
        b2=new JButton("Clear");
b2.setBounds(500,500,100,50);
        c.add(lb1);
c.add(lb2);
c.add(t1);
c.add(lb3);
c.add(jc);
c.add(lb4);
```

```
                c.add(t2);
                c.add(lb5);
                c.add(t3);
                c.add(lb6);
                c.add(t4);
                c.add(lb7);
                c.add(t5);
                c.add(lb8);
                c.add(t6);
                c.add(b1);
                c.add(b2);
                b1.addActionListener(this);
                b2.addActionListener(this);
        }
        public static void main(String[] args)
        {
                StudentApp obj1=new StudentApp();
                obj1.setTitle("Student Details");
                obj1.setSize(800,600);
                obj1.setVisible(true);
obj1.setDefaultCloseOperation
   (JFrame.EXIT_ON_CLOSE); // close window
        }
public void actionPerformed(ActionEvent arg)
        {
                str1=arg.getActionCommand();
                if(str1.equals("Calculate"))
                {
```

```
str2=t1.getText();
str3=t2.getText();
try
{
int len=str3.length();
if(len==10)
        {
        try
                {
String s11=str3.substring(7,8);
        Choice2 c1=new Choice2();
        String bb=c1.valid(s11);
        boolean br1=bb.equals("1");
        boolean br2=bb.equals("2");
        boolean br3=bb.equals("3");
        boolean br4=bb.equals("4");
        boolean br5=bb.equals("5");
        String ss=null;
        if(br1)
                ss="CIVIL";
        else if(br2)
                ss="EEE";
        else if(br3)
                ss="mech";
        else if(br4)
                ss="ECE";
        else if(br5)
                ss="CSE";
```

```
if(((jc.getSelectedItem().toString())
                                        .equals(ss)))
                                {
                                        try
                                        {
                                        str4=t3.getText();
StringTokenizer st=
                new StringTokenizer(str4," ");
                                        int a,b,c,d,e,f;
                                        String s1=st.nextToken();
                                        String s2=st.nextToken();
                                        String s3=st.nextToken();
                                        String s4=st.nextToken();
                                        String s5=st.nextToken();
                                        String s6=st.nextToken();
                                a=Integer.parseInt(s1);
                                b=Integer.parseInt(s2);
                                c=Integer.parseInt(s3);
                                d=Integer.parseInt(s4);
                                e=Integer.parseInt(s5);
                                f=Integer.parseInt(s6);
            if(!((a<0 || a>100) || (b<0 || b>100) ||
                        (c<0 || c>100)
        || (d<0 || d>100) || (e<0 || e>100) ||
                        (f<0 || f>100)))
                                        {
                                        int total=a+b+c+d+e+f;
                                        t4.setText(" "+total);
```

```
                    float per=total/6;
                    t5.setText(" "+per);
if((a<35 || b<35 || c<35 || d<35 || e<35 ||
                f<35))
                        {
                    t6.setText("fail");
                        }
                        else
                        {
                    t6.setText("pass");
                        }
                }
                else
                {
JOptionPane.showMessageDialog
  (this,"values between 0 to 100");
                }
                }
            catch(NumberFormatException nfe)
                {
JOptionPane.showMessageDialog
(this,"only enter the number in marks");
                }
            }
            else
            {
JOptionPane.showMessageDialog
(this,"mismatch of rno and branch");
```

```
                    }
                        }
            catch(NullPointerException npe)
                    {
        JOptionPane.showMessageDialog
            (this,"invalid rno");
                    }
                    }
                else
                    {
        JOptionPane.showMessageDialog
        (this,"rno must be 10 digits");
                    }
                }
        catch(NoSuchElementException nsee)
                {
JOptionPane.showMessageDialog
(this," plz enter 6 sub marks");
                }
        }
            else
            {
                t1.setText("");
                t2.setText("");
                t3.setText("");
                t4.setText("");
                t5.setText("");
                t6.setText("");
```

```
                }
            }
}
class Choice2
{
  String b;
  String valid(String s1)
  {
    switch(s1)
            {
              case "1":
                      b="1";
                      break;
              case "2":
                      b="2";
                      break;
              case "3":
                      b="3";
                      break;
             case "4":
                      b="4";
                      break;
             case "5":
                      b="5";
                      break;
            }
 return b;
   }
```

}

(c)javaFx:

=>JavaFx introduced by Java8 version(2014) and which is also used to construct GUI Components.

=>JavaFx is PlatForm independent and supports MVC.

=>JavaFx is rich in in-built UI Controls

=>JavaFx is in-built with CSS.

==========

Assignment-1:(Solution)

wap to read a String and display the sum of numbers from the given String?

(without using Built-in methods)

```
package maccess;
```

```java
import java.util.*;
public class Assignment1 {
     public static void main(String[] args) {
          Scanner s = new Scanner(System.in);
          System.out.println("Enter the String:");
        String str = s.nextLine();
        int len = str.length();
        int sum=0;
        for(int i=0;i<=len-1;i++)
        {
          char ch = str.charAt(i);
          if((int)ch>=48 && (int)ch<=57)
          {
             int k = Integer.parseInt(String.valueOf(ch));
             sum=sum+k;
          }//end of if
        }//end of loop
        System.out.println("The sum of numbers:"+sum);
        s.close();
     }
}
```

I/P : java17 by 2021

O/P : sum = 1+7+2+0+2+1 = 13

Assignment-1:

Update above application Using Anonymous InnerClasses with Exception

Handling process.

Balance.java

```java
package test;
public class Balance {
    public double bal=2000;
    public void getBalance() {
     System.out.println("Balance Amt:"+bal);
    }
}
```

CheckPinNo.java

```java
package test;
@SuppressWarnings("serial")
public class CheckPinNo extends Exception{
     public CheckPinNo() {}
    public CheckPinNo(String msg) {
```

```
        super(msg);
      }
      public void verify(int pinNo)throws CheckPinNo
      {
        try {
              switch(pinNo)
              {
              case 1111:
                    break;
              case 2222:
                    break;
              case 3333:
                    break;
              default: //raise exception
                    CheckPinNo cpn1 =
                            new CheckPinNo("Invalid PinNo");
                    throw cpn1;
              }//end of switch
        }//end of try
        catch(CheckPinNo cpn1)
        {
              throw cpn1;//re-throwing process
        }
      }
}
```

Transaction.java

```
package test;
public interface Transaction {
    public Balance b = new Balance();
    public abstract void process(int amt)throws Exception;
}
```

Dexception6.java

```
package maccess;

import test.*;

import java.util.*;

@SuppressWarnings("serial")

public class DException6 extends Exception{

        public DException6(String msg) {

                super(msg);

        }

        public static void main(String[] args) {

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

```
try(s;){
    int count=0;
    xyz:
    while(true) {
            try {
                    System.out.println("Enter the pinNo:");
                    int pinNo = s.nextInt();
                    CheckPinNo cpn2 = new CheckPinNo();
                    cpn2.verify(pinNo);//method_Call
                    System.out.println("===Choice===");
                    System.out.println("1.WithDraw\n2.Deposit");
                    System.out.println("Enter the Choice:");
                    int choice = s.nextInt();
                    switch(choice)
                    {
                    case 1:
                            System.out.println("Enter the amt:");
                            int a1 = s.nextInt();
                            if(!(a1>0 && a1%100==0))//Exception
                            {
                                    DException6 ob =
                                    new DException6("Invalid amt");
                                    throw ob;
                            }
                            Transaction wd = new Transaction()
                            {
                                    @Override
                                public void process(int amt) throws Exception
```

```
                    {
                        try {
                            if(amt>b.bal)//Exception
                            {
                                Exception wd = new Exception
                                        ("Insufficient fund");
                                throw wd;
                            }
                            System.out.println("Amt withDrawn:"+amt);
                            b.bal = b.bal-amt;
                            b.getBalance();
                            System.out.println("Transaction
completed...");
                        }//end of try
                        catch(Exception wd)
                        {
                            throw wd;//re-throwing
                        }
                    }
                };
                wd.process(a1);//Method_call
                break xyz;
            case 2:
                System.out.println("Enter the amt:");
                int a2 = s.nextInt();
                if(!(a2>0 && a2%100==0))//Exception
                {
                    DException6 ob =
                    new DException6("Invalid amt");
```

```
                    throw ob;
                }
                Transaction dp = new Transaction()
                {
                    @Override
                        public void process(int amt)
                    {
                        System.out.println("Amt deposited:"+amt);
                        b.bal=b.bal+amt;
                        b.getBalance();
                        System.out.println("Transaction
completed...");
                    }
                };
                dp.process(a2);//Method_call
                break xyz;
            default:
                System.out.println("Invalid Choice..");
                break xyz;
            }//end of switch
        }//end of try
        catch(CheckPinNo | DException6 ob)
        {
          System.out.println(ob.getMessage());
          if(ob.getMessage().equals("Invalid PinNo"))
          {
              count++;
          if(count==3)
          {
```

```
                System.out.println("Transaction blocked");
                break xyz;
            }
            continue;
            }//End of if
            break xyz;
        }
        catch(Exception e) {
            System.out.println(e.getMessage());
            break xyz;
        }
    }//end of loop
  }//end of try-with-resource
    }
}
```

Assignment-2:

Update above application Using LambdaExpressions with Exception Handling process.

Balance.java

```
package test;
public class Balance {
    public double bal=2000;
    public void getBalance() {
     System.out.println("Balance Amt:"+bal);
    }
}
```

CheckPinNo.java

```
package test;
@SuppressWarnings("serial")
public class CheckPinNo extends Exception{
     public CheckPinNo() {}
    public CheckPinNo(String msg) {
     super(msg);
```

```
    }
    public void verify(int pinNo)throws CheckPinNo
    {
     try {
          switch(pinNo)
          {
          case 1111:
               break;
          case 2222:
               break;
          case 3333:
               break;
          default: //raise exception
               CheckPinNo cpn1 =
                       new CheckPinNo("Invalid PinNo");
               throw cpn1;
          }//end of switch
     }//end of try
     catch(CheckPinNo cpn1)
     {
          throw cpn1;//re-throwing process
     }
    }
}
```

Transaction.java

```
package test;
public interface Transaction {
    public Balance b = new Balance();
    public abstract void process(int amt)throws Exception;
}
```

Dexception7.java

```
package maccess;

import test.*;

import java.util.*;

@SuppressWarnings("serial")

public class Dexception7 extends Exception{

        public DException6(String msg) {

                super(msg);

        }

        public static void main(String[] args) {

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

```
try(s;){
    int count=0;
    xyz:
    while(true) {
            try {
                    System.out.println("Enter the pinNo:");
                    int pinNo = s.nextInt();
                    CheckPinNo cpn2 = new CheckPinNo();
                    cpn2.verify(pinNo);//method_Call
                    System.out.println("===Choice===");
                    System.out.println("1.WithDraw\n2.Deposit");
                    System.out.println("Enter the Choice:");
                    int choice = s.nextInt();
                    switch(choice)
                    {
                    case 1:
                            System.out.println("Enter the amt:");
                            int a1 = s.nextInt();
                            if(!(a1>0 && a1%100==0))//Exception
                            {
                                    DException6 ob =
                                    new DException6("Invalid amt");
                                    throw ob;
                            }
                            Transaction wd = (int amt)->
                                 {
                                        try {
                                                if(amt>Transaction.b.bal)//Exception
```

```
                                        {
                                                Exception wd1 = new Exception
                                                        ("Insufficient fund");
                                                throw wd1;
                                        }
                                        System.out.println("Amt withDrawn:"+amt);
                                        Transaction.b.bal = Transaction.b.bal-amt;
                                        Transaction.b.getBalance();
                                        System.out.println("Transaction
completed...");
                                }//end of try
                                catch(Exception wd1)
                                {
                                        throw wd1;//re-throwing
                                }
                        };
                wd.process(a1);//Method_call
                break xyz;
        case 2:
                System.out.println("Enter the amt:");
                int a2 = s.nextInt();
                if(!(a2>0 && a2%100==0))//Exception
                {
                        DException6 ob =
                        new DException6("Invalid amt");
                        throw ob;
                }
                Transaction dp = (int amt)->
                        {
```

```
                                        System.out.println("Amt deposited:"+amt);
                                        Transaction.b.bal=Transaction.b.bal+amt;
                                        Transaction.b.getBalance();
                                        System.out.println("Transaction
completed...");
                                    };
                        dp.process(a2);//Method_call
                        break xyz;
                    default:
                        System.out.println("Invalid Choice..");
                        break xyz;
                    }//end of switch
                }//end of try
                catch(CheckPinNo | DException6 ob)
                {
                  System.out.println(ob.getMessage());
                  if(ob.getMessage().equals("Invalid PinNo"))
                  {
                        count++;
                  if(count==3)
                  {
                        System.out.println("Transaction blocked");
                        break xyz;
                  }
                  continue;
                  }//End of if
                  break xyz;
                }
                catch(Exception e) {
```

```
                    System.out.println(e.getMessage());
                    break xyz;
                }
        }//end of loop
    }//end of try-with-resource
        }
}
```

==========