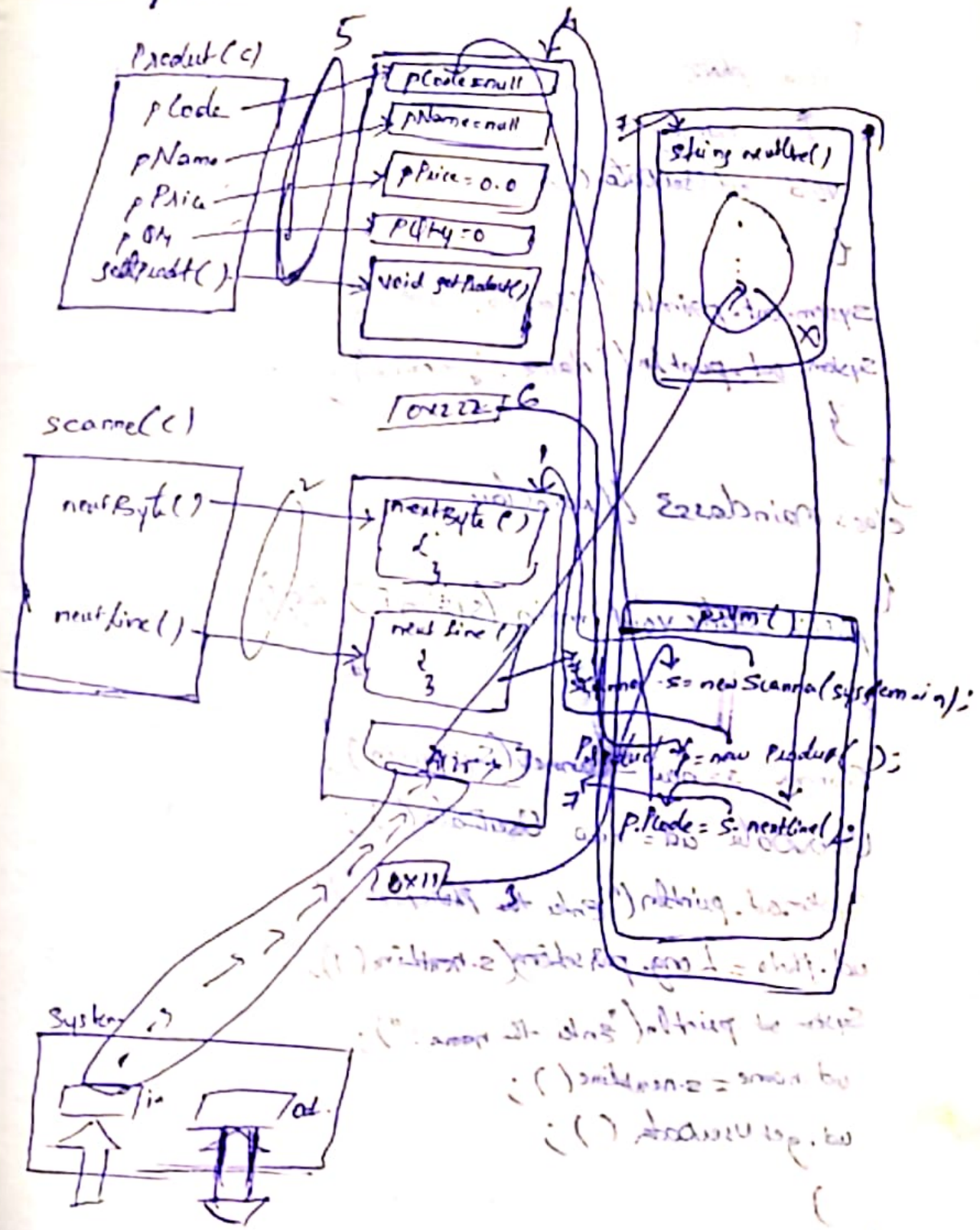
```
import java.lang.*;
import java.util.Scanner;
class Product{ sub class.
{
   String pcode,pname;
   float price;
   int qty;
   void getproduct()
   {
System.out.println("=== Product Info. ===");
System.out.println("pcode: "+pcode);
System.out.println("pname: "+pname);
System.out.println("price: "+price);
System.out.println("qty: "+qty);
   }
}
class MainClass2 //main class
{
   public static void main(String[] args)
   {
Scanner s=new Scanner(System.in);
Product p= new Product(); // Object Creation
System.out.println("Enter the pcode:");
p.pcode = s.nextLine();
System.out.println("Enter the price:");
p.price = s.nextFloat();
System.out.println("Enter the qty:");
p.qty=s.nextInt();
```

```
    }
}
```

Execution flow of above program :-

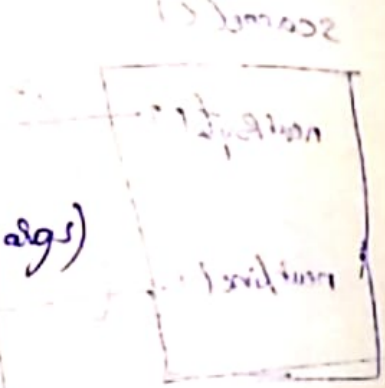
class files :-



15/12/15

WAP to read & display UserData (phNo & name..

```
import java.lang.*;
import java.util.Scanner;
class UserData //sub class
{
  long phNo;
  String name;
  void getUserData()
  {
System.out.println("PhoneNo: "+phNo);
System.out.println("Name : "+name);
  }
}
class mainclass3 //main class
{
  public static void main(String[] args)
  {
  Scanner s=new Scanner(System.in);
  UserData ud=new UserData();
  System.out.println("Enter the PhoneNo:");
  ud.phNo=Long.parseLong(s.nextLine());
  System.out.println("Enter the name:");
   ud.name=s.nextLine();
   ud.getUserData();
    }
  }
```



Note:-

When we read String data after Numeric data from console input, the String data will be skipped. This Disadvantage can be overcomed using the following methods.

Method Signature

public static byte parseByte(java.lang.String);

Syntax:

```
byte b = Byte.parseByte(s.nextLine());
```

public static short parseShort(java.lang.String);

Syntax

```
short s1 = short.parseShort(s.nextLine());
```

public static int parseInt(java.lang.String);

Syntax

```
int i = Integer.parseInt(s.nextLine());
```

public static long parseLong(java.lang.String);

Syntax

```
long l = Long.parseLong(s.nextLine());
```

public static float parseFloat(java.lang.String);

Syntax

```
float f = Float.parseFloat(s.nextLine());
```

public static double parseDouble(java.lang.String);

Syntax

```
double d = Double.parseDouble(s.nextLine());
```

Assignment :-

Update Mainclass1.java program by reading the data from console & display the data.

Methods in Java :-

⇒ Methods are the actions performed on the data in the process of generating result.

⇒ These Methods are categorized into two types:

1. Built-in Methods
2. User Defined Methods.

(1) Built-in Methods :-

⇒ The methods which are available from JavaLib classes and Interfaces are known as Built-in Methods.

⇒ Based on static keyword built-in methods are categorized into two types.

(i) Non-Static Built-in Methods
(ii) Static Built-in Methods.

(i) Non-Static Built-in Methods :-

⇒ The Built-in Methods which are declared without static keyword are known as Non-Static Built in Methods.

Exp: nextLine()
nextInt()
:

(ii) Static Built-in Methods :-

⇒ The built-in Methods which are declared with static keyword are known as static Built-in methods

Exp parseInt()
parseLong()...

(2) User Defined Methods:

⇒ The methods which are defined by the programmer are known as user defined methods. Based on static keyword the user defined methods are of two types. (i) Non-Static User Defined Methods.

(ii) Static user Defined Methods.

(i) Non-Static User Defined Methods:

⇒ The user defined methods which are declared without static keyword are known as Non static Userdefined methods or Instance methods.

⇒ These methods will get the memory within the object creation and accessed with the Object name

Structure of Instance Method:-

```
return_type method_name (para_list)
{
// method body;
}
```

Ques: Define parameters.

⇒ Parameters are the variables which act as data carriers from one method to another method.

⇒ Based on parameters the methods are categorized into two types.

a) Methods without parameters
b) Methods with parameters.

Methods without parameters:

methods which are declared without parameters are known as 0-parameter methods (zero) or methods without parameters.

Exp: above programs.

(MainClass1.java, MainClass2.java, MainClass3.java)

(b) Methods with parameters:

⇒ the methods which are declared with parameters are known as Methods parameterized methods or methods with parameters.

Exp Program:

WAP to read two int values and perform Arithmatic Operation based on User choice.

Note: If the values are greater than zero then perform

```
import java.lang.*;
import java.util.Scanner;
class Addition //subclass1
{
  Void add(int x, int y)
  {
  System.out.println("Sum:"+(x+y));
  }
}
class Subtraction //sub class
{
  void sub(int x, int y)
  {
```

```
System.out.println("sub:"+(x-y));
  }
}
class Multiplication //subclass
{
  void mul(int x, int y)
  {
  System.out.println("Mul:"+(x*y));
  }
}
  class Division //sub class
  {
    void div(int x, int y)
  {
  System.out.println("Div:"+(float)x/y);
  }
  }
  class ModDivision //subclass
  {
    void modDiv(int x, int y)
    {
  System.out.println("modDiv:"+(x/y));
}
}
class Main class //Main class
{
  public static void main (String[] args)
  {
```

```
Scanner s = new Scanner(System.in);
System.out.println("Enter the val1 : ");
int V1 = Integer.parseInt(s.nextLine());
System.out.println("Enter the val2 : ");
int V2 = Integer.parseInt(s.nextLine());
    if (V1>0 && V2>0)
    {
System.out.println("===choice===");
System.out.println
        ("1.add\n2.sub\n3.mul\n4.div\n5.mod");
System.out.println("Enter the choice:");
int choice = Integer.parseInt(s.nextLine());
        switch(choice)
        {
        case 1;
Addition a1 = new Addition();
            a1.add(V1,V2); //method call
            break;
        case 2;
Subtraction S1 = new Subtraction();
            S1.sub(V1,V2); //method call
        break;
        case 3;
Multiplication m1 = new Multiplication();
            m1.mul(V1,V2); //method call
            break;
```

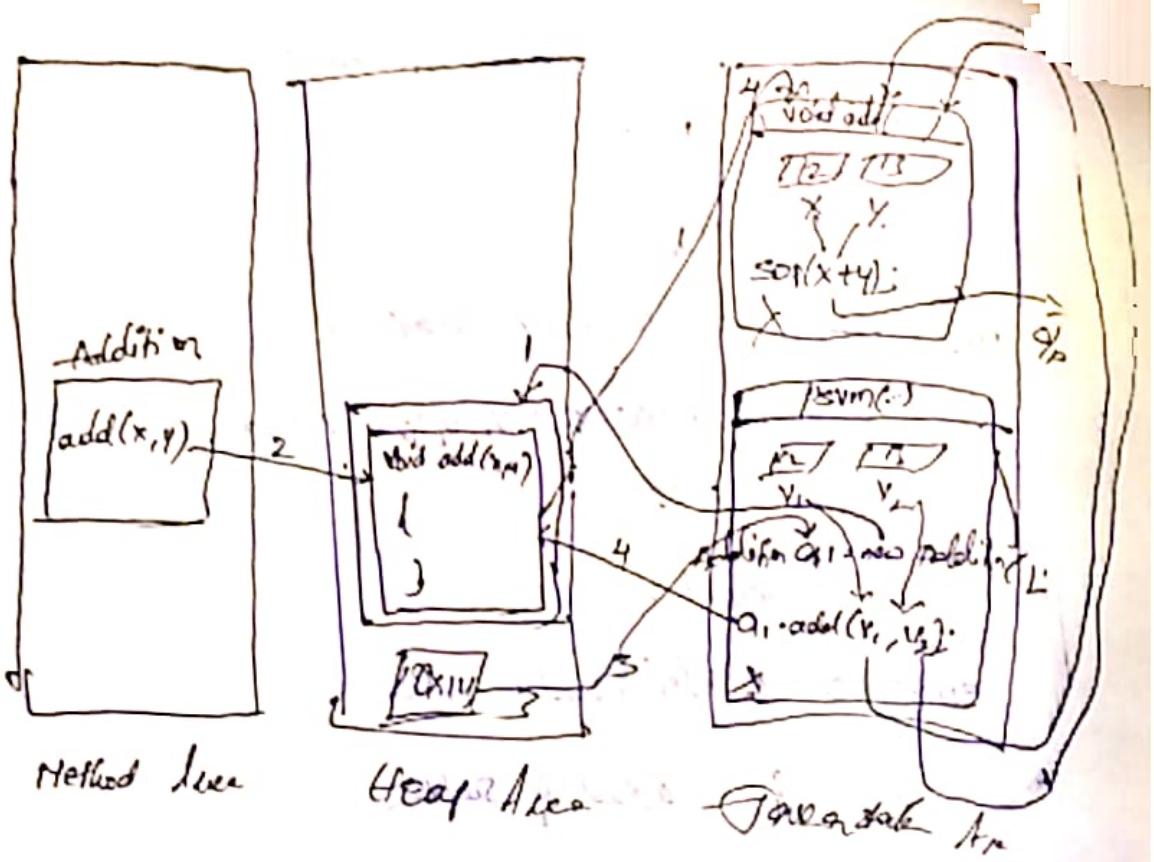
```
        case 4;
Division di=new Division();
            di.div(v1,v2);= //method call
            break;
        case 5;
ModDivision md = new ModDivision();
            md.modDiv(v1,v2); // Method call
            break;
            default :
System.out.println("Invalid choice....");
        } // End of Switch
    } // End of if
    else
    {
System.out.println("Invalid value....");
    }
}
}
```

16/12/19

Execution flow of above diagram :-

Class files

Addition.class
Subtraction.class
Multiplication.class
Division.class
moldivision.class
MainClass4.class (Main Class)



Note:

⇒ "v_1" and "v_2" are actual parameters because they hold the data coming from the console I/p and passed as parameters while method call.

⇒ 'x' and 'y' are formal parameters because they both are declared part of method signature and used in calculation process

⇒ We can have same names in actual parameters and formal parameters.

⇒ In the above prog only addition class is loaded for execution and remaining sub classes are not loaded, in this process the loading time is saved part of execution process & generates high performance

multiple available options & cases.

Behavior :-

⇒ The switch value is compared with the available options and if any option matched with the switch-value then the option is executed and the switch-case execution is stopped using 'break' statement

⇒ If the switch value is not matched with any available option then the default is executed.

Note :-

⇒ In real-time switch-case is used to show the options to the end user and also used in verification process.

Define Return type :-

⇒ 'return_type' specifies the methods returns the value after execution & not.

⇒ Based on return type the methods are categorized into two types

(i) Non-return type methods

(ii) return_type methods.

(i) Non-return type methods :-

⇒ the methods which do not return any value after execution are known as non-return type methods.

Eg: Above programs.

[Mainclass1.java, Mainclass2.java, Main class3.java, Mainclass4.java]

Note:

The methods which are declared with 'void' are known as Non-return type methods.

(ii) Return-type Methods:-

=> The methods which return the value after execution are known as return-type methods.

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

=> The returned value will come back to the method call..

Eg Prog:-

WAP to display employee details?

1. Read empName
2. Read empDesg
   => desg must be 2 characters (validation)
   => desg must be in HO SE, SE and ME (Verification)
3. Read empId
   => Id must be 4 characters (validation)
   => Id must be in the S111, 7222 & M333 (Verification)
4. If desg & id, validated and verified then read bSal.
   => bSal must be min 5000/- (validation)
5. If bSal is validated then calculate totSal.

totSal = bSal + HRA + DA;

HRA = 95% of bSal

DA = 63% of bSal

```
class EmpDesgCheck // subclass
{
  boolean k;
  boolean verify(String desg)
  {
    switch (desg)
    {
      case "SE" : k=true;
      break;
      case "TE" : k=true;
      break
      case "ME" : k=true;
      break;
      default : k=false;
    } // End of switch
    return(k);
  }
}
class EmpId chk // subclass
{
  boolean z;
  boolean verify(String id)
  {
```

```
switch (id)
{
  case "111": z = true;
  break;
  case "222": z = true;
  Break;
  case "333": z = true;
  break;
  default: z = false;
  } // End of switch
return(z);
}
}
class EmpSalary //Subclass
{
  float totsal;
  void cal(int bsal)
  {
totSal = bsal + (0.95f * bsal) + 0.6f * bsal);
  }
  void getTotsal()
  {
System.out.println("totsal:" + totsal);
  }
}
```

```
class Mainclass5 //Main class
{
  public static void main (String[] args)
  {
  Scanner s=new Scanner(System.in);
  System.out.println ("Enter the eName:");
  String eName= s.nextLine( );
  System.out.println ("Enter the desg:");
    String eDesg = s.nextLine( );
      if (eDesg.length( )==2)
      {
    EmpDesgCheck edc = new EmpDesgCheck( );
    boolean k=edc.verify (eDesg.toUpperCase( ));
        if (k)
        {
    System.out.println("Enter the empId:");
    String eId= s.nextLine( )
        if (eId.length( )==4)
        {
    EmpIdCheck eic = new EmpIdCheck( );
    boolean z = eic.verify (eId.toUpperCase( ));
            if (z)
            {
    System.out.println ("Enter the Sal:");
```

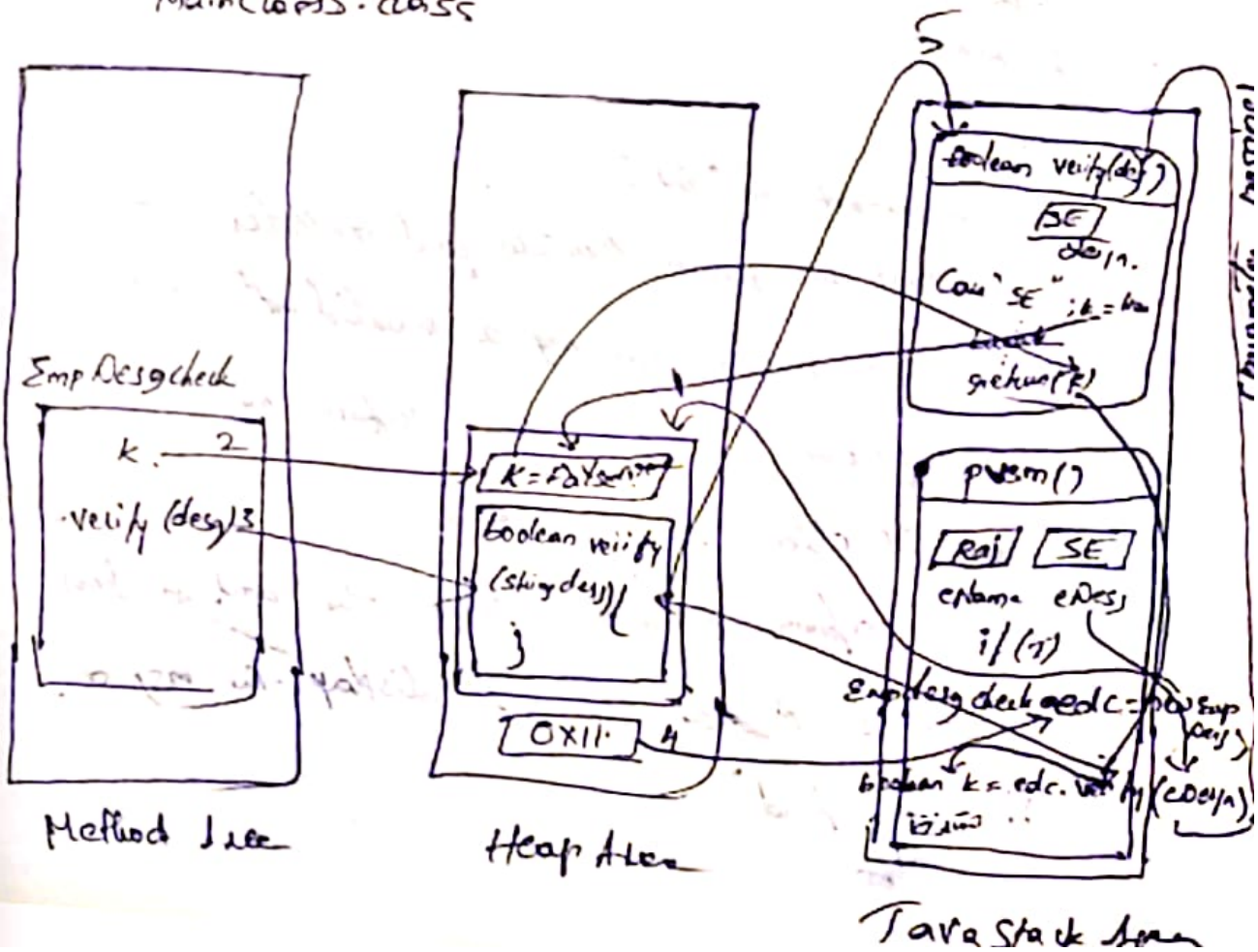
```
int bSal = Integer.parseInt(s.nextLine());
    if (bSal >= 5000)
    {
System.out.println("EmpName:" +eName);
System.out.println("EmpDesg:" +eDesg);
System.out.println("EmpId:" +eId);
System.out.println("Bsal:" +bSal);
EmpSalary es = new EmpSalary();
 es.cal(bSal);
 es.getNetsal();

    } // end of if
    else
{
System.out.println("Invalid bSal...");
    }
  } // end of if
  else
{
System.out.println("Emp Id do not exist...");
    }
  } // end of if
  else
{
System.out.println("Invalid empId...");
  } } end of if
```

```
else
{
System.out.println("Desg doesnot exist...");
    }
} // End of if
else
{
System.out.println("Invalid desg...");
    }
  }
}
```

17/12/19

Execution flow of above prog:-

Compile EmpDesgCheck.class
EmpIdCheck.class
EmpSalary.class
MainClass.class



Define Validation process.

⇒ The process of checking the entered data is in correct format or not is known as validation process.

Define Verification process.

⇒ The process of checking the entered data is available or not is known as verification process.

Assignment →

WAP to perform Bank transaction process.

a. Read pinNo

⇒ pinNo must be 4 digits. (Validation)

⇒ pinNo must be in the 1111, 2222 and 3333 (verification)

b. After pinNo validation and verification, show the following options.

1. WithDraw
2. Deposit.

1. Withdraw:

⇒ Enter the amount in "int".

⇒ The amt must be greater than zero and multiples of 100, if not display the msg as "Invalid amt"

⇒ If the amount is valid then perform the withdraw operation using method.

⇒ Before performing operation check the amt is less than 2000 or not, if not display the msg as "Insufficient fund".

Ctrl

Class Name: Withdraw

⇒ void wDraw(int amt).

---

2. Deposit :

⇒ Read the amount in "int"

⇒ the amt must be greater than zero and multiple of 100, if not display the msg as "Invalid amt"

⇒ If the amount is valid then perform deposit operation in method.

Class Name : Deposit

⇒ void deposit(int amt)

O/p:

Withdraw

⇒ Amt WithDrawn :

⇒ Bal Amt :

⇒ Transaction Completed.

Deposit

⇒ Amt Deposited :-

⇒ Bal Amt :

⇒ Transaction completed

ClassNew : pinCheck

⇒ boolean b

⇒ boolean Verify(int pinNo)

---

nit.venkatesh@gmail.com.

Class (Subclass) generating Multiple Objects

⇒ The class can generate any number of objects without restrictions.

⇒ The multiple objects which are generated from the class are independent by their memory location on HeapArea.

⇒ The modification done in one object will not effect remaining multiple objects.

Exp! program

```
class Display //Sub class
{
  int k=10; //Instance Variable
  void dis() //Instance method
  {
    k++
    System.out.println("The val k:"+k);
  }
}
class MainClass6 //main class
{
  public static void main(String[] args)
  {
    Display d1= new Display(); //object
```

```
Display d2 = new Display( ); //object 2
d1.dis( ); //method call
d2.dis( ); //method call
  }
}
```