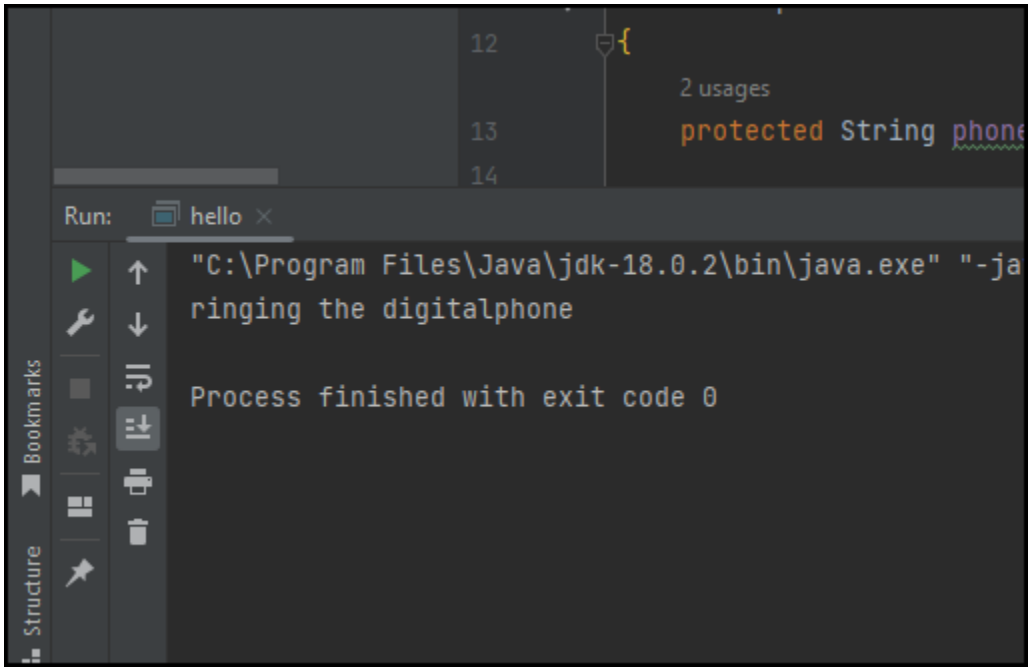
Hands on Practice for Java

1.Create a base class, `Telephone`, and derive a class `ElectronicPhone` from it. In `Telephone`, create a protected string member `phonetype`, and a public method `Ring( )` that outputs a text message like this: "Ringing the <phonetype>." In `ElectronicPhone`, the constructor should set the `phonetype` to "Digital." In the Run( ) method, call Ring( ) on the ElectronicPhone to test the inheritance.

```
package com.example.hello;
import java.util.*;
import java.lang.*  ;

// sout-- printing single lines
// psvm --> main file

import java.util.Scanner;


class telephone
{
   protected String phonetype;

   public void ring()
   {
       System.out.println("ringing the " + phonetype + "phone" );
   }
}

class electronicphone extends  telephone {

   public electronicphone(){
       phonetype = "digital";
       }

public void run(){
       ring();
   }
}

public class hello {

   public static void main(String[] args) {
       electronicphone ep = new electronicphone();
       ep.run();
   }
}
```



2). Extend Exercise 1 to illustrate a polymorphic method. Have the derived class override the Ring( ) method to display a different message.

```
package com.example.hello;
import java.util.*;
import java.lang.*  ;

// sout-- printing single lines
// psvm --> main file

import java.util.Scanner;
class telephone
{
   protected String phonetype;
   public  void ring()
{
   System.out.println("ringing the " + phonetype);
}
}
class electronicphone extends telephone
       {
public electronicphone()
       {
       phonetype = "digital";
```

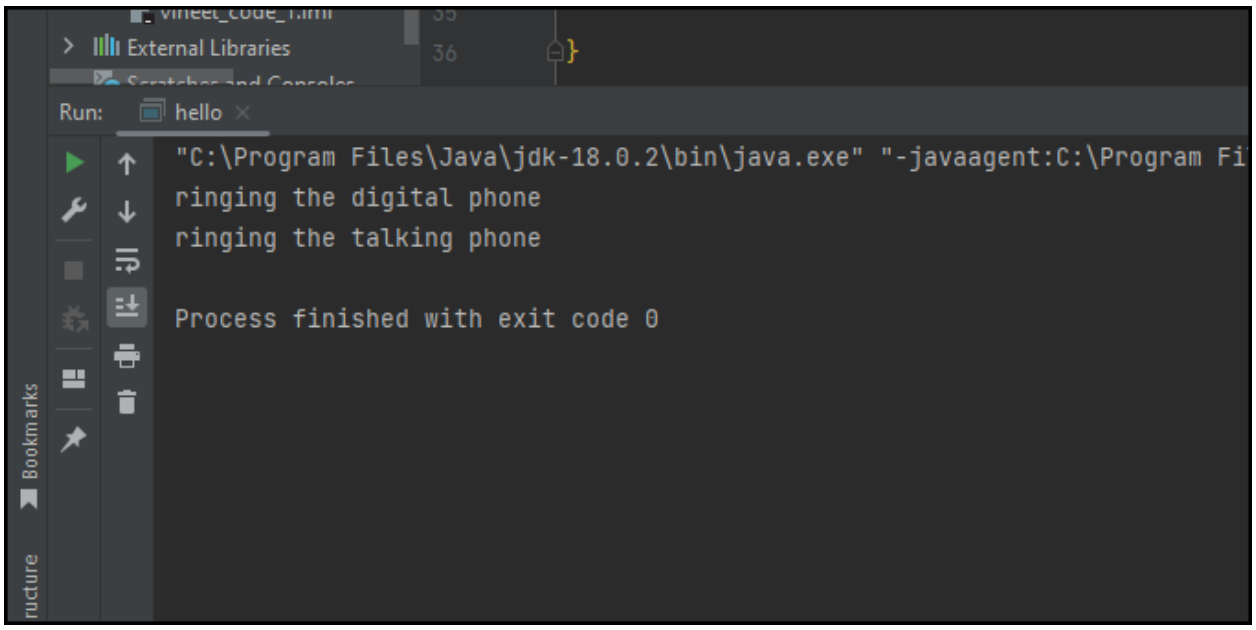
```
        }
public void run()
        {
        ring();
        }
public  void ring()
        {
            System.out.println("the "  + phonetype  + "phone is using a
different ring-tone");
        }
        }
public class hello {
   public static void main(String[] args) {
        electronicphone ep = new electronicphone();
        ep.run();
    }
}
```

```
Run:  hello
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" "-javaagent:C:\Pro
the digitalphone is using a different ring-tone

Process finished with exit code 0
```

3) Change the Telephone class to abstract, and make Ring( ) an abstract method. Derive two new classes from Telephone: DigitalPhone and TalkingPhone. Each derived class should set the phonetype, and override the Ring( ) method.

```
package com.example.hello;
import java.util.*;
import java.lang.*  ;

// sout-- printing single lines
// psvm --> main file

import java.util.Scanner;

abstract class telephone
{
   protected String phonetype;

   public abstract void ring();
}

class digitalphone extends telephone
{

   public digitalphone()
   {
      phonetype = "digital";
   }
   public  void ring()
   {
      System.out.println("ringing the "  + phonetype +  " phone");
   }
   public void run()
   {
      ring();
   }
}
class talkingphone  extends telephone {
   public talkingphone() {
      phonetype = "talking";
   }
   public void ring() {
      System.out.println("ringing the " + phonetype + " phone");
   }
   public void run() {
      ring();
```

```
  }
}
public class hello {
  public static void main(String[] args) {
      digitalphone dp = new digitalphone();
      dp.run();
      talkingphone tp = new talkingphone();
      tp.run();
  }
}
```



4) A Bank

Look at the `Account` class Account.java write a `main` method in a different class to briefly experiment with some instances of the `Account` class.

- Using the `Account` class as a base class, write two derived classes called `SavingsAccount` and `CurrentAccount`. A `SavingsAccount` object, in addition to the attributes of an `Account` object, should have an interest variable and a method which adds interest to the account. A `CurrentAccount` object, in addition to the attributes of an `Account` object, should have an overdraft limit variable. Ensure that you have overridden methods of the `Account` class as necessary in both derived classes.

- Now create a `Bank` class, an object of which contains an array of `Account` objects. Accounts in the array could be instances of the `Account` class, the `SavingsAccount` class, or the `CurrentAccount` class. Create some test accounts (some of each type).
- Write an update method in the bank class. It iterates through each account, updating it in the following ways: Savings accounts get interest added (via the method you already wrote); CurrentAccounts get a letter sent if they are in overdraft.
- The `Bank` class requires methods for opening and closing accounts, and for paying a dividend into each account.

- Note that the balance of an account may only be modified through the `deposit(double)` and `withdraw(double)` methods.
- The `Account` class should not need to be modified at all.

Be sure to test what you have done after each step.

```
package com.example.hello;
import java.util.*;
import java.lang.*  ;

// sout-- printing single lines
// psvm --> main file

import java.util.Scanner;


class BankAccount
{
   private double balance;
   private String accno;
   private String name;

   Scanner sc = new Scanner(System.in);
   public BankAccount()
   {
       balance = 0.0;
       accno = "Null" ;
       name = "default_name" ;
   }
   public void openAccount() {
       System.out.print("Enter Account No: ");
```

```
        accno = sc.next();
        System.out.print("Enter Name: ");
        name = sc.next();
        System.out.print("Enter Balance: ");
        balance = sc.nextDouble();
    }
    public void showAccount() {
        System.out.println("Name of account holder: " + name);
        System.out.println("Account no.: " + accno);
        System.out.println("Balance: " + balance);
    }

    public void deposit(double amount) {

        //System.out.println("Enter the amount you want to deposit: ");
        double amt = amount ;

        //amt = sc.nextDouble();
        balance = balance + amt;
    }



    public void withdrawal() {
        double amt;
        System.out.println("Enter the amount you want to withdraw: ");
        amt = sc.nextDouble() ;

        if (balance >= amt) {
            balance = balance - amt;
            System.out.println("Balance after withdrawal: " + balance);
        } else {
            System.out.println("Your balance is less than " + amt +
"\tTransaction failed...!!" );
        }
    }

    public double getBalance()
    {
        return balance;
    }
}


//Savings account::

class SavingsAccount extends BankAccount {
    private double interest;
    public SavingsAccount() {
        System.out.println("Enter the interest amount ::");
        interest = sc.nextDouble() ;

    }
    public void addInterest() {
        deposit(getBalance() * interest);
    }
```

```
}
//Current Account::
class CurrentAccount extends BankAccount {
   private boolean overdraft;

   public CurrentAccount() {
       System.out.println("Enter true or false for overdraft");
       overdraft = sc.nextBoolean() ;
   }
   public void Checkov(){
       if(overdraft)
           System.out.println("Send the letter");
       else
           System.out.println("Do not Send the letter");
   }
}
public class hello {
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.print("How many number of customers(bank) do you want to
input? ");
       int n = sc.nextInt();


       System.out.print("How many number of customers(savings) do you want to
input? ");
       int a = sc.nextInt();


       System.out.print("How many number of customers(current) do you want to
input? ");
       int b = sc.nextInt();

       BankAccount C[] = new BankAccount[n] ;
       SavingsAccount D[] = new SavingsAccount[a] ;
       CurrentAccount E[] = new CurrentAccount[b] ;

       if(n!=0) {
           System.out.println("BankAccount info");
           for (int i = 0; i < C.length; i++) {
               C[i] = new BankAccount();
               C[i].openAccount();
           }
       }

       if(a!=0) {
           System.out.println("Saving acc info :: ");
           for (int i = 0; i < D.length; i++) {
               D[i] = new SavingsAccount();
               D[i].openAccount();
               D[i].addInterest();
               D[i].getBalance() ;
           }
       }
       if(b!=0) {
           System.out.println("Current Account info");
```

```
            for (int i = 0; i < E.length; i++) {
                E[i] = new CurrentAccount();
                E[i].openAccount();
                E[i].Checkov();
            }
        }
    }
}
```

```
Run: hello
Enter the interest amount ::
100
Enter Account No: 101
Enter Name: acb
Enter Balance: 2000
Current Account info
Enter true or false for overdraft
true
Enter Account No: 2011
Enter Name: bnvv
Enter Balance: 300
```

```
How many number of customers(bank) do you want to input? 0
How many number of customers(savings) do you want to input? 1
How many number of customers(current) do you want to input? 1
Saving acc info ::
Enter the interest amount ::
1.56
Enter Account No: 100AB
Enter Name: vineet
Enter Balance: 50000
Balance : 128000.0
Current Account info
Enter true or false for overdraft
true
Enter Account No: 100BV
Enter Name: alex
Enter Balance: 20000
Send the letter

Process finished with exit code 0
```

5) Employees

Create a class called `Employee` whose objects are records for an employee. This class will be a derived class of the class Person which you will have to copy into a file of your own and compile. An employee record has an employee's name (inherited from the class `Person`), an annual salary represented as a single value of type `double`, a year the employee started work as a single value of type `int` and a national insurance number, which is a value of type `String`.

Your class should have a reasonable number of constructors and accessor methods, as well as an `equals` method. Write another class containing a `main` method to fully test your class definition.

```java
package com.example.hello;
import java.util.*;
import java.lang.*  ;

// sout-- printing single lines
// psvm --> main file

import java.util.Scanner;
class Employee
{
   private String firstName;    private String lastName;
   private double monthlySalary;
   /
   public Employee (String fname, String lname, double msalary)
   {
       firstName = fname;
       lastName = lname;        monthlySalary = msalary;
       if (msalary < 0.0)
           monthlySalary = 0.0;
   }
      public void setFirstName (String fname)
   {
       firstName = fname;
   }
      public String getFirstName ()
   {
       return firstName;
   }
   public void setLastName (String lname)
   {
       lastName = lname;
   }
   public String getLastName ()
   {
       return lastName;
   } // end method getLastName

   // method to set the monthly salary
   public void setMonthlySalary (double msalary)
   {
       monthlySalary = msalary;   // store the monthly salary
   } // end method setMonthlySalary
   // method to retrieve monthly salary
   public double getMonthlySalary ()
   {
       return monthlySalary;
   }
   public double getYearlySalary()
   {
       double yearlySalary = monthlySalary * 12;
       return yearlySalary;
   }
   public double getRaiseSalary()
```

```java
    {
        double raise =  monthlySalary * 0.1 ;
        double raiseSalary = ( monthlySalary + raise ) * 12;
        return raiseSalary;
    } // end method getRaiseSalary
}
public class hello {
    public static void main(String[] args) {
        Employee e = new Employee("vineet" , "verma" , 50000.560) ;
        System.out.println("first name ::" +  e.getFirstName()) ;
        System.out.println("last name ::" + e.getLastName())  ;
        System.out.println("month salary ::" + e.getMonthlySalary()) ;
        System.out.println("Yearly salary :: " + e.getYearlySalary() ) ;
        System.out.println("Raised Salary :: " + e.getRaiseSalary()) ;
    }
}
```

```
Run:  hello
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrain
first name ::vineet
last name ::verma
month salary ::50000.56
Yearly salary :: 600006.72
Raised Salary :: 660007.392

Process finished with exit code 0
```

6) The LotteryAccount uses an instance of a Lottery object for adding interests. Under some lucky circumstances, the owner of a LotteryAccount will get a substantial amount of interest. In most cases, however, no interests will be added.

There exists a single file which contains the classes BankAcount, CheckAccount, SavingsAccount, Lottery, together with a sample client class. (Note:- Define these classes with proper behaviours and properties).

Program a specialization of the LotteryAccount, called LotteyPlusAccount, with the following redefinitions of Deposit and Withdraw.

· The Deposit method doubles the deposited amount in case you draw a winning lottery number upon deposit. If you are not lucky, Deposit works as in LottoryAccount, but an administrative fee of 1500 will be withdrawn from your LotteryPlusAccount.

- The Withdraw method returns the withdrawn amount without actually deducting it from the LotteryPlusAccount if you draw a winning lottery number upon withdrawal. If you are not lucky, Withdraw works as in LottoryAccount, and an additional administrative fee of 500 will be withdrawn from the account as well.

Notice that the Deposit and Withdraw methods in LotteryPlusAccount should combine with the method in LotteryAccount (method combination). Thus, use the Deposit and Withdraw methods from LotteryAccount as much as possible when you program the LotteryPlusAccount.

Test-drive the class LotteryPlusAccount from a sample client class.