一、 实验名称

Inheritance

二、 实验目的

- Be able to derive a class from an existing class
- Be able to define a class hierarchy in which methods are overridden and fields are hidden
- Be able to use derived-class objects
- Implement a copy constructor

三、 实验内容

Task #1 Extending BankAccount
Task #2 Creating a Second Subclass

四、 实验方法(原理、流程图)

1. Written by Intellij IDEA Community edition 2020.3

2.

Task 1:

（1）Copy the files AccountDriver.java (code listing 10.1) and BankAccount.java (code listing 10.2) from the Student CD

（2）Create a new class called CheckingAccount that extends BankAccount.

（3）Create a static constant FEE that represents the cost of clearing one check. Set it equal to 15 cents.

（4）Write a constructor that takes a name and an initial amount as parameters. It should call the constructor for the superclass. It should initialize accountNumber to be the current value in accountNumber concatenated with –10 (All checking accounts at this bank are identified by the extension –10). There can be only one checking account for each account number.

(5) Write a new instance method, withdraw, that overrides the withdraw method in the superclass. This method should take the amount to withdraw, add to it the fee for check clearing, and call the withdraw method from the superclass. Remember that to override the method, it must have the same method heading. Notice that the withdraw method from the superclass returns true or false depending if it was able to complete the withdrawal or not.

Task 2:

（1）Create a new class called SavingsAccount that extends BankAccount

（2）Create an instance variable called rate that represents the annual interest rate. Set it equal to 2.5%.

（3）Create an instance variable called savingsNumber, initialized to 0.

（4）In this bank, you have one account number, but can have several savings accounts with that same number. Each individual savings account is identified by the number following a dash. For example, 100001-0 is the first savings account you

open, 100001-1 would be another savings account that is still part of your same account.

（5）Create an instance variable called accountNumber that will hide the accountNumber from the superclass.

（6）Write a constructor that takes a name and an initial balance as parameters and calls the constructor for the superclass. It should initialize accountNumber to be the current value in the superclass accountNumber (the hidden instance variable) concatenated with a hyphen and then the savingsNumber.

（7）Write a method called postInterest that has no parameters and returns no value. This method will calculate one month's worth of interest on the balance and deposit it into the account.

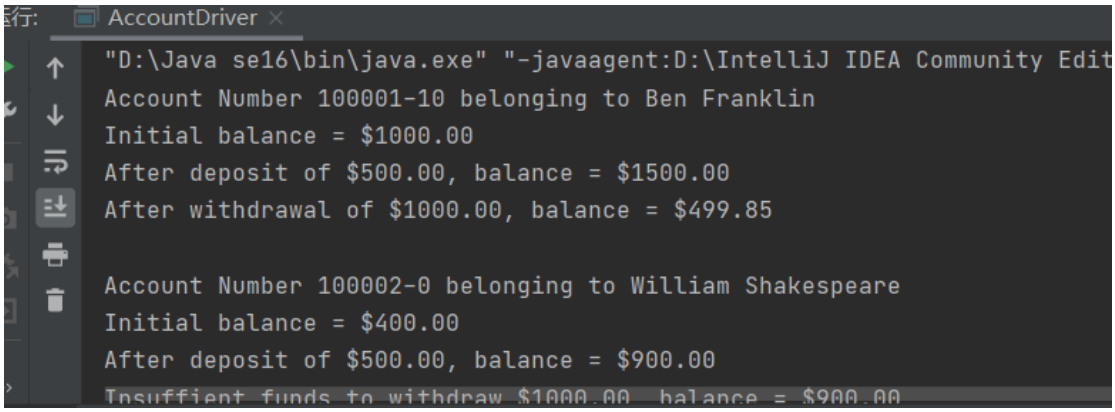
（8）Write a method that overrides the getAccountNumber method in the superclass.

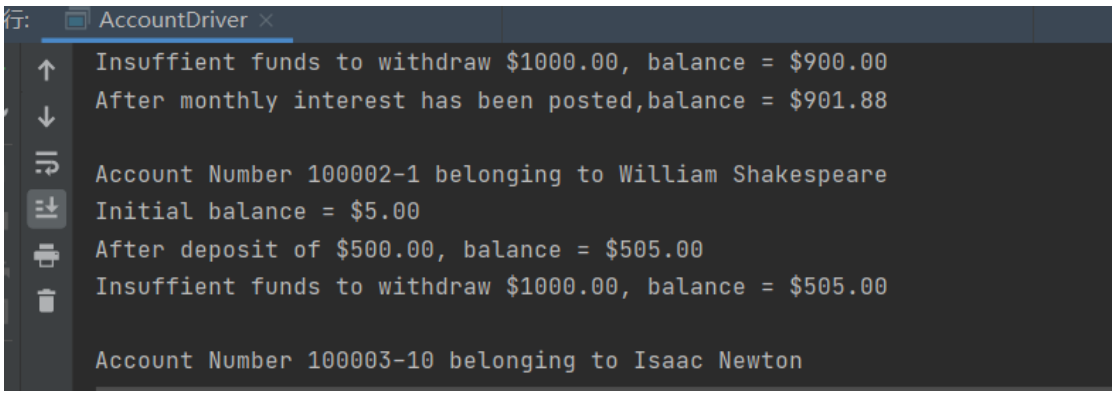
（9）Write a copy constructor that creates another savings account for the same person. It should take the original savings account and an initial balance as parameters. It should call the copy constructor of the superclass, assign the savingsNumber to be one more than the savingsNumber of the original savings account. It should assign the accountNumber to be the accountNumber of the superclass concatenated with the hypen and the savingsNumber of the new account.

五、 实验结论

The experimental requirements have been successfully realized.

The given results are same as those calculated by my codes.





六、实验体会和收获

1. By writing these code,I have a deeper realize about inheritance,subclass and superclass and other relationships between classes.

2. By finishing this task,I become more familiar with how to inherirance attributes and operations from superclass.

3. When writing these codes,i meet some difficults.But I solve these problemss by search relevant information on the Internet and review PPT.

4. By writing this task I am more interested in Java.

七、程序代码

(1) CheckingAccount.java

```
class CheckingAccount extends BankAccount
{
    private final double FEE = 0.15;
    public CheckingAccount(String name,double amount)
    {
        super(name, amount);
        setAccountNumber(getAccountNumber()+"-10");
    }
    @Override
    public boolean withdraw(double amount)
    {
        boolean completed = true;
        double balance=getBalance();
       if(amount+FEE <= balance)
       {
```

```
            setBalance(balance-amount-FEE);
        }
        else
        {
            completed = false;
        }
        return completed;
    }
}
```

(2) SavingsAccount. java

```
public class SavingsAccount extends BankAccount
{
    private final double rate = 0.025;
    private int savingsNumber = 0;
    private final String accountNumber = getAccountNumber();
    public SavingsAccount(String name,double amount)
    {
        super(name,amount);
        setAccountNumber(accountNumber+"-"+savingsNumber);
    }
    public void postInterest()
    {
```

```
        double interest;
        double balance = getBalance();
        interest = balance*rate;
        deposit(interest/12);
    }
    @Override
    public String getAccountNumber()
    {
        return super.getAccountNumber();
    }
    public SavingsAccount(SavingsAccount oldAccount,double amount)
    {
        super(oldAccount,amount);
        setAccountNumber(oldAccount.accountNumber+"-"+(savingsNumber += 1));
    }
}
```

(3) BankAccount. java

```
/**Defines any type of bank account*/
public abstract class BankAccount
{
    /**class variable so that each account has a unique number*/
    protected static int numberOfAccounts = 100001;
```

```
    /**current balance in the account*/
    private double balance;
    /** name on the account*/
    private String owner;
    /** number bank uses to identify account*/
    private String accountNumber;
    /**default constructor*/
    public BankAccount()
    {
        balance = 0;
        accountNumber = numberOfAccounts + "";
        numberOfAccounts++;
    }
    /**standard constructor
      @param name the owner of the account
      @param amount the beginning balance*/
    public BankAccount(String name, double amount)
    {
        owner = name;
        balance = amount;
        accountNumber = numberOfAccounts + "";
        numberOfAccounts++;
```

```
    }
    /**copy constructor creates another account for the same owner
      @param oldAccount the account with information to copy
      @param amount the beginning balance of the new account*/
    public BankAccount(BankAccount oldAccount, double amount)
    {
        owner = oldAccount.owner;
        balance = amount;
        accountNumber = oldAccount.accountNumber;
    }
    /**allows you to add money to the account
      @param amount the amount to deposit in the account*/
    public void deposit(double amount)
    {
        balance = balance + amount;
    }
    /**allows you to remove money from the account if
      enough money is available,returns true if the transaction was
      59
      ©2013 Pearson Education, Inc. Upper Saddle River, NJ. All Rights Reserved.
      completed, returns false if the there was not enough money.
      @param amount the amount to withdraw from the account
```

```
     @return true if there was sufficient funds to complete
     the transaction, false otherwise*/
    public boolean withdraw(double amount)
    {
        boolean completed = true;
        if (amount <= balance)
        {
            balance = balance - amount;
        }
        else
        {
            completed = false;
        }
        return completed;
    }
    /**accessor method to balance
     @return the balance of the account*/
    public double getBalance()
    {
        return balance;
    }
    /**accessor method to owner
```

```
     @return the owner of the account*/
    public String getOwner()
    {
        return owner;
    }
    /**accessor method to account number
     @return the account number*/
    public String getAccountNumber()
    {
        return accountNumber;
    }
    /**mutator method to change the balance
     @param newBalance the new balance for the account*/
    public void setBalance(double newBalance)
    {
        balance = newBalance;
    }
    /**mutator method to change the account number
     @param newAccountNumber the new account number*/
    public void setAccountNumber(String newAccountNumber)
    {
        accountNumber = newAccountNumber;
```

```
    }
}
```

(4) AccountDriver

```
import java.text.*; // to use Decimal Format
/**Demonstrates the BankAccount and derived classes*/
public class AccountDriver
{
    public static void main(String[] args)
    {
        double put_in = 500;
        double take_out = 1000;
        DecimalFormat myFormat;
        String money;
        String money_in;
        String money_out;
        boolean completed;
        // to get 2 decimals every time
        myFormat = new DecimalFormat("#.00");
        //to test the Checking Account class
        CheckingAccount myCheckingAccount =
                new CheckingAccount ("Ben Franklin", 1000);
        System.out.println ("Account Number "
```

```
                + myCheckingAccount.getAccountNumber() +
                " belonging to " + myCheckingAccount.getOwner());
        money = myFormat.format(myCheckingAccount.getBalance());
        System.out.println ("Initial balance = $" + money);
        myCheckingAccount.deposit (put_in);
        money_in = myFormat.format(put_in);
        money = myFormat.format(myCheckingAccount.getBalance());
        System.out.println ("After deposit of $" + money_in
                + ", balance = $" + money);
        completed = myCheckingAccount.withdraw(take_out);
        money_out = myFormat.format(take_out);
        money = myFormat.format(myCheckingAccount.getBalance());
        if (completed)
        {
            System.out.println ("After withdrawal of $" + money_out
                    + ", balance = $" + money);
        }
        else
        {
            System.out.println ("Insuffient funds to withdraw $"
                    + money_out + ", balance = $" + money);
        }
```

```
        System.out.println();
        //to test the savings account class
        SavingsAccount yourAccount =
                new SavingsAccount ("William Shakespeare", 400);
        System.out.println ("Account Number "
                + yourAccount.getAccountNumber() +
                " belonging to " + yourAccount.getOwner());
        money = myFormat.format(yourAccount.getBalance());
        System.out.println ("Initial balance = $" + money);
        yourAccount.deposit (put_in);
        money_in = myFormat.format(put_in);
        money = myFormat.format(yourAccount.getBalance());
        System.out.println ("After deposit of $" + money_in
                + ", balance = $" + money);
        completed = yourAccount.withdraw(take_out);
        money_out = myFormat.format(take_out);
        money = myFormat.format(yourAccount.getBalance());
        if (completed)
        {
            System.out.println ("After withdrawal of $" + money_out
                    + ", balance = $" + money);
        }
```

```
        else
        {
            System.out.println ("Insuffient funds to withdraw $"
                    + money_out + ", balance = $" + money);
        }
        yourAccount.postInterest();
        money = myFormat.format(yourAccount.getBalance());
        System.out.println ("After monthly interest has been posted,"
                + "balance = $" + money);
        System.out.println();
        // to test the copy constructor of the savings account class
        SavingsAccount secondAccount =
                new SavingsAccount (yourAccount,5);
        System.out.println ("Account Number "
                + secondAccount.getAccountNumber()+
                " belonging to " + secondAccount.getOwner());
        money = myFormat.format(secondAccount.getBalance());
        System.out.println ("Initial balance = $" + money);
        secondAccount.deposit (put_in);
        money_in = myFormat.format(put_in);
        money = myFormat.format(secondAccount.getBalance());
        System.out.println ("After deposit of $" + money_in
```

```
                + ", balance = $" + money);
        secondAccount.withdraw(take_out);
        money_out = myFormat.format(take_out);
        money = myFormat.format(secondAccount.getBalance());
        if (completed)
        {
            System.out.println ("After withdrawal of $" + money_out
                    + ", balance = $" + money);
        }
        else
        {
            System.out.println ("Insuffient funds to withdraw $"
                    + money_out + ", balance = $" + money);
        }
        System.out.println();
        //to test to make sure new accounts are numbered correctly
        CheckingAccount yourCheckingAccount =
                new CheckingAccount ("Isaac Newton", 5000);
        System.out.println ("Account Number "
                + yourCheckingAccount.getAccountNumber()
                + " belonging to "
                + yourCheckingAccount.getOwner());
```

```
    }
```

```
}
```