1. A Bank

public class Account  
{  
  private double balance; //The current balance  
  private int acctNum; //The account number  public Account (int num)  
  {  
     balance = 0.0;  
     acctNum = num;  
  }  
  public void deposit (double amt)  
  {  
     if (amt>0)  
        balance += amt;  
     else  
        System.out.println("Can not deposit negative amount!");  
     // add the amount to the balance only if positive  
  }  
  public void withdraw (double amt)  
  {  
     if (amt>0)  
        balance -=amt;  
     else  
        System.out.println("Can not withdraw negative amount!");  
     // subtract the amount from the balance only if poisitve  
  }  
  public double getBalance()  
  {  
     return balance; //returns the balance  
  }  
  public int getAccountNumber()  
  {  
     return acctNum; //returns the account number  
  }  
  @Override  
  public String toString()  
  {  
     return "Account number is " + acctNum + " the balance is " + balance + ".";  
  }  
  public final void print ()  
  {  
     //Don't override this, override the toString method above  
     System.out.println( toString());  
  }  
}

Look at the Account class Account.java and 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.

Hints:

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.

2. Create an inheritance hierarchy of Rodent: Mouse, Gerbil, Hamster, etc. In the base class, provide methods that are common to all Rodents, and override these in the derived classes to perform different behaviors depending on the specific type of Rodent. Create an array of Rodent, fill it with different specific types of Rodents, and call your base-class methods to see what happens.

3. Modify exercise 2 so that it demonstrates the order of initialization of the base classes and derived classes. Now add member objects to both the base and derived classes, and show the order in which their initialization occurs during construction.

4 -Create an interface with at least 3 methods that describes behavior of a real life object.

- create at least 3 different implementations of that interfaces.

- create class that contain one static method that will return different implementation of the interface without changing method signature

-in main method call the static method you just created and show that code does not change but behaves differently based on implementation

5 -Create an abstract class with at least 3 abstract methods and one non abstract that describes behavior of a real life object.

- create at least 3 different implementations of that class.

- create class that contain one static method that will return different implementation of the abstract class without changing method signature

-in main method call the static method you just created and show that code does not change but behaves differently based on implementation