



Department of Computer Science and Engineering (Data Science)

NAME: **Bhuvi Ghosh**

BATCH: **D22**

SAP ID: **60009210191**

COURSE CODE: **DJ19DSL504**

COURSE NAME: **Java and Scala Laboratory**

CLASS: **T.Y.B.Tech (D2)**

Experiment No 4

Aim: -To Implement Inheritance and Interface in Java Theory: - 1. Inheritance in Java

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends Keyword

extends is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

Syntax

```
class Super {  
    ....  
    ....  
} class Sub extends Super  
{  
    ....  
    ....  
}
```

The new class that is created is known as subclass (child or derived class) and the existing class from where the child class is derived is known as superclass (parent or base class).

The extends keyword is used to perform inheritance in Java.

For example,

```
class Animal { //  
    methods and fields  
}
```

```
// use of extends keyword //  
to perform inheritance  
class Dog extends Animal {
```



```
// methods and fields of Animal
```

```
// methods and fields of Dog
```

```
}
```

In the above example, the Dog class is created by inheriting the methods and fields from the Animal class. Here, Dog is the subclass and Animal is the superclass.

is-a relationship

In Java, inheritance is an is-a relationship. That is, we use inheritance only if there exists an is-a relationship between two classes. For example,

Car is a Vehicle

Orange is a Fruit

Surgeon is a Doctor

Dog is an Animal

Here, Car can inherit from Vehicle, Orange can inherit from Fruit, and so on.

super Keyword in Java Inheritance

Super Keyword in Inheritance

Previously we saw that the same method in the subclass overrides the method in superclass.

In such a situation, the super keyword is used to call the method of the parent class from the method of the child class. Example

class Animal {

```
// method in the superclass
```

```
public void eat() {
```

```
    System.out.println("I can eat");
```

```
}
```

```
}
```

```
// Dog inherits Animal class
```

```
Dog extends Animal {
```

```
    // overriding the eat() method
```

```
    @Override
```

```
public void eat() {
```

```
    // call method of superclass
```

```
    super.eat();
```

```
    System.out.println("I eat dog food");
```

```
}
```

```
// new method in subclass
```

```
public void bark() {
```

```
    System.out.println("I can bark");
```

```
}
```

```
}
```

```
class Main {    public static void
```

```
main(String[] args) {
```

```
    // create an object of the subclass
```

```
    Dog labrador = new Dog();
```

```
    // call the eat() method
```

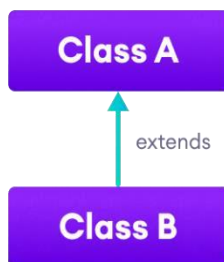


```
labrador.eat();  
labrador.bark();  
}  
}
```

2. Types of Inheritance in Java

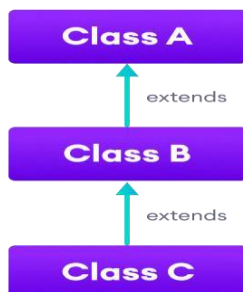
1. Single Inheritance

In single inheritance, a single subclass extends from a single superclass. For example,



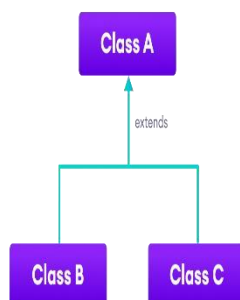
2. Multilevel Inheritance

In multilevel inheritance, a subclass extends from a superclass and then the same subclass acts as a superclass for another class. For example,



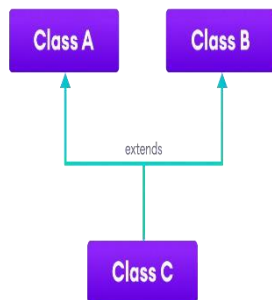
3. Hierarchical Inheritance

In hierarchical inheritance, multiple subclasses extend from a single superclass. For example,



4. Multiple Inheritance

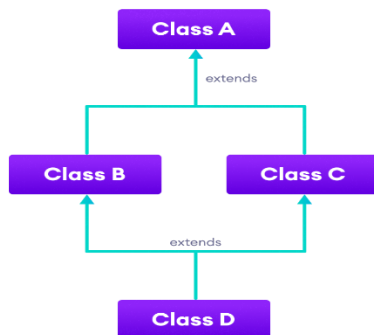
In multiple inheritance, a single subclass extends from multiple super classes. For example,



Note: Java doesn't support multiple inheritance. However, we can achieve multiple inheritance using interfaces. To learn more, visit [Java implements multiple inheritance](#).

5. Hybrid Inheritance

Hybrid inheritance is a combination of two or more types of inheritance. For example,



Here, we have combined hierarchical and multiple inheritance to form a hybrid inheritance.

2. Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java. In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body. Java Interface also **represents the IS-A relationship**. **It cannot be instantiated just like the abstract class**. Since Java 8, we can have default and static methods in an interface. Since Java 9, we can have private methods in an interface.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling.

How to declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.



Syntax:

```
interface <interface name> {
```

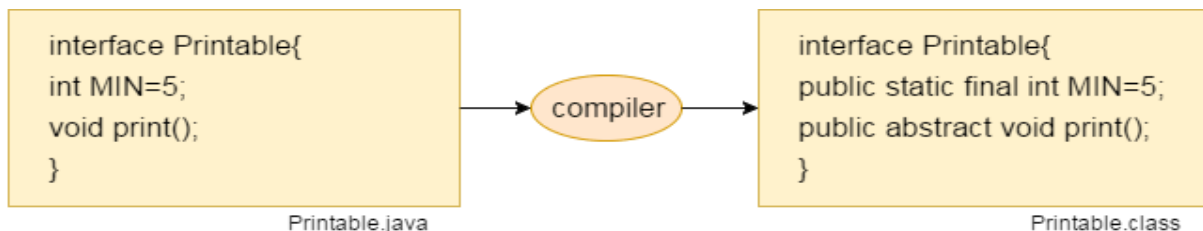
```
    // declare constant fields    //  
    declare methods that abstract  
    // by default.  
}
```

Internal addition by the compiler

The **Java compiler** adds **public** and **abstract** keywords before the interface method.

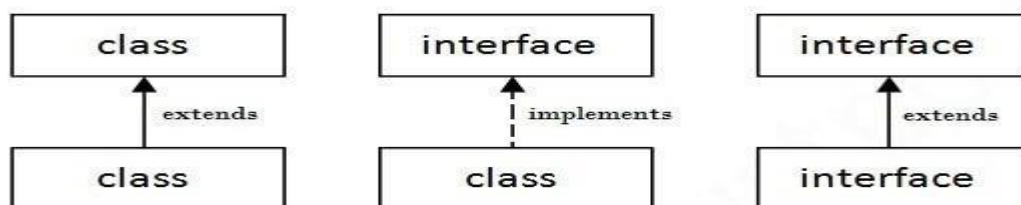
Moreover, it adds **public**, **static** and **final** keywords before data members.

In other words, Interface fields are public, static and final by default, and the methods are public and abstract.



The relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface, but a **class implements an interface**.



Java Interface Example

In this example, the Printable interface has only one method, and its implementation is provided in the A6 class.

```
interface printable { void  
print ();  
}  
class A6 implements printable {  
public void print () {System.out.println("Hello");}  
public static void main (String args []) {
```



```
A6 obj = new A6(); obj.print  
();  
}  
}
```

3. Polymorphism in Java

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.

Real-life Illustration: Polymorphism

A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possesses different behavior in different situations. This is called polymorphism.

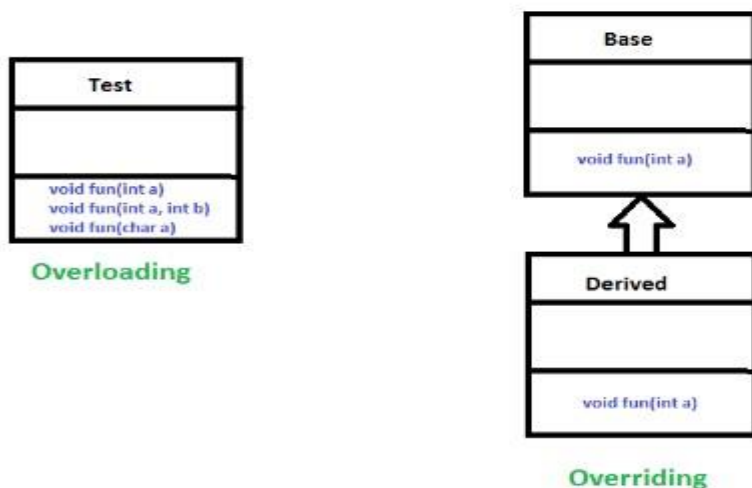
Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, so it means many forms.

Types of polymorphism

In Java polymorphism is mainly divided into two types:

Compile-time Polymorphism

Runtime Polymorphism



Method Overloading:

When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by change in the number of arguments or/and a change in the type of arguments.

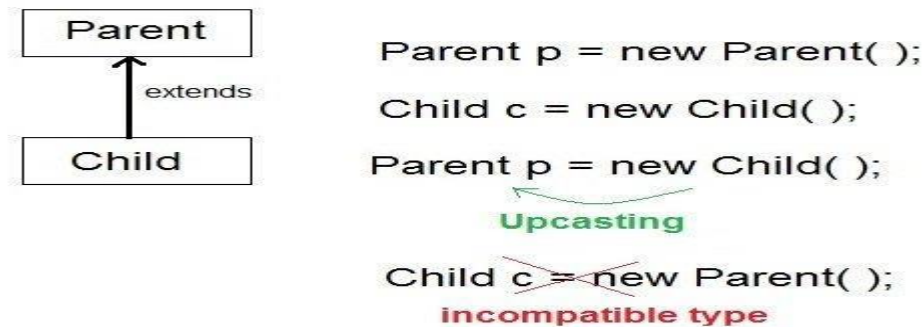
Runtime polymorphism

It is also known as Dynamic Method Dispatch. It is a process in which a function call to the overridden method is resolved at Runtime. This type of polymorphism is achieved by Method Overriding. **Method overriding**, on the other hand, occurs when a derived class has a definition for one of the member functions of the base class. That base function is said to be **overridden**.



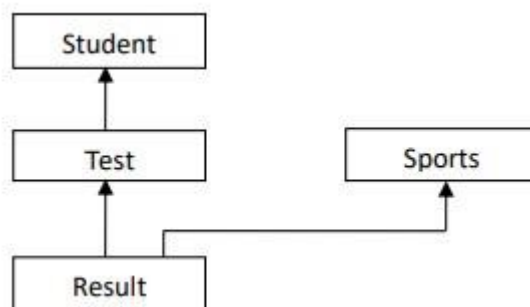
Dynamic method dispatch using base class and interface reference in Java

Dynamic method dispatch is a mechanism by which a call to an overridden method is resolved at runtime. This is how java implements runtime polymorphism. When an overridden method is called by a reference, java determines which version of that method to execute based on the type of object it refer to. In simple words the type of object which it referred determines which version of overridden method will be called.



4. Lab Assignments to complete in this session

- WAP to implement three classes namely Student, Test and Result. Student class has member as rollno, Test class has members as sem1_marks and sem2_marks and Result class has member as total. Create an interface named sports that has a member score (). Derive Test class from Student and Result class has multiple inheritances from Test and Sports. Total is formula based on sem1_marks, sem2_mark and score.





J Q1.java > ...

```
1  import java.util.Scanner;
2  interface Sports {
3      void score();
4  }
5  class Student {
6      int rollno;
7      Student(int rollno) {
8          this.rollno = rollno;
9      }
10 }
11 class Test extends Student {
12     int sem1_marks;
13     int sem2_marks;
14
15     Test(int rollno, int sem1_marks, int sem2_marks) {
16         super(rollno);
17         this.sem1_marks = sem1_marks;
18         this.sem2_marks = sem2_marks;
19     }
20 }
21 class Result extends Test implements Sports {
22     int total;
23     int score;
24     Result(int rollno, int sem1_marks, int sem2_marks, int score) {
25         super(rollno, sem1_marks, sem2_marks);
26         this.score = score;
27     }
28     @Override
29     public void score() {
30         this.total = sem1_marks + sem2_marks + score;
31     }
32 }
33 public class Q1 {
34     Run | Debug
35     public static void main(String[] args) {
36         Scanner scanner = new Scanner(System.in);
37         System.out.print(s:"Enter Roll Number: ");
38         int rollno = scanner.nextInt();
39         System.out.print(s:"Enter Semester 1 Marks: ");
40         int sem1Marks = scanner.nextInt();
41         System.out.print(s:"Enter Semester 2 Marks: ");
42         int sem2Marks = scanner.nextInt();
43         System.out.print(s:"Enter Sports Score: ");
44         int sportsScore = scanner.nextInt();
45
46         Result result = new Result(rollno, sem1Marks, sem2Marks, sportsScore);
47         result.score();
48         System.out.println("Roll Number: " + result.rollno);
49         System.out.println("Semester 1 Marks: " + result.sem1_marks);
50         System.out.println("Semester 2 Marks: " + result.sem2_marks);
51         System.out.println("Sports Score: " + result.score);
52         System.out.println("Total Marks: " + result.total);
53         scanner.close();
54     }
55 }
```




Shri Vile Parle Kelavani Mandal's

DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING

(Autonomous College Affiliated to the University of Mumbai)

NAAC Accredited with "A" Grade (CGPA : 3.18)



```
s\JSL4_2add9d58\bin' 'Q1'  
Enter Roll Number: 68  
Enter Semester 1 Marks: 98  
Enter Semester 2 Marks: 93  
Enter Sports Score: 89  
Roll Number: 68  
Semester 1 Marks: 98  
Semester 2 Marks: 93  
Sports Score: 89  
Total Marks: 280  
PS D:\DS\JSL4> █
```



- ii. Write an abstract class program to calculate area of circle, rectangle and triangle

```
J Q2.java > ...
1  import java.util.Scanner;
2  abstract class Shape {
3      abstract double calculateArea();
4  }
5
6  class Circle extends Shape {
7      private double radius;
8      Circle(double radius) {
9          this.radius = radius;
10     }
11     @Override
12     double calculateArea() {
13         return Math.PI * radius * radius;
14     }
15 }
16
17 class Rectangle extends Shape {
18     private double length;
19     private double width;
20     Rectangle(double length, double width) {
21         this.length = length;
22         this.width = width;
23     }
24     @Override
25     double calculateArea() {
26         return length * width;
27     }
28 }
29
30 class Triangle extends Shape {
31     private double base;
32     private double height;
33     Triangle(double base, double height) {
34         this.base = base;
35         this.height = height;
36     }
37     @Override
38     double calculateArea() {
39         return 0.5 * base * height;
40     }
41 }
```



```
42 public class Q2 {
43     Run | Debug
44     public static void main(String[] args) {
45         Scanner scanner = new Scanner(System.in);
46
47         System.out.println(x:"Choose a shape:");
48         System.out.println(x:"1. Circle");
49         System.out.println(x:"2. Rectangle");
50         System.out.println(x:"3. Triangle");
51
52         int choice = scanner.nextInt();
53
54         switch (choice) {
55             case 1:
56                 System.out.print(s:"Enter the radius of the circle: ");
57                 double radius = scanner.nextDouble();
58                 Circle circle = new Circle(radius);
59                 System.out.println("Area of the circle: " + circle.calculateArea());
60                 break;
61             case 2:
62                 System.out.print(s:"Enter the length of the rectangle: ");
63                 double length = scanner.nextDouble();
64                 System.out.print(s:"Enter the width of the rectangle: ");
65                 double width = scanner.nextDouble();
66                 Rectangle rectangle = new Rectangle(length, width);
67                 System.out.println("Area of the rectangle: " + rectangle.calculateArea());
68                 break;
69             case 3:
70                 System.out.print(s:"Enter the base of the triangle: ");
71                 double base = scanner.nextDouble();
72                 System.out.print(s:"Enter the height of the triangle: ");
73                 double height = scanner.nextDouble();
74                 Triangle triangle = new Triangle(base, height);
75                 System.out.println("Area of the triangle: " + triangle.calculateArea());
76                 break;
77             default:
78                 System.out.println(x:"Invalid choice");
79         }
80         scanner.close();
81     }
82 }
```



```
PS D:\DS\JSL4> d.; cd 'd:\DS\JSL4'; & 'C:\
s\JSL4_2add9d58\bin' 'Q2'
Choose a shape:
1. Circle
2. Rectangle
3. Triangle
1
Enter the radius of the circle: 5
Area of the circle: 78.53981633974483
PS D:\DS\JSL4> d.; cd 'd:\DS\JSL4'; & 'C:\
'-cp' 'C:\Users\djsce.student\AppData\Roam
s\JSL4_2add9d58\bin' 'Q2'
Choose a shape:
1. Circle
2. Rectangle
3. Triangle
2
Enter the length of the rectangle: 5
Enter the width of the rectangle: 3
Area of the rectangle: 15.0
PS D:\DS\JSL4> d.; cd 'd:\DS\JSL4'; & 'C:\
'-cp' 'C:\Users\djsce.student\AppData\Roam
s\JSL4_2add9d58\bin' 'Q2'
Choose a shape:
1. Circle
2. Rectangle
3. Triangle
3
Enter the base of the triangle: 3
Enter the height of the triangle: 4
Area of the triangle: 6.0
```



- iii. Create 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.



J Q3.java > Q3 > main(String[])

```
1  import java.util.ArrayList;
2  import java.util.List;
3
4  class Account {
5      private String accountNumber;
6      protected double balance;
7
8      public Account(String accountNumber) {
9          this.accountNumber = accountNumber;
10         this.balance = 0.0;
11     }
12
13     public void deposit(double amount) {
14         balance += amount;
15         System.out.println("Deposited $" + amount + " into account " + accountNumber);
16     }
17
18     public void withdraw(double amount) {
19         if (balance >= amount) {
20             balance -= amount;
21             System.out.println("Withdrawn $" + amount + " from account " + accountNumber);
22         } else {
23             System.out.println("Insufficient balance in account " + accountNumber);
24         }
25     }
26
27     public double getBalance() {
28         return balance;
29     }
30
31     public String getAccountNumber() {
32         return accountNumber;
33     }
34
35     @Override
36     public String toString() {
37         return "Account Number: " + accountNumber + ", Balance: $" + balance;
38     }
39 }
40
41 class SavingsAccount extends Account {
42     private double interestRate;
43
44     public SavingsAccount(String accountNumber, double interestRate) {
45         super(accountNumber);
46         this.interestRate = interestRate;
47     }
48
49     public void addInterest() {
50         double interestAmount = balance * (interestRate / 100);
51         balance += interestAmount;
52         System.out.println("Added interest of $" + interestAmount + " to account " + getAccountNumber());
53     }
54
55     @Override
56     public String toString() {
57         return super.toString() + ", Interest Rate: " + interestRate + "%";
58     }
59 }
60
61 class CurrentAccount extends Account {
62     private double overdraftLimit;
63
64     public CurrentAccount(String accountNumber, double overdraftLimit) {
65         super(accountNumber);
66         this.overdraftLimit = overdraftLimit;
67     }
68 }
```



```
68
69
70     @Override
71     public void withdraw(double amount) {
72         if (balance + overdraftLimit >= amount) {
73             balance -= amount;
74             System.out.println("Withdrawn $" + amount + " from account " + getAccountNumber());
75         } else {
76             System.out.println("Exceeded overdraft limit in account " + getAccountNumber());
77         }
78     }
79
80     @Override
81     public String toString() {
82         return super.toString() + ", Overdraft Limit: $" + overdraftLimit;
83     }
84 }
85
86 class Bank {
87     private List<Account> accounts;
88
89     public Bank() {
90         accounts = new ArrayList<>();
91     }
92
93     public void openAccount(Account account) {
94         accounts.add(account);
95         System.out.println("Opened a new account: " + account);
96     }
97
98     public void closeAccount(Account account) {
99         accounts.remove(account);
100         System.out.println("Closed account: " + account.getAccountNumber());
101     }
102
103     public void payDividend(double amount) {
104         for (Account account : accounts) {
105             account.deposit(amount);
106         }
107     }
108
109     public void updateAccounts() {
110         for (Account account : accounts) {
111             if (account instanceof SavingsAccount) {
112                 ((SavingsAccount) account).addInterest();
113             } else if (account instanceof CurrentAccount) {
114                 CurrentAccount currentAccount = (CurrentAccount) account;
115                 if (currentAccount.getBalance() < 0) {
116                     System.out.println("Letter sent for overdraft in account " + currentAccount.getAccountNumber());
117                 }
118             }
119         }
120     }
121
122     public void displayAccounts() {
123         for (Account account : accounts) {
124             System.out.println(account);
125         }
126     }
127 }
```




```
128     public class Q3 {  
129         Run | Debug  
130         public static void main(String[] args) {  
131             Bank bank = new Bank();  
132  
133             // Create and open accounts  
134             SavingsAccount savingsAccount = new SavingsAccount(accountNumber:"SA001", interestRate:2.5);  
135             CurrentAccount currentAccount = new CurrentAccount(accountNumber:"CA001", overdraftLimit:1000.0);  
136             bank.openAccount(savingsAccount);  
137             bank.openAccount(currentAccount);  
138  
139             // Deposit and withdraw  
140             savingsAccount.deposit(amount:5000);  
141             currentAccount.deposit(amount:2000);  
142             savingsAccount.withdraw(amount:1000);  
143             currentAccount.withdraw(amount:3000);  
144  
145             // Pay dividend  
146             bank.payDividend(amount:100);  
147  
148             // Update accounts  
149             bank.updateAccounts();  
150  
151             // Display account details  
152             bank.displayAccounts();  
153  
154             // Close accounts  
155             bank.closeAccount(savingsAccount);  
156             bank.closeAccount(currentAccount);  
157         }  
158     }
```

```
PS D:\DS\JSL4> & 'C:\Program Files\Java\jdk-19\bin\java.exe' '-XX:+ShowCodeDetailsInIntrospection' 'Q3'  
Opened a new account: Account Number: SA001, Balance: $0.0, Interest Rate: 2.5%  
Opened a new account: Account Number: CA001, Balance: $0.0, Overdraft Limit: $1000.0  
Deposited $5000.0 into account SA001  
Deposited $2000.0 into account CA001  
Withdrawn $1000.0 from account SA001  
Withdrawn $3000.0 from account CA001  
Deposited $100.0 into account SA001  
Deposited $100.0 into account CA001  
Added interest of $102.5 to account SA001  
Letter sent for overdraft in account CA001  
Account Number: SA001, Balance: $4202.5, Interest Rate: 2.5%  
Account Number: CA001, Balance: $-900.0, Overdraft Limit: $1000.0  
Closed account: SA001  
Closed account: CA001  
PS D:\DS\JSL4>
```

iv. 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

**J** Q4.java > ...

```
1  import java.util.Objects;
2
3  // Person class for employee's personal information
4  class Person {
5      private String name;
6
7      public Person(String name) {
8          this.name = name;
9      }
10
11     public String getName() {
12         return name;
13     }
14
15     @Override
16     public boolean equals(Object o) {
17         if (this == o) return true;
18         if (o == null || getClass() != o.getClass()) return false;
19         Person person = (Person) o;
20         return Objects.equals(name, person.name);
21     }
22
23     @Override
24     public int hashCode() {
25         return Objects.hash(name);
26     }
27 }
28
29 // Employee class as a derived class of Person
30 class Employee extends Person {
31     private double annualSalary;
32     private int startYear;
33     private String nationalInsuranceNumber;
34
35     public Employee(String name, double annualSalary, int startYear, String nationalInsuranceNumber) {
36         super(name);
37         this.annualSalary = annualSalary;
38         this.startYear = startYear;
39         this.nationalInsuranceNumber = nationalInsuranceNumber;
40     }
41
42     public double getAnnualSalary() {
43         return annualSalary;
44     }
45
46     public int getStartYear() {
47         return startYear;
48     }
49
50     public String getNationalInsuranceNumber() {
51         return nationalInsuranceNumber;
52     }
53
54     @Override
55     public String toString() {
56         return "Employee Name: " + getName() +
57             "\nAnnual Salary: $" + annualSalary +
58             "\nStart Year: " + startYear +
59             "\nNational Insurance Number: " + nationalInsuranceNumber;
60     }
61 }
62
```



```
63 public class Q4 {
64     Run | Debug
65     public static void main(String[] args) {
66         // Create Employee objects and test the class
67         Employee employee1 = new Employee(name:"John Doe", annualSalary:50000.0, startYear:2020, nationalInsuranceNumber:"AB12345");
68         Employee employee2 = new Employee(name:"Jane Smith", annualSalary:60000.0, startYear:2018, nationalInsuranceNumber:"CD67890");
69
70         // Display employee information
71         System.out.println(x:"Employee 1 Information:");
72         System.out.println(employee1);
73
74         System.out.println(x:"\nEmployee 2 Information:");
75         System.out.println(employee2);
76
77         // Test equality
78         if (employee1.equals(employee2)) {
79             System.out.println(x:"\nEmployee 1 and Employee 2 are the same person.");
80         } else {
81             System.out.println(x:"\nEmployee 1 and Employee 2 are different individuals.");
82         }
83     }
84 }
```

```
PS D:\DS\JSL4> & 'C:\Program Files\Java\jdk-19\bin\javac.exe' student\AppData\Roaming\Code\User\workspaceStorage\1\workspace\Q4\Q4.java
```

Employee 1 Information:

Employee Name: John Doe

Annual Salary: \$50000.0

Start Year: 2020

National Insurance Number: AB12345

Employee 2 Information:

Employee Name: Jane Smith

Annual Salary: \$60000.0

Start Year: 2018

National Insurance Number: CD67890

Employee 1 and Employee 2 are different individuals.

```
PS D:\DS\JSL4>
```