**9.Inheritence:**

**Class inheritance:**

**Overview:**

**Derived class relationship to base class**

**Member hiding and overriding**

**Role of object class**

**Implementing support for equality checks**

-Use extends keyword

-Derived has characteristics of base

-derived can add specialization.

package com.org.inhertence;  
  
public class Baseclass {  
  
 public void m1() {  
  
 System.*out*.println("m1 base class method");  
 }  
}

package com.org.inhertence;  
  
public class Derivedclass extends Baseclass {  
  
 public void m2()  
 {  
 System.*out*.println("m2 Derived class method");  
 }  
}

package com.org.inhertence;  
  
public class Main {  
  
 public static void main(String[] args) {  
  
  
//  
// Baseclass baseclass=new Baseclass();  
// baseclass.m1();  
  
  
 Derivedclass derivedclass=new Derivedclass();  
 derivedclass.m1();  
 derivedclass.m2();  
 }  
}

Class inheritance with variable and super keyword

package com.org.inhertence;  
  
public class Bicycle {  
  
 private int gear;  
 private int speed;  
  
  
 public Bicycle()  
 {  
 }  
  
 public Bicycle(int gear,int speed)  
 {  
 this.gear=gear;  
 this.speed=speed;  
 }  
  
 public void applyBreak(int decrement)  
 {  
 speed=speed-decrement;  
 }  
  
 public void speedUp(int increment)  
 {  
 speed=speed+increment;  
 }  
  
 public String toString()  
 {  
 return ("NO of gears are "+gear+ "speed of bicycle is "+speed);  
 }  
}

package com.org.inhertence;  
  
public class MoutainBike extends Bicycle{  
 public int seatHeight;  
  
 public MoutainBike(int gear,int speed,int seatHeight)  
 {  
 super(gear,speed);  
 this.seatHeight=seatHeight;  
 }  
  
 public void setSeatHeight(int newValue)  
 {  
 seatHeight=newValue;  
 }  
  
 public String toString()  
 {  
 return (super.toString()+"seat height is"+seatHeight);  
 }  
}

package com.org.inhertence;  
  
public class Main {  
  
 public static void main(String[] args) {  
 MoutainBike moutainBike = new MoutainBike(3, 100, 2);  
 System.*out*.println(moutainBike);  
 }  
}

Calengine with inheritance and overriding:

package com.org.inhertence;  
  
public class CalculateBase {  
 private double leftValue;  
 private double rightValue;  
 private double result;  
  
 public double getLeftValue() {  
 return leftValue;  
 }  
  
 public void setLeftValue(double leftValue) {  
 this.leftValue = leftValue;  
 }  
  
 public double getRightValue() {  
 return rightValue;  
 }  
  
 public void setRightValue(double rightValue) {  
 this.rightValue = rightValue;  
 }  
  
 public double getResult() {  
 return result;  
 }  
  
 public void setResult(double result) {  
 this.result = result;  
 }  
  
 public void calculate()  
 {  
  
 }  
  
  
}

package com.org.inhertence;  
  
public class Multiply extends CalculateBase {  
  
 public void calculate() {  
 double value = getLeftValue() \* getRightValue();  
 setResult(value);  
 }  
  
}

package com.org.inhertence;  
  
public class

MainInhertence {  
  
 public static void main(String[] args) {  
  
 Multiply multiply=new Multiply();  
 *doCalculation*(multiply,100.0d,50.0d);  
  
 }  
  
  
 static void doCalculation(CalculateBase calculateBase,double leftValue,double rightValue)  
 {  
 calculateBase.setLeftValue(leftValue);  
 calculateBase.setRightValue(rightValue);  
 calculateBase.calculate();  
 System.*out*.println("calculation result ="+calculateBase.getResult());  
 }  
}

**Object class:**

**Root of the java class hierarchy:**

-Every class has characteristics of object

- Object references can reference any array or class instance

- Every class inherits directly or indirectly from object class

**Object References:**

**Preventing Inheritance and Method Overring:**

**Default inheritance behavior**

* Each class can be extended
* Derived(child) class can override any method

**Can change default behavior with final**

* Can prevent class extending
* Can prevent method overriding.

package com.org.inhertence;  
 public class Baseclass {  
  
 public final void m1() {  
 System.*out*.println("m1 base class method");  
 }  
  
 public void m2() {  
 System.*out*.println("m2 base class method");  
 }  
  
 public void m3() {  
 System.*out*.println("m3 base class method");  
 }  
}

package com.org.inhertence;  
  
public class Derivedclass extends Baseclass {  
  
 public void m2()  
 {  
 System.*out*.println("m2 Derived class method");  
 }  
  
 public void m3()  
 {  
 System.*out*.println("m3 Derived class method");  
 }  
  
}

if override m3 we are going to get a compilation error.

Final class we can not extend

**Inheritance and constructor:**

**Constructers are not inherited**

* **Each class has its own constructors.**

**Derived class constructors:**

**Constructing a derived class instance**

**A base class constructor is always called by default no argument version called.**

**Can explicitly call a base class constructer use super keyword**

package com.org.inhertence;  
 public class Baseclass {  
 public Baseclass()  
 {  
 System.*out*.println("base class constructor");  
 }  
  
 public void m1() {  
 System.*out*.println("m1 base class method");  
 }  
  
 public void m2() {  
 System.*out*.println("m2 base class method");  
 }  
  
}

package com.org.inhertence;  
  
public class Derivedclass extends Baseclass {  
  
 public Derivedclass() {  
 System.*out*.println("Derived class constructor");  
 }  
  
 public void m1() {  
 System.*out*.println("m1 Derived class method");  
 }  
  
 public void m2() {  
 System.*out*.println("m2 Derived class method");  
 }  
  
}

package com.org.inhertence;  
  
public class Maindummy {  
  
 public static void main(String[] args) {  
 Derivedclass baseclass = new Derivedclass();  
 baseclass.m2();  
 }  
}

package com.org.inhertence;  
 public class Baseclass {  
  
 private int var;  
 public Baseclass()  
 {  
 System.*out*.println("base class constructor");  
 }  
  
 public Baseclass(int var)  
 {  
 this.var=var;  
 System.*out*.println("base class parameter constructor");  
 }  
  
 public void m1() {  
 System.*out*.println("m1 base class method");  
 }  
  
 public void m2() {  
 System.*out*.println("m2 base class method");  
 }  
  
}

package com.org.inhertence;  
  
public class Derivedclass extends Baseclass {  
  
 public Derivedclass() {  
 System.*out*.println("Derived class constructor");  
 }  
  
 public Derivedclass(int var) {  
 super(var);  
 System.*out*.println("Derived class constructor");  
 }  
  
 public void m1() {  
 System.*out*.println("m1 Derived class method");  
 }  
  
 public void m2() {  
 System.*out*.println("m2 Derived class method");  
 }  
  
}

package com.org.inhertence;  
  
public class Maindummy {  
  
 public static void main(String[] args) {  
 Derivedclass baseclass = new Derivedclass(10);  
 baseclass.m2();  
 }  
}

**Interface:**

package com.org.interfaceexamples;  
  
public interface InterfaceExample {  
 public abstract void m1();  
}

package com.org.interfaceexamples;  
  
public class Account implements InterfaceExample{  
 @Override  
 public void m1() {  
 System.*out*.println("m1 method of account class");  
 }  
}

package com.org.interfaceexamples;  
  
public class Employee implements InterfaceExample {  
 @Override  
 public void m1() {  
 System.*out*.println("m1 method of employee class");  
 }  
}

package com.org.interfaceexamples;  
  
public class Main {  
  
 public static void main(String[] args) {  
 InterfaceExample employeeInterface=new Employee();  
 employeeInterface.m1();  
 InterfaceExample employeeInterface1=new Account();  
 employeeInterface1.m1();  
 }  
}

**Implementing abstact:**

package com.org.interfaceexamples;  
  
public abstract class CalculateBase {  
 private double leftValue;  
 private double rightValue;  
 private double result;  
  
 public CalculateBase()  
 {  
  
 }  
 public CalculateBase(double leftValue,double rightValue)  
 {  
 this.leftValue=leftValue;  
 this.rightValue=rightValue;  
 }  
  
 public double getLeftValue() {  
 return leftValue;  
 }  
  
 public void setLeftValue(double leftValue) {  
 this.leftValue = leftValue;  
 }  
  
 public double getRightValue() {  
 return rightValue;  
 }  
  
 public void setRightValue(double rightValue) {  
 this.rightValue = rightValue;  
 }  
  
 public double getResult() {  
 return result;  
 }  
  
 public void setResult(double result) {  
 this.result = result;  
 }  
  
 public abstract void calculate();  
  
}

package com.org.interfaceexamples;  
  
public class Adder extends CalculateBase {  
 public Adder() {  
  
 }  
  
 public Adder(double leftValue, double rightValue) {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() + getRightValue();  
 setResult(value);  
 }  
}

package com.org.interfaceexamples;  
  
public class Divide extends CalculateBase {  
 public Divide()  
 {  
 }  
 public Divide(double leftValue,double rightValue)  
 {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() / getRightValue();  
 setResult(value);  
 }  
}

package com.org.interfaceexamples;  
  
public class Multiply extends CalculateBase {  
  
 public Multiply() {  
 }  
  
 public Multiply(double leftValue, double rightValue) {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() \* getRightValue();  
 setResult(value);  
 }  
}

package com.org.interfaceexamples;  
  
public class MainInhertence {  
  
 public static void main(String[] args) {  
  
 CalculateBase[] calculateBases={  
 new Adder(100.0d,50.0d),  
 new Multiply(100.0d,50.0d),  
 new Divide(100.0d,50.0d),  
 };  
  
 System.*out*.println();  
 System.*out*.println("Array calculations");  
  
  
 for (CalculateBase calculateBase:calculateBases)  
 {  
 calculateBase.calculate();  
 System.*out*.println("result==="+calculateBase.getResult());  
 }  
  
 }  
  
  
}

**Summary:**

**Super reference:**

* **Refence to the current object**
* **Treats as instance of base class**

**Preventing inheritance:**

* **Mark class private**

**Preventing method overriding**

* **Mark method as final**

**Requiring Inherence**

* **Mark class as abstract**

**Requiring method overriding**

**Constructors are not inherited:**

* **Each class has its own constructors**

**Constructing a derived class instance**

* **A base class constructor is always called**
* **By default, call no-argument version**
* **Can explicitly call specific constructor**

**=**

**08/05/2024:**

**Interface:**

* **The need for more than interface**
* **Implementing an interface**
* **Generic interface**
* **Implementing multiple interfaces**
* **Declaring interface**
* **Default methods**

**Effective software development**

* **Relies on reusability**

**Class inheritance is part of the solution**

* **Allows one class to leverage the implementation details of another**

**Inheritance has limitations**

* **A class directly extends only one class**
* **Constrains realistically available reusability**

**Interfaces:**

**An Interface defines a contract**

**Provides a list of operations does not focus on implementation details**

**Classes implements interfaces**

**Express conformance to contract provides necessary methods**

**Comparable interface demonstrates the value of interfaces**

**Challenge: Objects often need to be ordered rules of ordering different for each class.**

**Comparable interface provides a contract for ordering.**

**Enables broadly reusable soring utilizes , Need no knowledge of specific class**

**Expose one method- compareTo**

* **Receives a reference to another object**

**Method indicates relative relationship**

* **Indicates ordering between current a**

package com.org.interfaceexamples;  
  
public interface InterfaceExample {  
 //any method which dont have body is called abstarct methods  
 public abstract void m1();  
 public abstract void m2();  
 default public void m3(){  
 }  
 default public void m4(){  
 }  
}

package com.org.interfaceexamples;  
  
public class Account implements InterfaceExample {  
  
 @Override  
 public void m1() {  
 System.*out*.println("account class from m1 interface example");  
 }  
  
 @Override  
 public void m2() {  
 System.*out*.println("account class m1 from interface example");  
 }  
  
 @Override  
 public void m3() {  
 System.*out*.println("account class m3 from interface example");  
 }  
}

package com.org.interfaceexamples;  
  
public class Employee implements InterfaceExample {  
  
 @Override  
 public void m1() {  
 System.*out*.println("Empoyee class m1 interface example");  
 }  
  
 @Override  
 public void m2() {  
 System.*out*.println("Empoyee class m2 interface example");  
 }  
}

Example 2:

package com.org.interfaceexamples;  
  
public abstract class CalculateBase {  
 private double leftValue;  
 private double rightValue;  
 private double result;  
  
 public CalculateBase()  
 {  
  
 }  
 public CalculateBase(double leftValue,double rightValue)  
 {  
 this.leftValue=leftValue;  
 this.rightValue=rightValue;  
 }  
  
 public double getLeftValue() {  
 return leftValue;  
 }  
  
 public void setLeftValue(double leftValue) {  
 this.leftValue = leftValue;  
 }  
  
 public double getRightValue() {  
 return rightValue;  
 }  
  
 public void setRightValue(double rightValue) {  
 this.rightValue = rightValue;  
 }  
  
 public double getResult() {  
 return result;  
 }  
  
 public void setResult(double result) {  
 this.result = result;  
 }  
  
 public abstract void calculate();  
  
}

package com.org.interfaceexamples;  
  
public class Adder extends CalculateBase {  
 public Adder() {  
  
 }  
  
 public Adder(double leftValue, double rightValue) {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() + getRightValue();  
 setResult(value);  
 }  
}

package com.org.interfaceexamples;  
  
public class Divide extends CalculateBase {  
 public Divide()  
 {  
 }  
 public Divide(double leftValue,double rightValue)  
 {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() / getRightValue();  
 setResult(value);  
 }  
}

package com.org.interfaceexamples;  
  
public class Multiply extends CalculateBase {  
  
 public Multiply() {  
 }  
  
 public Multiply(double leftValue, double rightValue) {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() \* getRightValue();  
 setResult(value);  
 }  
}

package com.org.interfaceexamples;  
  
public class MainInhertence {  
  
 public static void main(String[] args) {  
  
 CalculateBase[] calculateBases={  
 new Adder(100.0d,50.0d),  
 new Multiply(100.0d,50.0d),  
 new Divide(100.0d,50.0d),  
 };  
  
 System.*out*.println();  
 System.*out*.println("Array calculations");  
  
  
 for (CalculateBase calculateBase:calculateBases)  
 {  
 calculateBase.calculate();  
 System.*out*.println("result==="+calculateBase.getResult());  
 }  
  
 }  
  
  
}

One more with interface (cal engine):

package com.org.interfacecal;  
  
public abstract class CalculateBase {  
 private double leftValue;  
 private double rightValue;  
 private double result;  
  
 public CalculateBase()  
 {  
  
 }  
 public CalculateBase(double leftValue,double rightValue)  
 {  
 this.leftValue=leftValue;  
 this.rightValue=rightValue;  
 }  
  
 public double getLeftValue() {  
 return leftValue;  
 }  
  
 public void setLeftValue(double leftValue) {  
 this.leftValue = leftValue;  
 }  
  
 public double getRightValue() {  
 return rightValue;  
 }  
  
 public void setRightValue(double rightValue) {  
 this.rightValue = rightValue;  
 }  
  
 public double getResult() {  
 return result;  
 }  
  
 public void setResult(double result) {  
 this.result = result;  
 }  
  
 public abstract void calculate();  
  
}

package com.org.interfacecal;  
  
public interface MathProcessing {  
  
 String *SEPARATOR* = " ";  
  
 String getKeyword();  
  
 double doCalculation(double leftVal, double rightVal);  
}

package com.org.interfacecal;  
  
public class Adder extends CalculateBase implements MathProcessing{  
 @Override  
 public String getKeyword() {  
 return "add";  
 }  
  
 @Override  
 public double doCalculation(double leftVal, double rightVal) {  
 setLeftValue(leftVal);  
 setRightValue(rightVal);  
 calculate();  
 return getResult();  
 }  
  
 public Adder() {  
  
 }  
  
 public Adder(double leftValue, double rightValue) {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() + getRightValue();  
 setResult(value);  
 }  
}

package com.org.interfacecal;  
  
public class Divide extends CalculateBase implements MathProcessing{  
 public Divide()  
 {  
 }  
 public Divide(double leftValue,double rightValue)  
 {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() / getRightValue();  
 setResult(value);  
 }  
  
 @Override  
 public String getKeyword() {  
 return "divide";  
 }  
  
 @Override  
 public double doCalculation(double leftVal, double rightVal) {  
 setLeftValue(leftVal);  
 setRightValue(rightVal);  
 calculate();  
 return getResult();  
 }  
}

package com.org.interfacecal;  
  
public class Multiply extends CalculateBase implements MathProcessing{  
  
 public Multiply() {  
 }  
  
 public Multiply(double leftValue, double rightValue) {  
 super(leftValue, rightValue);  
 }  
  
 public void calculate() {  
 double value = getLeftValue() \* getRightValue();  
 setResult(value);  
 }  
  
 @Override  
 public String getKeyword() {  
 return "multiply";  
 }  
  
 @Override  
 public double doCalculation(double leftVal, double rightVal) {  
 setLeftValue(leftVal);  
 setRightValue(rightVal);  
 calculate();  
 return getResult();  
 }  
}

package com.org.interfacecal;  
  
public class DynamicHelper {  
  
 private final MathProcessing[] handlers;  
  
 public DynamicHelper(MathProcessing[] handlers) {  
 this.handlers = handlers;  
 }  
  
 public void procesing(String statement)  
 {  
 String[] parts = statement.split(MathProcessing.*SEPARATOR*);  
 String keyword=parts[0];  
 double leftVal=Double.*parseDouble*(parts[1]);  
 double rightVal=Double.*parseDouble*(parts[2]);  
 MathProcessing theHandler = null;  
 for(MathProcessing handler: handlers)  
 {  
 if(keyword.equalsIgnoreCase(handler.getKeyword())){  
 theHandler=handler;  
 break;  
 }  
 }  
 double result=theHandler.doCalculation(leftVal,rightVal);  
 System.*out*.println("Result+ "+result);  
 }  
  
  
}

package com.org.interfacecal;  
  
import java.util.Scanner;  
  
public class MainInhertence {  
  
 public static void main(String[] args) {  
 *dynamicinteractivity*();  
 }  
  
 private static void dynamicinteractivity() {  
 DynamicHelper helper = new DynamicHelper(new MathProcessing[]{  
 new Divide()  
 });  
  
 System.*out*.println("Enter an operation divide two numbers");  
 Scanner scanner = new Scanner(System.*in*);  
  
 String userInput = scanner.nextLine();  
 helper.procesing(userInput);  
 }  
}