36.1.1. Understanding Inheritance

40:10

Inheritance is a way by which a class can inherit what is already there in another existing class.

We can classify inheritance into two kinds:

1. **Implementation Inheritance** (or code inheritance)
2. **Interface Inheritance** (or inheritance of behaviour or contract)

Here we will understand *implementation inheritance* and later learn about *interface inheritance* after we learn *interfaces*.

In Java, a class can **inherit** implementation from **only one class**, it is also referred as **single-inheritance**. (Other languages like c++ support multiple inheritance of implementation)

In Java, we use `extends` keyword when we want a class to inherit (extend) from another class. For example:

```
class A {
    ...
}
class B extends A {
    ...
}
```

In the above code B extends A. B is called the subclass of A. A is called the super class of B.

Note: Every class in Java automatically extends the root class Object if it does not explicitly extends another class. Which means in our above example, class A is a subclass of Object and Object is the superclass of A.

Using the above concepts select all the correct statements from the below code:

```
class W { // statement 1
}
class X { // statement 2
    ...
}
class Y extends X { // statement 3
    ...
}
class Z1 extends W, Y { // statement 4
    ...
}
class Z2 extends Y { // statement 5
    ...
}
```

- [ ] Statement 1 is wrong. Since every class in Java extends `Object` class, statement 1 should have been

```
class W extends Object {
```

- [ ] As per the class declaration in statement 3, `Y` is the superclass of `X`.
- [ ] As per statement 2, the class declaration statement of `X` states that there is no superclass for `X`.
- [ ] Statement 4, which states that class `Z1` is the subclass of both `W` and `Y` is correct.
- [x] As per statement 5, `Z2` is the subclass of `Y`, `X` and `Object`.

‹ Prev Reset Submit Next ›

Home Learn Anywhere

36.1.2. Understanding implementation inheritance usage

00:10

In implementation inheritance the subclass inherits the implementation facilitating code reuse. For example:

```
class A {
    public int aValue = 1;
    public int getAValue() {
        return aValue;
    }
}
class B extends A {
    public int bValue = 2;
    public int getBValue() {
        return bValue;
    }
}
```

In the above code, class `B` also contains the inherited field `aValue` and the inherited method `getAValue()`.

Observe the given code in editor and complete the incomplete code:

- It demonstrates inheritance with two classes: **Person** and **Student**.
- The **Person** class has instance variables **name** and **age**, along with **getInfo()** and **displayInfo()** methods.
- In the **getInfo()** method of the **Student** class, the user is prompted to enter the name, age, and roll number of a student. The input is stored in the respective variables using the **super** keyword to access the parent class's **getInfo()** method.
- The **displayInfo()** method of the **Student** class overrides the parent class's method and displays the student's information along with the roll number.
- In the **main()** method, a **Student** object is created, and the **getInfo()** and **displayInfo()** methods are called to get and display the student's details.

Note: Please don't change the package name.

Sample Test Cases

Explorer

Inheritan...

```
import java.util.Scanner;
class Person {
    protected String name;
    protected int age;
    public void getInfo() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the name: ");
        name = scanner.nextLine();
        System.out.print("Enter the age: ");
        age = scanner.nextInt();
    }
    public void displayInfo() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
//Complete the code
class Student extends Person {
    private int rollNumber;
    @Override
    public void getInfo(){

        super.getInfo();
            Scanner sc = new Scanner(System.in);
            System.out.print("Enter the roll number: ");
            rollNumber = sc.nextInt();

    }
    @Override
    public void displayInfo(){
        super.displayInfo();
        System.out.println("Roll Number: "+rollNumber);
    }
}
public class InheritanceExample {
    public static void main(String[] args) {
        Student student = new Student();
        student.getInfo();
        student.displayInfo();
    }
}
```

Test cases

‹ Prev Reset Submit Next ›

29, 2024, 23:29

Home Learn Anywhere

36.1.3. Understanding implementation inheritance usage

35:10

In implementation inheritance, when the subclass inherits from a superclass (also callled base class), the subclass instance can be referred by the base class reference. For example:

```
class A {
    public int aValue = 1;
    public int getAValue() {
        return aValue;
    }
}
class B extends A {
    public int bValue = 2;
    public int getBValue() {
        return bValue;
    }
}
Then the below statement is valid:
A a = new B();
```

Note: In the above code, we can access all members of class `A` via the reference `a` even though the actual object created in memory using the `new` keyword is an instance of class `B`.

According to the above code, the statementa.`getBValue()` will result in compilation error. This is because reference `a` which is of type class `A` does not know about the members declared in its subclass `B`.

See and observe the code in the editor and complete it:

- The **A** class has an instance variable **aValue**, along with a **getAValue()** method.
- The **B** class must be inherited from **A** and will have an additional instance variable **bValue** and a **getBValue()** method.
- In the **main()** method, the user is prompted to enter the value for **A** and **B** using the **Scanner** class.
- A **B** object is created, and the **aValue** and **bValue** are set based on the user's input.
- The **getAValue()** and **getBValue()** methods are called on the **B** object to display the values of **A** and **B** respectively.

Sample Test Cases

Explorer

Inheritan...

Submit

Plots

```
1  package q35966;
2  import java.util.Scanner;
3  class A {
4      public int aValue;
5
6      public int getAValue() {
7          return aValue;
8      }
9  }
10 class B extends A{
11     public int bValue;
12     public int getBValue(){
13         return bValue;
14     }
15 }
16 public class InheritanceExample2 {
17     public static void main(String[] args) {
18         Scanner scanner = new Scanner(System.in);
19         //define a reference variable of class B
20         B b = new B();
21         System.out.print("Enter the value for A: ");
22         int aValue = scanner.nextInt();
23         b.aValue = aValue;
24
25         System.out.print("Enter the value for B: ");
26         int bValue = scanner.nextInt();
27         b.bValue = bValue;
28
29         System.out.println("b.getAValue(): " + b.getAValue());
30         System.out.println("b.getBValue(): " + b.getBValue());
31     }
32 }
33
```

Test cases

‹ Prev Reset Submit Next ›

Home Learn Anywhere

36.1.4. Write a Java program to implement Single Inheritance

06:50

Write a Java program to illustrate the **single inheritance** concept.

Create a class `Marks`
- contains the data members **id** of **int** data type, **javaMarks**, **cMarks** and **cppMarks** of **float** data type
- write a method **setMarks()** to initialize the data members
- write a method **displayMarks()** which will display the given data

Create another class `Result` which is derived from the class `Marks`
- contains the data members **total** and **avg** of **float** data type
- write a method **compute()** to find total and average of the given marks
- write a method **showResult()** which will display the total and avg marks

Write a class `SingleInheritanceDemo` with **main()** method it receives four arguments as **id**, **javaMarks**, **cMarks** and **cppMarks**.

Create object only to the class `Result` to access the methods.

If the input is given as command line arguments to the **main()** as **"101", "45.50", "67.75", "72.25"** then the program should print the output as:

```
Id : 101
Java marks : 45.5
C marks : 67.75
Cpp marks : 72.25
Total : 185.5
Avg : 61.833332
```

Note: While computing the total marks, add the marks in the following order only **javaMarks**, **cMarks** and **cppMarks**

Sample Test Cases

Explorer SingleInh...

```
1  package q11263;
2  class Marks{
3      protected int id;
4      protected float javaMarks;
5      protected float cMarks;
6      protected float cppMarks;
7      void setMarks(int id, float javaMarks, float cMarks, float cppMarks){
8          this.id = id;
9          this.javaMarks = javaMarks;
10         this.cMarks = cMarks;
11         this.cppMarks = cppMarks;
12     }
13     void displayMarks(){
14         System.out.println("Id : "+id);
15         System.out.println("Java marks : "+javaMarks);
16         System.out.println("C marks : "+cMarks);
17         System.out.println("Cpp marks : "+cppMarks);
18     }
19 }
20 class Result extends Marks{
21     float total =0;
22     float avg =0;
23     void compute(){
24         total = javaMarks +cMarks +cppMarks;
25         avg = (float) total/3;
26     }
27     void showResult(){
28         System.out.println("Total : "+total);
29         System.out.println("Avg : "+avg);
30     }
31 }
32 public class SingleInheritanceDemo{
33     public static void main(String s[]){
34         Result R = new Result();
35         R.setMarks(Integer.parseInt(s[0]),Float.parseFloat(s[1]),
   Float.parseFloat(s[2]), Float.parseFloat(s[3]));
36         R.compute();
37         R.displayMarks();
38         R.showResult();
39     }
40 }
```

Terminal Test cases

‹ Prev Reset Submit Next ›

Home Learn Anywhere

36.1.5. Write a Java program to implement Multilevel Inheritance

00:10

Write a Java program to illustrate the **multilevel inheritance** concept.

Create a class `Student`
- contains the data members **id** of **int** data type and **name** of **string** type
- write a method **setData()** to initialize the data members
- write a method **displayData()** which will display the given **id** and **name**

Create a class `Marks` which is derived from the class `Student`
- contains the data members **javaMarks**, **cMarks** and **cppMarks** of **float** data type
- write a method **setMarks()** to initialize the data members
- write a method **displayMarks()** which will display the given data

Create another class `Result` which is derived from the class `Marks`
- contains the data members **total** and **avg** of **float** data type
- write a method **compute()** to find total and average of the given marks
- write a method **showResult()** which will display the total and avg marks

Write a class `MultilevelInheritanceDemo` with the **main()** method which will receive five arguments as **id**, **name**, **javaMarks**, **cMarks** and **cppMarks**.

Create object only to the class `Result` to access the methods.

If the input is given as command line arguments to the **main()** as **"99", "Lakshmi", "55.5", "78.5", "72"** then the program should print the output as:

```
Id : 99
Name : Lakshmi
Java marks : 55.5
C marks : 78.5
Cpp marks : 72.0
Total : 206.0
Avg : 68.666664
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

Multilevel...

Submit

```
1  package q11264;
2  class Student{
3      private int id;
4      private String name;
5      void setData(int id, String name){
6          this.id = id;
7          this.name = name;
8      }
9      void displayData(){
10         System.out.println("Id : "+id);
11         System.out.println("Name : "+name);
12     }
13 }
14 class Marks extends Student{
15     float javaMarks;
16     float cMarks;
17     float cppMarks;
18     void setMarks(float javaMarks, float cMarks, float
   cppMarks){
19         this.javaMarks = javaMarks;
20         this.cMarks = cMarks;
21         this.cppMarks = cppMarks;
22     }
23     void displayMarks(){
24         System.out.println("Java marks : "+ javaMarks);
25         System.out.println("C marks : "+ cMarks);
26         System.out.println("Cpp marks : "+ cppMarks);
27     }
28 }
29 class Result extends Marks{
30     private float total;
31     private float avg;
32     void compute(){
33         total = javaMarks + cMarks + cppMarks;
```

Terminal Test cases

Prev Reset Submit Next

Notes hub

Home Learn Anywhere

36.1.5. Write a Java program to implement Multilevel Inheritance

00:10

Write a Java program to illustrate the **multilevel inheritance** concept.

Create a class `Student`
- contains the data members **id** of **int** data type and **name** of **string** type
- write a method **setData()** to initialize the data members
- write a method **displayData()** which will display the given **id** and **name**

Create a class `Marks` which is derived from the class `Student`
- contains the data members **javaMarks**, **cMarks** and **cppMarks** of **float** data type
- write a method **setMarks()** to initialize the data members
- write a method **displayMarks()** which will display the given data

Create another class `Result` which is derived from the class `Marks`
- contains the data members **total** and **avg** of **float** data type
- write a method **compute()** to find total and average of the given marks
- write a method **showResult()** which will display the total and avg marks

Write a class `MultilevelInheritanceDemo` with the **main()** method which will receive five arguments as **id**, **name**, **javaMarks**, **cMarks** and **cppMarks**.

Create object only to the class `Result` to access the methods.

If the input is given as command line arguments to the **main()** as **"99"**, **"Lakshmi"**, **"55.5"**, **"78.5"**, **"72"** then the program should print the output as:

```
Id : 99
Name : Lakshmi
Java marks : 55.5
C marks : 78.5
Cpp marks : 72.0
Total : 206.0
Avg : 68.666664
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

Multilevel...

Submit

```
cppMarks){
19      this.javaMarks = javaMarks;
20      this.cMarks = cMarks;
21      this.cppMarks = cppMarks;
22    }
23    void displayMarks(){
24      System.out.println("Java marks : "+ javaMarks);
25      System.out.println("C marks : "+ cMarks);
26      System.out.println("Cpp marks : "+ cppMarks);
27    }
28  }
29  class Result extends Marks{
30    private float total ;
31    private float avg ;
32    void compute(){
33      total = javaMarks + cMarks + cppMarks;
34      avg = total/3;
35    }
36    void showResult(){
37      System.out.println("Total : "+total);
38      System.out.println("Avg : "+avg);
39    }
40  }
41  public class MultilevelInheritanceDemo{
42    public static void main(String s[]){
43    Result R = new Result();
44      R.setData(Integer.parseInt(s[0]),s[1]);
45      R.displayData();
46      R.setMarks( Float.parseFloat(s[2]),
    Float.parseFloat(s[3]), Float.parseFloat(s[4]));
47      R.displayMarks();
48      R.compute();
49      R.showResult();
50  }}
```

Terminal Test cases

‹ Prev Reset Submit Next ›

Q No.	Q. Type	Status	Marks	
1	Compilation Errors	✔	5.0	Hide Answer

Create a class named **NumberIn** with the following attributes:
- An int variable **num**.
- A method **inputNum()** that takes user input for the number.

Create a class named SquareOut that inherits from **NumberIn** and has the following characteristics:
- A method **displaySquare()** that prints the square of the entered number to the console.

Note: The main class has been provided to you in the editor.

Execution Results — Show Results

Submitted Source Code :

```
//File Name: q23112/MainFunction.java
//================================
package q23112;
import java.util.Scanner;


// public class MainFunction {
//    public static void main(String[] args) {
//     SquareOut squareout = new SquareOut();
//       squareout.inputNum();
//       squareout.displaySquare();
//    }
// }
public class MainFunction{
      public static void main(String[] args){
            Scanner st=new Scanner(System.in);
            System.out.print("Enter number: ");
            int x=st.nextInt();
            System.out.println(x*x);
      }
}
```

2	Compilation Errors	✔	5.0	View Answer
3	Compilation Errors	✔		View Answer
4	Compilation			View Answer

You are tasked with creating a simple calculator application that performs addition and multiplication operations.

Define a base class **Calculation** with a constructor that takes two parameters. Implement a method **addition** in the **Calculation** class. The **addition** method should take the two parameters and return their sum.

Create a child class **My_Calculation** that inherits from the **Calculation** class. In the child class, implement a method **multiplication** that returns the product of the two parameters that are inherited from the base class.

Input format:
The two lines of the input are integers.

Output format:
The first line is the integer representing the result of the addition operation.
The second line is the integer representing the result of the multiplication operation.

Note:
- The code for handling inputs, creating objects, invoking methods, and printing results is already provided in the editor. Your task is to implement the **Calculation** class based on the given specifications.

Execution Results

View Results

Submitted Source Code :

```
//File Name: q23013/MainCalculation.java
//================================
package q23013;

import java.util.Scanner;


      // write your code here..

// public class MainCalculation {
//     public static void main(String[] args) {
//         Scanner scanner = new Scanner(System.in);

//         int num1 = scanner.nextInt();
//         int num2 = scanner.nextInt();

//         My_Calculation myCalculation = new My_Calculation(num1, num2);

//         int sum = myCalculation.addition();
//         int product = myCalculation.multiplication();

//         System.out.println(sum);
//         System.out.println(product);

//         scanner.close();
```

3	Compilation	View Answer
4	Compilation	View Answer

You are tasked with creating a simple calculator application that performs addition and multiplication operations.

Define a base class **Calculation** with a constructor that takes two parameters. Implement a method **addition** in the **Calculation** class. The **addition** method should take the two parameters and return their sum.

Create a child class **My_Calculation** that inherits from the **Calculation** class. In the child class, implement a method **multiplication** that returns the product of the two parameters that are inherited from the base class.

Input format:
The two lines of the input are integers.

Output format:
The first line is the integer representing the result of the addition operation.
The second line is the integer representing the result of the multiplication operation.

Note:

- The code for handling inputs, creating objects, invoking methods, and printing results is already provided in the editor. Your task is to implement the **Calculation** class based on the given specifications.

Execution Results

View Results

Submitted Source Code :

```
//      int num2 = scanner.nextInt();

//      My_Calculation myCalculation = new My_Calculation(num1, num2);

//      int sum = myCalculation.addition();
//      int product = myCalculation.multiplication();

//      System.out.println(sum);
//      System.out.println(product);

//      scanner.close();
//  }
// }

public class MainCalculation{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        int a=st.nextInt();
        int b=st.nextInt();
        System.out.println(a+b);
        System.out.println(b*a);
    }
}
```

3 Compilation View Answer

4 Compilation View Answer

3	Compilation Errors	✔	5.0	Hide Answer

Create a class named **Person** with the following characteristics:
- A String variable **name**.
- A method **inputName()** that takes user input for the name.
- A method **displayName()** that prints the name to the console.

Create a class named **Citizen** that inherits from **Person** and has the following characteristics:
- An **int** variable **age**.
- A method **inputAge()** that takes user input for the age.
- A method **displayAge()** that prints the age to the console.

Note: The main class has been provided to you in the editor. The **MainPerson** class creates an instance of **Citizen**, takes user input for name and age, and then displays the entered name and age. The program demonstrates the use of inheritance, where the Citizen class inherits attributes and methods from the Person class.

Execution Results — Show Results

Submitted Source Code :

```
package q23042;
import java.util.Scanner;

    // write your code here..

// public class MainPerson {
//     public static void main(String[] args) {
//         Citizen citizen = new Citizen();
//         citizen.inputName();
//         citizen.displayName();
//         citizen.inputAge();
//         citizen.displayAge();
//     }
// }

public class MainPerson{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("Enter name: ");
        String a=st.nextLine();
        System.out.println("Name: "+a);
        System.out.print("Enter age: ");
        String b=st.nextLine();
        System.out.println("Age: "+b);
    }
}
```

4	Compilation			View Answer

4 Compilation Errors ✔ 5.0

Design a Java program to model a simple banking system.

Create a class named **Account** with the following specifications:
- A String variable **accountNumber**.
- A double variable **balance**.
- A method **inputDetails()** that takes user input for the account number.
- A method **displayBalance()** that prints the current balance.

Create a class named **Transaction** that extends **Account** with the following characteristics:
- A double variable **amount**.
- A method **inputAmount()** that takes user input for the transaction amount for the deposit.
- Create a new method **performTransaction()** in the **Transaction** class that updates the balance based on the transaction amount.

Note: The main class has been provided to you in the editor.

Execution Results

Submitted Source Code :

Show Results

```
//File Name: q23127/BankingSystem.java
//================================
package q23127;
import java.util.Scanner;

//     // write your code here...

// public class BankingSystem {
//     public static void main(String[] args) {
//         Transaction transaction = new Transaction();
//         transaction.inputDetails();
//         transaction.displayBalance();
//         transaction.inputAmount();
//         transaction.performTransaction();
//     }
// }

public class BankingSystem{
        public static void main(String[] args){
                System.out.print("Enter account number: ");
                Scanner st=new Scanner(System.in);
                int a=st.nextInt();
                System.out.println("Current balance: 0.0");
                System.out.print("Enter transaction amount: ");
                double c=st.nextDouble();
```

4	Compilation Errors	✔	5.0

Design a Java program to model a simple banking system.

Create a class named **Account** with the following specifications:

- A String variable **accountNumber**.
- A double variable **balance**.
- A method **inputDetails()** that takes user input for the account number.
- A method **displayBalance()** that prints the current balance.

Create a class named **Transaction** that extends **Account** with the following characteristics:

- A double variable **amount**.
- A method **inputAmount()** that takes user input for the transaction amount for the deposit.
- Create a new method **performTransaction()** in the **Transaction** class that updates the balance based on the transaction amount.

Note: The main class has been provided to you in the editor.

Execution Results — Show Results

Submitted Source Code :

```
// public class BankingSystem {
//     public static void main(String[] args) {
//         Transaction transaction = new Transaction();
//         transaction.inputDetails();
//         transaction.displayBalance();
//         transaction.inputAmount();
//         transaction.performTransaction();
//     }
// }

public class BankingSystem{
    public static void main(String[] args){
        System.out.print("Enter account number: ");
        Scanner st=new Scanner(System.in);
        int a=st.nextInt();
        System.out.println("Current balance: 0.0");
        System.out.print("Enter transaction amount: ");
        double c=st.nextDouble();
        System.out.println("Transaction Successful!");
        System.out.println("Current balance: "+(c));

    }
}
```

37.1.1. Understanding Method Overloading

35:10

Method overloading means the ability to have multiple methods with same name, which vary in their parameters. For example:

```
public void concatenate(String text, int num) {
    return text + num;
}
public void concatenate(String text, boolean flag) {
    return text + flag;
}
public void concatenate(String text, char ch) {
    return text + ch;
}
```

In the above code, `concatenate` method is overloaded **three** times.

Note: In the above example, the variation in the parameters list can be by their count or type or both.

Select all the correct statements given below regarding methods present in the String class.

[**Hint:** You can explore the methods present in the String class by clicking on String and scrolling down until you reach a section heading named **Method Summary**. In that you will see table containing a list of all methods in the String class.]

- [x] `charAt` method is not overloaded.
- [x] `indexOf` method is overloaded **4 times**.
- [x] The `String` constructor is overloaded **15 times** .
- [ ] `valueOf` method is overloaded **8 times**.

< Prev Reset Submit Next >

2024, 23:36

Home Learn Anywhere

37.1.2. Write a Java program to implement Method overloading

30:10

Write a Java program with a class name `Addition` with the methods `add(int, int)`, `add(int, float)`, `add(float, float)` and `add(float, double, double)` to add values of different argument types.

Write the **main(String[])** method within the class and assume that it will always receive a total of **6** command line arguments at least, such that the first **2** are **int**, next **2** are **float** and the last **2** are of type **double**.

If the **main()** is provided with arguments : **1, 2, 1.5f, 2.5f, 1.0, 2.0** then the program should print the output as:

```
Sum of 1 and 2 : 3
Sum of 1.5 and 2.5 : 4.0
Sum of 2 and 2.5 : 4.5
Sum of 1.5, 1.0 and 2.0 : 4.5
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

Addition.j...

Submit

Plots

```
1  package q11266;
2  public class Addition{
3      void add(int num1,int num2){
4          System.out.println("Sum of "+num1+" and "+num2+" : "+
   (num1+num2));
5      }
6      void add(int num1, float num2){
7          System.out.println("Sum of "+num1+" and "+num2+" : "+
   (num1+num2));
8      }
9      void add(float num1, float num2){
10         System.out.println("Sum of "+num1+" and "+num2+" : "+
   (num1+num2));
11     }
12     void add(float num1,double num2 , double num3){
13         System.out.println("Sum of "+num1+", "+num2+" and
   "+num3+" : "+(num1+num2+num3));
14     }
15
16     public static void main(String s[]){
17         Addition ad = new Addition();
18         int a = Integer.parseInt(s[0]);
19         int b = Integer.parseInt(s[1]);
20         float c = Float.parseFloat(s[2]);
21         float d = Float.parseFloat(s[3]);
22
23         double e = Double.parseDouble(s[4]);
24         double f = Double.parseDouble(s[5]);
25         ad.add(a,b);
26         ad.add(c,d);
27         ad.add(b,d);
28         ad.add(c,e,f);
29     }
30 }
```

Terminal Test cases

‹ Prev Reset Submit Next ›

Home Learn Anywhere

37.1.3. Write a Java program to implement Constructor overloading

60:10

Write a class `Box` which contains the data members **width**, **height** and **depth** all of type **double**.

Write the implementation for the below **3** *overloaded constructors* in the class `Box` :

- **Box()** - default constructor which initializes all the members with **-1**
- **Box(length)** - parameterized constructor with one argument and initialize all the members with the value in **length**

the members with the corresponding arguments

- **Box(width, height, depth)** - parameterized constructor with three arguments and initialize

Write a method `public double volume()` in the class `Box` to find out the **volume** of the given box.

Write the **main** method within the `Box` class and assume that it will receive either **zero** arguments, or **one** argument or **three** arguments.

For example, if the **main()** method is passed **zero** arguments then the program should print the output as:

```
Volume of Box() is : -1.0
```

Similarly, if the **main()** method is passed **one** argument : **2.34**, then the program should print the output as:

```
Volume of Box(2.34) is : 12.812903999999998
```

then the program should print the output as: Likewise, if the **main()** method is passed **three** arguments : **2.34, 3.45, 1.59**, then the program should print the output as:

```
Volume of Box(2.34, 3.45, 1.59) is : 12.836070000000001
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

Box.java

Submit

```
1  package q11267;
2  public class Box{
3      double width;
4      double length;
5      double depth;
6      Box(){
7          width = -1;
8          length = -1;
9          depth = -1;
10     }
11     Box(double l){
12         length = l;
13         width = l;
14         depth = l;
15     }
16     Box(double width,double length,double depth){
17         this.width = width;
18         this.length = length;
19         this.depth = depth;
20
21     }
22     public Double volume(){
23         return width*length*depth;
24
25     }
26
27     public static void main(String s[]){
28         double a ,b,c;
29         if(s.length == 0 ){
30             Box bb = new Box();
31             System.out.println("Volume of Box() is :
"+bb.volume());
32         }
33         if(s.length == 1 ){
```

Terminal Test cases

‹ Prev Reset Submit Next ›

Home Learn Anywhere

37.1.3. Write a Java program to implement Constructor overloading

50:10

Write a class `Box` which contains the data members **width**, **height** and **depth** all of type **double**.

Write the implementation for the below **3** *overloaded constructors* in the class `Box` :

- **Box()** - default constructor which initializes all the members with **-1**
- **Box(length)** - parameterized constructor with one argument and initialize all the members with the value in **length**

the members with the corresponding arguments

- **Box(width, height, depth)** - parameterized constructor with three arguments and initialize

Write a method `public double volume()` in the class `Box` to find out the **volume** of the given box.

Write the **main** method within the `Box` class and assume that it will receive either **zero** arguments, or **one** argument or **three** arguments.

For example, if the **main()** method is passed **zero** arguments then the program should print the output as:

```
Volume of Box() is : -1.0
```

Similarly, if the **main()** method is passed **one** argument : **2.34**, then the program should print the output as:

```
Volume of Box(2.34) is : 12.812903999999998
```

then the program should print the output as: Likewise, if the **main()** method is passed **three** arguments : **2.34, 3.45, 1.59**, then the program should print the output as:

```
Volume of Box(2.34, 3.45, 1.59) is : 12.836070000000001
```

Note: Please don't change the package name.

Sample Test Cases

Box.java

Submit

```
17          this.width = width;
18          this.length = length;
19          this.depth = depth;
20
21      }
22      public Double volume(){
23          return width*length*depth;
24
25      }
26
27      public static void main(String s[]){
28          double a ,b,c;
29          if(s.length == 0 ){
30              Box bb = new Box();
31              System.out.println("Volume of Box() is : "+bb.volume());
32          }
33          if(s.length == 1 ){
34              a = Double.parseDouble(s[0]);
35              Box bb = new Box(a);
36              System.out.println("Volume of Box("+a+") is : "+bb.volume());
37          }
38          else if(s.length == 3 ){
39              a = Double.parseDouble(s[0]);
40              b = Double.parseDouble(s[1]);
41              c = Double.parseDouble(s[2]);
42              Box bb = new Box(a,b,c);
43              System.out.println("Volume of Box("+s[0]+", "+s[1]+", "+s[2]+") is : "+bb.volume());
44          }
45
46      }
47  }
```

Terminal Test cases

Prev Reset Submit Next

Home Learn Anywhere

37.1.4. Write a Java program to implement Method overloading

06:50

Write a Java program with a class name `OverloadArea` with overload methods `area(float)` and `area(float, float)` to find area of **square** and **rectangle**.

Write the **main** method within the class and assume that it will receive a total of **2** command line arguments of type **float**.

If the **main()** is provided with arguments : **1.34, 1.98** then the program should print the output as:

```
Area of square for side in meters 1.34 : 1.7956
Area of rectangle for length and breadth in meters 1.34, 1.98 : 2.6532001
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

Overload...

Submit

```
1  package q11268;
2  public class OverloadArea {
3      static float area(float a){
4          return a*a;
5      }
6      static float area(float a ,float b){
7          return a*b;
8      }
9      public static void main (String[] s) {
10         OverloadArea a = new OverloadArea();
11         float f1 = Float.parseFloat(s[0]);
12         float f2 = Float.parseFloat(s[1]);
13         System.out.println("Area of square for side in meters
           "+f1+":"+area(f1));
14         System.out.println("Area of rectangle for length and
           breadth in meters "+f1+","+f2+":"+area(f1,f2));
15     }
16 }
```

Terminal Test cases

Prev Reset Submit Next

Notes hub

Create a class named **Vehicle** with the following attributes:
- String variable make (represents the vehicle's make or manufacturer).
- String variable model (represents the vehicle's model).

Create a class named **CarInfo** that inherits from **Vehicle** and has the following characteristics:
- A method **inputCarDetails()** that takes user input for the car's make and model.
- A method **displayCarInfo()** that prints the car's make and model to the console.

Note: Driver code has already been provided, you just need to write the code for the classes.

Execution Results — Show Results

Submitted Source Code :

```
//File Name: q24933/Main.java
//================================
package q24933;
import java.util.Scanner;

// write your code here...


// public class Main {
//     public static void main(String[] args) {
//         CarInfo carData = new CarInfo();
//         carData.inputCarDetails();
//         carData.displayCarInfo();
//     }
// }

public class Main{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("Enter car make: ");
        String a=st.nextLine();
        System.out.print("Enter car model: ");
        String b=st.nextLine();
        System.out.println("Car Make: "+a);
        System.out.println("Car Model: "+b);
```

2	Compilation Errors	✗	2.75	View Answer
3	Compilatio			View Answer

Create a class named **Vehicle** with the following attributes:

- String variable make (represents the vehicle's make or manufacturer).
- String variable model (represents the vehicle's model).

Create a class named **CarInfo** that inherits from **Vehicle** and has the following characteristics:

- A method **inputCarDetails()** that takes user input for the car's make and model.
- A method **displayCarInfo()** that prints the car's make and model to the console.

Note: Driver code has already been provided, you just need to write the code for the classes.

Execution Results — Show Results

Submitted Source Code :

```
// write your code here...


// public class Main {
//     public static void main(String[] args) {
//         CarInfo carData = new CarInfo();
//         carData.inputCarDetails();
//         carData.displayCarInfo();
//     }
// }

public class Main{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("Enter car make: ");
        String a=st.nextLine();
        System.out.print("Enter car model: ");
        String b=st.nextLine();
        System.out.println("Car Make: "+a);
        System.out.println("Car Model: "+b);
    }
}
```

2	Compilation Errors	✗	2.75	View Answer
3	Compilatio			View Answer

2	Compilation Errors	✖	2.75	Hide Answer

Create a class named **Product** with the following attributes:
- String variable productName (represents the name of the product).
- double variable price (represents the price of the product).

Create a class named **ProductInfo** that inherits from **Product** and has the following characteristics:
- A method **inputProductDetails()** that takes user input for the product's name and price.
- A method **displayProductInfo()** that prints the product's name and price to the console.

Note: Driver code has already been provided, you just need to write the code for the classes.

Execution Results — Show Results

Submitted Source Code :

```
//File Name: q24935/Main.java
//=================================
package q24935;
import java.util.Scanner;

// write your code here...


// public class Main {
//     public static void main(String[] args) {
//         ProductInfo productData = new ProductInfo();
//         productData.inputProductDetails();
//         productData.displayProductInfo();
//     }
// }

public class Main{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("product name: ");
        String a=st.nextLine();
        System.out.print("price: $");
        String b=st.nextLine();
        System.out.println("product name: "+a);
        System.out.println("price: $"+b);
```

None of the participants of this test answered this question correctly. For the solution, please consult your faculty.

3	Compilatio			View Answer

2	Compilation Errors	✖	2.75	Hide Answer

Create a class named **Product** with the following attributes:

- String variable productName (represents the name of the product).
- double variable price (represents the price of the product).

Create a class named **ProductInfo** that inherits from **Product** and has the following characteristics:

- A method **inputProductDetails()** that takes user input for the product's name and price.
- A method **displayProductInfo()** that prints the product's name and price to the console.

Note: Driver code has already been provided, you just need to write the code for the classes.

Execution Results — Show Results

Submitted Source Code :

```
// write your code here...


// public class Main {
//     public static void main(String[] args) {
//         ProductInfo productData = new ProductInfo();
//         productData.inputProductDetails();
//         productData.displayProductInfo();
//     }
// }

public class Main{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("product name: ");
        String a=st.nextLine();
        System.out.print("price: $");
        String b=st.nextLine();
        System.out.println("product name: "+a);
        System.out.println("price: $"+b);
    }
}
```

None of the participants of this test answered this question correctly. For the solution, please consult your faculty.

3	Compilatio			View Answer

Create a class named **MathOperation** with the following attributes:

- int variable number1.
- int variable number2.

Create a class named **ArithmeticOperations** that inherits from **MathOperation** and has the following characteristics:

- A method **inputNumbers()** that takes user input for number1 and number2.
- A method **displaySum()** that calculates and prints the sum of the numbers.
- A method **displayDifference()** that calculates and prints the difference (number1 - number2).
- A method **displayProduct()** that calculates and prints the product of the numbers.
- A method **displayQuotient()** that calculates and prints the quotient (number1 / number2).

Note: Driver code has already been provided, you just need to write the code for the classes.

Execution Results Show Results

Submitted Source Code :

```
//File Name: q24937/Main.java
//================================
package q24937;
import java.util.Scanner;

// write your code here...


// public class Main {
//     public static void main(String[] args) {
//         ArithmeticOperations mathOps = new ArithmeticOperations();
//         mathOps.inputNumbers();
//         mathOps.displaySum();
//         mathOps.displayDifference();
//         mathOps.displayProduct();
//         mathOps.displayQuotient();
//     }
// }

public class Main{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("first number: ");
        int a=st.nextInt();
        System.out.print("second number: ");
```

4 Compilatio View Answer

2024 23:40

Create a class named **MathOperation** with the following attributes:

- int variable number1.
- int variable number2.

Create a class named **ArithmeticOperations** that inherits from **MathOperation** and has the following characteristics:

- A method **inputNumbers()** that takes user input for number1 and number2.
- A method **displaySum()** that calculates and prints the sum of the numbers.
- A method **displayDifference()** that calculates and prints the difference (number1 - number2).
- A method **displayProduct()** that calculates and prints the product of the numbers.
- A method **displayQuotient()** that calculates and prints the quotient (number1 / number2).

Note: Driver code has already been provided, you just need to write the code for the classes.

Execution Results Show Results

Submitted Source Code :

```
//      mathOps.displayDifference();
//      mathOps.displayProduct();
//      mathOps.displayQuotient();
//   }
// }

public class Main{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("first number: ");
        int a=st.nextInt();
        System.out.print("second number: ");
        int b=st.nextInt();
        System.out.println("Sum: "+(a+b));
        System.out.println("Difference: "+(a-b));
        System.out.println("Product: "+(a*b));
        double x=(double)a/b;
        if(b==0)
        System.out.println("Cannot divide by zero. Quotient is undefined.");
        else
        System.out.println("Quotient: "+x);
    }
}
```

4 Compilatio View Answer

4	Compilation Errors	✔	5.0	Hide Answer

Create a class named **NumberIn** with the following attributes:
- An int variable **num**.
- A method **inputNum()** that takes user input for the number.

Create a class named **CubeOut** that inherits from **NumberIn** and has the following characteristics:
- A method **displayCube()** that prints the square of the entered number to the console.

Note: The main class has been provided to you in the editor.

Execution Results — Show Results

Submitted Source Code :

```
//File Name: q24924/MainFunction.java
//================================
package q24924;
import java.util.Scanner;

      // write your code here..

// public class MainFunction {
//    public static void main(String[] args) {
//     CubeOut cubeeout = new CubeOut();
//       cubeeout.inputNum();
//       cubeeout.displayCube();
//    }
// }

public class MainFunction{
      public static void main(String[] args){
            System.out.print("Enter number: ");
            Scanner st=new Scanner(System.in);
            int a=st.nextInt();
            System.out.println(a*a*a);
      }
}
```

Home Learn Anywhere

38.1.1. Understanding Overriding

In implementation of inheritance, when a class B inherits from a class A, the subclass B can modify the implementation present in its superclass A. For example:

```
class A {
    public int aValue = 2;
    public int getAValue() {
        return aValue;
    }
}
class B extends A {
    public int bValue = 3;
    public int getBValue() {
        return bValue;
    }
    public int getAValue() {  // this method overrides the implementation in class A
        return 2 * aValue; // returning  double of value stored in aValue
    }
}
Then the below will print 4 and not 2:
B b = new B();
System.out.println(b.getAValue());
```

Note: In the above code, the method `getAValue()` in `class B` is overriding the method with same name present in its superclass `A`.

Whenever a method is overridden in the subclass, the new method implementation will be called when the method is invoked on the instance of subclass.

Observe and understand the code given in the editor and complete it:

- The **A** class has an instance variable **aValue**, along with a **getAValue()** method.
- The **B** class will extend **A** and should have an additional instance variable **bValue** and a **getBValue()** method.
- In the **getAValue()** method of class **B**, we override the implementation from class **A** directly to access the implementation in class **A** and multiply it by 2.
- In the **main()** method, the user is prompted to enter the value for **A** and **B** using the Scanner class.
- A **B** object is created, and the **aValue** and **bValue** are set based on the user's input.

Sample Test Cases

Explorer

Overridin...

Submit

```
1  import java.util.Scanner;
2  class A {
3      public int aValue;
4      public int getAValue() {
5          return aValue;
6      }
7  }
8  class B extends A {
9      public int bValue;
10     public int getBValue(){
11         return bValue;
12     }
13     @Override
14     public int getAValue(){
15         return 2* aValue;
16     }
17 }
18 public class OverridingExample {
19     public static void main(String[] args) {
20         Scanner scanner = new Scanner(System.in);
21         B b = new B();
22         System.out.print("Enter the value for A: ");
23         int aValue = scanner.nextInt();
24         b.aValue = aValue;
25
26         System.out.print("Enter the value for B: ");
27         int bValue = scanner.nextInt();
28         b.bValue = bValue;
29
30         System.out.println("b.getAValue(): " + b.getAValue());
31         System.out.println("b.getBValue(): " + b.getBValue());
32     }
33 }
```

Terminal Test cases

Prev Reset Submit Next

Plots

Notes hub

29, 2024, 23:42

38.1.2. Difference between overloading and overriding

Method overloading is a feature which allows multiple methods with same name and different parameters in the same class.

Method overriding is a feature where we specialize (or modify) a method behaviour which is already present in the superclass. This takes places only when we have a class extending another class.

Select all the correct statements from the below code:

```
class A {
    public void printMe(int number) { // statement 1
        System.out.println(number);
    }
    public void printMe(boolean flag) { // statement 2
        System.out.println(flag);
    }
    public void printMe(int number, boolean flag) { // statement 3
        System.out.println(number + " : " + flag);
    }
}
class B extends A {
    public void printMe(int number) { // statement 4
        System.out.println("The double of " + number + " is : " + (number * 2));
    }
}
```

- [x] Statements 1, 2 and 3 in class A contain overloaded versions of method `printMe`.
- [ ] Statements 1, 2 and 3 in class A contain overridden versions of method `printMe`.
- [ ] The method in statement 4, overrides the method declared in statement 2.
- [x] The method in statement 4, overrides the method declared in statement 1.

The method declaration in statement 3 has more parameters than the methods declared in statements 1 and 2, hence it will not be considered as a overloaded version of `printMe`.

Home Learn Anywhere

38.1.3. Write a Java program to achieve concept of Method Overriding

05:10

Assume there is a class called `Bank` with method `calculateInterest(float principal, int time)`.

Create sub-classes of `Bank` with names `SBI`, `ICICI` and `AXIS` and override the `calculateInterest(float principal, int time)` method.

Create a constant of type `float` called `INTEREST_RATE` in classes `SBI`, `ICICI` and `AXIS` with values `10.8`, `11.6` and `12.3` respectively.

Use the formula **(principal * INTEREST_RATE * time) / 100** to calculate the **interest** for given **principal** and **time** and return the value as `float` in the **overriden method**.

For example, if the **two** arguments passed to the **main** method are 1000 and 5, (principal and time) below is the expected output:

```
SBI rate of interest = 540.0
ICICI rate of interest = 580.0
AXIS rate of interest = 615.0
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

TestOverr...

Submit

```
1  package q11271;
2  class Bank {
3      float calculateInterest(float principal, int time){
4          return 0;
5      }}
6  class SBI extends Bank {
7      private static final float INTEREST_RATE = 10.8f;
8      float calculateInterest(float principal, int time){
9          return (principal*time*INTEREST_RATE)/100;
10     }}
11 class ICICI extends Bank{
12     private static final float INTEREST_RATE = 11.6f;
13     float calculateInterest(float principal, int time){
14         return (principal*INTEREST_RATE*time)/100;
15     }}
16 class AXIS extends Bank{
17     private static final float INTEREST_RATE = 12.3f;
18     float calculateInterest(float principal, int time){
19         return (principal*time*INTEREST_RATE)/100;}
20 }
21 public class TestOverriding {
22     public static void main(String[] args) {
23         Bank sbiBank = new SBI();
24         Bank  iciciBank = new ICICI();
25         Bank axisBank = new AXIS();
26         float principal = Float.parseFloat(args[0]);
27         int time = Integer.parseInt(args[1]);
28         System.out.println("SBI rate of interest = " +
   sbiBank.calculateInterest(principal,time));
29         System.out.println("ICICI rate of interest = "
   +iciciBank.calculateInterest(principal,time) );
30         System.out.println("AXIS rate of interest = " +
   axisBank.calculateInterest(principal,time));
31     }}
```

Terminal Test cases

< Prev Reset Submit Next >

39.1.1. Understanding super keyword

15:10

The `super` keyword in **java** is used to refer/access either a member field, method or a constructor present in the super class hierarchy of the current class where the `super` keyword is used.

The **super** keyword can be used to :

- access the member fields of parent class when both parent and child class have member fields with same name
- explicitly call the default or parameterized constructors of parent class
- access the method of parent class when child class has overridden that method

When both the child and parent class contain variables with same names, we can access the member field of parent class inside the child class by using the syntax:

```
super.variableName
```

When an **instance** of subclass is created, the **new** keyword invokes the **constructor** of child class. This invocation of constructor in the subclass **implicitly** invokes the constructor of the **parent** class, if we do not explicitly write code to invoke the constructor of super class using **super**.

Hence, we should always remember that when we create an object of child class, the **parent** class constructor is executed first and then the **child** class constructor is executed.

Supposing we do not write code to call the super class constructor in the child class constructor, then the compiler implicitly adds `super()` (this invokes the **no argument constructor** of parent class) as the first statement in the constructor of **child** class. If the parent class does not have a default constructor (constructor with no parameters), then the programmer should explicitly call the parameterized constructor else the compiler will flag and error saying there is no default constructor in the super class.

Below is the syntax for calling the **parameterized** constructor of a **parent** class :

```
super(parameter-list);
```

Sample Test Cases

Explorer

SuperKey...

Submit

```
1   package q11272;
2   class SuperClass {
3       int num;
4       public SuperClass(int value) {
5           num = value;
6       }
7       public void printHello() {
8           System.out.println("Hello from SuperClass");
9       }
10  }
11
12  class SubClass extends SuperClass {
13      int num;
14      SubClass(int value){
15          super(value);
16          num = value +5;
17          System.out.println("SuperClass number = "+super.num);
18          System.out.println("SubClass number = "+num);
19      }
20      public void printHello(){
21      super.printHello();
22
23      System.out.println("Hello from SubClass");
24      }
25  }
26
27  public class SuperKeyword {
28      public static void main(String[] args) {
29          SubClass obj = new SubClass(10);
30          obj.printHello();
31      }
32  }
```

Terminal Test cases

‹ Prev Reset Submit Next ›

Home Learn Anywhere

39.1.2. Write a Java program to illustrate super keyword

30:10

Write a Java program to illustrate the usage of **super** keyword.

Create a class called `Animal` with the below members:

- a constructor which prints **Animal is created**
- a method called **eat()** which will print **Eating something** and returns nothing.

Create another class called `Dog` which is derived from the class `Animal`, and has the below members:

- a constructor which calls **super()** and then prints **Dog is created**
- a method **eat()** which will print **Eating bread** and returns nothing
- a method **bark()** which will print **Barking** and returns nothing
- a method **work()** which will call **eat()** of the **superclass** first and then the **eat()** method in the current class, followed by the **bark()** method in the current class.

Write a class `ExampleOnSuper` with the **main()** method, create an object to `Dog` which calls the method **work()**.

Note: Please don't change the package name.

Sample Test Cases

Explorer

Example...

Submit

Plots

```
1  package q11273;
2  class Animal {
3      public Animal() {
4          System.out.println("Animal is created");
5      }
6      void eat() {
7          System.out.println("Eating something");
8      }
9  }
10 class Dog extends Animal {
11     public Dog() {
12         super();
13         System.out.println("Dog is created");
14     }
15     void eat() {
16         System.out.println("Eating bread");
17     }
18     void bark() {
19         System.out.println("Barking");
20     }
21     void work() {
22         super.eat();
23         eat();
24         bark();
25     }
26 }
27 public class ExampleOnSuper {
28     public static void main(String args[]) {
29         Dog d = new Dog();
30         d.work();
31     }
32 }
```

Terminal Test cases

< Prev Reset Submit Next >

Feb 29, 2024, 23:45

Home Learn Anywhere

39.1.3. Write a Java program to Access the Class members using super Keyword

16:55

Write a Java program to access the class members using **super** Keyword.

Create a class called `SuperClass` with the below members:

- declare two member fields **value1** and **value2** of type **int**
- a parameterized constructor with **two** arguments, which assigns two arguments to the members respectively
- a method called **show()** which will print **This is super class show() method** as well as the value of **value1**.

Create another class called `SubClass` which is derived from the class `SuperClass`, and has the below members:

- declare two member fields **value3** and **value4** of type **int**
- a parameterized constructor with **four** arguments, which assigns the first two arguments with **SuperClass** members and next two values with **SubClass** members
- a method called **show()** which
  1. will print **This is sub class show() method**
  2. will call **show()** of **SuperClass**
  3. will print **value2** from **SuperClass**
  4. will print **value3** of **SubClass**
  5. will print **value4** of **SubClass**

Write a class `AccessUsingSuper` with the **main()** method, create an object to `SubClass` which calls the method **show()**.

For example, if the input is given as [10, 20, 30, 40] then the output should be:

```
This is sub class show() method
This is super class show() method
value1 = 10
value2 from super class = 20
value3 = 30
value4 = 40
```

Sample Test Cases

Explorer

AccessU...

Submit

```
1  package q11274;
2  class SuperClass {
3      int value1, value2;
4      SuperClass(int val1 , int val2){
5          value1 = val1;
6          value2 = val2;
7      }
8      void show(){
9          System.out.println("This is super class show()
   method\nvalue1 = "+value1);
10     }}
11 class SubClass extends SuperClass {
12     int value3, value4;
13     SubClass(int val1 , int val2 ,int val3 , int val4)
   {super(val1,val2);
14         value3 = val3;
15         value4 = val4;}
16     void show(){
17         System.out.println("This is sub class show() method");
18         super.show();
19         System.out.println("value2 from super class =
   "+super.value2);
20         System.out.println("value3 = "+value3);
21         System.out.println("value4 = "+value4);
22     }}
23
24 public class AccessUsingSuper {
25     public static void main(String[] args) {
26         SubClass obj = new SubClass(Integer.parseInt(args[0]),
   Integer.parseInt(args[1]), Integer.parseInt(args[2]),
   Integer.parseInt(args[3]));
27         obj.show();
28     }
29 }
```

Terminal Test cases

Prev Reset Submit Next

Plots

Notes hub

Feb 29, 2024, 23:46

Q No.	Q. Type	Status	Marks	
1	Compilation Errors	✖	2.75	Hide Answer

You are tasked with implementing a 3D shape calculator in Java.

- Design a class hierarchy with a base class **Shape** and two derived classes **Cube** and **Cuboid**.
- The **Shape** class should have a method **calculateVolume()** that prints "Calculating volume of Shape."
- The **Cube** class should override the **calculateVolume()** method to calculate and display the volume of a cube by taking any side (double) of the cube as input from the user.
- The **Cuboid** class should also override the **calculateVolume()** method to calculate and display the volume of a cuboid by taking the length (double), width (double), and height (double) of the cuboid as input from the user.

Formulas:

Volume of cube = $(side)^3$

Volume of the cuboid = length * width * height

Note:

- Print the volume up to 2 decimal places.
- The main class has been provided to you in the editor.

Execution Results — Show Results

Submitted Source Code :

```
//File Name: q24936/ShapeCalculator.java
//================================
package q24936;

import java.util.Scanner;
import java.text.DecimalFormat;
class Cube{
	public static void calculateVolume(){
		Scanner st=new Scanner(System.in);
		System.out.print("Side of the cube: ");
		double a= st.nextDouble();
		System.out.print("Volume of Cube: ");
		DecimalFormat format=new DecimalFormat("##.00");
		System.out.printf(format.format(a*a*a));
		System.out.println();
	}
}
class Cuboid{
	public static void calculateVolume(){
		Scanner pt = new Scanner(System.in);
		System.out.print("Length of the cuboid: ");
		double a=pt.nextDouble();
		System.out.print("Width of the cuboid: ");
		double b=pt.nextDouble();
		System.out.print("Height of the cuboid: ");
```

Q No.	Q. Type	Status	Marks	
1	Compilation Errors	✖	2.75	Hide Answer

You are tasked with implementing a 3D shape calculator in Java.

- Design a class hierarchy with a base class **Shape** and two derived classes **Cube** and **Cuboid**.
- The **Shape** class should have a method **calculateVolume()** that prints "Calculating volume of Shape."
- The **Cube** class should override the **calculateVolume()** method to calculate and display the volume of a cube by taking any side (double) of the cube as input from the user.
- The **Cuboid** class should also override the **calculateVolume()** method to calculate and display the volume of a cuboid by taking the length (double), width (double), and height (double) of the cuboid as input from the user.

Formulas:

Volume of cube = $(side)^3$

Volume of the cuboid = length * width * height

Note:

- Print the volume up to 2 decimal places.
- The main class has been provided to you in the editor.

Execution Results — Show Results

Submitted Source Code :

```
        Scanner pt = new Scanner(System.in);
        System.out.print("Length of the cuboid: ");
        double a=pt.nextDouble();
        System.out.print("Width of the cuboid: ");
        double b=pt.nextDouble();
        System.out.print("Height of the cuboid: ");
        double c=pt.nextDouble();
        DecimalFormat format=new DecimalFormat("##.00");
        System.out.println("Volume of Cuboid: "+format.format(a*b*c));
    }
}

public class ShapeCalculator {
    public static void main(String[] args) {
        // Creating instances of the subclasses
        Cube cubeInstance = new Cube();
        Cuboid cuboidInstance = new Cuboid();

        // Calling calculateVolume() for each instance
        cubeInstance.calculateVolume();
        cuboidInstance.calculateVolume();
    }
}
```

You are tasked with implementing a basic shape calculator in Java.

- Design a class hierarchy with a base class **Shape** and two derived classes **Square** and **Triangle**.
- The **Shape** class should have a method **calculateArea()** that prints "Calculating area of Shape."
- The **Square** class should override the **calculateArea()** method to calculate and display the area of a square by taking the side (double) of the square as input from the user.
- The **Triangle** class should also override the **calculateArea()** method to calculate and display the area of a triangle by taking the base (double) and height (double) of the triangle as input from the user.

Formulas:

Area of the square = $(side)^2$

Area of the triangle = 0.5 * base * height

Note:

- Print the area up to 2 decimal places.
- The main class has been provided to you in the editor.

Execution Results

Show Results

Submitted Source Code :

```
//File Name: q24934/BasicShapeCalc.java
//================================
package  q24934;
import java.util.Scanner;
import java.text.DecimalFormat;
class Square{
    public static void calculateArea(){
        Scanner st=new Scanner(System.in);
        DecimalFormat format = new DecimalFormat("##.00");
        System.out.print("Side of the square: ");
        double a=st.nextDouble();
        System.out.println("Area of Square: " + format.format(a*a));


    }
}
class Triangle{
    public static void calculateArea(){
        Scanner st=new Scanner(System.in);
        DecimalFormat format=new DecimalFormat("##.00");
        System.out.print("Base of the triangle: ");
        double a=st.nextDouble();
        System.out.print("Height of the triangle: ");
        double b=st.nextDouble();
        System.out.println("Area of Triangle: "+format.format(0.5*a*b));
```

3 Compilation Errors

View Answer

Feb 29, 2024, 23:47

- The **Triangle** class should also override the **calculateArea()** method to calculate and display the area of a triangle by taking the base (double) and height

Formulas:
Area of the square = $(side)^2$
Area of the triangle = 0.5 * base * height

Note:
- Print the area up to 2 decimal places.
- The main class has been provided to you in the editor.

Execution Results — Show Results

Submitted Source Code :

```
        Scanner st=new Scanner(System.in);
        DecimalFormat format=new DecimalFormat("##.00");
        System.out.print("Base of the triangle: ");
        double a=st.nextDouble();
        System.out.print("Height of the triangle: ");
        double b=st.nextDouble();
        System.out.println("Area of Triangle: "+format.format(0.5*a*b));
    }
}



public class BasicShapeCalc {
  public static void main(String[] args) {
    // Creating instances of the subclasses
    Square squareInstance = new Square();
    Triangle triangleInstance = new Triangle();

    // Calling calculateArea() for each instance
    squareInstance.calculateArea();
    triangleInstance.calculateArea();
  }
}
```

3	Compilation Errors	✔	5.0	View Answer

3	Compilation Errors	✔	5.0

You are tasked with implementing a basic shape calculator in Java.

- Design a class hierarchy with a base class **Shape** and two derived classes **Circle** and **Rectangle**.
- The **Shape** class should have a method **calculateArea()** that prints "Calculating area of Shape."
- The **Circle** class should override the **calculateArea()** method to calculate and display the area of a circle by taking the radius (double) of the circle as input from the user,
- The **Rectangle** class should also override the **calculateArea()** method to calculate and display the area of a rectangle by taking the length (double) and width (double) of the rectangle as input from the user.

Note:

- Print the area up to 2 decimal places.
- The main class has been provided to you in the editor.

Execution Results

Show Results

Submitted Source Code :

```
//File Name: q23030/ShapeTest.java
//=================================
package q23030;
import java.util.Scanner;
import java.text.DecimalFormat;
class Circle{
    public static void calculateArea(){
        Scanner st=new Scanner(System.in);
        System.out.print("radius of circle: ");
        double a=st.nextDouble();
        DecimalFormat format=new DecimalFormat("##.00");
        System.out.println("Area of circle: "+format.format((Math.PI)*(a*a)));
    }
}

class Shape{
    public static void calculateArea(){
        System.out.println("Calculating area of Shape");

    }
}
class Rectangle{
    public static void calculateArea(){
        Scanner st=new Scanner(System.in);
        System.out.print("length of rectangle: ");
```

3	Compilation Errors	✔	5.0

You are tasked with implementing a basic shape calculator in Java.

- Design a class hierarchy with a base class **Shape** and two derived classes **Circle** and **Rectangle**.
- The **Shape** class should have a method **calculateArea()** that prints "Calculating area of Shape."
- The **Circle** class should override the **calculateArea()** method to calculate and display the area of a circle by taking the radius (double) of the circle as input from the user,
- The **Rectangle** class should also override the **calculateArea()** method to calculate and display the area of a rectangle by taking the length (double) and width (double) of the rectangle as input from the user.

Note:

- Print the area up to 2 decimal places.
- The main class has been provided to you in the editor.

Execution Results

Show Results

Submitted Source Code :

```
            System.out.print("length of rectangle: ");
            double a=st.nextDouble();
            System.out.print("width of rectangle: ");
            double b=st.nextDouble();
            DecimalFormat format=new DecimalFormat("##.00");
            System.out.println("Area of rectangle: "+format.format(a*b));
        }
}
    // write your code here..


public class ShapeTest {
    public static void main(String[] args) {
        Shape shape = new Shape();
        shape.calculateArea();

        Circle circle = new Circle();
        circle.calculateArea();

        Rectangle rectangle = new Rectangle();
        rectangle.calculateArea();
    }
}
```

Home Learn Anywhere

40.1.1. Understanding polymorphism and IS-A relationship

25:10

In Java, since Object class is the superclass (root class) of every class, instance of any class **is also an instance** of Object class.

For example:

```
class Person {
}
Person p = new Person();
```

In the above code, the instance referred by `p` IS-A `Person`. Since every class in Java, including `Person` is a subclass of Object, the statement `p` IS-A Object is also correct.

Which means every object in Java will have more than one IS-A relationship (One with its own class type and one with Object class type).

Any object which satisfies more than one IS-A relation is called polymorphic. For example, let us write classes `A`, `B` and `C` which override **toString()** method of **Object** class.

Observe the given code carefully to understand the polymorphic behaviour and complete the incomplete code.

1. You will notice that `System.out.println(a);` is same as `System.out.println(a.toString());`, because the **println** method which takes any object internally calls `toString()` method on that reference.
2. You will notice that the overridden `toString()` method in class `B` first invokes the `toString()` method present in class `A` using the `super` keyword.
3. Similar is the case with class `C`.
4. Both the overridden methods in `B` and `C` append their own information to the value returned by the superclass's `toString()` method.

Notice that references `a`, `b`, `c` are declared to be of type class `Object`, since all classes are subtypes of class `Object`.

For the above stated reason we are able to hold the instances of type `A`, `B` and `C` in references

Sample Test Cases

Explorer

Polymorp...

Submit

Plots

```
1  package q11276;
2  public class PolymorphismExample2 {
3      public static void main(String[] args) {
4          /* create three objects a, b, c of the
5             classes A, B ,C respectively */
6          A a = new A();
7          B b = new B();
8          C c = new C();
9          System.out.println(a);
10         System.out.println(b);
11         System.out.println(c.toString());
12     }
13 }
14 class A {
15     public String toString() {
16         return "A";
17     }
18 }
19 class B extends A {
20     public String toString() {
21         return super.toString() + " " + "B";
22     }
23 }
24 class C extends B {
25     public String toString() {
26         return super.toString() + " " + "C";
27     }
28 }
```

< Prev | Reset | Submit | Next >

, 2024, 23:48

Home Learn Anywhere

40.1.2. Write a Java program that implements Runtime Polymorphism

13:39

Write a Java program that implements **runtime polymorphism**.

Create a class `Animal` with one method `whoAmI()` which will print **I am a generic animal**.

Create another class `Dog` which extends `Animal`, which will print **I am a dog**.

Create another class `Cow` which extends `Animal`, which will print **I am a cow**.

Create another class `Snake` which extends `Animal`, which will print **I am a snake**.

Write a class `RuntimePolymorphismDemo` with the **main()** method, create objects to all the classes `Animal`, `Dog`, `Cow`, `Snake` and call `whoAmI()` with each object.

Note: Please don't change the package name.

Sample Test Cases

Explorer

RuntimeP...

Submit

Plots

```
1  package q11277;
2  public class RuntimePolymorphismDemo {
3      public static void main(String[] args) {
4          Animal ref1 = new Animal();
5          Animal ref2 = new Dog();
6          Animal ref3 = new Cow();
7          Animal ref4 = new Snake();
8          ref1.whoAmI();
9          ref2.whoAmI();
10         ref3.whoAmI();
11         ref4.whoAmI();
12     }
13 }
14 // Write all the classes with methods
15 class Animal{
16     public void whoAmI(){
17     System.out.println("I am a generic animal");
18     }
19 }
20 class Dog extends Animal{
21     public void whoAmI(){
22     System.out.println("I am a dog");
23     }
24 }
25 class Cow extends Animal{
26     public void whoAmI(){
27     System.out.println("I am a cow");
28     }
29 }
30 class Snake extends Animal{
31     public void whoAmI(){
32     System.out.println("I am a snake");
33     }
34 }
```

Terminal Test cases

Prev Reset Submit Next

Notes hub

eb 29, 2024, 23:49

41.1.1. Understanding Object class

00:21

In Java **Object** is the root class of all classes. Which means that all the methods of Object class described below are also present in each and every class in Java.

1. **clone()** - creates and returns a copy of this object. However, for this method to work the class has to implement Cloneable interface.
2. **equals(Object obj)** - it compares if two references are pointing to the same address. However, you can override this method to provide custom implementation which will verify the contents and not just references.We will learn more about it in the later sections.
3. **finalize()** - this method is called by the garbage collector on an object when garbage collection determines that there are no more references to the object. Programmers do not call it, but can override and write cleanup code in it.
4. **getClass()** - this method returns the in-memory runtime representation of the **Class** object that was loaded and used to create the instance. For example:

```
Student st = new Student();
Class clazz = st.getClass();
```

   The reference **clazz** points to the Class object which has the details of the Student class loaded in memory.
5. **hashCode()** - this method returns a hash code value (an integer) for the object. (we will learn more about it in data structures)
6. **toString()** - this method returns the string representation of the object.

There are many other methods like **notify**, **notifyAll**, and **wait** which are used to synchronize data access and method calls when multiple threads are involved.

Can we directly create an object of the **Object** class using the **new** keyword?

- Yes, it is allowed
- (•) No, it is not allowed

2024, 23:49

Home Learn Anywhere

42.1.1. Understanding the toString method in Object class

30:10

The `toString` present in the `Object` class ensures that an object of any class in Java can be converted into a String representation.

If we do not override the `toString` method, by default the `toString` method present in the Object's class will be called (which is of no much use).

Observe the Code given in the Editor

- The **ToStringExample** class demonstrates how the **toString()** method can be used to obtain a customized string representation of an object.
- It concatenates the string **a.toString() :** with the result of the **toString()** method of the A class.

Now, Complete the code for a **class A** that matches the following requirements

1. The class should have a private integer instance variable.
2. The class should have a constructor that takes an integer parameter and initializes the instance variable.
3. The class should override the **toString()** method inherited from the Object class.
4. The **toString()** method should return a string in the format: **"The value is : "** where represents the value of the instance variable.

Note: Please don't change the package name.

Sample Test Cases

Explorer

ToStringE...

Submit

Plots

```
1  package q11279;
2  public class ToStringExample {
3      public static void main(String[] args) {
4          A a = new A(4);
5          System.out.println("a.toString() : " + a.toString());
6          System.out.println("a : " + a);
7      }
8  }
9  class A{
10     int a;
11     A(int a){
12         this.a = a;
13     }
14     public String toString (){
15         return "The value is : "+a;
16     }
17     
18 }
```

Terminal Test cases

‹ Prev Reset Submit Next ›

eb 29, 2024, 23:49

Home Learn Anywhere

42.1.2. Understanding the equals method in Object class

If we do not override the `equals` method, by default the equals method present in the Object's class will be called.

The implementation of `equals` present in the root class Object, only verifies if the references are the same. It does not verify if the contents are the same. Content verification is the responsibility of the overriding implementation.

There is an incomplete code provided to you in the Editor, follow the comment lines in order to complete the code in the Main class and The functionality of class A is described below:

- `class A` is a simple class with a private int field named value and a constructor that takes an `int` parameter to initialize the value field.
- The class also defines an `equals()` method that overrides the default `equals()` method inherited from the Object class.
- If the objects are not the same instance, it checks if the `otherObject` is an instance of class A using the `instanceof` operator
  1. If `otherObject` is an instance of class A, it casts `otherObject` to type A and assigns it to a variable `otherARef`
  2. It then compares the value of the current object (this. value) with the value of `otherARef`.If the values are the same, it means the objects are equal, so it returns true.

Now complete the code in the Editor By following the above statements and Observe the Output.

Note:

- In the main() method, the code prompts the user to input values for a_1, a_2, and a_3, which are used to create three instances of **class A**. It then calls the equals() method on a1 to compare it with a2 and a3, printing the results.
- In the same way, the code prompts the user to input values for b_1, b_2, and b_3, which are used to create three instances of **class B**
- Please don't change the package name

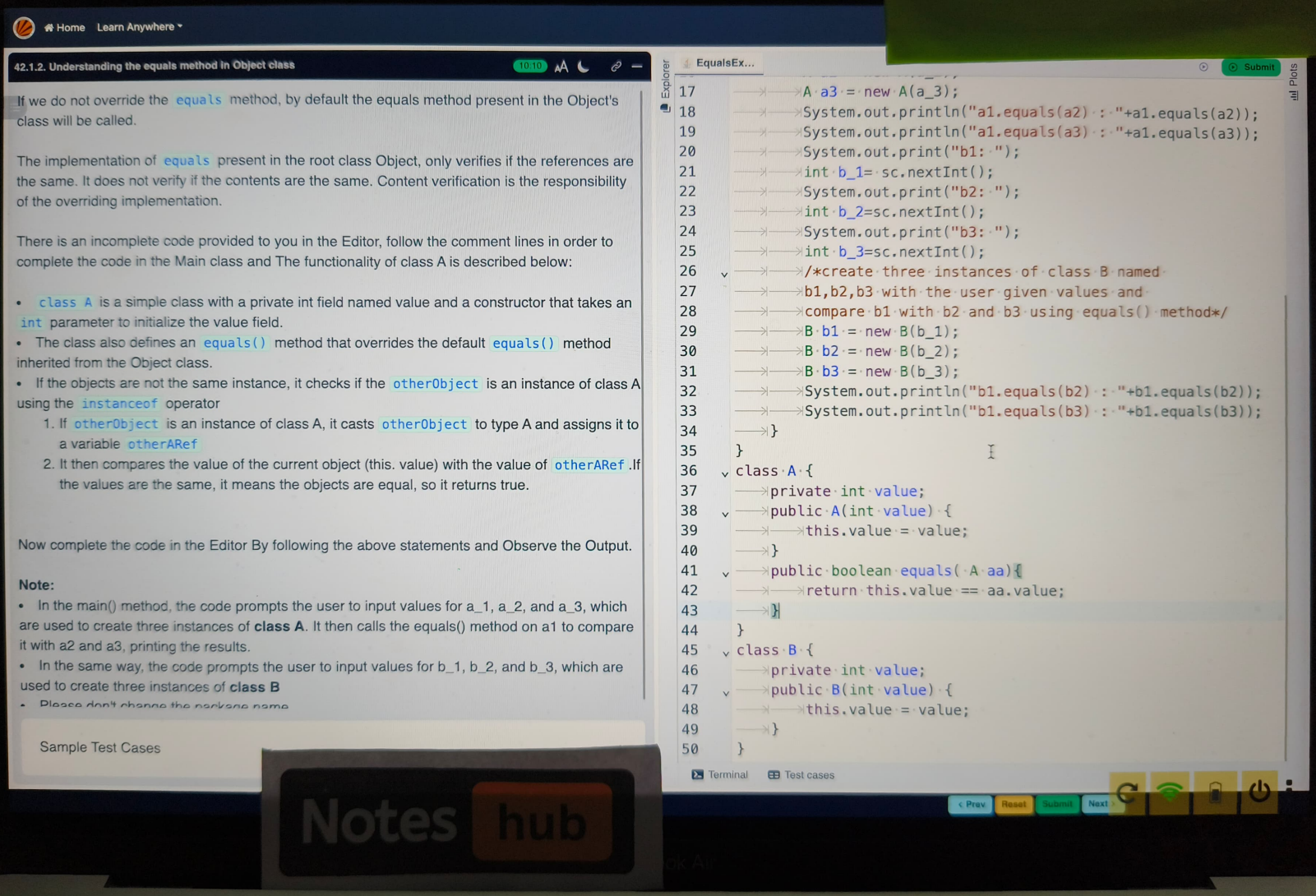
Sample Test Cases

Explorer EqualsEx... Submit

```
1  package q11280;
2  import java.util.*;
3  public class EqualsExample {
4      public static void main(String[] args) {
5          Scanner sc= new Scanner(System.in);
6          System.out.print("a1: ");
7          int a_1= sc.nextInt();
8          System.out.print("a2: ");
9          int a_2=sc.nextInt();
10         System.out.print("a3: ");
11         int a_3=sc.nextInt();
12     /*create three instances of class A named a1,a2,a3
13     with the user given values and compare a1 with
14     a2 and a3 using equals() method*/
15         A a1 = new A(a_1);
16         A a2 = new A(a_2);
17         A a3 = new A(a_3);
18         System.out.println("a1.equals(a2) : "+a1.equals(a2));
19         System.out.println("a1.equals(a3) : "+a1.equals(a3));
20         System.out.print("b1: ");
21         int b_1= sc.nextInt();
22         System.out.print("b2: ");
23         int b_2=sc.nextInt();
24         System.out.print("b3: ");
25         int b_3=sc.nextInt();
26         /*create three instances of class B named
27         b1,b2,b3 with the user given values and
28         compare b1 with b2 and b3 using equals() method*/
29         B b1 = new B(b_1);
30         B b2 = new B(b_2);
31         B b3 = new B(b_3);
32         System.out.println("b1.equals(b2) : "+b1.equals(b2));
33         System.out.println("b1.equals(b3) : "+b1.equals(b3));
34     }
```

Terminal Test cases

Prev Reset Submit Next


Home
Learn Anywhere
42.1.2. Understanding the equals method in Object class
10:10
If we do not override the equals method, by default the equals method present in the Object's class will be called.
The implementation of equals present in the root class Object, only verifies if the references are the same. It does not verify if the contents are the same. Content verification is the responsibility of the overriding implementation.
There is an incomplete code provided to you in the Editor, follow the comment lines in order to complete the code in the Main class and The functionality of class A is described below:
• class A is a simple class with a private int field named value and a constructor that takes an int parameter to initialize the value field.
• The class also defines an equals() method that overrides the default equals() method inherited from the Object class.
• If the objects are not the same instance, it checks if the otherObject is an instance of class A using the instanceof operator
1. If otherObject is an instance of class A, it casts otherObject to type A and assigns it to a variable otherARef
2. It then compares the value of the current object (this. value) with the value of otherARef .If the values are the same, it means the objects are equal, so it returns true.
Now complete the code in the Editor By following the above statements and Observe the Output.
Note:
• In the main() method, the code prompts the user to input values for a_1, a_2, and a_3, which are used to create three instances of class A. It then calls the equals() method on a1 to compare it with a2 and a3, printing the results.
• In the same way, the code prompts the user to input values for b_1, b_2, and b_3, which are used to create three instances of class B
Sample Test Cases
Explorer
EqualsEx...
Submit
Plots
A a3 = new A(a_3);
System.out.println("a1.equals(a2) : "+a1.equals(a2));
System.out.println("a1.equals(a3) : "+a1.equals(a3));
System.out.print("b1: ");
int b_1= sc.nextInt();
System.out.print("b2: ");
int b_2=sc.nextInt();
System.out.print("b3: ");
int b_3=sc.nextInt();
/*create three instances of class B named
b1,b2,b3 with the user given values and
compare b1 with b2 and b3 using equals() method*/
B b1 = new B(b_1);
B b2 = new B(b_2);
B b3 = new B(b_3);
System.out.println("b1.equals(b2) : "+b1.equals(b2));
System.out.println("b1.equals(b3) : "+b1.equals(b3));
}
}
class A {
private int value;
public A(int value) {
this.value = value;
}
public boolean equals(A aa){
return this.value == aa.value;
}
}
class B {
private int value;
public B(int value) {
this.value = value;
}
}
Terminal
Test cases
< Prev
Reset
Submit
Next >
Notes hub

Home Learn Anywhere

42.2.1. Understanding scope

Scope can be defined as a portion or block of code in which a variable is visible.

We apply the word scope to variables/references and not so much to methods as methods are members of classes and interfaces.

Variables and references can be declared in:

1. Blocks - these can be if/else/switch, for/while loops, methods and constructors or named blocks. These are also called **local variables**. For example:

```
public static void main(String[] args) {
    int x = Integer.parseInt(args[0]); // x is visible only inside this main method
    if (x > 1) { //if-block-start
        int y = 2 * x;  // y is visible only inside this if block
        System.out.println("y = " + y);
    } //if-block-end
}
```

2. Method and Constructor parameters. For example:

```
public int sum(int num1, int num2) { //method-block-start
    return num1 + num2; // num1 and num2 are only visible inside this method block
} //method-block-end
```

3. Class - as instance fields

```
class A { //class-block-start
    private String name; //reference name is visible anywhere inside the class block
    public String getName() {
        return name; // example usage 1
    }
    public String toString() {
        return "A [ name = " + name + " ]"; // example usage 2
    }
} //class-block-end
```

In Java, before using a variable/reference we need to first declare it.

Select all the correct statements for the below code:

```
class A {
    public A(int value1) { //statement 1
        this.value1 = value1; //stateme
        value2 = value1 * 2; //statemen
```

- [ ] Statement 5 is in the wrong location, it should have been declared above statement 1.
- [ ] Statement 2 will give a compilation error, because there is no variable called `this` declared.
- [x] Statements 3 and 4 will produce compiler errors saying `value2` cannot be resolved, since `value2` is neither declared locally in the enclosing block nor as a field in the class.
- [ ] Statement 5 will produce a compilation error because `value1` is not initialized.

‹ Prev Reset Submit Next ›

Home Learn Anywhere

Section - 1

Q No.	Q. Type	Status	Marks
1	Compilation Errors	✔	5.0

Hide Answer

Note: This question appeared in the Recruitment/Technical Interviews of De Shaw, Hexaware, Tech Mahindra, Accenture, and MAQ Software, Years 2021 and 2022.

In the vibrant city of Objectville, a developer named Taylor is keen on enhancing the representation of objects in their system. Taylor envisions a class named **CustomObject** that goes beyond the default string representation provided by the **Object** class. To achieve this, Taylor plans to override the **toString()** method in the **CustomObject** class, tailoring it to display meaningful information about the object's attributes.

Can you assist Taylor in implementing the **CustomObject** class with a customized **toString()** method?

Execution Results — Show Results

Submitted Source Code :

```
//File Name: q22058/Main.java
//================================
package q22058;
import java.util.*;
public class Main{
	public static void main(String[] args){
		Scanner st=new Scanner(System.in);
		String a=st.nextLine();
		int b=st.nextInt();
		System.out.println("CustomObject{attribute1='"+a+"', attribute2="+b+"}");
	}
}
```

```
// public class Main {
```

2 Comp… .5 View Answer

29, 2024, 23:50

Home Learn Anywhere

43.1.1. Understanding Interfaces

35/10

As the meaning of the word `interface` suggests, it is the point where two systems or subjects meet to interact.

Interface holds the same meaning in object oriented programming languages too. An `interface` defines a contract using which two classes/programs/systems can interact with each other.

In Java, `interface` is like a `class`, it is a reference type. It can have constants (they are not called fields), method signatures and nested members (we will learn more about nested members later).

The methods declared in interfaces should not contain the method body.
[Note: In **Java 8** and later versions, interfaces can contain default and static methods which can contain method body. We will learn more about them in later sections.]
Only the method signature should be present with a semicolon as terminator. For example:

```
interface Person {
    public static final int RETIREMENT_AGE = 60;  // example of a constant
    void setName(String name);  // only method signature without method body
    String getName();
    void setAge(int age);
    int getAge();
}
```

All methods declared in an interface are by default public and hence, we did not use the keyword `public` in the method signatures.

Interfaces cannot be instantiated. Meaning, Java compiler will throw out an error for a statement like : `Person p = new Person();` .

We can instantiate classes that implement the interface. Meaning, we can instantiate classes that provide the implementation for all the methods declared in the interface. For example:

```
public class Teacher implements Person {
    private String name;
    private int age;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
```

- [ ] Statement 1 will result in compilation error, since class Teacher cannot implement two interfaces at the same time.
- [x] Statement 2 **will result** in a compilation error because we cannot instantiate an interface.
- [ ] Statement 3 **will not result** in a compilation error because we are trying to instantiate Person and assign to Citizen.
- [x] Statements 4 and 5 do not result in compilation errors.

Prev Reset Submit Next

Total Questions	Attempted Questions	Correct Questions	Incorrect Questions		
3	2	2	0	1	0

List of Sections

Section - 1

Marks per question : 5.0 **Marks Scored : 10.0**

Q No.	Q. Type	Status	Marks	
1	Compilation Errors	⚠	0.0	Hide Answer

Imagine you are building an online shopping system that allows customers to make payments using different methods. To achieve this, you decide to create an interface **PaymentMethod** that defines the standard methods for processing payments and displaying receipts. One of the payment methods you want to support is credit card payments.

Challenge Details:

PaymentMethod Interface:

- Create an interface named **PaymentMethod** with the following methods:
- **processPayment():** A method that prompts the user for payment details
- **displayReceipt():** A method that displays a receipt for the payment.

CreditCardPayment Class:

- Implement the **PaymentMethod** interface in a class named **CreditCardPayment**.The **CreditCardPayment** class should have private attributes for storing credit card number (String), expiration date (String), and payment amount (double). Implement the **processPayment()** method to take user input for credit card details (card number, expiration date, and payment amount).Implement the **displayReceipt()** method to display a receipt with the credit card number, expiration date, and the amount paid.

Note:

- The main class has been provided to you in the editor.
- Refer to the displayed test cases for the input and output format.

Execution Results — Show Results

Submitted Source Code :

No Submissions found

None of the participants of this test answered this question correctly. For the solution, please consult your faculty.

2	Compilation Errors	✔	5.0	View Answer
3	Compilation Errors	✔	5.0	View Answer

2	Compilation Errors	✔	5.0	Hide Answer

You are working on a utility to transform numbers. Create an interface named **NumberTransformer** with a method **transform()** responsible for transforming and displaying the given number.

- Implement classes **SquareTransformer** and **CubeTransformer** that implement the **NumberTransformer** interface.
- **SquareTransformer** should transform and display the square of the input number, and **CubeTransformer** should transform and display the cube of the input number.

Note: The main class has been provided to you in the editor.

Execution Results

Show Results

Submitted Source Code :

```
//File Name: q23765/NumberTransform.java
//================================
// package q23765;
// import java.util.Scanner;
// interface NumberTransformer {



// public class NumberTransform {
//     public static void main(String[] args) {
//         Scanner scanner = new Scanner(System.in);

//         System.out.print("Enter a number: ");
//         int inputNumber = scanner.nextInt();

//         NumberTransformer squareTransformer = new SquareTransformer();
//         squareTransformer.transform(inputNumber);

//         NumberTransformer cubeTransformer = new CubeTransformer();
//         cubeTransformer.transform(inputNumber);

//         scanner.close();
//     }
// }
```

Feb 29, 2024, 23:52

3	Compilation Errors	✔	5.0	View Answer

2	Compilation Errors	✔	5.0	Hide Answer

You are working on a utility to transform numbers. Create an interface named **NumberTransformer** with a method **transform()** responsible for transforming and displaying the given number.

- Implement classes **SquareTransformer** and **CubeTransformer** that implement the **NumberTransformer** interface.
- **SquareTransformer** should transform and display the square of the input number, and **CubeTransformer** should transform and display the cube of the input number.

Note: The main class has been provided to you in the editor.

Execution Results — Show Results

Submitted Source Code :

```
//      int inputNumber = scanner.nextInt();

//      NumberTransformer squareTransformer = new SquareTransformer();
//      squareTransformer.transform(inputNumber);

//      NumberTransformer cubeTransformer = new CubeTransformer();
//      cubeTransformer.transform(inputNumber);

//      scanner.close();
//  }
// }

package q23765;
import java.util.Scanner;
public class NumberTransform{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("Enter a number: ");
        int a=st.nextInt();
        System.out.println("Square: "+a*a);
        System.out.println("Cube: "+a*a*a);
    }
}
```

3	Compilation Errors	✔	5.0	View Answer

3	Compilation Errors	✔	5.0	Hide Answer

You are developing a graphics application and need to represent various shapes. Design an interface named **Shape** with methods **calculateArea()** and **calculatePerimeter()** of type double. Implement a class named **Circle** that implements the Shape interface. In the **Circle** class, declare an attribute **radius** of type double and a constructor and implement the methods to calculate and display the area and perimeter of the circle.

Note:

- The main class has been provided to you in the editor.
- Refer to the displayed test cases for the input and output format.

Execution Results — Show Results

Submitted Source Code :

```
//File Name: q24927/AreaCalc.java
//==================================
package q24927;
import java.util.Scanner;
// interface Shape {

      // write your code here..

// public class AreaCalc {
//    public static void main(String[] args) {
//       Scanner scanner = new Scanner(System.in);

//       System.out.print("Radius: ");
//       double radius = scanner.nextDouble();

//       // Create an instance of the Circle class
//       Circle circle = new Circle(radius);

//       // Display area and perimeter of the circle
//    System.out.printf("Area: %.2f%n",circle.calculateArea());
//       System.out.printf("Perimeter: %.2f%n",circle.calculatePerimeter());

//       scanner.close();
//    }
// }
```

3	Compilation Errors	✔	5.0	Hide Answer

You are developing a graphics application and need to represent various shapes. Design an interface named **Shape** with methods **calculateArea()** and **calculatePerimeter()** of type double. Implement a class named **Circle** that implements the Shape interface. In the **Circle** class, declare an attribute **radius** of type double and a constructor and implement the methods to calculate and display the area and perimeter of the circle.

Note:

- The main class has been provided to you in the editor.
- Refer to the displayed test cases for the input and output format.

Execution Results — Show Results

Submitted Source Code :

```
//      // Create an instance of the Circle class
//      Circle circle = new Circle(radius);

//      // Display area and perimeter of the circle
//    System.out.printf("Area: %.2f%n",circle.calculateArea());
//      System.out.printf("Perimeter: %.2f%n",circle.calculatePerimeter());

//      scanner.close();
//  }
// }
import java.text.DecimalFormat;
public class AreaCalc{
    public static void main(String[] args){
        Scanner st=new Scanner(System.in);
        System.out.print("Radius: ");
        DecimalFormat format=new DecimalFormat("##.00");
        Double a=st.nextDouble();
        System.out.println("Area: "+format.format(Math.PI*a*a));
        System.out.println("Perimeter: "+format.format(Math.PI*2*a));



    }
}
```

44.1.1. Understanding the usage of interfaces

16:50

An interface is primarily used to create a type.

In Java, a class is also a type which has behaviour attached to it.

However, in large object-oriented systems where multiple classes interact with each other an interface is the preferred means of creating a type. Classes are still needed which implement the type, however the publicly visible contract (methods) is published by the interface and the implementation is provided in the implementing classes.

By doing this we achieve what is called **loose-coupling**, where a class does not depend on the actual implementation (of another class) but rather depends on the contract (i.e. the methods) published through the interface. This ensures that the classes which rely on the interface, need not change whenever the underlying implementation in the implementing classes changes.

See and understand the below code to know how the class InterfaceDemo need not know the difference in the implementation present in classes A and B, when it uses the interface.

```
public class InterfaceDemo {
    public static void main(String[] args) {
        Greeting g1 = new A();
        Greeting g2 = new B();
        System.out.println(g1.getGreetings("Thor"));
        System.out.println(g2.getGreetings("Thor"));
    }
}
interface Greeting {
    String getGreetings(String name);
}
class A implements Greeting {
    public String getGreetings(String name) {
        return "Hi " + name;
    }
}
class B implements Greeting {
    public String getGreetings(String name) {
        return "Hola " + name;
```

Sample Test Cases

Explorer

Interface...

Submit

```
1  import java.util.*;
2
3  interface Printable {
4      void print();
5  }
6
7  // Implement the interface in a class
8  class Document implements Printable {
9      public void print() {
10         Scanner sc = new Scanner(System.in);
11         System.out.print("Enter the first value: ");
12         int m = sc.nextInt();
13         System.out.print("Enter the second value: ");
14         int n = sc.nextInt();
15         System.out.println("You entered: "+m+" and "+n);
16     }
17 }
18
19
20 public class InterfaceDemo {
21     public static void main(String[] args) {
22         // Create an instance of the class implementing the
       interface
23         Printable document = new Document();
24
25         // Call the method defined in the interface
26         document.print();
27     }
28 }
```

Terminal Test cases

Plots

< Prev Reset Submit Next >

Home Learn Anywhere

44.1.2. Write a Java program to Implement Interface

Write a Java program that implements an **interface**.

Create an interface called `Car` with two abstract methods `String getName()` and `int getMaxSpeed()`. Also declare one **default** method `void applyBreak()` which has the code snippet

```
System.out.println("Applying break on " + getName());
```

In the same interface include a **static** method `Car getFastestCar(Car car1, Car car2)`, which returns **car1** if the **maxSpeed** of **car1** is greater than or equal to that of **car2**, else should return **car2**.

Create a class called `BMW` which implements the interface `Car` and provides the implementation for the abstract methods **getName()** and **getMaxSpeed()** (make sure to declare the appropriate fields to store **name** and **maxSpeed** and also the constructor to initialize them).

Similarly, create a class called `Audi` which implements the interface `Car` and provides the implementation for the abstract methods **getName()** and **getMaxSpeed()** (make sure to declare the appropriate fields to store **name** and **maxSpeed** and also the constructor to initialize them).

Create a **public** class called `MainApp` with the **main()** method.
Take the input from the command line arguments. Create objects for the classes `BMW` and `Audi` then print the fastest car.

Note:
Java 8 introduced a new feature called `default` methods or `defender` methods, which allow developers to add new methods to the interfaces without breaking the existing implementation of these interface. These **default** methods can also be overridden in the implementing classes or made abstract in the extending interfaces. If they are not overridden, their implementation will be shared by all the implementing classes or sub interfaces.

Sample Test Cases

MainApp....

Submit

```
1  package q11284;
2  interface Car {
3      String getName();
4      int getMaxSpeed();
5      default void applyBreak(){
6          System.out.println("Applying break on "+getName());
7      }
8      static Car getFastestCar(Car car1, Car car2){
9          if (car1.getMaxSpeed()>= car2.getMaxSpeed()){
10             return car1;
11         }
12         else {
13             return car2;
14         }
15     }
16 }
17 class BMW implements Car {
18     String name;
19     int maxSpeed;
20     BMW (String name, int maxSpeed){
21         this.name = name;
22         this.maxSpeed = maxSpeed;
23     }
24     public String getName(){
25         return name;
26     }
27     public int getMaxSpeed(){
28         return maxSpeed;
29     }
30 }
31 class Audi implements Car {
32     String name;
33     int maxSpeed;
34     public Audi (String name, int maxSpeed){
```

Terminal Test cases

Prev Reset Submit Next

Feb 29, 2024, 23:54

Home Learn Anywhere

44.1.2. Write a Java program to implement Interface

16:50

Write a Java program that implements an **interface**.

Create an interface called `Car` with two abstract methods `String getName()` and `int getMaxSpeed()`. Also declare one **default** method `void applyBreak()` which has the code snippet

```
System.out.println("Applying break on " + getName());
```

In the same interface include a **static** method `Car getFastestCar(Car car1, Car car2)`, which returns **car1** if the **maxSpeed** of **car1** is greater than or equal to that of **car2**, else should return **car2**.

Create a class called `BMW` which implements the interface `Car` and provides the implementation for the abstract methods **getName()** and **getMaxSpeed()** (make sure to declare the appropriate fields to store **name** and **maxSpeed** and also the constructor to initialize them).

Similarly, create a class called `Audi` which implements the interface `Car` and provides the implementation for the abstract methods **getName()** and **getMaxSpeed()** (make sure to declare the appropriate fields to store **name** and **maxSpeed** and also the constructor to initialize them).

Create a **public** class called `MainApp` with the **main()** method.
Take the input from the command line arguments. Create objects for the classes `BMW` and `Audi` then print the fastest car.

Note:
Java 8 introduced a new feature called `default` methods or `defender` methods, which allow developers to add new methods to the interfaces without breaking the existing implementation of these interface. These **default** methods can also be overridden in the implementing classes or made abstract in the extending interfaces. If they are not overridden, their implementation will be shared by all the implementing classes or sub interfaces.

Sample Test Cases

Explorer

MainApp....

Submit

```
20      BMW (String name, int maxSpeed){
21          this.name = name;
22          this.maxSpeed = maxSpeed;
23      }
24      public String getName(){
25          return name;
26      }
27      public int getMaxSpeed(){
28          return maxSpeed;
29      }
30  }
31  class Audi implements Car {
32      String name;
33      int maxSpeed;
34      public Audi (String name, int maxSpeed){
35          this.name = name;
36          this.maxSpeed = maxSpeed;
37      }
38      public String getName(){
39          return name;
40      }
41      public int getMaxSpeed(){
42          return maxSpeed;
43      }
44  }
45  public class MainApp {
46      public static void main(String s[]) {
47          Audi a = new Audi(s[2],Integer.parseInt(s[3]));
48          BMW b = new BMW(s[0],Integer.parseInt(s[1]));
49          Car v = Car.getFastestCar(b,a);
50          System.out.println("Fastest car is : "+v.getName());
51      }
52  }
53
```

Terminal Test cases

Plots

‹ Prev Reset Submit Next ›

Home Learn Anywhere

45.1.1. Write a Java program to find Areas of different Shapes using abstract class

04:20

Write a Java program to illustrate the **abstract class** concept.

Create an abstract class `CalcArea` and declare the methods **triangleArea(double b, double h)**, **rectangleArea(double l, double b)**, **squareArea(double s)**, **circleArea(double r)**.

Create a class `FindArea` which extends the abstract class `CalcArea` used to find areas of triangle, rectangle, square, circle.

Write a class `Area` with the **main()** method which will receive **two** arguments and convert them to **double** type.

If the input is given as command line arguments to the **main()** as **"1.2","2.7"** then the program should print the output as:

```
Area of triangle : 1.62
Area of rectangle : 3.24
Area of square : 1.44
Area of circle : 22.890600000000006
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

Area.java

Submit

```
1  package q11286;
2  public class Area {
3      public static void main(String args[]) {
4          FindArea area = new FindArea();
5          area.triangleArea(Double.parseDouble(args[0]),
   Double.parseDouble(args[1]));
6          area.rectangleArea(Double.parseDouble(args[0]),
   Double.parseDouble(args[1]));
7          area.squareArea(Double.parseDouble(args[0]));
8          area.circleArea(Double.parseDouble(args[1]));
9      }
10 }
11 //Write all the classes with definitions
12 abstract class calcArea {
13     abstract void triangleArea(double b, double h);
14     abstract void rectangleArea(double l, double b);
15     abstract void squareArea(double s);
16     abstract void circleArea(double r);
17 }
18 class FindArea extends calcArea {
19     void triangleArea(double b, double h)
20     {
21 double area = (b*h)/2;
22 System.out.println("Area of triangle : "+area);}
23 void rectangleArea(double l, double b)
24     {
25 double area = l*b;
26 System.out.println("Area of rectangle : "+area);
27     }
28     void squareArea(double s)
29     {
30     double area = s*s;
31     System.out.println("Area of square : "+area);}
32         void circleArea(double r)
33     {
34     double area = 3.14*r*r;
35 System.out.println("Area of circle : "+area);}
36 }
```

Terminal Test cases

Prev Reset Submit Next

Home Learn Anywhere

45.1.2. Write a Java program to illustrate the abstract class concept

00:43

Write a Java program to illustrate the **abstract class** concept.

Create an abstract class `Shape`, which contains an empty method **numberOfSides()**.

Define three classes named `Trapezoid`, `Triangle` and `Hexagon` extends the class `Shape`, such that each one of the classes contains only the method **numberOfSides()**, that contains the number of sides in the given geometrical figure.

Write a class `AbstractExample` with the **main()** method, declare an object to the class `Shape`, create instances of each class and call numberOfSides() methods of each class.

Sample Input and Output:

```
Number of sides in a trapezoid are 4
Number of sides in a triangle are 3
Number of sides in a hexagon are 6
```

Note: Please don't change the package name.

Sample Test Cases

Explorer

Abstract...

Submit

Plots

```
1  package q11287;
2  //Write the code
3  public class AbstractExample {
4      public static void main(String[] args) {
5          Shape s;
6          s= new Trapezoid();
7          s.numberofsides();
8          s = new Triangle();
9          s.numberofsides();
10         s = new Hexagon();
11         s.numberofsides();
12     }
13 }
14 abstract class Shape{
15     abstract void numberofsides();
16     }
17 class Trapezoid extends Shape{
18     void numberofsides()
19     {
20 System.out.println("Number of sides in a trapezoid are 4");
21     }
22     }
23 class Triangle extends Shape
24     {
25         void numberofsides()
26     {
27 System.out.println("Number of sides in a triangle are 3");
28 }
29     }
30 class Hexagon extends Shape{
31         void numberofsides()
32     {
33 System.out.println("Number of sides in a hexagon are 6");
34     }
35     }
```

Terminal Test cases

‹ Prev Reset Submit Next ›

cBook Air

Home Learn Anywhere

45.1.3. What is an abstract class?

01:25

In Java, a class when declared with abstract keyword becomes an abstract class. For example:

```
public abstract class A {
}
```

As per the dictionary, **abstract** means **not concrete**. In object oriented languages also, when a class is marked with abstract keyword, it indicates that the class is not concrete. **It cannot be instantiated**. Meaning, the Java compiler will give an error, saying abstract class `A` cannot be instantiated, if we write a statement like `A a = new A();`.

Abstract classes like normal (concrete) classes can have fields, constructors and methods.

Abstract classes can also have one or more abstract methods. An abstract method should have the keyword `abstract` in its signature and it should not have a method body. For example:

```
public abstract int sum(int num1, num2);
```

A concrete class is not allowed to have a method declared as `abstract`. Java compiler will throw an compilation error if an attempt is made to declare an abstract method in a concrete class.

In interfaces, the method declarations with signatures and without method bodies are by default marked `abstract`. However, in a abstract class we have to mark a method explicitly with `abstract` keyword.

Abstract classes are present so only to be inherited by other classes. Abstract classes usually contain common code that can be shared by the subclasses.

Observe the given code and declare an abstract class **AbstractGreeting** that implements interface **Greeting** including an abstract string method **getCustomMessage**.

After executing, from the output you will realize that the classes `EnglishGreeting` and `SpanishGreeting` have inherited the implementation of `getStandardMessage` method from the abstract class `AbstractGreeting`.

If the classes `EnglishGreeting` and `SpanishGreeting` were written such that they did not extend from the abstract class and have instead implemented the `Greeting` interface directly, they would have been forced to individually provide the implementation for `getStandardMessage` method, resulting in duplication

Sample Test Cases

Explorer

Abstract...

Submit

```
1  package q35957;
2  public class AbstractDemo {
3      public static void main(String[] args) {
4          Greeting english = new EnglishGreeting();
5          Greeting spanish = new SpanishGreeting();
6
7          System.out.println(english.getStandardMessage("Winston"));
8          System.out.println(english.getCustomMessage("Winston"));
9          System.out.println(spanish.getStandardMessage("Martin"));
10         System.out.println(spanish.getCustomMessage("Martin"));
11     }
12 }
13 interface Greeting {
14     public String getStandardMessage(String name);
15     public String getCustomMessage(String name);
16 }
17 abstract class AbstractGreeting implements Greeting {
18     public String getStandardMessage(String name) {
19         return "Hi " + name;
20     }
21     public abstract String getCustomMessage(String name);
22 }
23 class EnglishGreeting extends AbstractGreeting {
24     public String getCustomMessage(String name) {
25         return "Hello " + name;
26     }
27 }
28 class SpanishGreeting extends AbstractGreeting {
29     public String getCustomMessage(String name) {
30         return "Holla " + name;
31     }
```

Terminal Test cases

‹ Prev Reset Submit Next ›

Mar 1, 2024, 00:24

Home Learn Anywhere

46.1.1. Default methods in interfaces

12:26

Default methods in interfaces:

Before Java 8, interfaces could have only abstract methods. The implementation of these methods has to be provided in a separate class. So, if a new method is to be added to an interface, its implementation code must be provided in the class implementing the same interface. To overcome this issue, Java 8 has introduced the concept of default methods, allowing the interfaces to have methods with implementation without affecting the classes that implement the interface.

Let's go through a clear example to illustrate the use of default methods in Java interfaces.
Suppose you have an interface called `Shape` representing different geometric shapes, and you want to add a method to calculate the area. However, adding this method to the interface might break existing implementations. Default methods come to the rescue in this scenario.

```
public interface Shape {
  // Abstract method (to be implemented by concrete classes)
  void draw();

  // Default method to calculate area
  default double calculateArea() {
    return 0.0; // Default implementation for simplicity
  }
}
```

Here, `Shape` has an abstract method `draw()` that concrete classes must implement. Additionally, there's a default method `calculateArea()` with a default implementation returning 0.0. Concrete classes can choose to override this method.
Now, let's create a class `Circle` that implements the `Shape` interface:

```
public class Circle implements Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  @Override
  public void draw() {
    System.out.println("Drawing a circle");
```

Sample Test Cases

Explorer

Animal.java Dog.java

```
1  public interface Animal {
2    ...
3    void makeSound();
4    default String eatFood(){
5      return "eating generic food";
6    }
7  }
8
```

Terminal Test cases

‹ Prev Reset Submit Next ›

Mar 1, 2024, 00:25

Home Learn Anywhere

46.1.1. Default methods in interfaces

12:26

Default methods in interfaces:

Before Java 8, interfaces could have only abstract methods. The implementation of these methods has to be provided in a separate class. So, if a new method is to be added to an interface, its implementation code must be provided in the class implementing the same interface. To overcome this issue, Java 8 has introduced the concept of default methods, allowing the interfaces to have methods with implementation without affecting the classes that implement the interface.

Let's go through a clear example to illustrate the use of default methods in Java interfaces.
Suppose you have an interface called `Shape` representing different geometric shapes, and you want to add a method to calculate the area. However, adding this method to the interface might break existing implementations. Default methods come to the rescue in this scenario.

```
public interface Shape {
  // Abstract method (to be implemented by concrete classes)
  void draw();

  // Default method to calculate area
  default double calculateArea() {
    return 0.0; // Default implementation for simplicity
  }
}
```

Here, `Shape` has an abstract method `draw()` that concrete classes must implement. Additionally, there's a default method `calculateArea()` with a default implementation returning 0.0. Concrete classes can choose to override this method.
Now, let's create a class `Circle` that implements the `Shape` interface:

```
public class Circle implements Shape {
  private double radius;

  public Circle(double radius) {
    this.radius = radius;
  }

  @Override
  public void draw() {
    System.out.println("Drawing a circle");
```

Sample Test Cases

Explorer

Animal.java Dog.java

Submit

```
1  public class Dog implements Animal{
2
3      public void makeSound(){
4          System.out.println("Dog barks: Woof! Woof!");
5      }
6      public String eatFood(){
7          return "Dog is eating bones";
8      }
9
10 }
11
12
```

Terminal Test cases

‹ Prev Reset Submit Next ›

cBook Air

0:25