**LAB 4.1**

*Write an abstract class; Try following variations*

* *create an abstract class without keeping any method abstract*
* *Make some methods abstract, but do not use abstract word (other modifier) in class’s signature*
* *Try to create object of the abstract class, and call abstract method*

**Steps:**

* Create Java classes MyAbstractClass and try all above mentioned variations

**Part 1 -**

/\*\*

\* Creating an abstract class without having any abstract method

\*/

**abstract** **class** ForAbstractPractice{

**public** **void** nonAbstractDisplay(){

System.*out*.println("non abstract method of an abstract class");

}

}

**OUTPUT**

Abstract is a valid modifier even if class do not have any abstract method

**Part 2 -**

/\*\*

\* Writting and abstract method; without using abstract modifier

\*/

**abstract** **class** ForAbstractPractice{

**public** **void** nonAbstractDisplay(){

System.*out*.println("non abstract method of an abstract class");

}

**public** **void** abstractDisplay(); // COMPILATION ERROR

}

**OUTPUT**

Abstract modifier is must to write an abstract method

**Part 3 -**

**public** **class** AbstractClassPractice {

**public** **static** **void** main(String[] args) {

// trying to create object of an abstract class

ForAbstractPractice practice = **new** ForAbstractPractice();

// COMPILATION ERROR

}

}

/\*\*

\* Creating an abstract class without having any abstract method

\*/

**abstract** **class** ForAbstractPractice{

**public** **void** nonAbstractDisplay(){

System.*out*.println("non abstract method of an abstract class");

}

}

**OUTPUT**

Abstract class cannot be instantiated

**Learning:**

* Abstract is mandatory modifier for method and class to be abstract
* Object of abstract class cannot be instantiated

**LAB 4.2**

*Write an interface(IMyMath) , then write a class MyMath, which should implements this interface. In the MainClass (class having main method) create reference (instance) of the interface.*

**Steps:**

* Create interface IMyMath having add, substract, multiply and Divide method signatures
* Create Java classes MyMath which will implements IMyMath (try to write an method with body, as we write in any class, in the interface and check what happens)
* Provide implementation to all the methods (try not to provide implementation to one or more methods and check what happens)

**IMath Interface**

/\*\*

\* This Interface has definition of various mathematical operation

\*/

**public** **interface** IMath {

**public** **double** add(**double** firstNumber, **double** secondNumber);

**public** **double** substract(**double** firstNumber, **double** secondNumber);

**public** **double** multiple(**double** firstNumber, **double** secondNumber);

**public** **double** divide(**double** firstNumber, **double** secondNumber);

**public** **double** modulus(**double** firstNumber, **double** secondNumber);

}

**MyMath class – implementing IMath Interface**

/\*\*

\* This class implements IMath Interface - To performs various mathematical

\* operations like addition, substract, multiple, divide etc

\* This class will provide body to all methods of IMath

\*/

**public** **class** MyMath **implements** IMath{

**public** **double** add(**double** firstNumber, **double** secondNumber){

**return** firstNumber + secondNumber;

}

**public** **double** substract(**double** firstNumber, **double** secondNumber){

**return** firstNumber - secondNumber;

}

**public** **double** multiple(**double** firstNumber, **double** secondNumber){

**return** firstNumber \* secondNumber;

}

**public** **double** divide(**double** firstNumber, **double** secondNumber){

**return** firstNumber / secondNumber;

}

**public** **double** modulus(**double** firstNumber, **double** secondNumber){

**return** firstNumber % secondNumber;

}

}

**Another class – which will create reference object of IMath using MyMath Class**

/\*\*

\* This is main class – having p.s.v.main method

\* This class will create a ref object of IMath using MyMath class

\*/

**public** **class** MainClassForMySubMath {

**public** **static** **void** main(String[] args) {

IMath math = **new** MyMath();

**double** mathResult = math.add(12, 13);

System.*out*.println("Add of 12 & 13 in MyMath class "

+ mathResult);

}

}

**OUTPUT -** Add of 12 & 13 in MyMath class 25.0

**Learning:**

* What is interface, how to create an interface
* All method of an interface must be abstract
* It is compulsory to provide implementation (body) to all the method of interface being implementing by a class.
* Interface reference instance can be created with the help of implementing class

**LAB 4.3**

*Now write some additional methods in MyMath class (which is implementing IMyMath), i.e. apart from method in interface add some more methods in class. Now in MainClass (main method) create reference instance of interface and instance of MyMath class. Check which instance have which all methods.*

*Similarly, write one more class implanting IMyMath, so now one interface is being implementing by two different classes; write different implementation (body) in both the classes. Create reference instance of using both the classes; now check which instance is accessing which method.*

**Not applicable here - as code snippet is same as Lab 4.3 with some variations (mentioned below)**

// IMath object reference from MyMath Class

IMath math = **new** MyMath();

// MyMath object from MyMath Class

MyMath myMath = **new** MyMath();

**OUTPUT**

Math Object will have only those method and properties of MyMath Class which are defined in IMath interface

While myMath Object will have all the method and properties of MyMath class

**Learning:**

* Even if a class is implementing an interface; the reference instance of the interface will have access of methods in interface only and not to extra/other methods of the class.
* The implementation/body of method will be considered for that class, whose constructor is being used to create reference instance of interface.

**LAB 4.4**

*Write an interface which extends another interface (this exercise is to check inheritance in interfaces)*

*Now write a class implementing the sub-interface and check weather this class have to provide implantation to all the methods of sub and super interface.*

**Steps:**

* Write an interface (say ISplMath) which will extends IMath Interface
* Write a class (say SplMath) which implements ISplMath

/\*\*

\* This Interface has definition of various mathematical operation

\*/

**public** **interface** IMath {

**public** **double** add(**double** firstNumber, **double** secondNumber);

**public** **double** substract(**double** firstNumber, **double** secondNumber);

}

/\*\*

\* This interface extends IMath interface

\*/

**public** **interface** ISplMath **extends** IMath {

**public** **double** sqrt(**double** number);

}

/\*\*

\* This class implements ISplMath interface; While ISplMath extends IMath

\* interface. Thus this class will have to provide body of all the

\* methods of both ISplMath and IMath

\*/

**public** **class** SplMath **implements** ISplMath {

@Override

**public** **double** add(**double** firstNumber, **double** secondNumber) {

// **TODO** Auto-generated method stub

**return** 0;

}

@Override

**public** **double** substract(**double** firstNumber, **double** secondNumber) {

// **TODO** Auto-generated method stub

**return** 0;

}

@Override

**public** **double** sqrt(**double** number) {

// **TODO** Auto-generated method stub

**return** 0;

}

}

**Learning:**

* An Interface can extends another interface

**LAB 4.5**

*Write two or more interface and write a class implementing all interfaces (one or more than one interfaces)*

*Try to extends more than one class*

**Steps:**

* Write 2-3 interfaces
* Write a class which implements all of them

**public** **class** MyMath **implements** IMath, ISplMath{

// all methods of both the interface

}

**Learning:**

* A class can implements many interfaces but can extends only one class.

**LAB 4.6**

*Create two packages and write few classes and subclasses in packages; now check the various access conditions and find out behavior of various access modifier of method*

**Steps:**

* Create two packages (package 1 and package2); now write one class (Cls1) in package1 and another class (Cls2) in package2;
* Write some methods in Cls2; keep different access modifier for (public, provide, default, protected) for different method of Cls2.
* Create instance ( object) of Cls2 in Cls1 and try to access/call the methods.
* Now Create sub class of Cls2 in package1 and another subclass in package2.
* Create instance of sub classes in Cls1 and try to access method of super class through the instances

**Learning:**

* What is package, standard nomenclature for packages
* Benefit of packages
* Easy to manage
* Same class name can be in different package
* easy to achieve security/visibility conditions
* First line of any class will be package, then import
* Below is matrix of access modifier restriction/visibility

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method access modifier | Within same package | In diff package | Sub class in same package | Sub class in different package |
| Public | Y | Y | Y | Y |
| Protected | Y | N | Y | Y |
| Default | Y | N | Y | N |
| Private | N | N | N | N |