Abstract Methods in Java with Examples

Difficulty Level : Basic Last Updated : 26 Dec, 2020

Sometimes, we require just method declaration in super-classes. This can be achieve by specifying the **abstract** type modifier. These methods are sometimes referred to as *subclasser responsibility* because they have no implementation specified in the super-class. Thus, a subclass must override them to provide method definition. To declare an abstract method, use this general form:

```
abstract type method-name(parameter-list);
```

As you can see, no method body is present. Any concrete class(i.e. class without abstract keyword) that extends an abstract class must override all the abstract methods of the class.

Important rules for abstract methods:

- Any class that contains one or more abstract methods must also be declared abstract
- The following are various **illegal combinations** of other modifiers for methods with respect to *abstract* modifier:
  1. final
  2. abstract native
  3. abstract synchronized
  4. abstract static
  5. abstract private
  6. abstract strictfp

Consider the following Java program, that illustrate the use of *abstract* keyword with classes and methods.

```
// A java program to demonstrate
```

```
// use of abstract keyword.

// abstract class
abstract class A {
    // abstract method
    // it has no body
    abstract void m1();

    // concrete methods are still
    // allowed in abstract classes
    void m2()
    {
        System.out.println("This is "
                           + "a concrete method.");
    }
}

// concrete class B
class B extends A {
    // class B must override m1() method
    // otherwise, compile-time
    // exception will be thrown
    void m1()
    {
        System.out.println("B's "
                           + "implementation of m1.");
    }
}

// Driver class
public class AbstractDemo {
    public static void main(String args[])
    {
        B b = new B();
        b.m1();
        b.m2();
    }
}
```

Output:

```
B's implementation of m1.
This is a concrete method.
```

Note: Although abstract classes cannot be used to instantiate objects, they can be used to create object references, because Java's approach to run-time polymorphism is implemented through the use of super-class references. Thus, it must be possible to

create a reference to an abstract class so that it can be used to point to a subclass object.