Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java : Abstract Class/5b1e5ef97becc0459deae3a7

TUTORIAL

# Java : Abstract Class

## Topics

1.2 Abstract Class Implementation

As you move up the inheritance hierarchy, classes become more general and probably more abstract. At some point, the ancestor class becomes so general that you think of it more as a basis for other classes than as a class with specific instances you want to use. Consider, for example, an extension of our Employee class hierarchy. An employee is a person, and so is a student. Let us extend our class hierarchy to include classes Person and Student. There are some attributes that make sense for every person, such as the name. Both students and employees have names. Now let's add another method, getDescription, whose purpose is to return a brief description of the person, such as an employee with a salary of $50,000.00 a student majoring in computer science. It is easy to implement this method for the Employee and Student classes. But what information can you provide in the Person class? The Person class knows nothing about the person except the name. Of course, you could implement Person.getDescription() to return an empty string. But there is a better way. If you use the abstract keyword, you do not need to implement the method at all.

```
public abstract String getDescription();
// no implementation required
```

In other words, You can require that certain methods be overridden by subclasses by specifying the abstract type modifier. These methods are sometimes referred to as subclasser responsibility because they have no implementation specified in the superclass. Thus, a subclass must override them—it cannot simply use the

version defined in the superclass. To declare an abstract method, use this general form:

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

As you can see, no method body is present.

For added clarity, a class with one or more abstract methods must itself be declared abstract.

```
abstract class Person  { . . .
public abstract String getDescription();
}
```

In addition to abstract methods, abstract classes can have fields and concrete methods. For example, the Person class stores the name of the person and has a concrete method that returns it.

```
abstract class Person
{
        public Person(String n)
        {
                name = n;
        }
        public abstract String getDescription();
        public String getName()
        {
                return name;
        }
        private String name;
}
```

Abstract methods act as placeholders for methods that are implemented in the subclasses. When you extend an abstract class, you have two choices. You can leave some or all of the abstract methods undefined. Then you must tag the subclass as abstract as well. Or you can define all methods. Then the subclass is no longer abstract. Here is a simple example of a class with an abstract method, followed by a class which implements that method:

## Abstract Class Implementation

Java

```
// A Simple demonstration of abstract.
abstract class A {
  abstract void callme();
  // concrete methods are still allowed in abstract classes
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}

class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  }
}

class Main {
  public static void main(String args[]) {
    B b = new B();
    b.callme();
    b.callmetoo();
  }
}

```

Notice that no objects of class A are declared in the program. As mentioned, it is not possible to instantiate an abstract class. One other point: class A implements a concrete method called callmetoo( ). This is perfectly acceptable. Abstract classes can include as much implementation as they see fit. 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 superclass 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.