Java Abstract Class and Abstract Methods

In this tutorial, we will learn about Java abstract classes and methods with the help of examples. We will also learn about abstraction in Java.

Java Abstract Class

The abstract class in Java cannot be instantiated (we cannot create objects of abstract classes). We use the `abstract` keyword to declare an abstract class. For example,

```
// create an abstract class
abstract class Language {
  // fields and methods
}
...

// try to create an object Language
// throws an error
Language obj = new Language();
```

An abstract class can have both the regular methods and abstract methods. For example,

```
abstract class Language {

  // abstract method
  abstract void method1();

  // regular method
  void method2() {
    System.out.println("This is regular method");
  }
}
```

To know about the non-abstract methods, Java methods . Here, we will learn about abstract methods.

Java Abstract Method

A method that doesn't have its body is known as an abstract method. We use the same `abstract` keyword to create abstract methods. For example,

```
abstract void display();
```

Here, `display()` is an abstract method. The body of `display()` is replaced by `;`.

If a class contains an abstract method, then the class should be declared abstract. Otherwise, it will generate an error. For example,

```
// error
// class should be abstract
class Language {

  // abstract method
  abstract void method1();
}
```

Example: Java Abstract Class and Method

Though abstract classes cannot be instantiated, we can create subclasses from it. We can then access members of the abstract class using the object of the subclass. For example,

```
abstract class Language {

  // method of abstract class
  public void display() {
    System.out.println("This is Java Programming");
  }
}

class Main extends Language {

  public static void main(String[] args) {

    // create an object of Main
    Main obj = new Main();

    // access method of abstract class
    // using object of Main class
    obj.display();
  }
}
```

Output

```
This is Java programming
```

In the above example, we have created an abstract class named `Language`. The class contains a regular method `display()`.

We have created the Main class that inherits the abstract class. Notice the statement,

```
obj.display();
```

Here, `obj` is the object of the child class `Main`. We are calling the method of the abstract class using the object `obj`.

Implementing Abstract Methods

If the abstract class includes any abstract method, then all the child classes inherited from the abstract superclass must provide the implementation of the abstract method. For example,

```
abstract class Animal {
  abstract void makeSound();

  public void eat() {
    System.out.println("I can eat.");
  }
}

class Dog extends Animal {

  // provide implementation of abstract method
  public void makeSound() {
    System.out.println("Bark bark");
  }
}

class Main {
  public static void main(String[] args) {

    // create an object of Dog class
    Dog d1 = new Dog();

    d1.makeSound();
    d1.eat();
  }
}
```

Output

```
Bark bark
I can eat.
```

In the above example, we have created an abstract class `Animal`. The class contains an abstract method `makeSound()` and a non-abstract method `eat()`.

We have inherited a subclass `Dog` from the superclass `Animal`. Here, the subclass `Dog` provides the implementation for the abstract method `makeSound()`.

We then used the object `d1` of the `Dog` class to call methods `makeSound()` and `eat()`.

> **Note**: If the `Dog` class doesn't provide the implementation of the abstract method `makeSound()`, `Dog` should also be declared as abstract. This is because the subclass `Dog` inherits `makeSound()` from `Animal`.

Accesses Constructor of Abstract Classes

An abstract class can have constructors like the regular class. And, we can access the constructor of an abstract class from the subclass using the `super` keyword. For example,

```
abstract class Animal {
   Animal() {
      ….
   }
}

class Dog extends Animal {
   Dog() {
      super();
      ...
   }
}
```

Here, we have used the `super()` inside the constructor of `Dog` to access the constructor of the `Animal`.

Note that the `super` should always be the first statement of the subclass constructor. Visit Java super keyword to learn more.

Java Abstraction

The major use of abstract classes and methods is to achieve abstraction in Java.

Abstraction is an important concept of object-oriented programming that allows us to hide unnecessary details and only show the needed information.

This allows us to manage complexity by omitting or hiding details with a simpler, higher-level idea.

A practical example of abstraction can be motorbike brakes. We know what brake does. When we apply the brake, the motorbike will stop. However, the working of the brake is kept hidden from us.

The major advantage of hiding the working of the brake is that now the manufacturer can implement brake differently for different motorbikes, however, what brake does will be the same.

Let's take an example that helps us to better understand Java abstraction.

Example 3: Java Abstraction

```
abstract class MotorBike {
  abstract void brake();
}

class SportsBike extends MotorBike {

  // implementation of abstract method
  public void brake() {
    System.out.println("SportsBike Brake");
  }
}

class MountainBike extends MotorBike {

  // implementation of abstract method
  public void brake() {
    System.out.println("MountainBike Brake");
  }
}

class Main {
  public static void main(String[] args) {
    MountainBike m1 = new MountainBike();
    m1.brake();
    SportsBike s1 = new SportsBike();
    s1.brake();
  }
}
```

Output:

```
MountainBike Brake
SportsBike Brake
```

In the above example, we have created an abstract super class `MotorBike`. The superclass `MotorBike` has an abstract method `brake()`.

The `brake()` method cannot be implemented inside `MotorBike`. It is because every bike has different implementation of brakes. So, all the subclasses of `MotorBike` would have different implementation of `brake()`.

So, the implementation of `brake()` in `MotorBike` is kept hidden.

Here, `MountainBike` makes its own implementation of `brake()` and `SportsBike` makes its own implementation of `brake()`.

> **Note**: We can also use interfaces to achieve abstraction in Java. To learn more, visit Java Interface.

Key Points to Remember

- We use the `abstract` keyword to create abstract classes and methods.
- An abstract method doesn't have any implementation (method body).
- A class containing abstract methods should also be abstract.
- We cannot create objects of an abstract class.
- To implement features of an abstract class, we inherit subclasses from it and create objects of the subclass.
- A subclass must override all abstract methods of an abstract class. However, if the subclass is declared abstract, it's not mandatory to override abstract methods.
- We can access the static attributes and methods of an abstract class using the reference of the abstract class. For example,

```
Animal.staticMethod();
```