ABSTRACTION IN JAVA

Velocity

AUGUST 5, 2022

VELOCITY

Pune

Abstraction in java

Abstraction-

It is the process of hiding the certain details and showing the important information to the end user called as “Abstraction”. Or

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Realtime Examples of Abstraction in Java

1. Let’s first take ATM machine as a real-time example. We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc. in our daily life.

But we don’t know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.

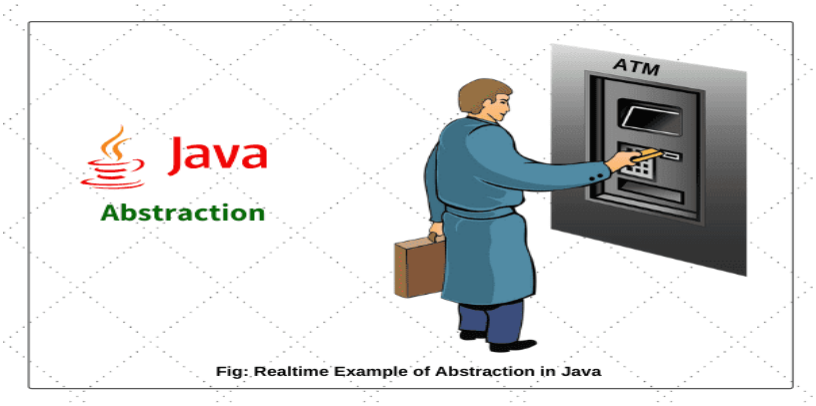

Fig: Realtime Example of Abstraction in Java

2. When you need to send SMS from your mobile, you only type the text and send the message. But you don’t know the internal processing of the message delivery.

How to achieve Abstraction in Java?

There are two ways to achieve or implement abstraction in java program. They are as follows:

Abstract class (0 to 100%)

Interface (100%)

Abstract Class in Java

An abstract class in Java is a class, which is declared with an abstract keyword. It has following features.

- Abstract class have constructor
- It contains abstract methods or concrete methods or empty class or combination of both methods.
- To use abstract method of class, we should extend the abstract class and use those methods.
- If we don't want to implement or override that method, make those methods as abstract.
- If any method is abstract in a class, then that class must be declared as abstract
- We cannot create the object of abstract class.
- Abstract class is not a pure abstraction in java.

Note- Multiple inheritances are not allowed in abstract class but allowed in interfaces

Note: Abstract is a non-access modifier in java which is applicable for classes, interfaces, methods, and inner classes. It represents an incomplete class that depends on subclasses for its implementation. Creating subclass is compulsory for abstract class.

Note: An abstract concept is not applicable to variables.

When to use Abstract class in Java?

An abstract class can be used when we need to share the same method to all non-abstract subclasses with their own specific implementations.

Syntax for abstract class

```
<Access specifier>abstract class <Class_name> {

}
```

Abstract Method in Java

A method that is declared with abstract modifier in an abstract class and has no implementation (means no body) is called abstract method in java. It does not contain any body.

Syntax for abstract method

<Access specifiers> <abstract><Returntype>method_name (Arguments);

There are the following uses of abstract method in Java. They are as follows:

1. An abstract method can be used when the same method has to perform different tasks depending on the object calling it.

2. A method can be used as abstract when you need to be overridden in its non-abstract subclasses.

```
package com.abstraction;

public abstract class Test {

	abstract void example(); // abstract method

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

How to implement that methods?

We need to create the class which extends from abstract class as shown in below.

```
package com.abstraction;

public class C extends Test {

	@Override
	void example() {

		System.out.println("this is the example method");

	}

	@Override
	void demo() {

		System.out.println("this is the demo method");

	}

}
```

```
package com.abstraction;

public class TestMain {

     public static void main(String[] args) {

          C c= new C();
          c.demo();
          c.example();

     }
}
```

Note- Suppose in the sub class, I don't want to override the abstract methods then make that subclass as abstract.

Examples

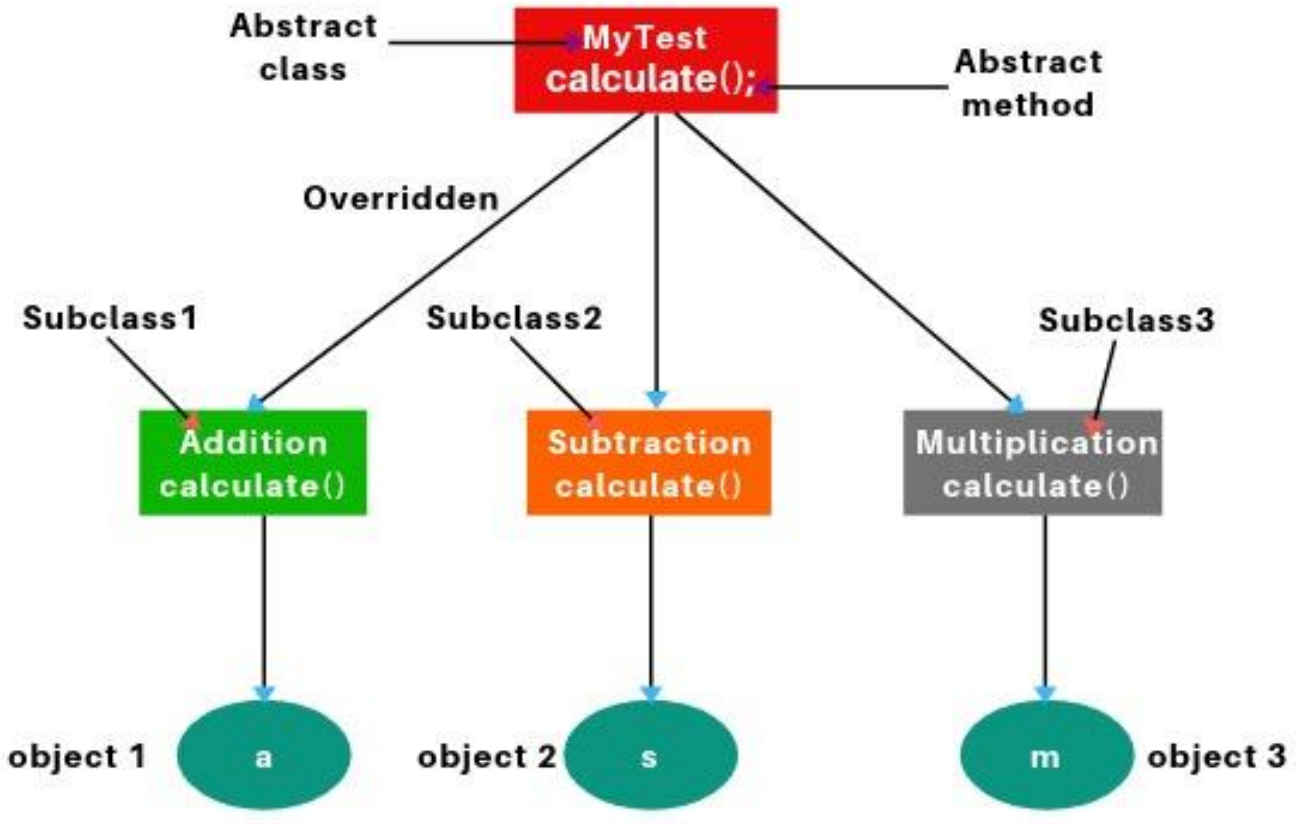

Fig: Abstract class and its subclasses

```java
package com.velocity.abstractclass;

public abstract class MyDemo {

    abstract void calculate(int a, int b);

}
```

```java
public class Addition extends MyDemo {

    @Override
    void calculate(int a, int b) {
        int x = a + b;
        System.out.println("The sum of numbers is >> " + x);

    }

}
```

```java
package com.velocity.abstractclass;

public class Subtraction extends MyDemo {

    @Override
    void calculate(int a, int b) {

        int y = b - a;
        System.out.println(" The subtracted value is >>" + y);

    }

}
```

```java
package com.velocity.abstractclass;

public class Multiplication extends MyDemo {

    @Override
    void calculate(int a, int b) {

        int z = a * b;
        System.out.println("The multiplied value is >>" + z);

    }

}
```

```java
package com.velocity.abstractclass;

public class Test {

    public static void main(String[] args) {

        Addition addition = new Addition();
        Subtraction subtraction = new Subtraction();
        Multiplication multiplication = new Multiplication();

        addition.calculate(10, 5);
        subtraction.calculate(10, 5);
        multiplication.calculate(10, 5);

    }

}
```

Example 2:

```java
package com.velocity.abstractclass;

public abstract class Main {

    int x = 10;

    Main() {
        System.out.println("Abstract class constructor");
    }

    final void m1() {

        System.out.println("This is a final method");
    }

    static void m2() {

        System.out.println("This is a static method");
    }

    void m3() {

        System.out.println("This is a normal method");
    }

    abstract void m4();

}
```

```java
package com.velocity.abstractclass;

public class Impl extends Main {

    @Override
    void m4() {
        System.out.println("This is overriding method from sub class");

    }

    Impl() {

        System.out.println("This is sub class constructor");
    }

}
```

```java
package com.velocity.abstractclass;

public class Testing {

    public static void main(String[] args) {

        Impl obj = new Impl();
        System.out.println(obj.x);
        obj.m1();
        obj.m2();
        obj.m3();
        obj.m4();

    }

}
```

Console × Problems @ Javadoc

<terminated> Testing [Java Application] C:\Users\praveen bhosle\.p2\pool\

```
Abstract class constructor
This is sub class constructor
10
This is a final method
This is a static method
This is a normal method
This is overriding method from sub class
```

In this example program, after object creation, the constructor of non-abstract subclass will be called immediately.

In the first line of constructor, internally super will call the constructor of an abstract class. The control of execution will be immediately transferred to the constructor of abstract class.

Therefore, the first output is “Abstract Class constructor”. After executing abstract class constructor, control of execution again comes back to execute subclass constructor. The second output is “sub-class constructor”.

Why abstract class has constructor even though we cannot create object?

We cannot create an object of abstract class but we can create an object of subclass of abstract class. When we create an object of subclass of an abstract class, it calls the constructor of subclass.

This subclass constructor has super in the first line that calls constructor of an abstract class. Thus, the constructors of an abstract class are used from constructor of its subclass.

If the abstract class doesn't have a constructor, a class that extends that abstract class will not get compiled.