Scala Trait

A trait is like an interface with a partial implementation. In scala, trait is a collection of abstract and non-abstract methods. You can create trait that can have all abstract methods or some abstract and some non-abstract methods.

A variable that is declared either by using val or var keyword in a trait get internally implemented in the class that implements the trait. Any variable which is declared by using val or var but not initialized is considered abstract.

Traits are compiled into Java interfaces with corresponding implementation classes that hold any methods implemented in the traits.

Scala Trait Example

```
trait Printable{
   def print()
}

class A4 extends Printable{
   def print(){
      println("Hello")
   }
}
object MainObject{
   def main(args:Array[String]){
      var a = new A4()
      a.print()
   }
}
```

Output:

Hello

If a class extends a trait but does not implement the members declared in that trait, it must be declared abstract. Let's see an example.

Scala Trait Example

```
trait Printable{
   def print()
}

abstract class A4 extends Printable{          // Must declared as abstract class
  def printA4(){
     println("Hello, this is A4 Sheet")
  }
}
```

Scala Trait Example: Implementing Multiple Traits in a Class

If a class implements multiple traits, it will extend the first trait, class, abstract class. with keyword is used to extend rest of the traits.

You can achieve multiple inheritances by using trait.

```
trait Printable{
   def print()
}

trait Showable{
  def show()
}
```

```
class A6 extends Printable with Showable{
   def print(){
      println("This is printable")
   }
   def show(){
      println("This is showable");
   }
}

object MainObject{
   def main(args:Array[String]){
      var a = new A6()
      a.print()
      a.show()
   }
}
```

Output:

```
This is printable
This is showable
```

Scala Trait having abstract and non-abstract methods

You can also define method in trait as like in abstract class. I.e. you can treat trait as abstract class also. In scala, trait is almost same as abstract class except that it can't have constructor. You can't extend multiple abstract classes but can extend multiple traits.

Scala Trait Example

```
trait Printable{
```

```
   def print()        // Abstract method
   def show(){        // Non-abstract method
      println("This is show method")
   }
}

class A6 extends Printable{
   def print(){
      println("This is print method")
   }
}

object MainObject{
   def main(args:Array[String]){
      var a = new A6()
      a.print()
      a.show()
   }
}
```

Output:

```
This is print method
This is show method
```

Scala Trait Mixins

In scala, trait mixins means you can extend any number of traits with a class or abstract class. You can extend only traits or combination of traits and class or traits and abstract class.

It is necessary to maintain order of mixins otherwise compiler throws an error.

You can use mixins in scala like this:

Scala Trait Example: Mixins Order Not Maintained

In this example, we have extended a trait and an abstract class. Let's see what happen.

```
trait Print{
   def print()
}


abstract class PrintA4{
   def printA4()
}
class A6 extends Print with PrintA4{
   def print(){              // Trait print
      println("print sheet")
   }
   def printA4(){            // Abstract class printA4
      println("Print A4 Sheet")
   }
}
object MainObject{
   def main(args:Array[String]){
      var a = new A6()
      a.print()
      a.printA4()
   }
```

```
}
```

Output:

```
error: class PrintA4 needs to be a trait to be mixed in
class A6 extends Print with PrintA4{
                    ^
one error found
```

The above program throws a compile time error, because we didn't maintain mixins order.

Scala Mixins Order

The right mixins order of trait is that any class or abstract class which you want to extend, first extend this. All the traits will be extended after this class or abstract class.

Scala Trait Example: Mixins Order Maintained

```
trait Print{
   def print()
}

abstract class PrintA4{
   def printA4()
}

class A6 extends PrintA4 with Print{          // First one is abstract class second one is trait
   def print(){                               // Trait print
      println("print sheet")
   }
   def printA4(){                             // Abstract class printA4
```

```
        println("Print A4 Sheet")
    }
}

object MainObject{
    def main(args:Array[String]){
        var a = new A6()
        a.print()
        a.printA4()
    }
}
```

Output:

print sheet

Print A4 Sheet

Another Example of Scala Trait

Here, we have used one more approach to extend trait in our program. In this approach, we extend trait during object creation. Let's see an example.

```
trait Print{
    def print()
}
abstract class PrintA4{
    def printA4()
}
class A6 extends PrintA4 {
    def print(){                    // Trait print
```

```
        println("print sheet")
    }
    def printA4(){                    // Abstract class printA4
        println("Print A4 Sheet")
    }
}

object MainObject{
    def main(args:Array[String]){
        var a = new A6() with Print          // You can also extend trait during object creation
        a.print()
        a.printA4()
    }
}
```

Output:

```
print sheet
Print A4 Sheet
```