

Abstract Members

Introduction

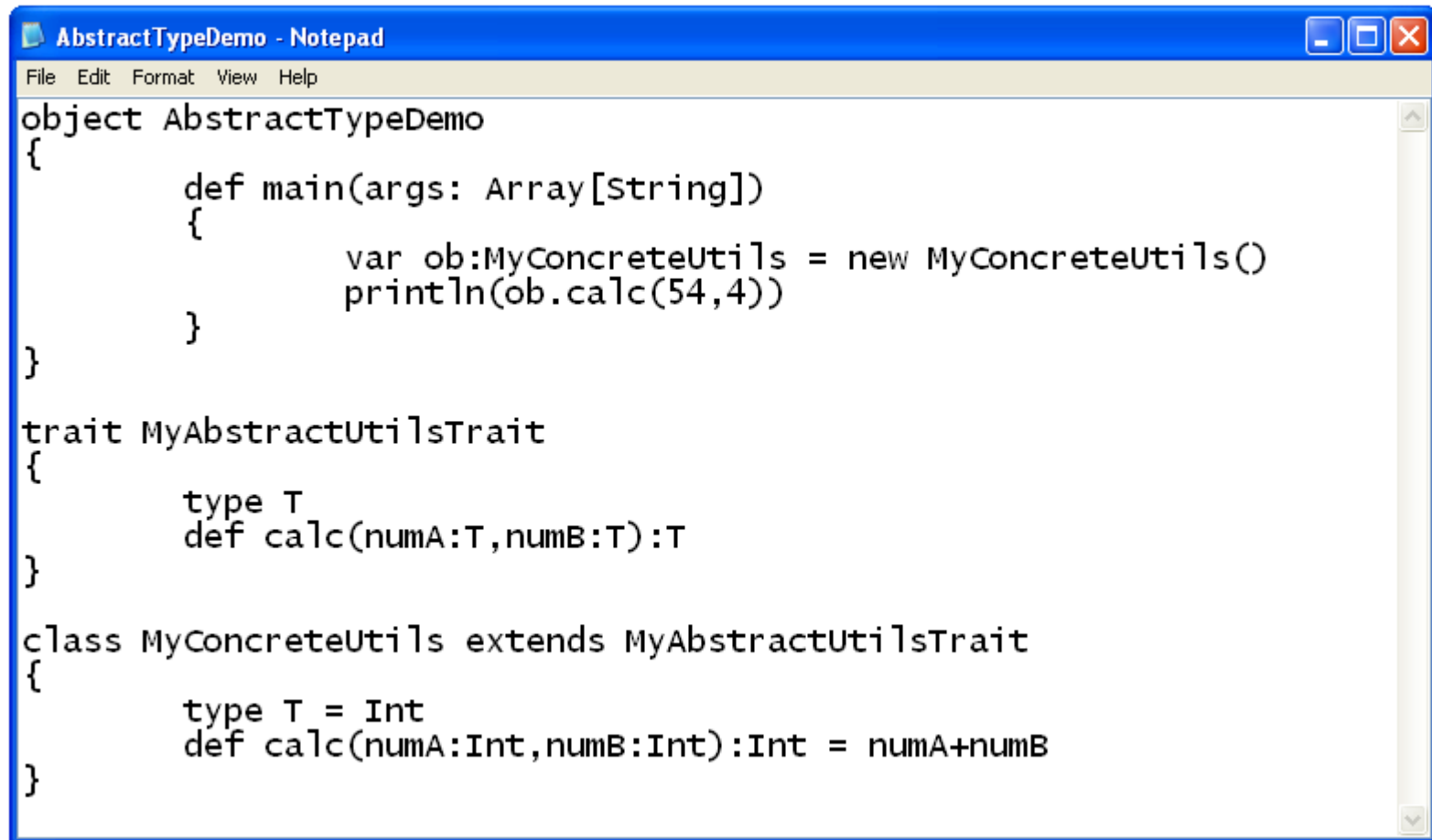
- ❖ When declaring a class or a trait we can declare an abstract member.
- ❖ An abstract member is a member that doesn't have a complete definition.
- ❖ Once a member was declared as abstract we should implement it within the subclass.
- ❖ Scala allows us to declare abstract methods, abstract fields (both vals and vars) and abstract types.

Abstract Types

- ❖ We define an abstract type using the `type` keyword.

```
trait MyAbstractUtilsTrait
{
  type T
  def calc(numA:T, numB:T) :T
}
```

Abstract Types

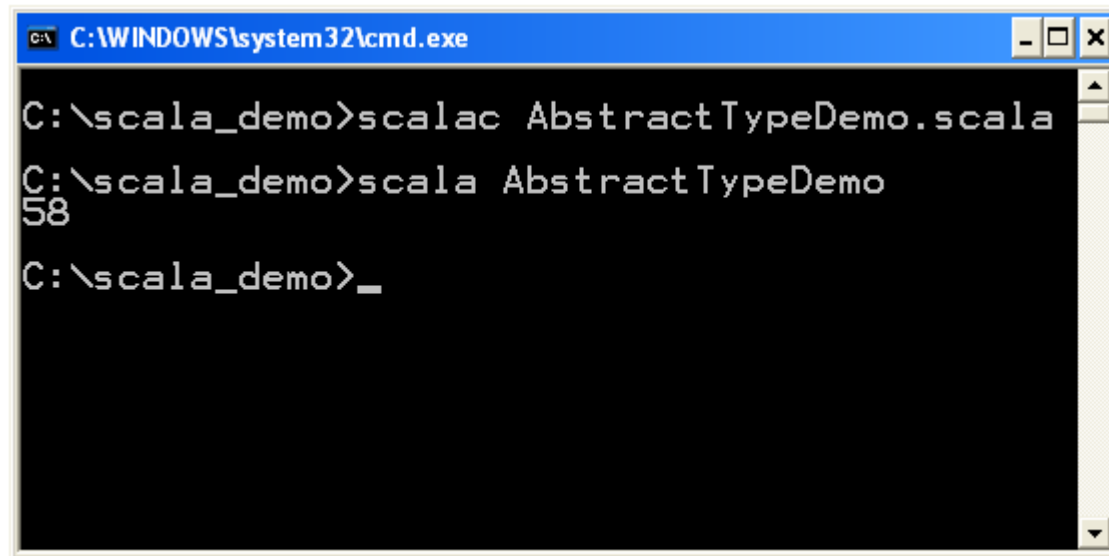


```
AbstractTypeDemo - Notepad
File Edit Format View Help
object AbstractTypeDemo
{
    def main(args: Array[String])
    {
        var ob:MyConcreteUtils = new MyConcreteUtils()
        println(ob.calc(54,4))
    }
}

trait MyAbstractUtilsTrait
{
    type T
    def calc(numA:T,numB:T):T
}

class MyConcreteUtils extends MyAbstractUtilsTrait
{
    type T = Int
    def calc(numA:Int,numB:Int):Int = numA+numB
}
```

Abstract Types



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac AbstractTypeDemo.scala
C:\scala_demo>scala AbstractTypeDemo
58
C:\scala_demo>_
```

Abstract Vals

- ❖ We define an abstract val when we define it without assigning a value.

```
abstract class Food
{
    val name:String
}
```

```
class Orange extends Food
{
    val name:String = "orange"
}
```

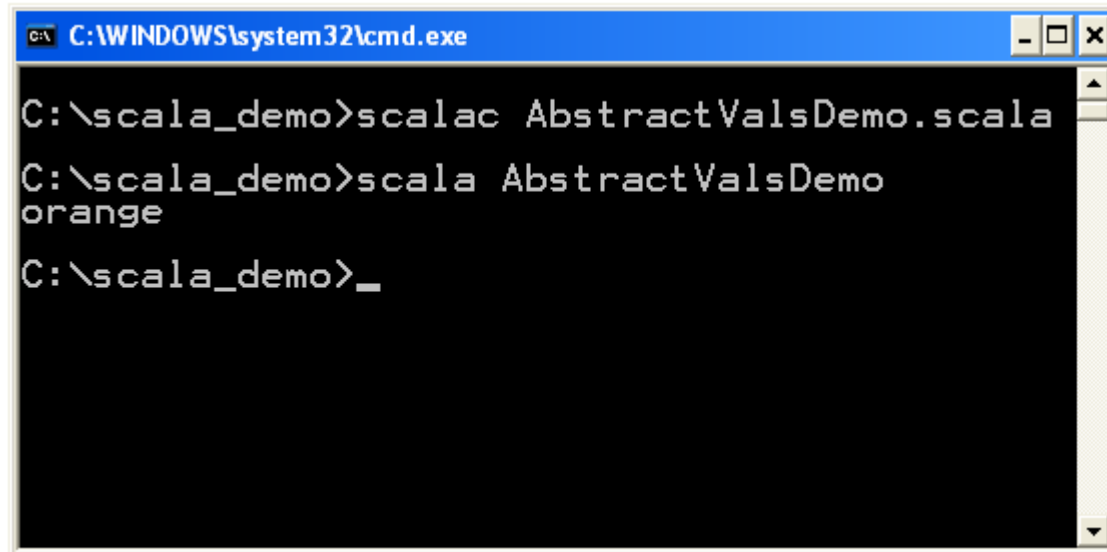
Abstract Vals

```
AbstractValsDemo - Notepad
File Edit Format View Help
object AbstractValsDemo
{
    def main(args: Array[String])
    {
        var ob:Food = new Orange()
        println(ob)
    }
}

abstract class Food
{
    val name:String
}

class Orange extends Food
{
    val name:String = "orange"
    override def toString:String = name
}
```

Abstract Vals



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac AbstractValsDemo.scala
C:\scala_demo>scala AbstractValsDemo
orange
C:\scala_demo>_
```


Abstract Vars

- ❖ As with `abstract val`, an `abstract var` declares just a name and a type and it doesn't include an initial value.

```
abstract class Currency
{
    var amount: Long
}
```

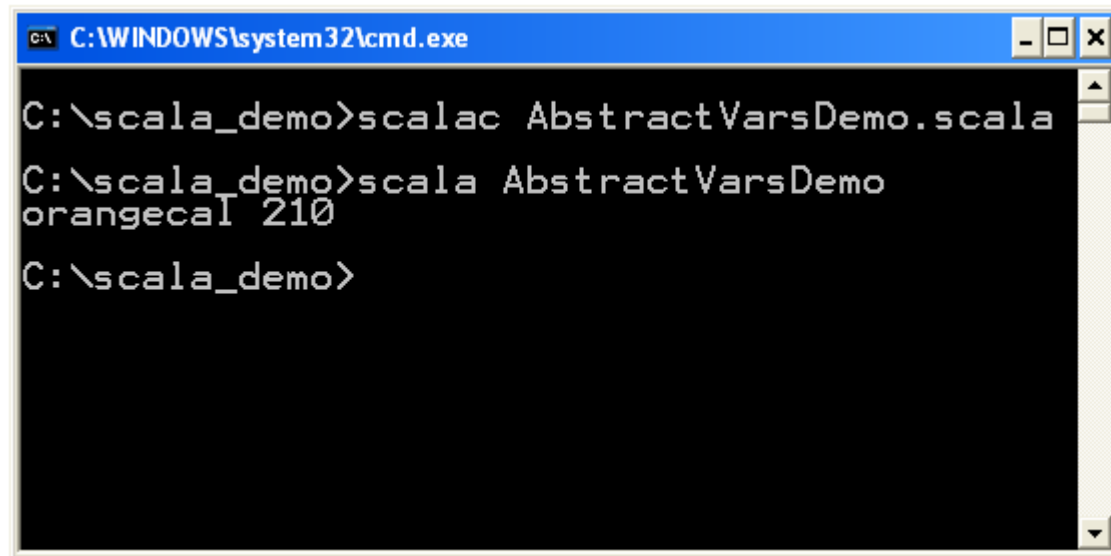
Abstract Vars

```
AbstractVarsDemo - Notepad
File Edit Format View Help
object AbstractVarsDemo
{
    def main(args: Array[String])
    {
        var ob:Food = new Orange()
        println(ob)
    }
}

abstract class Food
{
    val name:String
    var calorie:Int
}

class Orange extends Food
{
    val name:String = "orange"
    var calorie:Int = 210
    override def toString:String = name + "cal " + calorie
}
```

Abstract Vars



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac AbstractVarsDemo.scala
C:\scala_demo>scala AbstractVarsDemo
orangeCat 210
C:\scala_demo>
```

Abstract Method

- ❖ Defining an abstract method means defining a method without a body.

```
abstract class Currency
{
    val amount: Long
    def details: String
}
```

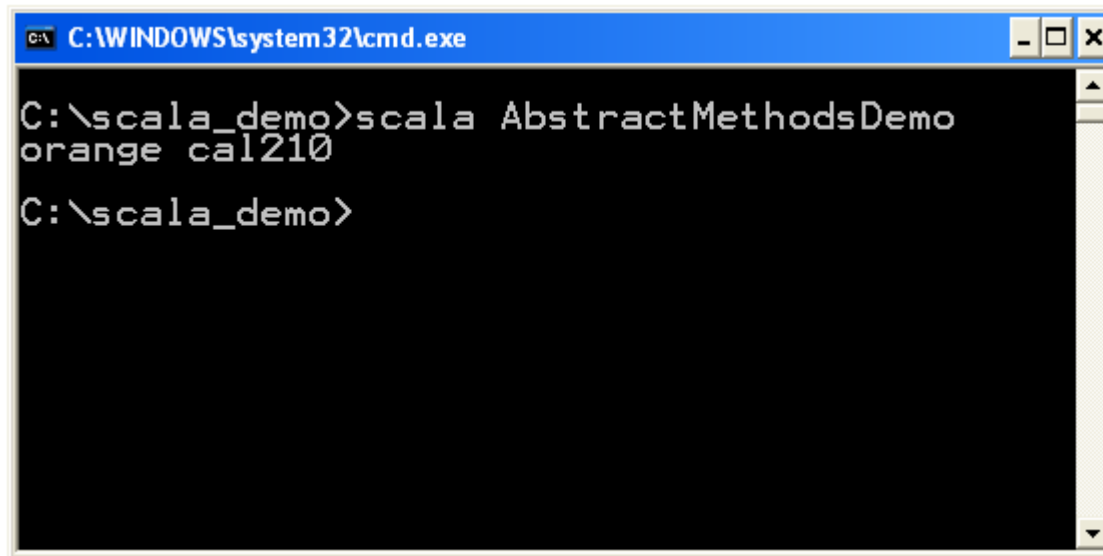
Abstract Method

```
AbstractMethodsDemo - Notepad
File Edit Format View Help
object AbstractMethodsDemo
{
    def main(args: Array[String])
    {
        var ob:Food = new Orange()
        ob.details()
    }
}

abstract class Food
{
    val name:String
    var calorie:Int
    def details():Unit
}

class Orange extends Food
{
    val name:String = "orange"
    var calorie:Int = 210
    override def details():Unit =
    {
        println(name+" cal"+calorie)
    }
}
```

Abstract Method



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scala AbstractMethodsDemo
orange cal210
C:\scala_demo>
```

Abstract Members

© 2008 Haim Michael (Scala, Abstract Menbers)

Introduction

- ❖ When declaring a class or a trait we can declare an abstract member.
- ❖ An abstract member is a member that doesn't have a complete definition.
- ❖ Once a member was declared as abstract we should implement it within the subclass.
- ❖ Scala allows us to declare abstract methods, abstract fields (both vals and vars) and abstract types.

© 2008 Haim Michael (Scala, Abstract Members)

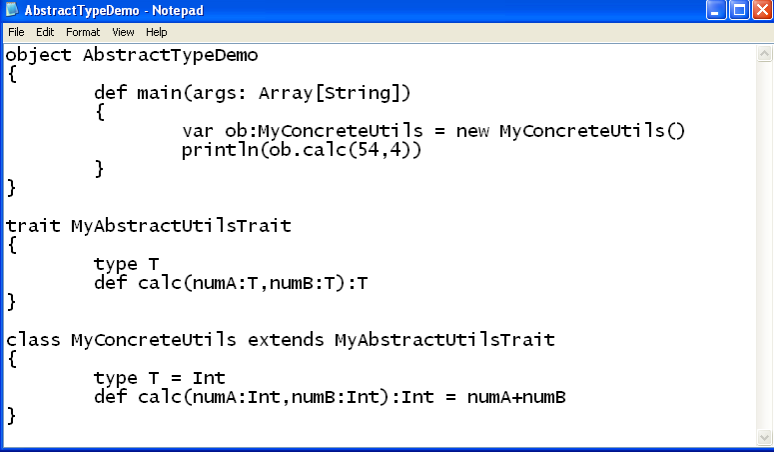
Abstract Types

- ❖ We define an abstract type using the `type` keyword.

```
trait MyAbstractUtilsTrait
{
  type T
  def calc(numA:T,numB:T):T
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Types



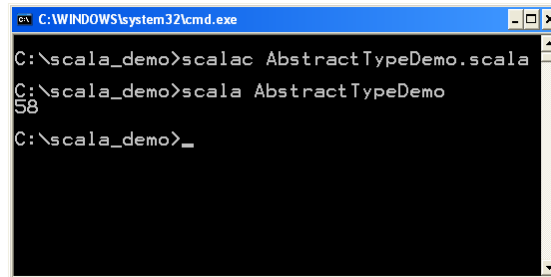
```
object AbstractTypeDemo
{
    def main(args: Array[String])
    {
        var ob:MyConcreteUtils = new MyConcreteUtils()
        println(ob.calc(54,4))
    }
}

trait MyAbstractUtilsTrait
{
    type T
    def calc(numA:T,numB:T):T
}

class MyConcreteUtils extends MyAbstractUtilsTrait
{
    type T = Int
    def calc(numA:Int,numB:Int):Int = numA+numB
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Types



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac AbstractTypeDemo.scala
C:\scala_demo>scala AbstractTypeDemo
58
C:\scala_demo>_
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Vals

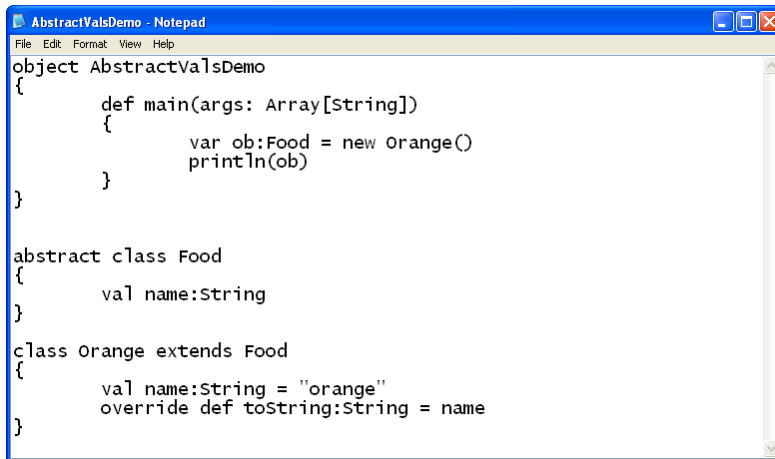
- ❖ We define an abstract val when we define it without assigning a value.

```
abstract Class Food
{
    val name:String
}

class Orange extends Food
{
    val name:String = "orange"
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Vals

A screenshot of a Notepad window titled "AbstractValsDemo - Notepad". The window contains Scala code defining an object, an abstract class, and a class. The code is as follows:

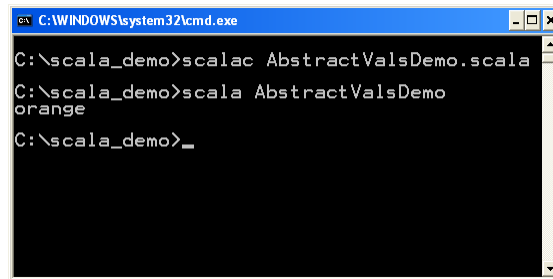
```
object AbstractValsDemo
{
    def main(args: Array[String])
    {
        var ob:Food = new Orange()
        println(ob)
    }
}

abstract class Food
{
    val name:String
}

class Orange extends Food
{
    val name:String = "orange"
    override def toString:String = name
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Vals



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac AbstractValsDemo.scala
C:\scala_demo>scala AbstractValsDemo
orange
C:\scala_demo>
```

© 2008 Haim Michael (Scala, Abstract Members)

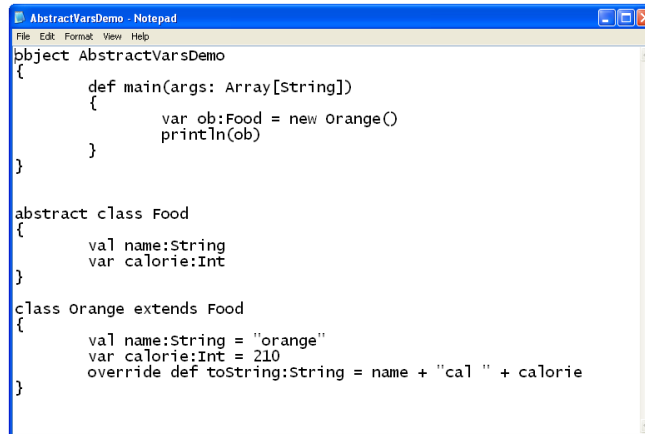
Abstract Vars

- ❖ As with `abstract val`, an `abstract var` declares just a name and a type and it doesn't include an initial value.

```
abstract class Currency
{
    var amount: Long
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Vars

A screenshot of a Notepad window titled "AbstractVarsDemo - Notepad". The window contains Scala code defining an object, an abstract class, and a concrete class. The code is as follows:

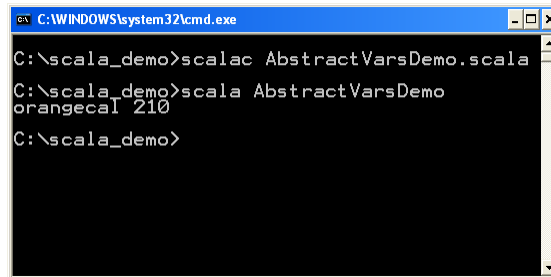
```
object AbstractVarsDemo
{
    def main(args: Array[String])
    {
        var ob:Food = new Orange()
        println(ob)
    }
}

abstract class Food
{
    val name:String
    var calorie:Int
}

class Orange extends Food
{
    val name:String = "orange"
    var calorie:Int = 210
    override def toString:String = name + "cal " + calorie
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Vars



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac AbstractVarsDemo.scala
C:\scala_demo>scala AbstractVarsDemo
orangeCat 210
C:\scala_demo>
```

© 2008 Haim Michael (Scala, Abstract Members)

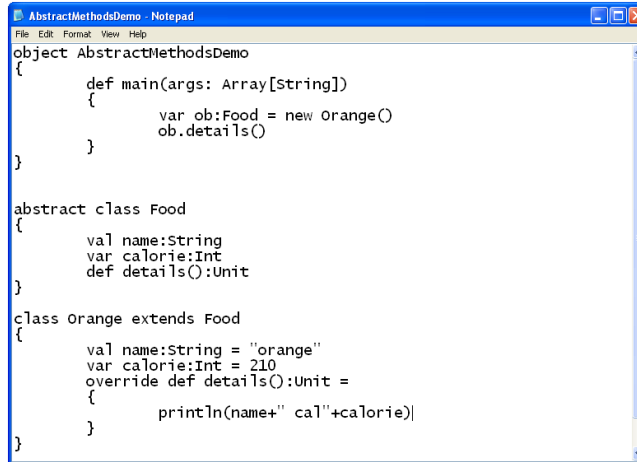
Abstract Method

- ❖ Defining an abstract method means defining a method without a body.

```
abstract class Currency
{
    val amount: Long
    def details: String
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Method



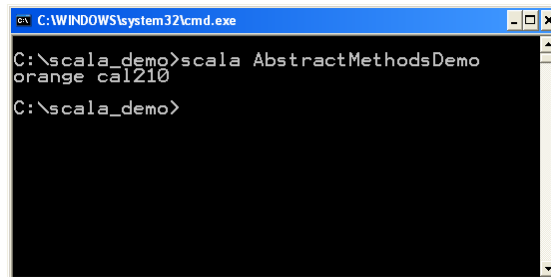
```
AbstractMethodsDemo - Notepad
File Edit Format View Help
object AbstractMethodsDemo
{
    def main(args: Array[String])
    {
        var ob:Food = new Orange()
        ob.details()
    }
}

abstract class Food
{
    val name:String
    var calorie:Int
    def details():Unit
}

class Orange extends Food
{
    val name:String = "orange"
    var calorie:Int = 210
    override def details():Unit =
    {
        println(name+" cal"+calorie)
    }
}
```

© 2008 Haim Michael (Scala, Abstract Members)

Abstract Method



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scala AbstractMethodsDemo
orange cal210
C:\scala_demo>
```

© 2008 Haim Michael (Scala, Abstract Members)