

Traits

Introduction

- ❖ Traits encapsulate methods and fields definitions we can reuse by mixing them into classes we define.
- ❖ Unlike classes inheritance that allow each class to inherit one class only, a class can mix in any number of traits.

Trait Definition

- ❖ The syntax is the same syntax we use when defining a class. The only difference is using the 'trait' keyword instead of 'class'.

```
trait Academic
{
  def think()
  {
    println("i think... i exist.")
  }
}
```

Trait Definition

- ❖ Once a trait was defined it can be mixed in to a class using either the keyword `extends` or the keyword `with`.

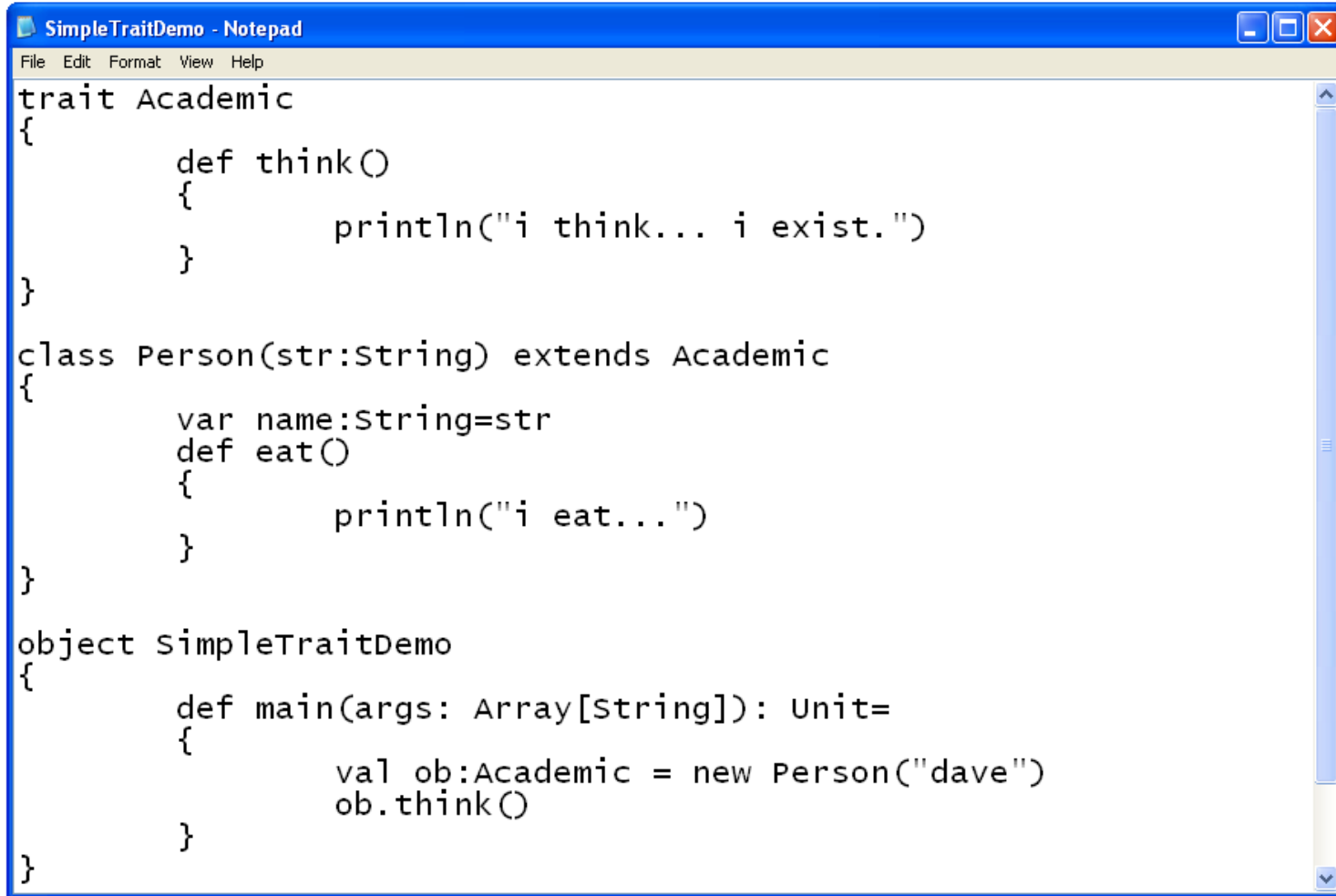
```
class Person extends Academic
{
    ...
}
```

- ❖ We can use the keyword `with` when our class already extends a specific other class or trait. We cannot use `with` if our class extends one trait only.

Trait Definition

- ❖ We can use methods inherited from a trait just as any method inherited from a super class.
- ❖ Once a trait is defined we get a new type, similarly to defining a new class.

Trait Definition



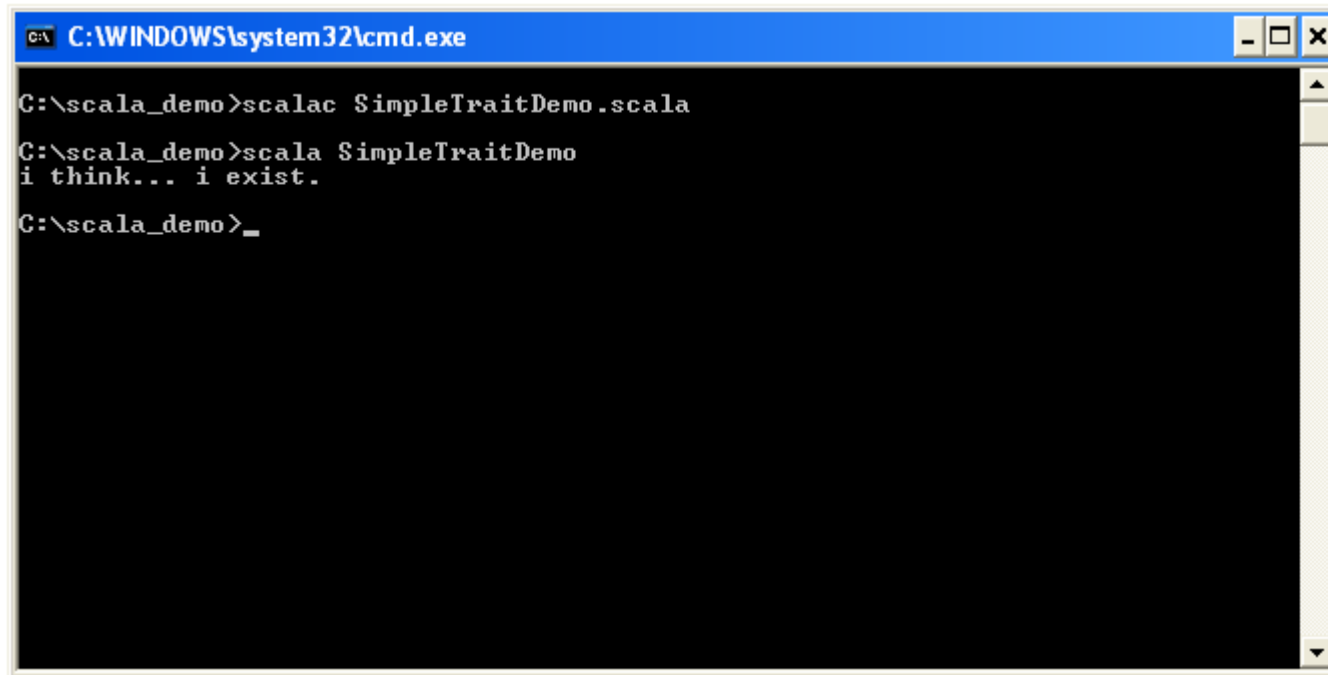
```
SimpleTraitDemo - Notepad
File Edit Format View Help

trait Academic
{
    def think()
    {
        println("i think... i exist.")
    }
}

class Person(str:String) extends Academic
{
    var name:String=str
    def eat()
    {
        println("i eat...")
    }
}

object SimpleTraitDemo
{
    def main(args: Array[String]): Unit=
    {
        val ob:Academic = new Person("dave")
        ob.think()
    }
}
```

Trait Definition



```
C:\WINDOWS\system32\cmd.exe
C:\scala_demo>scalac SimpleTraitDemo.scala
C:\scala_demo>scala SimpleTraitDemo
i think... i exist.
C:\scala_demo>_
```

Trait Definition

- ❖ When we want to mix a trait into a class that explicitly extends another class we should use `extends` in order to show the extension from the other class and `with` in order to show that we mix-in the trait.

```
class Teacher extends Student with Academic with Personal
{
    ...
}
```


Trait Definition

- ❖ We cannot define a trait with class parameters. Traits don't have a primary constructor. Traits don't have constructors at all.

Traits Multiple Inheritance

- ❖ Using Traits we can inherit from multiple class-like constructs and yet stay away of the problematic behavior we know from multiple inheritance in C++.
- ❖ We cannot define a class that extends multiple traits and get the same implemented method from more than one trait.