Programming Methodology

Lecture 05 – classes, objects, tools, and patterns

UMASSCS

SCHOOL OF COMPUTER SCIENCE

Objectives

- To learn how to use scala, scalac, and fsc
- To learn how to recognize “functional”
- To learn about classes and objects in Scala
- To learn a little “design pattern” in Java
  - What is a design pattern?
  - The **Singleton** pattern in Java
  - The **Factory Method** in Java
  - How does Scala handle this?
- To learn how to create Scala applications
- To learn about basic types and operations in Scala

Project Assignment 01

- How hard was Project Assignment 01?
  a. piece of cake!
  b. moderately easy
  c. reasonable
  d. hard
  e. very hard

Project Assignment 01

- How much time on Project Assignment 01?
  a. 1-2 hours
  b. 2-4 hours
  c. 4-6 hours
  d. 6-8 hours
  e. 8+ hours

Project Assignment 01

- Do you understand Java packages?
  a. completely
  b. mostly
  c. a little
  d. not at all

Recognizing Functional

- Scala is:
  - object-oriented
  - functional
- Important to become acquainted with this.
- Let us look at some script examples:
  - Recognize Functional (scala-script/06-functional.scala)
  - Formatting Lines (scala-script/07-fmt-lines.scala)

Classes and Objects

- Scala has classes!
  - including fields and methods
- Scala has objects!
  - Instantiated from classes
- **Similar to Java, but Different!**

Class Declaration/Instantiation


```
class ChecksumAccumulator {
  // class definition goes here
}
```

```
new ChecksumAccumulator
```

Class Fields


```
class ChecksumAccumulator {
  var sum = 0
}
```

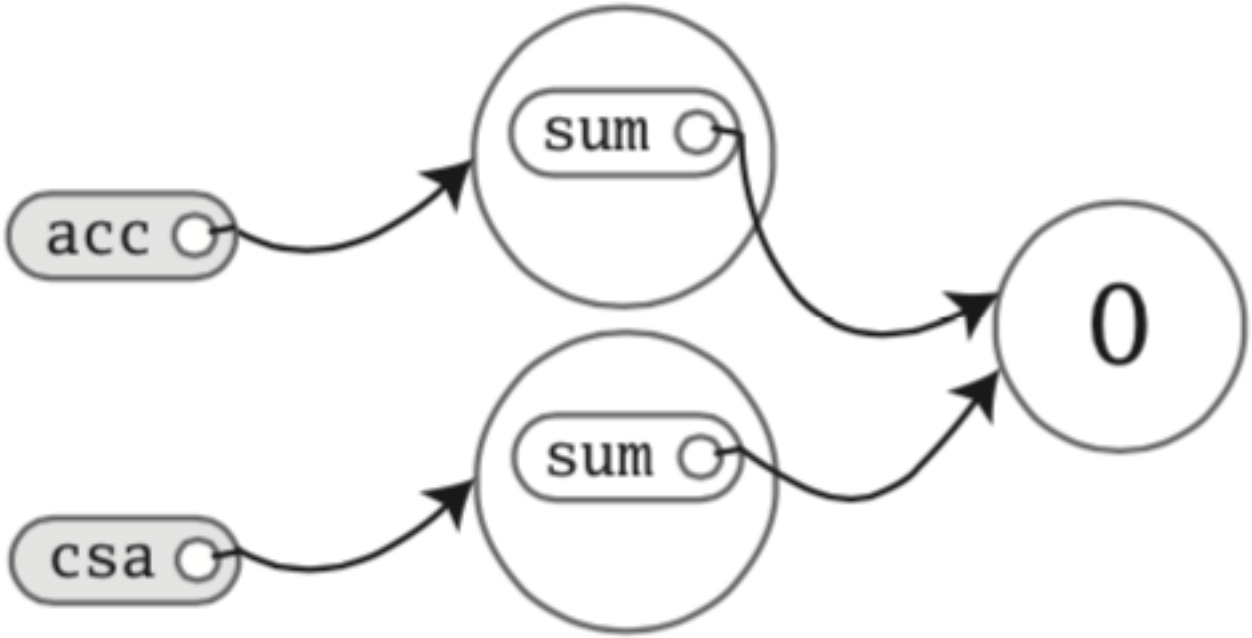
Class Fields


```
class ChecksumAccumulator {
  var sum = 0
}

val acc = new ChecksumAccumulator
val csa = new ChecksumAccumulator
```

Class Fields

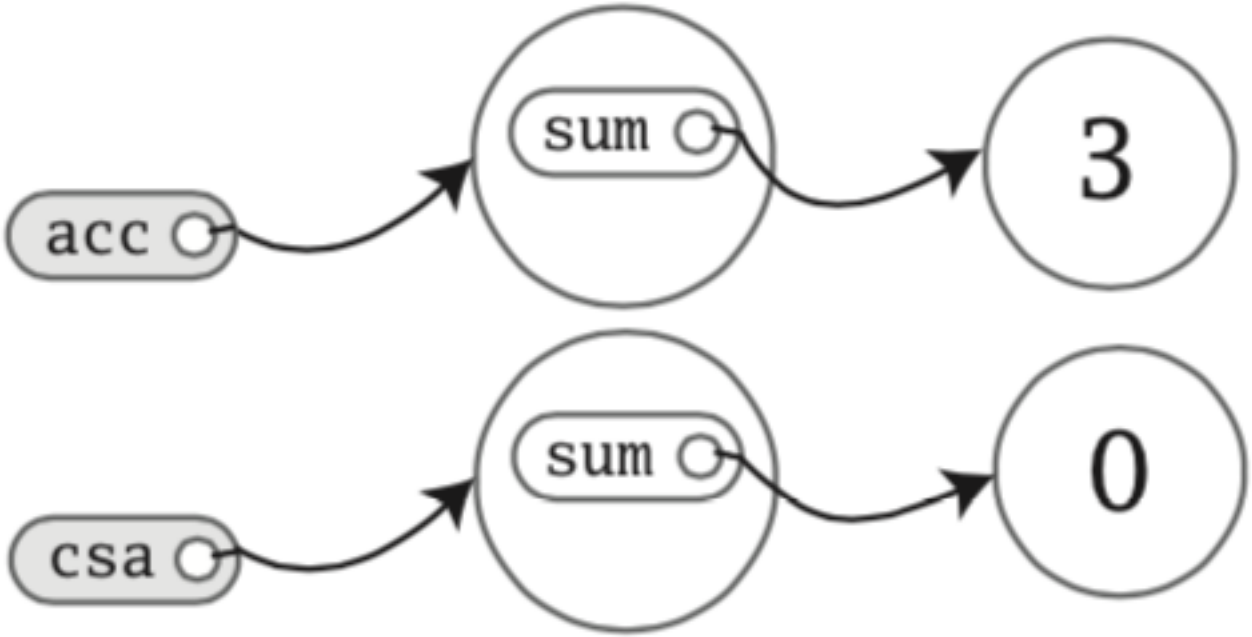
```
val acc = new ChecksumAccumulator
val csa = new ChecksumAccumulator
```



Class Fields


```
val acc = new ChecksumAccumulator
val csa = new ChecksumAccumulator

acc.sum = 3
```



Class Fields


```
val acc = new ChecksumAccumulator
val csa = new ChecksumAccumulator


acc.sum = 3


// Won't compile, because acc is a val
acc = new ChecksumAccumulator
```

Privacy


```
class ChecksumAccumulator {
  private var sum = 0
}
```

Privacy


```
class ChecksumAccumulator {
  private var sum = 0
}

val acc = new ChecksumAccumulator
acc.sum = 5 // Won't compile, because sum is private
```

The way you make members **public** in Scala is to simply specify nothing. In Java, you would be required to use the public keyword.

Methods


```
class ChecksumAccumulator {

  private var sum = 0

  def add(b: Byte): Unit = {
    sum += b
  }

  def checksum(): Int = {
    return ~(sum & 0xFF) + 1
  }
}
```

Methods


```
class ChecksumAccumulator {

  private var sum = 0

  def add(b: Byte): Unit = {
    sum += b
  }

  def checksum(): Int = {
    return ~(sum & 0xFF) + 1
  }
}
```

An important note to make here is that all function/method parameters in Scala are *vals* not *vars*.

This means you can't do this:

```
def add(b: Byte): Unit =
{
  b = 1
  sum += b
}
```

Conciseness


```
// In file ChecksumAccumulator.scala
class ChecksumAccumulator {
  private var sum = 0
  def add(b: Byte) { sum += b }
  def checksum(): Int = ~(sum & 0xFF) + 1
}
```

Because these methods are short you would typically put them on a single line!

Method Notes


```
scala> def f(): Unit = "this String gets lost"
f: ()Unit
```

Method Notes


```
scala> def f(): Unit = "this String gets lost"
f: ()Unit

scala> def g() { "this String gets lost too" }
g: ()Unit
```

Method Notes


```
scala> def f(): Unit = "this String gets lost"
f: ()Unit

scala> def g() { "this String gets lost too" }
g: ()Unit

scala> def h() = { "this String gets returned!" }
h: ()java.lang.String

scala> h
res0: java.lang.String = this String gets returned!
```

Compiling Scala

- We saw Scala in the small
  - Use scripts
- How do we compile Scala to class files?
  - scalac
  - scala
  - fsc
- **Example:** scala-compiling

What is a “Design Pattern”?

In software engineering, a **design pattern** is a general reusable solution to a commonly occurring problem within a given context in software design.

- **Recognizing the problem**
  - Recognizing the solution
- **Different languages offer different solutions**
  - Programming languages are tools
  - Designed for different purposes
  - Design patterns often occur when a language does not provide support to solve a typical problem!

Singletons

In class-based programming, the **singleton** pattern is implemented by creating a class with a method that creates a new instance of the class if one does not exist. If an instance already exists, it simply returns the a reference to that object. To make sure that the object can't be instantiated in any other way, the constructor is made private.

- How do we typically create objects in Java?
  - Yes, we use the **new** operator
- What if we wanted to ensure that only **one** instance of a class exists?
- Why would we want to do this?
- Let us look at an example:
  - **Example:** java-singleton