Classes

Java

```
class StudentJ {
    private int age;
    public StudentJ(int age){
        this.age = age;
    }
}
```

Scala

```
class StudentS1 (val name: String)
val student1 = new StudentS1("me")
//student1.name = "bye" //COMPILER ERROR
```

```
class StudentS2 (var name: String)
val student2 = new StudentS2("me")
student2.name="badnews" //Frowned upon. Don't do this if you want to live
println (student2.name)
```

Traits

Traits are just like interfaces but also like abstract classes in the sense that we can have implementations inside the trait but also have multiple traits inherited by a class

```
trait Pet
trait Awesomeness
```

(Give the audience a hint that we would be covering the following in detail in the next talk)

Case classes

```
case class Dog(name: String) extends Pet with Awesomeness
case class Cat (bellcolor: String) extends Pet
```

Pattern matching

```
def guessTheType(any: Any): String = any match {
  case Dog(name) => s"this is a dog: $name}"
  case StudentS2(name) => "this is a student and it doesn't compile"
  case _ => "stranger things"
}
```

Give a hint about apply and equals overriding.

```
val dog = Dog ("cheddar") //No `new` huh !
```

Objects and companion objects

Singletons are so important that it is a language feature in Scala.

```
object Student{

}
```

Companion objects

Now, if your object has the same name as your class, it's called a companion object. Static methods must be in companion objects. The classes are just data holders (aka) Anaemic classes

Companion objects have the same name as the class

```
class Student (val name: String)

object Student {

    def combine (stu1: Student, stu2: Student) ={
        new Student (stu1.name + stu2.name) //or whatever the hell that means
    }

}
```