

# Understanding Object-Oriented Scala

---



**Harit Himanshu**

@harittweets



# Overview



Understanding Classes and Objects

Creating Classes and Objects

Understanding Singleton Objects

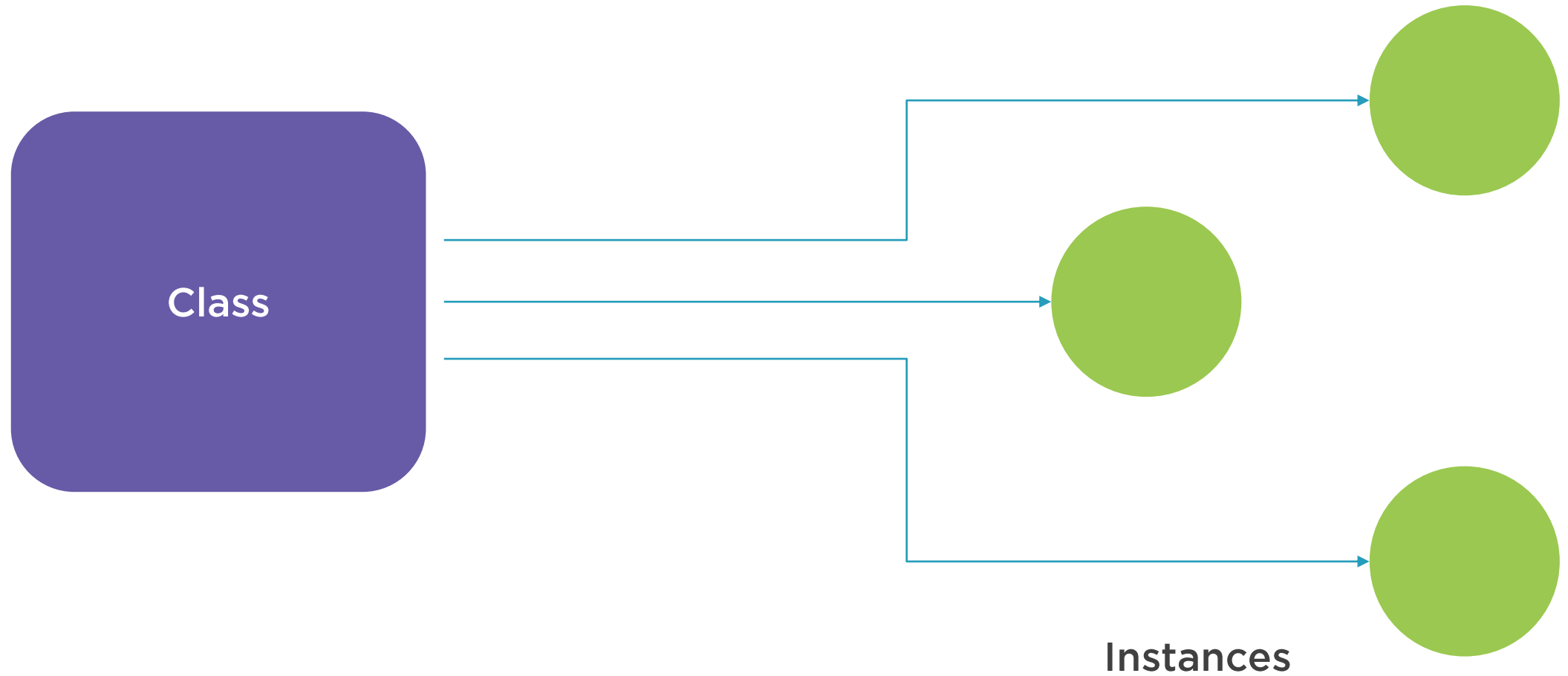
Understanding Functional Objects

Understanding Abstract Classes,  
Inheritance, and Composition

Project Demo



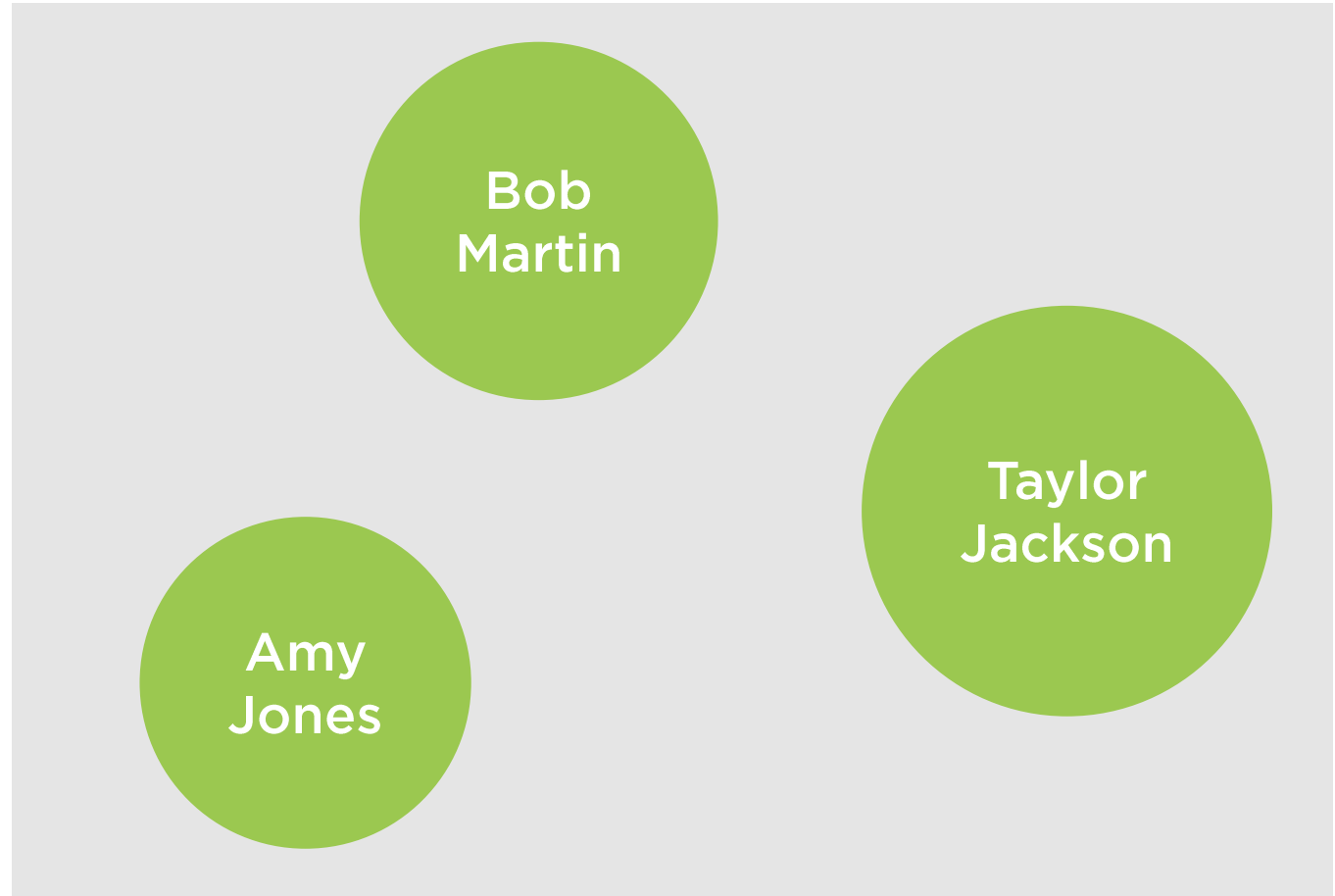
# Class and Instance



# Class and Instance Example



Employee



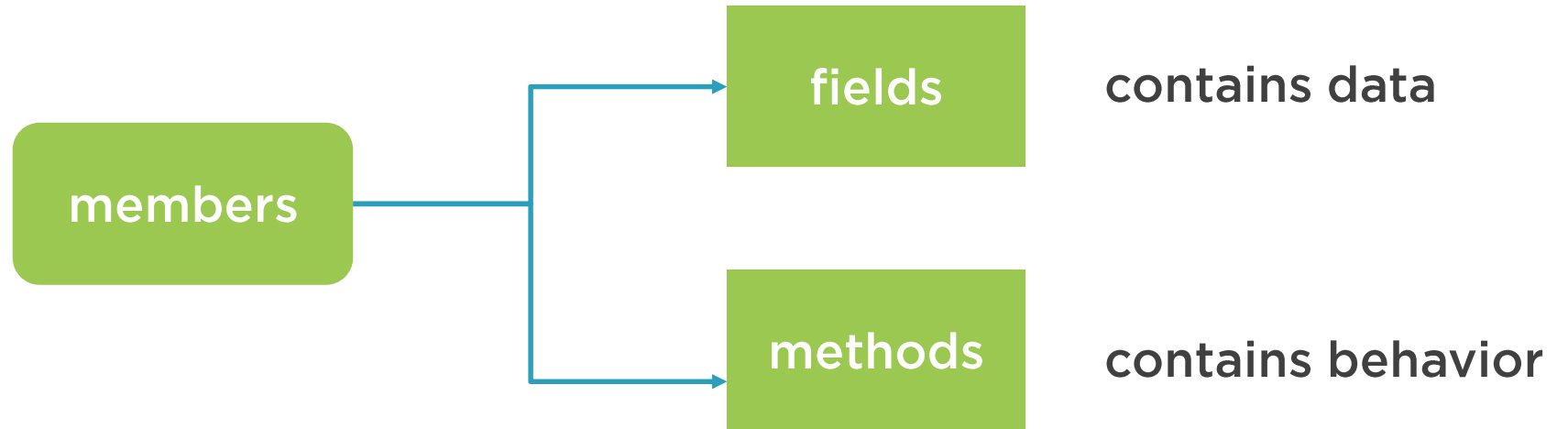
Memory



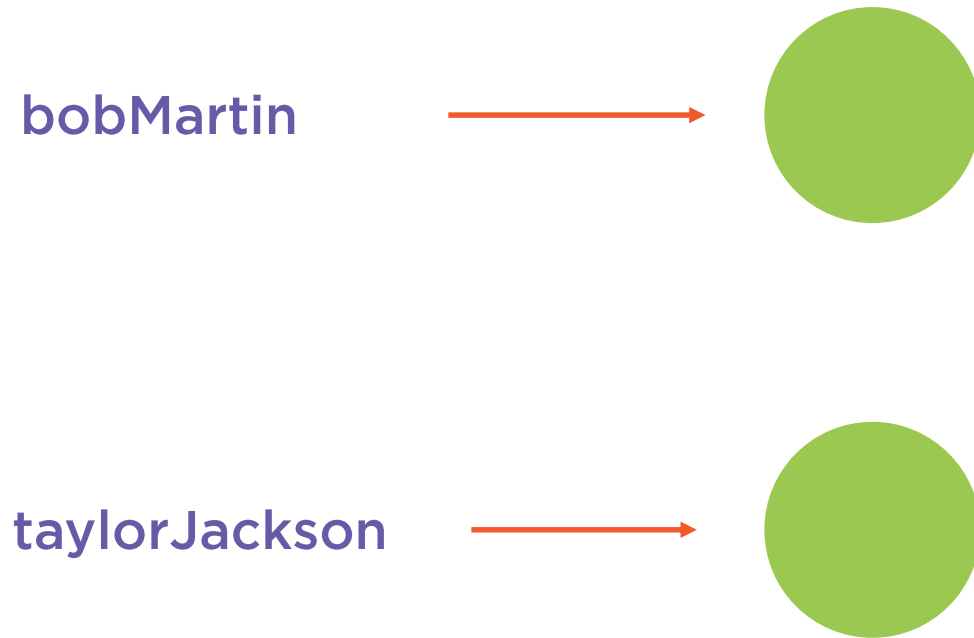
# Class Structure in Scala

```
class Employee {
```

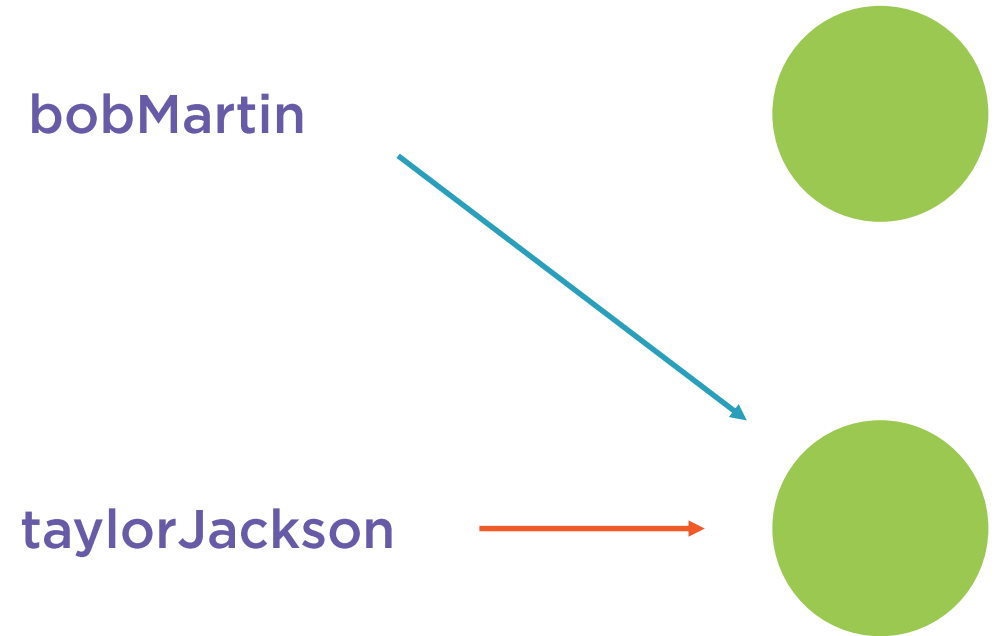
```
}
```



# var References



Before



After



# val References

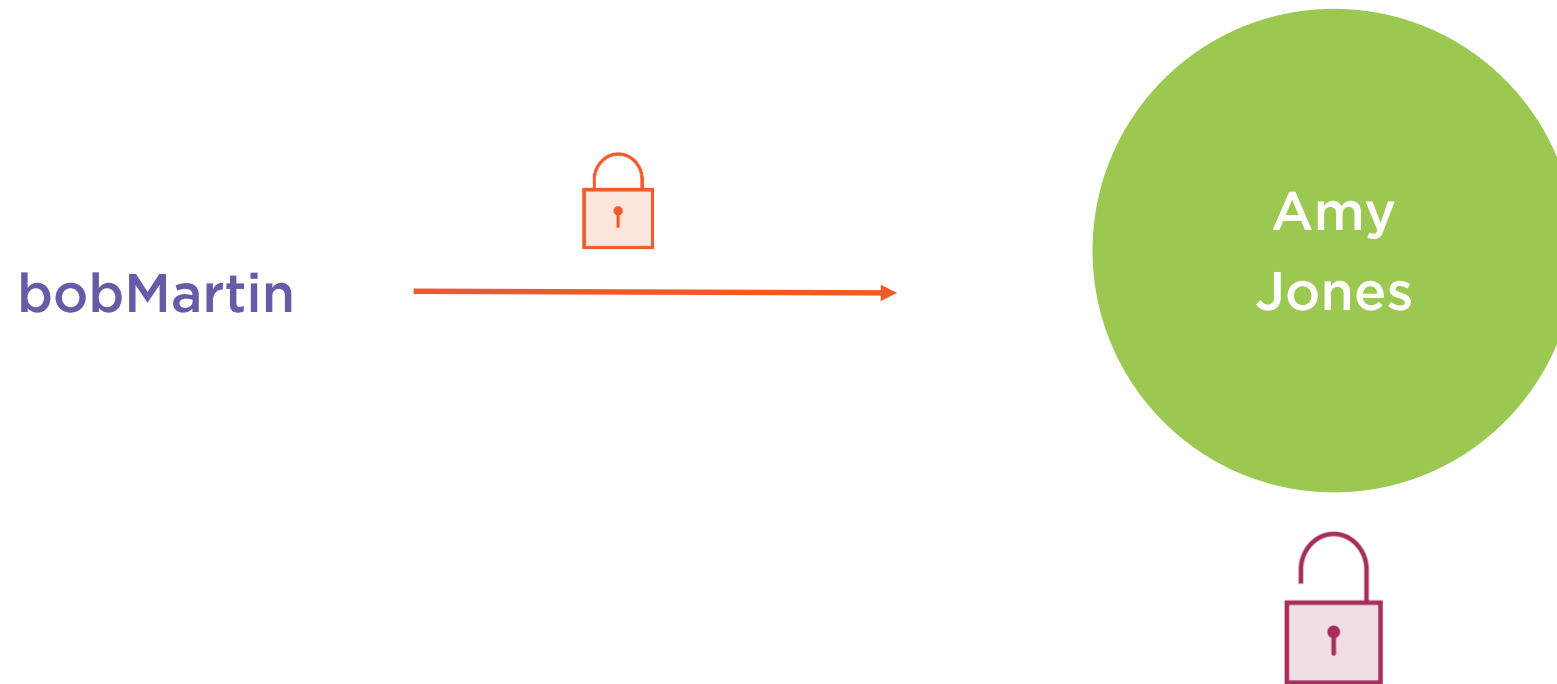


Before

After



# val Reference var State





# How to Avoid Open State Change

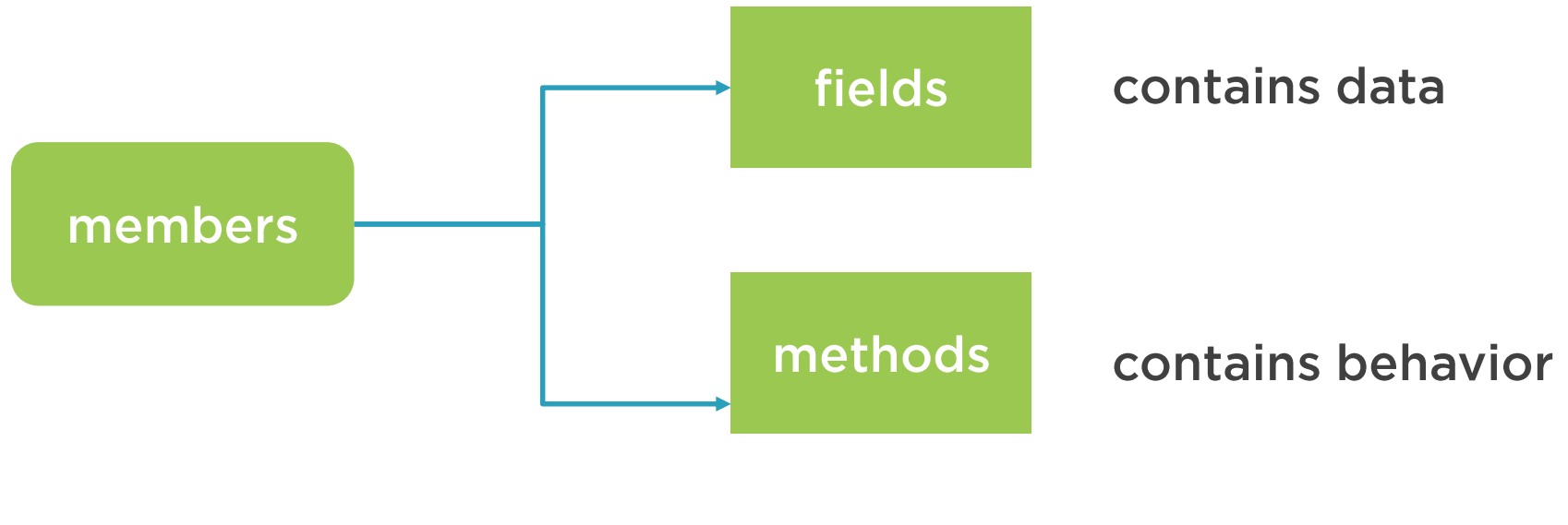
**Class Parameters**

**Constructor Fields**



# Class Parameters

```
class Employee(f: Type) {
```



# Visibility Detour

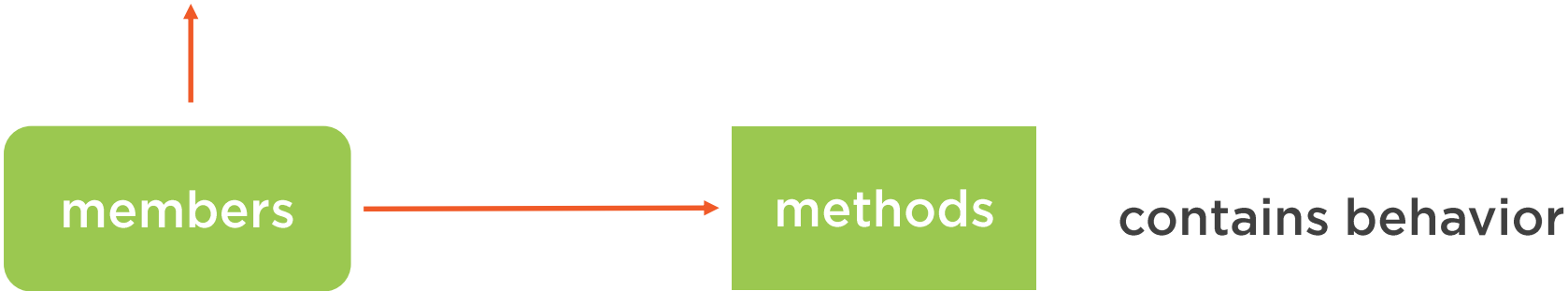
Visibility	Access Allowed	Change Allowed
<b>var</b>	Yes	Yes
<b>val</b>	Yes	No
<b>default</b> (nothing provided)	No	No
<b>private</b> (val or var)	No	No

---



# Instantiating with Fields

```
class Employee(f: String, l: String) {  
    val first: String = f  
    val last: String = l  
}
```



members → methods contains behavior

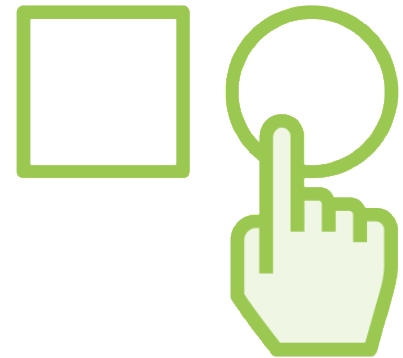
# Singleton Objects



No instantiation



Just one in memory



Initialized on first access

# Companion Object Properties

Employee.scala

2

Object Employee {  
 private val objectSecret = 1  
 def apply(f: String, l:String) = new Employee(f, l)

4

// classSecret   
}

3

class Employee(f: String, l: String) {  
 private val classSecret = 1

// objectSecret   
}



# Singleton Object Example

```
object EmployeeUtils {  
    def getOfficeLocations: Set[Location] = // implementation  
    def getSocialHandles: Set[SocialHandle] = // implementation  
    // more utility methods  
}
```



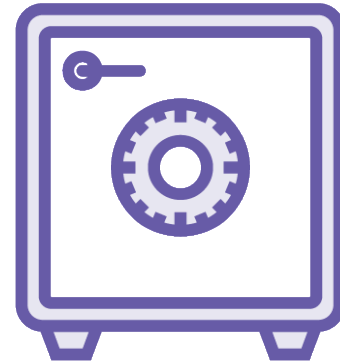
# Immutability benefits



Easy to reason



Share freely



concurrent safety

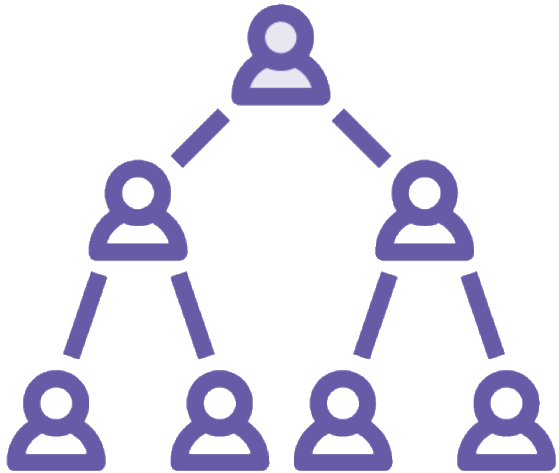


Safe hash keys

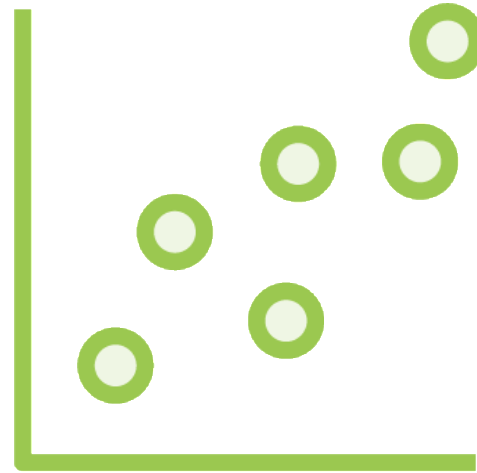




# Relationships



Organization Hierarchy



Stock Prices

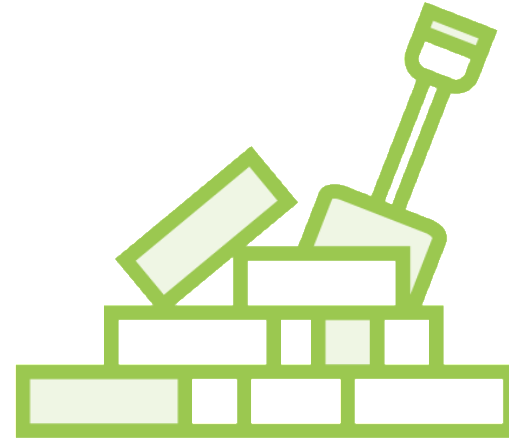


Transport networks

# Inheritance vs. Composition



Inheritance

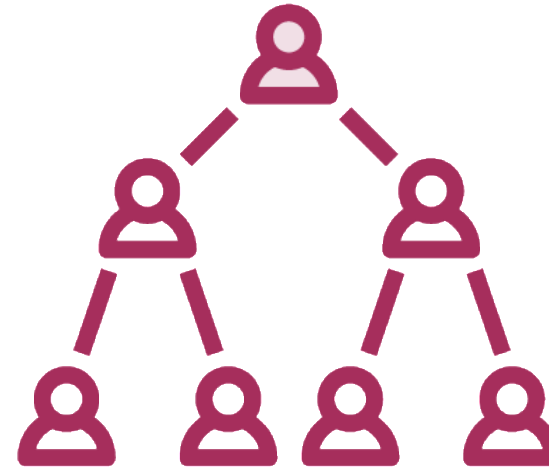


Composition

# Inheritance Examples



Family wealth



Organization chart

# Abstract Class

```
abstract class Employee {
```

```
  val first: String
```

```
  val last: String
```

abstract  
members

declaration

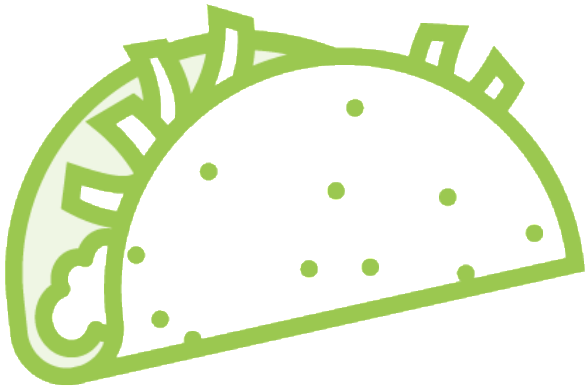
```
  def toString: String = first + “,” + last
```

definition

```
}
```



# Composition Examples



Food



Movie

# Is-a vs. Has-a relationship

## Is-a

Sparrow **is a** bird

Pluralsight **is a** company

Canada **is a** country

## Has-a

Car **has** wheels

Class **has** students

Book **has** pages



# Demo



Creating the Customer class

Creating the Product hierarchy

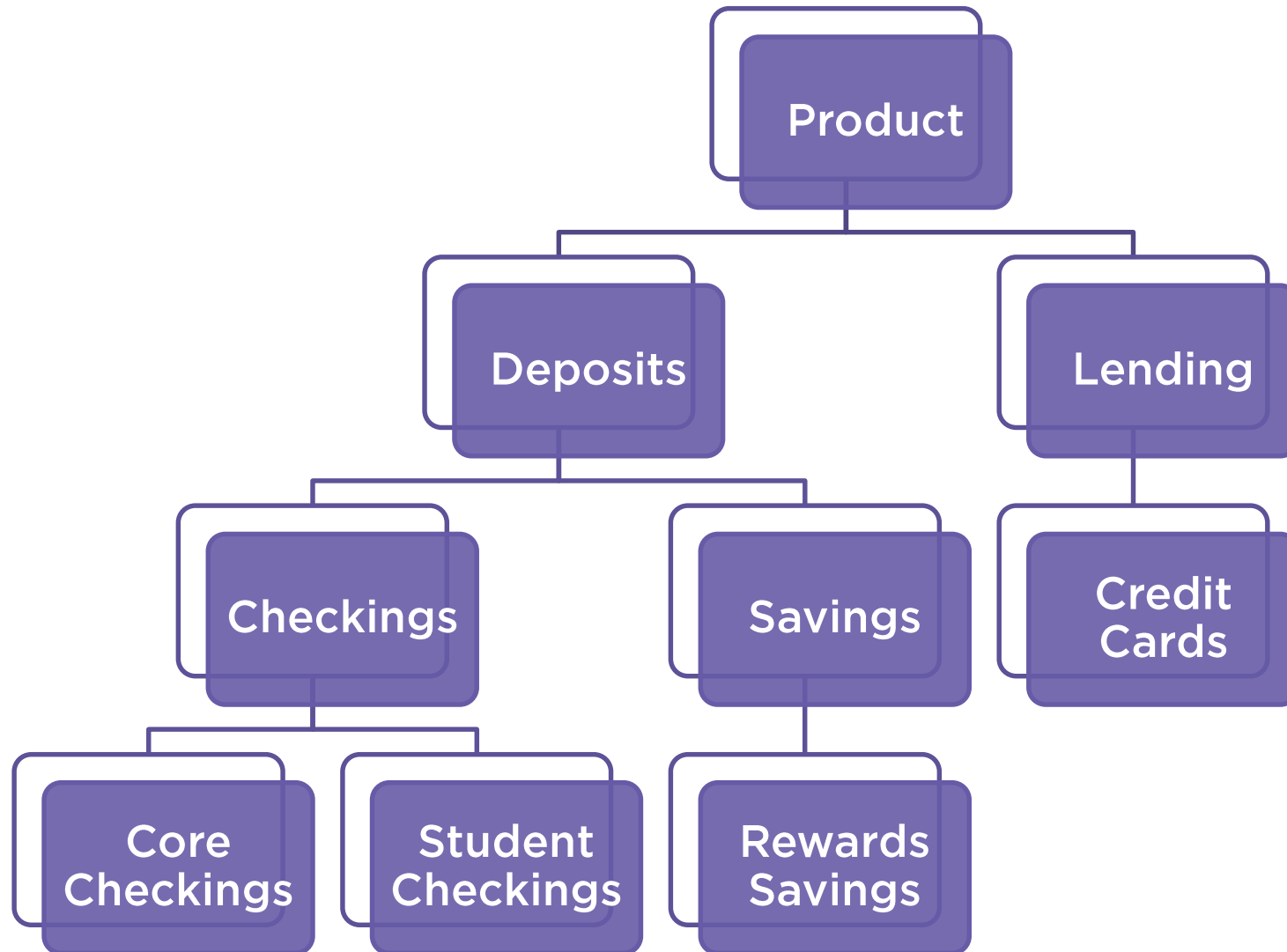
Creating the Account class

Creating the Bank class

Running the project



# Product hierarchy





# Summary



Understanding Classes and Objects

Creating Classes and Objects

Understanding Singleton Objects

Understanding Functional Objects

Understanding Abstract Classes,  
Inheritance and Composition

Project Demo

