Immutability and Thread Safety

Lesson Objectives

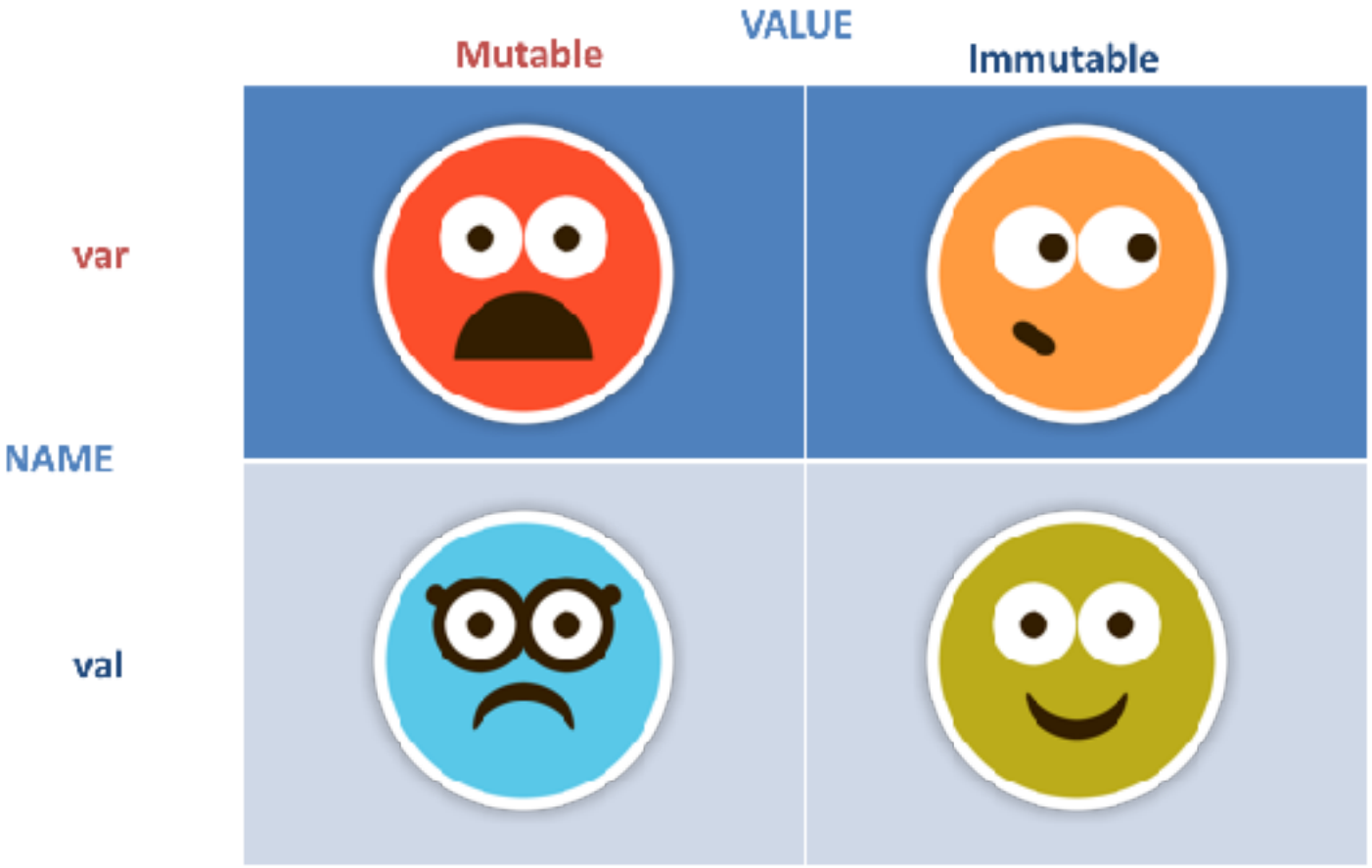
- After completing this lesson, you should be able to:
  - Understand basic thread safety in the JVM
  - Describe the importance of immutability in multithreaded applications
  - Outline how to use snapshots to preserve thread safety with case classes

What is Thread Safety

- The JVM has a well-defined memory model with specific guarantees
- There are two concerns:
  - **Synchronize-With:** Who is able to change state and in what order (locks)
  - **Happens-Before:** When to publish changes on one thread to all other threads (memory barriers)

Names versus Values

Typesafe



Typesafe

The Left Side of the Equals Sign

- Represents a pointer to the current value
- We want this to be “final” as much as possible, using a **`val`**
- Reassignment to a new value is possible when using a **`var`**

The Right Side of the Equals Sign

- Represents the value of the current state
- This should always be immutable, meaning that the class instance contains only fields that are defined as **`val`**
- If not, you must protect the state and who can change it using Mutually Exclusive Locks

Using a `var` for Snapshots

- Allows us to keep the value on the right side of the equals immutable, but still change our current state by replacing what the **`var`** points to with another instance
- The case class **`copy()`** method will help you do this

`@volatile`

- The `@volatile` annotation must be used when you follow the snapshot strategy, to ensure that all threads see your updates
- The case class `copy()` method will help you do this

@volatile

```
scala> case class Customer(firstName: String = "",
                           lastName: String = "")
defined class Customer

scala> @volatile var customer = Customer("Martin", "Odersky")
customer: Customer = Customer(Martin,Odersky)

scala> customer = customer.copy(lastName = "Doe")
customer: Customer = Customer(Martin,Doe)
```

Lesson Summary

- Having completing this lesson, you should be able to:
  - Understand basic thread safety in the JVM
  - Describe the importance of immutability in multithreaded applications
  - Outline how to use snapshots to preserve thread safety with case classes