Immutability

CIS-3030, Vermont Technical College
Peter C. Chapin

What is Immutability?

- A data object is *immutable* if once initialized it can not be changed (given a new value).
- What's good about immutability?
  - Immutable objects can't change unexpectedly.
    - Easier to reason about your program.
    - Fewer bugs.
  - Certain optimizations are easier.
  - Easier to use objects in a multi-threaded program
    - No need for locking since no thread can change object.

Immutability and FP

- In a pure functional language, all data objects are immutable
  - This gives functional programming unique flavor.
  - Enables the advantages.
  - Explains the strangeness.
- Oddnesses:
  - No variables (no “destructive update”)
  - No loops (can’t update loop control variable)
  - No “in place” modifications. Changes done by creating new objects instead.

Scala…

- … Is an OO (imperative) functional hybrid.
  - Supports variables and *mutable* objects in the usual sense
  - BUT… you are encouraged to create and use immutable objects and immutable references (vals) whenever you can

Just a Label

- A `val` is just a label attached to a value
  - Once bound, that label can not be used (in the same scope) to refer to a different value.
    - Some languages (F#) do allow rebinding of names.
- Compare
  - `(x + y) / (x - y)`
  - ```
    val numerator = x + y
    val denominator = x - y
    numerator / denominator
    ```
- Use `val` by default! Use immutability by default!
    ```

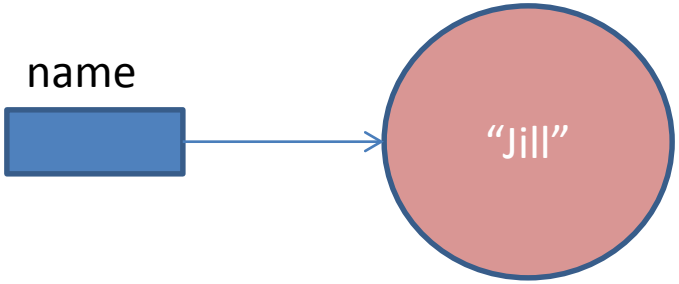
Visualization

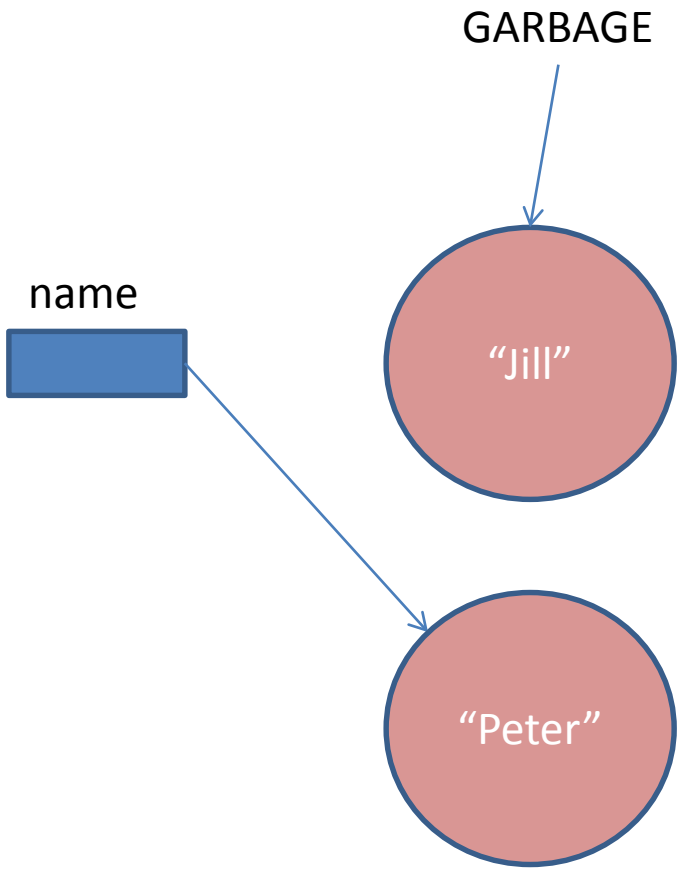
```
val name = "Jill"
```

Binding between the `val` and the object to which it refers can't be changed

Mutable References

```
var name = “Jill”
name = “Peter”
```





Object Mutability

- Objects can be mutable or immutable
  - Strings are immutable
    - Methods that “change” a string really return a new string with the changed value.
    - References to original string still see original value.

```
val name = “Jill”
val upperCaseName = name.toUpperCase

println(name)           // Prints “Jill”
println(upperCaseName) // prints “JILL”
```

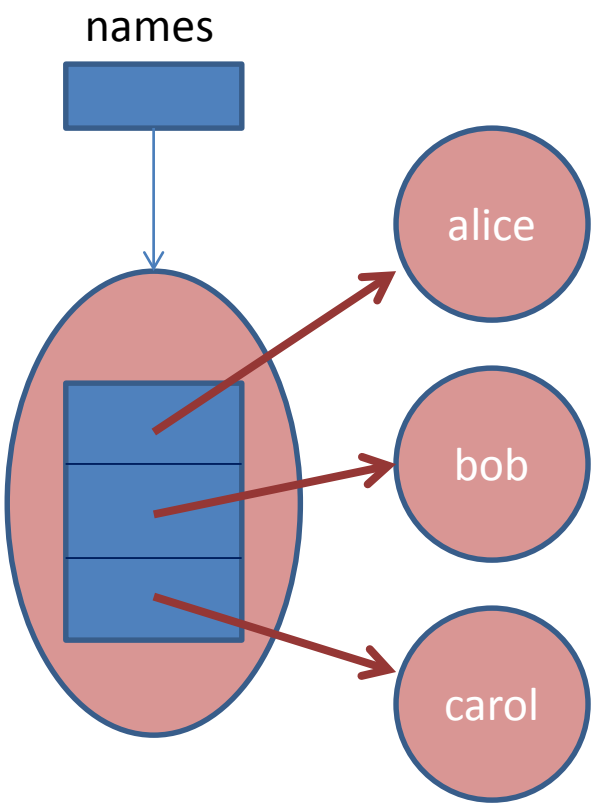
Arrays are Mutable

- Each array element can be modified in-place
  - Note: `val` below always refers to same array!
  - Note: individual String objects not modified!

```
val names = Array(“alice”, “bob”, “carol”)
names(0) = “dave”
for (name <- names) println(name)
  // Prints “dave”, “bob”, “carol”
```

Here's the Picture

BEFORE



AFTER