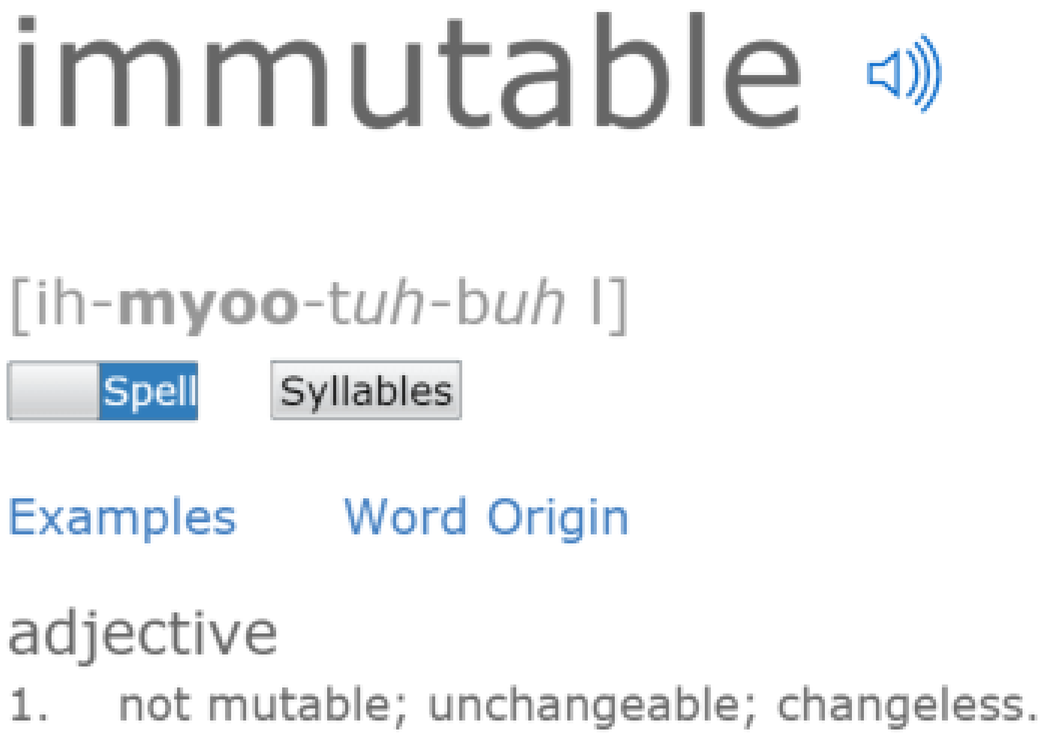
Java Programming 2
Immutable classes

Mary Ellen Foster

MaryEllen.Foster@glasgow.ac.uk

“Contemplating orange juice” by Hannah Webb
https://flic.kr/p/bkqXSy

“Immutable”?



immutable

[ih-**myoo**-tuh-buh l]

adjective

1. not mutable; unchangeable; changeless.

Immutability in Java

Immutable: internal state cannot change after it is constructed

Examples:

- `String`
- Wrapper classes: `Integer`, `Long`, `Character`, etc.
- `Stream`

Advantages of immutability

Immutable objects can be safely shared between data structures or threads

Can save memory:

- Two Strings with the same value are effectively identical …
- … so they can be mapped onto the same object at runtime

Ideal for lookup keys in “dictionary” structures

- Value will never change, so lookup is reliable

What does that mean in practice?

You always need to create new objects for new contents

- No possibility to change state, e.g., `setColour (RED)`

But isn’t it more expensive to create new objects all the time instead of reusing them?

- Yes (very, very slightly) …
- … but there are also efficiencies:
  - *Decreased garbage collection overhead*
  - *No need for code to protect objects from corruption*

String operations

Lots of constructors and static initialisers …

Lots of getters …

`charAt`, `indexOf`, `length`

Lots of methods to check the state

`contains`, `compareTo`, `equalsIgnoreCase`, `startsWith`

Other methods all **return a new string** – do not modify current string

`concat`, `toLowerCase`, `replace`, `trim`

What does this mean?

```
public void doStuff() {
    String s = "Hello world";
    // Doesn't actually change s at all
    s.toUpperCase();
    // s2 now contains "HELLO WORLD"
    String s2 = s.toUpperCase();
}
```

Creating an immutable class

Instance fields:

- Must be `private` and `final`
- Must have getters but no setters

Constructor:

- Must set complete internal state of object

Methods:

- Don’t allow overriding
  - *Easy: declare class* `final`
  - *Fancy: make constructor* `private` *and use static factory methods to create instances*

Creating an immutable class (2)

- If instance fields can be mutable objects, don’t let them be changed
  - Don’t provide methods to modify them
  - Don’t return the mutable objects directly from getters; return copies instead

Immutable class example

BEFORE

```
public class Person {

    private List<String> names;

    public Person(String[] names) {
        this.names =
                Arrays.asList(names);
    }

    public List<String> getNames() {
        return names;
    }

}
```

AFTER

```
public final class Person {

    private final List<String> names;


    public Person(String[] names) {

        this.names = new
ArrayList<>(Arrays.asList(names));

    }


    public List<String> getNames() {

        return new ArrayList<>(names);

    }

}
```