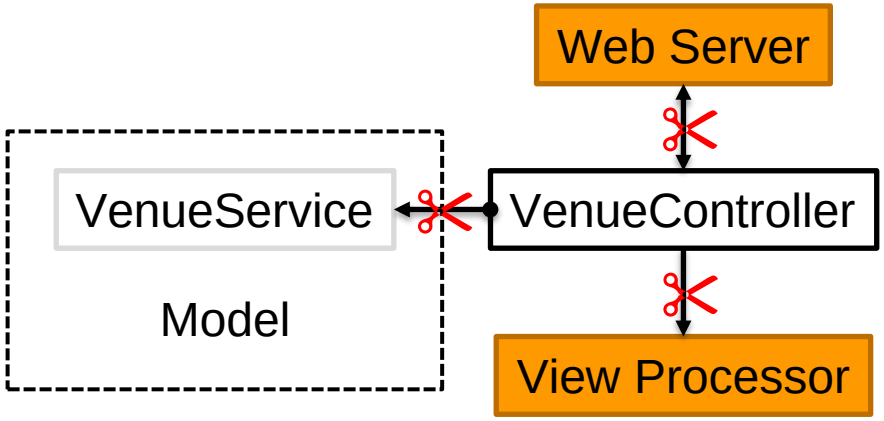
Testing Functionality in Isolation

Why isolate everything?



- Dummy
  - A dummy is passed around but never used
- Fake
  - A fake generally works as expected, but has some shortcut unsuitable for full production
- Stub
  - A stub provides a canned answer to a particular invocation
- Mock
  - A mock has pre-programmed expectations of how it will be called, and what will happen internally when it is called

Stub example

Main program loop

```
if (outside.readTemperature() <= 0) {
    System.out.println("Freezing!");
}
```

Stubbed method

```
public int readTemperature(void) {
    return(0);
}
```

Using a mock to stub methods

```
@Mock
private Event event;

@Mock
private Venue venue;

@MockBean
private EventService eventService;

@MockBean
private VenueService venueService;
```

Using a mock to stub methods

```
when(<condition>).then<do something>

when(event.getName()).thenReturn(“Hello”);

when(venueService.findAll())
        .thenThrow(new Exception());

when(event.getName())
        .thenThrow(new Exception())
        .thenReturn(“Hello”);
```

Adding behaviour verification to a mock

- Exactly once:

```
verify(venueService).delete(1L)
verify(venueService, times(1)).delete(1L)
```

- Exactly twice:

```
verify(venueService, times(2)).delete(1L)
```

- At most five times:

```
verify(venueService, atMost(5)).delete(1L)
```

- Never:

```
verify(venueService, never()).delete(1L)
```

Mocking example

"A venue cannot be deleted if it has one or more events."

- Return an empty list of events from our mocked venue:

```
when(venue.getEvents())
    .thenReturn(Collections.<Event> emptyList());
```

- Return that venue instance from our mocked DAO:

```
when(venueService.findOne(1L)).thenReturn(venue);
```

- Do the operation:

```
mvc.perform(delete("/venues/1")
                    .accept(MediaType.TEXT_HTML))
                    .andExpect(status().is(302));
```

- Verify the internal behaviour of VenueService:

```
verify(venueService).delete(1L);
```