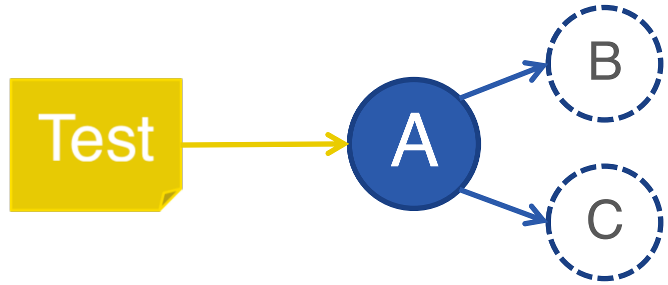
Unit Testing

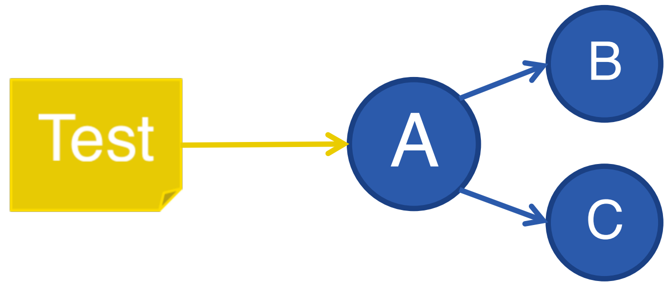
How to write better unit tests

ARTEM TABALIN

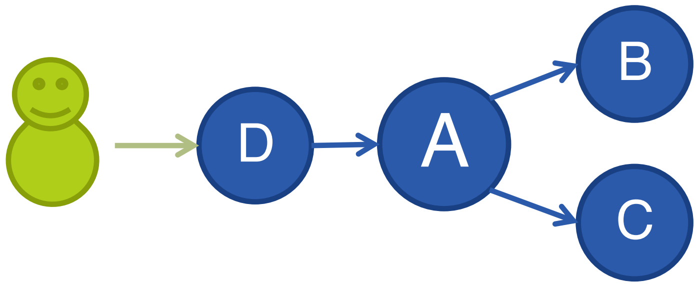
Tests Classification



Unit Tests



Integration Tests



Functional Tests (e2e)

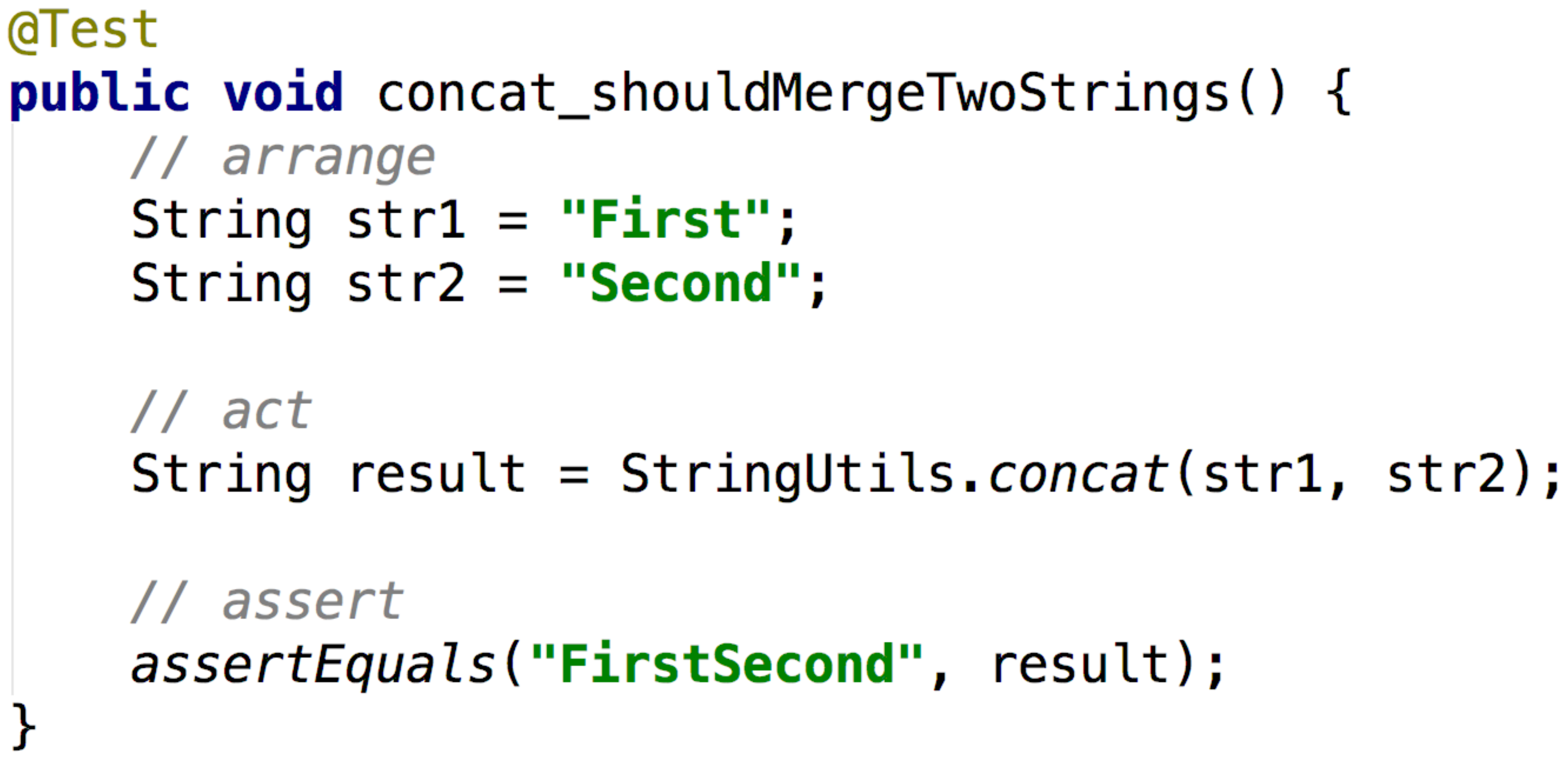
Why Unit Testing?

- Assurance of correctness
- Eliminates risk of changes
- Ensures better design
- Provides documentation
- Reduce debug time
- Makes to write loosely coupled code
- Gives confidence

Unit Tests Best Practices

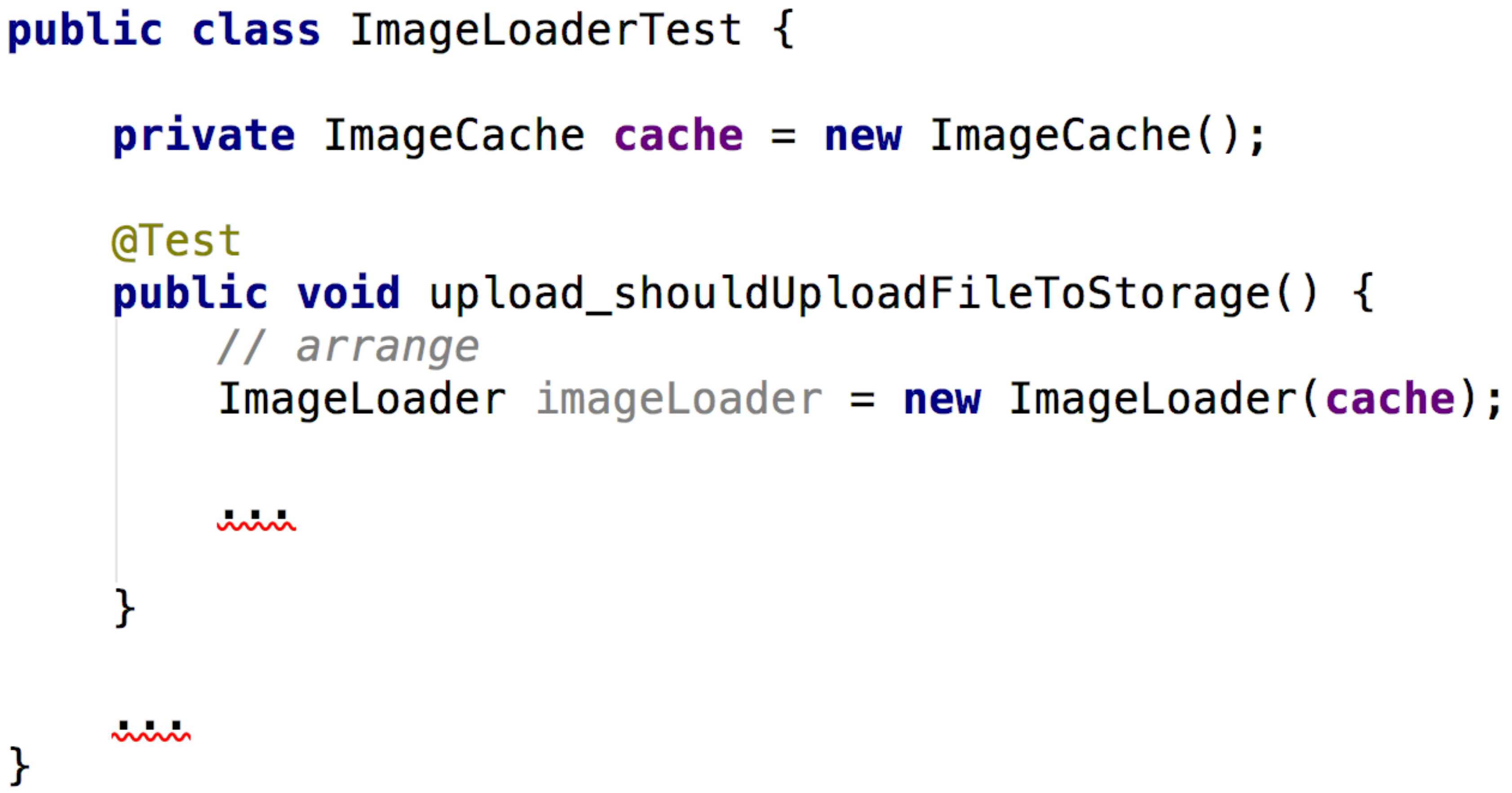
- Fast as possible
- Test first
- Arrange → Act → Assert
- Assert first
- Tests isolation
- Test doubles

Arrange → Act → Assert



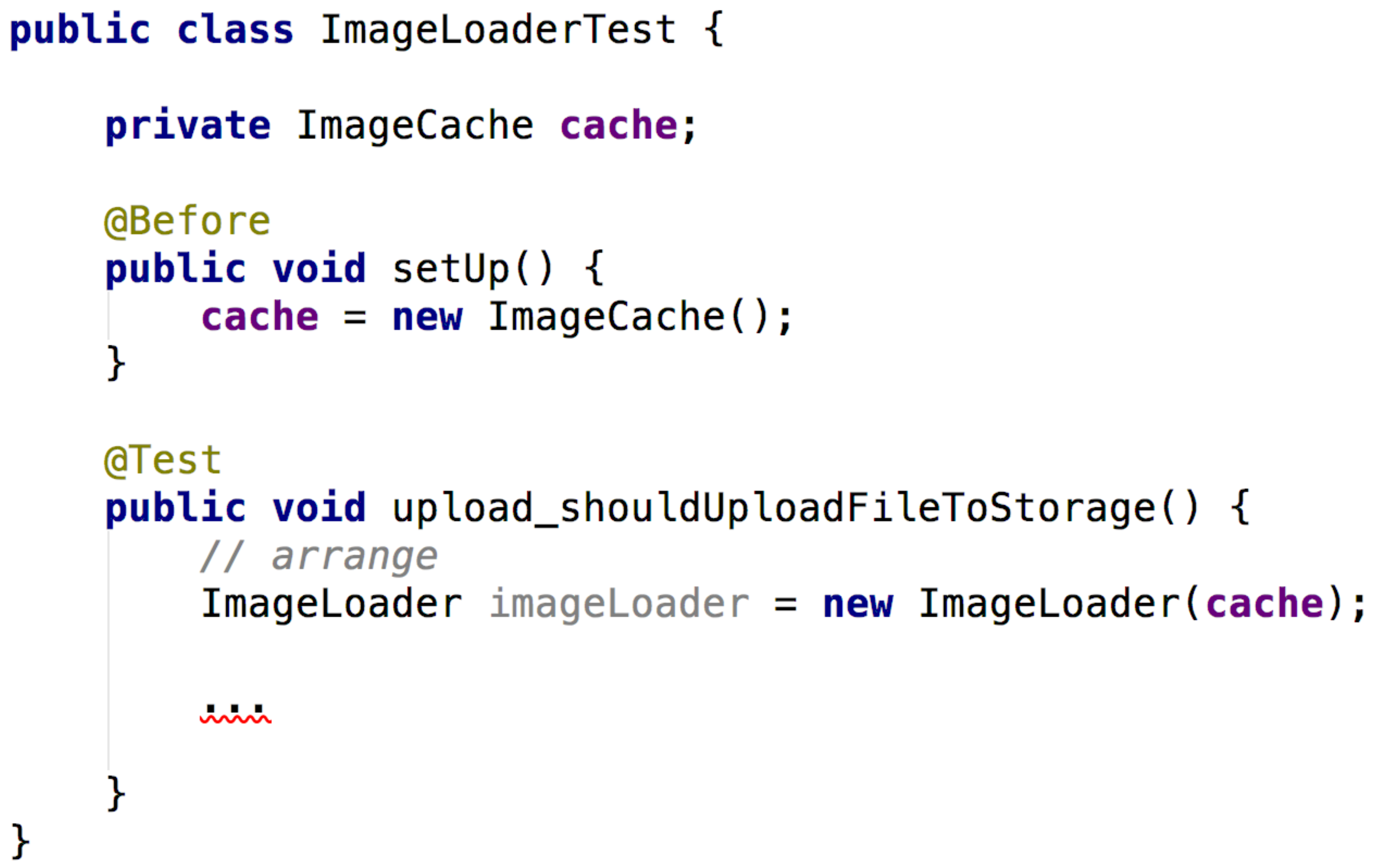
```java
@Test
public void concat_shouldMergeTwoStrings() {
    // arrange
    String str1 = "First";
    String str2 = "Second";

    // act
    String result = StringUtils.concat(str1, str2);

    // assert
    assertEquals("FirstSecond", result);
}
```

Test Isolation



```java
public class ImageLoaderTest {

    private ImageCache cache = new ImageCache();

    @Test
    public void upload_shouldUploadFileToStorage() {
        // arrange
        ImageLoader imageLoader = new ImageLoader(cache);

        ...

    }

    ...
}
```

Test Isolation



```
public class ImageLoaderTest {

    private ImageCache cache;

    @Before
    public void setUp() {
        cache = new ImageCache();
    }

    @Test
    public void upload_shouldUploadFileToStorage() {
        // arrange
        ImageLoader imageLoader = new ImageLoader(cache);


        ...

    }
}
```

Test Doubles

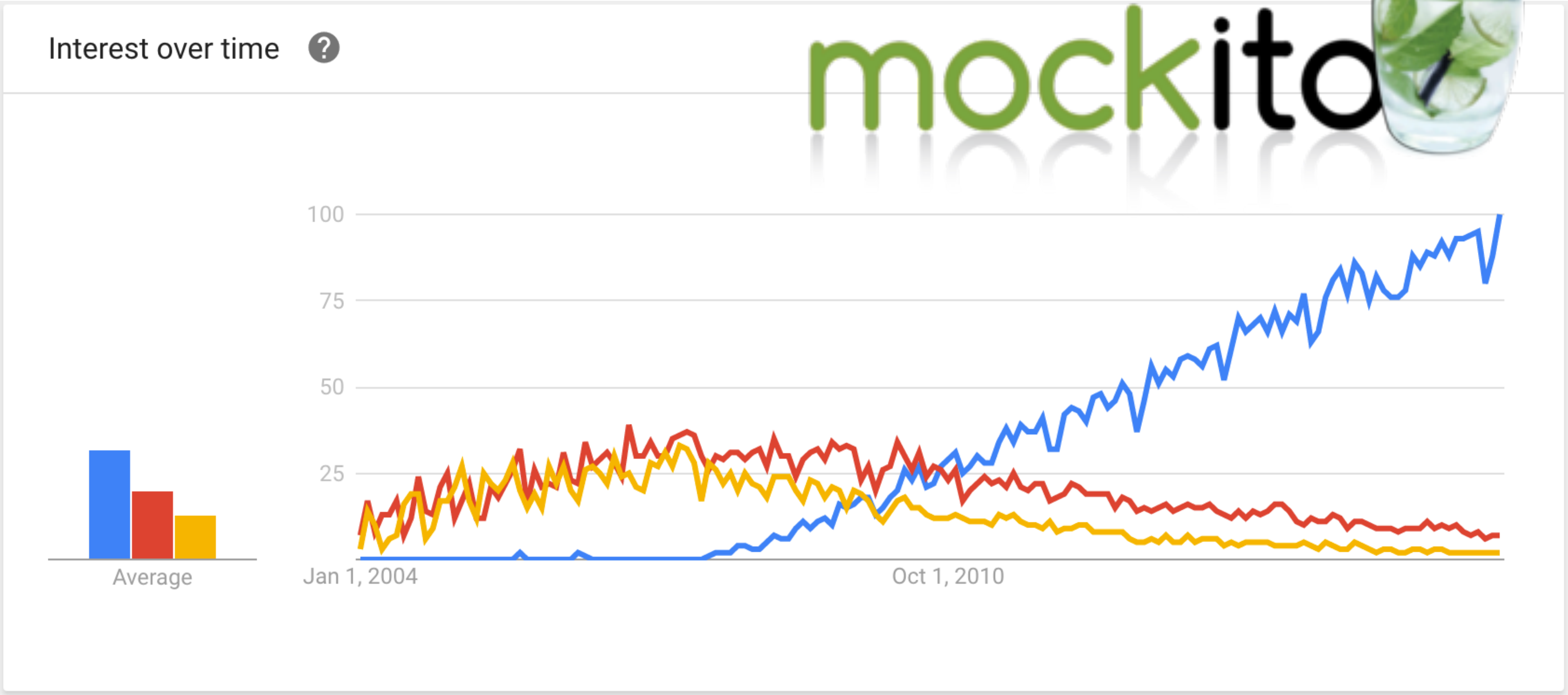
Dummy

Stub **Mock** Spy

Fake

- Isolate testing component
- Test integration (not only state)

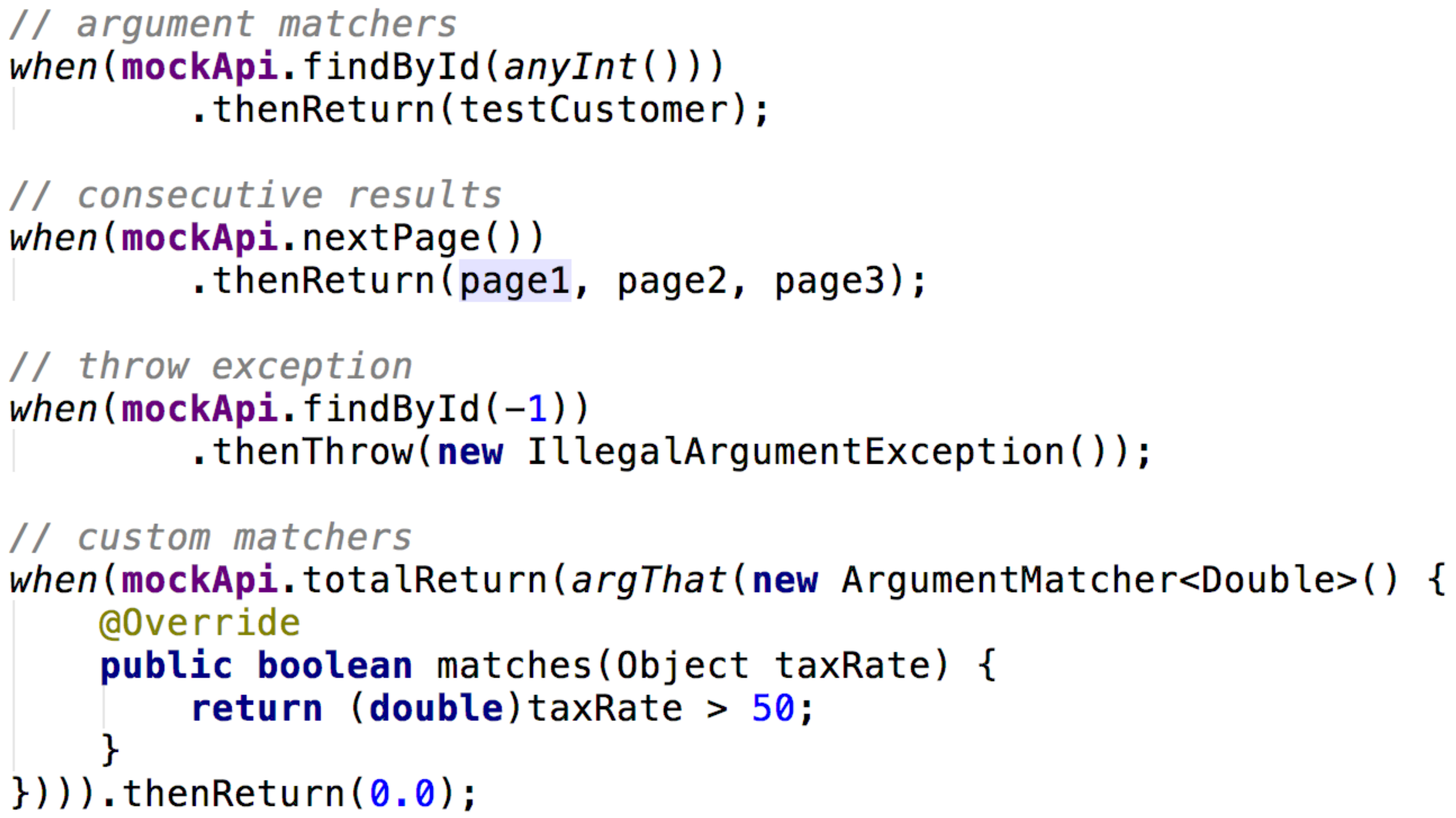
Mockito


Interest over time
100
75
50
25
Average
Jan 1, 2004
Oct 1, 2010
mockito

Mockito – Example

```java
public class CustomerServiceTest {
    private CustomerService service;
    private CustomerDAO mockDAO;

    @Before
    public void setUp() {
        mockDAO = Mockito.mock(CustomerDAO.class);
        service = new CustomerService(mockDAO);
    }

    @Test
    public void findCustomer_shouldReadCustomerFromDAO() {
        // arrange
        Customer testCustomer = new Customer(1, "Test Customer");
        when(mockDAO.findById(1)).thenReturn(testCustomer);

        // act
        Customer resultCustomer = service.findCustomer(1);

        // arrange
        assertEquals(testCustomer, resultCustomer);
        verify(mockDAO).findById(1);
    }
}
```

Mockito - Mocking

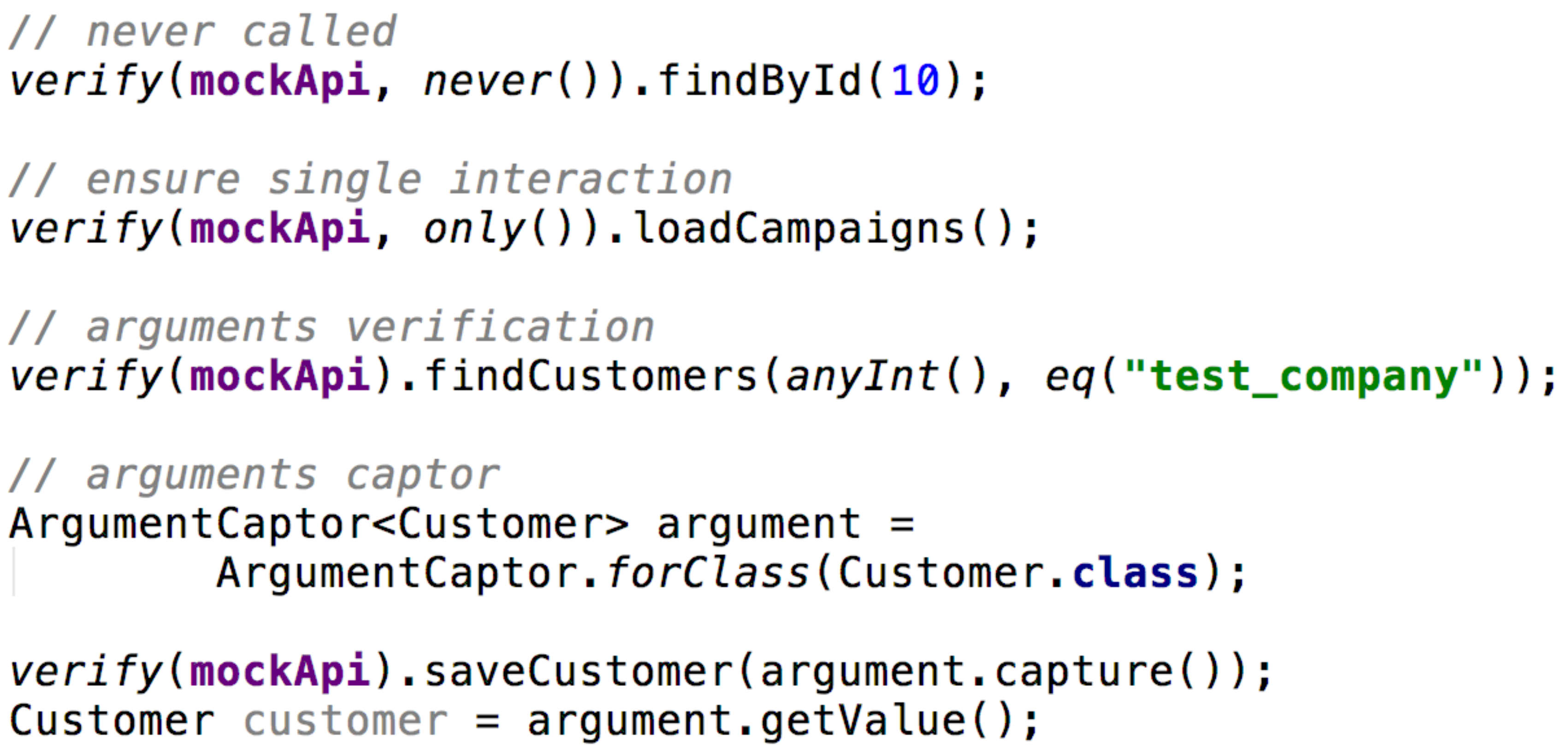


```java
// argument matchers
when(mockApi.findById(anyInt()))
        .thenReturn(testCustomer);

// consecutive results
when(mockApi.nextPage())
        .thenReturn(page1, page2, page3);

// throw exception
when(mockApi.findById(-1))
        .thenThrow(new IllegalArgumentException());

// custom matchers
when(mockApi.totalReturn(argThat(new ArgumentMatcher<Double>() {
    @Override
    public boolean matches(Object taxRate) {
        return (double)taxRate > 50;
    }
}))).thenReturn(0.0);
```

Mockito - Mocking

```java
// throw for void methods
doThrow(new IllegalArgumentException())
        .when(mockApi).deleteCustomer(0);

// call real implementation
doCallRealMethod().when(mockApi).resetCache();

// ignore method
doNothing().when(mockApi).preloadCache();

// custom answer
doAnswer(new Answer<Object>() {
    @Override
    public Object answer(InvocationOnMock invocation)
            throws Throwable {
        Customer customer = (Customer)invocation.getArguments()[0];
        customer.isVerified = true;
        return customer;
    }
}).when(mockApi).verifyCustomer(any(Customer.class));
```

Mockito - Verifying



```java
// never called
verify(mockApi, never()).findById(10);

// ensure single interaction
verify(mockApi, only()).loadCampaigns();

// arguments verification
verify(mockApi).findCustomers(anyInt(), eq("test_company"));

// arguments captor
ArgumentCaptor<Customer> argument =
        ArgumentCaptor.forClass(Customer.class);

verify(mockApi).saveCustomer(argument.capture());
Customer customer = argument.getValue();
```

Mockito Limitations

- Final classes & methods
- Static methods
- Private methods
- Enums
- Primitive types
- Anonymous classes

Thank you!