

I have been building  
pyramid until I started  
mining diamonds



**Jan Koszela**

# TL;DR

1. `System.out.println(„Hello World!!“);`

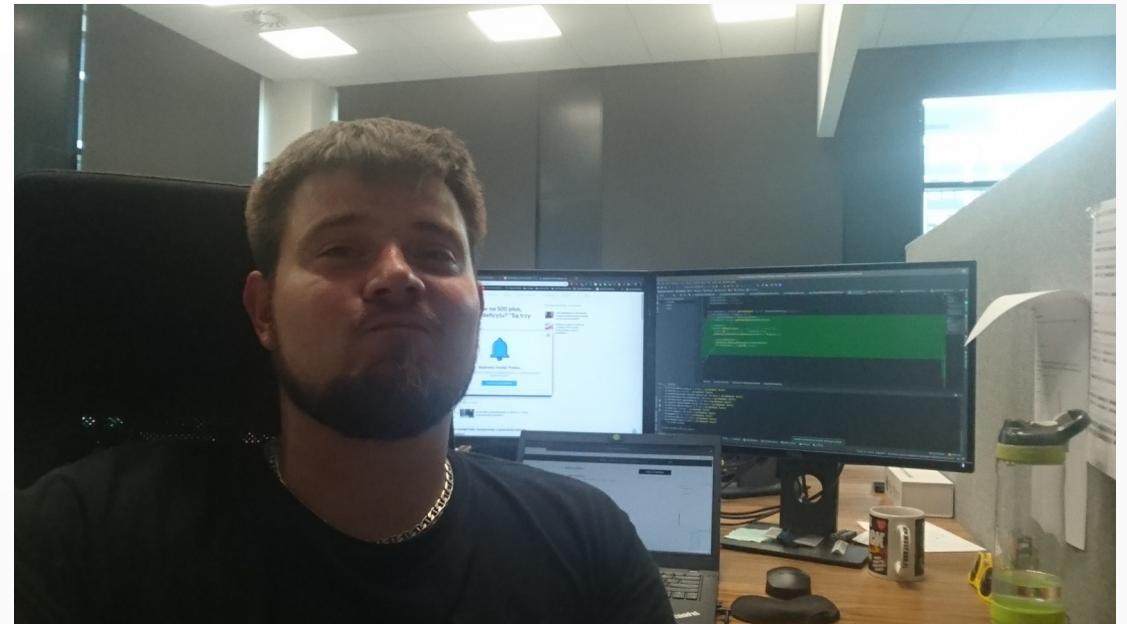
```
System.out.println(„Hello World!!“);
```

**PRINT "HELLO WORLD" FOR THE  
FIRST TIME FEELS**



# System.out.println(„Hello World!!“);

- Jan Koszela
- Eclipse Poland Limited at Wrocław
- Java Developer / Analyst
- 4.5 y. of comm. exp. based on Java web stack,
- automotive, e-commerce
- [linkedin.com/in/jan-koszela-6bb38583](https://www.linkedin.com/in/jan-koszela-6bb38583)



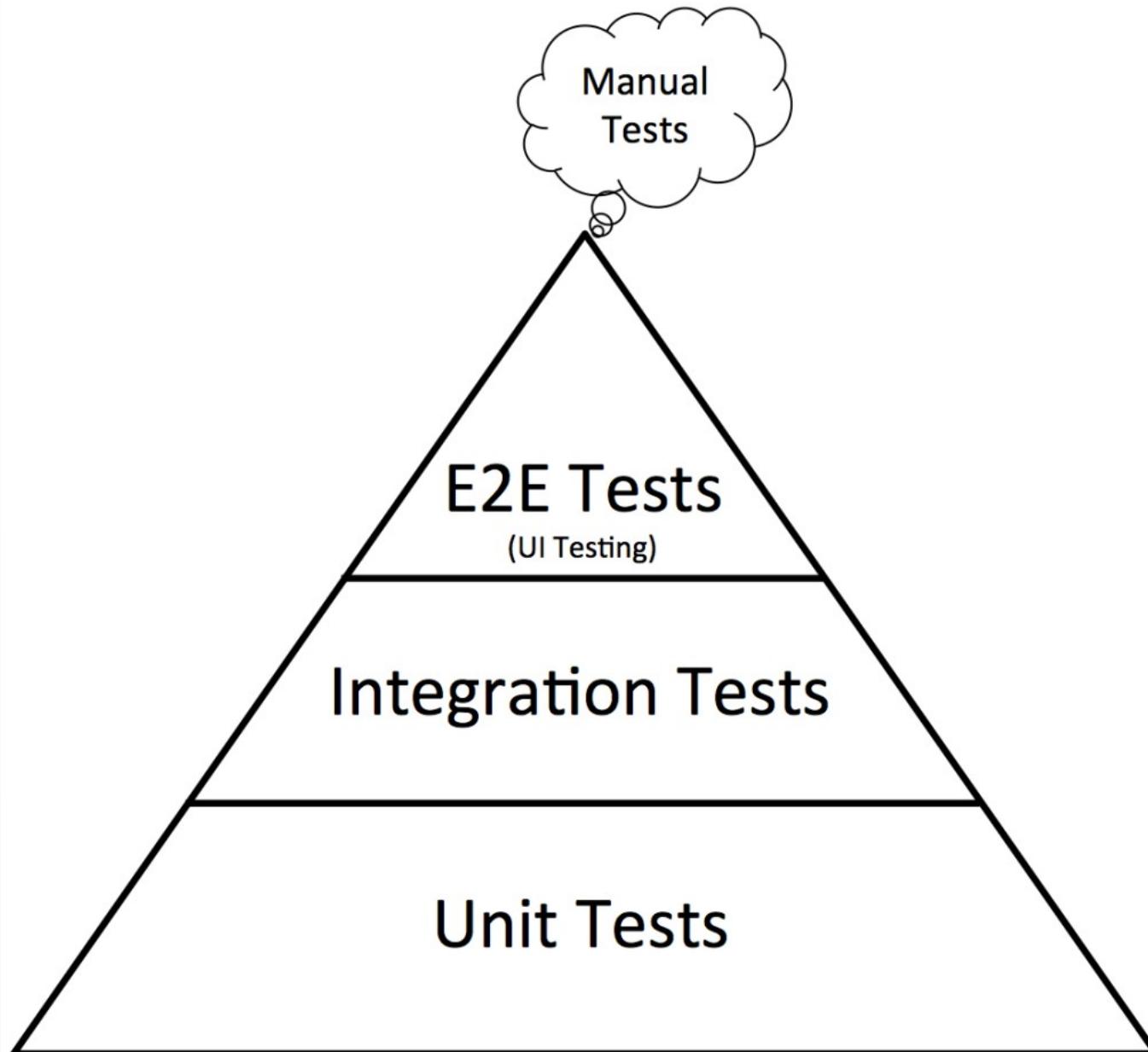
# TL;DR

1. `System.out.println(„Hello World!!“);`
2. Software testing pyramid

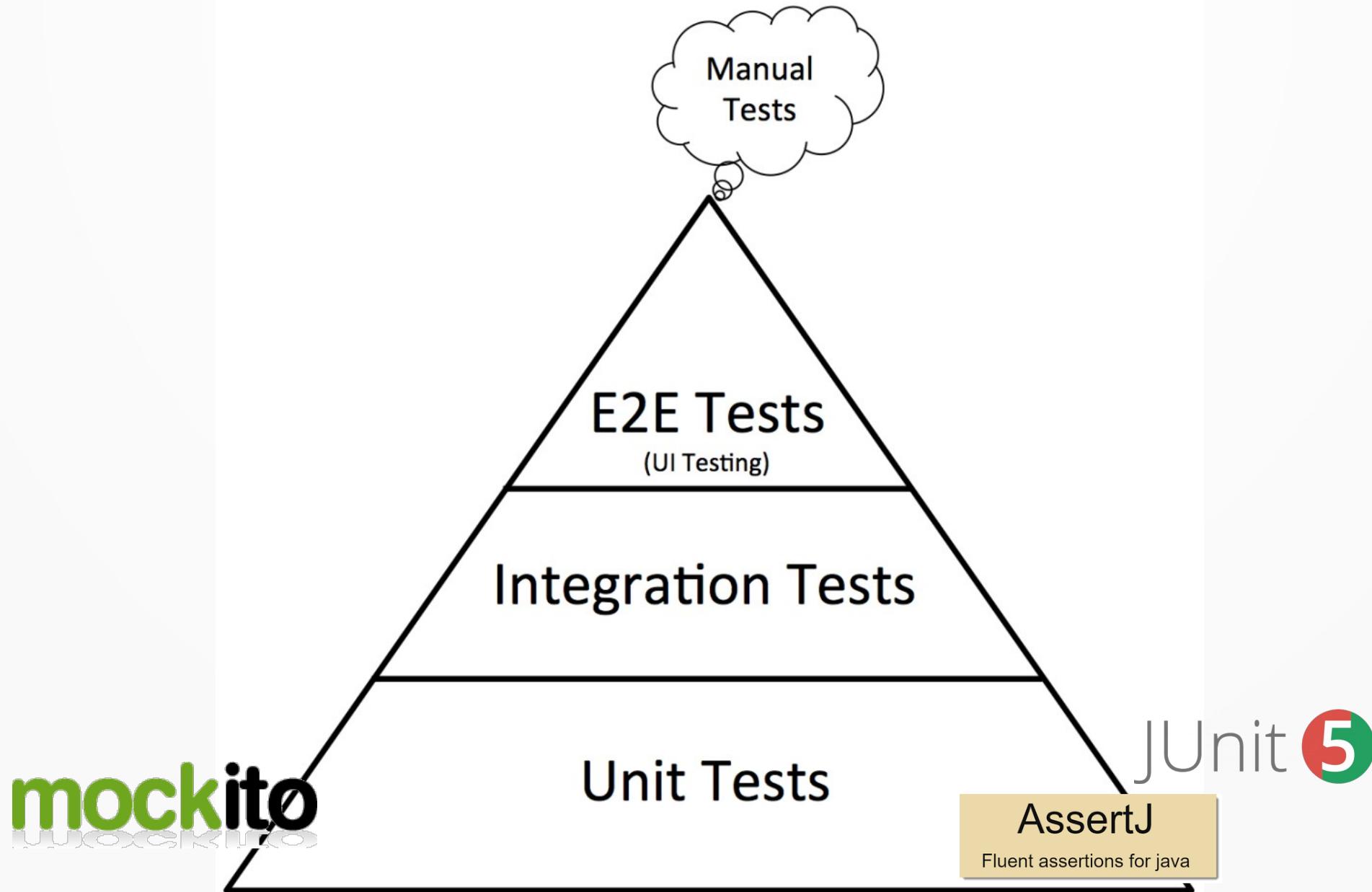
# Software testing Pyramid



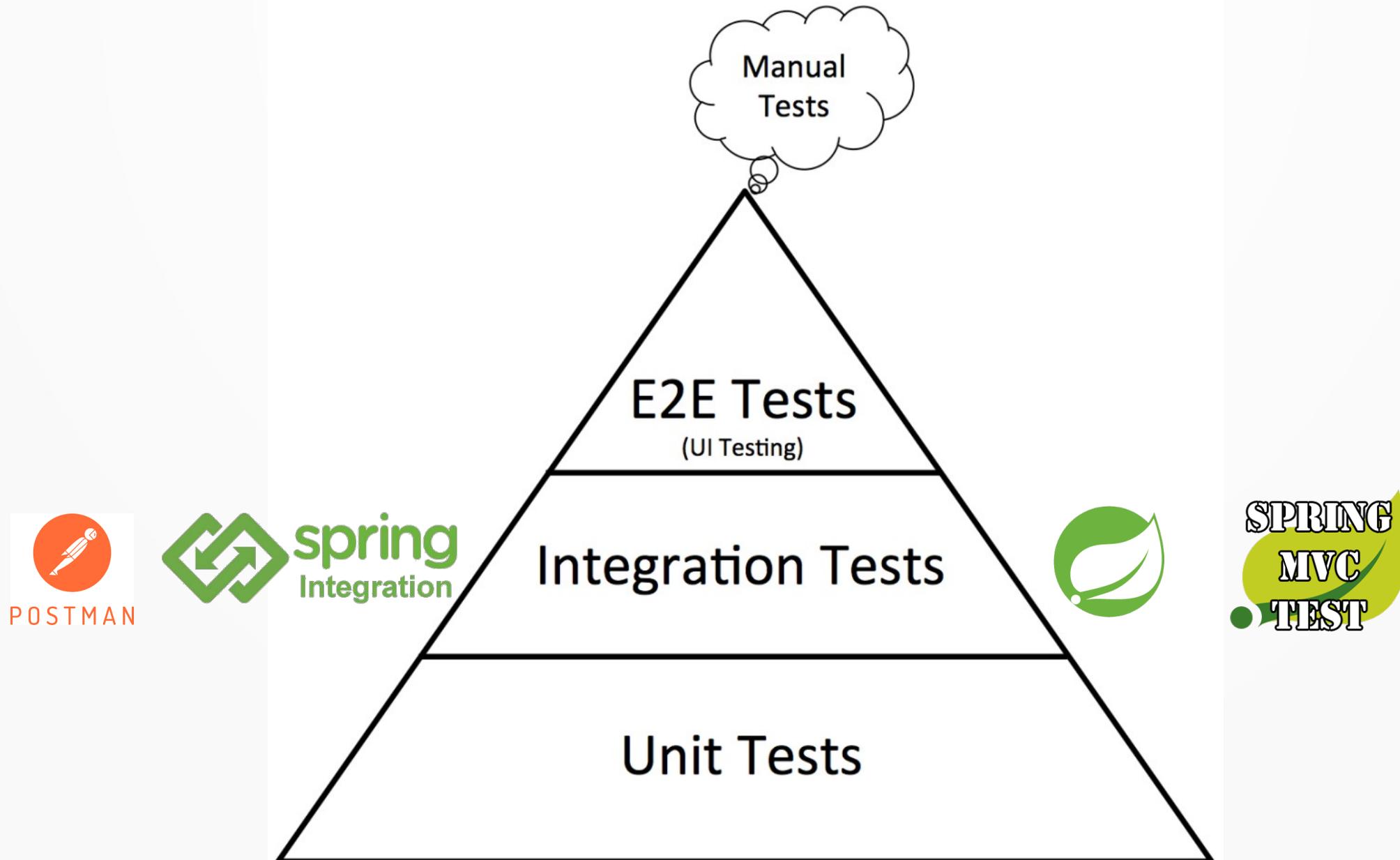
# Software testing Pyramid



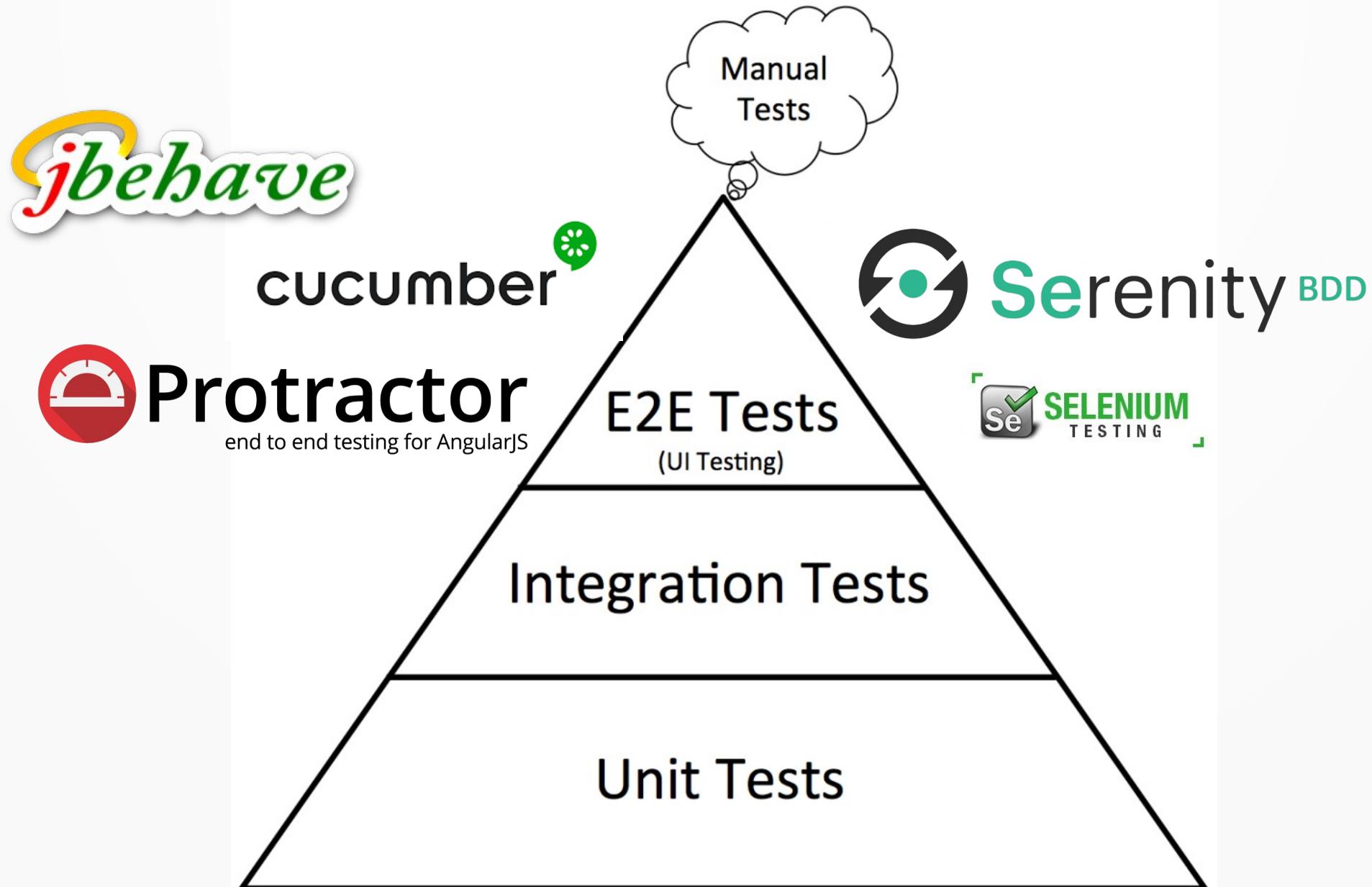
# Software testing Pyramid



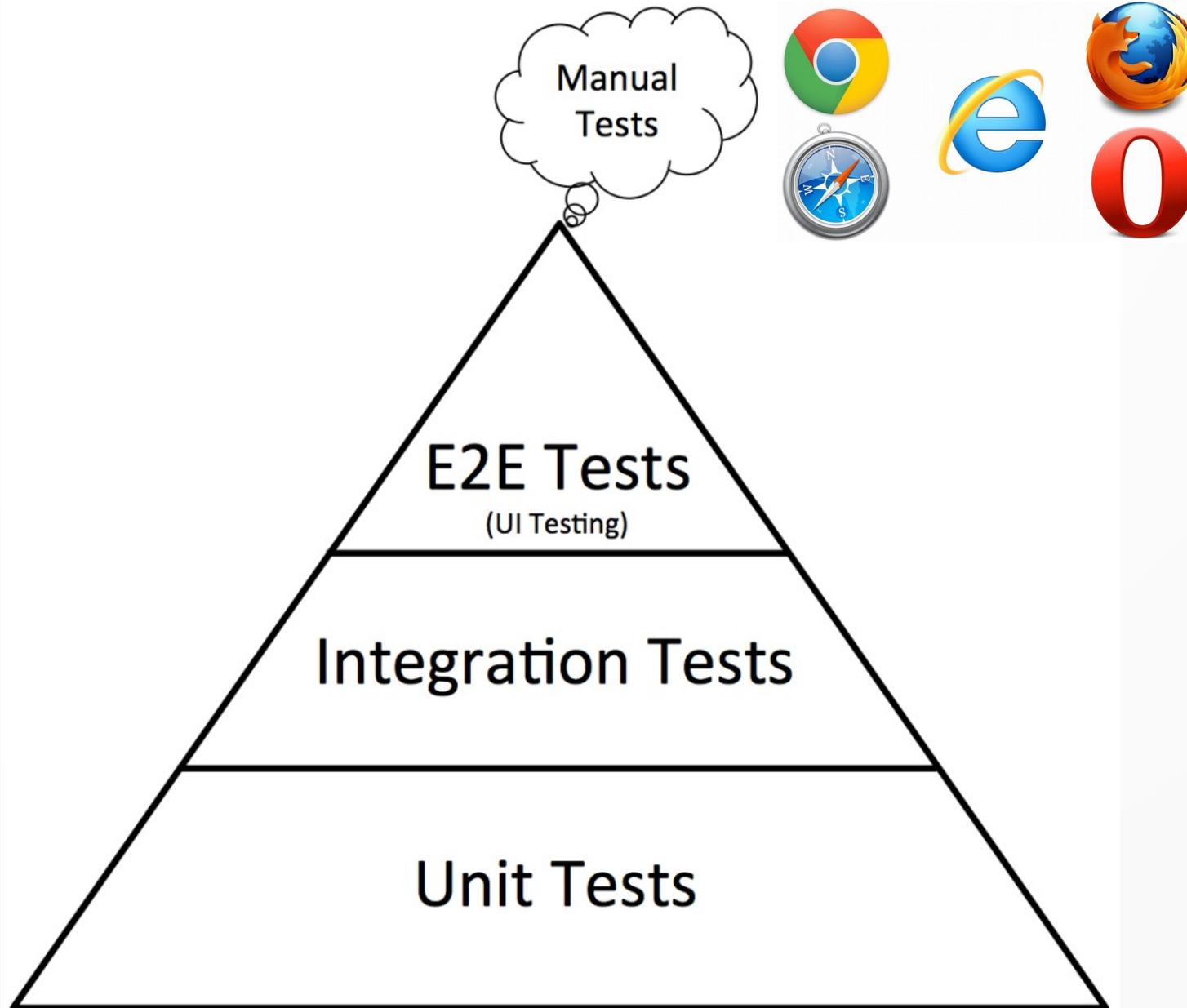
# Software testing Pyramid



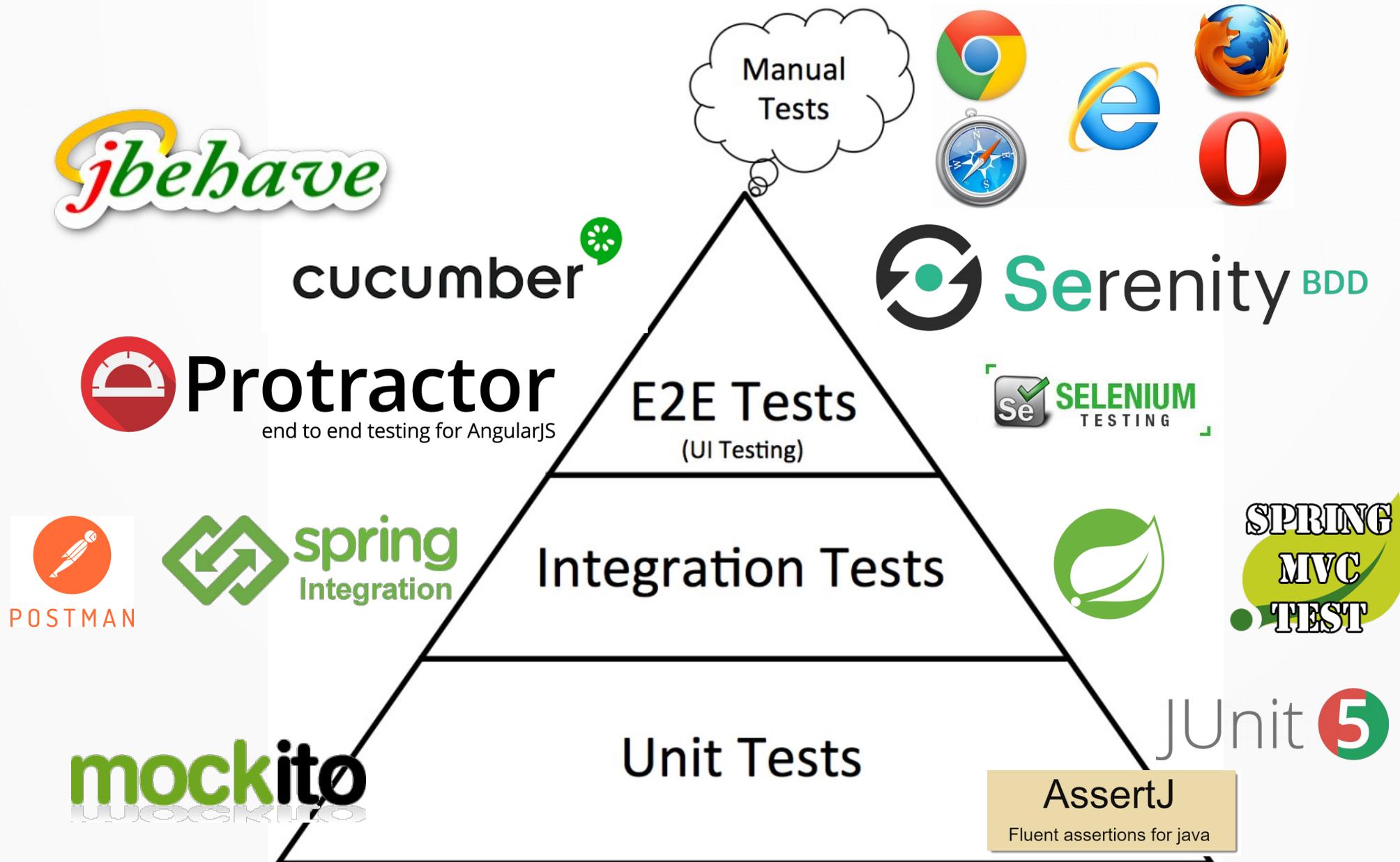
# Software testing Pyramid



# Software testing Pyramid



# Software testing Pyramid



# TL;DR

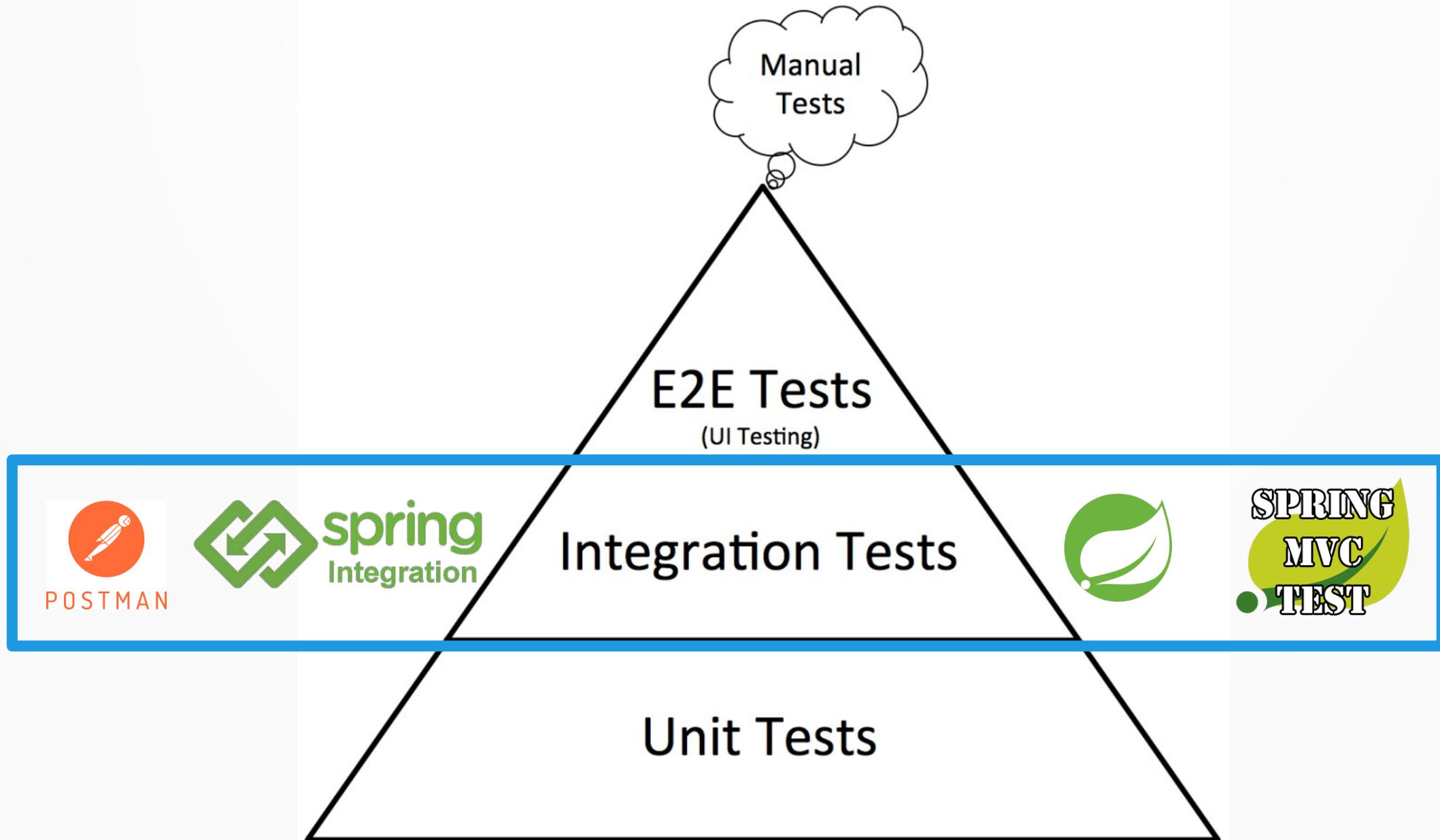
1. `System.out.println(„Hello World!!“);`
2. Software testing pyramid
3. Integration tests

# Integration tests



unit tests passing - no  
integration tests

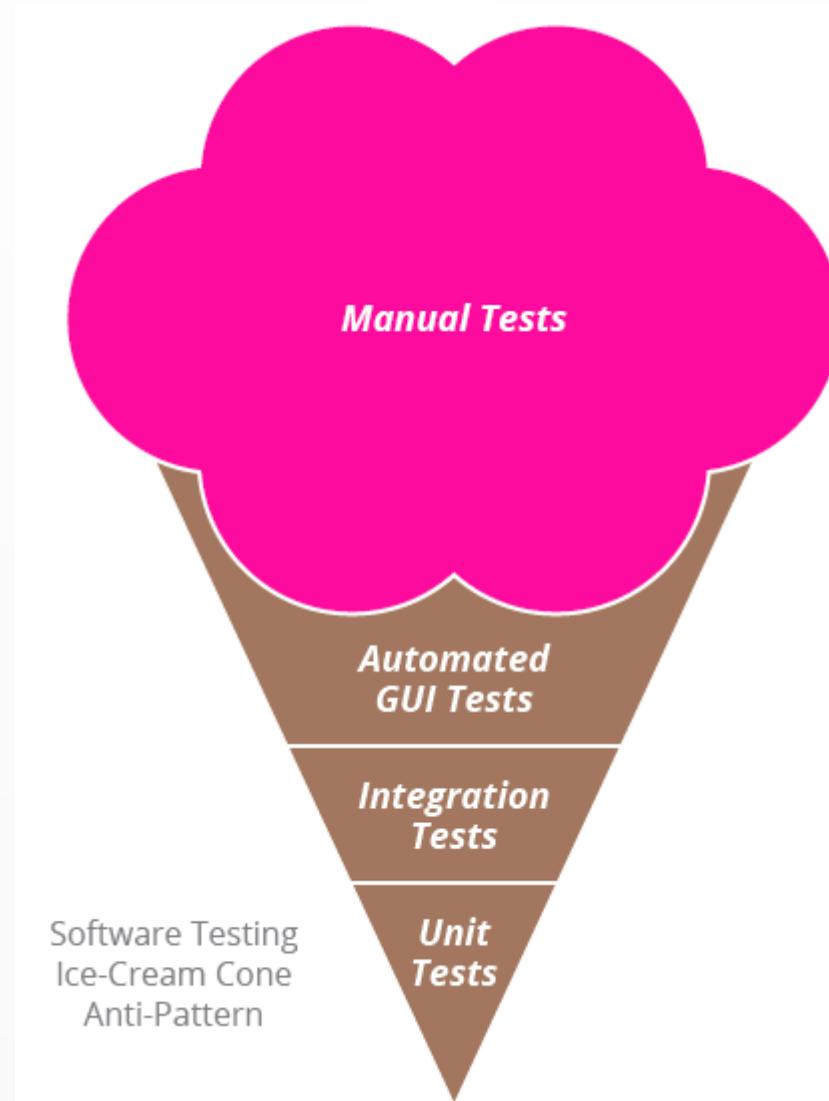
# Integration tests



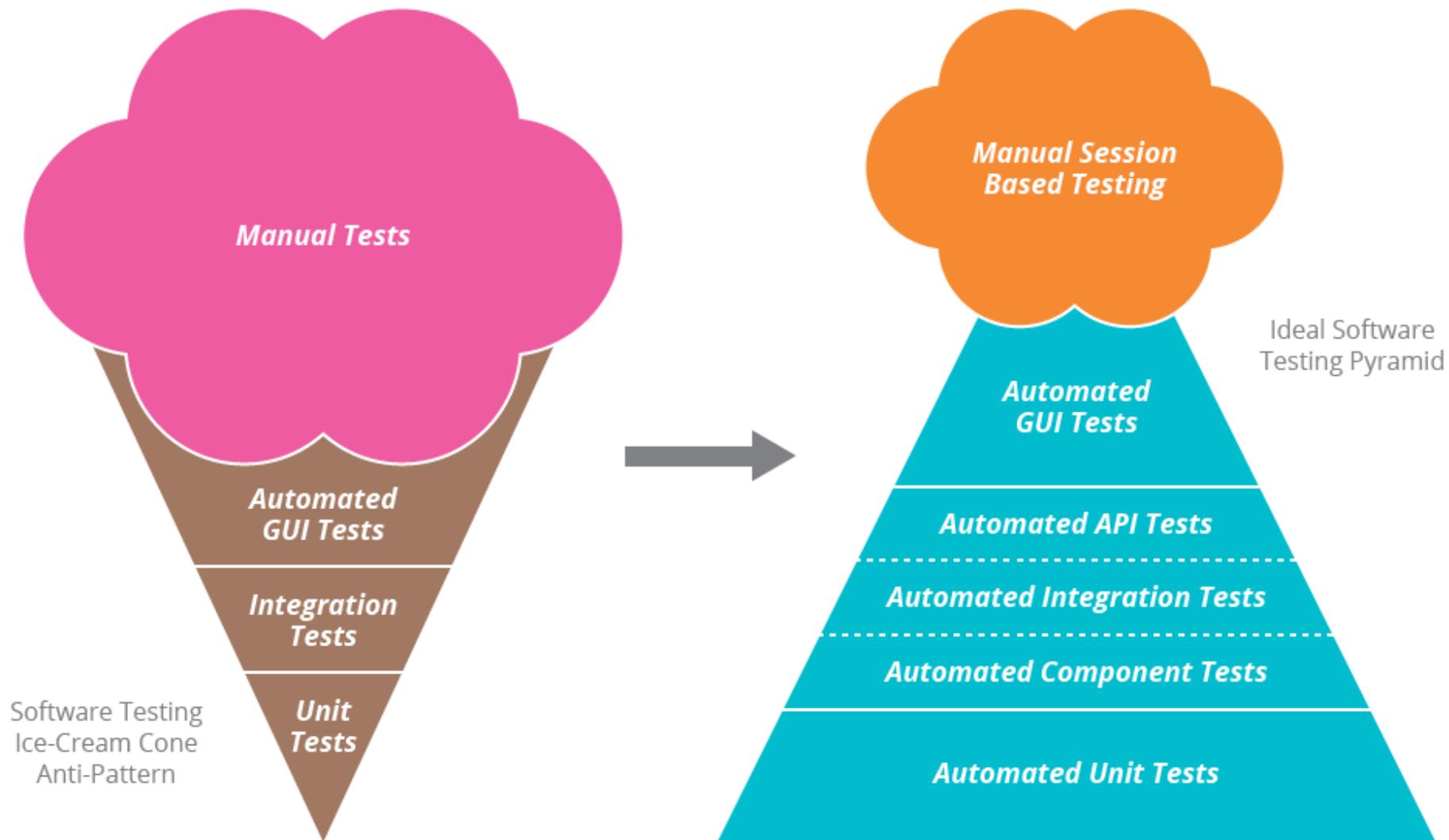
# Integration tests



# Integration tests



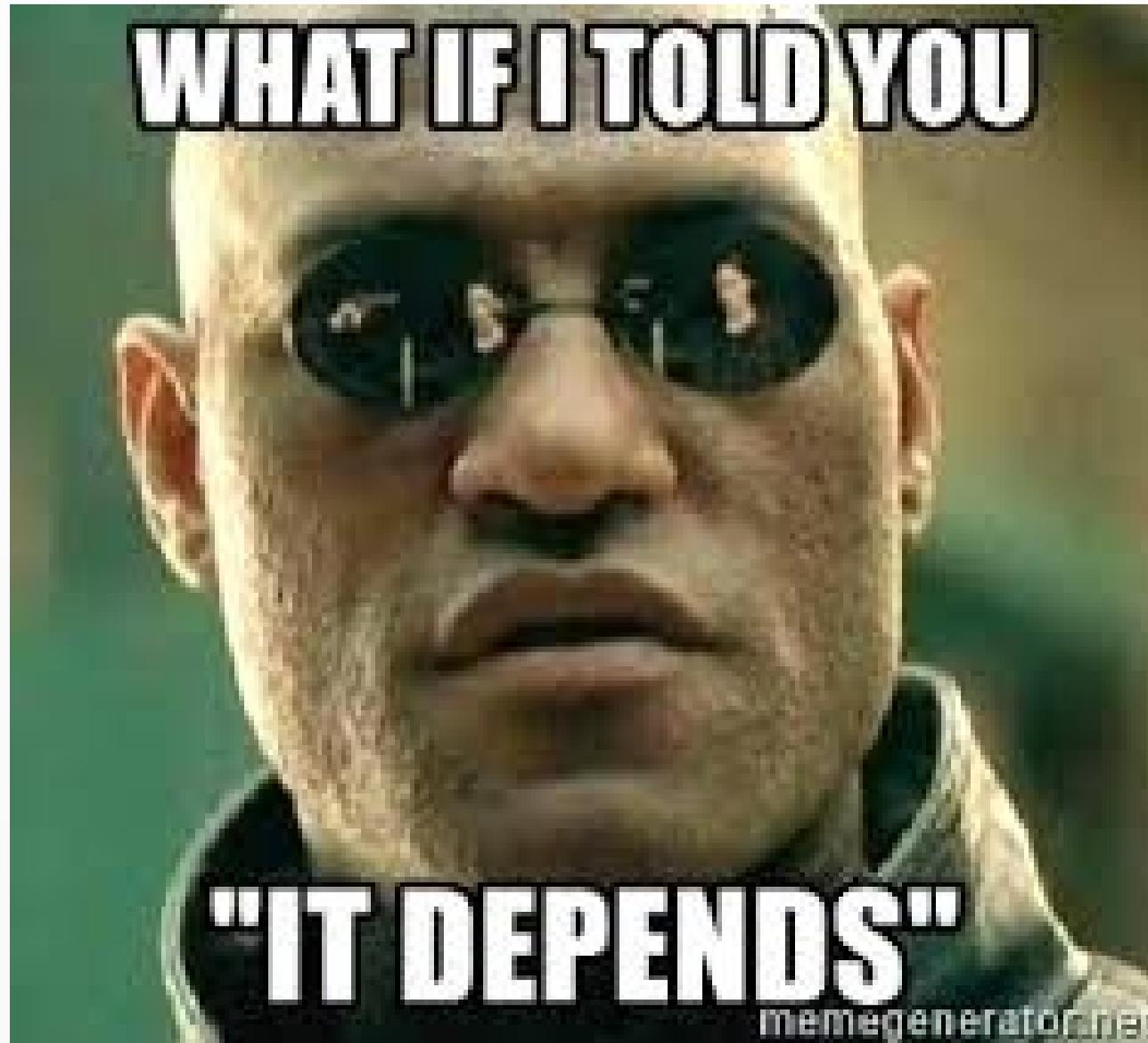
# Integration tests



# Question 1

**Is the inverted test pyramid  
really anti-pattern?**

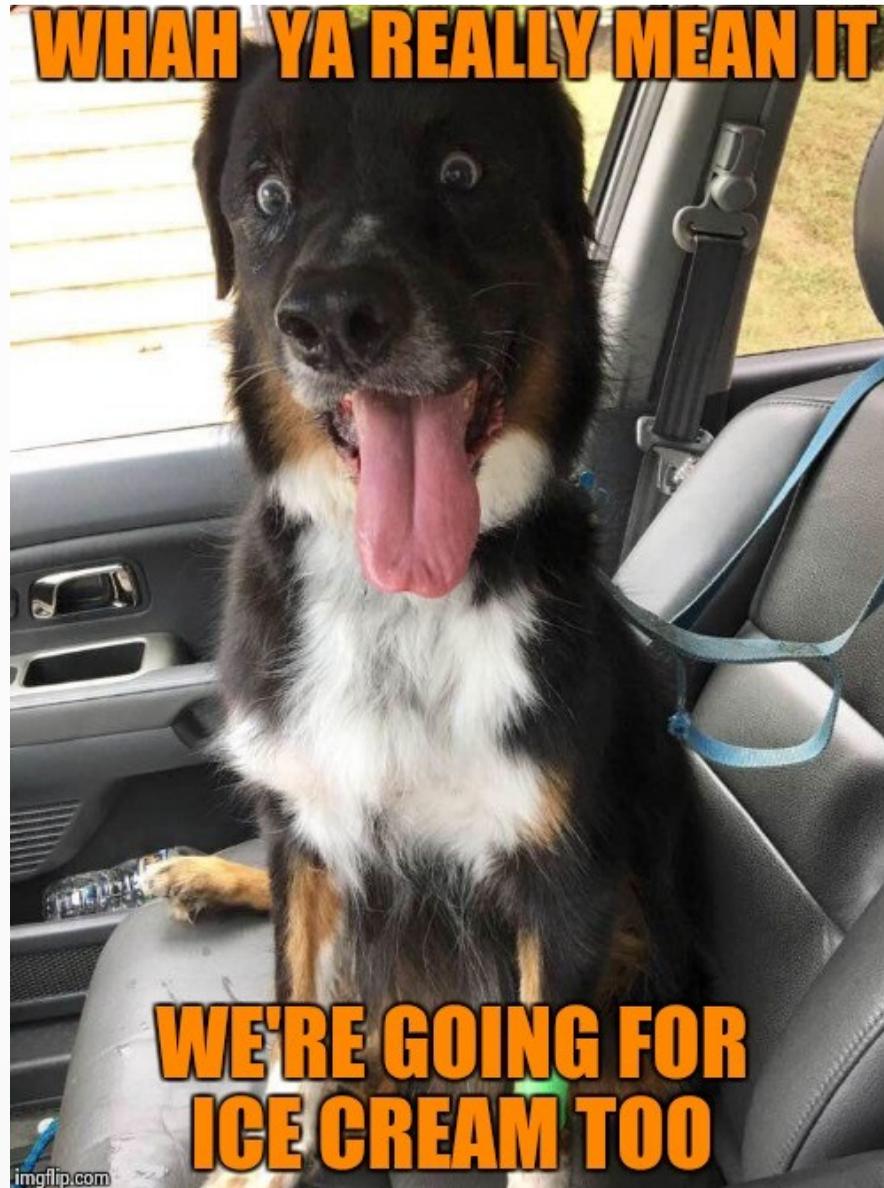
# Answer



# TL;DR

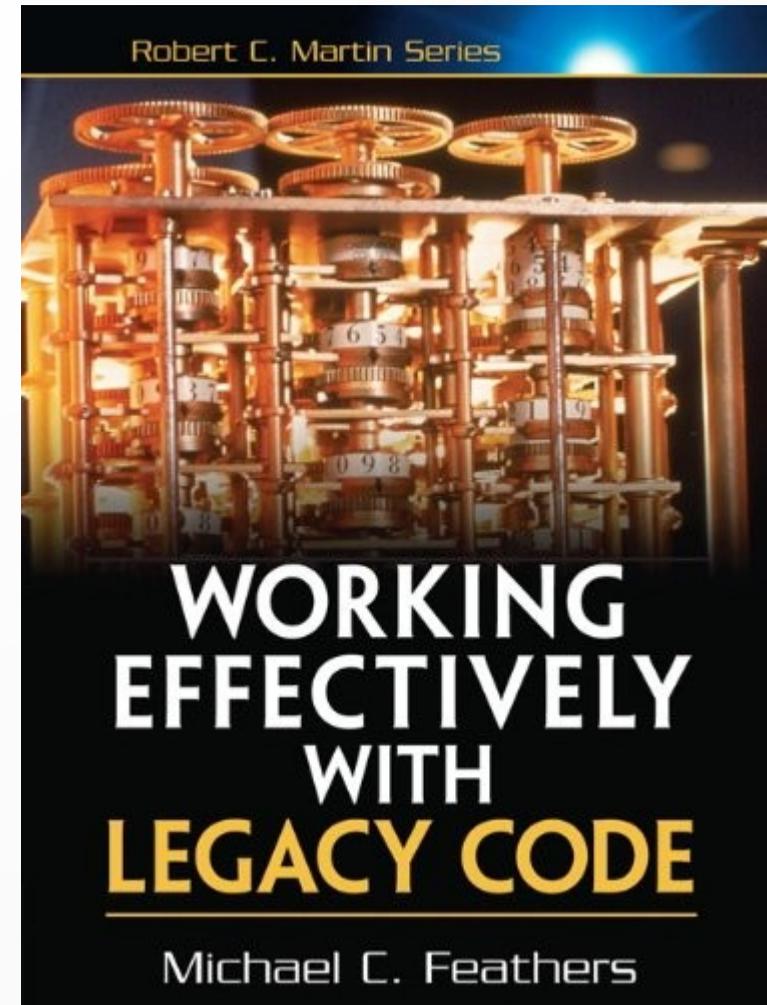
1. `System.out.println(„Hello World!!“);`
2. Software testing pyramid
3. Integration tests
4. Ice-cream cone „pattern“ cases

# Ice-cream cone „pattern” cases



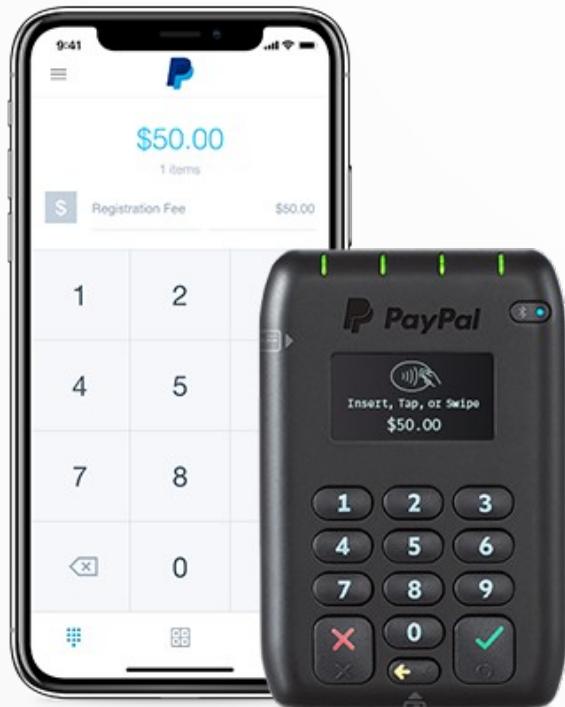
# Ice-cream cone „pattern” cases

**You are working with  
legacy code**



# Ice-cream cone „pattern” cases

**You have an integration to a specialized physical device**



# Ice-cream cone „pattern” cases

**You have an integration to a third party API**



**EXTERNAL APIs**

- Adoption
- New Business

**PARTNER APIs**

- Integration
- Business Development

**INTERNAL APIs**

- Mobile Apps
- LOB Apps

# Question 2



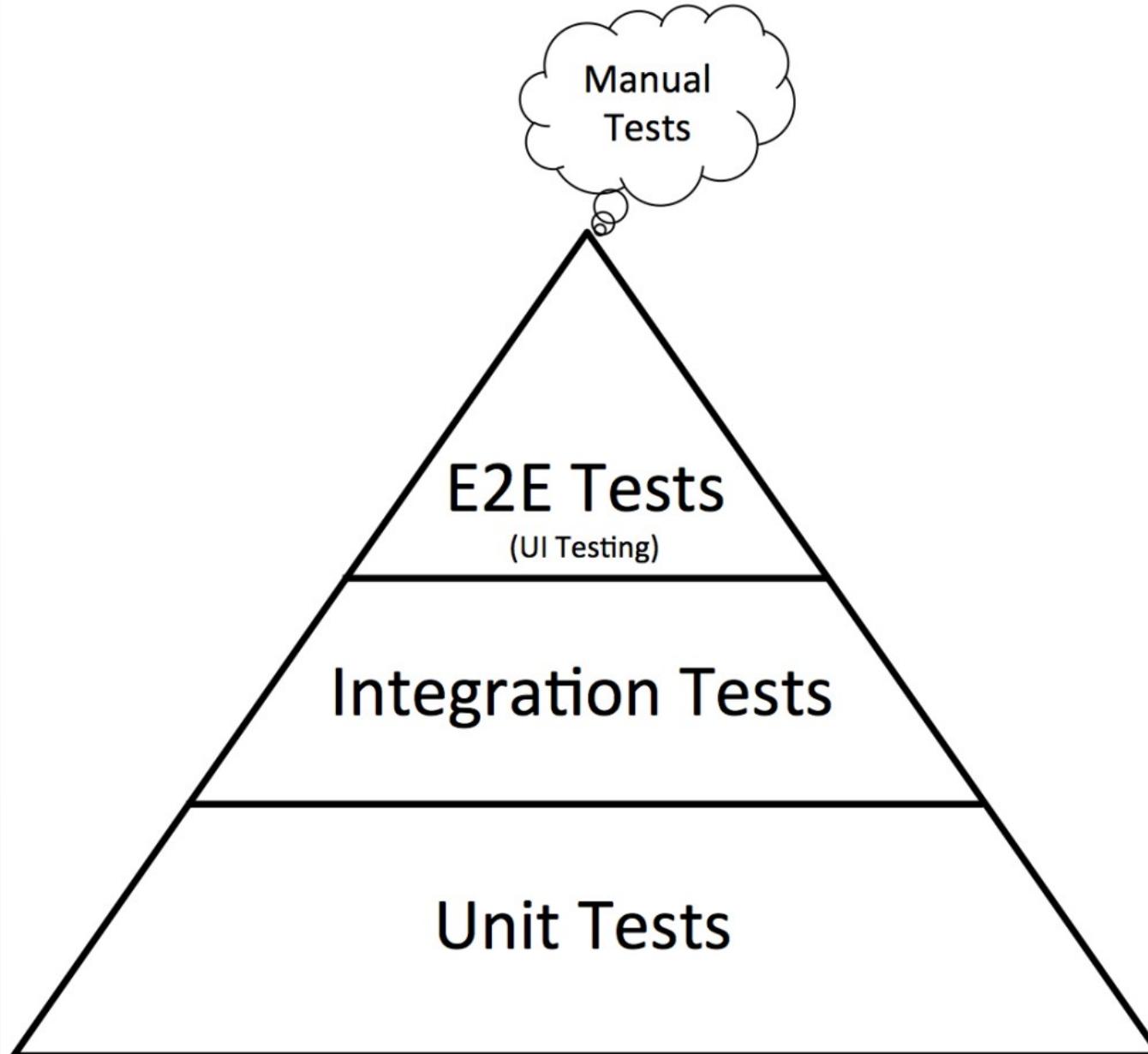
# Answer



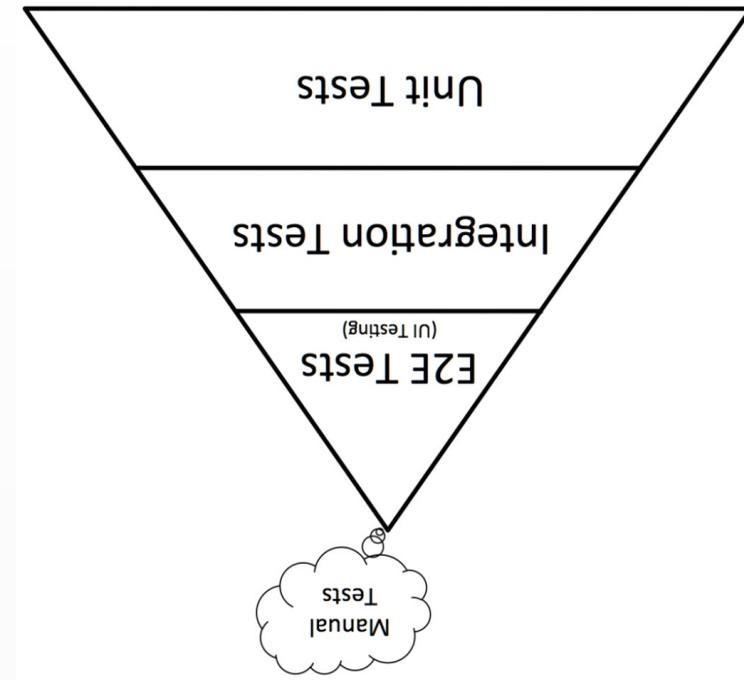
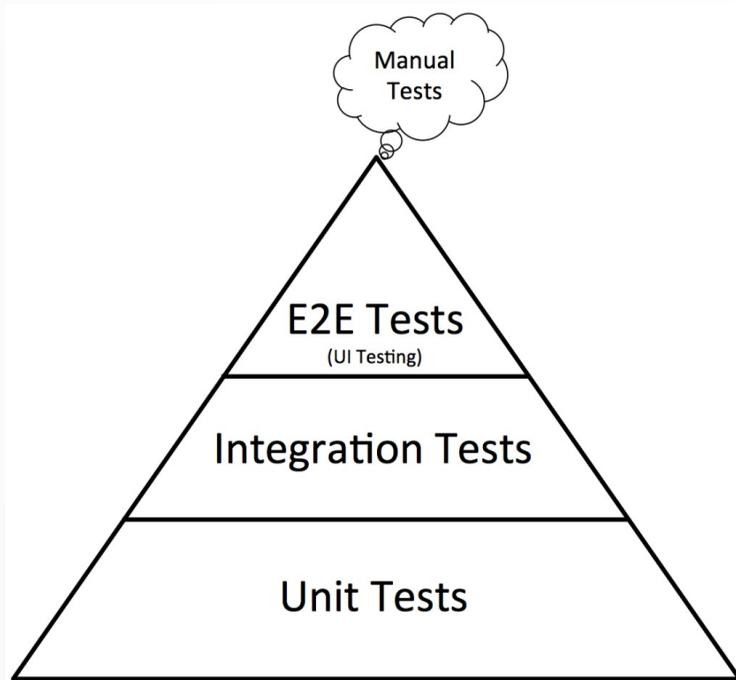
# Question 3



# Solution??



# Solution??



:)



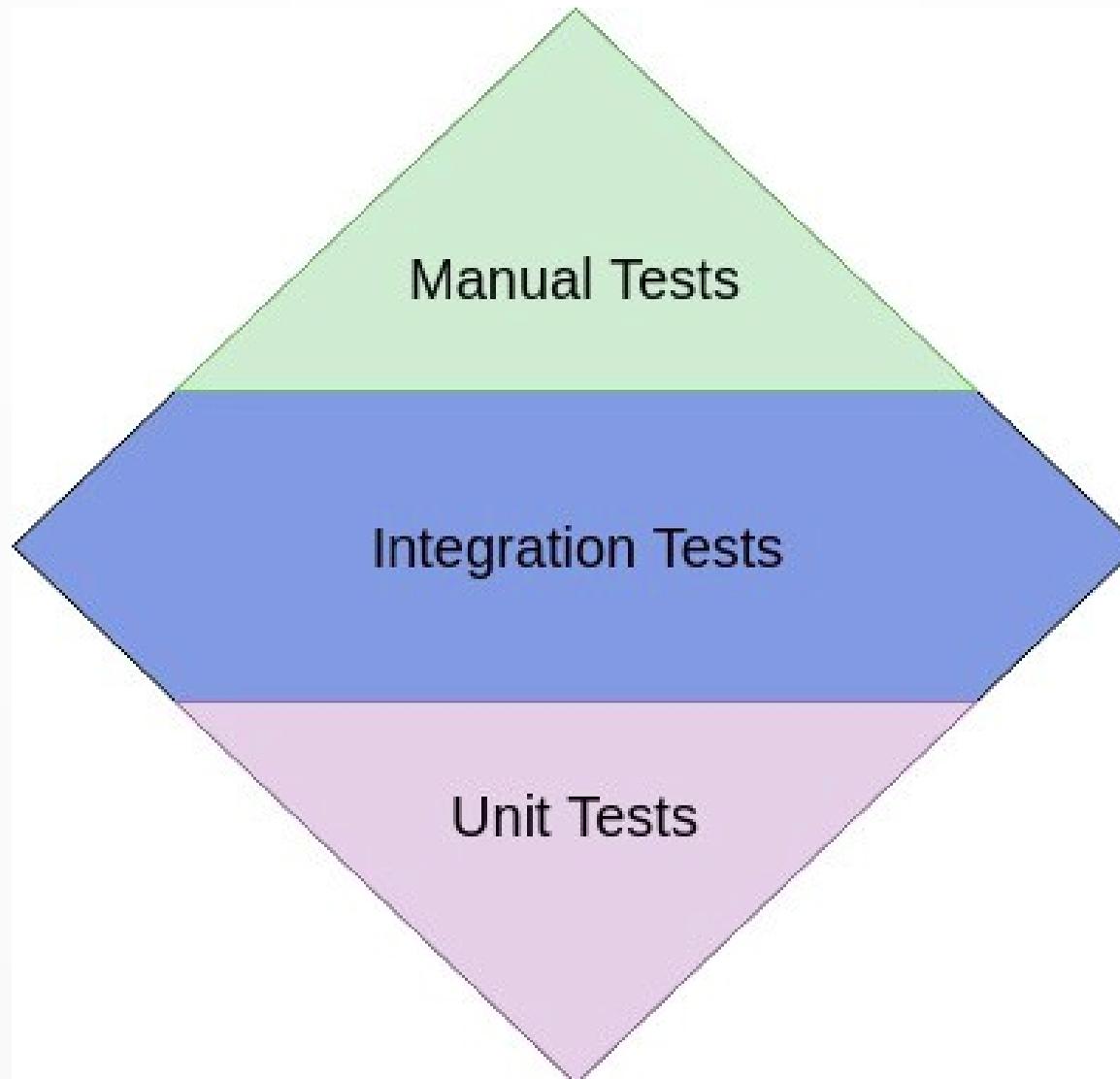
# TL;DR

1. `System.out.println(„Hello World!!“);`
2. Software testing pyramid
3. Integration tests
4. Ice-cream cone „pattern“ cases
5. Diamond test

# Diamond test



# Diamond test



# Diamond test

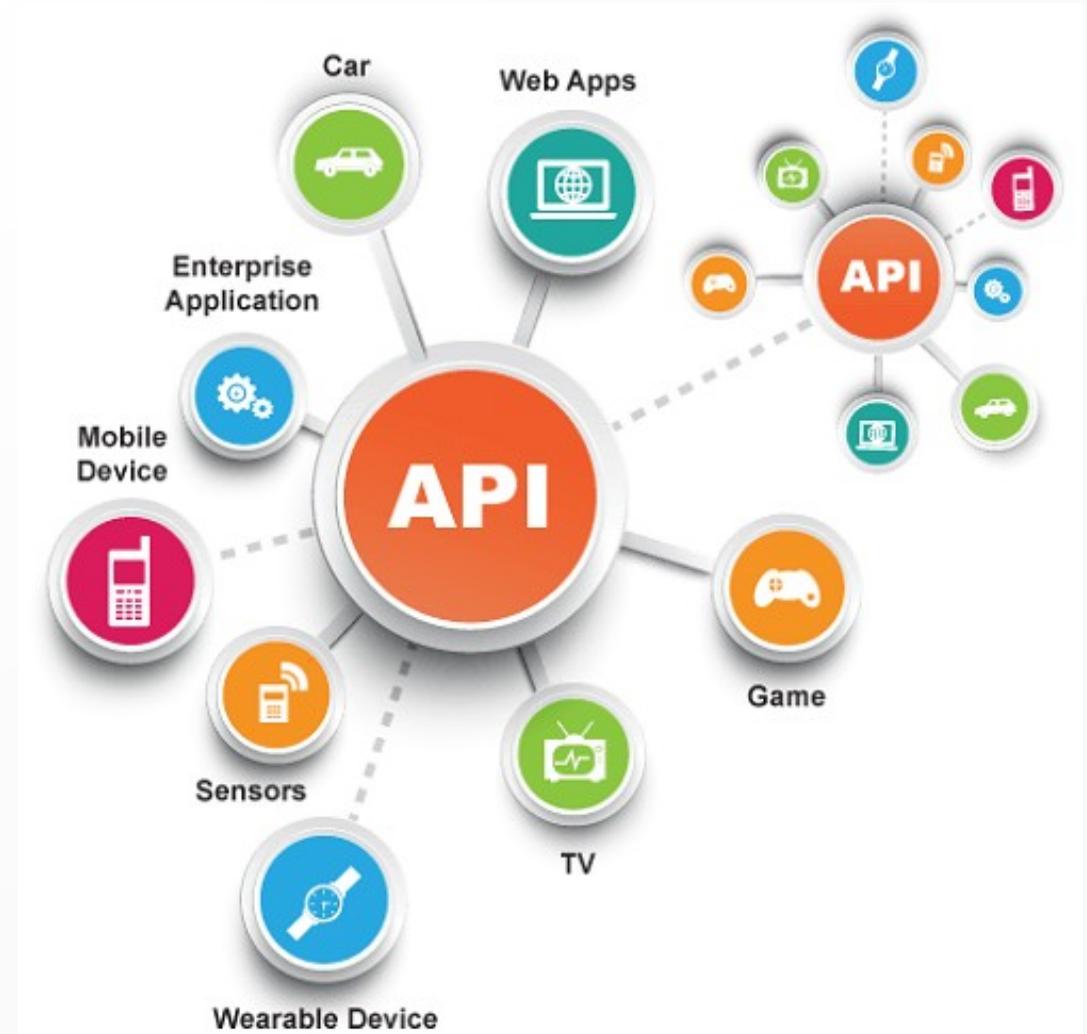
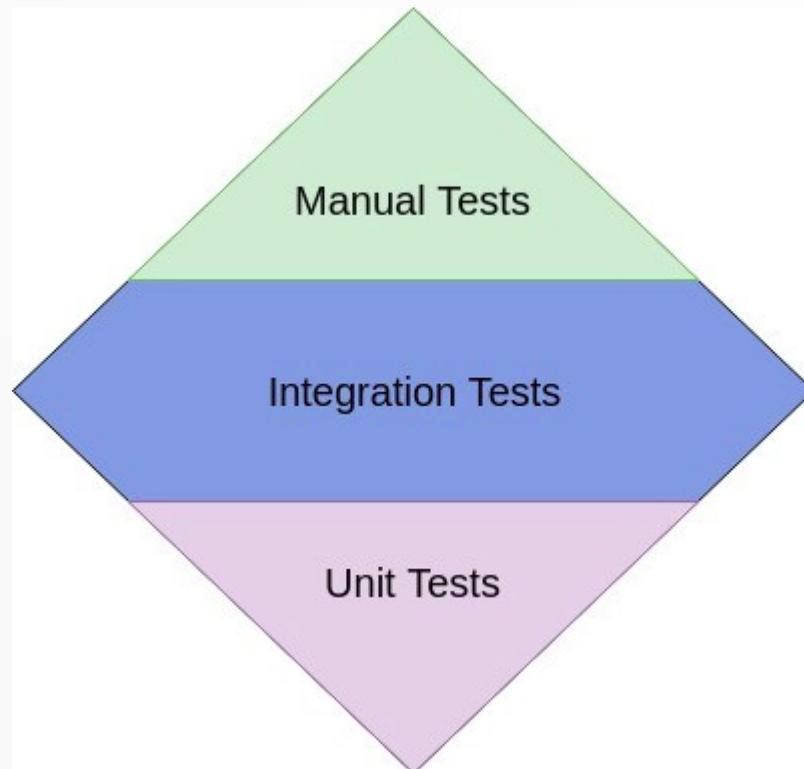
- Benefits:
  - We have confidence that the code does what it should
  - Immediate stacktrace feedback thanks to fast-fail convention
  - Realistic inputs, expected outputs, we don't care about code, because it does what it should
  - Easier maintenance

# Diamond test

- Don't hesitate to adjust the project to your architecture needs
- Best case of the pyramid usage, is use it as a guide
- In real-world projects more accurately is to use sometimes what we called the DIAMOND TEST
- Integration tests as the major part diamond test are still reasonable and relevant for test models

# Diamond test

**You have an integration to a third party API**



# TL;DR

1. `System.out.println(„Hello World!!“);`
2. Software testing pyramid
3. Integration tests
4. Ice-cream cone „pattern“ cases
5. Diamond test
6. API first approach

# API first approach

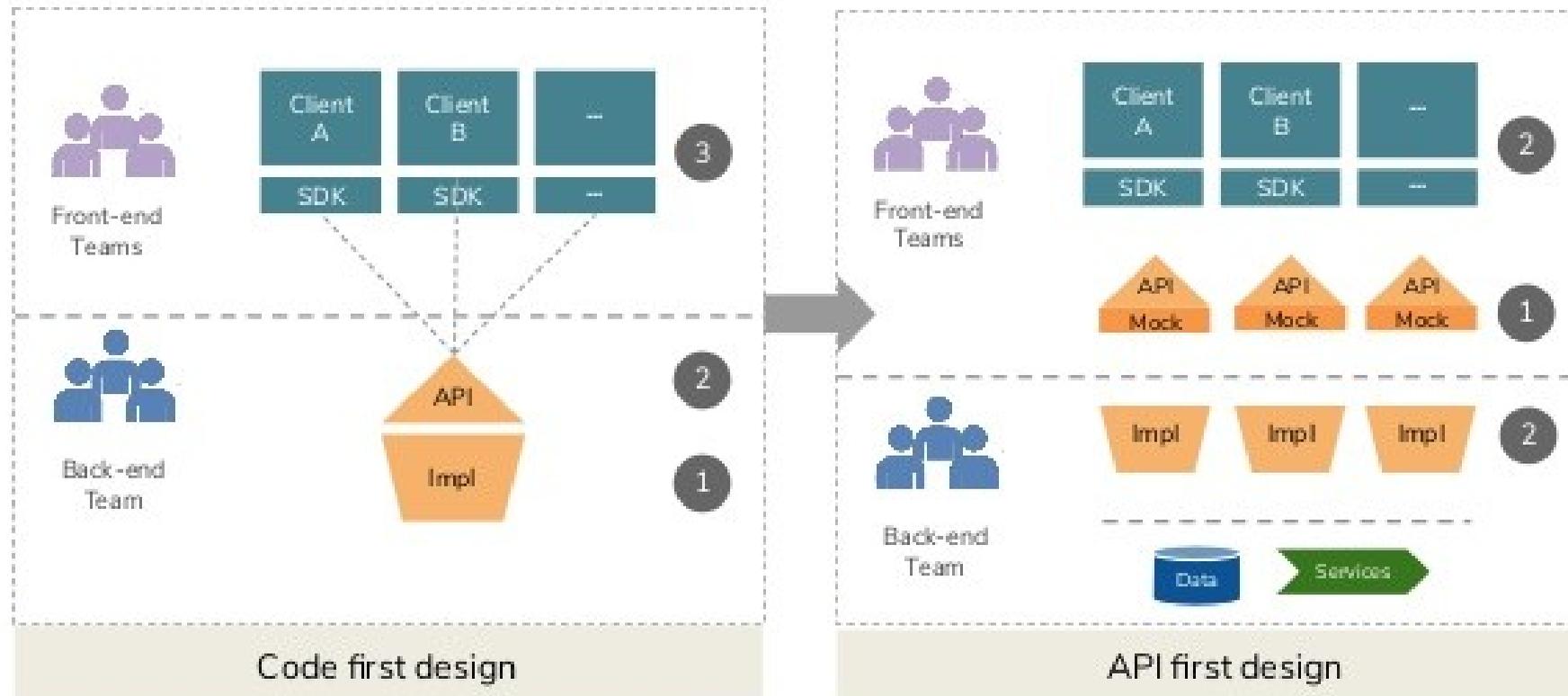


# Question 4

**What are the benefits of an API-First Approach?**

# Answer

## Advocate API First Design



INTEGRATION SUMMIT 2019

# Question 5

**How we are able to defend our microservice from external API?**

# Answer

Try defensive programming

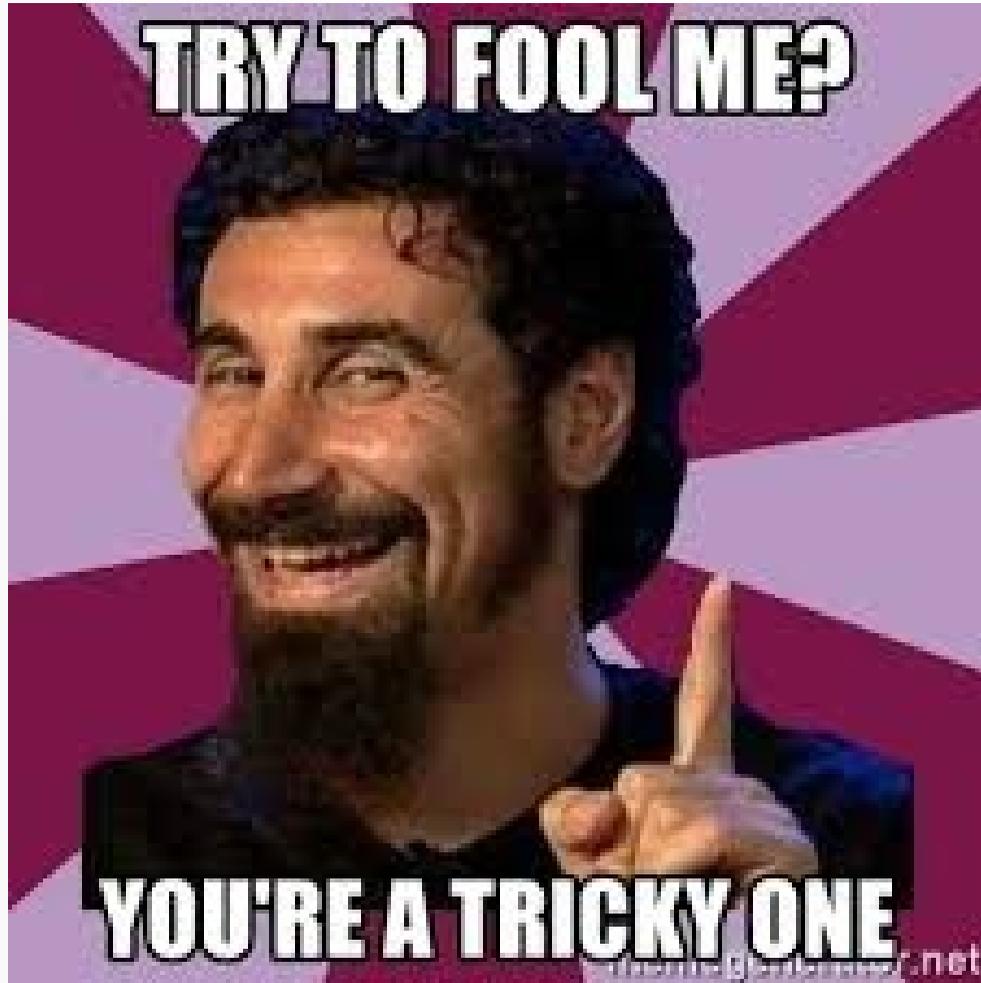


# Quiz

**JDBC == DbC**

**???**

# Indeed x2 :D



- JDBC -> JavaDatabase Connectivity
- DbC -> Design by Contract

# TL;DR

1. `System.out.println(„Hello World!!“);`
2. Software testing pyramid
3. Integration tests
4. Ice-cream cone „pattern“ cases
5. Diamond test
6. API first approach
7. Design by Contract

# Design by Contract

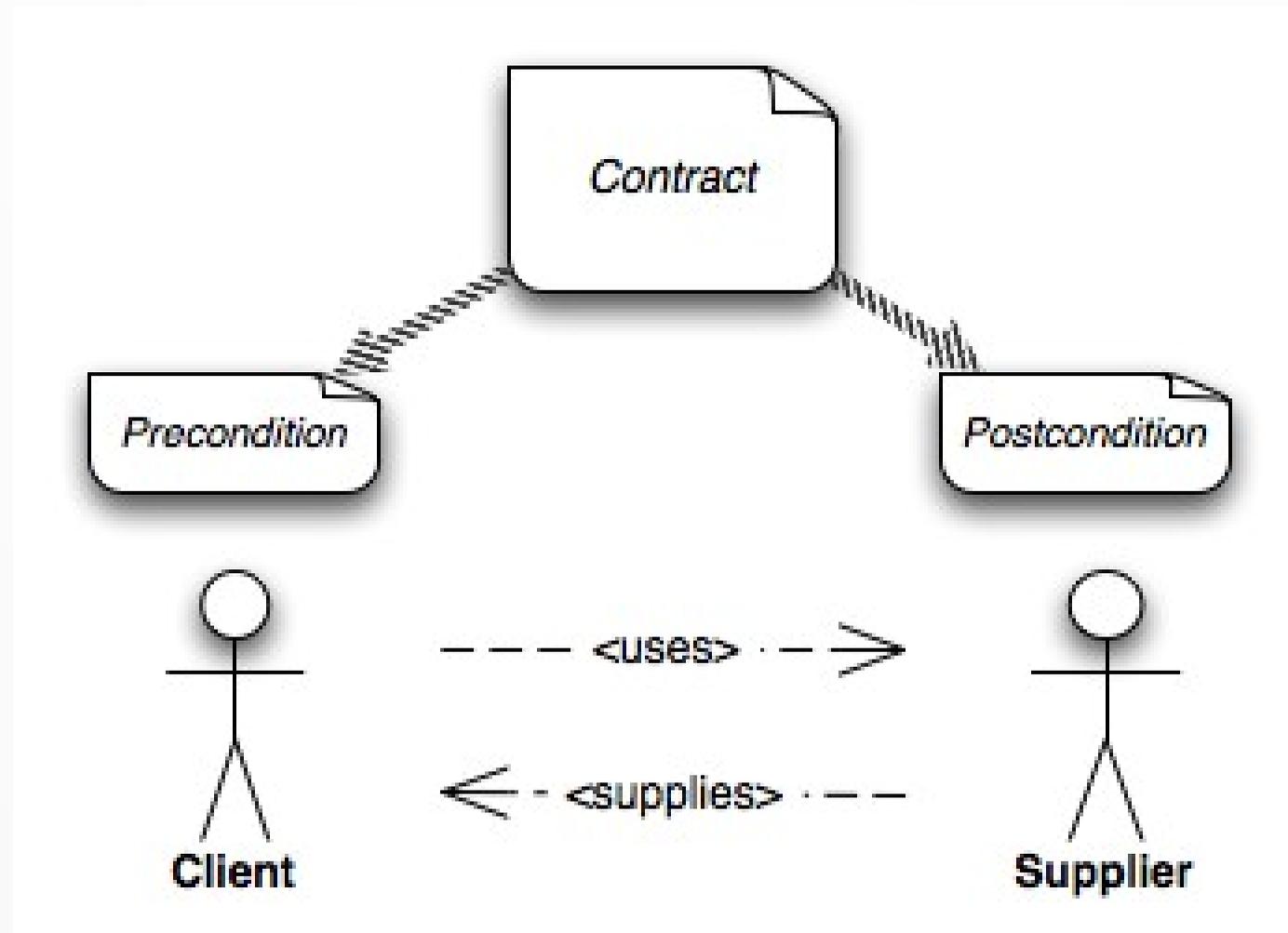


## DESIGN BY CONTRACT

DESIGN BY CONTRACT LEADS TO BEHAVIOR PRESERVATION. BEHAVIOR PRESERVATION LEADS TO LESS CRAPY SOFTWARE.

DIY.DESPAIR.COM

# Design by Contract



# Design by Contract

## Preconditions

```
public Cart(Long cartId, String totalPrice, String totalTax, String totalItemsPrice, String totalShippingPrice, List<ShippingGroup>
    checkArgument(Objects.nonNull(cartId), errorMessage: "'cartId' cannot be null");
    checkArgument(!Strings.isNullOrEmpty(totalTax), errorMessage: "'totalTax' cannot be null or empty");
    checkArgument(!Strings.isNullOrEmpty(totalPrice), errorMessage: "'totalPrice' cannot be null or empty");
    checkArgument(!Strings.isNullOrEmpty(totalItemsPrice), errorMessage: "'totalItemsPrice' cannot be null or empty");
    checkArgument(!Strings.isNullOrEmpty(totalShippingPrice), errorMessage: "'totalShippingPrice' cannot be null or empty");
    checkArgument(Objects.nonNull(shippingGroups), errorMessage: "'shippingGroups' cannot be null");
    checkArgument(Objects.nonNull(totalNumberOfItems), errorMessage: "'totalNumberOfItems' cannot be null");
    checkArgument(Objects.nonNull(currency), errorMessage: "'currency' cannot be null");
    this.cartId = cartId;
    this.totalPrice = totalPrice;
    this.totalTax = totalTax;
    this.totalItemsPrice = totalItemsPrice;
    this.totalShippingPrice = totalShippingPrice;
    this.shippingGroups = shippingGroups;
    this.affiliateId = affiliateId != null ? affiliateId : 0;
    this.totalNumberOfItems = totalNumberOfItems;
    this.currency = currency;
}
```

# Design by Contract

## Preconditions

```
@PostMapping("billing/validate")
public boolean validateBilling(@Valid @RequestBody AddressForm billingAddressForm, BindingResult bindingResult) {
    return validate(billingAddressForm).isValid();
}
```

```
@Documented
@Constraint(validatedBy = AddressValidator.class)
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface AddressConstraint {
    String message() default "{checkout.step1Delivery.validation.addressError}";

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};
}
```

# Design by Contract

## Preconditions

```
public Optional<Cart> getCart(Long cartId, String cartIdHash, Locale locale) {  
    if (isProvidedHashCorrect(cartId, cartIdHash)) {  
        return Optional.ofNullable(getCart(cartId, locale));  
    }  
    return Optional.empty();  
}
```

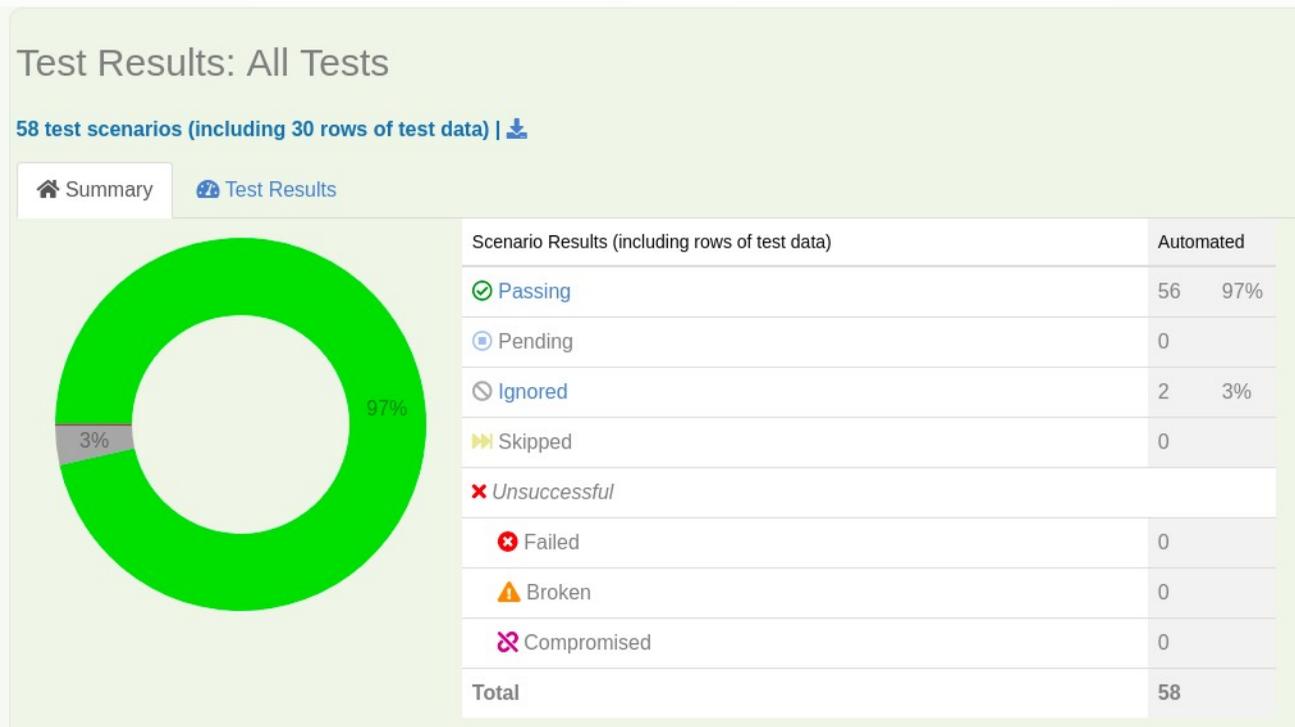
# Design by Contract

## Preconditions



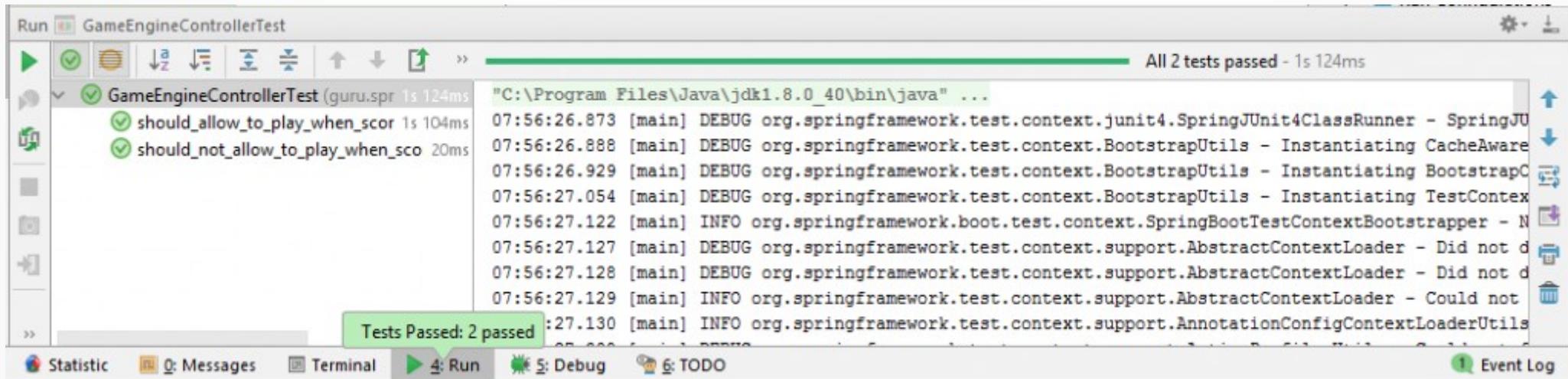
# Design by Contract

## Postconditions: E2E



# Design by Contract

**Postconditions: Integration tests :)**



The screenshot shows the Run console of an IDE. At the top, it says "Run GameEngineControllerTest" and "All 2 tests passed - 1s 124ms". Below this, the test results are listed:

- GameEngineControllerTest (guru.spr 1s 124ms) [Passed]
- should\_allow\_to\_play\_when\_scor 1s 104ms [Passed]
- should\_not\_allow\_to\_play\_when\_sco 20ms [Passed]

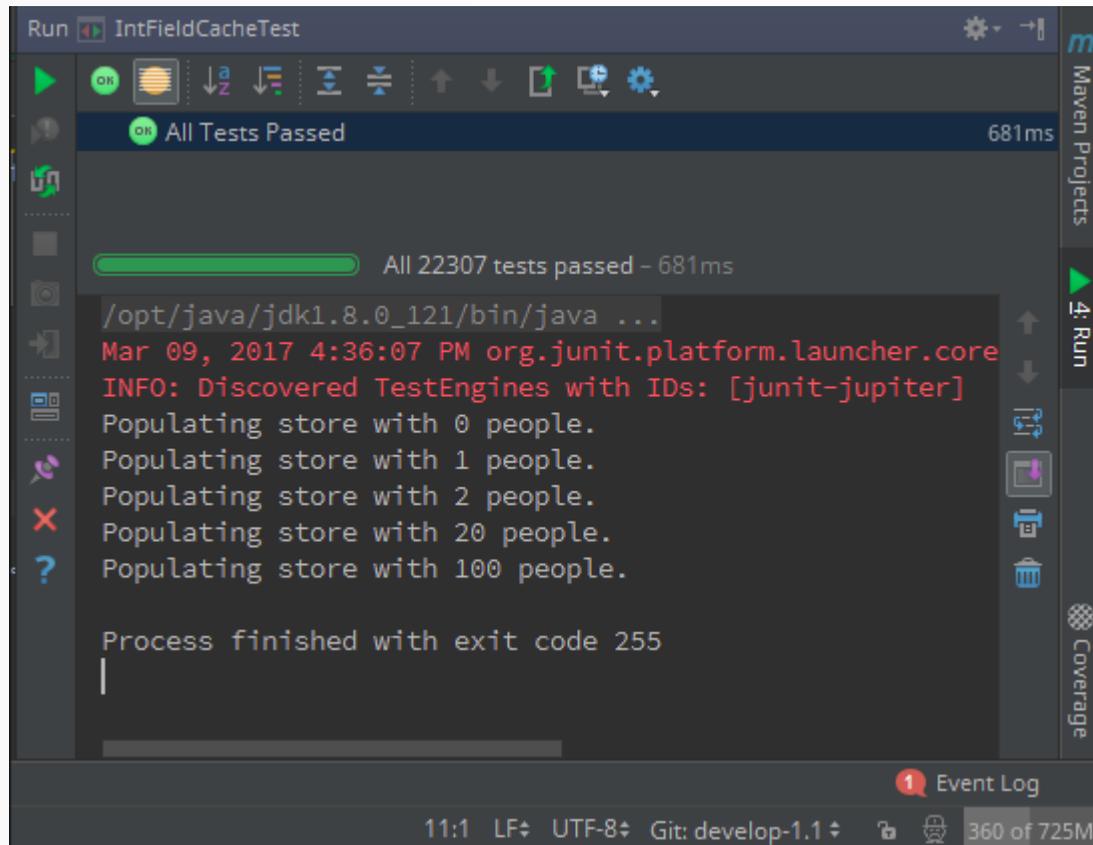
The console output shows the following log messages:

```
"C:\Program Files\Java\jdk1.8.0_40\bin\java" ...  
07:56:26.873 [main] DEBUG org.springframework.test.context.junit4.SpringJUnit4ClassRunner - SpringJU  
07:56:26.888 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAware  
07:56:26.929 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapC  
07:56:27.054 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContex  
07:56:27.122 [main] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper - N  
07:56:27.127 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not d  
07:56:27.128 [main] DEBUG org.springframework.test.context.support.AbstractContextLoader - Did not d  
07:56:27.129 [main] INFO org.springframework.test.context.support.AbstractContextLoader - Could not  
07:56:27.130 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils
```

A green box at the bottom of the console indicates "Tests Passed: 2 passed". The bottom of the IDE shows the "Run" button and other toolbars.

# Design by Contract

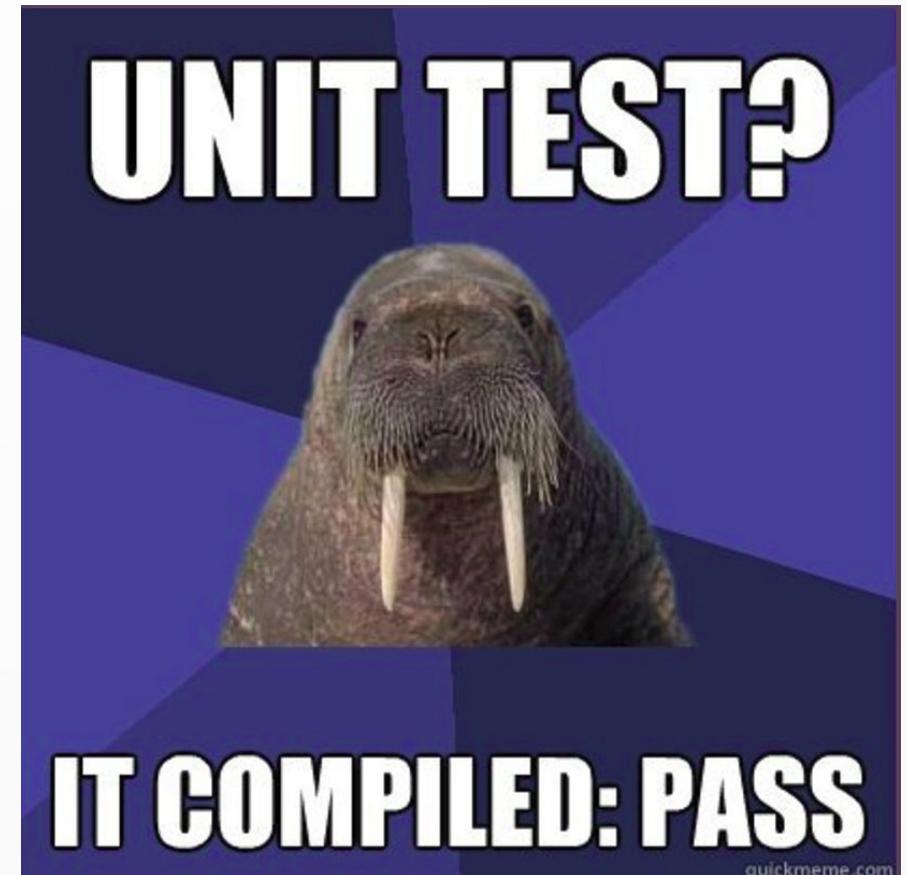
## Postconditions: Unit Tests



The screenshot shows an IDE window titled "Run IntFieldCacheTest". The status bar at the top indicates "All Tests Passed" in green. Below this, a progress bar shows "All 22307 tests passed - 681ms". The main console area displays the following output:

```
/opt/java/jdk1.8.0_121/bin/java ...  
Mar 09, 2017 4:36:07 PM org.junit.platform.launcher.core  
INFO: Discovered TestEngines with IDs: [junit-jupiter]  
Populating store with 0 people.  
Populating store with 1 people.  
Populating store with 2 people.  
Populating store with 20 people.  
Populating store with 100 people.  
  
Process finished with exit code 255
```

The IDE interface includes a toolbar with various icons, a sidebar with "Maven Projects" and "Run" buttons, and a status bar at the bottom showing "11:1 LF UTF-8 Git: develop-1.1 360 of 725M".



# Integration tests:

- Not only important from the Diamond Test POV, but also for Pyramid or Ice-cream cone models
- Potentially more effective compared to unit tests, because we operate on real data. The more data we have, the more resistant application we have got
- Potentially are slower compared to unit tests, but we could improve it, by using in-memory data (quasi integration)

# Take aways:

- Software test pyramid is a guide for preference depends on software architecture and client needs. It is not a panacea for everything
- Ice-cream cone anti-pattern still may apply to real use case mostly for legacy code at monolith architecture
- Diamond test model is well suited for microservice-based architecture in real world projects

# Take aways:

- Building applications based on API-first strategy allows organizations to develop and maintained efficiently for all devices, platforms a operations systems
- To reduce a risk of failure from integration with third party API try defensive programming
- Contract programming should be used in key areas of our application, like inputs, on which data we depend
- It's worth writing integration tests because they are based on real data

# Source of knowledge

- <https://sqa.stackexchange.com/questions/37623/is-inverted-test-pyramid-really-anti-pattern>
- <https://leeorengel.com/software-testing-best-practices/>
- <https://blog.restcase.com/internal-vs-external-apis/>
- <https://labs.spotify.com/2018/01/11/testing-of-microservices/>
- <https://www.slideshare.net/Apltools/visual-regression-testing-at-the-speed-of-unit-testing-by-gil-tayar>
- <https://swagger.io/resources/articles/adopting-an-api-first-approach/>
- <https://luisespinal.wordpress.com/2012/11/19/design-by-contract-by-example/>

**Questions?**

# Feedback



Google Forms

[http://bit.do/piramidy\\_i\\_diamenty](http://bit.do/piramidy_i_diamenty)

[https://docs.google.com/forms/d/e/1FAIpQLSdZiUukHvMOupvQnjY6\\_bJgiVeFMjbQ18rgUUxVU3Jh8FGgaQ/viewform](https://docs.google.com/forms/d/e/1FAIpQLSdZiUukHvMOupvQnjY6_bJgiVeFMjbQ18rgUUxVU3Jh8FGgaQ/viewform)

## Contact



<https://www.facebook.com/jan.koszela>



[https://www.twitter.com/jan\\_koszela](https://www.twitter.com/jan_koszela)



[jan.koszela@gmail.com](mailto:jan.koszela@gmail.com)