JUnit 5

The new Architecture

JUnit 5 tutorial

#16: The new Architecture

/RaminZare

IMPORTANCE OF SOFTWARE ARCHITECTURE

- Simplicity
- Modularity

JUnit 4 architecture

JUNIT 5 VINTAGE

To move to JUnit 5 and keep old codes

MONOLOTIC

JUnit 4 was only one jar file junit.jar

JUNIT 4 RUNNER

Not efficient way to extend functionallity

JUNIT 4 @RULE

It would allow you to do something before and after a test method has run

JUnit 5 Concerns

For developers

Test runners

Interactions with IDES/Build tools

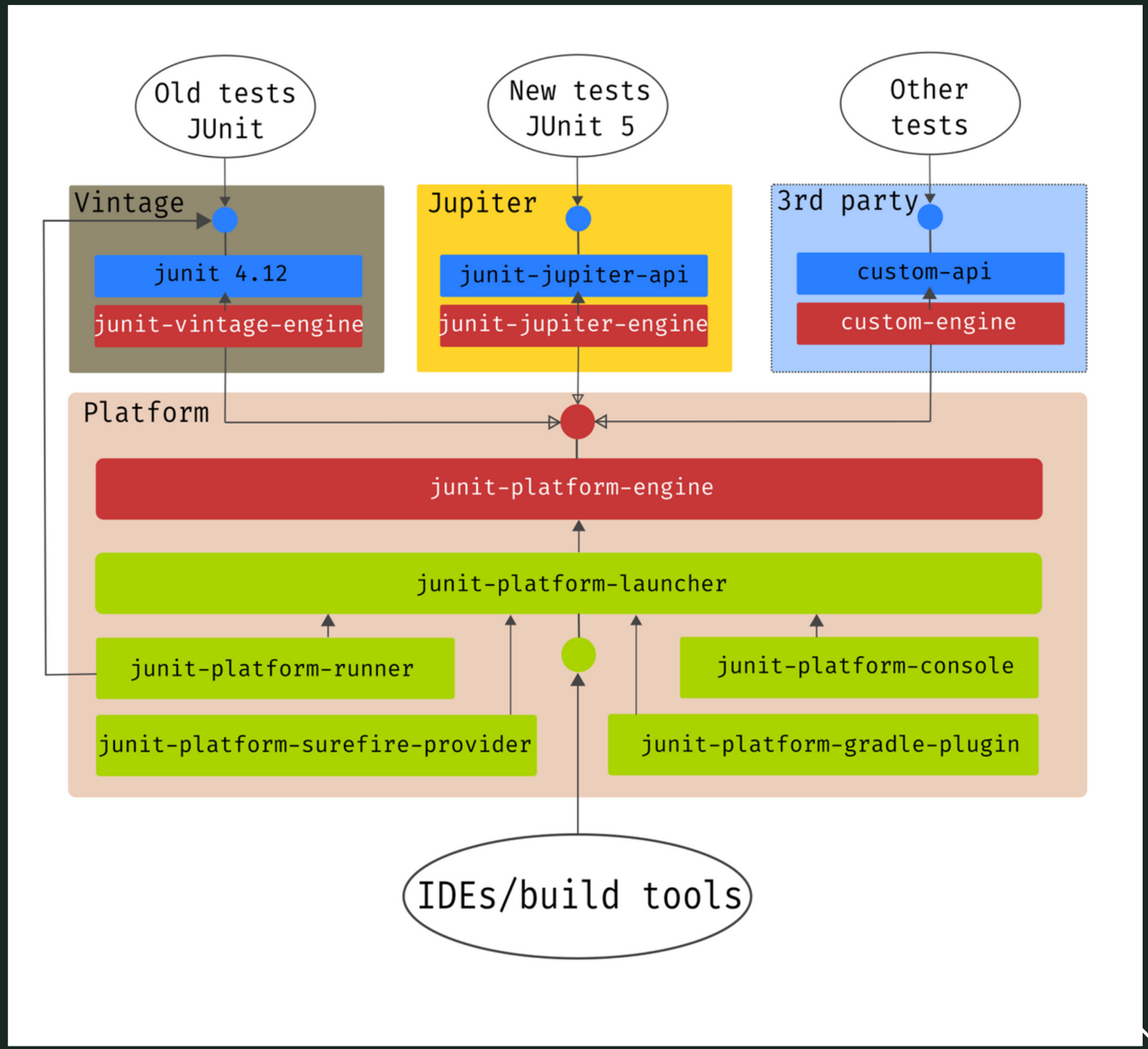
JUnit 5 Modules

JUnit Platform

JUnit Jupiter

JUnit Vintage

Big picture


Old tests
JUnit
New tests
JUnit 5
Other
tests
Vintage
junit 4.12
junit-vintage-engine
Jupiter
junit-jupiter-api
junit-jupiter-engine
3rd party
custom-api
custom-engine
Platform
junit-platform-engine
junit-platform-launcher
junit-platform-runner
junit-platform-console
junit-platform-surefire-provider
junit-platform-gradle-plugin
IDEs/build tools

JUnit 5

17

Software testing principles

JUnit 5 tutorial

#17: Software testing principles

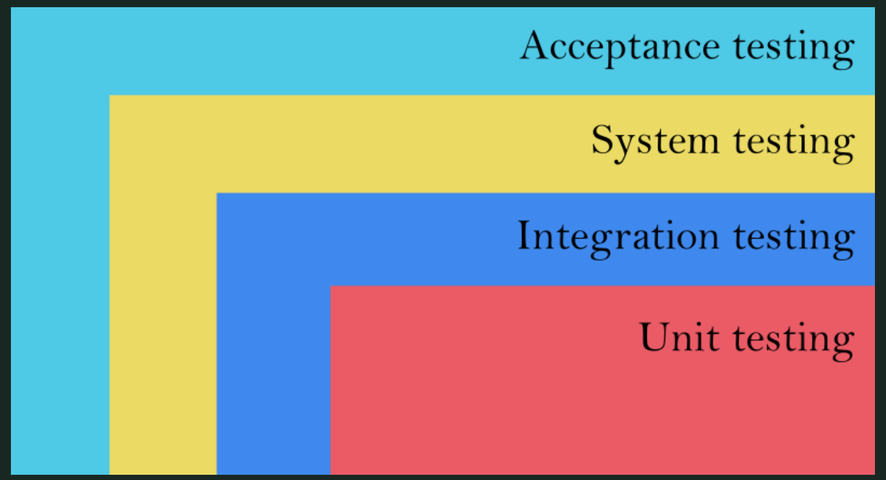
/RaminZare

Unit testing is not something you do without planning and preparation. To become a top-level developer, you need to contrast unit tests with functional and other types of tests

The need for unit tests

- Allow greater test coverage than functional tests
- Increase team productivity
- Detect regressions and limit the need for debugging
- Give us the confidence to refactor
- Improve implementation
- Document expected behavior
- Enable code coverage and other metrics

Test types



- Unit testing
  - method or classes
- Integration software testing
  - Objects
  - Services
  - Subsystems
- System software testing
  - evaluate system's compliance with requirements.
- Acceptance test
  - customer's need
    - given - when - then

Black-box tesing

White-box testing

Black-box vs. White-box testing

- USER-CENTRIC APPROACH
- TESTING DIFFICULTIES
- TEST COVERAGE

JUnit 5

18

Test coverage & Writing testable codes

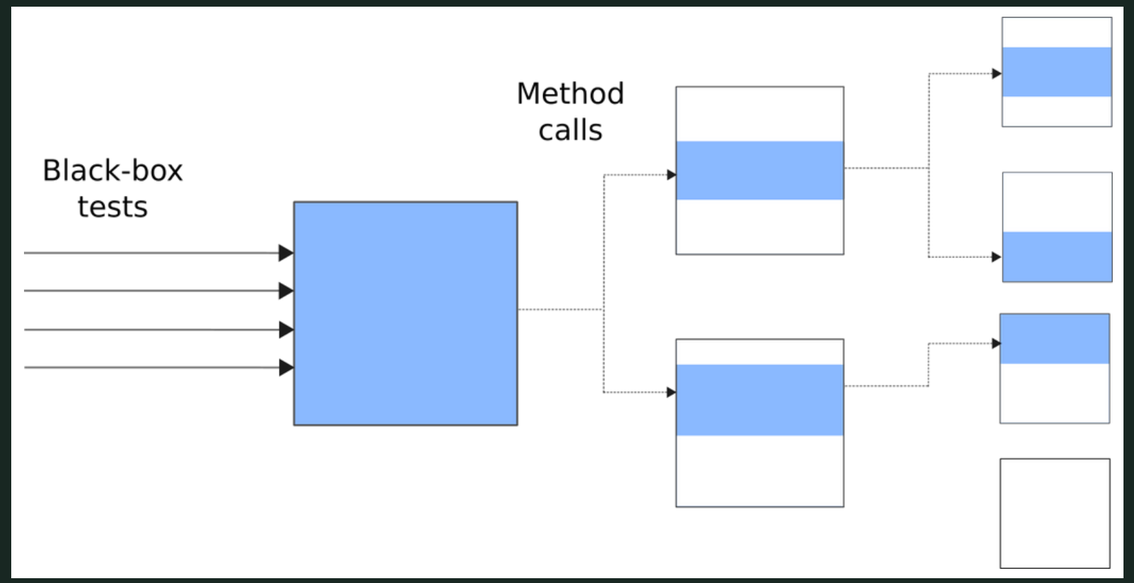
JUnit 5 tutorial

#18: Test coverage & Writing testable codes

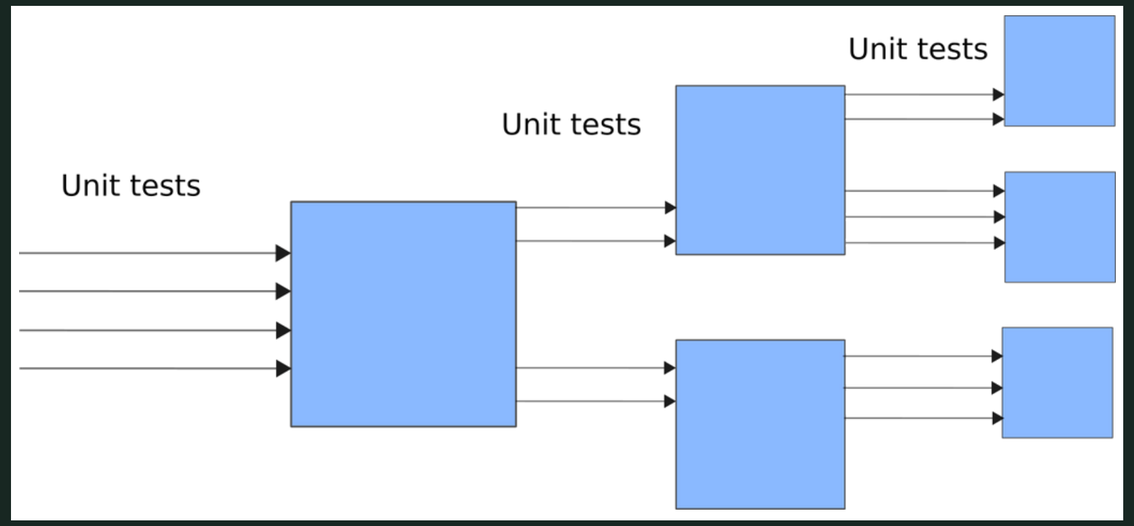
/RaminZare

Measuring test coverage

BLACK BOX TEST



UNIT TEST



Tools for measuring code coverage

- IntelliJ IDEA
- JaCoCo Java Code Coverage Library

Writing testable code

- Understanding that public APIs are contracts
- Following the Law of Demeter
- Reducing dependencies
- Creating simple constructors
- Avoiding hidden dependencies and global state
- Favoring generic methods
- Favoring polymorphism over conditionals

Following the Law of Demeter

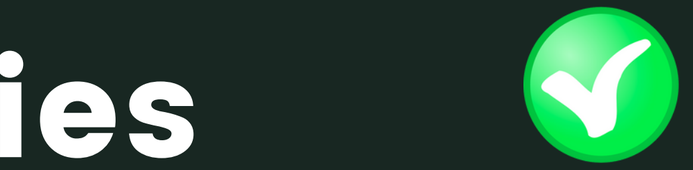
```
class Car{
    private Driver driver;

    Car(Context context) {
        this.driver = context.getDriver();
    }
}
```

```
class Car{
    private Driver driver;

    Car(Driver driver) {
        this.driver = driver;
    }
}
```

Reducing dependencies

```
class Vehicle {

    Driver d = new Driver();
    boolean hasDriver = true;

    private void setHasDriver(boolean hasDriver) {
        this.hasDriver = hasDriver;
    }
}
```

Reducing dependencies



```
class Vehicle {

    Driver d;
    boolean hasDriver = true;

    Vehicle(Driver d) {
        this.d = d;
    }

    private void setHasDriver(boolean hasDriver) {
        this.hasDriver = hasDriver;
    }
}
```

Creating simple constructors

```
class Car
    private int maxSpeed;

    Car() {
        this.maxSpeed = 180;
    }
}
```

Creating simple constructors

```
class Car
    private int maxSpeed;

    public void setMaxSpeed(int maxSpeed) {
        this.maxSpeed = maxSpeed;
    }
}
```

Avoiding hidden dependencies and global state

```
public void makeReservation() {
    Reservation reservation = new Reservation();
    reservation.makeReservation();
}


public class Reservation {
    public void makeReservation() {
        manager.initDatabase();
        // More ations
    }
}
```

Favoring generic methods

```
public static Set union(Set s1, Set s2) {
    Set result = new HashSet(s1);
    result.addAll(s2);
    return result;
}
```

```
public static <E> Set<E> union(Set<E> s1, Set<E> s2) {
    Set<E> result = new HashSet<>(s1);
    result.addAll(s2);
    return result;
}
```

Favoring polymorphism over conditionals

```
public class DocumentPrinter {
    void printDocument() {
        switch (document.getDocumentType()) {
        case WORD_DOCUMENT:
            printWORDDocument();
            break;
        case PDF_DOCUMENT:
            printPDFDocument();
            break;
        case TEXT_DOCUMENT:
            printTextDocument();
            break;
        default:
            printBinaryDocument();
            break;
        }
    }
}
```

JUnit 5

19

TDD, BDD, Mutation testing & Development cycle

JUnit 5 tutorial

#19: TDD, BDD, Mutation testing & Development cycle

/RaminZare

TDD

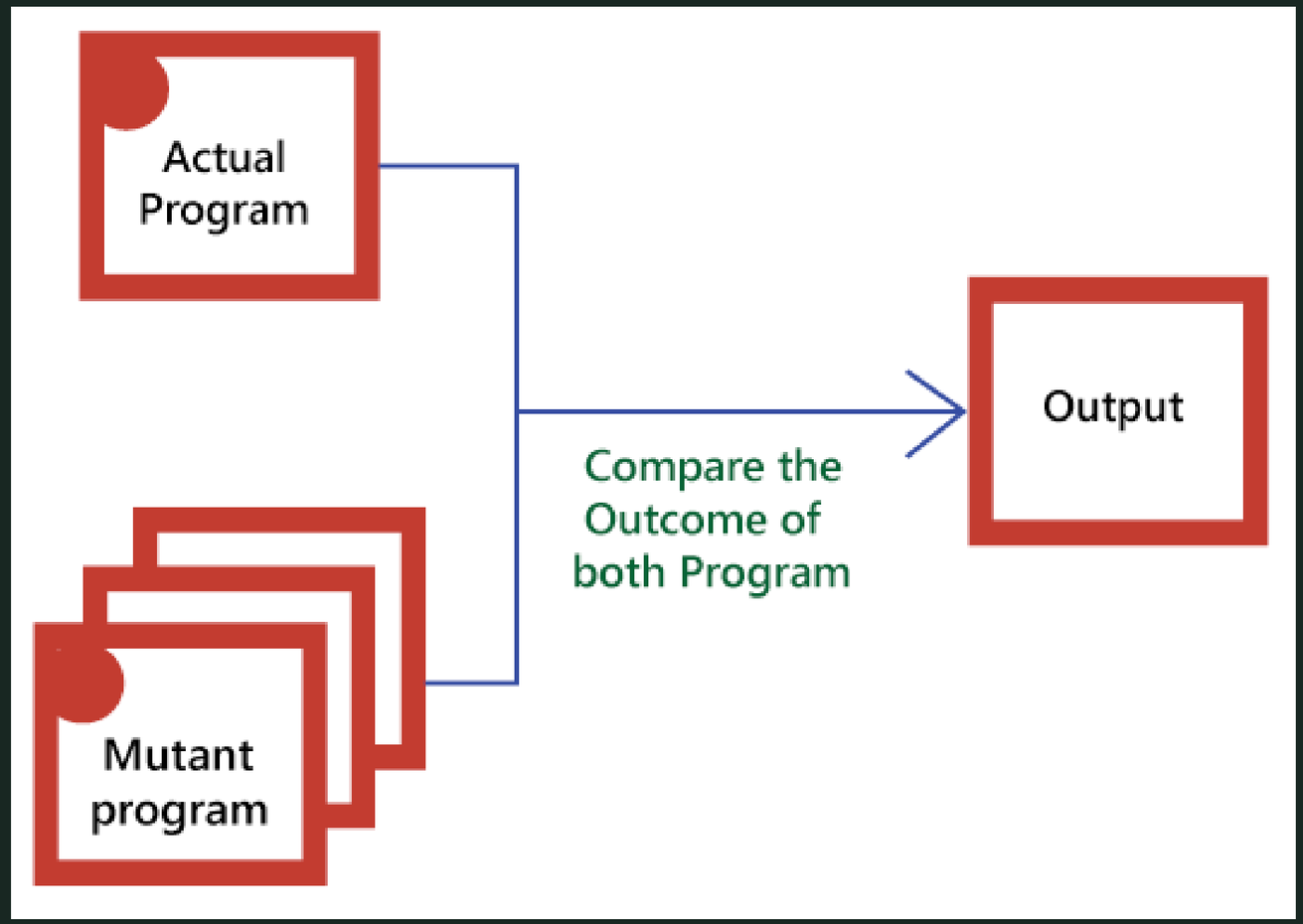
Test-driven development

BDD

Behavior-driven development

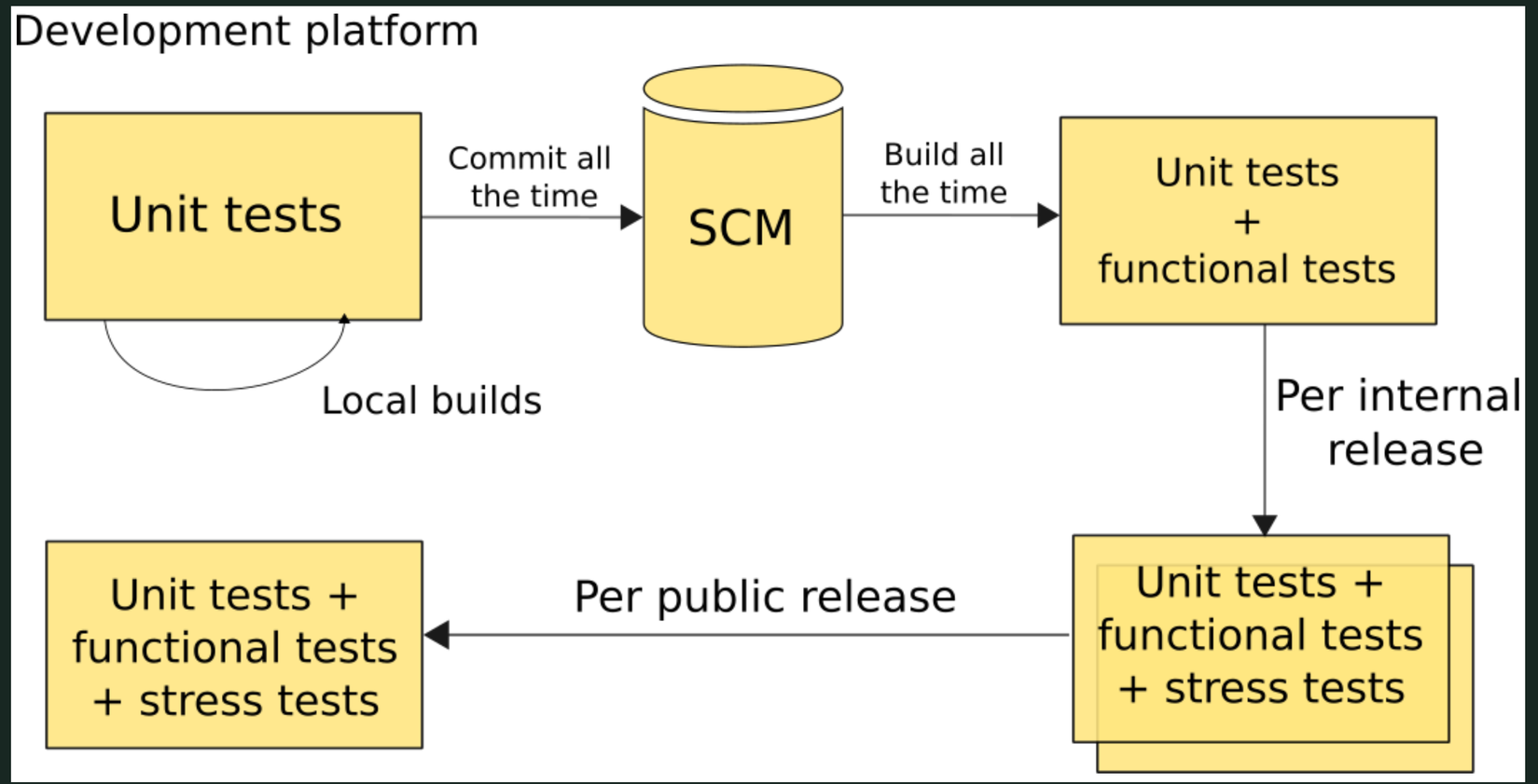
```
Scenario: Successful withdrawal from an account in credit

  Given I have $100 in my account # the context

  When I request $20              # the event(s)

  Then $20 should be dispensed    # the outcome(s)
```

Mutation testing



pitest.org

Testing in development cycle



JUnit 5

20

Stubbing and using Mock objects

JUnit 5 tutorial

#20: Stubbing and using Mock objects

/RaminZare

During testing, you will find that the code depends on other classes and environment or some parts are not ready. You can replace the missing part with fake behavior. Strategies are stubbing and using mock objects.

Stubbing

- Provide a predetermined behavior
- Is written outside the test
- Have a fixed behavior

initialize stub > execute test > verify assertions

Example

```
import java.net.URL;
import java.net.HttpURLConnection;
import java.io.InputStream;
import java.io.IOException;
 public class WebClient {
   public String getContent(URL url) {
      StringBuffer content = new StringBuffer();
      try {
         HttpURLConnection connection = (HttpURLConnection)
            url.openConnection();
         connection.setDoInput(true);
         InputStream is = connection.getInputStream();
         byte[] buffer = new byte[2048];
         int count;
         while (-1 != (count = is.read(buffer))) {
            content.append(new String(buffer, 0, count));
         }
      } catch (IOException e) {
         throw new RuntimeException(e);
      }
      return content.toString();
   }
}
```

Solution 1: Stubbing the web server resources

```
public class TestWebClient {
    private WebClient client = new WebClient();

  @BeforeAll
  public static void setUp() throws Exception() {
      Server server = new Server(8081);

      Context contentOkContext = new Context(server, "/testGetContentOk");
      contentOkContext.setHandler(new TestGetContentOkHandler());

      server.setStopAtShutDown(true);
      server.start();
  }

  @Test
  public void testGetContentOk() throws Exception {
      String workingContent = client.getContent(new URL(
          "http://localhost:8081/testGetContentOk"));
      assertEquals("It works", workingContent);
  }
}
```

Solution 2: Stubbing the connection

```
public class StubHttpURLConnection extends HttpURLConnection {
    @Override
    public InputStream getInputStream() throws IOException {
        ByteArrayInputStream readStream = new ByteArrayInputStream(
            new String("It works").getBytes());
        return readStream;
    }
}

public class TestWebClient1 {
    @BeforeAll
    public static void setUp() {
        URL.setURLStreamHandlerFactory(new StubStreamHandlerFactory());
    }
    // private static class StubStreamHandlerFactory implements URLStreamHandlerFactory
    // private static class StubHttpURLStreamHandler extends URLStreamHandler {
    @Test
    public void testGetContentOk() throws MalformedURLException {
        WebClient client = new WebClient();
        String workingContent = client.getContent(
                                    new URL("http://localhost"));
        assertEquals("It works", workingContent);
    }
}
```

Mock objects

- Suited for testing a portion of code logic in isolation from the rest of the code
- no side effects resulting from other objects

Initialize mock > set expectations > execute test > verify assertions

Example

```
@Test  public void testTransferOk() {
        Account senderAccount = new Account("1", 200);
        Account beneficiaryAccount = new Account("2", 100);

        MockAccountManager mockAccountManager = new MockAccountManager();
        mockAccountManager.addAccount("1", senderAccount);
        mockAccountManager.addAccount("2", beneficiaryAccount);

        AccountService accountService = new AccountService();
        accountService.setAccountManager(mockAccountManager);

        accountService.transfer("1", "2", 50);

        assertEquals(150, senderAccount.getBalance());
        assertEquals(150, beneficiaryAccount.getBalance());
    }
}

public class MockAccountManager implements AccountManager {
   private Map<String, Account> accounts = new HashMap<String, Account>();
       public void addAccount(String userId, Account account) {
           this.accounts.put(userId, account);
       }
       public Account findAccountForUser(String userId) {
           return this.accounts.get(userId);
       }
}
```

Example

```
@Test
public void testGetContentOk() throws Exception {
    MockHttpURLConnection mockConnection = new MockHttpURLConnection();
    mockConnection.setupGetInputStream(
            new ByteArrayInputStream("It works".getBytes()));
    MockURL mockURL = new MockURL();
    mockURL.setupOpenConnection(mockConnection);
    WebClient client = new WebClient();
    String workingContent = client.getContent(mockURL);
    assertEquals("It works", workingContent);
}
```

Using mocks as Trojan horses

TROJAN HORSES

It is possible to use mocks as probes by letting them monitor the method calls that the object under test makes

EXAMPLES

Monitoring the inputs passing to the object or checking resource leak

```
public class MockInputStream extends InputStream {

  private int closeCount = 0;

  public void close() throws IOException {
    closeCount++;
    super.close();
  }
  public void verify() throws java.lang.AssertionError {
    if (closeCount != 1) {
      throw new AssertionError ("close() should "
                                + "have been called once and once only");
    }
  }
}
```

Introducing mock frameworks

EASYMOCK

https://easymock.org/

JMOCK

http://jmock.org/

MOCKITO

https://site.mockito.org/

JUnit 5

21

Testing Spring Boot applications

JUnit 5 tutorial

#21: Testing Spring Boot applications

/RaminZare

JUnit 5

22

Integration test in Spring Boot with Database

JUnit 5 tutorial

#22: Integration test in Spring Boot with Database

/RaminZare