APA INI PART 2

JUnit Annotation

- `@Test` annotation specifies that method is the test method.
- `@Test(timeout=1000)` annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).
- `@BeforeClass` annotation specifies that method will be invoked only once, before starting all the tests.
- `@Before` annotation specifies that method will be invoked before each test.
- `@After` annotation specifies that method will be invoked after each test.
- `@AfterClass` annotation specifies that method will be invoked only once, after finishing all the tests.

Type of Testing

Test Strategies :

- **Blackbox Testing** Test overall software functionality, does not cover each statement.
- **Whitebox Testing** Test every statement and condition given in the code

Once strategies are decided, testing can be done at various level :

- **UNIT TESTING** is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed.
- **Integration testing** (sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group(sometimes called integration and testing, abbreviated I&T) is the phase in software testing in which individual software modules are combined and tested as a group.
- **System testing** is performed on a complete, integrated system. It allows checking system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.
- **Acceptance testing** is a test conducted to find if the requirements of a specification or contract are met as per its delivery. Acceptance testing is basically done by the user or customer. However, other stockholders can be involved in this process.
- **Regression testing** is the process of testing changes to computer programs to make sure that the older programming still works with the new changes.
- **Performance Testing** is a type of software testing and part of performance engineering that is performed to check some of the quality attributes of software like Stability, reliability, availability.

CI Testing Tools

HUDSON and JENKINS

JMeter TestPlan

A test plan describes a series of steps JMeter will execute when run. A complete test plan will consist of one or more Thread Groups, logic controllers, sample generating controllers, listeners, timers, assertions, and configuration elements.

You can define a sample set of users inside a test plan using a **thread group** element.

Adding elements to a test plan can be done by right-clicking on an element in the tree, and choosing a new element from the `add` list. Alternatively, elements can be loaded from file and added by choosing the `merge` or `open` option.

To remove an element, make sure the element is selected, right-click on the element, and choose the `remove` option

Data Driven Testing Benefits

- **Data Mapping** verfies whether a ui component is mapped with the correct columns of the table in database
- **ACID Properties Validation** Testing a Database for its ACID property makes the database stable and reliable

  ACID Properties and their Aspect :

  - **Atomicity** guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely.
  - **Consistency** A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.
  - **Isolation.** A transaction in process and not yet committed must remain isolated from any other transaction.
  - **Durability.** Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state.
- **Data Integrity** Testing a database for its data integrity refers to the verification of a change in one table being reflected in another, as desired.

Interpreting CoCo Results

- **Green** colors, means the code area is completely covered by the test cases
- **Yellow** colors, means the code area is partially covered by the test cases
- **Red** colors, means the code area is left uncovered by the test cases

Importing JWebUnit in Maven Project

```
<dependency>
    <groupId>net.sourceforge.jwebunit</groupId>
    <artifactId>jwebunit-htmlunit-plugin</artifactId>
    <version>3.2</version>
    <scope>test</scope>
</dependency>
```

DatabaseOperation Class Method

apa ini 2

Operation	Description
DatabaseOperation.UPDATE	This operation updates the database from the dataset contents. This operation assumes that table data already exists in the target database and fails if this is not the case.
DatabaseOperation.INSERT	This operation inserts the dataset contents into the database. This operation assumes that table data does not exist in the target database and fails if this is not the case. To prevent problems with foreign keys, tables must be sequenced appropriately in the dataset.
DatabaseOperation.DELETE	This operation deletes only the dataset contents from the database. This operation does not delete the entire table contents but only data that are present in the dataset.
DatabaseOperation.DELETE_ALL	Deletes all rows of tables present in the specified dataset. If the dataset does not contains a particular table, but that table exists in the database, the database table is not affected. Table are truncated in reverse sequence.
DatabaseOperation.TRUNCATE_TABLE	Truncate tables present in the specified dataset. If the dataset does not contains a particular table, but that table exists in the database, the database table is not affected. Table are truncated in reverse sequence.
DatabaseOperation.REFRESH	This operation literally refreshes dataset contents into the target database. This means that data of existing rows are updated and non-existing row get inserted. Any rows which exist in the database but not in dataset stay unaffected. This approach is more appropriate for tests that assume other data may exist in the database. if they are correctly written, tests using this strategy can even be performed on a populated database like a copy of a production database.
DatabaseOperation.CLEAN_INSERT	This composite operation performs a DELETE_ALL operation followed by an INSERT operation. This is the safest approach to ensure that the database is in a known state. This is appropriate for tests that require the database to only contain a specific set of data.
DatabaseOperation.NONE	Empty operation that does absolutely nothing.

Parameterized Test Annotation

Create a public static method annotated with `@Parameters` annotation that returns a collection of object (e.g Array) and for the **TEST** classes annotate the class with `@RunWith(Parameterized.Class)`. Import the following classes to use it.

```
import org.junit.runner.RunWith;
import org.junit.runner.Parameterized;
```

Code Coverage

- Statement Coverage, checks the execution of various statements.
- Function Coverage, checks whether test invoke each function.
- Condition, Checks the outcome of condition (e.g `if` clause, require the condition to be true and false so, `else` block are executed)

- Loop Coverage, ensure a loop was executed for atleast once

STLC Phases

- Requirement Analysis
- Test Planning
- Test case development
- Test Environment setup
- Test Execution
- Test Cycle closure

JWebUnit most used methods

Man there is still a fuckton of method in the books that are not covered killmyself.

Assert Method	What's It Used For
assertArrayEquals("message",A,B)	Asserts the equality of the A and B arrays
assertEquals("message",A,B)	Asserts the equality of objects A and B.This assert will actually invokes the equals() method on the first object against second.
assertSame("message",A,B)	Asserts that the A and B objects have the same value.
assertTrue("message",A)	Asserts that A condition is evaluated to true
assertNotNull("message",A)	Asserts that A isn?t null

@runwith parameterized `org.junit.runners.Parameterized.Parameters;`

Parameterized tests allow a developer to run the same test over and over again using different values.

step

- Annotate test class with `@RunWith(Parameterized.class)`.
- Create a public static method annotated with `@Parameters` that returns a Collection of Objects (as Array) as test data set.
- Create a public constructor that takes in what is equivalent to one "row" of test data.
- Create an instance variable for each "column" of test data.
- Create your test case(s) using the instance variables as the source of the test data.

Example

apa ini 2

```
...
@RunWith(Parameterized.class)
public class CalcTest {
    private int a,b,expected;
    public CalculatorTest(int a, int b, int expected){
        this.a=a;
        this.b=b;
        this.expected=expected;
    }
    ...

    @Test
    public void testAdd(){
    Calculator instance = new Calculator();
    int actual = instance.add(a,b);
    Assert.assertEquals(expected, actual);
    }
@Parameters
    public static Collection<Integer[]> getParameters(){
    Integer [][]inputArr;

    ...
```

Stub

A stub is a controllable replacement for an existing dependency (or collaborator) in the system. By using a stub, you can test your code without dealing with the dependency directly.

example stub code

```
interface Service {
    String doSomething();
}

class ServiceStub implements Service {
    public String doSomething(){
        return "my stubbed return";
    }
}
```

JWebUnit

apa ini 2

```java
import static net.sourceforge.jwebunit.junit.JWebUnit.*;

public class ExampleWebTestCase {

    @Before
    public void prepare() {
        setBaseUrl("http://localhost:8080/test");
    }

    @Test
    public void test1() {
        beginAt("home.xhtml"); //Open the browser on http://localhost:8080/test/home.xhtml
        clickLink("login");
        assertTitleEquals("Login");
        setTextField("username", "test");
        setTextField("password", "test123");
        submit();
        assertTitleEquals("Welcome, test!");
    }
}
```

JUnit Report

- **Tools** : Maven
- **Plugin** : Surefire

Summary

Tests	Errors	Failures	Skipped	Success Rate	Time
Total Test functions	Error in function	Failures in Function (Assert not true)	skipped	percentage of total testfunc-failfunc/testfunc	time elapsed

Package List

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
exampletestpkg	3	0	0	0	100%	0.01

exampletestpkg

Class	Tests	Errors	Failures	Skipped	Success Rate	Time
exampleclass	3	0	0	0	100%	0.01

Test Class

exampleclass

apa ini 2

test1func	0.001
test2func	0.001
test3func	0.001

Common issue