SPECIFICATION BASED TESTING TECHNIQUES / BLACK BOX TECHNIQUES

LEARNING OBJECTIVES

- Describe the benefits and construction of decision tables and state transition models in the context of test case design
- Create and use decision tables and state transition models in tests
- Explain the concept and need for data-driven testing and getting test data from files
- Perform repeated, parameterized and dynamic tests with uniform input values from different sources, such as arrays, lists and files
- Explain the advantages and the disadvantages of different assertion libraries and using alternative matchers

DECISION TABLES

Issues with equivalence partitioning and boundary value analysis

They do not explore combinations of input

A program can fail because of a combination of certain values causes an error

Testing combinations of input can be a challenge, as the number of combinations can be huge and impossible to test completely

Selecting a subset of combinations needs to be done with consideration

Decision tables are good for testing combinations of inputs which result in different actions being taken

A systematic way of stating complex business logic and rules

A decision table is made up of conditions, condition alternatives, actions and action entries.

A decision table captures all combinations of variables and possible outcomes.

Find subsystem or function with behavior that reacts to a combination of inputs, but without too many inputs

Identify aspects that need to be combined and put them into a table listing all combinations of true and false for each of the aspects

The number of inputs determines the number of combinations, so tackle small sets of combinations one at a time

Number of conditions / input	Number of columns / rules
1	2
2	4
3	8
4	16
5	32

Number of columns / rules = $2^{conditions}$

Enter conditions along with actions / outcomes in table and determine true and false values

Input-output behavior is transformed into a Boolean function

Decision tables help to find alternative scenarios and discover omissions and ambiguities

One test for each column or rule

Full table gives overview of combinations for selection

Helps to decide which combinations to test

Might test a combination of things otherwise not tested and find defects

With many combinations, prioritize and test the most important combinations

STATE TRANSITION MODELS

Used when some aspects of a system can be described in a finite state machine

For systems where output changes with same input, depending on what has happened before

For systems, classes and methods, where output can be different with the same input depending on earlier actions

State transition models are good tools when system must remember something about that has happened before

A state transition model has four basic parts
- The states that the software may occupy
- The transitions between states
- The events that cause a transition
- The actions that result from a transition

Test cases can test different states and transitions derived from state graph

Draw state transition model with states and transitions, then create tests based on the state transition model

Valid and invalid orders of operations exist
Test cases are designed to execute valid and invalid state transitions
For a system, state-based testing compares the resulting state of the system with the expected state
For a class, state-based testing compares the resulting state of the class with the expected state

Tests can be derived from state transition diagram to test sequences of states and transitions.
As a minimum, every state and transition should be tested and sequence length tested determines switch coverage.
- If all individual transitions are covered, there is 0 switch coverage
- If all transitions of length one and two are covered, there is 1 switch coverage

0 switch coverage refers to testing the individual transitions
1 switch coverage means that pairs of transitions are tested

JUNIT5 PROJECT

Set up a JUnit 5 project with params and hamcrest…

Dependencies

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.3.2</version>
</dependency>
<dependency>
  <groupId>org.junit.platform</groupId>
  <artifactId>junit-platform-runner</artifactId>
  <version>1.3.2</version>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-params</artifactId>
  <version>5.3.2</version>
</dependency>
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest-core</artifactId>
  <version>2.1</version>
</dependency>
<dependency>
  <groupId>org.hamcrest</groupId>
  <artifactId>hamcrest-library</artifactId>
  <version>2.1</version>
</dependency>
```

Plugin

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>2.22.1</version>
</plugin>
```

DATA-DRIVEN TESTING

Software testing methodology
Data-driven testing is the creation of test scripts to run together with their related data sets in a framework
The data comprises variables used for both input values and output verification values
Testing done using a table with test inputs and verifiable outputs
Supplied inputs from a row in the table expects the output which occur in the same row

REPEATED TESTS

Repeat a test a specified number of times with annotation @RepeatedTest and specifying with value the total number of repetitions desired for the test

A custom display name can be configured for each repetition via the name attribute of the @RepeatedTest annotation

Furthermore, the displayname can be a pattern composed of a combination of static text and dynamic placeholders

{displayName}: display name of the @RepeatedTest method
{currentRepetition}: the current repetition count
{totalRepetitions}: the total number of repetitions

In order to retrieve information about the current repetition and the total number of repetitions programmatically, a developer can choose to have an instances, such as TestInfo and RepetitionInfo injected into the @RepeatedTest, @BeforeEach, or @AfterEach methods

PARAMETERIZED TESTS

Parameterized tests make it possible to run a test multiple times with different arguments.
Parameterized tests are ideal when inputs can be compared to known expected results

Run the same test many times, but with different values?

Use parameterized tests!

Example: Triangle program parameters

ID	Test Case Description	Test Case Input			Expected Output
		a	b	c	
1	Valid scalene triangle	5	3	4	Scalene
2	Valid isosceles triangle	3	3	4	Isosceles
3	Valid equilateral triangle	3	3	3	Equilateral
4	First permutation of two equal sides	50	50	25	Isosceles
5	Second permutation of two equal sides	25	50	50	Isosceles
6	Third permutation of two equal sides	50	25	50	Isosceles
7	One side zero length	1000	1000	0	Invalid
8	One side has negative length	3	3	-4	Invalid
9	Three sides greater than zero, sum of two smallest is equal to the largest	1	2	3	Invalid
10	2nd permutation of 9	1	3	2	Invalid
11	3rd permutation of 9	3	1	2	Invalid
12	Three sides greater than zero, sum of two smallest is less than the largest?	2	5	8	Invalid
13	2nd permutation of 12	2	8	5	Invalid
14	3rd permutation of 12	8	5	2	Invalid
15	All sides zero	0	0	0	Invalid

Parameterized tests make it possible to run a test multiple times with different arguments.
They are declared just like regular @Test methods but use the @ParameterizedTest annotation instead.

In addition, at least one source must be declared that will provide the arguments for each invocation.
Parameterized test methods typically consume arguments directly from the configured source

Different sources can be used for parameterized tests

- ValueSource — Specify array of values providing a single argument per parameterized test invocation
- EnumSource — Use Enum constants for arguments
- CsvSource — Express argument lists as comma-separated values
- CsvFileSource — Read CSV files for arguments
- MethodSource — Refers to one or more factory methods of the test class or external classes

DYNAMIC TESTS

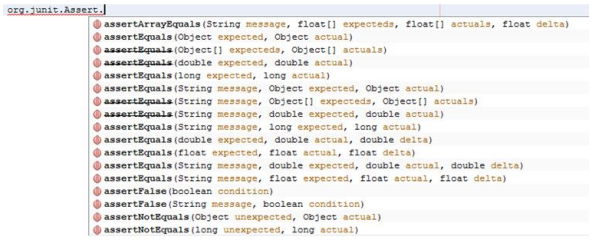
Test generated during runtime by a factory method annotated with the *@TestFactory* annotation
A @TestFactory method is not itself a test case but rather a factory for test cases
A dynamic test is the product of a factory
Does not support lifecycle callbacks, meaning that the @BeforeEach and the @AfterEach methods will not be called for the DynamicTests.

HAMCREST

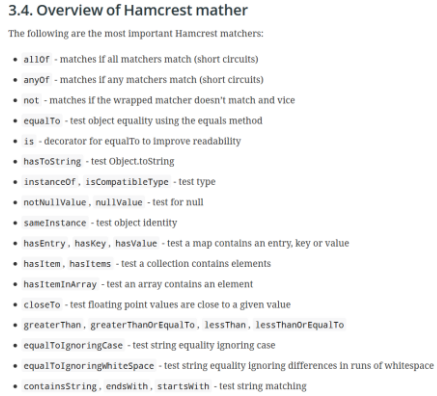
Hamcrest is a framework for software tests
Hamcrest comes bundled with lots of useful matchers and it is possible to create custom matchers
Assertion library with matchers to be used in tests instead of asserting with JUnit asserts
Hamcrest is a framework that assists writing software tests and writing matcher objects allowing 'match' rules to be defined declaratively instead of imperatively

JUnit Asserts / Hamcrest matchers
Write JUnit tests using Hamcrest matchers

But why would we use Hamcrest? What's the problem with all the asserts we get for free with JUnit?
One of the problems lies in the word all and also that they are not very readable



These are a bit more clear



When it comes to checks that are somewhat more complex the advantage becomes more visible

Declaratively / Imperatively

There is a great trend these days in the software community to favor declarative strategies in favor of imperative strategies

Declarative:	Describe what we want
Imperative:	Describe what to do to get what we want

Using Hamcrest should improve tests and provide higher readability and better error messages over JUnit asserts

RESOURCES

JUnit
https://junit.org/junit5/docs/5.3.0/user-guide/index.pdf

Decision tables
https://www.guru99.com/decision-table-testing.html

State transition models
https://www.guru99.com/state-transition-testing.html

Data driven testing
https://en.wikipedia.org/wiki/Data-driven_testing
http://xunitpatterns.com/Data-Driven%20Test.html

Repeated tests
https://www.baeldung.com/junit-5-repeated-test

Parameterized tests
https://www.baeldung.com/parameterized-tests-junit-5
https://blog.codefx.org/libraries/junit-5-parameterized-tests/

Dynamic tests
https://www.baeldung.com/junit5-dynamic-tests
https://blog.codefx.org/libraries/junit-5-dynamic-tests/

Hamcrest
http://hamcrest.org/
https://code.google.com/archive/p/hamcrest/wikis/Tutorial.wiki
https://www.javacodegeeks.com/2015/11/hamcrest-matchers-tutorial.html
https://www.vogella.com/tutorials/Hamcrest/article.html#overview-of-hamcrest-mather