# JUNIT Document

A theory is a kind of a JUnit test but different from the typical example-based JUnit tests, where we assert a specific data set and expect a specific outcome. JUnit theories are an alternative to JUnit's parameterized tests. A JUnit theory encapsulates the tester's understanding of an object's universal behavior. This means whatever a theory asserts is expected to be true for all data sets. Theories are useful for finding bugs in boundary-value cases

Parameterized tests allow us to write flexible data-driven tests and separate data from the test methods. Theories are similar to parameterized tests—both allow us to specify the test data outside of the test case.

Parameterized tests are good but they have the following drawbacks:

* Parameters are declared as member variables. They pollute the test class and unnecessarily make the system complex.
* Parameters need to be passed to the single constructor or variables need to be annotated, simply making the class incomprehensible.
* Test data cannot be externalized.

Theory comes up with many annotations and a runner class. Let's examine the important annotations and classes in theory, as follows:

* @Theory: Like @Test, this annotation identifies a theory test to run. The @Test annotation doesn't work with a theory runner.
* @DataPoint: This annotation identifies a single set of test data (similar to @Parameters), that is, either a static variable or a method.
* @DataPoints: This annotation identifies multiple sets of test data, generally an array.
* @ParametersSuppliedBy: This annotation provides the parameters to the test cases.
* Theories: This annotation is a JUnit runner for the theory-based test cases and extendsorg.junit.runners.BlockJUnit4ClassRunner.
* ParameterSupplier: This is an abstract class that gives us the handle on the parameters that we can supply to the test case.

## Dealing with JUnit rules

Rules allow very flexible addition or redefinition of the behavior of each test method in a test class. Rules are like **Aspect Oriented Programming** (**AOP**); we can do useful things before and/or after the actual test execution.

**Integration tests**

<http://zeroturnaround.com/rebellabs/the-correct-way-to-use-integration-tests-in-your-build-process/>