JUnit cheat sheet

Assertions and assumptions

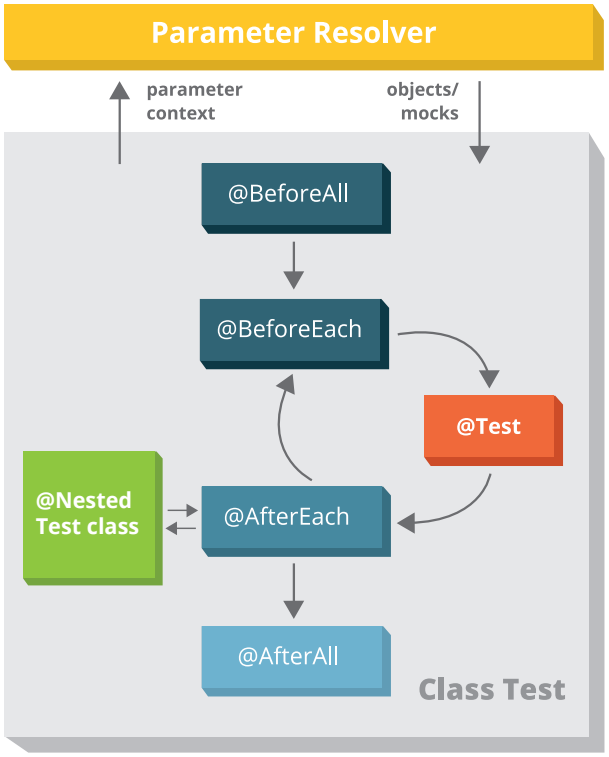
Use assertions to verify the behavior under test:

```
Assertions.assertEquals(Object expected,
Object actual, Supplier<String> message)

// Group assertions are run all together and
reported together.
Assertions.assertAll("heading",
  () -> assertTrue(true),
  () -> assertEquals("expected",
              objectUnderTest.getSomething());

// To check for an exception:
expectThrows(NullPointerException.class,
  () -> { ((Object) null).getClass();});
```

Lifecycle of standard tests



Parameter resolution

ParameterResolver - extension interface to provide parameters

```
public class MyInfoResolver implements ParameterResolver {
  public Object resolve(ParameterContext paramCtx,
                        ExtensionContext extCtx) {
    return new MyInfo();
  }
}
```

Extend your tests with your parameter resolver.

```
@ExtendWith(MyInfoResolver.class)
class MyInfoTest { … }
```

Useful code snippets

```
@TestFactory
Stream<DynamicTest> dynamicTests(MyContext ctx) {
// Generates tests for every line in the file
 return Files.lines(ctx.testDataFilePath).map(l ->
dynamicTest("Test:" + l, () -> assertTrue(runTest(l)));
}

@ExtendWith({ MockitoExtension.class,
DataParameterProvider.class })
class Tests {
  ArrayList<String> list;

  @BeforeEach
  void init() { /* init code */ }

  @Test
  @DisplayName("Add elements to ArrayList")
  void addAllToEmpty(Data dep) {
    list.addAll(dep.getAll());
    assertAll("sizes",
      () -> assertEqual(dep.size(), list.size(),
              () -> "Unexpected size:" + instance),
      () -> assertEqual(dep.getFirst(), list.get(0),
              () -> "Wrong first element" + instance));
  }
}
```

Useful annotations

@Test - marks a test method
@TestFactory - method to create test cases at Runtime
@DisplayName - make reports readable with proper test names
@BeforeAll/@BeforeEach - lifecycle methods executed prior testing
@AfterAll/AfterEach - lifecycle methods for cleanup
@Tag - declare tags to separate tests into suites
@Disabled - make JUnit skip this test.

Use **@Nested** on an inner class to control the order of tests.

Use **@ExtendWith()** to enhance the execution: provide mock parameter resolvers and specify conditional execution.

Use the lifecycle and **@Test** annotations on the default methods in interfaces to define contracts:

```
@Test
interface HashCodeContract<T> {
  <T> getValue();
  <T> getAnotherValue();
  @Test
  void hashCodeConsistent() {
    assertEquals(getValue().hashCode(),
                 getAnotherValue().hashCode(),
                 "Hashes differ");
}
```

"Never trust a test you haven't seen fail."

— Colin Vipurs