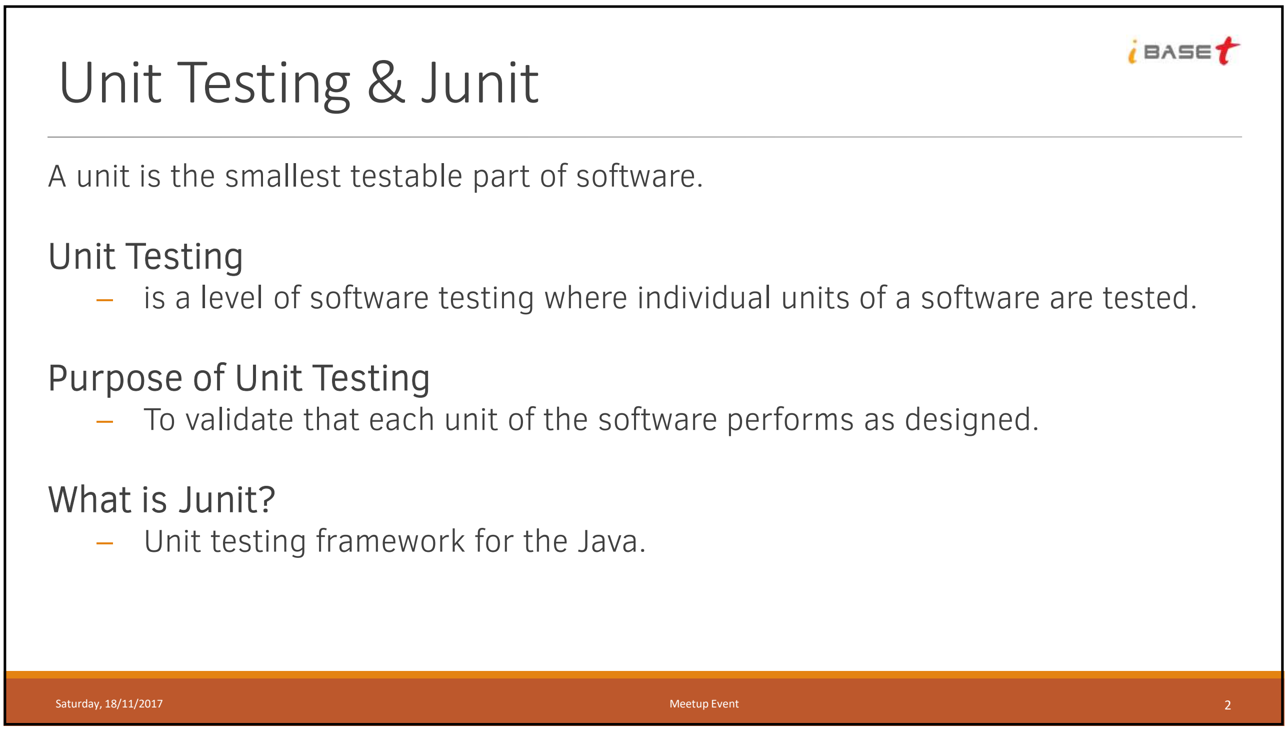
Unit Testing With Junit 5

The next generation of unit testing

SOFTWARE DEVELOPERS Meetup

ORGANISED BY: iBASEt



Unit Testing & Junit

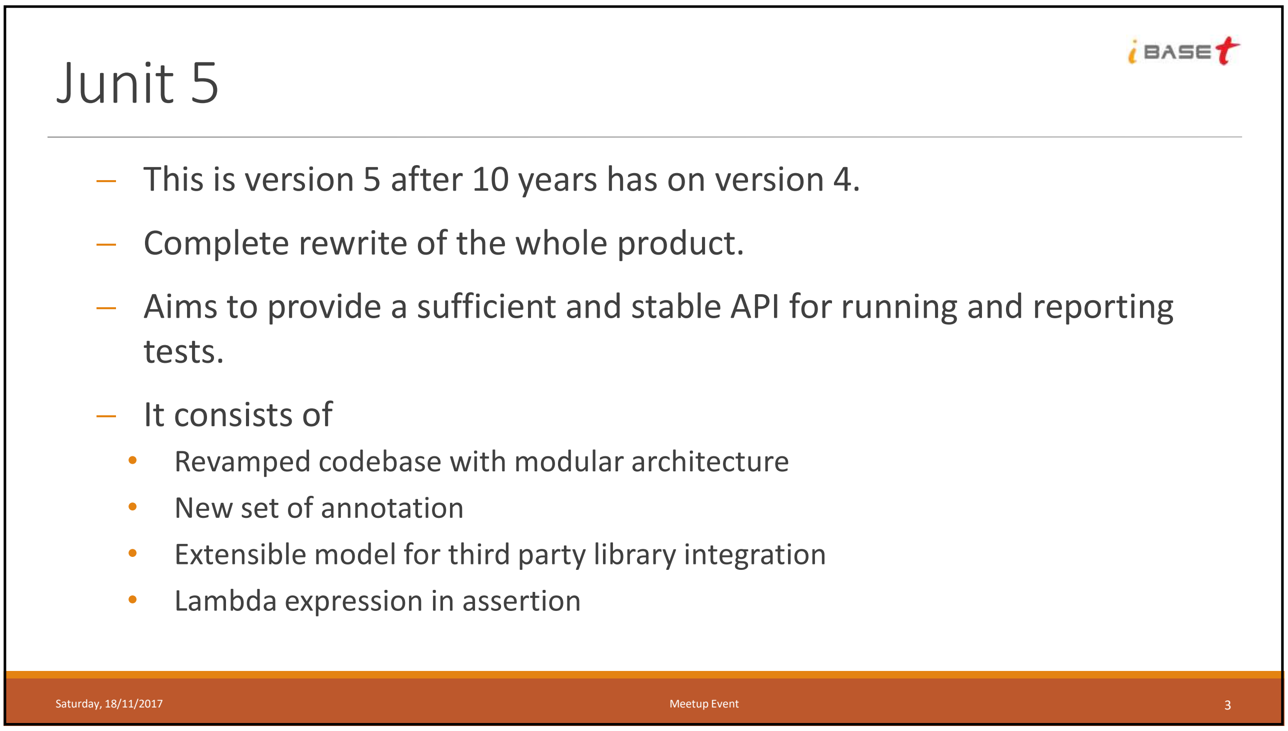
A unit is the smallest testable part of software.

Unit Testing
- is a level of software testing where individual units of a software are tested.

Purpose of Unit Testing
- To validate that each unit of the software performs as designed.

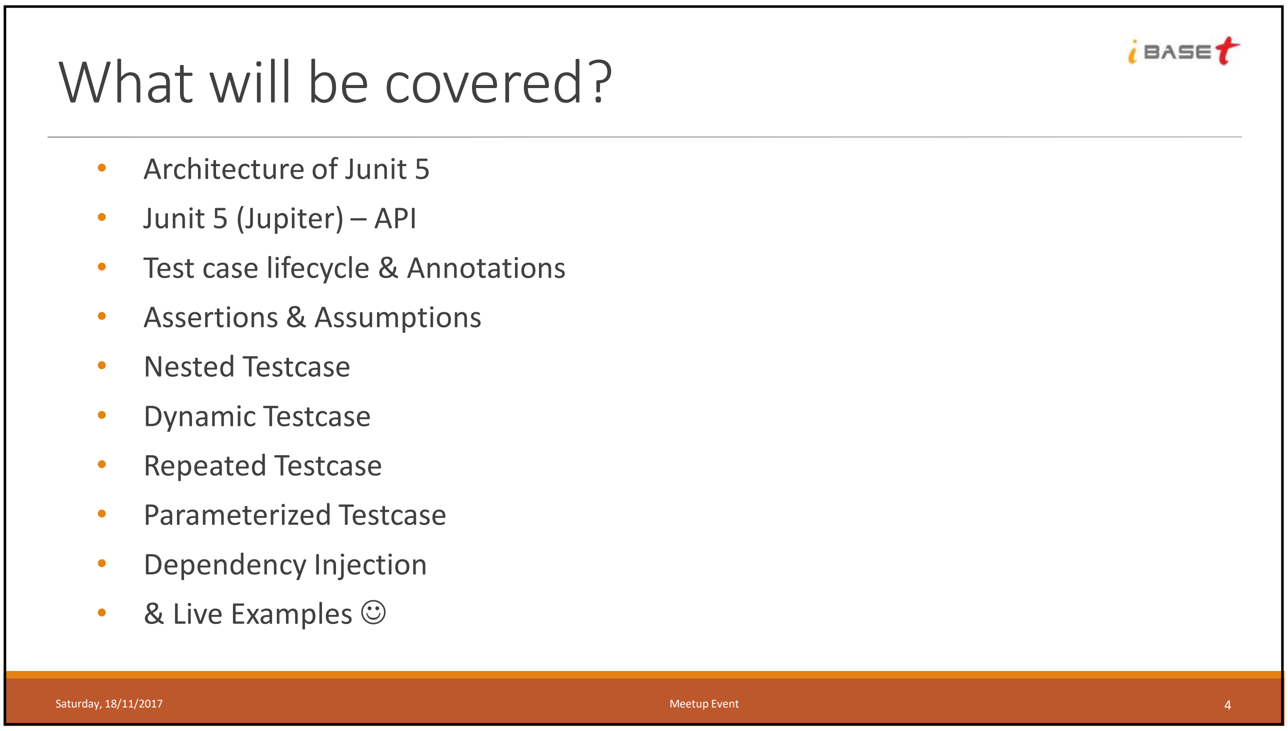
What is Junit?
- Unit testing framework for the Java.

Saturday, 18/11/2017 Meetup Event 2



Junit 5

- This is version 5 after 10 years has on version 4.
- Complete rewrite of the whole product.
- Aims to provide a sufficient and stable API for running and reporting tests.
- It consists of
  - Revamped codebase with modular architecture
  - New set of annotation
  - Extensible model for third party library integration
  - Lambda expression in assertion



What will be covered?

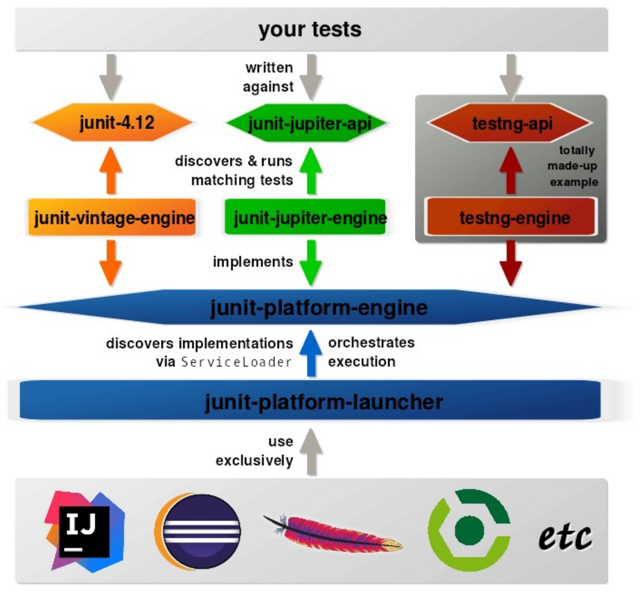
- Architecture of Junit 5
- Junit 5 (Jupiter) – API
- Test case lifecycle & Annotations
- Assertions & Assumptions
- Nested Testcase
- Dynamic Testcase
- Repeated Testcase
- Parameterized Testcase
- Dependency Injection
- & Live Examples ☺

Junit 5 - Architecture

Junit 5 = Platform + Jupiter + Vintage

- Junit Platform
  - Serves as a foundation for launching testing frameworks on the JVM.
  - Launcher and Test Engine API
  - Console Launcher, Gradle Plugin, Maven Surefire provider
- Junit Jupiter
  - Is the combination of the new programming model and extension model for writing tests and extensions in JUnit 5.
- Junit Vintage
  - Provides a Test Engine for running JUnit 3 and JUnit 4 based tests on the platform.

Architectural Overview



Test case – First Look

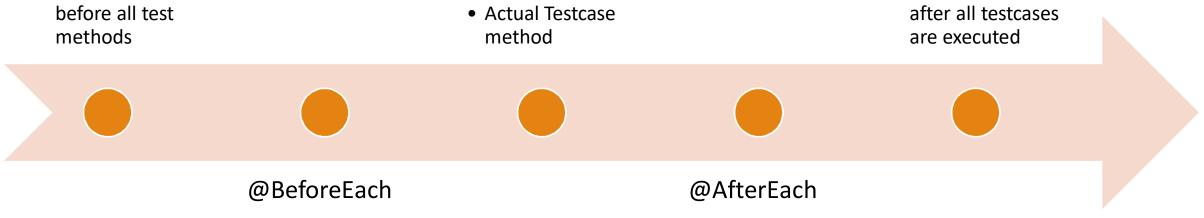
```
import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class FirstJUnit5Tests {

    @Test
    void myFirstTest() {
        assertEquals(2, 1 + 1);
    }

}
```

Test Lifecycle

@BeforeAll
- Identical to @BeforeClass
- Executes once before all test methods

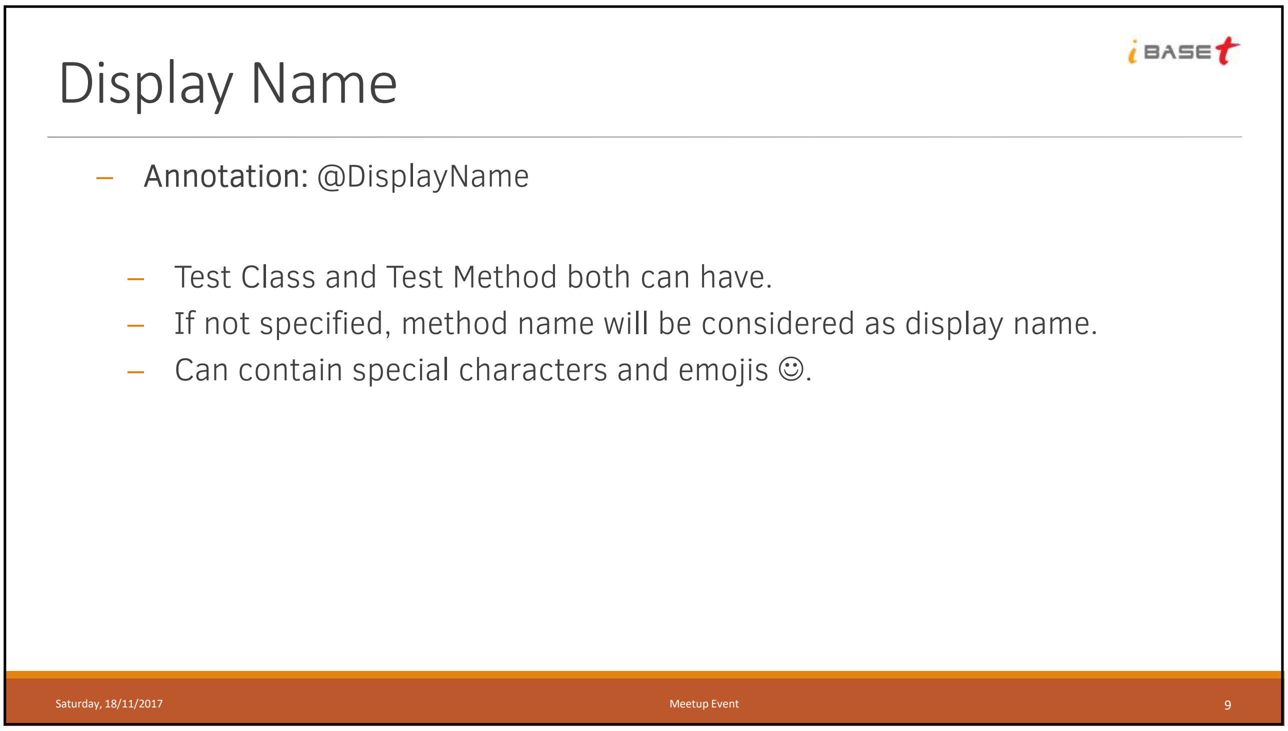
@Test
- Actual Testcase method

@AfterAll
- Identical to @AfterClass
- Executes once after all testcases are executed



@BeforeEach
- Identical to @Before
- Executes before each testcase

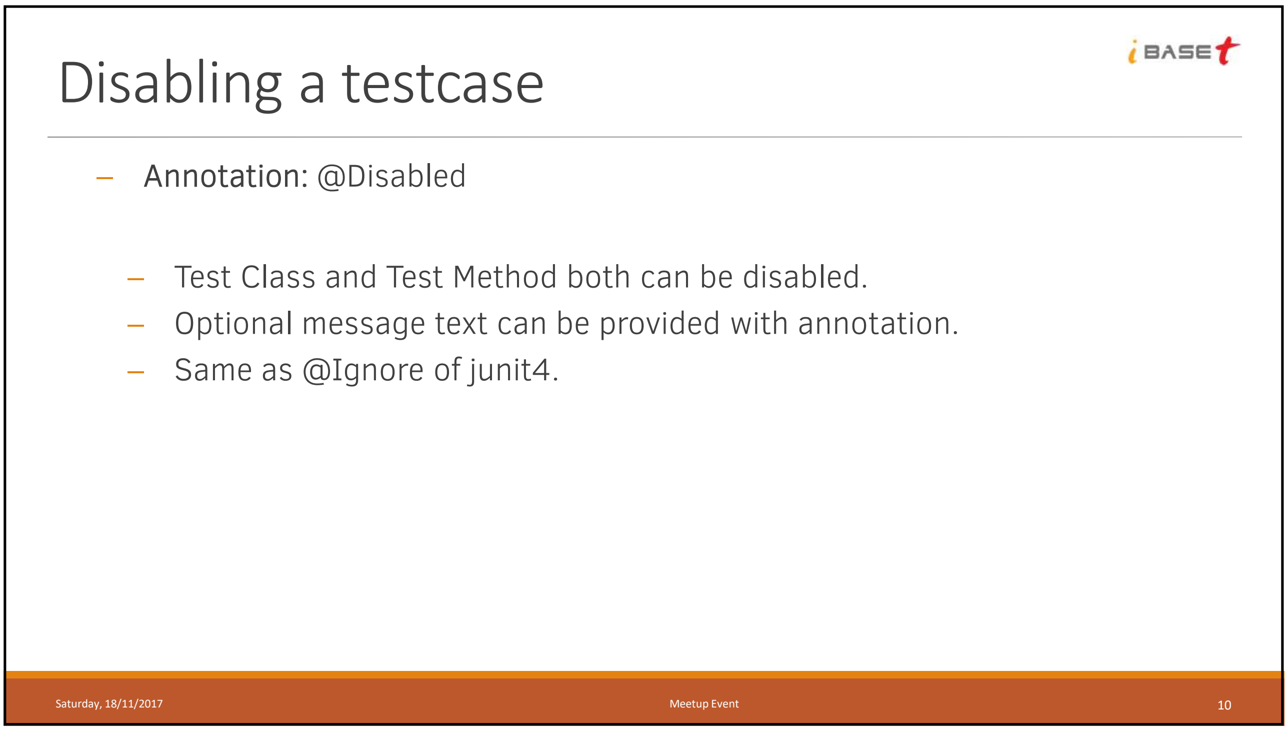
@AfterEach
- Identical to @After
- Executes after each testcase



Display Name

- Annotation: @DisplayName

  - Test Class and Test Method both can have.
  - If not specified, method name will be considered as display name.
  - Can contain special characters and emojis ☺.



Disabling a testcase

- Annotation: @Disabled

  - Test Class and Test Method both can be disabled.
  - Optional message text can be provided with annotation.
  - Same as @Ignore of junit4.

Tagging & Filtering

- Test classes and methods can be tagged using @Tag annotation
- Tagging can be used for test discovery and execution
- Tag must not be null or blank
- Trimmed
- Should not contain ISO control characters & reserved words like,
  & | !) (,

Assertions

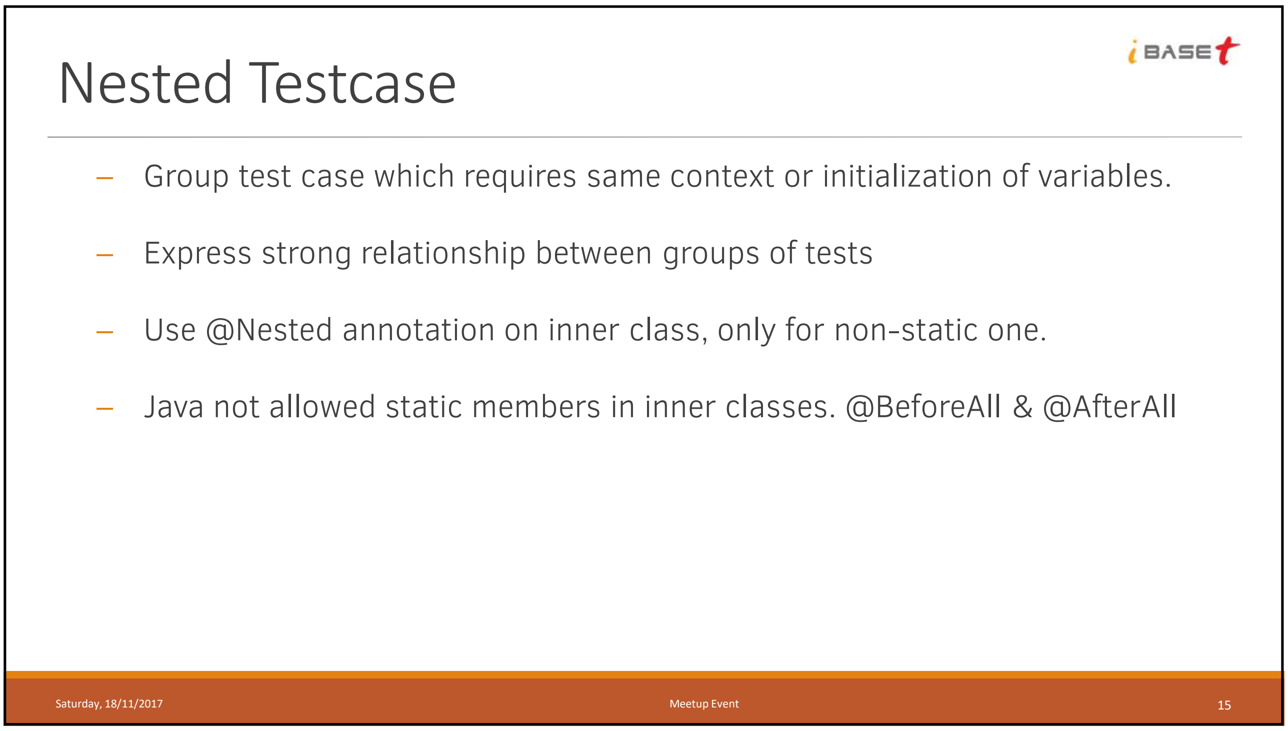
- **Package:** org.junit.jupiter.Assertions
  - **Standard Assertions:** assertEquals(), assertTrue()
  - **Grouped Assertions:** assertAll()
  - **Dependent Assertions:** sequence of assertion statements
  - **Timeout Exceed Assertion:** assertTiemout()

Assumptions

- Package: org.junit.jupiter.Assumptions
  - assumeTrue()
  - assumeFalse()
  - assumingThat()

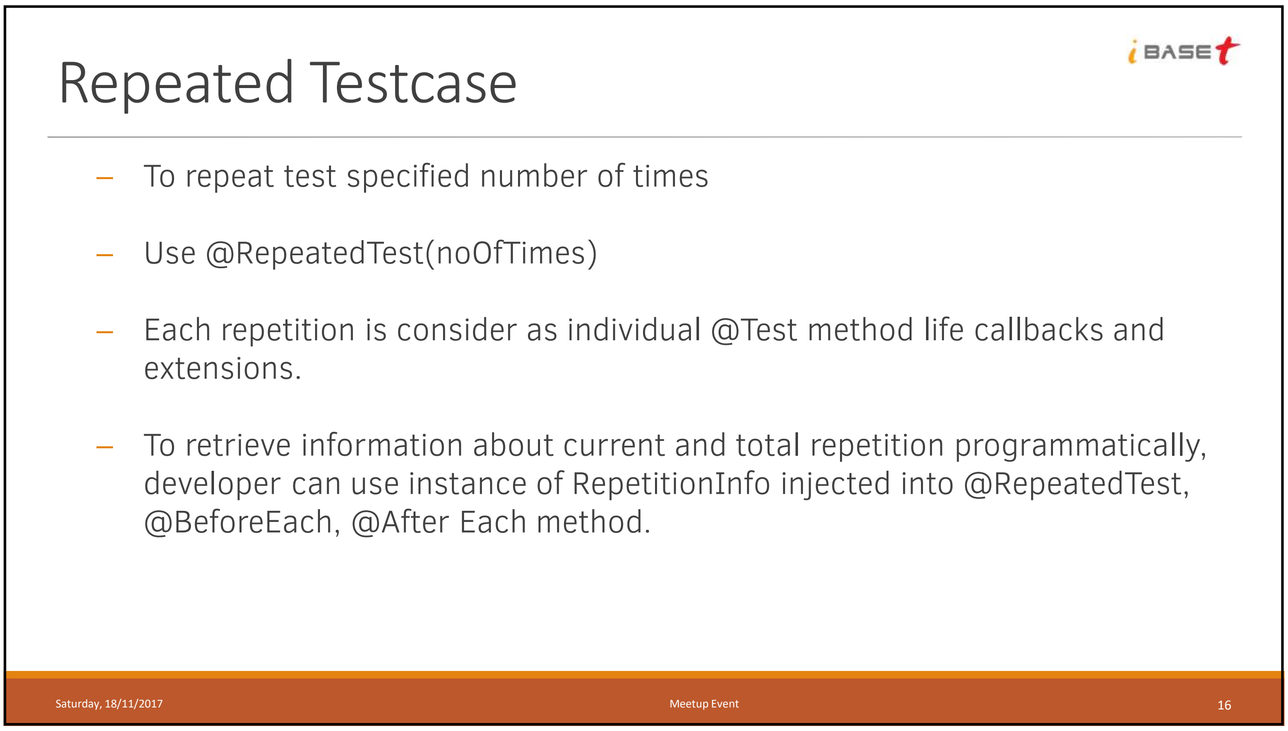
Composed Annotations

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

import org.junit.jupiter.api.Tag;

@Target({ ElementType.TYPE, ElementType.METHOD })
@Retention(RetentionPolicy.RUNTIME)
@Tag("fast")
public @interface Fast {
}

@Fast
```



Nested Testcase

- Group test case which requires same context or initialization of variables.
- Express strong relationship between groups of tests
- Use @Nested annotation on inner class, only for non-static one.
- Java not allowed static members in inner classes. @BeforeAll & @AfterAll



Repeated Testcase

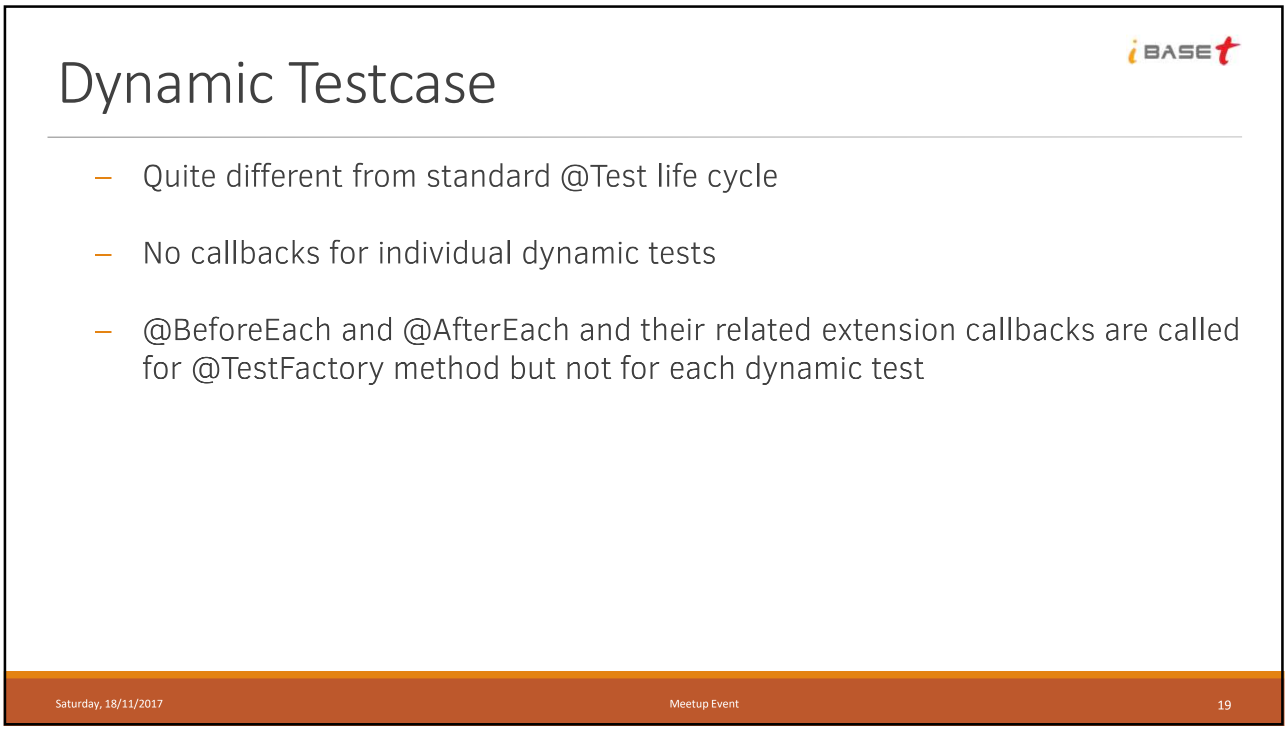
- To repeat test specified number of times
- Use @RepeatedTest(noOfTimes)
- Each repetition is consider as individual @Test method life callbacks and extensions.
- To retrieve information about current and total repetition programmatically, developer can use instance of RepetitionInfo injected into @RepeatedTest, @BeforeEach, @After Each method.

Parameterized Testcase

- To run test multiple times with different arguments
- Use *@ParameterizedTest* instead of *@Test* annotation
- Need to provide one source, which provides arguments to each invocation.
- Need to dependency for *junit-jupiter-params* artifact.
- Sources of Arguments
  - @ValueSource
  - @EnumSource
  - @MethodSource
  - @CsvSource
  - @CsvFileSource

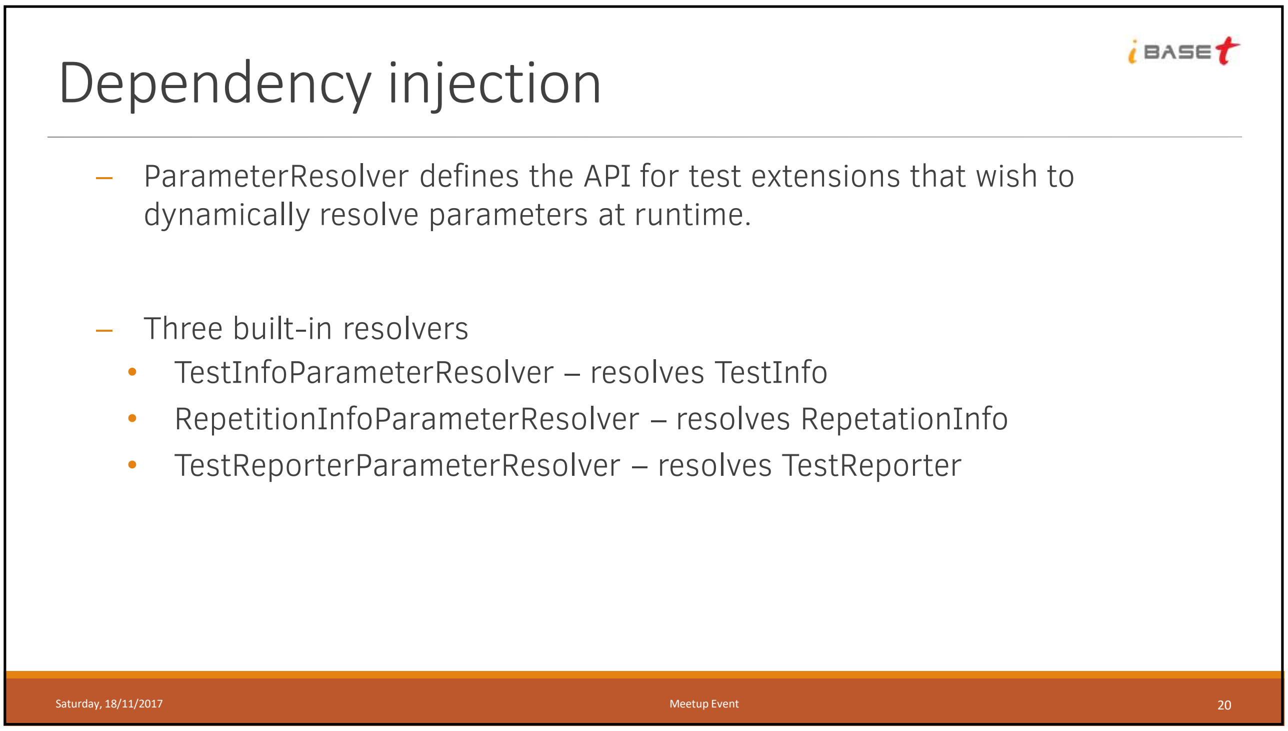
Dynamic Testcase

- Method annotated with @Test are fully specified at compile time and its behavior can't be changed at runtime.
- Dynamic Test case are generated at runtime by factory method that is annotated with @TestFactory
- @TestFactory is not a test case rather it is a factory of generating test cases at runtime.
- @TestFactory must return a Stream, Collection, Iterable, Iterator



Dynamic Testcase

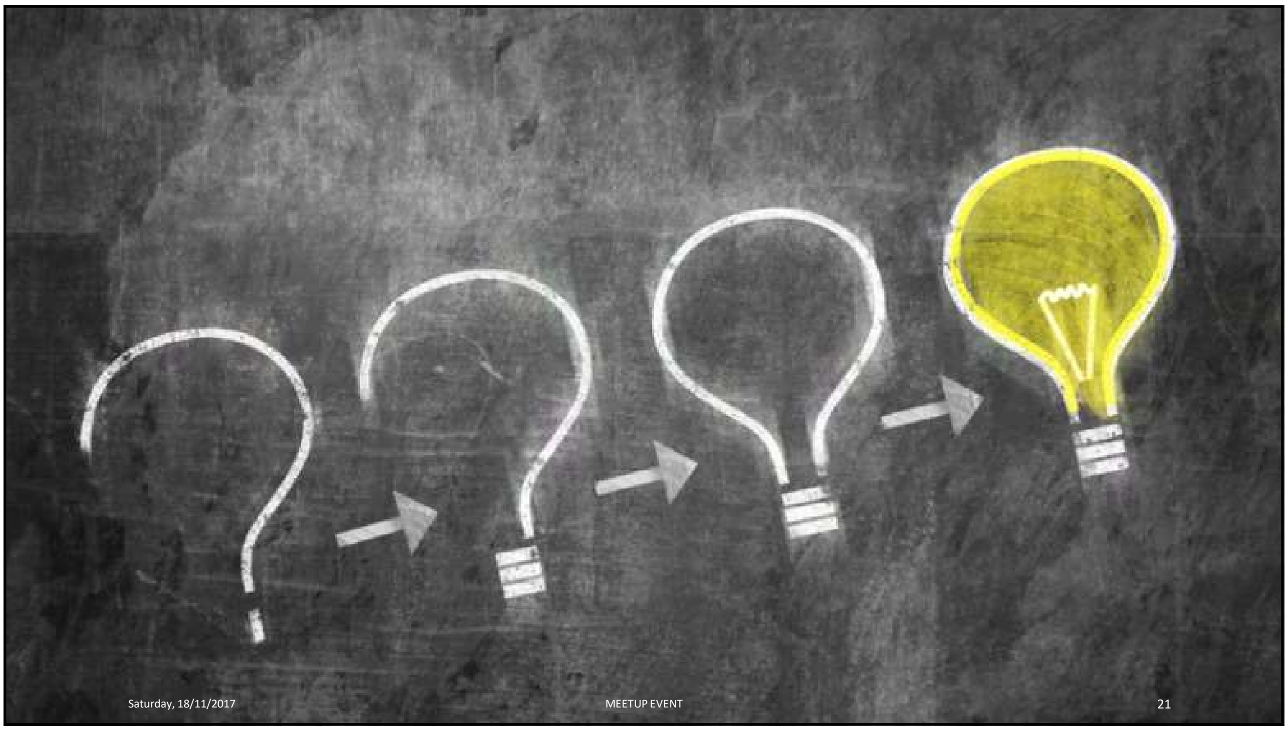
– Quite different from standard @Test life cycle
– No callbacks for individual dynamic tests
– @BeforeEach and @AfterEach and their related extension callbacks are called for @TestFactory method but not for each dynamic test



Dependency injection

– ParameterResolver defines the API for test extensions that wish to dynamically resolve parameters at runtime.

– Three built-in resolvers
  - TestInfoParameterResolver – resolves TestInfo
  - RepetitionInfoParameterResolver – resolves RepetationInfo
  - TestReporterParameterResolver – resolves TestReporter



Saturday, 18/11/2017 MEETUP EVENT 21

Thank You!
iBASEt
SOFTWARE DEVELOPERS
Meetup
</>
ORGANISED BY: iBASEt

Ankit, Parth, Ketul & Pravin

Saturday, 18/11/2017 Meetup Event