Ch2 Code Unit Testing

Write Code to Test Code(3)

Instructor: Haiying SUN

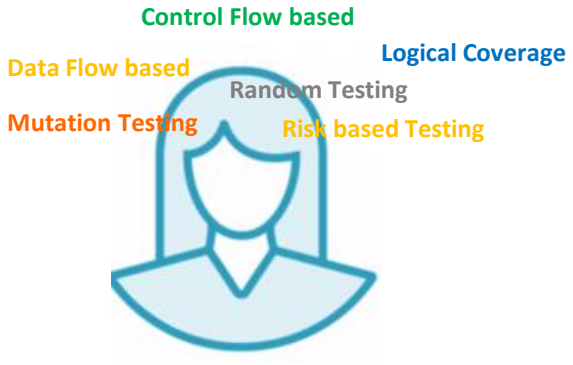
E-mail: hysun@sei.ecnu.edu.cn

Office: ECNU Science Build B1104

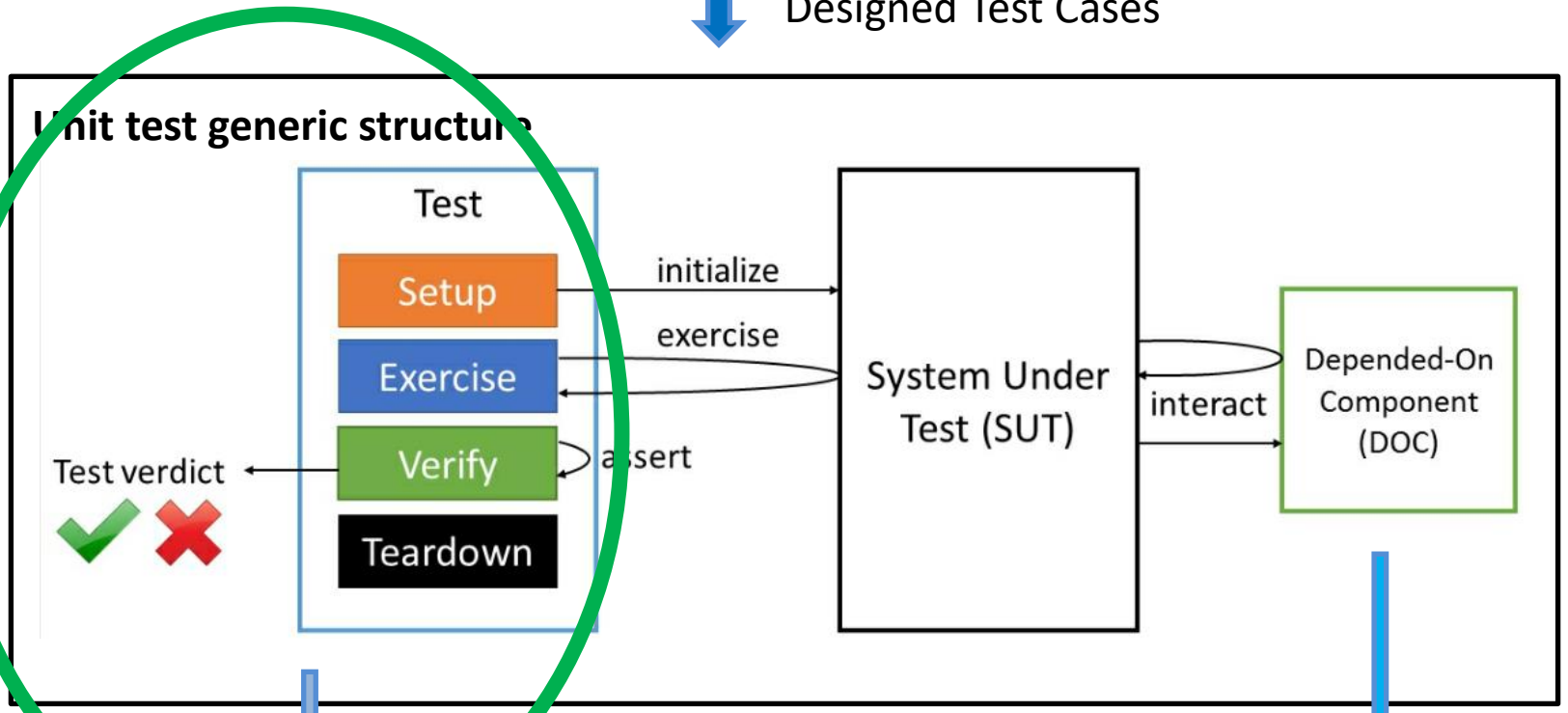
Available Time: Wednesday 8:00 -12:00 a.m.

Agenda

- Code Test Techniques
  - Logical Testing & Tools
  - Heuristic Rules
  - Junit & Qualified test scripts
- Code Test Generation
  - Control flow based
  - Data flow based
  - Mutation Based
  - Test Automation Tool Development



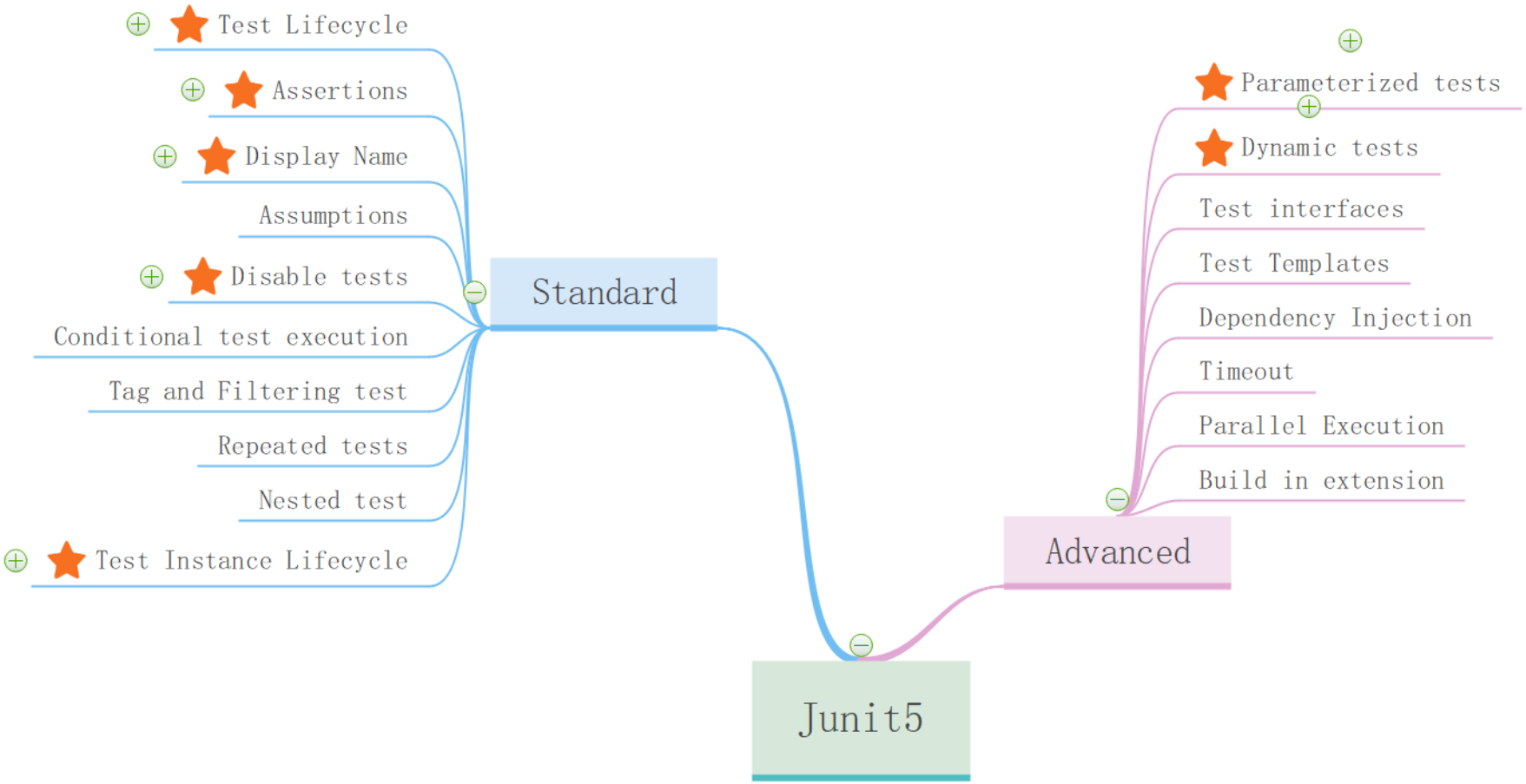
Designed Test Cases



JUnit 5

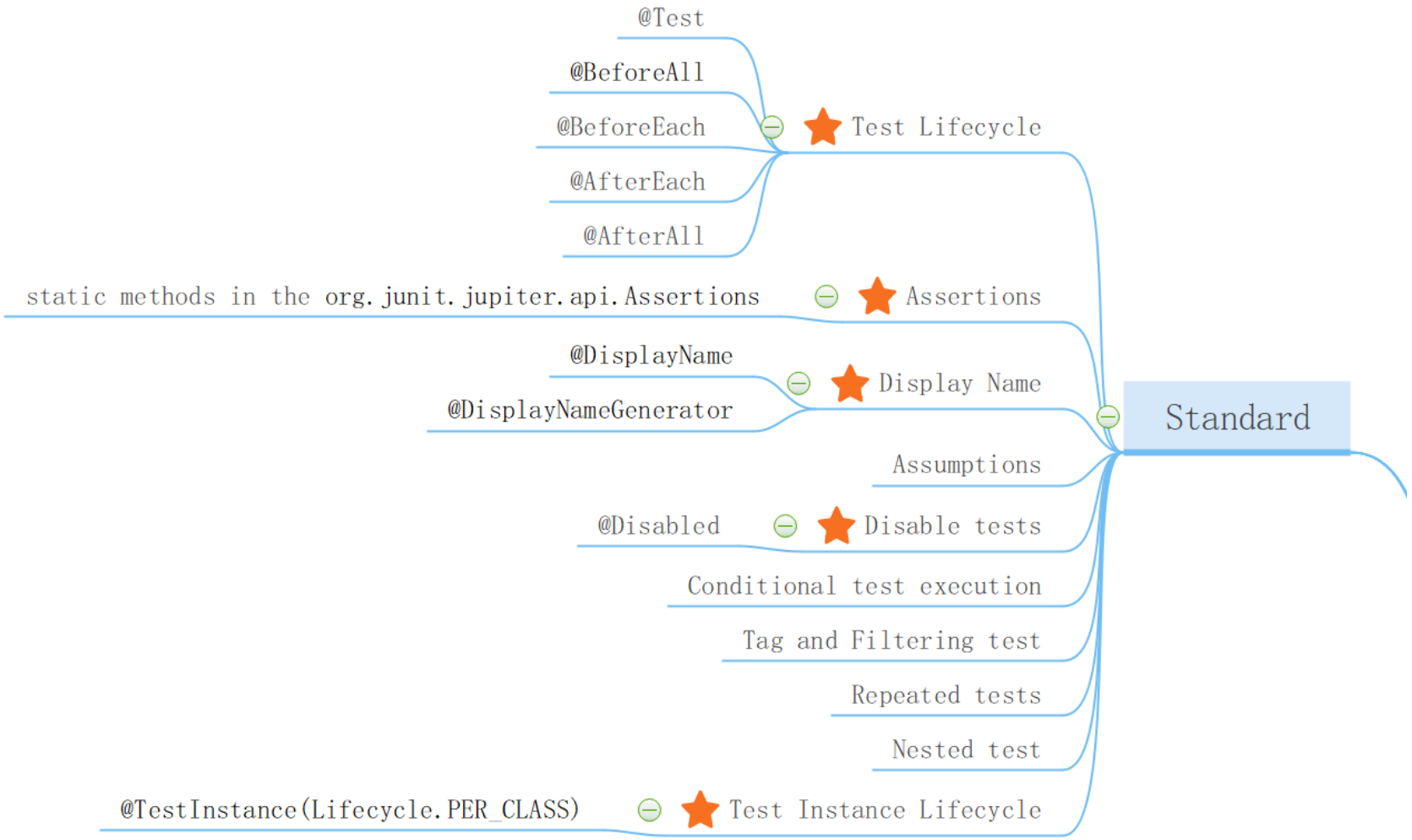
mockito

Junit5 Features

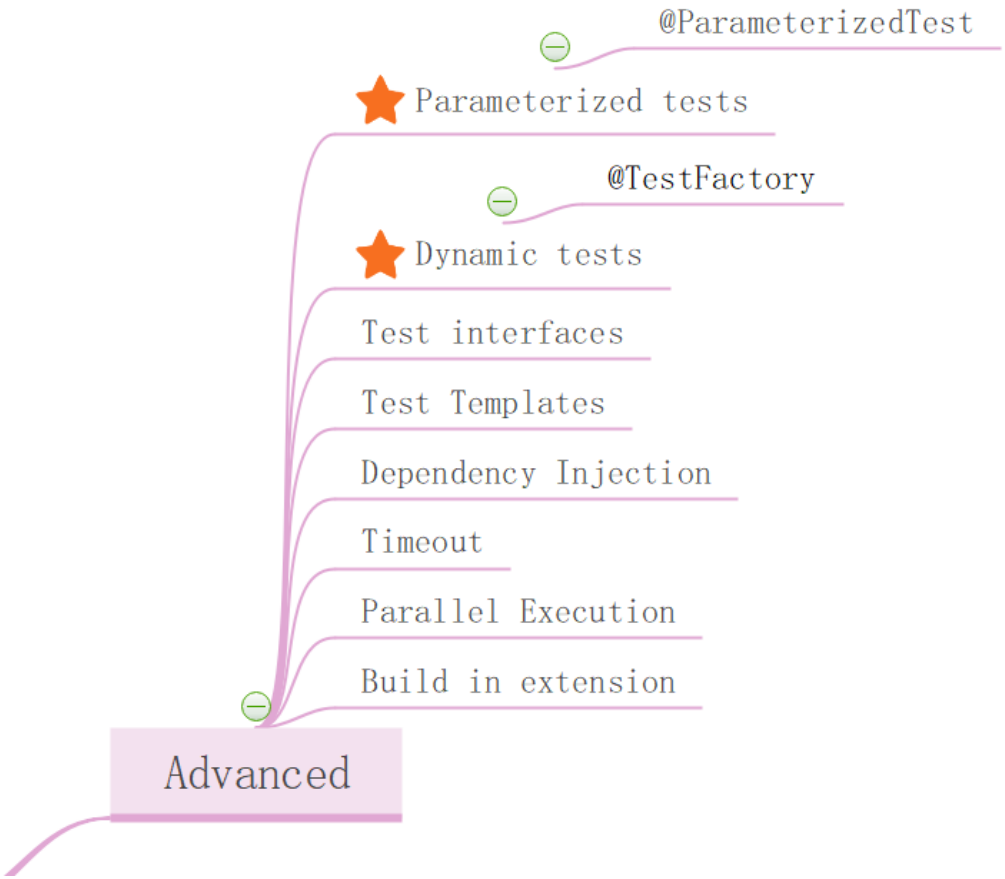


https://junit.org/junit5/docs/current/user-guide/

Junit5 Features



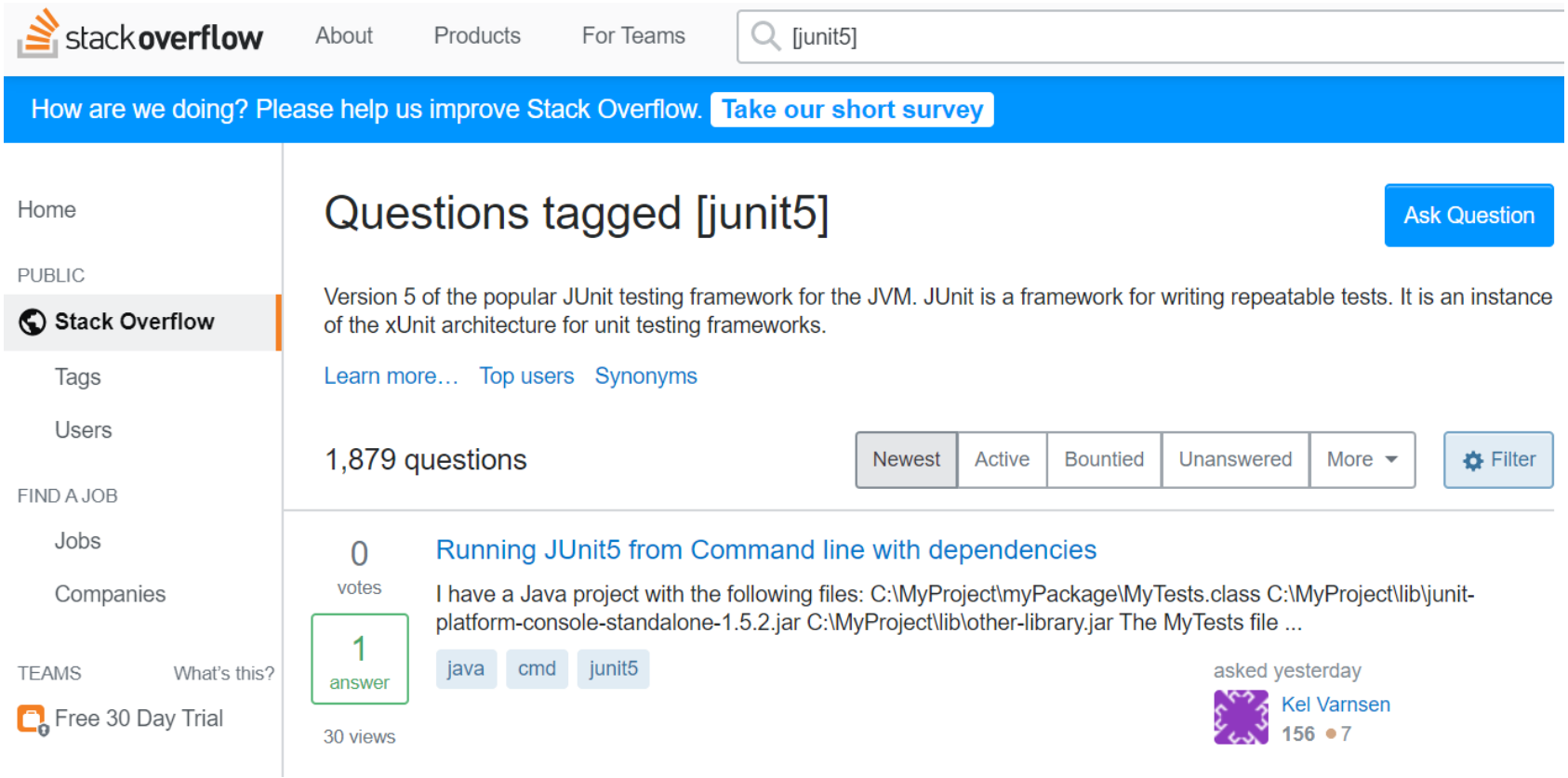
Junit5 Features



More about Junit5

Junit org： https://junit.org/junit5/

More About Junit5



Questions on stackoverflow : https://stackoverflow.com/questions/tagged/junit5

More About Junit5

Junit Gitter: an instant chat room which can talk directly with the Junit5 members

https://gitter.im/junit-team/junit5

Agenda

- Junit Foundations
- Test Lifecycle
- Assertions
- Parameterized tests

Overview

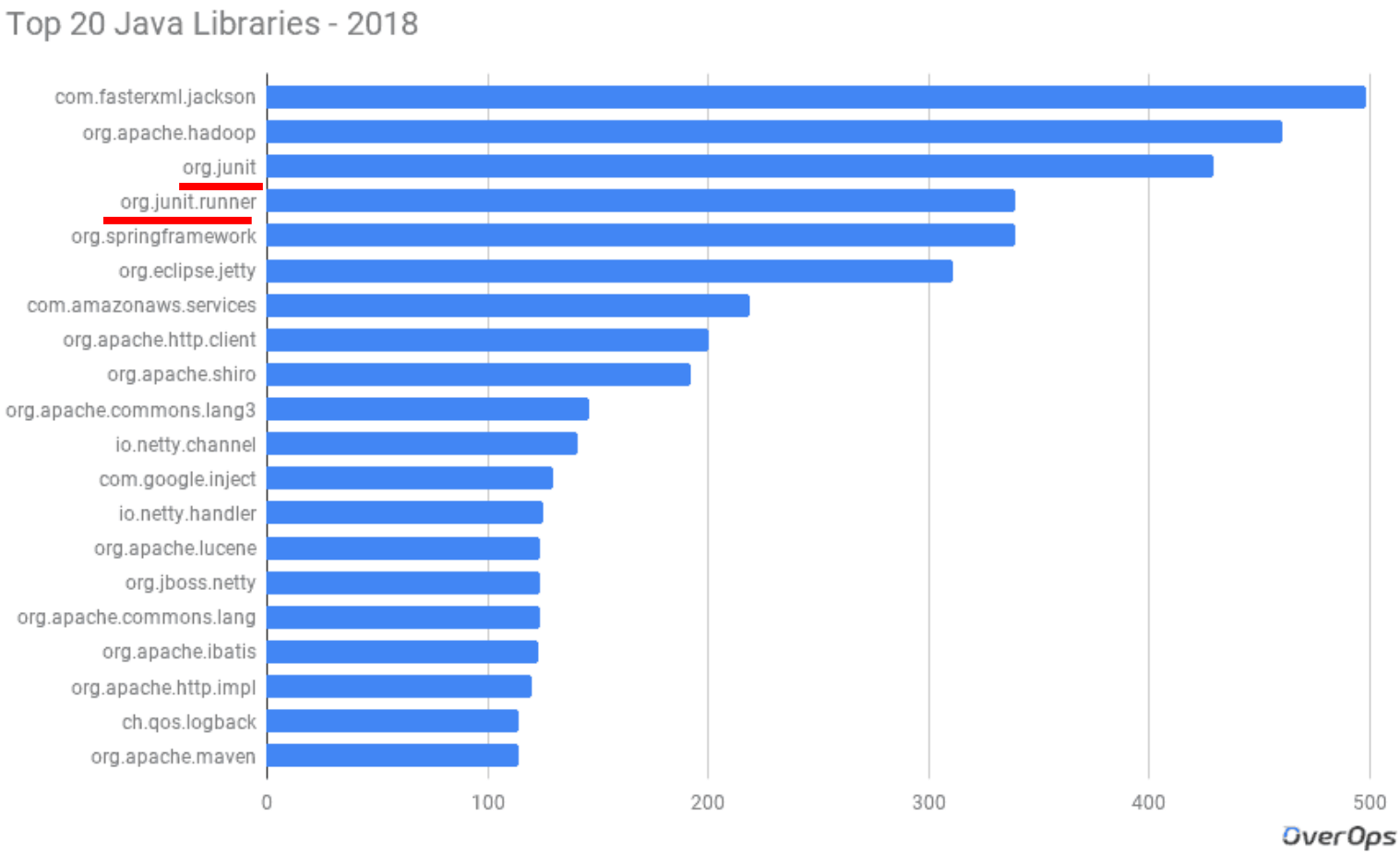
- Junit Foundations
- Test Lifecycle
- Assertions
- Parameterized tests

What is JUnit

JUnit is a simple, open source framework to write and run repeated tests

-- Org Junit

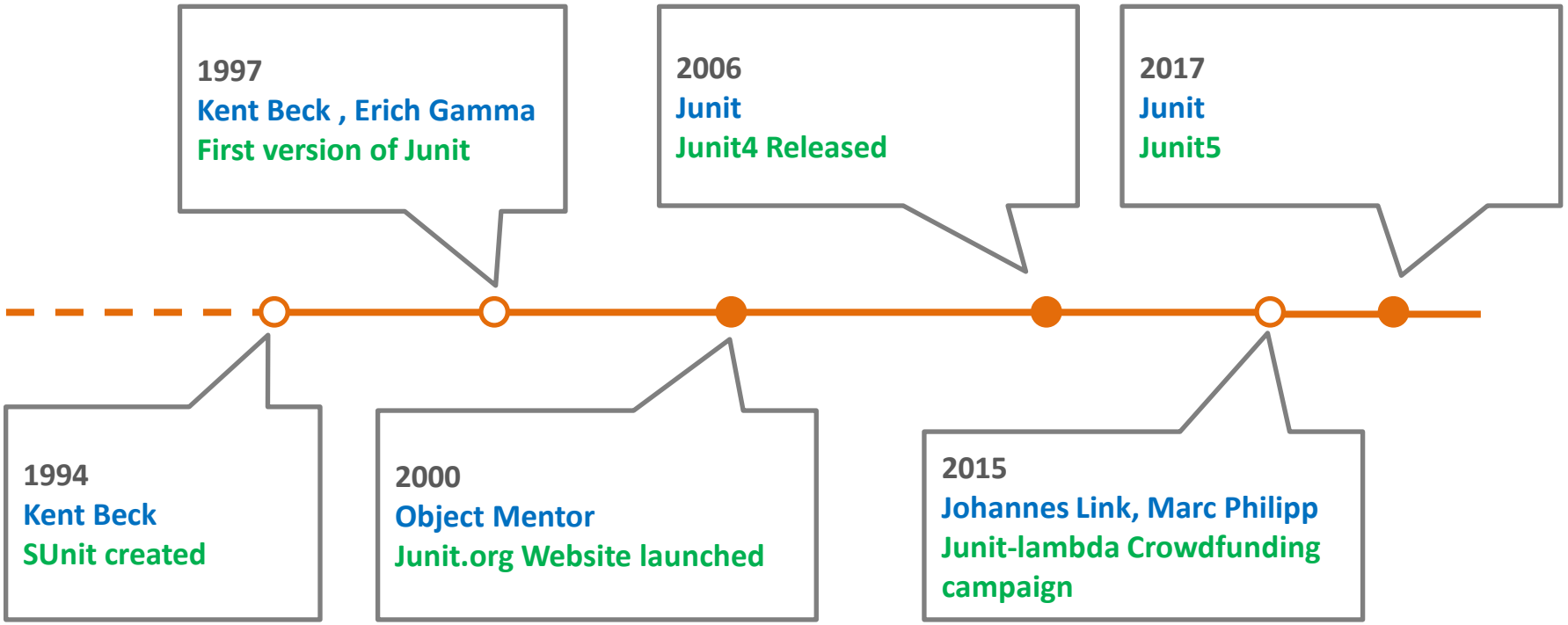
Top 20 Java Library (2018)



Based on 277,975 Github Source File

(https://blog.overops.com/the-top-100-java-libraries-in-2018-based-on-277975-source-files/)

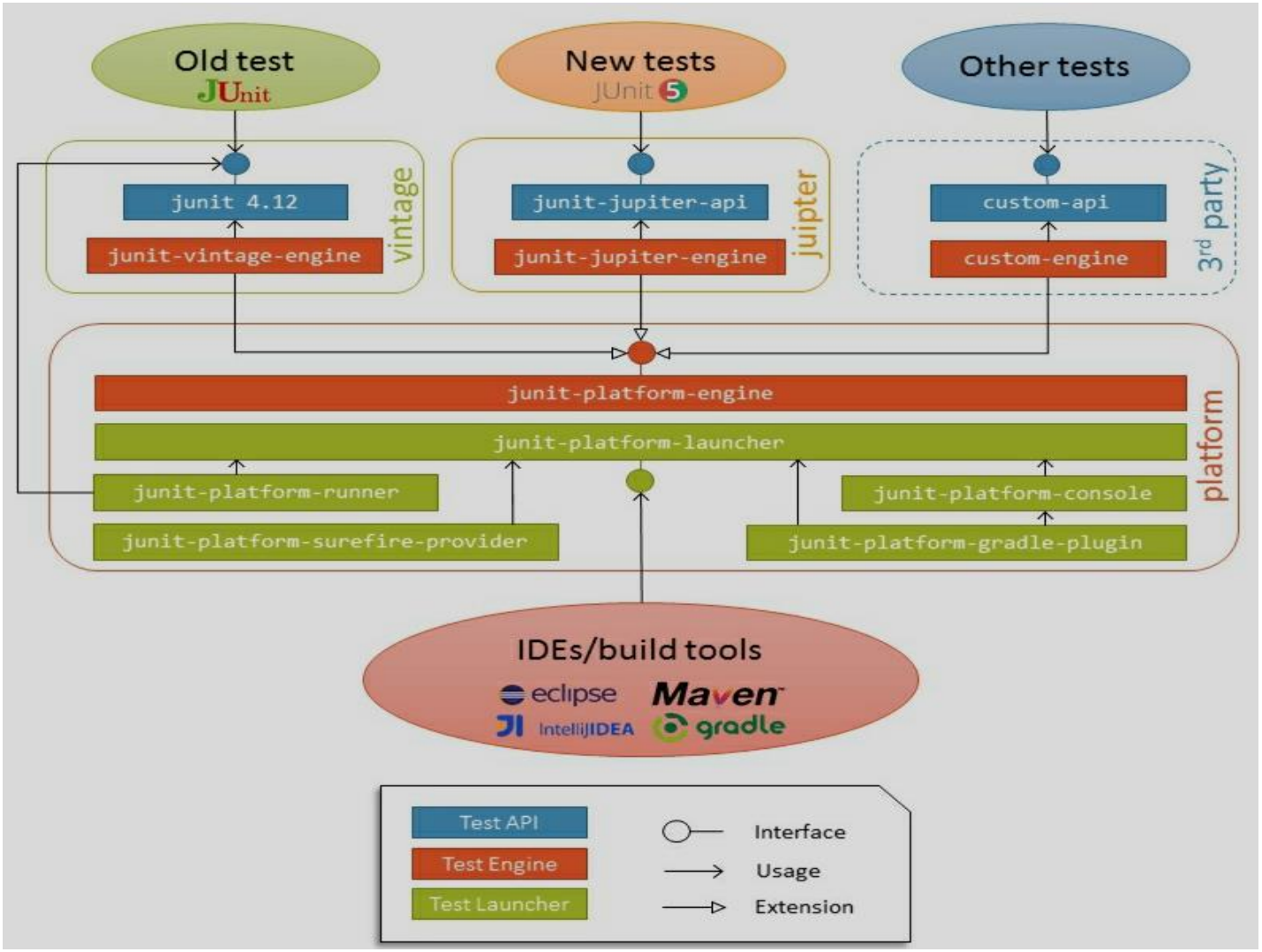
JUnit Timeline



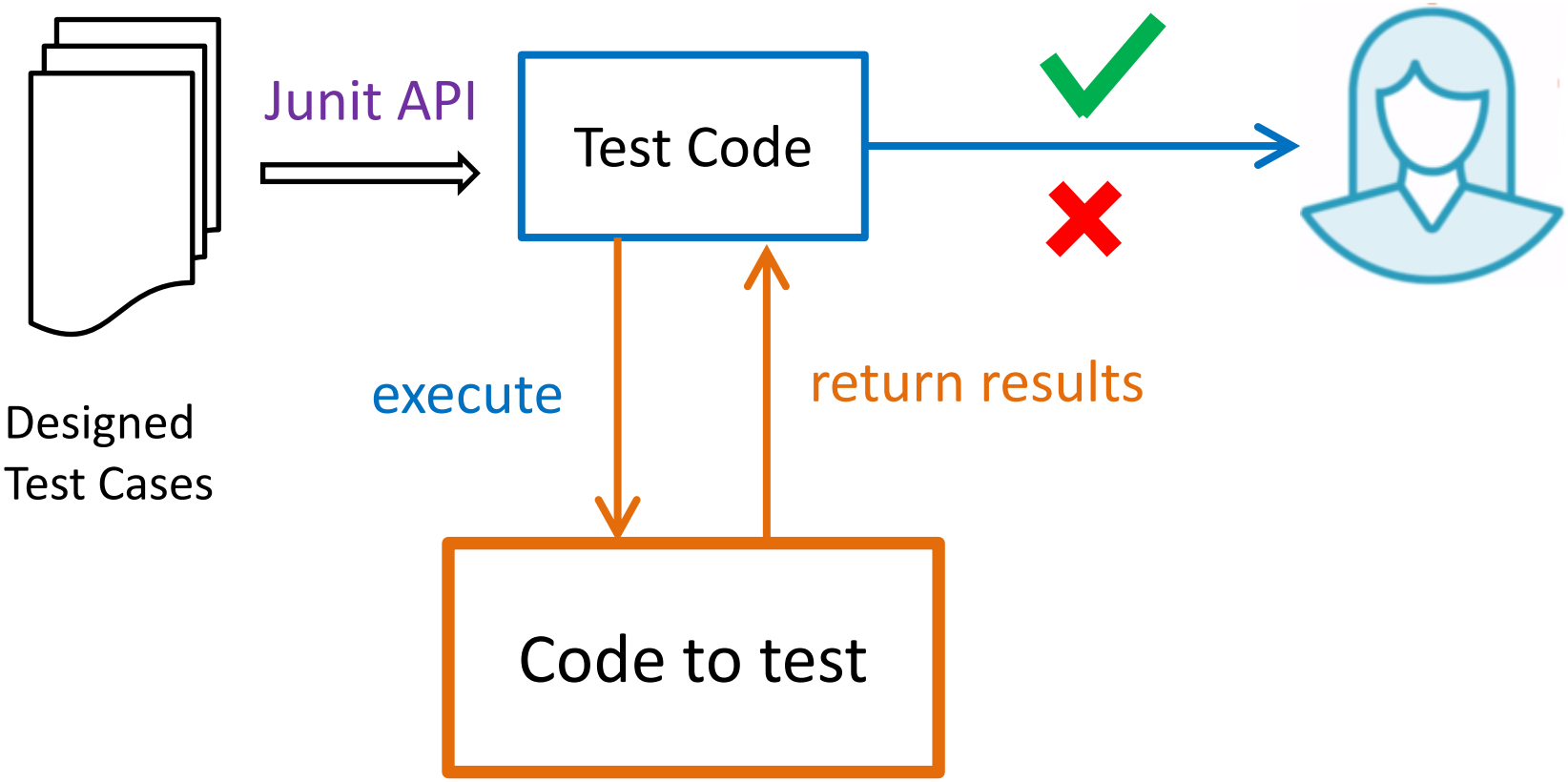
JUnit

- Junit3.8
  - Slogan: Keep your bar green to keep your code clean
- Junit4.x
  - TDD: Red -> Green -> Refactor
- Junit5
  - Ecosystem

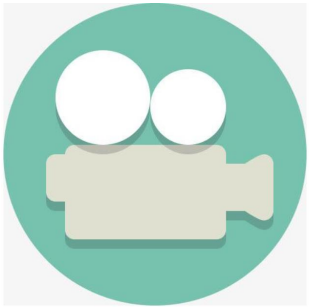
JUnit5 Ecosystem



JUnit Testing Process



Practice

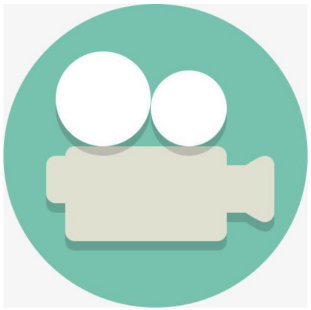


相关视频

4-The Very First Junit Test

- 实验准备
  1. 课程网站下载**MeetHereJava.zip**： 视频资料→ JUnit + mockito视频
  2. 解压**MeetHereJava. zip**到当前目录（**目录中不要有中文**）
  3. Idea中打开**MeetHereJava（注意JDK版本）**
- 实验任务1
  - 查看MeetHereJava项目结构，包括：
  1. 源代码目录src和测试代码目录test在项目结构中的位置
  2. src下的类和test下的类所属包的情况
  3. test目录下的类的命名
  4. MeetCalendarTest中使用@Test标注的方法的命名

Practice

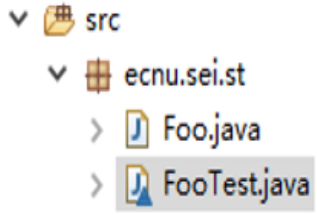


相关视频

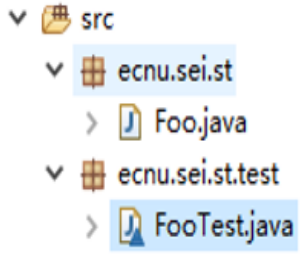
4-The Very First Junit Test

- 实验任务2
  1. 查看MeetCalendarTest代码
  2. 执行用于验证成功增加预约信息的测试方法addAnReservation()
- 实验任务3
  1. 为测试类MeetCalendarTest增加一个测试方法，用以验证使用关键字“今天”增加预约信息是否正确，并将其命名为addATodayReservation()
  2. 执行addATodayReservation()
  3. 思考测试结果的合理性

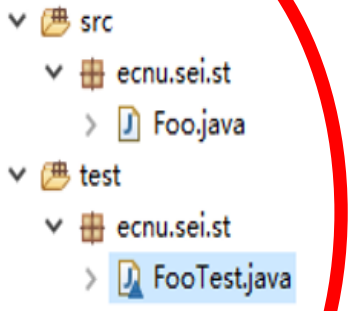
Organize Test Codes



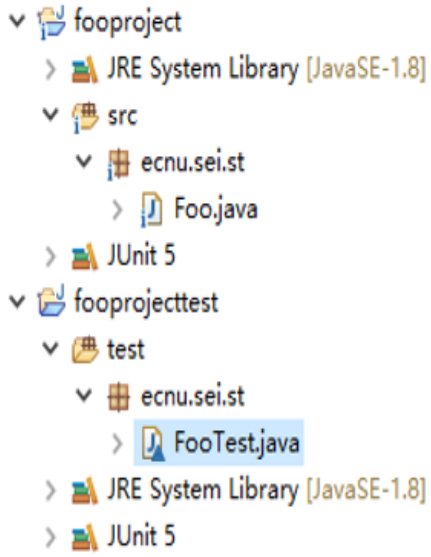
(a) A single-source tree with the same package



(b) A single-source tree with a separate test package

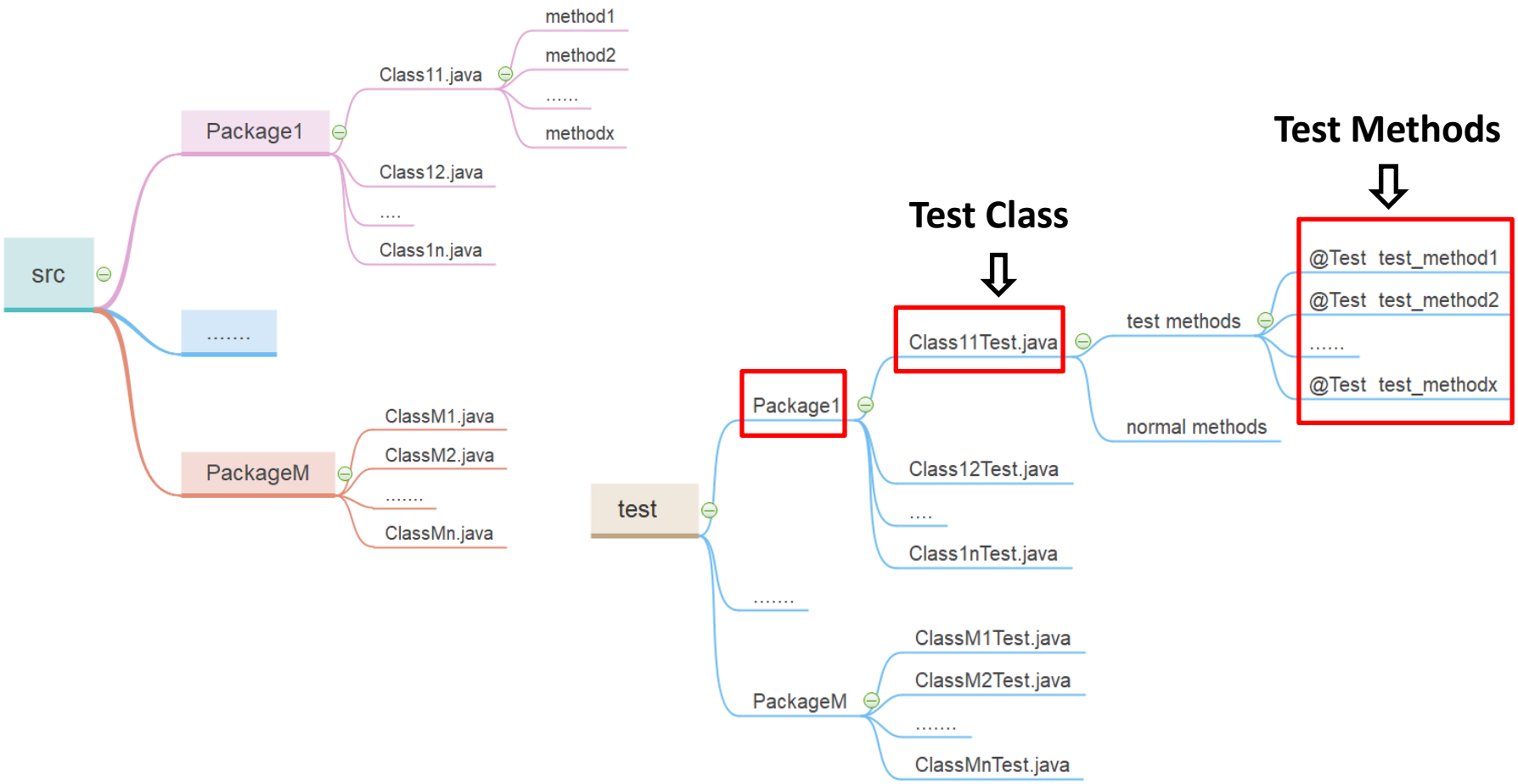


(c) A Parallel-source tree



(d) A Parallel-source tree with separate test project

JUnit Test Code Structure



JUnit Test Code Structure

- Test Class
  - 命名方式：被测类名 + Test ， 例， MeetCalendarTest
- Test Method
  - 真正执行测试发现缺陷的地方
  - 使用@Test, @RepeatedTest, @ParameterizedTest, @TestFactory, @TestTemplate 标注的方法
  - 3A Pattern
    1. Arrange: preparation for the coming test
    2. Action: execute code under test
    3. Assertion: compare actual outputs with expected ones

JUnit Test Code Structure

- Test Method
  - 命名方式

① [经典] 在被测试方法名前加上test前缀，例如，testSetTime（）

② [推荐] 意图命名法，测试方法名应能表明一个具体的测试需求

- AddItem (Add_Item)
- AddItemToEmptyList (Add_Item_to_Empty_List)
- AddItemWithMissingDate (Add_Item_With_Missing_Date)
- should_retrieve_correct_user_when_the_user_exists

③ 对于非正常行为场景的测试方法的命名非常有效

④ 不论采用哪种方式，需要保证：**可读性！可读性！还是可读性！**
要做到清楚、简单并且表达力强，能够使阅读者从其名称中掌握到它的行为意图

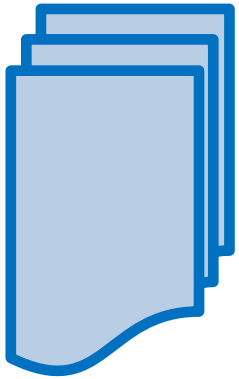
Overview

- Junit Foundations
- Test Lifecycle
- Assertions
- Parameterized tests
- Dynamic tests
- Integrating Junit

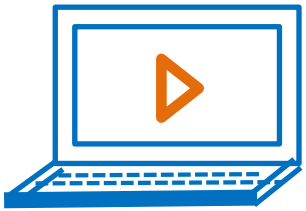
Test Lifecycle

- A test composed of 4 stages:
  1. Setup (optional): the test initializes the test fixture
  2. Exercise: the test interacts with the SUT, getting some outcome from it as a result.
  3. Verify: the outcome from the system under test is compared to the expected value using one or several assertions (also known as predicates). As a result, a test verdict is created.
  4. Teardown (optional): the test releases the test fixture to put the SUT back into the initial state.

Test Lifecycle

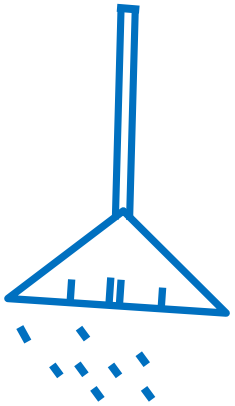


Setup:
preparation for the coming test



Exercise:
execute code under test

Verify:
compare actual outputs with expected ones

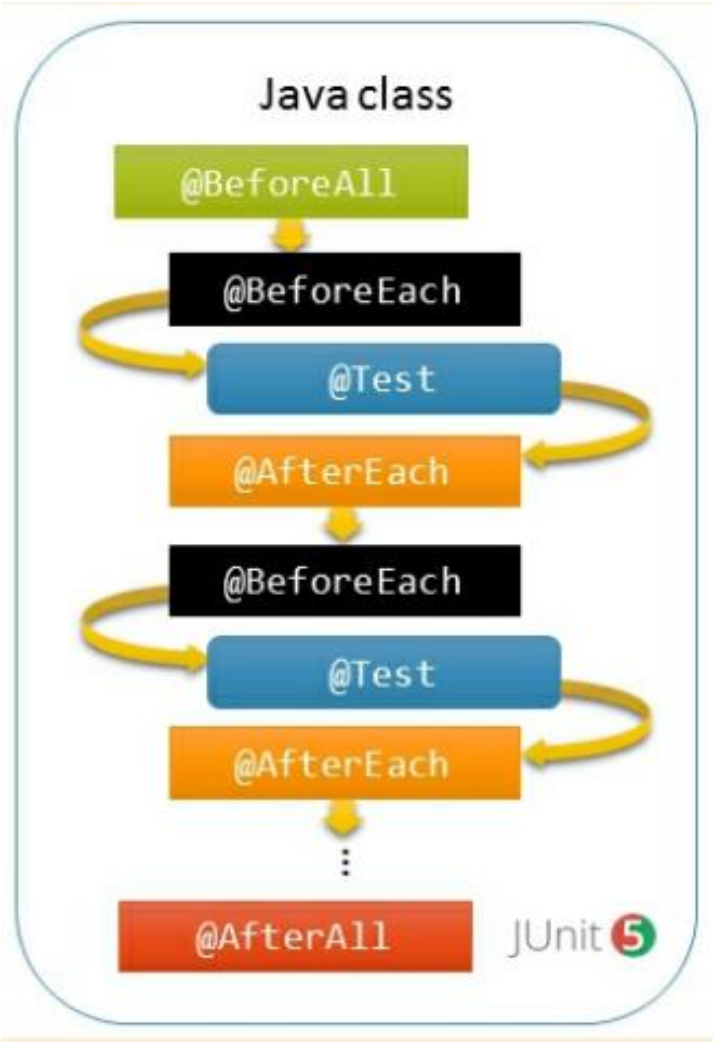


tearDown:
clean up test environment

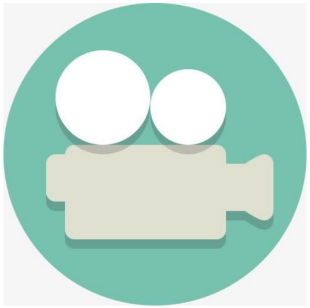
@BeforeAll→@BeforeEach→@Test→@AfterEach→
@BeforeEach→@Test→@AfterEach →…… @AfterAll

Test Lifecycle

- @BeforeAll (static method): executed before all @Test
- @BeforeEach: executed before each @Test
- @AfterEach: executed after each @Test
- @AfterAll (static method): executed after all @Test



Practice



相关视频

5-Test Lifecycle Methods-1

6-Test Lifecycle Methods-2

- 实验任务
  - 阅读并运行MeetHereJava项目位于test目录下的TestLifeCycle
  - 查看运行结果加深对各个lifecycle方法作用域的理解
  - 阅读并运行MeetHereJava项目位于test目录下的MeetCalendarTestPerMethod，查看运行结果
  - 阅读并运行MeetHereJava项目位于test目录下的MeetCalendarTestPerClass ，查看运行结果，思考错误原因
  - 修改MeetCalendarTestPerClass代码，将其构造函数改名为init()，并增加标注@BeforeAll，运行、查看、思考结果
  - 修改MeetCalendarTestPerClass代码，将init()的标注改为@BeforeEach，运行、查看、思考结果

Test Instance Lifecycle

- In order to run test method in isolation, in default, Junit create a new test class instance for each test method

```
@TestInstance(TestInstance.Lifecycle.PER_METHOD)
```

- In Junit5, in order to declare BeforeAll as non-static method to support nested tests, Junit just create one test class instance for all test method

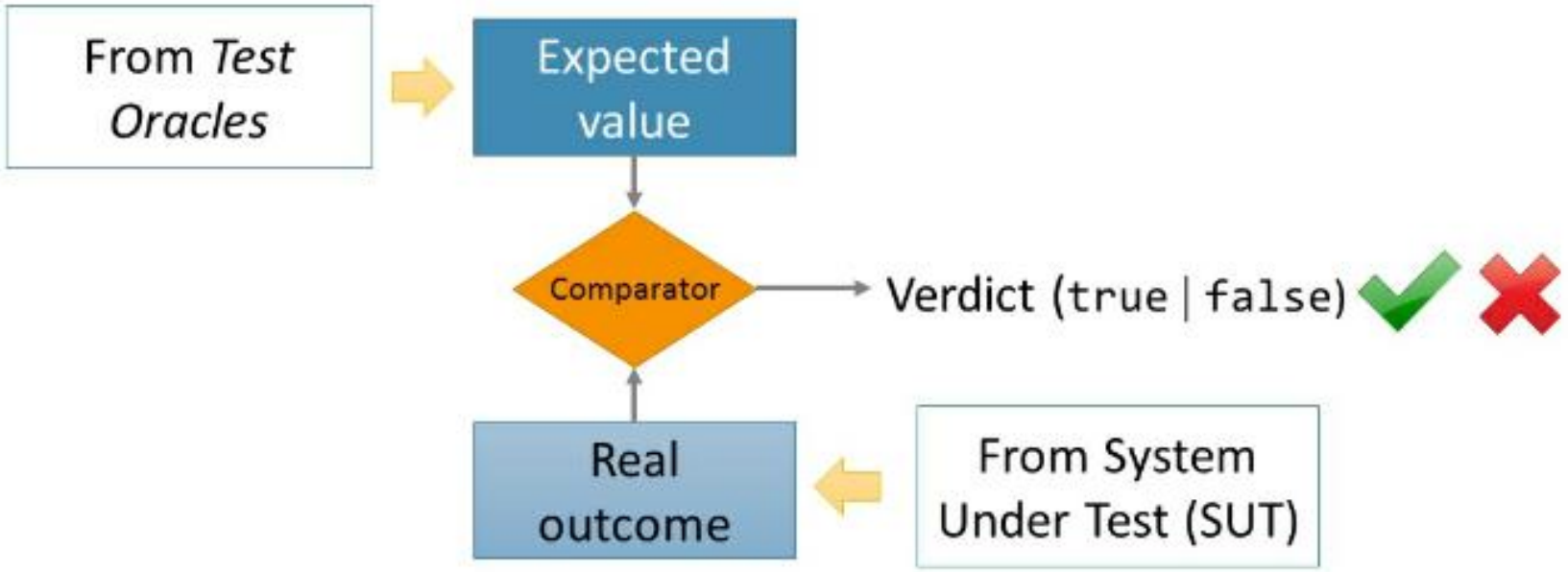
```
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
```

Overview

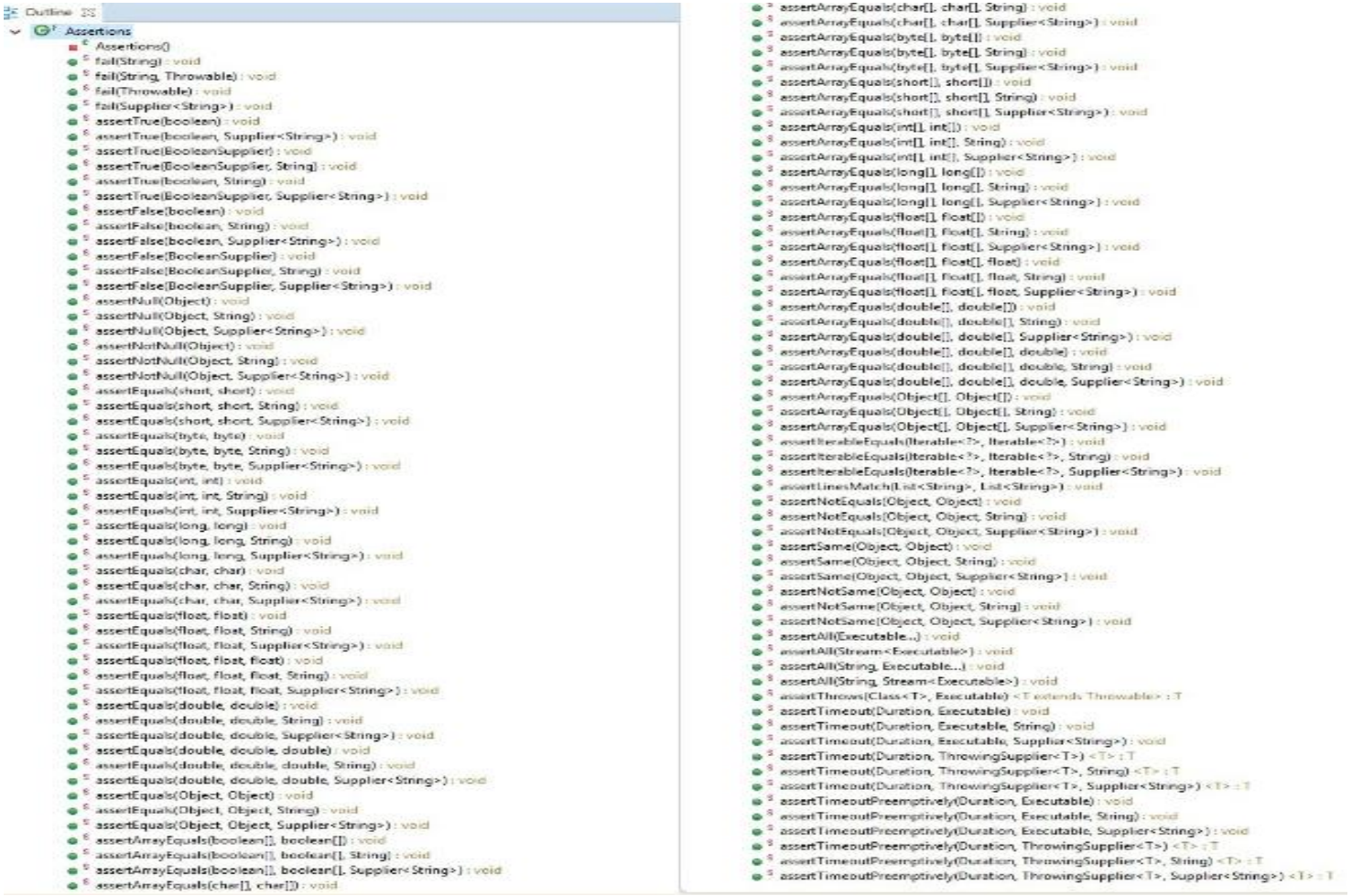
- Junit Foundations
- Test Lifecycle
- Assertions
- Parameterized tests

Assertions

- An assertion (predicate) is a boolean statement typically used to reason about software correctness.

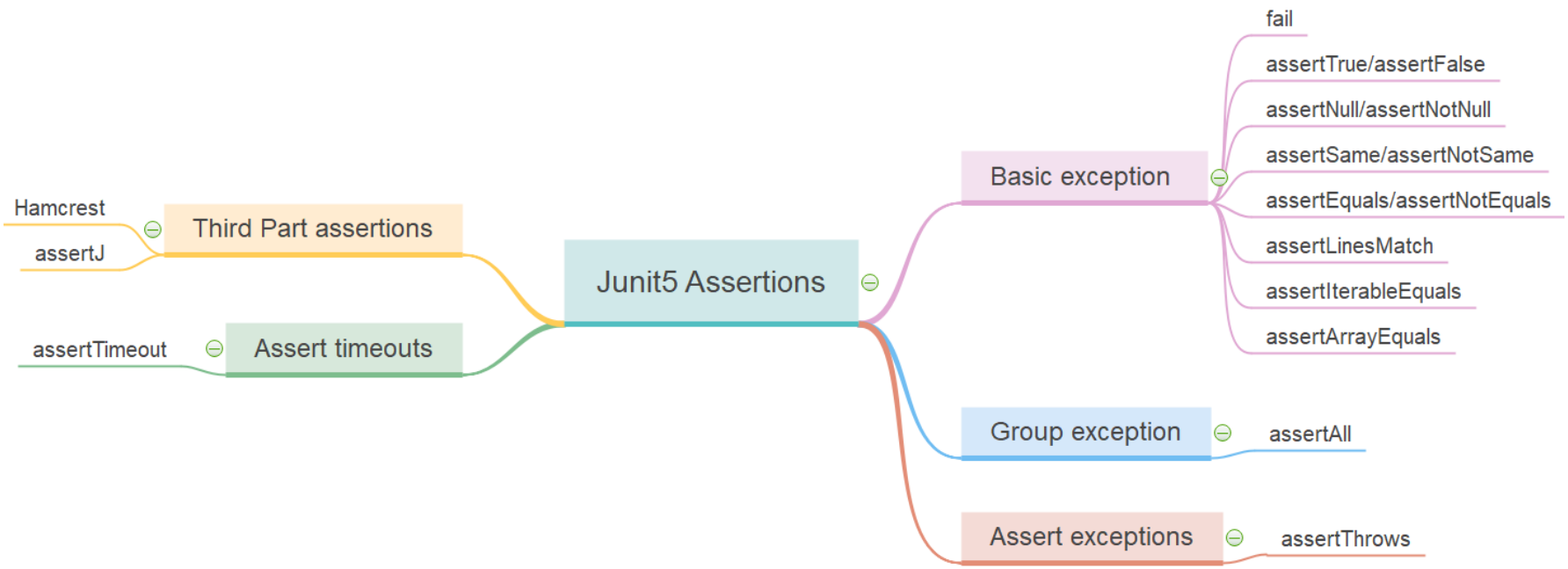


Junit5 Assertions



org.junit.jupiter.api.Assertions

Junit5 Assertions

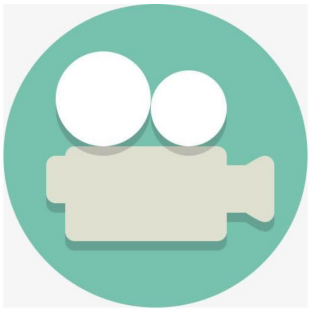


Junit5 Assertion

- assertAll()
  - group different assertions
  - all assertions are always executed
  - any failures will be reported together.

```
public static void assertAll(String heading, Stream<Executable> executables) throws MultipleFailuresError
```

Practice



相关视频

7-Testing Exceptions

- 实验任务

1. 修改MeetCalendarTest 的addAnReservation()中的Action部分的测试数据：Sun→Sun1, gymb1→gymb2
2. 运行测试代码，查看测试结果测试失效情况
3. 使用assertAll修改MeetCalendarTest 的addAnReservation()中检查预约信息的断言
4. 运行测试代码，查看测试结果报告的测试失效情况

Testing Exceptions

- 异常测试需要先后验证
  - 系统抛出期望的异常: assertThrows（）
  - 对异常的处理正确
  - 两个验证内容必须同时满足

```
public static <T extends Throwable> T assertThrows(Class<T> expectedType, Executable executable, String message)
```

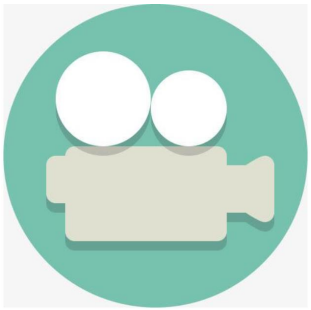
Assert that execution of the supplied `executable` throws an exception of the `expectedType` and return the exception.

If no exception is thrown, or if an exception of a different type is thrown, this method will fail.

If you do not want to perform additional checks on the exception instance, simply ignore the return value.

Fails with the supplied failure `message`.

Practice



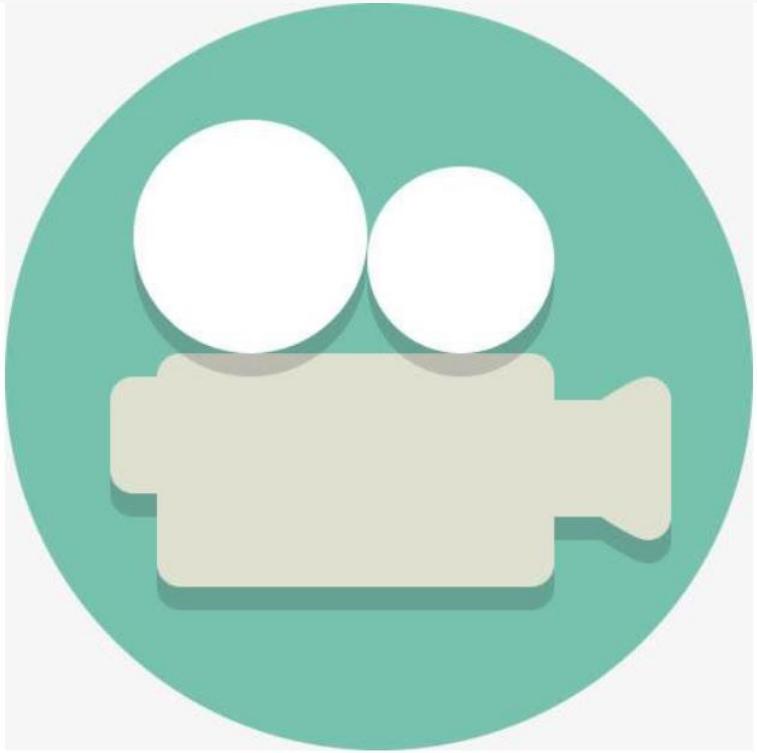
相关视频

7-Testing Exceptions

- 实验任务
  - 补全DataTimeConvertTest的convertInvalidDateString()对不合法日期的异常处理进行测试，要求验证：

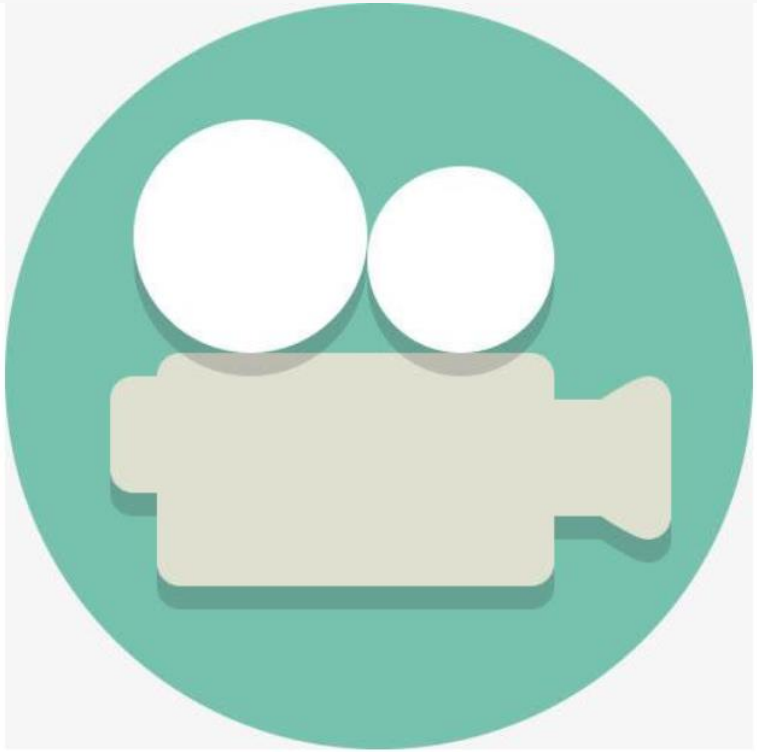
1. 系统抛出的异常是否符合预期
2. 对异常的处理是否符合预期

Display Name



8-Improving Test Message Readability

Disable Tests



9-Running Particular Tests

Overview

- Junit Foundations
- Test Lifecycle
- Assertions
- Parameterized tests

Parameterized Tests

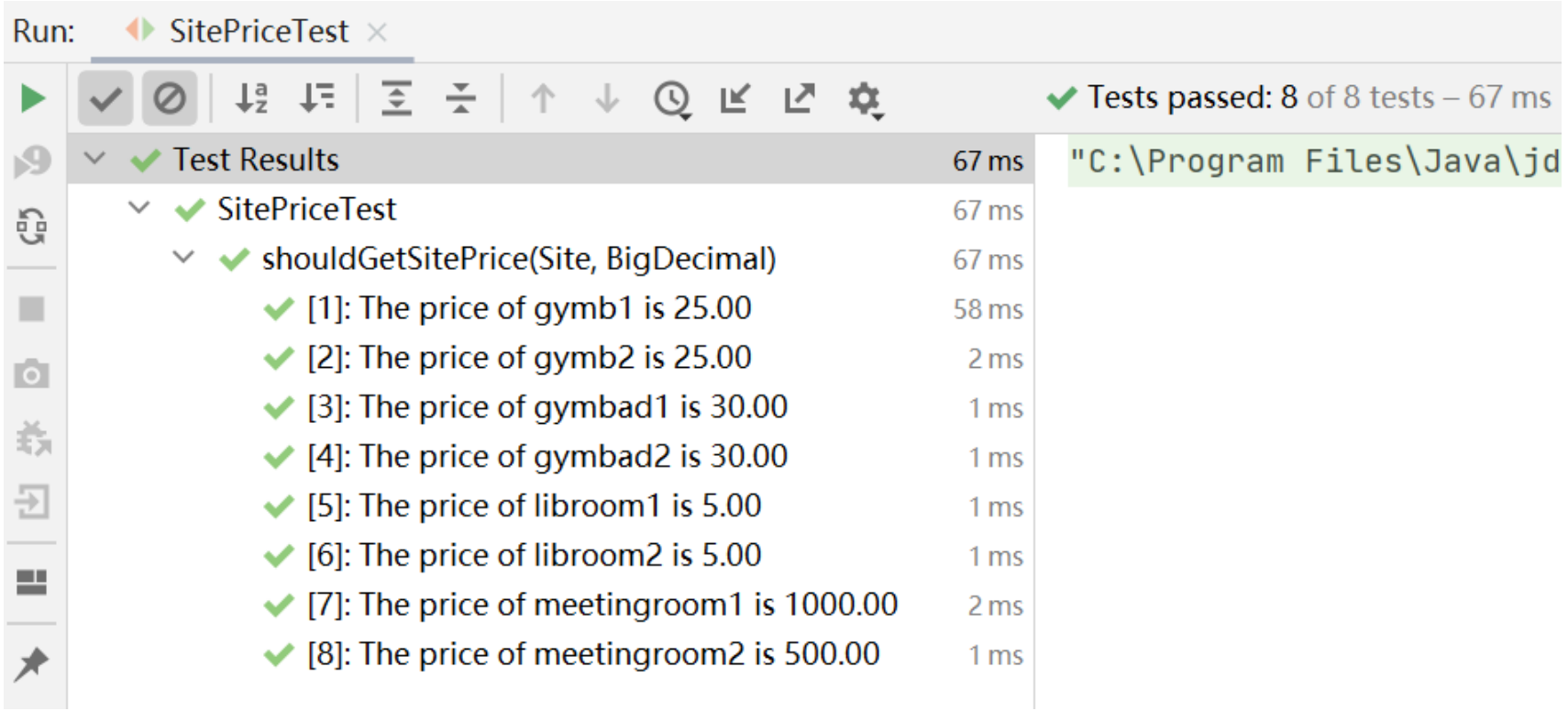
- @ParameterizedTest
  - Data driven automated testing technique implemented in Junit5
  - run a test multiple times with different arguments
  - same lifecycle as @Test
  - need to declare at least one argument source
  - Experimental API that may be changed in future version of Junit

Practice

送你一朵小红花

- 实验准备
  1. 课程网站下载**MeetHereMaven.zip**：视频资料　JUnit + mockito视频
  2. 解压**MeetHereMaven. zip**到当前目录（**目录中不要有中文**）
  3. Idea中打开**MeetHereMaven**，等待Maven更新完毕
- 实验任务1
  - 查看test目录下SitePriceTest代码
  - 运行SitePriceTest
  - 查看运行测试的个数及测试执行情况

Practice


Run:
SitePriceTest
Tests passed: 8 of 8 tests – 67 ms
Test Results 67 ms
SitePriceTest 67 ms
shouldGetSitePrice(Site, BigDecimal) 67 ms
[1]: The price of gymb1 is 25.00 58 ms
[2]: The price of gymb2 is 25.00 2 ms
[3]: The price of gymbad1 is 30.00 1 ms
[4]: The price of gymbad2 is 30.00 1 ms
[5]: The price of libroom1 is 5.00 1 ms
[6]: The price of libroom2 is 5.00 1 ms
[7]: The price of meetingroom1 is 1000.00 2 ms
[8]: The price of meetingroom2 is 500.00 1 ms
"C:\Program Files\Java\jd

Parameterized Tests

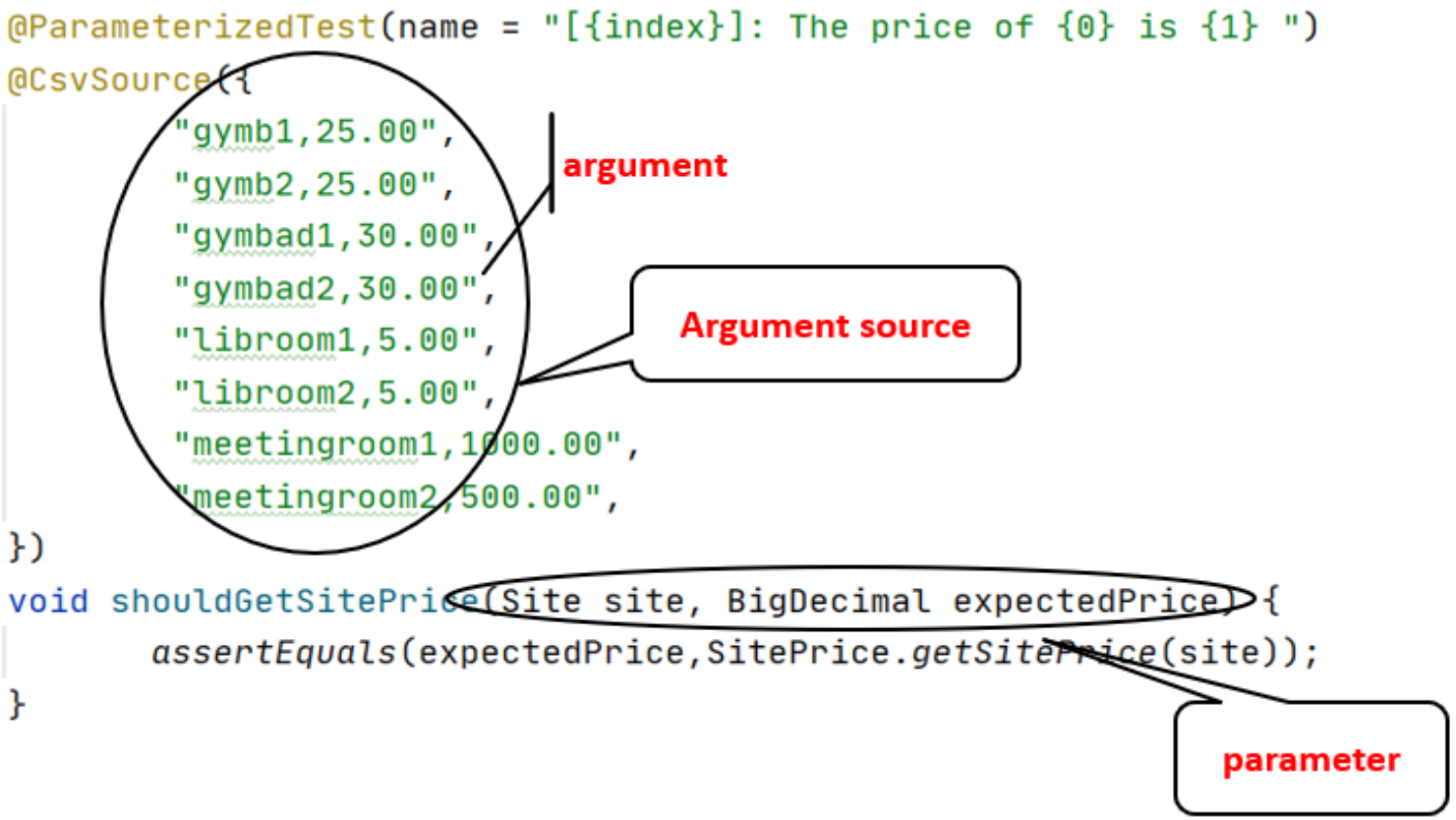
- @ParameterizedTest Relative Topics
  1. Argument Source(测试数据源): define test data for test
     - Predefined type
     - Customized type
  2. Argument Convention : keep type consistent between given test data and test parameter
     - Implicit
     - Explicit
  3. @ParameterizedTest attributes

Parameterized Tests

- @ParameterizedTest Relative Topics
  1. Argument Source: define test data for test
     - Predefined type
     - Customized type
  2. Argument Convention : keep type consistent between given test data and test parameter
     - Implicit
     - Explicit
  3. @ParameterizedTest attributes

Argument Sources

- **[Argument Source (测试数据源)]** 由若干测试数据构成的信息资源，为测试方法的每次执行提供相应的测试数据



Argument Sources

- Argument Source的基本使用规则
  - AS规则1：每个@Parameterized测试方法可以使用多个测试数据源，但是至少需要有一个
    - **[Multiple AS Practice]** 运行MeetHereMaven项目中ParameterizedTestDemo的testMethodWithMultipleArgumentSource
    - **[No AS Practice]** 运行MeetHereMaven项目中ParameterizedTestDemo的testMethodWithoutArgumentSource

Argument Sources

- Argument Source的使用规则
  - AS规则2：每个测试数据源必须为测试方法的所有参数提供测试数据。例如，测试方法若有2个参数，参数1不能使用测试数据源1，而参数2使用其他数据源，即参数2也必须使用测试数据源1
  - **[Different AS Practice]** 在MeetHereMaven项目中的ParameterizedTestDemo增加如下测试方法：

```
@ParameterizedTest
@ValueSource(ints = {1,2,3})
@MethodSource("range")
void parameterUseDifferentArgumentSource(int candidate1,int candidate2) {
    assertNotEquals( unexpected: 9,candidate1);
    assertNotEquals( unexpected: -1,candidate2);
}
```

Argument Sources

- Junit predefined argument sources
  1. @ValueSource
  2. Null and Empty Sources： @NullSource， @EmptySource， @NullAndEmptySource
  3. @EnumSource
  4. @MethodSource
  5. @CsvSource
  6. @CsvFileSource
- Customize your own argument sources
  - @ArgumentsSource

Argument Sources Predefined Type 1

- @ValueSource
  - use array to provide test data for test methods which have only one parameter
  - arrays of type

1. int
2. String
3. double
4. long
5. ….

@ValueSource

- Some Examples
  - short values: @ValueSource(shorts = {1,2,3})
  - byte values: @ValueSource(bytes = {-128,127,101})
  - integer value: @ValueSource(ints = {202, Integer.MAX_VALUE})
  - long values: @ValueSource(longs = {2L, 4L, 8L})
  - float values: @ValueSource(floats = {1.2F, 4.3F, 6.008F})
  - double values: @ValueSource(doubles = {1.5D, 2.2D, 3.0D})
  - char values: @ValueSource(chars = {‘a’ , ‘b’, ‘c’})
  - boolean values: @ValueSource(boolean = {true,false,true,true})
  - string values: @ValueSource(strings = {“software testing”, “hysun”})
  - class values: @ValueSource(classes = {Integer.class,String.class})

Practice

- 实验任务
  - 查看test目录下ParameterizedTestDemo代码
  - 运行ParameterizedTestDemo中使用@ValueSource的测试方法
  - 查看运行测试的个数及测试执行情况

Argument Sources Predefined Type 2

- Null and Empty Sources
  - Provide **null** and **empty** values for a single argument

    (that is , the corresponding test methods must have only one parameter)

  1. @NullSource
  2. @EmptySource
  3. @NullAndEmptySource

@NullSource

- @NullSource
  - provides a single null argument, **not** applied to **primitive type**.
- @EmptySource
  - provides a single empty argument
  - applied to : java.lang.String, java.util.List, java.util.Set, java.util.Map, primitive arrays (e.g., int[], char[][], etc.), object arrays (e.g.,String[], Integer[][], etc.).
  - Subtypes of the supported types are **not supported**.
- @NullAndEmptySource
  - combines the functionality of @NullSource and @EmptySource.

Example

```
@ParameterizedTest
@NullSource
@EmptySource
@ValueSource(strings = { " ", "   ", "\t", "\n" })
void nullEmptyAndBlankStrings(String text) {
    assertTrue(text == null || text.trim().isEmpty());
}
```

两个测试方法都运行6次，测试方法执行时，测试输入分别为null，empty，” ”，　” ”,“\t”,”\n”

```
@ParameterizedTest
@NullAndEmptySource
@ValueSource(strings = { " ", "   ", "\t", "\n" })
void nullEmptyAndBlankStrings(String text) {
    assertTrue(text == null || text.trim().isEmpty());
}
```

Argument Sources Predefined Type 3

- @EnumSource
  - use values of an enum to provide test data for test methods which have only one parameter
  - 1 must attribute

    value: @EnumSource(TimeUnit.class)
  - 2 Optional attributes

    name: @EnumSource(value = TimeUnit.class, names = { "DAYS", "HOURS" })

    mode: @EnumSource(value = TimeUnit.class, mode = EXCLUDE, names = { "DAYS", "HOURS" })

@EnumSource

- name: The names of enum constants to provide
  1. regular expressions to select the names of enum constants to provide.
  2. If no names or regular expressions are specified, all enum constants declared in the specified enum type will be provided.
  3. The mode() determines how the names are interpreted.
- mode: The enum constant selection mode.
  - Defaults: INCLUDE.
  - INCLUDE, EXCLUDE, MATCH_ALL,MATCH_ANY,

Practice

- 实验任务
  - 查看test目录下ParameterizedTestDemo代码
  - 运行ParameterizedTestDemo中使用@EnumSource的测试
  - 查看运行测试的个数，对比测试执行情况

Argument Sources Predefined Type 4

- @MethodSource(“methodname”)
  - refer one or more methods (factory method)to provide test data for test methods

```
@ParameterizedTest
@MethodSource("stringProvider")
void testWithExplicitLocalMethodSource(String argument) {
    assertNotNull(argument);
}

static Stream<String> stringProvider() {
    return Stream.of("apple", "banana");
}
```

测试数据源提供者,factory method

Practice

- 实验任务
  - 查看test目录下ParameterizedTestMethodASDemo代码
  - 运行ParameterizedTestMethodASDemo中的测试
  - 总结@MethodSource的使用规则
    1. 如何指定Factory method
    2. 如何定义Facotry method，包括可访问性、参数、返回值等

@MethodSource

- @MethodSource Using Rules
  - Factory method must be static unless using @TestInstance(Lifecycle.PER_CLASS)
  - Factory method should have no parameters
  - for test methods only have one parameter
    - factory method returns a **stream of Parameter type**
    - factory method returns a **stream of primitive type**
    - **stream :** anything that JUnit can reliably convert into a Stream（not only refer to the Java Stream Class）, such as Stream, DoubleStream, LongStream, IntStream, Collection, Iterator, Iterable, an array of objects, or an array of primitives.

@MethodSource

- @MethodSource Using Rules
  - for test methods have multiple parameters
    - factory method returns a Stream, Iterable, Iterator or array of type Arguments
  - If no factory method is given in @MethodSource , the method which has the same name with the test will be selected as the factory method

@MethodSource

- @MethodSource Using Rules
  - Factory method can be methods in the test class or methods of other classes
    - As for factory method from external class, Full class name is needed, that, don’t forget package name !!

@MethodSource

```java
package example;

import java.util.stream.Stream;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.MethodSource;

class ExternalMethodSourceDemo {

    @ParameterizedTest
    @MethodSource("example.StringsProviders#tinyStrings")
    void testWithExternalMethodSource(String tinyString) {
        // test with tiny string
    }
}

class StringsProviders {

    static Stream<String> tinyStrings() {
        return Stream.of(".", "oo", "OOO");
    }
}
```

Argument Sources Predefined Type 5

- @CsvSource
  - Using comma-separated String literals to provide test data for test methods

```
@ParameterizedTest
@CsvSource({
    "apple,          1",
    "banana,         2",
    "'lemon, lime', 0xF1"
})
void testWithCsvSource(String fruit, int rank) {
    assertNotNull(fruit);
    assertNotEquals(0, rank);
}
```

@CsvSource

- @CsvSource
  - Use a single quote (‘) as a quote character

Example Input	Resulting Argument List
`@CsvSource({ "apple, banana" })`	`"apple", "banana"`
`@CsvSource({ "apple, 'lemon, lime'" })`	`"apple", "lemon, lime"`
`@CsvSource({ "apple, ''" })`	`"apple", ""`
`@CsvSource({ "apple, " })`	`"apple", null`

@CsvSource

- @CsvSource
  - 1 attribute
    - delimiter: specify delimiter between parameters

```
@ParameterizedTest
@CsvSource(delimiter = ';',value = {"apple;1","banana;2"})
void testWithCsvFile(String fruit, int rank) {
    assertNotNull(fruit);
    assertNotEquals( unexpected: 0,rank);
}
```

Argument Sources

```
@ParameterizedTest(name = "[{index}]: The price of {0} is {1} ")
@CsvSource({
        "gymb1,25.00",
        "gymb2,25.00",
        "gymbad1,30.00",
        "gymbad2,30.00",
        "libroom1,5.00",
        "libroom2,5.00",
        "meetingroom1,1000.00",
        "meetingroom2,500.00",
})
void shouldGetSitePrice(Site site, BigDecimal expectedPrice) {
        assertEquals(expectedPrice,SitePrice.getSitePrice(site));
}
```

Argument Sources Predefined Type 6

- @CsvFileSource
  - Using csv File from classpath to provide test data for test methods

```
@ParameterizedTest
@CsvFileSource(resources = "/two-column.csv", numLinesToSkip = 1)
void testWithCsvFileSource(String country, int reference) {
    assertNotNull(country);
    assertNotEquals(0, reference);
}
```

	A	B
1	Country	reference
2	Sweden	1
3	Poland	2
4	“U.S.A”	3
5		

⇨ Excel表格内容可以保存成CSV

@CsvFileSource

- @CsvFileSource
  - Use a double quote (“) as a quote character
  - 4 attributes

1. Encoding
2. Line separator
3. Parameter Delimiter
4. Line to skip

```
@ParameterizedTest
@CsvFileSource(resources = "/testdata.csv",e)
void testWithCsvFile(String fruit, int
    assertNotNull(fruit);
    assertNotEquals( unexpected: 0,rank
}
```

```
encoding default "UTF-8"
delimiter default ','
lineSeparator default "\n"
numLinesToSkip default 0
```

Practice

- 实验任务1
  - 根据SitePriceTes的shouldGetSitePrice使用到的参数，新建一个SiteAndPrice.csv文件
  - 将SiteAndPrice.csv保存到项目的resource目录下
  - 编写shouldGetSitePriceFromFile方法，使用SiteAndPrice.csv作为数据源对场馆价格进行测试

Customized Argument Sources

- @ArgumentsSource
  - Customized argument sources by implementing ArgumentProvider interface

```
public interface ArgumentsProvider {
    Stream<? extends Arguments>
            provideArguments(ExtensionContext var1)
             throws Exception;
}
```

获取到各种测试类的相关信息

Practice

- 实验任务
  - 阅读GradeArgumentsProvider并运行testFinalGrateGradeWithArgumentProvider，注意接口ArgumentProvider的实现

Parameterized Tests

- @ParameterizedTest Relative Topics
  1. Argument Source: define test data for test
     - Predefined type
     - Customized type
  2. Argument Convention : keep type consistent between given test data and test parameter
     - Implicit
     - Explicit
  3. @ParameterizedTest attributes

Argument Convention

- When the argument type and the parameter type don't match
  1. Implicit convention
     - Junit5 built-in implicit type converters: String convert to the type of parameter
     - Fallback String-to-Object Conversion: Factory method & factory constructor
  2. Explicit convention
     - Implement ArgumentConverter interface/extends SimpleArgumentConverter
     - @ConvertWith in parameter

Implicit Argument Convention

Target Type	Example
`boolean` / `Boolean`	`"true"` → `true`
`byte` / `Byte`	`"15"`, `"0xF"`, or `"017"` → `(byte) 15`
`char` / `Character`	`"o"` → `'o'`
`short` / `Short`	`"15"`, `"0xF"`, or `"017"` → `(short) 15`
`int` / `Integer`	`"15"`, `"0xF"`, or `"017"` → `15`
`long` / `Long`	`"15"`, `"0xF"`, or `"017"` → `15L`
`float` / `Float`	`"1.0"` → `1.0f`
`double` / `Double`	`"1.0"` → `1.0d`
Enum subclass	`"SECONDS"` → `TimeUnit.SECONDS`
`java.io.File`	`"/path/to/file"` → `new File("/path/to/file")`
`java.lang.Class`	`"java.lang.Integer"` → `java.lang.Integer.class` *(use* `$` *for nested classes, e.g.* `"java.lang.Thread$State"` *)*
`java.lang.Class`	`"byte"` → `byte.class` *(primitive types are supported)*

https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests-sources-MethodSource

Fallback String-to-Object Conversion

```java
@ParameterizedTest
@ValueSource(strings = "42 Cats")
void testWithImplicitFallbackArgumentConversion(Book book) {
    assertEquals("42 Cats", book.getTitle());
}
```

```java
public class Book {

    private final String title;

    private Book(String title) {
        this.title = title;
    }

    public static Book fromTitle(String title) {
        return new Book(title);
    }

    public String getTitle() {
        return this.title;
    }
}
```

factory method

a non-private, static method declared in the target type that accepts a single argument and returns an instance of the target type.

Fallback String-to-Object Conversion

```java
@ParameterizedTest
@ValueSource(strings = {"Software testing","Junit"})
void fallbackConversion(Book book) {
    assertNotNull(book.getTitle());
}
```

```java
public class Book {
    private final String title;

    public Book(String title){
        this.title = title;
    }
    public String getTitle() {
        return title;
    }
}
```

factory Constructor

a non-private constructor in the target type that accepts a single String argument

Explicit Conversion

```
public interface ArgumentConverter {
    Object convert(Object var1, ParameterContext var2)
            throws ArgumentConversionException;
}
```

ArgumentConverter Interface

Explicit Conversion

```
public abstract class SimpleArgumentConverter
        implements org.junit.jupiter.params.converter.ArgumentConverter {
    public SimpleArgumentConverter() {
    }

    public final Object convert(Object source, ParameterContext context)
            throws ArgumentConversionException {
        return this.convert(source, context.getParameter().getType());
    }

    protected abstract Object convert(Object var1, Class<?> var2)
            throws ArgumentConversionException;
}
```

SimpleArgumentConverter class

```java
@ParameterizedTest
@ValueSource(strings = {"17001;hysun;95","17002;hymao;80"})
void explicitConversion(
        @ConvertWith(StudentConverter.class) Student student) {
    System.out.println("The Student is: " + student.getName());
    assertNotEquals( unexpected: 0,student.getGrade());
}
```

```java
public class StudentConverter extends SimpleArgumentConverter {
    @Override
    protected Object convert(Object source, Class<?> targetType){

        assertEquals(String.class,source.getClass(), message: "Argument should be a String");
        assertEquals(Student.class,targetType, message: "can only convert to Student type");

        String[] studentString = source.toString().split( regex: ";");
        Student student = new Student(
                studentString[0].trim(),
                studentString[1].trim(),
                Integer.parseInt(studentString[2]));
        return student;
    }
}
```

Explicit Conversion Example

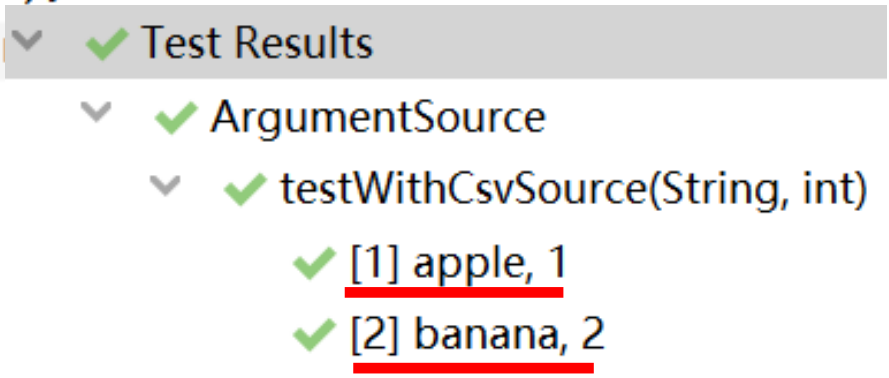
Parameterized Tests

- Custom Display Names
  - Default name format:

    @ParameterizedTest(name="[{index}] {arguments}")

    [Index] String consists of all arguments for the invocation

```
@ParameterizedTest
@CsvSource(delimiter = ';',value = {"apple;1","banana;2"})
void testWithCsvSource(String fruit, int rank) {
    assertNotNull(fruit);
    assertNotEquals( u
}
```



Parameterized Tests

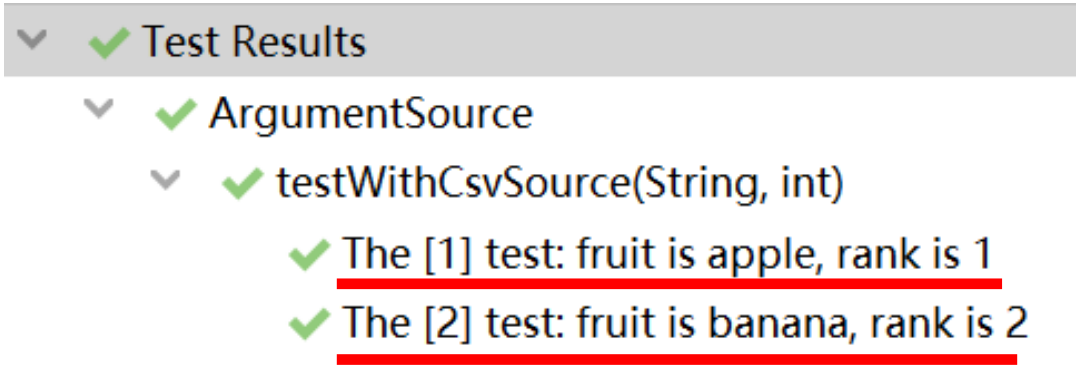
- Custom Display Names
  - customize name format (not @DisplayName)

    @ParameterizedTest(name=“ The [{index}] test: fruit is {0}, rank is {1} ”)

Placeholder	Description
`{index}`	the current invocation index (1-based)
`{arguments}`	the complete, comma-separated arguments list
`{0}`, `{1}`, …	an individual argument

Parameterized Tests

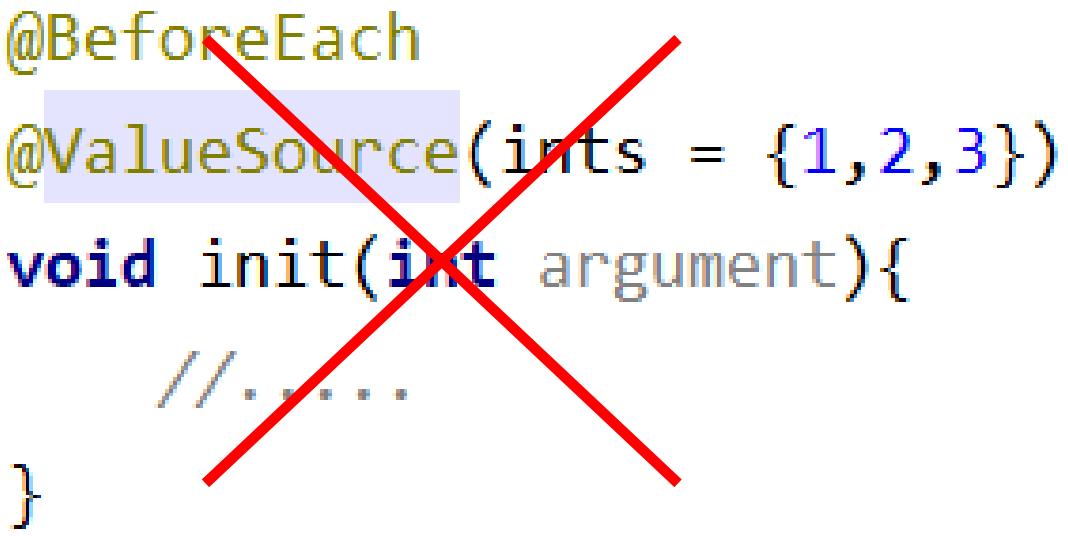
```
@ParameterizedTest(name = "The [{index}] test: fruit is {0}, rank is {1}")
@CsvSource(delimiter = ';',value = {"apple;1","banana;2"})
void testWithCsvSource(String fruit, int rank) {
    assertNotNull(fruit);
    assertNotEquals( unexpected: 0,rank);
}
```



Customized Display Name Example

Parameterized Tests

- Lifecycle method and parameterized
  - ParameterizedTest parameters can’t be injected into lifecycle methods

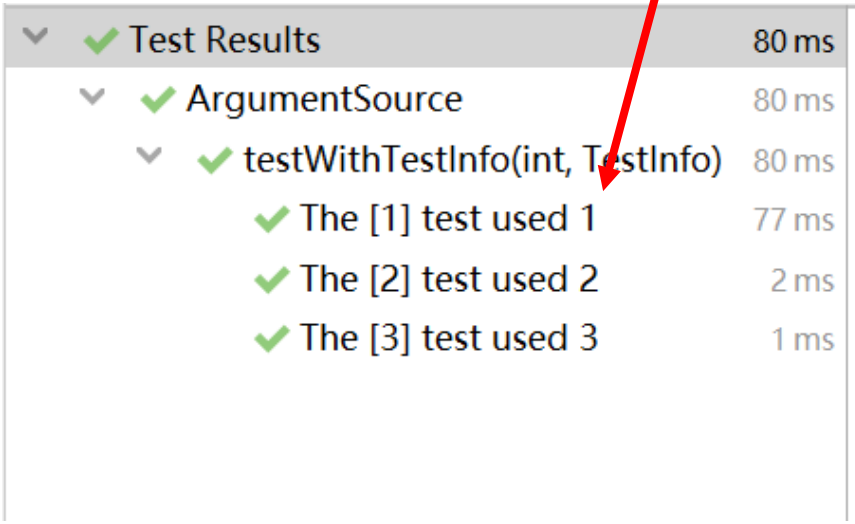


Parameterized Tests

- Lifecycle method and parameterized
  - Test information parameters
    1. TestInfo
    2. TestReporter
- Parameterized tests run independently
  - In case some tests fail, other tests can still run

```java
@BeforeEach
 void init(TestInfo testInfo){
    System.out.println("The " + testInfo.getDisplayName() + " is going to be executed.");
}

@ParameterizedTest(name = "The [{index}] test used {0}" )
@ValueSource(ints = {1,2,3})
void testWithTestInfo(int argument,TestInfo info) {
    assertNotEquals( unexpected: 9, argument);
}
```




```
"C:\Program Files\Java\jdk-11.0.5\bin\java.exe"

The The [1] test used 1 is going to be executed.

The The [2] test used 2 is going to be executed.

The The [3] test used 3 is going to be executed.
```

The End