Running Tests in Order

Remember this?

Order???

- In general
  - Order should not be a factor in unit tests
  - There should be no dependency between tests
  - All tests should pass regardless of the order in which they are run

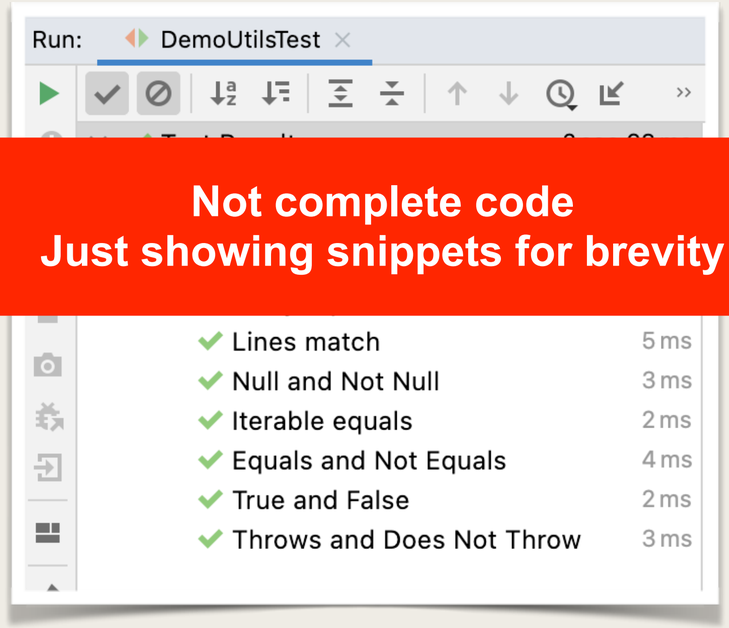
Use Cases

- However, there are some uses cases when you want to control the order
  - You want tests to appear in alphabetical order for reporting purposes
  - Sharing reports with project management, QA team etc...
  - Group tests based on functionality or requirements

Existing ... before any changes

DemoUtilsTest.java

```
class DemoUtilsTest {
    ...

    @DisplayName("Equals and Not Equals")
    void testEqualsAndNotEquals()

    @DisplayName("Null and Not Null")
    void testNullAndNotNull()

    @DisplayName("Same and Not Same")
    void testSameAndNotSame()

    @DisplayName("True and False")
    void testTrueFalse()

    @DisplayName("Array Equals")
    void testArrayEquals()

    @DisplayName("Iterable equals")
    void testIterableEquals()

    @DisplayName("Lines match")
    void testLinesMatch()

    @DisplayName("Throws and Does Not Throw")
    void testThrowsAndDoesNotThrow()

    @DisplayName("Timeout")
    void testTimeout()
}
```



Not complete code
Just showing snippets for brevity

JUnit Docs

By default, test classes and methods will be ordered using an algorithm that is deterministic

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

Annotation

Annotation	Description
`@TestMethodOrder`	Configures the order / sort algorithm for the test methods

Specify Method Order

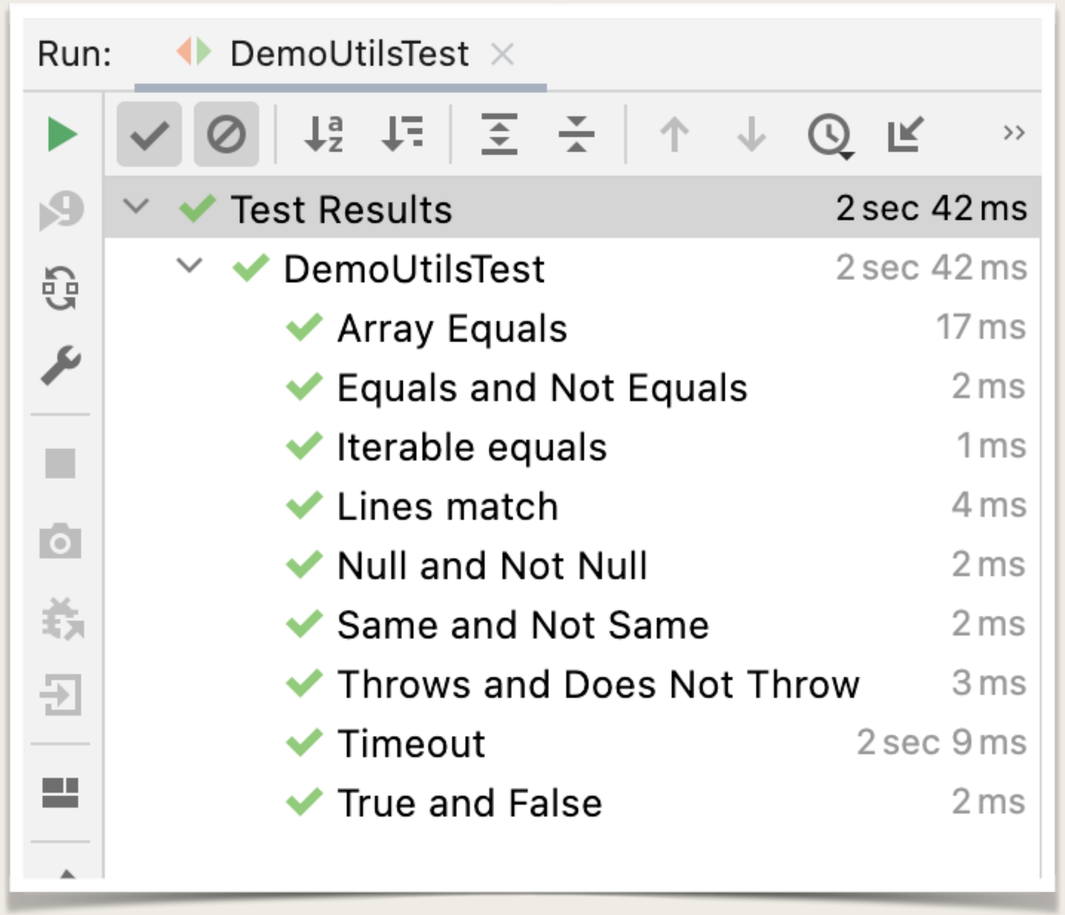
Name	Description
`MethodOrderer.DisplayName`	Sorts test methods alphanumerically based on display names
`MethodOrderer.MethodName`	Sorts test methods alphanumerically based on method names
`MethodOrderer.Random`	Pseudo-random order based on method names
`MethodOrderer.OrderAnnotation`	Sorts test methods numerically based on @Order annotation

Order by Display Name

Sorted alphanumerically

DemoUtilsTest.java

```java
@TestMethodOrder(MethodOrderer.DisplayName.class)
class DemoUtilsTest {
    ...

    @DisplayName("Equals and Not Equals")
    void testEqualsAndNotEquals()

    @DisplayName("Null and Not Null")
    void testNullAndNotNull()

    @DisplayName("Same and Not Same")
    void testSameAndNotSame()

    @DisplayName("True and False")
    void testTrueFalse()

    @DisplayName("Array Equals")
    void testArrayEquals()

    @DisplayName("Iterable equals")
    void testIterableEquals()

    @DisplayName("Lines match")
    void testLinesMatch()

    @DisplayName("Throws and Does Not Throw")
    void testThrowsAndDoesNotThrow()

    @DisplayName("Timeout")
    void testTimeout()
}
```



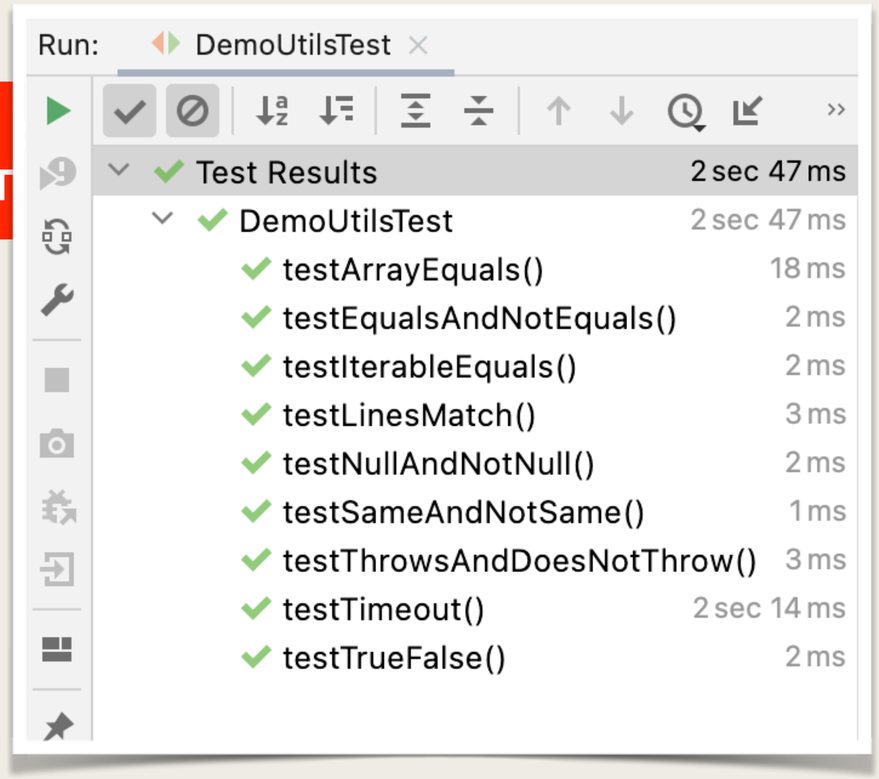
Order by Method Name

Sorted alphanumerically

DemoUtilsTest.java

```java
@TestMethodOrder(MethodOrderer.MethodName.class)
class DemoUtilsTest {
    ...

    // @DisplayName("Equals and Not Equals")
    void testEqualsAndNotEquals()

    void testNullAndNotNull()

    void testSameAndNotSame()

    void testTrueFalse()

    void testArrayEquals()

    void testIterableEquals()

    void testLinesMatch()

    void testThrowsAndDoesNotThrow()

    void testTimeout()
}
```

NOT

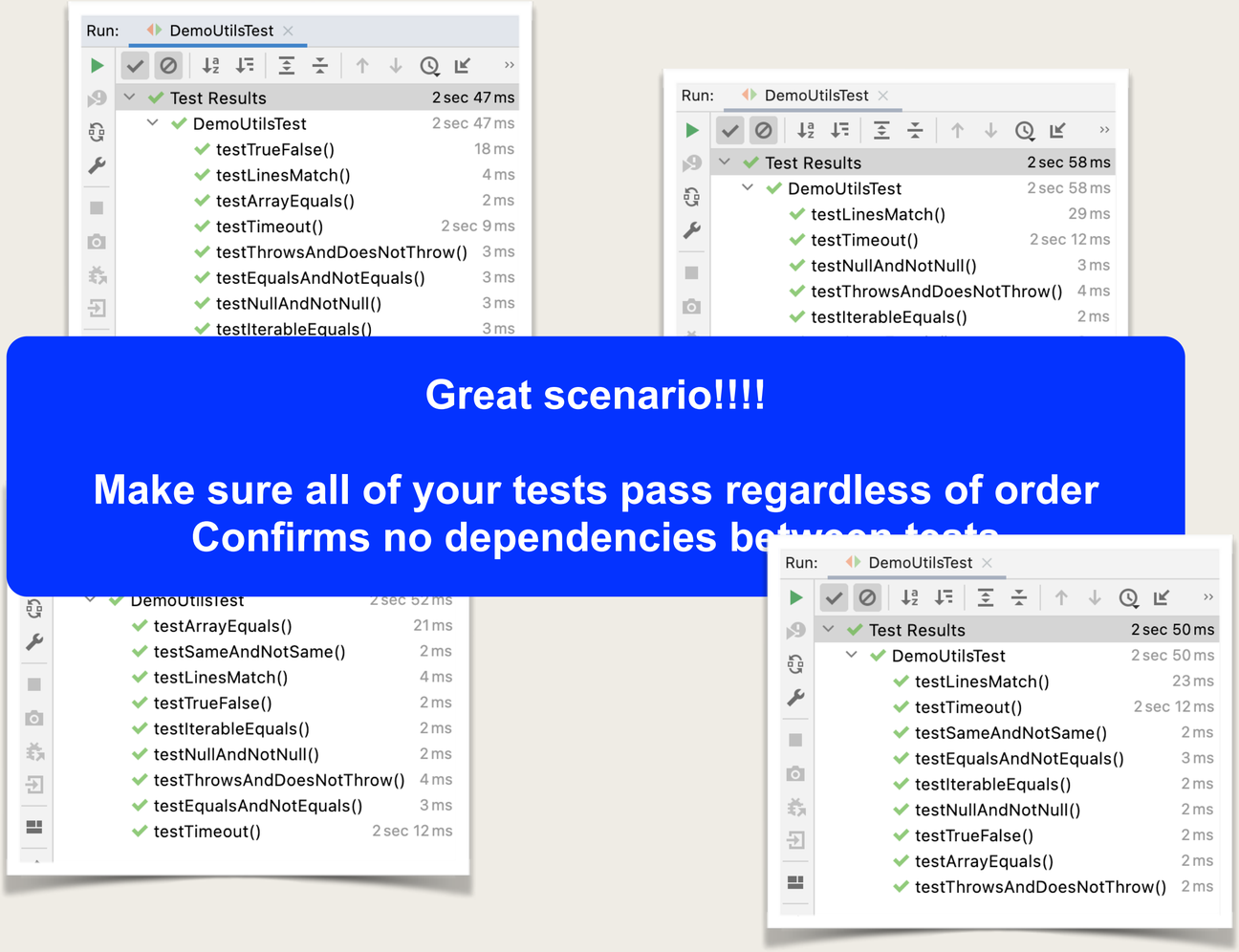


Random by Method Name

DemoUtilsTest.java

```
@TestMethodOrder(MethodOrderer.Random.class)
class DemoUtilsTest {
    ...

    void testEqualsAndNotEquals()

    void testNullAndNotNull()

    void testSameAndNotSame()

    void testTrueFalse()

}
```

Can also use DisplayName since everything is random ... order doesn't matter



Annotation

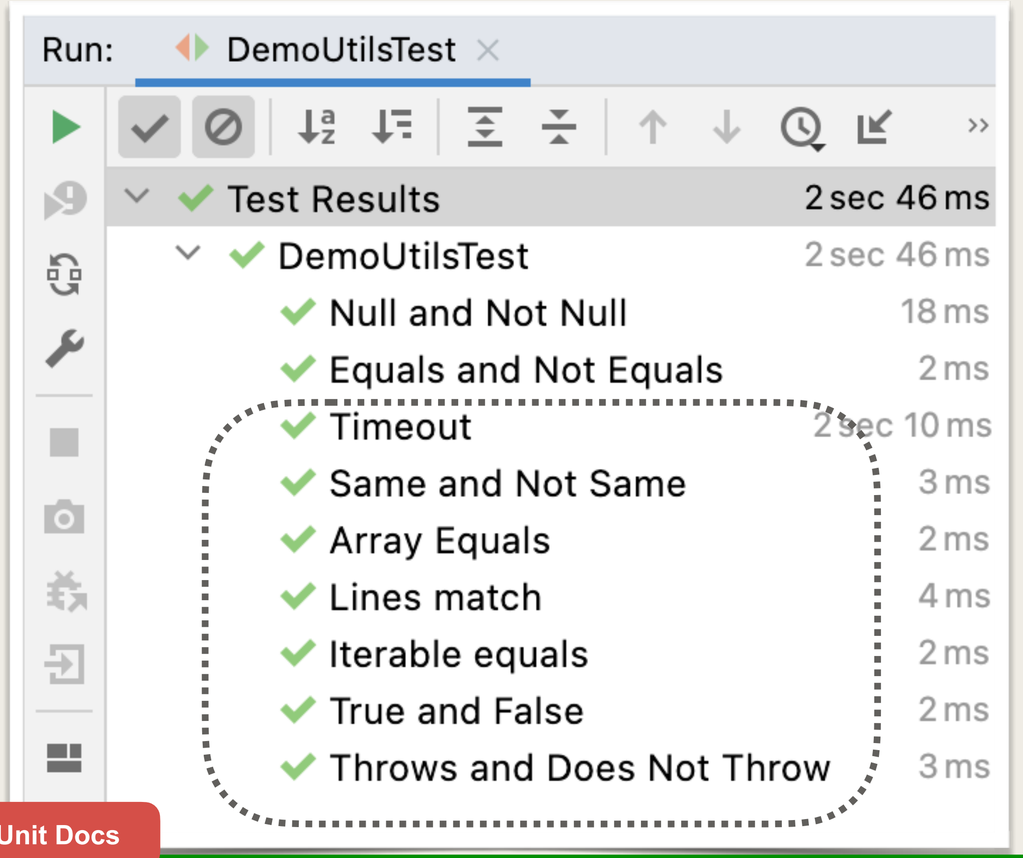
Annotation	Description
`@Order`	Manually specify the order with an `int` number - Order with lowest number has highest priority - Negative numbers are allowed

@Order

Remember
Lowest number has highest priority

DemoUtilsTest.java

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class DemoUtilsTest {
    ...

    @DisplayName("Equals and Not Equals")
    @Order(3)
    void testEqualsAndNotEquals()

    @DisplayName("Null and Not Null")
    @Order(1)
    void testNullAndNotNull()

    @DisplayName("Same and Not Same")
    void testSameAndNotSame()

    @DisplayName("True and False")
    void testTrueFalse()

    @DisplayName("Array Equals")
    void testArrayEquals()

    @DisplayName("Iterable equals")
    void testIterableEquals()

    @DisplayName("Lines match")
    void testLinesMatch()

    @DisplayName("Throws and Does Not Throw")
    void testThrowsAndDoesNotThrow()

    @DisplayName("Timeout")
    void testTimeout()
}
```



JUnit Docs

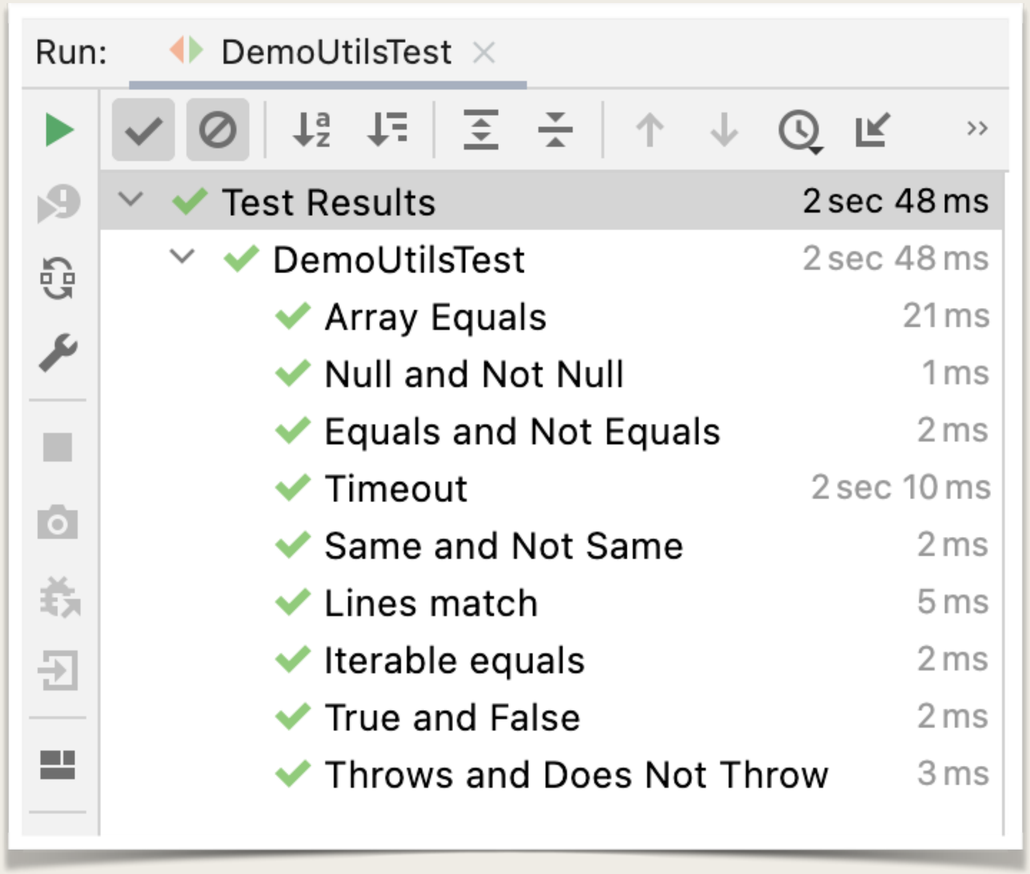
By default, test classes and methods will be ordered using an algorithm that is deterministic but intentionally nonobvious.

@Order - Negative Numbers

Remember
Lowest number has highest priority

DemoUtilsTest.java

```
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
class DemoUtilsTest {
    ...

    @DisplayName("Equals and Not Equals")
    @Order(3)
    void testEqualsAndNotEquals()

    @DisplayName("Null and Not Null")
    @Order(1)
    void testNullAndNotNull()

    @DisplayName("Same and Not Same")
    void testSameAndNotSame()

    @DisplayName("True and False")
    void testTrueFalse()

    @DisplayName("Array Equals")
    @Order(-7)
    void testArrayEquals()

    @DisplayName("Iterable equals")
    void testIterableEquals()

    @DisplayName("Lines match")
    void testLinesMatch()

    @DisplayName("Throws and Does Not Throw")
    void testThrowsAndDoesNotThrow()

    @DisplayName("Timeout")
    void testTimeout()
}
```



@Order - Duplicate Order Values

If duplicate @Order values then ...

JUnit Docs

> By default, test classes and methods will be ordered using an algorithm that is deterministic but intentionally nonobvious.

Additional Features

- If you have multiple test classes, you can order the classes
- Define custom order implementation
- Configure default order in properties file

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