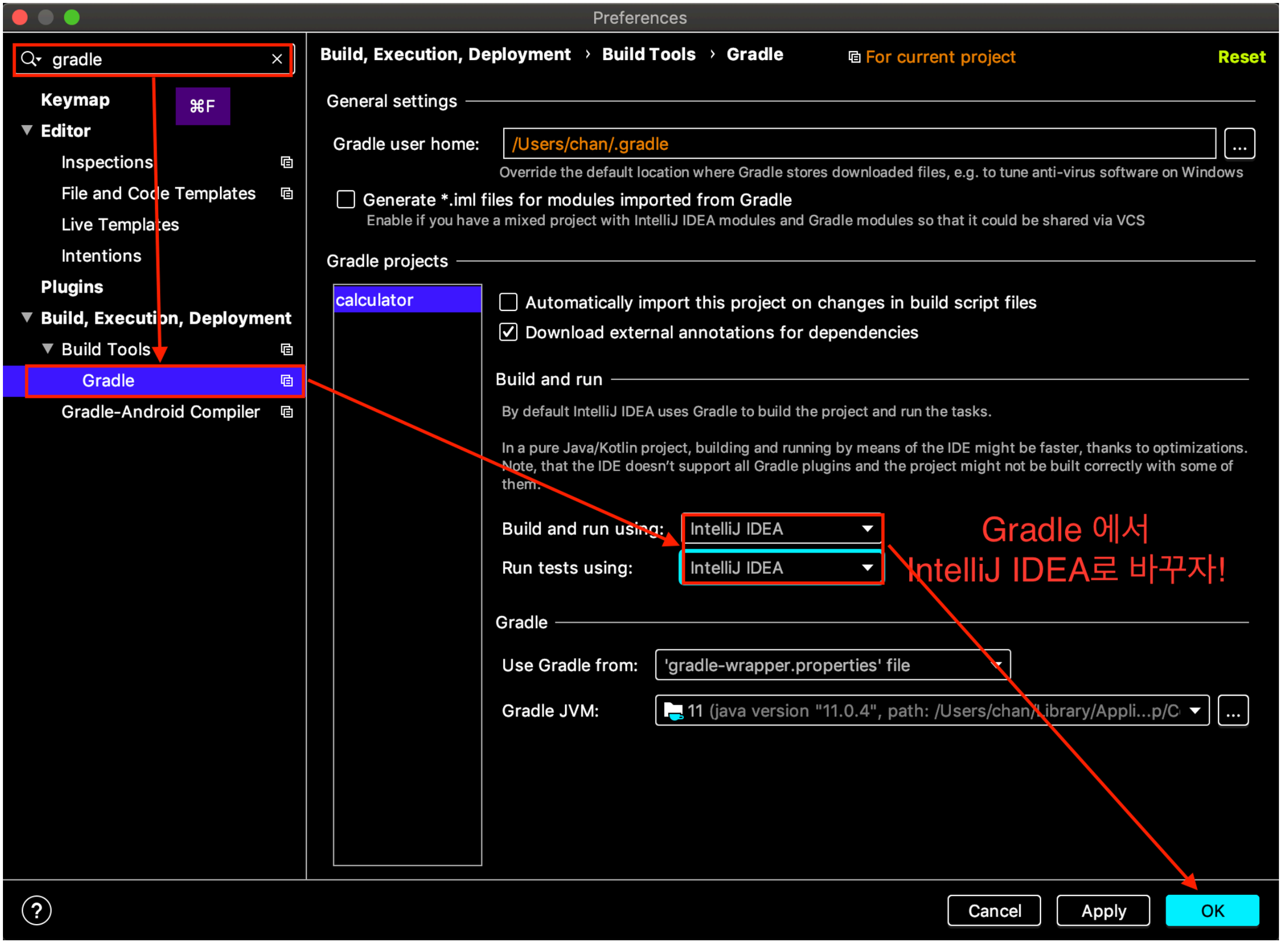
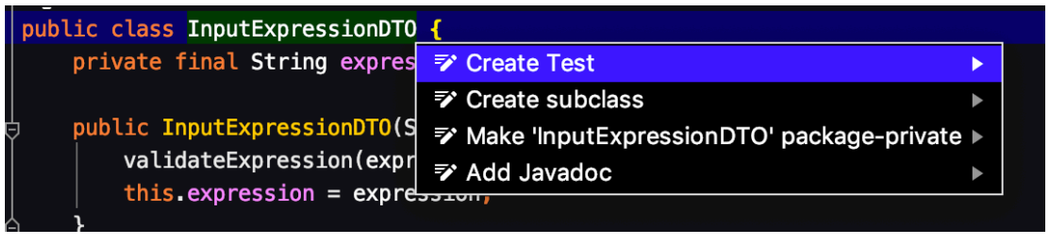
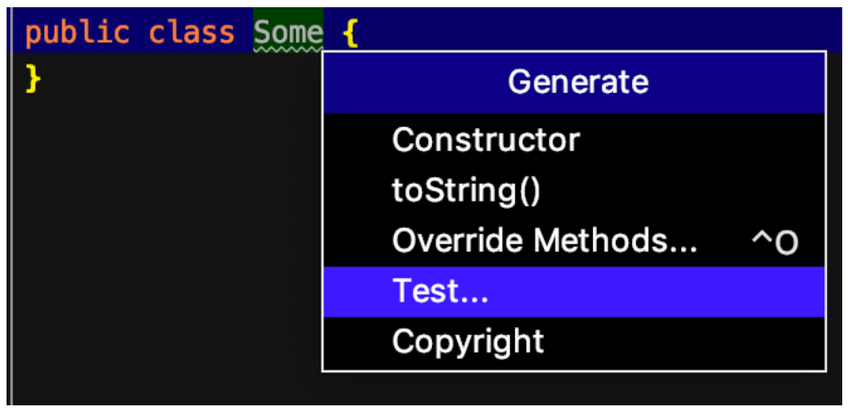
먼저 설정해 줍시다.


Preferences
gradle
⌘F
Keymap
Editor
Inspections
File and Code Templates
Live Templates
Intentions
Plugins
Build, Execution, Deployment
Build Tools
Gradle
Gradle-Android Compiler
Build, Execution, Deployment › Build Tools › Gradle
For current project
Reset
General settings
Gradle user home:
/Users/chan/.gradle
Override the default location where Gradle stores downloaded files, e.g. to tune anti-virus software on Windows
Generate *.iml files for modules imported from Gradle
Enable if you have a mixed project with IntelliJ IDEA modules and Gradle modules so that it could be shared via VCS
Gradle projects
calculator
Automatically import this project on changes in build script files
Download external annotations for dependencies
Build and run
By default IntelliJ IDEA uses Gradle to build the project and run the tasks.
In a pure Java/Kotlin project, building and running by means of the IDE might be faster, thanks to optimizations. Note, that the IDE doesn't support all Gradle plugins and the project might not be built correctly with some of them.
Build and run using:
IntelliJ IDEA
Run tests using:
IntelliJ IDEA
Gradle 에서
IntelliJ IDEA로 바꾸자!
Gradle
Use Gradle from:
'gradle-wrapper.properties' file
Gradle JVM:
11 (java version "11.0.4", path: /Users/chan/Library/Appli...p/C
Cancel
Apply
OK

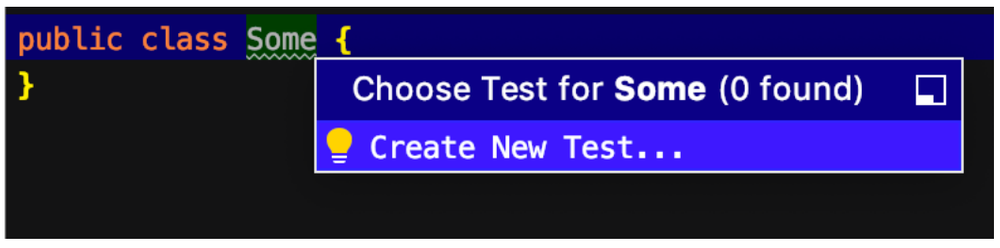
테스트 클래스 만들기



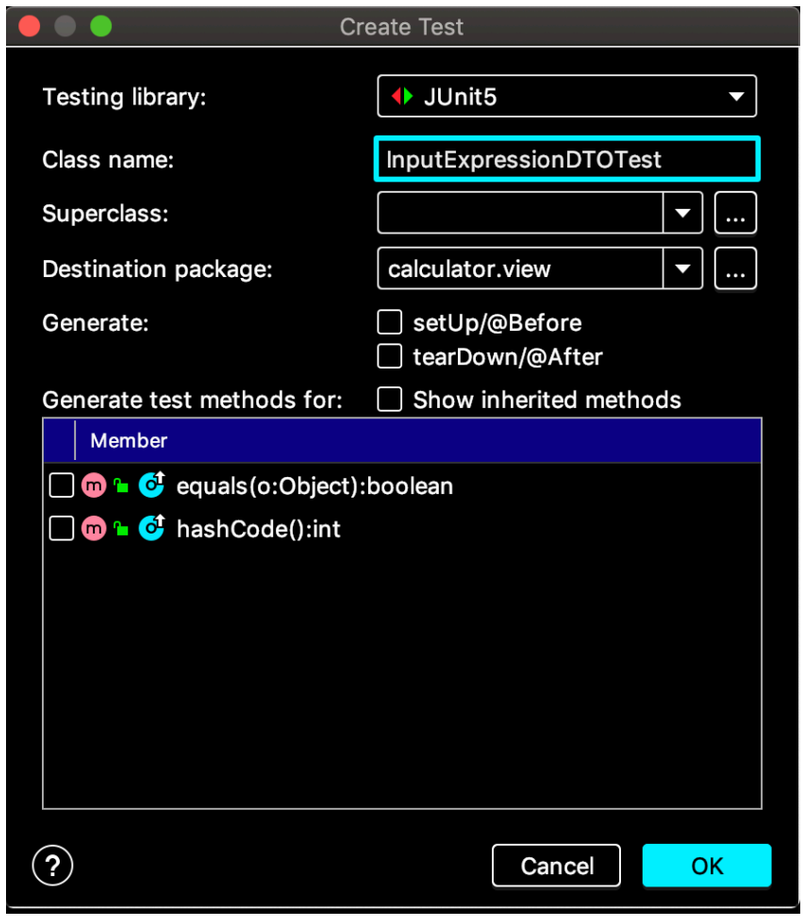
이름에 커서를 두고 option + enter



이름에 커서를 두고 cmd + n



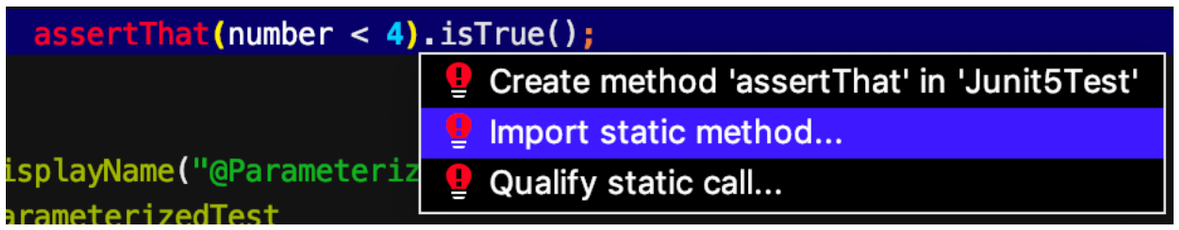
Cmd + shift + t

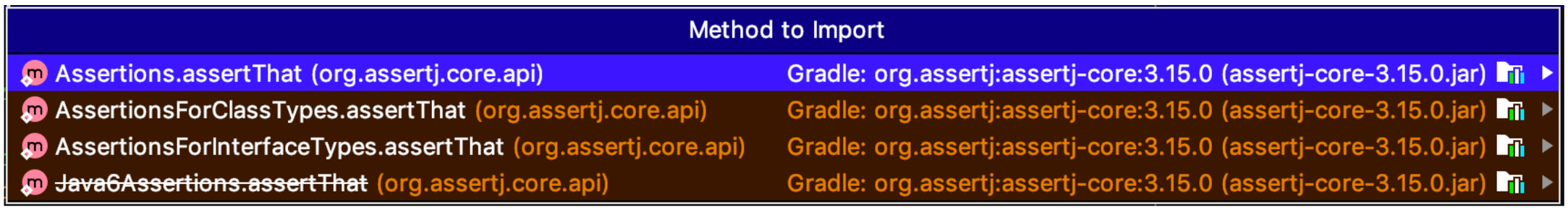


윈도우는 모르겠음

assertJ.core

```
import static org.assertj.core.api.Assertions.assertThat;
```





assertThat() 이렇게 까지 작성하면 밑줄이 생기면서 임포트 하기 쉬워진다.

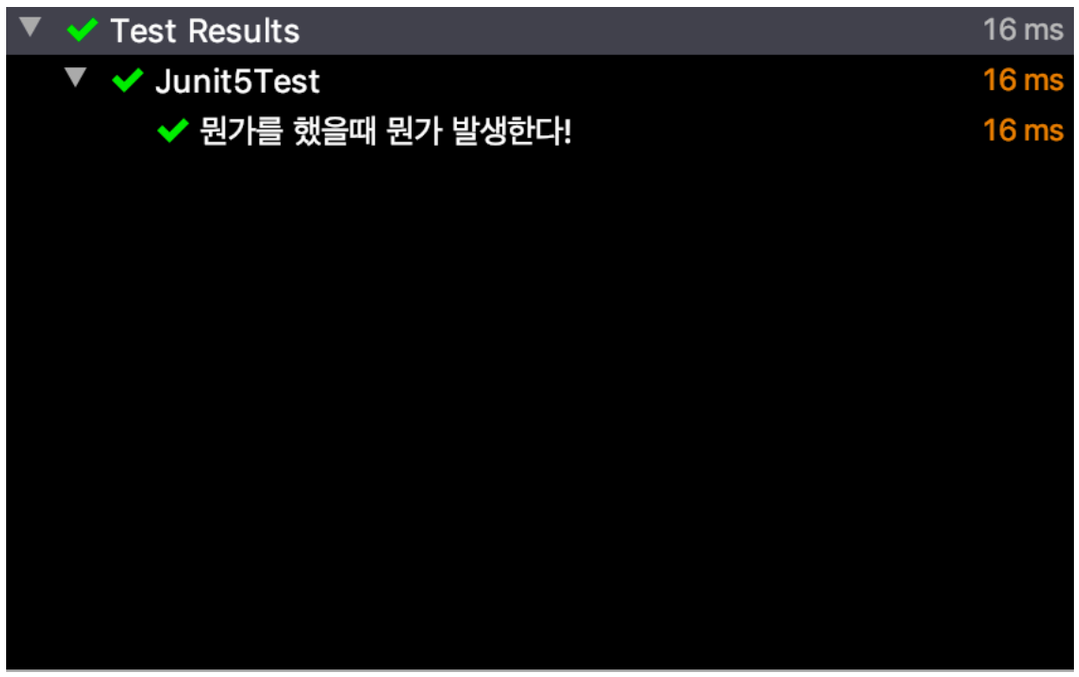
임포트는

윈도우 : alt + enter

맥 : option + enter

@DisplayName

```java
@DisplayName("뭔가를 했을때 뭔가 발생한다!")
@Test
void someTest() {
    //given
    //when
    //then
}
```



테스트가 어떤걸 테스트하는지 명시적으로 적어주도록 하자!

@ParameterizedTest

```java
@DisplayName("@ParameterizedTest 는 이렇게 쓸수 있다!")
@ParameterizedTest
@ValueSource(ints = {1, 2, 3})
void name(int number) {
    assertThat( actual: number < 4).isTrue();
}
```

```java
@DisplayName("@ParameterizedTest 는 이렇게 쓸수 있다 Csv!")
@ParameterizedTest
@CsvSource(value = {"1,0,true", "1,2,false"})
void name(int number1, int number2, boolean result) {
    assertThat( actual: number1 > number2).isEqualTo(result);
}
```

```java
@DisplayName("@ParameterizedTest 는 이렇게 쓸수 있다 nullAndEmpty!")
@ParameterizedTest
@NullAndEmptySource
public void dummy(String foo) {
    boolean result = false;

    if (foo == null || foo.equals("")) {
        result = true;
    }

    assertThat(result).isTrue();
}
```

* 예시를 위해 테스트코드 내부에 로직이 들어가 있다.
  위와 같이 로직을 추가하는 테스트는 작성하지 말자!

@ParameterizedTest

```java
class Foo {
    private int number;
    private String message;

    public Foo(int number, String message) {
        this.number = number;
        this.message = message;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Foo foo = (Foo) o;
        return number == foo.number &&
                Objects.equals(message, foo.message);
    }

    @Override
    public int hashCode() {
        return Objects.hash(number, message);
    }
}
```

```java
private static Stream<Arguments> someProvider() {
    return Stream.of(
            Arguments.of(new Foo( number: 1,  message: "a"), true),
            Arguments.of(new Foo( number: 112341234,  message: "asdfasdfasdfasd"), false)
    );
}

@DisplayName("@ParameterizedTest 는 이렇게 쓸수 있다 methodSource!")
@ParameterizedTest
@MethodSource("someProvider")
public void some(Foo provideFoo, boolean result) {
    Foo foo = new Foo( number: 1,  message: "a");

    assertThat(provideFoo.equals(foo)).isEqualTo(result);
}
```

매개변수에 객체를 넘길수도 있다!