

## A basic example of the JUnit 5 extension mechanism

Verisort framework relies on 3 major frameworks: JUnit5, Selenium, and Appium. A good and flexible framework allows it to intervene in its lifecycle. In this section, we will demonstrate JUnit 5 ability to allow interventions during its life cycle. Specifically, we will demonstrate the basic extension mechanism. To read more about JUnit 5 extensions, you can visit [this section](#) in the [JUnit 5 user guide](#).

### How does it work?

#### JUnit 5 Life cycle

The following diagram illustrates JUnit 5 life cycle. The diagram was copied from the JUnit 5 user guide:

---

#### BeforeAllCallback (1)

##### @BeforeAll (2)

```
LifecycleMethodExecutionExceptionHandler  
#handleBeforeAllMethodExecutionException (3)
```

#### BeforeEachCallback (4)

##### @BeforeEach (5)

```
LifecycleMethodExecutionExceptionHandler  
#handleBeforeEachMethodExecutionException (6)
```

#### BeforeTestExecutionCallback (7)

##### @Test (8)

```
TestExecutionExceptionHandler (9)
```

#### AfterTestExecutionCallback (10)

##### @AfterEach (11)

```
LifecycleMethodExecutionExceptionHandler  
#handleAfterEachMethodExecutionException (12)
```

#### AfterEachCallback (13)

##### @AfterAll (14)

```
LifecycleMethodExecutionExceptionHandler  
#handleAfterAllMethodExecutionException (15)
```

---

#### AfterAllCallback (16)

The orange parts are annotation based and are used in the code itself. The blue ones (callbacks) are extensions that are added as a separate file. **This extension file is the main demonstration of this page.**

### What will we need?

In order to use an extension class we will need to do 2 things:

1. Create an extension class
2. Connect the extension to the test code

### Create an extension class

An extension class is a class that implements at least one of the mentioned above callbacks. Here is a basic example:

```
1 public class ExampleExtension implements BeforeEachCallback {
2
3     @Override
4     public void beforeEach(ExtensionContext context) {
5         Report.info("This will be written before each test");
6     }
7 }
```

That's it. To see a more complex example of this class, you can take a look at its full implementation [here](#).

### Connect the extension to the test code

Actually, it's very simple:

```
1 @ExtendWith(ExampleExtension.class)
2 public class ExtensionsExampleTests {
3
4     @Test
5     @DisplayName("Use extensins 1")
6     public void useExtensions1() {
7         Report.info("Test 1 Starts");
8     }
9
10    @Test
11    @DisplayName("Use extensins 2")
12    public void useExtensions2() {
13        Report.info("Test 2 Starts");
14    }
15 }
```

and the log will show:

```
1 This will be written before each test
2 Test 1 Starts
3 This will be written before each test
4 Test 2 Starts
```