Automation Testing

Default Methods

A Day in the Life of an Automation Test Engineer

Marcelo acknowledges the work of assumptions and their application, which works in different modes of assumptions.

Now, Marcelo has to recognize the default methods, which are used to add new functionalities to the existing interfaces, and compatibility with code written for older versions of interfaces.

This lesson will help Marcelo to understand the concept of his project.

Learning Objectives

By the end of this lesson, you will be able to:

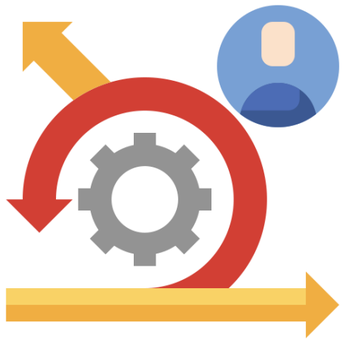
- Explain what is a default method
- Identify types of default methods
- Recognize how every default method works with example

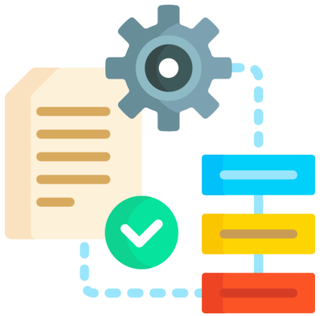
.

What Is Default Method?

What Is Default Method?

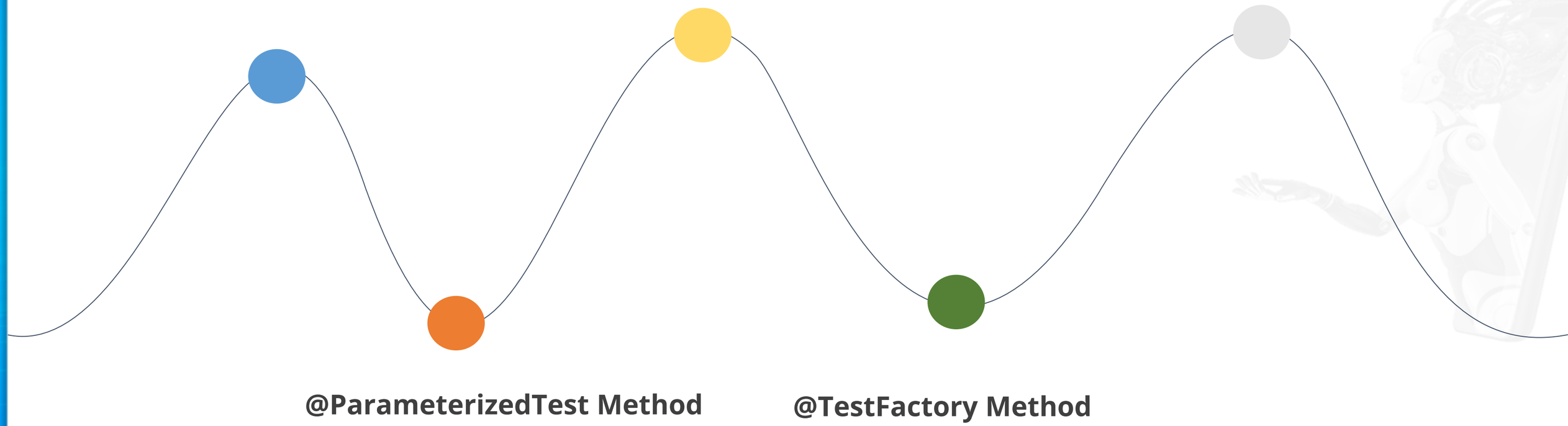
- The default methods are methods that can have a body.
- The most important use of default methods in interfaces is to provide additional functionality to a given type without breaking down the implementing classes.
- The default methods are also known as defender methods, virtual extension methods, and static methods.
- The interfaces can have the static methods as well, which is similar to static methods of classes.





Types of Default Method

Types of Default Method



@Test Method

A JUnit test is a method contained in a class that is only used for testing. This is called a test class. To define that a particular method is a test method, annotate it with the @Test annotation. This method executes the code under test.

Example of @Test Method

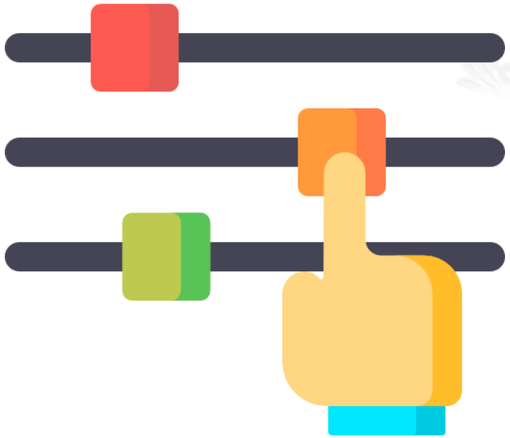
```
@TestInstance(Lifecycle.CLASS)
    interface TestLifecycleLogger
 {

    @BeforeAll
    default void beforeAllTests()
    {
     logger.info("Before all tests");
    }

 }
```

@ParameterizedTest Method

JUnit 5, the next generation of JUnit, facilitates writing developer tests with shiny new features. One such feature is parameterized tests. This feature enables us to execute a single test method multiple times with different parameters.



Example @ParameterizedTest Method

```
public class Junit5_Parameterized_Test
{
  @ParameterizedTest
  @ValueSource(ints = {8,4,2,6,10})

   void test_int_arrays(int arg)
    {
      System.out.println("arg => "+arg);
      assertTrue(arg % 2 == 0);
    }
}
```

@RepeatedTest Method

It is used to signal that the annotated method is a test template method, that should be repeated a specified number of times with a configurable display name.

Example of @RepeatedTest Method

```
public class RepeatedTest
{

  @RepeatedTest (value=3, name= SHORT_DISPLAY_NAME)
  void multiply ()
  {
    int a, b;
    a=10;
    b=15;
    assertEquals(150, (a*b), "Matched. Test status - Passed");
   }
}
```

@TestFactory Method

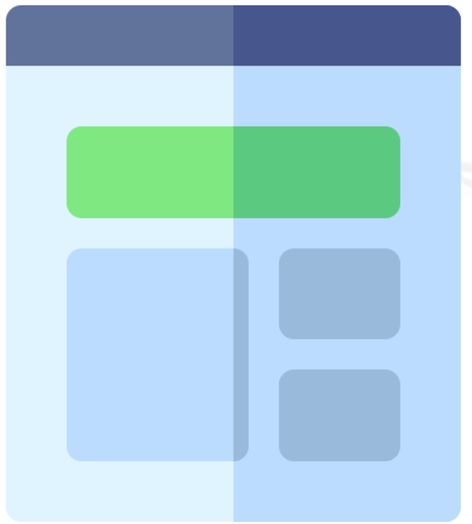
It tells JUnit that this is a factory for creating dynamic tests. As user can see, we're only returning a Collection of Dynamic Test. Each of the dynamic test consists of two parts, the name of the test or the display name, and an Executable.

Example of @TestFactory Method

```
@TestFactory
 Stream<DynamicTest> dynamicTestStream()
 {
   return IntStream.of(0, 3, 6, 9)
   .mapToObj(v ->
   dynamicTest(v + " is a multiple of 3“,() -> assertEquals(0,v % 3)));
 }
```

@TestTemplate Method

It is used to signal that the annotated method is a test template method. In contrast to @Test methods, a test template is not itself a test case but rather a template for test cases. It must be used together with at least one provider, otherwise the execution will fail.



Example of @TestTemplate Method

```
private TestTemplateInvocationContextfeatureEnabledContext(
  UserIdGeneratorTestCase userIdGeneratorTestCase)
 {
     return new TestTemplateInvocationContext()
     {
        @Override
        public String getDisplayName(int invocationIndex)
        {
            return userIdGeneratorTestCase.getDisplayName();
        }
     }
 }
```

Key Takeaways

- The Default methods are methods that can have a body.
- The Default methods are also known as defender methods, virtual extension methods, and static methods.
- Some default method which are test, repeat test, parameterized test, test factory, and test templates methods.
- A Repeat test is used to signal that the annotated method is a test template method.