Interview Questions and Answers on JUnit 5

1. What is JUnit 5?

Answer: JUnit 5 is the next generation of JUnit, a widely-used testing framework for Java. It introduces a modular structure, consisting of three sub-projects: JUnit Platform, JUnit Jupiter, and JUnit Vintage. JUnit 5 provides powerful new features and improvements over JUnit 4, including support for Java 8 and above, a more flexible architecture, and the ability to run tests written with older versions of JUnit.

2. What are the main components of JUnit 5?

Answer: JUnit 5 is composed of three main components:

- **JUnit Platform:** It serves as a foundation for launching testing frameworks on the JVM and defining the TestEngine API.
- **JUnit Jupiter:** It provides the new programming model and extension model for writing tests and extensions in JUnit 5.
- **JUnit Vintage:** It provides backward compatibility, enabling you to run JUnit 3 and JUnit 4 based tests on the JUnit 5 platform.

3. How do you create a simple test case in JUnit 5?

Answer: To create a simple test case in JUnit 5, you need to use the @Test annotation and a method to define your test logic. Here is an example:

java

Copy code

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;


class CalculatorTest {


  @Test

  void addition() {

    Calculator calculator = new Calculator();

    assertEquals(2, calculator.add(1, 1));

  }

}
```

4. What are some of the new annotations introduced in JUnit 5?

Answer: JUnit 5 introduces several new annotations, including:

- @Test: Marks a method as a test method.

- @BeforeEach: Indicates that the annotated method should be executed before each test method.
- @AfterEach: Indicates that the annotated method should be executed after each test method.
- @BeforeAll: Indicates that the annotated method should be executed once before all test methods in the current class.
- @AfterAll: Indicates that the annotated method should be executed once after all test methods in the current class.
- @DisplayName: Defines a custom display name for a test class or test method.
- @Disabled: Marks a test method or test class as disabled, so it will not be executed.

5. How can you run the same test with different parameters in JUnit 5?

Answer: In JUnit 5, you can use the @ParameterizedTest annotation to run the same test with different parameters. Here's an example using @ValueSource to provide parameters:

java

Copy code

```
import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

import static org.junit.jupiter.api.Assertions.assertTrue;


class ParameterizedTestExample {


  @ParameterizedTest

  @ValueSource(ints = {1, 2, 3, 4, 5})

  void testWithParameters(int number) {

    assertTrue(number > 0);

  }

}
```

6. What is the purpose of the @ExtendWith annotation in JUnit 5?

Answer: The @ExtendWith annotation is used to register extensions in JUnit 5. Extensions provide additional functionality to test classes and methods. For example, you can use extensions for setting up external resources, mocking, dependency injection, etc. Here's an example using the MockitoExtension:

java

Copy code

```
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;

@ExtendWith(MockitoExtension.class)
class MyMockitoTest {

  // Test methods using Mockito go here
}
```

7. How does JUnit 5 support dynamic tests?

Answer: JUnit 5 introduces the concept of dynamic tests, which allow you to generate test cases at runtime. You can use the @TestFactory annotation to create a factory method that returns a collection of DynamicTest instances. Here's an example:

java

Copy code

```
import org.junit.jupiter.api.DynamicTest;
import org.junit.jupiter.api.TestFactory;
import java.util.stream.Stream;
import static org.junit.jupiter.api.DynamicTest.dynamicTest;
import static org.junit.jupiter.api.Assertions.assertTrue;

class DynamicTestExample {

  @TestFactory
  Stream<DynamicTest> dynamicTests() {
    return Stream.of(1, 2, 3, 4, 5)
      .map(number -> dynamicTest("Test number: " + number,
        () -> assertTrue(number > 0)));
  }
}
```

8. What is an Assumption in JUnit 5 and how is it used?

Answer: Assumptions in JUnit 5 are used to halt the execution of a test if a certain condition is not met. They are useful for tests that should only run under certain conditions. If an

assumption fails, the test is aborted, not marked as failed. Here's an example using assumeTrue:

java

Copy code

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assumptions.assumeTrue;

import static org.junit.jupiter.api.Assertions.assertEquals;


class AssumptionExample {


  @Test

  void testOnlyOnWindows() {

    assumeTrue(System.getProperty("os.name").startsWith("Windows"));

    assertEquals(42, 42);

  }

}
```

9. How can you handle expected exceptions in JUnit 5?

Answer: In JUnit 5, you can handle expected exceptions using the assertThrows method. This method allows you to specify the exception type and a lambda expression containing the code that should throw the exception. Here’s an example:

java

Copy code

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertThrows;


class ExceptionHandlingTest {


  @Test

  void testException() {

    assertThrows(IllegalArgumentException.class, () -> {

      throw new IllegalArgumentException("Invalid argument");

    });
```

```
  }
}
```