1. What is TestNG and how is it different from JUnit?

Answer: TestNG isatesting framework inspired by JUnit and NUnit but introduces new functionalities that make itmore powerful and easier to use. It supports annotations, parallel execution, and flexible test configuration. Unlike JUnit, TestNG allows grouping of tests, data-driven testing using @DataProvider, and has built-in support for multi-threaded testing [1].

2. How do you install TestNG in Eclipse?

Answer:

1.Open Eclipse and go to Help -> Eclipse Marketplace.

2.Search for "TestNG" and click Go.

3.Click Install next to the TestNG plugin.

4.Follow the installation steps and restart Eclipse [1].

3. What are the main annotations in TestNG?

Answer:

- @Test: Marks a method as a test method.
- @BeforeMethod: Executes before each test method.
- @AfterMethod: Executes after each test method.
- @BeforeClass: Executes before the first method in the current class is invoked.
- @AfterClass: Executes after all the methods in the current class have been run.
- @BeforeSuite: Executes before the suite runs.
- @AfterSuite: Executes after the suite has run [1].

4. How do you run a TestNG test suite?

Answer:

- Create a testng.xml file that defines the suite and test classes.
- Right-click on the testng.xml file and select Run As -> TestNG Suite [1].

5. How do you handle dependencies in TestNG?

Answer:

- Use the dependsOnMethods attribute in the @Test annotation to specify method dependencies.

```
@Test
public void method1() {
    // Test code
 }

@Test(dependsOnMethods = {"method1"})
public void method2() {
    // Test code
}
```

6. What is the use of the @DataProvider annotation in TestNG?

Answer:

- The@DataProvider annotation is used to create a method that provides data to a test method. It allows running a test method multiple times with different data sets.

```
@DataProvider(name = "data-provider")
public Object[][] dpMethod() {
    return new Object[][] {{"data1"}, {"data2"}};
 }

@Test(dataProvider = "data-provider")
public void testMethod(String data) {
    System.out.println("Data is: " + data);
}
```

7. How do you run tests in parallel using TestNG?

Answer:

- Configure parallel execution in the testng.xml file using the parallel attribute.

```
<suite name="Suite" parallel="tests" thread-count="2">
```

```
    <test name="Test1">
        <classes>
            <class name="com.example.TestClass1"/>
        </classes>
    </test>
    <test name="Test2">
        <classes>
            <class name="com.example.TestClass2"/>
        </classes>
    </test>
</suite>
```

8. How do you generate reports in TestNG?

Answer:

- TestNG generates default HTML and XML reports. You can find these reports in the test-output folder of your project.
- For custom reports, implement the IReporter or ITestListener interfaces [1].

9. What is the @Factory annotation in TestNG?

Answer:

- The@Factory annotation is used to create instances of test classes dynamically. It allows running multiple test instances with different data.

```
@Factory
public Object[] factoryMethod() {
    return new Object[] {new TestClass("data1"), new TestClass("data2")};
}
```

10. How do you skip a test case in TestNG?

Answer:

- Usethe throw new SkipException("Skipping this test") statement within the test method to skip it.

```
@Test
```

```
public void testMethod() {
    if (someCondition) {
        throw new SkipException("Skipping this test");
    }
    // Test code
}
```

11. What is the default priority of a test method in TestNG?

Answer: The default priority of a test method in TestNG is 0. If no priority is specified, TestNG assigns a priority of 0 to the test method [1].

12. How do you disable a test method in TestNG?

Answer: You can disable a test method by setting the enabled attribute to false in the @Test annotation.

```
@Test(enabled = false)
public void testMethod() {
    // This test will be ignored
}
```

13. What is the use of the @Parameters annotation in TestNG?

Answer: The @Parameters annotation is used to pass parameters to test methods from the testng.xml file.

```
@Parameters({"param1", "param2"})
@Test
public void testMethod(String param1, String param2) {
 System.out.println("Parameter 1: " + param1);
 System.out.println("Parameter 2: " + param2);
}
```

In the testng.xml file:

```
<test name="Test">
 <parameter name="param1" value="value1"/>
 <parameter name="param2" value="value2"/>
```

```
    <classes>
        <class name="com.example.TestClass"/>
    </classes>
</test>
```

14. How do you create a custom listener in TestNG?

Answer: Implement the ITestListener interface to create a custom listener. Override the methods to define custom behavior.

```
public class CustomListener implements ITestListener {
 @Override
    public void onTestStart(ITestResult result) {
        System.out.println("Test started: " + result.getName());
    }

    @Override
    public void onTestSuccess(ITestResult result) {
        System.out.println("Test passed: " + result.getName());
    }

    @Override
    public void onTestFailure(ITestResult result) {
        System.out.println("Test failed: " + result.getName());
    }

 // Implement other methods as needed
}
```

Register the listener in the testng.xml file:

```
<listeners>
.CustomListener"/>
</listeners>
```

15. How do you retry a failed test in TestNG?

Answer: Implement the IRetryAnalyzer interface to define retry logic and associate it withthe test methodusing the@Test annotation.

```
public class RetryAnalyzer implements IRetryAnalyzer {
    private int retryCount = 0;
 private static final int maxRetryCount = 3;

 @Override
    public boolean retry(ITestResult result) {
        if (retryCount < maxRetryCount) {
            retryCount++;
            return true;
        }
        return false;
    }
}
Associate the retry analyzer with the test method:
@Test(retryAnalyzer = RetryAnalyzer.class)
// Test code
}
```

16. How do you group test methods in TestNG?

Answer: Use the groups attribute in the @Testannotation to group test methods.

```
@Test(groups = {"group1"})
public void testMethod1() {
    // Test code
}

@Test(groups = {"group2"})
```

```
public void testMethod2() {
    // Test code
}
```

In the testng.xml file, you can include or exclude groups:

```
<test name="Test">
    <groups>
        <run>
            <include name="group1"/>
            <exclude name="group2"/>
        </run>
    </groups>
    <classes>
        <class name="com.example.TestClass"/>
    </classes>
</test>
```

17. What is the @BeforeGroups and @AfterGroups annotation in TestNG?

Answer: The @BeforeGroups annotation is used to specify a method that should run before any test method belonging to a specified group. Similarly, the @AfterGroups annotation specifies a method that should run after all test methods belonging to a specified group have run.

```
@BeforeGroups("group1")
public void beforeGroup() {
 System.out.println("Before group1");
}


@AfterGroups("group1")
public void afterGroup() {
System.out.println("After group1");
}
```

18. How do you handle test dependencies in TestNG?

Answer: UsethedependsOnMethodsordependsOnGroupsattribute in the@Test annotationtospecifydependencies.

```
@Test
public void method1() {
    // Test code
}

@Test(dependsOnMethods = {"method1"})
public void method2() {
    // Test code
}
```

19. What is the default priority of a test method in TestNG?

Answer: ThedefaultpriorityofatestmethodinTestNGis0.Ifnopriority is specified, TestNG assigns a priority of 0 to the test method [1].

20. How do you disable a test method in TestNG?

Answer: You can disableatestmethodbysettingtheenabledattribute to falseinthe@Test annotation.

```
@Test(enabled = false)
public void testMethod() {
    // This test will be ignored
}
```

21. What is the use of the @Parameters annotation in TestNG?

Answer: The @Parametersannotationisusedtopassparameterstotest methods from thetestng.xmlfile.

```
@Parameters({"param1", "param2"})
@Test
public void testMethod(String param1, String param2) {
 System.out.println("Parameter 1: " + param1);
```

```
 System.out.println("Parameter 2: " + param2); } In
the testng.xml file: <test name="Test">
<parameter name="param1" value="value1"/>
<parameter name="param2" value="value2"/>



    <classes>
        <class name="com.example.TestClass"/>
    </classes>
</test>
```

22. How do you create a custom listener in TestNG?

Answer: Implement the ITestListener interface to create a custom listener. Override the methods to define custom behavior.

```
public class CustomListener implements ITestListener {
 @Override
    public void onTestStart(ITestResult result) {
        System.out.println("Test started: " + result.getName());
    }

    @Override
    public void onTestSuccess(ITestResult result) {
        System.out.println("Test passed: " + result.getName());
    }

    @Override
    public void onTestFailure(ITestResult result) {
        System.out.println("Test failed: " + result.getName());
    }
```

```
 // Implement other methods as needed
}
```

Register the listener in the testng.xml file:

```
<listeners>
.CustomListener"/>
</listeners>
```

23. How do you group test methods in TestNG?

Answer: Use the groups attribute in the @Test annotation to group test methods.

```
@Test(groups = {"group1"})
public void testMethod1() {
    // Test code
}

@Test(groups = {"group2"})
public void testMethod2() {
    // Test code
}
```

In the testng.xml file, you can include or exclude groups:

```
<test name="Test">
    <groups>
        <run>
            <include name="group1"/>
            <exclude name="group2"/>
        </run>
    </groups>
    <classes>
        <class name="com.example.TestClass"/>
    </classes>
```

</test> 24. What is the @BeforeGroups and @AfterGroups annotation in TestNG?

Answer:

The @BeforeGroups annotation is used to specify a method that should run before any test method belonging to a specified group. Similarly, the @AfterGroups annotation specifies a method that should run after all test methods belonging to a specified group have run.

```
@BeforeGroups("group1")
public void beforeGroup() {
 System.out.println("Before group1");
}

@AfterGroups("group1")
public void afterGroup() {
System.out.println("After group1");
}
```

25. How do you handle test dependencies in TestNG?

Answer: Use the dependsOnMethods or dependsOnGroups attribute in the @Test annotation to specify dependencies.

```
@Test
public void method1() {
    // Test code
}

@Test(dependsOnMethods = {"method1"})
public void method2() {
    // Test code
}
```