TestNG Annotations

In this lesson, we will see some of the annotations that TestNG provides.

WE'LL COVER THE FOLLOWING

- List of annotations
- Sample TestNG class explaining the annotations
  - Understanding the above code snippet

List of annotations

TestNG supports annotations for performing various operations like:

- `@Test` annotated method is considered as a test method. This annotation can be added at class level as well. When given at test method level, the one at method level will take precedence. A test method can be disabled by setting `@Test(enabled = false)`. By default, `enabled = true`.
- `@BeforeSuite` annotated method will run once per test suite before all the tests.
- `@AfterSuite` annotated method will run once per test suite after all the tests.
- `@BeforeTest` annotated method will run once per test before all the test methods.
- `@AfterTest` annotated method will run once per test after all the test methods.
- `@BeforeClass` annotated method will run once per every test class instance before all the test methods.
- `@AfterClass` annotated method will run once per every test class instance after all the test methods.

after all the test methods.

- `@BeforeMethod` annotated method will run once per every test method instance before all the test methods.
- `@AfterMethod` annotated method will run once per every test method instance after all the test methods.
- `@BeforeGroup` annotated method will run once per every test method instance before all the test methods that belong to the given group.
- `@AfterGroup` annotated method will run once per every test method instance after all the test methods that belong to the given group.
- `@Parameter` annotation on test method is to pass parameters to test methods.
- `@DataProvider` annotated method is used to create test methods or test classes at runtime with different parameters. In conjunction with `@Test` on the test method, TestNG will create multiple test methods at runtime. It can also be added to the test class constructor in conjunction with `@Factory` to pass parameters to create test class instances at runtime.
- `@Factory` can be used on a method that returns instances of test classes or on a test class constructor in conjunction with `@DataProvider`.
- `@Listeners` is used at test class level to takes the array of classes that implements a plethora of implementations of `ITestNGListener` interface like `IAlterSuiteListener`, `IAnnotationTransformer`, `IMethodInterceptor`, `IReporter`, etc. for different purposes.

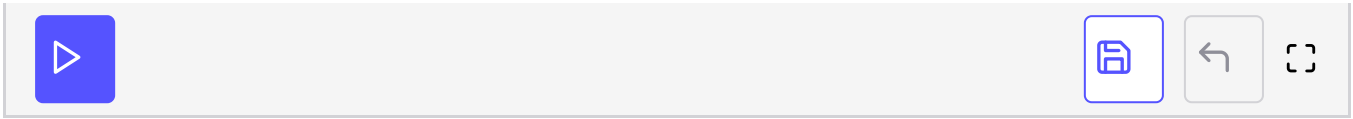
To know more about each of these annotations, please follow the link.

Sample TestNG class explaining the annotations

```
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.AfterSuite;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeSuite;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;
```

```
public class Main extends ParentClass {


	public Main() {
		System.out.println("testclass - inside constructor()");
	}

	@BeforeClass
	public void beforeClass() {
		System.out.println("testclass - inside beforeClass()");
	}

	@AfterClass
	public void afterClass() {
		System.out.println("testclass - inside afterClass()");
	}

	@BeforeMethod
	public void beforeMethod() {
		System.out.println("testclass - inside beforeMethod()");
	}

	@AfterMethod
	public void afterMethod() {
		System.out.println("testclass - inside afterMethod()");
	}

	@BeforeTest
	public void beforeTest() {
		System.out.println("testclass - inside beforeTest()");
	}

	@AfterTest
	public void afterTest() {
		System.out.println("testclass - inside afterTest()");
	}

	@BeforeSuite
	public void beforeSuite() {
		System.out.println("testclass - inside beforeSuite()");
	}

	@AfterSuite
	public void afterSuite() {
		System.out.println("testclass - inside afterSuite()");
	}

	@Test
	public void test() {
		System.out.println("testclass - inside test()");
	}
}

class ParentClass {

	public ParentClass() {
		System.out.println("parentclass - inside constructor()");
	}

	@BeforeClass
	public void parentBeforeClass() {
		System.out.println("parentclass - inside beforeClass()");
```

```
        }

        @AfterClass
        public void parentAfterClass() {
                System.out.println("parentclass - inside afterClass()");
        }

        @BeforeMethod
        public void parentBeforeMethod() {
                System.out.println("parentclass - inside beforeMethod()");
        }

        @AfterMethod
        public void parentAfterMethod() {
                System.out.println("parentclass - inside afterMethod()");
        }

        @BeforeTest
        public void parentBeforeTest() {
                System.out.println("parentclass - inside beforeTest()");
        }

        @AfterTest
        public void parentAfterTest() {
                System.out.println("parentclass - inside afterTest()");
        }

        @BeforeSuite
        public void parentBeforeSuite() {
                System.out.println("parentclass - inside beforeSuite()");
        }

        @AfterSuite
        public void parentAfterSuite() {
                System.out.println("parentclass - inside afterSuite()");
        }

}
```



Understanding the above code snippet

The following is the order for the execution of annotated methods.

- `@BeforeSuite`
- `@BeforeTest`
- `@BeforeClass`
- `@BeforeMethod`
- `@Test`
- `@AfterMethod`
- `@AfterClass`

- `@AfterTest`
- `@AfterSuite`

Configuration methods of the parent class will be called first and then the child class if the test class is extending another class having configuration methods. This parent hierarchy continues until the parent class is not in the Object class.

Now that a quick overview of annotations available in TestNG is covered, we can move on to running tests in parallel.