```
Case7:(Differnce b/w error and failure)
public class BankLoanService {
	public float calculateSimpleInterest(float pamount,float rate,float time) {
		System.out.println("BankLoanService.calculateSimpleInterest()");

		if(pamount<=0 || rate<=0 || time <=0)
			throw new IllegalArgumentException("Invalid inputs");
		return (pamount*rate*time)/100.0f;
	}
}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
@Test
public void calculateSimpleTestInterestWithInvalidInputs() {
	BankLoanService service = new BankLoanService();
	float actualInterest = service.calculateSimpleInterest(0,0,0);
	float expected = 2400000.12f;
	assertEquals(actualInterest, expected);//Exception generated in service class
is not handled in Test case , so it is "Error".
}

Run 1/1  Failure : 0 Error : 1
Output: java.lang.IllegalArgumentException: Invliad inputs
	at
in.ineuron.nitin.service.BankLoanService.calculateSimpleInterest(BankLoanService.ja
va:7)
	at
in.ineuron.nitin.test.TestBankLoanService.testCalcSimpleInterestWithInvalidInputs(T
estBankLoanService.java:38)
	at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native
Method)
	at
java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessor
Impl.java:77)
	at
java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethod
AccessorImpl.java:43)
	at java.base/java.lang.reflect.Method.invoke(Method.java:568)
	at
org.junit.platform.commons.util.ReflectionUtils.invokeMethod(ReflectionUtils.java:7
25)

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++
Case8:(TimeDuration:300000,TimeRequired for Executed is ::200000)
public class BankLoanService {
	public float calculateSimpleInterest(float pamount,float rate,float time) {
		System.out.println("BankLoanService.calculateSimpleInterest()");

		if(pamount<=0 || rate<=0 || time <=0)
			throw new IllegalArgumentException("Invalid inputs");

		try {
			Thread.sleep(10000);
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		return (pamount*rate*time)/100.0f;
	}
}
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
@Test
public void calculateSimpleTestInterestWithTimer() {
      BankLoanService service = new BankLoanService();
      assertTimeout(Duration.ofMillis(200000), () ->
service.calculateSimpleInterest(0, 0, 0));
}
Output: org.opentest4j.AssertionFailedError: execution exceeded timeout of 20000 ms
by 10007 ms
      at
org.junit.jupiter.api.AssertionFailureBuilder.build(AssertionFailureBuilder.java:15
2)
      at
org.junit.jupiter.api.AssertionFailureBuilder.buildAndThrow(AssertionFailureBuilder
.java:132)
      at org.junit.jupiter.api.AssertTimeout.assertTimeout(AssertTimeout.java:81)
      at org.junit.jupiter.api.AssertTimeout.assertTimeout(AssertTimeout.java:53)
      at org.junit.jupiter.api.Assertions.assertTimeout(Assertions.java:3356)
      at
in.ineuron.test.BankLoanServiceTest.calculateSimpleTestInterestWithTimer(BankLoanSe
rviceTest.java:41)
      at java.base/java.lang.reflect.Method.invoke(Method.java:577)
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)
      at java.base/java.util.ArrayList.forEach(ArrayList.java:1511)


=> assertTimeout(): To check whether business method execution is completed in the
specified time or not.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
Note:
@BeforeEach: To place common logic that should execute before each Test method
execution.
@AfterEach : To place common logic that should execute after the each Test method
execution.

case9:
class TestBankLoanService{
      private BankLoanService service;
      @BeforeEach
      public void setUp() {
            service = new BankLoanService();
      }
      @AfterEach
      public void clear() {
            service = null;
      }
}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
Case10:
@BeforeAll: To write common logic only for 1 time for all test methods.
@AfterAll : To place cleanup logic for all test methods.
These methods must be taken as static methods

public class TestBankLoanService{
      private static BankLoanService service;
      @BeforeAll
```

```
	public static void setUpOnce() {
		service = new BankLoanService();
	}
	@AfterAll
	public static void clearOnce() {
		service=null;
	}
}

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
Case11:
@Disabled   : Marks test method as skipped/ disabled/ ignored test method
@DisplayName: To give programmer choice non-technical names to Test case class and
test methods.

@DisplayName("TestBankLoanService class")
public class TestBankLoanService {
	@Test
	@DisplayName("Testing with small numbers")
	public void testcalcSimpleIntrestAmountWithSmallNumber() {
		float actual = service.calcSimpleIntrestAmount(100000, 2, 12);
		float expected = 24000.0f;
		assertEquals(expected, actual, "may results not matching");
	}

	@Test
	@Disabled
	@DisplayName("Testing with timer")
	public void testcalcSimpleIntrestAmountWithTimer() {
		assertTimeout(Duration.ofMillis(20000), ()->{
			service.calcSimpleIntrestAmount(100000, 2, 12);
		});
	}
}
Output: 2 testcases skipped
 Testing with small numbers
 Testing with timer


Case12:
@TestMethodOrder: Useful to specify execution order of test methods with different
possibilities.
@DisplayName("TestBankLoanService class")
@TestMethodOrder(value = MethodOrderer.OrderAnnotation.class)
public class TestBankLoanService {

	@Order(10)
	@Test
	public ....(){}

	@Order(0)
	@Test
	public ....(){}

	@Order(5)
	@Test
	public ....(){}
```

```
}
=> OrderAnnotation (best, we should add @Order(n) on top of test methods while
using this option.
=> n-> priority number high value indicates low priority and low value indicate
high priority.

Case13: Based on the MethodName(Alphabetical order)
@TestMethodOrder: Useful to specify execution order of test methods with different
possibilities.

@DisplayName("TestBankLoanService class")
@TestMethodOrder(value = MethodName.class)
public class TestBankLoanService {

}

Case14: Based on the DisplayName of the method
@TestMethodOrder(MethodOrderer.DisplayName.class)
public class BankLoanServiceTest {

}

Case15: Based on the Order of names(default followed by engine)
@DisplayName("BankLoanService")
@TestMethodOrder(MethodOrderer.Random.class)
public class BankLoanServiceTest {

}

Note:
MethodOrderer (l):
It is having multiple inner classes implementing same MethodOrderer (l) they are
a. MethodName
b. DisplayName (gives ambiguity with @DisplayName, so specify
MethodOrderer.DisplayName.class)
c. OrderAnnotation (best, we should add @Order(n) on top of test methods while
using this option.
   n-> priority number high value indicates low priority and low value indicate
high priority)
d. Random (default but gives ambiguity, so specify MethodOrderer.Random.class)
e. AlphaNumeric (deprecated)

In real Scenarios we need to execute the application/ project in 4 environments:
 a. Development(dev)
 b. Testing(test)
 c. UserAcceptance(uat)
 d. Production(prod)

@Tag: Useful to mark test methods to execute only in certain environment like
"dev", "test", "uat", "prod" and etc.
      So, that we can write separate test methods for "dev", "test" environment
based light weight setup like using MySQL,
      Tomcat server and etc. and similarly we can write separate test methods for
"uat", "prod" environment based heavy weight
      production ready setup like using Oracle, WebLogic,Wildfly and etc.

While running Test case class, we need to specify tags to include and exclude
a. In Eclipse Environment
      Right click Testcase class --> Run As --> Run Configurations -->Test tab -->
```

```
Configure
            Include tags: uat
            exclude tags: dev --> ok
            Test runner: JUnit 5
            Apply --> Run

++++++++++++++++++++++++
TestBankLoanService.java
++++++++++++++++++++++++
@Test
@DisplayName("Testing with Small Numbers")
@Tag("dev")
@Tag("uat")
public void testCalcSimpleInterestWithSmallNumbers() {
      System.out.println("\
nTestBankLoanService.testCalcSimpleInterestWithSmallNumbers()");
      float actualOutput = service.calculateSimpleInterest(100000, 2, 12);
      float expectedOutupt = 24000.0f;

      // Method checking for test-case and generating the output
      assertEquals(expectedOutupt, actualOutput);
}

@Test
@DisplayName("Testing with Big Numbers")
@Tag("uat")
public void testCalcSimpleInterestWithBigNumbers() {
      System.out.println("\
nTestBankLoanService.testCalcSimpleInterestWithBigNumbers()");
      float actualOutput = service.calculateSimpleInterest(10000000, 2, 12);
      float expectedOutupt = 2400000.345f;

            // Method checking for test-case and generating the output
      assertEquals(expectedOutupt, actualOutput, 0.5f, "Results are not Matching");
}
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
b. In Maven Environment
Specify the tag names as shown below under surefire plugin.
+++++++
pom.xml
+++++++
<plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>3.0.0-M5</version>
      <configuration>
            <goal>dev</goal>
            <excludedGroups>uat</excludedGroups>
      </configuration>
</plugin>
Right click on the Project --> Run As --> Maven test and check the output in
console.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++

Note: We can pass TestInfo (I) type parameter in the test method to know more the
current test method and its executing environment like
      tag name,display name, test class name, test method name and etc.
@Test
@Tag("dev")
```

```
public void calculateSimpleTestInterestWithWhenNoException(TestInfo info) {
      BankLoanService service = new BankLoanService();
      System.out.println(info.getClass()+" "+info.getTags()+"
"+info.getDisplayName()+" "+info.getTestMethod().get().getName()
                        +" "+info.getTestClass().get().getClass());
      assertDoesNotThrow(()->service.calculateSimpleInterest(10000, 2, 2));
}

Output
class org.junit.jupiter.engine.extension.TestInfoParameterResolver$DefaultTestInfo
[dev]
calculateSimpleTestInterestWithWhenNoException(TestInfo)
calculateSimpleTestInterestWithWhenNoException class java.lang.Class
```