

Test-Driven Development

A thick, hand-drawn style orange line that underlines the title, extending from the left edge of the text area towards the center of the slide.

Your way to write clean code

Agenda



TDD Introduction

Know what is TDD
Know how TDD works
Know why TDD



Get started with TDD



Practice TDD with simple examples



So let's get started

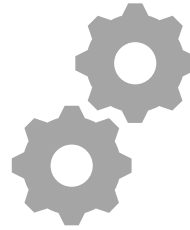
TDD Introduction

- **Test-driven development (TDD)** is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the code is improved so that the tests pass
- This is opposed to software development that allows code to be added that is not proven to meet requirements
- TDD is related to the test-first programming concepts of extreme programming, begun in 1999, but more recently has created more general interest in its own right
- TDD rediscovered by Kent Beck in 2003

Why TDD?



TDD can encourage simple designs and inspires confidence



TDD is an important part of agile programming



TDD is your way to write clean code

Tests drive the code – TDD process

1. Add a test for the new functionality or behavior
2. See it fail
3. Write enough code to make the test pass
4. Make sure all the previous tests pass as well
5. Refactor the code
6. Repeat until done



TDD benefits

TDD helps us achieve two important goals:

- Detect regression errors:
 - The sooner we detect a bug in the code, the faster and cheaper it is to fix
- Keep system design simple:
 - TDD is essentially a design tool and testing is just a side effect

Software Testing Phase Where Bug Found	Estimate Cost per Bug
System Test	\$5,000
Integration Test	\$500
Full Build	\$50
Unit Test/Test-Driven Development	\$5

Manual testing

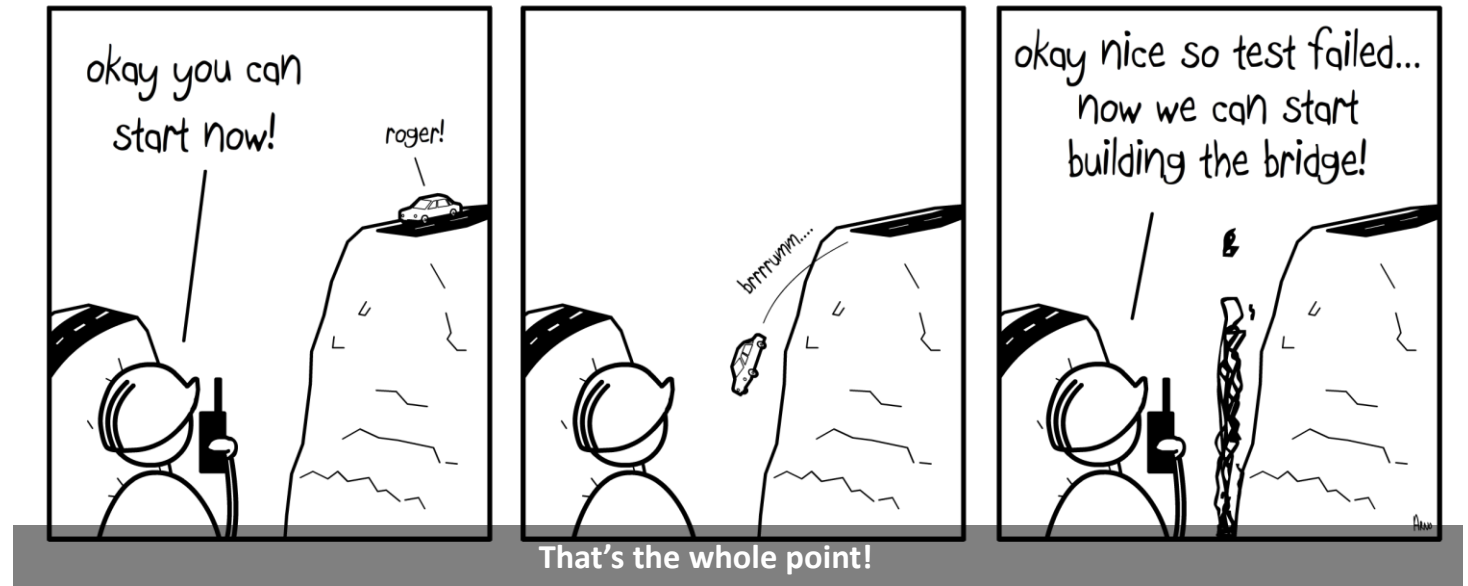
- Manual testing is bulky and time-consuming
- It is difficult to manually cover all the corner cases of any nontrivial software
- Humans are usually good at checking the happy paths (i.e., scenarios which always work)
- There is a great possibility that manual testing will miss some corner cases



TDD as a design tool

TDD as a design tool requires a change in mind-set

It will not happen in a day; it takes constant practice to master and reap the benefits



Levels of Testing

Your application should be composed of tests in each of the following levels:

- Unit testing
- Integration testing
- Acceptance testing

Define the list of testing types above and consider which one is strongly related to TDD?

Quality Matters



Work together in groups
and answer the following
question:



What is a GOOD TEST?



What is a BAD TEST?



Why should I strive to
write GOOD Tests?



JUnit Introduction

- JUnit, developed by Kent Beck and Erich Gamma, is one of the most popular unit-testing frameworks for Java developers
 - The first version of JUnit was released in 1997 developed by Kent Beck
 - It provided a lightweight framework, which enabled test creation by writing code in Java
 - It allowed developers to build test suites for every piece of their code
 - JUnit was integrated with all kinds of build tools and integrated development environments (IDEs).
-



JUnit Introduction

- Different versions of JUnit is available for Java developers
 - Check the documentation and associated techniques [here](#)
 - JUnit 5 is the latest version, and it is composed of several different modules from three different sub-projects, check it [here](#)
 - JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage
-



JUnit Introduction

- JUnit Platform = a TestEngine API for developing a testing framework that runs on the platform
 - JUnit Jupiter = a TestEngine for running JUnit 5 based tests on the platform
 - JUnit Vintage = a TestEngine for running JUnit 3 and JUnit 4 based tests on the platform
-

Assertions

assertEquals: verifies that the expected and the actual values are equal

assertArrayEquals: to assert that two arrays are equals, we can use the *assertArrayEquals*

assertNotNull* and *assertNull: to test if an object is *null* we can use the *assertNull* assertion

assertNotSame* and *assertSame: With *assertNotSame*, it's possible to verify if two variables don't refer to the same object

assertTrue* and *assertFalse: to verify that a certain condition is *true* or *false*, we can respectively use the *assertTrue* assertion or the *assertFalse* one

Fail: The *fail* assertion fails a test throwing an *AssertionFailedError*. It can be used to verify that an actual exception is thrown or when we want to make a test failing during its development



TDD Examples

Workflow:

Red -> Green -> Refactor

Start with *Red* to make sure that the new test executes as expected

To get to *Green*: Only write the minimum amount of code to make all tests pass (even if you know it's wrong in the long run)

Refactor to remove redundancy

TDD Example – Password Validator

Password rules:

- The password should be between 5 to 10 characters.
- The password should contain at least one uppercase letter.
- The password should contain at least one lowercase letter.
- The password should contain at least one special character.
- The password should contain at least one number.

The screenshot shows an IDE with a JUnit test runner on the left and a code editor on the right. The test runner indicates that the test 'ValidatePasswordTest' failed, with 1 failure and 0 errors. The code editor shows the implementation of the test, which currently fails with the message 'Not yet implemented'.

Package Explorer JUnit

Finished after 0.146 seconds

Runs: 1/1 Errors: 0 Failures: 1

ValidatePasswordTest [Runner: JUnit 5] (0.001 s)

- test() (0.001 s)

```
OtherClass.java Person.java Car.java
1+ import static org.junit.jupiter.api
4
5 class ValidatePasswordTest {
6
7-     @Test
8         void test() {
9             fail("Not yet implemented")
10        }
11
12    }
13
```


TDD Example – Password Validator

Start with Red – failed test

The screenshot shows an IDE with the JUnit test runner. The Package Explorer on the left shows the test results for `ValidatePasswordTest`. The test `test()` failed, as indicated by the red bar and the 'Failures: 1' count. The test results table shows:

Test	Time	Result
<code>test()</code>	0.001 s	Failed

The main editor shows the source code for `ValidatePasswordTest.java`:

```
1 import static org.junit.jupiter.api
2
3
4
5 class ValidatePasswordTest {
6
7     @Test
8     void test() {
9         fail("Not yet implemented")
10    }
11
12 }
13
```

TDD Example – Password Validator

- Follow with Green – passed test

```
1 public class ValidatePassword {
2     public boolean isValid(String password) {
3         if (password.length() >= 5 && password.length() <= 10)
4             return true;
5         else
6             return false;
7     }
8 }
9
10
11 }
```

Finished after 0.132 seconds

Runs: 1/1 Errors: 0 Failures: 0

> ValidatePasswordTest [Runner: JUnit 5] (0.000 s)

```
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.Test;
4
5 class ValidatePasswordTest {
6
7     @Test
8     void testPasswordLength() {
9         ValidatePassword vp = new ValidatePassword();
10         assertEquals(true, vp.isValid("Abc12"));
11     }
12
13 }
```

TDD Example – Password Validator

- Follow with Refactor – Delete duplicate code

The screenshot shows an IDE interface. On the left, the 'JUnit' tab is active, displaying a green progress bar and the text 'Finished after 0.13 seconds'. Below this, it shows 'Runs: 1/1', 'Errors: 0', and 'Failures: 0'. The test name 'ValidatePasswordTest [Runner: JUnit 5] (0.018 s)' is listed. On the right, the 'App.java' file is open, showing the following code:

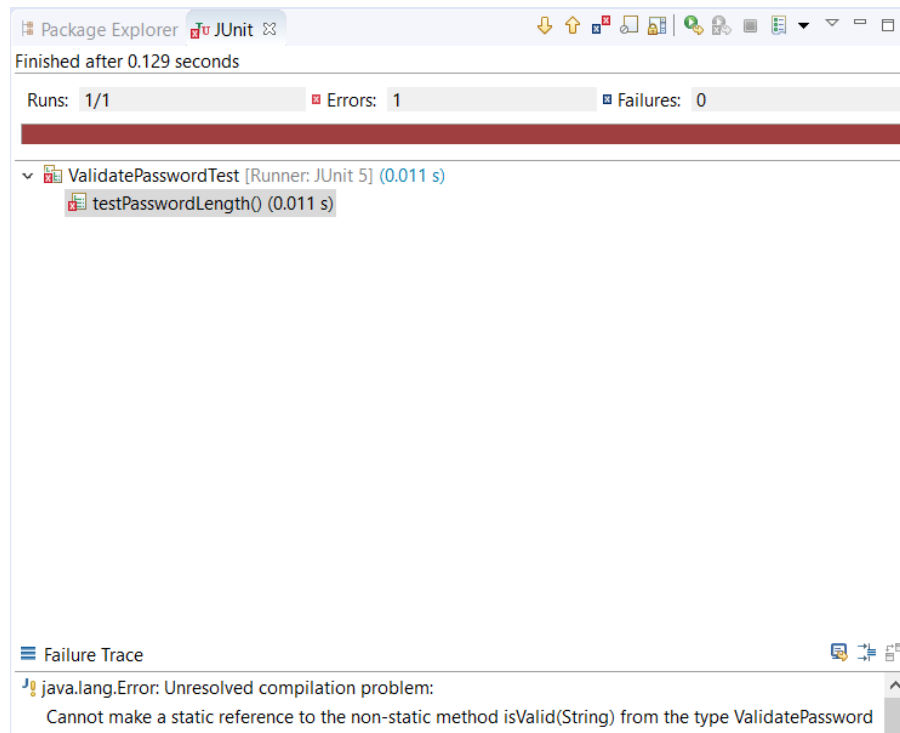
```
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.Test;
4
5 class ValidatePasswordTest {
6
7     @Test
8     void testPasswordLength() {
9         assertEquals(true, ValidatePassword.isValid("Abc12"));
10    }
11
12 }
```

An orange arrow points from a text box to the `ValidatePassword.isValid("Abc12")` call in the code.

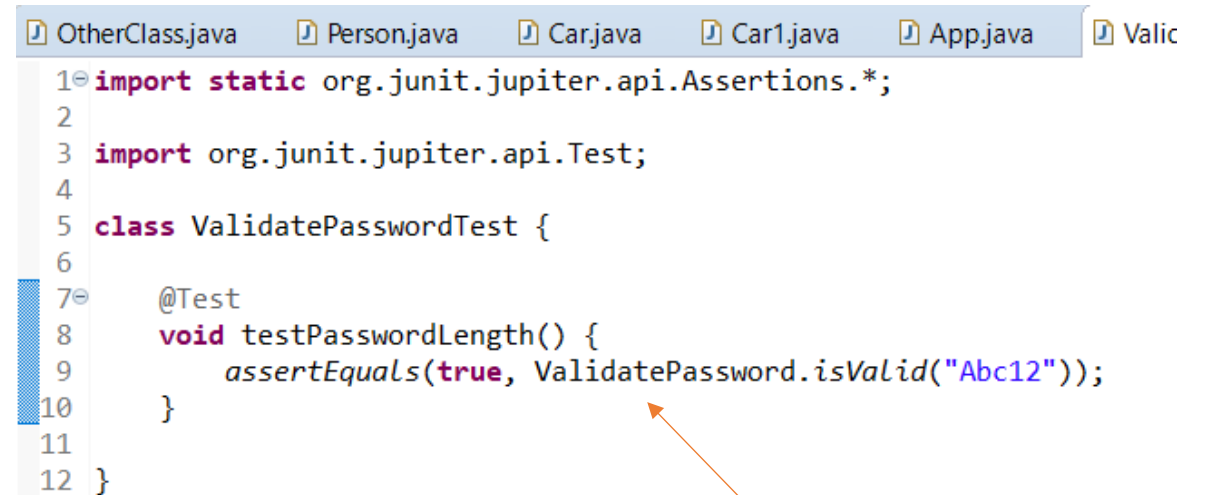
Refactor code as there is no need of creating instance of class `ValidatePassword()`

TDD Example – Password Validator

- Follow with Refactor – Delete duplicate code



The screenshot shows an IDE window with the Package Explorer on the left and the JUnit runner on the right. The runner shows a test run for `ValidatePasswordTest` with 1/1 runs, 1 error, and 0 failures. The error message is: `java.lang.Error: Unresolved compilation problem: Cannot make a static reference to the non-static method isValid(String) from the type ValidatePassword`. The test method `testPasswordLength()` is highlighted.



```
OtherClass.java Person.java Car.java Car1.java App.java Valid
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.Test;
4
5 class ValidatePasswordTest {
6
7     @Test
8     void testPasswordLength() {
9         assertEquals(true, ValidatePassword.isValid("Abc12"));
10    }
11
12 }
```

Check if the changes would pass?

TDD Example – Password Validator

- Follow with Refactor – Correct errors to get a passed test

The screenshot shows an IDE with two main panels. The left panel displays the JUnit test results, indicating a successful run. The right panel shows the Java code for the `ValidatePasswordTest` class, which includes a test method `testPasswordLength()` that calls `ValidatePassword.isValid("Abc12")`. An arrow points from a text box to the `isValid` method call in the code.

Package Explorer JUnit

Finished after 0.13 seconds

Runs: 1/1 Errors: 0 Failures: 0

> ValidatePasswordTest [Runner: JUnit 5] (0.018 s)

```
1 import static org.junit.jupiter.api.Assertions.*;
2
3 import org.junit.jupiter.api.Test;
4
5 class ValidatePasswordTest {
6
7     @Test
8     void testPasswordLength() {
9         assertEquals(true, ValidatePassword.isValid("Abc12"));
10    }
11
12 }
```

Change `isValid()` method to static in `ValidatePassword` class and run the test again.

TDD Example

- Password Validator

**Add test cases
for the other
rules**

Best Practice

Each Test must check one single piece of logic

Only one assert() in a test when the same logic is tested with multiple values

Test Business-Logic, not methods

Tests must be repeatable

Always check for null values and empty Optionals where appropriate

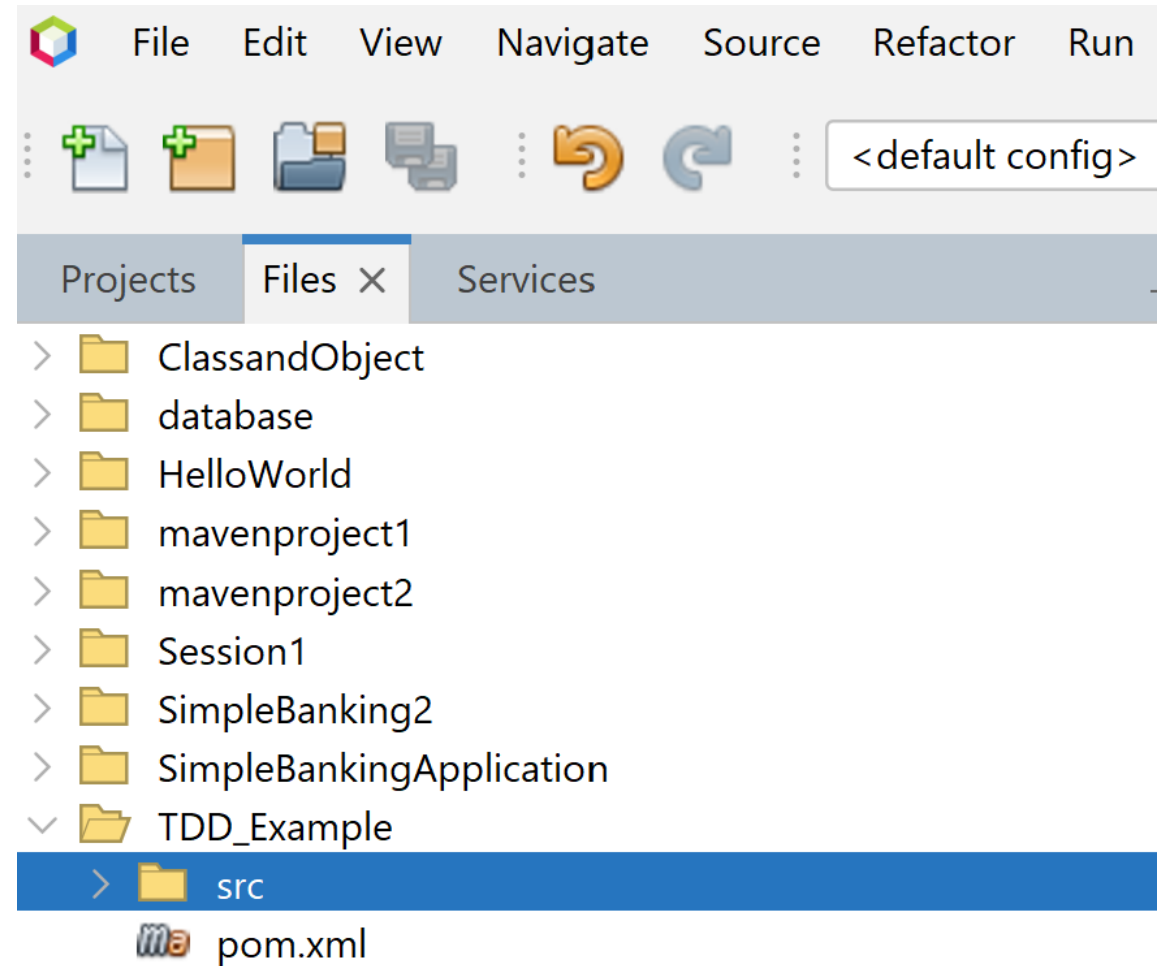
How to start in NetBeans

- Step 1: Set Up Your Project for JUnit 5
 - Create a New Java Project:
 1. Open NetBeans.
 2. Go to File > New Project.
 3. Select Java Application or Java with Maven > Java Application, depending on your preference, and click Next.

How to start in NetBeans

- 2. Click on Files and find pom.xml. Open the pom.xml file and add the following dependencies:

- `<dependencies>`
- `<dependency>`
- `<groupId>org.junit.jupiter</groupId>`
- `<artifactId>junit-jupiter-engine</artifactId>`
- `<version>5.8.2</version> <!-- Check for latest version -->`
- `<scope>test</scope>`
- `</dependency>`
- `</dependencies>`



How to start in NetBeans

- Allow some time to complete the download of Junit package after adding the dependency

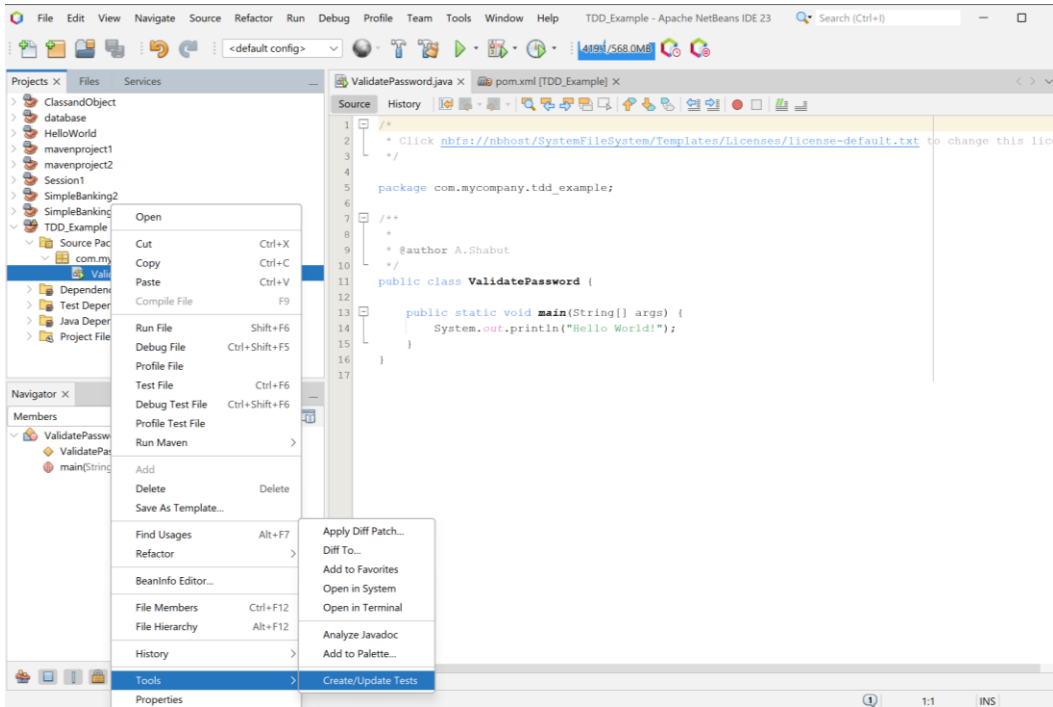
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany</groupId>
    <artifactId>TDD_Example</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.release>23</maven.compiler.release>
        <exec.mainClass>com.mycompany.tdd_example.ValidatePassword</exec.mainClass>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.8.2</version> <!-- Check for latest version -->
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```



How to start in NetBeans

- Step 2: Create a JUnit 5 Test Class
 1. Generate a Test Class:
 - In the Projects panel, right-click on the class you want to test (ValidatePassword).
 - Select Tools > Create Tests.
 - In the dialog that appears, choose JUnit 5.x as the test framework and click OK.
 - NetBeans will automatically create a new test class for you in the src/test/java directory, named ClassNameTest.java by default.
 - Change the name to ValidatePasswordTest.
 2. Add Test Methods:
 - In the generated test class, add test methods using the @Test annotation.
-

1) Create ValidatePasswordTest class

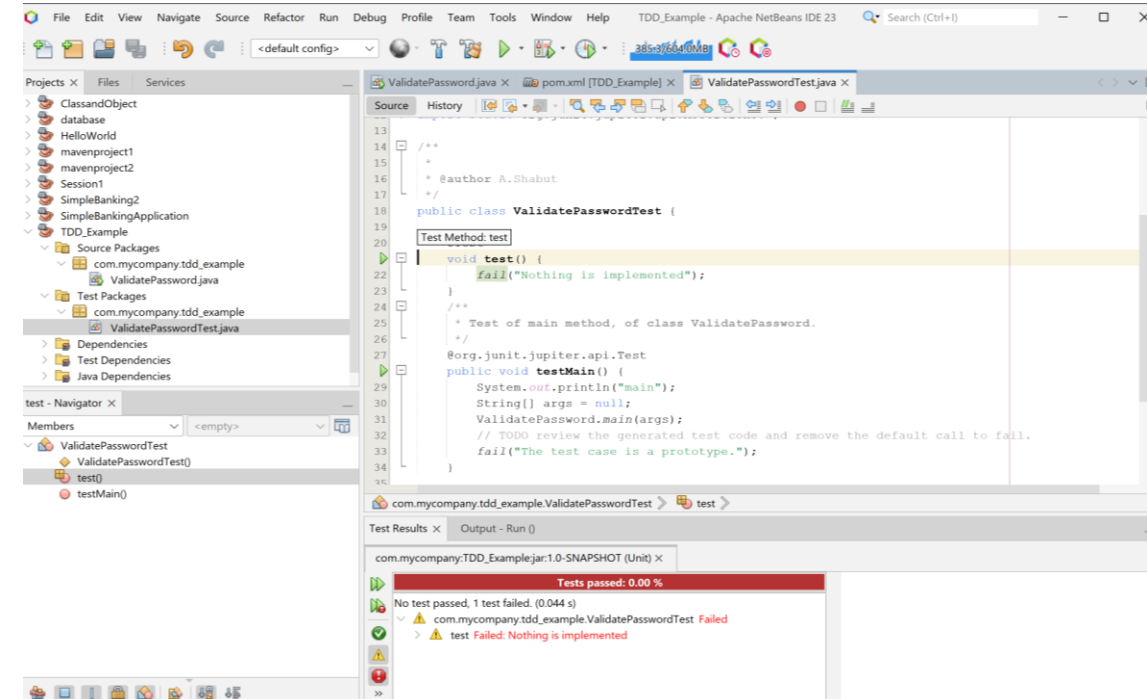


2) Fail the test

@Test

```
void test() {  
    fail("Nothing is Implemented");  
}
```

3) Click on the green button next to the method (Test Method test)



PasswordValidate

4) Add code to the PasswordValidate to test the password length

```
public static boolean isValidLength(String password) {  
    return password != null && password.length() >= 5 && password.length() <=11 ;  
}
```

PasswordValidateTest

5) Pass the test

@Test

```
public void testIsValidLength() {
```

```
    assertTrue(ValidatePassword.isValidLength("Password12!"), "Password with 12 characters should be valid.");
```

```
}
```

6) Add more Test Cases

```
    */  
    public class ValidatePasswordTest {  
  
        @Test  
        public void testIsValidLength() {  
            assertTrue(ValidatePassword.isValidLength("Pa  
        }  
        /**  
        * Test of main method, of class ValidatePassword  
        */  
        @org.junit.jupiter.api.Test  
        .mycompany.tdd_example.ValidatePasswordTest >> testIsValidLength >>  
        lts x Output - Run ()  
        rcompany:TDD_Example:jar:1.0-SNAPSHOT (Unit) x  
        Tests passed: 100.00 %  
        test passed. (0.037 s)
```

Resources

- [Test Driven Development: By Example \(oreilly.com\)](https://oreil.ly/) by Kent Beck - Reading this book will set your mind up for TDD and it really extracts the essence of test-driven development
- [How to unit test with JUnit 4 \(Junit 4 tutorial with examples\) \(javacodehouse.com\)](https://javacodehouse.com/junit4-tutorial-with-examples/) - Reading this blog will set your mind up for how to unit test with JUnit 4 (Junit 4 tutorial with examples)