Unit testing

Barbara Russo

SwSE - Software and Systems Engineering research group

unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Unit testing

- Each time you write a code module, you should write test cases for it
  - A possible exception: accessor methods (i.e., getters and setters)
    - Generally, accessor methods will be written error-free

Unit testing

- It focuses on faults within modules and code that could easily be broken

Test with annotation
The case of JUnit 4 / 5

Developers: Kent Beck, Erich Gamma, David Saff, Kris Vasudevan

Barbara Russo

SwSE - Software and Systems Engineering research group

JUnit 5

- **JUnit 5 = Platform + Jupiter + Vintage**
- **Platform** launches testing frameworks on the JVM
  - It also provides a *Console Launcher* to launch the platform from the command line and a *JUnit 4 based Runner* for running any TestEngine on the platform in a JUnit 4 based environment
- **Jupiter** is new model for writing tests and extensions in JUnit 5
  - Jupiter provides a TestEngine for running Jupiter based tests
- **Vintage** provides a TestEngine for running JUnit 3 and JUnit 4 based tests

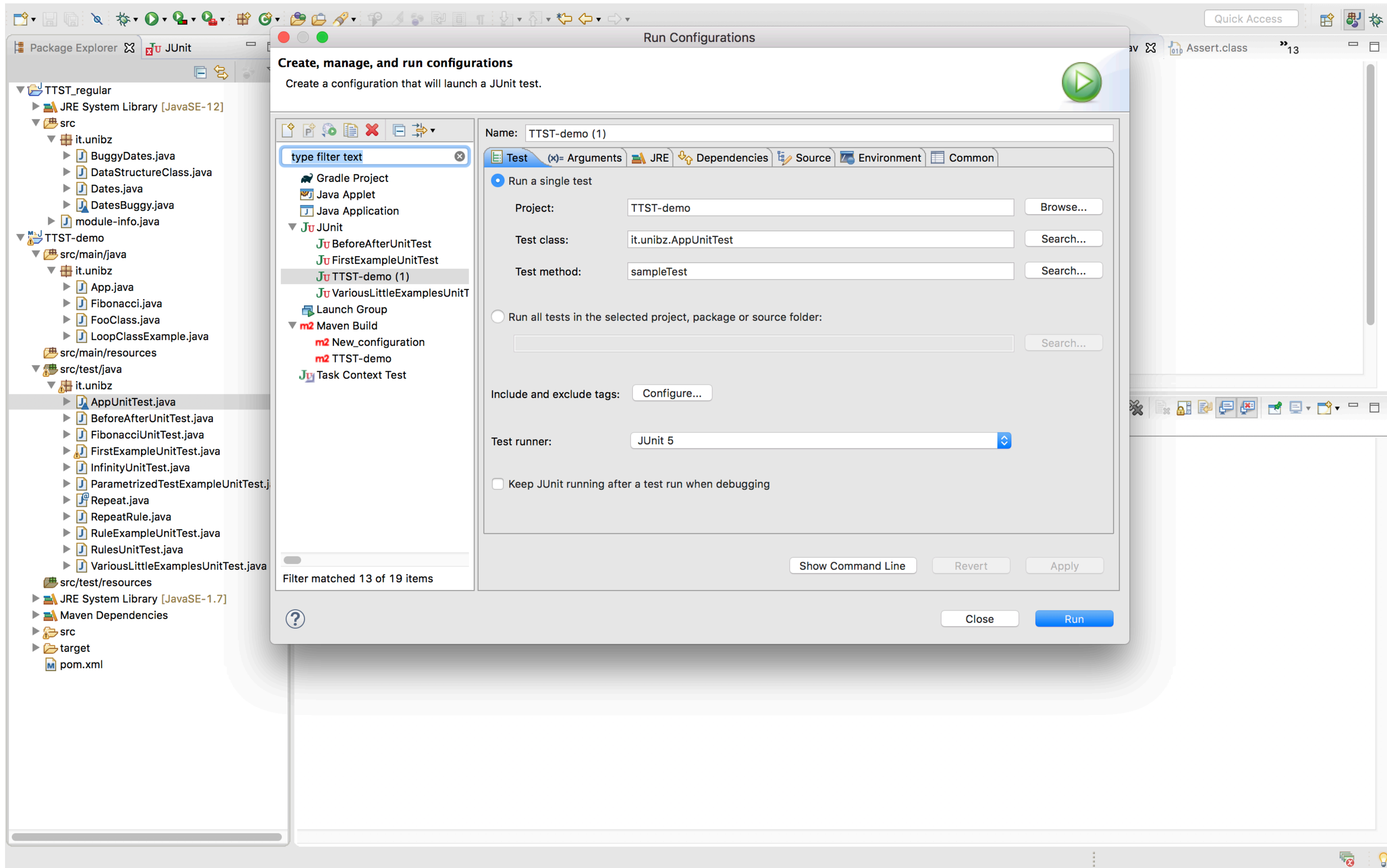
Annotations

- Test annotations characterize methods as test methods
- Annotations are strongly typed, so the compiler will flag any mistakes right away
- Test classes no longer need to extend anything (such as TestCase for JUnit 3)
- We can pass additional parameters to annotations

Runners

- We use JUnit runners to execute the test methods
- The runners can be configured in Eclipse
  - for whole project
  - for a single class
  - for a single method

Run a single test class or method



JUnit5

- https://junit.org/junit5/docs/current/user-guide/#writing-tests-annotations

- all core annotations are located in the org.junit.jupiter.api

Maven

- We use it to build and test java projects
- In particular, it provides
  - Dependency list
  - Unit test reports including coverage
- Maven has a central repository for jar and dependencies
  - https://maven.apache.org/repository/

The POM

- Project's configuration file in Maven
- XML structure
- It contains the majority of information required to build and test a project
  - It contains info on dependencies

Let’s have a look at it

workspace - TTST-demo/pom.xml - Eclipse IDE

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <project xmlns="http://maven.apache.org/POM/4.0.0"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
5      <modelVersion>4.0.0</modelVersion>
6
7      <groupId>it.unibz</groupId>
8      <artifactId>TTST-demo</artifactId>
9      <version>0.0.1-SNAPSHOT</version>
10
11     <dependencies>
12         <dependency>
13             <groupId>junit</groupId>
14             <artifactId>junit</artifactId>
15             <version>4.12</version>
16             <scope>test</scope>
17         </dependency>
18
19         <dependency>
20             <groupId>org.junit.jupiter</groupId>
21             <artifactId>junit-jupiter-engine</artifactId>
22             <version>5.5.1</version>
23             <scope>test</scope>
24         </dependency>
25
26         <dependency>
27             <groupId>org.junit.jupiter</groupId>
```

Overview | Dependencies | Dependency Hierarchy | Effective POM | pom.xml

Problems | Javadoc | Declaration | Console | Coverage

<terminated> TTST-demo [Maven Build] /Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java (Oct 14, 2019, 8:35:57 AM)

```
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------------< it.unibz:TTST-demo >-------------------------
[INFO] Building TTST-demo 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ TTST-demo ---
[INFO] Deleting /Users/barbaramini/unibz/Dropbox/Courses2019_2020/workspace/TTST-demo/target
```

project/dependencies/dependency | Writable | Smart Insert | 12 : 11

First example: use @Test

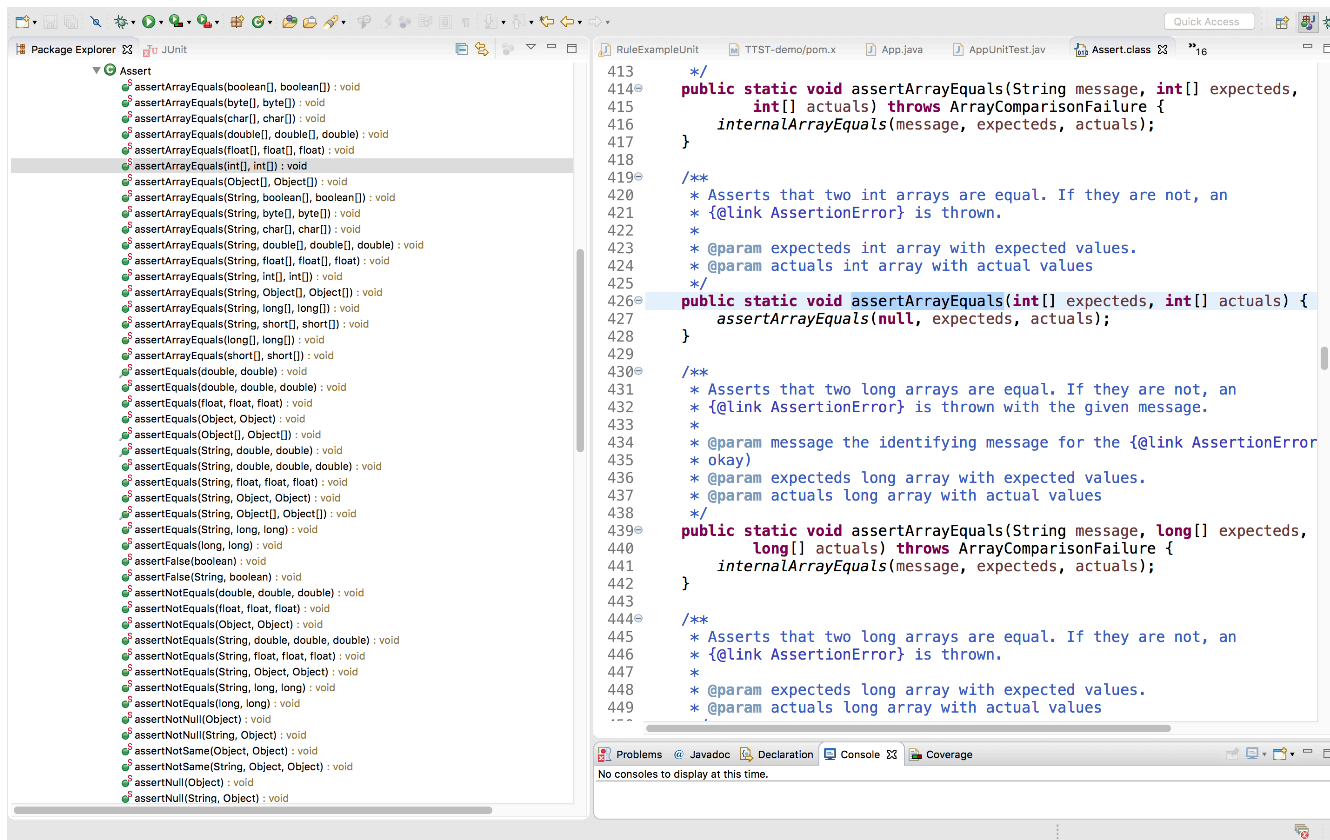
```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```

Example

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

 @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
```

Assertions



@Test

- It tags **public method that returns void** to run as a test method
  - JUnit first constructs a new instance of the class then invokes the annotated method
- Any *expected exceptions thrown* by the test will be *reported as a failure*
- Any *bug* is reported as *error*
- If *no exceptions/bugs* are thrown, the *test succeeds*

Run with JUnit configuration

Eclipse File Edit Source Refactor Navigate Search Project Run Window Help

workspace - TTST-demo/src/test/java/it/unibz/AppUnitTest.java - Eclipse IDE

Package Explorer | JUnit

Finished after 0.462 seconds

Runs: 36/36 Errors: 1 Failures: 3

- AppUnitTest [Runner: JUnit 5] (0.001 s)
  - test() (0.001 s)
- VariousLittleExamplesUnitTest [Runner: JUnit 5] (0.081 s)
- CalculatorMixedUnitTest [Runner: JUnit 5] (0.006 s)
- ParametrizedTestExampleUnitTest [Runner: JUnit 5] (0.009 s)
- RuleExampleUnitTest [Runner: JUnit 5] (0.016 s)
- VariousLittleExamplesUnitTest [Runner: JUnit 5] (0.002 s)
- FibonacciUnitTest [Runner: JUnit 5] (0.006 s)
- InfinityUnitTest [Runner: JUnit 5] (0.105 s)
  - infinity (0.105 s)
- RulesUnitTest [Runner: JUnit 5] (0.020 s)
- ExampleAnnotationsUnitTest [Runner: JUnit 5] (0.005 s)
  - testSize (0.001 s)
  - throwsException (0.001 s)
  - outOfBounds (0.000 s)
  - shouldTestExceptionMessage (0.001 s)
  - testRemove (0.001 s)

Failure Trace

```
org.opentest4j.AssertionFailedError: Not yet implemented
at it.unibz.AppUnitTest.test(AppUnitTest.java:12)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
```

RuleExampleUnit | TTST-demo/pom.x | App.java | AppUnitTest.jav

```java
package it.unibz;

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

class AppUnitTest {

    @Test
    void test() {
        //I want the test to fail. Look at the JUnit report
        fail("Not yet implemented");
    }

}
```

Optional parameters of @Test

- **expected and timeout**
- **expected:** checks a test method throws the expected exception
- The test returns an error if
  - If it *does not throw* an exception
  - If it *throws a different* exception than the one declared
  - If there is *no expected exception parameter* and an exception is thrown

Example: test succeeds

```
@Test(expected=IndexOutOfBoundsException.class)
public void outOfBounds() {
       new ArrayList<Object>().get(0);
 }
```

Finished after 0.022 seconds

Runs: 2/2 Errors: 0 Failures: 0

com.codebind.Calculator [Runner: JUnit 4] (0.000

Failure Trace

```
1  package com.codebind;
2  import static org.junit.Assert.assertEquals;
3
4  import java.util.ArrayList;
5
6  import org.junit.Test;
7
8  public class Calculator {
9      public int evaluate(String expression) {
10         int sum = 0;
11         for (String summand: expression.split("\\+"))
12             sum += Integer.valueOf(summand);
13         return sum;
14     }
15     @Test
16       public void evaluatesExpression() {
17         Calculator calculator = new Calculator();
18         int sum = calculator.evaluate("1+2+3");
19         assertEquals(6, sum);
20       }
21     @Test(expected=IndexOutOfBoundsException.class)
22     //@Test
23     public void outOfBounds() {
24            new ArrayList<Object>().get(0);
25      }
26 }
27
```

Console Problems Debug Shell

<terminated> Calculator [JUnit] /Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java (Oct 8, 2019, 6:49:20 PM)

Finished after 0.043 seconds

Runs: 2/2 Errors: 1 Failures: 0

- com.codebind.Calculator [Runner: JUnit 4] (0.000 s)
  - outOfBounds (0.000 s)
  - evaluatesExpression (0.000 s)

Failure Trace

```
java.lang.IndexOutOfBoundsException: Index 0 out of l
at java.base/jdk.internal.util.Preconditions.outOfBoun
at java.base/jdk.internal.util.Preconditions.outOfBoun
at java.base/jdk.internal.util.Preconditions.checkIndex
at java.base/java.util.Objects.checkIndex(Objects.java
at java.base/java.util.ArrayList.get(ArrayList.java:458)
at com.codebind.Calculator.outOfBounds(Calculator.ja
```

```java
1  package com.codebind;
2  import static org.junit.Assert.assertEquals;
3  
4  import java.util.ArrayList;
5  
6  import org.junit.Test;
7  
8  public class Calculator {
9      public int evaluate(String expression) {
10         int sum = 0;
11         for (String summand: expression.split("\\+"))
12             sum += Integer.valueOf(summand);
13         return sum;
14     }
15     @Test
16       public void evaluatesExpression() {
17         Calculator calculator = new Calculator();
18         int sum = calculator.evaluate("1+2+3");
19         assertEquals(6, sum);
20       }
21     //@Test(expected=IndexOutOfBoundsException.class)
22     @Test
23     public void outOfBounds() {
24             new ArrayList<Object>().get(0);
25      }
26 }
27 
```

Console | Problems | Debug Shell

<terminated> Calculator [JUnit] /Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java (Oct 8, 2019, 6:48:04 PM)

Finished after 0.039 seconds

Runs: 5/5 Errors: 1 Failures: 1

- it.unibz.FirstExampleUnitTest [Runner: JUnit 4] (0.000 s)
  - outOfBounds1 (0.000 s)
  - throwsException (0.000 s)
  - outOfBounds (0.000 s)
  - evaluatesExpression (0.000 s)
  - shouldTestExceptionMessage (0.000 s)

Failure Trace

```
java.lang.Exception: Unexpected exception, expected<java.lang.ArithmeticException> but was<java.lang.IndexOutOfBoundsException>
Caused by: java.lang.IndexOutOfBoundsException: Index 0 out of bounds for length 0
at java.base/jdk.internal.util.Preconditions.outOfBounds(Preconditions.java:64)
at java.base/jdk.internal.util.Preconditions.outOfBoundsCheckIndex(Preconditions.java:70)
at java.base/jdk.internal.util.Preconditions.checkIndex(Preconditions.java:248)
at java.base/java.util.Objects.checkIndex(Objects.java:372)
at java.base/java.util.ArrayList.get(ArrayList.java:458)
at it.unibz.FirstExampleUnitTest.outOfBounds1(FirstExampleUnitTest.java:37)
... 17 more
```

BeforeAfterUnit | FirstExampleUni | AppUnitTest.jav | Assert.class

```java
6
7  import org.hamcrest.CoreMatchers;
8  import org.junit.Rule;
9  import org.junit.Test;
10 import org.junit.rules.ExpectedException;
11
12 public class FirstExampleUnitTest {
13     ArrayList<Integer> myList;
14
15     public int evaluate(String expression) {
16         int sum = 0;
17         for (String summand: expression.split("\\+"))
18             sum += Integer.valueOf(summand);
19         return sum;
20     }
21     @Test
22     public void evaluatesExpression() {
23         FirstExampleUnitTest calculator = new FirstExampleUnitTe
24         int sum = calculator.evaluate("1+2+3");
25         assertEquals(6, sum);
26     }
27     @Test(expected=IndexOutOfBoundsException.class)
28     //@Test
29     public void outOfBounds() {
30         new ArrayList<Integer>().get(0);
31     }
32
33     //Check whether the exception is the one expected
34     //@Test(expected=IndexOutOfBoundsException.class)
35     @Test(expected=ArithmeticException.class)
36     public void outOfBounds1() {
37         new ArrayList<Object>().get(0);
38         myList.get(1);
39     }
40
41     @Rule
42     public ExpectedException thrown = ExpectedException.none();
43
```

Problems | Javadoc | Declaration | Console | Coverage

<terminated> FirstExampleUnitTest [JUnit] /Library/Java/JavaVirtualMachines/jdk-12.0.1.jdk/Contents/Home/bin/java (Oct 14, 2019

Writable | Smart Insert | 34 : 7

Optional parameters of @Test

- **timeout** causes a test to fail if it takes longer than a specified amount of clock time (measured in milliseconds)
- The test execution returns a time-out **error**

```
@Test(timeout=100)
public void infinity() {
     while(true);
}
```

Test Fixtures

- A **test fixture** is a fixed state of a set of objects used as a baseline for running tests

- JUnit provides annotations so that test classes can have fixture run **before or after** tests

Test Fixtures

- When a test class contains multiple methods to test, you can define **two void methods** that initialize and release respectively the common objects used in all tests
- You can call them *setup()* and *tearDown()*
- Use the tag @BeforeAll and @AfterAll to identify them

Example

```
ArrayList<Integer> myList;

@BeforeAll
public void initialize() {
    myList= new ArrayList<Integer>();
}

@Test
 public void testSize() {
    System.out.println(myList+" uses sizeList");
}
```

Example

```
public class Example {
   List myList;
   @BeforeAll
    public void setUp() {
        myList= new ArrayList();
   }
   @Test
    public void testSize() {
      System.out.println{myList + “it uses sizeList”};
   }
   @Test
    public void testRemove() {
     System.out.println{“it uses removeList”};
   }
}
```

@SuiteClasses

- The @SuiteClasses annotation specifies the classes to be executed when a class annotated with @RunWith(Suite.class) is run

Imports

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses(ATest.class, BTest.class, CTest.class)

public class ABCSuite {
}
```

Parametrised tests

Barbara Russo
SwSE - Software and Systems Engineering research group

Separation of concerns

- A parameterised unit test is simply a test method that *takes parameters, calls the code under test, and states assertions*

Parameterized.class runner

- The custom runner **Parameterized** implements parameterised tests
- Instances are created for the cross-product of the test methods and the test data elements
- Tests are run in parallel!

Examples

JUnit 4

```
import static org.junit.Assert.assertEquals;
import java.util.Arrays;
import java.util.Collection;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
```

```java
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 },
{ 5, 5 }, { 6, 8 } });
    }
    private int fInput;
    private int fExpected;
    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }
    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
public class Fibonacci {
    public static int compute(int n) {
      int result = 0;
        if (n <= 1) {
           result = n;
        } else {
           result = compute(n - 1) + compute(n - 2);
        }
        return result;
    }
}
```

Representative values that may have found with test strategies (e.g., category partition)

```java
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 },
{ 5, 5 }, { 6, 8 } });
    }
    private int fInput;
    private int fExpected;
    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }
    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
public class Fibonacci {
    public static int compute(int n) {
      int result = 0;
        if (n <= 1) {
           result = n;
        } else {
           result = compute(n - 1) + compute(n - 2);
        }
        return result;
    }
}
```

Representative values that may have found with test strategies (e.g., category partition)

```java
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 },
{ 5, 5 }, { 6, 8 } });
    }
    private int fInput;
    private int fExpected;
    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }
    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
public class Fibonacci {
    public static int compute(int n) {
      int result = 0;
        if (n <= 1) {
           result = n;
        } else {
           result = compute(n - 1) + compute(n - 2);
        }
        return result;
    }
}
```

Representative values that may have found with test strategies (e.g., category partition)

```java
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 },
{ 5, 5 }, { 6, 8 } });
    }
    private int fInput;
    private int fExpected;
    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }
    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
public class Fibonacci {
    public static int compute(int n) {
      int result = 0;
        if (n <= 1) {
           result = n;
        } else {
           result = compute(n - 1) + compute(n - 2);
        }
        return result;
    }
}
```

Representative values that may have found with test strategies (e.g., category partition)

```java
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 },
{ 5, 5 }, { 6, 8 } });
    }
    private int fInput;
    private int fExpected;
    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }
    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
public class Fibonacci {
    public static int compute(int n) {
      int result = 0;
        if (n <= 1) {
           result = n;
        } else {
           result = compute(n - 1) + compute(n - 2);
        }
        return result;
    }
}
```

Representative values that may have found with test strategies (e.g., category partition)

```java
@RunWith(Parameterized.class)
public class FibonacciTest {
    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][] {{ 0, 0 }, { 1, 1 }, { 2, 1 }, { 3, 2 }, { 4, 3 },
{ 5, 5 }, { 6, 8 } });
    }
    private int fInput;
    private int fExpected;
    public FibonacciTest(int input, int expected) {
        fInput= input;
        fExpected= expected;
    }
    @Test
    public void test() {
        assertEquals(fExpected, Fibonacci.compute(fInput));
    }
}
public class Fibonacci {
    public static int compute(int n) {
      int result = 0;
        if (n <= 1) {
           result = n;
        } else {
           result = compute(n - 1) + compute(n - 2);
        }
        return result;
    }
}
```

Representative values that may have found with test strategies (e.g., category partition)

Identify Individual test cases

- To easily identify the individual test cases in a Parameterized test, use a name in the @Parameters annotation.
- This name is allowed to contain placeholders that are replaced at runtime:
  - {index}: the current parameter index
  - {0}, {1}, …: the first, second, and so on, parameter value

Barbara Russo

SwSE - Software and Systems Engineering research group

Dynamic testing

- **Dynamic testing** concerns testing the **operations (behaviour)** of a program

- Unit tests, integration tests, system tests and acceptance tests and regression tests utilize dynamic testing

Dynamic testing

- Tests with @Test annotation are static tests as they are fully specified at compile-time
- A dynamic test is a test generated during run-time

- In JUnit 5 there are new annotations that support it
-

Dynamic testing

- A dynamic test is generated by a factory annotation: **@TestFactory**
- @TestFactory methods *must not be private or static* and may optionally declare parameters

JUnit 5

Example

```
@TestFactory
public DynamicTest createTest(){

   return dynamicTest("1st dynamic test", () -> assertTrue(isPalindrome("madam")));

}
```

test case

Factory method

JUnit 5

DynamicTest class

- DynamicTest is a test case generated at runtime
- It is determined of a *display name and an Executable* that are passed to its method dynamicTest

Example

```
@TestFactory
public DynamicTest createTest(){

return dynamicTest("1st dynamic test", () -> assertTrue(isPalindrome("madam")));

}
```

display name

executable

JUnit 5

Key methods of DynamicTest class

public static DynamicTest

```
dynamicTest(String displayName, Executable executable)
```

- Factory method creates a new DynamicTest instance with the given display name and executable code block

```
dynamicTest("testName", () -> assertTrue(isPalindrome(“madam")))
```

```
stream(Iterator<T> inputGenerator, Function<? super
T,String> displayNameGenerator, ThrowingConsumer<?
super T> testExecutor)
```

- Factory method to generate **a stream of dynamic tests**
- *inputGenerator* generates input values. A DynamicTest is added to the resulting stream for each dynamically generated input value, using the *displayNameGenerator* and *testExecutor*

- `inputGenerator` - an Iterator that serves as a dynamic input generator
- `displayNameGenerator` - a function that generates a display name based on an input value
- `testExecutor` - a consumer that executes a test based on an input value

```java
@TestFactory
Stream<DynamicTest> generateRandomNumberOfTests() {

    Iterator<Integer> inputGenerator = new Iterator<Integer>() {

        Random random = new Random();
        int current;

        @Override
        public boolean hasNext() {
            current = random.nextInt(100);
            return current % 7 != 0;
        }

        @Override
        public Integer next() {
            return current;
        }
    };

    // Generates display names like: input:5, input:37, input:85, etc.
    Function<Integer, String> displayNameGenerator = (input) -> "input:" + input;

    // Executes tests based on the current input value.
    ThrowingConsumer<Integer> testExecutor = (input) -> assertTrue(input % 7 != 0);

    // Returns a stream of dynamic tests.
    return DynamicTest.stream(inputGenerator, displayNameGenerator, testExecutor);
}
```

Generates random positive integers between 0 and 100 until a number evenly divisible by 7 is encountered.

Functional Interfaces. This one is throwing a Throwable exception

FunctionalInterfaces w. Dynamic Test

```
stream(
Iterator<T> inputGenerator,
Function<? super T,String> displayNameGenerator,
ThrowingConsumer<? super T> testExecutor)

dynamicTest(String displayName, Executable executable)
```

- The implementations of DynamicTest can be provided as lambda expressions or method references for the Functional Interfaces

Function faces w. Dynamic Test

Regular Interfaces

```
stream(
Iterator<T> inputGenerator,
Function<? super T,String> displayNameGenerator,
ThrowingConsumer<? super T> testExecutor)

dynamicTest(String displayName, Executable executable)
```

- The implementations of DynamicTest can be provided as lambda expressions or method references for the Functional Interfaces

Function… fac… Dynamic Test

Regular Interfaces

Functional Interfaces

```
stream(
Iterator<T> inputGenerator,
Function<? super T,String> displayNameGenerator,
ThrowingConsumer<? super T> testExecutor)

dynamicTest(String displayName, Executable executable)
```

- The implementations of DynamicTest can be provided as lambda expressions or method references for the Functional Interfaces

Function… fac… Dynamic Test

Regular Interfaces

Functional Interfaces

Functional Interfaces of JUnit5

```
stream(
Iterator<T> inputGenerator,
Function<? super T,String> displayNameGenerator,
ThrowingConsumer<? super T> testExecutor)

dynamicTest(String displayName, Executable executable)
```

- The implementations of DynamicTest can be provided as lambda expressions or method references for the Functional Interfaces

unibz Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Functional Interface

- A functional interface has **only one abstract method** but it can have *multiple default methods*
- **@FunctionalInterface** annotation is used to ensure **at compile time** that an interface cannot have more than one abstract method
  - The use of this annotation is optional

```java
//some imports

@RunWith(JUnitPlatform.class)
public class TranslatorEngineTest {
  private TranslatorEngine translatorEngine;
  @BeforeEach
  public void setUp() {
    translatorEngine = new TranslatorEngine();
  }

  @Test
  public void testTranslateHello() {
    assertEquals("Bonjour",
translatorEngine.translate("Hello"));
  }
  @Test
  public void testTranslateYes() {
    assertEquals("Oui",
translatorEngine.translate("Yes"));
  }
@Test
  public void testTranslateYes() {
    assertEquals("No",
translatorEngine.translate("No"));
  }
}
```

```
//some imports

@RunWith(JUnitPlatform.class)
public class TranslatorEngineTest {
  private TranslatorEngine translatorEngine;
  @BeforeEach
  public void setUp() {
    translatorEngine = new TranslatorEngine();
  }

  @Test
  public void testTranslateHello() {
    assertEquals("Bonjour",
translatorEngine.translate("Hello"));
  }
  @Test
  public void testTranslateYes() {
    assertEquals("Oui",
translatorEngine.translate("Yes"));
  }
@Test
  public void testTranslateYes() {
    assertEquals("Non",
translatorEngine.translate("No"));
  }
}
```

```java
//some imports

@RunWith(JUnitPlatform.class)
public class TranslatorEngineTest {
  private TranslatorEngine translatorEngine;
  @BeforeEach
  public void setUp() {
    translatorEngine = new TranslatorEngine();
  }

  @Test
  public void testTranslateHello() {
    assertEquals("Bonjour",
translatorEngine.translate("Hello"));
  }
  @Test
  public void testTranslateYes() {
    assertEquals("Oui",
translatorEngine.translate("Yes"));
  }
@Test
  public void testTranslateYes() {
    assertEquals("Non",
translatorEngine.translate("No"));
  }
}
```

```java
@RunWith(Parameterized.class)
public static class Example{
	@Parameters(name = "{index}: translation({0})={1}")
	public static Object[][] data(){
		return new Object[][]{{"Hello", "Bonjour"},{"Yes","Oui"},
{"No","Non"}};
	}
	private String input;
	private String output;
	public Example(String input, String output){
		this.input=input;
		this.output=output;
	}
	@org.junit.Test
	public void test(){
		TranslatorEngine translatorEngine = new TranslatorEngine();
		assertEquals(output, translatorEngine.translate(input));
	}
}
```

```java
//some imports

@RunWith(JUnitPlatform.class)
public class TranslatorEngineTest {
  private TranslatorEngine translatorEngine;
  @BeforeEach
  public void setUp() {
    translatorEngine = new TranslatorEngine();
  }

  @Test
  public void testTranslateHello() {
    assertEquals("Bonjour",
translatorEngine.translate("Hello"));
  }
  @Test
  public void testTranslateYes() {
    assertEquals("Oui",
translatorEngine.translate("Yes"));
  }
@Test
  public void testTranslateYes() {
    assertEquals("No",
translatorEngine.translate("No"));
  }
}
```

```java
//some imports and class declaration

@TestFactory
public Collection<DynamicTest> translateDynamicTests() {
      List<String> inPhrases = new
ArrayList<>(Arrays.asList("Hello", "Yes", “No"));
      List<String> outPhrases = new
ArrayList<>(Arrays.asList("Bonjour", "Oui", "Non"));
      Collection<DynamicTest> dynamicTests = new ArrayList<>();
      TranslatorEngine translatorEngine = new TranslatorEngine();

      for (int i = 0; i < inPhrases.size(); i++) {
            String phr = inPhrases.get(i);
            String outPhr = outPhrases.get(i);
            // create a test execution
            Executable exec = () -> assertEquals(outPhr,
translatorEngine.translate(phr));
            // create a test display name
            String testName = " Test translate " + phr;
            // create dynamic test
            DynamicTest dTest = DynamicTest.dynamicTest(testName,
exec);
            // add the dynamic test to collection
            dynamicTests.add(dTest);
      }
      return dynamicTests;
}
```

```java
//some imports

@RunWith(JUnitPlatform.class)
public class TranslatorEngineTest {
  private TranslatorEngine translatorEngine;
  @BeforeEach
  public void setUp() {
    translatorEngine = new TranslatorEngine();
  }

  @Test
  public void testTranslateHello() {
    assertEquals("Bonjour",
translatorEngine.translate("Hello"));
  }
  @Test
  public void testTranslateYes() {
    assertEquals("Oui",
translatorEngine.translate("Yes"));
  }
@Test
  public void testTranslateYes() {
    assertEquals("No",
translatorEngine.translate("No"));
  }
}
```

```java
//some imports and class declaration

@TestFactory
public Collection<DynamicTest> translateDynamicTests() {
      List<String> inPhrases = new
ArrayList<>(Arrays.asList("Hello", "Yes", "No"));
      List<String> outPhrases = new
ArrayList<>(Arrays.asList("Bonjour", "Oui", "Non"));
      Collection<DynamicTest> dynamicTests = new ArrayList<>();
      TranslatorEngine translatorEngine = new TranslatorEngine();

      for (int i = 0; i < inPhrases.size(); i++) {
            String phr = inPhrases.get(i);
            String outPhr = outPhrases.get(i);
            // create a test execution
            Executable exec = () -> assertEquals(outPhr,
translatorEngine.translate(phr));
            // create a test display name
            String testName = " Test translate " + phr;
            // create dynamic test
            DynamicTest dTest = DynamicTest.dynamicTest(testName,
exec);
            // add the dynamic test to collection
            dynamicTests.add(dTest);
      }
      return dynamicTests;
}
```

We can also generate such I/O