**Parameterized Tests**

**JUnit Parameterized Tests**

While testing, it’s common to execute a series of tests which differ only by input values and expected results. As an example, if you are testing a method that validates a emailID, you should test it with different email ID formats to check whether the validations are getting correctly done. But testing each email ID format separately, will result in duplicate or boilerplate code. It is better to abstract the email ID test into a single test method and provide it a list of all input values and expected results. JUnit supports this functionality through parameterized tests.

To see how parameterized test works, we’ll start by modifying our Robot constructor and adding a new method to our Robot class.

**public class** Robot {  
  
 **private** String **name**;  
 **private boolean working** = **false**;  
 **private int age**;  
  
  
 **public** Robot(String name, **int** age) {  
 setName(name);  
 setAge(age);  
 }  
 **private void** setName (String name){  
 **if** (name.isEmpty()) {  
 **throw new** IllegalArgumentException();  
 }  
 **this**.**name** = name;  
 }  
  
  
 **private void** setAge(**int** age) {  
 **if** (age <= 30) {  
 **this**.**age** = age;  
 } **else** {  
 **throw new** IllegalArgumentException();  
  
 }  
  
 }

then add the following method.

**public** String checkage() {  
 **if** (age <= 20){  
 **return "Age ok"**;  
 }  
 **else** {  
 **return “too old"**;  
 }

We now need to create a new junit test class. Select the robot class name and use alt- enter. Select the test class option. Call this file RobotParamTests.

JUnit runs a parameterized test with a special runner, Parameterized and we need to declare it with the @RuntWith annotation.

In a parameterized test class, we declare instance variables corresponding to the number of inputs to the test and the output. As the checkage() method under test takes a single int parameter and returns a String, we declare two corresponding variables.

For a parameterized test, we need to provide a constructor, which will initialize the variables.

**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.junit.runners.Parameterized;  
  
**import** java.util.Arrays;  
**import** java.util.Collection;  
  
**import static** org.junit.Assert.\*;  
  
@RunWith(value = Parameterized.**class**)

**public class** RobotParamTests {  
  
 **private** String **expected**;  
 **private int age**;

**public** RobotParamTests( String expected, **int** age){  
 **this**.**age** = age;  
 **this**.**expected** = expected;  
}

We also need to provide a public static method annotated with @Parameters annotation. This method will be used by the test runner to feed data into our tests.

@Parameterized.Parameters (name= **"{index}: checkage({1})={0}"**)  
**public static** Collection<Object[]> getTestParameters() {  
 **return** Arrays.*asList*(**new** Object[][] {  
 {**"Age ok"**, 10},  
 {**"Age ok"** , 20},  
 {**"Too old enough"**, 25},  
 });  
}

The @Parameters annotated method above returns a collection of test data elements (which in turn are stored in an array). Test data elements are the different variations of the data, including the input as well as expected output needed by the test. The number of test data elements in each array must be the same with the number of parameters we declared in the constructor.

When the test runs, the runner instantiates the test class once for each set of parameters, passing the parameters to the constructor that we wrote. The constructor then initializes the instance variables we declared.

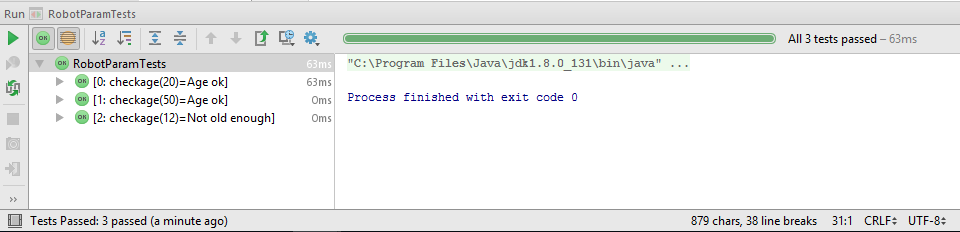
Notice the optional name attribute we wrote in the @Parameters annotation to identify the parameters being used in the test run. This attribute contains placeholders that are replaced at run time.

* **{index}**: The current parameter index, starting from 0.
* **{0}, {1}, …**: The first, second, and so on, parameter value.

Finally, we write the test method annotated with @Test. The complete code of the parameterized test is at the end of this document. The output on running the parameterized test in IntelliJ is this.

@Test  
  
**public void** testcheckAll() {  
  
 Robot buddy = **new** Robot(**"buddy"**, **this**.age);  
  
 assertEquals(expected, buddy.checkage());  
  
  
  
}

The full code is at the end of the handout.



**Exercise**

Add the following method to the Robot class. Then create a new test class that uses the paramterised runner to test the method.

**public double** checkCost() {  
 **if** (**age** <= 5) {  
 **return** 10000.00;  
 } **else if** (**age** <= 10) {  
 **return** 7500.00;  
 } **else return** 5000.00;  
}

**import** org.junit.Test;  
**import** org.junit.runner.RunWith;  
**import** org.junit.runners.Parameterized;  
  
**import** java.util.Arrays;  
**import** java.util.Collection;  
  
**import static** org.junit.Assert.\*;  
  
@RunWith(value = Parameterized.**class**)  
**public class** RobotParamTest1 {  
  
 **private** String **expected**;  
 **private int age**;  
  
  
 **public** RobotParamTest1( String expected, **int** age){  
 **this**.**age** = age;  
 **this**.**expected** = expected;  
 }  
  
  
 @Parameterized.Parameters (name= **"{index}: checkage({1})={0}"**)  
 **public static** Collection<Object[]> getTestParameters() {  
 **return** Arrays.*asList*(**new** Object[][] {  
 {**"Age ok"**, 10},  
 {**"Age ok"** , 20},  
 {**"Too old"**, 25},  
 });  
 }  
  
  
 @Test  
 **public void** testcheckageAll() {  
 Robot buddy = **new** Robot(**"buddy"**, **age**);  
 *assertEquals*(**expected**, buddy.checkage() );  
  
 }  
}