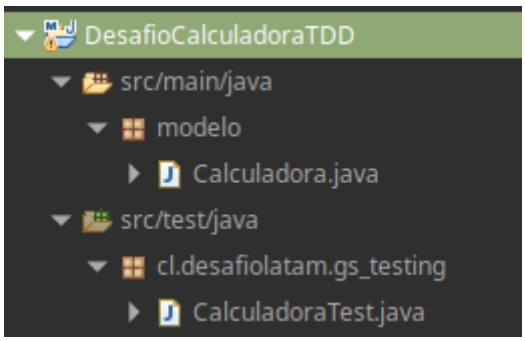
Solución Desafío - Calculadora (Parte II)

Paso 1: Se crea un nuevo proyecto con Maven. El proyecto generado contiene esta estructura de carpetas.



Se pueden borrar las clases Java generadas en main y en test (App.java, AppTest.java) para crear las propias.

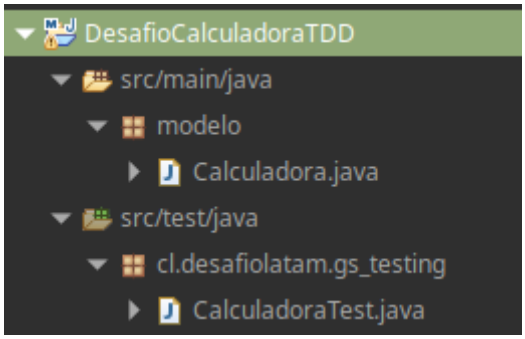
Paso 2: Agregar dependencias de JUnit 4. Se usa la versión más reciente de JUnit que consiste en una librería adicional la cual Maven se encarga de gestionar y agregar al proyecto. Corresponde a esta dependencia:

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>
```

Se debe ir al archivo `pom.xml` en la raíz del proyecto, se borra la dependencia de JUnit que viene por defecto y se agrega la nueva dentro del tag dependencies. El archivo `pom.xml` sería el siguiente:

```
<?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/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>cl.desafiolatam</groupId>
<artifactId>calculadora-tdd</artifactId>
<version>1.0-SNAPSHOT</version>
<name>calculadora-tdd</name>

<!-- FIXME change it to the project's website -->
<url>http://www.example.com</url>
<properties>
<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<maven.compiler.source>1.7</maven.compiler.source>
<maven.compiler.target>1.7</maven.compiler.target>
</properties>

<dependencies>
 <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
   </dependency>
</dependencies>
<!--resto del archivo-->
</project>
```

Paso 3: Fase Roja. Para orientar el desarrollo a TDD, se debe escribir las pruebas en primer lugar sin pensar en implementaciones. Creamos la clase `CalculadoraTest` dentro de la carpeta src/test del proyecto, en la siguiente ruta:



Esta será la clase que contendrá las pruebas. Inicialmente se instancia la clase `Calculadora` inexistente, pero se deberá crear solo para que la prueba compile.

```
import org.junit.jupiter.api.Test;
public class CalculadoraTest {
 private Calculadora calculadora = new Calculadora();
}
```

Paso 4: La clase `CalculadoraTest` debe tener sus métodos para operar, prueba para método sumar:

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculadoraTest {
  private Calculadora calculadora = new Calculadora();
 @Test
    public void testSumar() {
       Integer resultado = calculadora.sumar(1, 1);
       assertEquals(Integer.valueOf(2), resultado);
    }

}
```

Paso 5: La clase `CalculadoraTest` debe tener sus métodos para operar, prueba para método restar:

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculadoraTest {
  private Calculadora calculadora = new Calculadora();
 @Test
    public void testRestar() {
       Integer resultado = calculadora.restar(111, 11);

       assertEquals(Integer.valueOf(100), resultado);
    }}
```

Paso 6: La clase `CalculadoraTest` debe tener sus métodos para operar, prueba para método multiplicar:

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculadoraTest {
  private Calculadora calculadora = new Calculadora();
 @Test
    public void testMultiplicar() {
       Integer resultado = calculadora.multiplicar(4, 4);
       assertEquals(Integer.valueOf(16), resultado);
    }
}
```

Paso 7: La clase CalculadoraTest debe tener sus métodos para operar, prueba para método dividir:

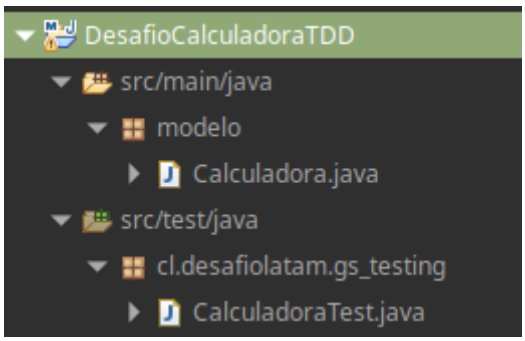
```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class CalculadoraTest {
  private Calculadora calculadora = new Calculadora();
 @Test
    public void testDividir() {
       Integer resultado = calculadora.dividir(9, 3);
       assertEquals(Integer.valueOf(3), resultado);
    }
}
```

La clase `CalculadoraTest` finalizada debe lucir así:

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

public class CalculadoraTest {
  private Calculadora calculadora = new Calculadora();

 @Test
    public void testSumar() {
       Integer resultado = calculadora.sumar(1, 1);
       assertEquals(Integer.valueOf(2), resultado);
    }

    @Test
    public void testRestar() {
       Integer resultado = calculadora.restar(111, 11);

       assertEquals(Integer.valueOf(100), resultado);
    }

    @Test
    public void testMultiplicar() {
       Integer resultado = calculadora.multiplicar(4, 4);
       assertEquals(Integer.valueOf(16), resultado);
    }
```

```
    @Test
    public void testDividir() {
       Integer resultado = calculadora.dividir(9, 3);
       assertEquals(Integer.valueOf(3), resultado);
    }}
```

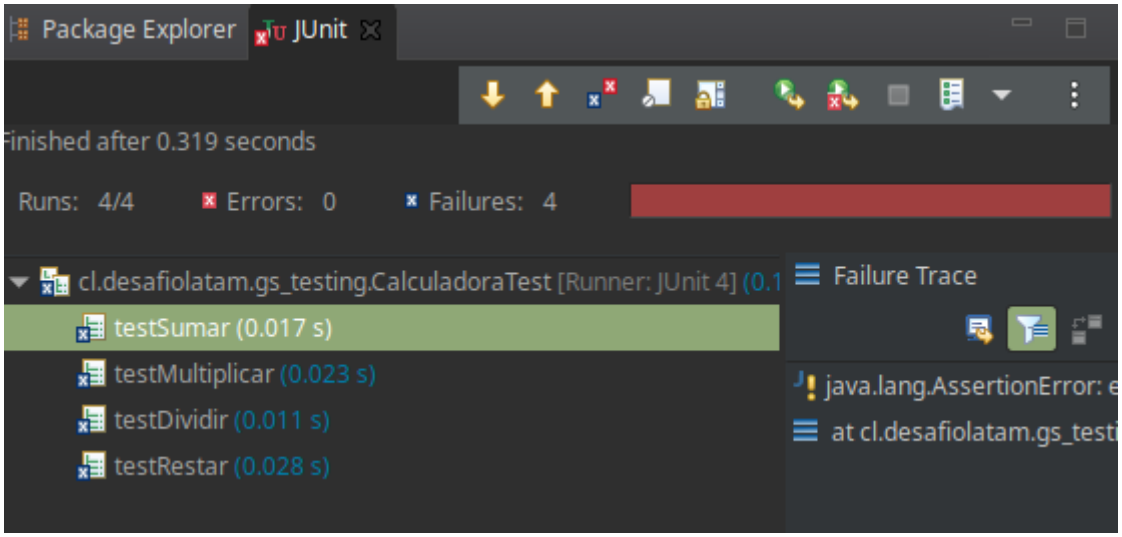
Paso 8: Crear clase `Calculadora` dentro de la carpeta *src/main* del proyecto. La clase de prueba luce bien, sin embargo, la clase Calculadora no existe y sus métodos tampoco. Para eliminar los errores, se debe crear la clase con sus métodos pero sin su implementación dentro.



La implementación mínima de los métodos de la clase Calculadora para compilar:

```
public class Calculadora {
  public Integer sumar(int i, int i1) {
  return 0;
}
  public Integer restar(int i, int i1) {
  return 0;
}
  public Integer multiplicar(int i, int i1) {
  return 0;
}
  public Integer dividir(int i, int i1) {
  return 0;
}
}
```

Paso 9: Ejecutar mvn test la salida es:

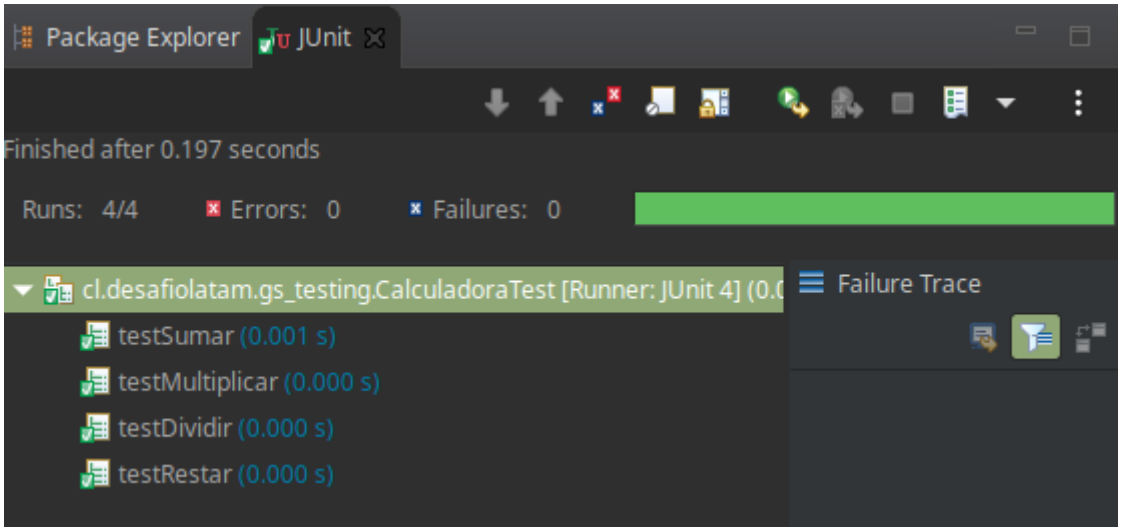



```
[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR] CalculadoraTest.testDividir:40 expected: <3> but was: <0> [ERROR]
CalculadoraTest.testMultiplicar:33 expected: <16> but was: <0> [ERROR]
CalculadoraTest.testRestar:26 expected: <100> but was: <0> [ERROR]
CalculadoraTest.testSumar:19 expected: <2> but was: <0> [INFO]
[ERROR] Tests run: 4, Failures: 4, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------ -
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------ -
[INFO] Total time: 2.792 s
[INFO] Finished at: 2019-07-09T12:31:48-04:00
[INFO] ------------------------------------------------------------------------ -
```

Paso 10: Escribir código a probar - Fase Verde. Lo siguiente es crear solo el código suficiente para pasar las pruebas de CalculadoraTest. En esta etapa solo se tiene una tarea, escribir una solución sencilla que haga pasar la prueba.

```
package cl.desafiolatam;
public class Calculadora {
  public Integer sumar(int a, int b) {
  return a+b;
}
  public Integer restar(int a, int b) {
  return a-b;
}
  public Integer multiplicar(int a, int b) {
  return a*b;
}
  public Integer dividir(int a, int b) {
  return a/b;
 }
}
```

Paso 11: Ejecutar pruebas. Ejecutar el comando mvn test para que la salida sea:

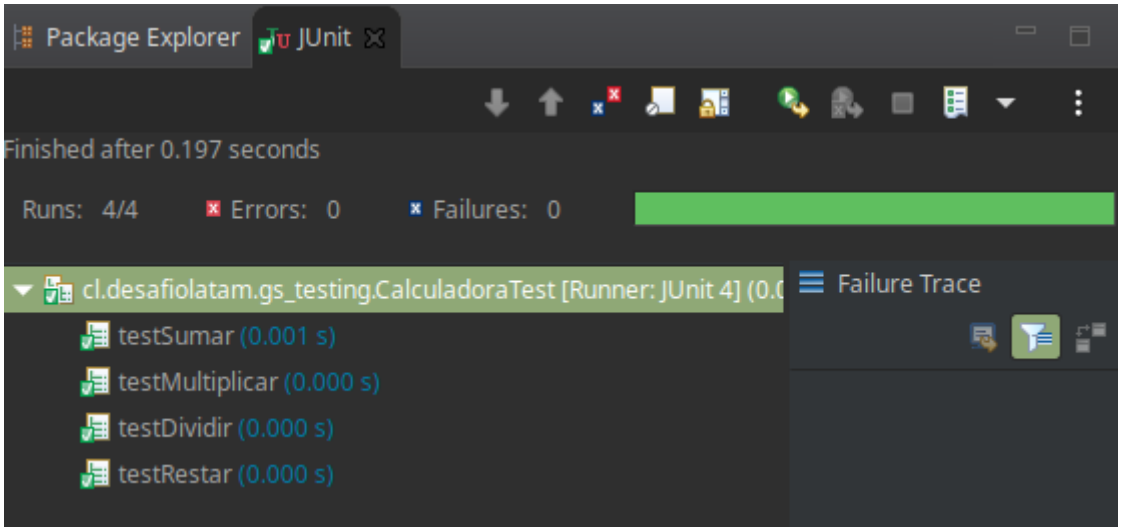



```
[INFO] ------------------------------------------------------- [INFO] T E S T S
[INFO] ------------------------------------------------------- [INFO] Running
cl.desafiolatam.CalculadoraTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.025 s -
in cl.desafiolatam.CalculadoraTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------ -
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------ -
[INFO] Total time: 2.960 s
[INFO] Finished at: 2019-07-09T12:22:28-04:00
[INFO] ------------------------------------------------------------------------ -
```

Paso 12: En la fase de refactorización, se le permite cambiar el código, manteniendo todas las pruebas en verde para que sea mejor. Lo que significa mejor depende del desarrollador. Se agrega validaciones para evitar errores en caso de pasar valores nulos a la calculadora, la clase queda así:

```
public class Calculadora {
  public Integer sumar(Integer a, Integer b) {
    if (a!=null && b!=null) {
      return a + b;
    } else {
        return 0;
    }
  }

  public Integer restar(Integer a, Integer b) {
    if (a!=null && b!=null) {
      return a - b;
    } else {
      return 0;
    }
  }

  public Integer multiplicar(Integer a, Integer b) {
    if (a!=null && b!=null) {
      return a * b;
    } else {
      return 0;
    }
  }

  public Integer dividir(Integer a, Integer b) {
    if (a!=null && b!=null) {
      return a / b;
    } else {
      return 0;
    }
  }
}
```

Paso 13: Ejecutar mvn test la salida debe ser:




```
[INFO] ------------------------------------------------------- [INFO] T E S T S
[INFO] ------------------------------------------------------- [INFO] Running
cl.desafiolatam.CalculadoraTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.045 s -
in cl.desafiolatam.CalculadoraTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------ -
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------ -
[INFO] Total time: 3.202 s
[INFO] Finished at: 2019-07-10T00:46:20-04:00
[INFO] ------------------------------------------------------------------------ -
```