# CSE12 - Lecture 2

Wednesday, April 5, 2023 8:00 AM

PA1 released → due Tuesday

Test Driven Development - TDD
Write the tests first!!

1) Write method header → public static int abs(int value) { return 0; }
2) Write test cases
3) Write code
4) Fix code until all tests pass

default value

What is a test? What are we checking in the code?

Check for errors
Check for correctness > confidence code works

JUnit is a simple framework to write repeatable tests.

```
Write a test for the absolute value method in the
Math library: int Math.abs(int value);

assertEquals( ____3_____ , Math.abs(-3));

assertNotEquals( ____-3_____ , Math.abs(-3) );

fail( “some error message” );
```

Fill in the blanks for for assertEquals and assertNotEquals.

assertEquals(expected, actual)
==
.equals()

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

public class TestJUnit {

      @Test
      public void testQuestion1() {

      }

}

Command line:
javac -cp hamcrest-core-1.3.jar;junit-4.12.jar;. TestJava.java
java -cp hamcrest-core-1.3.jar;junit-4.12.jar;. org.junit.runner.JUnitCore TestJava

Or use Eclipse (IDE)
```

class path

What is @Test in the above code? What does it do?

jUnit annotation → tells jUnit → this is a test → run it

Name: _____ PID: _____ Code: 9314

sumNumbers

```
Given a string, return the sum of the numbers appearing in the string, ignoring all other
characters. A number is a series of 1 or more digit chars in a row. (Note:
Character.isDigit(char) tests if a char is one of the chars '0', '1', .. '9'.
Integer.parseInt(string) converts a string to an int.)

sumNumbers("abc123xyz") → 123
sumNumbers("aa11b33") → 44
sumNumbers("7 11") → 18
```

\" escape sequence

What test cases should we write to confirm that our implementation works?

int sumNumbers(String s) { return 0; }

| input | expected |
|---|---|
| null | 0 |
| "" | 0 |
| "abc" | 0 |
| "5" | 5 |
| "4" | 4 |
| "1a2b3d" | 7 |

edge cases

"0b-1" → 1
"3.14" → 17
"ab\"cd" → 0

"0a1" → 1
"10a1" → 11

sumNumbers

```
Return an array that contains the exact same numbers as the given array, but rearranged so
that all the even numbers come before all the odd numbers. Other than that, the numbers can
be in any order. You may modify and return the given array, or make a new array.


evenOdd([1, 0, 1, 0, 0, 1, 1]) → [0, 0, 0, 1, 1, 1, 1]
evenOdd([3, 3, 2]) → [2, 3, 3]
evenOdd([2, 2, 2]) → [2, 2, 2]
```

What test cases should we write to confirm that our implementation works?