package nz.ac.aut.hsv5433.dsa2017.lab01.tests;

import static org.junit.Assert.\*;

import java.util.Random;

import org.junit.Test;

import nz.ac.aut.hsv5433.dsa2017.lab01.IntPair;

public class IntPairTest {

private Random rng = new Random(42);

// to check if the left matches 10 and right matches -10 without nulls.

@Test

// Creates pair of numbers, 10, -10, stores in variable p. asserts that variable p

// is not NULL, and then asserts that 10 is equal to the left number of pair p (10)

// and -10 is equal to the right side of pair p(-10)

public void testMakeBasic () {

IntPair p = IntPair.make(10, -10);

assertNotNull(p);

assertEquals(10, p.left());

assertEquals(-10, p.right());

}

//

@Test

// an attempt to create an IntPair will fail if either of the values are nulls.

public void testNewLeftRejectsNull () {

try {

IntPair.newLeft(null, 10);

fail("newLeft should reject nulls");

} catch (NullPointerException e) {}

}

@Test

public void testNewRightRejectsNull () {

try {

IntPair.newRight(null, -10);

fail("newRight should reject nulls");

} catch (NullPointerException e) {}

}

// checks to see if the left is 20 the right is -10 and the new left is 10 and the right is

// -10 with out throwing is fail

@Test

public void testNewLeftBehaviour () {

IntPair old = IntPair.make(10, -10);

IntPair p = IntPair.newLeft(old, 20);

assertNotNull(p);

assertEquals(20, p.left());

assertEquals(-10, p.right());

assertEquals(10, old.left());

assertEquals(-10, old.right());

}

@Test

public void testNewRightBehaviour () {

IntPair old = IntPair.make(10, -10);

IntPair p = IntPair.newRight(old, 20);

assertNotNull(p);

assertEquals(10, p.left());

assertEquals(20, p.right());

assertEquals(10, old.left());

assertEquals(-10, old.right());

}

@Test

public void testEqualityBasic () {

for (int i = 0; i < 1000000; i++) {

int[] bases = new int[4];

for (int j = 0; j < bases.length; j++) {

bases[j] = rand();

}

IntPair p1 = IntPair.make(bases[0], bases[1]); // 14, 17

IntPair p2 = IntPair.make(bases[2], bases[3]); // 15, 41

boolean matchLeft = bases[0] == bases[2]; // false

boolean matchRight = bases[1] == bases[3]; // false

assertEquals(matchLeft && matchRight, p1.equals(p2)); // 14 == 15 && 17 == 41,14,17 == 15,41

}

}

@Test

public void testEqualitySelf () {

for (int i = 0; i < 1000000; i++) {

IntPair p = IntPair.make(rand(), rand());

assertTrue(p.equals(p));

}

}

@Test

public void testEqualityNull () {

for (int i = 0; i < 1000000; i++) {

IntPair p = IntPair.make(rand(), rand());

assertFalse(p.equals(null));

}

}

@Test

public void testEqualityCommutes () {

for (int i = 0; i < 1000000; i++) {

IntPair p1 = IntPair.make(rand(), rand());

IntPair p2 = IntPair.make(rand(), rand());

assertEquals(p1.equals(p2), p2.equals(p1));

}

}

@Test

public void testEqualityAssociates () {

for (int i = 0; i < 1000000; i++) {

int left = rand();

int right = rand();

IntPair[] ps = new IntPair[3];

for (int j = 0; j < ps.length; j++) {

ps[j] = IntPair.make(left, right);

}

boolean eq = ps[0].equals(ps[1]) && ps[1].equals(ps[2]);

assertEquals(eq, ps[0].equals(ps[2]));

}

}

@Test

public void testHashCode () {

for (int i = 0; i < 1000000; i++) {

int x = rand();

int y = rand();

IntPair p1 = IntPair.make(x, y);

IntPair p2 = IntPair.make(x, y);

assertEquals(p1.equals(p2), p1.hashCode() == p2.hashCode());

}

}

// Private

private int rand () { return rng.nextInt(20) - 10; }

}