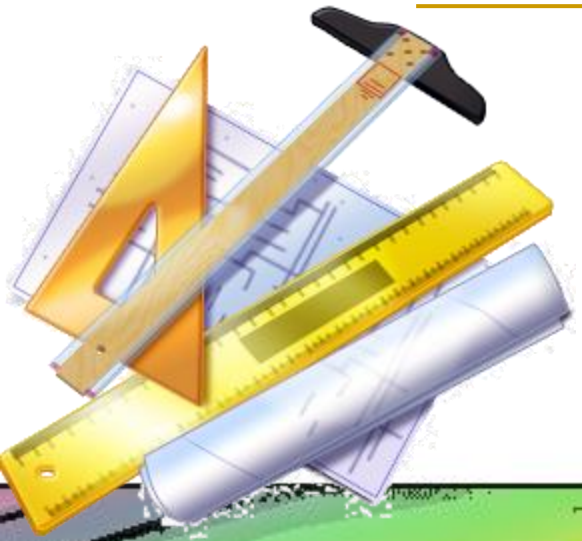
การทดสอบโปรแกรมระดับหน่วยย่อย (Unit Testing) โดยใช้ JUnit



Lecture 10

ทรงศักดิ์ รองวิริยะพานิช

เยาวดี เต็มธนาภัทร์

วัตถุประสงค์ของการเรียนวันนี้

- เพื่อแนะนำเกี่ยวกับวิธีการทดสอบความถูกต้อง
- เพื่อเรียนรู้การทดสอบโปรแกรมหน่วยย่อย (Unit Testing) โดยใช้ JUnit

การทดสอบโดยทั่วไป

- **การทดสอบ (Testing)**
  - เพื่อ (พยายาม) แสดงให้เห็นถึงความถูกต้องของซอฟต์แวร์
  - โดยการเปรียบเทียบผลลัพธ์ที่ได้จากการรันโปรแกรม (Actual output) ด้วยชุดข้อมูล Inputs เทียบกับผลลัพธ์ที่ควรจะเป็น (Expected output)
  - และพยายามคิดหากรณีทดสอบ (Test cases) ที่เป็นชุดข้อมูล Inputs ที่เมื่อรันแล้วทำให้พบจุดข้อผิดพลาดของโค้ดที่พัฒนา
- แนวคิดการพัฒนาโดยวิธี Test-Driven Development (TDD)
  - โค้ดที่พัฒนาควบคู่กับการทดสอบที่ดีเป็นระยะ ๆ จะช่วยให้ได้ซอฟต์แวร์ที่มีคุณภาพ
  - ทดสอบตั้งแต่ในระดับย่อยที่สุด หรือ unit testing
    - เพิ่มความมั่นใจในช่วงพัฒนา โค้ดที่ผิดพลาดถูกตรวจจับได้ตั้งแต่แรก ๆ
    - นักพัฒนา (ผู้เขียนโค้ด) เป็นผู้ทดสอบ

การทดสอบโดยทั่วไป

- **ช่วง Phases ต่าง ๆ ของการทดสอบ**
  - Unit testing
    - ทดสอบแต่ละโมดูล (unit หรือ component) อย่างเป็นอิสระต่อกัน ทดสอบว่าแต่ละโมดูลทำงานได้ตามที่กำหนด
    - ส่วนใหญ่ดำเนินการโดยผู้พัฒนาโมดูลนั้น พยายามทำตัวเป็นนักสืบ (Inspection)
  - Integration และ System testing
    - ทดสอบการเชื่อมโยงระหว่าง 2 ระบบ (Integration Testing)
    - ทดสอบความถูกต้องของการทำงานทั้งระบบโดยรวม (System Testing)
    - ส่วนใหญ่ดำเนินการโดยทีมทดสอบหรือทีมประกันคุณภาพ (QA)
  - Acceptance testing
    - เพื่อทดสอบความถูกต้องของฟังก์ชั่นการทำงานของโปรแกรมทั้งระบบว่าทำได้ถูกต้อง ครบถ้วนตามความต้องการของผู้ใช้ (เพื่อให้ลูกค้ายอมรับว่าโปรแกรมทำงานได้ถูกต้อง)
  - Regression testing
    - ทดสอบโมดูล (unit หรือ component) ที่มีการแก้ไข ทดสอบซ้ำให้มั่นใจว่า โปรแกรมที่แก้ไขสามารถทำงานได้ถูกต้องเท่ากับก่อนการแก้ไข ไม่มีการถดถอยของโปรแกรมคือทำงานที่เคยทำถูกต้องได้แม้จะมีการแก้โค้ดโปรแกรม

รูปภาพแสดงการทำ Regression Testing

Class 1 ⇔ {Test case1, …., Test case 10}
หมายถึง การทดสอบ 10 กรณี เพื่อทดสอบความถูกต้องของคลาส Class1

Modified Class 1 ⇔ {Test case1, …., Test case 10}
U {Test case11,…, Test case 15}
หมายถึง เมื่อมีการแก้ไขโค้ดของคลาส Class1 โปรแกรมเมอร์ต้องทดสอบ 5 กรณี เพิ่มจาก10 กรณีเพื่อทดสอบความถูกต้องของคลาส Class1
ว่าไม่มีการถดถอย ทำงานได้ถูกต้องไม่น้อยกว่าคลาสก่อนแก้ไข

Unit Testing

- **การทดสอบ (Testing)**
  - กระบวนการในการตรวจสอบความถูกต้อง (Verification) ของการทำงานของโปรแกรมกับชุดของ inputs
- **Unit**: โมดูลหรือเซตของโมดูลเล็ก ๆ
  - ใน OO, unit หมายถึงคลาสหรือ interface หรือเซตของคลาส/interface เช่น
    - Unit 1 หน่วย ประกอบด้วย interface และคลาส 3 ตัวที่ implement interface นั้น
    - หรือ Unit 1 หน่วย ประกอบด้วย public class พร้อมกับ helper classes ที่เกี่ยวข้องกับคลาสนั้น
- **Unit testing**
  - การทดสอบแต่ละ unit นั้น (verification of single modules)

ทำไมทำ unit testing

โค้ดที่ไม่ถูกทดสอบ ไม่สามารถรู้ได้ว่าถูกต้องหรือไม่

- ใช้แนวทาง Divide-and-conquer
  - แบ่งระบบเป็น units ดีบักแต่ละ unit
    - สามารถค้นหาบักได้ง่ายขึ้น โดยบีบขอบเขตที่ต้องค้นให้แคบลง
    - ไม่ต้องวิ่งไล่ตามบักข้ามไปยัง units อื่น ๆ
    - สนับสนุนการทดสอบแบบ regression ทำให้แก้ไขโค้ดได้อย่างมั่นใจ
- **แนวคิด**:
  - เขียนกรณีทดสอบ (test cases) สำหรับคลาส โดยพยายามให้แต่ละกรณีทดสอบเป็นอิสระจากกันเท่าที่เป็นไปได้
  - อาจไม่สามารถจับทุก error ในโปรแกรม (e.g., ขึ้นกับความครอบคลุมของกรณีทดสอบ)

คลาสทำงานที่ต้องการทดสอบ

```
package prepLesson;

public class SimpleCompare {

    /**
     * Returns the max value of two integers
     */
    public int max(int x, int y) {
        if (x >= y)
            return x;
        return y;
    }
}
```

การทดสอบแบบดั้งเดิม

```
package prepLesson;
/** A class to test the class SimpleCompare. */
public class TestSimpleCompare {
    /** Runs the tests. */
    public static void main(String[] args) {
        printTestResult(-10, -5);   // negative case x < y
        printTestResult(-12, -15);  // negative case y < x
        printTestResult(10, 30);    // simple case x < y
        printTestResult(20, 12);    // simple case y < x
        printTestResult(3, 3);      // equals positive case
        printTestResult(-7, -7);    // equals negative case
    }
    private static void printTestResult(int a, int b) {
        SimpleCompare comp = new SimpleCompare();
        System.out.print("max(" + a + "," + b + ") ==>  ");
        System.out.println(comp.max(a, b));
    }
}
```

ผลลัพธ์การทดสอบแบบดั้งเดิม

```
max(-10,-5) ==>  -5
max(-12,-15) ==>  -12
max(10,30) ==>  30
max(20,12) ==>  20
max(3,3) ==>  3
max(-7,-7) ==>  -7
```

- ทราบได้อย่างไรว่า การทดสอบนี้ตกหรือผ่าน

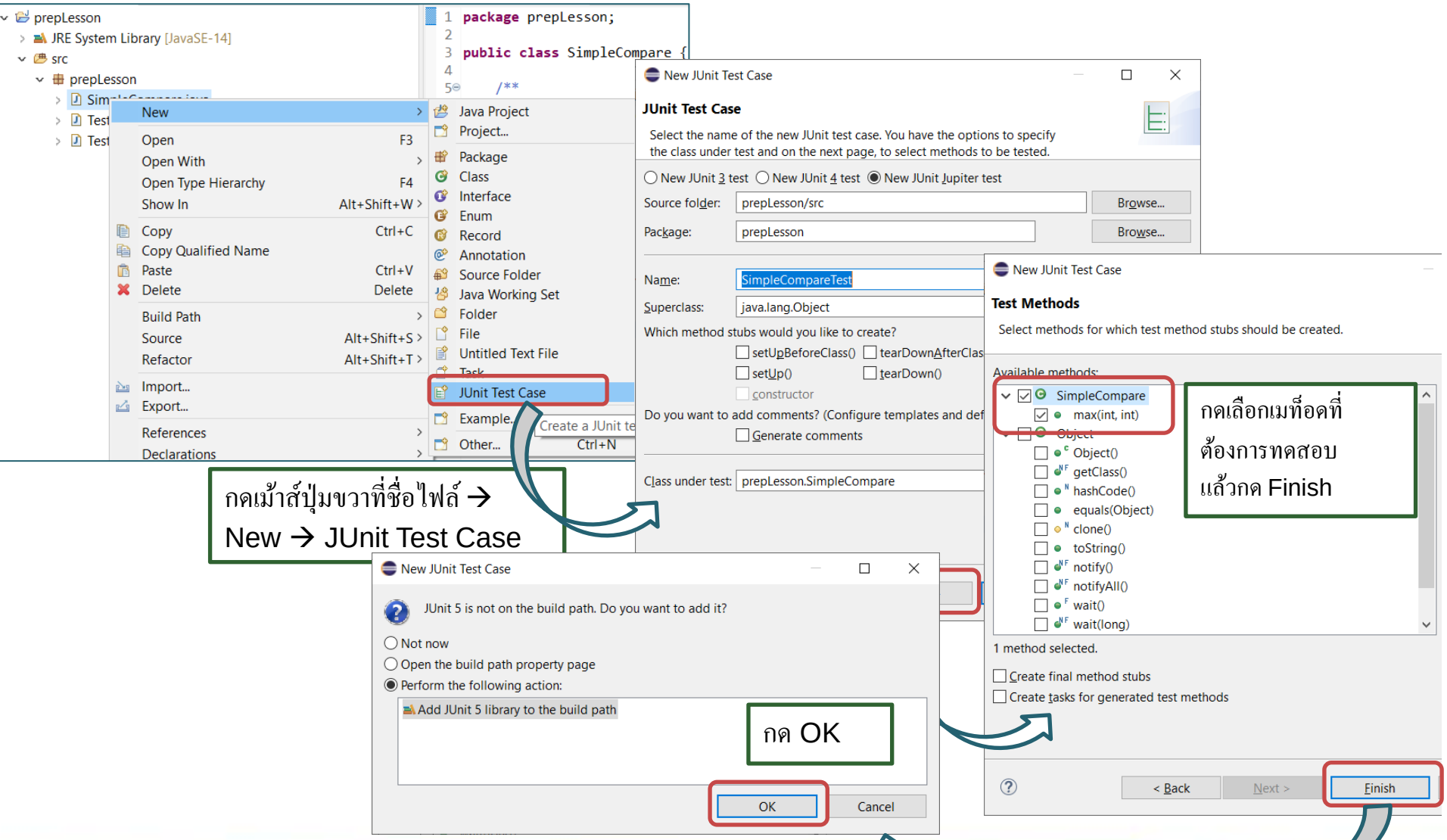
ลักษณะการทดสอบที่ดี

- อัตโนมัติ (Automatic)
- ครอบคลุม (Thorough)
- แม่นยำ (Accurate)
- ทำซ้ำได้ (Repeatable)
- เป็นอิสระ (Independent)
- เป็นมืออาชีพ (Professional)

ทางแก้ปัญหา

- การตรวจสอบแบบอัตโนมัติ (Automatic verification) โดยโปรแกรมทดสอบ
  - สามารถเขียนโปรแกรมทดสอบได้ด้วยตนเอง หรือ
  - ใช้เครื่องมือทดสอบเช่น JUnit
- **JUnit (www.junit.org)**
  - เป็นเครื่องมือช่วยในการทดสอบที่ง่าย ยืดหยุ่น เป็น open-source
  - ง่ายในการรัน (และ rerun) สำหรับหลาย ๆ การทดสอบ
    - เปรียบเทียบค่าที่คาดหวัง (Expected) กับผลลัพธ์ที่ได้จริง (actual)
  - สามารถทำงานได้กับเซตของ test cases ขนาดใหญ่
  - อย่างไรก็ตาม
    - ยังจำเป็นต้องออกแบบกรณีทดสอบที่ดี (ภายใต้สมมติฐาน)

การสร้าง JUnit Test File ใน Eclipse



คลาส SimpleCompareTest ที่ได้จาก JUnit

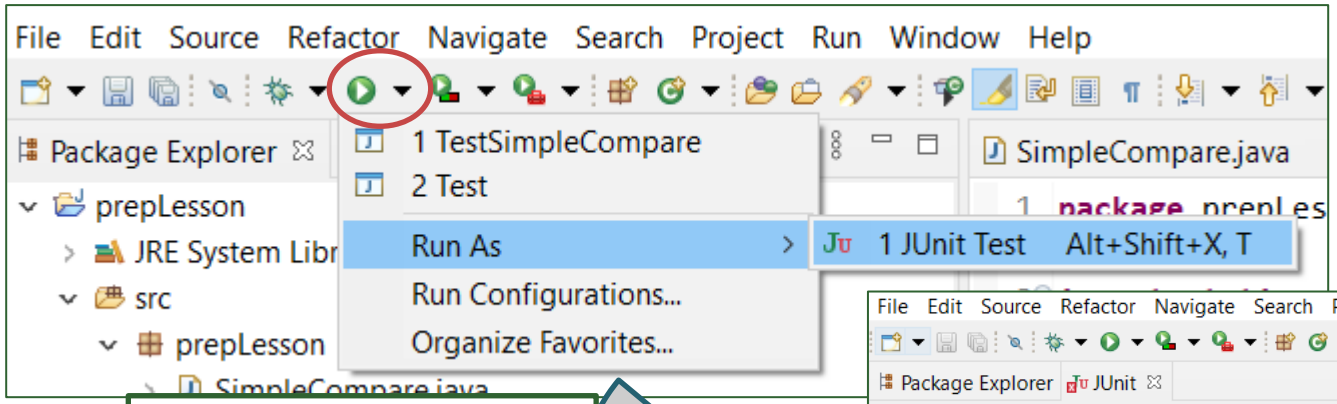
```
package prepLesson;

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

class SimpleCompareTest {

    @Test
    void testMax() {
        fail("Not yet implemented");
    }

}
```

@Test annotation
ระบุเพื่อให้รัน Junit test.

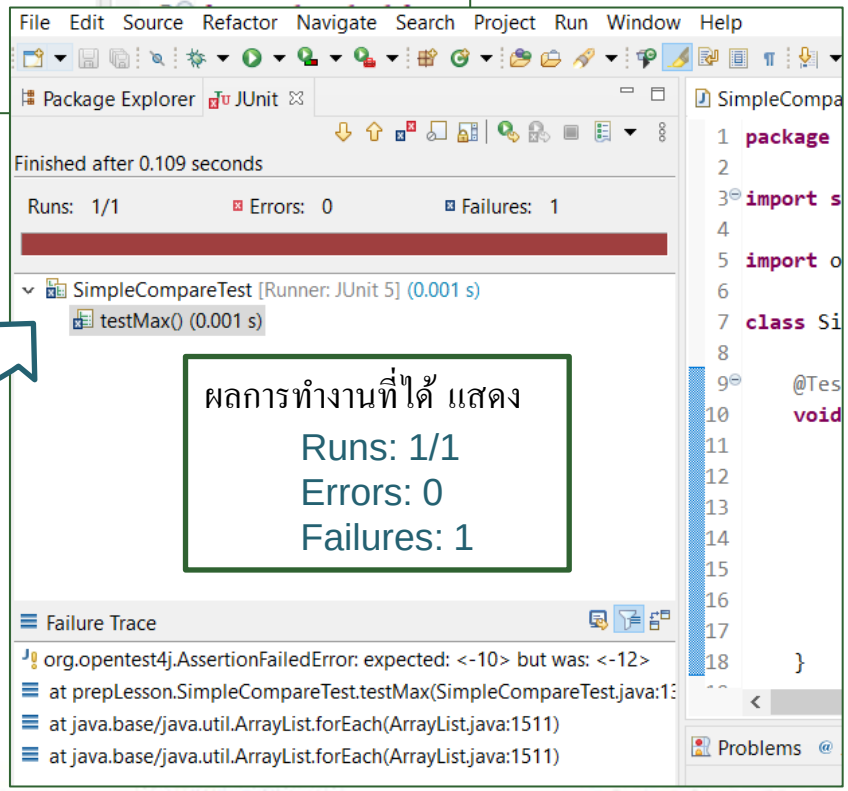
แก้ไขคลาสทดสอบเพื่อสร้างการทดสอบ

```
package prepLesson;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
class SimpleCompareTest {
    @Test
    void testMax() {
        SimpleCompare comp = new SimpleCompare();
        assertEquals(-5, comp.max(-10, -5));
        assertEquals(-10, comp.max(-12,-15)); // assume abnormal case
        assertEquals(30, comp.max(10,30));
        assertEquals(20, comp.max(20,12));
        assertEquals(3, comp.max(3,3));
        assertEquals(-7, comp.max(-7,-7));
    }
}
```

ทดสอบการรัน



กดลูกศร รันที่เมนู เลือก Run As > → JUnit Test



ผลการทำงานที่ได้ แสดง

Runs: 1/1
Errors: 0
Failures: 1

เปรียบเทียบกับผลการทดสอบแบบเดิม

```
max(-10,-5) ==>  -5
max(-12,-15) ==>  -12
max(10,30) ==>  30
max(20,12) ==>  20
max(3,3) ==>  3
max(-7,-7) ==>  -7
```

เทคนิคการทดสอบ Class

- เทคนิค black-box และ white-box สามารถนำมาใช้ในการทดสอบได้
  - การทดสอบค่าขอบ (boundary values)
  - ภายในแต่ละเมท็อด: อย่างน้อยต้องทดสอบความครอบคลุมของแต่ละทางเลือกที่เป็นไปได้ (Conditional Paths)
  - การจัดการกับความผิดพลาด (Error handling)
  - การทดสอบเมท็อดที่เป็นคู่กัน เช่นทดสอบเมท็อดที่กำหนดค่าให้กับ field (Setter Methods) ร่วมกับเมท็อดการอ่านค่า field นั้น (Getter Methods)

Naming Convention

- Test methods ขึ้นต้นด้วยคำว่า “test”

  เช่น testMax, testMin, testAdd

- Test classes ลงท้ายด้วยคำว่า “Test”

  เช่น SimpleCompareTest, MyClassTest

ตัวอย่างการทดสอบ คลาส Point โดยเทคนิค black-box และ white-box testing

```
public class Point {
    private int x ;
    private int y;

    public Point(){
        x = 0;
        y = 0 ;
    }

    public Point(int x1, int y1){
        x = x1 ;
        y = y1 ;
    }

    public int getX(){
        return x;
    }
    public int getY(){
        return y;
    }

    public void setpoint(int x1, int y1){
        this.x =x1 ;
        this.y =y1 ;
    }

    public boolean isTheSame(Point p1){
        if((x ==p1.getX())&&
           (y ==p1.getY()))
            return true;
        else
            return false;
    }
}
```

Default Constructor

Parameterized Constructor

สร้างคลาสทดสอบ PointTest ด้วย JUnit

```
package prepLesson;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class PointTest {

    @Test
    void testPoint() {
        fail("Not yet implemented");
    }

    @Test
    void testPointIntInt() {
        fail("Not yet implemented");
    }

    @Test
    void testGetX() {
        fail("Not yet implemented");
    }

    @Test
    void testGetY() {
        fail("Not yet implemented");
    }

    @Test
    void testSetpoint() {
        fail("Not yet implemented");
    }

    @Test
    void testIsTheSame() {
        fail("Not yet implemented");
    }
}
```

เมท็อดทดสอบ Default Constructor

เมท็อดทดสอบ Parameterized Constructor

การกำหนดวัตถุเริ่มต้นก่อนการทดสอบ (Setup) และ การยกเลิกการกำหนดคลาสต่าง ๆ ของวัตถุหลังการทดสอบ (Tear down)

```
@BeforeEach
void setup() {
    System.out.println("Before Test");
    p1 = new Point(Integer.MAX_VALUE,
             Integer.MAX_VALUE);
}

@AfterEach
void tearDown() {
}
```

- ตัวอย่างการ set up ก่อนการทดสอบ class Point โดยให้ก่อนทดสอบสร้าง Point หนึ่งจุดที่มีค่าพิกัด (Max INT, Max INT)

การทดสอบค่าขอบ (boundary values)

```
@Test
public void testConstrMaxBoundary() {
     assertEquals((double) Integer.MAX_VALUE,
               (double) (p1.getX()),
               "X should be equal to MAX INT");
    assertEquals((double) Integer.MAX_VALUE,
               (double) (p1.getY()),
               "Y should be equal to MAX INT");
}
```

- กรณีทดสอบว่าการทำงานของเมท็อด (constructor) ทำงานถูกต้องหรือไม่

การทดสอบเส้นทางของเงื่อนไข (conditional paths)

```
//ทดสอบจุด 2 จุดที่พิกัดแกน X คนละค่ากัน  ดังนั้น จุดทั้งสองจึงเป็นคนให้คำตอบผิด
@Test
void testNotSamePoint_X() {
    Point p2 = new Point(Integer.MIN_VALUE, Integer.MAX_VALUE);
    assertFalse(p1.isTheSame(p2),
           "To be the same point, Coordinate X must be the same");
}

//ทดสอบจุด 2 จุดที่พิกัดแกน Y คนละค่ากัน  ดังนั้น จุดทั้งสองจึงเป็นคนให้คำตอบผิด
@Test
void testNotSamePoint_Y() {
    Point p2 = new Point(Integer.MAX_VALUE, 1);
    assertFalse(p1.isTheSame(p2),
          "To be the same point, Coordinate Y must be the same");
}

//ทดสอบจุด 2 จุดที่พิกัดแกน X และแกน  Y  คนละค่ากัน  ดังนั้น จุดทั้งสองจึงเป็นคนให้คำตอบผิด
@Test
void testNotSamePoint_XY() {
    Point p2 = new Point(1, 1);
    assertFalse(p1.isTheSame(p2),
          "To be the same point, Coordinate X and Y must be the same");
}
```

การทดสอบเส้นทางของเงื่อนไข (conditional paths)

```
@Test
public void testSamePoint(){
    Point p2 = new Point(Integer.MAX_VALUE,
                         Integer.MAX_VALUE);
    assertTrue(p1.isTheSame(p2),
        "To be the same point, coordinate"
         + " X and Y must be the same."
    );
}
```

- กรณีทดสอบ เป็นการทดสอบจุดพิกัด x, y ของจุด 2 จุดที่มีพิกัดเหมือนกัน ทำให้ผลการทดสอบว่าเป็นจุดเดียวกันต้องเป็น true

การทดสอบเมท็อดที่เป็นคู่กัน เช่นทดสอบเมท็อดที่กำหนดค่าให้กับ field (Setter Methods) ร่วมกับเมท็อดการอ่านค่า field นั้น (Getter Methods)

```
@ParameterizedTest
@ValueSource(ints = { -1, 0,
                      Integer.MAX_VALUE })
void testSetpoint(int v) {
    p1.setpoint(v, v);
    assertEquals(v, p1.getX());
    assertEquals(v, p1.getY());
}
```

- ทดสอบว่าหลังการใช้เมท็อดประเภท Setter เพื่อกำหนดค่าจุดพิกัด Point -1, 0, หรือ Integer.MAX_VALUE แล้วค่า x, y เปลี่ยนแปลงจริงตามค่าเหล่านั้น ตามลำดับ

@JUnit 5 Annotations (1)

ดูเพิ่มเติมที่ https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/package-summary.html

annotation	ความหมาย/ตัวอย่าง

@Test	เมท็อดเพื่อใช้ทดสอบ
	`@Test`
	`void testGetX() { assertEquals(0, p1.getX()); }`

@ParameterizedTest	เมท็อดทดสอบด้วยพารามิเตอร์หลายค่า (คล้าย @Test) แต่ทำงานหลายครั้งกับหลายค่าอาร์กิวเมนต์
	`@ParameterizedTest`
	`@ValueSource(ints = { -1, 0, Integer.MAX_VALUE })`
	`void testSetpoint(int v) {`
	`  p1.setpoint(v, v);`
	`  assertEquals(v, p1.getX());`
	`  assertEquals(v, p1.getY());`
	`}`

@DisplayName	ให้ชื่อกับการทดสอบ (ใช้ได้กับทั้งคลาสและเมท็อด) แทนชื่อปริยาย
	`@Test`
	`void testGetX() { assertEquals(0, p1.getX()); }`

@Disabled	ใช้เพื่อระบุให้ไม่ต้องทดสอบ (disable)
	`@Disabled`
	`@Test`
	`void testIsTheSame() { fail("Not yet implemented"); }`

@JUnit 5 Annotations (2)

ดูเพิ่มเติมที่ https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/package-summary.html

annotation	ความหมาย/ตัวอย่าง

@RepeatedTest	เมท็อดทดสอบซ้ำตามจำนวนครั้งที่ระบุ
	`@RepeatedTest(value = 5, name = "{displayName} {currentRepetition}/{totalRepetitions}")`
	`@DisplayName("RepeatingTest")`
	`void repeatedTestSetPoint(RepetitionInfo repInfo) {`
	`  int i = 3;`
	`  p1.setpoint(repInfo.getCurrentRepetition(),`
	`  repInfo.getCurrentRepetition());`
	`  assertEquals(repInfo.getCurrentRepetition(), p1.getX());`
	`  assertEquals(i, p1.getY());`
	`}`

@BeforeEach	ระบุให้ execute เมท็อดก่อนทุกเมท็อดทดสอบ (@Test) (เท่ากับ @Before ใน v.4)
	`@BeforeEach`
	`void setup() {`
	`  System.out.println("Before Test");`
	`  p1 = new Point();`
	`}`

@BeforeAll	ระบุให้ execute เมท็อดสถิตย์นี้ก่อนทดสอบคลาสครั้งเดียว (= @BeforeClass ใน v.4)
	`@BeforeAll`
	`static void initConfig() {`
	`  System.out.println("Start the test");`
	`}`

เมท็อด Assertion

ดูเพิ่มที่ https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html

Method	Description
assertEquals(a,b)	Test if a is equal to b
assertFalse(a)	Test if a is false
assertNotSame(a, b)	Test if a and b do not refer to the identical object
assertNull(a)	Test if a is null
assertSame(a,b)	Test if a and b refer to the identical object
assertTrue(a)	Test if a is true

- เมท็อดทั้งหมดเป็น static methods กำหนดไว้ใน org.junit.jupiter.api.Assertions
- มีอีกรูปแบบ ที่มี error messages (ชนิด String) เป็นพารามิเตอร์เพิ่ม เช่น

```
assertEquals(a, b, "Failed on a equals b");
```

ตัวอย่างโค้ด JUnit คลาสทดสอบ PointTest (1/5)

```
package prepLesson;

import static org.junit.jupiter.api.Assertions.*;
    // ละไม่แสดงการ import
class PointTest {
        private Point p1;

        @BeforeAll
        static void initConfig() {
                System.out.println("Start the test");
        }

        @BeforeEach
        void setup() {
                System.out.println("Before Test");
                p1 = new Point(Integer.MAX_VALUE, Integer.MAX_VALUE);
        }

        @AfterEach
        void tearDown() {
        }
```

ตัวอย่างโค้ด JUnit คลาสทดสอบ PointTest (2/5)

```
@Test
void testPoint() {
        p1 = new Point();
        assertNotNull("Not Null");
        assertEquals(0, p1.getX());
        assertEquals(0, p1.getY());
}

@Test
void testPointIntInt() {
        p1 = new Point(Integer.MAX_VALUE, Integer.MIN_VALUE);
        assertNotNull("Not Null");
        assertEquals(Integer.MAX_VALUE, p1.getX());
        assertEquals(Integer.MIN_VALUE, p1.getY());
}

@Test
void testGetX() { assertEquals(0, p1.getX()); }

@Test
void testGetY() { assertEquals(Integer.MAX_VALUE, p1.getY()); }
```

ตัวอย่างโค้ด JUnit คลาสทดสอบ PointTest (3/5)

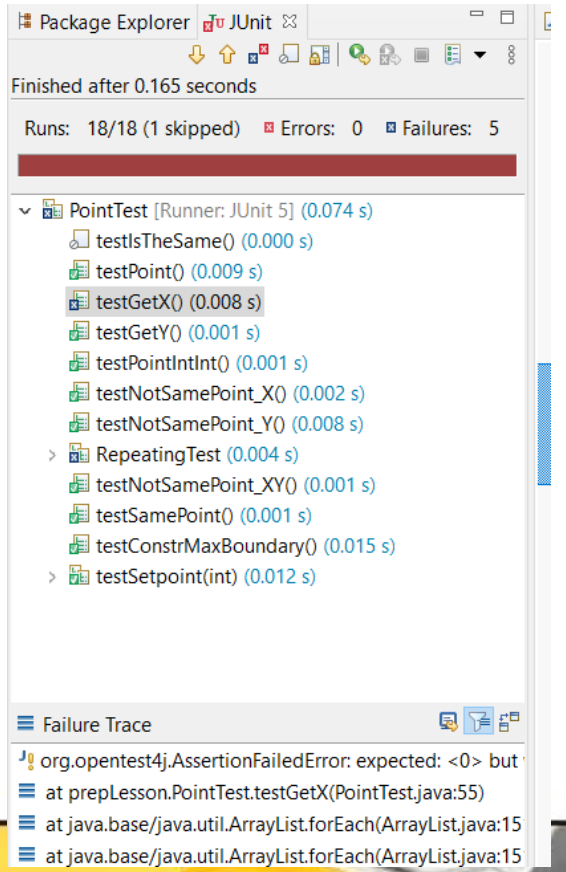
```
@Test
public void testConstrMaxBoundary() {
          assertEquals((double) Integer.MAX_VALUE, (double) (p1.getX()),
                    "X should be equal to MAX INT");
          assertEquals((double) Integer.MAX_VALUE, (double) (p1.getY()),
                    "Y should be equal to MAX INT");
}

@ParameterizedTest
@ValueSource(ints = { -1, 0, Integer.MAX_VALUE })
void testSetpoint(int v) {
          p1.setpoint(v, v);
          assertEquals(v, p1.getX());
          assertEquals(v, p1.getY());
}

@RepeatedTest(value = 5,
     name = "{displayName} {currentRepetition}/{totalRepetitions}")
@DisplayName("RepeatingTest")
void repeatedTestSetPoint(RepetitionInfo repInfo) {
          int i = 3;
          p1.setpoint(repInfo.getCurrentRepetition(),
                    repInfo.getCurrentRepetition());
          assertEquals(repInfo.getCurrentRepetition(), p1.getX());
          assertEquals(i, p1.getY());
}
```

ตัวอย่างโค้ด JUnit คลาสทดสอบ PointTest (4/5)

```
@Disabled
@Test
void testIsTheSame() {  fail("Not yet implemented"); }

@Test
public void testSamePoint(){
        Point p2 = new Point(Integer.MAX_VALUE, Integer.MAX_VALUE);
        assertTrue(p1.isTheSame(p2), "To be the same point, coordinate"
                + " X and Y must be the same.");
}

@Test
void testNotSamePoint_X() {
        Point p2 = new Point(Integer.MIN_VALUE, Integer.MAX_VALUE);
        assertFalse(p1.isTheSame(p2),
           "To be the same point, Coordinate X must be the same");
}
```

ตัวอย่างโค้ด JUnit คลาสทดสอบ PointTest (5/5)

```
        @Test
        void testNotSamePoint_Y() {
                Point p2 = new Point(Integer.MAX_VALUE, 1);
                assertFalse(p1.isTheSame(p2),
                    "To be the same point, Coordinate Y must be the same");
        }

        @Test
        void testNotSamePoint_XY() {
                Point p2 = new Point(1, 1);
                assertFalse(p1.isTheSame(p2),
               "To be the same point, "
              + "Coordinate X and Y must be the same");
        }
}
```



Resources:

- JUnit: http://www.junit.org
- JUnit User Guide: https://junit.org/junit5/docs/current/user-guide
- JUnit APIs: https://junit.org/junit5/docs/current/api/