ตัวควบคุมและคลาสอรรถประโยชน์สำหรับตัวตรวจโปรแกรมอัตโนมัติ

นายเฉลิมวุฒิ น้อยอุ่นแสน

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2552

CONTROLLER AND UTILITY CLASS FOR AUTOMATIC PROGRAM CHECKER

Mr. Chaluemwut Noyunsan

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2009

หัวข้อวิทยานิพนธ์	ตัวควบคุมและคลาสอรรถประโยชน์สำหรับตัวตรวจโปรแกรมอัตโนมัติ
โดย	นายเฉลิมวุฒิ น้อยอุ่นแสน
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์

(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล)

..... กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.สุกรี สินธุภิญโญ)

..... กรรมการภายนอกมหาวิทยาลัย

(อาจารย์ ดร.วรเศรษฐ สุวรรณิก)

เฉลิมวุฒิ น้อยอุ่นแสน : ตัวควบคุมและคลาสอรรถประโยชน์สำหรับตัวตรวจโปรแกรมอัตโนมัติ. (CONTROLLER AND UTILITY CLASS FOR AUTOMATIC PROGRAM CHECKER) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร.สมชาย ประสิทธิ์จูตระกูล, 75 หน้า.

วิทยานิพนธ์ฉบับนี้นำ เสนอตัวควบคุมการตรวจและคลาสอรรถประโยชน์ ที่อำนวยความสะดวกในการพัฒนาตัวตรวจโปรแกรมภาษาจาวา ตัวควบคุมอาศัยกลไกต่าง ๆ ที่มีอยู่ในระบบจาวา เช่น การใช้ตัวจัดการความมั่นคงระบบของจาวาเพื่อป้องกันไม่ให้มีการทำงานที่ไม่ได้รับอนุญาต การสั่งตัวตรวจทำงานแบบแยกหน่วยประมวลย่อยเพื่อควบคุมเวลาการทำงานไม่ให้เกินเวลาที่จำกัดไว้ การใช้ annotation ของจาวาเพื่อกำหนดลักษณะกำกับการตรวจของตัวตรวจ การใช้แฟ้มข้อมูลทดสอบ การห้ามใช้คลาสมาตรฐานที่มีอยู่ในจาวา เป็นต้น นอกจากนี้ยังมีคลาสอรรถประโยชน์ให้บริการ เช่น การจัดเตรียมข้อมูลขาเข้าและอ่านผลจากจอภาพ การเปรียบเทียบผลลัพธ์ทั้งแบบประมาณแบบแม่นตรง การทดสอบการสุ่มคำตอบหรือการคงคำตอบเดี่ยวของเมท็อดแบบบูลีน การทำสำเนาอ็อบเจกต์และการตรวจสอบการเปลี่ยนแปลงสถานะของอ็อบเจกต์ การอ่านค่าและเปลี่ยนค่าภายในอ็อบเจกต์ เป็นต้น โปรแกรมนี้มีขนาดไม่ถึง 91 กิโลไบต์ สามารถแนบไปกับชุดแบบฝึกหัดแต่ละข้อพร้อมตัวตรวจเพื่อให้ผู้เรียนเขียนโปรแกรมได้ฝึกปฏิบัติจริง และตรวจทราบผลของโปรแกรมที่เขียนได้ทันทีทันใด

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก .....
ปีการศึกษา 2552

5071412021 : MAJOR COMPUTER SCIENCE

KEYWORDS : PROGRAMMING / PROGRAM ASSESSMENT

CHALUEMWUT NOYUNSAN : CONTROLLER AND UTILITY CLASS FOR AUTOMATIC PROGRAM CHECKER. THESIS ADVISOR : ASSC. PROF. SOMCHAI PRASITJUTRAKUL, Ph.D. , 75 pp.

This paper presents a controller and a utility class, named JTest101, which facilitates tester developments for assessing Java programs. The test controller uses existing Java platform mechanisms such as using a security manager to protect system resources, limiting tested program's execution time by running the tested program in another thread, using annotation feature to specify testing behaviors. Moreover, prohibit use standard class java, read input and output from file, the system provides frequently used services such as input data preparations, readoutput, compare output for exact match and approximate match, testing random and fixed value of boolean methods, compare state of object, get and set field of object. JTest101's size is only 91KB which can be embedded into each exercise package so that students can practice programmings and instantaneously get assessment feedback

Department : Computer Engineering Student's Signature .....

Field of Study : Computer Science Advisor's Signature .....

Academic Year : 2009

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี เนื่องมาจากความช่วยเหลืออย่างดียิ่งของท่าน รศ.ดร.สมชาย ประสิทธิ์จูตระกูล อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้สละเวลาเพื่อให้คำปรึกษา รวมทั้งให้ความช่วยเหลือ และให้ความรู้ที่เป็นประโยชน์ตลอดมา และผู้วิจัยขอกราบขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ทุกท่านที่ได้ให้คำแนะนำในการพัฒนางานวิจัยนี้

ขอขอบคุณ นาง ศิวนาถ น้อยอุ่นแสน และ ศิวัช น้อยอุ่นแสน ที่คอยให้กำลังใจและให้คำแนะนำในการทำงานวิจัย

ท้ายที่สุด ผู้วิจัยขอกราบขอบพระคุณ คุณพ่อและคุณแม่ ผู้ซึ่งเป็นผู้ให้กำเนิดและเลี้ยงดูมาอย่างดี อีกทั้งคอยสนับสนุนด้านการเงินเป็นอย่างดี

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

สารบัญรหัส

หน้า

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ปัจจัยสำคัญประการหนึ่งในการเรียนวิชาการเขียนโปรแกรมให้ได้ผล คือ การฝึกปฏิบัติการพัฒนาโปรแกรมจริงอันประกอบด้วยการวิเคราะห์ความต้องการ การออกแบบขั้นตอนการทำงาน การลงรหัสคำสั่ง การสั่งทำงาน และการตรวจหาจุดบกพร่อง ทักษะเหล่านี้จะได้มาก็ด้วยการลงมือฝึกเขียนโปรแกรมจริง คงปฏิเสธไม่ได้ว่า การได้ฝึกปฏิบัติบ่อย ๆ กับโจทย์มาก ๆ ย่อมเสริมทักษะการเขียนโปรแกรมได้ดีกว่าการอ่านและท่องจำรูปแบบและกฎเกณฑ์การใช้งานของแต่ละคำสั่งในภาษา แต่ผู้เริ่มเรียนมักขาดทักษะด้านการทดสอบโปรแกรมจึงมักไม่แน่ใจว่าโปรแกรมที่เขียนขึ้นทำงานได้ถูกต้องครบถ้วนตามข้อกำหนดของโจทย์หรือไม่ ปัญหานี้แก้ไขได้ไม่ยาก หากผู้ออกโจทย์ เขียนตัวตรวจโปรแกรมเพื่อให้ผู้เรียนนำไปใช้ทดสอบของตน ทำให้ทราบผลการทำงานว่าถูกหรือผิด ย่อมเป็นการเร่งเวลาการเรียนรู้ให้รวดเร็วขึ้น แต่โดยทั่วไป การพัฒนาตัวตรวจโปรแกรมใช้เวลาผู้ออกโจทย์มาก ส่งผลให้ผู้ออกโจทย์พร้อมตัวตรวจได้ไม่มากนัก

งานวิจัยนี้นำเสนอตัวควบคุมการตรวจและคลาสอรรถประโยชน์ที่ช่วยให้สามารถพัฒนาตัวตรวจได้ง่ายขึ้น รหัสที่1.1 แสดงตัวอย่างตัวตรวจเมท็อดหาผลรวมของแถวลำดับ การสั่งทำงานและการตรวจผล การเรียกใช้ของบริการคลาสอรรถประโยชน์ `TestUtil` และเขียนบรรทัด `@Test` กำกับการตรวจ เพื่อให้ตัวควบคุมการตรวจปฏิบัติงานตามที่เขียนกำกับไว้ (รหัสที่ 1.1 เขียนกำกับให้ทำการเรียกตัวตรวจ 10 ครั้ง ให้ครั้งละ 1 คะแนน โดยในแต่ละครั้งอนุญาตให้ทำงานได้ไม่เกิน 1000 มิลลิวินาที)

```
@Test(loops = 10, points = 1, timeout=1000)
public double testSum() {
    int n = TestUtil.random(100, 200); // สุ่มจำนวนเต็มค่าอยู่ระหว่าง 100 ถึง 200
    int[] p = TestUtil.fillRandom(new int[n], 10, 1000);
    int sum = Calculation.sum(p); // เรียกเมท็อดผู้เรียน
    int sumAns = 0;
    for (int i : p) { // คำนวณหาผลรวม
        sumAns += i;
    }
    TestUtil.assertTrue(sumAns == sum); // เปรียบเทียบผลเฉลยกับคำตอบของผู้เรียน
    return 1; // คืนค่าคะแนน
}
```

รหัสที่ 1.1 ตัวอย่างตัวตรวจเมท็อดหาผลรวมของแถวลำดับ

1.2 วัตถุประสงค์ของการวิจัย

ออกแบบและพัฒนาตัวควบคุมและคลาสอรรถประโยชน์สำหรับตัวตรวจโปรแกรมอัตโนมัติ เพื่ออำนวยความสะดวกให้การพัฒนาตัวตรวจ

1.3 ขอบเขตของการวิจัย

1.3.1 พัฒนาตัวควบคุมการตรวจที่สามารถกำหนดพฤติกรรมการตรวจดังต่อไปนี้

1 จำนวนรอบที่ทำการตรวจ

2 คะแนนเต็มต่อการตรวจหนึ่งรอบ

3 วิธีการคำนวณคะแนน (คะแนนเฉลี่ย, คะแนนรวม, คะแนนมากสุด)

4 เวลามากที่สุดที่อนุญาตให้ทำงานต่อการตรวจหนึ่งรอบ

5 จำนวนครั้งที่อนุญาตให้ผู้เรียนเรียกตรวจ

6 แฟ้มข้อมูลทดสอบและผลที่คาดว่าจะได้ของแต่ละข้อมูลทดสอบ

7 กำหนดคลาสมาตรฐานของจาวาที่ไม่อนุญาตให้มีการเรียกใช้ในโปรแกรมที่ถูกตรวจ

1.3.2 พัฒนาคลาสอรรถประโยชน์ที่เรียกใช้ได้ในตัวตรวจ ที่มีบริการดังต่อไปนี้

1 บริการการสร้างชุดข้อมูลทดสอบ

2 บริการป้อนข้อมูลให้กับแป้นพิมพ์และอ่านผลที่แสดงจอภาพของโปรแกรมผู้เรียน

3 บริการเปรียบเทียบผลลัพธ์ทั้งแบบประมาณและแบบแม่นตรง

4 บริการแสดงผลการตรวจสอบ และแสดงผิดพลาดในการทำงาน

5 บริการตรวจพฤติกรรมการคืนค่าคงตัวหรือค่าสุ่มในเมท็อดที่คืนผลแบบบูลีน

6 บริการทำสำเนาอ็อบเจกต์เพื่อตรวจสอบการไม่เปลี่ยนแปลงสถานะของอ็อบเจกต์

7 บริการอ่านค่าและเปลี่ยนค่าภายในอ็อบเจกต์

1.4 ข้อจำกัดของการวิจัย

งานวิจัยนี้จะครอบคลุมเฉพาะการตรวจความถูกต้องของผลการทำงานเท่านั้นโดยจะไม่ครอบคลุมไปถึงการตรวจแบบวิเคราะห์รหัสต้นฉบับ

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ตัวควบคุมการตรวจและคลาสอรรถประโยชน์ในงานวิจัยนี้ อำนวยความสะดวกในการพัฒนาตัวตรวจโปรแกรมอัตโนมัติ ที่สามารถใช้ได้กับโจทย์ ที่เกี่ยวข้องกับทักษะการเขียนโปรแกรม เช่น ในวิชาการเขียนโปรแกรม โครงสร้างข้อมูล และอัลกอริทึม เป็นต้น

1.6 วิธีดำเนินการวิจัย

1 ศึกษาลักษณะของโจทย์ฝึกปฏิบัติการเขียนโปรแกรมและวิชาโครงสร้างข้อมูล และวิธีการตรวจเพื่อประเมินผล

2 ศึกษาลักษณะสมบัติต่างๆ ของระบบจาวา ที่สามารถประยุกต์การทำงานของตัวควบคุมการตรวจและคลาสอรรถประโยชน์สำหรับการตรวจ

3 ออกแบบ พัฒนา และทดสอบตัวควบคุมการตรวจและคลาสอรรถประโยชน์สำหรับการตรวจ

4 สรุปผลการวิจัยและจัดทำวิทยานิพนธ์เป็นรูปเล่ม

1.7 โครงสร้างของวิทยานิพนธ์

เนื้อหาของวิทยานิพนธ์ฉบับนี้ถูกแบ่งออกเป็น 5 บท ซึ่งประกอบด้วย บทที่ 1 เป็นบทนำ บทที่ 2 กล่าวถึงทฤษฎีและงานวิจัยที่เกี่ยวข้อง บทที่ 3 กล่าวถึงการดำเนินงาน บทที่ 4 เป็นการแสดงตัวอย่างบางส่วนของตัวตรวจ และท้ายสุดคือบทที่ 5 เป็นการสรุปผลและข้อเสนอแนะของงานวิจัย ซึ่งอาจมีประโยชน์ต่องานวิจัยอื่นๆ ต่อไปในอนาคต

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 วิวัฒนาการการตรวจโปรแกรม

ตัวตรวจโปรแกรมคอมพิวเตอร์เป็นงานวิจัยที่มีผู้ดำเนินการมาตั้งแต่ปี ค.ศ.1960 (ที่ตรวจโปรแกรมภาษาแอสเซมบลี้บนบัตรเจาะรู [1]) มาจนถึงปัจจุบัน [2][3] ในยุคแรกเป็นการสร้างระบบอย่างง่าย ๆ เพื่อใช้ในการตรวจโปรแกรมของผู้เรียน ข้อดีของระบบในยุคนี้คือสามารถตรวจโปรแกรมของผู้เรียนที่มีจำนวนมากได้ ในยุคต่อมาเป็นการสร้างเครื่องมือที่มีการสร้างรายคำสั่ง (command-line) หรือ ส่วนต่อประสานกราฟิกกับผู้ใช้ (GUI) และปัจจุบันเป็นการพัฒนาระบบการตรวจโปรแกรมโดยผู้ใช้สามารถเขียนโปรแกรมและตรวจโปรแกรมผ่านทางเว็บเบราว์เซอร์

2.2 วิธีตรวจโปรแกรม

ในการตรวจโปรแกรมแบบดั้งเดิม ผู้สอนจะให้ผู้เรียนเขียนโปรแกรมในกระดาษ จากนั้นจึงพิจารณาโปรแกรมในกระดาษแต่ละฉบับ และให้คะแนนตามความถูกต้องและเกณฑ์ต่าง ๆ ที่กำหนดไว้ การตรวจโปรแกรมจำนวนมากในลักษณะเช่นนี้ใช้เวลามาก จึงนำมาสู่การออกแบบและพัฒนาตัวตรวจโปรแกรมอัตโนมัติ การตรวจโปรแกรมนั้นมี 2 ลักษณะ ดังนี้ [3]

2.2.1 การตรวจผลการทำงานของโปรแกรม (Dynamic Assessment)

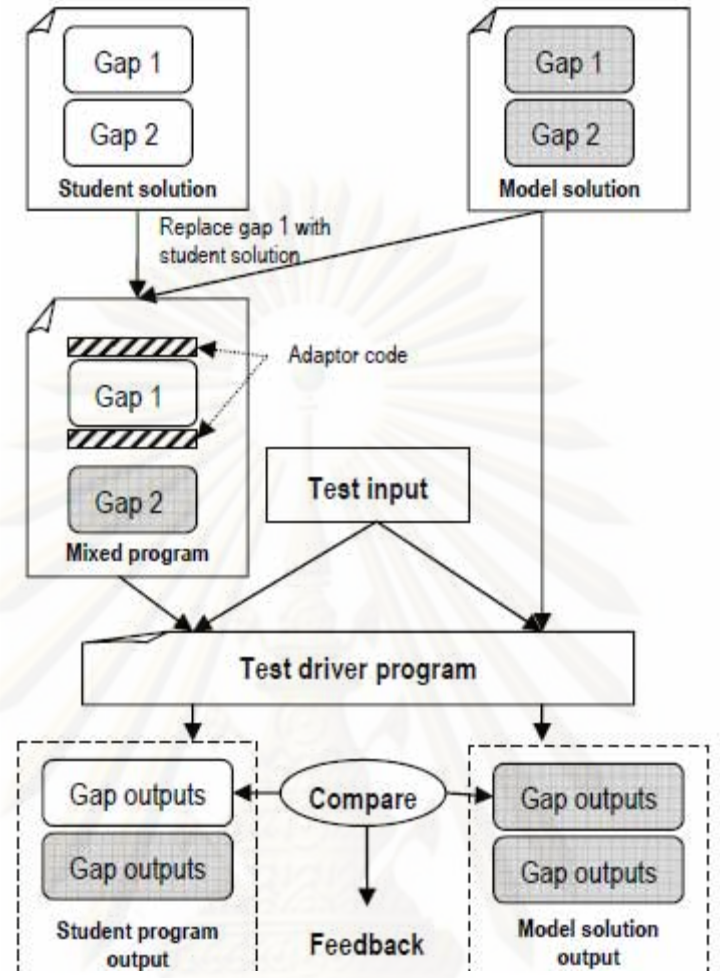
วิธีนี้นำรหัสต้นฉบับของโปรแกรมไปแปลเป็นรหัสเครื่อง จากนั้นสั่งให้ทำงานจริงกับข้อมูลทดสอบเพื่อตรวจว่า ผลลัพธ์ที่ได้ตรงตามที่คาดไว้หรือไม่ นอกจากการตรวจสอบความถูกต้องของการทำงานแล้ว ตัวตรวจยังสามารถทดสอบประสิทธิภาพของการทำงานได้อีกด้วย อย่างไรก็ตามวิธีนี้มีความเสี่ยงต่อความบกพร่อง ของโปรแกรมผู้เรียน ที่อาจสร้างความเสียหายให้กับระบบ หรือฐานข้อมูลในระบบของตัวตรวจได้ หรือโปรแกรมอาจทำงานผิดพลาดจนทำให้มีการใช้หน่วยความจำมากเกินไป ดังนั้นจึงมีความจำเป็นที่ตัวตรวจต้องเตรียมสภาพแวดล้อมของการทำงานของโปรแกรมให้ปลอดภัยจากสถานการณ์ดังกล่าว การตรวจในลักษณะนี้สามารถตรวจโปรแกรมของผู้เรียนได้ในหลายแง่มุมดังนี้

2.2.1.1 การตรวจความถูกต้อง (Functionality)

โดยทั่วไปการตรวจแบบฝึกหัดการเขียนโปรแกรมเป็นการตรวจความถูกต้องของการทำงานตามประเด็นที่ตั้งไว้ การตรวจกระทำได้โดยสั่งโปรแกรมผู้เรียนให้ทำงานกับชุดข้อมูลทดสอบหลาย ๆ ชุด หลาย ๆ ครั้ง เพื่อให้แน่ใจว่าการทำงานถูกต้อง และนำผลที่ได้จากโปรแกรม เช่น การคืนค่า หรือการแสดงทางหน้าจอมาเปรียบเทียบกับค่าที่ถูกต้อง

CourseMarker [4] เป็นระบบที่พัฒนามาจาก Ceilidh [5] ที่ใช้อยู่ที่มหาวิทยาลัยนอตติงแฮม CourseMarker เป็นระบบรับ-ให้บริการ (client-server) ใช้จาวาในการพัฒนา โดยใช้JFC (Java Foundation Classes) ในการพัฒนาลูกข่าย (client) ระบบสามารถตรวจได้ทั้งโปรแกรมภาษาจาวา และ ภาษาซีพลัสพลัส การตรวจให้คะแนนใช้หลายวิธีในการวัดผล เช่น การวิเคราะห์โครงสร้างโปรแกรมของผู้เรียน การใส่หมายเหตุในโปรแกรม (comment) เป็นต้น สำหรับการตรวจความถูกต้องของ CourseMarker คือสั่งโปรแกรมทำงานด้วยข้อมูลทดสอบที่ได้เตรียมไว้ แล้วดูผลลัพธ์เทียบกับผลลัพธ์ของโปรแกรมที่ถูกต้อง ระบบนี้ได้ดูแลความมั่นคงจากการทำงานของโปรแกรมผู้เรียน โดยให้โปรแกรมผู้เรียนอ่านและเขียนแฟ้มต่าง ๆ ได้เฉพาะในพื้นที่กำหนดไว้เท่านั้น และใช้ความสามารถที่มีอยู่ของระบบปฏิบัติการ เพื่อควบคุมสิทธิ์ของผู้ใช้ โดยใช้คำสั่ง run as ที่มีอยู่ในระบบปฏิบัติการวินโดวส์ และใช้ SUID/GUID ในระบบปฏิบัติการยูนิกซ์เพื่อควบคุมสิทธิ์ จุดเด่นของระบบนี้อีกอย่างหนึ่งคือ การกำหนดประเภทของผู้ใช้งานไว้ชัดเจน เช่น ผู้เรียน ผู้สอน ผู้ดูแลระบบ เป็นต้น เพื่อรองรับการทำงานของผู้ใช้งานแต่ละกลุ่ม เช่น ผู้เรียนสามารถเห็นโจทย์และเรียกตรวจหลังจากทำเสร็จ ส่วนผู้ออกโจทย์ก็สามารถออกและเลือกโจทย์ที่จะแสดงต่อผู้เรียนได้

ELP (Environment for Learning to Program) ได้รับการพัฒนาจากมหาวิทยาลัย Queensland University of Technology (QUT) [6] ระบบนี้มีการเว้นช่องเพื่อให้ผู้เรียนเติมรหัสต้นฉบับที่หายไปให้สมบูรณ์เรียกว่า “fill-in the gap” ELP รองรับการตรวจความถูกต้องของการทำงาน และการตรวจโดยการวิเคราะห์รหัสต้นฉบับ การตรวจความถูกต้องของการทำงานมี 2 ลักษณะคือ การทดสอบหน้าที่การทำงาน (black box testing) และการทดสอบความถูกต้องการทำงาน (white box testing) โดยการทดสอบหน้าที่การทำงานเป็นการทดสอบโปรแกรมทั้งหมด โดยจะรวมทุกช่องว่างที่ผู้เรียนเติมเพื่อนำมาทดสอบ การทดสอบจะทำโดยนำโปรแกรมที่สมบูรณ์ของผู้เรียนมาทดสอบกับข้อมูลชุดเดียวกับโปรแกรมที่ถูกต้องแล้วดูผลลัพธ์ ส่วนการทดสอบความถูกต้องการทำงานเป็นการนำแต่ละช่องว่างมาทดสอบ แน่นอนว่าแต่ละช่องว่างยังทำงานไม่ได้ ดังนั้นจึงต้องนำช่องว่างนั้นไปรวมกับโปรแกรมที่ถูกต้อง เพื่อทดสอบว่าช่องว่างนั้นทำงานถูกต้องหรือไม่ การทำงานสามารถแสดงได้ดังรูปที่ 2.1



รูปที่ 2.1 ภาพรวมการทดสอบความถูกต้องการทำงานของระบบ ELP [6]

2.2.1.2 **การตรวจประสิทธิภาพ** (Efficiency)

โดยทั่วไปการวัดประสิทธิภาพการทำงานจะวัดจากเวลาในการทำงานของโปรแกรมหรือสามารถวัดจากจำนวนคำสั่งที่ทำงาน งานวิจัยในกลุ่มนี้มีดังนี้

Assyst [7] ได้รับการพัฒนาโดย Jackson และ Usher ระบบนี้ตรวจโปรแกรมทั้งแบบ การตรวจสอบความถูกต้อง การตรวจสอบประสิทธิภาพ การตรวจสอบรูปแบบการเขียน การตรวจสอบความซับซ้อนของโปรแกรม และทักษะการสร้างข้อมูลทดสอบของผู้เรียน ในหัวข้อนี้จะพูดถึงการวัดประสิทธิภาพโปรแกรมของผู้เรียน

การวัดประสิทธิภาพของ Assyst มี 2 ลักษณะ การวัดแบบแรกคือ การวัดเวลาที่โปรแกรมของผู้เรียนทำงาน โดยเทียบเวลากับโปรแกรมที่ทำงานถูกต้อง ผู้เรียนจะได้คะแนนลดลงเมื่อโปรแกรมของผู้เรียนใช้เวลามากกว่า แต่วิธีนี้มีปัญหาคือการตรวจวิชาเขียนโปรแกรมเบื้องต้นนั้นเป็นการตรวจโปรแกรมขนาดเล็ก ดังนั้นเวลาทำงานของโปรแกรมจึงไม่ต่างกันมาก วิธีที่สองคือการวัดจำนวนคำสั่งโดยเมื่อระบบสั่งให้โปรแกรมของผู้เรียนทำงานระบบจะบันทึกจำนวนคำสั่งที่

โปรแกรมของผู้เรียนทำงาน เมื่อโปรแกรมของผู้เรียนทำงานเสร็จก็จะนำจำนวนคำสั่งมาเทียบกับโปรแกรมที่ทำงานถูกต้อง

Ceilidh เป็นซอฟต์แวร์ที่ใช้ในการเรียนวิชาการเขียนโปรแกรมของมหาวิทยาลัยนอตติง-แฮม นอกจากทดสอบความถูกต้องแล้ว ระบบยังสามารถทดสอบประสิทธิภาพของโปรแกรมผู้เรียน ได้โดยการจับเวลาของแต่ละบรรทัด และจะนำเวลาของเส้นทางโปรแกรมที่ยาวที่สุดของผู้เรียนมาเปรียบเทียบกับโปรแกรมของผู้ออกโจทย์

2.2.1.3 **การตรวจทักษะการทดสอบโปรแกรมของผู้เรียน**

การทดสอบโปรแกรมเป็นทักษะที่จำเป็นต่อผู้เขียนโปรแกรม เนื่องจากซอฟต์แวร์ที่มีคุณภาพจะทำให้ลดค่าใช้จ่ายและเวลาลง ซึ่งการทดสอบโปรแกรมเป็นสิ่งที่ทำให้ซอฟต์แวร์มีคุณภาพมากขึ้น ถึงแม้ว่าเรามีระบบที่สามารถตรวจข้อสอบได้อย่างอัตโนมัติก็ตาม แต่ผู้เขียนโปรแกรมก็ไม่ควรละเลยการทดสอบโปรแกรม

การฝึกทักษะในการทดสอบโปรแกรมสามารถทำได้โดย การให้ผู้เรียนทำข้อมูลทดสอบเพื่อส่งมาในการตรวจ จากนั้นตัวตรวจโปรแกรมจะนำข้อมูลนั้นมาทดสอบกับโปรแกรมของผู้เรียนและโปรแกรมที่ถูกต้อง โดยในการทดสอบจะวัดว่าข้อมูลนั้นสามารถทำงานได้ครบทุกเส้นทางของโปรแกรมหรือไม่

Web-CAT [8] เป็นระบบที่ให้ความสำคัญกับการพัฒนาทักษะด้านการทดสอบโปรแกรมของผู้เรียน โดยมีการทดสอบโปรแกรมที่ถูกต้องกับข้อมูลทดสอบที่ผู้เรียนได้สร้างและส่งมาให้กับระบบ โดยระบบจะดูว่าข้อมูลทดสอบที่นักเรียนสร้างสามารถทดสอบได้ครบทุกเส้นทางของโปรแกรมที่ถูกต้องหรือไม่

Assyst สามารถวัดทักษะการทดสอบโปรแกรมของผู้เรียน โดยสั่งให้โปรแกรมทำงานกับข้อมูลที่ผู้เรียนสร้างขึ้น แล้วดูว่าการทำงานของโปรแกรมครอบคลุมทุกบรรทัดของโปรแกรมมากน้อยเพียงใด

2.2.1.4 **การตรวจเฉพาะทาง**

การตรวจดังข้อที่กล่าวมาแล้วข้างต้นเป็นการตรวจโดยทั่วไป ซึ่งมีความจำเป็นสำหรับการตรวจโปรแกรม ภาษาโปรแกรมแต่ละภาษานั้นมีลักษณะเฉพาะของแต่ละภาษาซึ่งยากสำหรับการเรียนรู้และยากสำหรับการตรวจด้วย เช่น การจัดการหน่วยความจำในภาษาซี ปัญหาของการจองหน่วยความจำแล้วไม่ได้คืนกลับเมื่อไม่ได้ใช้งาน (malloc แล้วไม่ได้ free) Tutnew [9] เป็นระบบที่มีการประเมินโปรแกรมของผู้เรียนในลักษณะนี้ ด้วยการดักและควบคุมการทำงานของตัวจัดการหน่วยความจำ ทำให้เราวัดการใช้งานของหน่วยความจำขณะทำงานได้

2.2.2 **การตรวจโดยการวิเคราะห์รหัสต้นฉบับ** (Static Assessment)

การตรวจแบบนี้อาศัยการวิเคราะห์รหัสต้นฉบับของโปรแกรมผู้เรียน เพื่อดูรูปแบบการเขียนคำสั่งว่าตรงตามธรรมเนียมปฏิบัติในวงการ (coding convention) โดยไม่ต้องสั่งให้โปรแกรมนั้นทำงาน ในอดีตการตรวจโปรแกรมเป็นการตรวจแบบวิเคราะห์รหัสต้นฉบับ เนื่องจากผู้เรียนจะส่งโปรแกรมที่เขียนเป็นกระดาษ จากนั้นผู้ออกโจทย์จะพิจารณาเพื่อให้คะแนน ซึ่งอาจมีมาตรฐานการให้คะแนนไม่ตรงกัน ปัจจุบันมีการใช้วิเคราะห์รหัสต้นฉบับเพื่อตรวจว่าโปรแกรมนั้นเขียนได้ดีอย่างไรก็ตามการตรวจแบบนี้เป็นการตรวจที่โครงสร้างของโปรแกรมแต่ละภาษา ความถูกต้องขึ้นอยู่กับไวยากรณ์ และความหมายของคำสั่งต่าง ๆ การตรวจลักษณะนี้ สามารถตรวจโปรแกรมของผู้เรียนได้ในหลายแง่มุมดังนี้

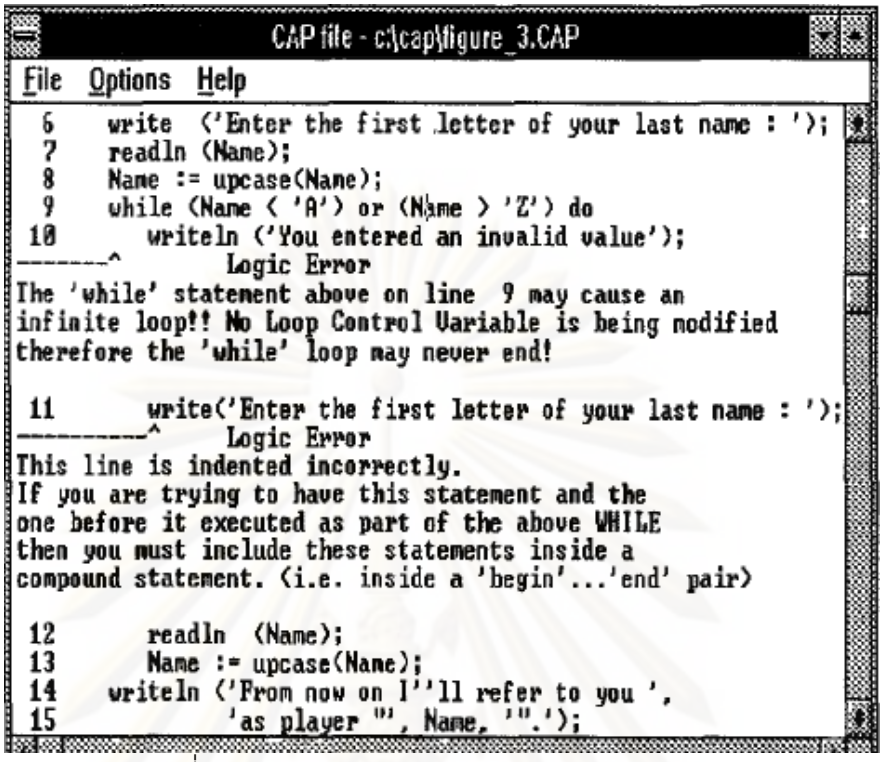
2.2.2.1 **รูปแบบการเขียนโปรแกรม** (Coding Style)

การตรวจแบบนี้เป็นการตรวจว่าผู้เรียนมีรูปแบบการเขียนที่ดีหรือไม่ เช่น การประกาศตัวแปรแล้วไม่ได้ใช้ การตั้งชื่อตัวแปรไม่ถูกต้อง เป็นต้น ปัจจุบันตัวแปลโปรแกรม(compiler) สามารถเตือนรูปแบบการเขียนโปรแกรมที่ไม่ถูกต้องเหล่านี้ได้ดีขึ้นเช่น จีซีซีสามารถแจ้งเรื่องตัวแปรที่ไม่ได้ใช้ การใช้ตัวแปรไม่ถูกชนิด และลักษณะของโปรแกรมที่ไม่เป็นไปตามมาตรฐานของภาษา จากคุณลักษณะข้างต้นทำให้การตรวจรูปแบบการเขียนทำได้ง่ายในภาษาซีพลัสพลัส รูปแบบการเขียนโปรแกรมเป็นสิ่งจำเป็นของนักเขียนโปรแกรม หากผู้เรียนเขียนโปรแกรมตามรูปแบบที่ดี จะทำให้โปรแกรมอ่านและบำรุงรักษาได้ง่าย [10]

2.2.2.2 **การตรวจการเขียนโปรแกรมผิดพลาด** (Programming Error)

โดยทั่วไป การตรวจความถูกต้องของการทำงานอาศัยการสั่งให้โปรแกรมทำงานจริงกับข้อมูลทดสอบ แต่ความผิดพลาดบางอย่างของโปรแกรมอาจต้องใช้การวิเคราะห์รหัสต้นฉบับ

CAP (Code Analyzer for Pascal) [11] เป็นระบบที่ใช้ตรวจโจทย์ในวิชาภาษาปาสกาล (pascal) โดยแสดงข้อความของข้อผิดพลาดและวิธีการแก้ปัญหา รวมถึงการแสดงตัวอย่างของรหัสที่ถูกต้อง CAP มีการตรวจ 3 อย่างคือ การตรวจเชิงวากยสัมพันธ์ การตรวจเชิงตรรกะ การตรวจเชิงรูปแบบ สำหรับการตรวจเชิงตรรกะนั้นจะตรวจว่ารหัสส่วนไหนที่อาจทำให้โปรแกรมผิดพลาด รูปที่ 2.2 แสดงผลการตรวจพบว่าส่วนของโปรแกรมอาจเกิดวงวนไม่รู้จบที่บรรทัดที่ 9



รูปที่ 2.2 การแสดงข้อผิดพลาดของระบบ CAP [11]

2.2.2.3 **การตรวจตัววัดซอฟต์แวร์** (Software Metrics)

Mengel and Ulans [12] ทำการวิจัยเกี่ยวกับตัววัดซอฟต์แวร์ เพื่อใช้กับโปรแกรมของผู้เรียน โดยงานวิจัยนี้ได้นำโปรแกรมของผู้เรียนจาก 3 แบบฝึกหัดเพื่อนำมาใช้ในการวัดผล ความต้องการของทั้ง 3 แบบฝึกหัดคือ ให้โปรแกรมมีขนาดเล็ก มีความซับซ้อนน้อย และมีสภาพเป็นส่วนจำเพาะสูง (high modularity) ซึ่งงานวิจัยนี้ได้ใช้ตัววัดในหลายรูปแบบ เช่น Mccabe Cyclomatic จำนวนฟังก์ชันและจำนวนข้อความสั่งในโปรแกรม จำนวนระดับการเรียกซ้อนในโปรแกรม จำนวนอักขระของหมายเหตุในหนึ่งข้อความสั่ง

2.2.2.4 **การตรวจการออกแบบ** (Design)

ผู้ออกโจทย์สามารถตรวจการออกแบบโปรแกรมของผู้เรียนได้โดยการตรวจว่าโปรแกรมของผู้เรียนมีความสอดคล้องกับโครงที่ผู้ออกโจทย์ได้กำหนดรหัสต้นฉบับบางส่วนไว้ให้

Roe and Bancorft [13] ได้พัฒนาตัวตรวจแบบวิเคราะห์รหัสต้นฉบับ โดยเป็นส่วนหนึ่งของระบบ ELP งานวิจัยนี้เน้นการแสดงผลป้อนกลับ โดยจะแสดงรูปแบบการเขียนโปรแกรมเพื่อให้ผู้เรียนพัฒนาการเขียนโปรแกรมได้ดีขึ้น งานวิจัยนี้แบ่งการวัดออกเป็น 2 ส่วน คือ การใช้ตัววัดทางวิศวกรรมซอฟต์แวร์ (software engineering metrics) ระบบมีการเตรียมชุดของฟังก์ชันเพื่อวัดคุณภาพของโปรแกรม เช่น ตัวแปรที่ไม่ได้ใช้ การนับจำนวนตัวแปร หรือข้อความสั่ง (statement)ทั้งหมด เป็นต้น การวัดอีกส่วนหนึ่งคือใช้การตรวจความเหมือนของโครงสร้างรหัส

ต้นฉบับโดยการแปลรหัสต้นฉบับเป็นเอกซ์เอ็มแอล และนำโครงสร้างของรหัสต้นฉบับที่ได้มาเปรียบเทียบกับของโปรแกรมที่ถูกต้อง

2.2.2.5 **การตรวจเฉพาะทาง** (Special Features)

มีความต้องการหลายอย่างในการวัดรหัสต้นฉบับแบบเฉพาะทาง ขึ้นอยู่กับความต้องการของผู้ออกโจทย์ เช่น ผู้ออกโจทย์ไม่อนุญาตให้ใช้ฟังก์ชัน sort แต่ให้เขียนการเรียงลำดับข้อมูลเอง ในกรณีเช่นนี้ระบบ Scheme-robo [14] ใช้วิธีการไม่อนุญาตให้ผู้เรียนใช้ หรือในกรณีการลอกโปรแกรมส่ง ซึ่งผู้เรียนทำสำเนาได้อย่างง่ายดาย หลาย ๆ ระบบ [4][15][22] ได้ทำการพัฒนาความสามารถของโปรแกรมเพื่อตรวจสอบการลอก

2.2.3 **การทดสอบความถูกต้องของโปรแกรม**

ในปัจจุบันอุตสาหกรรมซอฟต์แวร์ได้มีการทดสอบโปรแกรมหลายรูปแบบ เช่น การทดสอบเบ็ดเสร็จ (integration test) การทดสอบเพื่อยอมรับ (acceptance test) การทดสอบแบบหน่วย (unit testing) เป็นต้น เจยูนิต [16] เป็นซอฟต์แวร์ที่ได้รับความนิยมอย่างกว้างขวางในการทดสอบแบบหน่วยสำหรับภาษาจาวา โดยเจยูนิตในรุ่นที่ 4 นำ annotation ของจาวามาใช้ในส่วนการประกาศว่าเมท็อดนี้เป็นเมท็อดทดสอบ [17]

2.2.3.1 **การสร้างเมท็อดทดสอบ**

ผู้ใช้งานเจยูนิตต้องประกาศ `@Test` ไว้ที่เมท็อดเพื่อบอกว่าเมท็อดนั้นเป็นเมท็อดที่ต้องการทดสอบ เจยูนิตจะนำเมท็อดนี้ไปทำงาน ในหนึ่งคลาสมีเมท็อดทดสอบได้หลายเมท็อด โดยแต่ละเมท็อดไม่ต้องคืนค่า แต่ใช้การเรียกเมท็อดเพื่อทดสอบเงื่อนไขตามข้อกำหนดหากตรงตามเงื่อนไขก็ถือว่าผ่านการทดสอบ แต่หากไม่ตรงตามเงื่อนไขที่ผู้ใช้ระบุไว้ระบบจะแสดงข้อผิดพลาด และการทดสอบของเมท็อดนั้นก็จะไม่ผ่าน

```
import static org.junit.Assert.assertTrue;
import org.junit.Before;
import org.junit.Test;

public class TestArrayList {
    ArrayList lst;
    @Before
    public void setUp(){
        lst = new ArrayList(); // สร้างอ็อบเจกต์ ArrayList
    }
    @Test
    public void testAddList() {
        lst.add("test");
        lst.add("test2");
        assertTrue("Size not correct", lst.size() == 2); // ตรวจขนาดของ lst
    }
}
```

รหัสที่ 2.1 การทดสอบโดยใช้เจยูนิต

รหัสที่ 2.1 แสดงเมท็อดทดสอบ `testAddList` โดยมีการประกาศ `@Test` ไว้ที่หัวเมท็อดนั้น ภายในทำการเพิ่มข้อมูลในรายการ จากนั้นตรวจสอบว่าขนาดของรายการเท่ากับข้อมูลที่เพิ่มหรือไม่

2.2.3.2 การกำหนดเมท็อดเพื่อทำงานเริ่มต้นและก่อนสิ้นสุดของการทดสอบ

ผู้ใช้งานสามารถกำหนดค่าเริ่มต้น ทำการจัดการโครงแบบสภาพแวดล้อม (environment) หรือจัดการเกี่ยวกับลงบันทึกต่าง ๆ โดยการประกาศ `@Before` ไว้ที่เมท็อดโดยก่อนเริ่มทำงานเจยู-นิต จะเข้ามาทำเมท็อดนี้ก่อนการทำงานแต่ละเมท็อดทดสอบ โดยผู้ใช้สามารถกำหนดการทำงานเริ่มต้นได้หลายเมท็อดแต่จะไม่รับประกันลำดับของการทำงาน

```
@Before
protected void findTestDataDirectory() {
    inputDir = new File("data");
    inputDir = new File(inputDir, "xslt");
    inputDir = new File(inputDir, "input");
}
@Before
protected void redirectStderr() {
    System.setErr(new PrintStream(new ByteArrayOutputStream()));
}
```

รหัสที่ 2.2 การเรียกใช้ `@Before` ของเจยูนิต

การกำหนดเมท็อดเพื่อทำงานก่อนสิ้นสุดการทดสอบ จะเป็นการทำงานกลับกันกับการทำงานตอนเริ่มต้น เช่น ลบค่าที่กำหนดไว้ตอนเริ่มต้น หรือทำการกำหนดค่าว่างให้กับอ็อบเจกต์เพื่อจัดการกับหน่วยความจำ การกำหนดเมท็อดเพื่อทำงานก่อนสิ้นสุดทำได้โดยการกำหนด `@After` ไว้ที่เมท็อดดังนี้

```
@After
protected void disposeDocument() {
    doc = null;
    System.gc();
}
```

รหัสที่ 2.3 การกำหนดเมท็อดก่อนสิ้นสุด

2.2.3.3 การกำหนดเมท็อดเพื่อทำงานเริ่มต้นและก่อนสิ้นสุดระดับคลาส

การทำงานจะแตกต่างจากหัวข้อก่อนหน้านี้เนื่องจากเป็นการกำหนดเมท็อดก่อนเริ่มทำการทดสอบระดับคลาส เมท็อดนี้จะทำงานครั้งเดียวไม่ได้ทำงานทุกครั้งของเมท็อดการทดสอบ มีการนำคุณสมบัตินี้ไปใช้ในการติดต่อกับฐานข้อมูล การติดต่อกับเครือข่ายคอมพิวเตอร์ หรือการจัดการกับข้อมูลขนาดใหญ่ซึ่งจะใช้เวลามาก หากสร้างใหม่ทุกครั้ง จึงต้องสร้างครั้งแรกและให้ใช้ด้วยกัน ผู้ใช้สามารถทำได้โดยการใช้ `@BeforeClass` เพื่อเป็นการกำหนดเมท็อดก่อนการทดสอบระดับคลาส และใช้ `@AfterClass` เพื่อเป็นการกำหนดเมท็อดหลังการทำงานของการทดสอบระดับคลาส

```
@BeforeClass
protected void redirectStderr() {
    systemErr = System.err; // Hold on to the original value
    System.setErr(new PrintStream(new ByteArrayOutputStream()));
}
@AfterClass
protected void tearDown() {
    // restore the original value
    System.setErr(systemErr);
}
```

รหัสที่ 2.4 การกำหนดเมท็อดก่อนและหลังการทดสอบระดับคลาส

2.2.3.4 **การทดสอบชุดคำสั่งจัดการสิ่งผิดปรกติ** (Testing Exceptions)

การทดสอบชุดคำสั่งจัดการสิ่งผิดปรกติเป็นสิ่งหนึ่งที่ได้รับการพัฒนาในเจยูนิตรุ่นที่ 4 ในรุ่นเก่าจะมีการใช้บล็อก try เพื่อทดสอบดังรหัสที่ 2.5

```
public void testDivisionByZero() {
    try {
        int n = 2 / 0; // เพื่อให้เกิด exception
        fail("Divided by zero!");
    }
    catch (ArithmeticException success) {
        assertNotNull(success.getMessage());
    }
}
```

รหัสที่ 2.5 การทดสอบที่เกิดสิ่งผิดปรกติ

โดยในเจยูนิตรุ่นที่ 4 มีจะใช้การคาดการณ์ว่าจะเกิดข้อผิดปกติแบบนี้ขึ้น จะทำให้รหัสสั้นลง แต่หากทดสอบแล้วเกิดข้อผิดพลาดต่างจากที่คาดการณ์ไว้จะถือว่าไม่ผ่านการทดสอบ แต่ผู้เขียนตัวตรวจโปรแกรมสามารถใช้บล็อก `try` ได้ ถ้าต้องการทดสอบที่มีรายละเอียดมากกว่านี้ ผู้ใช้สามารถทดสอบโปรแกรมในลักษณะนี้ โดยการกำหนดค่าให้กับ expected ใน `@Test` แล้วใส่ชุดคำสั่งที่อาจเกิดสิ่งผิดปกติดังรหัสที่ 2.6 ตัวตรวจนี้ต้องเกิด exception `ArithmeticException` จึงจะถือว่าผ่านการทดสอบ หากไม่เกิด exception ดังที่ประกาศไว้ใน `@Test` จะถือว่าไม่ผ่านการทดสอบ

```
@Test(expected=ArithmeticException.class)
public void divideByZero() {
    int n = 2 / 0; // เกิด exception ทำให้เมท็อดนี้ผ่าน
}
```

รหัสที่ 2.6 การทดสอบโดยใช้การคาดกรณีสิ่งผิดปรกติของเจยูนิตรุ่นที่ 4

2.2.3.5 **การสั่งไม่ทดสอบในบางเมท็อด**

บางครั้งการทดสอบอาจใช้เวลานานมาก โดยรากฐานการทดสอบนั้นอาจมีความซับซ้อนมากหรืออาจทำงานได้ช้า หรืออาจเกิดข้อผิดพลาดจากการติดต่อกับเครื่องแม่ข่ายทำให้การทดสอบไม่สำเร็จ ผู้ใช้อาจไม่ต้องการทดสอบเมท็อดนั้นผู้ใช้สามารถทำได้โดยการเรียกใช้ `@Ignore` เพื่อไม่ต้องการให้ทดสอบในเมท็อดนั้น

2.2.3.6 เวลาของการทดสอบ

ผู้ใช้งานสามารถกำหนดเวลาให้กับการทดสอบได้โดยการกำหนดค่า `timeout` ที่มีอยู่ใน `@Test` โดยมีการกำหนดเวลาเป็นมิลลิวินาที หากตัวตรวจทำงานเกินจะถือว่าทดสอบไม่ผ่านดังรหัสที่ 2.7

```
@Test(timeout=2000)
public void remoteBaseRelativeResolutionWithDirectory()
    throws IOException, ParsingException {
        builder.build("http://www.ibiblio.org/xml");
}
```

รหัสที่ 2.7 การกำหนดค่าหมดเวลารอของตัวตรวจ

2.2.3.7 ข้อความยืนยัน

เจยูนิตรุ่นที่ 4 ได้มีการเพิ่มความสามารถในการทดสอบผลลัพธ์ในหลายรูปแบบ เช่น สามารถทดสอบแถวลำดับของข้อมูลดังรหัสด้านล่าง

```
public static void assertArrayEquals(Object[] expecteds, Object[] actuals)
public static void assertArrayEquals(String message, Object[] expecteds,
                                                        Object[] actuals)
```

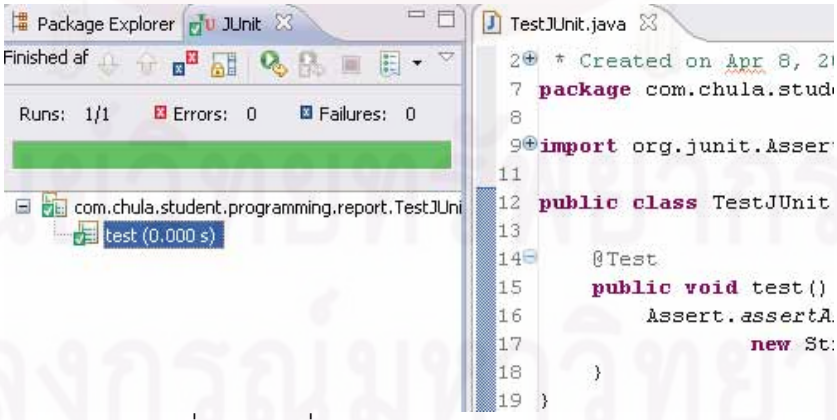
รหัสที่ 2.8 เมท็อดการเปรียบเทียบแถวลำดับของเจยูนิตรุ่นที่ 4

2.2.3.8 การสั่งดำเนินการ

ผู้ใช้สามารถสั่งดำเนินการตัวทดสอบหลังจากเขียนคำสั่งการทดสอบได้โดยใช้รายคำสั่งดังรูปที่ 2.3 หรือใช้โปรแกรมเสริมที่มีอยู่ในสิ่งแวดล้อมสำหรับการพัฒนาแบบเบ็ดเสร็จ (ไอดีอี) ดังรูปที่ 2.4

```
$ java -classpath .:junit.jar org.junit.runner.JUnitCore {class test}
  TestA TestB TestC...
JUnit version 4.5
Time: 0.003
OK (0 tests)
```

รูปที่ 2.3 การสั่งดำเนินการของตัวตรวจเจยูนิตโดยใช้รายคำสั่ง



รูปที่ 2.4 การสั่งดำเนินการเจยูนิตโดยผ่านไอดีอี

บทที่ 3

ตัวตรวจโปรแกรมอัตโนมัติ

งานวิจัยนี้ศึกษา ออกแบบ และพัฒนาระบบที่อำนวยความสะดวกให้ผู้ออกโจทย์ฝึกหัดการเขียนโปรแกรม สามารถพัฒนาตัวตรวจโปรแกรมตามข้อกำหนดของโจทย์ โดยอาศัยแนวคิดของการตรวจผลลัพธ์หลังจากสั่งโปรแกรมทำงานจริง ตัวตรวจและโจทย์ที่เขียนสามารถแนบกันไปเพื่อใช้กับซอฟต์แวร์ช่วยพัฒนาโปรแกรมทั่วไปได้เนื่องจากมีขนาดเล็ก

เนื้อหาในบทนี้จะประกอบไปด้วย การวิเคราะห์โจทย์ของการเขียนโปรแกรม ภาพรวมของระบบ คลาสอรรถประโยชน์ที่ใช้ในตัวตรวจ ตัวควบคุมการตรวจ ขั้นตอนการเขียนตัวตรวจ

3.1 การวิเคราะห์โจทย์ของการเขียนโปรแกรม

มหาวิทยาลัยต่าง ๆ มีการออกโจทย์ และนำใช้ในการเรียนการสอนของแต่ละวิชา เพื่อให้ผู้เรียนได้เข้ามาใช้งานโดยผู้เรียนสามารถเข้าดูโจทย์ผ่านทางโฮมเพจของแต่ละรายวิชา โดยงานวิจัยนี้ได้มีการศึกษาโจทย์ของมหาวิทยาลัยต่าง ๆ และนำมาวิเคราะห์เพื่อจะได้ออกแบบระบบให้สอดคล้องกับโจทย์โดยทั่วไป

งานวิจัยนี้ได้รวบรวมโจทย์ของมหาวิทยาลัยต่าง ๆ และทำการจัดกลุ่มโจทย์เหล่านั้น โดยจัดกลุ่มตามคุณลักษณะของตัวตรวจ รายวิชาที่ได้นำมาใช้ประกอบไปด้วย COS226 (Algorithms and Data Structures) 6 โจทย์ [18], CS1101 (Programming Methodology) 19 โจทย์ [19], CS107 (Intro to Programming) 11 โจทย์ [20] , 2110101 (การทำโปรแกรมคอมพิวเตอร์) 25 โจทย์ [21] และ JavaBat 13 โจทย์ [22] รวมแล้วมีโจทย์ทั้งหมด 74 โจทย์ หลังจากการรวบรวมโจทย์ของแต่ละมหาวิทยาลัยแล้ว ได้มีการนำข้อมูลไปวิเคราะห์ โดยได้มีการจัดกลุ่มโจทย์ตามคุณลักษณะ เช่น เมท็อดที่คืนค่าเป็นบูลีน การเปรียบผลแบบแม่นตรง การรับข้อมูลทางแป้นพิมพ์ เป็นต้น โดยโจทย์หนึ่งข้ออาจมีการใช้หลายคุณสมบัติได้ เช่น มีโจทย์ข้อหนึ่งของรายวิชา CS1101 ให้รับค่าละติจูดและค่าลองจิจูดแล้วให้โปรแกรมตรวจสอบว่าพิกัดที่รับมาอยู่ในประเทศสิงคโปร์หรือไม่ โดยโจทย์ข้อนี้ต้องมีการใช้ 3 คุณสมบัติคือ การรับข้อมูลทางแป้นพิมพ์ การสุ่มตัวเลขสองตัวเพื่อส่งให้โปรแกรม การอ่านค่าจากจอภาพ จากนั้นรวมคุณสมบัติของโจทย์แต่ละมหาวิทยาลัย คุณสมบัติที่มีการใช้กันมาก และใช้ทุกมหาวิทยาลัยคือการเปรียบเทียบผลแบบแม่นตรง โดยจะเห็นข้อมูลได้จากตารางที่ 3.1

	COS226	CS1101	JavaBat	2110101	CS107
การเปรียบผลแบบแม่นตรง	4	18	11	22	11
การตรวจเมท็อดที่คืนค่าเป็นบูลีน	3	1	2	2	
การตรวจสอบผลข้างเคียงจากการทำงาน	1			1	
การเปรียบเทียบค่าแบบประมาณ			1	1	1
การรับข้อมูลทางแป้นพิมพ์		14		12	
การตรวจผลลัพธ์ที่แสดงออกทางหน้าจอ		14		14	9
ลักษณะของข้อมูลขาเข้าแบบจำนวน		14	6	11	5
ลักษณะของข้อมูลขาเข้าแบบสายอักขระ	1	2	6	4	5
การอ่านและเขียนแฟ้มข้อมูล		1			

ตารางที่ 3.1 ข้อมูลโจทย์ของมหาวิทยาลัยต่าง ๆ

3.1.1 **การตรวจสอบผล**

การตรวจความถูกต้องเป็นสิ่งสำคัญของตัวตรวจโปรแกรม โจทย์จากการสำรวจเป็นโจทย์ที่ใช้การตรวจแบบสั่งให้ทำงานเพื่อตรวจสอบผล ดังนั้นการตรวจจึงใช้วิธีการใส่ข้อมูลขาเข้าแล้ววัดผลจากข้อมูลขาออก ซึ่งเกิดจากการทำงานของโปรแกรมของผู้เรียน โดยการเปรียบเทียบผลนั้นจะขึ้นอยู่กับลักษณะของโจทย์ และการให้คะแนนของผู้ออกโจทย์ สามารถแบ่งได้ 2 ประเภทคือ

3.1.1.1 **การเปรียบผลแบบแม่นตรง**

การเปรียบเทียบผลแบบแม่นตรงสามารถใช้ได้กับสายอักขระ ตัวเลข หรือแถวลำดับของข้อมูล เป็นการเปรียบเทียบผลลัพธ์ที่ได้จากการทำงานของโปรแกรมของผู้เรียนกับค่าที่ถูกต้อง โดยค่าทั้ง 2 ค่าต้องเป็นค่าที่เหมือนกันเท่านั้น จากตารางที่ 3.1 จะเห็นได้ว่าตัวตรวจส่วนมากมีการใช้คุณลักษณะนี้มากในทุกมหาวิทยาลัย การเปรียบเทียบผลแบบแม่นตรงนั้นยังรวมถึงการตรวจเมท็อดที่คืนค่าเป็นบูลีนอีกด้วย โดยโจทย์ที่พบบ่อยของวิชาโครงสร้างข้อมูลคือการตรวจสอบว่าในโครงสร้างข้อมูลมีข้อมูลหรือไม่ ดังนั้นผู้ออกโจทย์จึงมีความจำเป็นต้องออกโจทย์ที่มีการคืนค่าเป็นบูลีน หรือมีการออกโจทย์ที่ตรวจสอบข้อมูล เช่น มีโจทย์ข้อหนึ่งของวิชาการทำโปรแกรมคอมพิวเตอร์ให้รับค่าข้อมูลแถวลำดับแบบ 2 มิติ แล้วตรวจสอบว่าข้อมูลที่รับเข้ามาถูกต้องตามหลักการหาผลเฉลยของเกมซูโดกุ (sudoku) หรือไม่ ดังนั้นโจทย์จึงต้องการการคืนค่าเป็นบูลีน

```
public static boolean isSudoku(int[][] table) {
    // เขียน code เพื่อตรวจสอบ array table ถูกต้องตามกฎของ sudoku หรือไม่

    return true;
}
```

รหัสที่ 3.1 เมท็อดที่คืนค่าเป็นบูลีน

3.1.1.2 การเปรียบเทียบค่าแบบประมาณ

ในกรณีที่โจทย์ให้แสดงผลลัพธ์เป็นข้อความ มักพบเสมอว่าผู้เรียนอ่านข้อกำหนดของโจทย์ไม่ละเอียด ส่งผลให้เขียนคำสั่งที่ยังมีข้อผิดพลาดบ้าง เช่น สั่งให้แสดง Hello World ก็กลับเขียนคำสั่งให้แสดง hello world ซึ่งถือได้ว่าเกือบถูก ซึ่งหากจะไม่ให้คะแนนเลยก็จะไม่ถูกต้อง ดังนั้นตัวตรวจจึงมีความจำเป็นในการใช้เปรียบเทียบแบบประมาณของสายอักขระ เพื่อสามารถให้คะแนนกับผู้เรียนในกรณีดังกล่าว

สำหรับการเปรียบเทียบจำนวนจริงในคอมพิวเตอร์จะใช้การเปรียบเทียบแบบแม่นตรงไม่ได้ เนื่องจากคอมพิวเตอร์เก็บจำนวนจริงเป็นแบบประมาณ ดังนั้นหากเปรียบเทียบจำนวนจริงจึงต้องมีการใช้การเปรียบเทียบโดยใช้ค่าผิดพลาดสัมพัทธ์ (relative error) [23] โดยค่านี้จะเกิดจาก abs((result - expectedResult) / expectedResult) หากค่านี้น้อยมาก ผู้เรียนจะได้คะแนนมาก

3.1.2 การรับข้อมูลเข้า/ส่งผลออกของโปรแกรมผู้เรียน

ผู้ออกโจทย์สามารถกำหนดให้โปรแกรมของผู้เรียนรับข้อมูลเข้า/ส่งผลออกได้หลายวิธี เช่น การรับข้อมูลจากแป้นพิมพ์ หรือจากแฟ้มข้อมูล หรือผู้ออกโจทย์กำหนดไว้ในรหัสต้นฉบับ หากข้อมูลขาเข้ามีจำนวนมากและซับซ้อน นอกจากนี้ยังมีการรับเข้าส่งออกข้อมูลเป็นเมท็อดด้วย

3.1.2.1 การรับข้อมูลทางแป้นพิมพ์

การทำงานของโปรแกรมในวิชาการเขียนโปรแกรมเบื้องต้น จะเป็นการรับค่าจากแป้นพิมพ์และนำมาคำนวณค่า โดยเมื่อสั่งให้โปรแกรมดำเนินงานโปรแกรมจะหยุดรอเพื่อรับค่าจากผู้ใช้ผ่านทางแป้นพิมพ์ ผู้ออกโจทย์อาจออกแบบให้โจทย์รับข้อมูลจากแป้นพิมพ์หลายครั้งได้ ในปัจจุบันมีการใช้ `Scanner` เป็นตัวอ่านข้อมูลจากแป้นพิมพ์ และให้เป็นตัวอ่านจากแฟ้มข้อมูลได้ด้วย เช่น `Scanner(new File(file name))` จากรูปที่ 3.2 เป็นโปรแกรมรับคำนามเข้ามาแล้วเปลี่ยนคำนั้นเป็นพหูพจน์ โดยจะสร้าง `Scanner` โดยส่งพารามิเตอร์เป็น `System.in` เพื่อให้รับค่าข้อมูลจากแป้นพิมพ์ และรับค่าที่ป้อนผ่านทางแป้นพิมพ์โดยใช้ `kb.next()`และส่วนที่เปลี่ยนจากคำนามเป็นพหูพจน์จะเว้นช่องว่างเพื่อให้ผู้เรียนได้เขียน

```
import java.util.Scanner;

public class Plural {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in); // รับข้อมูลขาจากแป้นพิมพ์
        System.out.print("singular noun = ");
        String noun = kb.next(); // รับค่าจากแป้นพิมพ์แล้วกำหนดให้ noun

        System.out.println("noun+s = "); // สั่งพิมพ์คำตอบออกทางจอภาพ
  }
}
```

รหัสที่ 3.2 การรับข้อมูลจากแป้นพิมพ์โดยใช้ Scanner

3.1.2.2 **การตรวจผลลัพธ์ที่แสดงออกทางหน้าจอ**

โจทย์ส่วนมากมีข้อกำหนดให้แสดงผลลัพธ์ออกทางจอภาพโดยใช้ `System.out` หรือไม่ก็สร้างผลลัพธ์ลงแฟ้มข้อมูล ซึ่งต้องกำหนดรูปแบบการแสดงให้เด่นชัด สำหรับการเขียนเป็นเมท็อดก็ต้องกำหนดหัวเมท็อดให้ชัดเจน ดังรูปที่ 3.2 ผู้เรียนจะต้องพิมพ์คำว่า `noun+s =` ตามด้วยผลลัพธ์

3.1.3 **ลักษณะของข้อมูลขาเข้า**

ผู้ออกโจทย์ต้องการสร้างข้อมูลขาเข้าเพื่อใช้ทดสอบโปรแกรมของผู้เรียน ลักษณะของข้อมูลขาเข้าที่พบในโจทย์คือ จำนวนเต็มสุ่มและเต็มสุ่มในช่วง จำนวนจริงสุ่มและสุ่ม สายอักขระสุ่มไม่ซ้ำ แถวลำดับของจำนวนและสายอักขระแบบสุ่ม

3.1.4 **การตรวจสอบผลข้างเคียงจากการทำงาน**

สำหรับโจทย์ที่เกี่ยวกับเมท็อดของโครงสร้างข้อมูล บ่อยครั้งที่ตัวเมท็อดทำงานถูกต้อง แต่กลับทำเกินข้อกำหนดทำให้เกิดผลข้างเคียงต่อตัวข้อมูล เช่น โจทย์กำหนดให้เขียนเมท็อดหา และคืนข้อมูลที่มีค่าน้อยสุดเป็นอันดับสองที่เก็บในฮีปทวินาม (binary heap) หากผู้เรียนใช้วิธีเรียกเมท็อด `removeMin` หนึ่งครั้ง (เพื่อลบตัวน้อยสุดออกจากฮีป) ตามด้วยการเรียกเมท็อด `getMin` ก็จะได้ตัวน้อยสุดอันดับสองตามต้องการ แต่เป็นการทำงานที่ผิดเนื่องจากไปทำลายข้อมูลที่เก็บไว้

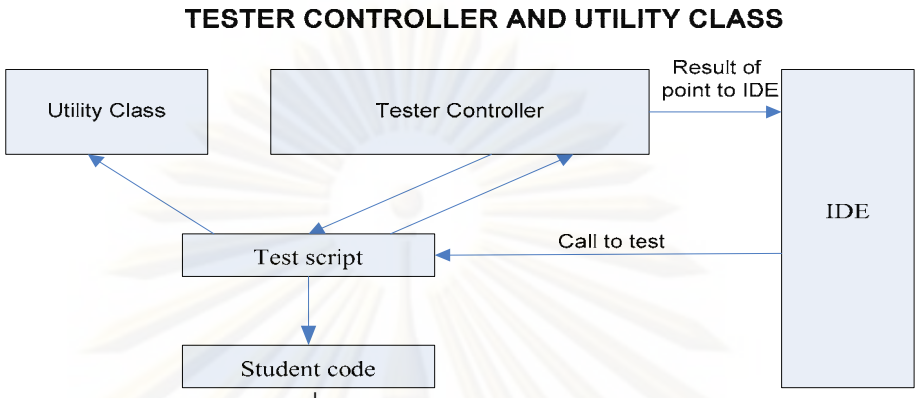
3.1.5 **คุณลักษณะอื่น**

บางครั้งผู้ออกโจทย์ต้องการออกโจทย์ที่มีคุณลักษณะพิเศษแล้วแต่ความต้องการของผู้ออกโจทย์ เช่น การห้ามใช้คลาสมาตรฐานของจาวา เป็นต้น

3.2 **ภาพรวมของระบบ**

ระบบการตรวจโปรแกรมอัตโนมัติแบ่งออกเป็น 4 ส่วนย่อย คือ รหัสต้นฉบับของผู้เรียน (student code) ตัวตรวจ (test script) ตัวควบคุมการตรวจ (test controller) และคลาส

อรรถประโยชน์เพื่อการตรวจ (utility class) รูปที่ 3.1 แสดงความสัมพันธ์ของแต่ละส่วนในระบบ งานวิจัยนี้เน้นการออกแบบและพัฒนาตัวควบคุมการตรวจและคลาสอรรถประโยชน์ เพื่ออำนวยความสะดวกในการพัฒนาตัวตรวจ

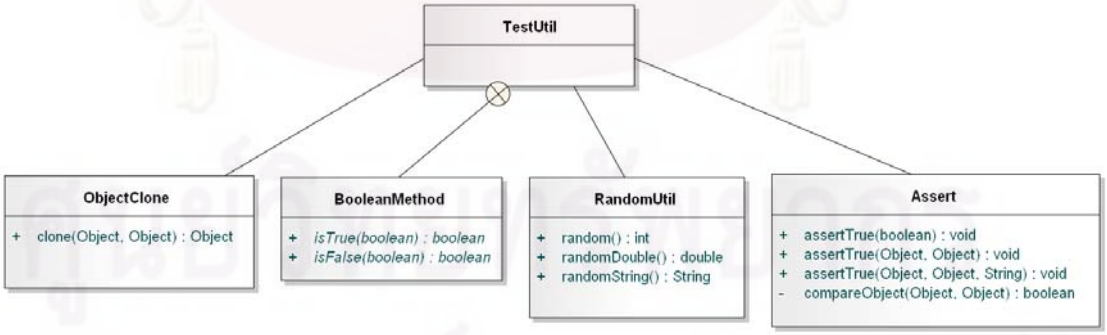


รูปที่ 3.1 ภาพโครงสร้างของระบบ

การตรวจโปรแกรมเริ่มด้วยการสั่งให้ตัวตรวจทำงาน โดยตัวตรวจจะไปเรียกตัวควบคุมการตรวจเพื่อให้อ่าน และทำงานตามที่ตัวตรวจเขียนกำกับไว้ด้วย annotation ที่เมท็อดการตรวจภายในตัวตรวจอาจเรียกใช้บริการต่าง ๆ ของคลาสอรรถประโยชน์ เมื่อตัวควบคุมทำงานเสร็จจะแสดงผลการตรวจให้ทราบ

3.3 คลาสอรรถประโยชน์ที่ใช้ในตัวตรวจ

คลาสอรรถประโยชน์มีบริการต่าง ๆ ที่อำนวยความสะดวกในการเขียนตัวตรวจ เช่น บริการสร้างข้อมูลทดสอบ บริการป้อนข้อมูลขาเข้าและอ่านผล บริการเปรียบเทียบผลลัพธ์ เป็นต้น งานวิจัยนี้ได้ทำการพัฒนาระบบซึ่งสามารถนำมาเขียนเป็นแผนภาพคลาสได้ดังรูปที่ 3.2



รูปที่ 3.2 แผนภาพคลาสของคลาสอรรถประโยชน์

แผนภาพคลาสประกอบไปด้วยคลาส `TestUtil` ซึ่งผู้ออกโจทย์จะเห็นคลาสนี้เป็นหลัก `TestUtil` มีการเรียกใช้งานคลาสอื่นเพื่อทำหน้าที่ในส่วนต่าง ๆ ดังนี้

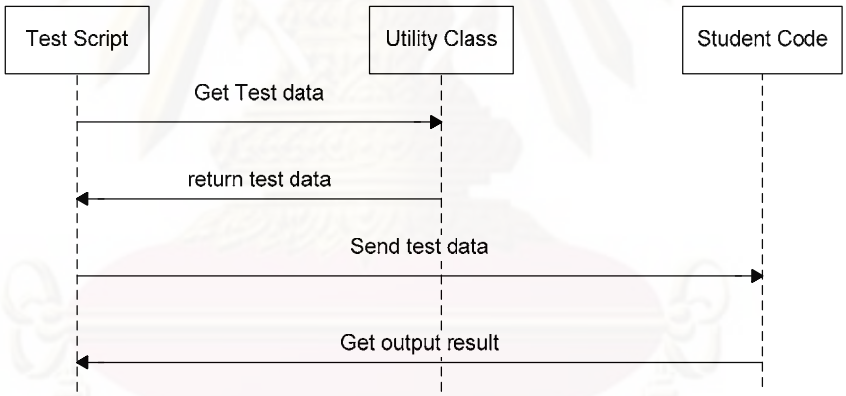
- `RandomUtil` ทำหน้าที่เพื่อสุ่ม ทั้งจำนวนและสายอักขระ

- `Assert` ทำหน้าที่เปรียบเทียบผลแบบต่างๆ
- `BooleanMethod` เป็นคลาสภายใน (inner class) ของ `TestUtil` ทำหน้าที่ตรวจเมท็อดที่คืนค่าเป็นบูลีน
- `ObjectClone` เพื่อทำสำเนาของอ็อบเจกต์

3.3.1 บริการการสร้างชุดข้อมูลทดสอบ

โจทย์ในวิชาการเขียนโปรแกรมทั่วไปมักมีการรับข้อมูลขาเข้าจากผู้ใช้ ตัวตรวจโปรแกรมจึงต้องสร้างข้อมูลทดสอบป้อนให้กับโปรแกรมผู้เรียน เพื่อทดสอบการทำงานในหลายลักษณะ

บริการสร้างข้อมูลทดสอบจึงเป็นสิ่งจำเป็นที่ระบบต้องให้บริการผ่านคลาสอรรถประโยชน์ โดยตัวตรวจเรียกใช้บริการสร้างข้อมูลทดสอบแบบต่าง ๆ จากคลาสอรรถประโยชน์จากนั้นจะส่งข้อมูลทดสอบที่ได้ให้กับโปรแกรมของผู้เรียนแล้วนำข้อมูลขาออกเพื่อมาตรวจว่าได้ผลที่ถูกต้องหรือไม่ ดังรูปที่ 3.3



รูปที่ 3.3 การเรียกคลาสอรรถประโยชน์เพื่อให้สร้างข้อมูลขาเข้า

บริการสร้างข้อมูลทดสอบมีหลายแบบเพื่อให้ผู้ออกโจทย์เรียกใช้ตามความเหมาะสมและความต้องการของโจทย์แต่ละข้อดังนี้

3.3.1.1 บริการสร้างจำนวนแบบสุ่ม

วิชาการเขียนโปรแกรมและวิชาโครงสร้างข้อมูลมีโจทย์จำนวนมากที่มีความต้องการใช้ข้อมูลทดสอบแบบจำนวน ทั้งจำนวนจริงและจำนวนเต็ม ดังนั้นคลาสอรรถประโยชน์จึงมีบริการซึ่งจะมีทั้งการคืนค่าเป็นจำนวนเต็มและจำนวนจริงแต่ในที่นี้ขอแสดงการคืนค่าเป็นจำนวนเต็มดังตารางที่ 3.2

`int random()`	
หน้าที่	คืนค่าของจำนวนเต็มแบบสุ่ม
ผลที่คืน	จำนวนเต็มแบบสุ่ม
`int random( int lower, int upper )`	
หน้าที่	คืนค่าของจำนวนเต็มแบบสุ่มมีค่าระหว่าง `lower` ถึง `upper`
พารามิเตอร์	`lower` - ค่าน้อยที่สุดที่ต้องการของการสุ่ม `upper` - ค่ามากที่สุดที่ต้องการของการสุ่ม
ผลที่คืน	จำนวนเต็ม
`int[] fillRandom(int [] d, int lower, int upper)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนเต็มแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยมีค่าระหว่าง `lower` ถึง `upper`
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า `lower` - ค่าน้อยที่สุดที่ต้องการของการสุ่ม `upper` - ค่ามากที่สุดที่ต้องการของการสุ่ม
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนจริงแบบสุ่ม
`int[] fillRandomDistinct(int [] d, int lower, int upper)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนเต็มแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยมีค่าระหว่าง `lower` ถึง `upper` โดยค่าจะไม่ซ้ำกัน
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า `lower` - ค่าน้อยที่สุดที่ต้องการของการสุ่ม `upper` - ค่ามากที่สุดที่ต้องการของการสุ่ม
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนเต็มแบบสุ่ม
`int[] fillRandomContain(int[] d, int [] contain)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนเต็มแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยค่าที่

	สุ่มคือค่าภายในแถวลำดับ `contain`
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า `contain` - แถวลำดับที่ต้องการให้สุ่มค่าเพื่อเติมลง `d`
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนเต็มแบบสุ่ม

ตารางที่ 3.2 ส่วนต่อประสานโปรแกรมประยุกต์คลาสอรรถประโยชน์ของจำนวน

3.3.1.2 **บริการสร้างสายอักขระแบบสุ่ม**

ตัวตรวจสามารถสร้างสายอักขระที่สุ่มโดยผู้ใช้สามารถระบุความยาว หรือให้ระบบสุ่มความยาวของสายอักขระให้ หรือสามารถสร้างแถวลำดับของสายอักขะได้ การสร้างสายอักขระแบบสุ่มนั้นทำได้ง่าย ๆ โดยการเรียกใช้บริการที่มีอยู่ในคลาสอรรถประโยชน์ซึ่งมีให้เรียกใช้ดังนี้

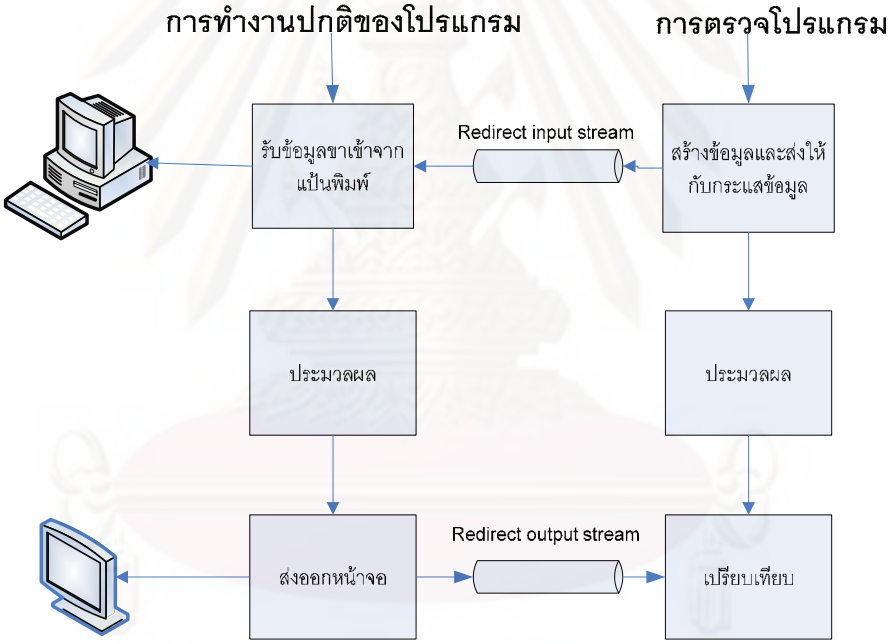
`String randomString()`	
หน้าที่	คืนค่าสายอักขระแบบสุ่ม โดยระบบจะมีการกำหนดความยาวของสายอักขระให้เอง
ผลที่คืน	สายอักขระแบบสุ่ม
`String randomString(int length)`	
หน้าที่	คืนค่าสายอักขระแบบสุ่ม ซึ่งมีความยาว `length`
พารามิเตอร์	`length` - ค่าความยาวของสายอักขระที่ต้องการให้สร้าง
ผลที่คืน	สายอักขระแบบสุ่มความยาว `length`
`String[] fillRandom(String[] d)`	
หน้าที่	คืนค่าแถวลำดับของสายอักขระแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d`
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า
ผลที่คืน	แถวลำดับ `d` ซึ่งภายในมีจำนวนเต็มสุ่ม
`String randomStringContain(char… c)`	
หน้าที่	คืนค่าสายอักขระแบบสุ่ม ซึ่งประกอบไปด้วยอักขระภายใน `c`
พารามิเตอร์	`c` - แถวลำดับของอักขระที่ต้องการให้สร้างเป็นสายอักขระ
ผลที่คืน	สายอักขระแบบสุ่ม

String radomStringDistinct()	
หน้าที่	คืนค่าสายอักขระแบบสุ่ม ซึ่งอักขระไม่ซ้ำกันเลย
ผลที่คืน	สายอักขระแบบสุ่ม

ตารางที่ 3.3 ส่วนต่อประสานโปรแกรมประยุกต์คลาสอรรถประโยชน์ของสายอักขระ

3.3.2 บริการป้อนข้อมูลให้กับแป้นพิมพ์และอ่านผลจากจอภาพ

ปกติการทำงานของโปรแกรมจะรับค่าจากการป้อนค่าผ่านทางแป้นพิมพ์ของผู้ใช้ ดังนั้น ตัวตรวจจะต้องเปลี่ยนทิศทางของสายข้อมูลจากการรับจากแป้นพิมพ์ ให้มารับจากตัวตรวจแทน ส่วนการแสดงออกทางหน้าจอนั้นก็เช่นกัน ตัวตรวจจะต้องเปลี่ยนทิศทางของข้อมูลซึ่งแสดงออกทางหน้าจอ ให้ตัวตรวจสามารถดึงข้อมูลที่แสดงออกทางหน้าจอเพื่อมาเปรียบเทียบกับข้อมูลที่ถูกต้องดังรูปที่ 3.4



รูปที่ 3.4 การรับข้อมูลเข้า/ส่งผลออก

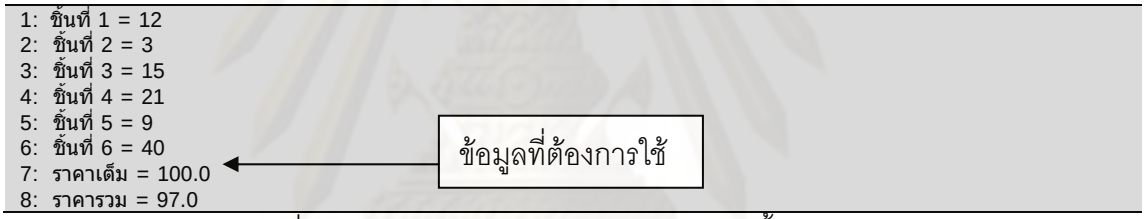
สำหรับการเรียกใช้บริการป้อนข้อมูลให้กับแป้นพิมพ์ และอ่านผลที่แสดงทางจอภาพของโปรแกรมผู้เรียน ตัวตรวจสามารถป้อนข้อมูลขาเข้าให้กับแป้นพิมพ์ของโปรแกรมผู้เรียน ได้โดยการเรียกใช้บริการ `TestUtil.writeSystemIn(Object)` โดยเมื่อตัวตรวจเรียก และส่งข้อมูลเข้าไปในเมท็อดจะเหมือนว่าโปรแกรมของผู้เรียนรับข้อมูลจากแป้นพิมพ์ จากนั้นผู้ออกโจทย์สามารถอ่านผลลัพธ์จากการทำงานของโปรแกรมผู้เรียน (หากเป็นการทำงานแบบปกติ จะต้องแสดงออกทางหน้าจอ) โดยเรียกเมท็อด `TestUtil.readSystemOut()`

```
@Test(loops = 10, points = 1, timeout=1000)
public double test() {
    // เติมจำนวนสุ่มตั้งแต่ 100 ถึง 1000 ใส่ในแถวลำดับ
    int[] in = TestUtil.fillRandom(new int[6], 100, 1000);
    TestUtil.writeSystemIn(in); // ส่งค่า in ให้เป็น input ของโปรแกรมของผู้เรียน
    Buy5Get1.main(new String[0]); // สั่งให้โปรแกรมของผู้เรียนทำงาน
    String[] line = TestUtil.readSystemOut(); // อ่านผลลัพธ์จากโปรแกรมของผู้เรียน
    ...
    return 1; // คืนสัดส่วนของคะแนนที่ได้ (กรณีนี้ได้เต็ม)
}
```

รหัสที่ 3.3 แสดงการป้อนข้อมูลให้แป้นพิมพ์

3.3.3 บริการคัดแยกข้อมูล

การตรวจผลที่แสดงทางจอภาพ มักสนใจเฉพาะข้อมูลบางส่วนที่แสดงเท่านั้น จึงต้องมีบริการคัดแยกข้อมูล จากรูปที่ 3.5 จะเห็นว่าข้อมูลที่แสดงออกทางจอภาพมีหลายบรรทัดแต่ข้อมูลที่ผู้ใช้ต้องการจะเริ่มตั้งแต่บรรทัดที่ 7 และอยู่หลังเครื่องหมายเท่ากับ ดังนั้นงานวิจัยนี้จึงได้สร้างบริการคัดแยกข้อมูลเพื่ออ่านข้อมูลจากจอภาพโดยสามารถเลือกได้ว่าจะเริ่มอ่านจากบรรทัดที่เท่าไรและตัวอักษรใดเป็นตัวแรก



รูปที่ 3.5 การแสดงออกทางจอภาพของโจทย์ซื้อ 5 แถม 1

การคัดแยกข้อมูลทำได้โดยเรียกใช้คลาสอรรถประโยชน์ `TestUtil.filter(lines, start_postion, start_string)` โดย `lines` เป็นแถวลำดับที่ต้องการทำการคัดแยกข้อมูล และ `start_position` เป็นตำแหน่งของข้อมูลในแถวลำดับ `lines` ที่ให้เริ่มต้นคัดแยกข้อมูล และ `start_string` เป็นสายอักขระภายในข้อมูลที่ให้เริ่มคัดแยกข้อมูล ดังรหัสที่ 3.4 ให้เริ่มคัดแยกข้อมูลตั้งแต่ตัวที่ 7 และเริ่มตั้งแต่คำว่า =

```
@Test(loops = 10, points = 1, timeout=1000)
public double test() {
    int[] in = TestUtil.fillRandom(new int[6], 100, 1000);
    TestUtil.writeSystemIn(in);
    Buy5Get1.main(new String[0]);
    String[] lines = TestUtil.readSystemOut();
    // คัดแยกข้อมูลโดยเลือกข้อมูลตั้งแต่ตัวที่ 7 โดยเริ่มจาก =
    String[] outResult = TestUtil.filter(lines,7,"= ");
    ...
    return 1;
}
```

รหัสที่ 3.4 การอ่านข้อมูลทางจอภาพโดยกำหนดบรรทัดและตัวอักษรที่ให้เริ่มอ่าน

3.3.4 บริการเปรียบเทียบผลลัพธ์แบบแม่นตรง

ปกติการเปรียบเทียบข้อมูลแบบแม่นตรงในจาวามี 2 แบบขึ้นอยู่กับชนิดของข้อมูลที่ทำการเปรียบเทียบ หากเป็นข้อมูลพื้นฐาน สามารถเปรียบเทียบค่าแบบแม่นตรงได้โดยการใช้เครื่องหมาย == แต่หากเป็นข้อมูลแบบอ็อบเจกต์จะต้องใช้เมท็อด `equals` ซึ่งเป็นบริการพื้นฐานของจาวาอยู่แล้ว แต่สำหรับการเปรียบเทียบแถวลำดับแบบเป็นช่วงจะเป็นสิ่งที่ผู้ออกโจทย์ต้องทำเอง ดังนั้นงานวิจัยนี้จึงได้พัฒนาบริการเพื่อเปรียบเทียบแถวลำดับแบบเป็นช่วง โดยตัวตรวจสามารถเรียกใช้บริการ `TestUtil.equals(Object [] a, int aStart, Object [] b , int bStart, int length)` `a` และ `b` เป็นอ็อบเจกต์ที่ต้องการเปรียบเทียบ `aStart` และ `bStart` เป็นตำแหน่งของแถวลำดับ และ `length` เป็นความยาวของแถวลำดับที่ต้องการเปรียบเทียบ

3.3.5 บริการเปรียบเทียบผลลัพธ์แบบประมาณ

ตัวตรวจสามารถเรียกใช้บริการเปรียบเทียบผลลัพธ์แบบประมาณของสายอักขระได้โดยเรียก `TestUtil.editDistance(source, dest)` ได้ผลคืนเป็นจำนวนจริงมีค่า 0 (เหมือนหมด) ถึง 1 (ต่างกันโดยสิ้นเชิง) การเปรียบเทียบแบบประมาณของสายอักขระแสดงได้ดังรหัสที่ 3.5

```
public static double editDistance( String source, String dest ) {
    int n = source.length(); // หาความยาวของสายอักขระ source
    int m = dest.length(); // หาความยาวของสายอักขระ dest
    int [][] d = new int[n + 1][m + 1]; // สร้างแถวลำดับเพื่อเก็บค่าจากการคำนวณ
    for ( i = 0; i <= n; i++ ) {
        d[i][0] = i; // กำหนดค่าต้นเริ่มแนวตั้งให้แถวลำดับ โดยเป็นค่าเรียงกัน
    }
    for ( j = 0; j <= m; j++ ) {
        d[0][j] = j; // กำหนดค่าต้นเริ่มแนวนอนให้แถวลำดับ โดยเป็นค่าเรียงกัน
    }
    for ( i = 1; i <= n; i++ ) {
        char s_i = source.charAt(i - 1);
        for ( j = 1; j <= m; j++ ) {
            char d_j = dest.charAt(j - 1);
            if ( s_i == d_j ) {
                cost = 0;
            } else {
                cost = 1;
            }
            // หาค่าน้อยที่สุดของ d[i-1][j]+1, d[i][j-1]+1 และ d[i-1][j-1]+cost
            d[i][j] = Math.min(Math.min(d[i - 1][j] + 1, d[i][j - 1] + 1),
                                        d[i - 1][j - 1] + cost);
        }
    }
    return d[n][m]; // คืนค่า edit distance
}
```

รหัสที่ 3.5 รหัสการเปรียบเทียบผลลัพธ์แบบประมาณของสายอักขระ

รหัสที่ 3.5 สร้าง `d` คือแถวลำดับ 2 มิติ ของจำนวนเต็มเพื่อเก็บการคำนวณโดยใช้หลักการของ Levenshtein distance[25] โดยให้ `n` เป็นความยาวของสายอักขระ `source` และ `m` เป็นความยาวของสายอักขระ `dest` และกำหนดค่าแบบเรียกลำดับให้กับแถวแนวนอนและแนวตั้งของ `d` และวงวนตามขนาดของ `n` และนำค่าของอักขระภายในสายอักขระ `source` ตำแหน่งที่ `i-1`

กำหนดให้ `s_i` และวงวนตามขนาดของ m และนำค่าของอักขระภายในสายอักขระ `dest` ตำแหน่งที่ `j-1` กำหนดให้ `d_j` แล้วเปรียบเทียบค่าของ `s_i` และ `d_j` หากเท่ากัน `cost` เป็น 0 หากไม่เท่ากัน `cost` เป็น 1 แล้วคำนวณหาค่าน้อยที่สุดจากค่า `d[i-1][j]+1, d[i][j-1]+1` และ `d[i-1][j-1]+cost` เมื่อคำนวณครอบทุกตัว คำตอบจะอยู่ที่ `d[n][m]`

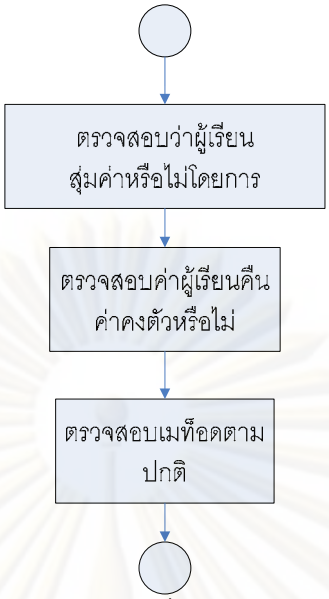
สำหรับการเปรียบเทียบจำนวนจริงควรใช้การเปรียบเทียบแบบประมาณ โดยผู้ออกโจทย์สามารถเรียกใช้บริการ `TestUtil.equals(src, dest, distance)` เพื่อให้คืนค่าจริงหากค่าสัมพัทธ์ของ `src` เทียบกับ `dest` แล้วต่างกันน้อยกว่าค่า `distance` และคืนค่าเท็จหากเปรียบกับแล้วค่าสัมพัทธ์มากกว่าค่า `distance`

3.3.6 บริการแสดงผลการตรวจและสาเหตุความผิดพลาด

การแสดงผลของการตรวจ เป็นการบอกคะแนนเมื่อผู้เรียนเรียกตรวจข้อนั้น ผู้ออกโจทย์สามารถช่วยผู้เรียนได้โดยการแสดงข้อผิดพลาดผ่านทางคลาสอรรถประโยชน์ เพื่อนำไปแสดงออกทางหน้าจอให้ผู้เรียนเห็น เพื่อให้ผู้เรียนได้แก้ไขข้อผิดพลาดของโปรแกรมเพื่อให้ถูกต้องในข้อนั้น การแสดงข้อผิดพลาดสามารถทำได้โดยใช้ `TestUtil.assertTrue(condition, “สาเหตุ”)` หากค่า `condition` เป็นค่าเท็จระบบจะนำข้อความ “สาเหตุ” ออกมาแสดงที่จอภาพ

3.3.7 บริการตรวจเมท็อดที่คืนผลแบบบูลีน

การออกโจทย์วิชาเขียนโปรแกรมและวิชาโครงสร้างข้อมูลนั้นมีโจทย์ที่คืนค่าเป็นบูลีน (ตารางที่ 3.1) เช่น การตรวจสอบว่าข้อมูลในกองซ้อนเป็นค่าว่างหรือไม่ ซึ่งเมท็อดที่คืนค่าเป็นบูลีนสามารถคืนค่าได้ 2 ค่า คือจริงหรือเท็จ หากผู้เรียนไม่สามารถทำโจทย์ข้อนี้ได้ ผู้เรียนอาจใช้วิธีกำหนดค่าคงตัว หรือสุ่มการคืนค่า ซึ่งหากตรวจแบบปกติผู้เรียนที่ทำพฤติกรรมเช่นนี้ อาจได้คะแนนบาง ดังนั้นงานวิจัยนี้จึงได้ตรวจสอบเมท็อดที่คืนค่าบูลีนโดย การตรวจสอบว่าผู้เรียนสุ่มการคืนค่าหรือไม่ โดยการส่งชุดข้อมูลทดสอบเดิมให้กับโปรแกรมของผู้เรียน แล้วตรวจผลลัพธ์จากการเรียกโปรแกรมของผู้เรียนว่าได้ค่าเดิมหรือไม่ หากไม่ได้ค่าเดิมแสดงว่าผู้เรียนสุ่มคำตอบ และตรวจสอบว่าผู้เรียนกำหนดค่าคงตัวของการคืนค่าหรือไม่ โดยสร้างข้อมูลใหม่ทุกครั้งที่เรียก และเรียกทั้งกรณีที่เป็นจริงและเท็จ หากได้ค่าผลลัพธ์จากการเรียกเป็นค่าเดิมแสดงว่าผู้เรียนกำหนดค่าคงตัวของการคืนค่า หากผ่าน 2 กรณีนี้ก็แสดงว่าผู้เรียนไม่ได้สุ่มการคืนค่าหรือการกำหนดค่าคงตัวของการคืนค่า จากนั้นจึงจะตรวจแบบปกติเพื่อให้คะแนน การทำงานนี้แสดงได้ดังรูปที่ 3.6



รูปที่ 3.6 การทำงานเพื่อทดสอบเมท็อดบูลีน

จากหลักการทำงานของการตรวจเมท็อดแบบบูลีน ผู้ออกโจทย์สามารถเขียนตัวตรวจ โดยงานวิจัยนี้ได้เตรียมคลาสอรรถประโยชน์สำหรับตรวจสอบบูลีนเมท็อด โดยอาศัยการสร้างอ็อบเจกต์แบบ `BooleanMethod` ซึ่งเป็นอินเทอร์เฟซที่บังคับให้เขียนเมท็อด `isTrue` และ `isFalse` เมท็อด `isTrue` เรียกเมท็อดแบบบูลีนที่คาดว่าต้องได้ผลเป็นจริง ในขณะที่ `isFalse` เรียกเมท็อดแบบบูลีนที่คาดว่าต้องได้ผลเป็นเท็จ รหัสที่ 3.6 แสดง `isTrue` ที่ภายในสร้างกองซ้อนว่าง แล้วเรียก `isEmpty` ส่วน `isFalse` สร้างกองซ้อนที่มีข้อมูลแล้วเรียก `isEmpty` ทั้งสองเมท็อดนี้รับพารามิเตอร์ `newData` ซึ่งถ้าเป็นจริง คือการระบุว่าต้องการให้สร้างชุดข้อมูลใหม่ก่อนเรียก หากค่าของ `newData` เป็นเท็จ คือการระบุให้ใช้ชุดข้อมูลทดสอบชุดที่ใช้ครั้งล่าสุด เพียงเท่านี้บริการตรวจเมท็อดที่คืนค่าเป็นบูลีน โดยเรียกใช้ `testBooleanMethod` จะทดสอบพฤติกรรมของเมท็อดแบบบูลีนด้วยการเรียก `isTrue` และ `isFalse` หลาย ๆ ครั้ง ก่อนจะทำการตรวจจริงเพื่อให้คะแนนความถูกต้อง

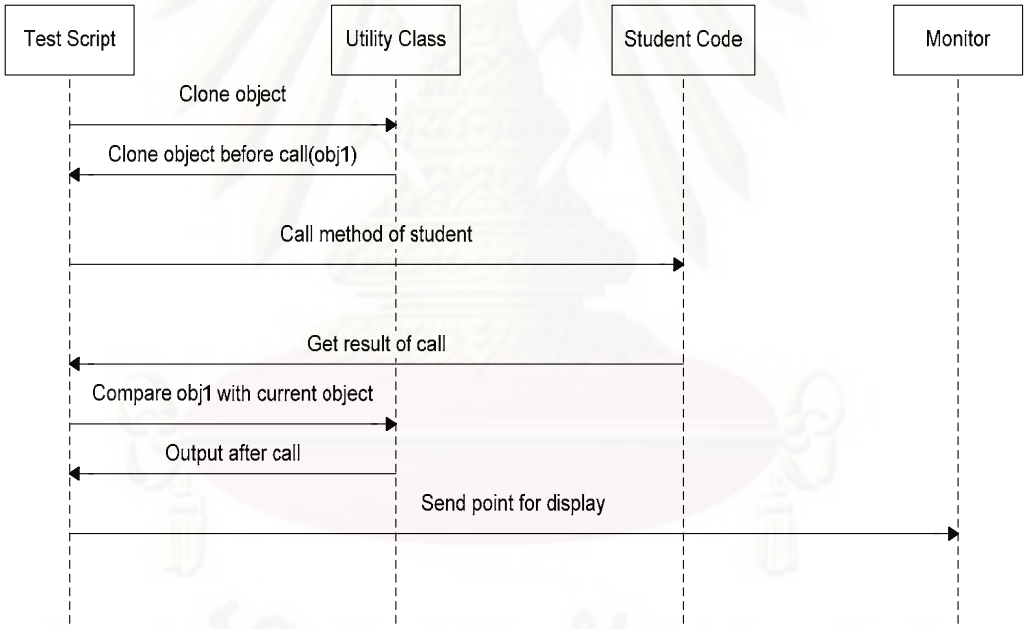
```
@Test(loops = 10, points = 1, timeout=1000)
public double testIsEmpty() {
    TestUtil.BooleanMethod bm = new TestUtil.BooleanMethod() {
        IntStack st, sf;
        public boolean isTrue(boolean newData) {
           if (newData) st = createRandomStack(0);
           return st.isEmpty(); // ต้องเป็น true;
        }
        public boolean isFalse(boolean newData) {
           int n = Math.random(20, 30);
           if (newData) st = createRandomStack(n);
           return sf.isEmpty(); // ต้องเป็น false
        }
    };
    double p = TestUtil.testBooleanMethod(bm);
    return TestUtil.assert(p==1, "ผิดบางกรณี", p);
}
```

รหัสที่ 3.6 ตัวอย่างตัวตรวจเมท็อดแบบบูลีน

3.3.8 บริการทำสำเนาอ็อบเจกต์และการตรวจสอบการเปลี่ยนสถานะของอ็อบเจกต์

ปกติการทำสำเนาอ็อบเจกต์ในจาวาทำได้โดย การสร้างเมท็อด `clone` การทำสำเนาตัวสร้าง (copy constructor) การเขียนอ็อบเจกต์โดยใช้ Serializable [27] แต่ไม่ใช่ทุกอ็อบเจกต์จะมีบริการดังกล่าว จึงมีบริการทำสำเนาอ็อบเจกต์ด้วยการใช้รีเฟลคชันของจาวา[29] โดยทำสำเนาของอ็อบเจกต์ทีละค่าในทุก ๆ ระดับจนถึงคลาสมาตรฐานของจาวา

รูปที่ 3.7 แสดงการทำงานของการตรวจสอบการไม่เปลี่ยนแปลงสถานะของอ็อบเจกต์ โดยการทำสำเนาของอ็อบเจกต์ก่อนการเรียกเมท็อดของผู้เรียน จากนั้นเรียกเมท็อดของผู้เรียน หลังจากการเรียกเมท็อดของผู้เรียน ตัวตรวจจะนำอ็อบเจกต์ที่ทำสำเนาไว้ก่อนการเรียกกับอ็อบเจกต์หลังการเรียกเมท็อดของผู้เรียนมาเปรียบเทียบกันว่าทุกค่าเหมือนกันหรือไม่หากค่าของอ็อบเจกต์ก่อนเรียกและหลังจากการเรียกไม่เหมือนกัน แสดงว่าอ็อบเจกต์เปลี่ยนแปลงสถานะ



รูปที่ 3.7 การตรวจสอบเปลี่ยนแปลงสถานะของอ็อบเจกต์

สำหรับการทำสำเนาอ็อบเจกต์โดยใช้รีเฟลคชันสามารถทำได้ดังรหัสที่ 3.7

```
public static <T> T clone(T obj) throws IllegalArgumentException,
                    IllegalAccessException, InstantiationException {
    Class objClass = obj.getClass();
    T result;
    if (objClass.isArray()) {
        int length = Array.getLength(obj);
        Class componentType = objClass.getComponentType();
        result = (T) Array.newInstance(componentType, length);
        for (int i = 0; i < length; ++i) {
            Object slot = Array.get(obj, i);
            if (slot != null) {
                Object slotClone = clone(slot);
                Array.set(result, i, slotClone);
            }
        }
    } else if (objClass.getName().indexOf("java.") != -1) {
        if (obj instanceof Cloneable){
            try {
                Method clone = obj.getClass().getMethod("clone");
                return (T) clone.invoke(obj);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return obj;
    } else {
        result = (T) obj.getClass().newInstance();
        for (Class c = objClass; c != Object.class; c = c.getSuperclass()) {
            Field[] f = c.getDeclaredFields();
            for (Field field : f) {
                field.setAccessible(true);
                Object value = field.get(obj);
                value = clone(value);
                field.set(result, value);
            }
        }
    }
    return result;
}
```

รหัสที่ 3.7 การทำสำเนาอ็อบเจกต์โดยใช้รีเฟลคชัน

รหัสที่ 3.7 เริ่มจากการรับอ็อบเจกต์ที่ต้องการทำสำเนาเข้ามา และตรวจสอบว่าอ็อบเจกต์นั้นเป็นแถวลำดับหรือไม่โดยใช้ `objClass.isArray()` หากเป็นจริงจะสร้างอ็อบเจกต์ใหม่เป็นแถวลำดับโดยสามารถเรียกใช้ `Array.newInstance(componentType, length)` โดยอ็อบเจกต์ `componentType` ได้จากการเรียก `objClass.getComponentType` และความยาวของแถวลำดับได้จากการเรียก `Array.getLength(obj)` จากนั้นวงวนเพื่อนำค่าจากต้นฉบับของอ็อบเจกต์แต่ละตัวโดยใช้การเรียกซ้ำที่เมท็อด `clone` เพื่อให้ลงไปถึงอ็อบเจกต์มาตรฐานของจาวา จากนั้นทำการกำหนดค่าของอ็อบเจกต์นั้นให้กับอ็อบเจกต์ใหม่โดยใช้ `Array.set`

หากอ็อบเจกต์ที่รับเข้ามาไม่ใช้แถวลำดับจะตรวจสอบว่าเป็นคลาสมาตรฐานของจาวาหรือไม่ โดยตรวจสอบชื่อของอ็อบเจกต์ ว่ามี `java.` ซึ่งหากมีก็แสดงว่าเป็นคลาสมาตรฐานของจา-

วา จากนั้นตรวจสอบว่าอ็อบเจกต์นั้นมีเมท็อด `clone` ให้เรียกใช้หรือไม่ หากมีก็เรียกเมท็อด clone ของอ็อบเจกต์นั้น หากไม่มีก็สามารถคืนค่าของอ็อบเจกต์นั้นได้เลย

หากอ็อบเจกต์ที่รับเข้ามาไม่ใช้แถวลำดับหรืออ็อบเจกต์มาตรฐานของจาวาก็สามารถสร้างอ็อบเจกต์ใหม่โดยใช้ `obj.getClass().newInstance()` จากนั้นวงวนจนถึง `Object.class` และนำค่าของข้อมูลทุกตัวในคลาสนั้นมากำหนดให้อ็อบเจกต์ใหม่เพื่อคืนค่า จะต้องมีการเรียก `field.setAccessible(true)` เพื่อให้สามารถเข้าถึงข้อมูล `private` ได้ จากนั้นจะนำค่าจากอ็อบเจกต์ต้นฉบับโดยใช้ `field.get(obj)` แล้วกำหนดให้อ็อบเจกต์ `value` และเรียกเมท็อด clone ซ้ำ เพื่อกำหนดค่าให้กับอ็อบเจกต์ใหม่โดยใช้ `field.set(result, value)`

นอกจากทำสำเนาของอ็อบเจกต์แล้ว คลาสอรรถประโยชน์ยังสามารถตรวจสอบสถานะของอ็อบเจกต์โดยใช้รีเฟลคชัน ตรวจสอบทุกค่าภายในอ็อบเจกต์ว่าเท่ากันหรือไม่ ดังรหัสที่ 3.8

```
01:public static boolean compareObject(Object src, Object dest) {
02:    Class srcClass = src.getClass();
03:    Class destClass = dest.getClass();
04:    if(srcClass.isArray()) {
05:        if(srcClass.getComponentType().isPrimitive()){
06:            if (src instanceof byte[] && dest instanceof byte[])
07:                return Arrays.equals((byte[]) src, (byte[]) dest);
08:            else if (src instanceof short[] && dest instanceof short[])
09:                return Arrays.equals((short[]) src, (short[]) dest);
10:            else if (src instanceof int[] && dest instanceof int[])
11:                return Arrays.equals((int[]) src, (int[]) dest);
12:                //check condition all primitive type
13:        } else {
14:            int length = Array.getLength(src);
15:            for (int i = 0; i < length; i++) {
16:                Object srcValue = Array.get(src, i);
17:                Object destValue = Array.get(dest, i);
18:                if (!compareObject(srcValue, destValue))
19:                    return false;
20:            }
21:        }
22:    } else if (srcClass.getName().indexOf("java.") != -1) {
23:        return src.equals(dest);
24:    } else {
25:        for(Class c = srcClass;c != Object.class; c = c.getSuperclass(),
26:                                  destClass = destClass.getSuperclass()) {
27:            Field[] fSrc = c.getDeclaredFields();
28:            Field[] fDest = destClass.getDeclaredFields();
29:            for(int i =0; i<fSrc.length; i++) {
30:                Field fieldSrc = fSrc[i];
31:                Field fieldDest = fDest[i];
32:                fieldSrc.setAccessible(true);
33:                fieldDest.setAccessible(true);
34:                if(!compareObject(fieldSrc.get(src),fieldDest.get(dest)))
35:                    return false;
36:            }
37:    }
38:    return true;
39:}
```

รหัสที่ 3.8 การตรวจสอบสถานะการเปลี่ยนแปลงของอ็อบเจกต์

รหัสที่ 3.8 การทำสำเนาของอ็อบเจกต์ เริ่มด้วยการตรวจสอบว่าอ็อบเจกต์ `src` เป็นแถวลำดับหรือไม่โดยใช้ `srcClass.isArray()` จากนั้นตรวจสอบว่าข้อมูลถายในแถวลำดับเป็นข้อมูล

พื้นฐานหรือไม่ โดยเรียก `srcClass.getComponentType().isPrimitive()` หากเป็นข้อมูลพื้น สามารถใช้ `Arrays.equals` แต่ต้องตรวจสอบว่าเป็นแถวลำดับของข้อมูลชนิดใดก่อนจึงจะสามารถส่งข้อมูลได้ถูกต้อง จะต้องตรวจสอบข้อมูลพื้นฐานทุกประเภท แต่หากข้อมูลในแถวลำดับเป็นอ็อบเจกต์จะต่อดึงค่าข้อมูลแต่ละตัวเพื่อทำการเปรียบเทียบโดยการใช้รีเฟลคชัน เริ่มจากการหาความยาวของแถวลำดับและนำข้อมูลในแถวลำดับแต่ละตัวของ `src` และ `dest` (บรรทัดที่ 16 และ 17) เพื่อมาตรวจว่าเท่ากันหรือไม่โดยจะเรียกเรียกซ้ำเมท็อด `compareObject` (บรรทัดที่ 18) หากค่าที่คืนมาเป็นค่าเท็จแสดงว่ามีค่าภายในอ็อบเจกต์ที่ไม่เท่ากัน

หากอ็อบเจกต์ที่รับเข้ามาไม่ใช้แถวลำดับจะตรวจสอบว่าเป็นคลาสมาตรฐานของจาวาหรือไม่ (บรรทัดที่ 22) หากเป็นให้คืนค่า `src.equals(dest)`ได้เลย

หาก `src` ไม่ใช้แถวลำดับหรืออ็อบเจกต์มาตรฐานของจาวา จะมาเริ่มทำงานในบรรทัดที่ 24 คือการวงวนเพื่อนำข้อมูลของ `src` และ `dest` มาเปรียบเทียบ โดยจะเรียก `compareObject` ซ้ำ เนื่องจากภายในอ็อบเจกต์อาจมีข้อมูลที่เป็นแถวลำดับหรืออ็อบเจกต์ จึงเรียกซ้ำจนถึงอ็อบเจกต์-มาตรฐานของจาวา

เมท็อดนี้สามารถใช้ได้กับข้อมูลพื้นฐานเนื่องจากจาวาสามารถเปลี่ยนข้อมูลพื้นฐานเป็นข้อมูลแบบอ็อบเจกต์อัตโนมัติ (คุณสมบัติ Autoboxing [30])

3.3.9 บริการอ่านค่าและเปลี่ยนค่าภายในอ็อบเจกต์

ในกรณีที่ต้องตรวจหลายเมท็อดที่มีการทำงานที่พึ่งกันและกัน เช่น โจทย์ของการพัฒนาคลาส `StudentArrayCollection` ที่มีทั้ง การเพิ่ม การลบ การค้นหาข้อมูล ดังรหัสที่ 3.9

```
public class StudentArrayCollection {
    private int size; // เก็บจำนวนข้อมูล
    private Object[] elementData; // อาเรย์ที่เก็บข้อมูล
    public ArrayCollection() { ... }
    // เพิ่ม e เข้าในที่เก็บ e ห้ามเป็น null
    public void add(Object e) { ... }
    // ลบ e ออกจากที่เก็บหนึ่งตัว ถ้าไม่มี e ก็ไม่ต้องทำอะไร
    public void remove(Object e) { ... }
    // คืนจำนวนข้อมูลที่เก็บ
    public int size() { ... }
    // คืน true ถ้าไม่มีข้อมูลเก็บเลย ไม่เช่นนั้นคืนเท็จ
    public boolean isEmpty() { ... }
    // ค้นว่ามี e เก็บอยู่ในที่เก็บนี่หรือไม่
    public boolean contains(Object e) { ... }
}
```

รหัสที่ 3.9 โครงของคลาส `StudentArrayCollection`

เห็นได้ว่าเมท็อดต่าง ๆ จะต้องพึ่งเมท็อด `add` เช่น เราไม่สามารถลบข้อมูลหรือตรวจสอบขนาดของรายการได้หากเรายังไม่ได้เพิ่มข้อมูล ดังนั้นจะไม่สามารถตรวจเมท็อดอื่นได้เลยหากผู้เรียนทำเมท็อด `add` ไม่เสร็จ ซึ่งการมีข้อบังคับให้ผู้เรียนเขียนเมท็อดใดก่อน และเขียนเมท็อดใดหลังนั้นเป็นสิ่งไม่เหมาะสม หรือผู้เรียนบางคนอาจทำได้บางเมท็อด เช่น ทำได้เฉพาะเมท็อดหา

ขนาดของข้อมูลในรายการ แต่ไม่สามารถเขียนเมท็อดการเพิ่มรายการได้ ผู้เรียนก็ควรได้คะแนนตามส่วน เพื่อให้สามารถตรวจเมท็อดแบบไม่ขึ้นต่อกัน ตัวตรวจต้องสามารถเปลี่ยนค่าภายในอ็อบเจกต์ได้ เช่น หากตัวตรวจต้องการตรวจเมท็อดหาขนาดของข้อมูลในรายการ ตัวตรวจ ต้องตั้งค่าในอ็อบเจกต์ก่อนแล้วเรียกเมท็อดเพื่อหาขนาดข้อมูลในรายการ เพื่อดูว่าตรงกับข้อมูลที่ผู้ออกโจทย์กำหนดให้หรือไม่ ซึ่งจะเห็นได้ว่าผู้เรียนไม่จำเป็นต้องเขียนเมท็อดเพื่อเพิ่มรายการให้เสร็จก่อน แต่ผู้เรียนสามารถเขียนเมท็อดหาขนาดของข้อมูลในรายการได้เลย การตั้งค่าภายในอ็อบเจกต์กระทำได้โดยใช้รีเฟลคชันที่มีอยู่ในจาวา

การอ่านค่าและเปลี่ยนค่าภายในอ็อบเจกต์ ทำได้โดยเรียก `TestUtil.setField` (การที่ต้องทำผ่านเมท็อดนี้เนื่องจาก หากข้อมูลที่ต้องการกำหนดค่าเป็น `private` ทำให้ไม่สามารถกำหนดค่าได้) ซึ่งเป็นเมท็อดของคลาสอรรถประโยชน์ จากตัวอย่างของตัวตรวจที่แสดงในรหัสที่ 3.10 เป็นการตรวจเมท็อด `isEmpty` ของรหัสที่ 3.9 ตัวตรวจเป็นการตรวจเมท็อดบูลีน ที่เมท็อด `isFalse` เมื่อ `newData` เป็นจริงจะสร้างข้อมูลใหม่ โดยเรียกเมท็อด `createArray` เริ่มโดยการสร้างแถวลำดับแบบสุ่ม `data` และนำแถวลำดับนั้นมากำหนดให้อ็อบเจกต์ `studentArray` โดยใช้ `TestUtil.setField` เพื่อกำหนดค่าให้กับข้อมูล `elementData` และ `size` ดังรหัสที่ 3.10

```
TestUtil.BooleanMethod pm = new TestUtil.BooleanMethod(){
    StudentArrayCollection studentArray;

    @Override
    public boolean isFalse(boolean newData) {
        if (newData)
            studentArray = createArray(); // สร้างข้อมูลใหม่
        return studentArray.isEmpty();
    }

    private StudentArrayCollection createArray() {
        int n = TestUtil.random(10, 100);
        String[] data = TestUtil.fillRandom(new String[n]);
        StudentArrayCollection studentArray = new StudentArrayCollection();
        // กำหนดค่าให้ field elementData และ size ของ StudentArrayCollection
        TestUtil.setField(studentArray, "elementData", data);
        TestUtil.setField(studentArray, "size", data.length);
        return studentArray;
    }
    ...
};
```

รหัสที่ 3.10 การเปลี่ยนค่าภายในอ็อบเจกต์

3.3.10 การแสดงผลการทำงานเทียบกับผลที่ถูกต้อง

การแสดงผลป้อนกลับให้กับผู้เรียนนั้นจะทำให้ผู้เรียนเข้าใจข้อผิดพลาดของโปรแกรมและสามารถพัฒนาโปรแกรมได้ดีขึ้น ดังนั้นการจัดการเรื่องผลป้อนกลับที่ดีจึงเป็นเรื่องที่จำเป็นต่อผู้เรียน ปกติผลการป้อนกลับสามารถมีได้หลายรูปแบบ เช่น ผู้ออกโจทย์แสดงสาเหตุของข้อผิดพลาดให้ผู้เรียนเห็น การแสดงผลการทำงานเทียบกับผลที่ถูกต้อง เป็นต้น

ผู้ออกโจทย์สามารถแสดงผลการทำงานกับผลที่ถูกต้องให้กับผู้เรียนผ่านทางจอภาพด้วย `TestUtil.assertTrue` โดยระบุค่าที่ต้องการแสดงให้กับพารามิเตอร์ `out` และ `ans` ของบริการนี้ จากนั้นตัวควบคุมการตรวจจะนำเอาข้อมูลขาออกที่ได้จากการทำงานของโปรแกรมของผู้เรียนและข้อมูลที่ถูกต้องมาแสดงออกทางจอภาพ ดังตัวอย่างในรหัสที่ 3.11

```
@Test(loops = 10, points = 1, timeout=1000)
public double testHello() {
    HelloWorld.main(new String[0]);
    String[] out = TestUtil.readSystemOut();
    String[] ans = {"Hello World"};
    double d = TestUtil.editDistance(ans, out);
    return TestUtil.assertTrue(d==0,"ไม่ตรงข้อกำหนด", 1-d, ans, out);
}
```

รหัสที่ 3.11 ตัวตรวจที่แสดงข้อมูลขาออกและข้อมูลที่ถูกต้องออกทางจอภาพ

จากรหัสที่ 3.11 จะเห็นได้ว่าตัวตรวจจะสุ่มค่าและส่งให้กับโปรแกรมของผู้เรียนจากนั้นตัวตรวจจะรอรับผลจากโปรแกรมของผู้เรียน และตัวตรวจคำนวณค่าที่ถูกต้อง ตัวตรวจส่งผลการทำงานของผู้เรียนคือ `out` และผลที่ถูกต้องที่ได้จากการคำนวณของผู้ออกโจทย์คือ ans ออกทางหน้าจอเพื่อให้ผู้เรียนเห็นดังรูปที่ 3.8

```
testHello : x
ข้อมูลขาเข้า=HELLO world  ข้อมูลที่ถูกต้อง=Hello World (result x)
Your point : 0.00 ( Total 10.0 )

***************
```

รูปที่ 3.8 แสดงข้อมูลขาออกเทียบกับข้อมูลที่ถูกต้อง

จากรูปที่ 3.8 ให้ผู้เรียนแสดงคำว่า HELLO world ออกทางหน้าจอ แต่ผู้เรียนแสดงคำว่า Hello World การที่ระบบแสดงผลเปรียบเทียบทั้งสองกรณี จะทำให้ผู้เรียนเข้าใจมากยิ่งขึ้นว่าทำไมข้อนี้ถึงผิด

3.3.11 การวัดเวลา

วัตถุประสงค์หลักของการตรวจโปรแกรมคือการตรวจความถูกต้อง ของผลการทำงาน นอกจากนั้น ผู้ออกโจทย์อาจสนใจการตรวจเรื่องเวลาการทำงาน โดยวิธีการเทียบเวลาโปรแกรมผู้เรียนกับโปรแกรมผู้ออกโจทย์ การที่ไม่กำหนดเป็นค่าคงตัวของเวลาเนื่องจากสภาพแวดล้อมของเครื่องไม่เหมือนกัน เครื่องที่มีประสิทธิภาพดีจะทำให้โปรแกรมที่เขียนทำงานได้เร็ว การวัดเวลาของคลาสอรรถประโยชน์ทำได้โดยเรียก `testTime.startTime()` เพื่อเป็นการเริ่มจับเวลา และเรียก `testTime.getTimeUsage()` เพื่อจะได้คืนค่าเวลาของการทำงานดังรหัส 3.12

```
@Test(loops = 10, points = 1, timeout=1000)
public double testTime() {
    int[] in = TestUtil.fillRandom(new int[6], 100, 1000);
    TestUtil.writeSystemIn(in);

    TestTime testTime = new TestTime(); // สร้างอ็อบเจกต์ TestTime
    testTime.startTime(); // เริ่มจับเวลาของโปรแกรมผู้เรียน
    Buy5Get1.main(new String[0]); // สั่งให้โปรแกรมของผู้เรียนทำงาน
    long studentTimeUsage = testTime.getTimeUsage();// เวลาการทำงานของโปรแกรมผู้เรียน

    String[] line = TestUtil.readSystemOut(); // ได้ผลจาการทำงานของโปรแกรมผู้เรียน
    String[] outResult = TestUtil.filter(line,6,"= ");
    double[] out = TestUtil.string2double(outResult); // เปลี่ยนสายอักขระเป็นจำนวนจริง
    int sum = 0;
    int min = in[0];

    testTime.startTime(); // เริ่มจับเวลาการทำงานของโปรแกรมหาผลเฉลย
    for (int intData : in) {
        sum += intData;
            if (intData < min)
                min = intData;
    }
    long solutionTimeUsage = testTime.getTimeUsage();// เวลาการทำงานโปรแกรมผลเฉลย
    double ans = sum-min;
    TestUtil.assertTrue(out[1] == ans); // เปรียบเทียบคำตอบ
    // ผลต่างเวลาโปรแกรมของผู้เรียนกับโปรแกรมผลเฉลย
    long diffTime =  studentTimeUsage - solutionTimeUsage;
    double extraPoint = 0;
    if (diffTime < 1000 && diffTime > 9999) {
        extraPoint = 0.5;
    } else if (diffTime < 10000 && diffTime > 100000) {
        extraPoint = 0.2;
    }
    return 1+extraPoint; // คืนค่าคะแนนบวกกับคะแนนเพิ่มหากโปรแกรมของผู้เรียนทำเวลาได้เร็ว
}
```

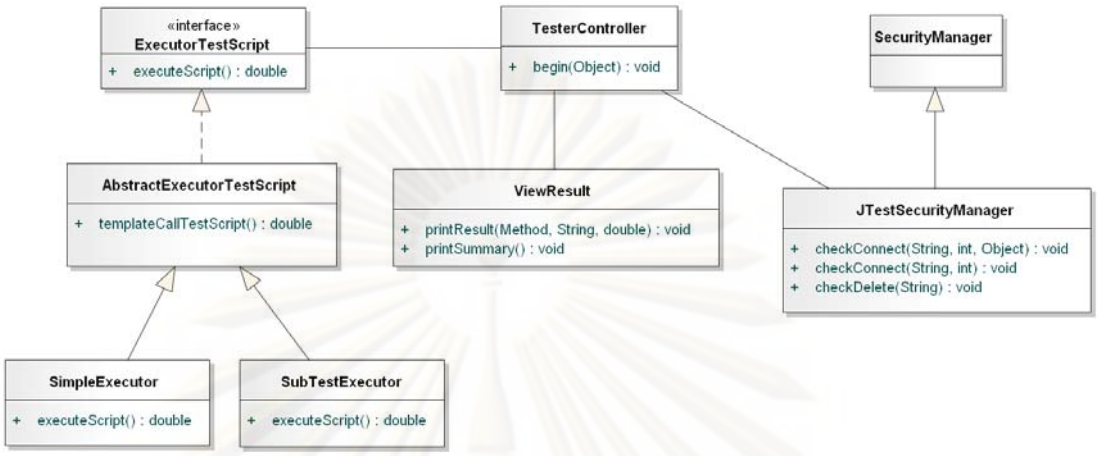
รหัสที่ 3.12 การตรวจเวลาของโจทย์ซื้อ 5 แถม 1

จากรหัสที่ 3.12 สุ่มแถวลำดับและส่งให้โปรแกรมของผู้เรียน จากนั้นทำการสร้างอ็อบเจกต์ `testTime` และเรียก `testTime.startTime` ก่อนเรียกเมท็อดของผู้เรียนหลังจากเรียกเมท็อดของผู้เรียนเสร็จจึงเรียก `testTime.getTimeUsage` เพื่อให้ได้เวลาในการทำงานของเมท็อดผู้เรียนและกำหนดค่าให้ตัวแปร `studentTimeUsage` จากนั้นทำงานของตัวตรวจตามปกติ เช่น อ่านผลจากจอภาพ คัดแยกข้อมูล เป็นต้น ก่อนเริ่มคำนวณคำตอบให้เรียก `testTime.startTime` เพื่อจับเวลาในการทำงานโปรแกรมหาผลเฉลยของตัวตรวจ แล้วจึงให้ตัวตรวจหาผลเฉลย เรียก `testTime.getTimeUsage` เมื่อคำนวณหาผลเฉลยเสร็จ โดยจะกำหนดค่าเวลาการทำงานของตัวตรวจให้ตัวแปร `solutionTimeUsage` จากนั้นนำผลต่างของเวลาเพื่อให้คะแนน

3.4 **ตัวควบคุมการตรวจ** (Tester Controller)

ตัวควบคุมการตรวจเป็นส่วนสำคัญในการทำงานของตัวตรวจโปรแกรม โดยตัวควบคุมการตรวจจะทำหน้าที่อ่าน annotation จากตัวตรวจ เพื่อทำงานตามที่ตัวตรวจได้ประกาศไว้ นอกจากนั้นตัวควบคุมการตรวจยังทำหน้าที่รวบรวมคะแนนแต่ละครั้งและควบคุมสภาพแวดล้อม

และความมั่นคงให้การทำงานของตัวตรวจเพื่อให้ตัวตรวจทำงานได้อย่างถูกต้อง ตัวควบคุมการตรวจสามารถแสดงเป็นแผนภาพคลาสได้ดังรูปที่ 3.8



รูปที่ 3.8 แผนภาพคลาสของตัวควบคุมการตรวจ

จากรูปที่ 3.8 แสดงแผนภาพคลาสของตัวควบคุมการตรวจซึ่งประกอบไปด้วย

- `TestController` เป็นคลาสหลักที่ทำหน้าที่เรียกคลาสต่าง ๆ เพื่อทำงาน
- `JTestSecurityManager` ทำหน้าที่ควบคุมความมั่นคง
- `ExecutorTestScript` เป็นคลาสที่ไปสั่งดำเนินการเมท็อดการตรวจ โดยจะมีคลาสลูกเพื่อดำเนินการตัวตรวจทั่วไป และตัวตรวจที่มีกรณีย่อย
- `ViewResult` ทำหน้าที่แสดงผลลัพธ์ออกทางจอภาพ

การตรวจโปรแกรมดำเนินการภายใต้การทำงานของตัวควบคุมการตรวจซึ่งสั่งให้เริ่มทำงานได้ด้วยคำสั่ง

```
TesterController.begin( testerObject )
```

อ็อบเจกต์ที่ส่งให้ `begin` นี้มีเมท็อดตรวจ (ซึ่งเป็นเมท็อดที่ถูกกำกับด้วย `@Test`) จากนั้นเป็นหน้าที่ของตัวควบคุมการตรวจที่จะวกกลับมาเรียก เมท็อดตรวจต่าง ๆ ในตัวตรวจ การทำงานของตัวควบคุมการตรวจสามารถแสดงได้ดังรหัสที่ 3.13

```
01:static void begin(Tester tester) {
02:    setSecurityManagerEnabled(true);
03:    redirectSystemInOut(true);
04:    List<Method> testMethods = getTestMethods(tester);
05:    for(Method method : testMethods) {
06:        double point = 0;
07:        Test annotation = method.getAnnotation(Test.class);
08:        List msg = new ArrayList();
09:        hasException = false;
10:        for (int i = 0; i < annotation.loops(); i++) {
11:            Thread thread = new Thread() {
12:             public void run() {
13:                ExecutorService service = Executors.newCachedThreadPool();
14:                Callable<Object> callable = new Callable<Object>() {
15:                    public Object call() throws Exception {
16:                        return annotation.points() * method.invoke();
17:                    }
18:                };
19:                Future<Object> result = service.submit(callable);
20:                try {
21:                     boolean terminated = service.awaitTermination
22:                             (annotation.timeout(),TimeUnit.MILLISECONDS);
23:                     if (!terminated)
24:                          service.shutdownNow();
25:                     point += result.get(0, TimeUnit.MILLISECONDS);
26:                } catch (TimeoutException e) {
27:                     e.printStackTrace();
28:                     msg.add("** time out **");
29:                     hasException = true;
30:                } catch (Exception e) {
31:                     e.printStackTrace();
32:                     msg.add(getStackTraceString(e));
33:                     hasException = true;
34:                }
35:             }
36:            };
37:            thread.start();
38:            if (hasException)
39:               break;
40:     }
41:     printResult(method, msg, point);
42:  }
43: redirectSystemInOut(false);
44: printSummary();
45: setSecurityManagerEnabled(false);
46:}
```

รหัสที่ 3.13 การทำงานของตัวควบคุมการครวจ

จากรหัสที่ 3.13 สามารถแสดงขั้นตอนการทำงานดังนี้

- **การรักษาความมั่นคง**

`setSecurityManagerEnabled` (บรรทัดที่ 2) ทำหน้าที่เพื่อจัดการความมั่นคงระบบ โดยป้องกันไม่ให้โปรแกรมที่ถูกตรวจทำงานที่ไม่ได้รับอนุญาต เมท็อดนี้อาศัย SecurityManager ของจาวาในการจัดการ เช่น การห้ามไม่ให้เลิกการทำงานโดยใช้ `System.exit()` (เพราะถ้าให้ทำทุกอย่างรวมทั้งตัวควบคุมการตรวจจะเลิกทำงานหมด) สามารถกระทำได้ด้วยคำสั่งที่เขียนเป็นตัวอย่างดังรหัสที่ 3.14

```
System.setSecurityManager(new SecurityManager()
{
    public void checkExit(int status) {
        if (!systemExitEnabled)
            throw new SecurityException("can't exit");
    }
});
```

รหัสที่ 3.14 การรักษาความมั่นคง

โดยปกติตัวควบคุมการตรวจจะห้ามทุกอย่าง เช่น ห้ามอ่าน เขียน ลบแฟ้มข้อมูล และห้ามเชื่อมต่อกับเครือข่าย เป็นต้น เนื่องจากโปรแกรมส่วนมากไม่ยอมให้ทำงานเหล่านั้น แต่หากโจทย์บางประเภทมีความจำเป็นต้องการให้เปิดความปลอดภัย เช่น โจทย์ให้เขียนโปรแกรมอ่านค่าจากเว็บไซต์เพื่อนับคำในหน้านั้น ผู้ออกโจทย์สามารถเปิดความปลอดภัยเพิ่มให้กับโปรแกรมของผู้เรียนโดยใช้ `openTestscriptSecurity` และกำหนดแถวลำดับของความปลอดภัยที่โปรแกรมผู้เรียนทำได้ ดังรหัสที่ 3.15

```
@Test(openTestscriptSecurity={TestScriptSecurityManager.connection})
public double test() {
    ...
    return 1;
}
```

รหัสที่ 3.15 การเปิดให้ตัวตรวจสามารถอ่านค่าจากเว็บไซต์ได้

- **การเปลี่ยนเส้นทางของข้อมูล**

`redirectSystemInOut` (บรรทัดที่ 3) เป็นการเปลี่ยน `System.in` เพื่อให้รับข้อมูลจาก `QueuedInputStream` และเปลี่ยน `System.out` ให้แสดงผลทาง `QueuedOutputStream` เพื่อให้สามารถตรวจโปรแกรมที่รับข้อมูลทางแป้นพิมพ์และแสดงผลทางจอภาพได้

- **การใช้รีเฟลคชันของจาวาในตัวควบคุมการตรวจ**

`getTestMethods` (บรรทัดที่ 4) หาเมท็อดในตัวตรวจที่มี `@Test` กำกับ ซึ่งอาศัยรีเฟลคชันของจาวา ดังแสดงตัวอย่างคร่าว ๆ ดังรหัสที่ 3.16

```
Class c = tester.getClass();
for( Method m : c.getDeclaredMethods()) {
    if (m.isAnnotationPresent(Test.class)) {
        ...
    }
}
```

รหัสที่ 3.16 การใช้รีเฟลคชันในตัวควบคุมการตรวจ

- **การวงวนเพื่อสั่งเมท็อดการตรวจทำงาน**

วงวนใหญ่ที่บรรทัดที่ 5 นำเมท็อดตรวจแต่ละเมท็อดมาสั่งทำงาน โดยหยิบลักษณะกำกับการตรวจที่เขียนใน `@Test` ออกมาด้วย `getAnnotation` (บรรทัดที่ 7)

วงวนภายในที่บรรทัดที่ 10 มีไว้วนตรวจด้วยเมท็อดตรวจเดียวกันซ้ำ ๆ (แต่ใช้ข้อมูลทดสอบไม่เหมือนกัน) ตามจำนวน `loops` ครั้งตามที่กำหนดใน `@Test` ของเมท็อด

- **การสั่งทำงานของหน่วยประมวลผลย่อย**

บรรทัดที่ 11 ถึง 36 สร้างหน่วยประมวลย่อยด้วย Thread เพื่อ ให้ตัวตรวจทำงานอยู่คนละหน่วยกับตัวควบคุมการตรวจ หลังจากสั่งตัวตรวจเริ่มทำงานโดยเรียก `start` (บรรทัดที่ 37) ตัวควบคุมการตรวจจะสร้าง Thread pool โดยใช้ `Executors.newCachedThreadPool()` ซึ่งจาวาจะกำหนดขนาดของ pool ให้ จากนั้นสร้างอ็อบเจกต์ `Callable` เพื่อเรียกเมท็อดการตรวจ โดยจะมีการเรียกเมท็อดของผู้เรียน (บรรทัดที่ 16) และจะนำผลลัพธ์จากการเรียก คูณกับคะแนนเต็มที่ประกาศไว้ใน `@Test` (เพื่อให้ตัวตรวจคืนค่าเป็น 1 หากให้คะแนนเต็มและคืนค่าน้อยกว่า 1 หากต้องการให้คะแนนตามส่วน เพื่อหากแก้คะแนนเต็มที่ `@Test` ก็ไม่จำเป็นต้องแก้ตัวตรวจ) ในเมท็อด `call` จากนั้นเรียก `service.submit` เพื่อส่ง `callable` (บรรทัดที่ 19) และกำหนดให้ `result` จากนั้นหยุดรอการทำงานจากการเรียกโปรแกรม เป็นเวลาตามที่ประกาศใน `@Test` โดยใช้ `service.awaitTermination` หาก thread ยังไม่หยุดทำงาน (`terminated` เป็นเท็จ) ให้หยุดทำงานทันทีโดยใช้ `service.shutdownNow()`แต่หากทำงานไม่เกินเวลาจะนำค่าคะแนนที่ได้จากการเรียกตัวตรวจมากำหนดให้ `point` เมื่อตัวตรวจทำงานครบทุกกรณีจะรายงานผลลัพธ์จากการทำงาน

- **ลักษณะกำกับการตรวจ**

ลักษณะกำกับการตรวจเป็น annotation ที่กำหนดไว้ที่เมท็อดการตรวจเช่น

```
@Test(loops=10, points=1, timeout=1000)
```

คือการให้เมท็อดตรวจ ตรวจ 10 รอบ (`loops=10`) รอบละหนึ่งคะแนน (`points=1`) และแต่ละรอบห้ามใช้เวลาเกิน 1000 มิลลิวินาที (`timeout=1000`) หากต้องมีการตรวจหลายกรณี ก็สามารถเขียนเมท็อดตรวจที่มี `@Test` กำกับเช่นนี้เพิ่มได้ โดยหน้าที่ซึ่งผู้ออกโจทย์สามารถทำได้มีดังต่อไปนี้

1) <u>ชื่อของการตรวจ</u> (name)

ผู้ออกโจทย์ตรวจสามารถกำหนดชื่อของการตรวจได้เองโดยตัวควบคุมจะนำชื่อของการตรวจไปแสดง เช่น `@Test(name="testAdd")` หากผู้ออกโจทย์ไม่ได้ระบุ ตัวควบคุมการตรวจจะใช้ชื่อของเมท็อดแทน และตัวควบคุมการตรวจจะนำชื่อนี้แสดงออกทางจอภาพ

2) <u>จำนวนรอบที่ทำการตรวจ</u> (loops)

ผู้ออกโจทย์สามารถกำหนดให้ตัวตรวจกระทำการตรวจหลาย ๆ รอบ เพื่อทดสอบการทำงานด้วยข้อมูลทดสอบหลาย ๆ แบบ เช่น `@Test(loops=10)` เป็นการกำหนดให้ทำงานเมท็อดนี้ 10 ครั้ง

3) คะแนนเต็มต่อการตรวจหนึ่งรอบ (points)

ผู้ออกโจทย์สามารถกำหนดคะแนนเต็มของแต่ละรอบการตรวจ เช่น `@Test(points=10)` แทนคะแนนเต็มต่อรอบจะเป็น 10 คะแนน

4) วิธีคำนวณคะแนน (CalculationType)

ในกรณีที่มีการตรวจหลายรอบ เรากำหนดให้ต้องมีวิธีคำนวณคะแนนรวมดังนี้

- คะแนนเฉลี่ยโดยใช้ `@Test(calcType=CalculationType.AVERAGE)`
- คะแนนรวม `@Test(calcType=CalculationType.TOTAL)`
- คะแนนมากที่สุด `@Test(calcType=CalculationType.MAX)`

การเลือกวิธีในการคำนวณคะแนนของแต่ละรอบ ขึ้นอยู่กับผู้ออกโจทย์เห็นเหมาะสม เช่น `@Test(loops=10,points=10,calcType=CalculationType.AVERAGE)` ซึ่งหมายความว่ามีการทำงาน 10 รอบ โดยหาค่าเฉลี่ยของคะแนนทั้ง 10 รอบเพื่อเป็นคะแนนจริง ๆ ที่ผู้เรียนจะได้รับ

5) เวลามากสุดที่อนุญาตให้ทำงานต่อการตรวจหนึ่งรอบ (timeout)

เราสามารถกำหนดเวลามากสุดของการทำงานของโปรแกรมผู้เรียน โดยจะสั่งให้โปรแกรมของผู้เรียนหยุดทำงานหากทำงานเกินเวลาที่กำหนดไว้ เช่น `@Test(timeout=1000)` คือเวลามากสุดที่อนุญาตให้โปรแกรมของผู้เรียนทำงานเป็น 1000 มิลลิวินาที

6) จำนวนครั้งที่อนุญาตให้ผู้เรียนเรียกตรวจ

ในบางครั้งผู้ออกโจทย์ต้องการให้คะแนนเต็มส่วนกับผู้ที่เรียกตรวจน้อยครั้งและมีการหักคะแนนกับผู้ที่เรียกตรวจบ่อยครั้ง เพื่อให้ผู้เรียนมีความละเอียดรอบคอบ เช่น `@Test(loops = 10, points = 10, reducePointAfterTest=5, reducePoint=10)` นั้นหมายความว่า หากผู้เรียนเรียกเกิน 5 ครั้งจะหักคะแนน 10% จากคะแนนที่ผู้เรียนสามารถทำได้ โดยจะเป็นการหักคะแนนเพียงครั้งเดียว

7) แฟ้มข้อมูลทดสอบ (testData)

ผู้ออกโจทย์สามารถเขียนรายการของข้อมูลทดสอบ และผลลัพธ์ลงในแฟ้มข้อมูลทดสอบ เพื่อให้ตัวตรวจอ่านข้อมูลจากแฟ้มนี้ส่งไปทดสอบโปรแกรมของผู้เรียนแล้วนำค่าผลที่ได้มาเปรียบเทียบกับผลลัพธ์ที่ถูกต้องที่ได้จากแฟ้มข้อมูลทดสอบ ทำให้ผู้ออกโจทย์ไม่ต้องเขียนรหัสในการทำงานเพื่อหาผลลัพธ์เหล่านั้น

การระบุแฟ้มข้อมูลทดสอบทำได้โดยการระบุตำแหน่งของแฟ้มข้อมูลไว้ที่ลักษณะกำกับการตรวจ ดังรหัสที่ 3.17 หรือผู้ออกโจทย์เรียกโดยใช้ `TestUtil.getVectorInput(fileName)` ก็ได้

```
@Test(loops=1, points=10, testData="test.dat")
public double mirrorEnds(){
    // ค่าข้อมูลขาเข้าจากแฟ้ม test.dat
    List<String> vector = TestUtil.getVectorInput();
    // ค่าผลลัพธ์จากแฟ้ม test.dat
    List<String> result = TestUtil.getVectorResult();
    for (int i=0; i<vector.size(); i++) {
        TestUtil.writeSystemIn(vector.get(i)); //ส่งข้อมูลขาเข้าให้โปรแกรมผู้เรียน
        Palindrome.main(new String[0]); //สั่งโปรแกรมผู้เรียนทำงาน
        String[] line = TestUtil.readSystemOut();//รับค่าผลจากการทำงานของโปรแกรมผู้เรียน
        String[] out = TestUtil.filter(line, 0, "= "); //คัดแยกข้อมูล
        // เปรียบเทียบผลลัพธ์จากการทำงานของโปรแกรมผู้เรียนกับของแฟ้ม test.dat
        TestUtil.assertTrue(result.get(i).equals(out[0]),"ข้อมูลไม่ถูกต้อง");
    }
    return 1;
}
```

รหัสที่ 3.17 การอ่านข้อมูลจากแฟ้มของตัวตรวจ

โจทย์กำหนดให้ ผู้เรียนทำการรับสายอักขระเข้ามาแล้วสายอักขระด้านหลังที่มีการเรียงกันเป็นกระจกเงาของด้านหน้า เช่น abXYZba จะเห็นว่าอักขระตัวท้ายเป็น ba ซึ่งเป็นตัวกลับของ ab โดยผู้เรียนต้องคืนค่า ab หรือสายอักขระ dlijd จะมี d เป็นกระจกเงาของสายอักขระด้านหน้า ผู้เรียนต้องคืนค่า d

ผู้ออกโจทย์สามารถกำหนดคำตอบไว้ที่แฟ้มข้อมูลทดสอบ test.dat จากนั้นนำข้อมูลขาเข้าจากแฟ้มข้อมูลโดยใช้ `TestUtil.getVectorInput()` และสามารถนำผลลัพธ์มาใช้โดยเรียก `TestUtil.getVectorResult()` จากนั้นวงวนเพื่อส่งข้อมูลขาเข้าให้โปรแกรมของผู้เรียน และรับข้อมูลผลลัพธ์จากโปรแกรมของผู้เรียน เพื่อมาเปรียบเทียบกับผลลัพธ์ที่ได้จากแฟ้มข้อมูลทดสอบ โดยรูปแบบของแฟ้มข้อมูลคือ มีบรรทัดแรกเพื่อบอกอักขระคั่น (delimiter) และบรรทัดต่อมาเป็นข้อมูลขาเข้า+อักขระคั่น+ผลลัพธ์ ซึ่งเห็นได้จากรูปที่ 3.9

```
,
abXYZba,ab
abca,a
aba,aba
dlijd,d
pekpkikep,pek
```

รูปที่ 3.9 แฟ้มข้อมูลทดสอบ test.dat

ผู้ออกโจทย์สามารถกำหนดรูปแบบของแฟ้มเป็นแถวลำดับแบบ 2 มิติ ซึ่งเป็นข้อมูลขาได้ และสามารถใช้ `TestUtil.getIntMatrices()` ซึ่งคืนค่าเป็น `List<int[][]>` ซึ่งสามารถเรียกใช้ได้ ดังรหัสที่ 3.18

```
@Test(loops=1, points=10, testData="test_matrices.dat")
public double test(){
    List<int[][]> lst = TestUtil.getIntMatrices();
    ...
    return 1;
}
```

รหัสที่ 3.18 การอ่านข้อมูลจากแฟ้มของตัวตรวจ

โดยรูปแบบของแฟ้มข้อมูล test_matrices.dat คือมีบรรทัดแรกเพื่อบอกอักขระคั่น และบรรทัดต่อมาเป็น ข้อมูล1+อักขระคั่น+ข้อมูล2…+ข้อมูลn และตารางแต่ละชุดข้อมูลมีอักขระคั่นเป็นบรรทัดใหม่ ดังรูปที่ 3.10

```
,
1,2,3,4
5,6,7,8

8,7,6,5
4,3,2,1
```

รูปที่ 3.10 แฟ้มข้อมูลทดสอบ test_matrices.dat

8) ลักษณะกำกับการตรวจระดับคลาส

ในกรณีผู้ออกโจทย์ต้องการกำหนดลักษณะกำกับการตรวจที่ใช้กับทุกเมท็อดกระทำได้โดยเขียนลักษณะกำกับการตรวจกำกับที่คลาส และที่เมท็อดก็มีการประกาศ `@Test` ตามปกติ ตัวควบคุมการตรวจจะตรวจเมท็อดนั้นโดยมีลักษณะกำกับการตรวจเหมือนกับของคลาส ซึ่งผู้ออกโจทย์สามารถแก้ไขหรือกำหนดลักษณะเพิ่มเต็มภายในเมท็อดได้ ดังรหัสที่ 3.19 มีการประกาศลักษณะกำกับการตรวจระดับคลาส โดยภายในมีเมท็อดการตรวจ 2 เมท็อดคือ `test1` และ `test2` ซึ่งเมท็อด `test2` จะมีการกำหนดจำนวนรอบของการทำงานเอง แต่ลักษณะกำกับการตรวจอื่นนอกจากจำนวนรอบจะเหมือนกับลักษณะกำกับการตรวจของคลาส

```
@Test(loops = 10, points = 1, timeout = 1000)
public class TestScript {
    @Test
    public double test1() {
        ...
        return 1;
    }
    @Test(loops=20)
    public double test2() {
        ...
        return 1;
    }
}
```

รหัสที่ 3.19 ตัวตรวจที่มีลักษณะกำกับการตรวจระดับคลาส

9) การห้ามใช้คลาสมาตรฐานที่มีอยู่ในจาวา

โจทย์วิชาโครงสร้างข้อมูล ต้องการวัดทักษะเกี่ยวกับการจัดการโครงสร้างข้อมูลเช่น ต้องการเรียงข้อมูลในแถวลำดับ ผู้เรียนจำเป็นต้องเขียนโปรแกรมจัดเรียงข้อมูลในแถวลำดับเอง หากผู้เรียนใช้คลาสมาตรฐานที่มีอยู่ในจาวา เช่น `Arrays.sort(Object[])` ซึ่งจะทำให้แถวลำดับของอ็อบเจกต์เรียงกัน แต่จะทำให้ผิดวัตถุประสงค์ของผู้ออกโจทย์

ดังนั้นตัวควบคุมการตรวจจึงมีการป้องกันการใช้คลาสมาตรฐาน โดยตัวควบคุมการตรวจจะค้นหาในตัวรหัส หากผู้เรียนมีการใช้คลาสมาตรฐานตัวควบคุมการตรวจจะสามารถแจ้งให้ผู้เรียนทราบว่าข้อนี้ห้ามใช้ โดยผู้ออกโจทย์สามารถกำหนดค่าให้ `prohibit` ในลักษณะกำกับการตรวจ ซึ่งสามารถระบุค่าในการป้องกันการใช้คลาสมาตรฐานได้หลายค่า และระบุคลาสของผู้เรียน

ค่า โดยใช้ `studentClass={SortData.class}` ซึ่งสามารถระบุได้หลายค่าเพื่อบอกตัวควบคุมการตรวจให้หาคำนี้ในรหัสของผู้เรียน หากผู้เรียนใช้คำนี้ในรหัสจะทำให้ผู้เรียนไม่ได้คะแนนในข้อนั้น ดังรหัสที่ 3.20

```
@Test(prohibit={"java.util.*"}, studentClass={SortData.class})
public double testSort(){
    String [] in = TestUtil.fillRandom(new String[10]);
    TestUtil.writeSystemIn(in);
    SortData.main(in);
    ...
    return 1;
}
```

รหัสที่ 3.20 ตัวตรวจที่ห้ามบางคำอยู่ในรหัสต้นฉบับของผู้เรียน

การทำงานของตัวควบคุมการตรวจในการห้ามใช้คลาสมาตรฐานของจาวา สามารถทำได้โดยการใช้นิพจน์ปรกติ (Regular Expression) ก่อนอื่นจะลบความคิดเห็นจากรหัสต้นฉบับก่อน แล้วจึงนำรหัสต้นฉบับหลังจากการลบความคิดเห็น มาค้นหาคำที่ผู้ออกโจทย์ระบุไว้ใน `prohibit` หากพบก็จะแสดงข้อความผิดพลาดให้ผู้ใช้เห็น ดังรหัสที่ 3.21

```
// remove comment
String sourceCodeNoComment=sourceCode.
            replaceAll("(?:/\\*(?:[^*]|(?:\\*+[^*/]))*\\*+/)|(?://.*)","");
Pattern pattern = Pattern.compile(notAllow);//ในกรณีนี้notAllowเป็น java.util.*
Matcher matcher = pattern.matcher(sourceCodeNoComment);
while ( matcher.find() ) {
    msg.add("error message");
}
```

รหัสที่ 3.21 การหาข้อความในรหัสต้นฉบับโดยใช้นิพจน์ปรกติ

10) <u>การตรวจที่มีกรณีย่อย</u>

โจทย์การเขียนโปรแกรมหนึ่งข้ออาจมีหลายกรณีที่จำเป็นต้องตรวจเช่น โจทย์ให้หาผลรวมของราคาสินค้า 10 ชิ้น โดยปกติราคาสินค้าจะไม่มีเป็นลบ หากมีสินค้าที่เป็นลบให้แจ้งกับผู้ใช้ว่าราคาไม่ถูกต้อง ดังนั้นตัวตรวจอาจแยกเป็น 2 กรณี คือ ตรวจในกรณีที่ถูกต้องคือราคาสินค้าเป็นบวกแล้วดูว่าโปรแกรมของผู้เรียนทำงานถูกหรือไม่ กรณีที่ 2 คือให้ข้อมูลสินค้าที่เป็นลบ แล้วดูว่าโปรแกรมของผู้เรียนแจ้งเตือนได้ถูกต้องหรือไม่ ผู้ออกโจทย์สามารถบอกตัวควบคุมการตรวจว่าเป็นกรณีย่อยได้โดยการตั้งชื่อและตามด้วยเครื่องหมาย ”$” และตามด้วยชื่อเมท็อดย่อย ดังแสดงที่รหัส 3.22

```
@Test(loops = 10, points = 1, timeout=1000)
public double sum$1_WithNormal(){
    ...
    return 1;
}
@Test(loops = 10, points = 1, timeout=1000)
public double sum$2_WithNegativeValue(){
    ...
    return 1;
}
```

รหัสที่ 3.22 ตัวตรวจที่มีกรณีย่อย

จากรหัสที่ 3.22 เห็นได้ว่ามี 2 เมท็อดย่อย คือ เมท็อด `sum$1_WithNormal` และเมท็อด `sum$2_WithNegativeValue` ตัวควบคุมจะสั่งให้ 2 เมท็อดนี้ทำงานสลับกัน โดยทำตามจำนวนที่กำหนดไว้ที่ลักษณะกำกับการตรวจ จากนั้นตัวควบคุมการตรวจจะรวมคะแนนของ 2 เมท็อดนี้เพื่อไปแสดงผล

```
sum : ok ok
You point : 10.00 ( Total 10.0 )
***************
```

รูปที่ 3.11 ผลลัพธ์ของการตรวจที่มีกรณีย่อย

จากรูปที่ 3.11 จะเห็นว่ามีการแสดงผลของการตรวจ 1 ครั้ง และคำว่า ok 2 อันคือการทำงานกรณีย่อยของเมท็อดนี้ถูกต้อง โดยคะแนนจะเป็นคะแนนรวมของทุกเมท็อด

11) การกำหนดเมท็อดก่อนเริ่มการตรวจและเมท็อดหลังจากการตรวจ

ในวิชาโครงสร้างข้อมูลมีรูทีน (Routine) ที่ทำทุกครั้งก่อนการตรวจ เช่น การเพิ่มข้อมูลจะต้องมีการทำก่อน การตรวจการลบข้อมูล การตรวจขนาดของโครงสร้างข้อมูล ซึ่งหากเขียนอยู่ในทุกเมท็อดของการตรวจการการบำรุงรักษา หรือการแก้ไขจะทำได้ยาก หากผู้ออกโจทย์ เขียนรูทีนนั้นเป็นเมท็อดแล้วจึงเรียกก่อนการทำงานผู้ออกโจทย์จะต้องเสียเวลาในการเรียกทุกครั้งดังรหัสที่ 3.21เห็นได้ว่ามีการเรียกเมท็อด `setUp` ก่อนการเรียกเมท็อดการตรวจ

```
private List lst;
public void setUp() {
    lst = new ArrayList();
}
@Test(loops = 10, points = 1, timeout=1000)
public double test1() {
    setUp(); // เรียกเมท็อดก่อนการตรวจเอง
    return 1;
}
@Test(loops = 10, points = 1, timeout=1000)
public double test2() {
    setUp(); // เรียกเมท็อดก่อนการตรวจเอง
    return 1;
}
```

รหัสที่ 3.23 การเรียกเมท็อดก่อนการตรวจเอง

จากรหัสที่ 2.23 ผู้ออกโจทย์มีการประกาศ `@Before` ไว้ที่เมท็อด `setUp` เพื่อให้ตัวควบคุมการตรวจเรียกก่อนเรียกเมท็อดการตรวจทุกเมท็อด และมีการประกาศ `@After` เพื่อให้เรียกทุกครั้งหลังจากเมท็อดการตรวจทำงานเสร็จ

```
private List lst;
@Before
public void setUp() {
    lst = new ArrayList();
}
@Test(loops = 10, points = 1, timeout=1000)
public double test1() {
    return 1;
}
@After
public void afterTest() {lst = null;}
```

รหัสที่ 3.24 การกำหนดเมท็อดก่อนเริ่มการตรวจและเมท็อดหลังจากการตรวจ

บทที่ 4

ตัวอย่างและผลการทดลอง

บทที่นี้จะแสดงตัวอย่างของโจทย์พร้อมตัวตรวจโดยมีการใช้งานตัวควบคุมการตรวจและคลาสอรรถประโยชน์ซึ่งจะครอบคลุมความต้องการพื้นฐานของการพัฒนาตัวตรวจโปรแกรมโดยทั่วไป

4.1 ตัวอย่างที่ 1 อีเมล

โจทย์กำหนดให้ผู้ใช้ป้อนสายอักขระทางแป้นพิมพ์ (โปรแกรมจะต้องพิมพ์คำว่า input : ก่อนจึงให้ผู้ใช้ใส่สายอักขระ) แล้วดึงค่าอีเมลออกมาเพื่อแสดงบนจอภาพ

จากโจทย์นี้ผู้เรียนสามารถเขียนโปรแกรมได้โดยเริ่มจากการสั่งพิมพ์ “input :” และสร้าง `Scanner` จากนั้นรับค่าจากแป้นพิมพ์เพื่อกำหนดให้ตัวแปร `s` และหาตำแหน่งของ mailto: ซึ่งเป็นตำแหน่งเริ่มต้นของอีเมลโดยจะกำหนดค่าให้ตัวแปร `i` และหาตำแหน่ง \ ซึ่งเป็นตำแหน่งสุดท้ายของอีเมลโดยจะกำหนดค่าให้ตัวแปร `j` แล้วหาสายอักขระย่อย (Substring) จากตำแหน่งหลัง mailto: ซึ่งเป็นตำแหน่งเริ่มต้นของอีเมล จนถึง \ ซึ่งเป็นตำแหน่งสิ้นสุดของอีเมล และสั่งพิมพ์ออกจอภาพ ดังรหัสที่ 4.1

```
public class EmailHarvester {
    public static void main(String[] args) {
        System.out.print("input : "); // สั่งพิมพ์ข้อความ input :
        Scanner kb = new Scanner(System.in); // สร้าง Scanner เพื่อรับข้อมูลจากแป้นพิมพ์
        String s = kb.nextLine(); // กำหนดค่าของข้อมูลที่ได้จากการป้อนของผู้ใช้ให้ s
        int i = s.indexOf("\"mailto:") + 1; // หาตำแหน่งเริ่มต้นของอีเมล
        int j = s.indexOf("\"", i); // หาตำแหน่งสุดท้ายของอีเมล
        System.out.println(s.substring(i + "mailto:".length(), j));
    }
}
```

รหัสที่ 4.1 โปรแกรมของผู้เรียนสำหรับโจทย์อีเมล

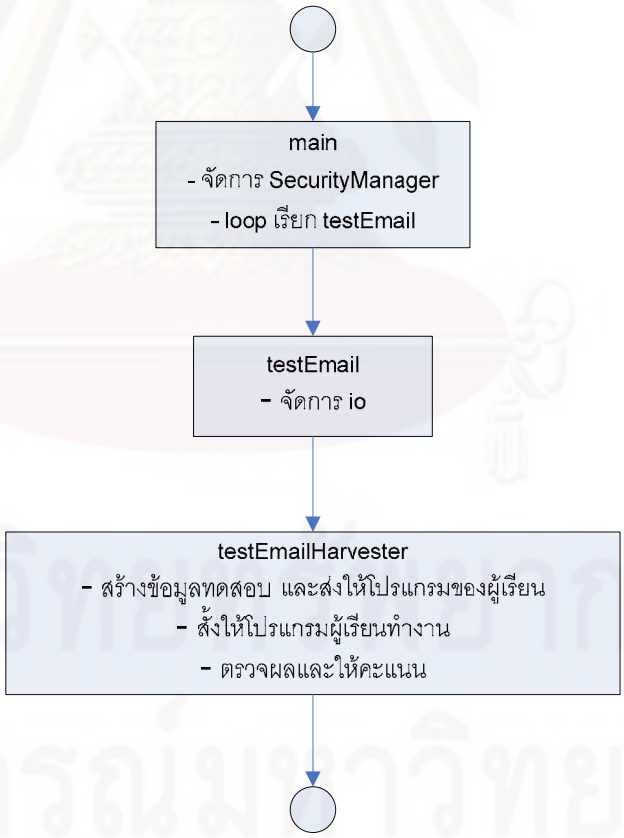
```
>java EmailHarvester
input : <a href="mailto:chaluemwut@hotmail.com">me</a>
chaluemwut@hotmail.com
```

รูปที่ 4.1 การสั่งดำเนินงานของโปรแกรมอีเมล

รูปที่ 4.1 แสดงการสั่งทำงานของโปรแกรม `EmailHarvester` ผู้ใช้ป้อนสายอักขระ me โปรแกรมจะสั่งพิมพ์อีเมลซึ่งอยู่ภายในสายอักขระนั้นออกทางจอภาพ

โดยก่อนอื่นจะแสดงตัวตรวจแบบเก่า ซึ่งไม่ได้ใช้ตัวควบคุมการตรวจและคลาสอรรถประโยชน์สามารถแสดงการทำงานได้ดังภาพที่ 4.2 โดยเริ่มจากเมท็อด `main` กำหนดความมั่นคงใหม่เพื่อการสั่งทำงานโปรแกรมของผู้เรียน (บรรทัดที่ 9 ถึง 14) จากนั้นวงวนการตรวจเป็นจำนวน 10 ครั้งโดยแต่ละครั้งจะไปเรียกเมท็อด `testEmail` เมท็อดนี้จัดการเกี่ยวกับข้อมูลขาเข้าและข้อมูลส่งออกของโปรแกรม โดยการสร้างสายข้อมูล `stream1` เป็นสายข้อมูลของผู้เรียนและ

`stream2` เป็นสายข้อมูลของตัวตรวจ จากนั้นกำหนดให้สายข้อมูลขาเข้าของระบบเป็นของตัวตรวจ (บรรทัดที่ 31) และสายข้อมูลส่งออกเป็นของโปรแกรมของผู้เรียน (บรรทัดที่ 30) จากนั้นเรียก `testEmailHarvester` เพื่อสร้างข้อมูลทดสอบและเรียกกรณีย่อยของการตรวจ สำหรับข้อมูลทดสอบคือจะสร้างสายอักขระสุ่มซึ่งภายในมีอีเมลอยู่ เริ่มด้วยการสร้างอีเมลแบบสุ่มจาก `userName` และ `mailServer` (บรรทัดที่ 38) จากนั้นสร้างสายอักขระแบบสุ่ม (บรรทัดที่ 41) และสุ่มเลือกสายอักขระนั้นเพื่อมาประกอบเป็นสายอักขระใหม่ เพื่อส่งให้โปรแกรมของผู้เรียน (บรรทัดที่ 48) จากนั้นส่งสายอักขระเข้าโปรแกรมของผู้เรียนโดยใช้ `out.println(test)` และอ่านข้อมูลซึ่งได้จากการทำงานของโปรแกรมผู้เรียน โดยสร้างเป็น `BufferedReader` จาก `stream1` ซึ่งเป็นสายข้อมูลของผู้เรียน และสั่งให้โปรแกรมผู้เรียนทำงาน และอ่านค่าหลังจากการทำงานของโปรแกรมผู้เรียน มีการคัดแยกข้อมูลโดยตัดข้อความ “input :” ออกก่อนนำมาเปรียบเทียบ (บรรทัดที่ 50) และนำสายอักขระหลังจากการคัดแยกข้อมูลมาเปรียบเทียบกับคำตอบที่ถูกต้อง (บรรทัดที่ 52) หากเท่ากันคะแนนจะเพิ่มขึ้น เมื่อครบ 10 รอบจะแสดงคะแนนที่ได้ออกทางจอภาพ ดังรหัสที่ 4.2



รูปที่ 4.2 การทำงานของตัวตรวจแบบเก่า

```
01:public class TestScript {
02:    private static int point = 0;
03:    private static final String[] userName = { "chaluemwut", "nanadev",
04:                                              "chula","jtest101","jlab" };
05:    private static final String[] mailServer = { "thai.com", "nana.com",
06:                                                 "mail.cp.eng.chula.ac.th",
07:                                                 "abc.def.ghi.com"};
08:    public static void main(String[] args) {
09:       System.setSecurityManager(new SecurityManager() {
10:           public void checkExit(int status) {
11:               throw new SecurityException("exit() is prohibited");
12:           }
13:            public void checkPermission(Permission perm) {}
14:       });
15:       System.out.print(">>JLab:Testing -> EmailHarvester : ");
16:       for (int i = 0; i < 10; i++) {
17:           testEmail();
18:       }
19:       System.out.println();
20:       System.out.println("Point : " + point);
21:    }
22:    public static void testEmail() {
23:       PrintStream stdOut = System.out;
24:       PipedOutputStream pipe1Out = new PipedOutputStream();
25:       PipedOutputStream pipe2Out = new PipedOutputStream();
26:       PrintStream stream1Out = new PrintStream(pipe1Out, true);
27:       PrintStream stream2Out = new PrintStream(pipe2Out, true);
28:       InputStream stream1In = new PipedInputStream(pipe1Out);
29:       InputStream stream2In = new PipedInputStream(pipe2Out);
30:       System.setOut(stream1Out);
31:       System.setIn(stream2In);
32:       testEmailHarvester(stream1In, stream2Out);
33:       System.setOut(stdOut);
34:       stream1In.close();
35:       stream2Out.close();
36:    }
37:    static void testEmailHarvester(InputStream in,PrintStream out){
38:       String email = userName[rand(0, userName.length - 1)] + "@"
39:                           + mailServer[rand(0, mailServer.length - 1)];
40:       String randText = "dfjf;laks j;lalsfj@ldk@ldj lksdjf@lkjsdf";
41:       String test = randText.substring(0,rand(5,randText.length()-6))+
42:                        "\"mailto:"+email+"\""+randText.substring
43:                        (0,rand(5,randText.length()-6));
44:       out.println(test);
45:       try {
46:           InputStreamReader inStudent = new InputStreamReader(in);
47:           BufferedReader br = new BufferedReader(inStudent);
48:           EmailHarvester.main(new String[0]);
49:           String lines = br.readLine();
50:           lines= lines.substring(lines.indexOf("input : ")+"input”+
51:                                  “ : ".length());
52:           point += (email.equals(lines.trim()) ? 1 : 0);
53:       } catch (Throwable e) {
54:           e.printStackTrace();
55:       }
56:    }
57:    private static int rand(int n1, int n2) {
58:       return n1 + (int)((n2 - n1 + 1) * Math.random());
59:    }
60:}
```

รหัสที่ 4.2 ตัวอย่างตัวตรวจแบบเก่าของอีเมล

สำหรับตัวตรวจหลังจากใช้ตัวควบคุมการตรวจและคลาสอรรถประโยชน์แล้วสามารถทำได้โดยสร้างอีเมลซึ่งสุ่มจากแถวลำดับ `userName` และ `mailServer` จากนั้นสุ่มสายอักขระเพื่อมา

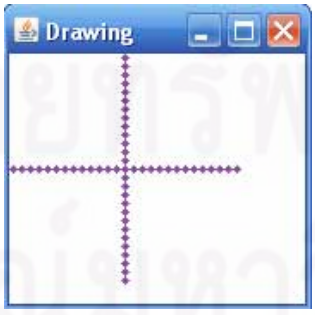
ประกอบเป็นด้านหน้าและด้านหลังของสายอักขระอีเมล เพื่อส่งให้โปรแกรมของผู้เรียนโดยเรียก `TestUtil.writeSystemIn` และสั่งทำงานโปรแกรมของผู้เรียน และอ่านผลจากจอภาพโดยเรียก `TestUtil.readSystemOut` มีการคัดแยกข้อมูลโดยเรียก `TestUtil.filter` และเปรียบเทียบคำตอบของผู้เรียนกับค่าที่ถูกต้องโดยเรียก `TestUtil.assertTrue` หากไม่เท่าจะแสดงข้อความ “คำนวณไม่ถูกต้อง” ออกทางจอภาพ ดังรหัสที่ 4.3

```
private static final String[] userName = { "chaluemwut", "nanadev",
                                           "chula", "jtest101", "jlab" };
private static final String[] mailServer = { "thai.com", "nana.com",
                                             "mail.cp.eng.chula.ac.th",
                                             "abc.def.ghi.com"};
@Test(loops = 10, points = 1, timeout=1000)
public double testEmail() {
    String mail = userName[TestUtil.random(0, userName.length)] + "@"
                        + mailServer[TestUtil.random(0, mailServer.length)];
    String s = TestUtil.radomString()+"\"mailto:"
                                    +mail+"\""+TestUtil.radomString();
    TestUtil.writeSystemIn(s);
    EmailHarvester.main(new String[0]);
    String[] lines = TestUtil.readSystemOut();
    String[] out = TestUtil.filter(lines, 0, "input : ");
    TestUtil.assertTrue(out[0].equals(mail), "คำนวณไม่ถูกต้อง");
    return 1;
}
```

รหัสที่ 4.3 ตัวอย่างตัวตรวจของอีเมลโดยใช้ตัวควบคุมการตรวจและคลาสอรรถประโยชน์

4.2 ตัวอย่างที่ 2 แถวลำดับ

โจทย์กำหนดให้สร้างเมท็อด `void plus(boolean[ ][ ] m)` ซึ่ง `m` เป็นค่าเมทริกซ์จัตุรัส (square matrix) ที่มีมิติเป็นจำนวนคี่ เช่น 11x11, 43x43 เป็นต้น (แถวลำดับที่รับมามีค่าในทุกๆ ช่องเป็นเท็จ) เมท็อด `plus` (ในโจทย์จริงมี 4 เมท็อด แต่ตัวอย่างนี้ของแสดงแต่เมท็อด `plus`) มีหน้าที่เติมค่า จริง/เท็จ ในแถวลำดับ ที่เราสามารถนำไปวาดเป็นรูปบนจอภาพ (ผู้เรียนไม่ต้องเขียนในส่วนของการวาดภาพ เนื่องจากผู้ออกโจทย์จัดการในส่วนนี้) โดยแต่ละช่องในแถวลำดับแทนจุดหนึ่งจุดบนจอภาพ (ค่าจริงแทนจุดที่มีสีสัน ในขณะที่ค่าเท็จแทนสีพื้น) ดังรูปที่ 4.3



รูปที่ 4.3 การวาดภาพหลังจากส่งแถวลำดับ `m` ให้ `plus`

ตัวตรวจเริ่มด้วยการสุ่มจำนวนเต็มเพื่อกำหนดให้ `s` แล้วหาค่าจำนวนคี่ โดยหาร `s` ด้วย 2 หากลงตัวแสดงว่า `s` เป็นคู่ ให้บวกค่า `s` อีกหนึ่ง เพื่อทำให้ `s` เป็นคี่ และสร้างแถวลำดับ `m1` และ `m2`

ขนาด s (ซึ่งเป็นจำนวนคี่) และส่ง m1 ให้เมท็อด plus ของตัวตรวจ (ซึ่งเป็นเมท็อดเฉลย) และส่ง m2 ให้เมท็อด plus ของผู้เรียน และเปรียบเทียบแถวลำดับ 2 อันนี้ว่าเท่ากันหรือไม่ ดังรหัสที่ 4.4

```
@Test(loops = 10, points = 1, timeout=1000)
public double testPlus() {
    int s = TestUtil.random(20,30);
    s = s % 2 == 0 ? s + 1 : s; // หาจำนวนคี่
    boolean[][] m1 = new boolean[s][s];
    boolean[][] m2 = new boolean[s][s];
    plus(m1); // เรียกเมท็อดของตัวตรวจ
    TDrawing.plus(m2); // เรียกเมท็อดของผู้เรียน
    TestUtil.assertTrue(m1, m2, "คำนวณไม่ถูกต้อง"); // เปรียบเทียบแถวลำดับ
    return 1;
}
private static void plus(boolean[][] m) {
    // เมท็อดหาผลเฉลย
    int w = m.length / 2;
    for (int i = 0; i < m.length; i++) {
        m[w][i] = m[i][w] = true;
    }
}
```

รหัสที่ 4.4 ตัวอย่างตัวตรวจของแถวลำดับ

4.3 ตัวอย่างที่ 3 ดัชนีมวลกาย

โจทย์กำหนดให้คำนวณดัชนีมวลกาย (body mass index เรียกย่อว่า BMI) โดยรับค่าจากแป้นพิมพ์ 2 ค่า w คือน้ำหนัก (หน่วยเป็นกิโลกรัม) และ h คือส่วนสูง (หน่วยเป็นเซนติเมตร) โดยดัชนีมวลกายคำนวณได้จากสูตร BMI = w/h^2

จากโจทย์นี้ผู้เรียนสามารถเขียนโปรแกรมได้โดยเริ่มจากการสร้าง Scanner เพื่อรับข้อมูลจากแป้นพิมพ์ และสั่งพิมพ์ “weight (kg.) :” และรับค่าน้ำหนักเพื่อกำหนดค่าให้ตัวแปร w และสั่งพิมพ์ “height (cm.) :” และรับค่าความสูงเพื่อกำหนดค่าให้ตัวแปร h จากนั้นหารความสูงด้วย 100 เพื่อเปลี่ยนให้เป็นหน่วยเมตร (สูตรต้องการหน่วยเป็นเมตร) และนำค่าที่ได้ยกกำลังสองและเอาไปหารค่าของน้ำหนักจะเป็นค่าดัชนีมวลกาย แล้วสั่งพิมพ์ออกทางจอภาพ ดังรหัสที่ 4.5

```
public class BMI {
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("weight (kg.) : ");
        int w = kb.nextInt(); // รับค่าน้ำหนักจากแป้นพิมพ์
        System.out.print("height (cm.) : ");
        int h = kb.nextInt(); // รับค่าความสูงจากแป้นพิมพ์
        double heightInMeter = h / (double)100; // เปลี่ยนหน่วยเป็นเมตร
        double bmi = w / (heightInMeter * heightInMeter);
        System.out.println(bmi, "คำนวณไม่ถูกต้อง");
    }
}
```

รหัสที่ 4.5 โปรแกรมหาดัชนีมวลกายของผู้เรียน

ตัวตรวจเริ่มด้วยการสุ่มเลขจาก 101 ถึง 250 เพื่อเป็นความสูง และสุ่มเลขจาก 50 ถึง 200 เพื่อเป็นน้ำหนัก และส่งทั้งคู่ให้กับโปรแกรมของผู้เรียนโดยเรียก TestUtil.writeSystemIn(new int[]{w, h}) แล้วจึงเรียกโปรแกรมของผู้เรียน จากนั้นอ่านผลจากจอภาพ คัดแยกข้อมูล และ

เปลี่ยนข้อมูลเป็นจำนวนจริง และคำนวณหาผลเฉลยของค่าดัชนีมวลกาย เนื่องจากผลเฉลยเป็นจำนวนจริง จึงใช้การเปรียบเทียบแบบประมาณ โดยเรียก `TestUtil.equals` และส่งค่า `bmi` และ `out[0]` หากทั้ง 2 มีค่าผิดพลาดสัมพัทธ์ไม่เกิน 0.001 จะคืนค่าจริง แต่หากเกินจะคืนค่าเท็จ ดังรหัสที่ 4.6

```
@Test(loops = 10, points = 1, timeout=1000)
public double testBMI() {
    int w = TestUtil.random(50, 200); // สุ่มหาค่าน้ำหนักอยู่ระหว่าง 50 ถึง 200
    int h = TestUtil.random(101, 250); // สุ่มหาค่าความสูงอยู่ระหว่าง 101 ถึง 250
    TestUtil.writeSystemIn(new int[]{w, h}); // ส่งน้ำหนักและความสูงให้โปรแกรมผู้เรียน
    BMI.main(new String[0]); // สั่งให้โปรแกรมของผู้เรียนทำงาน
    String[] lines = TestUtil.readSystemOut();
    lines = TestUtil.filter(lines, 0, "height (cm.) : ");
    double[] out = TestUtil.string2double(lines);
    double heightInMeter = h / (double) 100;
    double bmi = w / (heightInMeter * heightInMeter);
    // เปรียบเทียบแบบประมาณของจำนวนจริง ถ้า bmi และ out[0] มีค่าประมาณต่างกันน้อยกว่า 0.001 จะคืนค่าจริง
    boolean d = TestUtil.equals(bmi, out[0], 0.001);
    TestUtil.assertTrue(d, "คำนวณไม่ถูกต้อง");
    return 1;
}
```

รหัสที่ 4.6 ตัวอย่างตัวตรวจโปรแกรมหาดัชนีมวลกาย

4.4 ตัวอย่างที่ 4 การตรวจสอบผลเฉลยของเกมซูโดกุ

โจทย์กำหนดให้เขียนเมท็อด `isSudoku` ซึ่งตรวจสอบว่า ตารางที่ได้รับมาถูกต้องตามกฎของเกมซูโดกุหรือไม่ ตัวตรวจโจทย์นี้เป็นการตรวจเมท็อดที่คืนค่าเป็นบูลีน และใช้วิธีการอ่านแฟ้มข้อมูลทดสอบ ตัวตรวจเริ่มด้วยการเขียนคลาส `BooleanMethod` และสร้าง t เป็นแถวลำดับที่แทนตารางซูโดกุ เรียก `TestUtil.getIntMatrices` และส่งชื่อแฟ้ม sudoku_true.dat (เป็นแฟ้มที่เก็บตารางซูโดกุที่ถูกต้องทั้งหมดตัวอย่างแฟ้มดังรูปที่ 4.4) เพื่อให้คลาสอรรถประโยชน์นำข้อมูลมาเก็บไว้ในรายการ `tList` ซึ่งเป็นแถวลำดับที่มีการเก็บตารางซูโดกุที่ถูกต้องทั้งหมด จากนั้นเรียก `TestUtil.getIntMatrices` และส่งชื่อแฟ้ม sudoku_false.dat (ตัวอย่างแฟ้มดังรูปที่ 4.5)เพื่อให้คลาสอรรถประโยชน์นำข้อมูลมาเก็บไว้ในรายการ `fList` ซึ่งเป็นรายการของตารางซูโดกุที่ผิดกฎของซูโดกุ และสร้าง `isTrue` โดยหาก `newData` เป็นจริง ให้ไปสร้างข้อมูลใหม่ โดยไปดึงค่าจากรายการ `tList` แบบสุ่มและกำหนดให้ `t` หาก `newData` เป็นเท็จ ให้ใช้ข้อมูล `t` ชุดเดิม การสร้างเมท็อด `isFalse` ก็เหมือนกัน ต่างกันที่การสร้างข้อมูลใหม่เมื่อ `newData` เป็นจริง ให้สร้างข้อมูลทดสอบเพื่อส่งให้โปรแกรมจากรายการ `fList` ซึ่งเป็นรายการที่เก็บตารางที่ผิดกฎของซูโดกุ และเรียก `TestUtil.testBooleanMethod` เพื่อสั่งให้การตรวจเมท็อดบูลีนดำเนินการ หากถูกทุกกรณีจะคืนค่าคะแนนเต็มให้ผู้เรียน หากผิดบ้างจะได้คะแนนตามส่วน ดังรหัสที่ 4.7

```
@Test(loops = 10, points = 1, timeout=1000)
public double testSudoku() {
    TestUtil.BooleanMethod bm = new TestUtil.BooleanMethod() {
        int[][] t = new int[9][9]; // เก็บตารางของซูโดกุ
        // ได้รายการที่เก็บตารางที่ถูกตามกฎของซูโดกุ
        List<int[][]> tList = TestUtil.getIntMatrices("sudoku_true.dat");
        // รายการที่เก็บตารางที่ผิดกฎของซูโดกุ
        List<int[][]> fList = TestUtil.getIntMatrices("sudoku_false.dat");
        public boolean isTrue(boolean newData) {
            if (newData) {
                // สุ่มเลือกข้อมูลจากรายการที่เก็บตารางที่ถูก
                t = tList.get(TestUtil.random(0, tList.size() - 1));
            }return Sudoku.isSudoku(t);
        }
        public boolean isFalse(boolean newData) {
            if (newData) {
                // สุ่มเลือกข้อมูลจากรายการที่เก็บตารางที่ผิด
                t = fList.get(TestUtil.random(0, fList.size() - 1));
            }return Sudoku.isSudoku(t);
        }
    };
    double p = TestUtil.testBooleanMethod(bm);
    return TestUtil.assertTrue(p == 1, "ผิดบางกรณี", p);
}
```

รหัสที่ 4.7 ตัวอย่างตัวตรวจโปรแกรมหาผลเฉลยของเกมซูโดกุ

```
,

9,6,3,1,7,4,2,5,8

...

3,1,7,2,4,6,9,8,5

6,4,2,5,9,8,1,7,3



2,4,3,7,1,6,9,5,8

9,8,5,2,3,4,7,1,6

1,6,2,4,7,9,8,3,5

3,7,8,6,5,2,4,9,1

5,9,4,3,8,1,2,6,7
```

รูปที่ 4.4 ตัวอย่างแฟ้มข้อมูลของ sudoku_true.dat

```
,

2,4,3,7,1,6,9,5,8

9,8,5,2,3,4,7,1,6

...

3,7,8,6,5,2,4,9,1

1,2,3,4,5,6,7,8,9
```

```
2,4,3,7,1,6,9,5,8

9,8,5,2,3,4,7,1,6

...

3,7,8,6,5,2,4,9,1

1,2,0,4,5,6,7,8,9
```

รูปที่ 4.5 ตัวอย่างแฟ้มข้อมูลของ sudoku_false.dat

4.5 ตัวอย่างที่ 5 การเพิ่มข้อมูลของกองซ้อนจำนวนเต็ม

โจทย์กำหนดให้สร้างเมท็อด `push` ของกองซ้อน ถ้ากองซ้อนเต็ม ให้ขยายขนาดของแถวลำดับ ที่เก็บค่าเป็น 2 เท่า และห้ามใช้คลาสมาตรฐานใน `java.util` มาช่วยเก็บข้อมูล

จากโจทย์นี้ผู้เรียนสามารถเขียนโปรแกรมได้โดย เริ่มจากเมท็อด `push` มีจะตรวจสอบค่า `size` หากมากกว่าค่าของ `elementData.length` ให้สร้างแถวลำดับ `a` ที่มีขนาดเป็น 2 เท่าของ `elementData.length` และสำเนาข้อมูลใน `elementData` มาไว้ที่ `a` และกำหนดค่า `a` (ซึ่งมีข้อมูลของ `elementData` และขนาดเป็น 2 เท่า) กลับให้ `elementData` แต่หากค่าของ `size` ไม่มากกว่าขนาดของ `elementData` ให้เพิ่มขนาดของกองซ้อนตามปกติ โดยเพิ่มข้อมูลที่รับมาคือ `x` ให้กับ `elementData` โดยเพิ่มที่ตำแหน่งสุดท้ายของแถวลำดับ และเพิ่มค่า `size` ไปหนึ่ง ดังรหัสที่ 4.8

```
public class StudentIntStack {
    private int[] elementData; // ข้อมูลของ stack
    private int size;     // ขนาดของ stack
    // โจทย์สร้าง constructor ให้ 2 เมท็อด
    public StudentIntStack() {
        elementData = new int[1];
        size = 0;
    }
    public StudentIntStack(int initialCapacity) {
        elementData = new int[initialCapacity];
        size = 0;
    }
    // เมท็อด plush ที่ผู้เรียนต้องเขียน
    public void push(int x ) {
        if (size >= elementData.length) {
            int[] a = new int[elementData.length * 2];//เพิ่มขนาดของแถวลำดับเป็น2เท่า
            // ทำสำเนาข้อมูลของ elementData ให้กับแถวลำดับ a
            System.arraycopy(elementData, 0, a, 0, elementData.length);
            elementData = a;
        }
        elementData[size] = x; // เพิ่มข้อมูลต่อท้ายของ stack
        size++;
    }

}
```

รหัสที่ 4.8 คลาส `StudentIntStack` ซึ่งมีเมท็อด `plus` ของผู้เรียน

ตัวตรวจแยกเป็น 2 เมท็อดย่อย มีเมท็อด `setUp` เพื่อสร้างกองซ้อนของผู้เรียนโดยสุ่มค่าตัวเลขขนาด 10 ตัว และวงวนเพื่อใส่ค่าตัวเลขที่ได้จากการสุ่มใส่ลงในเมท็อด `push` จากนั้นนำค่าใน `elementData` กำหนดให้ `out` โดยใช้ `TestUtil.getField` (ไม่เรียกเมท็อด `topAndPop` ของ

ผู้เรียนเนื่องจาก อาจยังทำไม่เสร็จ และ elementData เป็นข้อมูล private) เมท็อดการตรวจแรก testPush$1_WithOutDoubling เป็นการตรวจความถูกต้องของข้อมูลว่าแถวลำดับของ in และ out เท่ากันทุกตัวหรือไม่ และตรวจขนาดของ elementData ว่าเท่ากับ n หรือไม่ ซึ่งเมท็อดนี้จะไม่มีการขยายขนาดของกองซ้อน เมท็อดที่สองคือ testPush$2_Doubling ซึ่งเป็นการตรวจสอบการขยายตัวของกองซ้อนเมื่อกองซ้อนเต็ม ว่ามีการขยายขนาดของ elementData เป็น 2 เท่าหรือไม่ โดยการ plus ค่าจำนวนเต็มสุ่มลงไปอีกหนึ่งค่า (กองซ้อนต้องขยายตัว) และตรวจสอบว่าขนาดของแถวลำดับ elementData เป็น 2 เท่าของ n หรือไม่ ดังรหัสที่ 4.9

```
@Test(loops = 10, points = 1, timeout=1000,
      prohibit = {"java.util.*"},
      studentClass = {StudentIntStack.class})
public class TestIntStack {
    StudentIntStack studentStack;
    int[] in, out;
    int n;
    @Before
    public void setUp() {
        n = TestUtil.random(10, 100);
        // สร้างแถวลำดับของจำนวนเต็มแบบสุ่มขนาด n
        in = TestUtil.fillRandom(new int[n], 10, 1000);
        studentStack = new StudentIntStack(n); // สร้าง stack ของผู้เรียน
        for (int i : in) {
            studentStack.push(i); // เรียกเมท็อด push ของผู้เรียน
        }
        // นำค่าข้อมูล elementData ของ stack ผู้เรียนกำหนดให้ out
        out = (int[]) TestUtil.getField(studentStack, "elementData");
    }
    @Test
    public double testPush$1_WithOutDoubling() {
        // ตรวจข้อมูลทุกตัวภายใน stack
        TestUtil.assertTrue(in, out, "ข้อมูลไม่ถูกต้อง");
        // ตรวจขนาดของ statck
        TestUtil.assertTrue(n == out.length, "ความยาวไม่ถูกต้อง");
        return 1;
    }
    @Test(loops = 20)
    public double testPush$2_Doubling() {
        // plus ข้อมูลเพื่อให้ stack ขยายออก
        studentStack.push(TestUtil.random());
        // get ค่า elementData จาก stack ของผู้เรียนอีกครั้ง
        out = (int[]) TestUtil.getField(studentStack, "elementData");
        // ตรวจสอบขนาดว่าเป็น 2 เท่าหรือไม่
        TestUtil.assertTrue(n*2 == out.length);
        return 1;
    }
}
```

รหัสที่ 4.9 ตัวอย่างตัวตรวจของการเพิ่มข้อมูลของคลาส StudentIntStack

4.6 ตัวอย่างที่ 6 การกลับลำดับข้อมูลภายในรายการ

โจทย์กำหนดให้สร้างเมท็อด reverse เพื่อใช้กลับลำดับข้อมูลในรายการ ซึ่งโครงสร้างโปรแกรมของผู้เรียนจะคล้ายกับตัวอย่างที่ 5 แต่ต่างกันที่ elementData เป็นแถวลำดับของสายอักขระ การเขียนตัวตรวจเริ่มด้วยการสร้างสายอักขระสุ่มขนาด 10 ตัว และสร้างอ็อบเจกต์ของผู้เรียน และกำหนดค่าให้กับ elementData และ size โดยใช้ TestUtil.setField และเรียกเมท็-

อด `reverse` จากนั้นนำค่า `elementData` ซึ่งเก็บข้อมูลในรายการโดยใช้ `TestUtil.getField` และวงวนเพื่อตรวจสอบว่ารายการมีการกลับลำดับจริงหรือไม่

```
@Test(loops = 10, points = 1, timeout=1000)
public double testReverse() {
    String [] in = TestUtil.fillRandom(new String[10]);
    StudentArrayList myLst = new StudentArrayList();
    // กำหนดค่าของ in ให้กับ elementData และกำหนดขนาดของ in ให้กับ size
    TestUtil.setField(myLst, "elementData", in);
    TestUtil.setField(myLst, "size", in.length);
    myLst.reverse(); // เรียกเมท็อดของผู้เรียน
    String[] out = (String[]) TestUtil.getField(myLst, "elementData");
    int counter = in.length;
    for (int i = 0; i < in.length; i++) { // ตรวจผลเฉลย
        TestUtil.assertTrue(in[i].equals(out[--counter]), "ข้อมูลไม่ถูกต้อง");
    }
    return 1;
}
```

รหัสที่ 4.10 ตัวอย่างตัวตรวจการกลับลำดับข้อมูลในรายการ

4.7 ตัวอย่างที่ 7 การหาค่าน้อยอันดับสองของฮีป

โจทย์กำหนดให้หาข้อมูลที่มีค่าน้อยสุดเป็นอันดับสองที่เก็บในฮีปทวินาม ตัวตรวจเริ่มจากการสุ่มสร้างฮีป 100 ตัว แบบสุ่มค่าอยู่ระหว่าง 100 ถึง 200 ตัว จากนั้นเพิ่มค่าฮีปเพื่อให้เป็นค่าน้อยที่สุด (ค่าอยู่ระหว่าง 0 ถึง 9) และหาค่าฮีปที่น้อยเป็นอันดับที่สองซึ่งค่าจะอยู่ระหว่าง 10 ถึง 99 และทำสำเนาอ็อบเจกต์ก่อนการเรียกเมท็อดผู้เรียน เพื่อหาค่าอันดับสองของผู้เรียนโดยเรียก `TestUtil.saveState` แล้วจึงเรียกเมท็อด `h.get2nd` ของผู้เรียนเพื่อเปรียบเทียบค่ากับ `min2` ซึ่งเป็นผลเฉลย หลังจากการเรียกเมท็อดของผู้เรียนจะตรวจสอบค่าอ็อบเจกต์ภายในโครงสร้างข้อมูลมีการเปลี่ยนแปลงหรือไม่โดยเรียก `state.sameState` ซึ่งจะคืนค่าเป็นบูลีน

```
@Test(loops = 10, points = 1, timeout=1000)
public double testGet2ndMin() {
    Heap h = createRandomHeap(100, 100, 200); // สร้างฮีปที่มีค่าสุ่มระหว่าง 100 ถึง 200
    h.add(TestUtil.random(0, 9));  // เพิ่มค่าสุ่มระหว่าง 0 ถึง 9 (ค่าน้อยสุด)
    int min2 = TestUtil.random(10, 99);  // เพิ่มค่าสุ่มระหว่าง 10 ถึง 99 (ค่าน้อยสุดอันดับสอง)
    h.add(min2);  // เพิ่มค่าสุ่มระหว่าง 10 ถึง 99 (ค่าน้อยสุดอันดับสอง)
    TestUtil.State state = TestUtil.saveState(h);  // ทำสำเนาอ็อบเจกต์
    TestUtil.assert(h.get2nd() == min2, "คืนผลผิด");  // ตรวจสอบผล
    //ตรวจสอบว่าต้องไม่เปลี่ยน
    return TestUtil.assert(state.sameState(h),"heap เปลี่ยน",1);
}
```

รหัสที่ 4.11 ตัวอย่างตัวตรวจการหาค่าน้อยอันดับสองของฮีป

4.8 สรุป

จากตัวอย่างที่ได้ผ่านมาเห็นได้ว่าเมื่อใช้ตัวควบคุมการตรวจและคลาสอรรถประโยชน์แล้วจะง่ายต่อการพัฒนาตัวตรวจ ทั้งยังทำให้ตัวตรวจสั้นลง เนื่องจากไม่ต้องเขียนในรหัสดังนี้

- การป้อนข้อมูลให้กับแป้นพิมพ์และอ่านผลจากจอภาพ ทำให้ไม่ต้องเขียนโปรแกรมในการจัดการเรื่องนี้เอง (หน้า 22)

- การสร้างข้อมูลแบบสุ่มเพื่อส่งให้โปรแกรมของผู้เรียน (หน้า 19)
- การเปรียบเทียบผลลัพธ์แบบประมาณและแบบแม่นตรง (หน้า 24)
- การตรวจเมท็อดที่คืนค่าเป็นบูลีน (หน้า 25)
- การทำสำเนาอ็อบเจกต์และการตรวจสอบการเปลี่ยนแปลงสถานะของอ็อบเจกต์ (หน้า 27)
- การอ่านค่าและเปลี่ยนค่าภายในอ็อบเจกต์ ทำให้ตรวจเมท็อดแบบไม่ต้องพึ่งกันและกันได้ (หน้า 30)
- การวงวนเพื่อทำการตรวจหลายครั้ง ทำให้โปรแกรมของผู้เรียนได้ทดสอบกับข้อมูลที่หลากหลาย (หน้า 36)
- การรักษาความมั่นคง (หน้า 35)
- แฟ้มข้อมูลทดสอบ ทำให้ผู้ออกโจทย์ไม่ต้องเขียนโปรแกรมหาผลเฉลย (หน้า 38)
- การห้ามใช้คลาสมาตรฐานที่มีอยู่ในจาวา (หน้า 40)

งานวิจัยนี้ได้เก็บข้อมูลตัวตรวจที่พัฒนาโดยไม่ใช้ตัวควบคุมการตรวจและคลาสอรรถประโยชน์กับตัวตรวจที่มีการใช้งานดังตารางที่ 4.1 ตัวตรวจแบบเดิมคือตัวตรวจที่ไม่มีการใช้ตัวควบคุมและคลาสอรรถประโยชน์ และตัวตรวจแบบใหม่คือตัวตรวจที่พัฒนาภายใต้ตัวควบคุมและคลาสอรรถประโยชน์ เห็นได้ว่าเมื่อมีการพัฒนาตัวตรวจภายใต้ตัวควบคุมการตรวจและคลาสอรรถประโยชน์ตัวตรวจจะสั้นลงถึง 48 เปอร์เซ็นต์ ทั้งในวิชาการเขียนโปรแกรมและวิชาโครงสร้างข้อมูล

	ตัวตรวจแบบเดิม	**ตัวตรวจแบบใหม่**
วิชาจาวา		
Email	116	43
Arrays	182	91
BMI	112	25
JEyes	145	75
Sudoku	174	145
วิชาโครงสร้างข้อมูล		
IntStack	617	218
ArrayList	304	97
List	139	118
Heap	202	140

ตารางที่ 4.1 ข้อมูลเปรียบเทียบบรรทัดของตัวตรวจ

บทที่ 5

สรุปผลการวิจัยและข้อเสนอแนะ

5.1 สรุปผลการวิจัย

งานวิจัยนี้ได้พัฒนาตัวควบคุมการตรวจและคลาสอรรถประโยชน์ เพื่อตัวตรวจโปรแกรมอัตโนมัติ โดยตัวควบคุมการตรวจมีหน้าที่ ควบคุมความมั่นคงจากการสั่งตัวตรวจทำงาน สั่งให้ตัวตรวจทำงานแบบแยกหน่วยประมวลย่อยเพื่อควบคุมเวลาการทำงานไม่ให้เกินเวลาที่จำกัดไว้ การใช้ annotation เพื่อกำหนดลักษณะกำกับการตรวจ การห้ามใช้คลาสมาตรฐานที่มีอยู่ในจาวา ลักษณะกำกับการตรวจระดับคลาส การตรวจที่มีกรณีย่อย การตรวจเมท็อดที่คืนค่าเป็นบูลีน การกำหนดการทำงานเมท็อด ก่อนเริ่มการตรวจและเมท็อดหลังจากการตรวจ

นอกจากนี้ยังมีคลาสอรรถประโยชน์ให้บริการต่าง ๆ ช่วยให้สามารถพัฒนาตัวตรวจได้ง่าย คลาสอรรถประโยชน์มีบริการดังนี้ การสร้างชุดข้อมูลทดสอบ การเปลี่ยนเส้นทางข้อมูลจากแป้นพิมพ์ การคัดแยกข้อมูล การเปรียบเทียบผลแบบแม่นตรงและประมาณทั้งสายอักขระและจำนวนจริง การแสดงผลการตรวจและสาเหตุความผิดพลาด การทำสำเนาและตรวจการเปลี่ยนสถานะของอ็อบเจกต์ การอ่านค่าและเปลี่ยนค่าภายในอ็อบเจกต์ การวัดเวลาทำงาน

เมื่อผู้ออกโจทย์พัฒนาตัวตรวจโดยใช้คลาสอรรถประโยชน์ร่วมกับตัวควบคุมการตรวจจะทำให้สามารถพัฒนาตัวตรวจได้ง่ายขึ้น ทั้งยังทำให้ตัวตรวจมีขนาดสั้นลง ทำให้ผู้ออกโจทย์ผลิตตัวตรวจได้จำนวนมาก และจะประโยชน์ต่อผู้เรียน

สำหรับรหัสที่สั้นลงของตัวตรวจจากการใช้ตัวควบคุมและคลาสอรรถประโยชน์ เกิดจากการจัดการของตัวควบคุมและคลาสอรรถประโยชน์ในส่วนต่าง ๆ ซึ่งผู้ออกโจทย์ไม่ต้องเขียนทำให้ตัวตรวจสั้นลงดังนี้

ตัวตรวจแบบเดิม	ตัวตรวจแบบใหม่
การรักษาความมั่นคง	
`System.setSecurityManager` `(new SecurityManager()` `{` `  public void checkExit(int status){` `  throw new SecurityException("exit");` `  }` `  public void checkPermission(` `  Permission perm) {` `  }` `  ...` `});`	**// ผู้เขียนออกโจทย์ไม่ต้องเขียนในส่วนนี้**

<table>
<tr><th>ตัวตรวจแบบเดิม</th><th>ตัวตรวจแบบใหม่</th></tr>
<tr><td colspan="2">การสร้าง thread เพื่อทำงาน</td></tr>
<tr><td><pre>
Thread thread = new Thread() {
  //เหมือนรหัสที่ 3.13 ซึ่งจะต้องจัดการเกี่ยวกับ
  //การให้คะแนนและ timeout เอง
  public void run() {
  ...
  }
};
thread.start();
</pre></td><td><pre>
// ผู้เขียนออกโจทย์ไม่ต้องเขียนในส่วนนี้
</pre></td></tr>
<tr><td colspan="2">การวงวนเพื่อเรียกตัวตรวจ</td></tr>
<tr><td><pre>
for(int i=0;i<10;i++) {
    testScript();
}
</pre></td><td><pre>
@Test(loops=10)
</pre></td></tr>
<tr><td colspan="2">การจัดการในตัวตรวจที่รับค่าจากแป้นพิมพ์และอ่านค่าจากจอภาพ</td></tr>
<tr><td><pre>
PrintStream stdOut = System.out;
PipedOutputStream pipe1Out = ...;
PipedOutputStream pipe2Out = ...;
PrintStream stream1Out = ...;
PrintStream stream2Out = ...;
InputStream stream1In = ...;
InputStream stream2In = ...;
System.setOut(stream1Out);
System.setIn(stream2In);
testScript(stream1In, stream2Out);
System.setOut(stdOut);
stream1In.close();
stream2Out.close();
</pre></td><td><pre>
TestUtil.writeSystemIn();
...
TestUtil.readSystemOut();
</pre></td></tr>
<tr><td colspan="2">การทำสำเนาอ็อบเจกต์</td></tr>
<tr><td><pre>
public static <T> T clone (T obj){
  //เหมือนรหัสที่ 3.7
  Class objClass = obj.getClass();
  T result;
  if (objClass.isArray()) {
    ...
  } else if (..indexOf("java.")!=-1){

  } else {
    return result;
}
</pre></td><td><pre>
TestUtil.saveState(h);
</pre></td></tr>
<tr><td colspan="2">การตรวจสอบการเปลี่ยนสถานะ</td></tr>
<tr><td><pre>
public boolean compareObject(Object
src, Object dest){
  //เหมือนรหัสที่ 3.8
  Class srcClass = src.getClass();
  Class destClass = dest.getClass();
  if(srcClass.isArray()) {
    ...
  } else if (..indexOf("java.")!=-1){
    ...
  } else {
    ...
  }
}
</pre></td><td><pre>
state.sameState(h)
</pre></td></tr>
</table>

ตารางที่ 5.1 ข้อมูลเปรียบเทียบตัวตรวจแบบเก่าและแบบใหม่

5.2 ข้อเสนอแนะ

1. งานวิจัยชิ้นนี้จะเน้นการตรวจโดยการสั่งให้ทำงานเพื่อตรวจสอบผลเพื่อตรวจความถูกต้องของโปรแกรมของผู้เรียน ทั้งยังสามารถวัดเวลาการทำงานของโปรแกรมได้อีกด้วย แต่จริง ๆ แล้ว การตรวจโปรแกรมยังสามารถใช้วิธีการตรวจโดยการวิเคราะห์รหัสต้นฉบับซึ่งสามารถตรวจรูปแบบการเขียนโปรแกรมว่าตรงตามธรรมเนียมปฏิบัติในวงการ ขึ้นอยู่กับความต้องการของผู้ตรวจโปรแกรม หากเป็นวิชาเขียนโปรแกรมเบื้องต้นอาจเน้นหนักที่การตรวจเพื่อความถูกต้อง แต่หากบางวิชาที่เน้นการเขียนโปรแกรมให้ตรงตามธรรมเนียมปฏิบัติในวงการก็จำเป็นต้องใช้การตรวจโดยการวิเคราะห์รหัสต้นฉบับรวม เพื่อสามารถตรวจรูปแบบการเขียนโปรแกรม
2. ควรมีบริการเสริมที่สามารถวัด ประสิทธิ์ภาพและหน่วยความจำจากการทำงานของโปรแกรมผู้เรียนได้ หรือบริการเสริมสำหรับวัดการออกแบบของโปรแกรมผู้เรียนเพื่อให้ผู้เรียนปรับปรุงการเขียนโปรแกรมได้ดีขึ้นในหลายรูปแบบ

รายการอ้างอิง

[1] Hollingsworth, J., (1960). Automatic graders for programming classe. Communication of ACM: 528-529.

[2] Ala-Mutka, K.M. (2005). A Survey of Automated Assessment Approaches for Programming Assignments. Computer Science Education 15: 83-102.

[3] Douce, C., Livingstone, D., and Orwell, J. (2005). Automatic Test-Based Assessment of Programming: A Review. ACM. J. of Educational Resources in Computing, 5:3.

[4] Higgins, C., Hergazy, T., Symeonidis, P., and Tsinsifas, A., (2003). The CourseMarker CBA system: Improvements over Ceilidh. Journal of Education and Information Technologies 8: 287 – 304.

[5] Foxley, E., Tsintsifas, A., Higgins, C., and Symeonidis P. (1999). Ceilidh, A system for the automatic evaluation of students' programming work. In Proceedings of the CBLISS 99 Conference (University of Twente, Holland, July 2 – 7).

[6] Truong, N., Roe, P., and Bancroft, P. (2005). Automated feedback for fill in the gap programming exercises. Proceedings of the 7th Australasian conference on Computing education - Volume 42.

[7] Jackson, D., and Usher, M. (1997). Grading Student Programs using ASSYST. Proceedings of the 28th SIGCSE technical symposium on Computer science education, USA: 335 – 339.

[8] Stephen H. Edwards. (2003).Teaching software testing: Automatic grading meets test-first coding. In Addendum to the 2003 Proceedings of the Conference on Object-oriented Programming, Systems, Languages, and Applications, ACM : 318-319.

[9] Tutnew memory management library. Available from: http://www.cs.tut.fi/~bitti [2009,May 11].

[10] Oman, P.W., and Cook, C.R. (1990). A taxonomy for programming style. In Proceedings of the 1990 ACM Annual Conference on Cooperation: 244 – 250.

[11] Schorsch, T. (1995). CAP: an automated self-assessment tool to check Pascal programs for syntax,logic and style errors. In Proceedings of the 26th SIGCSE technical symposium on Computer science education: 168 – 172.

[12] Mengel, S.A., and Ulans, J.V. (1999). A case study of the analysis of the quality of novice students programs. In Proceedings of the 12th Conference on Software Engineering Education and Training: 40 – 49.

[13] Truong, N., Roe, P., and Bancroft, P. (2003). A Web Based Environment for Learning to Program. In Proceedings of the 25th Australasian Computer Science Conference, Australia: 255 – 264.

[14] Saikkonen, R., Malmi, L., and Korhonen, A. (2001). Fully automatic assessment of programming exercises. Proceedings of 6th annual conference on Innovation and technology in computer science education, UK: 133 – 136.

[15] Cheang, B., Kurnia, A., Lim, A., and Oon, W.-C. (2003). On automated grading of Programming Assignments in an academic institution. Computers & Education, 41: 121 – 131.

[16] JUnit test, Available from: http://www.junit.org [2009,May 11].

[17] JUnit test. Available from: http://www.ibm.com/developerworks/java/library/junit4.html [2009,May 11].

[18] Program of University. Available from: http://www.cs.princeton.edu/courses/archive/spring09/cos226/assignments.html [2009,May 11]

[19] Program of University. Available from: http://www.comp.nus.edu.sg/~cs1101x/3_ca/labs.html [2009,May 11]

[20] Program of University. Available from: http://www.cp.eng.chula.ac.th/~somchai/JLab [2009, Jun 28]

[21] Program of University. Available from: http://www.cs.jhu.edu/~phf/2009/spring/cs107 [2009, Jun 28]

[22] Program of University. Available from: http://www.javabat.com [2009, Jun 28]

[23] Relative error. Available from: http://www.cygnus software.com/papers/comparingfloats/Comparing%20floating%20point%20numbers.htm [2009, Jun 28]

[24] Java. Available from: http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html [2009,May 11].

[25] Edit distance, Available from: http://en.wikipedia.org/wiki/Levenshtein_distance [2009,May 11].

[26] Java, Available from: http://java.sun.com/docs/books/tutorial/essential/concurrency/index.html [2009,May 11].

[27] Clone Object, Available from: http://weblogs.java.net/blog/emcmanus/archive/2007/04/cloning_java_ob.html [2009,May 11].

[28] BOSS, Available from: http://boss.org.uk [2004, Dec-19]

[29] Java Reflection, Available from: http://java.sun.com/docs/books/tutorial/reflect [2004, Dec-19]

[30] Java, Available from: http://java.sun.com/j2se/1.5.0/docs/guide/language/autoboxing.html [2004, Dec-19]

ภาคผนวก

ภาคผนวก ก
ส่วนต่อประสานโปรแกรมประยุกต์ของจาวาที่ใช้ในงานวิจัยนี้

งานวิจัยนี้ เป็นการพัฒนาตัวควบคุมการตรวจและคลาสอรรถประโยชน์โดยมีการใช้คุณสมบัติต่าง ๆ ของจาวาเพื่อให้ตัวตรวจเขียนได้ง่าย ๆ

1 Annotation [21]

ก่อนจะกล่าวถึง annotation จะขอกล่าวถึงภาษาเชิงประกาศ (declarative language) ก่อนเนื่องจาก annotation ใช้หลักการของภาษาเชิงประกาศ คือประกาศว่าจะสร้างมันได้อย่างไร เช่น เอชทีเอ็มแอล (HTML) เป็นภาษาเชิงประกาศ เนื่องจากเราประกาศว่าหน้านั้นประกอบไปด้วยอะไรบาง เช่น ข้อความ ภาพ เป็นต้น แต่ไม่ได้บอกว่าจริง ๆ แล้วสร้างเช่นไร ซึ่งต่างจาก ภาษาเชิงคำสั่ง (imperative language) เช่น ภาษาฟอร์แทรน (Fortran) ภาษาซี และ จาวา ซึ่งต้องให้ผู้เขียนบอกว่าใช้ชุดคำสั่งใดทำงาน อาจกล่าวได้ว่าภาษาเชิงคำสั่ง นั้นต้องบอกแต่ละคำสั่งให้แน่ชัดภาษาเชิงประกาศ บอกเป้าหมายแต่ไม่ได้บอกว่าใช้ชุดคำสั่งใดในการสร้าง

จาวารองรับการใช้งาน annotation ตั้งแต่จาวารุ่นที่ 5 โดย annotation เป็นการบอกข้อมูลเกี่ยวกับโปรแกรมซึ่งไม่ใช่ส่วนหนึ่งของโปรแกรมของมันเอง และ annotation ไม่ได้มีผลโดยตรงกับรหัสที่ annotation ทำการอธิบายไว้

ก.1.1 การเรียกใช้งาน annotation

การเรียกใช้ annotation สำหรับการแปลโปรแกรมโดย annotation สามารถใช้เพื่อให้ตัวแปลภาษา บอกข้อผิดพลาดของโปรแกรม ตัวอย่างการใช้ annotation เพื่อบอกข้อผิดพลาดของโปรแกรม คือ `@Override` ซึ่งเป็น annotation เพื่อเป็นข้อมูลให้กับตัวแปลภาษาว่าเมท็อดนี้มีการ override จากคลาสแม่จะเห็นการใช้ `@Override` ได้จากรหัสที่ ก-1

```
// mark method as a superclass method
// that has been overridden
@Override
int overriddenMethod() { }
```

รหัสที่ ก-1 การ override เมท็อด overriddenMethod โดยมีการใช้ annotation

การใส่ `@Override` ไว้บนหัวเมท็อดของคลาสลูกที่มีการ override นั้นจะทำให้ตัวแปลภาษาสามารถตรวจสอบได้ว่ามีเมท็อดที่มีชื่อและพารามิเตอร์อยู่ที่คลาสแม่หรือไม่ หากไม่มีก็อาจเป็นได้ว่าผู้เขียนมีการเขียนชื่อเมท็อดของคลาสลูกผิด ตัวแปลภาษาจะแจ้งให้ผู้เขียนทราบว่ามีข้อผิดพลาดเกิดขึ้น

ก.1.2 ตัวอย่างการใช้งาน annotation

การสร้าง annotation นั้นง่ายมาก โดยใช้รูปแบบเดียวกับการสร้างอินเตอร์เฟส ดังรหัสที่ ก-2 และสามารถเรียกใช้ annotation ที่สร้างได้ดังรหัสที่ ก-3

```
package com.chula.jtest101.tester;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
@Retention(RUNTIME)
public @interface Test {
    String name() default "";
    int loops() default 10;
    int timeout() default 100;
}
```

รหัสที่ ก-2 การสร้าง annotation `Test`

```
public class TestBuy5Get1 {
    @Test(loops = 10, points = 1,timeout=1000)
    public double testBuy5Get1() {
        ...
        return 0;
    }
}
```

รหัสที่ ก-3 การเรียกใช้ annotation

เราสามารถสร้าง annotation ได้ดังรหัสที่ ก-2 ซึ่งมีการสร้าง `@Test` มีสมาชิกเป็น `name` `loops` และ `timeout` ซึ่งกำหนดให้คุณสมบัติของ annotation มีจนถึงเวลาดำทำงานของโปรแกรม `@Retention(RUNTIME)` ส่วนการเรียกใช้งานนั้นสามารถทำได้ดังรหัสที่ ก-3 ซึ่งใช้ `@Test` ไว้บนหัวของเมท็อด `testBuy5Get1` ซึ่งทำให้สามารถดึงข้อมูล `@Test` ของเมท็อด `testBuy5Get1` ได้โดยรีเฟลคชัน

2 รีเฟลคชัน

รีเฟลคชัน หมายถึงภาพสะท้อนในกระจก โดยทั่วไปแล้วการเขียนโปรแกรมมีการใช้รีเฟลคชันสำหรับโปรแกรมที่ต้องการทดสอบและแก้ไขพฤติกรรมของโปรแกรมขณะที่โปรแกรมกำลังทำงาน โดยรีเฟลคชันทำให้เราสามารถดึงข้อมูล แก้ไขข้อมูลของคลาส ฟิลด์และเมท็อดผ่านทางจาวา API ดังในรหัสที่ ก-5 จะเห็นว่าเมท็อด `getTestMethod` มีการรับพารามิเตอร์เป็นอ็อบเจกต์ `Tester` จากนั้นนำข้อมูลคลาสของ `Tester` ออกมาโดยใช้ `tester.getClass()` แล้วจะดึงข้อมูลของเมท็อดที่อ็อบเจกต์นั้นมีโดยใช้ `c.getDeclaredMethods()` และตรวจสอบดูว่าเมท็อดนั้นมี `@Test` ที่เมท็อดหรือไม่โดยใช้ `m.isAnnotationPresent(Test.class)` ถ้าเมท็อดนั้นมี `@Test` จะทำการเพิ่มข้อมูลใน `lst` และทำการคืนค่า `lst` กลับไป

```
public List getTestMethod(Tester tester) {
     Class c = tester.getClass();
     List lst = new ArrayList();
     for( Method m : c.getDeclaredMethods()) {
         if (m.isAnnotationPresent(Test.class)) {
           lst.add(m);
         }
     }
     return lst;
}
```

รหัสที่ ก-4 การดึงเมท็อดที่มี `@Test` ของอ็อบเจกต์ `Tester`

3 ภาวะพร้อมกัน (Concurrency) [26]

เทรดพูล (Thread pool) เป็นการทำงานรวมกันของเทรดหลายตัว เทรดพูลเมื่อมีเทรดใดหยุดทำงานระบบจะนำเทรดตัวใหม่เข้ามาทำงานแทนโดยอัตโนมัติ จะมีเทรดที่กำลังทำงานอยู่ในพูลตามจำนวนที่ผู้ใช้กำหนดไว้ ส่วนที่เกินมาระบบจะนำเทรดเหล่านั้นมารอเข้าแถวคอยเพื่อรอเข้าไปในพูล จาวา5เป็นเวอร์ชันแรกที่เริ่มเปิดให้ใช้เทรดพูล ผู้ใช้สามารถสร้างเทรดพูลในลักษณะต่าง ๆ ได้ดังนี้

- สร้างโดยการให้ระบบสร้างพูล

จาวาเทรดพูลสามารถสร้างพูลโดยให้ระบบสร้างพูลให้เอง โดยระบบจะขยายและลดพูลลงได้เองโดยใช้ `Executors.newCachedThreadPool()` เทรดจะมีการใช้ทรัพยากรของเทรดก่อนหน้า หากเทรดใดไม่มีการทำงานภายใน 60 วินาทีจะถูกทำลายและนำออกจากที่เก็บ

```
public static void main(String[] args) {
     ExecutorService exec = Executors.newCachedThreadPool();
     exec.execute(new Runnable() {
          @Override
          public void run() {
               System.out.println("thread pool...");
          }
     });
     exec.shutdown();
}
```

รหัสที่ ก-5 การสร้างเทรดโดยให้จาวาสร้างพูลให้

- สร้างโดยการระบุจำนวนที่แน่นอนของพูล

พูลในลักษณะนี้ ผู้ใช้จะระบุจำนวนที่แน่นอนของการสร้างพูลซึ่งตลอดการทำงานขนาดของพูลจะมีจำนวนเท่ากับที่ผู้ใช้ระบุไว้ การสร้างพูลเช่นนี้สามารถใช้ได้ดีกับระบบที่ต้องการจำกัดจำนวนของการสร้างเทรดเพื่อจำกัดทรัพยากรโดยใช้ `Executors.newFixedThreadPool(10)`

ภาคผนวก ข
ลักษณะกำกับการตรวจในคลาสอรรถประโยชน์

@Test เป็นตัวกำกับคุณสมบัติของตัวตรวจ ซึ่งสามารถสั่งให้ตัวควบคุมการตรวจทำงานได้ งานวิจัยนี้ได้ทำการอธิบายรายละเอียดของต่าง ๆ ของตัวควบคุมการตรวจและคลาสอรรถประโยชน์ไปแล้ว ในภาคผนวกนี้จึงของรวบรวมและสรุปส้น ๆ ของ @Test โดยจะมีดังนี้

ประเภท	ชื่อ	คำอธิบาย
`String`	`name`	ชื่อเมท็อดตัวตรวจ
`CalculationType`	`calcType`	ชนิดของการคำนวณคะแนน
`int`	`loops`	จำนวนรอบของการทำงาน
`int`	`timeout`	กำหนดเวลาของการทำงานของตัวตรวจ
`int`	`points`	คะแนนต่อรอบ
`String`	`testData`	ตำแหน่งของแฟ้มข้อมูลขาเข้าและข้อมูลที่ถูกต้อง
`String[]`	`prohibit`	คำที่ห้ามใช้
`Class []`	`studentClass`	คลาสของผู้เรียน ซึ่งใช้ค้นหาคำที่ห้ามใช้
`TestScriptSecurityManager[]`	`openTestscriptSecurity`	การเปิดความมั่นคงเพิ่มเติมหากผู้ออกโจทย์ต้องการ

ตารางที่ ข-1 Annotation ทั้งหมดของ `@Test`

เนื่องจากคลาสอรรถประโยชน์มีจำนวนมากและยังมีความคล้ายคลึงกันดังนั้นจึงขอแสดงคลาสอรรถประโยชน์ทั้งหมดมี ดังต่อไปนี้

`int random()`	
หน้าที่	คืนค่าของจำนวนเต็มแบบสุ่ม
ผลที่คืน	จำนวนเต็มแบบสุ่ม
`double randomDouble()`	
หน้าที่	คืนค่าของจำนวนจริงแบบสุ่ม
ผลที่คืน	จำนวนจริงแบบสุ่ม
`int random( int lower, int upper )`	
หน้าที่	คืนค่าของจำนวนเต็มแบบสุ่มมีค่าระหว่าง `lower` ถึง `upper`
พารามิเตอร์	`lower` - ค่าน้อยที่สุดที่ต้องการของการสุ่ม `upper` - ค่ามากที่สุดที่ต้องการของการสุ่ม
ผลที่คืน	จำนวนเต็มแบบสุ่ม
`double random(double lower, double upper )`	
หน้าที่	คืนค่าของจำนวนจริงแบบสุ่มมีค่าระหว่าง `lower` ถึง `upper`
พารามิเตอร์	`lower` - ค่าน้อยที่สุดที่ต้องการของการสุ่ม `upper` - ค่ามากที่สุดที่ต้องการของการสุ่ม
ผลที่คืน	จำนวนจริงสุ่มแบบสุ่ม
`int[] fillRandom(int [] d, int lower, int upper)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนเต็มแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยมีค่าระหว่าง `lower` ถึง `upper`
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า `lower` - ค่าน้อยที่สุดที่ต้องการในแถวลำดับ `upper` - ค่ามากที่สุดที่ต้องการในแถวลำดับ

ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนจริงแบบสุ่ม
`double[] fillRandom(double [] d, double lower, double upper)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนจริงแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยมีค่าระหว่าง `lower` ถึง `upper`
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า `lower` - ค่าน้อยที่สุดที่ต้องการในแถวลำดับ `upper` - ค่ามากที่สุดที่ต้องการในแถวลำดับ
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนเต็มแบบสุ่ม
`int[] fillRandomDistinct(int [] d, int lower, int upper)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนเต็มแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยมีค่าระหว่าง `lower` ถึง `upper` โดยค่าจะไม่ซ้ำกัน
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า `lower` - ค่าน้อยที่สุดที่ต้องการในแถวลำดับ `upper` - ค่ามากที่สุดที่ต้องการในแถวลำดับ
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนเต็มแบบสุ่ม
`double[] fillRandomDistinct(double[] d, double lower, double upper)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนจริงแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยมีค่าระหว่าง `lower` ถึง `upper` โดยค่าจะไม่ซ้ำกัน
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า `lower` - ค่าน้อยที่สุดที่ต้องการในแถวลำดับ `upper` - ค่ามากที่สุดที่ต้องการในแถวลำดับ
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนจริงแบบสุ่ม
`int[] fillRandomContain(int[] d, int [] contain)`	

หน้าที่	คืนค่าแถวลำดับของจำนวนเต็มแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ d โดยค่าที่สุ่มคือค่าภายในแถวลำดับ contain
พารามิเตอร์	d - แถวลำดับที่ต้องการให้เติมค่า contain - แถวลำดับที่ต้องการให้สุ่มค่าเพื่อเติมลง d
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนเต็มแบบสุ่ม
double[] fillRandomContain(double[] d, double [] contain)	
หน้าที่	คืนค่าแถวลำดับของจำนวนจริงแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ d โดยค่าที่สุ่มคือค่าภายในแถวลำดับ contain
พารามิเตอร์	d - แถวลำดับที่ต้องการให้เติมค่า contain - แถวลำดับที่ต้องการให้สุ่มค่าเพื่อเติมลง d
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนจริงแบบสุ่ม
String randomString()	
หน้าที่	คืนค่าสายอักขระแบบสุ่ม โดยระบบจะมีการกำหนดความยาวของสายอักขระให้เอง
ผลที่คืน	สายอักขระแบบสุ่ม
String randomString(int length)	
หน้าที่	คืนค่าสายอักขระแบบสุ่ม ซึ่งมีความยาว length
พารามิเตอร์	length - ค่าความยาวของสายอักขระที่ต้องการให้สร้าง
ผลที่คืน	สายอักขระแบบสุ่มความยาว length
String randomStringContain(char… c)	
หน้าที่	คืนค่าสายอักขระแบบสุ่ม ซึ่งประกอบไปด้วยอักขระภายใน c
พารามิเตอร์	c - แถวลำดับของอักขระที่ต้องการให้สร้างเป็นสายอักขระ
ผลที่คืน	สายอักขระแบบสุ่ม
String radomStringDistinct()	

หน้าที่	คืนค่าสายอักขระแบบสุ่ม ซึ่งอักขระไม่ซ้ำกันเลย
ผลที่คืน	สายอักขระแบบสุ่ม
`String[] fillRandom(String[] d)`	
หน้าที่	คืนค่าแถวลำดับของสายอักขระแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d`
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า
ผลที่คืน	แถวลำดับ `d` ซึ่งภายในมีจำนวนเต็มสุ่ม
`String[] fillRandomDistinct(String[] d)`	
หน้าที่	คืนค่าแถวลำดับของสายอักขระแบบสุ่ม ซึ่งจะเติมค่าให้แถวลำดับ `d` โดยค่าจะไม่ซ้ำกัน
พารามิเตอร์	`d` - แถวลำดับที่ต้องการให้เติมค่า
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยสายอักขระแบบสุ่ม
`double editDistance(String src, String dest)`	
หน้าที่	คืนค่าผิดพลาดสัมพัทธ์ระหว่างสายอักขระ `source` และ `dest`
พารามิเตอร์	`src` – สายอักขระต้นฉบับที่ต้องการวัดระยะห่าง `dest` - สายอักขระเป้าหมายที่ต้องการวัดระยะห่าง
ผลที่คืน	ค่าผิดพลาดสัมพัทธ์ของสองสายอักขระ
`Object getField(Object obj, String name)`	
หน้าที่	คืนค่าข้อมูลของ name ที่อยู่ภายในอ็อบเจกต์ `obj`
พารามิเตอร์	`obj` - อ็อบเจกต์ที่ต้องการค่า `name` – ชื่อของข้อมูลที่ต้องการค่า
ผลที่คืน	ค่าของข้อมูล `name` ภายในอ็อบเจกต์ `obj`
`void setField(Object obj, String name, Object value)`	
หน้าที่	กำหนดค่า `value` ให้กับข้อมูล `name` ในอ็อบเจกต์ `obj`

พารามิเตอร์	`obj` - อ็อบเจกต์ที่ต้องการกำหนดค่า `name` – ชื่อของข้อมูลที่ต้องการกำหนดค่า `value` – ค่าที่ต้องการกำหนดให้ข้อมูล name
`double[] string2double(String [] strList)`	
หน้าที่	คืนค่าแถวลำดับของจำนวนจริง ซึ่งได้มาจากการเปลี่ยนแถวลำดับของสายอักขระ `strList` ให้เป็นแถวลำดับของจำนวนจริง
พารามิเตอร์	`strList` - แถวลำดับของสายอักขระที่ต้องการเปลี่ยนเป็นแถวลำดับของจำนวนจริง
ผลที่คืน	แถวลำดับซึ่งภายในประกอบด้วยจำนวนจริง
`void writeSystemIn(String str)`	
หน้าที่	เขียนข้อมูลลงในกระแสข้อมูล เพื่อใช้ในการตรวจโปรแกรมที่รับค่าจากแป้นพิมพ์
พารามิเตอร์	`str` - ข้อมูลที่ต้องการเขียนลงในกระแสข้อมูล
`String[] readSystemOut()`	
หน้าที่	คืนค่าข้อมูลภายในกระแสข้อมูล ใช้ในการนำข้อมูลหลังจากการสั่งดำเนินการโปรแกรมของผู้เรียน เพื่อใช้ในการเปรียบเทียบกับผลเฉลย
ผลที่คืน	ข้อมูลภายในกระแสข้อมูล
`String[] filter(String[] in, int startWithLine, String strStart)`	
หน้าที่	คืนค่าแถวลำดับซึ่งคัดแยกข้อมูล โดยสามารถระบุตำแหน่งและสายอักขระซึ่งให้เริ่มทำการคัดแยกข้อมูล
พารามิเตอร์	`in` - แถวลำดับที่ต้องการให้คัดแยกข้อมูล `startWithLine` - ตำแหน่งภายในแถวลำดับที่เริ่มให้คัดแยกข้อมูล `strStart` - สายอักขระที่ต้องการให้คัดแยกข้อมูลภายในข้อมูลของแถวลำดับ
ผลที่คืน	สายอักขระซึ่งผ่านการคัดแยกข้อมูลแล้ว
`boolean equals(double source, double dest, double distance)`	

หน้าที่	คืนค่าบูลีนที่ได้จากการเปรียบเทียบแบบประมาณของจำนวนจริง `source` และ `dest` โดยหากค่าการเปรียบเทียบแบบประมาณน้อยกว่าค่า `distance` จะคืนค่าจริง หากมากกว่าจะคืนค่าเท็จ
พารามิเตอร์	`source` - จำนวนจริงต้นฉบับที่ต้องการให้เปรียบเทียบแบบประมาณ `dest` - จำนวนจริงเป้าหมายที่ต้องการให้เปรียบเทียบแบบประมาณ `distance` - ค่าประมาณของการเปรียบเทียบ
ผลที่คืน	ค่าบูลีนที่ได้จากการเปรียบเทียบ
`boolean equals(Object[] soruce, int sourceStart, Object[] dest, int destStart, int length)`	
หน้าที่	คืนค่าบูลีนที่ได้จากการเปรียบเทียบแถวลำดับ `source` ตำแหน่ง `sourceStart` กับแถวลำดับ `dest` ตำแหน่ง `destStart` ยาว `length`
พารามิเตอร์	`source` - แถวลำดับต้นฉบับของการเปรียบเทียบ `start` - ตำแหน่งเริ่มต้นของแถวลำดับ source `dest` - แถวลำดับเป้าหมายที่ต้องการเปรียบเทียบ `destStart` - ตำแหน่งเริ่มต้นของแถวลำดับ dest `length` - ความยาวของข้อมูลที่ต้องการเปรียบเทียบ
ผลที่คืน	ค่าบูลีนที่ได้จากการเปรียบเทียบ
`double testBooleanMethod(BooleanMethod bm)`	
หน้าที่	คืนค่าระหว่าง 0 ถึง 1 เกิดจากการตรวจสอบเมท็อดที่คืนค่าเป็นบูลีน โดย 0 คือผิดทุกกรณี 1 คือถูกทุกกรณี
พารามิเตอร์	`BooleanMethod` - เป็นอินเทอร์เฟซที่ผู้ใช้ต้องสร้างเมท็อด `isTrue` และ `isFalse`
ผลที่คืน	ค่า 0 ถึง 1 เกิดจากการตรวจสอบเมท็อดที่คืนค่าเป็นบูลีน
`List<String> getVectorInput()`	

หน้าที่	คืนค่ารายการข้อมูลทดสอบ เพื่อส่งให้โปรแกรมของผู้เรียน ซึ่งจะได้มาจากแฟ้มข้อมูล โดยชื่อแฟ้มรายการของข้อมูล ผู้ใช้ต้องกำหนดผ่าน annotation
ผลที่คืน	รายการของข้อมูลทดสอบ
`List<String> getVectorInput(String fileName)`	
หน้าที่	คืนค่ารายการของข้อมูลทดสอบ เพื่อส่งให้โปรแกรมของผู้เรียน ซึ่งจะได้มาจากแฟ้มข้อมูล
พารามิเตอร์	`filename` - ชื่อแฟ้มที่เก็บข้อมูล
ผลที่คืน	รายการของข้อมูลขาเข้า
`List<String> getVectorResult()`	
หน้าที่	คืนค่ารายการของผลลัพธ์ เพื่อนำไปเปรียบเทียบกับผลจากการทำงานของโปรแกรมของผู้เรียน ซึ่งจะได้มาจากแฟ้มข้อมูล โดยชื่อแฟ้มผู้ใช้ต้องกำหนดผ่าน annotation
ผลที่คืน	รายการของผลลัพธ์
`List<String> getVectorResult(String fileName)`	
หน้าที่	คืนค่ารายการของผลลัพธ์ เพื่อนำไปเปรียบเทียบกับผลจากการทำงานของโปรแกรมของผู้เรียน ซึ่งจะได้มาจากแฟ้มข้อมูล
พารามิเตอร์	`filename` - ชื่อแฟ้มที่เก็บข้อมูล
ผลที่คืน	รายการของผลลัพธ์
`List<int[][]> getIntMatrices()`	
หน้าที่	คืนค่ารายการของแถวลำดับ ของข้อมูลจำนวนเต็มแบบ 2 มิติ ซึ่งจะได้มาจากแฟ้มข้อมูล
ผลที่คืน	รายการของผลลัพธ์
`List<double[][]> getDoubleMatrices()`	
หน้าที่	คืนค่ารายการของแถวลำดับ ของข้อมูลจำนวนจริงแบบ 2 มิติ ซึ่งจะได้มาจากแฟ้มข้อมูล

ผลที่คืน	รายการของผลลัพธ์
`List<int[][]> getIntMatrices(String fileName)`	
หน้าที่	คืนค่ารายการของแถวลำดับ ของข้อมูลจำนวนเต็มแบบ 2 มิติ ซึ่งจะได้มาจากแฟ้มข้อมูล
พารามิเตอร์	`filename` - ชื่อแฟ้มที่เก็บข้อมูล
ผลที่คืน	รายการของผลลัพธ์
`List<double[][]> getDoubleMatrices(String fileName)`	
หน้าที่	คืนค่ารายการของแถวลำดับ ของข้อมูลจำนวนจริงแบบ 2 มิติ ซึ่งจะได้มาจากแฟ้มข้อมูล
พารามิเตอร์	`filename` - ชื่อแฟ้มที่เก็บข้อมูล
ผลที่คืน	รายการของผลลัพธ์
`void assertTrue(boolean val)`	
หน้าที่	เปรียบเทียบค่า `val` หากเป็นเท็จจะโยน exception `AssertionError`
พารามิเตอร์	`val` – ค่าบูลีนที่ต้องการให้เปรียบเทียบ
`void assertTrue(boolean val,String msg)`	
หน้าที่	เปรียบเทียบค่า `val` หากเป็นเท็จจะโยน exception `AssertionError` และแสดงข้อความ `msg` ออกทางจอภาพ
พารามิเตอร์	`val` - ค่าที่ต้องการให้เปรียบเทียบ `msg` - ข้อความที่ต้องการแสดงออกทางจอภาพหาก `val` ป็นเท็จ
`void assertTrue(Object src, Object dest)`	
หน้าที่	เปรียบเทียบค่าระหว่าง `src` และ `dest` หากเป็นเท็จจะโยน exception `AssertionError` โดย `src` และ `dest` สามารถเป็นค่าแถวลำดับได้เช่น `int[]`, `double[]` เป็นต้น
พารามิเตอร์	`src` - อ็อบเจกต์ต้นฉบับที่ต้องให้เปรียบเทียบ

	dest - อ็อบเจกต์เป้าหมายที่ต้องให้เปรียบเทียบ
void assertTrue(Object src, Object dest, String msg)	
หน้าที่	เปรียบเทียบค่าระหว่าง src และ dest หากเป็นเท็จจะโยน exception AssertionError และแสดงข้อความ msg ออกทางจอภาพ โดย src และ dest สามารถเป็นค่าแถวลำดับได้เช่น int[], double[] เป็นต้น
พารามิเตอร์	src - อ็อบเจกต์ต้นฉบับที่ต้องให้เปรียบเทียบ dest - อ็อบเจกต์เป้าหมายที่ต้องให้เปรียบเทียบ msg – ข้อความที่ต้องการแสดงออกทางจอภาพหาก src และ dest ไม่เท่ากัน
double assertTrue(boolean val, String msg, double p)	
หน้าที่	คืนค่าของ p และจะแสดงข้อความ msg ออกทางจอภาพ หาก val เป็นเท็จ
พารามิเตอร์	val - ค่าบูลีนที่ต้องการเปรียบเทียบ msg - ข้อความที่ต้องการแสดงออกทางจอภาพหาก val เป็นเท็จ p - คะแนนที่จะคืนค่าให้
ผลที่คืน	ค่าของ p
double assertTrue(boolean val,String msg,double point,String studentAns,String ans)	
หน้าที่	คืนค่าของ p และจะแสดงข้อความ msg ออกทางจอภาพ หาก val เป็นเท็จ และยังแสดง studentAns ซึ่งเป็นคำตอบของผู้เรียน และ ans ซึ่งเป็นคำตอบที่ถูกต้องออกทางจอภาพ
พารามิเตอร์	val - ค่าบูลีนที่ต้องการเปรียบเทียบ msg - ข้อความที่ต้องการแสดงออกทางจอภาพหาก val เป็นเท็จ p - คะแนนที่จะคืนค่าให้ studentAns - เป็นคำตอบของผู้เรียน

	ans - เป็นคำตอบที่ถูกต้อง
ผลที่คืน	ค่าของ p

ประวัติผู้เขียนวิทยานิพนธ์

นายเฉลิมวุฒิ น้อยอุ่นแสน เกิดวันที่ 12 สิงหาคม พ.ศ. 2524 ที่จังหวัดขอนแก่น สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต (วศ.บ.) สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยขอนแก่น เมื่อปีการศึกษา 2546 และเข้าศึกษาต่อหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2550