# การพัฒนาดีบักเกอร์ภาษาจาวา

# Java Debugger Development

นางสาวจรณะ ธงภักดิ์

นางสาวชุติกา อุดมสิน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

# การพัฒนาดีบักเกอร์ภาษาจาวา

# Java Debugger Development

โดย

นางสาวจรณะ ธงภักดิ์

นางสาวชุติกา อุดมสิน

อาจารย์ที่ปรึกษา

ดร. วิศิษฏ์ หิรัญกิตติ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาดีบักเกอร์ภาษาจาวา

Java Debugger Development

ผู้จัดทำ

1. นางสาวจรณะ ธงภักดิ์ รหัสประจำตัว 39014066
2. นางสาวชุติกา อุดมสิน รหัสประจำตัว 39014134

อาจารย์ที่ปรึกษา

(ดร. วิศิษฏ์ หิรัญกิตติ)

# การพัฒนาดีบักเกอร์ภาษาจาวา

นางสาวจรณะ ธงภักดิ์ 39014066
นางสาวชุติกา อุดมสิน 39014134
ดร. วิศิษฏ์ หิรัญกิตติ อาจารย์ที่ปรึกษา
ปีการศึกษา 2542

## บทคัดย่อ

ปริญญานิพนธ์นี้ทำการพัฒนาดีบักเกอร์ชื่อว่า "KMITL Java Developer " ขึ้นมาในลักษณะที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส (Graphic User Interface) ที่มีฟังก์ชันพื้นฐานในการดีบักได้แก่ สเตป(Step) , เซตเบรคพ้อยท์ (Set Breakpoint),เคลียร์เบรคพ้อยท์ (Clear Breakpoint),เรียกดูค่าตัวแปร (Add Watch), แสดงเธรดที่กำลังทำงาน โดยดีบักเกอร์นี้สร้างมาจากเจพีดีเอ (JPDA: Java Platform Debugger Architecture) ซึ่งเป็นสถาปัตยกรรมที่บริษัท ซันไมโครซิสเต็มส์ มีให้สำหรับผู้พัฒนาดีบักเกอร์ ไว้เขียนโปรแกรมดีบักเกอร์ให้สามารถทำการติดต่อกับจาวาเวอร์ชวลแมชชีน(Java Virtual Machine) เพื่อที่จะนำข้อมูลสำหรับการดีบักออกมาใช้ในการดีบักโปรแกรมภาษาจาวา

นอกจากนี้ยังมีการเพิ่มเติมในส่วนสภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวาให้มีส่วนของอีดิทเตอร์(Editor) การคอมไพล์ และการเอ็กซีคิวต์ด้วย เพื่อที่จะช่วยให้ผู้เขียนโปรแกรมสามารถสร้างโปรแกรมขึ้นมาใหม่หรือทำการเรียกโปรแกรมที่มีอยู่ขึ้นมาแก้ไขโปรแกรมได้สะดวกและรวดเร็วยิ่งขึ้น

# Java Debugger Development

Jarana Thongpak
Chutika Udomsinn
Dr. Visit Hirankitti Advisor
1999

## Abstract

We developed an easy-to-use debugger with some basic debugging functions: Step, Next, Run, Set and Clear Breakpoint, and Add Watch function. This debugger, namely "KMITL Java Developer", is built upon Sun Microsystems' JPDA (Java Platform Debugger Architecture) which enables an access to Java Virtual Machine in order to fetch information about program execution that essential for debugging.

Beside the debugging feature, KMITL Java Developer also includes editing, compiling and executing features which help programmer develop Java program easily and rapidly.

# สารบัญ

# สารบัญตารางและสารบัญรูปภาพ

# บทที่ 1
# บทนำ

## 1.1 ความสำคัญและที่มา

ปัจจุบันภาษาจาวาเป็นภาษาที่ใช้ในการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) ที่ได้รับความนิยมเป็นอย่างมาก เนื่องจากความสามารถในด้านที่เป็นอิสระไม่ขึ้นกับแพลตฟอร์มใด ๆ (Platforms Independents) เพราะมีเวอร์ชวลแมชชีน ทำให้สามารถที่จะนำโปรแกรมไปใช้งานในคอมพิวเตอร์เครื่องใดก็ได้ และยังมีความสามารถสร้างโปรแกรมที่ทำงานในลักษณะที่เป็นมัลติเธรด (Multithreads) ซึ่งไม่เหมือนกับภาษาอื่น ๆ ทำให้ปัจจุบันมีความนิยมใช้ภาษานี้ในการพัฒนาโปรแกรมไปในหลายด้าน เช่น การเขียนโปรแกรมประยุกต์ทั่วไป, การเขียนโปรแกรมบนระบบเครือข่าย, การนำไปเขียนโปรแกรมควบคุมฮาร์ดแวร์ (Hardware)

นอกจากนี้จาวายังมีลักษณะที่เป็นทั้งคอมไพล์เลอร์(Compiler)และอินเตอร์พรีทเตอร์(Interpreter) ในส่วนที่เป็นอินเตอร์พรีทเตอร์จะมีลักษณะที่จะทำให้สามารถนำมาสร้างดีบักเกอร์ได้สะดวกยิ่งขึ้น ดังนั้นการที่จะสร้างดีบักเกอร์ (Debugger) ของภาษาจาวาจึงมีความน่าสนใจและจะช่วยสนับสนุนให้การพัฒนาโปรแกรมภาษาจาวามีความสะดวกมากยิ่งขึ้น

ปัจจุบันเครื่องมือที่ใช้ในการพัฒนาภาษาจาวายังมีไม่มากนัก โดยเฉพาะเครื่องมือในการตรวจสอบความผิดพลาดของโปรแกรมหรือดีบักเกอร์ การสร้างดีบักเกอร์ที่มีอยู่ในปัจจุบันสร้างมาจาก เจพีดีเอ (Java Platform Debugger Architecture : JPDA) ซึ่งเป็นสถาปัตยกรรมที่บริษัท ซันไมโครซิสเต็มส์ สร้างขึ้นมาเพื่ออำนวยความสะดวกในการติดต่อกับจาวาเวอร์ชวลแมชชีน เพื่อที่จะนำข้อมูลสำหรับการดีบักออกมาใช้

ในช่วงแรก ๆ การสร้างดีบักเกอร์ จะมีลักษณะที่ใช้เท็กซ์โหมด(Text mode) คือ เจดีบี(JDB:Java Debugger) ของบริษัท ซันไมโครซิสเต็มส์ ที่มีอยู่ในชุดพัฒนาจาวาหรือที่เรียกว่าเจดีเค(JDK : Java™ Development Kit) ดังนั้นการพัฒนาให้มีลักษณะที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส จะทำให้ผู้ใช้สามารถใช้งานได้สะดวกยิ่งขึ้น ปัจจุบันโปรแกรมในการพัฒนาจาวา ที่มีการดีบักในลักษณะกราฟฟิคยูสเซอร์อินเตอร์เฟส มีอยู่ไม่มากนัก เช่น

- เจบิลด์เดอร์(Jbuilder) สร้างโดย บริษัท บอร์แลนด์(Borland)
- วิชวล เอจ(Visual Age) สร้างโดย บริษัท ไอบีเอ็ม
- เจแฟคเตอรี่(JFactory) สร้างโดยบริษัท รูจเวฟ(Rouge Wave)
- วิชวลเจพลัสพลัส(Visual J++) สร้างโดยบริษัท ไมโครซอร์ฟ
- วิชวลคาเฟ่(Visual Café) สร้างโดยบริษัท ไซแมนเทค(Symantec)

ปัญหาที่เกิดขึ้นก็คือดีบักเกอร์มากับโปรแกรมเหล่านี้ เวลาทำการดีบักจะใช้หน่วยความจำจำนวนมากทำให้ช้า อีกทั้งดีบักเกอร์ที่มีอยู่นี้ ยังไม่สามารถนำมาพัฒนาต่อไปเนื่องจากเป็นโปรแกรมทางการค้า

จึงไม่มีการเปิดเผยซอร์สโค้ด(Source code) การพัฒนาดีบักเกอร์ จะเป็นไปในลักษณะที่เป็นเพิ่มความฉลาดให้กับตัวดีบักเกอร์ ซึ่งปัจจุบันการพัฒนาดีบักเกอร์ให้มีความฉลาดนั้นยังมีไม่มากนัก ที่มีอยู่คือภาษาโปรลอก ซึ่งมีลักษณะที่เป็นอินเตอร์พรีทเตอร์เหมือนกับภาษาจาวา โดยในปริญญานิพนธ์นี้เราจะสร้างดีบักเกอร์ขึ้นมาในลักษณะที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส ที่มีลักษณะพื้นฐานในการดีบักซึ่งจะได้กล่าวต่อไป

นอกจากนี้ส่วนของหลักการในการสร้างดีบักเกอร์นั้นจะต้องทราบหลักการในการคิดหาเหตุผลทางตรรก ซึ่งการคิดหาเหตุผลทางตรรกคือ การนำสมมุติฐานหรือความเชื่อที่มีอยู่แล้วมาหาผลลัพธ์ที่จะเกิดขึ้น แต่เมื่อผลลัพธ์นั้นไม่ตรงตามที่ต้องการ ดังนั้นต้องทำการตรวจสอบสมมุติฐานที่มีอยู่ว่ามีส่วนใดส่วนเกี่ยวข้องกับความผิดพลาดที่เกิดขึ้นบ้างแล้วทำการแก้ไขให้ถูกต้อง หลักการในการคิดเช่นนี้เป็นเหมือนกับหลักการการรัน(run)โปรแกรมคอมพิวเตอร์ คือมีโปรแกรมซึ่งถือว่าเป็นสมมุติฐาน เมื่อนำโปรแกรมมารันแล้วผลลัพธ์ที่ได้ไม่ตรงตามที่ต้องการ จะต้องทำการตรวจสอบเพื่อให้ทราบว่าผลลัพธ์โปรแกรมที่มีความผิดพลาดมาจากส่วนใดของโปรแกรม นั่นก็คือการดีบัก

การคิดหาเหตุผลทางตรรกจะประกอบได้ด้วย 2 ส่วน คือสมมุติฐาน(Assumption) และการวินิจฉัย(Inference)

1. สมมุติฐาน คือสิ่งที่มนุษย์สมมติขึ้นเพื่อใช้แทนความเป็นจริงในธรรมชาติได้แก่ ทฤษฎี ความเชื่อ

2. การวินิจฉัย คือผลที่ได้จากการนำสมมุติฐานมาคิดหาเหตุผลทางตรรก

เมื่อเปรียบเทียบกับโปรแกรมคอมพิวเตอร์แล้ว

1.โปรแกรมคอมพิวเตอร์ หมายถึง สมมุติฐาน ที่ผู้เขียนโปรแกรมกำหนดขึ้นเพื่อใช้แก้ปัญหา ซึ่งจะมีข้อมูลต่าง ๆ ซึ่งเป็นประโยคทางตรรก(Logic sentence) ที่ได้มาจากผู้เขียนโปรแกรม เช่น ประโยคในโปรแกรมคอมพิวเตอร์

```
Program R( )
    Procedure P( )
            {
                    Q( )
            }
    Begin
                    P( );
    End
```

จะมองเป็นประโยคทางตรรกได้ดังนี้

("การวินิจฉัยA :- สมมุติฐานB" หมายความว่า จะเกิด การวินิจฉัยA เมื่อมีสมมุติฐานB)

```
R( ) :- P( ),
P( ) :- Q( )
Q( ) :-……,
```

2.ผลลัพธ์ของโปรแกรม = ข้อสรุปของการวินิจฉัย

วิธีการที่จะรู้ว่าข้อสรุปของการวินิจฉัยที่ได้ไม่ตรงตามที่ต้องการเกิดมาจากส่วนใดของโปรแกรม วิธีการทำคือตรวจสอบว่าส่วนใดของโปรแกรมที่มีผลทำให้เกิดผลลัพธ์ที่ผิดพลาดนั้น นั่นคือไปดูว่าโปรแกรมมีเส้นทางการทำงานอย่างไรหรือที่เรียกว่าเธรด(Threads) แล้วพิจารณาแต่ละส่วนว่าส่วนใดมีความเกี่ยวข้องที่จะทำให้ผลลัพธ์ผิดพลาด แล้วทำการแก้ไขให้ถูกต้อง ตัวอย่างเช่น ถ้ามีสมมุติฐานดังนี้

P( ) :- Q( )

Q( )

R( ) :- A( )

A( )

ผลลัพธ์ที่ได้ คือ ¬P( )

การวิเคราะห์ ส่วนที่มีผลทำให้เกิดความผิดพลาด จะได้ว่า

¬(( P( ) :- Q( ) ) และ Q( ) )

ความผิดพลาดอาจจะเป็นไปได้ 3 กรณี

1. P( ) :- Q( )
2. Q( )
3. ( P( ) :- Q( ) ) และ Q( ) ) (ผิดทั้งคู่)

แล้วทำการตรวจสอบในแต่ละกรณีว่าส่วนใดบ้างที่ผิดพลาด

การสร้างดีบักเกอร์ด้วยภาษาจาวาจะต้องเข้าใจการทำงานของจาวาเวอร์ชวลแมชชีนว่ามีการเก็บข้อมูลขณะทำรันโปรแกรมอย่างไร มีลักษณะการทำงานอย่างไร จะทำให้เราสามารถเข้าใจฟังก์ชันการทำงานของจาวาซึ่งจะอธิบายต่อไปในบทที่ 2 สำหรับในส่วนการสร้างดีบักเกอร์ จะต้องนำข้อมูลขณะทำการรันโปรแกรมออกมาเพื่อที่จะทำการตรวจสอบหาส่วนที่ทำให้ผลลัพธ์ผิดพลาด โดยจะใช้ JPDA ในการติดต่อกับจาวาเวอร์ชวลแมชชีนจะอธิบายต่อไปในบทที่ 3

## 1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. ศึกษาการทำงานของดีบักเกอร์ว่ามีหลักการในการทำงานอย่างไรจึงจะดีบักโปรแกรมได้

2. ศึกษาการทำงานของจาวาเวอร์ชวลแมชชีน ว่ามีสถาปัตยกรรมอย่างไรในขณะทำการเอ็กซีคิวต์ไบต์โค้ด

3. สร้างดีบักเกอร์ที่เป็นภาษาจาวา เพื่อดีบักโปรแกรมภาษาจาวาโดยมีลักษณะเป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส และมีฟังก์ชันพื้นฐานที่ใช้ในการดีบัก เช่น สเตป(Step), ตั้งเบรคพ้อยท์ (Set Breakpoint), ดูตัวแปรโลคอล(Local variable) และเรียกดูค่าตัวแปรที่ต้องการได้ เป็นต้น

**1.3 ขอบเขตของปริญญานิพนธ์**

1. ปริญญานิพนธ์นี้ต้องการพัฒนาดีบักเกอร์เพื่อดีบักภาษาจาวาขึ้นมาโดยพัฒนาขึ้นมาจากภาษาจาวา

2. ใช้ JPDA มาช่วยในการติดต่อกับจาวาเวอร์ชวลแมชชีนเพื่อที่จะดึงข้อมูลขณะทำการเอ็กซีคิวโปรแกรม แล้วนำข้อมูลนั้นมาใช้ในการดีบักโปรแกรม

3. สร้างดีบักเกอร์ที่มีลักษณะเป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส เพื่อให้ผู้ใช้สามารถใช้งานได้สะดวกมากขึ้น

4. สร้างโปรแกรมที่มีลักษณะเป็น IDE ทำให้สามารถเปิดไฟล์มาแก้ไข , สร้างไฟล์ใหม่ , compile, execute และ debug ได้

**1.4 วิธีการดำเนินงาน**

1. การศึกษาทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้องกับงานวิจัย ซึ่งก็มีเรื่องอยู่ 3 เรื่อง ด้วยกัน คือ
   - ศึกษารายละเอียดของภาษาจาวาและของจาวาเวอร์ชวลแมชชีนดังที่ได้อธิบายไว้ในบทที่ 2
   - ศึกษาเกี่ยวกับหลักในการสร้างดีบักเกอร์ภาษาจาวาโดยใช้ JPDA ดังที่ได้อธิบายไว้ในบทที่ 3

2. ออกแบบและสร้างดีบักเกอร์ โดยอธิบายไว้ในบทที่ 4 และบทที่ 5 ที่จะกล่าวถึงการออกแบบโปรแกรม รวมทั้งอธิบายรายละเอียดการทำงานของคลาสหลัก ๆ ที่จำเป็นในการนำมาสร้างดีบักเกอร์ และการสร้างที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส

3. ทดสอบโปรแกรมโดยนำโปรแกรมต่าง ๆ มาทดสอบฟังก์ชันการดีบักที่สร้างขึ้นซึ่งจะแสดงไว้ในบทที่ 6

4. สรุปการทำงานของโปรแกรมว่าตรงตามเป้าหมายที่ได้ตั้งไว้หรือไม่และแนวทางการพัฒนาโปรแกรม กล่าวถึงในบทที่ 7

# บทที่ 2

# ภาษาจาวา

## 2.1 บทนำ

ภาษาจาวาเริ่มต้นในฐานะงานวิจัยเพื่อพัฒนาซอฟต์แวร์(software)ระดับสูงสำหรับอุปกรณ์ของเครือข่ายคอมพิวเตอร์และอุปกรณ์ที่ควบคุมโดยไมโครโพรเซสเซอร์ (embedded system) ที่มีขนาดเล็กและเชื่อถือได้ ที่สามารถนำไปรันบนแพลตฟอร์ม(platform)[1]ที่แตกต่างกันได้หรือกระจายส่วนของโปรแกรมไปยังเครื่องต่างๆ อย่างปลอดภัยและประหยัดทรัพยากรของระบบ

ในตอนเริ่มต้น ไวยากรณ์ของภาษาซีถูกเลือกให้เป็นไวยากรณ์ของภาษาใหม่นี้ แต่ต่อมามีปัญหาเกิดขึ้นมากมายจนต้องออกแบบภาษาใหม่ขึ้นมาโดยเรียกว่าภาษาจาวา ภาษาดังกล่าวมีพื้นฐานมาจากภาษาซี (C),ซีพลัส พลัส (C++), ไอเฟล (Eiffel), สมอลล์ทอร์ค (SmallTalk), ออบเจ็กทีฟซี (Objective C) และเซดาร์/เมซา (Cedar/Mesa) ภาษาจาวาได้รับการยอมรับว่าเป็นภาษาที่ดีที่สุดในการพัฒนาซอฟต์แวร์ที่ต้องการความปลอดภัยสูงแบบกระจาย (distributed) ผ่านเน็ตเวิร์ค (network) โดยสามารถใช้พัฒนาได้ตั้งแต่โปรแกรมของอุปกรณ์เน็ตเวิร์คที่ควบคุมโดยไมโครโพรเซสเซอร์ (network-embedded device) โปรแกรมในเวบบราวเซอร์จนถึงแอพพลิเคชั่นทั่วไป

## 2.2 ลักษณะของภาษาจาวา

จาวาเป็นภาษาระดับสูงเชิงวัตถุ (object-oriented high-level language) ซึ่งมีกลไกการแปลภาษาและเอ็กซิคิวต์ในลักษณะผสมผสานระหว่างคอมไพล์เลอร์ (compiler) และอินเตอร์พรีทเตอร์ (interpreter) เพื่อจุดประสงค์ที่จะรับข้อดีและขจัดข้อเสียของทั้งคอมไพล์เลอร์และอินเตอร์พรีทเตอร์

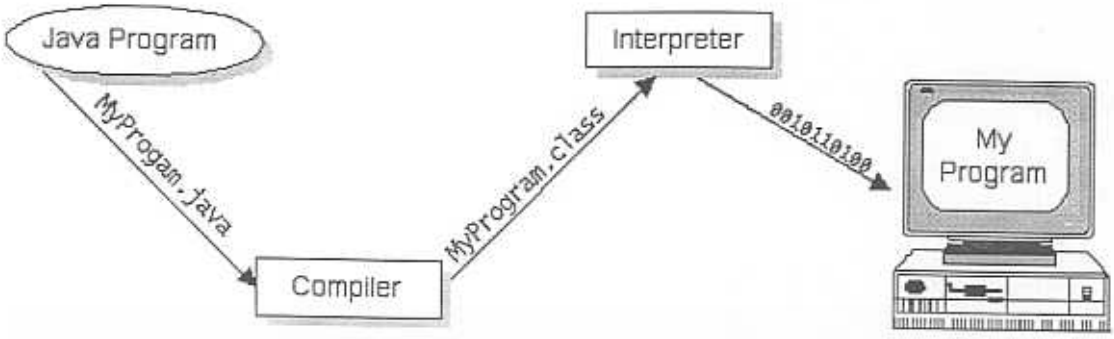
คอมไพล์เลอร์เป็นตัวแปลภาษาที่วิเคราะห์โปรแกรมในภาษาระดับสูงเพื่อแปลงเป็นภาษาเครื่อง (machine code) ซึ่งข้อดีของวิธีนี้คือโปรแกรมภาษาเครื่องจะทำงานเร็วมาก เพราะขั้นตอนในการแปลภาษาถูกแยกออกไปก่อนโปรแกรมทำงาน และคอมไพล์เลอร์ก็สามารถวิเคราะห์ทั้งโปรแกรมได้ก่อน จึงทำให้สามารถสร้างโปรแกรมภาษาเครื่องที่มีประสิทธิภาพในการทำงานและมีขนาดเล็ก แต่ข้อเสียของคอมไพล์เลอร์คือความไม่อิสระต่อแพลตฟอร์ม ภาษาเครื่องที่คอมไพล์เลอร์สร้างขึ้นมาจะไม่สามารถนำไปทำงานบนเครื่องที่ต่างแพลตฟอร์มได้

ส่วนอินเตอร์พรีทเตอร์เป็นตัวแปลภาษาที่ทำงานโดยอ่านโปรแกรมในภาษาระดับสูงทีละบรรทัด แล้วแปลโปรแกรมบรรทัดนั้นเป็นภาษาเครื่องและทำงานทันที ต่อจากนั้นก็อ่านโปรแกรมบรรทัดต่อไปเข้ามาทำเช่นเดิมอีกไปจนกว่าโปรแกรมจะหยุดหรือจบโปรแกรม วิธีนี้จะมีทั้งการแปลภาษาและทำงานโปรแกรมสลับกันไป จึงทำงานช้ากว่าคอมไพล์เลอร์ แต่ก็มีข้อดีคือสามารถสร้างได้ง่ายกว่า เพราะการวิเคราะห์โปรแกรมทีละบรรทัดทำได้ง่ายกว่าแปลทั้งโปรแกรม และที่สำคัญการใช้อินเตอร์พรีทเตอร์จะ

---

[1] ประกอบด้วยฮาร์ดแวร์ (hardware) และระบบปฏิบัติการ (operating system)

ทำให้การทำงานเป็นอิสระต่อแพลตฟอร์มได้ นั่นคือซอร์สโค้ดเดียวกัน เมื่อนำไปรันบนจาวาเวอร์ชวลแมชชีนของแต่ละแพลตฟอร์มก็จะได้ผลลัพธ์เหมือนกัน



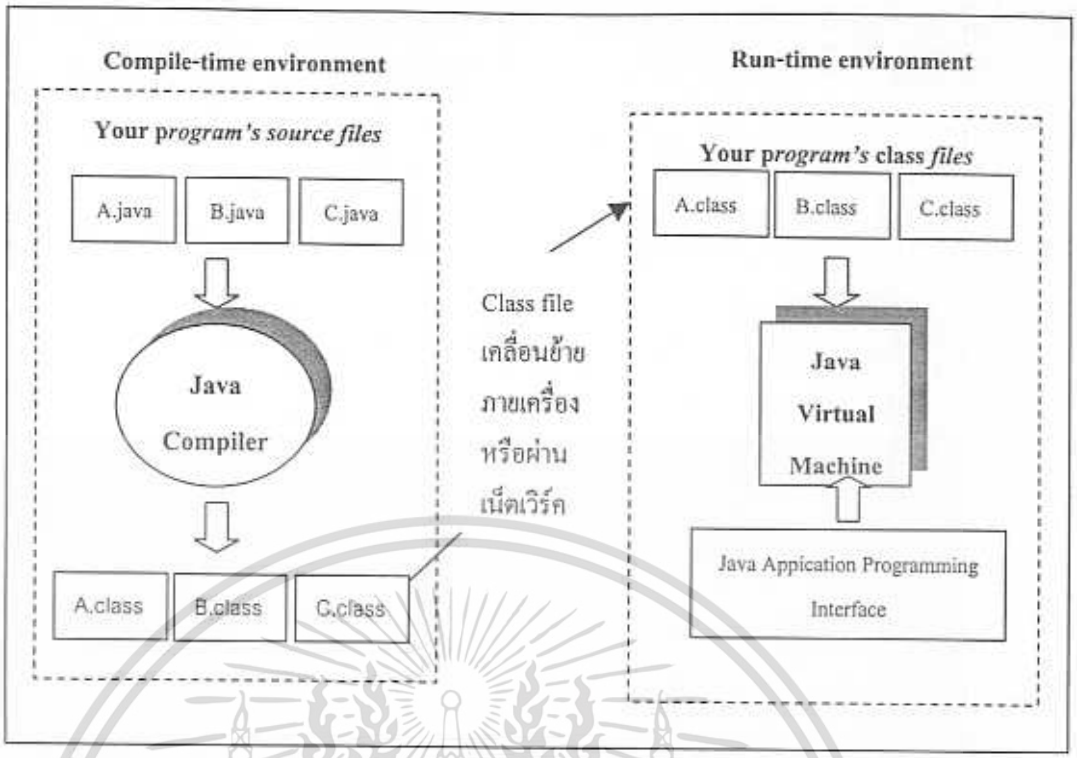

*รูปที่ 2-1 การทำงานของคอมไพล์เลอร์และอินเตอร์พรีทเตอร์ในภาษาจาวา*

จากรูปที่ 2-1 คอมไพล์เลอร์จะแปลโปรแกรมในภาษาจาวาจากไฟล์ MyProgram.java ไปเป็น จาวาไบต์โค้ด (Java Bytecode) ไว้ในไฟล์ MyProgram.class ที่มีเป็นอิสระต่อแพลตฟอร์ม การแปลงจากโปรแกรมในภาษาจาวาไปเป็นจาวาไบต์โค้ดก่อนทำให้อินเตอร์พรีทเตอร์สามารถทำงานได้รวดเร็วกว่าการอินเตอร์พรีทโดยตรงจากโปรแกรมในภาษาจาวา เพราะจาวาไบต์โค้ดมีลักษณะเป็นโค้ดที่มีความซับซ้อนระหว่างภาษาจาวาและภาษาเครื่อง (intermediate code) น้อยกว่าตัวโปรแกรมในภาษาจาวา จาวาไบต์โค้ดดังกล่าวสามารถนำไปรันบนจาวาเวอร์ชวลแมชชีนของแพลตฟอร์มต่างๆ ได้ทันที เนื่องจากจาวาไบต์โค้ดมีลักษณะไม่ขึ้นต่อแพลตฟอร์ม

เราสามารถมองจาวาไบต์โค้ดเป็นคำสั่งภาษาเครื่องที่รันอยู่บนคอมพิวเตอร์จำลองที่เรียกว่าจาวาเวอร์ชวลแมชชีน (Java Virtual Machine) อิมพลีเมนเตชั่นของจาวาเวอร์ชวลแมชชีน (implementation) เช่นจาวาอินเตอร์พรีทเตอร์ทั้งเครื่องมือพัฒนาซอฟต์แวร์และเวบบราวเซอร์ และนอกจากนี้ยังมีอยู่ในเครื่องใช้ไฟฟ้าภายในบ้านที่รันเอ็นไวรอนเมนท์ของเพอซันนอลจาวาแอพพลิเคชั่น (PersonalJava™ Application Environment), สมาร์ทการ์ด (smart card) ที่รันชุดพัฒนาจาวาการ์ด (Java Card™ Development Kit)

วิธีการของจาวาไบต์โค้ดทำให้สามารถโปรแกรมที่เขียนขึ้นในภาษาจาวาสามารถทำงานในลักษณะ "เขียนครั้งเดียว ใช้งานได้ทุกที่" กล่าวคือโปรแกรมภาษาจาวาสามารถถูกคอมไพล์ได้ในทุกแพลตฟอร์มที่มีคอมไพล์เลอร์ของภาษา และจาวาไบต์โค้ดก็สามารถรันได้ในทุกๆ อิมพลีเมนเตชั่นของจาวาเวอร์ชวลแมชชีน

รูปที่ 2-1 เป็นเพียงรูปแบบอย่างง่ายซึ่งทำเพื่อให้เข้าใจในเบื้องต้น โดยทั่วไปแล้วในทางปฏิบัติจะต้องมีการใช้งานหลายคลาส รวมทั้งต้องโหลดจาวาเอพีไอ ซึ่งเป็นไลบรารี่ของฟังก์ชันที่ใช้งานทั่วไปที่ให้มาพร้อมกันกับชุดพัฒนาภาษาจาวา (Java Development Kit) ด้วยดังรูปที่ 2-2




*รูปที่ 2-2 การทำงานของคอมไพล์เลอร์และอินเตอร์พรีทเตอร์ในภาษาจาวาในมุมมองที่ละเอียด*

**2.3 จาวาแพลตฟอร์ม (Java Platform)**

แพลตฟอร์มคือซอฟต์แวร์และ/หรือฮาร์ดแวร์เอ็นไวรอนเมนท์ (environment) ที่รันจาวาไบต์โค้ด จาวาแพลตฟอร์มจะแตกต่างจากแพลตฟอร์มทั่วไปที่เป็นแพลตฟอร์มแบบซอฟต์แวร์ล้วน (software-only platform) ที่รันบนแพลตฟอร์มที่มีฮาร์ดแวร์เป็นฐาน (hardware-based platform) อื่น ๆ ในขณะที่แพลตฟอร์มอื่นๆ โดยทั่วไปจะเป็นส่วนประกอบของฮาร์ดแวร์และระบบปฏิบัติการ รูปที่ 2–3 แสดงจาวาแพลตฟอร์มต่างๆ ของของภาษาจาวา


*รูปที่ 2–3 โปรแกรมภาษาจาวาบนแพลตฟอร์มต่างๆ*

จาวาไบต์โค้ดที่ถูกคอมไพล์แล้วสามารถนำไปรันบนแพลตฟอร์มใดๆ ก็ได้โดยไม่ต้องมีการเปลี่ยนแปลง ส่วนแต่ละจาวาแพลตฟอร์มจะถูกสร้างขึ้นสำหรับแต่ละชนิดของฮาร์ดแวร์และระบบปฏิบัติการเช่นจาวาแพลตฟอร์มสำหรับโอเอสทู ใช้กับพีซีที่รันระบบปฏิบัติการโอเอส/ทู จาวาแพลตฟอร์มมีองค์ประกอบ 2 ส่วนคือ

- จาวาเวอร์ชวลแมชชีน (Java Virtual Machine)
- จาวาแอพพลิเคชั่นโปรแกรมมิ่งอินเตอร์เฟส (Java Application Programming Interface) หรือเรียกโดยย่อว่าจาวาเอพีไอ (Java API)

ในหัวข้อย่อยต่อไปจะกล่าวถึงจาวาเวอร์ชวลแมชชีนและจาวาเอพีไอ แต่จะลงลึกในรายละเอียดในส่วนของจาวาเวอร์ชวลแมชชีน เนื่องจากเป็นพื้นฐานที่สำคัญในการสร้างดีบักเกอร์

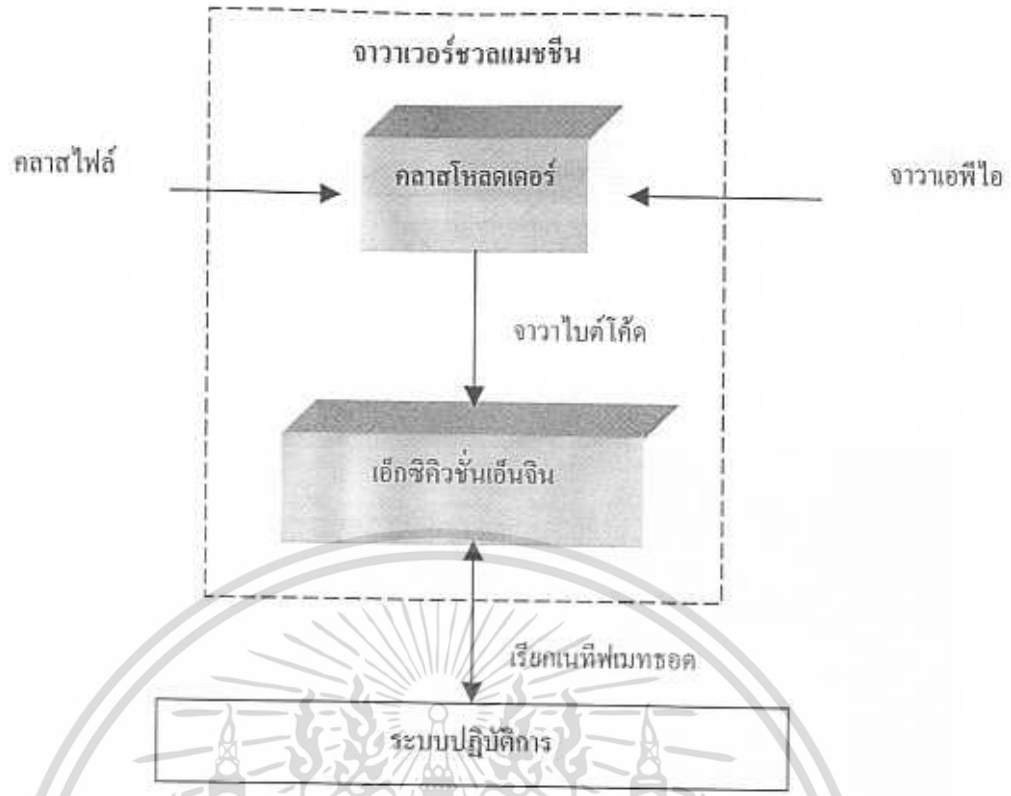
### 2.3.1 จาวาเวอร์ชวลแมชชีน

จาวาเวอร์ชวลแมชชีนเป็นส่วนที่รันบนแพลตฟอร์มซึ่งประกอบด้วยฮาร์ดแวร์และระบบปฏิบัติการเพื่อทำการจำลองตัวเองเป็นคอมพิวเตอร์ที่มีภาษาเครื่องเป็นจาวาไบต์โค้ดซึ่งลักษณะของจาวาเวอร์ชวลแมชชีนจะกำหนดรายละเอียด (specfication) ไว้ว่าจะต้องมีองค์ประกอบใดบ้าง แต่ว่าสามารถสร้างขึ้นได้หลายแบบ อย่างเช่นกำหนดไว้ว่าจาวาเวอร์ชวลแมชชีนจะต้องมีความสามารถในการเอ็กซิคิวต์จาวาไบต์โค้ดแต่ก็มีหลายวิธีที่สามารถทำได้ ซึ่งอาจจะมีเทคนิคที่แตกต่างไป

องค์ประกอบหลักของ จาวาเวอร์ชวลแมชชีน มี 2 ส่วน คือ

- คลาสโหลดเดอร์ (class loader) จะมีหน้าที่ในการโหลดคลาสไฟล์และจาวาเอพีไอ
- เอ็กซิคิวต์เอ็นจิน (execute engine) เป็นส่วนที่เอ็กซิคิวต์จาวาไบต์โค้ด ซึ่งจะสามารถสร้างได้หลายเทคนิค เช่นวิธีที่ง่ายที่สุดคือการอินเตอร์พรีทจาวาไบต์โค้ดในขณะเวลานั้น ๆ เลย แต่ก็มีอีกวิธีที่เร็ว แต่ว่าต้องใช้หน่วยความจำเยอะมาก ก็คือ วิธีจัสอินไทม์ (Just In Time -JIT) วิธีนี้จาวาไบต์โค้ดจะถูกคอมไพล์ไปเป็นภาษาเครื่องซึ่งจะทำให้ขาดคุณสมบัติเป็นอิสระต่อ แพลตฟอร์ม ในการเรียกใช้ครั้งแรก แล้วเก็บไว้ และจะถูกเรียกใช้ได้อีกในครั้งต่อไป

เมื่อจาวาเวอร์ชวลแมชชีนกำลังรันอยู่บนเครื่องคอมพิวเตอร์ (host machine) จะเรียกใช้ระบบปฏิบัติการโดยการเรียกใช้เนทีฟเมทธอด (native method) ซึ่งเขียนด้วยภาษาอื่น ๆ ที่ไม่ใช่ภาษาจาวาเช่น ภาษาซี, ซีพลัสพลัส, แอสเซมบลี (Assembly) หรือภาษาอื่นๆ แล้วคอมไพล์ให้เป็นภาษาเครื่องและเก็บไว้ในไดนามิคคอลลิ่งค์ไลบรารี่ (dynamically linked library : dll) ซึ่งไม่มีคุณสมบัติความเป็นอิสระต่อแพลตฟอร์ม เมื่อเวลาที่โปรแกรมภาษาจาวาต้องการเรียกเนทีฟเมทธอด จาวาเวอร์ชวลแมชชีนจะโหลดไดนามิคส์ลิงค์ไลบรารี่ที่มีเนทีฟเมทธอดนั้น ๆ แล้วก็จะเรียกมาใช้ ดังแสดงในรูปที่ 2-4




*รูปที่ 2-4 จาวาเวอร์ชวลแมชชีนเรียกเนทีฟเมทธอดผ่านระบบปฏิบัติการ*

จะเห็นได้ว่า การใช้เนทีฟเมทธอดทำให้คุณสมบัติการเป็นอิสระต่อแพลตฟอร์มหมดไป ปัญหานี้จะแก้โดยการใช้จาวาเนทีฟอินเตอร์เฟส (Java Native Interface –JNI) ซึ่งจะทำให้โปรแกรมที่เรียกใช้เนทีฟอินเตอร์เฟสสามารถรันภายใต้จาวาแพลตฟอร์มใดๆ ก็ได้

เพื่อที่จะเข้าใจ จาวาเวอร์ชวลแมชชีน จะต้องทราบความแตกต่างของ 3 สิ่งนี้ก่อน คือ

1. ข้อกำหนดเชิงนามธรรม (abstract specification) เป็นการกำหนดรายละเอียดว่าจาวาเวอร์ชวลแมชชีนควรมีองค์ประกอบใดบ้าง แต่ไม่ได้กำหนดว่าจะสร้างขึ้นมาได้อย่างไร

2. คอนกรีตอิมพลีเมนเตชั่น (concrete implementation) เป็นการสร้างจาวาเวอร์ชวลแมชชีนขึ้นมาจริงๆ ตามรายละเอียดที่กำหนดไว้ในข้อกำหนดเชิงนามธรรม ซึ่งอาจจะสร้างโดยใช้ซอฟต์แวร์ หรือทั้งซอฟต์แวร์และฮาร์ดแวร์ประกอบกัน ตัวอย่างเช่นจาวาอินเตอร์พรีทเตอร์

3. รันไทม์อินสแตนส์ (runtime instance) เป็นอินสแตนส์ของจาวาเวอร์ชวลแมชชีนที่ทำหน้าที่จัดการจาวาแอพพลิเคชั่นที่รันอยู่ หนึ่งอินสแตนส์ของจาวาเวอร์ชวลแมชชีนจะทำหน้าที่จัดการต่อหนึ่งจาวาแอพพลิเคชั่น

ความสัมพันธ์ของ 3 สิ่งนี้คือ จาวาแอพพลิเคชั่นจะรันในรันไทม์อินสแตนส์ของคอนกรีตอิมพลีเมนเตชั่นที่สร้างขึ้นตามข้อกำหนดเชิงนามธรรม เมื่อจาวาแอพพลิเคชั่นเริ่มทำงานรันไทม์อินสแตนส์จะเกิดขึ้น และจะทำลายตัวเองลงเมื่อจาวาแอพพลิเคชั่นทำงานเสร็จ หากมีจาวาแอพพลิเคชั่นทำงานพร้อมกัน 3 งาน ก็จะเป็นการใช้ คอนกรีตอิมพลีเมนเตชั่นเดียวกันแต่ว่าจะมีรันไทม์อินสแตนส์เกิดขึ้น 3 อินสแตนส์ โดยแต่ละจาวาแอพพลิเคชั่นจะทำงานในจาวาเวอร์ชวลแมชชีนของตนเอง

2.3.1.1 วงจรชีวิตของจาวาเวอร์ชวลแมชชีน

อินสแตนส์ของจาวาเวอร์ชวลแมชชีนจะเริ่มต้นการทำงานโดยจะเริ่มต้นเรียกเมทธอด main() ซึ่งเป็นจุดเริ่มต้นของในการรันของจาวาแอพพลิเคชั่น

```
class Echo {
    public static void  main(String[] args) {
        int  len = args.length;
        for ( int i= 0 ; i < len ; ++i) {
        System.out.print(args[i] + " ");
        }
        System.out.println();
    }
}
```

โปรแกรม Echo.java ข้างต้น ซึ่งเป็นโปรแกรมที่รับอาร์กิวเมนท์จากคอมมานด์ไลน์ (command-line) แล้วพิมพ์สตริงดังกล่าวออกทางหน้าจอ ผู้ใช้โปรแกรมจะต้องให้ชื่อของคลาสไฟล์ที่มีเมทธอด main( ) ให้กับจาวาเวอร์ชวลแมชชีน ซึ่งโดยทั่วไปแล้ววิธีการนี้จะแตกต่างกันไปในแต่ละแพลตฟอร์ม ตัวอย่างเช่นสำหรับอิมพลิเมนเตชั่นของจาวาเวอร์ชวลแมชชีนของซันเจดีเค (Sun's JDK) ที่เป็นโปรแกรมชื่อว่า java.exe จะรันโดยพิมพ์คำสั่งนี้ที่คอมมานด์ไลน์ของดอส

Java Echo CE, KMITL.

โดย Java คือ คำสั่งที่จะทำงานโดยระบบปฏิบัติการ

Echo คือ ชื่อของคลาสไฟล์ที่จะต้องใช้ในการเริ่มต้นทำงาน ซึ่งต้องมีเมทธอด main( ) ที่ รีเทิร์น void และรับพารามิเตอร์ซึ่งมีประเภทเป็นสตริง

CE, KMITL. คือสตริงดังกล่าวจะผ่านเข้าไปในเมทธอด main( ) ได้แก่ arg[0] คือ CE, arg [1] คือ KMITL.

เมทธอด main( ) จะเป็นจุดเริ่มต้นในการเอ็กซิคิวต์และเป็นเธรด (thread) เริ่มต้นของแอพพลิเคชั่นด้วย ในจาวาเวอร์ชวลแมชชีน จะมีเธรด อยู่ 2 ชนิด คือ เดมอนเธรด (daemon thread) และนอนเดมอนเธรด (nondaemon thread) ซึ่งเดมอนเธรดจะเป็นเธรดที่ถูกใช้โดยจาวาเวอร์ชวลแมชชีนเอง เช่น เธรดที่ทำการ์เบจคอแลคเตอร์ (Garbage Collector) ส่วนนอนเดมอนเธรดจะเป็นเธรดของแอพพลิเคชั่น เช่นเธรดของเมทธอด main()

หากยังมีนอนเดมอนเธรดรันอยู่ จาวาแอพพลิเคชั่นจะดำเนินการเอ็กซิคิวต์ไปเรื่อยๆ แต่หากนอนเดมอนสิ้นสุดการรันหมด เวอร์ชวลแมชชีนอินสแตนส์จะถูกทำลาย หรือหากได้รับการอนุญาตจากซีเคียวริตี้

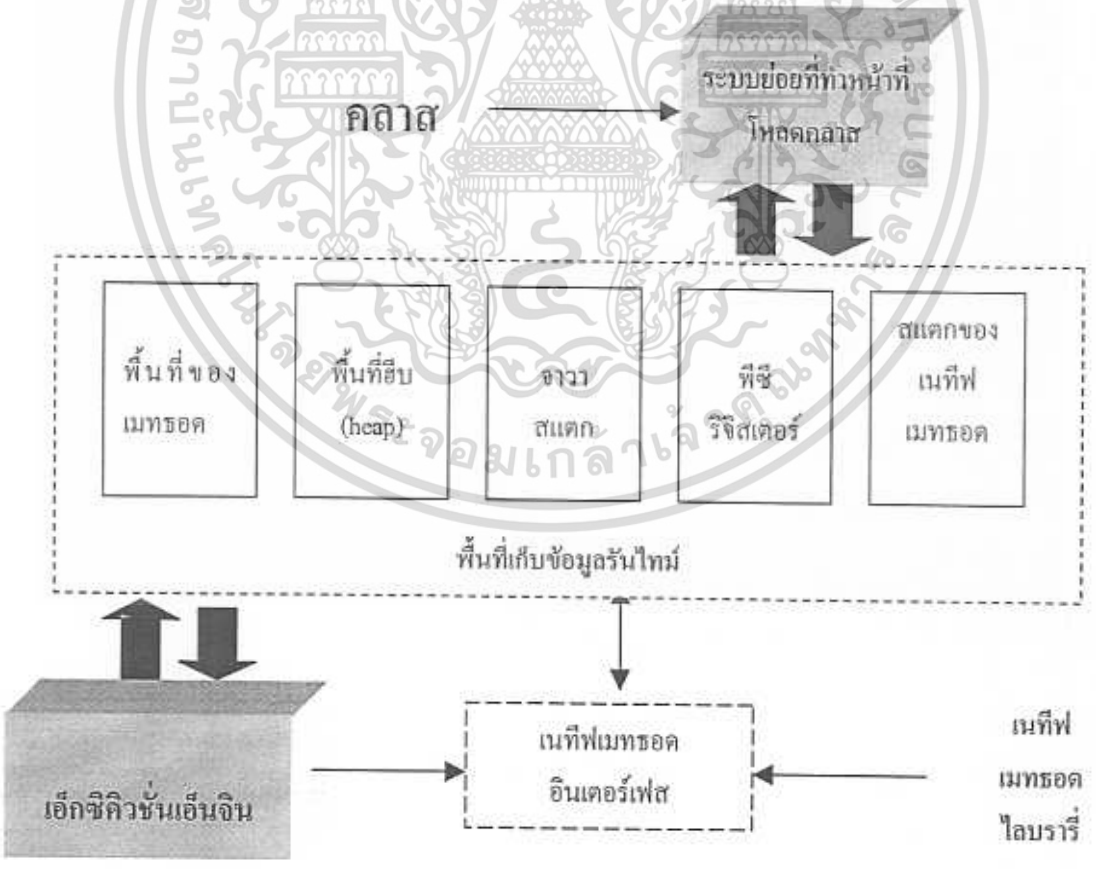
เมเนเจอร์ (Security Manager) แอพพลิเคชั่นก็สามารถจบการทำงานของจาวาเวอร์ชวลแมชชีนโดยการเรียกเมทธอด exit( )จากคลาสรันไทม์ (Runtime) หรือซิสเต็ม (System)

2.3.1.2 สถาปัตยกรรมของจาวาเวอร์ชวลแมชชีน

ในข้อกำหนดของจาวาเวอร์ชวลแมชชีน ลักษณะของจาวาเวอร์ชวลแมชชีนจะอธิบายในจากส่วนของระบบย่อย (subsystem), พื้นที่หน่วยความจำ (memory area), ชนิดของข้อมูล (data type) ไปจนถึงชุดคำสั่ง (instruction) ข้อกำหนดดังกล่าวจะไม่บังคับถึงสถาปัตยกรรมภายในว่าจะสร้างขึ้นมาอย่างไร แต่จะเคร่งครัดเรื่องลักษณะภายนอกและการติดต่อกับองค์ประกอบอื่นๆ ในสภาวะแวดล้อมของการพัฒนาโปรแกรม

รูปที่ 2–5 เป็นบล็อกไดอะแกรม (block diagram) ที่แสดงองค์ประกอบของจาวาเวอร์ชวลแมชชีน แสดงระบบย่อยที่สำคัญและพื้นที่หน่วยความจำ จากที่ได้กล่าวมาแล้วว่า จาวาเวอร์ชวลแมชชีน จะประกอบด้วยส่วนของตัวโหลดคลาสที่ทำหน้าที่ในการโหลดคลาสต่าง ๆ และเอ็กซีคิวต์เอ็นจิน ซึ่งเป็นกลไกที่รับผิดชอบในการเอ็กซิคิวต์คำสั่งที่อยู่ในเมทธอดของคลาสที่โหลดเข้ามา

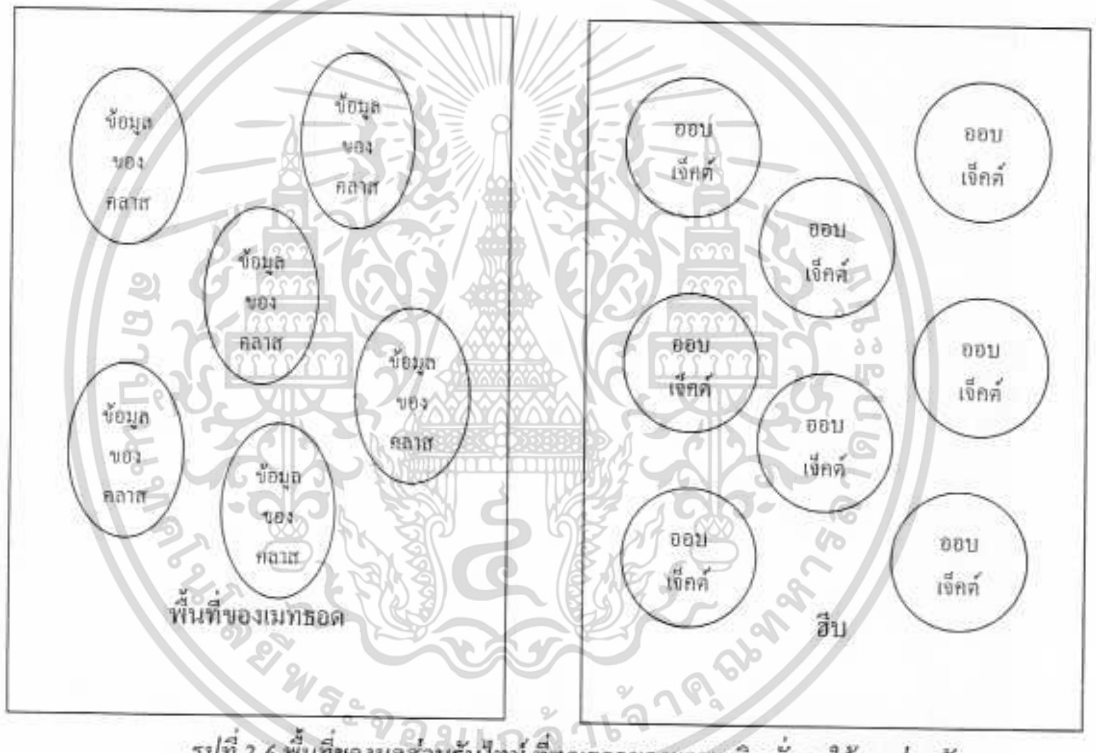
เมื่อจาวาเวอร์ชวลแมชชีนรันโปรแกรม จะต้องการหน่วยความจำเพื่อที่จะเก็บสิ่งต่าง ๆ เช่นจาวาไบต์โค้ดและข้อมูลอื่นๆ ที่ได้มาจากคลาสไฟล์ที่ถูกโหลดมาไว้ในจาวาเวอร์ชวลแมชชีน, พารามิเตอร์ที่ส่งเข้ามาในเมทธอด, ค่าที่ต้องส่งกลับ (return value), ตัวแปรโลคอล (local variable) และผลลัพธ์ต่างๆ ที่เกิดขึ้นระหว่างการรันโปรแกรม




*รูปที่ 2-5 บล็อกไดอะแกรมแสดงองค์ประกอบที่สำคัญของจาวาเวอร์ชวลแมชชีน*

ถึงแม้ชนิดของพื้นที่ข้อมูลจะมีเหมือนกันสำหรับแต่ละจาวาเวอร์ชวลแมชชีน แต่จะไม่มีการลงรายละเอียดถึงวิธีการสร้าง วิธีการสร้างและโครงสร้างภายในจะเป็นหน้าที่ของผู้ออกแบบ เนื่องจากแต่ละแพลตฟอร์มมีลักษณะเฉพาะที่แตกต่างกัน เช่น มีขนาดของหน่วยความจำที่แตกต่างกัน บางแพลตฟอร์มก็มีหน่วยความจำพิเศษเช่นหน่วยความจำเสมือน (virtual memory) การกำหนดองค์ประกอบของจาวาเวอร์ชวลแมชชีนแบบไม่ลงรายละเอียดถึงวิธีการสร้าง จะทำให้การสร้างจาวาเวอร์ชวลแมชชีนง่ายขึ้น

มีพื้นที่ข้อมูลบางชนิดที่ทุกๆ เธรดของแอพพลิเคชั่นหนึ่งๆ จะใช้ร่วมกันในขณะที่บางชนิดมีสำหรับแต่ละเธรด แต่ละอินสแตนส์ของจาวาเวอร์ชวลแมชชีนจะมีพื้นที่ของเมทธอดและฮีบ (heap) เพียงหนึ่งเดียวและจะใช้ร่วมกันโดยทุกๆ เธรดของที่รันอยู่ในจาวาเวอร์ชวลแมชชีน รูปที่ 2-6 จะแสดงพื้นที่ชนิดดังกล่าว เมื่อจาวาเวอร์ชวลแมชชีนโหลดคลาสไฟล์ ข้อมูลเกี่ยวกับชนิดของข้อมูลไว้ในพื้นที่ของเมทธอด และเมื่อโปรแกรมทำงาน จาวาเวอร์ชวลแมชชีนจะเก็บออบเจ็คต์ของโปรแกรมไว้ที่ส่วนของฮีบ

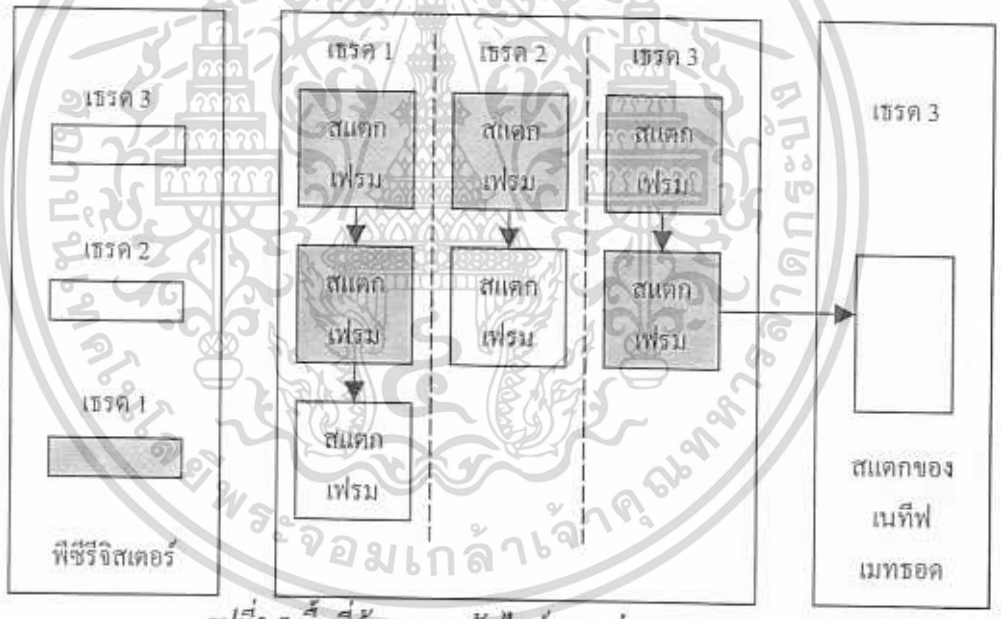



*รูปที่ 2-6 พื้นที่ของมูลส่วนรันไทม์ ที่ทุกเธรดของแอพพลิเคชั่นจะใช้งานร่วมกัน*

สำหรับพื้นที่ข้อมูลเมื่อรันไทม์ชนิดที่เป็นของแต่ละเธรดได้แก่จาวาสแตก,พีซีรีจิสเตอร์ และเนทีฟเมทธอดสแตก ทุกๆ ครั้งที่มีการสร้างเธรดขึ้นมาใหม่ แต่ละเธรดจะมีพีซีรีจิสเตอร์ที่เก็บค่าโปรแกรมเคาน์เตอร์ (program counter) และจาวาสแตกของตัวเอง ค่าของพีซีรีจิสเตอร์จะชี้ที่คำสั่งต่อไปและจาวาสแตกของแต่ละเธรดก็จะเก็บสถานะเกี่ยวกับการเรียกใช้จาวาเมทธอดที่เกี่ยวข้องนั้น ๆ เช่นค่าตัวแปรโลคอล, พารามิเตอร์ที่ถูกเรียกใช้, ค่าที่ได้จากการทำงานของเมทธอด(ถ้ามี) และค่าที่ได้จากการคำนวณภายในเมทธอด (intermediate calculation) และหากการเรียกใช้เนทีฟเมทธอดก็จะมีสแตกของเนทีฟเมทธอดไว้เก็บสถานะของเธรดแยกออกมาต่างหาก

จาวาสแตกจะประกอบด้วยสแตกเฟรม (stack frame) ซึ่งจะเป็นที่เก็บสถานะของการเรียกจาวาเมทธอดแต่ละครั้ง เมื่อเธรดเรียกจาวาเมทธอด จาวาเวอร์ชวลแมชชีนจะพุช (push) เฟรมใหม่ลงไปในสแตกของเธรดนั้นๆ เมื่อจาวาเมทธอดจบการทำงานแล้ว จาวาเวอร์ชวลแมชชีนจะป็อป (pop) เฟรมของเมทธอดนั้นๆ ทิ้งออกไป จะเห็นได้ว่าจาวาเวอร์ชวลแมชชีนจะไม่ใช้รีจิสเตอร์ในการเก็บข้อมูล แต่จะใช้สแตกแทนซึ่งจะทำให้คำสั่งของจาวาเวอร์ชวลแมชชีนมีความกะทัดรัดมากขึ้น และยังทำให้การสร้างจาวาเวอร์ชวลแมชชีนบนแพลตฟอร์มที่มีจำนวนรีจิสเตอร์น้อยได้ง่ายและสะดวกขึ้น

รูปที่ 2-7 แสดงส่วนของอินสแตนส์ของจาวาเวอร์ชวลแมชชีนที่มีเธรด 3 เส้นกำลังเอ็กซิคิวต์อยู่ เธรด1 และเธรด 2 เป็นเธรดที่กำลังเอ็กซิคิวต์ในจาวาเมทธอด ส่วนเธรด3 กำลังเอ็กซิคิวต์อยู่ในเนทีฟเมทธอด พื้นที่ข้อมูลสำหรับแต่ละเธรดจะมีจาวาสแตกและพีซีรีจิสเตอร์ซึ่งถูกสร้างขึ้นมาสำหรับแต่ละเธรด เธรดใดๆ จะไม่สามารถเข้าถึงจาวาสแตกและพีซีรีจิสเตอร์ของเธรดอื่นได้ สแตกเป็นสแตกที่ขยายตัวลงด้านล่าง สแตกเฟรมที่กำลังเอ็กซิคิวต์อยู่จะเป็นสีขาว สำหรับเธรดที่รันอยู่ในจาวาเนทีฟเมทธอด พีซีรีจิสเตอร์จะเก็บคำสั่งที่จะถูกเอ็กซิคิวต์ต่อไปไว้แทนด้วยสีขาว ส่วนเธรดที่รันอยู่ในเนทีฟเมทธอดจะแทนด้วยสีเทา ค่าในพีซีรีจิสเตอร์จะไม่ได้กำหนดไว้เนื่องจากไม่ได้รันอยู่ในจาวาเมทธอด




*รูปที่2-7 พื้นที่ข้อมูลขณะรันไทม์ของแต่ละเธรด*

2.3.1.3 การเข้าถึงข้อมูลการดีบักในจาวาเวอร์ชวลแมชชีน

เพื่อที่จะสามารถดึงข้อมูลการดีบักได้ ในการคอมไพล์ จะต้องใส่ออปชั่น "-g" เพื่อให้สร้างแอตธริบิวต์ (attribute) LineNumberTable และ LocalVariableTable ซึ่ง

### 2.3.2 จาวาแอพพลิเคชั่นโปรแกรมมิ่งอินเตอร์เฟส

จะเป็นชุดของคอมโพเนนต์ที่พร้อมใช้งาน (component) ที่ใช้ในการทำงานที่ใช้กันบ่อยๆ หรืองานเฉพาะบางอย่าง เช่นกราฟิคยูสเซอร์อินเตอร์เฟส (Graphic User Interface –GUI) เอพีไอเหล่านี้จะถูก

รวมกลุ่มเข้าไปเป็นแพคเก็จ (package) ของไลบรารี่ที่เกี่ยวข้องกัน ทำให้ผู้เขียนโปรแกรมสามารถเรียกใช้ฟังก์ชันต่างๆ ได้ทันที วิธีการใช้งานจะอยู่ในจาวาดอกคิวเมนเตชั่น (Java Documentation) ซึ่งสามารถดาวน์โหลดได้จากเวบไซต์ www.java.sun.com

# บทที่ 3
# สถาปัตยกรรมของดีบักเกอร์บนจาวาแพลตฟอร์ม

## 3.1 บทนำ

บริษัทซันไมโครซิสเต็มส์ (Sun Microsystems) ได้พัฒนาสถาปัตยกรรมของดีบักเกอร์บนจาวาแพลตฟอร์มหรือเรียกย่อๆ ว่าเจพีดีเอ (Java™ Platform Debugger Architecture Description -JPDA) เพื่อแก้ปัญหาการชนกันของทรัพยากรระบบ (resource contention) อันได้แก่การเข้าสู่สถานะตาย (deadlock) และการชนกัน (crashing) ของดีบักเกอร์แบบเก่า ก่อนที่จะพูดถึงสาเหตุของปัญหาของดีบักเกอร์แบบเก่าเป็นสิ่งสำคัญที่จะต้องเข้าใจรูปแบบในการสร้างเสียก่อน รูปแบบในการสร้างดีบักเกอร์มีดังนี้

- **in-process debugger**

เป็นรูปแบบการสร้างดีบักเกอร์ที่ดีบักเกอร์นั้นรันอยู่ภายใต้จาวาเวอร์ชวลแมชชีนเดียวกันกับดีบักกี้[1] (debuggee) ดังนั้นการควบคุมการทำงานจะทำโดยจาวาเวอร์ชวลแมชชีนเดียวกัน ซึ่งจาวาเวอร์ชวลแมชชีนจะไม่สามารถทราบได้เลยว่าภายในตัวมันมี 2 โปรแกรมทำงานอยู่ อันจะทำให้เกิดการชนกันของทรัพยากรระบบเพราะดีบักเกอร์และดีบักกี้ต่างก็ไม่รู้ลำดับการทำงานซึ่งกันและกัน ดังนั้นก็อาจเกิดกรณีที่ดีบักเกอร์รอทรัพยากรระบบที่ดีบักกี้ใช้อยู่และดีบักเกอร์ก็รอทรัพยากรที่ดีบักกี้ใช้อยู่ซึ่งทำให้เข้าสู่สถานะตายได้ วิธีการสร้างแบบนี้เป็นวิธีการสร้างของดีบักเกอร์แบบเก่า

- **out-process debugger**

รูปแบบการสร้างดีบักเกอร์แบบนี้การควบคุมการทำงานของดีบักเกอร์จะอยู่ภายใต้คนละจาวาเวอร์ชวลแมชชีนกับดีบักกี้ แต่อาจมีบางส่วนของโค้ดของดีบักกี้รันอยู่ภายใต้จาวาเวอร์ชวลแมชชีนเดียวกันกับดีบักกี้เพื่อกระทำการบางอย่าง ซึ่ง out-process debugger แบ่งออกเป็น 2 ประเภทดังนี้

— pure out-process debugger

ดีบักเกอร์จะใช้กลไกเช่นหน่วยความจำร่วม (shared memory) ในการเข้าถึงสถานะการทำงานของดีบักกี้ วิธีนี้จะไม่มีส่วนใดเลยของโค้ดของดีบักเกอร์รันอยู่ในจาวาเวอร์ชวลแมชชีนของดีบักกี้

การสร้างดีบักเกอร์แบบนี้จะมีข้อเสียที่วิธีการสร้างของดีบักเกอร์จะต้องขึ้นอยู่กับวิธีการสร้างของจาวาเวอร์ชวลแมชชีน, เวอร์ชั่น (version) และแพลตฟอร์ม (platform) ของจาวาเวอร์ชวลแมชชีนของทั้งดีบักเกอร์และดีบักกี้ ดังนั้นเมื่อมีการเปลี่ยนองค์ประกอบอย่างใดอย่างหนึ่งแล้ว จะต้องมีการเปลี่ยนแปลงวิธีการสร้างของดีบักเกอร์ให้สอดคล้องกับความเปลี่ยนแปลงนี้ และนอกจากนี้ยังสามารถใช้ได้เพียงเป็นการดีบักเพื่อตรวจสอบความถูกต้องระหว่างพัฒนาได้เท่านั้น อาจไม่สามารถใช้ดีบักระหว่าง

[1] โปรแกรมที่ถูก debug

ทำงานได้เนื่องจากเวลานำไปใช้งาน อาจเกิดกรณีที่จาวาเวอร์ชวลแมชชีนหรือจาวาแพลตฟอร์มไม่สอดคล้องกับดีบักเกอร์ก็ได้

— out-process debugger with in-process helper

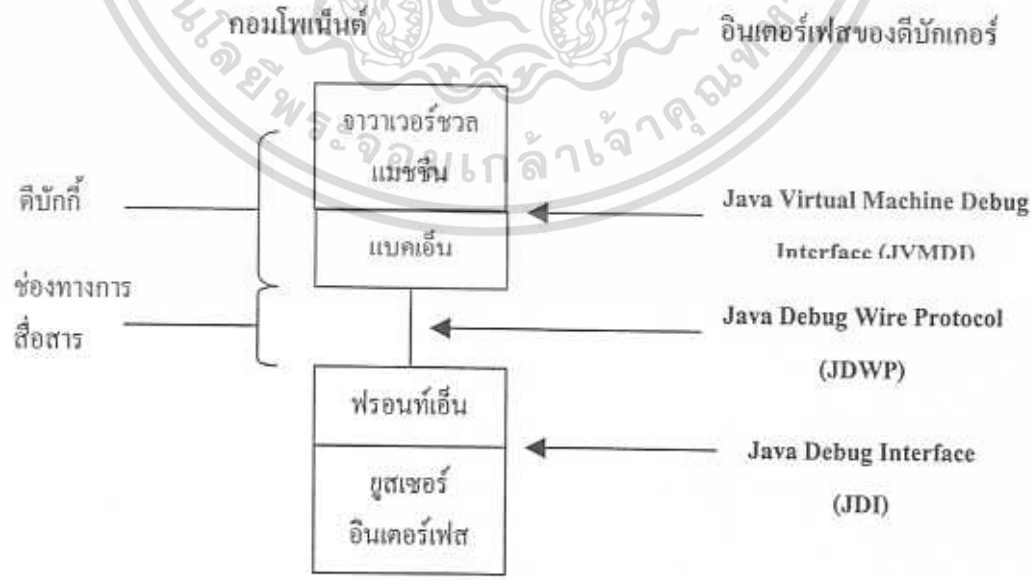
ส่วนหน้า (front-end) ของดีบักเกอร์จะรันอยู่คนละจาวาเวอร์ชวลแมชชีนกับดีบักกี้ แต่อาจจะมีโค้ดของดีบักเกอร์บางส่วนรันอยู่ในจาวาเวอร์ชวลแมชชีนเดียวกันกับดีบักกี้ โดยจะต้องมีกลไกการติดต่อสื่อสารระหว่าง out-process front-end วิธีนี้จะทำให้ปัญหาเรื่องความไม่สอดคล้องกันระหว่างจาวาเวอร์ชวลแมชชีนของดีบักเกอร์และดีบักกี้ดังปรากฏในดีบักเกอร์แบบ pure out-process และดีบักกี้ซึ่งวิธีนี้เป็นวิธีที่จะศึกษาต่อไป

วิธีนี้มีข้อดีที่สามารถสร้างดีบักเกอร์อย่างเป็นอิสระต่อจาวาเวอร์ชวลแมชชีนและแพลตฟอร์มได้ และหากมีการออกแบบที่ดีแล้ว จะสามารถกำจัดปัญหาการชนกันของทรัพยากรระบบได้ 100% สรุปก็คือปัญหาของดีบักเกอร์แบบเก่าเกิดจากการใช้รูปแบบ in-process debugger ดังนั้นเพื่อแก้ปัญหาการชนกันของทรัพยากรดังกล่าวบริษัทซันไมโครซิสเต็มส์จึงเลือกรูปแบบ out-process debugger with in-process helper เป็นวิธีการในการสร้างเจพีดีเอ

### 3.2 โครงสร้างสถาปัตยกรรมของดีบักเกอร์ในจาวาแพลตฟอร์ม

บริษัทซันไมโครซิสเต็มส์จึงได้กำหนดเจพีดีเอเพิ่มขึ้นมาในชุดพัฒนาซอฟต์แวร์ภาษาจาวารุ่นที่ 2 (Java 2 Software Development Kit) มีจุดประสงค์เพื่อใช้เป็นโครงสร้างพื้นฐาน (infrastructure) ในการสร้างดีบักเกอร์ระดับผู้เชี่ยวชาญ (professional)

ปรัชญาของการสร้างเจพีดีเอนี้คือ มันจะต้องใช้กับจาวาเวอร์ชวลแมชชีนใดก็ได้ ดังนั้นเพื่อที่จะให้บรรลุจุดประสงค์นี้ สถาปัตยกรรมนี้จะต้องเป็นแบบเลเยอร์ (layer) โดยแบ่งเป็นเลเยอร์ย่อยๆ 3 เลเยอร์ ซึ่งแต่ละเลเยอร์เป็นอิสระต่อกัน โครงสร้างของเจพีดีเอแสดงดังรูปที่ 3-1




*รูปที่ 3-1 โครงสร้างของ เจพีดีเอ*

จากรูปที่ 3-1 จะเห็นได้ว่าเจพีดีเอมีองค์ประกอบหลักอยู่ 2 ส่วนคือ

**3.2.1. คอมโพเน็นต์ (component) ประกอบด้วย**

- ดีบักกี้

เป็นโพรเซสที่เราต้องการดีบักซึ่งจะประกอบด้วยแอพพลิเคชั่นที่กำลังถูกดีบัก (ไม่ได้แสดงในรูป), จาวาเวอร์ชวลแมชชีนที่ใช้ในการรันแอพพลิเคชั่นและ แบคเอ็น (back-end ) ของดีบักเกอร์

- จาวาเวอร์ชวลแมชชีน

ก็คือจาวาเวอร์ชวลแมชชีนที่กำลังรันโปรแกรมที่ต้องการดีบักหรือดีบักกี้ ซึ่งในสถาปัตยกรรมของเจพีดีเอ นั้นออกแบบให้สามารถใช้จาวาเวอร์ชวลแมชชีนเวอร์ชั่นใดหรือแพลตฟอร์มใดก็ได้ในการรันดีบักกี้ และสามารถเปลี่ยนได้โดยไม่กระทบกับส่วนอื่น ตามวิธีของสถาปัตยกรรมแบบเลเยอร์

- แบคเอ็น (back-end)

ส่วนแบคเอ็นของดีบักเกอร์มีหน้าที่รับผิดชอบการติดต่อสื่อสารระหว่างฟรอนท์เอ็น ( front-end ) ของดีบักเกอร์และเวอร์ชวลแมชชีนของดีบักกี้ ทั้งในส่วนของการร้องขอจากฟรอนท์เอ็นของดีบักเกอร์และส่วนของการตอบสนองต่อฟรอนท์เอ็นของดีบักเกอร์

โดยแบคเอ็นจะติดต่อกับฟรอนท์เอ็นผ่านทางเส้นทางการติดต่อสื่อสาร (แสดงดังรูป) โดยใช้ Java Debug Wire Protocol (JDWP) และแบคเอ็นจะติดต่อกับเวอร์ชวลแมชชีนของดีบักกี้โดยใช้ Java Virtual Machine Debug Interface (JVMDI)

จากข้อเสียที่เราได้กล่าวมาแล้วว่าดีบักเกอร์ที่เขียนด้วยภาษาจาวาและรันในจาวาเวอร์ชวลแมชชีนเดียวกันกับดีบักกี้ จะทำให้เกิดการชนกันของทรัพยากรระบบดังนั้นเราจะใช้ เนทีฟโค้ด (native code) ในการสร้างแบคเอ็นเพื่อหลีกเลี่ยงปัญหานี้ (เนื่องจากเนทีฟโค้ดไม่ต้องรันในจาวาเวอร์ชวลแมชชีน) และดังนั้นเจวีเอ็มดีไอจะต้องเป็นเนทีฟอินเตอร์เฟสแบบแท้ด้วย (pure native interface) ด้วย

- เส้นทางการสื่อสาร (communication channel)

เป็นเส้นทางการติดต่อระหว่างฟรอนท์เอ็นและแบคเอ็นที่สามารถเลือกกลไกการติดได้โดยอิสระตามความเหมาะสม ไม่ว่าจะเป็นซอกเก็ตแบบทีซีพี/ไอพี (TCP/IP socket), หรือการใช้หน่วยความจำร่วม (shared memory) ส่วนรูปแบบ (format) และไวยากรณ์ (semantic) ของบิตสตรีมแบบอนุกรม (serialized bit-stream) ที่วิ่งในเส้นทางการติดต่อสื่อสารนี้จะระบุโดย Java Debug Wire Protocol (JDWP)

- ฟรอนท์เอ็น (front-end)

สร้างโดย Java Debug Interface (JDI) มีหน้าที่ติดต่อระหว่างฟรอนท์เอ็นของดีบักเกอร์และ ยูสเซอร์อินเตอร์เฟส (User InterfaceX ผ่านทาง Java Debug Wire Protocol (JDWP) โดยผลลัพธ์ของ front-end จะอยู่ในระดับซอร์สโค้ด (source code) ซึ่งผู้เขียนโปรแกรมสามารถเข้าใจได้

- ยูสเซอร์อินเตอร์เฟส

เป็นไคลเอนต์ (client) ที่เรียกใช้บริการจากเจดีไอซึ่งในสถาปัตยกรรมเจพีดีเอไม่ได้ระบุเฉพาะเจาะจง ดังนั้นในการสร้างเราสามารถใช้เครื่องมือ (tools) อะไรก็ได้ในการสร้างและเขียนในภาษาอะไรก็ได้ วิธีนี้จะสร้างความสะดวกแก้ผู้พัฒนาได้อย่างมาก

ส่วนนี้เป็นส่วนที่ผู้พัฒนาดีบักเกอร์แอพพลิเคชั่นจะต้องสร้างขึ้นมาเอง โดยเรียกใช้ส่วนของเจดีไอซึ่งเจดีไอจะไปเรียกใช้งานเจดีดับเบิลยูพีและเจวีเอ็มดีไอ อนึ่งทั้งอิมพลีเมนเตชั่นของ เจดีไอ, เจดีดับเบิ้ลยูพี, และเจวีเอ็มดีไอได้มีพร้อมมากับชุดพัฒนาเจพีดีเอแล้ว แต่ผู้พัฒนาสามารถสร้างขึ้นมาเองเพื่อให้สอดคล้องกับความต้องการเฉพาะได้

### 3.2.2 ดีบักเกอร์อินเตอร์เฟส (Debugger Interfaces) ประกอบด้วย

- Java Virtual Machine Debugger Interface (JVMDI)

เป็นจาวาเนทีฟอินเตอร์เฟสที่ถูกเรียกใช้โดยแบคเอ็นของดีบักเกอร์ เป็นการกำหนดบริการและข้อมูลที่จาวาเวอร์ชวลแมชชีนต้องให้ในการดีบักรวมทั้งการเรียกใช้ข้อมูล (ตัวอย่างเช่น สแตกเฟรมที่กำลังทำงานอยู่ (current stack frame)), การกระทำต่างๆ (เช่น การเซตเบรคพอยต์ (breakpoint)) และการเตือน (เช่นเตือนว่ารันมาถึงเบรคพอยต์ที่ตั้งไว้แล้ว) โดยเป็นเนทีฟอินเตอร์เฟสที่เรียกใช้ jvmdi.h ซึ่งมีมากับชุดพัฒนาโปรแกรมภาษาจาวารุ่นที่ 2 ขึ้นไป แต่เราก็สามารถเปลี่ยนแปลงไฟล์ดังกล่าวให้สอดคล้องกับความต้องการได้

การกำหนดอินเตอร์เฟสของเวอร์ชวลแมชชีนดังกล่าวจะทำให้จาวาเวอร์ชวลแมชชีนใดๆ[2] ก็ตามสามารถใช้สถาปัตยกรรมนี้ได้

- Java Debug Wire Protocol (JDWP)

เป็นการกำหนดรูปแบบของข้อมูลและการร้องขอที่เรียกผ่านระหว่างโพรเซสของดีบักกี้ และฟรอนท์เอ็นของดีบักเกอร์ แต่ว่าจะไม่กล่าวถึงกลไกการติดต่อ (transport mechanism) เพื่อให้ผู้สร้าง (implement) สามารถเลือกได้ตามเหมาะสม (ไม่ว่าจะเป็นsocket, serial line, shared memory หรืออื่นๆที่จะเพิ่มเติมมาในภายหลัง)

---

[2] นั่นคือไม่ว่าจะเป็น เวอร์ชั่นใดหรือแพลตฟอร์มใดก็ตาม

ในการสร้างจริง จะกำหนดไว้ในไดนามิคลอลี่ลิงค์ไลบรารี่ (dynamically linked library) ที่มีชื่อว่า jdwp.dll โดยจะเก็บไว้ในไดเรคทอรี่ย่อย jre\bin ซึ่งเป็นที่เก็บไดนามิคส์ลิงค์ไลบรารี่ที่จำเป็นต่อการใช้งานของชุดพัฒนาโปรแกรมภาษาจาวารุ่นที่ 2

วิธีการของโพรโตคอล (protocol) นี้ทำให้ดีบักกี้และฟรอนท์เอ็นของดีบักเกอร์ สามารถรันภายจาวาเวอร์ชวลแมชชีนต่างเวอร์ชั่น และ/หรือแพลตฟอร์มที่แตกต่างกันได้ และยังทำให้ฟรอนท์เอ็นสามารถพัฒนาในภาษาอื่นนอกเหนือจากภาษาได้

- Java Debug Interface (JDI)

เป็นจาวาอินเตอร์เฟสแท้ (Java interface) ที่ใช้โดยฟรอนท์เอ็นโดยจะเป็นอินเตอร์เฟสที่ใช้รวมคุณสมบัติการดีบักเข้ากับสภาพแวดล้อมการพัฒนา (development environment) โดยการเรียกใช้งานเลเยอร์ (layer) ล่างได้แก่เจดีดับเบิลยูพีและเจดีไอนั่นเอง ในการสร้างจะเป็นเอพีไอที่รวมอยู่ในแพคเก็จ com.sun.jdi ที่มาพร้อมกับเจพีดีเอ ซึ่งสามารถเรียกใช้โดยแอพพลิเคชั่นของดีบักเกอร์ เช่นเดียวกับอินเตอร์เฟสอื่นเจดีไอสามารถเปลี่ยนแปลงให้สอดคล้องกับความต้องการของผู้พัฒนาได้

## 3.3 รายละเอียดการเชื่อมต่อและเรียกใช้ (Connection and Invocation Details)

### 3.3.1.กลไกการติดต่อระหว่างดีบักเกอร์แอพพลิเคชั่นและดีบักกี้ (Transport)

ในส่วนนี้จะกล่าวถึงการเชื่อมต่อ (connection) ระหว่างดีบักกี้และดีบักเกอร์แอพพลิเคชั่นและการเรียกจาวาเวอร์ชวลแมชชีนของดีบักกี้ให้เรียกตัวเองขึ้นมา (invocation) เพื่อที่ดีบักเกอร์แอพพลิเคชั่นจะสามารถเข้าไปตรวจสอบได้

ก่อนที่จะกล่าวถึงการเชื่อมต่อจะต้องเข้าใจในเรื่องของรูปแบบการติดต่อสื่อสารระหว่างโพรเซส (transport) เสียก่อน เจพีดีเอสนับสนุนรูปแบบการติดต่อสื่อสาร 2 แบบ[3] นั่นคือ

- รูปแบบการติดต่อสื่อสารแบบซ็อกเก็ต (Socket Transport)

รูปแบบของการติดต่อสื่อสารระหว่างโพรเซสแบบนี้มีใช้ทั้งในแพลตฟอร์มของซันโซลาริส (Solaris) และวินโดวส์แบบ 32 บิต (Win32) โดยจะใช้ ซ็อกเก็ตแบบทีซีพี/ไอพี

ในการติดต่อระหว่างแอพพลิเคชั่นของดีบักเกอร์และดีบักกี้ในการติดต่อแบบนี้ทั้งดีบักเกอร์และดีบักกี้สามารถอยู่ทั้งบนเครื่องเดียวกันหรือต่างเครื่องกันได้ โดยจะแบ่งแยกแต่ละการติดต่อด้วยสตริงที่ไม่ซ้ำกันเลย (unique string) ที่เรียกว่า dt_socket ซึ่งค่าใน dt_socket นี้จะใช้ในการเลือกเส้นทางการติดต่อสื่อสาร เมื่อมีการรันจาวาเวอร์ชวลแมชชีนของดีบักกี้ [4] หรือเพื่อเลือกคอนเนคเตอร์(connector)ของเจดีไอ

---

[3] ใน JPDA เวอร์ชั่นปัจจุบันยังสนับสนุนแค่ 2 แบบแต่ในอนาคตอาจมีรูปแบบ transport ใหม่เพิ่มขึ้นมา และอาจมีอินเตอร์เฟสที่ใช้ในเพิ่มเติมรูปแบบของ transport

[4] หรือเรียกว่า target Virtual Machine

เมื่อไคลเอนต์จะต่อ (attach) เข้ากับเซิร์ฟเวอร์ (server) รูปแบบของที่อยู่ของรูปแบบการติดต่อสื่อสารแบบซ็อกเก็ต (socket transport address)[5] จะเป็น "<name>:<port>" โดยที่ <name> คือชื่อของโฮสต์ และ <port> คือ หมายเลขของพอร์ตของซ็อกเก็ต (socket port) ที่ไคลเอนท์ต้องการ attach หรือ listen แต่ในกรณีที่เซิร์ฟเวอร์รอไคลเอนต์มาต่อนั้น ค่าของที่อยู่อาจจะมีแค่หมายเลขพอร์ตของซ็อกเก็ตเท่านั้นก็ได้ (เพราะว่า ชื่อของโฮสต์นั้นได้ทราบโดยปริยายอยู่แล้ว)

- รูปแบบการติดต่อสื่อสารแบบใช้หน่วยความจำร่วม(Shared Memory Transport)

รูปแบบการติดต่อสื่อสารแบบนี้มีเฉพาะในแพลตฟอร์มของวินโดวส์แบบ 32 บิตเท่านั้น วิธีนี้จะใช้หน่วยความจำร่วม (shared memory primitive) ในการติดต่อระหว่างดีบักกี้และแอพพลิเคชั่น ของดีบักเกอร์ ซึ่งทั้งสองโพรเซสจะต้องอยู่ในเครื่องเดียวกันเท่านั้น

ในการอ้างอิงแต่ละ transport จะอ้างผ่านสตริงที่มีค่าไม่ซ้ำกัน (unique string) ที่มีชื่อว่า dt_shmem ค่านี้จะใช้ในการเลือกการติดต่อสื่อสารแบบใช้หน่วยความจำร่วมได้ตั้งแต่ตอนที่สร้างจาวาเวอร์ชวลแมชชีนหรือตอนเลือกเลือกคอนเน็คเตอร์ของเจดีไอ และสำหรับค่าที่อยู่ของการติดต่อสื่อสารแบบใช้หน่วยความจำร่วมจะต้องเป็นชื่อที่สามารถใช้เป็น ชื่อของ Win32 file-mapping object [6]ได้

### 3.3.2 คอนเน็กเตอร์ (Connector)

ใช้ในการสร้างการเชื่อมต่อระหว่างแอพพลิเคชั่นของดีบักเกอร์และดีบักกี้ ในรูปแบบอ้างอิงของการสร้างเจดีไอ (JDI reference implementation) สนับสนุนคอนเน็คเตอร์ในหลายรูปแบบ ซึ่งจะสามารถแมป (map) กับชนิดของรูปแบบการติดต่อสื่อสารที่มีอยู่ และโหมด (mode) ของการเชื่อมต่อ (launching, listening และ attaching) ซึ่งจะกล่าวต่อไปนี้

- Launching

ในวิธีนี้คอนเน็คเตอร์จะเป็นผู้เรียก (launch) จาวาเวอร์ชวลแมชชีนของดีบักกี้ และจาวาเวอร์ชวลแมชชีนอื่นๆ ที่เกี่ยวข้อง รายละเอียดของการสร้างและออปชั่นในการดีบัก (debug option) จะจัดการโดยคอนเน็คเตอร์

ส่วนรูปแบบการติดต่อสื่อสารที่ใช้ในคอนเน็คเตอร์แบบนี้จะขึ้นกับแพลตฟอร์ม นั่นคือ ในแพลตฟอร์มวินโดวส์แบบ 32 บิต จะใช้รูปแบบการติดต่อสื่อสารแบบ.ใช้หน่วยความจำร่วม แต่ในซันโซลาริสจะใช้รูปแบบการติดต่อสื่อสารแบบซ็อกเก็ต โดยอ้างผ่านชื่อ"com.sun.jdi.CommandLineLauch"

---

[5] ที่อยู่ที่ใช้ในการสื่อสาร (transport address) ซึ่งใช้ในการอ้างอิงแต่ละด้านของการติดต่อสื่อสาร

[6] นั่นคือชื่อดังกล่าวจะประกอบด้วยตัวอักษรอะไรก็ได้ยกเว้น "\"

- Attaching

คอนเน็คเตอร์นี้ถูกใช้โดยแอพพลิเคชั่นของดีบักเกอร์ในการต่อกับจาวาเวอร์ชวลแมชชีนเป้าหมาย (target VM) ที่กำลังทำงานอยู่ โดย การเรียกจาวาเวอร์ชวลแมชชีน (VM invocation) นั้นจะต้องสอดคล้องกับอาร์กิวเมนต์ (argument) ของคอนเน็คเตอร์

คอนเน็คเตอร์นี้สนับสนุนรูปแบบการติดต่อสื่อสารทั้งแบบหน่วยความจำร่วมและแบบซ็อกเก็ต โดยแบบซ็อกเก็ตจะอ้างอิงผ่านชื่อ "com.sun.jdi.SocketAttach" ส่วนแบบหน่วยความจำร่วม(มีเฉพาะในแพลตฟอร์มแบบวินโดวส์ 32 บิต) จะอ้างอิงผ่านชื่อ "com.sun.jdi.SharedMemoryAttach"

- Listening

คอนเน็คเตอร์นี้จะใช้โดยแอพพลิเคชั่นของดีบักเกอร์ เพื่อใช้ในการรับการเชื่อมต่อจากจาวาเวอร์ชวลแมชชีนเป้าหมายโดยการเรียกจาวาเวอร์ชวลแมชชีน (VM invocation) นั้นจะต้องสอดคล้องกับ อาร์กิวเมนต์ ของคอนเน็คเตอร์

คอนเน็คเตอร์แบบนี้สามารถรับการเชื่อมต่อจากหลายๆ จาวาเวอร์ชวลแมชชีนเป้าหมายได้ และสนับสนุนรูปแบบการติดต่อสื่อสารทั้งแบบหน่วยความจำร่วมและซ็อกเก็ต โดยรูปแบบการติดต่อสื่อสารแบบซ็อกเก็ตจะอ้างอิงผ่านชื่อ "com.sun.jdi.SocketListen" ส่วนรูปแบบการติดต่อสื่อสารแบบใช้หน่วยความจำร่วม (มีเฉพาะในแพลตฟอร์มของวินโดวส์ 32 บิต) จะอ้างอิงผ่านชื่อ "com.sun.jdi.SharedMemoryListen"

สำหรับในส่วนนี้จะกล่าวถึงออปชั่นที่มีความสำคัญต่อการเรียกจาวาเวอร์ชวลแมชชีนของดีบักกี้ (invoke debuggee VM) เพื่อใช้ในการดีบักโดยแอพพลิเคชั่นของดีบักเกอร์ แต่ถ้าแอพพลิเคชั่นของดีบักเกอร์ดังกล่าวใช้คอนเน็คเตอร์แบบ launching ออปชั่นที่เหมาะสมจะถูกเซตให้เราเรียบร้อยแล้ว แต่ถ้านอกเหนือจากนี้จาวาเวอร์ชวลจะถูกเรียกขึ้นมาแบบไม่อัตโนมัติ (manual) ซึ่งเราจะต้องทำการเซต ออปชั่นเอง

ออปชั่นต่างๆ มีดังต่อไปนี้

- -Xdebug ทำให้สามารถดีบักได้
- -Xnoagent ยกเลิก sun.tools.debug อันเก่าและเจพีดีเอจะใส่ "agent" ของตัวเองลงไปโดยมีรูปแบบตามข้างล่างนี้
- -Djava.compiler=NONE ยกเลิกจัสอินไทม์คอมไพเลอร์
- -Xrunjdwp:<sub option > โหลดเจดีดับเบิลยูพี ซึ่งไลบรารี่ดังกล่าวจะฝังตัวเองอยู่ที่จาวาเวอร์ชวลแมชชีนเป้าหมาย

ที่กล่าวมานั้นเป็นออปชั่นในกรณีที่ใช้จาวาเวอร์ชวลแมชชีนแบบคลาสสิค (Classic JVM) แต่หากในกรณีที่ใช้ Hotspot Performance Engine ไม่ต้องใส่ออปชั่น -Xnoagent และ -Djava.compiler=NONE

# บทที่ 4
# การออกแบบและการสร้างโปรแกรมดีบักเกอร์

## 4.1 การออกแบบดีบักเกอร์

1 ทำการสร้างดีบักเกอร์ในลักษณะที่เป็นเอาท์โพรเซสดีบักเกอร์(out-process debugger) ชนิดที่เป็นเอาท์โพรเซสดีบักเกอร์ที่มีอินโพรเซสด้วย(out-process debugger with in-process helper)

2 สร้างการเชื่อมต่อระหว่างแอพพลิเคชั่นของดีบักเกอร์และดีบักกี้ให้เป็นแบบ Launching connector

## 4.2 ขั้นตอนในการสร้างดีบักเกอร์

ขั้นตอนในการสร้างดีบักเกอร์จะมีอยู่ 2 ส่วนหลัก ๆ คือ

1 ศึกษาซอร์สโค้ดของ JDB ซึ่งเป็นเครื่องมือในที่ใช้ดีบักของ JDK

2 สร้างส่วนที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส เพิ่มเติมโดยมีพื้นฐานของ JDB จะอธิบายในบทต่อไป

## 4.3 ขั้นตอนการศึกษาซอร์สโค้ดของ JDB

1 ทำการดาวน์โหลดซอร์สโค้ดของ JDB มาจาก http://java.sun.com/products/jpda/

2 ทำการกำหนดค่าสภาวะแวดล้อม (Environment)เพื่อความพร้อมในการใช้งาน

3 ทดลองใช้งาน JDB เพื่อที่จะทราบฟังก์ชันการทำงานต่าง ๆ

4 ศึกษาคลาสหลักและคลาสที่มีเกี่ยวข้อง เพื่อนำมาพัฒนาต่อได้

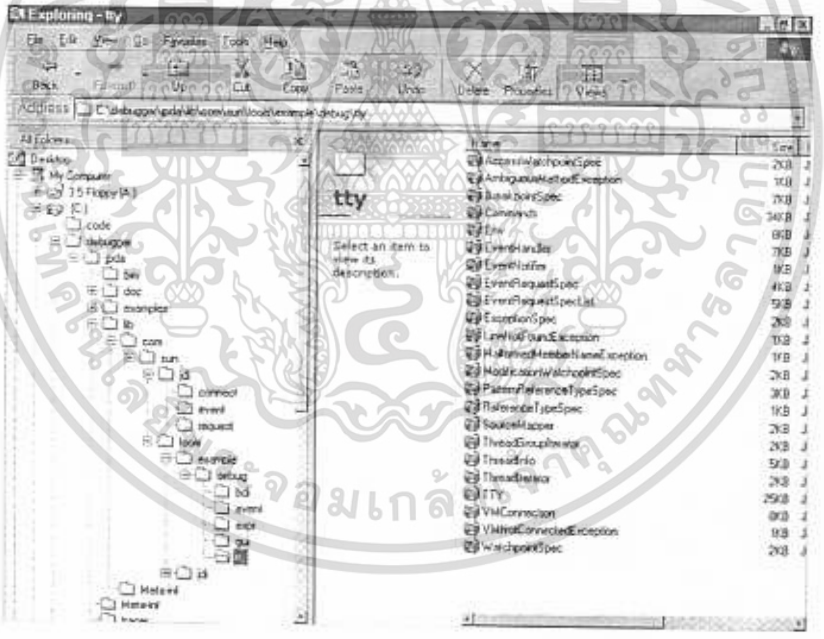
## 4.4 การกำหนดค่าสภาวะแวดล้อม

1 เมื่อทำการดาวน์โหลดแล้วจะได้เป็นไดเรกทอรี่ JPDA สมมุติว่านำมาเก็บไว้ในไดเรกทอรี่ย่อย (subdirectory) c:\debugger

2 คลาสหลัก (main) คือ คลาส TTY ซึ่งจะมีซอร์สโค้ดอยู่ในไดเรกทอรี่ย่อย C:\debugger\jpda\examples\com\sun\tools\example\debug\tty ดังแสดงในรูปที่ 4-1

3 ส่วนที่เป็นคลาสทั้งหมดที่จะอยู่ในไดเรกทอรี่ย่อย C:\debugger\jpda\lib\com\sun\tools\example\debug\tty และส่วนที่คลาสของส่วน JDI จะเป็นแพคเก็จ com.sun.jdi อยู่ในไดเรกทอรี่ย่อย C:\debugger\jpda\lib\com\sun\jdi ดังแสดงในรูปที่ 4-2


*รูปที่ 4-1 แสดงไดเรกทอรี่ย่อยที่เก็บคลาส TTY ซึ่งเป็นคลาสหลัก*




*รูปที่ 4-2 แสดงไดเรกทอรี่ย่อยที่เก็บคลาสทั้งหมดและที่เก็บคลาสของส่วน JDI*

4 ทำการกำหนดค่าสภาวะแวดล้อม ดังต่อไปนี้

4.1 คลาสพาธ (classpath) กำหนดที่คลาสของ JDI คือที่ C:\debugger\jpda\lib ได้เป็น classpath=%classpath%;C:\debugger\jpda\lib

4.2 พาธ(path) กำหนดที่เก็บ java.exe และ javac.exe ซึ่งจะต้องทำการลงโปรแกรม JDK1.2.2 ก่อน ถ้าลงไว้ที่ C:\JDK จะต้องกำหนดที่ C:\JDK\bin ได้เป็น

path = %path%;C:\JDK\bin

4.3 นำไฟล์ dt_shmem.dll, dt_socket.dll, jdwp.dll จาก C:\debugger\jpda\bin ไปใส่ไว้ใน C:\JDK\jre\bin เพื่อให้ผู้สร้างสามารถเลือกใช้ socket, serial line, shared memory หรืออื่นๆที่จะเพิ่มเติมมาในภายหลัง

4.4 การคอมไพล์เพื่อที่จะนำคลาสไปดีบักนั้นจะต้องทำการคอมไล์โดยมีออปชั่น " -g " จะทำให้สามารถเรียกดูข้อมูลระหว่างการดีบักได้ เช่น javac -g Hello.java

**4.5 ทดลองใช้งาน JDB เพื่อที่จะทราบฟังก์ชันการทำงานต่าง ๆ**

1 ฟังก์ชันที่จำเป็นจะต้องใช้งานมี ดังต่อไปนี้

| คำสั่ง | คำอธิบาย |
|---|---|
| 1. run [class [args]] | เริ่มต้นการเอ็กซีคิวต์คลาสหลักของโปรแกรม |
| 2. threads [threadgroup] | แสดงเธรดทั้งหมดที่มี |
| 3. thread <thread id> | เซต(set)เธรดปัจจุบัน |
| 4. suspend [thread id(s)] | หยุดเธรดไว้ชั่วคราว (ถ้าไม่กำหนดเธรดจะเป็นทุกเธรด) |
| 5. resume [thread id(s)] | ให้เธรดทำงานต่อดังเดิม(ถ้าไม่กำหนดเธรดจะเป็นทุกเธรด) |
| 6. where [thread id] \| all | แสดงแสตก(stack) ของเธรด |
| 7. wherei [thread id] \| all | แสดงแสตก(stack) ของเธรดและข้อมูลพีซี (pc) |
| 8. up [n frames] | เลื่อนขึ้นไปที่แสตกของเธรดตามจำนวน n |
| 9. down [n frames] | เลื่อนขึ้นไปที่แสตกของเธรดตามจำนวน n |
| 10. kill <thread> <expr> | ทำลายเธรดโดยให้เอ็กเซปชั่นออปเจค (exception object) |
| 11. print <expr> | แสดงค่าตัวแปรของ expr |
| 12. dump <expr> | แสดงข้อมูลทั้งหมดของออบเจ็คต์ |
| 13. eval <expr> | แสดงค่าของ expr เหมือนกับ print |
| 14. set <lvalue> = <expr> | เซตค่าให้กับฟิลด์/ตัวแปร/อาร์เรย์ใหม่ |
| 15. locals | แสดงตัวแปรโลคอลที่อยู่ในแสตกเฟรมปัจจุบัน(current stack frame) |
| 16. classes | แสดงคลาสที่รู้จักในปัจจุบัน |

| | |
|---|---|
| 17. class <class id> | แสดงรายละเอียดของชื่อคลาส |
| 18. methods <class id> | แสดงเมดธอด(methods)ทั้งหมดของคลาส |
| 19. fields <class id> | แสดงฟิลด์(fields)ทั้งหมดของคลาส |
| 20. threadgroups | แสดงกลุ่มของเธรดที่มีอยู่ทั้งหมด |
| 21. threadgroup <name> | เซตกลุ่มของเธรดปัจจุบัน |
| 22. stop in <class id>.<method> [(argument_type,...)] | เซตเบรคพ้อยท์(breakpoint)ในเมดธอด |
| 23. stop at <class id>:<line> | เซตเบรคพ้อยท์(breakpoint)ที่บรรทัดใด ๆ |
| 24. clear <class id>.<method> [(argument_type,...)] | เคลียร์(clear) เบรคพ้อยท์ในเมดธอด |
| 25. clear <class id>:<line> | เคลียร์(clear) เบรคพ้อยท์ที่บรรทัดใด ๆ |
| 26. clear | แสดงตำแหน่งเบรคพ้อยท์ทั้งหมด |
| 27. step | เอ็กซีคิวต์บรรทัดปัจจุบัน |
| 28. step up | เอ็กซีคิวต์จนกระทั่งเมดธอดปัจจุบันจนกระทั่งมีการรีเทิร์น(return) |
| 29. stepi | เอ็กซีคิวต์คำสั่งปัจจุบัน |
| 30. next | เสตปโอเวอร์(step over) คือเอ็กซีคิวท์โดยไม่เข้าไปในเมทธอดที่เรียกใช้งาน |
| 31. cont | เอ็กซีคิวต์ต่อไปจนถึงเบรคพ้อยท์ |
| 32. list [line number\|method] | แสดงซอร์สโค้ด |
| 33. use (or sourcepath) [source file path] | แสดงผลหรือเปลี่ยนซอร์สพาธ |
| 34. classpath | แสดงคลาสพาธของเวอร์ชวลแมชชีนเป้าหมาย |
| 35. !! | ทำคำสั่งล่าสุดซ้ำอีกครั้ง |
| 36. <n> <command> | ทำคำสั่งล่าสุดซ้ำอีก n ครั้ง |
| 37. help (or ?) | แสดงคำสั่งทั้งหมด |
| 38. exit (or quit) | ออกจากการดีบัก |

*หมายเหตุ*

*<class id>* *ชื่อคลาสที่มีชื่อของแพคเก็จหรือ ถ้าอยู่ในแพคเก็จ แล้วอาจใช้ '*' เติมข้างหน้าหรือหลังก็ได้*

*<thread id>* *เบอร์ของเธรดที่แสดงจากคำสั่ง 'threads'*

*<expr>* *นิพจน์ที่มีในภาษาจาวา*

*ตารางที่ 4-1 คำสั่งหลัก ๆ ที่มีอยู่ใน JDB*

จากที่กล่าวมาในการเซตสภาวะแวดล้อมนั้น เพื่อที่จะสามารถใช้งานฟังก์ชันการดีบักได้ เมื่อคอมไพล์จะต้องใส่ออปชั่น "-g" เพื่อสั่งให้คอมไพล์เลอร์สร้างข้อมูลที่จำเป็นต่อการดีบักอันได้แก่ตารางหมายเลขบรรทัด (Line Number Table) และตารางตัวแปรโลคอล (Local Variable Table)ไว้ในจาวาไบต์โค้ดที่สร้างขึ้น

ตารางหมายเลขบรรทัดมีความจำเป็นต่อการดีบัก เนื่องจากจะเป็นตัวแมป (map) จากโปรแกรมในแต่บรรทัดไปเป็นคำสั่งของจาวาไบต์โค้ด ซึ่งมีความจำเป็นต่อการสเตป โดยในการสเตปจะทำคำสั่งของจาวาไบต์โค้ดทีละบรรทัดและเมื่อเข้าสู่ส่วนของไบต์โค้ดที่เป็นของโปรแกรมบรรทัดถัดไป จะมีการเลื่อนบรรทัดในซอสโค้ดได้

ส่วนตารางตัวแปรโลคอลจะเป็นการจัด (assign) ตัวแปรโลคอลในจาวาเวอร์ชวลแมชชีนให้กับตัวแปรโลคอลในโปรแกรมภาษาจาวา การกระทำดังกล่าวทำให้สามารถดูค่าตัวแปรโลคอลในเวอร์ชวลแมชชีนได้ ซึ่งจะทำให้ได้ค่าที่เราสามารถมอนิเตอร์ได้ในระหว่างที่โปรแกรมทำงาน สำหรับฟังก์ชันหลัก ๆ ของการทำงานมีดังนี้

1. Step

เป็นการรันคำสั่งโปรแกรมทีละบรรทัด เพื่อดูผลความเปลี่ยนแปลงของตัวแปรต่างๆ เมื่อทำงานโปรแกรมในบรรทัดนั้น โดยกลไกการทำงานภายในจะทำโดยสั่งให้คำสั่งในระดับจาวาไบต์โค้ดรันไปทีละคำสั่งโดยเพิ่มค่าโปรแกรมเคาน์เตอร์รีจิสเตอร์ในจาวาเวอร์ชวลแมชชีน เมื่อคำสั่งเหล่านั้นทำงานจนไปตรงกับคำสั่งในระดับจาวาไบต์โค้ดที่ตรงกับโปรแกรมในภาษาจาวาในบรรทัดถัดไป (สามารถทราบได้โดยใช้ข้อมูลจากตารางหมายเลขบรรทัด) จะมีการเลื่อนแถบบรรทัดไปยังบรรทัดถัดไปเช่นกัน และนอกจากนี้ยังจะอัพเดตค่าของเธรดและตัวแปรต่างที่มอนิเตอร์อยู่

2. Next

คำสั่งนี้มีการใช้งานคล้ายกับคำสั่ง step คือจะเป็นรันคำสั่งโปรแกรมทีละบรรทัด แต่มีส่วนแต่งต่างที่จะไม่เข้าไปในส่วนของเมธอดที่เรียกใช้งาน เนื่องจากเมธอดนั้นทำงานอย่างถูกต้องอยู่แล้วจึงไม่มีความจำเป็นที่จะต้องเข้าไปดีบักในแต่ละคำสั่งของเมธอดนั้นอีก ซึ่งจะทำให้การดีบักรวดเร็วยิ่งขึ้น

3. Breakpoint

ในกรณีที่ทราบว่าโปรแกรมส่วนหนึ่งส่วนใดถูกต้องอยู่แล้ว จะไม่มีความจำเป็นที่จะต้องรันโปรแกรมทีละบรรทัด เพื่อดูความเปลี่ยนที่เกิดขึ้น ดังนั้นจะมีการใช้เบรคพอยต์เพื่อให้โปรแกรมทำงานไปเรื่อยๆ แล้วมาหยุดในจุดที่คาดว่าจะมีความผิดพลาด แล้วเริ่มต้นทำการโปรแกรมทีละบรรทัดเพื่อตรวจสอบข้อผิดพลาดของโปรแกรม

ในการใช้งานเบรคพอยต์ จะมีคำสั่งย่อย 3 คำสั่งคือ คำสั่ง set breakpoint เป็นการตั้งเบรคพอยต์ในบรรทัดที่กำหนด เพื่อให้โปรแกรมวิ่งมาหยุดที่บรรทัดนี้, คำสั่ง clear breakpoint เป็นการยกเลิกการตั้งเบรคพอยต์, และคำสั่ง run to breakpoint เป็นการสั่งให้โปรแกรมทำงานมาเรื่อยๆ จนถึงสุดแรกที่ตั้งเบรคพอยต์ไว้

1. Add watch

เป็นการดูค่าตัวแปรที่ต้องการ โดยการระบุชื่อตัวแปรที่ต้องการมา แล้วดีบักเกอร์จะทำการแสดงค่าของตัวแปรนั้นออกมา โดยการใช้คำสั่ง print <expr>

## 2 วิธีการใช้

จะมีหลายส่วนซึ่งจะอธิบายได้ดังนี้

1 เริ่มต้นการดีบัก

1.1 ทำการเซตเบรกพ้อยท์ก่อน โดยใช้คำสั่ง set in ,set at

1.2 เริ่มต้นทำการเอ็กซีคิวต์ โดยใช้คำสั่ง run

1.3 เมื่อคำสั่งปัจจุบันไปหยุดที่ตำแหน่งที่เซตเบรกพ้อยท์ไว้ จะทำการเอ็กซีคิวต์ทีละคำสั่ง,เสตปโอเวอร์ ให้ใช้คำสั่ง step, next ตามลำดับ

2 การเซตเธรดปัจจุบัน

2.1 ดูว่ามีเธรดกรุ๊ปอะไรบ้าง โดยใช้คำสั่ง threadgroups จะได้ผลลัพธ์ เช่น

1. (java.lang.ThreadGroup) 0xb9 system
2. (java.lang.ThreadGroup) 0xbd main

จะได้ว่า ชื่อของเธรดกรุ๊ปหรือ<name> คือ system และ main

thread id คือ เลข 1 และเลข 2

2.2 เซตเธรดกรุ๊ป โดยใช้คำสั่ง threadgroup <name> เช่น threadgroup main

2.3 ดูเธรดทั้งหมดที่มีอยู่ในเธรดกรุ๊ป โดยใช้คำสั่ง threads จะได้ผลลัพธ์ เช่น

1. (Simplethread) 0xbf Jamaica running
2. (Simplethread) 0xb3 Fiji running
3. (java.lang.Thread) Thread-0 cond. Waiting

จะได้ว่า ชื่อของเธรด คือ Jamica, Fiji, Thread-0

thread id คือ เลข 1, เลข 2 และเลข 3

2.4 เซตเธรดปัจจุบัน โดยใช้คำสั่ง thread <thread id> เช่น thread 2 จะได้ว่าเธรดปัจจุบันคือเธรด Fiji

2.5 สามารถที่จะเรียกดูค่าในแสตกของเธรดนี้ได้ โดยใช้คำสั่ง where [thread id]

3 การดูค่าตัวแปรต่าง ๆ

ดูได้จากตารางที่ 4-1 เช่นคำสั่งในการเรียกดูตัวแปรโลคอลจะใช้คำสั่ง local หรือต้องการดูค่าตัวแปรที่ต้องการใช้คำสั่ง print <expr>

4 การให้แสดงซอร์สโค้ด ใช้คำสั่ง list แต่ต้องทำการกำหนดพาธของซอร์สโค้ดโดยใช้คำสั่ง use <source path> แล้วจึงค่อยใช้คำสั่ง list

**4.6 ศึกษาคลาสหลักและคลาสที่มีเกี่ยวข้อง เพื่อนำมาพัฒนาต่อได้**

1 คลาส TTY

จะมีส่วนที่สามารถนำมาพัฒนาต่อได้ ดังนี้

1.1 ในเมทธอดเมนของ TTY จะมีการรับอาร์กิวเมนท์ (argument) ที่ส่งมาทางคอมมานด์ไลน์ แล้วส่งให้กับ คลาส Env เพื่อทำการสร้างการเชื่อมต่อระหว่างดีบักกี้และแอพพลิเคชั่นของดีบักเกอร์ ดังนั้นถ้าต้องการส่งให้เป็นแบบ launching connector ค่าคอนเนคเตอร์อาร์กิวเมนท์ (Connector argument) จะเป็นดังนี้

connectSpec = "com.sun.jdi.CommandLineLaunch:" + "main=" + "Hello" + ","+"options=-classpath"+ " " + "c:\\sorcecode" + ","

มีความหมายว่าต้องการ

- สร้างการเชื่อมต่อแบบ launching connectiot
- ดีบักกี้ คือ คลาส Hello
- การเซตคลาสพาทไว้ที่ c:\sourcecode

โดยจะส่งเข้าไปในคลาส Env ดังนี้

Env.init(connectSpec,true,VirtualMachine.TRACE_NONE);

หลังจากนี้จะทำการสร้างออบเจค TTY ขึ้นมาเพื่อที่จะทำหน้าที่ในการรับคำสั่งได้ตอบกับผู้ใช้งานแล้วนำมาทำงานกับเวอร์ชวลแมชชีนของดีบักเกอร์ ได้แสดงซอร์สโค้ดของคลาส TTY ที่ยังได้มีการแก้ไขไว้ในภาคผนวก ข. เพื่อจะแสดงให้เห็นว่าในเมทธอดเมนการทำงานดังที่ได้กล่าวไว้ข้างบน

1.2 ส่วนที่รับคำสั่งของการทำงาน คือส่วนเมดธอดที่ชื่อว่า executeCommand (StringTokenizer t) ซึ่งค่า t ที่ส่งไปให้คือค่าคำสั่งที่รับมาจากคอมมานด์ลายน์ (command-line) เช่น step, threads เป็นต้น ฟังก์ชันนี้จะไปเรียกให้คลาส Commands ทำงานอีกทีหนึ่ง ในส่วนนี้เราสามารถที่จะพัฒนาได้คือส่งคำสั่งเข้าไปโดยตรงโดยไม่รับจากคอมมานด์ลายน์

ตัวอย่างเช่น ถ้ามีการกดปุ่ม step ในหน้าจอกราฟฟิค จะเข้ามาที่การจัดการอีเวนท์เม้าส์คลิ๊ก (Event Mouse – click) โดยจะมีการใส่คำสั่งในส่วนของแอคชั่นลิสเทนเนอร์(Action Listener) ดังนี้

public void actionPerformed(ActionEvent e)

```
{
    StringTokenizer t =null;
    t = new StringTokenizer("step");
    tty.executeCommand(t);
     executeCommand("step)
}
```

2 คลาส Env

จะเป็นคลาสที่ทำหน้าที่ในการสร้างการเชื่อมต่อ และกำหนดสภาวะแวดล้อมต่างให้กับดีบักเกอร์ ซึ่งในการเซตค่าต่าง ๆ ตามส่งค่าตัวแปรเข้ามานั้น วิธีการจะมีการเรียกใช้งานคลาสต่าง ๆ อีกมากเช่น VMConnection ในคลาสนี้เราสามารถเรียกใช้งานได้โดยไม่ต้องทำการแก้ไข

3 คลาส Commands

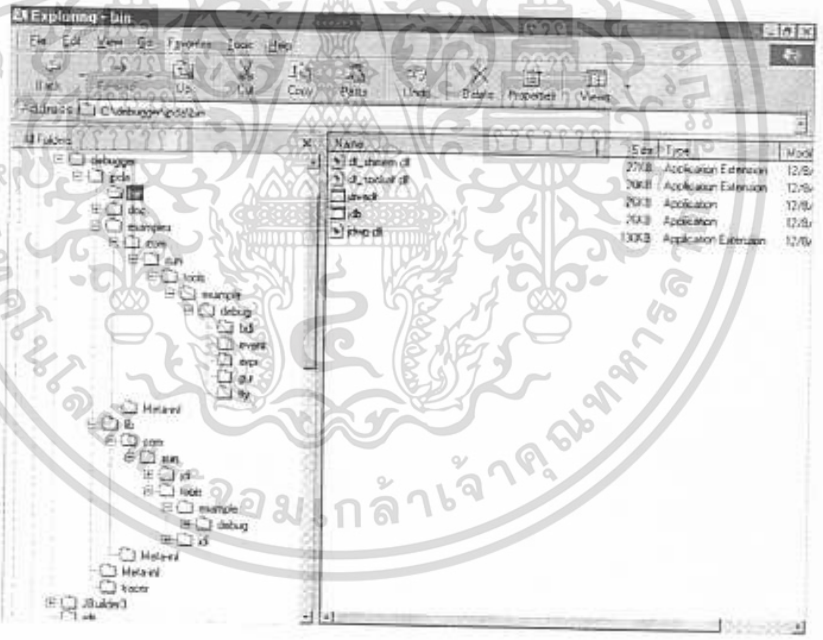
การพัฒนาโปรแกรมให้มีลักษณะกราฟฟิคยูสเซอร์อินเตอร์เฟส คลาสนี้เราสามารถนำมาพัฒนาได้โดยการเพิ่มเติมโค้ดเข้าไป โดยจากเดิมในส่วนที่มีการแสดงผลออกทางหน้าจอซึ่งเป็นสแตนดาร์ดเอาท์พุทของ JDB ก็ให้นำมาแสดงผลทางหน้าจอที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟสด้วย ซึ่งภายในคลาสนี้จะมีการเรียกใช้คลาสต่าง ๆ อีกมากมาย เช่น คำสั่งในการแสดงแสตกของเธรด commandWhere (StringTokenizer t, boolean showPC)

**4.7 อุปสรรคและการแก้ไขในการศึกษา JPDA**

เพื่อที่จะให้เข้าใจการติดตั้งและการแก้ปัญหา จะกำหนดพาธที่เก็บโปรแกรมแต่ละส่วนไว้ดังนี้

1. JDK จะเก็บไว้ดังนี้ D:\JDK ซึ่งในไดเรกทอรี่นี้จะมีไดเรกทอรี่ย่อย ดังแสดงในรูปที่ 4-3
2. JPDA จะเก็บไว้ดังนี้ C:\debugger\JPDA ซึ่งจะมีไดเรกทอรี่ย่อยดังแสดงในรูปที่ 4-4
3. โปรแกรมต่าง ๆ ที่ต้องการนำมาดีบัก จะเก็บไว้ที่ D:\code เช่น Hello.java จะเก็บดังนี้ C:\code\Hello.java ดังแสดงในรูป 4-3

*รูปที่ 4-3 แสดงพาธที่เก็บ JDK และ Code*



*รูปที่ 4-4 แสดงพาธที่เก็บ JPDA*

การเซตพาธจะต้องอ้างไปถึงพาธที่เก็บไฟล์ที่เราต้องการ เช่น ต้องการใช้ไฟล์ java.exe , javac.exe , jar.exe ที่อยู่ใน D:\JDK\bin เพราะฉะนั้นจะต้องเซตพาธ ดังนี้

PATH=D:\JDK\bin

ในส่วนของปัญหาที่เกิดขึ้น ส่วนใหญ่เนื่องมาจากเอกสารเกี่ยวกับ JPDA และ JDB มีไม่เพียงพอ ซึ่งจะมีส่วนสำคัญมากในการเซตค่าต่าง ๆ เพื่อที่จะทดลองใช้งานและทำความเข้าใจ ดังนั้นจะกล่าวถึงปัญหาต่าง ๆ และความรู้เกี่ยวกับ JPDA และ JDB เพื่อเป็นประโยชน์กับผู้ที่จะศึกษาต่อไป

1 ในไดเรกทอรี่ C:\debuger\JPDA มีรายละเอียดอย่างไรบ้าง

มีไดเรกทอรี่ย่อยดังนี้

1. bin

ในไดเรกทอรี่นี้จะประกอบด้วยไฟล์ทั้งหมด 4ไฟล์ dt_shmem.dll, dt_socket.dll, jdwp.dll, javadt.exe, jdb.exe ส่วนแรก ไฟล์ที่เป็น .dll นั้นจะต้องนำไปคัดลอกไว้ที่ D:\JDK\jre\bin ซึ่งเป็นที่เก็บไดนามิคลิงค์ไลบรารี่ที่จำเป็น ส่วนที่สองคือส่วน .exe จะมี 2 ไฟล์ คือ jdb เป็นแอพพลิเคชั่นของดีบักเกอร์ที่สร้างโดยบริษัท ซันไมโครซิสเต็มซึ่งมีลักษณะเป็นคอมมานด์ลายน์ ส่วน javadt คือแอพพลิเคชั่นของดีบักเกอร์เช่นเดียวกันแต่ว่ามีลักษณะที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส

2. doc

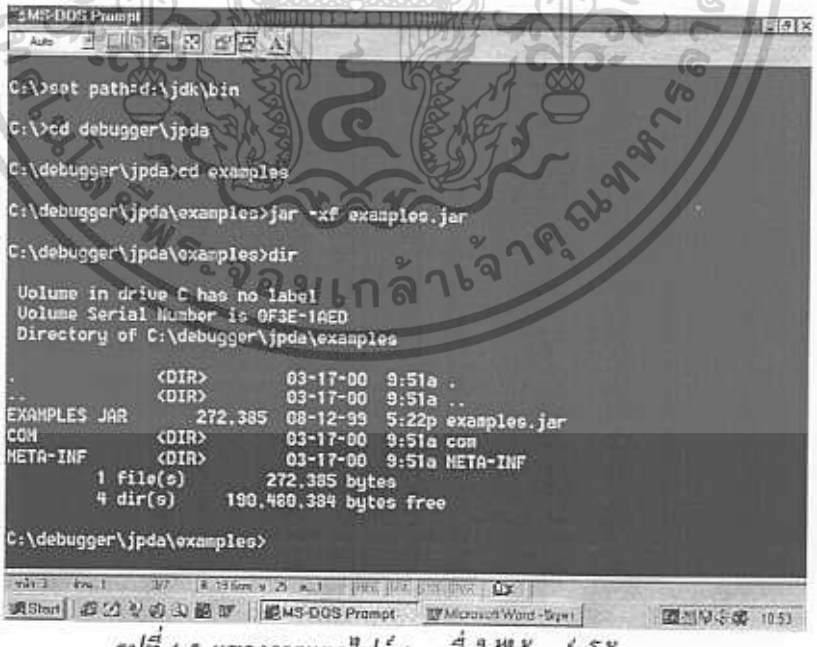
จะมีเอกสารเกี่ยวกับ JPDA ซึ่งจะอธิบายคลาสของ JDI และมีการอธิบายถึง JVMDI, JDWP, การเชื่อมต่อ(connector)แบบต่าง ๆ ด้วย

3. examples

เป็นส่วนที่มีไฟล์ examples.jar ซึ่งสามารถที่จะแตกออกเพื่อให้ได้ไฟล์ที่เป็นซอร์ทโค้ดของ JDB โดยการใช้คำสั่งที่คอมมานด์ลายน์ ดังนี้

jar -xf examples.jar

จะได้ผลลัพธ์เป็น ไดเรกทอรี่ย่อยที่เก็บซอร์ทโค้ดของ JDB ที่อยู่ในไดเรกทอรี่ย่อยคือ com\sun\tools\example\debug\tty ซึ่งในไดเรกทอรี่นี้จะประกอบด้วยซอร์ทโค้ดของคลาสทั้งหมดที่เกี่ยวข้องกับ JDB ดังแสดงในรูปที่ 4-5




*รูปที่ 4-5 แสดงการแตกไฟล์ .jar เพื่อให้ได้ซอร์สโค้ดของ JDB*

4. lib

เป็นส่วนที่มีไฟล์ jpda.jar ซึ่งจะมีคลาสของ JDI และ JDB ซึ่งจะใช้ในการเซตคลาส

พาธ

2 การทดลองรันคลาส TTY.java ทำอย่างไร

คลาส TTY.java อยู่ใน package com.sun.tools.examples.debug.tty ดังนั้นจะต้องมีการเซตพาธและคลาสพาธก่อนโดยให้มีค่าดังนี้

set path=%path%;D:\jdk\bin

set classpath=%classpath%;C:\debugger\jpda\lib\jpda.jar

วิธีการรันโปรแกรมจะทำโดย

c:\ > java com.sun.tools.example.debug.tty.TTY -classpath c:\code Hello

หมายถึงว่าสั่งให้ทำการรัน TTY โดยส่งอาร์กิวเมนท์เป็น –classpath c:\code Hello คือส่งพาธและชื่อไฟล์ที่จะทำการดีบักให้กับคลาส TTY เมื่อสามารถสร้างการเชื่อมต่อได้จะแสดงข้อความ "Initializing JDB" ดังแสดงในรูปที่ 4-6


*รูปที่ 4-6 แสดงการเซตพาธ คลาสพาธ และการรันคลาส TTY ที่เป็นเมนคลาสของ JDB*

3 การทำงานในคลาสเมน TTY เป็นอย่างไร

ที่คลาสเมนจะทำการรับค่าอาร์กิวเมนเข้ามาแล้วทำการเช็คค่าเพื่อที่จะกำหนดการเชื่อมต่อเพื่อที่จะส่งให้กับ Env.init(connectSpec, launchImmediately, traceFlags) ซึ่งคอนเนคเสปค(connectSpec) จะเป็นตัวกำหนดการเชื่อมต่อว่าจะให้เป็นการเชื่อมต่อแบบใด Launching , attaching , listening หลังจากนั้นจะทำการสร้าง(constructor) คลาส TTY ขึ้นมาอีกเพื่อที่จะรอรับคำสั่งต่าง ๆ จากคอมมานลายน์ เช่น คำสั่ง stop in , step แล้วทำการส่งให้เมทธอด executeCommand(StringTokenizer t) แล้วexecuteCommands จะทำการเรียกใช้คลาส Commands ดังโค้ดที่ตัดมาแสดงบางส่วน ต่อไปนี้

```
package com.sun.tools.example.debug.tty;

import com.sun.jdi.*;
import com.sun.jdi.event.*;
import com.sun.jdi.connect.*;
import java.util.*;
import java.io.*;

public class TTY implements EventNotifier
{
void executeCommand(StringTokenizer t) {
Commands evaluator = new Commands(out);
if (!Env.connection().isOpen() && !isDisconnectCmd(cmd)) {
        out.println("Command '" + cmd + "' is not valid until the VM is started
with the 'run' command");
    } else {
        try {
            if (cmd.equals("print")) {
                evaluator.commandPrint(t, false);
                showPrompt = false;     // asynchronous command
            } else if (cmd.equals("eval")) {
                evaluator.commandPrint(t, false);
                showPrompt = false;     // asynchronous command
            } else if (cmd.equals("set")) {
                evaluator.commandSet(t);
                showPrompt = false;     // asynchronous command
} else if (cmd.equals("step")) {
                evaluator.commandStep(t);
            } else if (cmd.equals("stepi")) {
                evaluator.commandStepi();
            } else if (cmd.equals("next")) {
                evaluator.commandNext();
}
}
```

จะเห็นได้ว่า executeCommands จะทำการเรียกคลาส Commands อีกครั้งหนึ่งโดยมีอินสแตนท์ (Instance) ชื่อว่า evaluator

เมื่อนำมาสร้างให้มีลักษณะที่เป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส สิ่งสำคัญคือ เราสามารถส่งคำสั่งเหล่านี้ผ่านใน executeCommands(StringTokenizer t)ได้โดยที่ไม่ต้องรับค่า t จากคอมมานด์ลายน์ แต่ว่าจะค่า t โดยการคลิ๊กปุ่มที่สร้างเอาไว้ซึ่งจะสามารถดีบักได้โดยมีฟังก์ชันที่เหมือนกับ JDB ได้

4 คลาสที่ทำหน้าที่ในการพิมพ์ผลลัพธ์การดีบักออกทางหน้าจอคือคลาสอะไร

คลาสที่ทำหน้าในการพิมพ์ผลลัพธ์คือคลาสที่ชื่อว่า Commands ดังตัวอย่าง ได้นำเอาส่วนที่แสดงสแตกของเธรดจากคำสั่ง where

```
package com.sun.tools.example.debug.tty;

import com.sun.jdi.*;
import com.sun.jdi.request.*;
import com.sun.tools.example.debug.expr.ExpressionParser;
import com.sun.tools.example.debug.expr.ParseException;
import java.net.URLClassLoader;
import java.util.*;
import java.io.*;

class Commands {

    private void dumpStack(ThreadInfo tinfo, boolean showPC) {
        List stack = null;
        try {
            stack = tinfo.stack();
        } catch (IncompatibleThreadStateException e) {
            out.println("Current thread isn't suspended.");
            return;
        }
        if (stack == null) {
            out.println("Thread is not running (no stack).");
        } else {
```

```
            int nFrames = stack.size();
            for (int i = tinfo.currentFrameIndex; i < nFrames; i++) {
                StackFrame frame = (StackFrame)stack.get(i);
                Location loc = frame.location();
                Method meth = loc.method();
                out.print("  [" + (i + 1) + "] ");
                out.print(meth.declaringType().name());
                out.print('.');
                out.print(meth.name());
                out.print(" (");
                if (meth instanceof Method && ((Method)meth).isNative()) {
                    out.print("native method");
                } else if (loc.lineNumber() != -1) {
                    try {
                        out.print(loc.sourceName());
                    } catch (AbsentInformationException e) {
                        out.print("<unknown>");
                    }
                    out.print(':');
                    out.print(loc.lineNumber());
                }
                out.print(')');
                if (showPC) {
                    long pc = loc.codeIndex();
                    if (pc != -1) {
                        out.print(", pc = " + pc);
                    }
                }
                out.println();
            }
        }
    }
    void commandWhere(StringTokenizer t, boolean showPC) {
        if (!t.hasMoreTokens()) {
```

```
            if (ThreadInfo.current == null) {
                out.println("No thread specified.");
                return;
            }
            dumpStack(ThreadInfo.current, showPC);
        } else {
            String token = t.nextToken();
            if (token.toLowerCase().equals("all")) {
                ThreadInfo[] list = ThreadInfo.threads();
                for (int i = 0; i < list.length; i++) {
                    ThreadInfo tinfo = list[i];
                    out.println(tinfo.thread.name() + ": ");
                    dumpStack(tinfo, showPC);
                }
            } else {
                ThreadInfo tinfo = getThread(token);
                if (tinfo != null) {
                    ThreadInfo.current = tinfo;
                }
                dumpStack(tinfo, showPC);
            }
        }
    }
```

จะเห็นได้ว่า ส่วนที่แสดงผลทางหน้าจอ out.println( ) เราสามารถให้แสดงผลทางหน้าจอกราฟฟิคที่เราออกแบบได้เช่นดียวกัน

# บทที่ 5
# การสร้างและออกแบบสภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวาแบบกราฟฟิคยูสเซอร์อินเตอร์เฟส

## 5.1 บทนำ

หลังจากการศึกษาตัวอย่างดีบักเกอร์ที่สร้างขึ้นตามสถาปัตยกรรมของดีบักเกอร์บนจาวาแพลตฟอร์มในบทที่ 4 แล้ว ต่อไปจะเป็นการพัฒนาสภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวาแบบกราฟิคยูสเซอร์อินเตอร์เฟส ซึ่งจะทำให้สามารถสร้างและแก้ไขโปรแกรมได้ และนอกจากนี้ยังทำหน้าที่เป็นเชลล์ (shell) ในการเรียกใช้คอมไพล์เลอร์, อินเตอร์พรีทเตอร์ และดีบักเกอร์ ในส่วนของดีบักเกอร์จะเชื่อมต่อกับตัวอย่างดีบักเกอร์ในบทที่ 4 ซึ่งโปรแกรมดังกล่าวเป็นแบบเปิด (open source software) ทำให้สามารถนำซอร์สโค้ดดังกล่าวมาแก้ไขให้สอดคล้องกับความต้องการได้และสามารถเชื่อมต่อกับส่วนกราฟฟิคยูสเซอร์อินเตอร์เฟสที่พัฒนาขึ้นมาใหม่ได้

## 5.2 รายละเอียดของระบบ

สภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวาแบบกราฟฟิคยูสเซอร์อินเตอร์เฟส (Graphic User Interface Integrated Development Environment for Java) ที่มีชื่อว่า KMITL Java Developer เป็นการรวบรวมเครื่องมือ (tools) ในการแก้ไข (edit), คอมไพล์ (compile), เอ็กซิคิวต์ (execute), และ ดีบัก (debug) โปรแกรมเข้าเป็นสภาพแวดล้อมการพัฒนา ทำให้การเขียนและตรวจสอบความผิดพลาดของโปรแกรมได้สะดวกและรวดเร็วยิ่งขึ้น

ผู้ใช้โปรแกรม KMITL Java Developer สามารถสร้างโปรแกรมในภาษาจาวาขึ้นมาใหม่ หรือเรียกโปรแกรมที่มีอยู่แล้วมาแก้ไข และบันทึกความเปลี่ยนแปลงได้

ส่วนในการคอมไพล์และเอ็กซิคิวต์ KMITL Java Developer จะเรียกใช้และส่งค่าอาร์กิวเมนต์และออปชั่นไปให้กับโปรแกรม javac.exe และ java.exe ซึ่งเป็นคอมไพล์เลอร์และอินเตอร์พรีทเตอร์ภาษาจาวาในชุดพัฒนาโปรแกรมภาษาจาวาของบริษัทซันไมโครซิสเต็มส์ Sun's JSDK (Java Software Development Kit)

สำหรับในส่วนของการดีบัก จะเชื่อมต่อกับโปรแกรม JDB ที่ได้กล่าวไปแล้วในบทที่ 4 โปรแกรมดังกล่าวเป็นลักษณะเปิด ซึ่งเราสามารถนำซอร์สโค้ดมาแก้ไขเพื่อให้เชื่อมต่อกับส่วนที่พัฒนาขึ้นมาใหม่ โดยจะมีฟังก์ชั่น (function) พื้นฐานเกี่ยวกับการดีบัก อันได้แก่ การตั้งและยกเลิกเบรคพอยต์ (breakpoint), การทำงานจนถึงเบรคพอยต์ที่ตั้งไว้ (run to breakpoint), การทำงานที่ละคำสั่ง (step), การทำงานทีละคำสั่งโดยไม่เข้าในเมทธอดย่อย (next) และการเรียกดูค่าตัวแปร (add watch) ซึ่งเพียงพอแก่การตรวจสอบและแก้ไขข้อผิดพลาด

เนื่องจากภาษาจาวาเป็นภาษาเชิงวัตถุ ดังนั้นในส่วนของการวิเคราะห์และออกแบบจะใช้วิธีการเชิงวัตถุ (object-oriented methodology) โดยใช้สัญลักษณ์ (notation) ที่มีชื่อว่า Unified Modeling Language ของ Object Management Group ซึ่งประกอบด้วยยูสเคส (use case) และการจำลองเหตุการณ์จริง (scenario) ซึ่งใช้ในการแสดงรายละเอียด (specification) ของระบบที่พัฒนาขึ้น และใช้คลาสไดอะแกรม (class diagram), ออบเจ็คต์ไดอะแกรม (object diagram) ในการวิเคราะห์และออกแบบ ซึ่งรายละเอียดของส่วนรายละเอียดของระบบจะอยู่ในภาคผนวก ก.

## 5.3 การออกแบบและพัฒนาระบบ

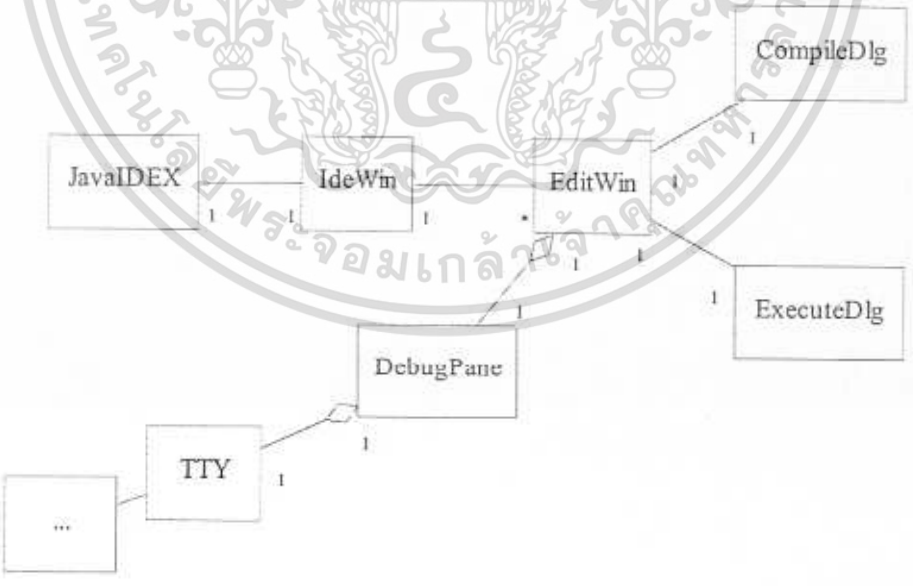
### 5.3.1 วิธีการพัฒนาซอฟต์แวร์ (Software process) ที่ใช้

จะใช้วิธีการพัฒนาซอฟต์แวร์แบบทีละเล็กละน้อย (incrementally changes) โดยเริ่มต้นพัฒนาจากส่วนที่มีความเข้าใจดีก่อนแล้วทดสอบให้มีความผิดพลาดน้อยที่สุดเท่าที่จะเป็นไปได้ แล้วค่อยๆ เพิ่มเติมส่วนต่างๆ จนกระทั่งเป็นระบบที่สมบูรณ์

### 5.3.2 การออกแบบและพัฒนาระบบ

หลังจากกำหนดรายละเอียดของโปรแกรมที่ต้องการพัฒนาแล้ว ขั้นตอนต่อไปจะเป็นการออกแบบระบบ เนื่องจากภาษาจาวาเป็นภาษาเชิงวัตถุ ดังนั้นวิธีการออกแบบก็จะใช้การออกแบบเชิงวัตถุด้วย โดยใช้สัญลักษณ์แบบ Unified Modeling Language ของ Object Management Group ซึ่งในส่วนของการออกแบบจะมีการสร้างคลาสไดอะแกรมซึ่งมีคลาสต่างๆ ที่จะต้องพัฒนา

คลาสไดอะแกรมของ KMITL Java Developer แสดงดังรูปที่ 5-1



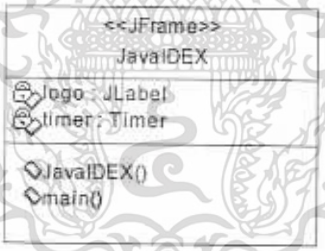

*รูปที่ 5-1 คลาสไดอะแกรมของ KMITL Java Developer*

โดยแต่ละคลาส มีหน้าที่ดังต่อไปนี้

- JavaIDEX: เป็นตัวแสดงโลโก้ (logo) ของโปรแกรมและเริ่มต้นโปรแกรม
- IdeWin: เป็นวินโดว์ (window) ของ ide สร้าง EditWin ตัวใหม่ผ่านคำสั่ง new และ open
- EditWin: เป็นวินโดว์ที่ใช้ในการสร้าง, แก้ไข และบันทึกความเปลี่ยนแปลงของไฟล์ รวมทั้งเป็นตัวเรียกฟังก์ชั่นการคอมไพล์, เอ็กซิคิวต์, และดีบัก
- CompileDlg: เป็นไดอะลอก (dialog) ที่ใช้รับออปชั่นของการคอมไพล์ และทำการคอมไพล์โปรแกรมที่อยู่ใน EditWin
- ExecuteDlg: เป็นไดอะลอกที่ใช้รับออปชั่นและอาร์กิวเมนต์ของการเอ็กซิคิวต์และเอ็กซิคิวต์
- DebugPane: เป็นหน้าต่างที่บรรจุฟังก์ชั่นต่างๆ ที่ใช้ในการดีบักเช่น step, set breakpoint และหน้าต่างที่แสดงผลจากการดีบัก โดยจะไปเรียกฟังก์ชั่นจากคลาส TTY ซึ่งเป็นโปรแกรมดีบักเกอร์ตัวอย่าง JDB
- TTY: เป็นคลาสหลักของโปรแกรมดีบักเกอร์ตัวอย่าง JDB ที่ถูกเปลี่ยนแปลงเพื่อสามารถเชื่อมต่อกับระบบที่พัฒนาขึ้นมาใหม่ได้ และเป็นส่วนที่ไปเรียกคลาสอื่นๆ ของโปรแกรม JDB ที่ใช้ในการดีบัก

สำหรับรายละเอียดของแต่ละคลาสจะแสดงในหัวข้อต่อไป

1. คลาส JavaIDEX




*รูปที่ 5-2 คลาส JavaIDEX*

จากรูปที่ 5-2 แสดงคลาส JavaIDEX ซึ่งมีส่วนของชื่อคือ JavaIDEX ซึ่งมีสเตอริโอไทพ์ (sterotype) เป็น JFrame ประกอบด้วยแอตทริบิวต์สองตัวได้แก่ logo และ timer และเมทธอดอีก 2 เมทธอดได้แก่ JavaIDEX และ main

การที่คลาส JavaIDEX มีสเตอริโอไทพ์เป็น JFrame หมายความว่ามันจะสามารถรับการถ่ายทอดคุณสมบัติ (inherit) อันได้แก่แอตทริบิวต์และเมทธอดที่มีอยู่ในคลาส JFrame ได้

สำหรับแอตทริบิวต์ จะแสดง วิธีการเข้าถึงแอตทริบิวต์, ชื่อของแอตทริบิวต์ และชนิดของมัน เช่น logo : JLabel จะมีวิธีการเข้าถึงแบบไพรเวต (private), มีชื่อของแอตทริบิวต์เป็น logo และมีชนิดเป็น JLabel สำหรับสัญลักษณ์ของวิธีการเข้าถึงแอตทริบิวต์มีดังนี้

- สำหรับการเข้าถึงแบบพับลิค (public) นั่นคือสามารถถูกใช้งานได้จากทุกๆ คลาส

- สำหรับการเข้าถึงแบบไพรเวต (private) นั่นคือสามารถถูกใช้งานได้ภายในคลาสเท่านั้น
- สำหรับการเข้าถึงแบบโพรเท็คเท็ด (protected) นั่นคือสามารถถูกใช้งานได้จากทุกๆ คลาสที่อยู่ในแพกเก็จเดียวกัน

สำหรับสัญลักษณ์ของวิธีการเข้าถึงของเมทธอด จะมีลักษณะเหมือนกันกับสัญลักษณ์ของวิธีการเข้าถึงของแอตทริบิวต์ เพียงแต่จะเปลี่ยนจากสีฟ้าเป็นสีม่วงเท่านั้น ซึ่งโดยทั่วไปแล้วแอตทริบิวต์จะมีวิธีการเข้าถึงเป็นแบบไพรเวต และเมทธอดจะมีวิธีการเข้าถึงเป็นแบบพับลิค ตามหลักการซ่อนรายละเอียด (encapsulation)

ส่วนของเมทธอดจะแสดงถึง วิธีการเข้าถึงเมทธอด, ชื่อของเมทธอด, อาร์กิวเมนต์ของเมทธอด (ถ้ามี) และชนิดของผลลัพธ์ที่ได้จากการใช้งานเมทธอด เช่น ◆ main ( arg : String[] ) : void จะมีวิธีการเข้าถึงเมทธอดแบบพับลิค, มีชื่อของเมทธอดคือ main, มีอาร์กิวเมนต์เป็นอาเรย์ของสตริงที่มีชื่อว่า arg และชนิดของผลลัพธ์ที่ได้จากการใช้งานเมทธอดเป็น void นั่นคือไม่ได้มีค่าส่งออกมาจากการใช้งานเมทธอดนี้เลย

ส่วนเมทธอด JavaIDEX จะมีลักษณะจำเพาะต่างจากเมทธอดทั่วไปที่มันจะไม่มีผลลัพธ์ที่ได้จากการใช้งานเมทธอด เนื่องจากมันเป็นคอนสตรัคเตอร์ (constructor) ซึ่งมีหน้าที่ในการเริ่มต้นการทำงานของคลาสนั้น ไม่ได้มีหน้าที่ในการคำนวณเพื่อให้ผลลัพธ์ใดๆ เลย

คลาส JavaIDEX ทำหน้าที่แสดงโลโก้ (logo) ของโปรแกรมและเริ่มต้นโปรแกรม โดยมีแอตทริบิวต์และเมทธอดของดังนี้

1.1 แอตทริบิวต์ label: JLabel

ใช้เก็บรูปภาพที่ใช้เป็นโลโกของโปรแกรม

1.2 แอตทริบิวต์ timer: Timer

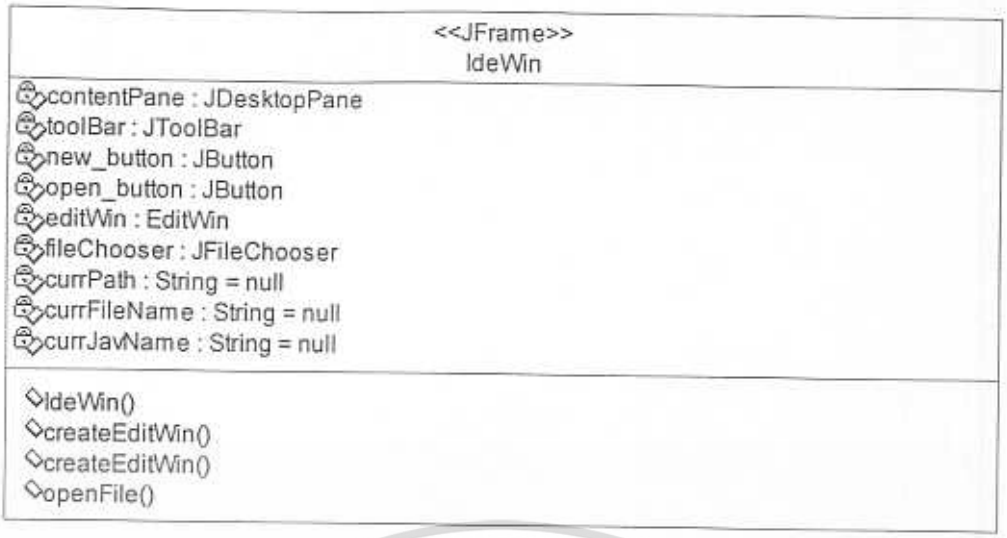
เป็นไทเมอร์ที่ใช้จับเวลาการแสดงภาพโลโกให้ได้ 3 วินาที ในแอตทริบิวต์นี้จะมีอินเนอร์คลาสที่ทำหน้าที่ลบภาพโลโกออกจากหน้าจอและสร้างออบเจ็คต์ของคลาส IdeWin

1.3 คอนสตรัคเตอร์ ◆JavaIDEX()

ทำหน้าที่เริ่มต้นการทำงานของโลโกและไทเมอร์

1.4 เมทธอด ◆main(arg: String[]):void

เป็นจุดเริ่มต้นการทำงานของคลาสนี้ ทำหน้าที่เรียกคอนสตรัคเตอร์ JavaIDEX




*รูปที่ 5-3 คลาส IdeWin*

2.2 แอตทริบิวต์ toolBar: JToolBar

เป็นทูลบาร์ (tool bar) ที่ใช้เก็บปุ่มเรียกคำสั่ง new และ open

2.3 แอตทริบิวต์ new_button: Jbutton

เป็นปุ่มที่เรียกเปิดหน้าต่าง EditWin ขึ้นมาใหม่ เพื่อใช้ในการสร้างและแก้ไขโปรแกรมใหม่ ผ่านแอคชั่นลิสเทนเนอร์ (action listener) ที่จะไปเรียกใช้งานเมทธอด createEditWin()

2.4 แอตทริบิวต์ open_button: Jbutton

เป็นปุ่มที่จะเรียก fileChooser ขึ้นมาเพื่อเลือกโหลดไฟล์ในภาษาจาวาขึ้นมาในหน้าต่าง EditWin เพื่อทำการแก้ไข ผ่านแอคชั่นลิสเทนเนอร์ที่จะไปเรียกใช้งานเมทธอด openFile()

2.5 แอตทริบิวต์ editWin: EditWin

เป็นหน้าต่างที่ใช้แก้ไขโปรแกรมภาษาจาวา

2.6 แอตทริบิวต์ fileChooser: JfileChooser

เป็นไดอะล็อกที่ใช้ในการเลือกและเปิดไฟล์ขึ้นมาแก้ไข ในที่นี้เราเซ็ตไฟล์ฟิลเตอร์ (file filter) ให้สามารถเลือกได้เฉพาะไฟล์ที่นามสกุล .java ขึ้นมาแก้ไขได้เท่านั้น โดยเซ็ตผ่านเมทธอด setFileFilter(FileFilter filefilter) ของ fileChooser เอง

2.7 แอตทริบิวต์ currFileName: String = null

เป็นสตริง (string) ที่ใช้เก็บชื่อของไดเรคทอรี่กับไฟล์ที่เลือกขึ้นมาโดย fileChooser เช่น หากเลือกไฟล์ test.java ในไดเรคทอรี่ C:\JavaProg จะได้ค่า currPathName เป็น "C:\JavaProg\test.java" เป็นต้น

2.8 แอตทริบิวต์ currJavName: String = null

เก็บชื่อไฟล์ (ไม่รวมนามสกุล) ของไฟล์ที่เลือกขึ้นมาโดย fileChooser เช่นในกรณีเดียวกับข้อ 7 จะได้ค่า currJavName เป็น "test"

2.9 แอตทริบิวต์ currPath: String = null

เก็บชื่อไดเรคทอรี่ ของไฟล์ที่เลือกขึ้นมาโดย fileChooser เช่นในกรณีเดียวกับข้อ 7 จะได้ค่า currPath เป็น "C:\JavaProg"

การเก็บค่า currFileName, currJavName และ currPath มีความสำคัญในการส่งค่าในการเรียกใช้งานโปรแกรม javac.exe และ java.exe ซึ่งเป็นคอมไพล์เลอร์และอินเตอร์พรีทเตอร์ภาษาจาวาในชุดพัฒนาโปรแกรมภาษาจาวาตามลำดับ

โดยการเรียกใช้งานโปรแกรม javac.exe จะมีรูปแบบเป็น "javac <options> <source files>" หากเราต้องการคอมไพล์โปรแกรม C:\JavaProg\test.java เราก็ส่งค่า currFileName เข้าไปเป็นในฟิลด์ <source files>

ส่วนการเรียกใช้งานโปรแกรม java.exe จะมีรูปแบบเป็น "java <options> class <arg...>" หากเราต้องการคอมไพล์โปรแกรม C:\JavaProg\test.java เราจะต้องเรียกตามรูปแบบดังนี้ "java –classpath C:\JavaProg test" ดังนั้นค่าที่จะต้องส่งให้จะเป็นรูปแบบ "java –classpath [currPath] [currJavName] นี่เป็นเหตุผลว่าเราทำไมจึงต้องแยกค่า currPath และ currJavName ออกมา

2.10 คอนสตรัคเตอร์ IdeWin()

ทำหน้าเริ่มต้นการทำงานให้กับแอตทริบิวต์ต่างๆ

2.11 เมทธอด createEditWin(): void

จะถูกเรียกเมื่อผู้ใช้กดปุ่ม new (ผ่านแอคชั่นลิสเทนเนอร์ของแอตทริบิวต์ new_Button) โดยจะทำหน้าที่ในการสร้างหน้าต่าง EditWin ตัวใหม่ขึ้นมา

2.12 เมทธอด createEditWin(String currPath, String currFileName, String currJavName, File file): void

จะถูกเรียกใช้งานโดยเมทธอด openFile() จะทำหน้าที่สร้างหน้าต่าง EditWin ขึ้นมาพร้อมทั้งโหลดไฟล์ file ที่ส่งผ่านพารามิเตอร์เข้ามาในหน้าต่าง EditWin ด้วย

2.13 เมทธอด openFile(): void

จะถูกเรียกใช้งานเมื่อผู้ใช้กดปุ่ม open โดยจะทำหน้าที่เรียก fileChooser ขึ้นมาเพื่อให้ผู้ใช้เลือกโปรแกรมในภาษาจาวาที่ต้องการแก้ไขขึ้นมา แล้วจะเรียกเมทธอด createEditWin(String currPath, String currFileName, String currJavName, File file) เพื่อให้ทำหน้าที่สร้างหน้าต่าง EditWin แล้วโหลดไฟล์ที่เลือกมาเข้าไป

3. คลาส EditWin


*รูปที่ 5-4 คลาส EditWin*

เป็นวินโดว์ที่ใช้ในการสร้าง, แก้ไข และบันทึกความเปลี่ยนแปลงของไฟล์ รวมทั้งเป็นตัวเรียกฟังก์ชั่นการคอมไพล์, เอ็กซิคิวต์, และดีบัก โดยมีแอตทริบิวต์และเมทธอดดังนี้

3.1 แอตทริบิวต์ openFrameCount: int=0

เก็บจำนวนของหน้าต่างของ EditWin ทั้งหมด เพื่อใช้จัดรูปแบบการแสดงหน้าต่าง EditWin ให้เยื้องกันตามค่า xOffset และ yOffset

3.2 แอตทริบิวต์ xOffset: int=30

เป็นค่าออฟเซ็ตที่ใช้รวมกันกับ openFrameCount และ yOffset เพื่อจัดรูปแบบการแสดงหน้าต่าง EditWin

3.3 แอตทริบิวต์ yOffset: int=30

เป็นค่าออฟเซ็ตที่ใช้รวมกันกับ openFrameCount และ xOffset เพื่อจัดรูปแบบการแสดงหน้าต่าง EditWin

3.4 แอตทริบิวต์ contentPane: JPanel

เป็นคอนเทนเนอร์ (container) หลักที่ใช้บรรจุคอมโพเน็นต์ (component) ของยูสเซอร์อินเตอร์เฟส (user interface) อื่นที่ใช้แสดงผลออกทางหน้าจอ

3.5 แอตทริบิวต์ spiltPane: JSplitPane

เป็นคอนเทนเนอร์ที่ใช้บรรจุ debugPane และ textPane ซึ่งเป็นหน้าต่างจะใช้สำหรับดีบักและเป็นอิดิทเตอร์ที่ใช้ในการแก้ไขโปรแกรมตามลำดับ โดยจะเป็นคอนเทนเนอร์ที่มีชนิดเป็น JSplitPane ซึ่งมีลักษณะที่จะแสดงสิ่งที่มันบรรจุไว้ นั่นคือ debugPane และ textPane แยกกันไว้ทางด้านซ้ายและขวา

3.6 แอตทริบิวต์ scollPane1: JScrollPane

ใช้บรรจุ textPane เป็นคอนเทนเนอร์ชนิด JScrollPane ที่จะมีสครอลบาร์ (scroll bar) ขึ้นมาหากขนาดของสิ่งที่จะต้องแสดงใหญ่กว่าเนื้อที่แสดงผลจริง

3.7 แอตทริบิวต์ scollPane1: JScrollPane

ใช้บรรจุ debugPane เป็นคอนเทนเนอร์ชนิด JScrollPane

3.8 แอตทริบิวต์ textPane: JTextPane

เป็นคอนเทนเนอร์ที่ใช้งานร่วมกับ dsd ในลักษณะ Model-view-controller โดย textPane จะทำหน้าที่เป็น view และ dsd เป็น model นั่นคือ textPane จะทำหน้าที่แสดงผลเท่านั้นส่วน dsd จะเป็นส่วนทำเก็บตัวอักษรและอัพเดตค่าตัวอักษรที่ถูกแก้ไขหรือเพิ่มเติมมา

3.9 แอตทริบิวต์ dsd: DefaultStyledDocument

เป็นส่วนของ model ที่ใช้เก็บตัวอักขระของโปรแกรมที่กำลังแก้ไขอยู่ ดังอธิบายในหัวข้อที่ 3.8

3.10 แอตทริบิวต์ debugPane: DebugPane

เป็นคอนเทนเนอร์ที่ใช้เก็บคอมโพเน็นต์ต่างๆ ที่เกี่ยวกับการดีบักได้แก่ปุ่ม step, break, unbreak, run, next watch, และ exit, หน้าต่างเธรด, และหน้าต่างตัวแปร add watch ไว้

3.11 แอตทริบิวต์ compileDlg: CompileDlg

เป็นไดอะล็อกที่ใช้ในการคอมไพล์โปรแกรมที่เรากำลังทำการแก้ไขอยู่ เหตุผลที่ต้องมีเป็นแอตทริบิวต์ชนิดโกลบอล (global) เนื่องจากจะต้องถูกเรียกใช้ก่อนการเอ็กซิคิวต์และดีบัก เพื่อที่จะได้เอ็กซิคิวต์และดีบักคลาสไฟล์ล่าสุดที่ได้แก้ไขไป

3.12 แอตทริบิวต์ file File

คือไฟล์โปรแกรมภาษาจาวาที่เรากำลังแก้ไขอยู่

3.14 แอตทริบิวต์ dirty: boolean = false

เป็นบูลลีนที่ใช้เก็บว่ามีการเปลี่ยนแปลงแก้ไขโปรแกรมในหน้าต่างอิดิทเตอร์หรือไม่ โดยจะมีค่าเป็นเท็จเมื่อไม่มีการเปลี่ยนแปลง โดยกลไกการทำงานคือ อินเนอร์คลาส MyDocumentListener จะถูกเซ็ตเป็นดอกคิวเมนต์ (document) ลิสเทนเนอร์ของ dsd เพื่อเก็บค่าความเปลี่ยนแปลงที่เก็บขึ้นใน dsd ผ่านคำสั่ง dsd.addDocumentListener(new MyDocumentListener()) โดยอินเนอร์คลาส MyDocumentListener จะมีเมทธอด 3 เมทธอดซึ่งโอเวอร์ไรด์มาจากคลาส DocumentListener ได้แก่ insertUpdate(), removeUpdate(), changedUpdate() โดยเมทธอดเหล่านี้จะทำการเซ็ตค่า dirty ให้เป็นจริงเมื่อมีการเปลี่ยนแปลง, ลบ หรือเปลี่ยนค่าใน dsd ตามลำดับ

3.15 แอตทริบิวต์ menuBar: JMenuBar

เป็นเมนูบาร์หลักของหน้าต่าง EditWin ซึ่งประกอบด้วยเมนู file, environment และ command

3.16 แอตทริบิวต์ fileMenu: JMenu

เป็นเมนูที่บรรจุเมนูย่อย save, save as และ exit

3.17 แอตทริบิวต์ envMenu: JMenu

ชื่อเต็มที่ปรากฏบนหน้าจอคือเมนู Environment เป็นเมนูที่บรรจุเมนูย่อย set environment

3.18 แอตทริบิวต์ commmandMenu: JMenu

เป็นเมนูที่บรรจุเมนูย่อย compile, execute และ debug

3.19 แอตทริบิวต์ file_Save_menuItem: JMenuItem

เมนูย่อย Save ที่ใช้บันทึกความเปลี่ยนแปลงของโปรแกรมที่เราแก้ไขอยู่ โดยเมื่อผู้ใช้เลือกเมนูย่อยนี้แล้ว แอคชั่นลิสเทนเนอร์จะไปเรียกใช้เมทธอด saveFile()

3.20 แอตทริบิวต์ file_SaveAs_menuItem: JMenuItem

เมนูย่อย Save as จะใช้บันทึกความเปลี่ยนแปลงของโปรแกรมที่เราแก้ไขไปยังไฟล์อื่น โดยจะมี fileChooser ขึ้นมาให้เลือกไฟล์เป้าหมายที่ต้องการบันทึก

โดยเมื่อผู้ใช้เลือกเมนูย่อยนี้แล้ว แอคชั่นลิสเทนเนอร์ของมันจะไปเรียกใช้เมทธอด SaveAsFile()

3.21 แอตทริบิวต์ file_Exit_menuItem: JMenuItem

เมนูย่อย Exit จะใช้ปิดหน้าต่าง EditWin โดยจะตรวจสอบว่ามีการเปลี่ยนแปลงในโปรแกรมหลังจากการบันทึกครั้งล่าสุดหรือไม่ ถ้ามีจะขึ้นไดอะล็อกมาถามว่าต้องการบันทึกความเปลี่ยนแปลงที่เกิดขึ้นหรือไม่

โดยเมื่อผู้ใช้เลือกเมนูย่อยนี้แล้ว แอคชั่นลิสเทนเนอร์ของมันจะไปเรียกใช้งานเมทธอด SaveChangeBeforeExit()

3.22 แอตทริบิวต์ env_setEnv_menuItem: JMenuItem

เมนูย่อย Set Environment ใช้ในการเซ็ตค่า JDKLoc ซึ่งเป็นตำแหน่งของ JDK โดยจะมี fileChooser ขึ้นมาให้ผู้ใช้กำหนดตำแหน่งที่ไฟล์ javac.exe อยู่ ซึ่งทำให้โปรแกรมสามารถทราบตำแหน่งของ JDK ได้

โดยเมื่อผู้ใช้เลือกเมนูย่อยนี้แล้ว แอคชั่นลิสเทนเนอร์ของมันจะไปเรียกใช้งานเมทธอด setJDKLoc()

3.23 แอตทริบิวต์ command_Compile_menuItem: JMenuItem

เมนูย่อย compile ใช้ในการคอมไพล์โปรแกรมที่เรากำลังแก้ไขอยู่ โดยเมื่อผู้ใช้เลือกเมนูย่อยนี้แล้ว แอคชั่นลิสเทนเนอร์ของมันจะไปสร้างและแสดงไดอะล็อก CompileDlg เพื่อให้ผู้ใช้ทำการคอมไพล์โปรแกรม

3.24 แอตทริบิวต์ command_Execute_menuItem: JMenuItem

เมนูย่อย execute ใช้ในการเอ็กซิคิวต์โปรแกรมที่เรากำลังแก้ไขอยู่ โดยเมื่อผู้ใช้เลือกเมนูย่อยนี้แล้ว แอคชั่นลิสเทนเนอร์ของมันจะไปสร้างและแสดงไดอะล็อก ExecuteDlg เพื่อให้ผู้ใช้ทำการเอ็กซิคิวต์โปรแกรม

3.25 แอตทริบิวต์ command_Debug_menuItem: JMenuItem

เมนูย่อย debug ใช้ในการดีบักโปรแกรมที่เรากำลังแก้ไขอยู่ โดยเมื่อผู้ใช้เลือกเมนูย่อยนี้แล้ว แอคชั่นลิสเทนเนอร์ของมันจะไปสร้างและแสดง debugPane เพื่อให้ผู้ใช้ทำการดีบักโปรแกรม

3.26 แอตทริบิวต์ currFileName: String = null

เป็นสตริง (string) ที่ใช้เก็บชื่อไดเรคทอรี่กับไฟล์ของไฟล์ที่เลือกขึ้นมาโดย fileChooser โดยค่านี้จะส่งมาจาก IdeWin

3.27 แอตทริบิวต์ currJavName: String = null

เก็บชื่อไฟล์ (ไม่รวมนามสกุล) ของไฟล์ที่เลือกขึ้นมาโดย fileChooser โดยค่านี้จะส่งมาจาก IdeWin

3.28 แอตทริบิวต์ currPath: String = null

เก็บชื่อไดเรคทอรี่ ของไฟล์ที่เลือกขึ้นมาโดย fileChooser โดยค่านี้จะส่งมาจาก IdeWin

3.29 แอตทริบิวต์ JDKLoc: String = "d:\\jdk\\bin"

ใช้เก็บค่าตำแหน่งของ JDK โดยจะมีค่าดีฟอลต์เป็น "d:\\jdk\\bin"

3.30 แอตทริบิวต์ SimpleAttributeSet[];

เก็บรูปแบบอันได้แก่ชื่อฟอนต์, ขนาด, ลักษณะ (ตัวหนา,ตัวเอียง, ตัวปกติ) และสี ของตัวอักษรที่จะแสดง โดยอักษรทุกแบบจะใช้ฟอนต์ Sans Sarif ขนาด 12 พอยต์ โดยที่อักษรธรรมดาจะเป็นสีดำ, อักษรที่แสดงการเซตเบรคพอยต์จะเป็นตัวหนาสีแดง, และอักษรที่แสดงการตำแหน่งที่โปรแกรมรันมาถึงจะเป็นตัวหนาสีน้ำเงิน

3.31 แอตทริบิวต์ attrs: SimpleAttributeSet[];

ใช้เก็บแอตทริบิวต์ของตัวอักษร ดังที่ได้กล่าวในข้อที่ 3.29

3.32 แอตทริบิวต์ dbp_viewport: JViewport

ในกรณีของการดีบักซึ่งจะมีฟังก์ชั่นการ step ที่ตำแหน่งของส่วนที่โปรแกรมกำลังรันอยู่จะเลื่อนลงไปเรื่อยๆ เมื่อผู้ใช้กดปุ่มในแต่ละครั้ง ซึ่งอาจทำให้ตำแหน่งที่โปรแกรมกำลังรันอยู่เลยหน้าจอลงไป ดังนั้นเราจึงจำเป็นต้องใช้ viewport ซึ่งมีหน้าที่เลื่อนตำแหน่งของสครอลบาร์ไปยังตำแหน่งที่เราต้องการแสดงผล

3.33 แอตทริบิวต์ dbv_point: Point

เป็นตำแหน่งของ dpb_viewport โดยจุดนี้จะเป็นจุดซ้ายบนสุดของ dpb_viewport ดังนั้นเมื่อเราต้องการให้ dpb_viewport เลื่อนไปที่ตำแหน่งใด เราก็ต้องมาเซ็ตที่แอตทริบิวต์นี้

3.34 แอตทริบิวต์ line2Pixel:int =17

เป็นค่าจำนวนพิกเซลต่อความกว้างหนึ่งบรรทัดของหน้าจอ textPane ซึ่งจะต้องใช้ในการแปลงค่าบรรทัดที่โปรแกรมกำลังรันอยู่ไปเป็นตำแหน่งพิกเซลเพื่อนำไปเซตค่า dbv_point

3.35 แอตทริบิวต์ fileChooser: JfileChooser

เป็นไดอะล็อกที่ใช้ในการเลือกและเปิดไฟล์ขึ้นมาแก้ไข ในที่นี้เราเซ็ตไฟล์ฟิลเตอร์ (file filter) ให้สามารถเลือกได้เฉพาะไฟล์ที่นามสกุล .java ขึ้นมาแก้ไขได้เท่านั้น โดยเซ็ตผ่านเมทธอด setFileFilter(FileFilter filefilter) ของ fileChooser เอง

3.36 แอตทริบิวต์ threshold: int

ใช้เก็บค่าจุดกึ่งกลางของหน้าจอเมื่อมีการ step โปรแกรมไปเรื่อยๆ เพื่อให้ dpv_point เลื่อนไปยังจุดที่แสดงตำแหน่งที่โปรแกรมรันอยู่ตรงกลางเสมอ

3.37 แอตทริบิวต์ char2line[]: int

ใช้เก็บค่าตำแหน่งตัวอักษรสุดท้ายของแต่ละบรรทัด เพื่อใช้ในการเซ็ตค่าตำแหน่งที่โปรแกรมรันอยู่

3.38 คอนสตรัคเตอร์ EditWin()

จะถูกเรียกเมื่อผู้ใช้กดปุ่ม new ในหน้าต่าง IdeWin โดยจะไปเรียกเมทธอด createWin() เพื่อสร้างหน้าต่าง EditWin ขึ้นมาใหม่

3.39 คอนสตรัคเตอร์ EditWin(String currPath, String currFileName, String currJavName, File file)

จะถูกเรียกเมื่อผู้ใช้กดปุ่ม open ในหน้าต่าง IdeWin เพื่อเลือกโปรแกรมภาษาจาวาขึ้นมาแก้ไข โดยจะไปเรียกเมทธอด createWin() เพื่อสร้างหน้าต่าง EditWin ขึ้นมาใหม่ แล้วโหลดโปรแกรมภาษาจาวาที่ได้เลือกไว้เข้าไปใน textPane

3.40 เมทธอด ◆createWin(): void

ทำหน้าที่สร้างและเริ่มต้นการทำงานให้กับหน้าต่าง EditWin

3.41 เมทธอด ◆setSDKLoc(): void

จะถูกเรียกโดยแอคชั่นลิสเทนเนอร์ของเมนูย่อย env_SetEnv_menuItem เมื่อผู้ใช้เลือกเมนูย่อย Set Environment ในเมนู Environment

เมทธอดนี้จะทำงานโดยแสดง fileChooser เพื่อให้ผู้ใช้เลือกตำแหน่งของไฟล์ javac.exe ซึ่งจะทำให้สามารถทราบตำแหน่งของ JDK ได้ ตำแหน่งกล่าวจะถูกเซ็ตไว้ในแอตทริบิวต์ JDKLoc

3.42 เมทธอด ◆SaveFile(): boolean

จะถูกเรียกโดยแอคชั่นลิสเทนเนอร์ของเมนูย่อย file_Save_menuItem เมื่อผู้ใช้เลือกเมนูย่อย Save ในเมนู File

โดยเริ่มต้นจะตรวจสอบว่า currFileName เป็น null หรือไม่ นั่นคือมีการตั้งชื่อให้ไฟล์นี้แล้วหรือยัง ถ้าเป็น null จะรีเทิร์นไปยังเมทธอด SaveAsFile() เพื่อเลือกตำแหน่งและชื่อของไฟล์เป้าหมายที่ต้องการบันทึก แล้วจะรีเทิร์นกลับมาที่เมทธอด SaveFile() อีกทีเพื่อบันทึกโปรแกรมลงไปยังไฟล์เป้าหมายที่เลือกไว้

ส่วนในกรณีที่ currFileName ไม่เป็น null ก็จะบันทึกโปรแกรมลงยังไฟล์ตามชื่อใน currFileName

3.43 เมทธอด ◆SaveAsFile(): boolean

จะถูกเรียกโดยแอคชั่นลิสเทนเนอร์ของเมนูย่อย file_SaveAs_menuItem เมื่อผู้ใช้เลือกเมนูย่อย Save as ในเมนู File

โดยจะมี fileChooser ขึ้นมาให้เลือกตำแหน่งและชื่อของไฟล์เป้าหมายที่ต้องการบันทึก แล้วจะรีเทิร์นมาที่เมทธอด SaveFile() อีกทีเพื่อบันทึกโปรแกรมลงไปยังไฟล์เป้าหมายที่เลือกไว้

3.44 เมทธอด ◆SaveChangeBeforeExit(): boolean

จะถูกเรียกโดยแอคชั่นลิสเทนเนอร์ของเมนูย่อย file_Exit_menuItem เมื่อผู้ใช้เลือกเมนูย่อย Exit ในเมนู File

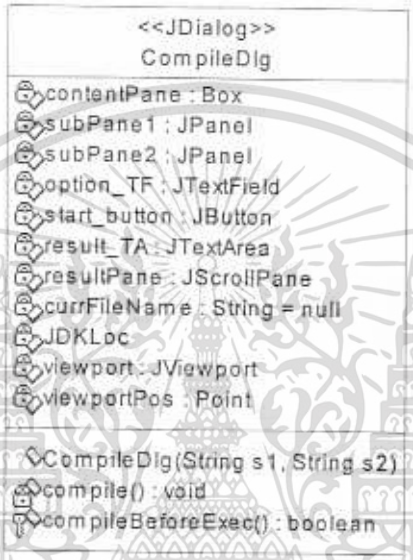
ในกรณีที่ค่าแอตทริบิวต์ dirty เป็นจริง นั่นคือมีการเปลี่ยนแปลงโปรแกรมหลังจากการบันทึกครั้งล่าสุด จะมีไดอะล็อกขึ้นมาถามผู้ใช้ว่าต้องการปิดหน้าต่าง EditWin จริงๆ หรือไม่ และต้องการบันทึกความเปลี่ยนแปลงของโปรแกรมที่ได้แก้ไขไปครั้งล่าสุดหรือไม่ โดยจะให้ผู้ใช้เลือก 3 ทางคือ yes, no, cancel คือ ต้องการบันทึกความเปลี่ยนแปลง, ไม่ต้องการบันทึกความ

เปลี่ยน, และยกเลิกการปิดหน้า EditWin ตามลำดับ โดยถ้าผู้ใช้เลือก yes จะมีการเรียกเมทธอด SaveFile() เพื่อบันทึกความเปลี่ยนแปลงที่เกิดขึ้น

ส่วนในกรณีที่ค่าแอตทริบิวต์ dirty เป็นเท็จ จะปิดหน้าต่าง EditWin ไปทันที

**4 คลาส CompileDlg**

เป็นไดอะล็อก (dialog) ที่ใช้รับออปชั่นของการคอมไพล์ และทำการคอมไพล์โปรแกรมที่อยู่ใน EditWin โดยในการทำงานจะมีแอตทริบิวต์และเมทธอดดังนี้




*รูปที่ 5-5 คลาส CompileDlg*

4.1 แอตทริบิวต์ contentPane: Box

เป็นคอนเทนเนอร์ (container) หลักที่ใช้บรรจุคอมโพเน็นต์ (component) ของยูสเซอร์อินเตอร์เฟส (user interface) อื่นที่ใช้แสดงผลออกทางหน้าจอ

4.2 แอตทริบิวต์ subPane1: JPanel

เป็นคอนเทนเนอร์ย่อยที่บรรจุ option_TF

4.3 แอตทริบิวต์ subPane2: JPanel

เป็นคอนเทนเนอร์ย่อยที่บรรจุ resultPane

4.4 แอตทริบิวต์ option_TF: JTextField

เป็นเท็กท์ฟิลด์ที่ให้ผู้ใช้ป้อนค่าออปชั่นที่ต้องการให้จาวาคอมไพล์เลอร์ทำงาน

4.5 แอตทริบิวต์ start_button: JButton

เป็นปุ่มที่เมื่อผู้ใช้กดแล้วจะเริ่มการทำงาน โดยแอคชั่นลิสเทนเนอร์ของมันจะไปเรียกใช้งานเมทธอด Compile() เพื่อทำการคอมไพล์โปรแกรม

4.6 แอตทริบิวต์ result_TA: JTextArea

เป็นพื้นที่แสดงข้อผิดพลาดที่เกิดจากการคอมไพล์

4.7 แอตทริบิวต์ currFileName: String = null

เป็นชื่อของไฟล์โปรแกรมภาษาจาวาที่ต้องการคอมไพล์ เป็นค่าที่ส่งมาจากคลาส EditWin ตอนเรียกสร้างออบเจ็คต์ของคลาส CompileDlg

4.8 แอตทริบิวต์ JDKLoc: String

เป็นเป็นตำแหน่งของ JDK เป็นค่าที่ส่งมาจากคลาส EditWin ตอนเรียกสร้างออบเจ็คต์ของคลาส CompileDlg

4.9 แอตทริบิวต์ resultPane: JScrollPane

เป็นคอนเทนเนอร์ของ result_TA ที่เป็นชนิด JScrollPane ซึ่งจะมีสครอลบาร์ขึ้นมาในกรณีที่ขนาดของ result_TA ใหญ่กว่าพื้นที่แสดงผลจริง

4.10 แอตทริบิวต์ viewport: JViewport

เป็นวิวพอร์ตที่บังคับให้สครอลบาร์ของ resultPane เลื่อนมาที่ตำแหน่งซ้ายบนเสมอ เพื่อให้ผู้ใช้สามารถดูผลลัพธ์จากการคอมไพล์ตั้งแต่เริ่มต้น เนื่องจากถ้าไม่ใช่วิวพอร์ตแล้วตำแหน่งของสครอลบาร์จะอยู่ที่ตำแหน่งสุดท้ายของข้อความเสมอ

4.11 แอตทริบิวต์ viewportPos: Point

เป็นจุดที่ใช้กำหนดตำแหน่งบนซ้ายสุดของ viewport โดยในที่นี้จะมีค่าเป็น [0,0] เสมอ เนื่องจากต้องการให้ตำแหน่งของ resultPane อยู่ซ้ายบนสุดเสมอ

4.12 คอนสตรัคเตอร์ compileDlg(String s1, String s2)

เป็นส่วนที่เริ่มการทำงานของคลาส โดยสตริง s1 และ s2 จะเป็นค่าของ currFileName และ JDKLoc ตามลำดับ

4.13 เมทธอด compile(): void

จะทำการคอมไพล์โปรแกรมโดยผ่านออบเจ็คต์ Runtime ของ "JDKLoc+ "\\javac.exe " + "-g " + options_TF.getText()+ " " + currFileName" โดยการคอมไพล์ที่ใช้ออปชั่น "-g" จำมีการสร้างข้อมูลที่จำเป็นต่อการดีบัก

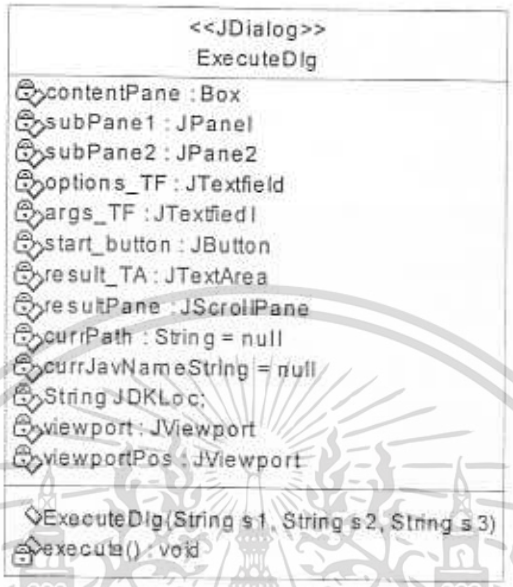
โดยเมื่อคอมไพล์เสร็จจะมีไดอะล็อกขึ้นมาให้ผู้ใช้ทราบว่าได้คอมไพล์เสร็จเรียบร้อย แล้วผู้ใช้จะตรวจดูข้อผิดพลาดที่เกิดจากการคอมไพล์โปรแกรมใน result_TA

4.14 เมทธอด compileBeforeExec(): void

จะถูกเรียกใช้ก่อนมีการเอ็กซิคิวต์หรือดีบักโดยคลาส EditWin เพื่อให้ได้จาวาไบต์โค้ดของโปรแกรมที่แก้ไขล่าสุด โดยถ้าหากมีข้อผิดพลาดจากการโปรแกรม จะแสดงไดอะล็อกแจ้งให้ยูสเซอร์ทราบว่ามีความผิดพลาด เพื่อให้ยูสเซอร์แก้ไขและคอมไพล์อีกครั้ง

**5 คลาส ExecuteDlg**

เป็นไดอะล็อก (dialog) ที่ใช้รับออปชั่นและอาร์กิวเมนต์ของการเอ็กซิคิวต์ และทำการเอ็กซิคิวต์จาวาไบต์โค้ดของโปรแกรมที่อยู่ใน EditWin โดยในการทำงานจะมีแอตทริบิวต์และเมทธอดดังนี้




*รูปที่ 5-6 คลาส ExecuteDlg*

5.1 แอตทริบิวต์ contentPane: Box

เป็นคอนเทนเนอร์ (container) หลักที่ใช้บรรจุคอมโพเน็นต์ (component) ของยูสเซอร์อินเตอร์เฟส (user interface) อื่นที่ใช้แสดงผลออกทางหน้าจอ

5.2 แอตทริบิวต์ subPane1: JPanel

เป็นคอนเทนเนอร์ย่อยที่บรรจุ option_TF

5.3 แอตทริบิวต์ subPane2: JPanel

เป็นคอนเทนเนอร์ย่อยที่บรรจุ resultPane

5.4 แอตทริบิวต์ option_TF: JTextField

เป็นเท็กท์ฟิลด์ที่ให้ผู้ใช้ป้อนค่าออปชั่นที่ต้องการให้จาวาอินเตอร์พรีทเตอร์ทำงาน

5.5 แอตทริบิวต์ args_TF: JTextField

เป็นเท็กท์ฟิลด์ที่ให้ผู้ใช้ป้อนค่าอาร์กิวเมนต์ที่ต้องการให้จาวาอินเตอร์พรีทเตอร์ทำงาน

5.6 แอตทริบิวต์ start_button: JButton

เป็นปุ่มที่เมื่อผู้ใช้กดแล้วจะเริ่มการทำงาน โดยแอคชั่นลิสเทนเนอร์ของมันจะไปเรียกใช้งานเมทธอด Execute() เพื่อทำการเอ็กซิคิวต์โปรแกรม

5.7 แอตทริบิวต์ result_TA: JTextArea

เป็นพื้นที่แสดงข้อผิดพลาดที่เกิดจากการเอ็กซิคิวต์

5.8 แอตทริบิวต์ currPath: String = null

เป็นพาธของโปรแกรมภาษาจาวาที่ต้องการเอ็กซิคิวต์ เป็นค่าที่ส่งมาจากคลาส EditWin ตอนเรียกสร้างออบเจ็คต์ของคลาส ExecuteDlg

5.9 แอตทริบิวต์ currJavName: String = null

เป็นชื่อของไฟล์โปรแกรมภาษาจาวาที่ต้องการเอ็กซิคิวต์ เป็นค่าที่ส่งมาจากคลาส EditWin ตอนเรียกสร้างออบเจ็คต์ของคลาส ExecuteDlg

5.10 แอตทริบิวต์ JDKLoc: String

เป็นเป็นตำแหน่งของ JDK เป็นค่าที่ส่งมาจากคลาส EditWin ตอนเรียกสร้างออบเจ็คต์ของคลาส ExecuteDlg

5.11 แอตทริบิวต์ resultPane: JScrollPane

เป็นคอนเทนเนอร์ของ result_TA ที่เป็นชนิด JScrollPane ซึ่งจะมีสครอลบาร์ขึ้นมาในกรณีที่ขนาดของ result_TA ใหญ่กว่าพื้นที่แสดงผลจริง

5.12 แอตทริบิวต์ viewport: Jviewport

เป็นวิวพอร์ตที่บังคับให้สครอลบาร์ของ resultPane เลื่อนมาที่ตำแหน่งซ้ายบนเสมอ เพื่อให้ผู้ใช้สามารถดูผลลัพธ์จากการคอมไพล์ตั้งแต่เริ่มต้น เนื่องจากถ้าไม่ใช่วิวพอร์ตแล้วตำแหน่งของสครอลบาร์จะอยู่ที่ตำแหน่งสุดท้ายของข้อความเสมอ

5.13 แอตทริบิวต์ viewportPos: Point

เป็นจุดที่ใช้กำหนดตำแหน่งบนซ้ายสุดของ viewport โดยในที่นี้จะมีค่าเป็น [0,0] เสมอ เนื่องจากต้องการให้ตำแหน่งของ resultPane อยู่ซ้ายบนสุดเสมอ

5.14 คอนสตรัคเตอร์ ◆ExecuteDlg(String s1, String s2, String s3)

เป็นส่วนที่เริ่มการทำงานของคลาส โดยสตริง s1, s2 และ s3 จะเป็นค่าของ currPath, currJavName และ JDKLoc ตามลำดับ

5.15 เมทธอด execute(): void

จะทำการคอมไพล์โปรแกรมโดยผ่านออบเจ็คต์ Runtime ของ "JDKLoc+ "\\java.exe " + "-classpath " + currPath + " " +options_TF.getText()+ " " + currFileName + " " args_TF.getText()"

โดยหากเกิดรันไทม์เออเรอร์ (runtime error) จะแสดงข้อความของความผิดพลาดที่ result_TA

**6. คลาส DebugPane**

เป็นหน้าต่างที่บรรจุฟังก์ชั่นต่างๆ ที่ใช้ในการดีบักเช่น step, set breakpoint และหน้าต่างที่แสดงผลจากการดีบัก โดยจะไปเรียกฟังก์ชั่นจากคลาส TTY ซึ่งเป็นโปรแกรมดีบักเกอร์ตัวอย่าง JDB โดยมีแอตทริบิวต์และเมทธอดดังนี้

6.1 แอตทริบิวต์ currJavName: String = null

เป็นชื่อของไฟล์โปรแกรมภาษาจาวาที่ต้องการดีบัก เป็นค่าที่ส่งมาจากคลาส EditWin ตอนเรียกสร้างออบเจ็คต์ของคลาส DebugPane

6.2 แอตทริบิวต์ currPath: String = null

เป็นพาธของโปรแกรมภาษาจาวาที่ต้องการดีบัก เป็นค่าที่ส่งมาจากคลาส EditWin ตอนเรียกสร้างออบเจ็คต์ของคลาส DebugPane

6.3 แอตทริบิวต์ connecSpec: String = null

เป็นตัวแปรที่ใช้ส่งให้กับ Env.init(String connectSpec, boolean openNow, int flags) ซึ่งจะเป็นการกำหนดการเชื่อมต่อระหว่างดีบักกี้และแอพพลิเคชั่นของดีบักเกอร์ ดังนั้นถ้าต้องการส่งให้เป็นแบบ launching connector คำคอนเนคเตอร์อาร์กิวเมนท์(Connector argument) จะเป็นดังนี้

connectSpec = "com.sun.jdi.CommandLineLaunch:" + "main=" + "Hello" + ","+"options=-classpath"+" " + "c:\\sorcecode" + ","

6.5 แอตทริบิวต์ tty: TTY

เป็นออบเจ็คต์ของคลาส TTY

6.6 แอตทริบิวต์ debugger: Thread = null

เป็นเธรดของคลาสนี้ ที่โอเวอร์ไรด์มาจากคลาส Runnable โดยทำให้สามารถแสดงหน้าต่างการดีบักได้พร้อมๆ กับอัพเดตค่าใน EditWin เช่นทำสีตัวอักษรให้ทราบว่ามีการเซ็ตเบรคพอยต์เป็นต้น โดยเธรดนี้จะเริ่มต้นเมื่อผู้ใช้เลือกเมนูย่อย debug ในเมนู command ของหน้าต่าง EditWin เพื่อเป็นการเริ่มต้นการดีบัก

6.7 แอตทริบิวต์ contentPane : Box

เป็นคอนเทนเนอร์ (container) หลักที่ใช้บรรจุคอมโพเน็นต์ (component) ของยูสเซอร์อินเตอร์เฟส (user interface) อื่นที่ใช้แสดงผลออกทางหน้าจอ

6.8 แอตทริบิวต์ jToolBar: JToolBar

เป็นทูลบาร์ (tool bar) ที่ใช้เก็บปุ่มเรียกคำสั่งต่างๆ ของดีบักเกอร์ ได้แก่ break, unbreak, run, step, next, add watch, และ exit


*รูปที่ 5-7 คลาส DebugPane*

6.9 แอตทริบิวต์ BreakPoint_bt: JButton

เป็นปุ่มที่ใช้เรียกเซ็ตเบรคพอยต์ โดยผู้ใช้จะต้องคลิ๊กที่บรรทัดที่ต้องการเซ็ตก่อนแล้วจึงกดปุ่มนี้ โดยแอคชั่นลิสเทนเนอร์ของมันจะนำค่า dot และ carerCoords มาเปลี่ยนตัวอักษรในบรรทัดให้เป็นแดง เพื่อให้ทราบว่าบรรทัดนี้มีการเซ็ตเบรคพอยต์

โดยจะทำการส่งค่าตัวแปร t ที่มีชนิดเป็น StringTokenizer เข้าไปใน tty.executeCommand(t); โดยกำหนดค่า t ดังนี้ t = new StringTokenizer("stop at"+" "+currJavName+":"+lineAct );

6.10 แอตทริบิวต์ unbreak_bt: JButton

เป็นปุ่มที่ใช้ในการเคลียร์เบรคพอยต์ที่เซ็ตไว้ โดยผู้ใช้จะต้องคลิ๊กไปที่บรรทัดที่ต้องการเคลียร์เบรคพอยต์ก่อน แล้วจึงกดปุ่มนี้ โดยแอคชั่นลิสเทนเนอร์ของมันจะนำค่า dot และ carerCoords มาเปลี่ยนตัวอักษรในบรรทัดให้เป็นสีดำดังเดิม

โดยจะทำส่งค่าตัวแปร t ที่มีชนิดเป็น StringTokenizer เข้าไปใน tty.executeCommand (t); โดยกำหนดค่า t ดังนี้ t = new StringTokenizer("clear "+currJavName+":"+lineAct);

6.11 แอตทริบิวต์ run_bt: JButton

เมื่อกดปุ่มนี้ โปรแกรมจะทำงานจนมาถึงเบรคพอยต์ที่ใกล้ที่สุด และจะเปลี่ยนตัวอักษรในบรรทัดดังกล่าวให้เป็นสีน้ำเงินเพื่อแสดงว่ามีการได้รันมาถึงตรงนี้แล้ว มีการอัพเดตค่าของเธรด, ตัวแปรโลคอล และ ตัวแปรที่แอดวอชไว้ ในหน้าต่าง ThreadWin, AWatchWin , addWatchWin ตามลำดับ

นอกจากนี้ยังมีการส่งค่าตัวแปร t ที่มีชนิดเป็น StringTokenizer เข้าไปใน tty.executeCommand(t); โดยกำหนดค่า t ดังนี้ t = new StringTokenizer("cont");

6.12 แอตทริบิวต์ step_bt: JButton

เมื่อกดปุ่มนี้ โปรแกรมจะทำงานทีละสเตทเมนต์ (statement) และจะเปลี่ยนตัวอักษรในบรรทัดดังกล่าวให้เป็นสีน้ำเงินเพื่อแสดงว่ามีการได้รันมาถึงตรงนี้แล้ว มีการอัพเดตค่าของเธรด, ตัวแปรโลคอล และ ตัวแปรที่แอดวอชไว้ ในหน้าต่าง ThreadWin, AWatchWin , addWatchWin ตามลำดับ

นอกจากนี้ยังมีการส่งค่าตัวแปร t ที่มีชนิดเป็น StringTokenizer เข้าไปใน tty.executeCommand(t); โดยกำหนดค่า t ดังนี้

```
t = new StringTokenizer("step");
tty.executeCommand(t);
t = new StringTokenizer("where all");
tty.executeCommand(t);
t = new StringTokenizer("locals");
tty.executeCommand(t);
t = new StringTokenizer("list");
tty.executeCommand(t);
```

เพื่อที่จะทำงานของคำสั่งตามลำดับ ดังต่อไปนี้

1. คำสั่ง step เพื่อทำการเอ็กซีคิวท์อีกหนึ่งบรรทัด
2. คำสั่ง where all เพื่อทำการแสดงเธรดปัจจุบันไว้ในหน้าต่าง ThreadWin
3. คำสั่ง localsเพื่อทำการค่าตัวแปรโลคอลไว้ในหน้าต่าง AWatchWin
4. คำสั่ง list เพื่อที่จะรับค่าบรรทัดปัจจุบันที่กำลังถูกเอ็กซีคิวท์

6.13 แอตทริบิวต์ next_bt: JButton

เมื่อกดปุ่มนี้ โปรแกรมจะทำงานทีละสเตทเมนต์ (statement) แต่จะไม่เข้าไปลึกเมื่อมีการเรียกเมทธอด และจะเปลี่ยนตัวอักษรในบรรทัดดังกล่าวให้เป็นสีน้ำเงินเพื่อแสดงว่ามีการได้รันมาถึงตรงนี้แล้ว มีการอัพเดตค่าของเธรด, ตัวแปรโลคอล และ ตัวแปรที่แอดวอชไว้ ในหน้าต่าง ThreadWin, AWatchWin , addWatchWin ตามลำดับ

นอกจากนี้ยังมีการส่งค่าตัวแปร t ที่มีชนิดเป็น StringTokenizer เข้าไปใน tty.executeCommand(t); โดยกำหนดค่า t ดังนี้

```
t = new StringTokenizer("next");
tty.executeCommand(t);
t = new StringTokenizer("where all");
tty.executeCommand(t);
t = new StringTokenizer("locals");
tty.executeCommand(t);
t = new StringTokenizer("list");
tty.executeCommand(t);
```

เพื่อที่จะทำงานของคำสั่งตามลำดับ ดังต่อไปนี้

1. คำสั่ง step เพื่อทำการเอ็กซีคิวท์อีกหนึ่งบรรทัด
2. คำสั่ง where all เพื่อทำการแสดงเธรดปัจจุบันไว้ในหน้าต่าง ThreadWin
3. คำสั่ง localsเพื่อทำการค่าตัวแปรโลคอลไว้ในหน้าต่าง AWatchWin
4. คำสั่ง list เพื่อที่จะรับค่าบรรทัดปัจจุบันที่กำลังถูกเอ็กซีคิวท์

6.14 แอตทริบิวต์ addWatch_bt: JButton

เมื่อกดปุ่มนี้จะมีไดอะล็อกขึ้นมาเพื่อให้ผู้ใช้ป้อนชื่อของตัวแปรที่ต้องการดูค่า โดยจะทำการส่งค่าตัวแปร t ที่มีชนิดเป็น StringTokenizer เข้าไปใน tty.executeCommand(t); โดยกำหนดค่า t ดังนี้ t = new StringTokenizer("print"+" "+input);

6.15 แอตทริบิวต์ exit_bt: JButton

เมื่อมีการกดปุ่มนี้จะเป็นการปิดการทำงานของดีบักเกอร์โดยไป เคลียร์ตัวอักษรในทุกๆ หน้าต่าง และเคลียร์เบรคพอยต์ที่เซ็ตไว้ทั้งหมด โดยจะทำการส่งค่าตัวแปร t ที่มีชนิดเป็น StringTokenizer เข้าไปใน tty.executeCommand(t); โดยกำหนดค่า t ดังนี้ t = new StringTokenizer("quit");

6.16 แอตทริบิวต์ ThreadWin: JTextArea

เป็นหน้าต่างที่ใช้แสดงเธรดที่กำลังรันอยู่ในขณะนั้น

6.17 แอตทริบิวต์ AWatchWin: JTextArea

เป็นหน้าต่างที่ใช้แสดงค่าตัวแปรโลคอล

6.18 แอตทริบิวต์ addWatchWin: JTextArea

เป็นหน้าต่างที่ใช้แสดงค่าตัวแปรที่ผู้ใช้ได้แอดวอชไว้

6.19 แอตทริบิวต์ ThreadWinPane: JScrollPane

เป็นคอนเทนเนอร์ชนิด JScrollPane ของ ThreadWin ซึ่งจะมีสครอลบาร์ออกมาในกรณีที่พื้นที่ของ ThreadWin ใหญ่กว่าพื้นที่แสดงผลจริง

6.20 แอตทริบิวต์ AWatchWinPane: JScrollPane

เป็นคอนเทนเนอร์ชนิด JScrollPane ของ AWatchWin ซึ่งจะมีสครอลบาร์ออกมาในกรณีที่พื้นที่ของ AWatchWin ใหญ่กว่าพื้นที่แสดงผลจริง

6.21 แอตทริบิวต์ addWatchWinPane: JScrollPane

เป็นคอนเทนเนอร์ชนิด JScrollPane ของ addWatchWin ซึ่งจะมีสครอลบาร์ออกมาในกรณีที่พื้นที่ของ addWatchWin ใหญ่กว่าพื้นที่แสดงผลจริง

6.22 แอตทริบิวต์ threadViewport: Jviewport

เป็นวิวพอร์ตที่บังคับให้สครอลบาร์ของ ThreadPane เลื่อนมาที่ตำแหน่งซ้ายบนเสมอ เพื่อให้ผู้ใช้สามารถดูค่าของเธรดได้ตั้งแต่เริ่มแรก เนื่องจากถ้าไม่ใช้วิวพอร์ตแล้วตำแหน่งของสครอลบาร์จะอยู่ที่ตำแหน่งสุดท้ายของข้อความเสมอ

6.23 แอตทริบิวต์ AWatchViewport: Jviewport

เป็นวิวพอร์ตที่บังคับให้สครอลบาร์ของ AWatchPane เลื่อนมาที่ตำแหน่งซ้ายบนเสมอ เพื่อให้ผู้ใช้สามารถดูค่าของตัวแปรโลคอลได้ตั้งแต่เริ่มแรก เนื่องจากถ้าไม่ใช้วิวพอร์ตแล้วตำแหน่งของสครอลบาร์จะอยู่ที่ตำแหน่งสุดท้ายของข้อความเสมอ

6.24 แอตทริบิวต์ addWatchViewport: Jviewport

เป็นวิวพอร์ตที่บังคับให้สครอลบาร์ของ addWatchWin เลื่อนมาที่ตำแหน่งซ้ายบนเสมอ เพื่อให้ผู้ใช้สามารถดูค่าของตัวแปรได้ตั้งแต่เริ่มแรก เนื่องจากถ้าไม่ใช้วิวพอร์ตแล้วตำแหน่งของสครอลบาร์จะอยู่ที่ตำแหน่งสุดท้ายของข้อความเสมอ

6.25 แอตทริบิวต์ viewportPos: Point

เป็นจุดที่ใช้กำหนดตำแหน่งบนซ้ายสุดของวิวพอร์ตทั้งสาม โดยในที่นี้จะมีค่าเป็น [0,0] เสมอ เนื่องจากต้องการให้ตำแหน่งของผลลัพธ์มีที่ส่วนเริ่มต้นเสมอ

6.26 แอตทริบิวต์ editWin: EditWin

จะถูกเซ็น.ให้เป็นค่าตัวแทนของออบเจ็คต์ของคลาส EditWin ที่เป็นตัวแรกการทำงานของ debugPane นี้ เพื่อที่จะสามารถอ้างอิงค่าต่างๆ และเปลี่ยนสีตัวอักษรให้กับ textPane ใน EditWin ได้

6.27 แอตทริบิวต์ caretCoords: Rectangle

ใช้เก็บค่าโคออดิเนต (coordinate) ของตำแหน่งที่เคเรต (caret) อยู่ เพื่อนำไปใช้ในการเซ็ตเบรคพอยต์เมื่อเข้าสู่โหมดการดีบัก โดยผู้ใช้จะต้องคลิ๊กเมาส์ไปตรงที่บรรทัดของโปรแกรมที่ต้องการเซ็ตเบรคพอยต์ แล้วกดปุ่ม break ใน debugPane ก็จะเป็นการเซ็ตเบรคพอยต์

กลไกการทำงานคือ อินเนอร์คลาส MyCaretListener ของคลาส EditWin จะถูกเซ็ตเป็นเคเรตลิสเทนเนอร์ของ textPane ผ่านคำสั่ง textPane.addCaretListener(new MyCaretListener()) ซึ่งอินเนอร์คลาสนี้จะเซ็ตค่า caretCoord ทุกครั้งเมื่อมีการคลิกเมาส์ไปบน textPane

แอตทริบิวต์นี้ต้องมีการเข้าถึงเป็นโพรเท็กเท็ด เนื่องจากต้องมีการเซ็ตค่าจากคลาส EditWin

6.28 แอตทริบิวต์ dot: int

เป็นค่าตำแหน่งของตัวอักษรที่ผู้ใช้คลิ๊กเคเรตไว้ถูกเซ็ตโดยอินเนอร์คลาส MyCaretListener ของคลาส EditWin จากค่านี้เราสามารถนำไปเปลี่ยนสีให้กับตัวอักษรในบรรทัดนั้นทับบรรทัดได้ โดยผ่านเมทธอด setHighLight()

6.29 แอตทริบิวต์ lineAct: int

เป็นบรรทัดที่ถูกเซ็ตเบรคพอยต์ ตามปกติแล้วเราน่าจะสามารถหาค่านี้ได้ผ่าน caretCoords.y แต่จากการใช้งานจริงปรากฏว่าค่าดังกล่าวไม่ตรงกับค่าบรรทัดที่แท้จริง ดังนั้นเราต้องมีการปรับค่านี้ให้ตรงกับค่าจริงแล้วใส่ค่านี้ให้กับตัวแปร lineAct โดยสูตรในการปรับค่าคือ lineAct = (caretCoord.y +13) / 16

6.30 แอตทริบิวต์ oldj:int = 60000

เป็นค่าตำแหน่งบรรทัดล่าสุดที่โปรแกรมทำงานไปถึงแล้วได้เปลี่ยนสีตัวอักษรในบรรทัดนั้นให้เป็นสีน้ำเงิน เก็บไว้เพื่อที่จะไปเซ็ตสีให้เป็นสีดำเหมือนเดิม เมื่อมีการปิดการใช้งานดีบักเกอร์

6.31 แอตทริบิวต์ line2Pixel: int = 16

เป็นค่าจำนวนพิคเซลต่อความกว้างหนึ่งบรรทัดของหน้าจอ textPane ซึ่งจะต้องใช้ในการแปลงค่าบรรทัดที่โปรแกรมกำลังรันอยู่ไปเป็นตำแหน่งพิกเซลเพื่อนำไปเซตค่า dbv_point ในคลาส EditWin

6.32 แอตทริบิวต์ bp_hist[]: int

เป็นอะเรย์ที่เก็บค่าบรรทัดของเบรคพอยต์ที่ผู้ใช้ได้เซ็ตไว้ เก็บไว้เพื่อที่จะไปเซ็ตสีตัวอักษรจากสีแดงให้เป็นสีดำเหมือนเดิม เมื่อมีการปิดการใช้งานดีบักเกอร์

6.33 แอตทริบิวต์ i: int

เป็นอินเด็กซ์ของอะเรย์ bphist[]

6.34 คอนสตรัคเตอร์ DebugPane(final EditWin eWin)

เป็นผู้สร้างยูสเซอร์อินเตอร์เฟสของ debugPane แต่จะถูกซ่อนไว้ก่อน

6.35 เมทธอด setHighLight(): void

ทำหน้าที่เซ็ตค่าของตัวอักษรให้เป็นสีน้ำเงินจะถูกเรียกจากแอคชั่นลิสเทนเนอร์ step_bt, next_bt และ run_bt

6.36 เมทธอด startIt(): void

เป็นเมทธอดสร้างเธรดที่ชื่อว่า debugger และทำการเริ่มต้นการทำงานของดีบักเกอร์โดยคำสั่ง debugger.start() ซึ่งจะไปเรียกใช้งานเมทธอด run()

6.37 เมทธอด stopIt(): void

จะเป็นการปิดการทำงานของดีบักเกอร์โดยปิดการทำงานของเธรด debugger, แล้วซ่อนส่วนของ DebugPanc ดังเดิม

6.38 เมทธอด run(): void

เป็นเมทธอดที่สร้างจาวาเวอร์ชวลแมชชีนของดีบักเกอร์ และกำหนดการเชื่อมต่อระหว่างดีบักกี้และแอพพลิเคชั่นของดีบักเกอร์ ให้เป็นแบบ launching connector โดยส่งค่าคอนเนคเตอร์อาร์กิวเมนท์ให้กับ Env.init(connectSpec,true,VirtualMachine.TRACE_NONE) และทำการเซตค่า ของซอร์สพาธให้เป็นค่าพาธของไฟล์ที่กำลังแก้ไขอยู่ในปัจจุบัน โดยส่งค่า currPath เข้าไปในEnv.setSourcePath(currPath) และจะแสดงยูสเซอร์อินเตอร์เฟสของ debugPanc ซึ่งถูกซ่อนไว้ก่อนโดยคอนสตรัคเตอร์

2.7 คลาส TTY

เป็นคลาสที่ทำการรับอาร์กิวเมนท์ (argument) ที่ส่งมาทางคอมมานด์ไลน์ แล้วส่งให้กับ คลาส Env เพื่อทำการสร้างการเชื่อมต่อระหว่างดีบักกี้และแอพพลิเคชั่นของดีบักเกอร์ และ ทำหน้าที่ในการรับคำสั่งโต้ตอบกับผู้ใช้งานแล้วนำมาทำงานกับเวอร์ชวลแมชชีนของดีบักเกอร์

7.1 แอตทริบิวต์ AWatchWin: JTextArea

เป็นหน้าต่างที่ใช้แสดงค่าตัวแปรโลคอล

7.2 แอตทริบิวต์ AThreadWin: JTextArea

เป็นหน้าต่างที่ใช้แสดงเธรดที่กำลังรันอยู่ในขณะนั้น

7.3 เมทธอด executeCommand(StringTokenizer t)

ส่วนที่รับคำสั่งของการทำงาน จากค่า t ที่ส่งเข้ามาคือค่าคำสั่งที่รับมาจากแอคชั่นลิสเทนเนอร์ของแต่ละปุ่มคำสั่งที่อยู่ในคลาส DebugPane เช่น run_bt, Step_bt เป็นต้น ฟังก์ชันนี้จะไปเรียกให้คลาส Commands ทำงานอีกทีหนึ่ง

7.4 เมทธอด help()

เป็นส่วนของการแสดงวิธีการใช้งานโปรแกรมเจดีบีซึ่งจะมีการเรียกใช้เมื่อมีการพิมพ์คำสั่ง help ที่คอมมานไลน์

7.5 คอนสตรัคเตอร์ TTY(PrintStream out,EditWin eWin)

ทำหน้าที่ในการรับคำสั่งแล้วมาทำงานตามคำสั่งโดยเรียกใช้จาวาเวอร์ชวลแมชชีนของดีบักเกอร์ ในส่วนนี้จะทำการเพิ่มเติมโดยจะรับ eWin เข้ามาด้วยเพื่อที่จะสามารถใช้ในการแสดงผลต่างๆ ที่หน้าจอกราฟฟิคได้โดยตรง และในคอนสตรัคเตอร์นี้จะมีการเรียกใช้เมทธอด

executeCommand(StringTokenizer t) เพื่อทำงานตามคำสั่งที่รับเข้าตามค่า t เดิมจะรับคำสั่งจากคอมมานไลน์แต่จะแก้ไขโดยไม่ทำการรับค่าจากคอมมานไลน์แต่ว่าจะส่งค่าผ่าน tty.executeCommand(t) ที่อยู่ในคลาส DebugPane โดยส่งคำสั่งมาทางตัวแปร t


*รูปที่ 5-8 คลาส TTY*

**8 คลาส Commands**

ทำหน้าที่ในการพิมพ์ผลลัพธ์การดีบักออกทางหน้าจอ ตามคำสั่งที่เรียกเข้ามาโดยจะมีเมทธอดที่ทำการรับคำสั่งแต่ละคำสั่งในคลาสนี้ด้วย

8.1 แอตทริบิวต์ sourcePath : String = "."

ค่าของซอร์สพาธเพื่อที่บอกพาธของซอร์สโค้ดซึ่งกำหนดให้เป็นคลาสปัจจุบัน

8.2 เมทธอด commandCont()

ทำคำสั่ง cont ที่เรียกมาจาก tty.executeCommand(t) เพื่อเอ็กซีคิวต์ต่อไปจนถึงเบรคพ้อยต์

8.3 เมทธอด commandList(StringTokenizer t)

ทำคำสั่ง list ที่เรียกมาจาก tty.executeCommand(t) เพื่อพิมพ์ซอร์สโค้ด

8.4 เมทธอด commandLocals()

ทำคำสั่ง locals ที่เรียกมาจาก tty.executeCommand(t) เพื่อพิมพ์ตัวแปรโลคอลที่อยู่ในแสตกเฟรมปัจจุบัน(current stack frame)

8.5 เมทธอด Next()

ทำคำสั่ง next ที่เรียกมาจาก tty.executeCommand(t) เพื่อเอ็กซีคิวท์โดยไม่เข้าไปในส่วนของเมธอดที่เรียกใช้งาน

8.6 เมทธอด Print(final StringTokenizer t , final boolean dumpObject)

ทำคำสั่ง print ที่เรียกมาจาก tty.executeCommand(t) เพื่อแสดงค่าตัวแปรที่ต้องการ

8.7 เมทธอด run(StringTokenizer t)

ทำคำสั่ง run ที่เรียกมาจาก tty.executeCommand(t) เพื่อทำการเริ่มต้นการเอ็กซีคิวต์คลาสหลักของโปรแกรม

8.8 คอนสตรัคเตอร์ Commands(PrintStream out , EditWin eWin)

เป็นเมทธอดเริ่มต้นการทำงานของคลาสนี้

8.9 เมทธอด commandStep(StringTokenizer t)

ทำคำสั่ง step ที่เรียกมาจาก tty.executeCommand(t) เพื่อเอ็กซีคิวต์บรรทัดปัจจุบัน

8.10 เมทธอด commandStepi()

ทำคำสั่ง stepi ที่เรียกมาจาก tty.executeCommand(t) เพื่อเอ็กซีคิวต์คำสั่งปัจจุบัน

8.11 เมทธอด commandstop(StringTokenizer t)

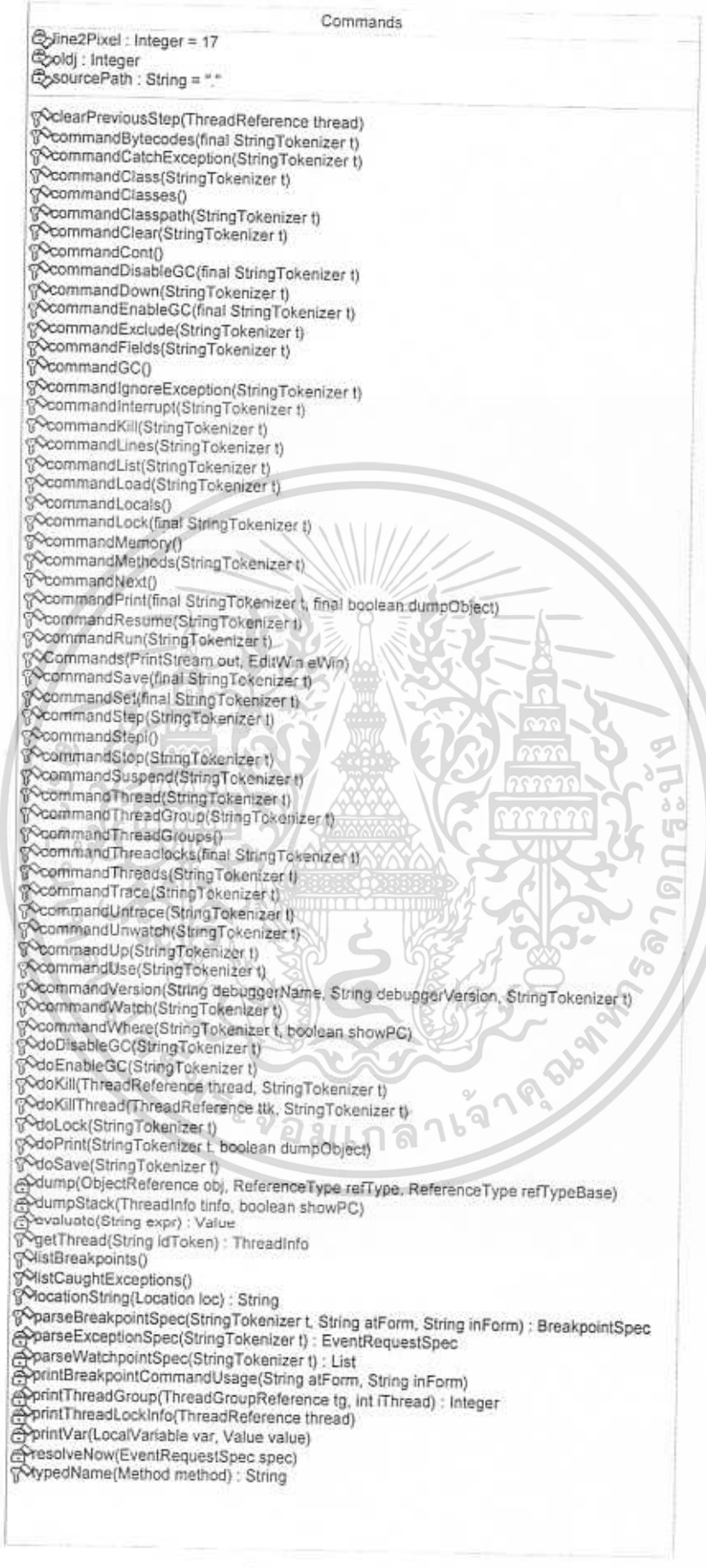
ทำคำสั่ง stop ที่เรียกมาจาก tty.executeCommand(t) เพื่อเซตเบรคพ้อยท์(breakpoint)

8.12 เมทธอด commandThread(StringTokenizer t)

ทำคำสั่ง thread ที่เรียกมาจาก tty.executeCommand(t) เพื่อเซต(set)เธรดปัจจุบัน

8.13 เมทธอด commandThreadGroup(StringTokenizer t)

ทำคำสั่ง threadgroup ที่เรียกมาจาก tty.executeCommand(t) เพื่อเซตกลุ่มของเธรดปัจจุบัน




*รูปที่ 5-9 คลาส Commands*

8.14 เมทธอด commandThreadGroups(StringTokenizer t)

ทำคำสั่ง threadgroups ที่เรียกมาจาก tty.executeCommand(t) เพื่อแสดงกลุ่มของเธรดที่มีอยู่ทั้งหมด

8.15 เมทธอด commandThreads(StringTokenizer t)

ทำคำสั่ง threads ที่เรียกมาจาก tty.executeCommand(t) เพื่อแสดงเธรดทั้งหมดที่มี

8.16 เมทธอด commandUse(StringTokenizer t)

ทำคำสั่ง use ที่เรียกมาจาก tty.executeCommand(t) เพื่อแสดงผลหรือเปลี่ยนซอร์สพาธ

8.17 เมทธอด commandWhere(StringTokenizer t, boolean showPC)

ทำคำสั่ง where ที่เรียกมาจาก tty.executeCommand(t) เพื่อแสดงแสตก(stack) ของเธรด

**9 คลาส Env**

จะเป็นคลาสที่ทำหน้าที่ในการสร้างการเชื่อมต่อระหว่างดีบักกี้และแอพพลิเคชั่นของดีบักเกอร์ และกำหนดสภาวะแวดล้อมต่างให้กับดีบักเกอร์ ซึ่งในการเซตค่าต่าง ๆ ตามส่งค่าตัวแปรเข้ามานั้น วิธีการจะมีการเรียกใช้งานคลาสต่าง ๆ อีกมากเช่น VMConnection ในคลาสนี้เราสามารถเรียกใช้งานได้โดยไม่ต้องทำการแก้ไข

9.1 แอตทริบิวต์ sourceMapper : SourceMapper = new SourceMapper(".")

ค่าของซอร์สพาธเพื่อที่บอกพาธของซอร์สโค้ดซึ่งกำหนดให้เป็นคลาสปัจจุบัน

9.2 เมทธอด init(String connectSpec, boolean openNow, int flags)

ทำการสร้างการเชื่อมต่อระหว่างดีบักกี้และแอพพลิเคชั่นของดีบักเกอร์โดยจะทำการเรียกใช้คลาส VMConnection(connectSpec, flags) โดยจะเรียกเมทธอด open()


*รูปที่ 5-10 คลาส Env*

# บทที่ 6

# การทดสอบและผลการทดลองการใช้งานสภาพแวดล้อมการพัฒนาโปรแกรมแบบกราฟิคยูสเซอร์อินเตอร์เฟส

## 6.1 ขั้นตอนการดำเนินการดำเนินงาน

- รวบรวมความต้องการต่างๆ ของคุณสมบัติที่ควรจะมีในสภาพแวดล้อมการพัฒนาโปรแกรมในภาษาจาวา
- ทำการติดตั้งชุดพัฒนาโปรแกรมภาษาจาวา ( Java Software Development Kit) ซึ่งสามารถดาวน์โหลดได้จากเวบไซต์ของภาษาจาวาของบริษัทซันไมโครซิสเต็มส์ (www.java.sun.com) ซึ่งเวอร์ชั่นที่จะสนับสนุนเจพีดีเอจะต้องเป็นเวอร์ชั่น 1.2 ขึ้นไป
- ทำการออกแบบและสร้างสภาพแวดล้อมการพัฒนาโปรแกรมในภาษาจาวาตามความต้องการที่รวบรวมไว้ข้างต้น
- ทดสอบระบบทั้งหมดเพื่อหาข้อบกพร่องและทำการปรับปรุง

## 7.2 การติดตั้งโปรแกรม KMITL Java Developer version 1.0

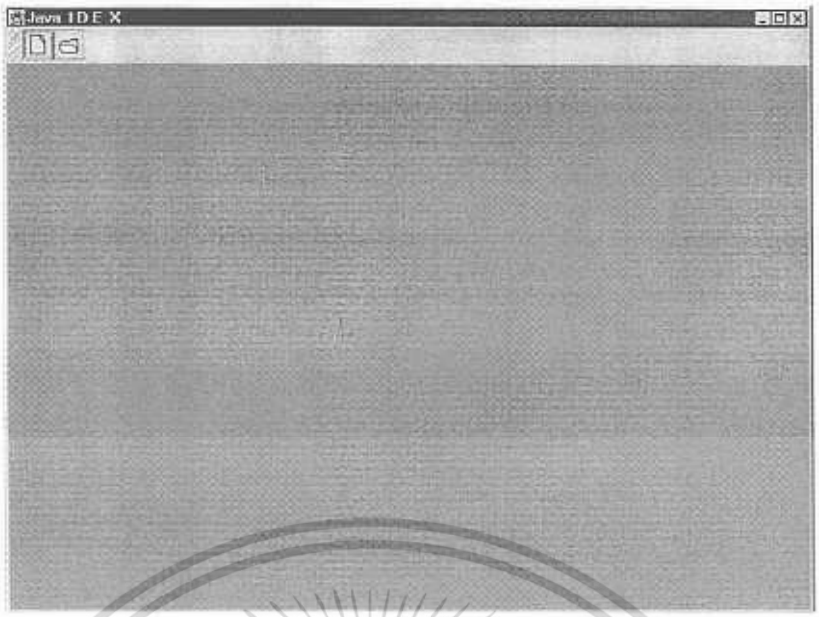
สภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวาแบบกราฟฟิคยูสเซอร์อินเตอร์เฟสที่สร้างขึ้นมีชื่อว่า KMITL Java Developer version 1.0 โดยจะรวมกันไว้ในแพคเก็จที่ชื่อว่า KMITL Java Developer version 1.0 ในการติดตั้งโปรแกรมดังกล่าวจะต้องทำตามขั้นตอนดังต่อไปนี้

- ติดตั้งชุดพัฒนาโปรแกรมภาษาจาวาของบริษัทซันไมโครซิสเต็มส์ (Sun's Java Development Kit) ในเวอร์ชั่นที่สูงกว่าเวอร์ชั่น 1.2 เพื่อให้สนับสนุน JPDA โดยสามารถดาวน์โหลดได้จาก www.java.sun.com โดยในที่นี้ สมมติว่าติดตั้งในไดเร็คทอรี่ c:\jdk1.2.2
- ติดตั้งสภาพแวดล้อมการพัฒนาโปรแกรมภาษาจาวาแบบกราฟฟิคยูสเซอร์อินเตอร์เฟส KMITL Java Developer version 1.0 ไว้ในไดเรกทอรี่ (ในที่นี้) ที่ชื่อว่า c:\KMITLJavaDev1.0
- เซตพาธดังนี้

  SET PATH=%PATH%;c:\jdk1.2.2\bin
- เซตคลาสพาธดังนี้

  SETCLASSPATH=.;%CLASSPATH%;c:\ KMITLJavaDev1.0\lib\jpda.jar
- ที่คอมมานด์ไลน์ในรูทไดเร็คทอรี่ไดร์ฟซี รันโปรแกรมโดยเรียกคำสั่ง

  ava JavaIDEX.JavaIDEX จะขึ้นโปรแกรมดังรูปที่ 6-1



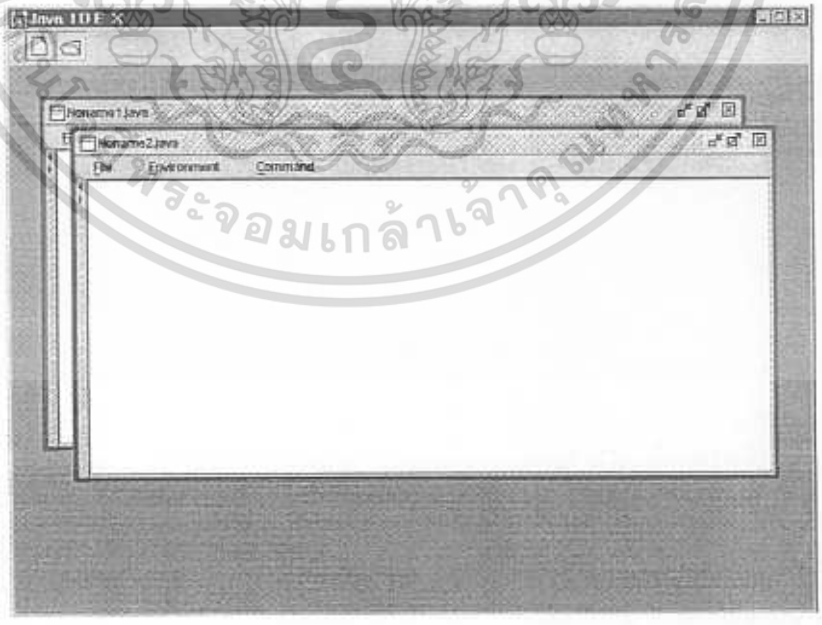

*รูปที่ 6-1 ส่วนเริ่มต้นของโปรแกรม*

7.3 การทดสอบและใช้งานโปรแกรม

จะทดสอบโปรแกรมตามการจำลองทำงานจริง (scenario) ดังในภาคผนวก ถ.

**1. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น new**

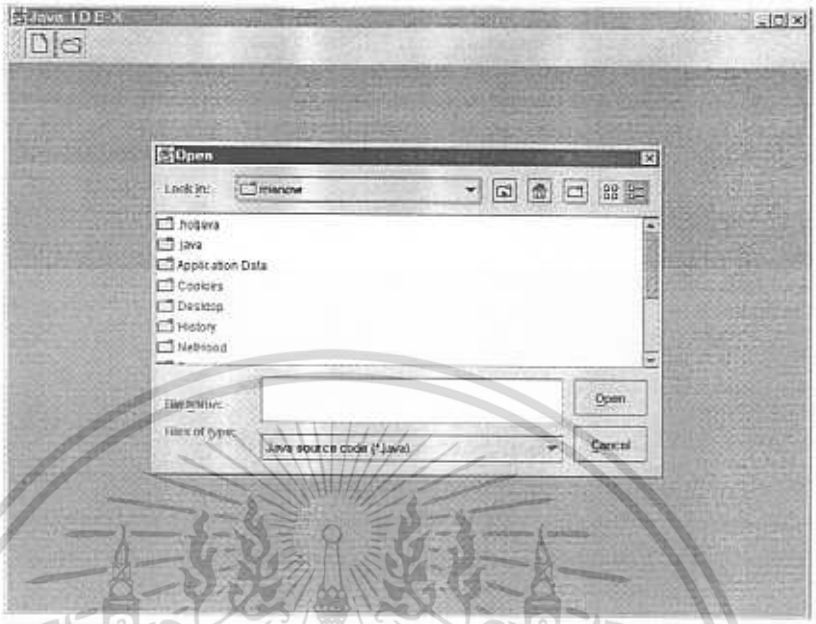
1.1 กดปุ่ม new (ซึ่งเป็นรูปกระดาษ) สองครั้งจะเกิดหน้าต่างให้สร้างและแก้ไขโปรแกรมขึ้นมา 2 หน้าต่างมีชื่อว่า Noname1.java และ Noname2.java ตามลำดับ




*รูปที่ 6-2 ผลการทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น new*

**2. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น open**

2.1 กดปุ่ม open (ซึ่งเป็นรูปไดเรคทอรี่) จะขึ้นไดอะลอกให้มาเลือกไฟล์

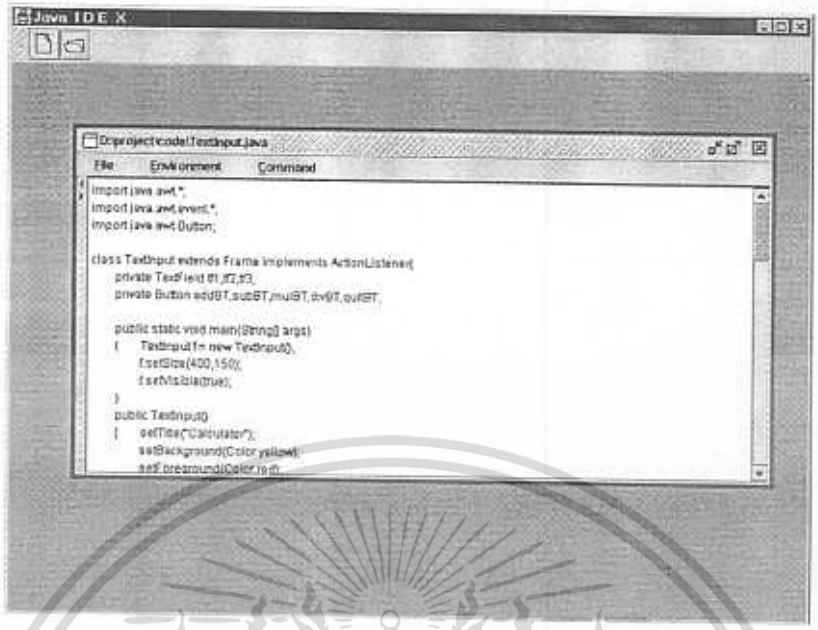



*รูปที่ 6-3 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น open – กดปุ่ม open*

2.2 เลือกไฟล์ TextInput.java


*รูปที่ 6-4 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น open – เลือกไฟล์*

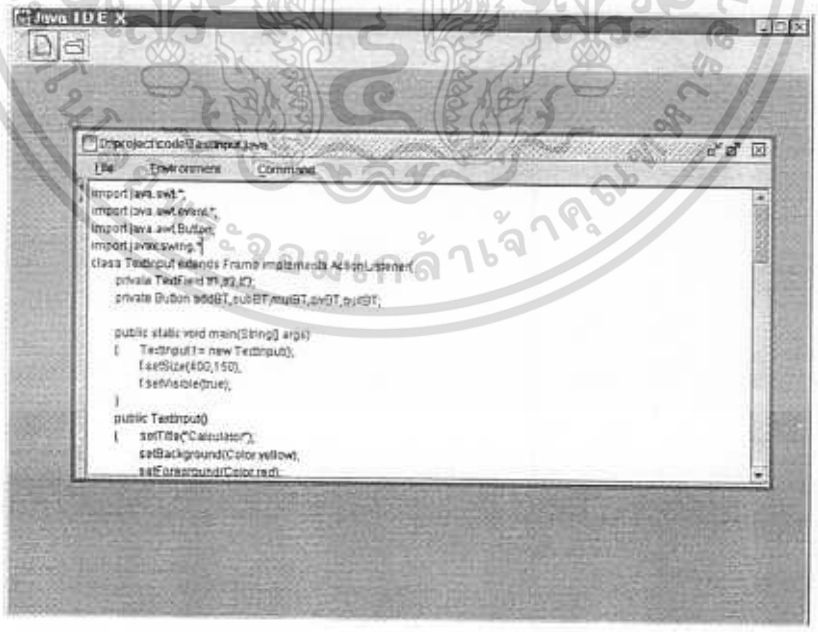
2.3 ไฟล์ TextInput.java ถูกโหลดไว้ในหน้าต่างที่ใช้แก้ไขโปรแกรม




*รูปที่ 6-5 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น open – โหลดไฟล์เข้าไว้ในหน้าต่างที่ใช้ในการแก้ไขโปรแกรม*

3. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น edit

3.1 ทดลองพิมพ์คำว่า "import javax.swing.*" ในบรรทัดที่ 4



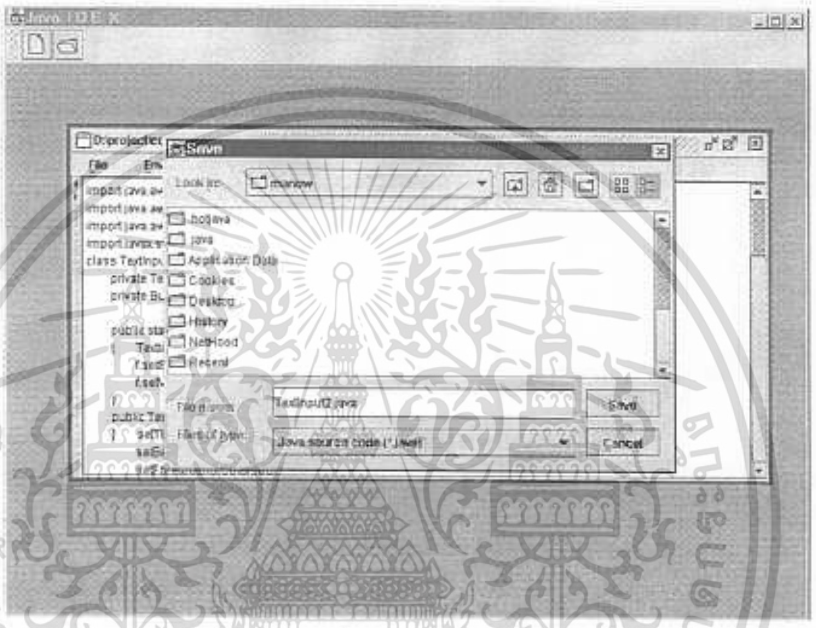

*รูปที่ 6-6 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น edit*

**4. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น save**

4.1 กดปุ่ม save ในเมนู file, ปิดหน้าต่าง EditWin แล้วเปิดขึ้นมาใหม่ ปรากฏว่าข้อมูลเหมือนกับที่ได้เปลี่ยนแปลงไป

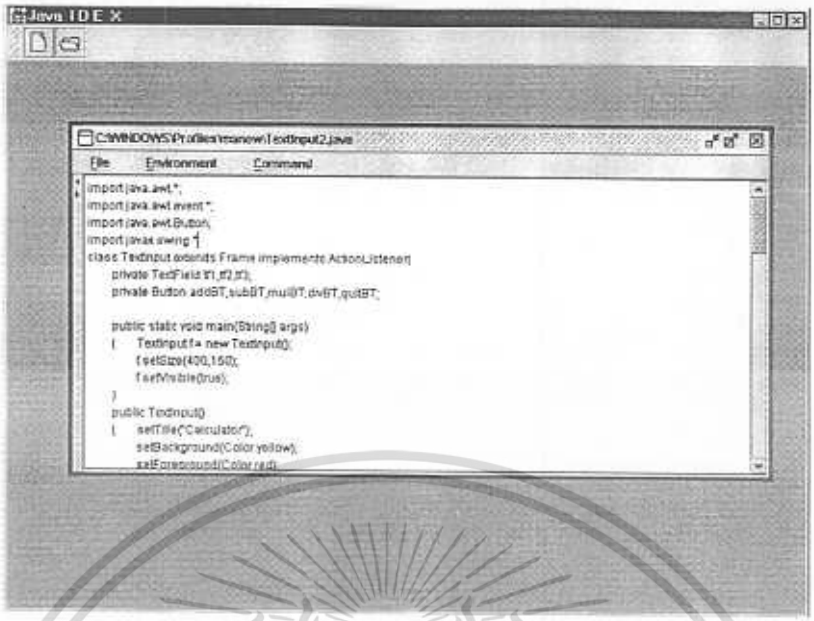
**5 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น save as**

5.1 เลือกคำสั่ง save as ในเมนู file แล้วจะปรากฏไดอะลอกไปเลือกไฟล์เป้าหมายที่ต้องการ บันทึก ในที่นี้ให้ชื่อไฟล์ว่า TextInput2.java




*รูปที่ 6-7 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น save as -เลือกไฟล์เป้าหมายที่ต้องการบันทึก*

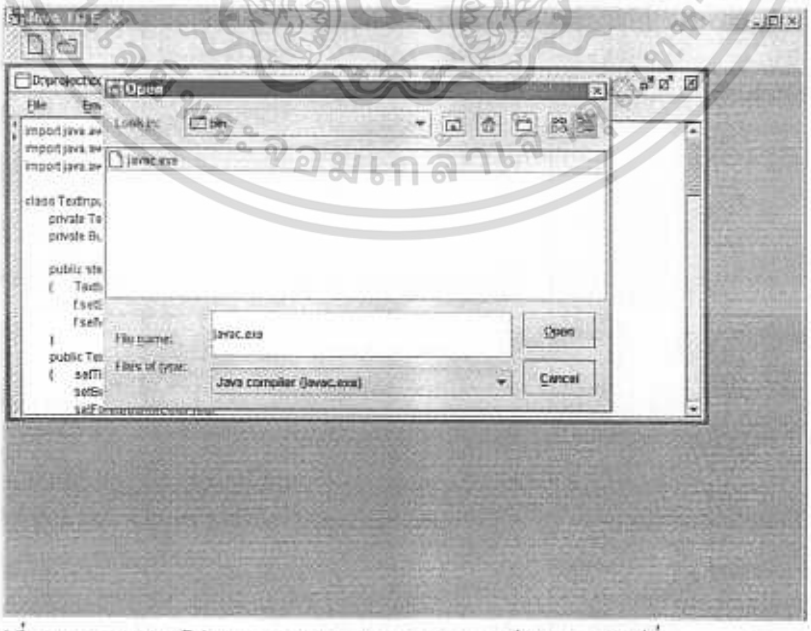
5.2 ชื่อไฟล์ที่กำลังแก้ไขเปลี่ยนแปลงเป็น TextInput2.java




*รูปที่ 6-8 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น save as -ชื่อไฟล์เปลี่ยนเป็น TextInput2.java*

**6. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น set environment**

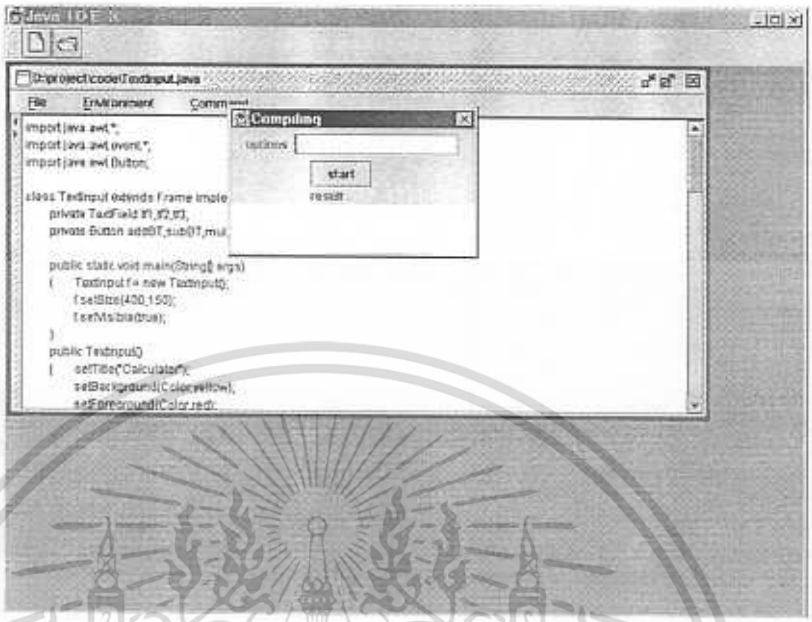
6.1 เมื่อเลือกคำสั่ง set environment ในเมนู environment จะขึ้นหน้าต่างมาให้เลือกตำแหน่งของโปรแกรม javac.exe (ทำให้ทราบตำแหน่งของชุดพัฒนาโปรแกรมภาษาจาวา (JDK) ได้) เพื่อใช้ในการคอมไพล์, เอ็กซิคิวต์และดีบัก




*รูปที่6-9 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น set environment*

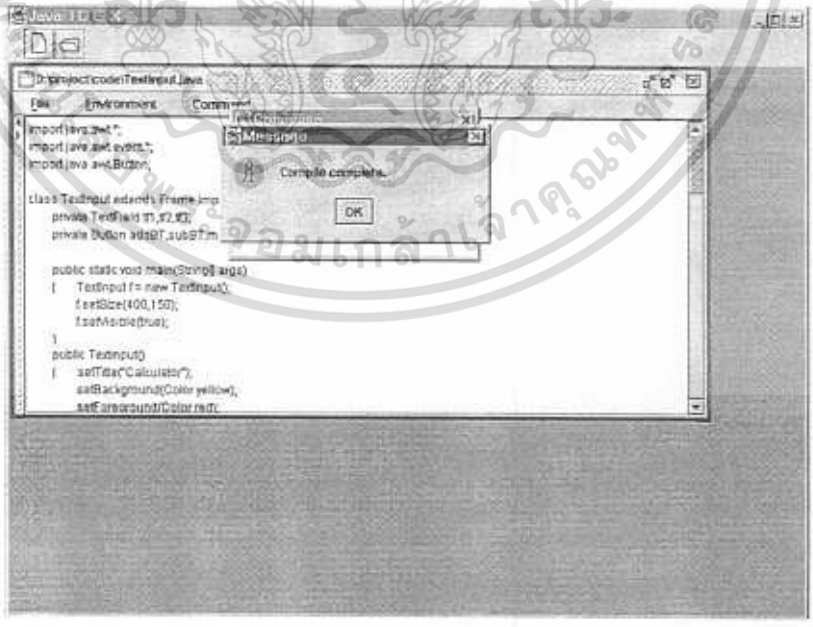
**7. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น compile**

7.1 เลือกคำสั่ง compile ในเมนู command จะปรากฏหน้าต่าง compile ให้ใส่ออปชั่นในการคอมไพล์



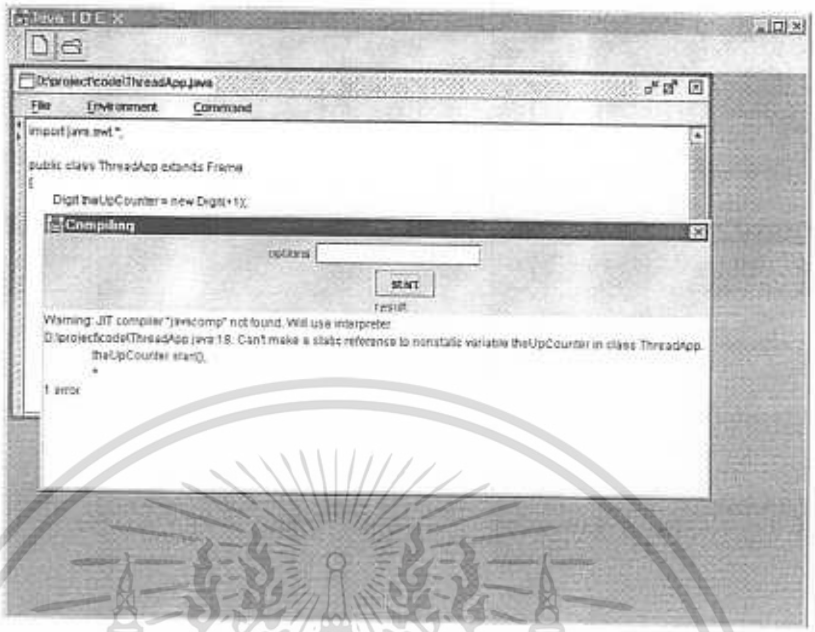

*รูปที่ 6-10 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น compile – หน้าต่างให้ใส่ออปชั่นในการคอมไพล์*

7.2 เมื่อคอมไพล์เสร็จ จะขึ้นหน้าต่างแสดงขึ้นว่าคอมไพล์เรียบร้อยแล้ว




*รูปที่ 6-11 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น compile- แสดงว่าคอมไพล์เสร็จเรียบร้อยแล้ว*

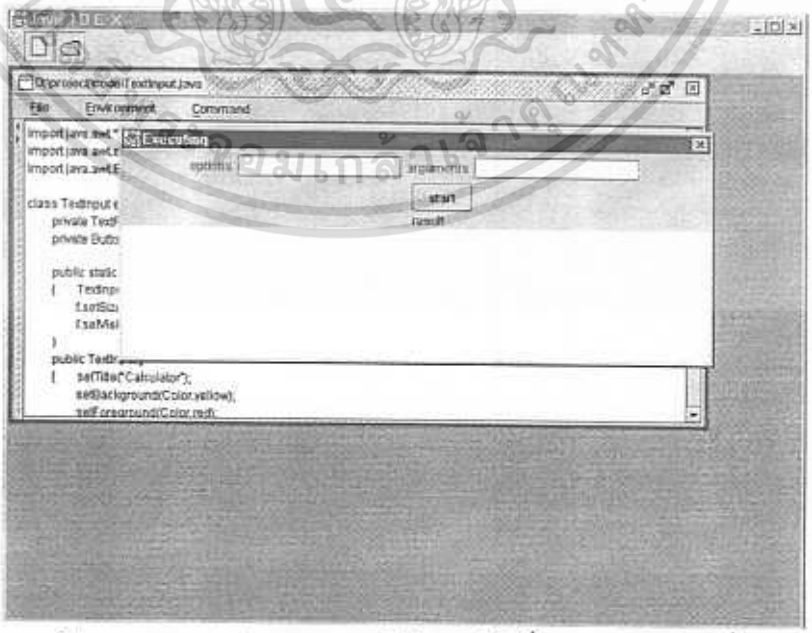
7.3 เมื่อกด OK แล้ว หากโปรแกรมไม่มีข้อผิดพลาด ที่หน้าต่าง results ก็จะไม่แสดงผลอะไร แต่หากโปรแกรมมีข้อผิดพลาดในการคอมไพล์ จะขึ้นข้อความแสดงการผิดพลาด




*รูปที่ 6-12 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น compile- แสดงผลของการคอมไพล์ที่ผิดพลาด*

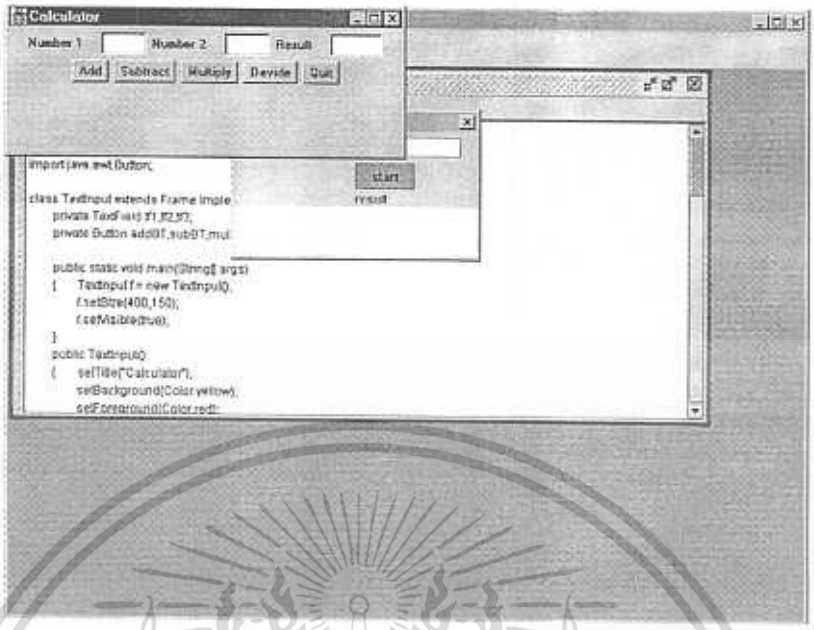
8. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น execute

8.1 เลือกคำสั่ง execute ในเมนู command จะปรากฏหน้าต่าง execute ให้ใส่ออปชั่นและอาร์กิวเมนต์ในการเอ็กซิคิวต์



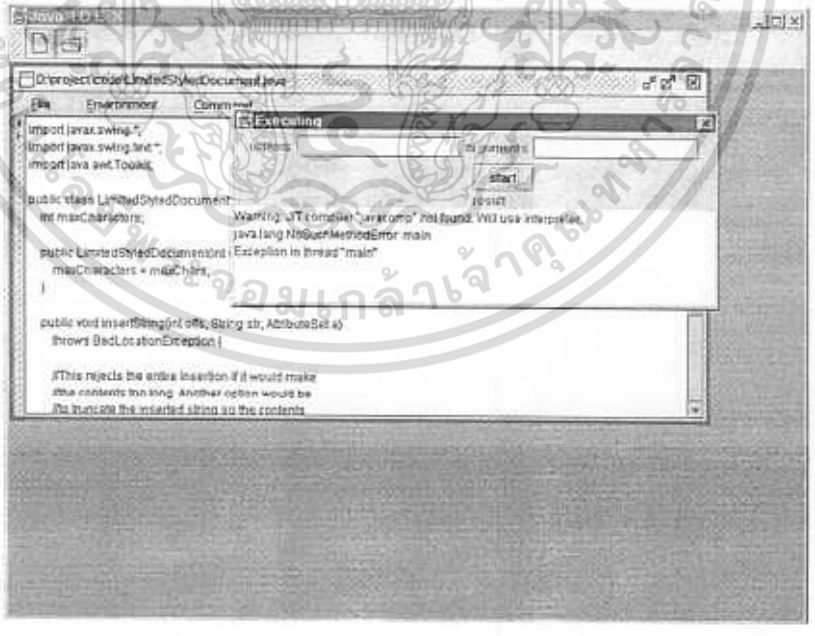

*รูปที่ 6-13 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น execute- ใส่ออปชั่นและอาร์กิวเมนต์ในหน้าต่างการเอ็กซิคิวต์*

8.2 หากไม่มีปัญหาแบบ runtime error โปรแกรมจะถูกเอ็กซิคิวต์ขึ้นมา




*รูปที่ 6-14 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น execute- กรณีไม่เกิด runtime error*

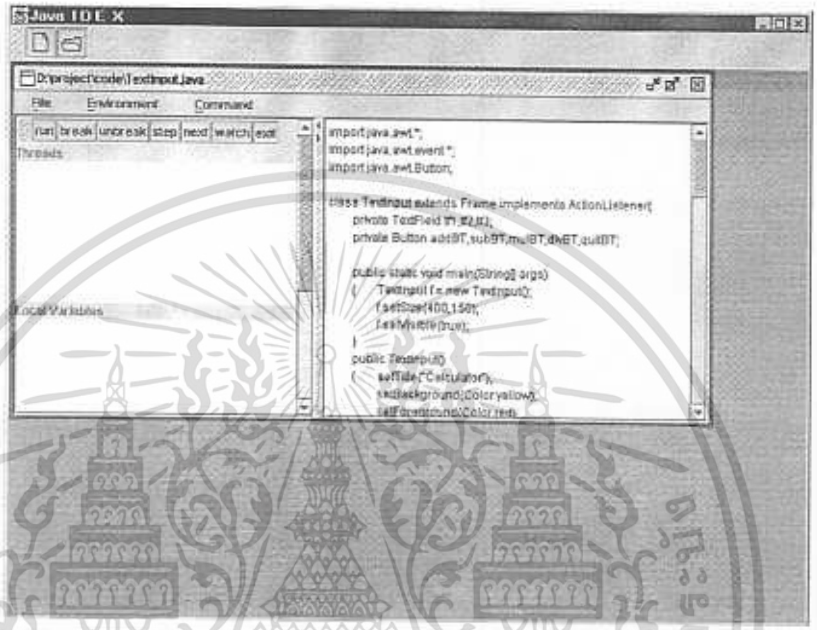
8.3 หากมีปัญหาแบบ runtime error ที่หน้าต่าง results จะแสดงข้อผิดพลาด เพื่อผู้เขียนโปรแกรมสามารถนำไปแก้ไขได้ต่อไป




*รูปที่ 6-15 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น execute- กรณีเกิด runtime error*

**9. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น debug**

9.1 เลือกคำสั่ง debug ในเมนู command จะปรากฏหน้าต่าง debug โดยมีคำสั่งต่าง ๆ ที่ใช้สำหรับการดีบักขึ้นมา ได้ step ใช้ในการรันโปรแกรมทีละคำสั่ง, next ใช้ในการรันโปรแกรมทีละคำสั่งเหมือนกัน step แต่จะไม่ลงลึกเข้าไปหากมีการเรียกใช้เมทธอด , break/unbreak ใช้ในการเซตและเคลียร์เบรคพอยต์ตามลำดับ, run ใช้ในการรันโปรแกรมไปถึงเบรคพอยต์ที่เซตไว้, watch ใช้ใส่ชื่อตัวแปรที่ต้องการมอนิเตอร์ดูค่า และ exit ใช้เพื่อออกจากโหมดการดีบัก



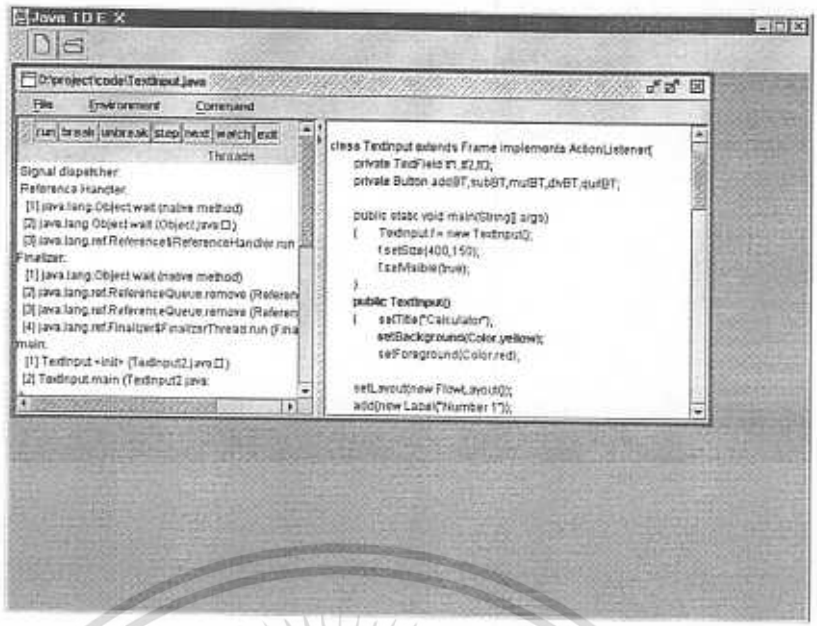

*รูปที่ 6-16 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่นการดีบัก*

**10. การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น break, step, next, unbreak, run, watch**

10.1 ถ้าต้องการเซตเบรคพ้อยต์ ให้เคอร์เซอร์อยู่ที่บรรทัดที่ต้องการแล้วกดปุ่ม break จะได้ผลดังรูปจะมี ตัวอักษรที่บรรทัดนั้นจะกลายเป็นสีแดงเพื่อแสดงการเซตเบรคพ้อยต์ และถ้าต้องการ unbreak ก็ทำเช่นเดียวกันแล้วผลที่ได้คือ ตัวอักษรที่บรรทัดนั้นจะกลายเป็นสีแดงกลายเป็นสีดำ

10.2 ถ้าต้องการ step หรือ next ให้ทำการกดปุ่ม แล้วอักษรที่บรรทัดปัจจุบันจะกลายเป็นสีน้ำเงินแสดงถึงว่าบรรทัดนั้นกำลังถูกเอ็กซิคิวต์อยู่

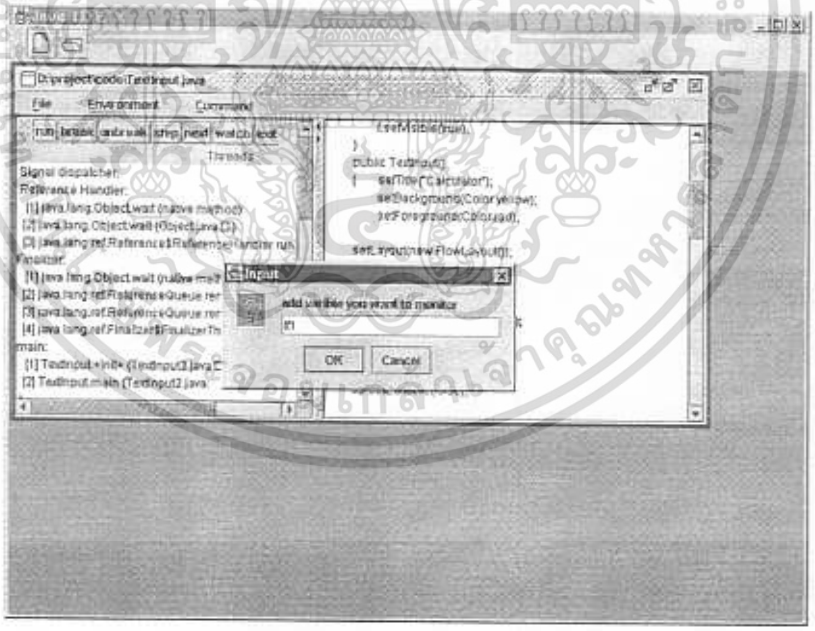
10.3 ถ้าต้องการ run ไปจนถึงเบรคพ้อยต์ ให้กดปุ่ม run แล้วจะทำการเอ็กซิคิวต์ไปจนกระทั่งเจอเบรคพ้อยต์ที่เซตไว้




*รูปที่ 6-17 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่นต่าง ๆ ในการดีบัก*

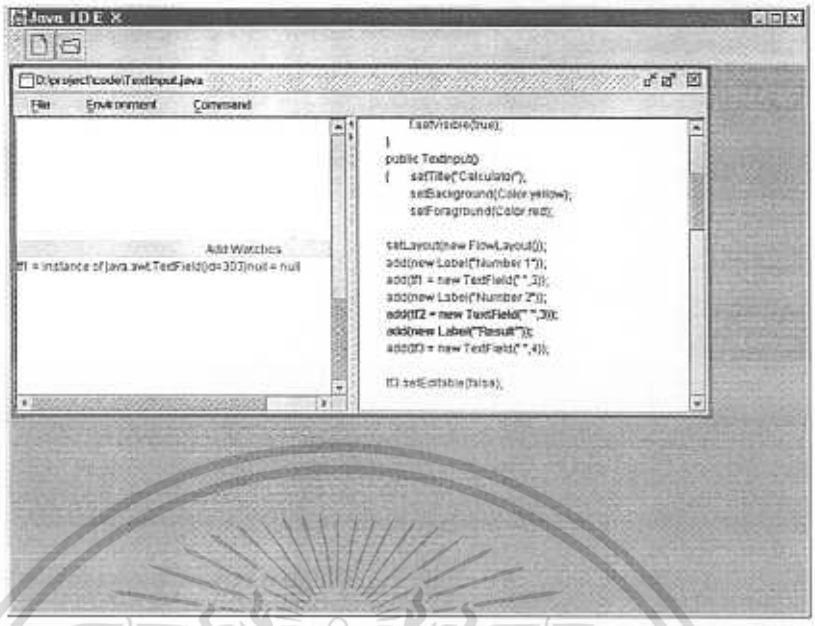
## 11 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น add watch

11.1 เมื่อกดปุ่ม watch จะขึ้นหน้าต่าง เพื่อรับชื่อตัวแปรที่ต้องการดูค่า ดังรูป




*รูปที่ 6-18 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น add watch ในการรับชื่อตัวแปร*

11.2 เมื่อรับชื่อตัวแปรแล้วจะทำการแสดงค่าในหน้าต่าง Add Watch




*รูปที่ 6-19 การทดสอบโปรแกรมตามการจำลองเหตุการณ์จริงของฟังก์ชั่น add watch*

# บทที่ 7
# บทสรุปและวิจารณ์

## 7.1 บทสรุปและวิจารณ์

ดีบักเกอร์ที่สร้างเป็นดีบักเกอร์ที่พัฒนามาจาก JDB ที่มาพร้อมกับชุดสถาปัตยกรรม JPDA โดยลักษณะเป็นกราฟฟิคยูสเซอร์อินเตอร์เฟส โดยมีฟังก์ชั่นการดีบักได้แก่ สเตป(step) ,เซตเบรคพ้อยท์ (set breakpoint),เคลียร์เบรคพ้อยท์(clear breakpoint), เรียกดูค่าตัวแปรได้(add watch), แสดงเธรดที่กำลังทำงานตรงตามเป้าหมายที่ตั้งไว้ แต่ก็ยังมีปัญหาอยู่บ้าง คือ การแสดงข้อผิดพลาดของโปรแกรมในดีบักเกอร์ที่สร้างขึ้นยังไม่สมบูรณ์ ซึ่งในเจดีบีสามารถแสดงผลได้เช่น เมื่อเกิดข้อผิดพลาดในการ step ไม่ได้แสดงข้อผิดพลาดที่เกิดขึ้นนี้ให้ผู้ใช้เห็น ซึ่งจะสรุปผลการทดลองได้ดังนี้

1. ได้สร้างดีบักเกอร์ที่เป็น GUI ซึ่งพัฒนามาจาก JPDA ซึ่งมีลักษณะรวมเป็น IDE ซึ่งสามารถเปิดไฟล์มาแก้ไข , สร้างไฟล์ใหม่ , คอมไพล์, เอ็กซิคิวต์ และดีบักได้

2. โปรแกรมไม่ต้องการใช้ทรัพยากรระบบมากนัก เมื่อเทียบกับ tools ที่มีลักษณะเป็น GUI อื่น ๆ ทำให้มีความรวดเร็วในการใช้งาน

## 7.2 ปัญหาที่เกิดขึ้น

1 เอกสารเกี่ยวกับ JPDA มีน้อย ทำให้การศึกษาเกี่ยวกับการสร้างดีบักเกอร์เป็นไปได้ช้าโดยเฉพาะวิธีการกำหนดค่าสภาวะแวดล้อมต่าง ๆ ที่จำเป็นสำหรับการดีบัก

2 การศึกษาในส่วนของซอร์ทโค้ด JDB นั้นทำได้ยากเพราะไม่มีอ็อบเจ็คต์ไดอะแกรมอธิบายไว้อีกทั้งการแสดงคำอธิบายในโปรแกรม(comment)มีน้อยทำให้การเข้าใจซอร์สโค้ดเป็นไปได้ยาก

3 การสร้างโปรแกรมจากส่วน JDI ที่มีให้นั้นเมื่อเกิดปัญหาหรือเกิดข้อผิดพลาดขึ้นเราไม่สามารถทราบข้อผิดพลาดที่แท้จริงได้ว่าเกิดจากอะไร ทำให้การแก้ปัญหาทำได้ยาก

## 7.3 วิธีการแก้ปัญหา

ปัญหาที่เกิดขึ้นส่วนใหญ่มาจากเอกสารไม่เพียงพอ ทำให้การทำงานเป็นไปได้ยาก วิธีการแก้ไขคือใช้อีเมล์ไปสอบถามกับผู้พัฒนา JPDA (debugger@java.sun.com)

## 7.4 แนวทางการพัฒนา

1 นำซอร์สโค้ดมาแก้ไขเพื่อเพิ่มฟังก์ชันในการใช้งาน เช่น การแสดงตัวแปรและเมดธอดทั้งหมดที่มีภายในคลาสนั้น,ให้สนับสนุนการสร้างที่มีลักษณะเป็นแพคเก็จ

2 พัฒนาให้มีความฉลาด(Intelligent) เช่นมีลักษณะโต้ตอบถามกับโปรแกรมเมอร์ได้ หรือสามารถแก้ไขความผิดพลาดบ้างอย่างได้ด้วยตนเอง

# ภาคผนวก ก.

# รายละเอียดของระบบ KMITL Java Developer version 1.0

## ก.1 ยูสเคส

รายละเอียดของฟังก์ชั่นต่างๆ ที่มีในระบบ KMITL Java Developer version 1.0แสดงได้ดังยูส-เคสรูปที่ ก-1


*รูปที่ ก-1 ยูสเคสโปรแกรม KMITL Java Developer version 1.0*

ในหัวข้อต่อไป จะกล่าวถึงรายละเอียดของแต่ละยูสเคส ในส่วนของเงื่อนไขก่อนการใช้งาน (precondition), เงื่อนไขหลังการใช้งาน (postcondition), เส้นทางหลัก (main flow) และเส้นทางเมื่อเกิดข้อผิดพลาด (error flow) ซึ่งจะทำให้สามารถกำหนดขอบเขตของการพัฒนาได้ดีขึ้น และส่งผลให้การออกแบบและสร้างรวดเร็วและถูกต้องมากขึ้น

ก.1.1 ยูสเคส new

เงื่อนไขก่อนการทำงาน: ต้องเปิดโปรแกรม KMITL Java Developer version 1.0 ขึ้นมาแล้ว

เงื่อนไขหลังการทำงาน: ไม่มี

เส้นทางหลัก: เมื่อกดเลือกฟังก์ชั่น new (รูปกระดาษ) บนแถบเครื่องมือ (tool bar) แล้วจะเกิดหน้าต่างอีดิตเตอร์ใหม่มีชื่อว่า Noname1.java และหากกดอีกจะเปลี่ยนชื่อไฟล์เป็น Noname2.java และ Noname3.java ไปเรื่อยๆ

เส้นทางเมื่อเกิดข้อผิดพลาด: ไม่มี

ก.1.2 ยูสเคส open

เงื่อนไขก่อนการทำงาน: ต้องเปิดโปรแกรม KMITL Java Developer version 1.0 ขึ้นมาแล้ว

เงื่อนไขหลังการทำงาน: ไม่มี

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น open จะมีไดอะลอกให้เลือกไฟล์ภาษาจาวาที่ต้องการเปิดขึ้นมาแก้ไข และโหลดเนื้อหาในไฟล์ดังกล่าวไว้ในหน้าต่างอีดิทเตอร์ และเซตชื่อตามชื่อไฟล์ที่ถูกเปิด

เส้นทางเมื่อเกิดข้อผิดพลาด: ไม่มี

ก.1.3 ยูสเคส edit

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกใช้คำสั่ง new และ open มาก่อนเพื่อให้มีหน้าต่างอีดิทเตอร์มาใช้ในการแก้ไขโปรแกรมได้

เงื่อนไขหลังการทำงาน: เมื่อผู้ใช้กดแป้นพิมพ์ (keyboard) เพื่อเปลี่ยนแปลงข้อมูลในหน้าต่างอีดิทเตอร์แล้ว ข้อมูลในหน้าต่างอีดิทเตอร์ต้องเปลี่ยนแปลงตามข้อมูลที่ผู้ใช้ป้อนให้

เส้นทางหลัก: เมื่อผู้ใช้กดแป้นพิมพ์ (keyboard) เพื่อเปลี่ยนแปลงข้อมูลในหน้าต่างอีดิทเตอร์แล้ว ข้อมูลในหน้าต่างอีดิทเตอร์จะเปลี่ยนแปลงตามข้อมูลที่ผู้ใช้ป้อนเข้าไป

เส้นทางเมื่อเกิดข้อผิดพลาด: ไม่มี

ก.1.4 ยูสเคส save

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกใช้คำสั่ง new และ open มาก่อนเพื่อให้มีหน้าต่างอีดิทเตอร์มาใช้ในการแก้ไขโปรแกรมได้

เงื่อนไขหลังการทำงาน: ไฟล์มีความเปลี่ยนแปลงตามที่แก้ไข

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น save จะมีการบันทึกความเปลี่ยนแปลงที่เกิดขึ้นในอีดิทเตอร์ลงในไฟล์ หากไฟล์เป็นไฟล์ใหม่ที่ไม่มีการบันทึกมาก่อน จะมีหน้าต่างขึ้นขึ้นมาให้ตั้งชื่อไฟล์หรือเปลี่ยนตำแหน่งของไฟล์ใหม่ได้

เส้นทางเมื่อเกิดข้อผิดพลาด: เมื่อเลือกชื่อไฟล์ที่ไม่ตรงตามรูปแบบที่อนุญาตให้มี (illegal format) หน้าต่างไฟล์จะให้กรอกชื่อไฟล์ใหม่ จนกว่าจะถูกต้อง

ก.1.5 ยูสเคส save as

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกใช้คำสั่ง new และ open มาก่อนเพื่อให้มีหน้าต่างอีดิทเตอร์มาใช้ในการแก้ไขโปรแกรมได้

เงื่อนไขหลังการทำงาน: ไฟล์มีความเปลี่ยนแปลงตามที่แก้ไข

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น save as จะมีการบันทึกความเปลี่ยนแปลงที่เกิดขึ้นในอีดิทเตอร์ลงในไฟล์ โดยจะมีหน้าต่างขึ้นขึ้นมาให้ตั้งชื่อไฟล์หรือเปลี่ยนตำแหน่งของไฟล์ใหม่ได้

เส้นทางเมื่อเกิดข้อผิดพลาด: เมื่อเลือกชื่อไฟล์ที่ไม่ตรงตามรูปแบบที่อนุญาติให้มี (illegal file format) หน้าต่างไฟล์จะให้กรอกชื่อไฟล์ใหม่ จนกว่าจะถูกต้อง

ก.1.6 ยูสเคส set environment

เงื่อนไขก่อนการทำงาน: ไม่มี

เงื่อนไขหลังการทำงาน: ตำแหน่งของชุดพัฒนาโปรแกรมภาษาจาวา (JDK๗) จะถูกเซ็ตตามค่าที่ส่งเข้ามา

เส้นทางหลัก: จะมีหน้าต่างเปิดไฟล์ขึ้นมาให้ผู้ใช้เลือกตำแหน่งของคอมไพล์เลอร์ภาษาจาวา (โปรแกรม javac.exe) ซึ่งสามารถทำให้ทราบตำแหน่งของ JDK ได้โดยปริยาย แล้วจะเปลี่ยนแปลงค่าตามข้อมูลที่ได้

เส้นทางเมื่อเกิดข้อผิดพลาด: ไม่มี เนื่องจากมีการตรวจสอบแล้วว่าที่ตำแหน่งนั้นมีโปรแกรม javac.exe หรือไม่

ก.1.7 ยูสเคส compile

เงื่อนไขก่อนการทำงาน: จะต้องมีโปรแกรมภาษาจาวาอยู่ในอีดิทเตอร์ที่ทำการบันทึกเรียบร้อยแล้ว

เงื่อนไขหลังการทำงาน: ในกรณีที่ไม่เกิดความผิดพลาดในการคอมไพล์โปรแกรม จะมีคลาสไฟล์ของโปรแกรมที่ถูกคอมไพล์เกิดขึ้น เช่นถ้าไฟล์โปรแกรมภาษาจาวาที่ถูกคอมไพล์ชื่อ A.java จะเกิดคลาสไฟล์ที่มีชื่อว่า A.class

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น compile จะมีหน้าต่างให้ใส่อาร์กิวเมนต์ (argument) ที่ใช้ในการคอมไพล์ จากนั้นแอพพลิเคชั่น KMITL Java Developer version 1.0 จะไปเรียกโปรแกรม javac.exe ซึ่งเป็นคอมไพล์เลอร์ในภาษาจาวาขึ้นมาเพื่อคอมไพล์โปรแกรม พร้อมทั้งมีการส่งออปชั่นที่ผู้ใช้ใส่เข้าไปในคอมไพล์เลอร์ดังกล่าวด้วย

เส้นทางเมื่อเกิดข้อผิดพลาด: เมื่อเกิดความผิดพลาดเมื่อคอมไพล์ (compile-time error) นั่นคือโปรแกรมจะคอมไพล์ไม่ผ่าน อันอาจจะสืบเนื่องมาจากความผิดพลาดทางไวยากรณ์ของ

โปรแกรม จะมีหน้าต่างแสดงความผิดพลาดดังกล่าว เพื่อให้ผู้ใช้สามารถแก้โปรแกรมภาษาจาวาให้ถูกต้องได้

ก.1.8 ยูสเคส execute

เงื่อนไขก่อนการทำงาน: ต้องมีจาวาไบต์โค้ดที่เกิดจากการคอมไพล์เรียบร้อยแล้ว

เงื่อนไขหลังการทำงาน: ไม่มี

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น execute จะมีหน้าต่างให้ใส่อาร์กิวเมนต์และออปชั่น (option) ที่ใช้ในการเอ็กซิคิวต์ จากนั้นแอพพลิเคชั่น KMITL Java Developer version 1.0 จะไปเรียกโปรแกรม java.exe ซึ่งเป็นอินเตอร์พรีทเตอร์ในภาษาจาวาขึ้นมาเพื่อเอ็กซิคิวท์โปรแกรม พร้อมทั้งมีการส่งอาร์กิวเมนต์และออปชั่นที่ผู้ใช้ใส่เข้าไปในอินเตอร์พรีทเตอร์ดังกล่าวด้วย

เส้นทางเมื่อเกิดข้อผิดพลาด: เมื่อมีความผิดพลาดเมื่อโปรแกรมทำงาน (run-time error) นั่นคือโปรแกรมจะไม่สามารถทำงานได้ อันอาจจะสืบเนื่องมาจากไม่มีคลาสไฟล์ที่ต้องการอยู่ หรืออาจมีการกระทำบางอย่างที่ผิดพลาดที่ไม่สามารถตรวจพบได้ในขณะคอมไพล์ เช่นอาจมีกรณีที่มีการหารด้วยศูนย์ จะมีหน้าต่างแสดงความผิดพลาดดังกล่าว เพื่อให้ผู้ใช้สามารถแก้โปรแกรมภาษาจาวาให้ถูกต้องได้

ก.1.9 ยูสเคส debug

เงื่อนไขก่อนการทำงาน: จะต้องมีคลาสไฟล์ที่คอมไพล์โดยใส่ออปชั่น "-g" เพื่อให้คอมไพล์เลอร์ใส่ข้อมูลที่จำเป็นต่อการดีบักเข้าไปในจาวาไบต์โค้ด

เงื่อนไขหลังการทำงาน: มีหน้าต่างที่ใช้ในการดีบักเกิดขึ้น และมีจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้นด้วย

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น debug จะมีหน้าต่างสำหรับการดีบักขึ้นมา โดยจะมีหน้าที่ในการเรียกฟังก์ชั่นย่อยของการดีบัก และจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้น

เส้นทางเมื่อเกิดข้อผิดพลาด: เมื่อมีความผิดพลาดในขณะที่โปรแกรมทำงาน (run-time error) นั่นคือจาวาเวอร์ชวลแมชชีนจะไม่สามารถทำงานได้ อันอาจจะสืบเนื่องมาจากไม่มีคลาสไฟล์ที่ต้องการอยู่ หรืออาจมีการกระทำบางอย่างที่ผิดพลาดที่ไม่สามารถตรวจพบได้ในขณะคอมไพล์ เช่นอาจมีกรณีที่มีการหารด้วยศูนย์ จะมีหน้าต่างแสดงความผิดพลาดดังกล่าว เพื่อให้ผู้ใช้สามารถแก้โปรแกรมภาษาจาวาให้ถูกต้องได้

ก.1.10 ยูสเคส set breakpoint

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกฟังก์ชั่น debug ซึ่งจะทำให้เกิดหน้าต่างที่ใช้ในการดีบักเกิดขึ้น และมีจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้นแล้ว

เงื่อนไขหลังการทำงาน: เบรคพอยต์ถูกเซ็ตในบรรทัดที่เลือกไว้ และตัวอักษรในบรรทัดนั้นจะกลายเป็นสีแดงเพื่อแสดงให้ทราบว่าที่บรรทัดนั้นมีการเซ็ตเบรคพอยต์

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น set breakpoint ค่าบรรทัดจะส่งเข้าไปเพื่อเซ็ตเบรคพอยต์ เพื่อให้โปรแกรมพักการทำงานชั่วคราวที่บรรทัดดังกล่าว และตัวอักษรในบรรทัดนี้จะกลายเป็นสีแดงขึ้นเพื่อให้รู้ว่าตรงจุดนี้มีการเซ็ตเบรกพอยต์

เส้นทางเมื่อเกิดข้อผิดพลาด: ไม่มี

ก.1.11 ยูสเคส clear breakpoint

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกฟังก์ชั่น debug ซึ่งจะทำให้เกิดหน้าต่างที่ใช้ในการดีบักเกิดขึ้น, มีจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้นเรียบร้อยแล้ว และมีเบรคพอยต์ถูกเซ็ตในบรรทัดดังกล่าว

เงื่อนไขหลังการทำงาน: เบรคพอยต์ที่เซ็ตไว้ถูกเคลียร์ รวมทั้งตัวอักษรในบรรทัดนั้นจะเปลี่ยนเป็นสีดำตามปกติ

เส้นทางหลัก: เมื่อมีการเรียกฟังก์ชั่น clear breakpoint ค่าเบรกพอยต์ที่ถูกเซ็ตไว้จะถูกเคลียร์ และเอาตัวอักษรในบรรทัดนั้นจะกลายเป็นสีดำตามเดิม

เส้นทางเมื่อเกิดข้อผิดพลาด: เมื่อมีการเคลียร์เบรกพอยต์ในบรรทัดที่ไม่มีการเซ็ตเบรคพอยต์เอาไว้ ก็จะไม่มีการเปลี่ยนแปลงใดๆ เกิดขึ้น

ก.1.12 ยูสเคส run to breakpoint

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกฟังก์ชั่น debug ซึ่งจะทำให้เกิดหน้าต่างที่ใช้ในการดีบักเกิดขึ้น, มีจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้นแล้ว และมีเบรคพอยต์ถูกเซ็ตในบรรทัดดังกล่าว

เงื่อนไขหลังการทำงาน: โปรแกรมทำงานไปจนถึงตำแหน่งที่ถูกเซ็ตเบรกพอยต์

เส้นทางหลัก: เมื่อเรียกฟังก์ชั่น run to breakpoint โปรแกรมจะรันไปถึงจุดที่ได้เซ็ตเบรกพอยต์ไว้ รวมทั้งแสดงค่าตัวแปรและเธรดเพื่อประโยชน์ในการดีบักโปรแกรม

เส้นทางเมื่อเกิดข้อผิดพลาด: หากไม่มีการเซ็ตเบรคพอยต์ไว้เลย โปรแกรมจะรันจนจบโปรแกรมไปเอง

ก.1.13 ยูสเคส step

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกฟังก์ชั่น debug ซึ่งจะทำให้เกิดหน้าต่างที่ใช้ในการดีบักเกิดขึ้น, มีจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้นด้วย

เงื่อนไขหลังการทำงาน: โปรแกรมหยุดการทำงานในสเตทเมนต์ (statement) ต่อไป

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น step โปรแกรมจะหยุดที่บรรทัดต่อไป รวมทั้งแสดงค่าตัวแปรและ เธรดเพื่อประโยชน์ในการดีบักโปรแกรม

เส้นทางเมื่อเกิดข้อผิดพลาด: ไม่มี

ก.1.14 ยูสเคส next

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกฟังก์ชั่น debug ซึ่งจะทำให้เกิดหน้าต่างที่ใช้ในการดีบักเกิดขึ้น, มีจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้นด้วย

เงื่อนไขหลังการทำงาน: โปรแกรมหยุดการทำงานในสเตตเมนต์ (statement) ต่อไป แต่จะไม่เข้าไปในส่วนย่อยของเมทธอด

เส้นทางหลัก: เมื่อเลือกฟังก์ชั่น next โปรแกรมจะหยุดที่บรรทัดต่อไป แต่จะไม่เข้าไปในส่วนย่อยของเมทธอด รวมทั้งแสดงค่าตัวแปรและ เธรดเพื่อประโยชน์ในการดีบักโปรแกรม

เส้นทางเมื่อเกิดข้อผิดพลาด: ไม่มี

ก.1.15 ยูสเคส add watch

เงื่อนไขก่อนการทำงาน: ต้องมีการเรียกฟังก์ชั่น debug ซึ่งจะทำให้เกิดหน้าต่างที่ใช้ในการดีบักเกิดขึ้น, มีจาวาเวอร์ชวลแมชชีนของโปรแกรมที่ต้องการดีบักเกิดขึ้นด้วย

เงื่อนไขหลังการทำงาน: แสดงค่าตัวแปรที่ add watch ไว้

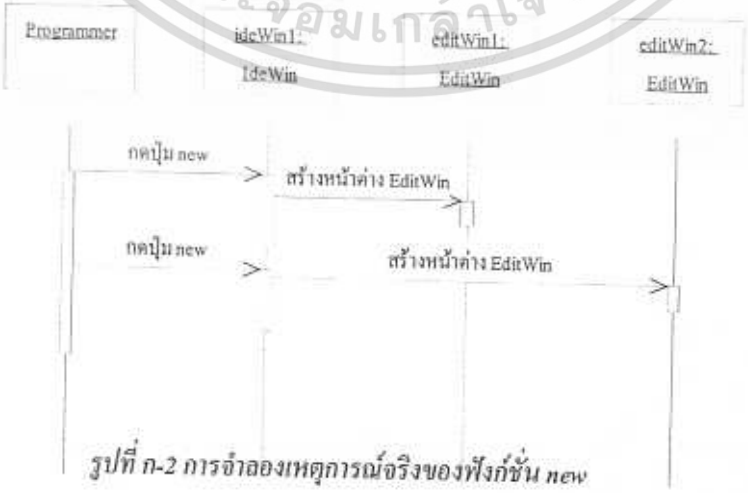
เส้นทางหลัก: เมื่อมีการเรียกฟังก์ชั่น add watch จะมีหน้าต่างขึ้นมาให้ใส่ตัวแปรที่ต้องการมอนิเตอร์ (monitor) และเมื่อมีการทำงานแต่ละสเตทเมนต์จะการอัพเดต (update) ค่าดังกล่าวด้วย

เส้นทางเมื่อเกิดข้อผิดพลาด: เมื่อมีการ add watch ตัวแปรที่ไม่มีอยู่จริงจะแสดงค่าเป็น "object not found"

**ก.2 การจำลองเหตุการณ์จริง (scenario)**

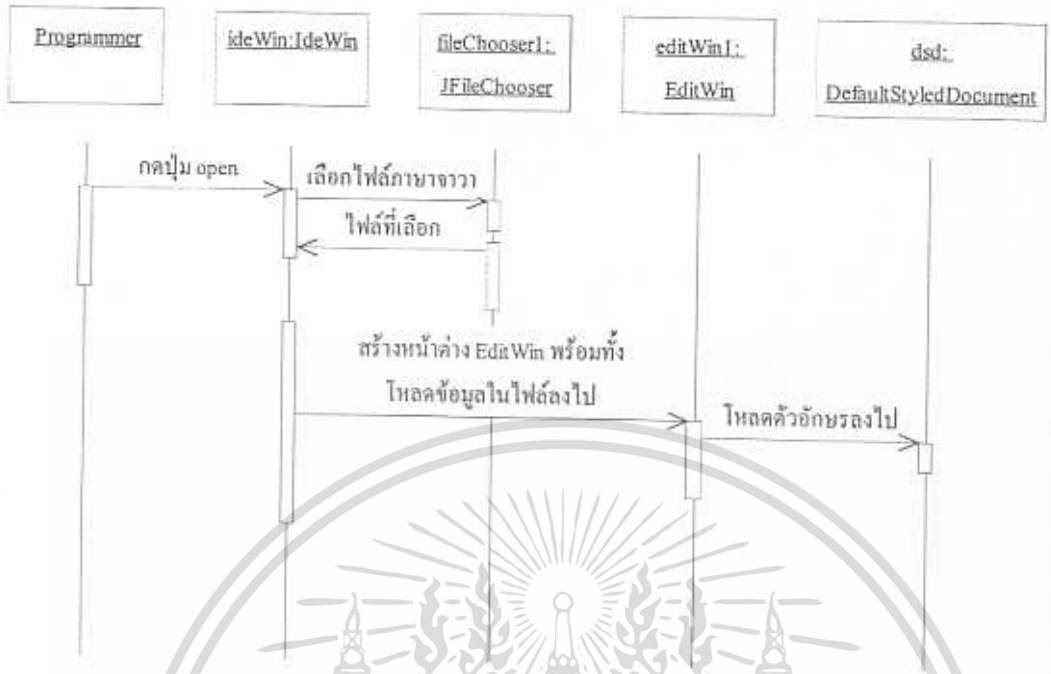
จะแสดงรายละเอียดระบบของ JavaIDEX ผ่านการจำลองเหตุการณ์จริงของแต่ละฟังก์ชั่นดังนี้

ก.2.1 การจำลองเหตุการณ์จริงของฟังก์ชั่น new




*รูปที่ ก-2 การจำลองเหตุการณ์จริงของฟังก์ชั่น new*

ก.2.2 การจำลองเหตุการณ์จริงของฟังก์ชั่น open

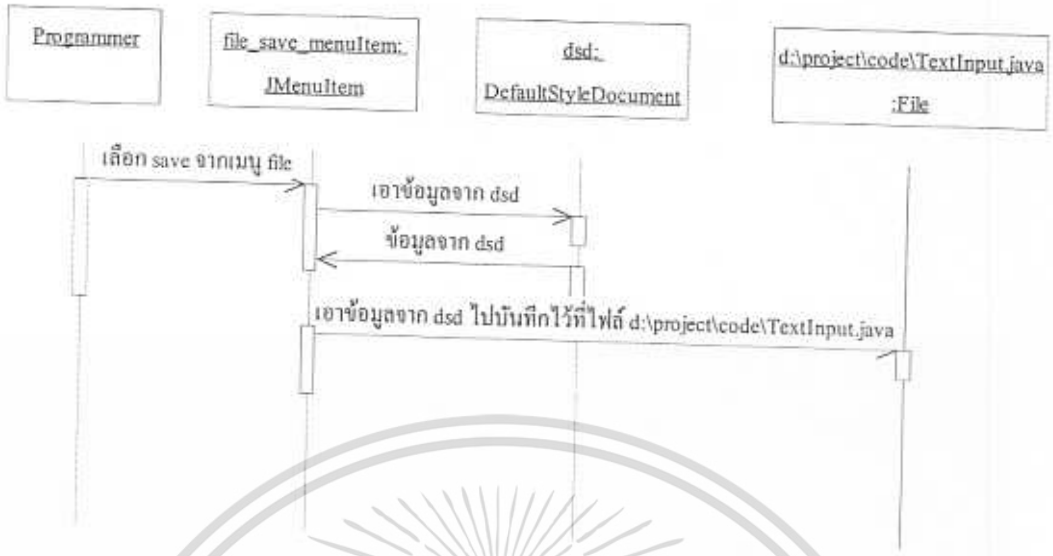



*รูปที่ ก-3 การจำลองเหตุการณ์จริงของฟังก์ชั่น open*

ก.2.3 การจำลองเหตุการณ์จริงของฟังก์ชั่น edit


*รูปที่ ก-4 การจำลองเหตุการณ์จริงของฟังก์ชั่น edit*

ก.2.4 การจำลองเหตุการณ์จริงของฟังก์ชั่น save

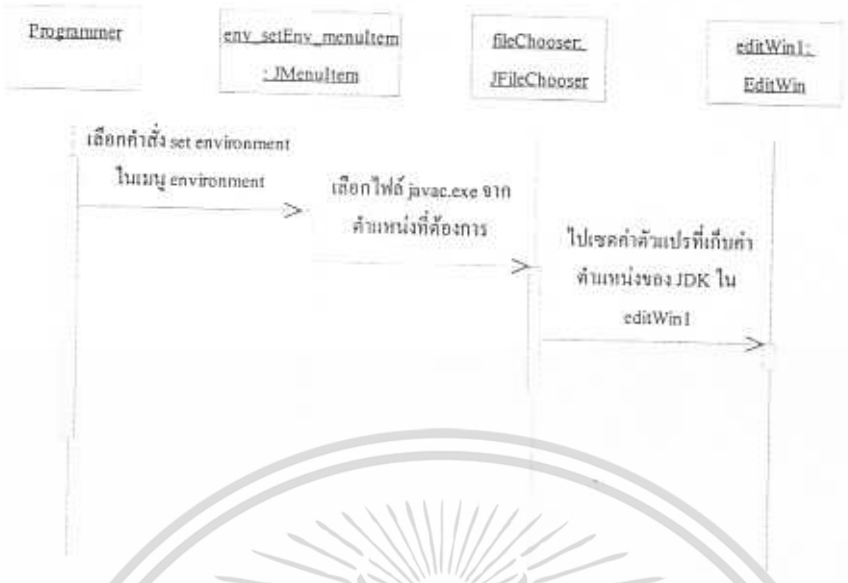



*รูปที่ ค-5 การจำลองเหตุการณ์จริงของฟังก์ชั่น save*

ก.2.5 การจำลองเหตุการณ์จริงของฟังก์ชั่น save as


*รูปที่ ก-6 การจำลองเหตุการณ์จริงของฟังก์ชั่น save as*

ก.2.6 การจำลองเหตุการณ์จริงของฟังก์ชั่น set environment

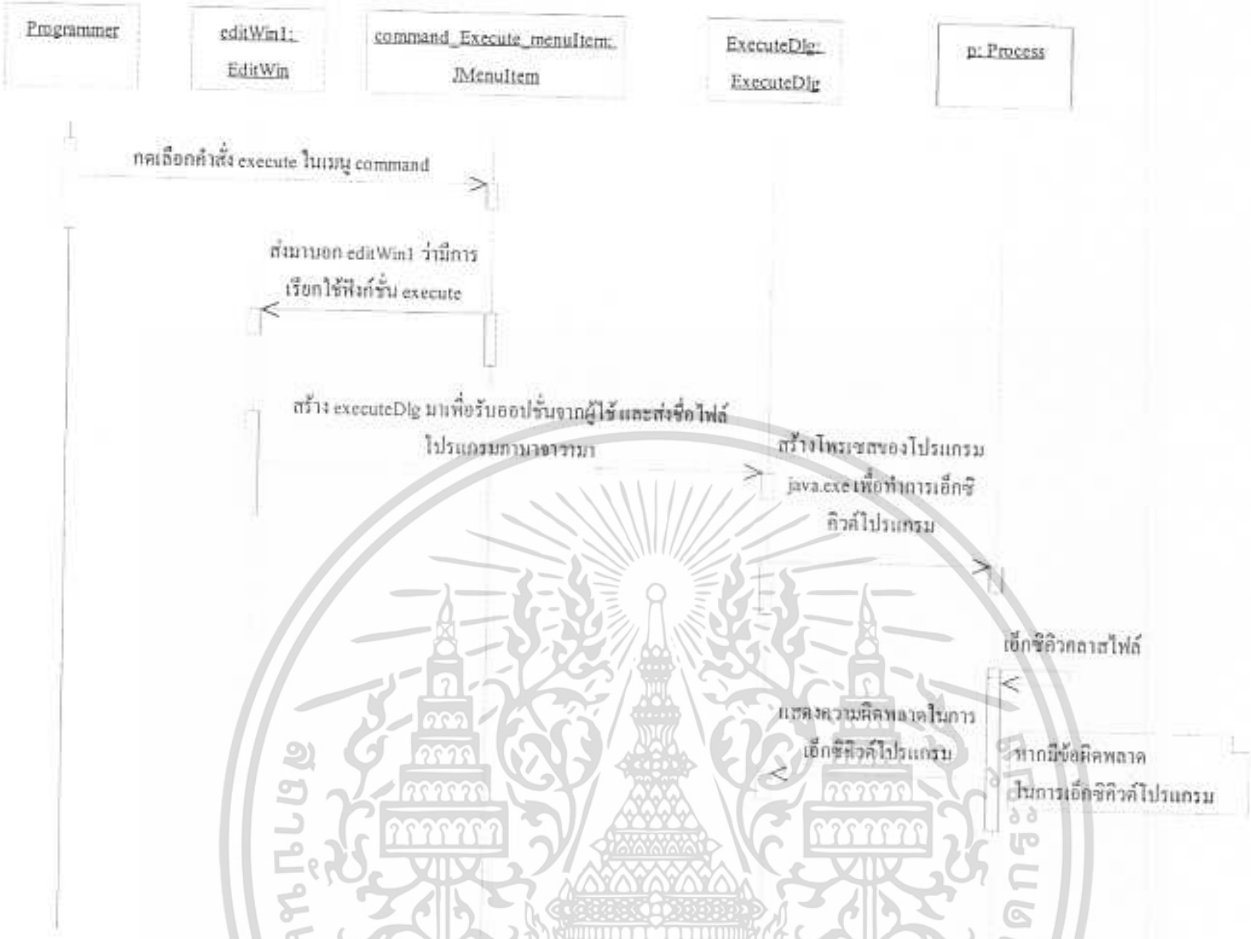



*รูปที่ ก-7 การจำลองเหตุการณ์จริงของฟังก์ชั่น set environment*

ก.2.7 การจำลองเหตุการณ์จริงของฟังก์ชั่น compile

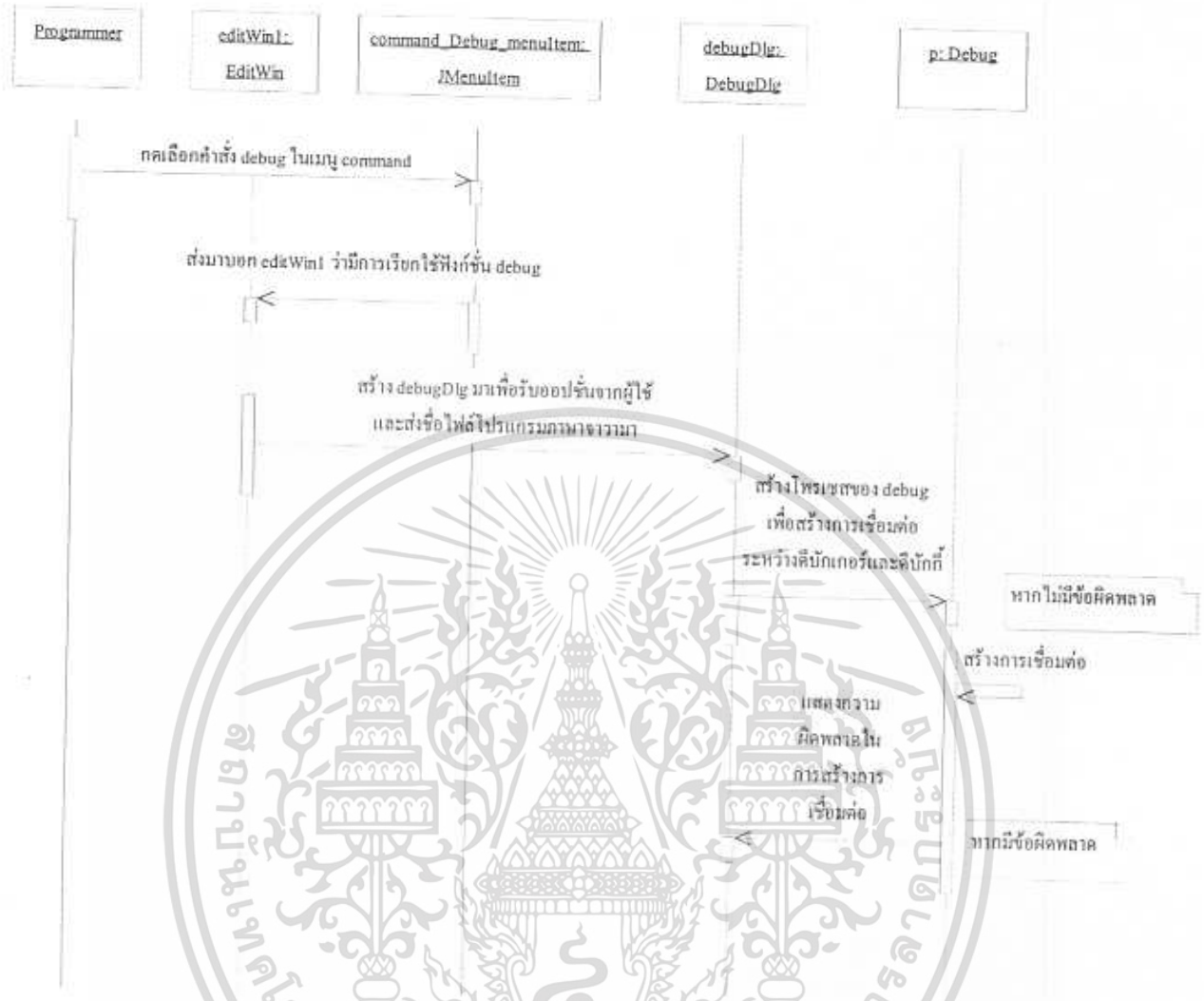

*รูปที่ ก-8 การจำลองเหตุการณ์จริงของฟังก์ชั่น compile*

## ก.2.8 การจำลองเหตุการณ์จริงของฟังก์ชั่น execute




*รูปที่ ก-9 การจำลองเหตุการณ์จริงของฟังก์ชั่น execute*

ก.2.9 การจำลองเหตุการณ์จริงของฟังก์ชั่น debug




*รูปที่ ก-10 การจำลองเหตุการณ์จริงของฟังก์ชั่น debug*

ก.2.10 การจำลองเหตุการณ์จริงของฟังก์ชั่น set breakpoint


*รูปที่ ก-11 การจำลองเหตุการณ์จริงของฟังก์ชั่น set breakpoint*

ก.2.11 การจำลองเหตุการณ์จริงของฟังก์ชั่น clear breakpoint


*รูปที่ ก-12 การจำลองเหตุการณ์จริงของฟังก์ชั่น clear breakpoint*

ก.2.12 การจำลองเหตุการณ์จริงของฟังก์ชั่น run to breakpoint


*รูปที่ ก-13 การจำลองเหตุการณ์จริงของฟังก์ชั่น run to breakpoint*

ก.2.13 การจำลองเหตุการณ์จริงของฟังก์ชั่น step


*รูปที่ ก-14 การจำลองเหตุการณ์จริงของฟังก์ชั่น step*

ก.2.14 การจำลองเหตุการณ์จริงของฟังก์ชั่น next


*รูปที่ ก-15 การจำลองเหตุการณ์จริงของฟังก์ชั่น next*

ก.2.15 การจำลองเหตุการณ์จริงของฟังก์ชั่น add watch


*รูปที่ ก-16 การจำลองเหตุการณ์จริงของฟังก์ชั่น add watch*

# บรรณานุกรม


[1] ดร.วีระศักดิ์ ซึงถาวร: *" Fundamental of Java Programming Volume 1"*, SUM Publishing Department, SUM System Company Limited, 390 หน้า, 2541.

[2] Bill Venners: *"Inside The Java Virtual Machine"*, McGraw Hill, 597 p, 1997.

[3] Bruce Powel Douglass: "Real-Time UML: Developing Efficient Objects for Embedded Systems", Addison-Wesley Longman Inc., Massachusetts, 1998.

[4] *"Java Platform Debugger Architecture"*, http://java.sun.com/product/jpda.

[5] Kathy Walrath, Mary Campione : *"The Java Tutorial Second Edition: Object-Oriented Programming for the Internet. "*, Addison-Wesley Publishing Co, 964 p, 1998.

[6] Mark Watson and Paul Harmon: "Understanding UML: The Developer's Guide With a Web-Based Application iin Java", Morgan Kaufman Publishers Inc., California, 1998.

# ภาคผนวก ข.

# ซอร์สโค้ด

## ข.1 ซอร์สโค้ดของคลาส JavaIDEX

```
package JavaIDEX;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class JavaIDEX extends JFrame{
        JLabel logo;
        Timer timer;
        final int THREE_SEC = 3000;
        public JavaIDEX() {
                getContentPane().setLayout(new BorderLayout());
                logo = new JLabel(new ImageIcon("images/logo.gif"));
                getContentPane().add(logo, BorderLayout.NORTH);
                setSize(405,350);
                setLocation(200,150);
                setVisible(true);
                timer = new Timer(THREE_SEC, new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                setVisible(false);
                                timer.stop();
                                IdeWin ideWin = new IdeWin();
                        }
                });
                timer.start();
        }
        public static void main(String[] args) {
                JavaIDEX javaIDEX = new JavaIDEX();
        }
}
```

## ข.2 ซอร์สโค้ดของคลาส IdeWin

```
package JavaIDEX;
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import javax.swing.*;
import javax.swing.filechooser.*;
public class IdeWin extends JFrame{
        JDesktopPane contentPane;
        JToolBar toolBar;
        JButton new_button, open_button;
        EditWin editWin;
        final JFileChooser fileChooser = new JFileChooser();
        static String currPath = null, currFileName = null, currJavName = null;
        public IdeWin() {
                toolBar = new JToolBar();
                new_button = new JButton(new ImageIcon("images/new.gif"));
                toolBar.add(new_button);
                open_button = new JButton(new ImageIcon("images/open.gif"));
                toolBar.add(open_button);
                contentPane = new JDesktopPane();
                contentPane.setLayout(new BorderLayout());
                contentPane.add(toolBar, BorderLayout.NORTH);
                setContentPane(contentPane);
                addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                        System.exit(0);
                        }
                });
                new_button.addActionListener(new ActionListener()  {
                        public void actionPerformed(ActionEvent e) {
```

```
                createEditWin();
            }
        });
        open_button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                openFile();
            }
        });
        setTitle("KMITL Java Developer v1.0");
        setSize(800, 575);
        setVisible(true);
    }
    protected void createEditWin() {
        editWin = new EditWin();
        contentPane.add(editWin);
            try {
                editWin.setSelected(true);
            } catch (java.beans.PropertyVetoException e) {}
    }
    protected void createEditWin(String currPath, String currFileName, String currJavName, File file) {
        editWin = new EditWin(currPath, currFileName, currJavName, file);
        contentPane.add(editWin);
            try {
                editWin.setSelected(true);
            } catch (java.beans.PropertyVetoException e) {}
    }
    protected void openFile() {
        // remove acceptAllFileFilter
        fileChooser.removeChoosableFileFilter(fileChooser.getAcceptAllFileFilter());
        // set file filter
        fileChooser.setFileFilter(new FileFilter() {
            public boolean accept(File f) {
                return f.getName().toLowerCase().endsWith(".java") || f.isDirectory();
            }
            public String getDescription() {
                return "Java source code (*.java)";
            }
        });
        int returnVal = fileChooser.showOpenDialog(this);
        // set file filter
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            // current path + file name"
            currFileName = fileChooser.getSelectedFile().getPath().toString();
            // current path of chosen file
            currPath = fileChooser.getCurrentDirectory().toString();
            // current path + file name exclude ".java"
            String tempN = fileChooser.getName(file);
            int deletePos = tempN.indexOf(".java");
            currJavName = tempN.substring(0,deletePos);
            createEditWin(currPath,currFileName,currJavName,file);
        }
    }
}
```

### ข.3 ซอร์สโค้ดของคลาส EditWin

```
package JavaIDEX;
import java.awt.*;
import java.awt.event.*;
import java.util.Hashtable;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.filechooser.*;
import javax.swing.text.*;
import javax.swing.undo.*;
import javax.swing.plaf.basic.*;
public class EditWin extends JInternalFrame{
    static int openFrameCount = 0;
    static final int xOffset = 30, yOffset = 30;
```

```
JPanel contentPane;
JSplitPane splitPane;
JScrollPane scrollPane1,scrollPane2 ;
JTextPane textPane;
DefaultStyledDocument dsd;
DebugPane debugPane;
CompileDlg compileDlg;
File file;
boolean dirty = false;
JMenuBar menuBar;
JMenu fileMenu, envMenu, commandMenu;
JMenuItem file_Save_menuItem, file_SaveAs_menuItem, file_Exit_menuItem,
        env_setEnv_menuItem,
        command_Compile_menuItem, command_Execute_menuItem, command_Debug_menuItem;
String currPath = null, currFileName = null, currJavName= null, JDKLoc = "d:\\jdk\\bin";
SimpleAttributeSet[] attrs;
JViewport dbp_viewport;
Point dbv_point;
final int line2Pixel = 17;
final JFileChooser fileChooser = new JFileChooser();
int threshold;
JLabel statusBar;
int char2Line[];
public EditWin() {
        super("Noname" + (++openFrameCount)+ ".java", true, true, true,true);
        createWin();
}
public EditWin(String currPath, String currFileName, String currJavName, File file) {
        super(currFileName,true, true, true, true);
        this.currPath = currPath;
        this.currFileName = currFileName;
        this.currJavName = currJavName;
        this.file = file;
        createWin();
        try {
                int size = (int)file.length();
                int chars_read = 0;
                FileReader in = new FileReader(file);
                char[] data = new char[size];
                char2Line = new int[200];
                while(in.ready()) {
                        chars_read += in.read(data,chars_read,size-chars_read);
                }
                char[] realString = new char[size];
                int line = 1;
                int j = 0;
                for (int i=0; i <= size-1 ; i++) {
                        if (data[i] == '\r') {
                                char2Line[line] = j;
                                line++;
                        } else {
                                data[j] = data[i];
                                j++;
                        }
                }
                try {
                        String temp = new String(data,0,j);
                        attrs = initAttributes(size);
                        dsd.insertString(dsd.getLength(), temp, attrs[0]);
                } catch (BadLocationException ble) {}
                in.close();
        } catch(IOException e) {}
}
public void createWin(){
        setVisible(true);
        setSize(700, 500);
        setLocation(xOffset*openFrameCount, yOffset*openFrameCount);
        // 1. create content pane
        contentPane = new JPanel();
        contentPane.setLayout(new BorderLayout());
        setContentPane(contentPane);
        // 2. create menu bar
        menuBar = new JMenuBar();
        setJMenuBar(menuBar);
        // 2.1 create menu file
```

```
fileMenu = new JMenu("File");
fileMenu.setMnemonic(KeyEvent.VK_F);
menuBar.add(fileMenu);
file_Save_menuItem = new JMenuItem("Save",KeyEvent.VK_S);
        file_Save_menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,
                ActionEvent.CTRL_MASK));
fileMenu.add(file_Save_menuItem);
file_SaveAs_menuItem = new JMenuItem("Save As...",KeyEvent.VK_A);
fileMenu.add(file_SaveAs_menuItem);
file_Exit_menuItem = new JMenuItem("Exit");
fileMenu.add(file_Exit_menuItem);
// 2.3 create menu environment
envMenu = new JMenu("Environment");
envMenu.setMnemonic(KeyEvent.VK_E);
menuBar.add(envMenu);
env_setEnv_menuItem = new JMenuItem("Set Environment",KeyEvent.VK_S);
envMenu.add(env_setEnv_menuItem);
// 2.3 create menu command
commandMenu = new JMenu("Command");
commandMenu.setMnemonic(KeyEvent.VK_C);
menuBar.add(commandMenu);
command_Compile_menuItem = new JMenuItem("Compile",KeyEvent.VK_C);
command_Compile_menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C,
        ActionEvent.CTRL_MASK)); // set shortcut key
commandMenu.add(command_Compile_menuItem);
command_Execute_menuItem = new JMenuItem("Execute",KeyEvent.VK_E);
command_Execute_menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,
        ActionEvent.CTRL_MASK)); // set shortcut key
commandMenu.add(command_Execute_menuItem);
command_Debug_menuItem = new JMenuItem("Debug",KeyEvent.VK_D);
command_Debug_menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D,
        ActionEvent.CTRL_MASK)); // set shortcut key
commandMenu.add(command_Debug_menuItem);
// 4. create text editor and Debugger Pane
dsd = new DefaultStyledDocument();
dsd.addDocumentListener(new MyDocumentListener());
textPane = new JTextPane(dsd); // row, column size
textPane.addCaretListener(new MyCaretListener());
scrollPane2 = new JScrollPane(textPane);
debugPane = new DebugPane(this);
scrollPane2.setAutoscrolls(true);
dbp_viewport = new JViewport();
scrollPane2.setViewportView(textPane);
dbp_viewport = scrollPane2.getViewport();
dbv_point = new Point(0,0);
dbp_viewport.setViewPosition(dbv_point);
splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT, debugPane , scrollPane2);
splitPane.setOneTouchExpandable(true);
splitPane.setDividerLocation(0);
contentPane.add(splitPane, BorderLayout.CENTER);
// 5. create Event Listeners
file_Save_menuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                SaveFile();
        }
});
file_Exit_menuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
    SaveChangeBeforeExit();
        }
});
file_SaveAs_menuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                SaveAsFile();
        }
});
env_setEnv_menuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                setJDKLoc();
        }
});
command_Compile_menuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                if (currFileName == null) {
                        JOptionPane.showMessageDialog(contentPane,
```

```
                                        "No Java source code to compile",
                                        "Error",JOptionPane.ERROR_MESSAGE);
                                } else {
                                        SaveFile();
                                        compileDlg = new CompileDlg(currFileName, JDKLoc);
                                        Dimension compileDlgSize = compileDlg.getSize();
                                        Dimension compileFrmSize = getSize();
                                        Point loc = getLocation();
                                        compileDlg.setLocation((compileFrmSize.width -
compileDlgSize.width) / 2 + loc.x, (compileFrmSize.height - compileDlgSize.height) / 2 + loc.y);
                                        compileDlg.setModal(true);
                                        compileDlg.show();
                                }
                        }
                });
                command_Execute_menuItem.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                if (currFileName == null) {
                                        JOptionPane.showMessageDialog(contentPane,
                                                "No Java class file to execute",
                                                "Error",JOptionPane.ERROR_MESSAGE);
                                } else {
                                        SaveFile();
                                        compileDlg = new CompileDlg(currFileName, JDKLoc);
                                        if (compileDlg.compileBeforeExec()) {
                                                JOptionPane.showMessageDialog(contentPane, "There
are some errors in your source code; please edit and compile it again.");
                                                return;
                                        }
                                        ExecuteDlg executeDlg = new ExecuteDlg(currPath,
                                                currJavName, JDKLoc);
                                        Dimension executeDlgSize = executeDlg.getSize();
                                        Dimension executeFrmSize = getSize();
                                        Point loc = getLocation();
                                        executeDlg.setLocation((executeFrmSize.width -
executeDlgSize.width) / 2 + loc.x, (executeFrmSize.height - executeDlgSize.height) / 2 + loc.y);
                                        executeDlg.setModal(true);
                                        executeDlg.show();
                                }
                        }
                });
                command_Debug_menuItem.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                if (currFileName != null) {
                                        SaveFile();
                                        compileDlg = new CompileDlg(currFileName, JDKLoc);
                                        if (compileDlg.compileBeforeExec()) {
                                                JOptionPane.showMessageDialog(contentPane,
                                                        "There are some errors in your source code;
                                                        please edit and compile it again.");
                                                return;
                                        }
                                        int editHeight = scrollPane2.getHeight();
                                        editHeight = editHeight / 2;
                                        threshold = (int) editHeight / 2;
                                        debugPane.startIt();
                                        splitPane.setDividerLocation(270);
                                } else {
                                        JOptionPane.showMessageDialog(contentPane,
                                        "No Java source code to debug",
                                                "Error",JOptionPane.ERROR_MESSAGE);
                                }
                        }
                });
        }
        void setJDKLoc() {
                // remove acceptAllFileFilter
                fileChooser.removeChoosableFileFilter(fileChooser.getAcceptAllFileFilter());
                // set file filter
                fileChooser.setFileFilter(new FileFilter() {
                        public boolean accept(File f) {
                                return f.getName().toLowerCase().equals("javac.exe") || f.isDirectory();
                        }
                        public String getDescription() {
                                return "Java compiler (javac.exe)";
```

```
                }
            });
            int returnVal = fileChooser.showOpenDialog(this);
            //set file filter
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                JDKLoc = fileChooser.getCurrentDirectory().toString();
            }
    }
    boolean SaveFile() {
            if (currFileName == null) {
                return SaveAsFile();
            }
            try {
                File file = new File(currFileName);
                FileWriter out = new FileWriter(file);
                String text = textPane.getText();
                out.write(text);
                out.close();
                dirty = false;
                this.setTitle(currFileName);
                return true;
            }
            catch(IOException e) {}
            return false;
    }
    boolean SaveAsFile() {
            // remove acceptAllFileFilter
            fileChooser.removeChoosableFileFilter(fileChooser.getAcceptAllFileFilter());
            // set file filter
            fileChooser.setFileFilter(new FileFilter() {
                public boolean accept(File f) {
                    return f.getName().toLowerCase().endsWith(".java") || f.isDirectory();
                }
                public String getDescription() {
                    return "Java source code (*.java)";
                }
            });
            if (JFileChooser.APPROVE_OPTION == fileChooser.showSaveDialog(this)) {
                currFileName = fileChooser.getSelectedFile().getPath();
                if (!currFileName.endsWith(".java")) {
                    currFileName = currFileName+".java";
                }
                return SaveFile();
            } else {
                this.repaint();
                return false;
            }
    }
    boolean SaveChangeBeforeExit() {
            if (dirty) {
                int optionPane = JOptionPane.showConfirmDialog(contentPane,
                        "Save change before exit?",
                        "Save change?", JOptionPane.YES_NO_CANCEL_OPTION);
                if (optionPane == JOptionPane.YES_OPTION) {
                    SaveFile();
                } else if (optionPane == JOptionPane.NO_OPTION) {
                    this.dispose();
                }
            }
            return true;
    }
    protected SimpleAttributeSet[] initAttributes(int length) {
            //Hard-code some attributes.
             SimpleAttributeSet[] attrs = new SimpleAttributeSet[length];
            // normal text
            attrs[0] = new SimpleAttributeSet();
            StyleConstants.setFontFamily(attrs[0], "SansSerif");
             StyleConstants.setFontSize(attrs[0], 12);
            // breakpoint setted text...
            attrs[1] = new SimpleAttributeSet(attrs[0]);
            StyleConstants.setBold(attrs[1], true);
            StyleConstants.setForeground(attrs[1], Color.red);
            StyleConstants.setBackground(attrs[1], Color.green);
             // step text
            attrs[2] = new SimpleAttributeSet(attrs[0]);
```

```
                StyleConstants.setBold(attrs[2], true);
                StyleConstants.setForeground(attrs[2], Color.blue);
                return attrs;
        }
        protected class MyDocumentListener
          implements DocumentListener {
                public void insertUpdate(DocumentEvent e) {
                        dirty = true;
                }
                public void removeUpdate(DocumentEvent e) {
                        dirty = true;
                }
                public void changedUpdate(DocumentEvent e) {
                        dirty = true;
                }
        }
        public class MyCaretListener
         implements CaretListener {
                public void caretUpdate(CaretEvent e) {
                        debugPane.dot = e.getDot();
                        try {
                                debugPane.caretCoords = textPane.modelToView(debugPane.dot);
                        } catch (BadLocationException ble) {}
                }
        }
}
.
```

## ข.4 ซอร์สโค้ดของคลาส CompileDlg

```
package JavaIDEX;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
public class CompileDlg extends JDialog {
        final Box contentPane = Box.createVerticalBox();
        JPanel subPane1,subPane2;
        JTextField options_TF;
        JButton start_button;
        JTextArea result_TA;
        String currFileName = null;
        String JDKLoc;
        JScrollPane resultPane;
        JViewport viewport;
        Point viewportPos;
        public CompileDlg(String s1, String s2) {
                currFileName = s1;
                JDKLoc = s2;
                setTitle("Compiling");
                // create content pane
                Box contentPane = Box.createVerticalBox();
                contentPane.add(Box.createGlue());
                // create sub pane 1
                subPane1 = new JPanel();
                subPane1.setLayout(new FlowLayout());
                subPane1.add(new JLabel("options"));
                options_TF = new JTextField(15);
                subPane1.add(options_TF);
                contentPane.add(subPane1);
                start_button = new JButton("start");
                contentPane.add(start_button);
                 // create sub pane 2
                result_TA = new JTextArea(5,5);
                resultPane = new JScrollPane(result_TA);
                contentPane.add(new JLabel("result"));
                contentPane.add(resultPane);
                resultPane.setAutoscrolls(true);
                viewport = new JViewport();
                resultPane.setViewportView(result_TA);
                viewportPos = new Point(0,0);
                viewport.setViewPosition(viewportPos);
                getContentPane().add(contentPane,"Center");
                setSize(250,250);
                // create event listener
```

```
                start_button.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                        compile();
                        }
                });
        }
        private void compile() {
                Runtime rt;
                //Properties property;
                rt = Runtime.getRuntime();
                try {
                        Process p = rt.exec(JDKLoc + "\\javac.exe" + " "+"-g" +" "+
                        options_TF.getText() +" " +currFileName);
                        try {
                                p.waitFor();
                                InputStream in = p.getErrorStream();
                                InputStreamReader rin = new InputStreamReader(in);
                                BufferedReader bin = new BufferedReader(rin);
                                String temp;
                                result_TA.setText("");
                                temp = bin.readLine();
                                while (temp != null){
                                        result_TA.append(temp+"\n");
                                        temp = bin.readLine();
                                }
                                viewport.setViewPosition(viewportPos);
                                result_TA.repaint();
                                JOptionPane.showMessageDialog(this, "Compile complete.");
                                rt.freeMemory();
                        }
                        catch (InterruptedException e){}
                } catch (IOException e) {}
        }

        protected boolean compileBeforeExec() {
                Runtime rt;
                String temp="xx";
                rt = Runtime.getRuntime();
                try {
                        Process p = rt.exec(JDKLoc + "\\javac.exe" + " "+"-g" +" "+
                        options_TF.getText() +" " +currFileName);
                        try {
                                p.waitFor();
                                InputStream in = p.getErrorStream();
                                InputStreamReader rin = new InputStreamReader(in);
                                BufferedReader bin = new BufferedReader(rin);
                                temp = bin.readLine();
                                temp = bin.readLine();
                                rt.freeMemory();
                        } catch (InterruptedException e){}
                } catch (IOException e) {}
                if (temp != null)
                        return true; //error, didn't complete compilation
                else return false;
        }
}
```

## ข.5 ซอร์สโค้ดของคลาส ExecuteDlg

```
package JavaIDEX;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
public class ExecuteDlg extends JDialog {
        final Box contentPane = Box.createVerticalBox();
        JPanel subPane1,subPane2;
        JTextField options_TF, args_TF;
        JButton start_button;
        JTextArea result_TA;
        JScrollPane resultPane;
        String currPath = null;
        String currJavName = null;
        String JDKLoc;
```

```
JViewport viewport;
Point viewportPos;
public ExecuteDlg(String s1, String s2, String s3) {
        currPath = s1;
        currJavName = s2;
        JDKLoc = s3;
        setTitle("Executing");
        // create content pane
        Box contentPane = Box.createVerticalBox();
        contentPane.add(Box.createGlue());
        // create sub pane 1
        subPane1 = new JPanel();
        subPane1.setLayout(new FlowLayout());
        subPane1.add(new JLabel("options     "));
        options_TF = new JTextField(15);
        subPane1.add(options_TF);
        contentPane.add(subPane1);
        subPane2 = new JPanel();
        subPane2.add(new JLabel("arguments"));
        args_TF = new JTextField(15);
        subPane2.add(args_TF);
        contentPane.add(subPane2);
        start_button = new JButton("start");
        contentPane.add(start_button);
        result_TA = new JTextArea(5,10);
        resultPane = new JScrollPane(result_TA);
        contentPane.add(new JLabel("result"));
        contentPane.add(resultPane);
        resultPane.setAutoscrolls(true);
        viewport = new JViewport();
        resultPane.setViewportView(result_TA);
        viewportPos = new Point(0,0);
        viewport.setViewPosition(viewportPos);
        getContentPane().add(contentPane,"Center");
        setSize(250,250);
        // create event listener
        start_button.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                        execute();
                }
        });
}
private void execute() {
        Runtime rt;
        //Properties property;
        rt = Runtime.getRuntime();
        try {
                Process p = rt.exec(JDKLoc + "\\java.exe" +" -classpath "+ currPath + " " +
                        options_TF.getText() +" " +currJavName + " " + args_TF.getText());
                try {
                        p.waitFor();
                        InputStream in = p.getErrorStream();
                        InputStreamReader rin = new InputStreamReader(in);
                        BufferedReader bin = new BufferedReader(rin);
                        String temp;
                        result_TA.setText("");
                        temp = bin.readLine();
                        while (temp != null){
                                result_TA.append(temp+"\n");
                                temp = bin.readLine();
                        }
                        in = p.getInputStream();
                        rin = new InputStreamReader(in);
                        bin = new BufferedReader(rin);
                        temp = bin.readLine();
                        while (temp != null){
                                result_TA.append(temp+"\n");
                                temp = bin.readLine();
                        }
                        viewport.setViewPosition(viewportPos);
                        result_TA.repaint();
                        rt.freeMemory();
                }catch (InterruptedException e){}
        } catch (IOException e) {}
}
```

```
}
```

## ข.6 ซอร์สโค้ดของคลาส DebugPane

```
package JavaIDEX;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.sun.jdi.*;
import java.util.StringTokenizer;
import javax.swing.text.BadLocationException;
public class DebugPane extends JPanel implements Runnable {
        String currJavName = null, currPath = null, connectSpec = null;
        VirtualMachine debuggeeVM = null;
        VMConnection connection;
        TTY tty;
        Thread debugger = null;
        Box contentPane = Box.createVerticalBox();
        JToolBar jToolBar;
        JButton BreakPoint_bt, Step_bt, addWatch_bt, run_bt, next_bt, exit_bt, unbreak_bt;
        JTextArea ThreadWin, AWatchWin, addWatchWin;
        JScrollPane ThreadWinPane, AWatchWinPane, addWatchWinPane;
        JViewport threadViewport, AWatchViewport, addWatchViewport;
        Point viewportPos;
        EditWin eWin;
        protected Rectangle caretCoords;
        int dot;
        int lineAct;
        int oldj=60000;
        final int line2Pixel = 16;
        int bp_hist[];
        int i=1;
        // contructor
        public DebugPane(final EditWin eWin) {
            this.currJavName = eWin.currJavName;
                this.currPath = eWin.currPath;
                this.eWin = eWin;
                bp_hist = new int[50];
                jToolBar = new JToolBar();
                jToolBar.setName("Debug Command");
                BreakPoint_bt = new JButton("break");
                Step_bt = new JButton("step");
                addWatch_bt = new JButton("watch");
                run_bt = new JButton("run");
                next_bt = new JButton("next");
                exit_bt = new JButton("exit");
                unbreak_bt = new JButton("unbreak");
                jToolBar.add(run_bt);
                jToolBar.add(BreakPoint_bt);
                jToolBar.add(unbreak_bt);
                jToolBar.add(Step_bt);
                jToolBar.add(next_bt);
                jToolBar.add(addWatch_bt);
                jToolBar.add(exit_bt);
                BreakPoint_bt.setEnabled(false);
                Step_bt.setEnabled(false);
                addWatch_bt.setEnabled(false);
                run_bt.setEnabled(false);
                next_bt.setEnabled(false);
                exit_bt.setEnabled(false);
                unbreak_bt.setEnabled(false);
                this.setLayout(new BorderLayout());
                add(jToolBar, BorderLayout.NORTH);
                JLabel label1 = new JLabel("Threads    ");
                ThreadWin = new JTextArea(6,2);
                contentPane.add(label1);
                ThreadWin.setEditable(false);
                ThreadWinPane = new JScrollPane(ThreadWin);
                ThreadWinPane.setAutoscrolls(true);
                threadViewport = new JViewport();
                ThreadWinPane.setViewportView(ThreadWin);
                threadViewport = ThreadWinPane.getViewport();
                viewportPos = new Point(0,0);
                viewportPos.setLocation(0,0);
```

```
threadViewport.setViewPosition(viewportPos);
contentPane.add(ThreadWinPane);
JLabel label2 = new JLabel("Local Variables    ");
contentPane.add(label2);
AWatchWin = new JTextArea(6,2);
AWatchWin.setEditable(false);
AWatchWinPane = new JScrollPane(AWatchWin);
AWatchViewport = new JViewport();
AWatchWinPane.setViewportView(AWatchWin);
AWatchViewport = AWatchWinPane.getViewport();
AWatchViewport.setViewPosition(viewportPos);
contentPane.add(AWatchWinPane);
JLabel label3 = new JLabel("Add Watches    ");
contentPane.add(label3);
addWatchWin = new JTextArea(6,8);
addWatchWin.setEditable(false);
addWatchWinPane = new JScrollPane(addWatchWin);
addWatchViewport = new JViewport();
addWatchWinPane.setViewportView(addWatchWin);
addWatchViewport = addWatchWinPane.getViewport();
addWatchViewport.setViewPosition(viewportPos);
contentPane.add(addWatchWinPane);
his.add(contentPane);
// action listeners
exit_bt.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
                StringTokenizer t =null;
                t = new StringTokenizer("quit");
                tty.executeCommand(t);
                stopIt();

        }
});
unbreak_bt.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
                if (caretCoords != null) {
                        lineAct = caretCoords.y + 13 ;
                        lineAct = lineAct / 16;
                        int temp = dot;
                        eWin.dsd.setParagraphAttributes(dot, 1, eWin.attrs[0], true);
                        StringTokenizer t =null;
                        t = new StringTokenizer("clear "+currJavName+":"+lineAct);
                        tty.executeCommand(t);
                }
        }
});
next_bt.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
                StringTokenizer t =null;
                t = new StringTokenizer("next");
                tty.executeCommand(t);
                t = new StringTokenizer("where all");
                tty.executeCommand(t);
                t = new StringTokenizer("locals");
                tty.executeCommand(t);
                t = new StringTokenizer("list");
                tty.executeCommand(t);
                setHighLight();
                threadViewport.setViewPosition(viewportPos);
                AWatchViewport.setViewPosition(viewportPos);
                addWatchViewport.setViewPosition(viewportPos);
                ThreadWinPane.repaint();
                AWatchWinPane.repaint();
                addWatchWinPane.repaint();
        }
});
BreakPoint_bt.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent e) {
                if (caretCoords != null) {
                        lineAct = caretCoords.y + 13 ;
                        lineAct = lineAct / 16;
                        int temp = dot;
                        eWin.dsd.setParagraphAttributes(dot, 1, eWin.attrs[1], true);
                        bp_hist[i++] = dot;
                        StringTokenizer t =null;
```

```
                                        t = new StringTokenizer("stop at"+" " +currJavName+":"+lineAct );
                                        tty.executeCommand(t);
                                }
                        }
                });
                addWatch_bt.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e) {
                                int xx=0;
                                String input = JOptionPane.showInputDialog("add varible
                                        you want to monitor");
                                StringTokenizer t =null;
                                t = new StringTokenizer("print"+" "+input);
                                tty.executeCommand(t);
                        }
                });
                Step_bt.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e) {
                                StringTokenizer t =null;
                                t = new StringTokenizer("step");
                                tty.executeCommand(t);
                                t = new StringTokenizer("where all");
                                tty.executeCommand(t);
                                t = new StringTokenizer("locals");
                                tty.executeCommand(t);
                                t = new StringTokenizer("list");
                                tty.executeCommand(t);
                                setHighLight();
                                viewportPos.setLocation(0,0);
                                AWatchViewport.setViewPosition(viewportPos);
                                addWatchViewport.setViewPosition(viewportPos);
                                threadViewport.setViewPosition(viewportPos);
                                ThreadWinPane.repaint();
                                AWatchWinPane.repaint();
                                addWatchWinPane.repaint();
                        }
                });
                run_bt.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e) {
                                StringTokenizer t =null;
                                t = new StringTokenizer("cont");
                                tty.executeCommand(t);
                                setHighLight();
                                threadViewport.setViewPosition(viewportPos);
                                AWatchViewport.setViewPosition(viewportPos);
                                addWatchViewport.setViewPosition(viewportPos);
                                ThreadWinPane.repaint();
                                AWatchWinPane.repaint();
                                addWatchWinPane.repaint();
                        }
                });
        }

        void setHighLight() {
                ThreadInfo tinfo = ThreadInfo.current;
                StackFrame frame;
                try {
                        frame = tinfo.getCurrentFrame();
                }
                catch (IncompatibleThreadStateException exc) {
                        if (lineno * line2Pixel > eWin.threshold) {
                                int lineSlide = lineno*line2Pixel - eWin.threshold;
                                eWin.dbv_point.setLocation(0,lineSlide+2);
                                eWin.dbp_viewport.setViewPosition(eWin.dbv_point);
                                eWin.scrollPane2.repaint();
                        }
                }
        }
        public void startIt() {
                eWin.textPane.setEditable(false);
                debugger = new Thread(this);
                debugger.start();
                BreakPoint_bt.setEnabled(true);
                Step_bt.setEnabled(true);
                addWatch_bt.setEnabled(true);
                run_bt.setEnabled(true);
```

```
                next_bt.setEnabled(true);
                exit_bt.setEnabled(true);
                unbreak_bt.setEnabled(true);
        }
        public void stopIt() {
                debugger.stop();
                eWin.textPane.setEditable(true);
                BreakPoint_bt.setEnabled(false);
                Step_bt.setEnabled(false);
                addWatch_bt.setEnabled(false);
                run_bt.setEnabled(false);
                next_bt.setEnabled(false);
                exit_bt.setEnabled(false);
                unbreak_bt.setEnabled(false);
                ThreadWin.setText("");
                AWatchWin.setText("");
                addWatchWin.setText("");
                // clear the last step
                eWin.dsd.setParagraphAttributes(oldj, 1, eWin.attrs[0], true);
                // clear all breakpoints
                for (int j = 1; j <= i; j++) {
                        eWin.dsd.setParagraphAttributes(bp_hist[j], 1, eWin.attrs[0], true);
                }
                eWin.splitPane.setDividerLocation(0);
        }
        public void run() {
                try {
                        connectSpec = "com.sun.jdi.CommandLineLaunch:" + "main=" +
                                currJavName + ","+"options=-classpath"+ "+ currPath+"," ;
                        Env.out= System.out;
                        Env.init(connectSpec,true,VirtualMachine.TRACE_NONE);
                        Env.setSourcePath(currPath);
                        tty = new TTY(Env.out, eWin);
                        jToolBar.setEnabled(true);
                        BreakPoint_bt.setEnabled(true);
                        Step_bt.setEnabled(true);
                        addWatch_bt.setEnabled(true);
                        run_bt.setEnabled(true);
                        next_bt.setEnabled(true);
                        exit_bt.setEnabled(true);
                        unbreak_bt.setEnabled(true);
                        ThreadWin.setText("");
                        AWatchWin.setText("");
                        addWatchWin.setText("");

                } catch(Exception e2) {
                        System.out.print("Internal exception: ");
                        System.out.flush();
                        e2.printStackTrace();
                }

        }
}
```

### ข.7 ซอร์สโค้ดของคลาส TTY

ส่วนที่เป็นตัวหนาเอียงคือส่วนที่ทำการเพิ่มเติม แต่ในส่วนที่เป็นเมทธอดเมนนั้น เป็นส่วนที่ไม่ต้องการดังนั้นจึงให้กลายเป็นส่วนคอมเมนต์ เพราะว่าส่วนเมทธอดเมนเป็นส่วนที่รับค่าของ *connectSpec* จากอาร์กิวเมนต์ที่ส่งมา แต่เราไม่ต้องการเพราะเราทำการกำหนดให้ *connectSpec* *="com.sun.jdi.CommandLineLaunch:" + "main=" + currJavName + ","+"options=-classpath"+" "+ currPath+"," ดังที่กำหนดไว้ในคลาส DebugPane

```
Package JavaIDEX;
import com.sun.jdi.*;
import com.sun.jdi.event.*;
import com.sun.jdi.connect.*;
import javax.swing.*;
```

```
import java.util.*;
import java.io.*;

public class TTY implements EventNotifier {

    PrintStream out;
    JTextArea AWatchWin,ThreadWin, messageWin;
    EventHandler handler = null;
    EditWin eWin;

private List monitorCommands = new ArrayList();
    private int monitorCount = 0;
    private static final String progname = "jdb";
    private static final String version = "99/06/11";
    public void vmStartEvent(VMStartEvent se)  {
        Thread.yield();  // fetch output
        out.print("\nVM Started: ");
        otherEvent(se);
    }

    public void breakpointEvent(BreakpointEvent be)  {
        Thread.yield();  // fetch output
        out.print("\nBreakpoint hit: ");
        otherEvent(be);
    }

    public void fieldWatchEvent(WatchpointEvent fwe)  {
        Field field = fwe.field();
        ObjectReference obj = fwe.object();
        Thread.yield();  // fetch output
        out.print("\nField (" + field + ") ");
        if (fwe instanceof ModificationWatchpointEvent) {
            out.print("is ");
            out.print(fwe.valueCurrent());
            out.print(", will be ");
            out.print(((ModificationWatchpointEvent)fwe).valueToBe());
            out.print(": ");
        } else {
            out.print("access encountered: ");
        }
        otherEvent(fwe);
    }

    public void stepEvent(StepEvent se)  {
        Thread.yield();  // fetch output
        Out.print("\nStep completed: ");
        OtherEvent(se);
    }

    public void exceptionEvent(ExceptionEvent ee) {
        Thread.yield();  // fetch output
        Out.print("\nException occurred: ");
        Out.print(ee.exception().referenceType().name());
        Location catchLocation = ee.catchLocation();
        If (catchLocation == null) {
            out.print(" (uncaught) ");
        } else {
            out.print(" (to be caught at: ");
            out.print(Commands.locationString(catchLocation));
            out.print(") ");
        }
        otherEvent(ee);
    }

    public void methodEntryEvent(MethodEntryEvent me) {
        Thread.yield();  // fetch output

        StringBuffer buffer = new StringBuffer("\nMethod Entered: ");
        Buffer.append(me.method().declaringType().name());
        Buffer.append(".");
        Buffer.append(me.method().name());
        Buffer.append(" ");
        out.print(buffer.toString());
        otherEvent(me);
```

```
    }

    public void methodExitEvent(MethodExitEvent me) {
        Thread.yield();  // fetch output

        StringBuffer buffer = new StringBuffer("\nMethod Exited: ");
        buffer.append(me.method().declaringType().name());
        buffer.append(".");
        buffer.append(me.method().name());
        buffer.append(" ");
        out.print(buffer.toString());
        otherEvent(me);
    }

    public void otherEvent(Event event) {
    }

    public void vmInterrupted() {
        Thread.yield();  // fetch output
        printCurrentLocation();
        Iterator it = monitorCommands.iterator();
        while (it.hasNext()) {
            StringTokenizer t = new StringTokenizer((String)it.next());
            t.nextToken();  // get rid of monitor number
            executeCommand(t);
        }
        printPrompt();
    }

    private void printPrompt() {
        Env.printPrompt();
    }

    private void printCurrentLocation() {
        ThreadInfo tinfo = ThreadInfo.current;
        StackFrame frame;
        try {
            frame = tinfo.getCurrentFrame();
        } catch (IncompatibleThreadStateException exc) {
            out.println("current thread isn't suspended anymore?!?");
            return;
        }
        if (frame == null) {
            out.println("No frames on the current call stack");
        } else {
            Location loc = frame.location();
            out.print("thread=\"" + tinfo.thread.name() +"\", ");
            out.print(", ");
            out.println(Commands.locationString(loc));
            if (loc.lineNumber() != -1) {
                String line;
                try {
                    line = Env.sourceLine(loc, loc.lineNumber());
                } catch (java.io.IOException e) {
                    line = null;
                }
                if (line != null) {
                    out.println("  " + loc.lineNumber() + " " + line);
                }
            }
        }
        out.println();
    }

    void help() {
        out.println("** command list **");
        out.println("run [class [args]]        -- start execution of application's main class");
        out.println();
        out.println("threads [threadgroup]     -- list threads");
        out.println("thread <thread id>        -- set default thread");
        out.println("suspend [thread id(s)]    -- suspend threads (default: all)");
        out.println("resume [thread id(s)]     -- resume threads (default: all)");
        out.println("where [thread id] | all   -- dump a thread's stack");
        out.println("wherei [thread id] | all  -- dump a thread's stack, with pc info");
        out.println("up [n frames]             -- move up a thread's stack");
```

```
        out.println("down [n frames]          -- move down a thread's stack");
        out.println("kill <thread> <expr>     -- kill a thread with the given exception object");
        out.println("interrupt <thread>       -- interrupt a thread");
        out.println();
        out.println("print <expr>            -- print value of expression");
        out.println("dump <expr>              -- print all object information");
        out.println("eval <expr>             -- evaluate expression (same as print)");
        out.println("set <lvalue> = <expr>    -- assign new value to field/variable/array element");
        out.println("locals                  -- print all local variables in current stack frame");
        out.println();
        out.println("classes                 -- list currently known classes");
        out.println("class <class id>         -- show details of named class");
        out.println("methods <class id>       -- list a class's methods");
        out.println("fields <class id>        -- list a class's fields");
        out.println();
        out.println("threadgroups            -- list threadgroups");
        out.println("threadgroup <name>       -- set current threadgroup");
        out.println();
        out.println("stop in <class id>.<method>[(argument_type,...)]");
        out.println("                        -- set a breakpoint in a method");
        out.println("stop at <class id>:<line> -- set a breakpoint at a line");
        out.println("clear <class id>.<method>[(argument_type,...)]");
        out.println("                        -- clear a breakpoint in a method");
        out.println("clear <class id>:<line>  -- clear a breakpoint at a line");
        out.println("clear                   -- list breakpoints");
        out.println("catch <class id>         -- break when specified exception thrown");
        out.println("ignore <class id>        -- cancel 'catch' for the specified exception");
        out.println("watch [access|all] <class id> <field name>");
        out.println("                        -- watch access/modifications to a field");
        out.println("unwatch [access|all] <class id> <field name>");
        out.println("                        -- discontinue watching access/modifications to a field");
        out.println("trace methods [thread]   -- trace method entry and exit");
        out.println("untrace methods [thread] -- stop tracing method entry and exit");
        out.println("step                    -- execute current line");
        out.println("step up                 -- execute until the current method returns to its caller");
        out.println("stepi                   -- execute current instruction");
        out.println("next                    -- step one line (step OVER calls)");
        out.println("cont                    -- continue execution from breakpoint");
        out.println();
        out.println("list [line number|method] -- print source code");
        out.println("use (or sourcepath) [source file path]");
        out.println("                        -- display or change the source path");
        out.println("exclude [class id ... | \"none\"]");
        out.println("                        -- do not report step or method events for specified classes");
        out.println("classpath               -- print classpath info from target VM");
        out.println();
        out.println("monitor <command>        -- execute command each time the program stops");
        out.println("monitor                 -- list monitors");
        out.println("unmonitor <monitor#>     -- delete a monitor");
        out.println("read <filename>          -- read and execute a command file");
        out.println();
        out.println("lock <expr>             -- print lock info for an object");
        out.println("threadlocks [thread id]  -- print lock info for a thread");
        out.println();
        out.println("disablegc <expr>         -- prevent garbage collection of an object");
        out.println("enablegc <expr>          -- permit garbage collection of an object");
        out.println();
        out.println("!!                      -- repeat last command");
        out.println("<n> <command>            -- repeat command n times");
        out.println("help (or ?)             -- list commands");
        out.println("version                 -- print version information");
        out.println("exit (or quit)          -- exit debugger");
        out.println();
        out.println("<class id>: full class name with package qualifiers or a ");
        out.println("pattern with a leading or trailing wildcard ('*').");
        out.println("<thread id>: thread number as reported in the 'threads' command");
        out.println("<expr>: a Java(tm) Programming Language expression.");
        out.println("Most common syntax is supported.");
        out.println();
        out.println("Startup commands can be placed in either \"jdb.ini\" or \".jdbrc\"");
        out.println("in user.home or user.dir");
    }

    private String[]  disconnectCmds = {
        "run", "catch", "ignore", "stop", "clear", "watch", "unwatch",
```

```
    "use", "sourcepath", "quit", "exit", "help", "?", "read",
    "version"
};

private boolean isDisconnectCmd(String cmd) {
    for (int i = 0; i < disconnectCmds.length; i++) {
        if (disconnectCmds[i].equals(cmd)) {
            return true;
        }
    }
    return false;
}

void executeCommand(StringTokenizer t) {
    String cmd = t.nextToken().toLowerCase();
    Commands evaluator = new Commands(out, eWin);

        boolean showPrompt = true;

        if (!Env.connection().isOpen() && !isDisconnectCmd(cmd)) {
        out.println("Command '" + cmd + "' is not valid until the VM is started with the 'run' command");
    } else {
        try {
            if (cmd.equals("print")) {
                evaluator.commandPrint(t, false);
                showPrompt = false;       // asynchronous command
            } else if (cmd.equals("eval")) {
                evaluator.commandPrint(t, false);
                showPrompt = false;       // asynchronous command
            } else if (cmd.equals("set")) {
                evaluator.commandSet(t);
                showPrompt = false;       // asynchronous command
            } else if (cmd.equals("dump")) {
                evaluator.commandPrint(t, true);
                showPrompt = false;       // asynchronous command
            } else if (cmd.equals("locals")) {
                evaluator.commandLocals();
            } else if (cmd.equals("classes")) {
                evaluator.commandClasses();
            } else if (cmd.equals("class")) {
                evaluator.commandClass(t);
            } else if (cmd.equals("methods")) {
                evaluator.commandMethods(t);
            } else if (cmd.equals("fields")) {
                evaluator.commandFields(t);
            } else if (cmd.equals("threads")) {
                evaluator.commandThreads(t);
            } else if (cmd.equals("thread")) {
                evaluator.commandThread(t);
            } else if (cmd.equals("suspend")) {
                evaluator.commandSuspend(t);
            } else if (cmd.equals("resume")) {
                evaluator.commandResume(t);
            } else if (cmd.equals("cont")) {
                evaluator.commandCont();
            } else if (cmd.equals("threadgroups")) {
                evaluator.commandThreadGroups();
            } else if (cmd.equals("threadgroup")) {
                evaluator.commandThreadGroup(t);
            } else if (cmd.equals("catch")) {
                evaluator.commandCatchException(t);
            } else if (cmd.equals("ignore")) {
                evaluator.commandIgnoreException(t);
            } else if (cmd.equals("step")) {
                evaluator.commandStep(t);
            } else if (cmd.equals("stepi")) {
                evaluator.commandStepi();
            } else if (cmd.equals("next")) {
                evaluator.commandNext();
            } else if (cmd.equals("kill")) {
                evaluator.commandKill(t);
            } else if (cmd.equals("interrupt")) {
                evaluator.commandInterrupt(t);
            } else if (cmd.equals("trace")) {
                evaluator.commandTrace(t);
```

```
    } else if (cmd.equals("untrace")) {
        evaluator.commandUntrace(t);
    } else if (cmd.equals("where")) {
        evaluator.commandWhere(t, false);
    } else if (cmd.equals("wherei")) {
        evaluator.commandWhere(t, true);
    } else if (cmd.equals("up")) {
        evaluator.commandUp(t);
    } else if (cmd.equals("down")) {
        evaluator.commandDown(t);
    } else if (cmd.equals("load")) {
        evaluator.commandLoad(t);
    } else if (cmd.equals("run")) {
        evaluator.commandRun(t);

        if ((handler == null) && Env.connection().isOpen()) {
            handler = new EventHandler(this, false,eWin);
        }
    } else if (cmd.equals("memory")) {
        evaluator.commandMemory();
    } else if (cmd.equals("gc")) {
        evaluator.commandGC();
    } else if (cmd.equals("stop")) {
        evaluator.commandStop(t);
    } else if (cmd.equals("clear")) {
        evaluator.commandClear(t);
    } else if (cmd.equals("watch")) {
        evaluator.commandWatch(t);
    } else if (cmd.equals("unwatch")) {
        evaluator.commandUnwatch(t);
    } else if (cmd.equals("list")) {
        evaluator.commandList(t);
    } else if (cmd.equals("lines")) {
        evaluator.commandLines(t);
    } else if (cmd.equals("classpath")) {
        evaluator.commandClasspath(t);
    } else if (cmd.equals("use") || cmd.equals("sourcepath")) {
        evaluator.commandUse(t);
    } else if (cmd.equals("monitor")) {
        monitorCommand(t);
    } else if (cmd.equals("unmonitor")) {
        unmonitorCommand(t);
    } else if (cmd.equals("lock")) {
        evaluator.commandLock(t);
        showPrompt = false;       // asynchronous command
    } else if (cmd.equals("threadlocks")) {
        evaluator.commandThreadlocks(t);
    } else if (cmd.equals("disablegc")) {
        evaluator.commandDisableGC(t);
        showPrompt = false;       // asynchronous command
    } else if (cmd.equals("enablegc")) {
        evaluator.commandEnableGC(t);
        showPrompt = false;       // asynchronous command
    } else if (cmd.equals("save")) {
        evaluator.commandSave(t);
        showPrompt = false;       // asynchronous command
    } else if (cmd.equals("bytecodes")) {
        evaluator.commandBytecodes(t);
    } else if (cmd.equals("exclude")) {
        evaluator.commandExclude(t);
    } else if (cmd.equals("read")) {
        readCommand(t);
    } else if (cmd.equals("help") || cmd.equals("?")) {
        help();
    } else if (cmd.equals("version")) {
        evaluator.commandVersion(progname, version, t);
    } else if (cmd.equals("quit") || cmd.equals("exit")) {
        if (handler != null) {
            handler.shutdown();
        }
        Env.shutdown();

    } else {
        if (t.hasMoreTokens()) {
            try {
```

```
    }

boolean readCommandFile(String filename) {
    File f = new File(filename);
    BufferedReader inFile = null;
    try {
      if (f.canRead()) {
        Env.out.println("*** Reading commands from " + f.getCanonicalPath());
        inFile = new BufferedReader(new FileReader(f));
        String ln;
        while ((ln = inFile.readLine()) != null) {
          StringTokenizer t = new StringTokenizer(ln);
          if (t.hasMoreTokens()) {
            executeCommand(t);
          }
        }
      }
    } catch (IOException e) {
    } finally {
      if (inFile != null) {
        try {
          inFile.close();
        } catch (Exception exc) {
        }
      }
    }
    return inFile != null;
}

public TTY(PrintStream out, EditWin eWin) throws Exception {
    System.out.println("Initializing " + progname + "...");
    this.out = out;
    this.eWin = eWin;
    if (Env.connection().isOpen()) {

      this.handler = new EventHandler(this, true,eWin);
    }
    try {

      BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

      String lastLine = null;

      Thread.currentThread().setPriority(Thread.NORM_PRIORITY);


      if (!readCommandFile(System.getProperty("user.home") +
                    File.separator + "jdb.ini")) {
        readCommandFile(System.getProperty("user.home") +
                    File.separator + ".jdbrc");
      }

      if (!readCommandFile(System.getProperty("user.dir") +
                    File.separator + "jdb.ini")) {
        readCommandFile(System.getProperty("user.dir") +
                    File.separator + ".jdbrc");
      }

      printPrompt();
      //while (true) {
        String ln=null;// = in.readLine();
        if (ln == null) {
          out.println("Input stream closed.");
          return;
        }

        if (ln.startsWith("!!") && lastLine != null) {
          ln = lastLine + ln.substring(2);
          out.println(ln);
        }

        StringTokenizer t = new StringTokenizer(ln);
        if (t.hasMoreTokens()) {
          lastLine = ln;
```

```
                VMConnection connection = Env.connection();
            if (!connection.isLaunch()) {
            if (!t.hasMoreTokens()) {
               Env.vm().resume();
            } else {
               out.println("'run <args>' command valid only with launched VMs");
            }
            return;
         }
            executeCommand(t);
               } else {
                  printPrompt();
               }

            //}
         } catch (VMDisconnectedException e) {
            handler.handleDisconnectedException();
         }
      }

      private static void usage() {
         String separator = File.pathSeparator;
         System.out.println("Usage: " + progname + " <options> <class> <arguments>");
         System.out.println();
         System.out.println("where options include:");
         System.out.println("    -help          print out this message and exit");
         System.out.println("    -sourcepath <directories separated by \"" +
                            separator + "\">");
         System.out.println("                   directories in which to look for source files");
         System.out.println("    -attach <address>");
         System.out.println("                   attach to a running VM at the specified address using standard connector");
         System.out.println("    -listen <address>");
         System.out.println("                   wait for a running VM to connect at the specified address using standard
connector");
         System.out.println("    -listenany");
         System.out.println("                   wait for a running VM to connect at any available address using standard
connector");
         System.out.println("    -launch");
         System.out.println("                   launch VM immediately instead of waiting for 'run' command");
         System.out.println("    -connect <connector-name>:<name1>=<value1>,... ");
         System.out.println("                   connect to target VM using named connector with listed argument values");
         System.out.println("    -dbgtrace [flags] print info for debugging " + progname);
         System.out.println("    -thotspot      run the application in the Hotspot(tm) Performance Engine");
         System.out.println("    -tclassic      run the application in the Classic VM");
         System.out.println();
         System.out.println("options forwarded to debuggee process:");
         System.out.println("    -v -verbose[:class|gc|jni]");
         System.out.println("                   turn on verbose mode");
         System.out.println("    -D<name>=<value> set a system property");
         System.out.println("    -classpath <directories separated by \"" +
                            separator + "\">");
         System.out.println("                   list directories in which to look for classes");
         System.out.println("    -X<option>         non-standard target VM option");
         System.out.println();
         System.out.println("<class> is the name of the class to begin debugging");
         System.out.println("<arguments> are the arguments passed to the main() method of <class>");
         System.out.println();
         System.out.println("For command help type 'help' at " + progname + " prompt");
      }

      static void usageError(String message) {
         System.err.println(message);
         System.err.println();
         Usage();
         System.exit(1);
      }

      private static Connector findConnector(String transportName, List availableConnectors) {
         Iterator iter = availableConnectors.iterator();
         While (iter.hasNext()) {
            Connector connector = (Connector)iter.next();
            if (connector.transport().name().equals(transportName)) {
               return connector;
            }
         }
```

```
        // not found
        throw new IllegalArgumentException("Invalid transport name: " + transportName);
    }

    private static boolean supportsSharedMemory() {
        List connectors = Bootstrap.virtualMachineManager().allConnectors();
        Iterator iter = connectors.iterator();
        While (iter.hasNext()) {

            Connector connector = (Connector)iter.next();
            if (connector.transport().name().equals("dt_shmem")) {
                return true;
            }
        }
        return false;
    }

    private static String addressToSocketArgs(String address) {
        int index = address.indexOf(':');
        if (index != -1) {
            String hostString = address.substring(0, index);
            String portString = address.substring(index + 1);
            return "hostname=" + hostString + ",port=" + portString;
        } else {
            return "port=" + address;
        }
    }

/*  public static void main(String argv[]) {
        String cmdLine = "";
        String javaArgs = "";
        Int traceFlags = VirtualMachine.TRACE_NONE;
        boolean launchImmediately = false;
        String connectSpec = null;

        For (int i = 0; i < argv.length; i++) {
            String token = argv[i];
            if (token.equals("-dbgtrace")) {
            if ((i == argv.length - 1) ||
                ! Character.isDigit(argv[i+1].charAt(0))) {
                traceFlags = VirtualMachine.TRACE_ALL;
            } else {
                String flagStr = argv[++i];
                TraceFlags = Integer.decode(flagStr).intValue();
            }
        } else if (token.equals("-X")) {
            usageError("Use 'java -X' to see the available non-standard options");
            return;
        } else if (
                // Standard VM options passed on
                token.equals("-v") || token.startsWith("-v:") || // -v[:...]
                token.startsWith("-verbose") ||                  // -verbose[:...]
                token.startsWith("-D") ||
                // -classpath handled below
                // NonStandard options passed on
                token.startsWith("-X") ||
                // Old-style options (These should remain in place as long as
                //  the standard VM accepts them)
                token.equals("-noasyncgc") || token.equals("-prof") ||
                token.equals("-verify") || token.equals("-noverify") ||
                token.equals("-verifyremote") ||
                token.equals("-verbosegc") ||
                token.startsWith("-ms") || token.startsWith("-mx") ||
                token.startsWith("-ss") || token.startsWith("-oss") ) {

            javaArgs += token + " ";
        } else if (token.equals("-tclassic")) {
            // -classic must be the first one
            javaArgs = "-classic " + javaArgs;
        } else if (token.equals("-thotspot")) {
            // -hotspot must be the first one
            javaArgs = "-hotspot " + javaArgs;
        } else if (token.equals("-sourcepath")) {
            if (i == (argv.length - 1)) {
```

```
            usageError("No sourcepath specified.");
            return;
        }
        Env.setSourcePath(argv[++i]);
    } else if (token.equals("-classpath")) {
        if (i == (argv.length - 1)) {
            usageError("No classpath specified.");
            return;
        }
        javaArgs += token + " " + argv[++i] + " ";
    } else if (token.equals("-attach")) {
        if (connectSpec != null) {
            usageError(token + " can't redefine existing connection");
            return;
        }
        if (i == (argv.length - 1)) {
            usageError("No attach address specified.");
            return;
        }
        String address = argv[++i];

        if (supportsSharedMemory()) {
            connectSpec = "com.sun.jdi.SharedMemoryAttach:name=" +
                          address;
        } else {
            String suboptions = addressToSocketArgs(address);
            connectSpec = "com.sun.jdi.SocketAttach:" + suboptions;
        }
    } else if (token.equals("-listen") || token.equals("-listenany")) {
        if (connectSpec != null) {
            usageError(token + " can't redefine existing connection");
            return;
        }
        String address = null;
        if (token.equals("-listen")) {
            if (i == (argv.length - 1)) {
                usageError("No attach address specified.");
                return;
            }
            address = argv[++i];
        }
        if (supportsSharedMemory()) {
            connectSpec = "com.sun.jdi.SharedMemoryListen";
            if (address != null) {
                connectSpec += (":name=" + address);
            }
        } else {
            String suboptions = addressToSocketArgs(address);
            connectSpec = "com.sun.jdi.SocketListen";
            if (address != null) {
                connectSpec += (":port=" + address);
            }
        }
    } else if (token.equals("-launch")) {
        launchImmediately = true;
    } else if (token.equals("-connect")) {

        if (connectSpec != null) {
            usageError(token + " can't redefine existing connection");
            return;
        }
        if (i == (argv.length - 1)) {
            usageError("No connect specification.");
            return;
        }
        connectSpec = argv[++i];
    } else if (token.equals("-help")) {
        usage();
        System.exit(0);
    } else if (token.equals("-version")) {
        System.out.println(progname + " version " + version);
        System.exit(0);
    } else if (token.startsWith("-")) {
        usageError("invalid option: " + token);
        return;
```

```
        } else {
            // Everything from here is part of the command line
            cmdLine = token + " ";
            for (i++; i < argv.length; i++) {
                cmdLine += argv[i] + " ";
            }
            break;
        }
    }

    if (connectSpec == null) {
        connectSpec = "com.sun.jdi.CommandLineLaunch:";
    }

    cmdLine = cmdLine.trim();
    javaArgs = javaArgs.trim();

    if (cmdLine.length() > 0) {
        if (!connectSpec.startsWith("com.sun.jdi.CommandLineLaunch:")) {
            usageError("Cannot specify command line with connector: " + connectSpec);
            return;
        }
        connectSpec += "main=" + cmdLine + ",";
    }

    if (javaArgs.length() > 0) {
        if (!connectSpec.startsWith("com.sun.jdi.CommandLineLaunch:")) {
            usageError("Cannot specify target VM arguments with connector: " + connectSpec);
            return;
        }
        connectSpec += "options=" + javaArgs + ",";
    }

    try {
        Env.out = System.out;
        Env.init(connectSpec, launchImmediately, traceFlags);
        new TTY(Env.out);
    } catch(Exception e) {
        System.out.print("Internal exception: ");
        System.out.flush();
        e.printStackTrace();
    }
    */
}
```