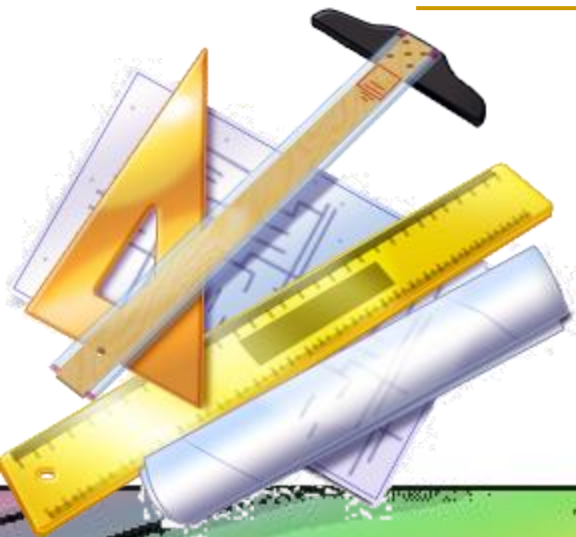


การทดสอบและทบทวน

Lecture 6

เยาวดี เต็มธนาภักดิ์ และ สุกัญญา รัตโนทยานนท์



ความผิดพลาด (Errors)

■ สาเหตุความผิดพลาดอาจแบ่งได้เป็น 3 อย่าง

- ❑ **COMPILE TIME** – ความผิดพลาดที่เกิดในช่วง compile ความผิดพลาดนี้ส่วนใหญ่เกิดจากการเขียนไม่ถูกต้องตามวากยสัมพันธ์ เช่น ไม่ใส่ ; (syntax errors) หรือผิดพลาดในด้านความหมาย เช่นการใช้ค่าโดยไม่ให้ค่าเริ่มต้น (semantic error)
- ❑ **RUN TIME** – ความผิดพลาดที่เกิดในช่วงที่โปรแกรมทำงาน โดยทั่วไป ใน Java จะปรากฏให้เห็นโดยเกิด Java exceptions และโปรแกรมหยุดการทำงาน ตัวอย่างเช่น การหารด้วยศูนย์ การใช้ตัวแปรวัตถุที่เป็น null ให้ทำงาน (NullPointerException)
- ❑ **LOGIC** – ความผิดพลาดในตรรกะของโปรแกรม โดยทั่วไปโปรแกรมทำงานได้เสร็จสิ้นแต่ผลการทำงานไม่เป็นไปตามที่คาดหวังไว้ (ความผิดพลาดนี้อาจนำหรือไม่นำไปสู่ exceptions)

การทดสอบ

■ การทดสอบ S/W เพื่อค้นหาความผิดพลาด

- ❑ การทดสอบจะช่วยให้มีโอกาสหาความผิดพลาดที่อาจยังไม่พบในการทำงานส่วนใหญ่หรือสามารถหาความผิดพลาดได้เจอ
- ❑ เราไม่กล้ารับประกันว่า S/W ไม่ผิดพลาด แต่เราสามารถบอกได้เพียงว่า เรายังไม่พบความผิดพลาดใน S/W

■ การตรวจสอบ (Verification) และความถูกต้อง (Validation)

- ❑ **Verification:** ยืนยันว่า S/W ถูกพัฒนาถูกต้องตามข้อกำหนด (specification)

Are we building the product right?

- ❑ **Validation:** ยืนยันว่า S/W ถูกพัฒนาได้ถูกต้องตามความต้องการ (requirement)

Are we building the right product?

Test Cases

■ แบ่งการทดสอบออกได้เป็น 2 ประเภท

□ **Black Box Testing:** ทดสอบโดยไม่พิจารณารายละเอียดการทำงานภายใน

- สามารถรับ Input ได้ถูกต้อง
- สามารถให้ผลลัพธ์การทำงานที่ถูกต้อง
- ผลลัพธ์ที่ได้เป็นไปตามคาคหมาย

□ **White Box Testing:** ทดสอบโดยพิจารณารายละเอียดการทำงานภายใน

- ทดสอบ logical paths ของการทำงานในทุกเงื่อนไขและทุกจุดที่เป็นไปได้

การทดสอบแต่ละหน่วยและทั้งระบบ

■ Unit Test: การทดสอบหน่วยย่อย

- โดยทดสอบ method หรือกลุ่มของ method ว่าทำงานได้ตามที่ต้องการและถูกต้อง เป็นการทดสอบที่ทำในช่วง Implementation ของแต่ละหน่วยย่อย

■ System Test: การทดสอบรวมทั้งระบบ

- ทดสอบโดยรวมของระบบหลังจากนำหน่วยย่อยมารวมกัน

Unit Tests

- ทดสอบแต่ละ method ให้ถูกต้องก่อนจะนำเข้าไปรวมในโปรแกรม โดยการทดสอบแต่ละ method หรือกลุ่มของ methods
 - Parameters ที่จะต้องใส่ให้กับ methods เพื่อการทดสอบได้จากแหล่งต่าง ๆ เช่น
 - User input
 - ค่าที่เป็นได้ระหว่างช่วงที่กำหนดใน loop
 - ค่าสุ่มที่สร้างขึ้น
 - ค่าจาก file หรือฐานข้อมูล
 - ทดสอบจากกรณีต่าง ๆ ที่ครอบคลุมค่าที่เป็นไปได้ให้มากที่สุด
 - กรณีของค่าที่เป็นไปได้ (Positive Test)
 - กรณีของค่าที่เป็นขอบเขต (Boundary Test)
 - กรณีของค่าที่เป็นไปไม่ได้ (Negative Test)

การดีบั๊ก: กระบวนการหาและแก้ไข bug

- **Reproduce the error:** พยายาม run ซ้ำ เพื่อให้เกิดพฤติกรรมที่ผิดอันเดิม
- **Simplify the error:** หา input ที่ง่ายที่สุดที่ทำให้เกิดความผิดพลาด
- **Divide and Conquer:** แบ่งเป็นส่วนย่อย ๆ ในการหา
 - เพิ่มจุด trace เพื่อแสดงการไหลและค่าที่สำคัญ เช่นพิมพ์ค่าตัวแปรออกมา จากนั้นวิเคราะห์ค่าที่พิมพ์
- **เข้าใจการทำงานของโปรแกรม:** โปรแกรมให้ผลอย่างไร ในขณะที่คาดหวังว่าจะต้องให้ผลอะไร
- **ดูรายละเอียดทั้งหมด:** โดยทั่วไปถึงจุดนี้เรามักตั้งสมมติฐานว่าความผิดพลาดเกิดจากอะไร อย่างไรก็ตามต้องไม่มองข้ามสาเหตุอื่น ๆ ที่อาจเป็นไปได้
- ควรเข้าใจ bug แต่ละอันก่อนที่จะลงมือแก้

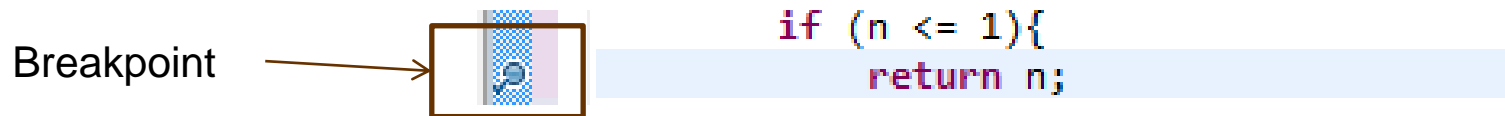
การดีบั๊กโดยใช้ Eclipse Debugger

- **ดีบั๊กเกอร์ (Debugger)** คือเครื่องมือที่ช่วยในการค้นหาข้อผิดพลาดในโปรแกรม โดยมีความสามารถคือ
 - สามารถตั้งจุดให้หยุดชั่วคราว (Breakpoint) เพื่อดูสถานะการทำงานของ
 - บอกรัฐสถานะการทำงานของโปรแกรม เช่น
 - แสดงค่าของตัวแปรต่างๆ และ เฝ้าดู (watch) ค่าของตัวแปรหรือนิพจน์ได้
 - สามารถแก้ไขค่าของตัวแปรได้
 - ควบคุมการทำงานของโปรแกรมเช่นหยุดการทำงาน ทำงานต่อ หรือ สั่งให้โปรแกรมทำงานทีละบรรทัด (Single Step)
 - ไม่ต้องเข้าไปทำงานในเมทอด (Step Over)
 - เข้าไปในเมทอด (Step Into)

การเริ่มใช้งานดีบั๊กเกอร์

■ เริ่มต้นการดีบั๊กใน Eclipse โดย

- สร้างจุดที่ต้องการให้โปรแกรมหยุด (Breakpoint) โดยการ double-click ที่ขอบซ้ายมือของ Editor บนบรรทัดที่ต้องการให้โปรแกรมหยุดทำงาน



- กดปุ่ม 

- เลือกเมนู Run -> Debug หรือ Run -> Debug As -> Java Application

■ โปรแกรมจะทำงานไปจนถึงบรรทัดที่มี Breakpoint และ Eclipse จะขอสลับมุมมอง จากมุมมองจาวาซึ่งใช้ในการเขียนโค้ด ไปยังมุมมองการดีบั๊ก

หน้าจอมุมมองการดีบั๊ก

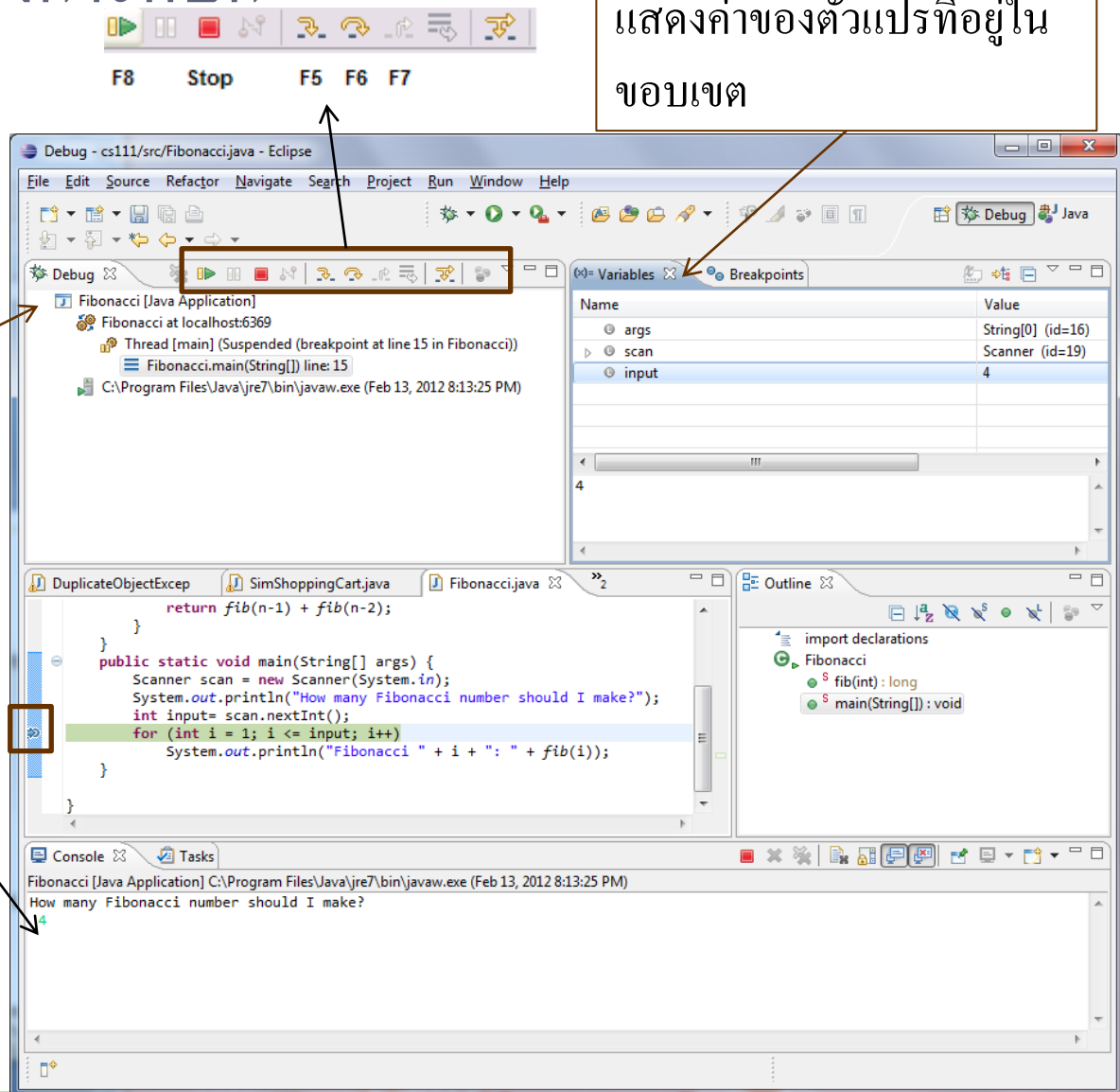
Debug View แสดง stack ของ thread การทำงานของโปรแกรม และมีเครื่องมือที่ใช้ควบคุมการทำงานของโปรแกรม

Console

ใช้ในการแสดงผลและรับอินพุตจากผู้ใช้

Variable View

แสดงค่าของตัวแปรที่อยู่ในขอบเขต



คำสั่งที่ใช้ควบคุมการทำงานของโปรแกรม

- **Resume:** สั่งให้โปรแกรมทำงานต่อจากจุดที่หยุด (จนถึงจุดที่หยุดถัดไป)
- **Suspend:** หยุดการทำงานของโปรแกรมชั่วคราวเพื่อดูสถานะ
 - เหมาะสำหรับดูสถานะในกรณีที่โปรแกรมทำงานนานๆ เช่นใน Loop ที่นาน
- **Terminate:** จบการทำงานของโปรแกรม
- **Step Into:** ให้โปรแกรมทำงานทีละบรรทัดและเข้าไปทำงานในเมทอดถ้าบรรทัดนั้นมีการเรียกเมทอด
- **Step Over:** ให้โปรแกรมทำงานทีละบรรทัดแต่ไม่เข้าไปในเมทอด
- **Step Return:** ให้ทำงานจนจบเมทอดแล้วจึงหยุดพัก
- **Drop to Frame:** ทิ้งผลการทำงานของเมทอดนั้นแล้วเริ่มใหม่

สถานการณ์จำลอง

- อ. ให้การบ้านนศ. เขียนโปรแกรมเพื่อสร้างซีรีส์ของตัวเลข Fibonacci โดยจำนวนตัวเลขในซีรีส์จะถูกกำหนดจากอินพุตของผู้ใช้
- ตัวอย่างการทำงาน

How many Fibonacci number should I make?

4

Fibonacci 1: 1

Fibonacci 2: 1

Fibonacci 3: 2

Fibonacci 4: 3

ตัวอย่างโค้ดที่มีข้อผิดพลาด

```
import java.util.Scanner;

public class Fibonacci {
    public static long fib(int n) {
        if (n < 1){
            return n;
        } else {
            return fib(n-1) + fib(n-2);
        }
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("How many Fibonacci number should I make?");
        int input = scan.nextInt();
        for (int i = 1; i < input; i++)
            System.out.println("Fibonacci " + i + ": " + fib(i));
    }
}
```

ผลการทำงาน

How many Fibonacci number should I make?

4

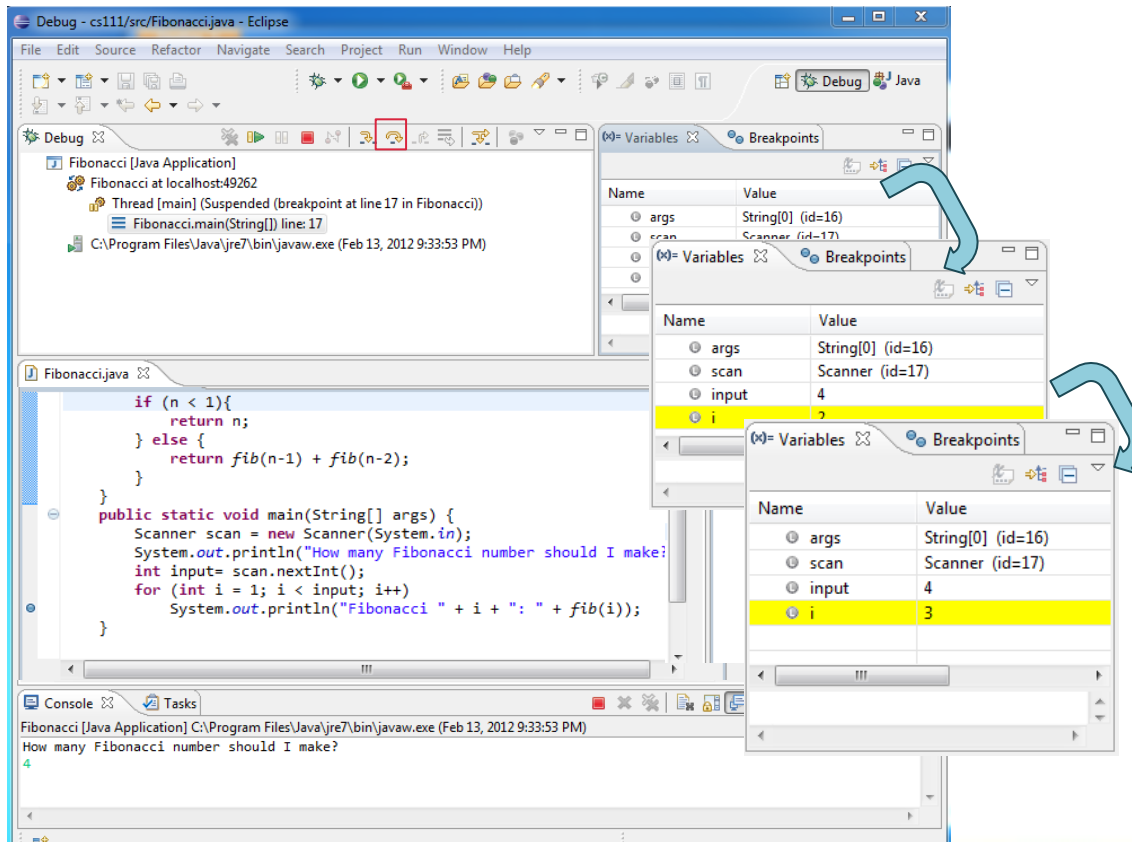
Fibonacci 1: -1

Fibonacci 2: -1

Fibonacci 3: -2

ดื่บักจำนวนตัวเลขที่สร้าง

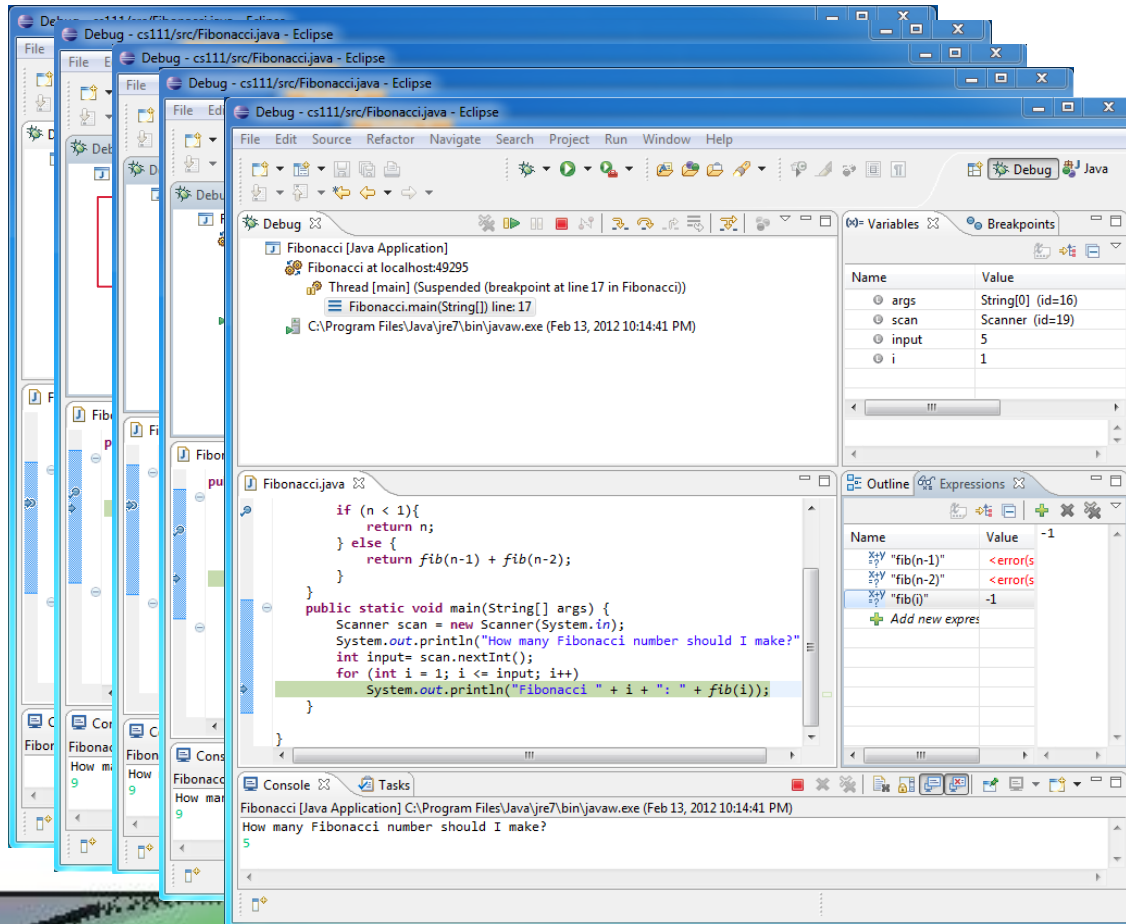
- ทำไมจำนวนตัวเลขถึงไม่ครบ
- วาง Breakpoint ที่บรรทัดที่พิมพ์ตัวเลขและใช้ Step Over



พบว่าเมื่อ $i = 3$ จะออก
จาก for loop เนื่องจาก
เงื่อนไขจบเป็นจริงทำให้
จบการทำงานเร็วไป 1
รอบ

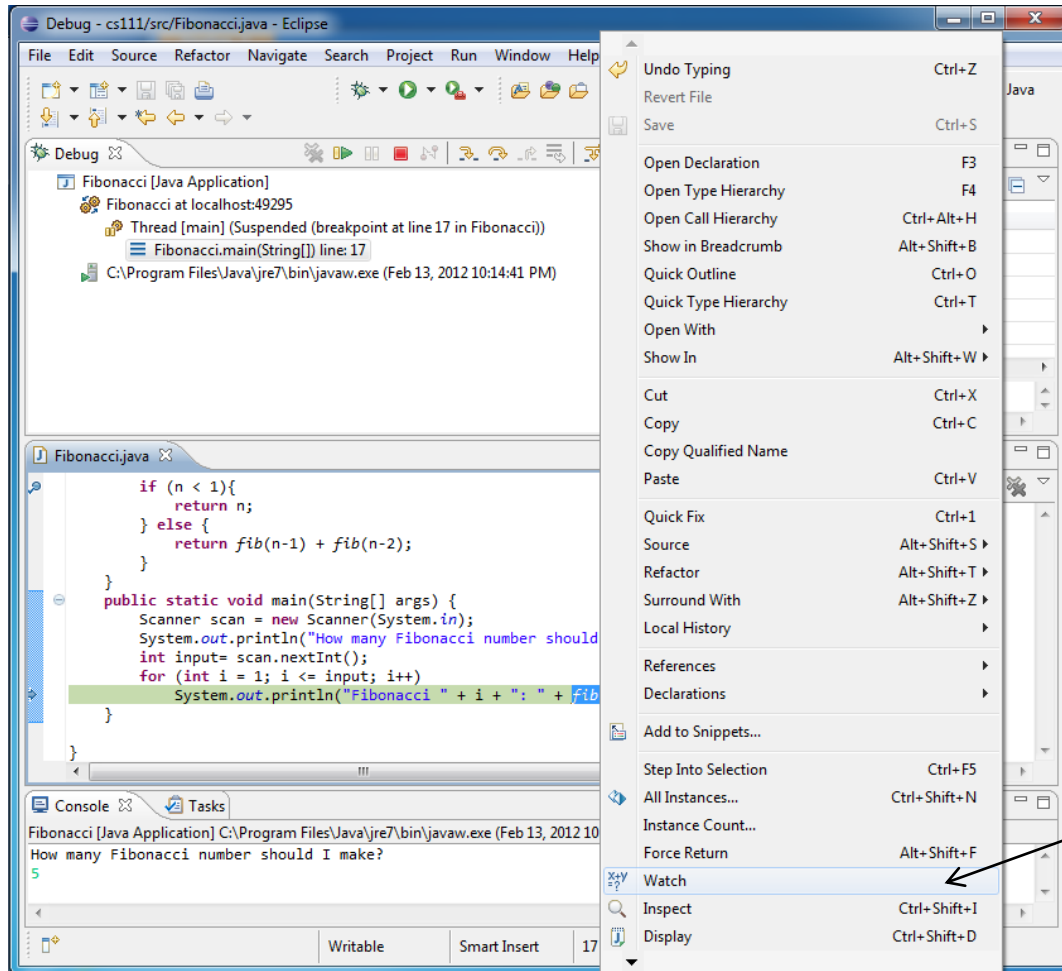
ดีบั๊กค่าของตัวเลข Fibonacci

- ทำไมตัวเลขติดลบ
- ตั้ง Breakpoint ใน fib และใช้ Step Into เพื่อดูการทำงานในแต่ละรอบ

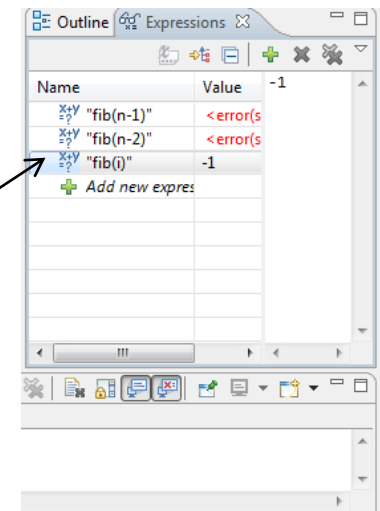


เมื่อ $n = 1$ โปรแกรม
ทำงานผิดพลาดโดย
ให้ค่าเป็นผลบวกของ
 $\text{fib}(0)$ และ $\text{fib}(-1)$

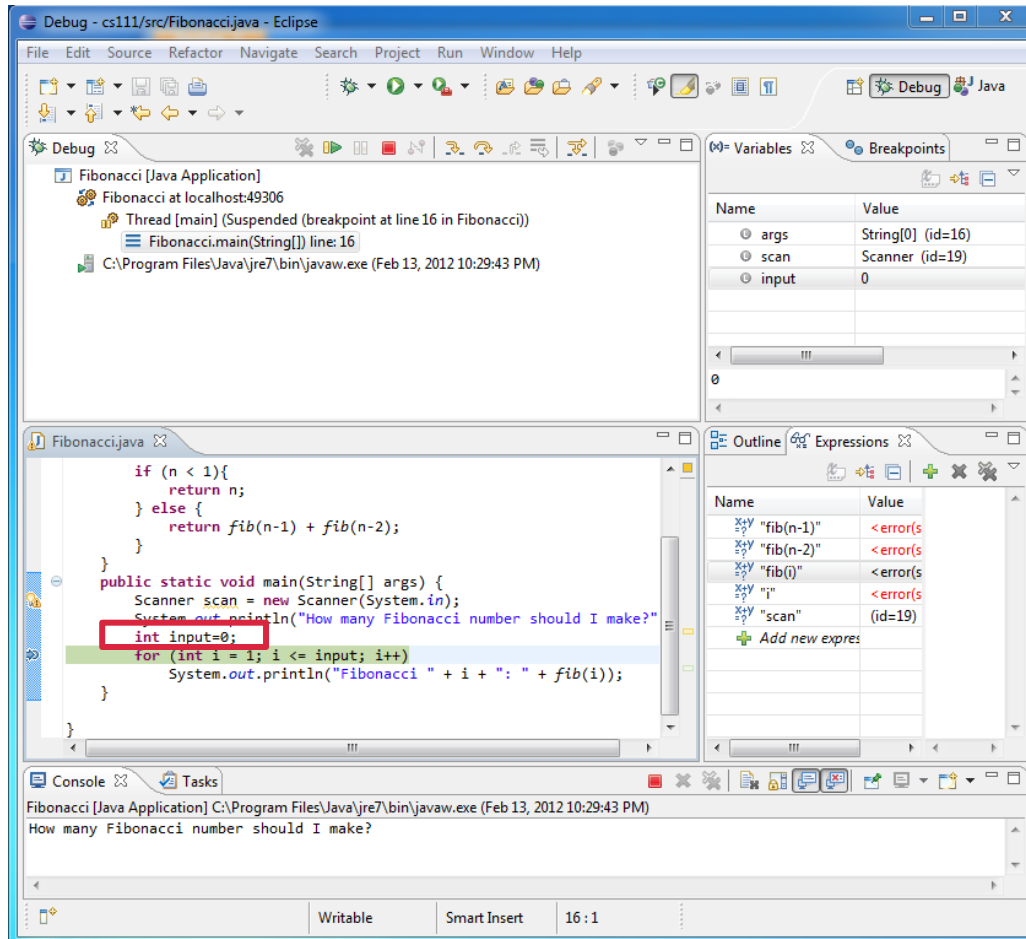
ความสามารถอื่นๆ – การเฝ้าดูค่า



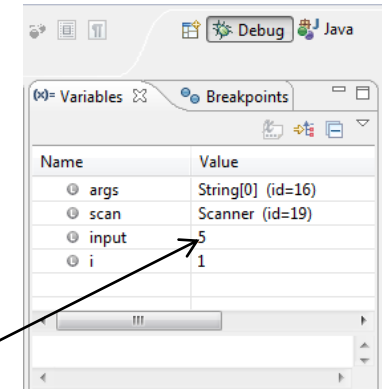
สามารถเฝ้าดูค่าของ
ตัวแปรหรือประโยค
การคำนวณค่าได้ ผล
จะแสดงใน
Expression View



ความสามารถอื่นๆ – การแก้ไขค่าตัวแปร

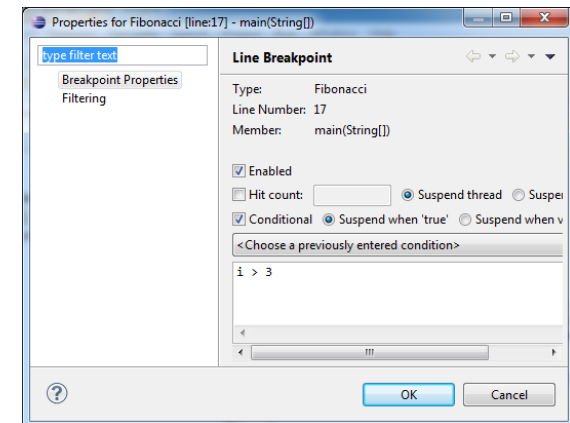
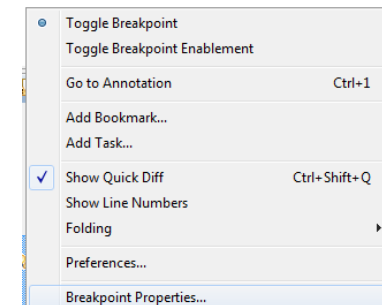


สามารถแก้ไขค่า
ของตัวแปรได้
โดยการพิมพ์ค่า
ในช่อง Value
ของ Variable
View



คุณสมบัติของ Breakpoints

- เราสามารถตั้งให้โปรแกรมหยุดทำงานตามเงื่อนไขต่างๆ ได้ด้วยการตั้งคุณสมบัติของ Breakpoint
- Line Breakpoint
 - ☐ ให้หยุดเมื่อทำงานครบตามจำนวนครั้ง
 - ☐ ให้หยุดเมื่อเงื่อนไขเป็นจริง
- Method Breakpoint
 - ☐ ให้หยุดเมื่อเข้าหรือออกจากเมทอด
- Breakpoint ที่ instance variable (Watchpoint)
 - ☐ หยุดเมื่อตัวแปรนั้นโดนอ่าน
 - ☐ หยุดเมื่อตัวแปรนั้นโดนเปลี่ยนค่า



เกมเก็บขุมทรัพย์

เกมนี้แต่ละรอบผู้เล่นจะวาร์ป (warp) ไปปรากฏในตำแหน่งแบบสุ่ม (ตั้งแต่ 0 ถึง 20 ทั้งแนวแกน x และ y) ในเกมมีสัตว์ประหลาดเฝ้าขุมทรัพย์อยู่ ขุมทรัพย์จะอยู่ที่เดียวกับตำแหน่งของสัตว์ประหลาด ถ้าสัตว์ประหลาดนอนหลับจะเดินเข้าไปเก็บขุมทรัพย์ได้ แต่ถ้าสัตว์ประหลาดตื่นอยู่และผู้เล่นอยู่ในระยะที่มันมองเห็น (สมมติให้เป็น 5 หน่วยของระยะทาง) ผู้เล่นจะโดนกินและเกมจบ ในแต่ละรอบ ถ้าโชคร้ายไปปรากฏในตำแหน่งเดียวกับสัตว์ประหลาดและมันตื่นอยู่ ผู้เล่นโดนกินและเกมจบทันที ถ้าไม่เช่นนั้นผู้เล่นมีโอกาสนี้ครั้งหนึ่งในการเดินในทิศทางตามแนวตั้งหรือแนวนอนตามระยะที่ต้องการ (ระยะเป็น + หรือ - ก็ได้) เพื่อหนีสัตว์ประหลาดหรือเข้าไปให้ใกล้มากพอ (ไม่เกิน 2 หน่วยระยะทาง) ที่จะเก็บสมบัติได้ ถ้าเกมยังไม่จบผู้เล่นจะถูกวาร์ปไปในตำแหน่งใหม่แบบสุ่มเพื่อเผชิญหน้ากับสัตว์ประหลาดเช่นนี้ไปเรื่อย ๆ เมื่อเกมจบผู้เล่นสามารถเลือกที่จะเล่นต่อหรือเลิกเล่นได้

ทางแก้ปัญห

สร้างผู้เล่น สัตว์ประหลาด และตัวสุ่มตัวเลข

do{

 เล่นเกม (play game)

}while (ผู้เล่นยังต้องการเล่นอีก)

การเล่นเกม (play game)

กำหนดค่าเริ่มต้นของคะแนนสมบัติเป็น 0

do{

 เลือกตำแหน่ง x,y ให้ผู้เล่นและสัตว์ประหลาด (generate random positions) อย่างสุ่ม

 เลือกสถานการณ์หลักของสัตว์ประหลาดอย่างสุ่ม (1 แทนหลับ 0 แทนตื่น)

 พิมพ์ตำแหน่งปัจจุบันของผู้เล่น ตำแหน่งและสถานะของสัตว์ประหลาด

 ตรวจสอบตำแหน่งผู้เล่นและสัตว์ประหลาดเป็นตำแหน่งเดียวกันหรือไม่

 ถ้าไม่ใช่

 รับค่าทิศในการเดินจากผู้เล่น (0 แทนแนวนอน 1 แทนแนวตั้ง)

 รับค่าระยะทางที่ต้องการเดิน

 คำนวณระยะห่างจากสัตว์ประหลาด

 ถ้าอยู่ในระยะที่สัตว์ประหลาดมองเห็นและมันตื่นอยู่

 จบเกม

 ไม่เช่นนั้น ถ้าสัตว์ประหลาดนอนหลับและผู้เล่นยืนอยู่ในระยะที่เก็บสมบัติได้

 เพิ่มคะแนนสมบัติขึ้น 1

 ไม่เช่นนั้น จบเกม

}while(เกมยังไม่จบ)

ตัวอย่างการทำงาน

Monster: x: 5 y: 12 Awakening

Player: x: 2 y: 3 Treasure: 0

Enter direction (0=horizontal, 1=vertical): 2

You chose the non-walkable direction.

Monster: x: 4 y: 17 Sleeping

Player: x: 3 y: 8 Treasure: 0

Enter direction (0=horizontal, 1=vertical): 1

Enter distance: 9

You chose vertical walk

Do you like to see your position (1=Yes)?1

My position is at x: 3 and y: 17

Monster: x: 16 y: 10 Awakening

Player: x: 19 y: 9 Treasure: 1

Enter direction (0=horizontal, 1=vertical): 0

Enter distance: -5

You chose horizontal walk

Do you like to see your position (1=Yes)?1

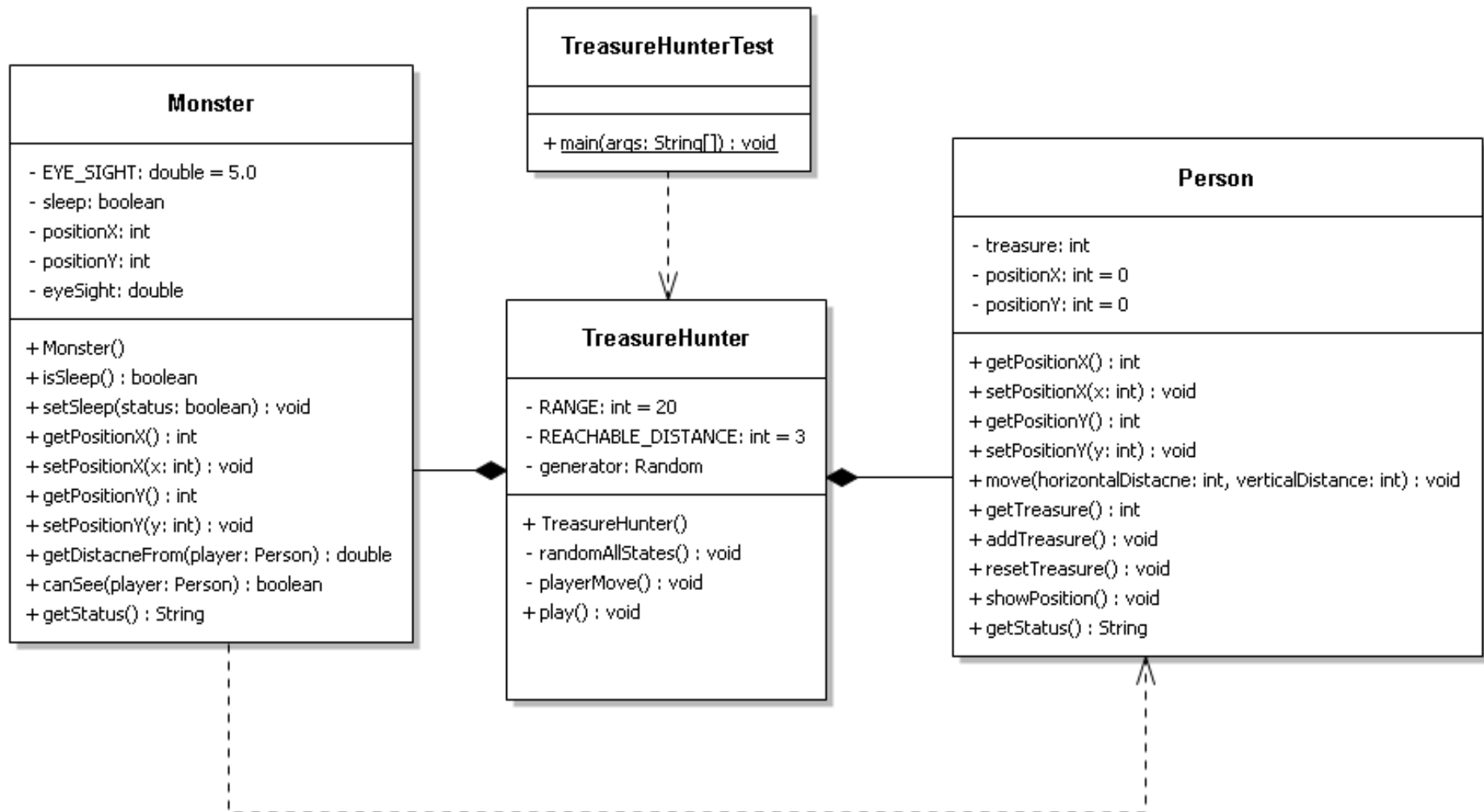
My position is at x: 14 and y: 9

Your total treasure: 1

Do you want to play again (1=Yes)?: 0

Good bye.

Class diagram



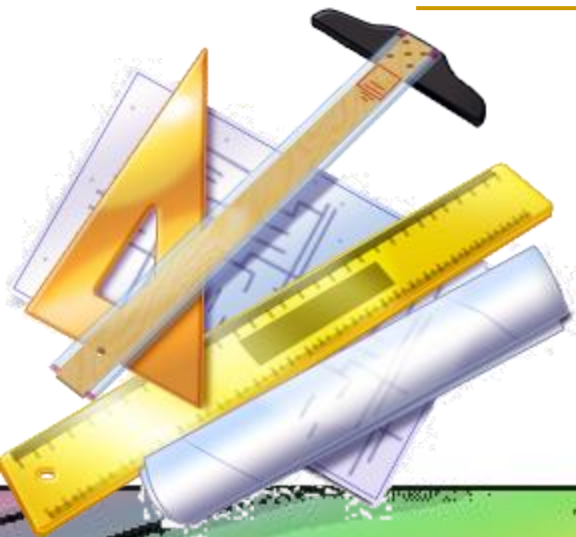
สรุปการเรียนรู้วันนี้

- เพื่อเรียนรู้และเข้าใจแนวคิดเกี่ยวกับการทดสอบ
- เพื่อเรียนรู้การดีบั๊กโดยใช้ Eclipse

ทบทวน

Lecture 6

เขาวดี เต็มธนาภักดิ์



คลาส

- **คลาสแทนหนึ่งแนวคิด/สิ่ง (single concept)** ในปัญหา
 - ชื่อของคลาสควรเป็น**คำนาม**ที่อธิบายถึงหนึ่งแนวคิด/สิ่งในปัญหา
- **คลาสทดสอบ**ในจาวา: เป็นคลาสที่มีเมทอด `main` เพื่อให้สามารถรันได้
- **การวิเคราะห์:** การหาคลาส
 - หาคำนามที่เกี่ยวข้องกับปัญหา → คลาส
 - กำหนดหน้าที่ความรับผิดชอบ → ความสามารถของวัตถุ
 - กำหนดสิ่งที่คลาสต้องมี เพื่อจำสถานะและช่วยให้วัตถุทำงานได้ → Attribute
 - กำหนดความสัมพันธ์กับวัตถุอื่น → ความสัมพันธ์

คลาส

■ คลาสประกอบด้วย

□ ลักษณะ (Attributes)

- ตัวแปรวัตถุ (Instance variable), ตัวแปรคลาส (Static variable), ตัวแปรค่าคงที่

□ ความสามารถ (Methods)

- เมทอดของวัตถุ และเมทอดของคลาส (Static method)

■ คลาสที่ดี ควรออกแบบให้มี Cohesion สูง และ Coupling ต่ำ

การค้นหาคلاسจากปัญหา (1)

- คลาสแทนหนึ่งแนวคิด/สิ่ง (single concept) ในปัญหา
- ชื่อของคลาสควรเป็นคำนามที่อธิบายถึงหนึ่งแนวคิด/สิ่งในปัญหา
- แนวคิด/สิ่งในชีวิตประจำวัน
 - Player
 - Room
- แนวคิด/สิ่งในเรขาคณิต
 - Point
 - Rectangle
 - Polygon

การค้นหาคلاسจากปัญหา (2)

- **Actors** (ลงท้ายด้วย -er, -or) – วัตถุที่ทำงานบางอย่างให้เรา เช่น:
 - Scanner
 - Random
- **Utility classes** – ไม่มีวัตถุ มีเพียง static methods และค่าคงที่:
 - Math
- **คลาสทดสอบ**: มีเพียงเมธอด `main`
- ไม่ควรพยายามทำ actions ให้เป็นคลาส
 - ชื่อเช่น Paycheck เหมาะสมมากกว่า ComputePaycheck

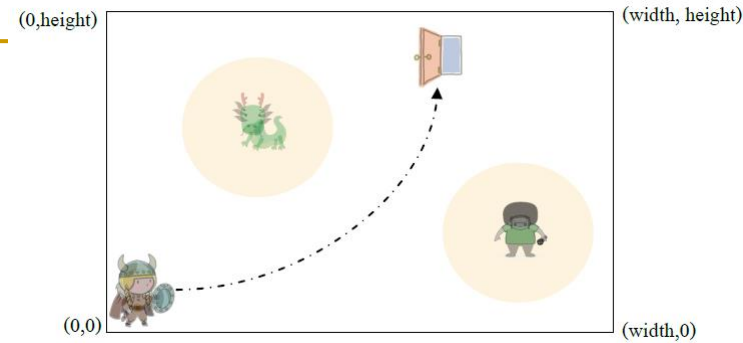
คลาสGame สำหรับเกมผจญภัย

Game

-player: Person
-mon1: Monster
-mon2: Monster
-width: int
-length: int
-exitX: int
-exitY: int

+init(): void
-inbound(int positionX, int positionY): boolean
+play(): void

```
public class Game {  
    private Person player;  
    private Monster mon1;  
    private Monster mon2;  
  
    private int width;  
    private int length;  
    private int exitX;  
    private int exitY;
```



ในการเล่นแต่ละครั้ง

- หอยลูกเต๋าเพื่อหาระยะทางในการเดิน
- เลือกทิศที่จะเดิน
- ตรวจสอบว่าเดินได้หรือไม่
- ถึงทางออกแล้วหรือไม่
- สัตว์ประหลาดมองเห็นหรือไม่

```
public void init() {  
    /* ค่าเริ่มต้นให้กับขนาดของห้อง ตำแหน่งสัตว์ประหลาด และตำแหน่งของประตู */  
}  
private boolean inbound(int positionX, int positionY) {  
    /* ตรวจสอบว่าอยู่ในขอบเขตของห้องหรือไม่ */  
}  
public void play() {  
    }
```

Cohesion

■ คลาสควรแทนหนึ่งแนวคิด/สิ่ง

□ ส่วนต่อประสานเชื่อมกับภายนอกของคลาส ถือว่ามี cohesion ถ้า *ทุก features* *เกี่ยวข้องกับแนวคิด/สิ่งที่คลาสแทนอยู่*

□ ในคลาสGame

■ แนวคิดของกลไกของเกมที่จะมีคนเดินตามจำนวนลูกเต๋าที่ทอยได้เพื่อหาทางออก

■ แนวคิดของห้องเกี่ยวกับตำแหน่งของสัตว์ประหลาด ทางออก ขนาดของห้อง และขอบเขตที่คนจะเดินได้

■ ปรับ Game ให้มีจำนวนห้องมากกว่า 1 ห้อง

```
public class Game {  
    private int room1_width;  
    private int room1_length;  
    private int room1_exitX;  
    private int room1_exitY;  
  
    private int room2_width;  
    private int room2_length;  
    private int room2_exitX;  
    private int room2_exitY;  
    . . .  
    public void init() { ... }  
    private boolean inbound(int positionX, int positionY) { ... }  
  
    ... }
```

ปัญหา หากจะแก้ไขเงื่อนไขเช่นจำนวน
สัตว์ประหลาด หรือเพิ่มจำนวนห้อง
คลาสตัวอย่างนี้ขาด *cohesion*
(> 1 concepts)

Cohesion

- Game เกี่ยวข้องกับ 2 สิ่ง/แนวคิด: *เกม* และ *ห้อง*

- การแก้ไข: สร้าง 2 คลาส

```
public class Room {  
    private int width;  
    private int length;  
    private int exitX;  
    private int exitY;  
    public Room(int width, int length) { ... }  
    public boolean inbound(int positionX, int positionY) { ... }  
    ...  
}
```

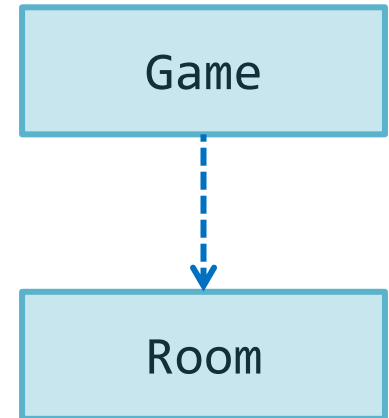
```
public class Game {  
    Private Room room;  
    . . .  
    public void init() { ... room = new Room(. . .);}  
    public void play() { ... }  
    ...  
}
```

Game ไม่จำเป็นต้องรู้รายละเอียดของ
ห้อง สามารถเรียกใช้บริการที่คลาส
Room เพื่อกำหนดค่าเริ่มต้น ตรวจสอบ
การเคลื่อนที่ และ ทางออก ได้

Coupling

- **Dependency**: คลาสหนึ่งขึ้นอยู่กับอีกคลาสหนึ่ง ถ้าใช้วัตถุของอีกคลาสนั้น

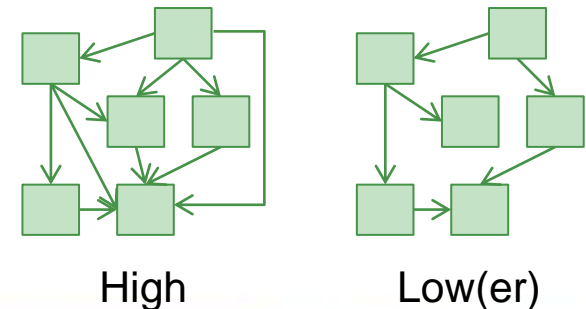
- Game ขึ้นกับ Room เพื่อจะรู้ตำแหน่งของทางออกและสัตว์ประหลาด
- Room ไม่ขึ้นกับ Game



- **Coupling สูง** = มี dependencies มากในระหว่างคลาส

- **Minimize coupling** เพื่อลดผลกระทบจากการเปลี่ยนแปลงส่วนต่อประสาน (interface)

- ดูได้จากความสัมพันธ์เมื่อวาด class diagrams



Encapsulation

■ **Abstract Data Type** (ชนิดผู้ใช้กำหนด) + **Information Hiding** (การซ่อนข้อมูล)

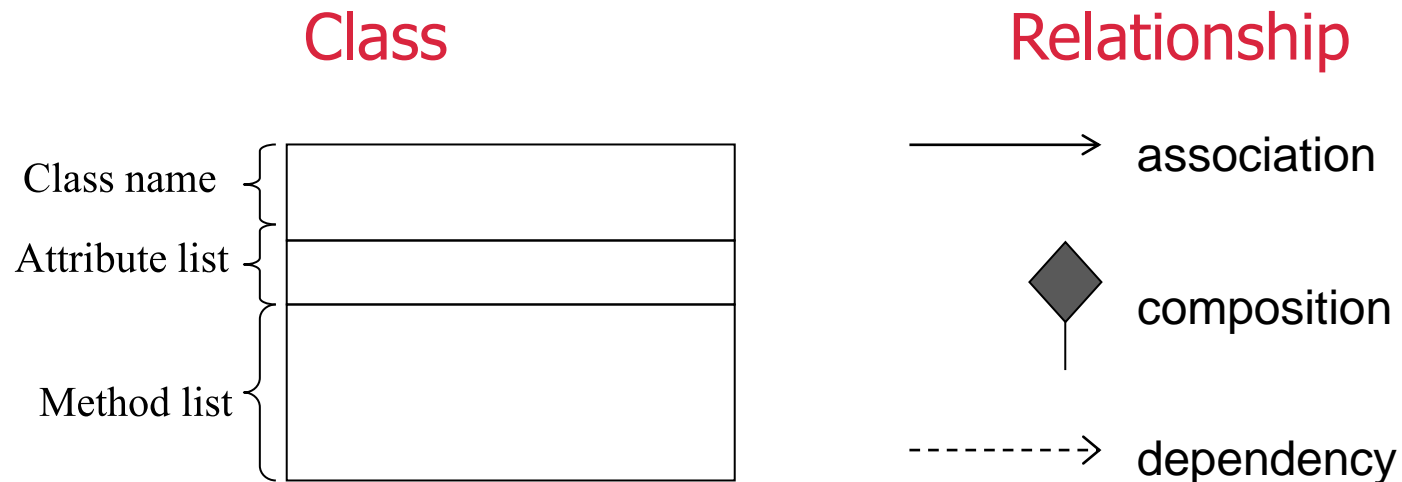
- ❑ ประกาศคลาส: เพื่อสร้าง ADT
- ❑ ให้ลักษณะหรือ/และเมทอดที่เห็นไม่ได้จากภายนอกเป็น private
- ❑ ให้ส่วนการใช้งาน (ทั่วไปคือเมทอด) เชื่อมต่อกับภายนอกเป็น public

■ ตัวอย่าง

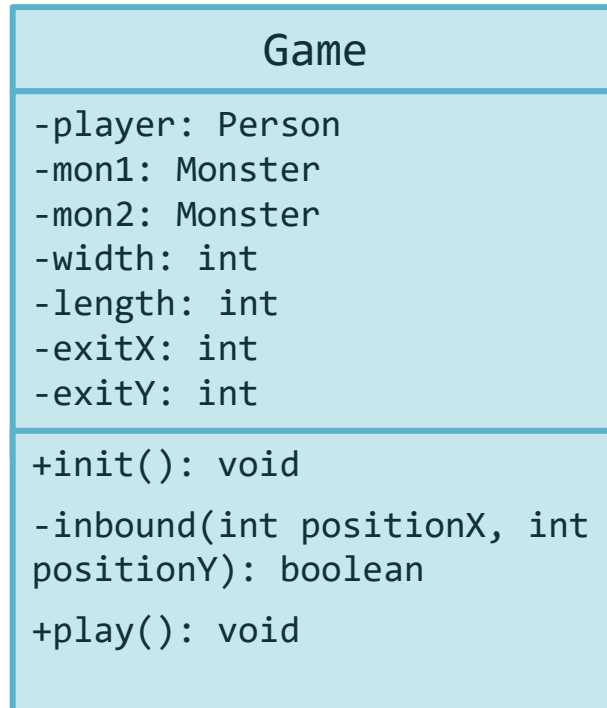
- ❑ ประกาศคลาส
- ❑ กำหนดตัวแปรวัตถุ (instance variables) ให้เป็น private
- ❑ กำหนดให้มี *getter* และ *setter* ที่เป็น public เพื่อเข้าถึงตัวแปรวัตถุเหล่านั้น

Class Diagram

- **Class Diagram** ช่วยในการออกแบบและช่วยให้เข้าใจความสัมพันธ์ระหว่างคลาส



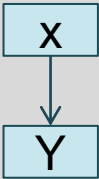
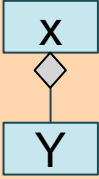
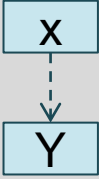
Class Diagram - Example



สรุปการแปลงไคอะแกรมเป็นโค้ด

ไคอะแกรม	จาวา	ตัวอย่าง
ชื่อคลาส	คลาส	Game → <code>public class Game { }</code>
+/-	public/private	<code>+init():void</code> → <code>public void init() { }</code>
Attribute	ตัวแปรวัตถุ	<code>-player:Person</code> → <code>private Person player;</code>
Method	เมทอด	<code>+play():void</code> → <code>public void play() { }</code>
= ค่าคงที่	final	<code>-VALUE:int = 2</code> → <code>private final int VALUE=2;</code>
จีดเส้นใต้	static	<code>-<u>lastNo</u>:int</code> → <code>private static int lastNo;</code>

สรุปการแปลงไคอะแกรมเป็นโค้ด

ไคอะแกรม	จาวา	ตัวอย่าง
ความสัมพันธ์ (Association)	ตัวแปรวัตถุ (ชนิดคลาสที่ สัมพันธ์ด้วย)	 <pre>→ public class X { private Y yObject; ... }</pre>
องค์ประกอบ (Composition)	ตัวแปรวัตถุ (ชนิดคลาสที่ สัมพันธ์ด้วย)	 <pre>→ public class X { private Y yObject; public X() { yObject = new Y(); } }</pre>
Dependency	พารามิเตอร์หรือ ตัวแปรท้องถิ่น ในเมทอด	 <pre>→ public class X { public void someMethod(Y y) { ... } }</pre>

วากยสัมพันธ์พื้นฐาน

- **ชื่อ Identifier:** ลำดับของตัวอักษร, ตัวเลข, \$ และ _ โดยที่ตัวแรกต้องไม่เป็นตัวเลข และชื่อต้องไม่ตรงกับคำสงวน (Reserved Word)
- ชนิดข้อมูลพื้นฐาน: `byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`
- Syntax:

- การประกาศคลาส

```
public class ClassName { ... }
```

- การประกาศตัวแปร

```
dataType variableList;
```

- การประกาศเมทอด

```
modifiers returnType methodName (parameterList) {  
    statements  
}
```

คำสั่งการควบคุมการไหล: ทางเลือก

■ Syntax

```
if ( <boolean expression> ) {  
    <then block statements>  
}  
else { // else block อาจมีหรือไม่ก็ได้  
    <else block statements>  
}
```

```
switch ( <arithmetic expression> ) {  
    case <label 1> : <case body 1>  
        ...  
    case <label n> : <case body n>  
        default : <default body>  
}
```


คำสั่งการควบคุมการไหล: การทำซ้ำ

■ Syntax

```
while ( <boolean expression> ) {  
    <statements>  
}
```

```
for (<initial>; <condition>; <update>) {  
    <statements>  
}
```

```
do {  
    <statements>  
} while ( <boolean expression> );
```

ตัวแปรในภาษาจาวา

- อายุหรือขอบเขตของตัวแปร ขึ้นกับ block ที่ครอบตัวแปรนั้น ๆ
 - ตัวแปรของวัตถุ
 - ตัวแปร static
 - ตัวแปร parameter
 - ตัวแปร local

```
public class Monster {  
    private int positionX, positionY; // instance variable  
    private static int lastAssignedNo // static variable  
    public double getDistanceFrom(int posX, int posY) {  
        // posX and posY are parameter variable  
        int xDiff= posX - getPositionX(); // local variable  
        .  
        .  
        .  
    } // scope of local and parameter variable end here  
}
```

Overloaded Method

- *ลายเซ็นของเมทอด (Method signature):* ส่วนเชื่อมต่อสำหรับการส่งสาร กำหนดจากชื่อเมทอด และพารามิเตอร์ของเมทอด
- *Overloaded Method* เมทอดที่ใช้ชื่อเดียวกันแต่มีพารามิเตอร์ที่ต่างกัน (ชนิด ลำดับ) เพื่อช่วยการโปรแกรมให้สะดวกและง่ายต่อการนำไปใช้
 - การทำ Overloading ไม่คำนึงถึง return type ของลายเซ็นของเมทอด

```
double getDistanceFrom(Monster m)
double getDistanceFrom(int positionX, int positionY)
```

เมทอดตัวสร้าง (Constructors)

- เมทอดพิเศษ ใช้สำหรับกำหนดค่าเริ่มต้นเมื่อสร้างวัตถุของคลาสนั้น
- ชื่อเมทอดต้องเหมือนกับชื่อคลาส
- ใช้ **new** เพื่อส่ง message ไปยัง constructor เพื่อให้เกิดการทำงาน
- Syntax:

```
public <class name> (<parameter>) {  
  
}
```

ชนิดของ Constructors

■ Constructor แบบไม่มีพารามิเตอร์ : ทั่วไปมีให้โดยปริยาย

```
public Monster( ) { <statements> }
```

- ❑ เป็น default constructor ที่ถูกกำหนดโดยนัยภายในคลาส หากไม่มีการกำหนด constructor อื่นใด -- สามารถเรียกใช้ได้โดยไม่ต้องประกาศ
- ❑ ถ้ากำหนด constructor อื่นไว้ ในคลาสต้องประกาศ constructor แบบนี้เอง

■ Constructor อื่น – ทำ Overloading

- ❑ สามารถประกาศ constructor ได้ > 1 constructor แต่ละ constructor ต้องมี parameter list ที่แตกต่างกัน (จำนวน และ/หรือ data type)

โครงสร้างแบบกลุ่ม

■ การประกาศตัวแปรและการสร้างวัตถุ

- ❑ **อาร์เรย์:** เก็บได้ทั้งกลุ่มของวัตถุและข้อมูลชนิดพื้นฐาน
`DataType[] variableName;`
`variableName = new DataType[size];`
- ❑ การเพิ่มสมาชิก กรณีนีชนิดพื้นฐาน (Primitive Data Type)
`variableName[index] = value;`
- ❑ การเพิ่มสมาชิก ต้องมีวัตถุเพื่อให้อ้างถึง (จึงอาจต้องสร้างวัตถุด้วย)
`variableName[index] = new DataType();`
- ❑ การหาขนาด
`variableName.length`
- ❑ ต้องบริหารจัดการ การในเรียงตำแหน่งขนาด และการเพิ่ม-ลดขนาดเอง

โครงสร้างแบบกลุ่ม

■ การประกาศตัวแปรและการสร้างวัตถุ

- ❑ **ArrayList**: เก็บกลุ่มของวัตถุเท่านั้น

```
ArrayList<DataType> variableName;  
variableName = new ArrayList<DataType>();
```

- ❑ การเพิ่มสมาชิก ต้องมีวัตถุเพื่อให้อ้างอิง (จึงอาจต้องสร้างวัตถุด้วย)

```
variableName.add(new DataType());  
variableName.add(index, new DataType());
```

- ❑ การลบสมาชิก

```
variableName.remove(index);
```

- ❑ การหาขนาด

```
variableName.size();
```

API (1)

■ Math

□ เมทีอด: sqrt, abs, ceil, pow, sin, cos, tan

■ Random

□ เมทีอด: constructor, nextInt, nextDouble

■ Scanner

□ เมทีอด: constructor, nextInt, nextDouble, nextLine

API (2)

■ String

- เมท็อด: charAt, equals, equalsIgnoreCase, compare

■ StringTokenizer

- เมท็อด: constructor, hasMoreTokens, nextToken, countTokens

สรุปการเรียนรู้วันนี้

- ทบทวนเนื้อหาในครั้งแรก
 - แนวคิดเชิงวัตถุและการออกแบบ
 - Encapsulation
 - วากยสัมพันธ์พื้นฐานของภาษาจาวา
 - Scope ของตัวแปร
 - Overloading methods
 - อาร์เรย์และอาร์เรย์ลิสต์
 - API ที่ควรรู้จัก