



## โครงการ

Numerical Method  
Calculator Website

## จัดทำโดย

6504062620175 นายอริชา เล็กสรรเสริญ

## เสนอ

ผู้ช่วยศาสตราจารย์สถิต ประสมพันธ์

**วิชา 040613204 Object-Oriented Programming**

ภาคเรียนที่ 1 ปีการศึกษา 2566

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ คณะวิทยาศาสตร์ประยุกต์  
มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

## บทที่ 1 ที่มาและความสำคัญของโครงการ

โครงการนี้จัดขึ้นเพื่อวัดผลความสามารถในการเรียนวิชา Object Oriented Programming โดยการนำเรื่องที่เรียนมาสร้างเป็นชิ้นงานในรูปแบบเว็บ โดยใช้แนวคิดการเขียนโปรแกรมแบบเชิงวัตถุ และยังช่วยให้ผู้จัดทำเรียนรู้อุปกรณ์และเครื่องมือ ผู้จัดทำได้สร้างเว็บนี้ขึ้นมา

### ประเภทของโครงการ

โปรแกรมเว็บแอปพลิเคชัน Full-Stack

### ประโยชน์

1. เพื่อให้สามารถคำนวณปัญหาทาง Numerical ได้อย่างสะดวกสบาย
2. เพื่อนำความรู้จากวิชา Database, Numerical Methods, Object-Oriented Programming มาประยุกต์ใช้
3. เพื่อนำแนวคิดการเขียนโปรแกรมแบบเชิงวัตถุมาประยุกต์ใช้

### ขอบเขตของโครงการ

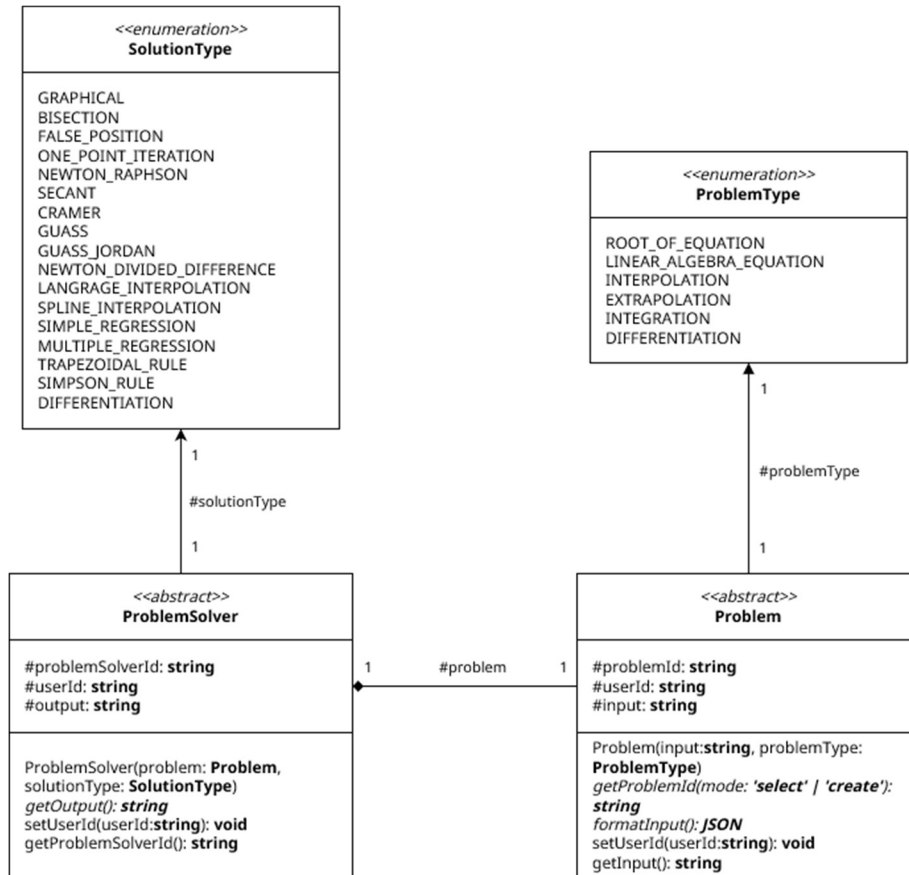
1. ความต้องการของระบบ (Functional Requirements)
  - สามารถคำนวณปัญหาทาง Numerical ดังนี้ได้
    - รากของสมการ (Root of Equation)
    - สมการพีชคณิตเชิงเส้น (Linear Algebra Equation)
    - การประมาณค่าในช่วง (Interpolation)
    - การประมาณค่านอกช่วง (Extrapolation)
    - ปริพันธ์ (Integration)
    - อนุพันธ์ (Differentiation)
  - สามารถแสดงผลลัพธ์ออกเป็นกราฟแบบโต้ตอบ (Interactive graph) โดยใช้ ployly.js
  - สามารถเก็บข้อมูลของปัญหาที่เคยคำนวณ โดยใช้ PlanetScale Database
  - สามารถเข้าสู่ระบบด้วย Google Account
  - สามารถแสดงปัญหาที่เคยคำนวณแล้ว และแสดงจำนวนครั้งที่ถูกคำนวณได้

## 2. ตารางเวลาการดำเนินโครงการ (Project Schedule)

งาน	สัปดาห์ 1 (ก.ย.)			สัปดาห์ 2 (ต.ค.)			สัปดาห์ 3 (ต.ค.)			สัปดาห์ 4 (ต.ค.)			สัปดาห์ 5 (ต.ค.)		
	25	27	29	2	4	6	9	11	13	16	18	20	23	25	27
1. ติดตั้งและออกแบบฐานข้อมูล	X														
2. ระบบเข้าสู่ระบบด้วย Google		X													
3. ปัญหา Root of Equation			X	X	X	X									
4. กราฟแบบโต้ตอบ (plotly.js)							X								
5. ปัญหา Linear Algebra Equation								X	X	X					
6. ปัญหา Interpolation											X	X			
7. ปัญหา Extrapolation													X		
8. ปัญหา Integration														X	
9. ปัญหา Differentiation															X
	0%	6%	13%	22%	28%	34%	44%	50%	56%	66%	72%	78%	88%	94%	100%

## บทที่ 2 การพัฒนา

### แผนภาพ Class Diagram



โครงการนี้จะมีคราสหลักอยู่สองคราสคือ

1. คราส Problem จะเก็บข้อมูลของปัญหาต่างๆ เช่น

- Id ของปัญหา
- Id ของผู้ใช้งาน (ที่ถามปัญหา)
- รายละเอียดของปัญหาซึ่งจะเก็บในรูปแบบของ **JSON** ที่เป็น **string**
- ประเภทของปัญหา เช่น Root of equation, ...

และจะมี Methods ต่างๆ ดังนี้

- **getProblemId(mode: 'select' | 'create')** ซึ่งเป็น Abstract method โดย methods นี้จะสามารถเลือกได้ว่าจะสร้าง Problem ใหม่หรือดึงจากที่มีอยู่บนฐานข้อมูล ซึ่งแต่ละปัญหาจะมีการดึงข้อมูลจากฐานข้อมูลที่แตกต่างกัน จึงสามารถ Inherit คราสนี้เพื่อไปสร้างเป็นปัญหาต่างๆ ได้
- **setUserId(userId: string)** ตั้ง Id ของผู้ใช้งาน (ที่ถามปัญหา)

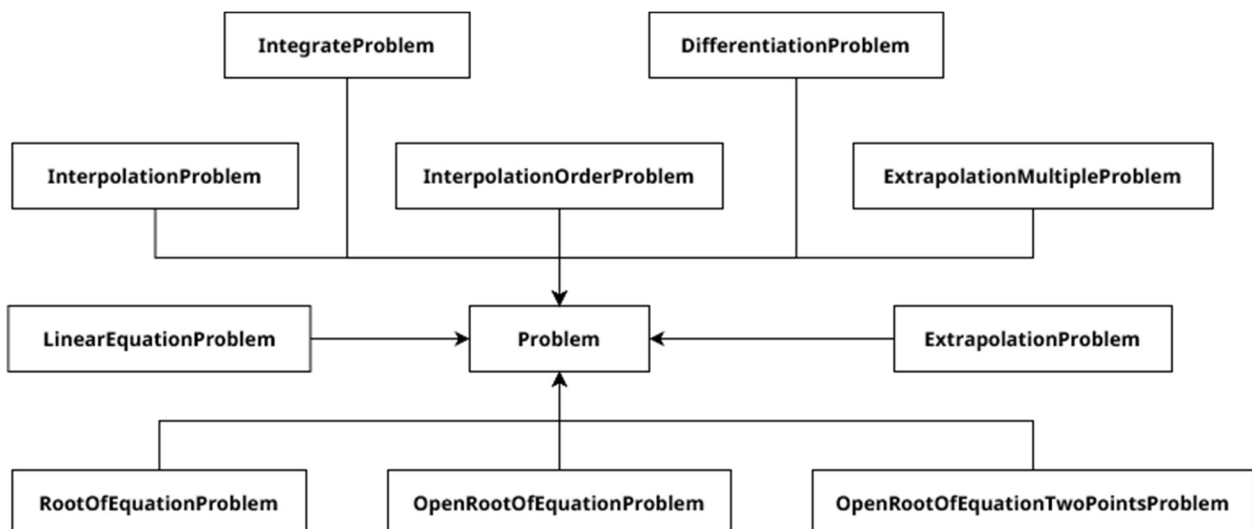
- getInput() ส่งรายละเอียดของปัญหา
- formatInput() เป็น Abstract class ที่จะเช็คข้อมูลที่ส่งมาจากผู้ใช้งานให้ตรงกับปัญหาที่ถาม หากไม่ตรงจะไม่สามารถแก้ปัญหานั้นได้ หรือข้อมูลเยอะเกินที่ระบบรองรับ

2. คราส ProblemSolver จะเก็บวิธีการแก้ปัญหารูปแบบต่างๆ ของปัญหานั้นๆ เช่น

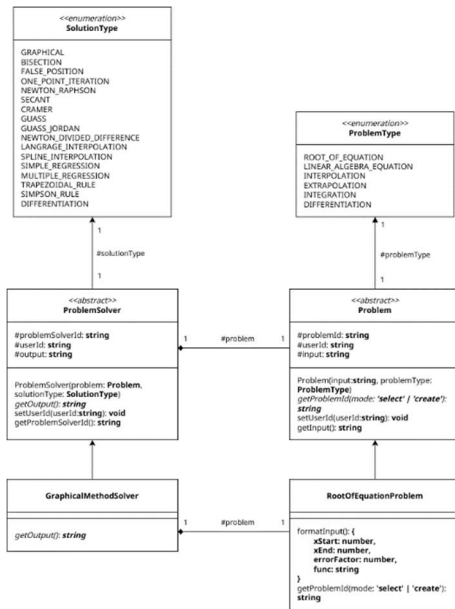
- Id ของ Solution
- Id ของผู้ใช้งาน (ที่ถามปัญหาด้วย Solution นี้)
- รายละเอียดของวิธีการแก้ปัญหาวงจะเก็บอยู่ในรูปของ **JSON** ที่เป็น **string**
- ประเภทของวิธีแก้ปัญห เช่น Graphical Method, ...

และจะมี Methods ต่างๆ ดังนี้

- getOutput() เป็น Abstract Method ที่จะดึง input ของคราส Problem มาแก้ปัญหาลักษณะต่างๆ ตามวิธีการแก้ปัญหของ Object นั้นๆ โดย Methods จะดึงวิธีแก้ปัญหามาจากฐานข้อมูลและเพิ่มจำนวนการแก้ ถ้าหากเคยมีคนถามคำถามนี้แล้ว หากไม่มีจะทำการเพิ่มเข้าไปในฐานข้อมูล
- setUserId(userId: string) ตั้ง Id ของผู้ใช้งาน (ที่ถามปัญหาด้วย Solution นี้)
- getProblemSolverId() โดย Method นี้จะดึง Id ของ Solution จากฐานข้อมูล จาก Id ของปัญหาและวิธีการแก้ปัญห หากมีจะตั้งให้ output เป็นคำตอบจากฐานข้อมูลเลย



แผนภาพการสืบทอดคราส ของคราส Problem  
(แบบง่าย)



```

export class RootOfEquationProblem extends Problem {
  constructor(input: string) {
    super(input, 'ROOT_OF_EQUATION');
  }

  formatInput(): [
    null | { xStart: number; xEnd: number; errorFactor: number; func: string },
    null | { message: string; status: number }
  ] {
    // ...
  }

  async getProblemId(
    mode: 'select' | 'create' = 'select'
  ): Promise<null | string | undefined, null | { message: string; status: number }> {
    // ...
  }
}

export class OpenRootOfEquationProblem extends Problem {
  constructor(input: string) {
    super(input, 'ROOT_OF_EQUATION');
  }

  formatInput(): [
    null | { xStart: number; errorFactor: number; func: string },
    null | { message: string; status: number }
  ] {
    // ...
  }

  async getProblemId(
    mode: 'select' | 'create' = 'select'
  ): Promise<null | string | undefined, null | { message: string; status: number }> {
    // ...
  }
}
  
```

## ตัวอย่างของการ Implement คราส Graphical Method และ ปัญหาของ Root Of Equation

### รูปแบบการพัฒนาโครงการ



- ภาษา: Svelte, TypeScript
- Framework: SvelteKit ( Svelte + Vite)
- ผู้ให้บริการโฮสติ้ง: Vercel
- ฐานข้อมูล
  - Prisma เป็นไลบรารีจับคู่เชื่อมโยงระหว่างโมเดลเชิงวัตถุและเชิงสัมพันธ์ หรือ ORM
  - PlanetScale ผู้ให้บริการฐานข้อมูลแบบ MySQL
- CSS และ Components:
  - TailwindCSS เป็น CSS Utility Framework
  - shadcn-svelte เป็นไลบรารีที่รวม Components ต่างๆ
  - KaTeX เป็นไลบรารีแสดงสมการทางคณิตศาสตร์
- กราฟแบบโต้ตอบ: Plotly.js
- ไลบรารีคำนวณทางคณิตศาสตร์: Math.js
- ไลบรารีระบบลือคอิน: Lucia

## แนวทางการเขียนโปรแกรมเชิงวัตถุ

### - Constructor

```
// src/lib/server/problem.ts
constructor(input: string, problemType: ProblemType) {
  this.input = input;
  this.problemType = problemType;
}
```

```
// src/lib/server/problem.ts
constructor(problem: Problem, solutionType: SolutionType) {
  this.problem = problem;
  this.solutionType = solutionType;
}
```

Constructor ของคลาส Problem และคลาส ProblemSolver เพื่อรับ Parameters ต่างๆ มาตั้ง Attribute

```
// src/lib/server/rootProblem.ts
constructor(input: string) {
  super(input, 'ROOT_OF_EQUATION');
}
```

เมื่อคลาส RootOfEquationProblem inherit คลาส Problem มาจึงไม่จำเป็นต้องใส่ solutionType แต่สามารถเรียก super class Constructor ได้เลย

### - Encapsulation

```
// src/lib/server/problem.ts
protected problemSolverId?: string;
protected solutionType: SolutionType;
protected userId?: string;
protected problem: Problem;
protected output?: string;
```

Attribute ของคลาสหลักๆ เช่นคลาส Problem และคลาส ProblemSolver จะเป็นแบบ Protected เพื่อให้สามารถมองเห็นได้เฉพาะคลาสที่ Inherit ไปเช่น คลาส CramerSolver, คลาส GuassEliminationSolver, ... โดยหากต้องการค่าของ Attribute สามารถใช้ได้จาก getter methods เช่น getInput(), getOutput()

### - Composition

```
// src/lib/server/problem.ts
protected problem: Problem;
```

ในคลาสของ ProblemSolver จะมี Attribute เป็นวัตถุ (Object) ของคลาส Problem ซึ่งไว้เก็บปัญหาที่คลาส ProblemSolver นั้นๆ พยายามแก้ ตัวอย่างเช่นคลาส TrapezoidalRuleSolver จะมี Class ปัญหาคือคลาส IntegrateProblem

```
// src/lib/server/problem.ts
const [problemId, problemIdError] = await this.problem.getProblemId('select');
```

## การเรียกใช้งานคลาส Problem ในคลาส ProblemSolver

### - Polymorphism

```
// src/routes/api/solution/integrate/trapezoidal/+server.ts
const problem = new IntegrateProblem(JSON.stringify(dataJson));
const problemSolved = new TrapezoidalRuleSolver(problem);

// src/lib/server/integrateProblem.ts
constructor(problem: Problem) {
  super(problem, 'TRAPEZOIDAL_RULE');
}
```

ในส่วนของ API ต่างๆ จะมีการสร้างคลาสปัญหา (Problem) และ คลาสการแก้ปัญหา (ProblemSolver) ซึ่งจากตัวอย่าง API trapezoidal จะมีการสร้างคลาส IntegrateProblem และ คลาส TrapezoidalRuleSolver ซึ่งการสร้างคลาส TrapezoidalRuleSolver, Constructor จะรับ คลาส IntegrateProblem เข้ามาเป็น Attribute problem ที่เป็นคลาส Problem จึงเป็นการที่ IntegrateProblem อยู่ในรูปของคลาส Problem

### - Abstract & Inheritance

```
// src/lib/server/problem.ts
abstract getOutput(): Promise<object | null, { message: string; status: number } | null>;

// src/lib/server/interpolationProblem.ts
export class NewtonDividedDifferenceSolver extends ProblemSolver {
  constructor(problem: Problem) {
    super(problem, 'NEWTON_DIVIDED_DIFFERENCE');
  }

  async getOutput(): Promise<object | null, { message: string; status: number } | null> {
    // ...
  }
}
```

โดยคลาส Problem และ ProblemSolver จะมี Abstract methods ต่างๆ ให้คลาสที่ inherit ไปสามารถ implement methods ได้ตามความต้องการและรายละเอียดของปัญหา

ตัวอย่างเช่นคลาส NewtonDividedDifferenceSolver เป็นคลาสที่สืบทอดมาจากคลาส ProblemSolver ซึ่งจะมี Abstract method เป็น getOutput() ซึ่งคลาส NewtonDividedDifferenceSolver ก็จะมี implement method โดยจะส่งคำตอบออกมาด้วยวิธีการของ Newton divided-difference