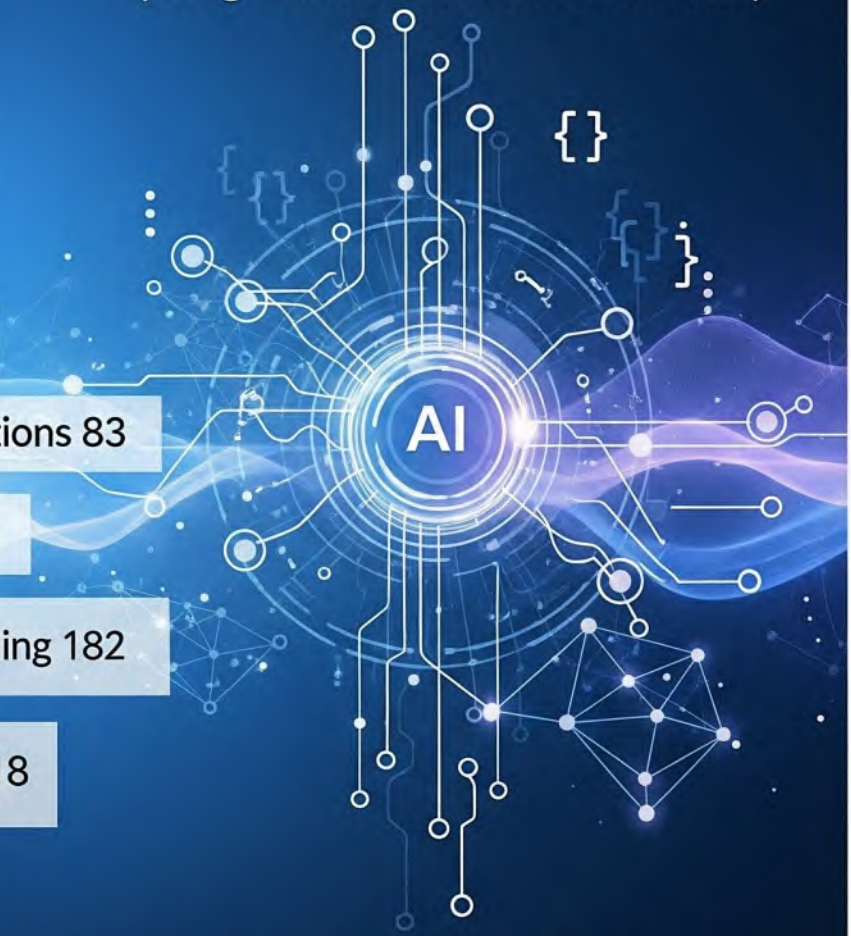
Dart Programming Language: Intermediate

(Integrative-Generative AI Edition)

- Collections 1
- Null Safety 50
- Advanced Functions 83
- Basic OOP 120
- Exception Handling 182
- Bibliography 218



Author: Student Price Book Center

คำนำ

ในโลกของการพัฒนาโปรแกรมสมัยใหม่ การพัฒนาซอฟต์แวร์ที่มีประสิทธิภาพและยืดหยุ่นจำเป็นต้องเข้าใจแนวคิดและเทคนิคที่ลึกซึ้งกว่าการเขียนโค้ดพื้นฐาน ภาษา Dart ซึ่งพัฒนาโดย Google เป็นภาษาที่ได้รับความนิยมอย่างรวดเร็ว โดยเฉพาะในงานพัฒนาแอปพลิเคชันข้ามแพลตฟอร์มร่วมกับ Flutter ด้วยความเรียบง่ายและประสิทธิภาพในการประมวลผล Dart สามารถรองรับทั้งการพัฒนาแอปมือถือ เว็บ และเดสก์ท็อปได้จากโค้ดฐานเดียว

หนังสือเล่มนี้ **Dart Programming Language: Intermediate** ถูกออกแบบมาสำหรับผู้ที่มีพื้นฐานการเขียนโปรแกรม Dart มาแล้วและต้องการพัฒนาความรู้ไปสู่ระดับที่ซับซ้อนมากขึ้น บทเรียนในเล่มจะเน้นการเข้าใจเชิงลึกและการประยุกต์ใช้งานจริง ไม่เพียงแต่การเขียนโค้ด แต่รวมถึงการออกแบบโครงสร้างข้อมูล การจัดการข้อผิดพลาด และการใช้ฟังก์ชันขั้นสูงเพื่อสร้างโปรแกรมที่มีคุณภาพสูง

บทที่ 7 **Collections** จะพาผู้อ่านเข้าสู่โลกของโครงสร้างข้อมูลแบบหลายค่า (multi-value) ใน Dart ทั้ง **List**, **Set**, และ **Map** เนื้อหาจะอธิบายทั้งการสร้าง แก้ไข ลบ และการเข้าถึงข้อมูล พร้อมตัวอย่างการวนลูปด้วยวิธีต่าง ๆ เช่น for, forEach, map และ where เพื่อให้ผู้อ่านสามารถจัดการข้อมูลจำนวนมากได้อย่างมีประสิทธิภาพ

บทที่ 8 **Null Safety** เป็นหัวใจสำคัญในการเขียนโค้ดที่ปลอดภัยจากข้อผิดพลาดประเภท null ที่มักเกิดขึ้นในหลายภาษา Dart แนะนำแนวคิดของ **Nullable** และ **Non-nullable variables** พร้อมทั้งเครื่องมือ **Null-aware Operators** เช่น ?, ??, ??=, ?. เพื่อช่วยให้โค้ดมีความปลอดภัย กระชับ และมั่นใจว่าการเข้าถึงค่าต่าง ๆ จะไม่ทำให้โปรแกรมล่ม

บทที่ 9 **Functions ขั้นสูง** จะเน้นการใช้ฟังก์ชันในรูปแบบที่ซับซ้อนขึ้น เช่น **Anonymous Functions** หรือฟังก์ชันไร้ชื่อ, **Higher-order Functions** ที่สามารถรับหรือส่งฟังก์ชันเป็นพารามิเตอร์ และ **Closures** ที่ช่วยให้ฟังก์ชันจดจำค่าจากสโคปภายนอกได้ การเข้าใจฟังก์ชันขั้นสูงจะช่วยให้ผู้อ่านสามารถเขียนโค้ดที่ยืดหยุ่นและสามารถนำกลับมาใช้ซ้ำได้

บทที่ 10 **Object-Oriented Programming (OOP) พื้นฐาน** จะปูพื้นฐานการออกแบบโปรแกรมเชิงวัตถุใน Dart ครอบคลุมการสร้าง **Class** และ **Object**, การกำหนด **Fields** และ **Methods**, การใช้ **Constructor** และคำสำคัญ **this** รวมถึงการใช้ **Getter** และ **Setter** เพื่อควบคุมการเข้าถึงข้อมูล นอกจากนี้ยังมีตัวอย่างบูรณาการเพื่อช่วยให้ผู้อ่านเข้าใจการประยุกต์ใช้งาน OOP ในสถานการณ์จริง

บทที่ 11 **Exception Handling** จะสอนวิธีจัดการข้อผิดพลาดอย่างเป็นระบบด้วยคำสั่ง **try / catch / finally** และการสร้าง **Custom Exception** รวมถึงการใช้ **throw** เพื่อโยนข้อผิดพลาดไปยังจุดที่สามารถจัดการได้ การเรียนรู้บทนี้จะช่วยให้ผู้อ่านสามารถเขียนโปรแกรมที่ทนทาน ปลอดภัย และสามารถควบคุมการไหลของโปรแกรมได้แม้เกิดข้อผิดพลาด

หนังสือเล่มนี้ออกแบบให้ผู้อ่านได้เรียนรู้ทั้งแนวคิดเชิงทฤษฎีและตัวอย่างเชิงปฏิบัติ เพื่อให้สามารถนำไปประยุกต์ใช้ในการพัฒนาโปรแกรมจริงได้อย่างมั่นใจและมีประสิทธิภาพ การศึกษาเรื่อง Collections, Null Safety, Functions ขั้นสูง, OOP และ Exception Handling จะช่วยยกระดับความเข้าใจและทักษะของผู้เรียนสู่การเป็นนักพัฒนาที่เชี่ยวชาญมากยิ่งขึ้น

ท้ายที่สุด หนังสือเล่มนี้มุ่งหวังที่จะเป็นคู่มือสำหรับนักพัฒนาที่ต้องการต่อยอดจากพื้นฐานไปสู่การใช้งาน Dart อย่างมืออาชีพ ทั้งในเชิงการออกแบบโค้ด การจัดการข้อมูล และการสร้างโปรแกรมที่มั่นคงและยืดหยุ่น

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 7 Collections (Collections) 1

- Collections
- เจาะลึกเรื่อง Collections ในภาษา Dart
- List: ข้อมูลที่เรียงตามลำดับ
- Set และความแตกต่างที่สำคัญระหว่าง Set กับ List ในภาษา Dart
- Map
- การวนลูปผ่าน Collections

บทที่ 8 Null Safety (Null Safety) 50

- Null Safety
- อธิบายเรื่อง Null Safety ให้ละเอียด
- ความหมายของ Null Safety ในภาษา Dart
- Non-nullable variables (ค่าเริ่มต้น)
- Null-aware Operators
- ตัวอย่างบูรณาการ

บทที่ 9 Functions ขั้นสูง (Advanced Functions) 83

- Functions ขั้นสูง
- เจาะลึกเรื่อง Functions ขั้นสูง ในภาษา Dart
- Anonymous Functions (ฟังก์ชันไร้ชื่อ)
- Higher-order Functions (ฟังก์ชันที่รับ/ส่งฟังก์ชัน)
- Closures

บทที่ 10 Object-Oriented Programming (OOP) พื้นฐาน (Basic Object-Oriented Programming (OOP)) 120

- Object-Oriented Programming (OOP) พื้นฐาน
- เจาะลึกเรื่อง Object-Oriented Programming (OOP) พื้นฐาน
- การสร้าง Class และ Object
- Fields และ Methods

- Constructor
- การใช้ this
- Getter และ Setter
- ตัวอย่างบูรณาการ

บทที่ 11 Exception Handling (Exception Handling) 182

- Exception Handling (การจัดการข้อผิดพลาด)
- เจาะลึกเรื่อง Exception Handling ในภาษา Dart
- try / catch / finally
- การสร้าง Exception เอง (Custom Exception)
- การใช้ throw
- ตัวอย่างบูรณาการ

บรรณานุกรม 218

บทที่ 7

Collections

(Collections)

เนื้อหา

- Collections
- เจาะลึกเรื่อง Collections ในภาษา Dart
- List: ข้อมูลที่เรียงตามลำดับ
- Set และความแตกต่างที่สำคัญระหว่าง Set กับ List ในภาษา Dart
- Map
- การวนลูปผ่าน Collections

บทนำ

โครงสร้างข้อมูลแบบ **Collections** ถือเป็นองค์ประกอบพื้นฐานที่สำคัญของการเขียนโปรแกรมด้วยภาษา Dart เนื่องจากช่วยให้สามารถจัดเก็บและจัดการข้อมูลหลายค่าภายในตัวแปรเดียวได้อย่างมีประสิทธิภาพ การเข้าใจและใช้งาน Collections อย่างถูกต้องจะทำให้นักพัฒนาสามารถสร้างโค้ดที่กระชับ อ่านง่าย และมีประสิทธิภาพมากขึ้น บทนี้จะนำเสนอแนวคิดและการใช้งาน Collections ประเภทหลักใน Dart ได้แก่ **List**, **Set** และ **Map** พร้อมทั้งวิธีการวนลูปเพื่อประมวลผลข้อมูลในคอลเลกชันเหล่านี้

List เป็นโครงสร้างข้อมูลที่ใช้จัดเก็บข้อมูลแบบลำดับ (ordered collection) ซึ่งแต่ละค่าจะถูกเข้าถึงผ่าน **index** ที่เริ่มจาก 0 นักพัฒนาสามารถสร้าง เพิ่ม ลบ และแก้ไขค่าภายใน List ได้อย่างยืดหยุ่น ความสามารถนี้ทำให้ List เหมาะสมกับงานที่ต้องการจัดเก็บข้อมูลตามลำดับหรือมีการเข้าถึงค่าด้วยตำแหน่งโดยตรง บทนี้จะอธิบายขั้นตอนการประกาศและใช้งาน List ตั้งแต่ระดับพื้นฐานไปจนถึงการประยุกต์ใช้งานในสถานการณ์จริง

นอกจาก List แล้ว **Set** เป็นอีกหนึ่งโครงสร้างข้อมูลที่ Dart ให้การสนับสนุน โดย Set ใช้เก็บข้อมูลที่ไม่ซ้ำกัน (unique values) และไม่รับประกันลำดับการจัดเก็บ ทำให้ Set เหมาะสมกับงานที่ต้องการป้องกันข้อมูลซ้ำโดยอัตโนมัติ ความแตกต่างระหว่าง Set และ List จะถูกอธิบายอย่างละเอียดในบทนี้ เพื่อช่วยให้ผู้อ่านสามารถเลือกใช้โครงสร้างข้อมูลที่เหมาะสมกับโจทย์ของตนเองได้

โครงสร้างข้อมูลประเภทที่สามที่มีความสำคัญคือ **Map** ซึ่งใช้เก็บข้อมูลในรูปแบบคู่ **Key-Value** โดย Key ต้องไม่ซ้ำกัน และสามารถใช้เพื่อเข้าถึงหรือแก้ไขค่าที่สอดคล้องกับ Key นั้น ๆ ได้อย่างรวดเร็ว

Map มีประโยชน์มากในการเก็บข้อมูลที่ต้องการการอ้างอิงแบบชื่อ เช่น ข้อมูลการตั้งค่า (configuration), ข้อมูลผู้ใช้ หรือพจนานุกรม

นอกจากการสร้างและจัดเก็บข้อมูลใน Collections แล้ว การประมวลผลข้อมูลที่อยู่ภายในก็มีความสำคัญไม่แพ้กัน ภาษา Dart มีคำสั่งและเมธอดที่ช่วยให้การวนลูปผ่านข้อมูลใน Collections ทำได้ง่ายและมีประสิทธิภาพ เช่น **for**, **forEach**, **map** และ **where** ซึ่งแต่ละวิธีมีความเหมาะสมในบริบทที่แตกต่างกัน ทั้งในการเปลี่ยนแปลงค่าภายใน, คัดกรองข้อมูล หรือสร้างชุดข้อมูลใหม่จากข้อมูลเดิม

การเลือกใช้วิธีการวนลูปที่เหมาะสมไม่เพียงส่งผลต่อความถูกต้องของโปรแกรม แต่ยังมีผลต่อความเร็วในการประมวลผลและความชัดเจนของโค้ดอีกด้วย บทนี้จะอธิบายข้อดีและข้อควรระวังของแต่ละวิธี รวมถึงยกตัวอย่างการประยุกต์ใช้งานที่ครอบคลุมทั้งกรณีทั่วไปและกรณีที่ซับซ้อน เพื่อให้ผู้อ่านสามารถนำไปประยุกต์ใช้ได้ทันที

โดยสรุป บทที่ 7 นี้มีจุดมุ่งหมายเพื่อสร้างความเข้าใจอย่างลึกซึ้งเกี่ยวกับการใช้งาน Collections ใน Dart ทั้งในแง่โครงสร้าง ข้อแตกต่าง และเทคนิคการวนลูปจัดการข้อมูล การมีพื้นฐานที่แข็งแรงในหัวข้อนี้จะช่วยให้การพัฒนาโปรแกรมด้วย Dart มีความยืดหยุ่นและมีประสิทธิภาพมากยิ่งขึ้น ตลอดจนเป็นรากฐานสำคัญสำหรับการทำงานกับข้อมูลในระดับที่ซับซ้อนกว่าในบทต่อ ๆ ไป

Collections

- List (สร้าง, เพิ่ม, ลบ, แก้ไข)
- Set และความแตกต่างจาก List
- Map (Key-Value pairs)
- การวนลูปผ่าน Collections (for, forEach, map, where)

---

1. List: รายการของข้อมูลที่มีลำดับ

List คือกลุ่มของข้อมูลที่มีการจัดเรียงตามลำดับ (index) โดย Index จะเริ่มจาก 0 เสมอ

การสร้าง List

เราสามารถสร้าง List ได้หลายวิธี เช่น:

- **สร้าง List เปล่า:**

  Dart

  ```
  List<String> fruits = [];
  ```

- **สร้าง List พร้อมข้อมูล:**

  Dart

  ```
  List<String> fruits = ['Apple', 'Banana', 'Orange'];
  ```

  ในตัวอย่างนี้ เราได้กำหนดชนิดข้อมูลเป็น String ให้กับ List ด้วย ทำให้มั่นใจได้ว่า List นี้จะเก็บได้แค่ข้อความเท่านั้น

การเพิ่ม ลบ และแก้ไขข้อมูล

- **เพิ่ม (Add):**

  Dart

  ```
  fruits.add('Grape'); // เพิ่ม 'Grape' เข้าไปท้าย List
  fruits.insert(1, 'Mango'); // เพิ่ม 'Mango' ที่ index 1
  ```

- **ลบ (Remove):**

  Dart

  ```
  fruits.remove('Banana'); // ลบ 'Banana' ออกจาก List
  fruits.removeAt(0); // ลบข้อมูลที่ index 0
  ```

- **แก้ไข (Update):**

  Dart

  ```
  fruits[0] = 'Pineapple'; // แก้ไขข้อมูลที่ index 0 เป็น 'Pineapple'
  ```

---

2. Set: กลุ่มข้อมูลที่ไม่ซ้ำกันและไม่มีลำดับ

Set คือกลุ่มของข้อมูลที่ **ไม่ยอมให้มีข้อมูลซ้ำกัน** และ **ไม่มีลำดับ** (ไม่มี Index)

ความแตกต่างที่สำคัญระหว่าง Set กับ List:

- **ข้อมูลซ้ำซ้อน:**
  - **List:** อนุญาตให้มีข้อมูลซ้ำได้

    Dart

    ```
    List<int> numbers = [1, 2, 2, 3]; // ทำงานได้
    ```

  - **Set:** ไม่อนุญาตให้มีข้อมูลซ้ำกัน หากพยายามเพิ่มข้อมูลที่ซ้ำเข้าไป Set จะไม่ทำการเพิ่มและจะไม่เกิดข้อผิดพลาด

    Dart

    ```
    Set<int> uniqueNumbers = {1, 2, 2, 3}; // Set จะเก็บแค่ {1, 2, 3}
    uniqueNumbers.add(1); // ข้อมูลซ้ำ ไม่มีการเปลี่ยนแปลง
    ```

- **การเข้าถึงข้อมูล:**
  - **List:** เข้าถึงข้อมูลด้วย Index เช่น fruits[0]
  - **Set:** ไม่มี Index จึงไม่สามารถเข้าถึงข้อมูลแบบ Index ได้ ต้องใช้วิธีอื่น เช่น uniqueNumbers.contains(2) เพื่อตรวจสอบว่ามีข้อมูลนั้นอยู่หรือไม่

---

3. Map: คู่ของข้อมูล (Key-Value pairs)

Map หรือที่เรียกว่า Dictionary ในภาษาอื่น คือกลุ่มของข้อมูลที่เก็บเป็น **คู่ (Pair)** โดยแต่ละคู่ประกอบด้วย **Key** (กุญแจ) และ **Value** (ค่า) Key จะต้องไม่ซ้ำกันและใช้สำหรับอ้างอิงถึง Value ที่เกี่ยวข้อง

การสร้าง Map

Dart

```
Map<String, String> person = {
  'name': 'John Doe',
  'city': 'New York',
  'job': 'Developer'
};
```

ในตัวอย่างนี้ Key เป็น String และ Value ก็เป็น String

การเพิ่ม ลบ และแก้ไขข้อมูล

- **เข้าถึง/เพิ่ม/แก้ไข:**

Dart

```
print(person['name']); // เข้าถึง Value ด้วย Key 'name'
person['age'] = '30'; // เพิ่ม Key 'age' พร้อม Value
person['city'] = 'San Francisco'; // แก้ไข Value ของ Key 'city'
```

- **ลบ:**

Dart

```
person.remove('job'); // ลบคู่ Key-Value ที่มี Key เป็น 'job'
```

---

4. การวนลูป (Looping) ผ่าน Collections

การวนลูปคือการทำงานซ้ำๆ กับข้อมูลทั้งหมดใน Collection

for loop (การวนลูปแบบดั้งเดิม)

วิธีนี้เหมาะสำหรับ List ที่เราต้องการเข้าถึง Index ของข้อมูลด้วย

Dart

```
List<String> fruits = ['Apple', 'Banana', 'Orange'];
for (int i = 0; i < fruits.length; i++) {
  print('Fruit at index $i is ${fruits[i]}');
}
```

for-in loop (การวนลูปที่อ่านง่ายกว่า)

เป็นวิธีที่ง่ายและนิยมมากที่สุดสำหรับการวนลูปผ่าน Collection

Dart

```
for (String fruit in fruits) {
  print(fruit);
}
```

.forEach method (การวนลูปแบบ Functional)

เป็นอีกวิธีที่นิยม ใช้ร่วมกับ Arrow function (=>) ได้เพื่อความกระชับ

Dart

```
fruits.forEach((fruit) => print(fruit));
```

.map method (การแปลงข้อมูล)

.map ใช้สำหรับ **แปลงข้อมูล** ใน Collection หนึ่งไปเป็น Collection ใหม่ เช่น การแปลง List ของ String เป็น List ของความยาวของ String นั้นๆ

Dart

```
List<int> lengths = fruits.map((fruit) => fruit.length).toList();
print(lengths); // [5, 6, 6]
```

ข้อสำคัญ: .map จะคืนค่าเป็น Iterable ดังนั้นจึงต้องใช้ .toList() เพื่อแปลงกลับเป็น List หากต้องการ

.where method (การกรองข้อมูล)

.where ใช้สำหรับ **กรองข้อมูล** ใน Collection โดยมีเงื่อนไขที่เรากำหนด

Dart

```
List<String> longFruits = fruits.where((fruit) => fruit.length > 5).toList();
print(longFruits); // [Banana, Orange]
```

คล้ายกับ .map .where จะคืนค่าเป็น Iterable เช่นกัน จึงต้องใช้ .toList() เพื่อแปลงเป็น List

หากมีข้อสงสัยเพิ่มเติมเกี่ยวกับ Collections หรือส่วนไหนที่อยากให้เจาะลึกเป็นพิเศษ สามารถสอบถามได้เลยนะครับ!

เจาะลึกเรื่อง Collections ในภาษา Dart

---

1. List: กลุ่มข้อมูลที่มีลำดับ (Ordered Collection)

นอกจากที่เราทราบแล้วว่า List มี Index และอนุญาตให้ข้อมูลซ้ำได้ สิ่งสำคัญที่ต้องรู้คือ List ใน Dart เป็น Generic Type ซึ่งหมายความว่าเราสามารถกำหนดชนิดข้อมูลที่ List จะเก็บได้ (e.g., List<int>, List<String>, List<User>) ซึ่งช่วยป้องกันข้อผิดพลาดในการเขียนโปรแกรมได้ตั้งแต่ขั้นตอนการพัฒนา

การสร้างและคุณสมบัติเพิ่มเติม:

- **List<T> list = [];**: สร้าง Empty List ที่สามารถเพิ่มข้อมูลภายหลังได้
- **var list = <T>[];**: เป็นอีกวิธีในการสร้าง Empty List แบบกระชับ (Literal syntax)
- **.length**: Property สำหรับตรวจสอบจำนวนสมาชิกใน List
- **.first, .last**: Property สำหรับเข้าถึงสมาชิกตัวแรกและตัวสุดท้ายของ List ได้โดยตรง
- **.isEmpty, .isNotEmpty**: Property สำหรับตรวจสอบว่า List ว่างเปล่าหรือไม่
- **.add()**: เมธอดสำหรับเพิ่มข้อมูลทีละตัว
- **.addAll()**: เมธอดสำหรับเพิ่มข้อมูลจาก Iterable (เช่น List อื่น) เข้าไปใน List

- **.removeWhere()**: เมธอดสำหรับลบข้อมูลตามเงื่อนไขที่กำหนด เช่น myList.removeWhere((item) => item.isNegative);

ตัวอย่างการใช้ประโยชน์:

Dart

```
List<int> numbers = [10, 20, 30, 40];
numbers.addAll([50, 60]); // [10, 20, 30, 40, 50, 60]
numbers.removeWhere((number) => number > 40); // [10, 20, 30, 40]
print(numbers.first); // 10
```

---

2. Set: กลุ่มข้อมูลที่ไม่ซ้ำกัน (Unique Collection)

Set เหมาะสำหรับการใช้งานที่ต้องการเก็บข้อมูลที่ไม่ซ้ำกัน เช่น การเก็บรายชื่อผู้ใช้ที่ลงทะเบียน หรือการตรวจสอบว่าข้อมูลนั้นเคยถูกเพิ่มเข้ามาแล้วหรือไม่ ด้วยประสิทธิภาพที่เหนือกว่าการวนลูปใน List เพื่อตรวจสอบข้อมูลซ้ำ

ความพิเศษของ Set:

- **ประสิทธิภาพการค้นหา (Look-up)**: การตรวจสอบว่าข้อมูลมีอยู่ใน Set หรือไม่ (set.contains(item)) มีประสิทธิภาพสูงกว่า List มาก (โดยเฉลี่ยคือ O(1) หรือ Constant Time)
- **.add() และ .addAll()**: เมธอดเหล่านี้จะเพิ่มข้อมูลเฉพาะเมื่อข้อมูลนั้นยังไม่มีอยู่ใน Set
- **.union(), .intersection(), .difference()**: เมธอดสำหรับใช้ในการจัดการ Set เช่น การหาข้อมูลที่เหมือนกันหรือแตกต่างกัน
  - set1.union(set2): รวมข้อมูลทั้งหมดจากสอง Set เข้าด้วยกัน
  - set1.intersection(set2): หาข้อมูลที่ซ้ำกันในสอง Set
  - set1.difference(set2): หาข้อมูลที่อยู่ใน set1 แต่ไม่อยู่ใน set2

ตัวอย่างการใช้ประโยชน์:

Dart

```
Set<String> fruits = {'Apple', 'Banana', 'Orange'};
Set<String> exoticFruits = {'Mango', 'Apple', 'Pineapple'};

Set<String> commonFruits = fruits.intersection(exoticFruits);
print(commonFruits); // {'Apple'}

Set<String> allFruits = fruits.union(exoticFruits);
print(allFruits); // {'Apple', 'Banana', 'Orange', 'Mango', 'Pineapple'}
```

---

3. Map: คู่ข้อมูล (Key-Value Pairs)

Map เป็นโครงสร้างข้อมูลที่ใช้ Key สำหรับเข้าถึง Value มีประสิทธิภาพสูงในการค้นหาข้อมูลด้วย Key (Look-up) เหมาะสำหรับการเก็บข้อมูลที่มีความสัมพันธ์กัน เช่น ข้อมูลผู้ใช้ (Key: 'id', Value: ข้อมูลผู้ใช้) หรือการตั้งค่า Configuration ต่างๆ

ความพิเศษของ Map:

- **Key ที่ไม่ซ้ำกัน**: แต่ละ Key ใน Map จะต้องไม่ซ้ำกัน หากมีการเพิ่ม Key ที่ซ้ำ Map จะทำการแก้ไข Value ของ Key นั้นแทน
- **ชนิดข้อมูลที่ยืดหยุ่น**: ทั้ง Key และ Value สามารถเป็นชนิดข้อมูลใดก็ได้ (e.g., Map<String, int>, Map<int, User>)
- **.keys, .values**: Property สำหรับดึงเฉพาะ Key หรือ Value ทั้งหมดออกมาเป็น Iterable
- **.containsKey(), .containsValue()**: เมธอดสำหรับตรวจสอบว่า Map มี Key หรือ Value ที่เราต้องการหรือไม่
- **.forEach()**: เมธอดสำหรับวนลูปผ่านทุกคู่ Key-Value
- **.putIfAbsent()**: เมธอดที่ช่วยเพิ่ม Key และ Value เข้าไปใน Map เฉพาะในกรณีที่ Key นั้นยังไม่มีอยู่

ตัวอย่างการใช้ประโยชน์:

Dart

```dart
Map<String, dynamic> user = {
  'id': 1,
  'name': 'John Doe',
  'is_active': true
};

user.putIfAbsent('age', () => 30); // เพิ่ม 'age' เข้าไป เพราะยังไม่มี Key นี้
user.forEach((key, value) {
  print('$key: $value');
});

// Output:
// id: 1
// name: John Doe
// is_active: true
// age: 30
```

---

4. การวนลูปเชิง Functional (Functional Iteration)

นอกเหนือจาก for loop แบบดั้งเดิมแล้ว Dart ยังมีเมธอดที่ช่วยให้การวนลูปและจัดการ Collections ง่ายและอ่านได้สะดวกขึ้นมาก

- **.map()**: เหมาะสำหรับการ **แปลง** ข้อมูลจาก Collection หนึ่งไปเป็นอีก Collection หนึ่ง
- **.where()**: เหมาะสำหรับการ **กรอง** ข้อมูลใน Collection โดยจะคืนค่าเฉพาะข้อมูลที่ตรงตามเงื่อนไขเท่านั้น
- **.every()**: ตรวจสอบว่า **ทุกสมาชิก** ใน Collection ตรงตามเงื่อนไขหรือไม่
- **.any()**: ตรวจสอบว่ามี **อย่างน้อยหนึ่งสมาชิก** ใน Collection ที่ตรงตามเงื่อนไขหรือไม่
- **.fold()**: เมธอดที่ทรงพลังมากสำหรับการ **รวม (Aggregate)** ข้อมูลทั้งหมดใน Collection ให้เหลือค่าเดียว เช่น การหาผลรวม, การหาค่าเฉลี่ย
- **.reduce()**: คล้ายกับ .fold() แต่จะใช้สมาชิกตัวแรกของ Collection เป็นค่าเริ่มต้น

ตัวอย่างการใช้ .fold():

Dart

```
List<int> numbers = [1, 2, 3, 4, 5];
int sum = numbers.fold(0, (previousValue, element) => previousValue + element);
print(sum); // 15
```

ในตัวอย่างนี้ 0 คือค่าเริ่มต้น และ previousValue จะเก็บผลรวมที่คำนวณมาได้ในแต่ละรอบการวนลูป การทำความเข้าใจในเชิงลึกเกี่ยวกับ Collections เหล่านี้จะช่วยให้คุณสามารถเลือกใช้โครงสร้างข้อมูลที่เหมาะสมกับงานได้อย่างมีประสิทธิภาพ และเขียนโค้ดได้กระชับและมีประสิทธิภาพมากขึ้น หากมีคำถามเกี่ยวกับเมธอดใดเป็นพิเศษ ถามเพิ่มเติมได้เลยครับ!

List: ข้อมูลที่เรียงตามลำดับ

List ใน Dart คือโครงสร้างข้อมูลที่ใช้เก็บชุดข้อมูลที่มีการจัดเรียงตามลำดับ (Ordered Collection) โดยแต่ละข้อมูลจะมีตำแหน่ง (index) กำกับไว้ ซึ่งจะเริ่มจาก **0** เสมอ

1. การสร้าง List (Create)

การสร้าง List สามารถทำได้หลายวิธี โดยคุณสามารถกำหนดหรือไม่กำหนดชนิดข้อมูลของสมาชิกใน List ก็ได้

1.1. การสร้าง List เปล่า

- **กำหนดชนิดข้อมูล (Type-safe):** เป็นวิธีที่แนะนำ เพราะช่วยป้องกันการเพิ่มข้อมูลผิดประเภท

Dart

```
List<String> fruits = [];
```

List นี้จะสามารถเก็บได้เฉพาะข้อมูลชนิด String เท่านั้น

- **ไม่กำหนดชนิดข้อมูล (Dynamic):** List นี้จะสามารถเก็บข้อมูลได้ทุกประเภท

Dart

```
List myDynamicList = [];
myDynamicList.add('Apple'); // ได้
myDynamicList.add(10);      // ได้
```

วิธีนี้เหมาะสำหรับกรณีที่ต้องการความยืดหยุ่นสูง แต่ก็เพิ่มความเสี่ยงเรื่อง Type-safety ได้

1.2. การสร้าง List พร้อมข้อมูลเริ่มต้น

Dart

```
List<String> fruits = ['Apple', 'Banana', 'Orange'];
List<int> numbers = [1, 2, 3, 4, 5];
```

2. การเพิ่มข้อมูลใน List (Add)

คุณสามารถเพิ่มข้อมูลใหม่เข้าไปใน List ได้หลายแบบ

- **.add(element)**: เพิ่มข้อมูลใหม่เข้าไปท้าย List

  Dart

  ```
  List<String> fruits = ['Apple', 'Banana'];
  fruits.add('Grape'); // fruits ตอนนี้คือ ['Apple', 'Banana', 'Grape']
  ```

- **.insert(index, element)**: เพิ่มข้อมูลใหม่เข้าไปในตำแหน่งที่ต้องการ โดยข้อมูลเดิมจะถูกเลื่อนออกไป

  Dart

  ```
  List<String> fruits = ['Apple', 'Grape'];
  fruits.insert(1, 'Banana'); // fruits ตอนนี้คือ ['Apple', 'Banana', 'Grape']
  ```

- **.addAll(iterable)**: เพิ่มข้อมูลจาก List หรือ Iterable อื่นๆ เข้าไปใน List

  Dart

  ```
  List<String> fruits = ['Apple', 'Banana'];
  List<String> newFruits = ['Grape', 'Mango'];
  fruits.addAll(newFruits); // fruits ตอนนี้คือ ['Apple', 'Banana', 'Grape', 'Mango']
  ```

3. การลบข้อมูลออกจาก List (Remove)

- **.remove(element)**: ลบข้อมูลตัวแรกที่ตรงกับค่าที่เรากำหนดออกจาก List

  Dart

  ```
  List<String> fruits = ['Apple', 'Banana', 'Apple'];
  fruits.remove('Apple'); // fruits ตอนนี้คือ ['Banana', 'Apple']
  ```

- **.removeAt(index)**: ลบข้อมูลที่ตำแหน่ง index ที่ระบุ

  Dart

  ```
  List<String> fruits = ['Apple', 'Banana', 'Orange'];
  fruits.removeAt(1); // fruits ตอนนี้คือ ['Apple', 'Orange']
  ```

- **.removeWhere(test)**: ลบสมาชิกทั้งหมดที่ตรงตามเงื่อนไขที่กำหนด

  Dart

```
List<int> numbers = [1, 5, 10, 15, 20];
numbers.removeWhere((number) => number > 10); // numbers ตอนนี้คือ [1, 5, 10]
```

4. การแก้ไขข้อมูลใน List (Update)

การแก้ไขข้อมูลใน List ทำได้โดยการเข้าถึงข้อมูลผ่าน index และกำหนดค่าใหม่

Dart

```
List<String> fruits = ['Apple', 'Banana', 'Orange'];

// แก้ไขข้อมูลที่ตำแหน่ง index 0
fruits[0] = 'Pineapple';
// fruits ตอนนี้คือ ['Pineapple', 'Banana', 'Orange']

// แก้ไขข้อมูลที่ตำแหน่ง index 2
fruits[2] = 'Grape';
// fruits ตอนนี้คือ ['Pineapple', 'Banana', 'Grape']
```

ตัวอย่างการใช้งานแบบครบวงจร

Dart

```
void main() {
  // 1. สร้าง List พร้อมข้อมูล
  List<String> shoppingList = ['Eggs', 'Milk', 'Bread'];
  print('Initial List: $shoppingList'); // Initial List: [Eggs, Milk, Bread]

  // 2. เพิ่มข้อมูล
  shoppingList.add('Chicken');
  print('After adding Chicken: $shoppingList'); // After adding Chicken: [Eggs, Milk, Bread,
Chicken]

  shoppingList.insert(1, 'Cheese');
  print('After inserting Cheese: $shoppingList'); // After inserting Cheese: [Eggs, Cheese, Milk,
Bread, Chicken]

  // 3. แก้ไขข้อมูล
  shoppingList[0] = 'Oatmeal';
  print('After updating at index 0: $shoppingList'); // After updating at index 0: [Oatmeal, Cheese,
Milk, Bread, Chicken]
```

```
  // 4. ลบข้อมูล
  shoppingList.remove('Milk');
  print('After removing Milk: $shoppingList'); // After removing Milk: [Oatmeal, Cheese, Bread,
Chicken]

  shoppingList.removeAt(2); // ลบ "Bread"
  print('After removing at index 2: $shoppingList'); // After removing at index 2: [Oatmeal,
Cheese, Chicken]
}
```

หวังว่าคำอธิบายและตัวอย่างเหล่านี้จะช่วยให้คุณเข้าใจการจัดการ List ใน Dart ได้อย่างละเอียดนะครับ หากมีคำถามเพิ่มเติม ถามได้เลยครับ!

ตัวอย่างโปรแกรมทั้งแบบพื้นฐานและแบบประยุกต์มาให้คุณทั้งหมด 6 โปรแกรม เพื่อให้เห็นภาพการใช้งานจริง

---

1. ตัวอย่างโปรแกรมพื้นฐาน

โปรแกรมเหล่านี้จะเน้นการสาธิตการใช้งาน List อย่างตรงไปตรงมา ทั้งการสร้าง, เพิ่ม, ลบ, และแก้ไขข้อมูล

โปรแกรมที่ 1: การสร้างและแก้ไขข้อมูลใน List

โครงสร้างไฟล์: main.dart

คำอธิบายโค้ด:

โปรแกรมนี้จะสร้าง List ของชื่อนักเรียน จากนั้นใช้การเข้าถึงด้วย index เพื่อแสดง, แก้ไข และแสดงผลข้อมูลอีกครั้งเพื่อให้เห็นความเปลี่ยนแปลง

โค้ดเต็มไฟล์:

Dart

```
void main() {
  // 1. สร้าง List ของชื่อนักเรียน
  List<String> students = ['Alice', 'Bob', 'Charlie'];

  print('--- รายชื่อนักเรียนเริ่มต้น ---');
  print(students);

  // 2. เข้าถึงข้อมูลด้วย index
```

```
  String firstStudent = students[0];
  print('\nนักเรียนคนแรก: $firstStudent');

  // 3. แก้ไขชื่อนักเรียนคนที่สอง
  students[1] = 'Ben';
  print('\n--- รายชื่อนักเรียนหลังแก้ไข ---');
  print(students);
}
```

ผลการรัน:

```
--- รายชื่อนักเรียนเริ่มต้น ---
[Alice, Bob, Charlie]

นักเรียนคนแรก: Alice

--- รายชื่อนักเรียนหลังแก้ไข ---
[Alice, Ben, Charlie]
```

โปรแกรมที่ 2: การเพิ่มและลบข้อมูลใน List

โครงสร้างไฟล์: main.dart

คำอธิบายโค้ด:

โปรแกรมนี้จะสร้าง List ของสินค้าในรถเข็น จากนั้นใช้เมธอด .add() เพื่อเพิ่มสินค้า และใช้ .remove() เพื่อลบสินค้าที่ต้องการออก

โค้ดเต็มไฟล์:

Dart

```
void main() {
  // 1. สร้าง List ของสินค้าในรถเข็น
  List<String> shoppingCart = ['Apple', 'Milk', 'Bread'];

  print('--- สินค้าในรถเข็นเริ่มต้น ---');
  print(shoppingCart);

  // 2. เพิ่มสินค้าใหม่
  shoppingCart.add('Eggs');
  print('\nเพิ่ม Eggs: $shoppingCart');
```

```
  shoppingCart.insert(0, 'Cheese');
  print('แทรก Cheese ที่ตำแหน่งแรก: $shoppingCart');

  // 3. ลบสินค้าออกจาก List
  shoppingCart.remove('Milk');
  print('\nลบ Milk: $shoppingCart');

  shoppingCart.removeAt(3); // ลบ Eggs (ที่ตำแหน่ง index 3)
  print('ลบสินค้าที่ index 3: $shoppingCart');
}
```

ผลการรัน:

```
--- สินค้าในรถเข็นเริ่มต้น ---
[Apple, Milk, Bread]

เพิ่ม Eggs: [Apple, Milk, Bread, Eggs]
แทรก Cheese ที่ตำแหน่งแรก: [Cheese, Apple, Milk, Bread, Eggs]

ลบ Milk: [Cheese, Apple, Bread, Eggs]
ลบสินค้าที่ index 3: [Cheese, Apple, Bread]
```

โปรแกรมที่ 3: การวนลูปผ่าน List ด้วย for-in และ forEach

โครงสร้างไฟล์: main.dart

คำอธิบายโค้ด:

โปรแกรมนี้จะใช้ for-in loop และ .forEach เพื่อแสดงข้อมูลใน List ทีละรายการ ซึ่งเป็นวิธีที่นิยมใช้สำหรับการอ่านค่าใน Collections

โค้ดเต็มไฟล์:

Dart

```
void main() {
  List<String> fruits = ['Apple', 'Banana', 'Orange', 'Grape'];

  print('--- แสดงผลด้วย for-in loop ---');
  for (String fruit in fruits) {
    print('ผลไม้: $fruit');
```

```
  }

  print('\n--- แสดงผลด้วย .forEach ---');
  fruits.forEach((fruit) {
    print('ผลไม้: $fruit');
  });
}
```

ผลการรัน:

```
--- แสดงผลด้วย for-in loop ---
ผลไม้: Apple
ผลไม้: Banana
ผลไม้: Orange
ผลไม้: Grape

--- แสดงผลด้วย .forEach ---
ผลไม้: Apple
ผลไม้: Banana
ผลไม้: Orange
ผลไม้: Grape
```

---

2. ตัวอย่างโปรแกรมประยุกต์

โปรแกรมเหล่านี้จะแสดงการใช้งาน List ที่ซับซ้อนขึ้นเล็กน้อย เช่น การจัดการข้อมูลที่ซับซ้อนขึ้น หรือการใช้เมธอดขั้นสูงอย่าง map และ where

โปรแกรมที่ 4: การจัดการข้อมูลนักเรียนด้วย Map และ List

โครงสร้างไฟล์: main.dart

คำอธิบายโค้ด:

โปรแกรมนี้จะสร้าง List ที่เก็บ Map โดยแต่ละ Map แทนข้อมูลนักเรียนหนึ่งคน จากนั้นใช้ .forEach เพื่อวนลูปและแสดงผลข้อมูลนักเรียนทั้งหมด

โค้ดเต็มไฟล์:

Dart

```
void main() {
  List<Map<String, dynamic>> students = [
    {'id': 1, 'name': 'Alice', 'score': 95},
    {'id': 2, 'name': 'Bob', 'score': 88},
```

```
    {'id': 3, 'name': 'Charlie', 'score': 72},
  ];

  print('--- ข้อมูลนักเรียนทั้งหมด ---');
  students.forEach((student) {
    print('ID: ${student['id']}, ชื่อ: ${student['name']}, คะแนน: ${student['score']}');
  });
}
```

ผลการรัน:

```
--- ข้อมูลนักเรียนทั้งหมด ---
ID: 1, ชื่อ: Alice, คะแนน: 95
ID: 2, ชื่อ: Bob, คะแนน: 88
ID: 3, ชื่อ: Charlie, คะแนน: 72
```

โปรแกรมที่ 5: การกรองข้อมูลด้วย .where()

โครงสร้างไฟล์: main.dart

คำอธิบายโค้ด:

โปรแกรมนี้ใช้ List จากโปรแกรมที่แล้ว จากนั้นใช้เมธอด .where() เพื่อกรองหานักเรียนที่มีคะแนนมากกว่า 90 และแสดงผลออกมา

โค้ดเต็มไฟล์:

Dart

```
void main() {
  List<Map<String, dynamic>> students = [
    {'id': 1, 'name': 'Alice', 'score': 95},
    {'id': 2, 'name': 'Bob', 'score': 88},
    {'id': 3, 'name': 'Charlie', 'score': 72},
  ];

  // ใช้ .where() เพื่อกรองนักเรียนที่ได้คะแนนมากกว่า 90
  var highAchievers = students.where((student) => student['score'] > 90);

  print('--- รายชื่อนักเรียนที่ได้คะแนนสูง (มากกว่า 90) ---');
  highAchievers.forEach((student) {
    print('ID: ${student['id']}, ชื่อ: ${student['name']}, คะแนน: ${student['score']}');
```

```
  });
}
```

ผลการรัน:

```
--- รายชื่อนักเรียนที่ได้คะแนนสูง (มากกว่า 90) ---
ID: 1, ชื่อ: Alice, คะแนน: 95
```

โปรแกรมที่ 6: การแปลงข้อมูลด้วย .map()

โครงสร้างไฟล์: main.dart

คำอธิบายโค้ด:

โปรแกรมนี้จะสร้าง List ของราคา จากนั้นใช้เมธอด .map() เพื่อแปลง List ของราคาเดิมให้กลายเป็น List ใหม่ที่ราคาเพิ่มขึ้น 10%

โค้ดเต็มไฟล์:

Dart

```dart
void main() {
  List<double> prices = [100.0, 150.0, 200.0];

  print('--- ราคาสินค้าเริ่มต้น ---');
  print(prices);

  // ใช้ .map() เพื่อสร้าง List ราคาใหม่ที่เพิ่มขึ้น 10%
  List<double> newPrices = prices.map((price) => price * 1.10).toList();

  print('\n--- ราคาสินค้าใหม่ (เพิ่มขึ้น 10%) ---');
  print(newPrices);
}
```

ผลการรัน:

```
--- ราคาสินค้าเริ่มต้น ---
[100.0, 150.0, 200.0]

--- ราคาสินค้าใหม่ (เพิ่มขึ้น 10%) ---
[110.0, 165.0, 220.0]
```

Set และความแตกต่างที่สำคัญระหว่าง Set กับ List ในภาษา Dart

Set: กลุ่มข้อมูลที่ไม่ซ้ำกัน

Set คือโครงสร้างข้อมูลที่ใช้เก็บข้อมูลโดยมีคุณสมบัติหลักคือ **ไม่อนุญาตให้มีข้อมูลซ้ำกัน** และ **ไม่มีการจัดเรียงตามลำดับ (Unordered)**

คุณสมบัติเหล่านี้ทำให้ Set เหมาะสำหรับสถานการณ์ที่ต้องการจัดการกับข้อมูลที่ไม่ซ้ำซ้อน เช่น:

- การเก็บรายชื่อ ID ผู้ใช้ที่เคยเข้าชมเว็บไซต์
- การตรวจสอบว่าข้อมูลหนึ่งๆ เคยถูกเพิ่มเข้ามาในกลุ่มแล้วหรือไม่
- การคำนวณทางคณิตศาสตร์แบบ Set (Union, Intersection, Difference)

การสร้าง Set

การสร้าง Set มีไวยากรณ์คล้ายกับ List แต่ใช้เครื่องหมาย {} (วงเล็บปีกกา)

Dart

```
// สร้าง Set เปล่า
Set<String> uniqueFruits = {};

// สร้าง Set พร้อมข้อมูลเริ่มต้น
Set<String> uniqueFruitsWithData = {'Apple', 'Banana', 'Orange'};

// ถ้าเราพยายามเพิ่มข้อมูลซ้ำ Set จะเก็บไว้เพียงตัวเดียว
Set<String> fruits = {'Apple', 'Banana', 'Apple'};
print(fruits); // Output: {Apple, Banana}
```

การเพิ่ม ลบ และการตรวจสอบข้อมูลใน Set

- **.add(element)**: เพิ่มข้อมูลใหม่เข้าไปใน Set หากข้อมูลนั้นยังไม่มีอยู่

  Dart

  ```
  Set<String> fruits = {'Apple', 'Banana'};
  fruits.add('Grape'); // เพิ่มสำเร็จ
  fruits.add('Apple'); // ไม่มีการเปลี่ยนแปลง เพราะ 'Apple' มีอยู่แล้ว
  print(fruits); // Output: {Apple, Banana, Grape}
  ```

- **.remove(element)**: ลบข้อมูลที่ระบุออกจาก Set

  Dart

  ```
  Set<String> fruits = {'Apple', 'Banana', 'Grape'};
  fruits.remove('Banana');
  print(fruits); // Output: {Apple, Grape}
  ```

- **.contains(element)**: ตรวจสอบว่า Set มีข้อมูลที่ระบุหรือไม่ (คืนค่าเป็น true หรือ false)

  Dart

  ```
  Set<String> fruits = {'Apple', 'Banana'};
  ```