Microsoft STUDY4 Build School

.NET Conf

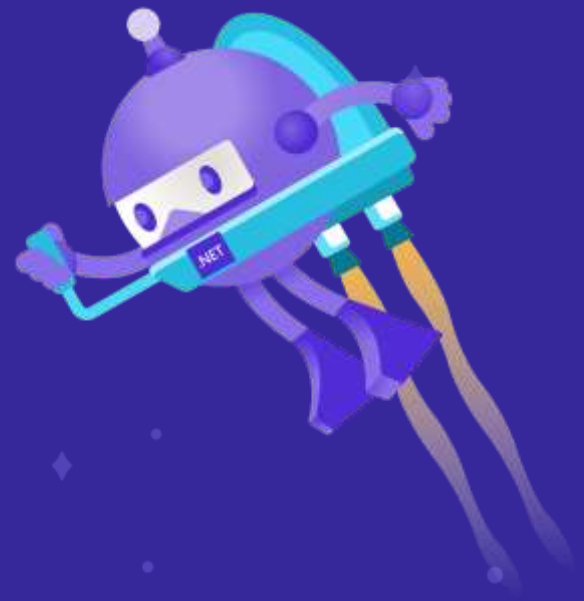
探索 .NET 新世界

Microsoft STUDY4 Build School

刻意練習:
如何鍛鍊你的抽象化思考能力?

// 建構軟體架構團隊的必要過程

Andrew Wu, 91APP / Chief Architect
Fion Yu, 91APP / Software Engineer



Microsoft STUDY4 Build School

講師簡介: Andrew Wu

現職: 91APP Chief Architect

負責帶領: 產品研發聯合處 / 架構設計部, 執行架構改善與規劃；
核心服務設計開發；以及大型整合專案的推動。

91APP

現任: Microsoft MVP (Azure, 2016 ~ 至今)

MVP Microsoft® Most Valuable Professional

專長與個人期許:

談論各種軟體開發與設計的大小事，有 20 年的大型與雲端服務的開發經驗。
喜歡研究各種技術背後的原理與實作細節，期許自己做個優秀的系統架構師。

主題以: .NET / C# / OOP / Container / Cloud Native / DevOps 為主軸，同時在部落格上也持續分享相關主題的一系列文章。期許能將這些實作經驗分享到社群。

https://columns.chicken-house.net/about/

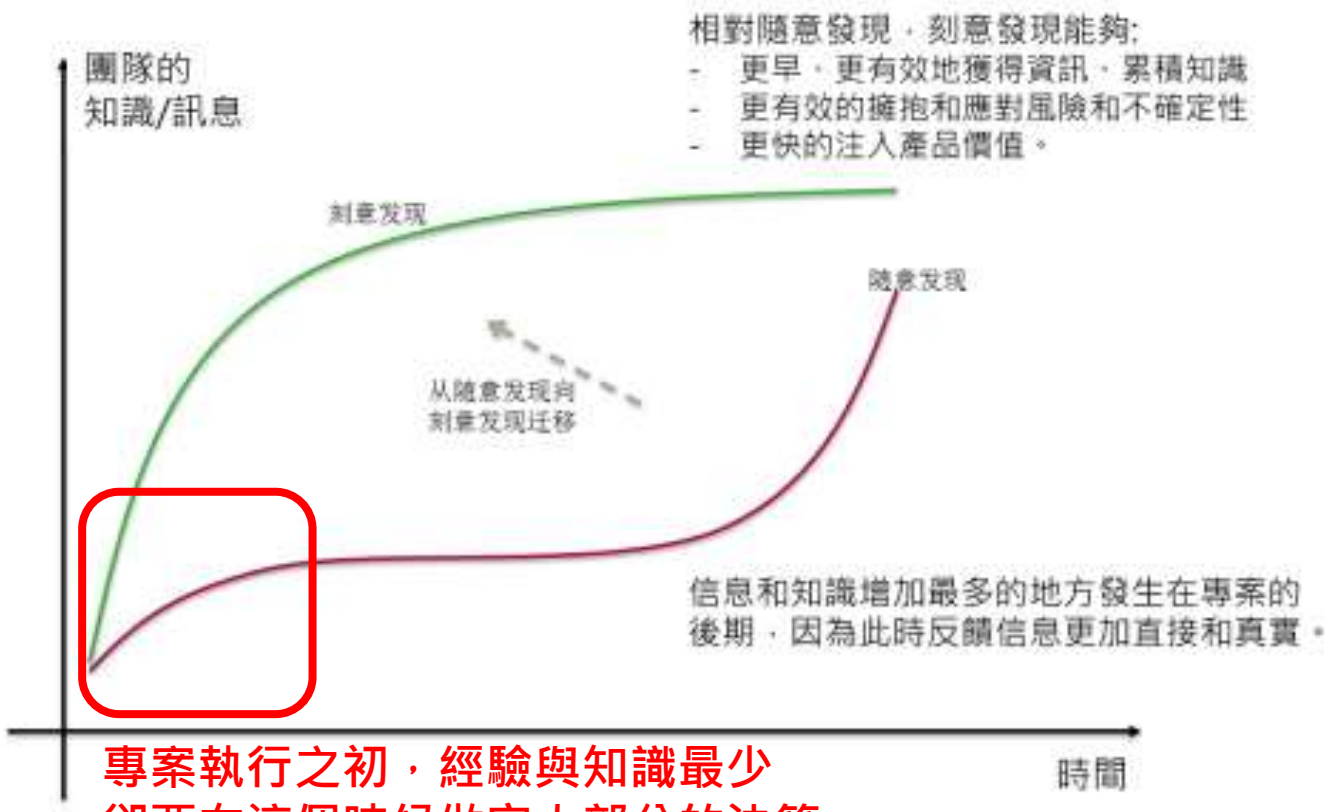
"刻意發現"

Dan North

刻意的發現

Deliberate Discovery

強調了無論是需求還是設計，都要**避免過多的預設計**，而要通過在日常功能開發過程**中不斷地、有紀律地、刻意地去發現，去抽象，去演進**，才會做出好的系統。**反覆運算貫穿整個產品開發的方方面面，是刻意的。**



專案執行之初，經驗與知識最少
卻要在這個時候做完大部分的決策

Ruddy Lee, 2020/08/04 刻意發現Deliberate Discovery,

是你最大的敵人

無知將是你最大的限制!

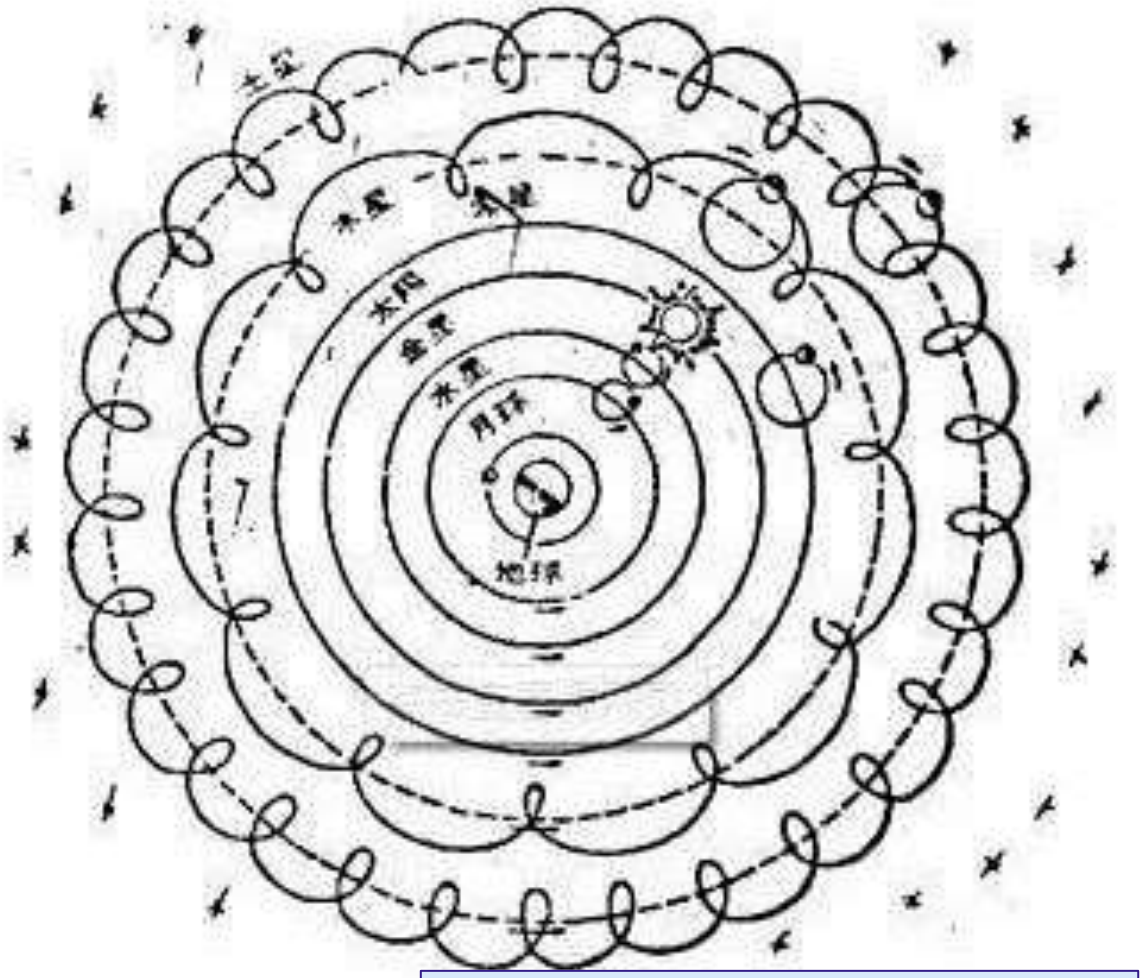
Ignorance is the constraint.

- Dan North; BDD 之父

無知：對目的物缺乏相關資訊或是錯誤的認知。

“好” 的 modeling 有多重要?

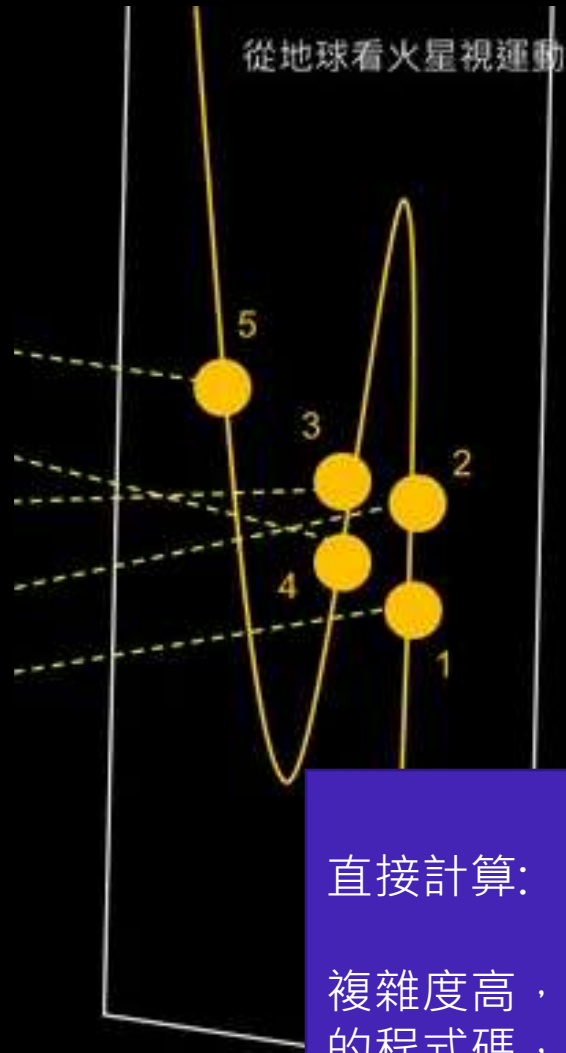
1543 年，哥白尼提出 “日心說”



地心說；宇宙萬物都繞著地球轉

Microsoft STUDY4 Build School

Q: 好的 Modeling 能造福多少開發人員?



簡單的模型:

只要個別計算單純的圓周運動，並且做好相對座標的計算即可解釋複雜的現象。容易理解，簡單，好維護好驗證的程式碼。

直接計算:

複雜度高，又難理解，而且一堆特例的程式碼，難以維護與驗證。

Microsoft STUDY4 Build School

“沒有最好的架構，只有最合適的架構”

// 年度幹話精選 TOP10 ...

// 事實上，大部分的人面對問題，連 “一個” 可行的架構都提不出來...

// 雖然沒有絕對的 “最好”, 但是架構仍然存在著好壞的區別, 設計好架構需要練習...

// 我們需要更有效率的練習方式，來面對未來的問題...

// **你要先有選擇，好壞才有意義**

重點是: 面對問題時, 你能有多少種選擇?

(知識) 首先, 你能 "**想出**" 幾種解決方案?

- **你想的出合理且可行的解決方案嗎?**
- 你對該領域的了解 (business domain knowhow) 有多深入 ?
- 你是否有能力讓團隊成員都了解這些知識?

透過刻意的練習，將知識轉換成你的能力。

(能力) 接著, 你能 "**執行**" 幾種解決方案?

- **你有能力把你想的方案做出來嗎?** 你有執行該解決方案的經驗或能力嗎?
- 你對相關的語言、工具、技術夠熟悉，掌握能力夠高嗎?
- 如果沒有完美的工具或技術，你有能力自行開發嗎?
- 你是否有能力也教會團隊的成員，該如何使用這些工具 or 技術?

刻意練習: 想辦法讓自己的 “進步” 比別人快

- 抽象化思考問題，找出邏輯上可行的解決方案
- 有能力用你手上的工具 (C#, OOP) 實作出來
- 越困難越抽象的問題，越需要透過練習才能掌握技巧
- **化繁為簡的設計能力**

- 有能力解決技術問題 (例如效能，交易正確性等等)
- 加上規模 (大流量情況下) 與時間 (即時處理) 的應變能力
- **最佳化的能力 (通常會讓實作變複雜；別太早最佳化，別讓最佳化破壞抽象化)**

1. 為什麼抽象化很重要?

// 這是我學 OOP 最大的理由

Wikipedia: 抽象與抽象層

抽象就是把一個問題或模型，以不同規則或方法所得出的不同的解（求解方法和解本身即抽象層），這些不同的解可以組合並還原成問題或模型的本身。

抽象的意義是可以忽略不是求解過程中必需的**解**。例如要用電腦程式去類比「人」，在描述了人的動作（飲食、思考、移動等）符合設計要求後（如可完整表達「人」在坐下時候的動作），其他「人」的細節（軀幹、器官、細胞活動乃至人際關係）都可以忽略，以集中設計**需要的功能**，並減低程式的複雜度。

為了使抽象的成品（演算法）不會出現問題，要注意抽象時是否漏掉**重要特徵**。

https://zh.wikipedia.org/wiki/%E6%8A%BD%E8%B1%A1%E5%8C%96_(%E8%A8%88%E7%AE%97%E6%A9%9F%E7%A7%91%E5%A

架構面試題 #4 - 抽象化設計；折扣規則的設計機制 (06/25 補完)

2020/03/10 系列文章: 架構師觀點 系列文章: 架構面試題 系列文章 架構師 C# OOP

5 Comments

Search

Search

Edit Post (Pull Request)

Post Directory

文章目录

https://columns.chicken-house.net/2020/03/10/interview-abstraction/

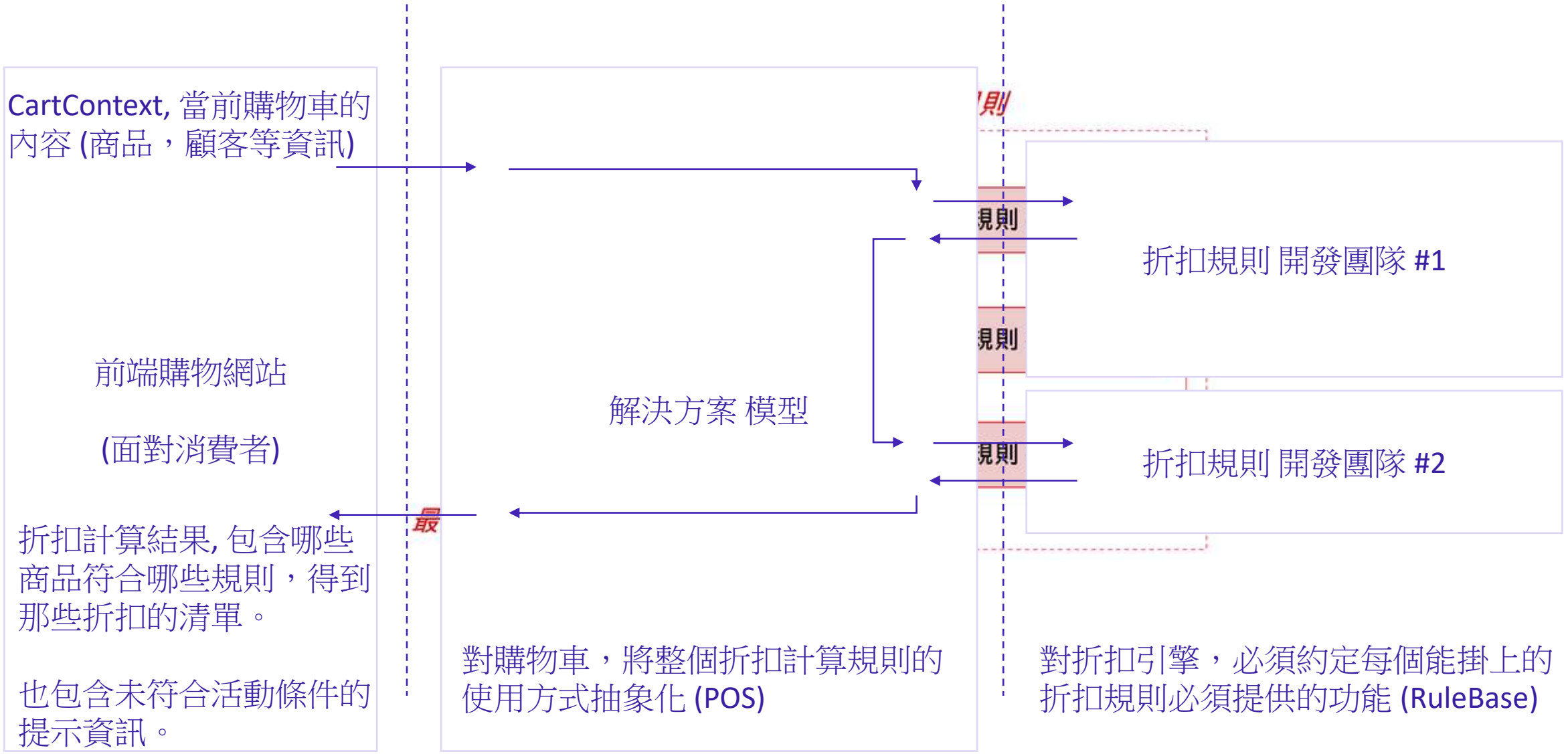
難題: 複雜的問題，該如何系統化的處理?

康寶
雞湯塊/
排骨湯塊
100gx3入/盒
單盒特價145元
第2件5折
平均一盒
108.5元

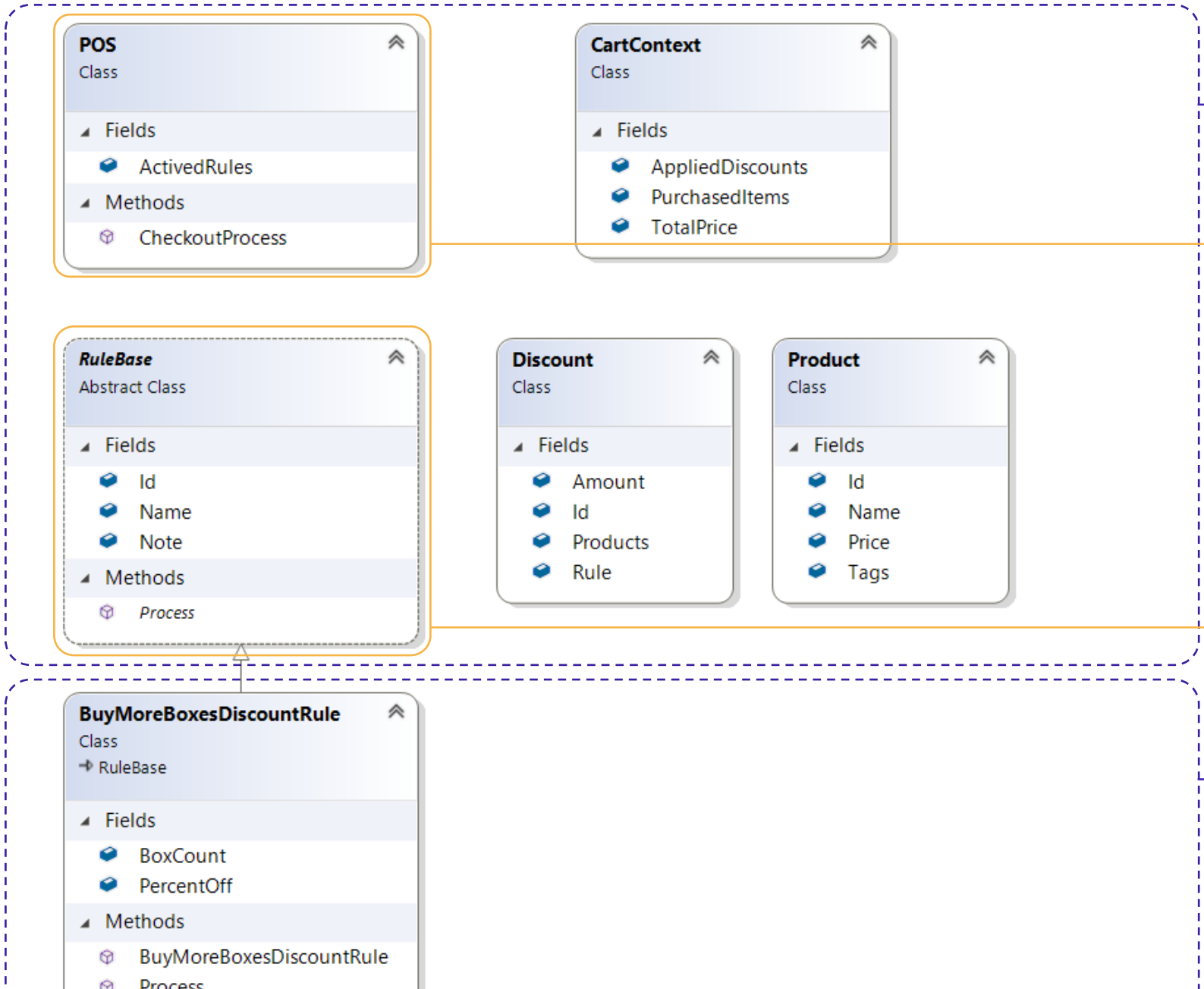
越複雜的問題，你越需要抽象化思考

前面購物車的折扣計算規則中:

1. 解題的模型 是什麼?
2. 需要的功能 是什麼?
3. 不能漏掉的 重要特徵 是什麼?



https://www.91app.tech/blog/Architecture/new-promotion-mechanism/



這些都是抽象化後的成品

前端購物網站開發團隊面對的介面 (interface)

規則開發團隊面對的擴充設計規範 (abstract class)

這些是實作後的細節

Q: 為什麼抽象化很重要?

A: 抽象化能讓你先忽略不必要的細節，專注在要解決的問題核心。

// 先專心思考核心問題，確認要採用哪個解決方案...
// 應該就能定義好 **interface** (有 interface 就可以開始寫測試)
// → 前台團隊就可以開始開發購物網站
// → 專案團隊就可以開始擴充客戶要的折扣規則
// → 補上前面忽略掉的細節跟實作

這些可以同時進行
(因為已經有定義好 interface 了)

// 團隊開始能夠: 用 **top down** 的角度來思考問題
// 團隊開始能夠: 重要的決策先做，而非先挑選細節
// 留意: **抽象化 != 過度設計**

可以拉高成員們思考問題的高度


我到底看了什麼？

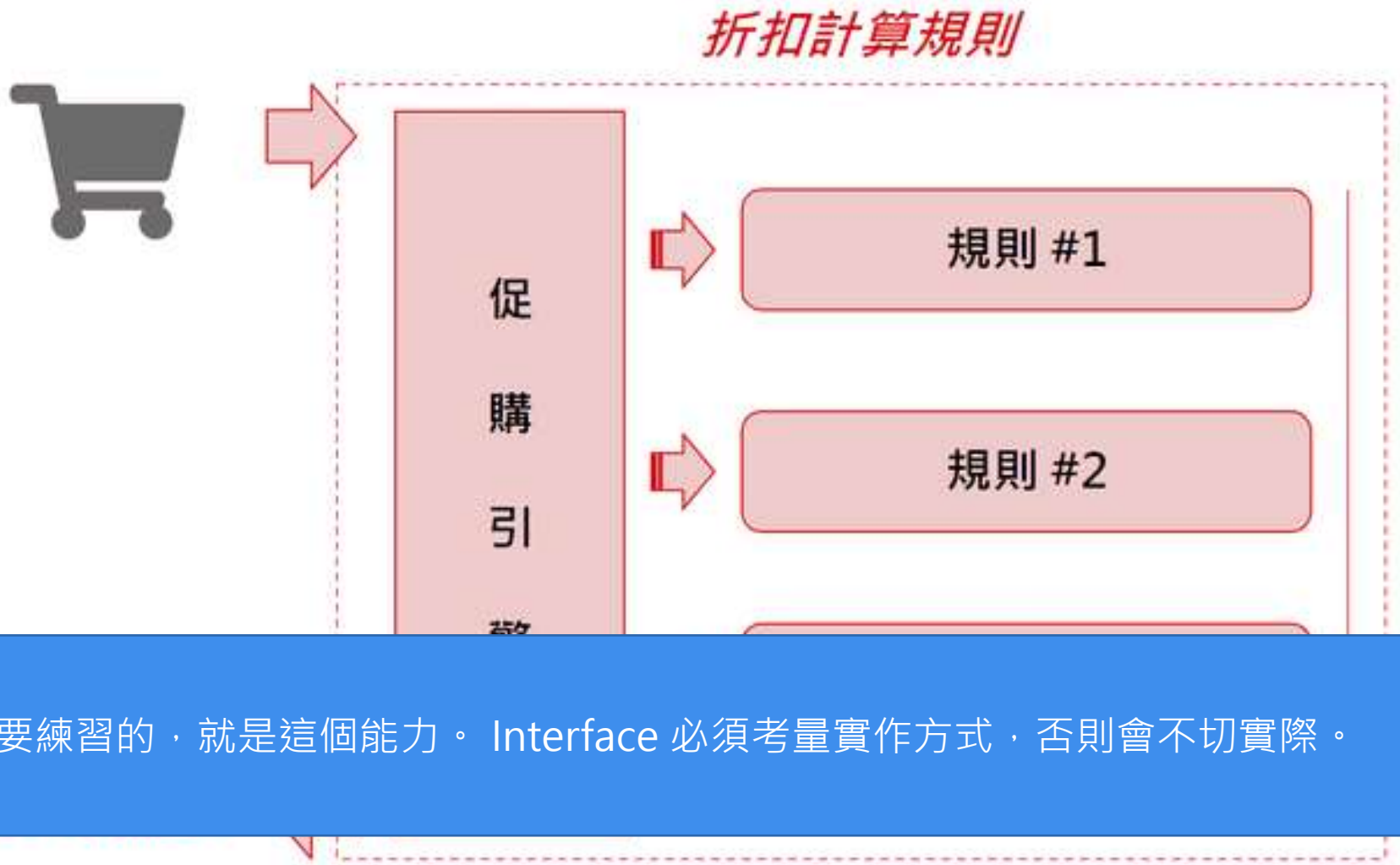
2. 我腦裡想的，真的都能寫成 code 嗎?

// 這是我學 OOP 最大的理由，C# 在這領域是首選

// 真正需要練習的，是這個部分

RECAP

Q: 看懂這張圖之後，你有能力寫成 code 嗎?



真正要練習的，就是這個能力。 Interface 必須考量實作方式，否則會不切實際。

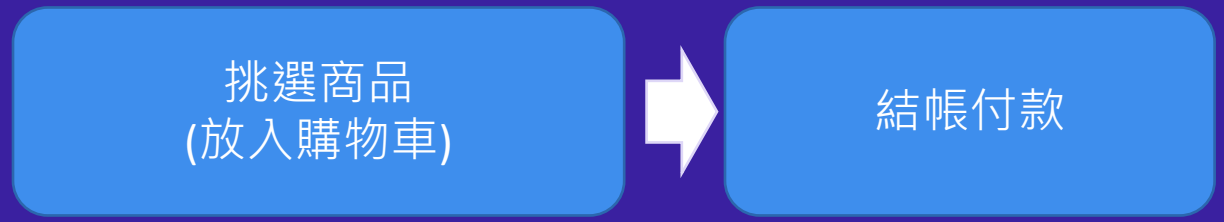
我們團隊實際上的難題

(前面說明過了) 想好解決問題的模型，**第一先定義出 interface ..**

實際應用時，還會碰到這些挑戰:

- 計算過程 (演算法) 能應付實際的資料量嗎? (時間複雜度)
- 計算流程 能應付即時回應的需求嗎?
- 時間驅動 vs 資料驅動 的差異...
- 實際狀況可能會碰到的各種例外處理 (例如退貨，算錯等等)

實際應用上的挑戰...



Q: 你的設計 (modeling) 足夠能應付嗎?

抽象化: 求**解**的過程 => 集中設計**需要的功能** => 保留**重要的特徵**

解: 計算引擎 (POS)

需要的功能:
- 輸入: 購物車 (CartContext)
- 輸出: 結帳結果，結帳提示 (Discount)

重要的特徵:
- 能不斷重複執行
- 輸出結果必須能滿足各階段需求 (加購提示，退貨時如何處理折扣提示...)

刻意練習: 善用 POC 盡早確認設計，不斷修正

獲得經驗的過程，通常是這樣:

1. 我有一個很好的**設計**...
2. 我設計出第一版概念驗證的 code (**POC**) ...
3. 我打造一個 **Prototype** 真正跑完關鍵流程 ...
4. 實際用在真正的開發專案身上，並且把關鍵流程寫成**測試**...

最好的練習方式: 自己找題目，用 (3) 來練習。因為 (3) 是真正能做到隱藏細節，卻又最貼近實際需求的場景。

Q: 刷題 (ex: LeetCode) 是個好的練習方式嗎?

刷題的優點:
- 演算法: 明確的讓你知道你程式碼寫法好不好 (正確性、複雜度)
- 題庫很廣；可以應付 FAANG 的面試 (誤

但是...
- 工作上碰到的應用題，在 LeetCode 上都找不到
- 難題都是 “你該用什麼演算法”，或是 “你該挑什麼演算法” ...

所以: **如果你是 Team Leader, 能否用刷題的形式，讓 Team Member 練習解決工作上的問題?**

3. 我們架構團隊，過去做過哪些“練習”？

為了什麼而練習?

目的: 有效率的處理大量批次作業
- **Process Pool**
- CLI + Pipeline Process
- Parallel Task Execution

目的: 有效率的處理來自 database 的排程任務需求
- Task Scheduling

目的: 抽象化的規則引擎
- POS: Discount Process

練習案例: 平行任務處理的思考練習

主要目標: 是否有確實完成每個 Task? 每個 Task 都必須按照順序完成每個 Step。

品質指標 1: 過程中最大的 WIP。
品質指標 2: 過程中占用的 memory size 最大值。
品質指標 3: 第一個 task 完成所需要花費的時間。
品質指標 4: 所有 Task 處理完畢的時間。

刷題題目:

```
73  public class AndrewTaskRunner : TaskRunnerBase
74  {
75      public override void Run(IEnumerable<MyTask> tasks)
76      {
77          foreach (var task in tasks)
78          {
79              task.DoStepN(1);
80              task.DoStepN(2);
81              task.DoStepN(3);
82          }
83      }
84  }
```

每個 step 背後都有不同的限制:
1. 執行時間不同
2. 允許同時執行的數量不同

Microsoft STUDY4 Build School

成員練習了什麼?

- 先思考問題該怎麼解決?
- 寫出第一版 (先能通過測試再說)
- 持續優化 (思考你的 code 已經 “夠” 好了嗎?)
    - > 切記: “無知” 是你最大的敵人
    - > 你是否知道理論的天花板在哪裡?
- 將你的 solution 送出 Pull Request
- 評比，參考別人的作法，互相討論

RunnerName	WIP_AL	MEM_P	TTFT	TTLT	AVG_WAIT
LexDemo:LexTaskRunner	12	27904	1443.48	174479.45	86654.78
EPDemo:EPTaskRunner	7	21760	1432.09	290307.54	144106.39
SeanDemo:SeanRunner	2	15232	1433.26	867875.35	434657.76
PhoenixDemo:PhoenixTaskRunner	10	26240	1459.45	174511.24	86170.37
JulianDemo:TaskRunner	16	32384	1448.28	174467.72	87715.99
GuiDemo:GuiTaskRunner	3	16896	1432.03	477727.39	234824.64
JWJWTaskRunner#5	12	27904	1436.46	174496.77	85855.17
AndyDemo:AndyTaskRunner	12	27904	1433.35	182791.88	87302.68
MazeDemo:MazeTaskRunner	11	26880	1432.57	174593.59	87746.53
NathanDemo:NathanTaskRunner	12	27904	1432.23	174471.30	87571.79
BorisDemo:BorisTaskRunner	12	27904	1435.82	174442.46	87657.74
JoinDemo:JoinTaskRunner	12	27904	1433.53	174680.88	87885.69
LeviDemo:LeviTaskRunner	9	25216	1531.44	174711.30	88077.96
AndrewDemo:AndrewBasicTaskRunner1	1	13824	1431.43	1430415.55	715922.35
AndrewDemo:AndrewBasicTaskRunner2	1000	1028480	1000370.41	1430364.68	1215376.61
AndrewDemo:AndrewThreadTaskRunner1	11	26880	1438.07	179261.11	87612.93
AndrewDemo:AndrewPipelineTaskRunner1	12	27904	1431.96	174450.21	85880.53
AndrewDemo:AndrewPipelineTaskRunner2	38	41728	1431.63	231005.40	113182.32

成員學到的解決方案有哪些?

- ~~無腦 foreach~~ ...
- Pipeline
- Thread Pool
- TPL (Task Parallel Library)
- Async Task
- Rx.NET (Reactive Extensions for .NET)
- ...

Microsoft STUDY4 Build School

練習案例: 排程任務的處理機制練習

需求: 我有很多排定時間執行的任務，儲存在資料庫內。我需要精準而且節省資源的方式，在預定的時間執行它...

DB 的效能是很昂貴且有限的運算資源，使用輪詢機制處理存在 DB 的排程任務，要在效能(成本)與精確度之間做取捨...。這次練習主要的目的，就是試著找出如何達到下列幾個要求?

- 對於 DB 額外的負擔，越小越好
- 對於每個任務的啟動時間精確度，越高越好
  (延遲越低越好，延遲的變動幅度越低越好)
- 需要支援分散式 / 高可用。排程服務要能執行多套互相備援
- 一個 Job 在任何情況下都不能執行兩次

評量指標

對於排程機制對 DB 額外產生的負擔，我看這些指標最後算出來的分數:

1. 所有查詢待執行任務清單的次數 (權重: 100)
2. 所有嘗試鎖定任務的次數 (權重: 10)
3. 所有查詢指定任務狀態的次數 (權重: 1)

花費成本評分(cost score): (1) x 100 + (2) x 10 + (3)

對於任務實際執行的精確度，我用這些指標最後算出來的分數:

1. 所有任務的平均延遲時間, Avg(ExecuteAt - RunAt)
2. 所有任務的延遲時間標準差, Stdev(ExecuteAt - RunAt)

精確度評分(efficient score): (1) + (2)

成員學到了什麼?

- 透過指標來評量優缺點。
- 評估好 application 跟 infrastructure 的分界。
- 掌握自行開發，處理 scheduling 的關鍵技巧 (modeling)。
- 透過測試找出最佳的組態。

Microsoft STUDY4 Build School

4. 最好的練習: 現在就開始寫 code 吧!

@FionYu

我們練習什麼題目?

1. 暖身: 生命遊戲 (經典計算機科學的題目: 康威生命遊戲)
2. 承上, 改為 OOP 版本 (**封裝**；繼承；多型；)
3. 承上, 加入**時間維度**，每個細胞有各自獨立的時間軸
4. 承上, 每個人的實作都放在同一個世界運行，演化 & 競爭。

這個 “練習” 的目的:

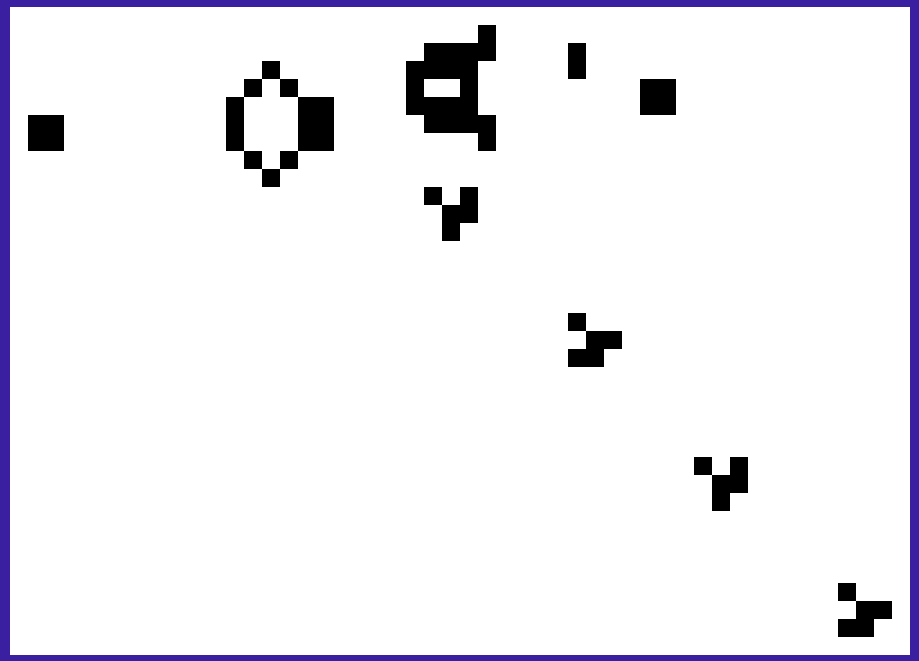
掌握設計 “interface” 的技巧
- 介面設計越簡單越好 vs 功能越多越好
- 介面能取得的資訊越多越好 vs 取得資訊的限制越少越好

找出可行的模型 (modeling)
- 處理時序相關問題的方法
- 正確的封裝，讓同樣目的的 code 集中在同樣的地方

開發技巧:
- 善用你選擇語言的描述能力 (尤其 OOPL 這點更為重要)

練習題目: 康威生命遊戲

- 每個細胞有兩種狀態 - 存活或死亡，每個細胞與以自身為中心的周圍八格細胞產生互動
- 當前細胞為存活狀態時，當周圍的存活細胞低於2個時（不包含2個），該細胞變成死亡狀態。
- 當前細胞為存活狀態時，當周圍有2個或3個存活細胞時，該細胞保持原樣。
- 當前細胞為存活狀態時，當周圍有超過3個存活細胞時，該細胞變成死亡狀態。（模擬生命數量過多）
- 當前細胞為死亡狀態時，當周圍有3個存活細胞時，該細胞變成存活狀態。（模擬繁殖）
- 可以把最初的細胞結構定義為種子，當所有在種子中的細胞同時被以上規則處理後，可以得到第一代細胞圖。按規則繼續處理當前的細胞圖，可以得到下一代的細胞圖，周而復始。



職責歸屬的思考:

封裝 (encapsulation)

- 如何限定每個細胞只 “能” 看到周圍 8 個細胞的狀態?
- 每個細胞 “應該” 知道自己的座標嗎?
- 誰來改變細胞的 “狀態” ? 自己改變? 還是被世界改變?

模型 (modeling)

- 細胞演化的規則，計算的時機，是否該掌握在細胞身上?
- 時間的驅動由誰來負責?
- 如果世界的運行不再是 “回合制” ?
- 如果世界存在不同生存規則的細胞?
- 如果...

Microsoft STUDY4 Build School

< To Fion’s Slides >

5.收穫: 學到抽象化 + 時間序的批次處理作法

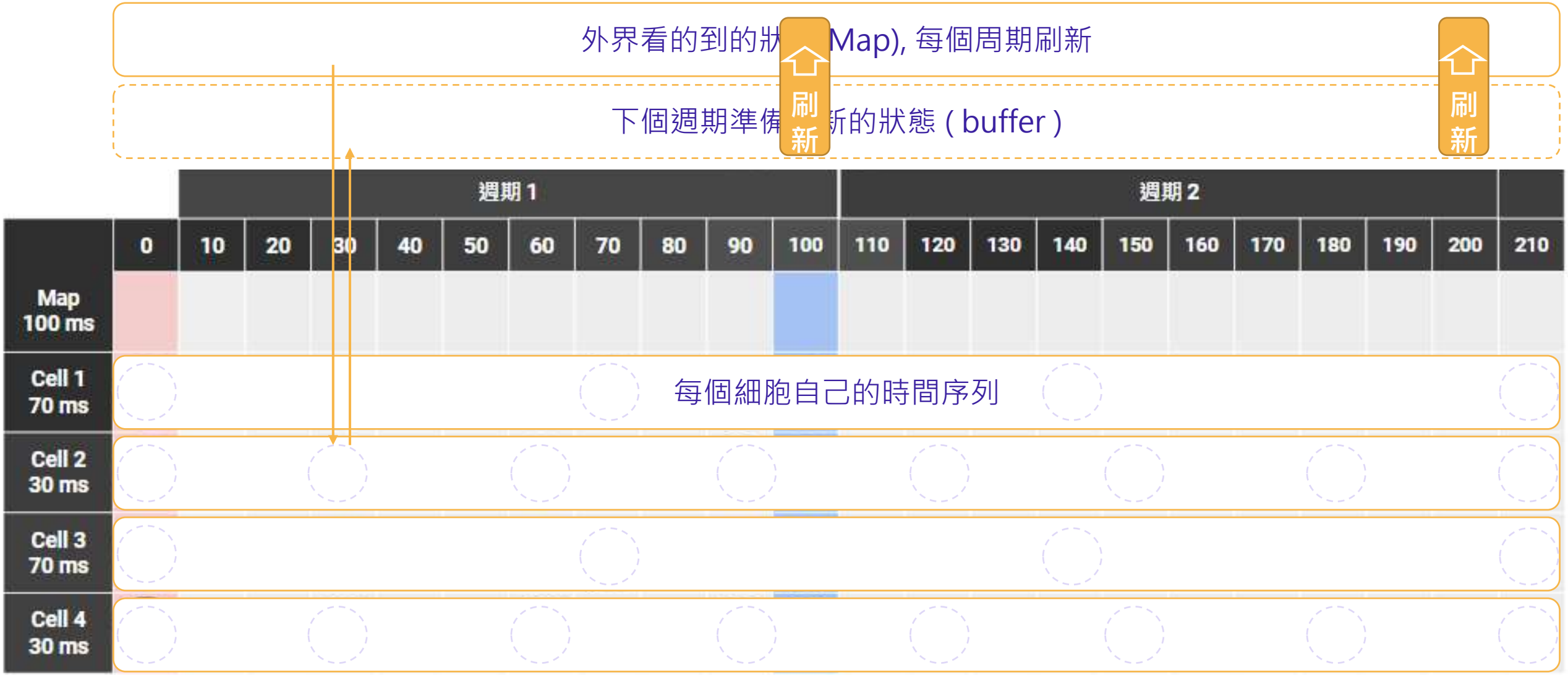
// C# 是很棒的語言，善用語言特性來解決邏輯問題，而非只有最佳化。

// 善用 Andres Hejlsberg 大神的恩賜，能用 C# 來解決問題是幸福的。

最簡單直覺的做法: 用執行緒 + Timer / Sleep

- 用 Event / Timer 控制時間 (缺點: 會將處理邏輯切碎)
- 用 Thread 保有連續的處理邏輯 (缺點: 競爭；耗費資源)

// 平行處理，都隱含了一部分的不確定性 (同步、競爭、鎖定...)
// 在不需要平行處理的情況下，是否有更簡單的作法?



其實: 每個細胞有專屬的 thread, 中間用 sleep(30ms) 就可以搞定了
可是: 我不想開一堆 thread(s), 而且我不要受限於真實世界的時間 (我想要加速世界), 能跑多快就跑多快...

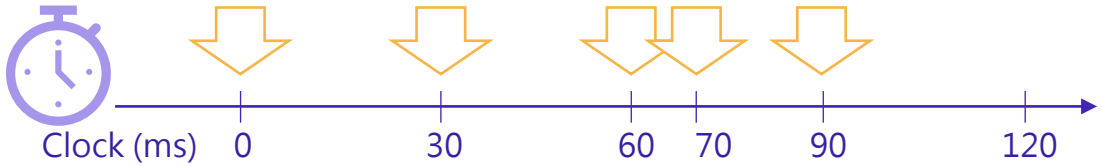
將 Thread + Sleep 換成 C# Yield Return

```
0 references
40  static void Main(string[] args)
41  {
42      var t1 = new Thread(MyLogic);
43      t1.Start("A");
44      t1.Join();
45  }
46
    1 reference
47  static void MyLogic(object context)
48  {
49      string name = context as string;
50
51      Console.WriteLine($"[{DateTime.Now}] doing step {name}#1...");
52      Thread.Sleep(1000);
53
54      Console.WriteLine($"[{DateTime.Now}] doing step {name}#2...");
55      Thread.Sleep(1000);
56
57      Console.WriteLine($"[{DateTime.Now}] doing step {name}#3...");
58      Thread.Sleep(1000);
59  }
```

```
10  static void Main(string[] args)
11  {
12      foreach (var sleep in MyLogic("A")) Thread.Sleep(sleep);
13  }
14
    1 reference
15  static IEnumerable<int> MyLogic(string name)
16  {
17      Console.WriteLine($"[{DateTime.Now}] doing step {name}#1...");
18      yield return 1000;
19
20      Console.WriteLine($"[{DateTime.Now}] doing step {name}#2...");
21      yield return 1000;
22
23      Console.WriteLine($"[{DateTime.Now}] doing step {name}#3...");
24      yield return 1000;
25  }
```

介面定義:
用 yield return {sleep msec} 來代替 Thread.Sleep(...)；將執行 Sleep() 的職責，統一交由 World 集中處理。

我不需要專屬的 thread 就能擁有獨立的處理序列。　　**下一步**: 如何只用一個 World 來協調多個序列 (生命)？



World:

- 30 ms 時叫醒 Cell-A
- 60 ms 時叫醒 Cell-A
- 70 ms 時叫醒 Cell-B
- 90 ms 時叫醒 Cell-A
- 140 ms 時叫醒 Cell-B

Cell-A (週期: 30ms):

- A: +30 ms 後叫我 (age: 30 ms)
- A: +30 ms 後叫我 (age: 60 ms)
- A: +30 ms 後叫我 (age: 90 ms)

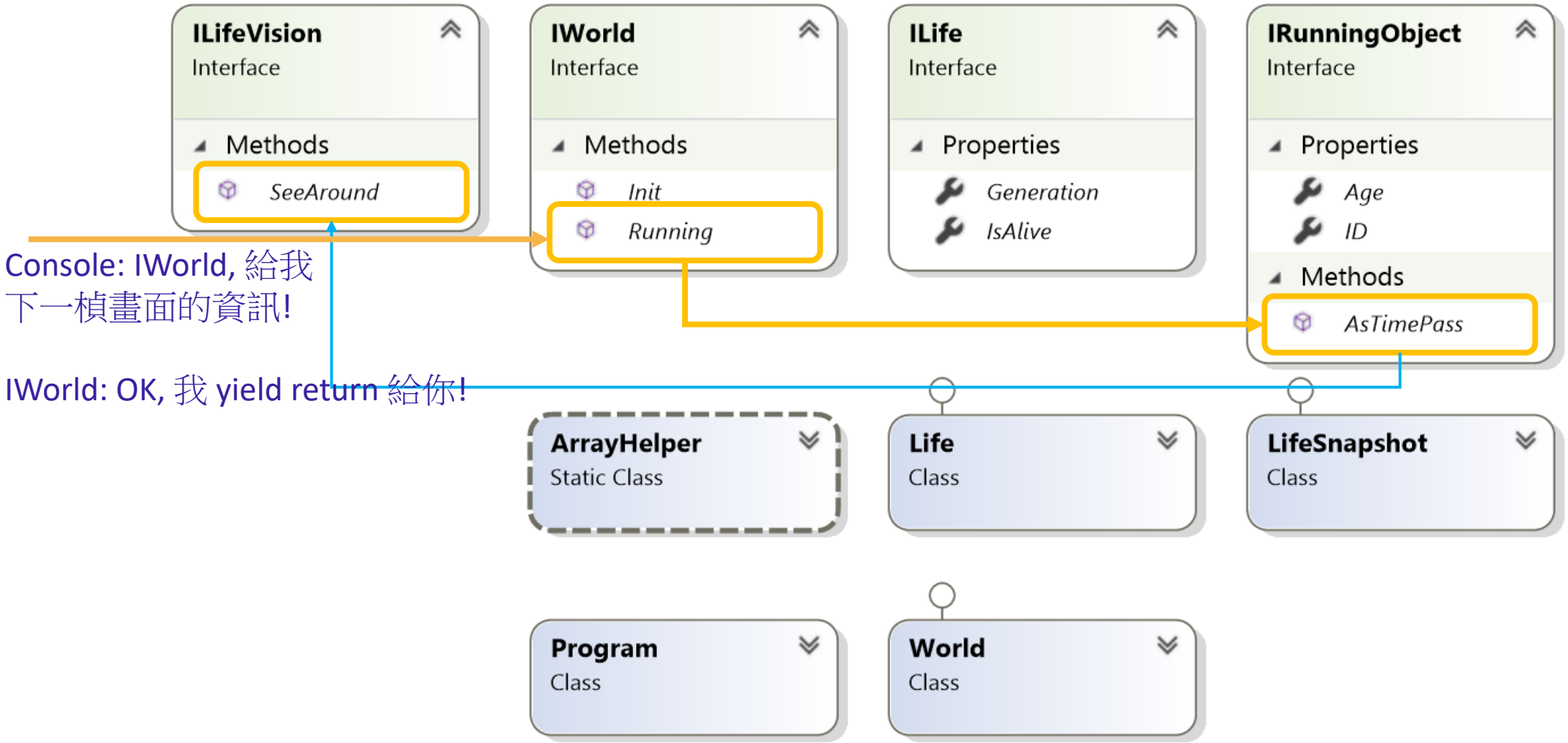
Cell-B (週期: 70ms):

- B: +70 ms 後叫我 (age: 70 ms)
- B: +70 ms 後叫我 (age: 140 ms)

World 每次喚醒，都會接收下次預計喚醒的時間。
統一用 SortedSet<int> 來做簡單的排程機制，隨時更新下一個瞬間要喚醒的對象。

Cell 只需要用 yield return 的方式來傳達 sleep 的資訊

抽象化 (class diagram)



抽象化: 隱藏細節；提取重點

重點是:

1.世界是如何驅動所有的生命的?
IWorld.Running(...)

2.每個生命如何描述自身的行為? 同時配合世界的驅動?
IRunningObject.AsTimePass(...)

3.生命如何 “看” 到世界? 如何提共生命有限度的視覺能力..
ILifeVision.SeeAround(...)

其他都是細節:

從 interface 移除，只保留在 implementation

```
4 references | Andrew Wu, 21 days ago | 1 author, 3 changes
public interface IWorld
{
    4 references | Andrew Wu, 24 days ago | 1 author, 1 change
    public bool Init(bool[,] init_matrix, int[,] init_cell_frame, int[,] init_cell_start_frame, int world_frame);

    4 references | Andrew Wu, 21 days ago | 1 author, 2 changes
    public IEnumerable<(TimeSpan time, ILife[,] matrix)> Running(TimeSpan until, bool realtime = true);
}


15 references | Andrew Wu, 20 days ago | 1 author, 4 changes
public interface ILife
{
    7 references | Andrew Wu, 20 days ago | 1 author, 3 changes
    public int Generation
    {
        get { return 0; }
    }
    15 references | Andrew Wu, 20 days ago | 1 author, 3 changes
    public bool IsAlive { get; }
}
```

```
110         int count = 0;
111         Console.CursorVisible = false;
112
113         Stopwatch realtime_timer = new Stopwatch();
114         realtime_timer.Restart();
115         foreach (var frame in world.Running(until, realtime))
116         {
117             count++;
118             int live_count = 0;
119             Console.SetCursorPosition(0, 0);
120
121             var current_matrix = frame.matrix;
122             var time = frame.time;
123
124             if (display)
125             {
126                 for (int y = 0; y < current_matrix.GetLength(1); y++)
127                 {
128                     for (int x = 0; x < current_matrix.GetLength(0); x++)
129                     {
130                         var c = current_matrix[x, y];
131                         if (c.IsAlive) live_count++;
132                         Console.Write(c.IsAlive ? '★' : '☆');
133                     }
134                     Console.WriteLine();
```

由 foreach loop 來驅動整個世界的運轉

以 下 略

```
21      /// <summary>
22      /// 驅動這整個世界 realtime 演進的所有生物必須實作的 interface.
23      /// 公開的介面代表: 每次演化後透過 yield return 告知環境, 下次演化預計的時間點 ( int, 單位 msec )。
24      ///
25      /// 透過 C# 編譯器支援的 yield return 來實作世界的排程器與每個生命之間的協作。
26      /// </summary>
        10 references | Andrew Wu, 21 days ago | 1 author, 2 changes
27      public interface IRunningObject
28      {
            8 references | Andrew Wu, 21 days ago | 1 author, 1 change
29          public int ID { get; }
            3 references | Andrew Wu, 24 days ago | 1 author, 1 change
30          public IEnumerable<int> AsTimePass();
            12 references | Andrew Wu, 21 days ago | 1 author, 1 change
31          public int Age { get; }
32      }
```

```
4 references | Andrew Wu, 19 days ago | 1 author, 7 changes
111  public IEnumerable<(TimeSpan time, ILife[,] matrix)> Running(TimeSpan until, bool realtime = false)
112  {
113      if (!this._is_init) throw new InvalidOperationException();
114
115      this.RefreshFrame();
116      int until_frames = (int)Math.Min(int.MaxValue, until.TotalMilliseconds);
117
118      SortedSet<RunningObjectRecord> todoset = new SortedSet<RunningObjectRecord>();
119      foreach (var (x, y) in ArrayHelper.ForEachPos<Life.Sensibility>(this._maps_current_life_sense))
120      {
121          var sense = this._maps_current_life_sense[x, y];
122          todoset.Add(new RunningObjectRecord(sense.Itself));
123      }
124      todoset.Add(new RunningObjectRecord(this));
125
126
127      // world start
128      Stopwatch timer = new Stopwatch();
129      timer.Restart();
130      int current_time = 0;
131
132      do
133      {
```

Initialize ToDo List ... (include cell & world)

接下頁

```
132         do
133         {
134             var item = todoset.Min;
135             todoset.Remove(item);
136
137             if (item.Enumerator.Current >= until_frames) break;
138             current_time = item.Enumerator.Current;
139
140             if (item.Enumerator.MoveNext())
141             {
142                 todoset.Add(item);
143                 if (realtime) Spin(() => timer.ElapsedMilliseconds >= current_time);
144             }
145             else
146             {
147                 // yield break. means the life was terminated.
148                 continue;
149             }
150
151             if (item.Source is World)
152             {
153                 Debug.WriteLine($"- running: {current_time}");
154                 this.RefreshFrame();
155                 yield return (TimeSpan.FromMilliseconds(current_time), this._maps_snapshot);
156             }
157
158         } while (true);
159     }
```

取出排在最前面的那一筆資料。

找到該細胞的下一筆，放進 SortedSet 就立刻排序完成。

Get Current, Do Current, Append Next

Microsoft STUDY4 Build School

6. Summary

// 透過練習，你有更充足的準備，思考如何 “做對的事”！

為什麼要 “刻意” 練習? 因為當你越來越資深的時候...

- 思考如何 “**做對的事**” 會越來越重要 (因為你的影響力越來越大)
  // 成語: 將帥無能，累死 ...
  // 如果你是部門主管，或是 Tech Leader，你就是最適合出題的人

- 練習需要時間仔細**思考**；
  // 當你需要的時候通常沒時間...

- 練習需要**排除不必要的細節**；
  //當你埋首在專案內，通常都拋不開細節...

- 問題越~~簡單~~單純，你才會思考的越透徹 (降低互相切磋的門檻)
- 越簡單的設計越困難；簡單 (simple) != 容易 (easy)

Microsoft STUDY4 Build School

我該怎麼開始 “刻意練習”？

適合練習的時間: 從你開始學軟體開發的那一天，或是**今天**。

適合練習的題目: 到處都是題目，自己找題材練習。

- 想辦法把工作上碰到的問題簡化成題目，也是種抽象化的訓練。
- 到我的 **部落格** / **GitHub**, 都看的到過去我們團隊練習的題目與內容。

適合練習的環境: 有能力自己找題目的話，到處都是適合的環境。

- **如果你想找一個這樣的工作環境，外面有 91APP 的攤位** XDD

Microsoft STUDY4 Build School

Thanks for joining!

Ask questions on Twitter using #dotNETConf

91APP 線上考題

91APP Tech 粉絲團

現場或對線上考題有任何問題
歡迎到 91APP 攤位與講師交流 ❤

.NET Conf 2020

特別感謝

91APP Technical Network

Red Hat

KKTIX

JET BRAINS

HackMD

MVP Microsoft® Most Valuable Professional

Microsoft

Build School

STUDY4 為學習而生

以及各位參與活動的你們