เฉลยแบบทดสอบข้อสอบ

ตอนที่ 1. แบบทดสอบปรนัย

❶ ข้อ 1) และ 2)

ข้อ 1) ถูกต้อง เพราะถ้าหากข้อมูลที่ใช้สื่อสารกันอยู่ในรูปของ XML จะสามารถเข้าใจได้ในทุกแพลตฟอร์ม ซึ่งตรงกับเนื้อหาในบทที่ 1. หัวข้อ “ประโยชน์ของ XML อยู่ที่ไหน” ในข้อย่อยที่ 2) ส่วนที่กล่าวว่าระบบนี้พัฒนาด้วยภาษาจาวา ก็อธิบายได้ว่าภาษานี้มี API ที่สามารถจัดการกับเอกสาร XML ได้เป็นอย่างดี ทั้ง DOM และ SAX นั่นเอง ดังนั้นจึงสามารถพัฒนาแอปพลิเคชันมารองรับข้อมูลที่อยู่ในรูป XML ได้เป็นอย่างดี

ข้อ 2) ถูกต้อง ทั้งสองแอปพลิเคชันพัฒนาด้วยภาษาที่ต่างกัน และบนแพลตฟอร์มที่ต่างกัน การแลกเปลี่ยนข้อมูลกันด้วย XML ซึ่งเป็นไฟล์ข้อความธรรมดา (text file) จึงเข้าใจได้ง่ายทั้งสองฝ่าย

ข้อ 3) ผิด เพราะเว็บไซต์ส่วนตัวไม่กี่หน้า ไม่จำเป็นต้องเก็บข้อมูลด้วย XML ก็ได้ เพราะสิ้นเปลืองเวลาเขียนโปรแกรมเปล่าๆ

ข้อ 4) ผิด เพราะเครื่องเมนเฟรมรุ่นเก่า คงจะไม่มีพลังพอที่จะประมวลผล XML ได้ ยิ่งหากใช้ DOM เป็น API ในการจัดการเอกสาร XML ด้วยแล้ว ยิ่งเป็นไปไม่ได้ เพราะ DOM จะใช้หน่วยความจำของเครื่องมาก

❷ ข้อ 1) และ 2)

ทั้ง CSS และ XSLT เป็นวิธีในการแสดงผลเอกสาร XML ส่วน DTD และ XML Schema เป็นวิธีในการกำหนดโครงสร้างเอกสาร XML

❸ ข้อ 4)

XHTML เป็นภาษาที่ถูกพัฒนามาจากภาษา HTML ที่เป็นภาษาที่ใช้แสดงผลบนเว็บเบราเซอร์ และมีแท็กต่างๆ เหมือนกับภาษา HTML (ข้อ 1) จึงผิด) ที่มีตัว X นำหน้า ก็คือเป็นภาษาที่ใช้ไวยากรณ์ของ XML นั่นเอง (ข้อ 3) จึงผิด) ดังนั้น หากมีการเปิดแท็ก ก็ต้องปิดแท็กทุกครั้ง ไม่สามารถละเลยได้ เหมือนภาษา HTML ดังนั้นข้อ 4. จึงถูกต้อง

ข้อ 2) ผิด เพราะภาษา XHTML ไม่ใช่ภาษาที่ใช้ในการเปลี่ยนรูปแบบของเอกสาร

❹ ข้อ 1)

ข้อ 1) เหมาะสมที่สุด เพราะเราเขียนโค้ด XSLT เพียงไม่กี่บรรทัด ก็สามารถแสดงผลเอกสาร XML ซึ่งมีข้อมูลลูกค้า 100 คนได้แล้ว ซึ่งสามารถใช้คำสั่ง for-each นั่นเอง

ส่วนข้อ 2) และ ข้อ 3) ก็สามารถทำได้ แต่เราคงต้องใช้เวลาลงมือทำพอสมควรทีเดียว สุดท้าย ข้อ 4) ผิดแน่นอน เพราะเบราเซอร์จะแสดงผลเอกสาร XML เป็นดังภาพ 1-8 ในบทที่ 1. ซึ่งไม่ใช่ในรูปของตาราง และผู้อ่านก็จะอ่านไม่รู้เรื่องแน่นอน

❺ ข้อ 1)

ข้อนี้ตรงกับภาพ 2-1 ซึ่งบอกว่าส่วนของ Prolog จะอยู่ก่อนเริ่มแท็กของเอกสาร XML นั่นเอง ซึ่ง Prolog ก็คือ XML Declaration และ DTD นั่นเอง

❻ ข้อ 2)

เพราะเอกสารนี้ ไม่มีรูตอิลิเมนต์ ซึ่งเป็นแท็กที่คลุมทุกๆอิลิเมนต์ในเอกสาร XML ซึ่งหากจะแก้ให้ถูกต้อง ผมจะขอสมมติรูตอิลิเมนต์ชื่อ folders ดังตัวอย่าง

```
<?xml version="1.0"?>
<folders>
<folder dt="31/10/2006">
    <file id="1">
        <name>Marketing Summary</name>
        <countents/>
        <!-- Note: This file does not contain anything yet -->
    </file>
</folder>
<folder dt="1/11/2006">
    <file id="2">
        <name>Finacial Summary</name>
        <countents/>
```

```
    <!-- Note: This file does not contain anything yet -->
  </file>
</folder>
</folders>
```

7 ข้อ 3)

บรรทัดที่ 3 ผิด เพราะค่าของแอตทริบิวต์ ต้องมีเครื่องหมาย single quote (‘) หรือ double quote (“) อย่างใดอย่างหนึ่งเสมอ

8 ข้อ 6)

บรรทัดที่ 6 ผิด เพราะคอมเมนต์ ต้องขึ้นต้นด้วย <!-- และลงท้ายด้วย --> ซึ่งจากโจทย์ขึ้นต้นคอมเมนต์ผิด

9 ข้อ 1) และ 3)

ข้อ 1) และ 3) ถูก เนื้อหาส่วนนี้อยู่ในบทที่ 2. หัวข้อ “กฎเกณฑ์เบื้องต้นว่าด้วยเรื่องของอิลิเมนต์” ครับ

ข้อ 1 ทุกอิลิเมนต์ต้องมีชื่อ ซึ่งความจริง ก็คือชื่อแท็กนั่นเอง เช่น แท็ก <person> ก็มีชื่ออิลิเมนต์ว่า person นั่นเอง

ข้อ 2 ผิด ชื่อของอิลิเมนต์ไม่สามารถมีช่องว่างได้ และต้องขึ้นต้นด้วยตัวอักษร หรือ เครื่องหมาย underscore (_)

ข้อ 3 ถูก เพราะจากที่อธิบายในข้อ 2 และขอเพิ่มเติมว่า ชื่อของอิลิเมนต์นั้น ตัวพิมพ์เล็กและพิมพ์ใหญ่ถือว่าเป็นคนละตัวกัน ซึ่งก็คือคุณสมบัติ case-sensitive

10 ข้อ 6)

 ไม่ใช่ Entity Reference เนื้อหาส่วนนี้อยู่ในบทที่ 2. หัวข้อ “กฎเกณฑ์เบื้องต้นว่าด้วยเรื่องของอิลิเมนต์” หัวข้อย่อยที่ 7. กล่าวว่า Entity Reference มี 5 ตัว ซึ่งก็คือตัวเลือก 1)-5) ในโจทย์ข้อนี้นั่นเอง

11 ข้อ 4)

เนื่องจาก CDATA มีไว้เพื่อเก็บข้อความที่ไม่ต้องการให้ XML Parser ประมวลผล โดยข้อความนั้นจะเป็นข้อความใดๆ หรือแม้กระทั่งแท็ก XML ก็ได้ และด้วยสาเหตุที่ไม่ต้องการให้ XML Parser ประมวลผล จึงไม่ต้องประกาศใน DTD

12 ข้อ 3)

เราสามารถนำข้อความที่ไม่ต้องการให้ XML Parser ประมวลผล เช่น อาจมีเครื่องหมาย <, > หรือมีโค้ดของโปรแกรมใดๆ อยู่ นำไปเก็บไว้ในส่วน CDATA ได้ โดยใช้รูปแบบตามข้อ 3 แต่อย่าลืมข้อห้ามอย่างหนึ่ง คือ ข้อความนั้นห้ามมีอักษร]]> อยู่ เพราะอักษรนี้ใช้เมื่อต้องการปิด CDATA เท่านั้น หากเราเผลอมีอักษร 3 ตัวนี้อยู่ในข้อความ เอกสาร XML นี้จะ Invalid ทันที

13 ข้อ 7)

ไม่มีข้อผิดพลาดใดๆ โจทย์ข้อนี้ “หลอก” จะให้เราตอบว่าบรรทัดที่ 2 ผิด โดยในบรรทัดที่ 2 นั้น ใช้เครื่องหมาย single quote (‘) ครอบค่าของแอตทริบิวต์ dt ซึ่งส่วนมากเราจะคุ้นเคยกับเครื่องหมาย double quote (“) ครอบมากกว่า แต่ไวยากรณ์ของ XML สามารถใช้ได้ทั้งสองเครื่องหมาย

14 ข้อ 4)

ในบรรทัดนี้ ดูผิวเผินเหมือนจะไม่ผิด แต่ต้องมองดีๆ ที่แท็กปิดในบรรทัดที่ 4) คือ </ name> จะพบว่ามีช่องว่างอยู่ ซึ่งถือว่าผิดกฎ

⓯ ข้อ 3)

ผู้อ่านลองพลิกดูบทที่ 3 ในตารางที่ 3.1 จะพบว่าข้อมูลชนิด ID ก็มีชื่อว่า ID นั่นเอง

⓰ ข้อ 3)

ผู้อ่านลองพลิกดูบทที่ 3 ในตารางที่ 3.1 จะพบว่าเราจะใช้คีย์เวิร์ด #IMPLIED เมื่อแอตทริบิวต์นั้นจะมีหรือไม่ก็ได้

⓱ ข้อ 4)

ข้อนี้อยู่ในบทที่ 3 เรื่อง “การประกาศค่าเอ็นทิตี้” ในหัวข้อ “Unparsed General Entity” มีบอกไว้ตรงๆ เลยครับ ซึ่งตรงกับข้อ 4 นั่นเอง

⓲ ข้อ 2)

จากบทที่ 3 หัวข้อ “ประกาศค่าแอตทริบิวต์” ได้กำหนดรูปแบบของการประกาศแอตทริบิวต์ไว้ว่า

<!ATTLIST ชื่ออิลิเมนต์ ชื่อแอตทริบิวต์ ชนิดข้อมูลของแอตทริบิวต์ (#REQUIRED|#IMPLIED|#FIXED) ค่าดีฟอลต์ของแอตทริบิวต์>

ซึ่งในโจทย์นี้ ค่าของแอตทริบิวต์ size คือ เลขศูนย์ (0) เราต้องไม่ลืมว่าใน DTD นั้น จะมองทุกอย่างเป็นข้อมูลตัวอักษร (character data) ดังนั้น เลขศูนย์ (0) ก็ต้องประกาศเป็น “0” กำหนดชนิดข้อมูลเป็น CDATA (ข้อมูลตัวอักษร) ดังนั้นเมื่อเปรียบเทียบกับรูปแบบการประกาศแอตทริบิวต์แล้วจึงตรงกับตัวเลือกข้อ 2 อย่างไม่ต้องสงสัย

ขอเสริมอีกนิดว่า ในตัวเลือกข้อ 4) ชนิดข้อมูลของแอตทริบิวต์ประเภท DATA นั้นไม่มีในข้อกำหนด DTD ซึ่งผู้อ่านลองดูในตารางที่ 3-1 ในบทที่ 3 ได้ครับ

⓳ ข้อ 4)

สำหรับข้อ 1) ผิด เพราะ) อิลิเมนต์ sender กำหนดให้เป็น #PCDATA ซึ่งหมายความว่าภายในอิลิเมนต์ต้องมีข้อความ เช่น <sender>Maria</sender> หรือจะไม่มีข้อความใดๆ เลย (เป็นค่าว่าง)ก็ได้ เช่น <sender></sender> หรือ <sender/>

ส่วนข้อ 2)-4) ต้องพลิกไปดูที่บทที่ 3 ในหัวข้อ “รู้แค่ 3 อย่างก็สร้าง DTD ได้แล้ว” ในหัวข้อย่อย “พลิกแพลงการประกาศอิลิเมนต์ โดยระบุ +,*,?” ซึ่งกล่าวไว้ว่า หากระบุเป็น...

+ หมายความว่า จะต้องมีอิลิเมนต์นั้นตั้งแต่ หนึ่งอิลิเมนต์ขึ้นไป (1,2,3,...)

* หมายความว่า จะมีอิลิเมนต์นี้หรือไม่ก็ได้ หรือมีกี่อิลิเมนต์ก็ได้ (0,1,2,3,...)

? หมายความว่า จะมีอิลิเมนต์นั้นอย่างมากเพียง 1 อิลิเมนต์หรือไม่มีเลยก็ได้ (0,1)

ดังนั้น ข้อ 2) ผิด เพราะอิลิเมนต์ attachment มีเครื่องหมาย + จำเป็นต้องมีอย่างน้อย 1 อิลิเมนต์

ข้อ 3) ผิด เพราะ messagelist(message*) หมายความว่าภายในอิลิเมนต์ messagelist จะมีอิลิเมนต์ message หรือไม่ก็ได้ หากมี จะมีกี่อิลิเมนต์ก็ได้

จากเหตุผลในข้อ 2) ทำให้ ข้อ 4) จึงเป็นคำตอบที่ถูก

⓴ ข้อ 3)

สำหรับข้อ 1 ผิด เพราะ) อิลิเมนต์ sender กำหนดให้เป็น #PCDATA ซึ่งหมายความว่าภายในอิลิเมนต์ต้องมีข้อความ เช่น <sender>Maria</sender> หรือจะไม่มีข้อความใดๆ เลย (เป็นค่าว่าง) ก็ได้ เช่น <sender></sender> หรือ <sender/>

ข้อ 2) ผิด เพราะถ้าห้ามไม่ให้อิลิเมนต์ body และ อิลิเมนต์ attachment ปรากฏพร้อมกัน ในบรรทัดที่ 3 ต้องมีระบุว่า body | attachment ซึ่งจากโจทย์ข้างต้นไม่มีคำสั่งนี้แต่อย่างใด

ข้อ 3) ถูก เพราะอิลิเมนต์ attachment ถูกกำหนดให้เป็น any ซึ่งหมายความว่าภายในอิลิเมนต์จะมีอะไรก็ได้ ไม่ว่าจะเป็นข้อความธรรมดาหรืออิลิเมนต์อื่น

ข้อ 4) ผิด เพราะในบรรทัดที่ 3 ข้างหลังอิลิเมนต์ receiver มีเครื่องหมาย * แสดงว่าจะมีอิลิเมนต์นี้หรือไม่ก็ได้ หรือมีกี่อิลิเมนต์ก็ได้ (0,1,2,3,...) ดังนั้นในข้อนี้ที่กล่าวว่าต้องมีอิลิเมนต์ receiver ตั้งแต่ 1 อิลิเมนต์ขึ้นไป จึงผิด

21 ข้อ 1)

ก่อนที่จะดูตัวเลือกทั้ง 4 ข้อ เราลองมาแปลความหมายจาก DTD ของเอกสาร XML นี้ กันก่อนครับ เริ่มทีละบรรทัด

<!DOCTYPE folder [แปลว่า เอกสารนี้มีรูตอิลิเมนต์ ชื่อ folder ซึ่งก็หมายความว่าแท็ก <folder> ต้องคลุมแท็กอื่นๆทั้งเอกสาร

<!ELEMENT folder(file)*> แปลว่า ภายในแท็ก <folder> จะมีแท็ก <file> ได้ไม่จำกัด หรือไม่มีเลยก็ได้ เพราะมีเครื่องหมาย * อยู่

<!ELEMENT file(name?,contents+)> แปลว่า ภายในแท็ก <file> จะมีแท็ก <name> ได้อย่างมาก 1 แท็ก หรือไม่มีเลยก็ได้ (เพราะมีเครื่องหมาย ?) และต้องมีแท็ก <contents> ตั้งแต่ 1 แท็กขึ้นไป (เพราะมีเครื่องหมาย +)

<!ELEMENT name(#PCDATA)> แปลว่า ข้อความภายในแท็ก <name> ต้องเป็นข้อมูลชนิดตัวอักษร (เพราะกำหนดให้เป็นชนิดข้อมูล #PCDATA) หรือจะไม่มีข้อความใดๆก็ได้ เช่น <name></name> หรือ <name/>

<!ELEMENT contents(#PCDATA)> แปลว่า ภายในแท็ก <contents> ต้องเป็นข้อมูลชนิดตัวอักษร (เพราะกำหนดให้เป็นชนิดข้อมูล #PCDATA) หรือจะไม่มีข้อความใดๆก็ได้ เช่น <contents></contents> หรือ <contents/>

เมื่อเราลองแปลความหมายของ DTD ได้ดังข้างต้นแล้ว จะสรุปได้เลยครับว่า ข้อ 1 เพียงข้อเดียวเท่านั้นที่ถูก

22 ข้อ 2)

สำหรับข้อ 1 ผิด เพราะ) อิลิเมนต์ sender กำหนดให้เป็น #PCDATA ซึ่งหมายความว่าภายในอิลิเมนต์ต้องมีข้อความ เช่น <sender>Maria</sender> หรือจะไม่มีข้อความใดๆ เลย (เป็นค่าว่าง)ก็ได้ เช่น <sender></sender> หรือ <sender/>

ข้อ 2 ถูก เนื่องจากคำสั่งที่ว่า body|attachment ในบรรทัดที่ 3 หมายความว่าจะต้องมีอิลิเมนต์ใดอิลิเมนต์หนึ่งเท่านั้น ซึ่งก็หมายความว่าไม่สามารถปรากฏพร้อมกันได้นั่นเอง

ข้อ 3 ผิด สืบเนื่องจากข้อ 2 อิลิเมนต์ body อาจจะไม่สามารถปรากฏเลยก็ได้ หากมีอิลิเมนต์ attachment

ข้อ 4 ผิด เนื่องจากคำสั่ง messagelist(message*) ในบรรทัดที่ 2 หมายความว่าภายในอิลิเมนต์ messagelist สามารถมีอิลิเมนต์ message กี่อิลิเมนต์ก็ได้ หรือไม่มีเลยก็ได้ (เพราะมีเครื่องหมาย *)

23 ข้อ 3) และ 5)

ข้อนี้ตรงกับเรื่อง XML Schema ในบทที่ 4 ซึ่งปรากฏในหัวข้อ “Simple Type และ Complex Type” มีคำอธิบายชัดเจนว่า Simple Type เป็นอิลิเมนต์ที่มีข้อมูลภายในเป็นข้อมูลพื้นฐาน เช่น สตริง, ตัวเลข เป็นต้น ส่วน Complex Type คืออิลิเมนต์ที่มีแอตทริบิวต์ หรือมีข้อมูลภายในเป็นอิลิเมนต์อื่นๆ

ดังนั้น อิลิเมนต์ name และ อิลิเมนต์ report มีข้อมูลภายในเป็นข้อความธรรมดา หรือสตริงนั่นเอง จึงเป็นอิลิเมนต์แบบ Simple Type ส่วนอิลิเมนต์

folder, file, contents ล้วนมีอิลิเมนต์อื่นอยู่ภายในทั้งสิ้น จึงจัดเป็นอิลิเมนต์แบบ Complex Type

㉔ ข้อ 3)

ในตาราง 3-1 ในบทที่ 3 มีอธิบายแอตทริบิวต์ประเภท NMTOKEN อยู่แล้วครับ คือมีความคล้ายกับข้อมูลประเภท CDATA (อักษร หรือข้อความปกติ) แต่มีข้อจำกัดคือห้ามมีช่องว่าง และต้องขึ้นต้นตัวอักษร หรือ underscore (_) แล้วตามด้วยตัวอักษร ตัวเลข และเครื่องหมายอื่นๆ ซึ่งตามคำนิยามตรงกับข้อ 3

㉕ ข้อ 4)

เนื่องจากแอตทริบิวต์ประเภท NMTOKEN มีนิยามคือ มีความคล้ายกับข้อมูลประเภท CDATA (อักษร หรือข้อความปกติ) แต่มีข้อจำกัดคือห้ามมีช่องว่าง และต้องขึ้นต้นตัวอักษร หรือ underscore (_) แล้วตามด้วยตัวอักษร ตัวเลข และเครื่องหมายอื่นๆ

ในตาราง 3-1 ในบทที่ 3 มีคำอธิบายแอตทริบิวต์ประเภท NMTOKENS ว่าคือข้อมูลประเภท NMTOKEN มากกว่า 1 ตัวขึ้นไป โดยมีช่องว่างเป็นตัวคั่นระหว่าง NMTOKEN แต่ละตัว ดังนั้นจึงตรงกับตัวเลือกข้อ 4

㉖ ข้อ 4)

ภาษาที่ใช้ไวยากรณ์ของ XML มักจะมีเนมสเปซเสมอครับ ดังนั้น ทั้งภาษา XHTML และภาษา XSL จึงต้องมีเนมสเปซด้วย

ในบทที่ 4 หัวข้อ “ไขข้อข้องใจ อะไรคือเนมสเปซ” ได้มีคำอธิบายอย่างละเอียดแล้ว ซึ่งสรุปว่า เนมสเปซใช้ป้องกันความสับสนในการระบุชื่อองค์ประกอบใดๆ ในเอกสาร XML (เช่น แท็ก, แอตทริบิวต์ เป็นต้น) ที่มาจากเอกสาร XML คนละไฟล์ ซึ่งรวมไปถึงมาจาก XML Vocabulary ต่างๆ กัน

ดังนั้นจึงถูกทุกข้อ

㉗ ข้อ 2)

ในบทที่ 4 หัวข้อ “ไขข้อข้องใจ อะไรคือเนมสเปซ” ได้มีคำอธิบายอย่างละเอียดแล้ว ซึ่งสรุปว่า เนมสเปซใช้ป้องกันความสับสนในการระบุชื่อองค์ประกอบใดๆ ในเอกสาร XML (เช่น แท็ก, แอตทริบิวต์ เป็นต้น) ที่มาจากเอกสาร XML คนละไฟล์ ซึ่งรวมไปถึงมาจาก XML Vocabulary ต่างๆ กันด้วยครับ

ดังนั้นข้อ 2 จึงถูก ส่วนตัวเลือกอื่นๆ ไม่ใช่เรื่องของเนมสเปซเลยครับ

㉘ ข้อ 4)

จากตาราง 3-1 ในบทที่ 3 ได้มีคำอธิบายสำหรับแอตทริบิวต์ประเภท IDREFS ว่าเป็นข้อมูล ID ของอิลิเมนต์อื่น ซึ่งอ้างถึงอิลิเมนต์อื่นได้ตั้งแต่ 1 ตัวขึ้นไป นอกจากนี้หากลองดูการประยุกต์ใช้ในซอร์ซโค้ดที่ 3-4 จะพบว่าในการใช้งาน จะใช้ช่องว่าง (white space) เป็นตัวแบ่ง ID แต่ละตัว ซึ่งสรุปแล้วตรงกันตัวเลือกข้อ 4 ครับ

㉙ ข้อ 2)

จากหัวข้อ “ประกาศอิลิเมนต์” ในบทที่ 3 ลองดูที่หัวข้อย่อย “ข้อความธรรมดา – ต้องระบุ PCDATA” จะพบว่ารูปแบบการประกาศอิลิเมนต์จะตรงกับตัวเลือกข้อ 2) ครับ

㉚ ข้อ 4)

จากบทที่ 4 ในหัวข้อ “ข้อดีของ XML Schema ที่เหนือกว่า DTD” กล่าวไว้โดยละเอียดและสมบูรณ์แล้วครับ ซึ่งคือรวมถึงตัวเลือกข้อ 1)-3) นั่นเอง

31 ข้อ 4)

อิลิเมนต์ใน XML Schema มีมากมายครับ กล่าวได้ไม่หมด แต่หากท่านผู้อ่านดูในบทที่ 4 โดยเฉพาะในซอร์ซโค้ดที่ 4-2 จะพบว่ามีแท็ก <complexType> ใช้ในการกำหนดโครงสร้างของ Complex Type แต่ละตัว มีแท็ก <attribute> ใช้ในการกำหนดแอตทริบิวต์ orderDate ในบรรทัดที่ 17 และมีแท็ก <element> อยู่จำนวนมาก ใช้ในการกำหนดลักษณะของอิลิเมนต์แต่ละตัว ดังนั้น ทั้ง 3 ตัวนี้ ก็ล้วนเป็นอิลิเมนต์ใน XML Schema ครับ

32 ข้อ 2)

รูปแบบของการประกาศเนมสเปซใน XML คือ

< ตัวย่อ prefix:ชื่ออิลิเมนต์ xmlns:ตัวย่อ prefix="ชื่อเนมสเปซ">

ซึ่งจากตัวเลือกทั้ง 4 พอจับทางได้ว่า ตัวย่อ prefix ในที่นี้คือ hns ชื่ออิลิเมนต์คือ body และชื่อเนมสเปซคือ http://www.w3.org/TR/html4/ เมื่อลองแทนค่าลงในรูปแบบข้างต้น จะเห็นได้ว่าคำตอบที่ถูกต้องคือข้อ 2

33 ข้อ 3)

จากบทที่ 3 ในหัวข้อ "ประกาศ DTD ในเอกสาร XML" หัวข้อย่อย "ประกาศแบบ External DTD" จะพบว่ารูปแบบเป็นไปตามตัวเลือกที่ 3

34 ข้อ 1)

จากบทที่ 3 หัวข้อ "ประกาศค่าแอตทริบิวต์" ได้กำหนดรูปแบบของการประกาศแอตทริบิวต์ไว้ว่า

<!ATTLIST ชื่ออิลิเมนต์ ชื่อแอตทริบิวต์ ชนิดข้อมูลของแอตทริบิวต์ (#REQUIRED|#IMPLIED|#FIXED) ค่าปกติของแอตทริบิวต์>

ซึ่งในโจทย์นี้ ค่าของแอตทริบิวต์ size คือ เลขศูนย์ (0) เราต้องไม่ลืมว่า ใน DTD นั้น จะมองทุกอย่างเป็นข้อมูลตัวอักษร (character data) ดังนั้น

เลขศูนย์ (0) ก็ต้องประกาศเป็น "0" กำหนดชนิดข้อมูลเป็น CDATA (ข้อมูลตัวอักษร) ดังนั้นเมื่อเปรียบเทียบกับรูปแบบการประกาศแอตทริบิวต์แล้ว จึงตรงกับตัวเลือกข้อ 1)

ขอเสริมอีกนิดว่า ในตัวเลือกข้อ 4 ชนิดข้อมูลของแอตทริบิวต์ประเภท DATA นั้นไม่มีในข้อกำหนด DTD ซึ่งผู้อ่านลองดูในตารางที่ 3-1 ในบทที่ 3 ได้ครับ

ส่วนตัวเลือกข้อ 2) ผิดเพราะ ค่าปกติต้องมีเครื่องหมาย single quote หรือ double quote ด้วย

35 ข้อ 1)

จากตารางที่ 8-1 ในบทที่ 8 จะพบคำอธิบายถึงเครื่องหมายกำหนดแพตเทิร์น จากนั้นก็มีตัวอย่างเสริมความเข้าใจ ซึ่งจากโจทย์ข้อนี้ ตรงกับตัวอย่างที่ (1) พอดี ซึ่งเมื่อเทียบกับโจทย์ข้อนี้ อธิบายว่าเป็นการเข้าถึงอิลิเมนต์ email ซึ่งเป็นอิลิเมนต์หลานของอิลิเมนต์ contacts หรือพูดอีกนัยหนึ่งว่าเป็นอิลิเมนต์ซึ่งอยู่ต่ำลงไปอีก 2 ระดับ นั่นเอง

36 ข้อ 1)

จากเนื้อหาในบทที่ 8 จะพบว่าซอร์ซโค้ดเกี่ยวกับ XSLT ทุกโค้ด จะมีแท็ก <stylesheet> คลุมแท็กทุกแท็กของ XSLT ดังนั้นอิลิเมนต์ stylesheet จึงเป็นรูตอิลิเมนต์

37 ข้อ 3)

select นั้นเป็นแอตทริบิวต์ ไม่ใช่อิลิเมนต์ โดยในบทที่ 8-9 จะพบว่ามีการใช้แอตทริบิวต์ select บ่อยๆ ใน 2 ในอิลิเมนต์ คือ อิลิเมนต์ apply-templates และ อิลิเมนต์ value-of

ส่วนตัวเลือกข้ออื่นๆ เป็นอิลิเมนต์ทั้งหมด

38 ข้อ 4)

จากตารางที่ 8-1 ในบทที่ 8 จะมีคำอธิบายถึงเครื่องหมายกำหนดแพตเทิร์น ทั้ง 3 อยู่แล้วครับ ทุกข้อถูกหมด

39 ข้อ 3)

โดยปกติการประกาศตัวแปรด้วยอิลิเมนต์ variable จะต้องมีแอตทริบิวต์ name เพื่อระบุชื่อตัวแปรเสมอ โดยสามารถประกาศได้ 2 แบบ คือ

เก็บค่าของตัวแปรไว้ในแท็กเปิด-แท็กปิดของแท็ก variable คือ

<xsl:variable name="ชื่อตัวแปร">ค่าคงที่</xsl:variable>

เก็บค่าของตัวแปรไว้ในแอตทริบิวต์ select ของแท็ก variable ซึ่งเป็นแท็กว่าง

<xsl:variable name="ชื่อตัวแปร" select="ค่าคงที่"/>

จะเห็นได้ว่าทั้ง 4 ตัวเลือกใกล้เคียงกับเป็นแบบที่ 2. ดังนั้น ตัวเลือก 1) และ 3) จึงเป็นไปได้ เพราะมีทั้งแอตทริบิวต์ name และ select

แต่โจทย์บอกว่าค่าของตัวแปร x เป็นสตริง "a" จึงต้องเลือกตัวเลือกที่ 3) เพราะมีการใช้เครื่องหมาย single quote ครอบค่า a อีกชั้นหนึ่งเพราะให้รู้ว่าเป็นสตริง

40 ข้อ 1)

เครื่องหมาย @ จะใช้ระบุนำหน้าแอตทริบิวต์ที่เราต้องการพิจารณา ซึ่งค่าของแอตทริบิวต์จะอยู่ในเครื่องหมาย single quote หรือ double quote ดังนั้นคำตอบจึงเป็นตัวเลือกข้อ 1

41 ข้อ 1)

เครื่องหมาย @ จะใช้ระบุนำหน้าแอตทริบิวต์ที่เราต้องการพิจารณา ซึ่งค่าของแอตทริบิวต์จะอยู่ในเครื่องหมาย single quote หรือ double quote นอกจากนี้ ยังต้องใช้เครื่องหมาย [และ] ล้อมรอบส่วนที่เป็นแพทเทิร์นของแอตทริบิวต์ด้วย ดังนั้นคำตอบจึงเป็นตัวเลือกข้อ 1

42 ข้อ 3)

จากตารางที่ 8-1 ในบทที่ 8 จะพบคำอธิบายถึงเครื่องหมายกำหนดแพตเทิร์น // ว่าเป็นการเข้าถึงอิลิเมนต์ที่อยู่ต่ำลงไปกี่ชั้นก็ได้ ซึ่งตรงกับตัวเลือกข้อ 3

43 ข้อ 3)

บรรทัดที่ 3 ผิดเพราะ อิลิเมนต์ value-of จะใช้ร่วมกับแอตทริบิวต์ select ในการนำค่าของอิลิเมนต์มาแสดง ซึ่งที่ถูกต้องต้องเป็น <xsl:value-of select ="John"/>

44 ข้อ 3)

เพราะ SAX จะเป็น XML Parser แบบ Event-based ไม่จำเป็นต้องสร้าง โหนดของเอกสาร XML เป็นแผนภูมิต้นไม้ในหน่วยความจำก่อน จึงใช้ทรัพยากรของเครื่องน้อยกว่าและทำงานเร็วกว่า

ข้อ 1 ผิดเพราะโทรศัพท์มือถือไม่สามารถอ่านเอกสาร HTML ได้ หรืออาจจะได้แต่รูปแบบก็จะไม่น่าดูเหมือนในจอคอมพิวเตอร์ เพราะข้อจำกัดของหน้าจอแสดงผล

ข้อ 2 ผิดเพราะเบราเซอร์รุ่นเก่าๆ บางค่าย รวมไปจนถึงโทรศัพท์มือถือไม่สามารถแสดงผลภาษา XML ได้ อีกทั้งลูกค้าคงไม่สะดวกที่จะดูเอกสาร XML เพียวๆ เพราะอ่านไม่เข้าใจ

ข้อ 4 ผิด เพราะการเปลี่ยนจากอิลิเมนต์มาเป็นแอตทริบิวต์ไม่ได้ช่วยให้ขนาดของไฟล์ลดลงไปเท่าใดนัก อีกทั้งยังมีข้อจำกัดในด้านการเก็บข้อมูลอีก เพราะภายในอิลิเมนต์หนึ่ง สามารถมีอิลิเมนต์ที่ชื่อเดียวกันหลายตัวได้ แต่ภายในอิลิเมนต์หนึ่ง ไม่สามารถมีแอตทริบิวต์ที่มีชื่อเหมือนกันมากกว่า 1 ตัวได้ เช่น

เราสามารถเก็บข้อมูลสินค้าไว้ในอิลิเมนต์ได้ ดังนี้

```
<products>
 <product>สินค้า 1</product>
 <product>สินค้า 2</product>
 <product>สินค้า 3</product>
</products>
```

แต่หากเราเปลี่ยนเป็นเก็บไว้ในรูปของแอตทริบิวต์ดังแสดงต่อไปนี้ จะทำไม่ได้ เพราะมีแอตทริบิวต์ที่มีชื่อซ้ำกัน ไม่จัดเป็น well-formed XML

<products product="สินค้า 1" product="สินค้า 2" product="สินค้า 3"/>

แต่หากเรายังยืนยันที่จะเก็บไว้ในรูปของแอตทริบิวต์ต่อไป โดยการเปลี่ยนชื่อแอตทริบิวต์เพื่อเป็นรายชื่อสินค้าแต่ละตัว ดังแสดงต่อไปนี้

<products product1="สินค้า 1" product2="สินค้า 2" product3 ="สินค้า 3"/>

ปัญหาที่ตามมาก็คือจะขาดความยืดหยุ่นของเอกสารนี้โดยทันที เพราะหากอนาคตมีสินค้าชนิดใหม่เกิดขึ้น ก็ต้องเพิ่มแอตทริบิวต์ product4, product5, ... ซึ่งจะเกิดปัญหากับการออกแบบ DTD หรือ XML Schema อย่างมาก รวมไปจนถึงการเขียนโปรแกรมอีกด้วย

45 ข้อ 2)

SAX เป็น API ตัวหนึ่งในการจัดการ, ประมวลผลเอกสาร XML ด้วยวิธีแบบ Event-Driven ซึ่งมีคำอธิบายในบทที่ 9 หัวข้อ "SAX จัดการ XML ด้วยแนวทาง Event-Driven Parser"

46 ข้อ 1)

SAX เป็น API ที่เกิดขึ้นมาจากข้อจำกัดของ DOM ในด้านการใช้พื้นที่หน่วยความจำนวนมากในการประมวล รวมถึงความเร็วในการประมวลผลด้วย ทั้งนี้เพราะ DOM จะสร้างโครงสร้างต้นไม้สมมติของเอกสาร XML ในหน่วยความจำ แต่ SAX จะเข้าถึงแต่ละส่วนของเอกสาร XML โดยการเดินไปในเอกสารทีละจุดโดยจะมีการสร้างอีเวนต์ให้กับจุดต่างๆ ซึ่งเราเรียกการทำงานลักษณะนี้ว่า Event-Driven

47 ข้อ 1) และ 3)

ในบทที่ 9 หัวข้อ "ใช้เหตุผลอะไรในการตัดสินใจเลือก DOM หรือ SAX" ได้มีคำอธิบายและจำแนกเป็นข้อๆ แล้ว สรุปได้ว่าหากต้องการแก้ไขเอกสาร ทั้งเอกสาร รวมถึงการเข้าถึงข้อมูลแบบสุ่ม หรือ random access จะเหมาะกับ Parser ชนิด DOM มากกว่า แต่หากมีข้อจำกัดด้านประสิทธิภาพและหน่วยความจำของแอปพลิเคชัน และต้องการเข้าถึงเอกสารแบบตามลำดับ จะเหมาะกับ Parser ชนิด SAX มากกว่า

48 ข้อ 2) และ 3)

ข้อ 1) ผิด เพราะปกติ XML Parser จะตรวจสอบคุณสมบัติ Valid ให้อยู่แล้ว

ข้อ 2), 3) ถูก เพราะ SAX เป็น API ที่เกิดขึ้นมาจากข้อจำกัดของ DOM ในด้านการใช้พื้นที่หน่วยความจำนวนมากในการประมวล รวมถึงความเร็วใน

การประมวลผลด้วย ทั้งนี้เพราะ DOM จะสร้างโครงสร้างต้นไม้สมมติของเอกสาร XML ในหน่วยความจำ ซึ่งหากเอกสารมีขนาดใหญ่มาก ก็จะกินหน่วยความจำมาก แต่ SAX จะเข้าถึงแต่ละส่วนของเอกสาร XML โดยการเดินไปในเอกสารทีละจุดตามลำดับ (sequential) โดยจะมีการสร้างอีเวนต์ให้กับจุดต่างๆ ซึ่งเราเรียกการทำงานลักษณะนี้ว่า Event-Driven ซึ่งถึงแม้เอกสารจะมีขนาดใหญ่ ก็จะไม่มีผลมากนัก

ข้อ 4) ผิด เพราะการพัฒนาบนแพลตฟอร์มบนภาษาจาวา มี XML Parser ให้เลือกทั้งที่ทำงานด้วยวิธี DOM และ SAX

49 ข้อ 2)

ข้อนี้ต้องนำความรู้ในบทที่ 3 ที่ว่าด้วยเรื่อง DTD และบทที่ 9 ที่ว่าด้วยเรื่อง DOM แบบเจาะลึกมาวิเคราะห์ครับ ในบทที่ 9 หัวข้อ “จัดการทั้งเอกสารด้วยอ็อบเจ็กต์ Document” ส่วนที่เป็นกรอบอธิบายเพิ่มเติม (มีหลอดไฟ) กล่าวว่าโหนดลูก (child node) คือโหนดของอิลิเมนต์ลูก หรือข้อความในอิลิเมนต์เท่านั้น ซึ่งผมได้แสดงตัวอย่างให้ดูว่า

```
<A id= “1”>123
   <B>456</B>
</A>
```

กรณีนี้โหนดลูกของโหนด A จะมี 2 โหนด คือโหนดของข้อความ “123” และโหนด B ซึ่งผู้อ่านลองพลิกไปทบทวนดูได้

ด้วยเหตุนี้ ตัวเลือกข้อ 1 และข้อ 4 จึงสามารถเป็นโหนดลูกของโหนดอิลิเมนต์ได้

และในบทที่ 3 หัวข้อ “ประกาศค่าเอ็นทิตี้” ในซอร์ซโค้ดที่ 3-5 ในบรรทัดที่ 27, 31, 35 คือ <editor>&witty;</editor> ซึ่งในที่นี้ &witty; เรียกว่าเป็น EntityReference ของเอ็นทิตี้ witty ดังนั้น โหนดลูกของโหนดอิลิเมนต์ก็สามารถเป็น EntityReference ได้ ตัวเลือกข้อ 3 จึงยังไม่ใช่คำตอบของเรา

ตัวเลือกข้อ 2 คือ Entity นั้น ไม่ใช่โหนดลูกของโหนดอิลิเมนต์ครับ เพราะในซอร์ซโค้ดที่ 3-5 เอ็นทิตี้ที่ชื่อ witty ซึ่งมีค่าคือ “วิทยา ต่อศรีเจริญ” จะประกาศใน DTD เท่านั้น (ในบรรทัดที่ 13) หากลองดูจะพบว่า Entity จะไม่มีการใช้งานในแท็กใดๆ เลย แต่การนำเอ็นทิตี้มาใช้งานในเอกสาร XML จะทำโดยผ่าน EntityReference ด้วยคำสั่ง &witty; ในบรรทัดที่ 27, 31, 35 นั่นเอง

50 ข้อ 3)

ผู้อ่านอาจตำหนิผมว่า หนังสือเล่มนี้ไม่มีการแนะนำวิธีการเขียนโปรแกรมภาษาจาวาเพื่อประมวลผลเอกสาร XML ด้วย SAX API เลย แต่ผมขอบอกว่า โจทย์ข้อนี้เอาส่วนโค้ดของ SAX มาหลอกให้เรางงเล่นครับ เพราะเราสามารถใช้ความรู้ในบทที่ 3 ที่ว่าด้วยเรื่องของ DTD ในหัวข้อ “ประกาศค่าเอ็นทิตี้” มาแก้โจทย์ปัญหานี้ได้

เราคงต้องเริ่มต้นที่เนื้อความในเอกสาร XML ของเราที่ว่า The Title is “&title;”

ขั้นที่ 1 “&title;” กล่าวถึง เอนทิตี้ที่ชื่อ tilte ซึ่งหากเราเหลือบไปดูที่ DTD จะพบว่าเอ็นทิตี้นี้แทนข้อความ “&option;Earth Moon” ดังนั้นผลลัพธ์ในขั้นนี้คือ The Title is “&option;Earth Moon”

ขั้นที่ 2 จากผลลัพธ์ในขั้นที่ 1 จะพบว่ายังติดเอนทิตี้ที่ชื่อ option อยู่อีกตัว ซึ่งหากเราเหลือบไปดูที่ DTD อีกครั้ง จะพบว่าเอ็นทิตี้นี้แทนข้อความ “Sun” ดังนั้นผลลัพธ์ในขั้นนี้คือ The Title is “Sun Earth Moon”

ได้แล้วครับ ดังนั้นผลลัพธ์ที่พิมพ์ออกมา คือ The Title is “Sun Earth Moon” ตรงกับตัวเลือกข้อ 3

ตอนที่ 2. แบบทดสอบอัตนัย

เฉลยข้อ 1

เฉลยข้อ 1.1

จากบรรทัดที่ 6 ใน DTD ที่ว่า floor (info?, (room | pda)*) แปลว่าภายในอิลิเมนต์ floor จะมีอิลิเมนต์ info อย่างมากได้เพียง 1 อิลิเมนต์ หรือจะไม่มีก็ได้ (เพราะมีเครื่องหมาย ?) จากนั้นตามมาด้วยอิลิเมนต์ room หรือ pda จะกี่อิลิเมนต์ก็ได้ หรือไม่มีก็ได้ (เพราะมีเครื่องหมาย *)

ดังนั้น

a. (info, room, pda, pda, room) ถูกต้อง

b. (room, info, pda) ไม่ถูกต้อง เพราะอิลิเมนต์ info ต้องมาก่อนอิลิเมนต์อื่น

c. (info, info, room) ไม่ถูกต้อง เพราะมีอิลิเมนต์ info มากกว่า 1 ตัว

d. () ถูกต้อง เพราะไม่มีอิลิเมนต์ info ก็ได้ อีกทั้งไม่มีอิลิเมนต์ที่ตามมา คือ room หรือ pda ก็ได้

เฉลยข้อ 1.2

ไม่ได้ (โปรดดูคำอธิบายในเฉลยจากข้อ 1.1)

เฉลยข้อ 1.3

```
<!ATTLIST building name ID #REQUIRED
                  type (public | private) "public">
```

เฉลยข้อ 1.4

ได้ เนื่องจากบรรทัดที่ 10 กำหนดว่าค่าของแอตทริบิวต์ name เป็นข้อมูลตัวอักษร (character data) ซึ่งจะเป็นสตริงอะไรก็ได้ ค่าจะซ้ำกับแอตทริบิวต์ name ตัวอื่นได้ก็ได้ แต่ถ้ากำหนดว่าเป็นข้อมูลชนิด ID จะไม่สามารถซ้ำกับแอตทริบิวต์ name ตัวอื่นได้

เฉลยข้อ 1.5

```
<complexType name="floor_type">
<sequence>
    <element ref="info" minOccurs="0"/>
    <choice maxOccurs="unbounded">
        <element ref="room"/>
        <element ref="pda"/>
    </choice>
</sequence>
</complexType>

<simpleType name="level_type">
<restriction base="integer">
    <minInclusive value="-1"/>
    <maxInclusive value="5"/>
</restriction>
</simpleType>
```

ก่อนอื่นเราต้องมองให้ออกก่อนว่าอิลิเมนต์ floor และแอตทริบิวต์ level เป็นข้อมูลแบบ Simple Type หรือ Complex Type ซึ่งตรงนี้ได้มีคำอธิบายไว้ในบทที่ 4 หัวข้อ "Simple Type และ Complex Type"

จากคำอธิบายในหัวข้อนี้ เราจะรู้ว่าอิลิเมนต์ floor เป็นข้อมูลชนิด Complex Type เพราะเป็นอิลิเมนต์ที่มีข้อมูลภายในเป็นอิลิเมนต์อื่นๆ (จาก DTD บรรทัดที่ 6 กำหนดว่าภายในจะเป็นอิลิเมนต์ info, room หรือ pda) ดังนั้นจึงเป็นที่มาของแท็ก <complexType> ในบรรทัดที่ 1 และ 9

หากจะดูบรรทัดที่ 6 ใน DTD ให้ลึกซึ้ง จะพบว่า floor (info?, (room | pda)*) แปลว่าภายในอิลิเมนต์ floor จะมีอิลิเมนต์ info อย่างมากได้เพียง 1 อิลิเมนต์ หรือจะไม่มีก็ได้ (เพราะมีเครื่องหมาย ?) จากนั้นตามมาด้วยอิลิเมนต์ room หรือ pda จะกี่อิลิเมนต์ก็ได้ หรือไม่มีก็ได้ (เพราะมีเครื่องหมาย *) ตีแผ่ออกมาได้เป็นดังนี้

อิลิเมนต์ลูกต้องเรียงลำดับ จึงต้องใช้อิลิเมนต์ sequence ในบรรทัดที่ 2 และ 8 ครอบอิลิเมนต์ลูกทั้งหมด (ดูรายละเอียดในบทที่ 4 หัวข้อ “เรียงลำดับด้วย sequence และกำหนดให้เลือกตัวใดตัวหนึ่งด้วย choice”)

อิลิเมนต์ลูกตัวแรก คือ info ซึ่งมีเครื่องหมาย ? กำกับอยู่ หมายความว่าจะมีหรือไม่ก็ได้ ถ้าจะมีก็มีได้เพียง 1 ตัวเท่านั้น จึงระบุว่า minOccurs=“0” (ดูรายละเอียดในบทที่ 4 หัวข้อ “กำหนดจำนวนด้วยอิลิเมนต์ minOccurs และ maxOccurs”)

จากนั้นตามมาด้วยอิลิเมนต์ room หรือ pda จึงต้องใช้อิลิเมนต์ choice ครอบอิลิเมนต์ room และ pda ซึ่งทั้ง 2 อิลิเมนต์นี้จะมีกี่อิลิเมนต์ก็ได้ หรือไม่มีก็ได้ จึงต้องกำหนดแอตทริบิวต์ maxOccurs=“unbounded” ลงไปในอิลิเมนต์ choice คำว่า “unbounded” หมายความว่ามีได้ไม่จำกัด (ดูรายละเอียดในบทที่ 4 หัวข้อ “เรียงลำดับด้วย sequence และกำหนดให้เลือกตัวใดตัวหนึ่งด้วย choice”)

สำหรับการสร้าง XML Schema ในส่วนของแอตทริบิวต์ level ซึ่งใน DTD คือบรรทัดที่ 2 กำหนดไว้ว่ามีค่าเป็นจำนวนเต็มมีค่าตั้งแต่ -1 ถึง 5 ลักษณะแบบนี้ใน XML Schema เรียกว่า “facet” ซึ่งอธิบายไว้ในบทที่ 4 หัวข้อ “สร้าง Simple Type ของตัวเองด้วย facet” ซึ่งอธิบายไว้อย่างชัดเจนแล้วว่าจะต้องมีอิลิเมนต์บังคับชื่อ restriction และมีค่าของแอตทริบิวต์ base เป็นชนิดของข้อมูล integer จากนั้นภายในก็กำหนดรูปแบบของข้อมูลลงไป เนื่องจากค่าที่เป็นไปได้มีได้ตั้งแต่ -1 ถึง 5 จึงใช้อิลิเมนต์ minInclusive และ maxInclusive ระบุค่า -1 และ 5 ลงในแอตทริบิวต์ value ตามลำดับ

เฉลยข้อ 1.6

```
<?xml version="1.0" encoding="UTF-8"?>
<canteens>
<canteen name="eatIT">Todays menu is hot tomato soup with bread crumbles!</canteen>
<canteen name="Kantinen">Today we have a buffet with chicken, salmon and veg. lasagne</canteen>
</canteens>
```

* หมายเหตุ: อาจจะไม่มี XML Declaration ก็ได้ (โปรดดูคำอธิบาย)

คำอธิบาย

เราลองมองโค้ด XSLT ในภาพรวมก่อน จะเห็นได้ว่าโค้ด q01.xsl นี้ใช้วิธีการแบ่งงานไปให้ template ย่อยๆ ทำ โดยอิลิเมนต์ apply-templates ในบรรทัดที่ 5 ถือว่าเป็น “ต้นทาง” โดยจะกรองก่อนหนึ่งชั้น และส่งอิลิเมนต์ที่ผ่านการกรองไปให้ template ย่อยในบรรทัดที่ 9-13 ซึ่งถือว่าเป็น "ปลายทาง" นอกจากนี้ หากสังเกตว่า อิลิเมนต์ apply-templates ถูกครอบด้วยแท็กเปิด <canteens> และ แท็กปิด </canteens> ดังนั้นผลลัพธ์จึงจะต้องอยู่ภายในแท็กนี้ ผลลัพธ์จึงต้องมีลักษณะดังนี้

```
<?xml version="1.0" encoding="UTF-8"?>
<canteens>
<!--ผลลัพธ์ที่ได้จากการประมวลผลของ template ย่อยในบรรทัดที่ 9-13-->
</canteens>
```

* หมายเหตุ: XSLT Processor บางตัว เช่น Altova XML 2007 จะสร้าง XML Declaration (ส่วนที่เป็นสีเข้ม) ขึ้นมาโดยอัตโนมัติ ดังนั้น คำตอบอาจจะมีหรือไม่มี XML Declaration ก็ได้

อิลิเมนต์ apply-templates ในบรรทัดที่ 5 จะกรองก่อนหนึ่งชั้น โดยอาศัยคำสั่ง XPath ที่ว่า //room[@type='canteen']"/> หมายถึงอิลิเมนต์ room ทุกตัวในทุกระดับความลึกที่มีแอตทริบิวต์ type มีค่า "canteen" เมื่อดูจากไฟล์ context.xml จะพบว่ามี 2 อิลิเมนต์ คือ อิลิเมนต์แรกคือโค้ดบรรทัดที่ 17-22 และอีกอิลิเมนต์หนึ่งคือบรรทัดที่ 33-37 ทั้งสองอิลิเมนต์นี้จะถูกส่งไปให้ template ย่อยทำงานในบรรทัดที่ 9 ของไฟล์ q01.xsl

บรรทัดที่ 9 โค้ดคือ <canteen name="{@name}"> ผู้อ่านลืมหรือยังครับว่าเป็นการใช้เครื่องหมายปีกกาคร่อม @name (ซึ่ง @name ก็คือคำสั่ง XPath นั่นเอง) มันคือการใช้เทคนิค Attribute Value Templates ในการดึงข้อมูลในโหนดของเอกสาร XML มาใช้สร้างแอตทริบิวต์ (หากลืมแล้วขอให้ไปทบทวนในบทที่ 9 หัวข้อ "ง่ายกว่า ถ้าใช้เทคนิค Attribute Value Templates") ซึ่ง {@name} ก็คือการดึงค่าของแอตทริบิวต์ name ของอิลิเมนต์ room นั่นเอง เมื่อพอรู้เช่นนี้แล้ว เราจะพักบรรทัดนี้ไว้ก่อน แล้วไปดูซิว่าโค้ดบรรทัดต่อไปเป็นอย่างไร

บรรทัดที่ 11 เป็นการพิมพ์ค่าในอิลิเมนต์ info ออกมา การใช้ฟังก์ชัน text() ก็คือการดึงข้อความในอิลิเมนต์ออกมานั่นเอง

ดังนั้นใน template ย่อยของเราจะได้ผลลัพธ์ลักษณะดังนี้

```
<canteen name="ค่าของแอตทริบิวต์ name ของอิลิเมนต์ room">
<!--ข้อความในอิลิเมนต์ info-->
</canteen>
```

ต่อมาเราก็พิจารณาอิลิเมนต์ room ทั้ง 2 ตัวที่ผ่านการกรองออกมา

room ในบรรทัดที่ 13-18 จะได้ผลลัพธ์ออกมาดังนี้

```
<canteen name="eatIT">
Todays menu is hot tomato soup with bread crumbles!
</canteen>
```

room ในบรรทัดที่ 30-34 จะได้ผลลัพธ์ออกมาดังนี้

```
<canteen name="Kantinen">
Today we have a buffet with chicken, salmon and veg. Lasagne
</canteen>
```

เราทราบมาแล้วว่าผลลัพธ์เหล่านี้ จะถูกครอบด้วยแท็กเปิด <canteens> และ แท็กปิด </canteens> ดังนั้นผลลัพธ์ที่ได้จึงเป็นไปดังที่เฉลยไว้

เฉลยข้อ 1.7

แบบที่ 1

```
<canteens>
{ for $r in doc("context.xml")//room[@type='canteen']
return <canteen name="{$r/data(@name)}">{$r/info/text()}</canteen>
}
</canteens>
```

แบบที่ 2

```
<canteens>
{ for $r in doc("context.xml")//room
where $r/@type='canteen'
return <canteen name="{$r/data(@name)}">{$r/info/text()}</canteen>
}
</canteens>
```

* หมายเหตุ: คำตอบมีมากกว่านี้ แต่ไม่สามารถยกตัวอย่างได้หมด ขอให้ผู้อ่านลองทำดูครับ

คำอธิบาย

ย้อนกลับไปในตัวอย่างที่ 18 ของบทที่ 14 ได้แสดงให้เห็นว่าหากต้องการให้ผลลัพธ์ที่ได้จาก XQuery อยู่ในรูป XML ทำได้โดยการเขียนรูตอิลิเมนต์ขึ้นมาก่อน จากนั้นเขียนเครื่องหมายปีกกาเปิด-ปิดขึ้นมา แล้วแทรกคำสั่ง FLWOR เข้าไปข้างในเครื่องหมายปีกกานั้น เมื่อหันมามองผลลัพธ์ของคำถามข้อที่แล้ว มีอิลิเมนต์ canteens เป็นรูตอิลิเมนต์ เราจึงต้องเขียนโค้ดลักษณะดังนี้

```
<canteens>
{
<!--โค้ด FLWOR-->
}
</canteens>
```

จากคำเฉลย ผมให้โค้ดไว้ 2 แบบ (ทั้งที่ความจริงสามารถเขียนโค้ด ได้หลายวิธี) เราจะมาดูแบบที่ 1 กันก่อน

แบบที่ 1

บรรทัดที่ 2 คำสั่ง XPath ที่ว่า doc("context.xml")//room[@type='canteen'] จะได้โหนดของอิลิเมนต์ room ทุกตัวในทุกระดับความลึกที่มีค่าของแอตทริบิวต์ type เป็น "canteen" เมื่อมองไปที่ไฟล์ context.xml จะมีเพียง 2 โหนดเท่านั้น คือ โหนดแรกคือบรรทัดที่ 13-18 และอีกโหนดหนึ่งคือบรรทัดที่ 30-34

ใช้คำสั่ง for ผูกค่าตัวแปร r กับโหนดทั้ง 2 ทีละโหนด แบบวนซ้ำ (iteration) จึงต้องวน 2 รอบ แต่ละรอบจะต้องประมวลผลคำสั่งในบรรทัดที่ 3 คือคำสั่ง return ที่ว่า

<canteen name="{$r/data(@name) }">{ $r/info/text() }</canteen>

โดยที่

{$r/data(@name) ก็คือค่าของแอตทริบิวต์ name ของอิลิเมนต์ room

$r/info/text() ก็คือข้อความในอิลิเมนต์ info ซึ่งเป็นลูกของอิลิเมนต์ room

ดังนั้น เมื่อลองไปดูที่ไฟล์ context.xml แล้วลองมาแทนค่าในแต่ละรอบจะได้ว่า

ในรอบที่ 1: โหนด room ในบรรทัดที่ 13-18 จะได้ว่า

<canteen name="eatIT">Todays menu is hot tomato soup with bread crumbles!</canteen>

ในรอบที่ 2: โหนด room ในบรรทัดที่ 30-34 จะได้ว่า

<canteen name="Kantinen">Today we have a buffet with chicken, salmon and veg. lasagne</canteen>

ซึ่งเมื่อครอบด้วยแท็กเปิด-ปิด <canteens> อีกชั้นหนึ่ง ก็ตรงกับผลลัพธ์ในข้อที่ผ่านมานั่นเอง

แบบที่ 2

จะเห็นว่าเราใช้คำสั่ง where มาช่วย โดยนำขั้นตอนการตรวจสอบว่าจะคัดเลือกเฉพาะอิลิเมนต์ room ที่แอตทริบิวต์ type มีค่า "canteen" มาไว้ในคำสั่ง where แทน ซึ่งทำความเข้าใจได้ไม่ยาก

ผู้อ่านคงเห็นพ้องกับผมนะครับว่า ในการสืบค้นหนึ่งๆ สามารถเขียน XQuery ได้มากกว่า 1 วิธี

เฉลยข้อ 2.

เฉลยข้อ 2.1

จากบรรทัดที่ 8 ใน DTD (ไฟล์ inventory.dtd) แปลได้ว่า อิลิเมนต์ item จะต้องมีอิลิเมนต์ลูกเรียงตามลำดับ เริ่มจาก อิลิเมนต์ name, อิลิเมนต์ description (มีหรือไม่ก็ได้ ถ้ามี ก็จะมีได้สูงสุดเพียง 1 อิลิเมนต์), อิลิเมนต์ price, อิลิเมนต์ sale (มีหรือไม่ก็ได้ ถ้ามี ก็จะมีได้สูงสุดเพียง 1 อิลิเมนต์) และปิดท้ายด้วยอิลิเมนต์ stock

ดังนั้น

a. (name, price, sale, stock) ถูกต้อง เพราะไม่ต้องมีอิลิเมนต์ description ก็ได้

b. (name, description, sale, stock) ไม่ถูกต้อง เพราะขาดอิลิเมนต์ price

c. (name, description, price, stock) ถูกต้อง เพราะไม่มีอิลิเมนต์ sale ก็ได้

เฉลยข้อ 2.2

เราแยกมาประกาศเป็นเอ็นทิตี้ไว้ก่อน โดยสมมติชื่อเป็น currencies จากนั้นเมื่อต้องการนำไปใช้ที่ใด ก็อ้างถึงชื่อ currencies ได้เลย ดังที่แสดงเป็นตัวเข้ม

```
<?xml version="1.0" ?>
<!ENTITY currencies 'currency (DKK|EUR) "DKK"'>
<!ELEMENT inventory (categories,items)>
<!ELEMENT categories (category+)>
<!ELEMENT category (category*)>
<!ATTLIST category name ID #REQUIRED>
<!ELEMENT items (item)*>
<!ELEMENT item (name,description?,price,sale?,stock)>
<!ATTLIST item category IDREF #REQUIRED
id ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

```
<!ATTLIST price currencies;>
<!ELEMENT sale (#PCDATA)>
<!ATTLIST sale currencies;
from CDATA #REQUIRED
until CDATA #REQUIRED>
<!ELEMENT stock (#PCDATA)>
```

เฉลยข้อ 2.3

ไม่ได้ เพราะจากไฟล์ inventory.dtd บรรทัดที่ 10 กำหนดไว้ว่าค่าของแอตทริบิวต์ id เป็นข้อมูลชนิด ID ซึ่งก็คือต้องมีค่าไม่ซ้ำกันเลย แต่เมื่อมองไปดูที่ไฟล์ inventory.xml พบว่ามีการใช้ค่า “alusink” ไปแล้วในบรรทัดที่ 11

เฉลยข้อ 2.4

ได้ เพราะเมื่อมองดูที่แอตทริบิวต์ id ของทุกๆอิลิเมนต์ item (ในไฟล์ inventory.xml) ยังไม่พบจุดใดที่มีค่า “bathalusink” เลย

เฉลยข้อ 2.5

//item[sale]/@id

เฉลยข้อ 2.6

ไฟล์ inventory.xq

```
1. <items>
2. {
3.      for $i in doc( “inventory.xml” )/inventory/items/item
4.      where $i/price/@currency = ‘DKK’
5.      order by xs:decimal( $i/price )
6.       return
7.              <item>
8.              <name>{$i/name/text()}</name>
9.              <price>{$i/price/text()}</price>
10.             </item>
11. }
12. </items>
```


* หมายเหตุ: คำตอบมีมากกว่านี้ แต่ไม่สามารถยกตัวอย่างได้หมด ขอให้ผู้อ่านลองทำดูครับ

คำอธิบาย

โจทย์ข้อนี้ดูผิวเผินเหมือนไม่มีอะไรยากครับ แต่ลองอ่านโจทย์ดีๆ จะพบว่าผู้ตั้งโจทย์ต้องการตรวจสอบความเข้าใจของเราในสองประเด็น คือ

1. การเรียงลำดับตัวเลข
2. ความเข้าใจในเอกสาร DTD

ในประเด็นที่ 1. หากเราเผลอเรอ หรือไม่รู้ว่าการจะเรียงลำดับตัวเลขด้วยคำสั่ง order by นั้น ต้องแปลงค่าจากสตริงให้เป็นตัวเลขก่อน ทั้งนี้เพราะ XML Processor ไม่รู้หรอกครับว่าข้อมูลในอิลิเมนต์ price เป็นตัวเลข รู้แต่ว่าเป็นสตริง “400.00”, “125.00” และ “1500.00” เราจึงต้องใช้กลไกการ casting โดยอาศัยคำสั่ง xs:decimal() มาช่วยในการแปลงให้เป็นตัวเลขเพื่อการเรียงลำดับได้ถูกต้อง ดังแสดงในเฉลยบรรทัดที่ 5 (ของไฟล์ inventory.xq) หากไม่ใช้คำสั่งนี้ช่วย XSLT Processor จะเรียงลำดับตามตัวอักษรแบบสตริง คือ “125.00”, “1500.00” และ “400.00” ซึ่งไม่ใช่การเรียงลำดับค่าน้อยไปมาก

(หากผู้อ่านลืมเรื่องนี้ไปแล้ว ขอให้ย้อนกลับไปดูรายละเอียดในบทที่ 14 ตัวอย่างที่ 21)

ในประเด็นที่ 2. เราต้องเข้าใจที่โจทย์บอกว่า “โดยให้ XQuery ที่เขียนสามารถสืบค้นเอกสาร XML ใดๆ ก็ตามที่เป็นไปตามข้อกำหนดในเอกสาร DTD ข้างต้นได้อย่างถูกต้อง” ดังนั้น ถึงเราจะเขียน XQuery ที่สืบค้นไฟล์ inventory.xml แล้วได้ผลลัพธ์ดังที่โจทย์ต้องการ ก็ไม่ได้หมายความว่าโค้ดของ XQuery ของเราจะถูกต้องเสมอนะครับ แต่ต้องเขียน XQuery ให้สืบค้นเอกสาร XML ทุกๆเอกสารที่เป็นตามข้อกำหนดของ DTD นี้ ดังนั้น เราจึงต้องเข้าใจข้อกำหนดใน DTD ด้วย

ผู้อ่านลองไปดูที่ไฟล์ inventory.dtd ในบรรทัดที่ 14. ที่ว่า...

<!ATTLIST price currency (DKK|EUR) “DKK”>

ข้อกำหนดนี้บอกว่าแอตทริบิวต์ currency ของอิลิเมนต์ price จะต้องมีค่าเป็น “DKK” หรือ “EUR” เท่านั้น โดยที่ค่าดีฟอลต์จะเป็น “DKK” (หมายความว่าหากไม่กำหนดไว้ ให้ถือว่าเป็น “DKK” นั่นเอง)

แต่เผอิญในไฟล์ inventory.xml นั้น สินค้าทุกตัวมีหน่วยเป็น DKK อยู่แล้ว ตรงนี้แหละครับที่สำคัญ หากเราไม่รอบคอบ ไม่วิเคราะห์ตรงนี้ให้ดี ก็อาจจะไม่ได้เขียนโค้ดบรรทัดที่ 4 ในไฟล์ inventory.xq เพื่อตรวจสอบว่า “เฉพาะสินค้าที่มีราคาเป็นสกุลเงิน DKK เท่านั้น” เพราะเรามัวแต่ไปคิดว่าสินค้าทุกตัวในไฟล์ inventory.xml ล้วนมีราคาเป็นสกุลเงิน DKK อยู่แล้ว

ยังมีสิ่งที่อยากจะเสริมอีก 2 ประเด็น คือ

1. คำสั่ง XPath ในบรรทัดที่ 3 ที่ว่า...doc(“inventory.xml”)/inventory/items/itemสามารถใช้คำสั่ง doc(“inventory.xml”)//item แทนได้ โดยที่ให้ผลลัพธ์เหมือนกัน

2. คำสั่ง $i/name/text() ในบรรทัดที่ 8 และ $i/price/text() ในบรรทัดที่ 9 นั้น สามารถใช้คำสั่ง string($i/name) และ string($i/price) แทนได้ โดยที่ผลลัพธ์ไม่เปลี่ยนแปลงทั้งนี้เพราะทั้งฟังก์ชัน text() และ string() ล้วนมีความสามารถในการดึงเฉพาะข้อความออกมา (ไม่รวมแท็ก) ด้วยกันทั้งคู่ (ดังที่ได้เคยกล่าวมาในบทที่ 14 ตัวอย่างที่ 8)

เฉลยข้อ 3.

ตัวอย่างหนึ่งของไฟล์ Q3.xml และผลลัพธ์ที่ได้จากการรัน XQuery เป็นดังนี้

Q3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<addressBook>
<address>
<name>
<lastName>last</lastName>
<firstName>first</firstName>
<middleName initial="no">middle</middleName>
</name>
<tel>1133</tel>
</address>
</addressBook>
```

ผลลัพธ์จาก XQuery

```
<address>
<name>
<lastName>last</lastName>
<firstName>first</firstName>
```

```
<middleName  initial="no">middle</middleName>
</name>
<tel>1133</tel>
</address>
```

คำอธิบาย

จากโค้ดของ XQuery เมื่อพิจารณาในคำสั่ง return จะเห็นได้ว่าเป็นคำสั่ง XPath ที่ยาวพอสมควร เราต้องดูให้ออกว่าเป็นการเข้าถึงส่วนใดของเอกสาร XML บอกตามตรงครับว่าคำสั่ง XPath รูปแบบนี้มักไม่ค่อยได้พบมากนัก แสดงว่าข้อสอบข้อนี้ค่อนข้างโหดที่หยิบเอาคำสั่งที่ไม่ค่อยได้พบเห็นบ่อยนักมาออก

ย้อนกลับไปในตาราง 14-1 ของบทที่ 14 ผมได้แสดงคำสั่ง XPath อันหนึ่ง คือ

/Transcript[Master][CrsTaken] แปลว่า เข้าถึงอิลิเมนต์ Transcript ใดๆก็ตามที่มีอิลิเมนต์ลูกชื่อ Master อย่างน้อย 1 อิลิเมนต์ และ CrsTaken อย่างน้อย 1 อิลิเมนต์

ลองเปรียบเทียบดู จะเห็นว่าคล้ายๆ กับคำสั่งในโจทย์ คือมีเครื่องหมาย [...] 2 ตัว

ตัวแรกคือ [.//name[middleName[@initial="no"]]] แปลได้ว่า อิลิเมนต์ name ในทุกระดับความลึกที่มีอิลิเมนต์ middleName เป็นอิลิเมนต์ลูก และอิลิเมนต์ middleName นั้นมีแอตทริบิวต์ initial ที่มีค่า "no"

ตัวที่สอง คือ [tel] แปลว่า อิลิเมนต์ tel

ดังนั้น $x//address[.//name[middleName[@initial="no"]]][tel] จึงแปลว่า อิลิเมนต์ address ทุกตัวในทุกระดับความลึกที่มีอิลิเมนต์ชื่อ name อย่างน้อย 1 อิลิเมนต์ และ tel อย่างน้อย 1 อิลิเมนต์ โดยที่อิลิเมนต์ name อยู่ลึกลงไปกี่ระดับก็ได้แต่ต้องมีอิลิเมนต์ middleName เป็นอิลิเมนต์ลูก

และอิลิเมนต์ middleName นั้นมีแอตทริบิวต์ initial ที่มีค่า "no"

คำสั่ง return จะไม่ส่งค่าว่างออกมา ก็ต่อเมื่อไฟล์ Q3.xml จะต้องมีอิลิเมนต์ที่เป็นไปตาม XPath ข้างต้น อีกทั้งยังต้องทำตามข้อกำหนดใน DTD ดังนั้น ไฟล์ Q3.xml จะต้องเป็นไปตามข้อกำหนดดังนี้

เอกสาร XML จะต้องมีรูตอิลิเมนต์ ในที่นี้คือ อิลิเมนต์ addressBook (ตามที่บอกไว้ใน DTD)

ต้องมีอิลิเมนต์ address อยู่ภายในอย่างน้อย 1 อิลิเมนต์ (ตามความหมายของ XPath) และต้องเป็นอิลิเมนต์ลูกของรูตอิลิเมนต์ (บอกในบรรทัดที่ 2ของ DTD)

ห้ามมีอิลิเมนต์ note (เพราะบรรทัดที่ 2 ใน DTD ที่ว่า address | note ความหมายคือถ้ามีอิลิเมนต์ address ก็ห้ามมี note)

ภายในอิลิเมนต์ address ต้องมีอิลิเมนต์ name และ tel เป็นอิลิเมนต์ลูกอย่างละ 1 อิลิเมนต์เป็นอย่างน้อย (ตามความหมายของ XPath)

ภายในอิลิเมนต์ name ต้องมีอิลิเมนต์ lastName, firstName, middleName เป็นอิลิเมนต์ลูก ขาดตัวใดตัวหนึ่งไม่ได้ (จากบรรทัดที่ 4 ของ DTD) และ และอิลิเมนต์ middle นั้นมีแอตทริบิวต์ initial ที่มีค่าเป็น "no" เสมอ (ตามความหมายของ XPath หรือจะดูจากบรรทัดที่ 12 ของ DTD ที่บ่งบอกโดยคีย์เวิร์ด #FIXED)

อิลิเมนต์ address ห้ามมีอิลิเมนต์ลูกชื่อ email เพราะมีอิลิเมนต์ tel เป็นอิลิเมนต์ลูกแล้ว (กำหนดไว้ด้วย email | tel ในบรรทัดที่ 3 ของ DTD)

เราสามารถใช้กฎเกณฑ์เหล่านี้ นำมาสร้างไฟล์ Q3.xml ได้มากมายหลายแบบ หากไฟล์ Q3.xml ที่ผู้อ่านคิดมาได้ ตรงตามข้อกำหนดที่ว่ามานี้ ก็ถือว่าเป็นคำตอบที่ถูก ซึ่งไฟล์ Q3.xml ที่ผมยกตัวอย่างมานี้ก็ทำตามกฎเกณฑ์นี้เช่นกัน

ส่วนผลลัพธ์จาก XQuery ก็จะแสดงออกมาเฉพาะอิลิเมนต์ address เท่านั้น (เพราะคำสั่ง return มีคำสั่ง XPath ที่เจาะจงเฉพาะอิลิเมนต์ address) จึงเป็นไปดังเฉลยข้างต้น