

CPE217 – Homework 3

Homework: Linked list data structures

Homework Due Date: Sun 8 September 2019

Patiwet Wuttisarnwattana, Ph.D.

Department of Computer Engineering

- คำชี้แจงการส่งงาน
- แต่ละกลุ่ม ควรให้ core person เป็นคนส่งงาน และในช่องข้อความต้องระบุรหัสประจำตัวนักศึกษาของทุกคนที่เป็นสมาชิกในกลุ่ม หาก Core Person ไม่สามารถส่งงานได้ ให้สมาชิกคนอื่นก็ได้ส่งงานแทน แต่ต้องบอกว่าส่งแทน Core Person ซึ่งก็คือใคร มีรหัสอะไร
- โค้ดของคุณต้องมีคอมเมนต์ (comment) เพื่ออธิบายว่าโค้ดดังที่เห็นอยู่นี้ทำงานอะไร หรือ if นี่ทำตรวจสอบอะไร หากกลุ่มไหนไม่มีคอมเมนต์ในโค้ดจะไม่ได้รับการตรวจ การเขียนคอมเมนต์ไม่ต้องเขียนแบบละเอียดยิบก็ได้เท่าที่คุณต้องการให้ผู้ตรวจทราบก็พอ
- งานที่ส่งต้องประกอบด้วย ZIP file ของ src folder ที่สามารถกด F6 รันได้เลย
- สามารถส่งการบ้านช้าได้ แต่หักคะแนนวันละ 10%
- การลอกงานเพื่อนมาส่ง เป็นการทุจริตและมีความผิดทางวินัย อาจารย์อาจพิจารณาให้คะแนนการบ้านนั้น หรือคะแนนการบ้านทั้งหมดของคุณ และ/หรือ คะแนนจิตพิสัยทั้งหมดได้ศูนย์คะแนน ซึ่งนั่นอาจเป็นปัจจัยของการตัดสินใจถอนกระบวนวิชาของคุณและลงทะเบียนใหม่ในปีการศึกษาหน้า
- การลอกงานเพื่อนมาส่งต้องรับผิดชอบพร้อมกันทั้งกลุ่ม จะให้คนทำผิดรับผิดชอบเพียงคนเดียวไม่ได้

การบ้านนี้ให้นักศึกษา implement “Linked List ADT” โดยใช้ Java โดยให้มีคุณสมบัติดังต่อไปนี้

1. ให้สร้าง class ชื่อว่า DoublyLinkedList โดย class นี้ มีคุณสมบัติของ “Doubly-linked list with tail” ตามที่ได้เรียนในห้องเรียน
2. ให้ object ของ DoublyLinkedList ทำหน้าที่บรรจุ objects ของ class Node โดย class Node นี้ มีสมาชิกที่เป็นตัวแปรอย่างน้อยสามตัว คือ student_id (ชนิด int), name (ชนิด String) และ gpa (ชนิด double)
 - a. ในการ implement Doubly-linked list with tail คุณสามารถใช้ references head และ tail ได้อย่างเต็มที่ แต่ถ้าจะทำการ implement Singly-linked list without tail (ตามข้อ 6) คุณสามารถใช้ reference head ได้แต่เพียงอย่างเดียว
3. ให้ class Node มี 2 Constructors (Overloading functions) ดังนี้
 - a. Node(int id, String name, double gpa)
 - ทำหน้าที่ กำหนดค่าเริ่มต้นของทั้งสามตัวแปรตามที่ผู้ใช้งานกำหนด
 - b. Node(String error_msg)
 - Constructor ชนิดนี้จะถูกใช้เพื่อสร้าง dummy Node เพื่อ return ให้กับ caller ในกรณีเกิด ERROR ขึ้น
 - โดย Error Message ที่เกิดขึ้นให้เซตไปที่ค่าของตัวแปร name ส่วนค่าเริ่มต้นสำหรับตัวแปรที่เหลือ ให้ใช้ค่า default ของ Java
 - c. ใน starter code ผมได้เติม constructor Node() ไว้อีกหนึ่ง เพื่อไม่ให้ dummy code ของผมมันฟ้อง error คุณไม่ต้องสนใจฟังก์ชันอันนี้นะครับ ให้สนใจเฉพาะสองฟังก์ชันข้างบนก็พอ
4. ให้ class Node มีฟังก์ชัน public void printIDName()
 - ฟังก์ชันนี้ทำหน้าที่ พิมพ์ค่าของ student_id และ name ของออกทาง console
 - ใน starter code มีให้ละครับ
5. ให้ class DoublyLinkedList มี public functions ดังต่อไปนี้
 - a. public LinkedList1(String name)
 - ฟังก์ชันนี้ทำหน้าที่เป็น Constructor ของ List โดยกำหนดชื่อของ List คือค่าของตัวแปร name
 - b. public void pushFront(Node node)
 - ฟังก์ชันนี้ทำหน้าที่นำ node มาเติมข้างหน้าสุดของ list (referenced by head)
 - c. public Node topFront()
 - ฟังก์ชันนี้ทำหน้าที่ return node ที่อยู่ข้างหน้าสุดของ list (referenced by head)
 - ถ้า List ว่าง ให้ print ออกทาง console ก่อนเลยว่า ERROR แล้ว ให้ return ด้วย Node ที่ถูกสร้างด้วย Constructor แบบที่สอง โดยให้มี Error Message ว่า “Empty List!”
 - d. public void popFront()
 - ฟังก์ชันนี้ทำหน้าที่ ลบ node ที่อยู่ข้างหน้าสุดของ list (referenced by head) ออกไป
 - ถ้า List ว่าง ให้ print ออกทาง console ว่า ERROR

- e. `public void pushback(Node node)`
- ฟังก์ชันนี้ทำหน้าที่นำ node ไปต่อท้าย node ที่อยู่หลังสุด
- f. `public Node topBack()`
- ฟังก์ชันนี้ทำหน้าที่ return node ที่อยู่ข้างหลังสุดของ list
 - ถ้า List ว่าง ให้ print ออกทาง console ก่อนเลยว่า ERROR แล้ว ให้ return ด้วย Node ที่ถูกสร้างด้วย Constructor แบบที่สอง โดยให้มี Error Message ว่า "Empty List!"
- g. `public void popBack()`
- ฟังก์ชันนี้ทำหน้าที่ ลบ node ที่อยู่ข้างหลังสุดของ list ออกไป
 - ถ้า List ว่าง ให้ print ออกทาง console ว่า ERROR
- h. `public Node findNode(int id)`
- ฟังก์ชันนี้ทำหน้าที่ return Node ที่มีค่า student_id เท่ากับ id
 - ถ้า List ว่าง ให้ return ด้วย Node ที่ถูกสร้างด้วย Constructor แบบที่สอง โดยให้มี Error Message ว่า "Empty List!" (ไม่ต้องแจ้งว่า ERROR)
 - ถ้า List ไม่ว่างแต่หา node นั้นไม่เจอให้ return ด้วย Node ที่ถูกสร้างด้วย Constructor แบบที่สอง โดยให้มี Error Message ว่า "Student Not Found!"
- i. `public Node eraseNode(int id)`
- ฟังก์ชันนี้ทำหน้าที่หา Node ที่มีค่า student_id เท่ากับ id ใน List เมื่อเจอแล้วให้ลบ Node นั้นออกจาก List นอกจากนี้ ให้ return Node ที่ลบนั้นออกไปให้ caller อีกด้วย
 - ถ้า List ว่าง ให้ print ออกทาง console ก่อนเลยว่า ERROR แล้ว ให้ return ด้วย Node ที่ถูกสร้างด้วย Constructor แบบที่สอง โดยให้มี Error Message ว่า "Empty List!"
 - ถ้าหา node นั้นไม่เจอให้ return ด้วย Node ที่ถูกสร้างด้วย Constructor แบบที่สอง โดยให้มี Error Message ว่า "Student Not Found!"
- j. `public boolean isEmpty()`
- ฟังก์ชันนี้ทำหน้าที่ return ว่า Data structure นี้ว่างหรือไม่
- k. `public void addNodeAfter(Node node1, Node node2)`
- ฟังก์ชันนี้ทำหน้าที่นำ node2 (ซึ่งเป็น Node ใดๆ) ไปแทรกใน list โดยนำไปแทรกด้านหลังของ node1 (ซึ่งมีอยู่ใน list อยู่แล้ว)
- l. `public void addNodeBefore(Node node1, Node node2)`
- ฟังก์ชันนี้ทำหน้าที่นำ node2 (ซึ่งเป็น Node ใดๆ) ไปแทรกใน list โดยนำไปแทรกด้านหน้าของ node1 (ซึ่งมีอยู่ใน list อยู่แล้ว)
- m. `public Node whoGotHighestGPA()`
- ฟังก์ชันนี้ทำหน้าที่หาว่า Node ไหนมีนักเรียนที่ได้ GPA สูงที่สุด

- ถ้า List ว่าง ให้ return ด้วย Node ที่ถูกสร้างด้วย Constructor แบบที่สอง โดยให้มี Error Message ว่า “Empty List!” (ไม่ต้องแจ้งว่า ERROR)
- ถ้าคนที่ได้ GPA สูงที่สุดมีมากกว่าหนึ่งคน ให้ return คนที่อยู่ใกล้ head (ใกล้ tail) มากที่สุด

n. `public void merge(List list)`

- ฟังก์ชันนี้ทำหน้าที่รวม List สองตัวเข้าด้วยกัน โดยนำ List (list) ที่นำเข้า เข้าต่อด้านหลังสุดของ List (this) ปัจจุบัน

o. `public void printStructure()`

- ให้ print สถานะข้อมูลของ Data structure ออกทาง console ด้วย pattern ดังต่อไปนี้
- หาก SinglyLinkedList ชื่อ list1 มีข้อมูลคือ [[59061, “Mathew”, 3.0], [59062, “Mark”, 2.0], [59063, “Luke”, 2.5], [59064, “John”, 3.25]] ให้แสดงว่า
 - list1: head -> [59061] -> [59062] -> [59063] -> [59064] -> null
- หาก DoublyLinkedList ชื่อ list2 มีข้อมูลคือ [[59064, “John”, 3.25], [59062, “Mark”, 2.0]] ให้แสดงว่า
 - list2: head <-> [59064] <-> [59062] <-> tail

6. ให้สร้าง class ชื่อว่า SinglyLinkedList โดย class นี้ มีคุณสมบัติของ “Singly-linked list without tail” (ไม่มีหาง) ตามที่ได้เรียนในห้องเรียน โดยกำหนดให้คลาสนี้สามารถดำเนินการได้ตามข้อ 5a. – 5o. เหมือนกับที่คุณทำมาแล้วได้ทั้งหมด

7. ตัวอย่างการทำงาน

Java code

```
public static void main(String[] args) {
    Node node;
    DoublyLinkedList list1 = new DoublyLinkedList("list1");
    node = new Node(5906001, "Matthew", 3.50);
    list1.pushFront(node);
    node = new Node(5906002, "Mark", 2.75);
    list1.pushFront(node);
    node = new Node(5906003, "Luke", 3.00);
    list1.pushFront(node);
    node = new Node(5906004, "John", 3.75);
    list1.pushFront(node);
    list1.printStructure();
    DoublyLinkedList list2 = new DoublyLinkedList("list2");
```

<pre> list2.pushBack(new Node(5906005, "James", 3.25)); list2.pushBack(new Node(5906006, "Peter", 2.85)); list2.pushBack(new Node(5906007, "John", 2.50)); list2.pushBack(new Node(5906008, "Jude", 3.15)); list2.printStructure(); list1.merge(list2); list1.printStructure(); } </pre>
Output
<pre> list1: head <-- [5906004] <-- [5906003] <-- [5906002] <-- [5906001] <-- tail list2: head <-- [5906005] <-- [5906006] <-- [5906007] <-- [5906008] <-- tail list1: head <-- [5906004] <-- [5906003] <-- [5906002] <-- [5906001] <-- [5906005] <-- [5906006] <-- [5906007] <-- [5906008] <-- tail </pre>
ผลลัพธ์เมื่อเปลี่ยนชื่อ class จาก DoublyLinkedList เป็น SinglyLinkedList
<pre> list1: head -> [5906004] -> [5906003] -> [5906002] -> [5906001] -> null list2: head -> [5906005] -> [5906006] -> [5906007] -> [5906008] -> null list1: head -> [5906004] -> [5906003] -> [5906002] -> [5906001] -> [5906005] -> [5906006] -> [5906007] -> [5906008] -> null </pre>

Java code
<pre> public static void main(String[] args) { Node node; DoublyLinkedList list = new DoublyLinkedList("list3"); node = list.topFront(); node.printIDName(); node = list.topBack(); node.printIDName(); list.pushBack(new Node(5906001, "Matthew", 3.50)); list.pushBack(new Node(5906002, "Mark", 2.75)); list.pushBack(new Node(5906003, "Luke", 3.00)); node = list.topFront(); node.printIDName(); node = list.topBack(); node.printIDName(); } </pre>

Output
ERROR StudentID: 0 , Name: Empty List! ERROR StudentID: 0 , Name: Empty List! StudentID: 5906001 , Name: Matthew StudentID: 5906003 , Name: Luke
ผลลัพธ์เมื่อเปลี่ยนชื่อ class จาก DoublyLinkedList เป็น SinglyLinkedList → เหมือนเดิม

Java code
<pre> public static void main(String[] args) { DoublyLinkedList list = new DoublyLinkedList("list4"); list.printStructure(); list.popFront(); list.pushBack(new Node(5906001, "Matthew", 3.50)); list.pushBack(new Node(5906002, "Mark", 2.75)); list.pushBack(new Node(5906003, "Luke", 3.00)); list.pushBack(new Node(5906004, "John", 3.75)); list.printStructure(); list.popFront(); list.printStructure(); list.popBack(); list.printStructure(); list.popFront(); list.printStructure(); list.popBack(); list.printStructure(); list.popBack(); } </pre>
Output
list4: head <--> tail ERROR

```
list4: head <-> {5906001} <-> {5906002} <-> {5906003} <-> {5906004} <-> tail
list4: head <-> {5906002} <-> {5906003} <-> {5906004} <-> tail
list4: head <-> {5906002} <-> {5906003} <-> tail
list4: head <-> {5906003} <-> tail
list4: head <-> tail
ERROR
```

ผลลัพธ์เมื่อเปลี่ยนชื่อ class จาก DoublyLinkedList เป็น SinglyLinkedList

```
list4: head -> null
ERROR
list4: head -> {5906001} -> {5906002} -> {5906003} -> {5906004} -> null
list4: head -> {5906002} -> {5906003} -> {5906004} -> null
list4: head -> {5906002} -> {5906003} -> null
list4: head -> {5906003} -> null
list4: head -> null
ERROR
```

Java code

```
public static void main(String[] args) {
    Node node;
    DoublyLinkedList list = new DoublyLinkedList("list5");
    node = list.whoGotHighestGPA();
    node.printIDName();
    list.pushBack(new Node(5906001, "Matthew", 3.25));
    list.pushBack(new Node(5906002, "Mark", 2.75));
    list.pushBack(new Node(5906003, "Luke", 3.00));
    list.printStructure();
    node = list.whoGotHighestGPA();
    node.printIDName();
    Node newNode = new Node(5906004, "John", 3.30);
    list.addNodeBefore(node, newNode);
    list.printStructure();
    list.addNodeAfter(list.whoGotHighestGPA(), new Node(5906005, "James", 3.40));
    list.printStructure();
    (list.whoGotHighestGPA()).printIDName();
}
```

}
Output
StudentID: 0 , Name: Empty List! list5: head <-> [5906001] <-> [5906002] <-> [5906003] <-> tail StudentID: 5906001 , Name: Matthew list5: head <-> [5906004] <-> [5906001] <-> [5906002] <-> [5906003] <-> tail list5: head <-> [5906004] <-> [5906005] <-> [5906001] <-> [5906002] <-> [5906003] <-> tail StudentID: 5906005 , Name: James
ผลลัพธ์เมื่อเปลี่ยนชื่อ class จาก DoublyLinkedList เป็น SinglyLinkedList
StudentID: 0 , Name: Empty List! list5: head -> [5906001] -> [5906002] -> [5906003] -> null StudentID: 5906001 , Name: Matthew list5: head -> [5906004] -> [5906001] -> [5906002] -> [5906003] -> null list5: head -> [5906004] -> [5906005] -> [5906001] -> [5906002] -> [5906003] -> null StudentID: 5906005 , Name: James

Java code
<pre> public static void main(String[] args) { Node node; DoublyLinkedList list = new DoublyLinkedList("list6"); list.pushBack(new Node(5906001, "Matthew", 3.00)); list.pushBack(new Node(5906002, "Mark", 2.75)); list.pushBack(new Node(5906003, "Luke", 3.25)); list.printStructure(); node = list.whoGotHighestGPA(); node.printIDName(); Node newNode = new Node(5906004, "John", 3.30); list.addNodeAfter(node, newNode); list.printStructure(); list.addNodeBefore(list.whoGotHighestGPA(), new Node(5906005, "James", 3.30)); list.printStructure(); (list.whoGotHighestGPA()).printIDName(); } </pre>

Output
list6: head <-> {5906001} <-> {5906002} <-> {5906003} <-> tail StudentID: 5906003 , Name: Luke list6: head <-> {5906001} <-> {5906002} <-> {5906003} <-> {5906004} <-> tail list6: head <-> {5906001} <-> {5906002} <-> {5906003} <-> {5906005} <-> {5906004} <-> tail StudentID: 5906004 , Name: John
ผลลัพธ์เมื่อเปลี่ยนชื่อ class จาก DoublyLinkedList เป็น SinglyLinkedList
list6: head -> {5906001} -> {5906002} -> {5906003} -> null StudentID: 5906003 , Name: Luke list6: head -> {5906001} -> {5906002} -> {5906003} -> {5906004} -> null list6: head -> {5906001} -> {5906002} -> {5906003} -> {5906005} -> {5906004} -> null StudentID: 5906004 , Name: John

Java code
<pre> public static void main(String[] args) { DoublyLinkedList list = new DoublyLinkedList("list7"); (list.findNode(5906001)).printIDName(); (list.eraseNode(5906001)).printIDName(); list.pushBack(new Node(5906001, "Matthew", 3.50)); list.pushBack(new Node(5906002, "Mark", 2.75)); list.pushBack(new Node(5906003, "Luke", 3.00)); list.pushBack(new Node(5906004, "John", 3.75)); list.pushBack(new Node(5906005, "James", 3.25)); list.pushBack(new Node(5906006, "Peter", 2.85)); list.printStructure(); (list.findNode(5906001)).printIDName(); (list.findNode(5906006)).printIDName(); (list.findNode(5906007)).printIDName(); Node node = list.findNode(5906003); list.addNodeAfter(node, new Node(5906007, "John", 2.50)); list.printStructure(); (list.eraseNode(5906001)).printIDName(); list.printStructure(); (list.eraseNode(5906006)).printIDName(); list.printStructure(); (list.eraseNode(5906003)).printIDName(); list.printStructure(); } </pre>

<pre> (list.eraseNode(5906003)).printIDName(); list.printStructure(); } </pre>
<p>Output</p> <p>StudentID: 0 , Name: Empty List!</p> <p>ERROR</p> <p>StudentID: 0 , Name: Empty List!</p> <p>list7: head <-> [5906001] <-> [5906002] <-> [5906003] <-> [5906004] <-> [5906005] <-> [5906006] <-> tail</p> <p>StudentID: 5906001 , Name: Matthew</p> <p>StudentID: 5906006 , Name: Peter</p> <p>StudentID: 0 , Name: Student Not Found!</p> <p>list7: head <-> [5906001] <-> [5906002] <-> [5906003] <-> [5906007] <-> [5906004] <-> [5906005] <-> [5906006] <-> tail</p> <p>StudentID: 5906001 , Name: Matthew</p> <p>list7: head <-> [5906002] <-> [5906003] <-> [5906007] <-> [5906004] <-> [5906005] <-> [5906006] <-> tail</p> <p>StudentID: 5906006 , Name: Peter</p> <p>list7: head <-> [5906002] <-> [5906003] <-> [5906007] <-> [5906004] <-> [5906005] <-> tail</p> <p>StudentID: 5906003 , Name: Luke</p> <p>list7: head <-> [5906002] <-> [5906007] <-> [5906004] <-> [5906005] <-> tail</p> <p>StudentID: 0 , Name: Student Not Found!</p> <p>list7: head <-> [5906002] <-> [5906007] <-> [5906004] <-> [5906005] <-> tail</p>
<p>ผลลัพธ์เมื่อเปลี่ยนชื่อ class จาก DoublyLinkedList เป็น SinglyLinkedList</p> <p>StudentID: 0 , Name: Empty List!</p> <p>ERROR</p> <p>StudentID: 0 , Name: Empty List!</p> <p>list7: head -> [5906001] -> [5906002] -> [5906003] -> [5906004] -> [5906005] -> [5906006] -> null</p> <p>StudentID: 5906001 , Name: Matthew</p> <p>StudentID: 5906006 , Name: Peter</p> <p>StudentID: 0 , Name: Student Not Found!</p> <p>list7: head -> [5906001] -> [5906002] -> [5906003] -> [5906007] -> [5906004] -> [5906005] -> [5906006] -> null</p> <p>StudentID: 5906001 , Name: Matthew</p> <p>list7: head -> [5906002] -> [5906003] -> [5906007] -> [5906004] -> [5906005] -> [5906006] -> null</p> <p>StudentID: 5906006 , Name: Peter</p> <p>list7: head -> [5906002] -> [5906003] -> [5906007] -> [5906004] -> [5906005] -> null</p> <p>StudentID: 5906003 , Name: Luke</p> <p>list7: head -> [5906002] -> [5906007] -> [5906004] -> [5906005] -> null</p> <p>StudentID: 0 , Name: Student Not Found!</p> <p>list7: head -> [5906002] -> [5906007] -> [5906004] -> [5906005] -> null</p>

8. โปรดใช้ Starter code ที่อาจารย์แนบให้