**Doubly Linked List Worksheet**

Use the DoublyLinkedNode class.

**1.** Consider this portion of a doubly linked list of characters:

--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->

│ │ B │ │ │ │ F │ │ │ │ K │ │ │ │ Q │ │ │ │ W │ │

<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---

p1 p2

How is the linked list pictured after the code segment below is executed?

DoublyLinkedNode p1;

DoublyLinkedNode p2;

**while** (p1 != p2)

{

p1 = p1.getNext();

p2 = p2.getPrev();

}

p1.getNext().setPrev(p2.getPrev());

p2.getPrev().setNext(p1.getNext());

[A] --->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->

│ │ B │ │ │ │ F │ │ │ │ Q │ │ │ │ W │ │

<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---

[B] --->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->

│ │ B │ │ │ │ F │ │ │ │ Q │ │ │ │ W │ │

<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---

p1 p2

[C] --->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->

│ │ B │ │ │ │ Q │ │ │ │ F │ │ │ │ W │ │

<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---

[D] --->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->

│ │ B │ │ │ │ Q │ │ │ │ K │ │ │ │ F │ │ │ │ W │ │

<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---

[E] --->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->┌─┬───┬─┐--->

│ │ B │ │ │ │ Q │ │ │ │ K │ │ │ │ F │ │ │ │ W │ │

<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---└─┴───┴─┘<---

p1 p2

**2.** Examine this class that maintains a double linked list:

**public class** DLList

{

**private** DoublyLinkedNode front, rear; //front points to first node //if the list is not empty; otherwise null

//rear points to the last node in the list   
 //if the list is not empty; otherwise null

**private void** makeCircular()

{ /\* to be implemented \*/ }

//other methods and constructors not shown

}

}

Furthermore, assume that a non-empty doubly linked list has been created that is represented by the following general representation:

rear

front

1300

1000

1500

1400

1200

1100

Complete method makeCircularbelow that turns the doubly linked list into a circular doubly linked list.

//precondition : The list is not empty

// front points to the front of the doubly linked list

// rear points to the rear of the doubly linked list

// rear.getNext() is null, front.getPrev() is null

//postcondition: None of the list internal pointers are null

**private void** makeCircular()

{

}

**3.** Consider the following incomplete class that creates a doubly linked list.

**public class** DLList

{

**private** DoublyLinkedNode front, rear;//front points to the first node

// if the list is not empty; otherwise null

//rear points to the last node in the list //if the list is not empty; otherwise null

//precondition: front and rear are not null, t is not equal to front,

t is not equal to rear

//postcondition: The node referenced by t has been moved to the front.

**public void** moveNode(DoublyLinkedNode t)

{ /\* to be implemented \*/ }

//other methods and constructors not shown

}

Furthermore, assume that a non-empty doubly linked list with at least three nodes has been created that is represented by the following general representation. The precondition says that **t** points at some node in the middle of the list, neither at the first node nor at the last node in the list.

t rear

1300

1000

1500

1400

1200

1100

front

Complete method moveNode that moves the node pointed to by t and places it at the front of the list. Make sure that the list is reconnected. (**Hint**: You are NOT creating a new node.)

**public void** moveNode(DoublyLinkedNode t)

{

t rear

1300

1000

1500

1400

1200

1100

front

**4.** Complete method insert that inserts a new node, containing obj***,*** before the node pointed to by t. If the list is empty, just insert a new node containing obj as the first node in the list. How many cases do you have? \_\_\_\_ Draw them below.

**public void** insert(DoublyLinkedNode t, Object obj)

{

}

t

rear

1300

1000

1500

1400

1200

1100

front

**5.** Complete method remove that removes the node pointed to by t from the list. This node can be **ANY** node in the list. How many cases do you have? \_\_\_\_\_\_.

**public void** remove(DoublyLinkedNode t)

{