CS 102 Computer Science II
Lab 6

The class **ListDL** is intended to be an implementation of the **List** interface. The **ListDL** class implements the interface as a doubly-linked list. A **ListDL** object includes three instance variables: **head** (a reference to the first element of the list); **tail** (a reference to the last element of the list) and **count** (an integer designating the number of elements on the list). Each element of the doubly-linked list is a **ListElementDL** object with three instance variables: **next** (a reference to the next element of the list); **previous** (a reference to the previous element of the list); and **data** (the value stored in the list element). A picture of a three-element **ListDL** is found at the end of this document.

In this lab you must define the **List** interface as discussed in class. Then you must implement the **ListDL** and **ListElementDL** classes so that **ListDL** implements the **List** interface. The methods you must implement are described below. Once you have finished your implementation, you should test it by compiling and running a version of the **ParkingLot.java** application that uses the **ListDL** and **ListElementDL** classes.

ListElementDL Class:

public ListElementDL(Object d, ListElementDL n, ListElementDL p)

1. Store the parameters **d**, **n** and **p** in the **data**, **next** and **previous** instance variables.
2. If **n** is not **null**, then make the **previous** field of **n** point to **this** object.
3. If **p** is not **null**, then make the **next** field of **p** point to **this** object.

public ListElementDL next() Similar to method in ListElementSL.

public void setNext(ListElement DL n) Similar to method in ListElementSL.

public ListElementDL previous() Similar to method in ListElementSL.

public void setPrevious(ListElementDL p) Similar to method in ListElementSL.

public Object data() Similar to method in ListElementSL.

ListDL Class:

public void add(Object data) Implement by calling **addToHead** or **addToTail**.

public void addToTail(Object data)

1. Construct a new **ListElementDL** to hold **data**, with the current **tail** in its **previous** field and with **null** in its **next** field. Store the new list element in a local variable **temp**.
2. If the **tail** is not **null**, then set **tail**'s **next** field to be **temp**.
3. Set the **tail** field to be **temp**.
4. If the **head** is null, set the **head** field be the same as the **tail**.
5. Increment the **count** field.

public void addToHead(Object data) Similar to **addToTail**.

public Object removeFromTail()

1. Store the current **tail** in a temporary variable.
2. Set the **tail** field to be the element **previous** to the current **tail**.
3. If the new tail is **nul**l, then set the **head** field to **null**, otherwise, set the **next** field of the new **tail** to null.
4. Decrement the **count** field.
5. Return the data stored in the old **tail**.

public Object removeFromHead() Similar to **removeFromTail**.

public boolean contains(Object data)

1. Declare and initialize a **ListElementDL** variable "**current**" to equal the **head** of the list.
2. While **current** is not **null** and the data stored in **current** is not equal to **data**, move **current** down the list.
3. If **current** is not **null**, then **data** was found, so return **true**, otherwise return **false**.

public Object remove(Object data)

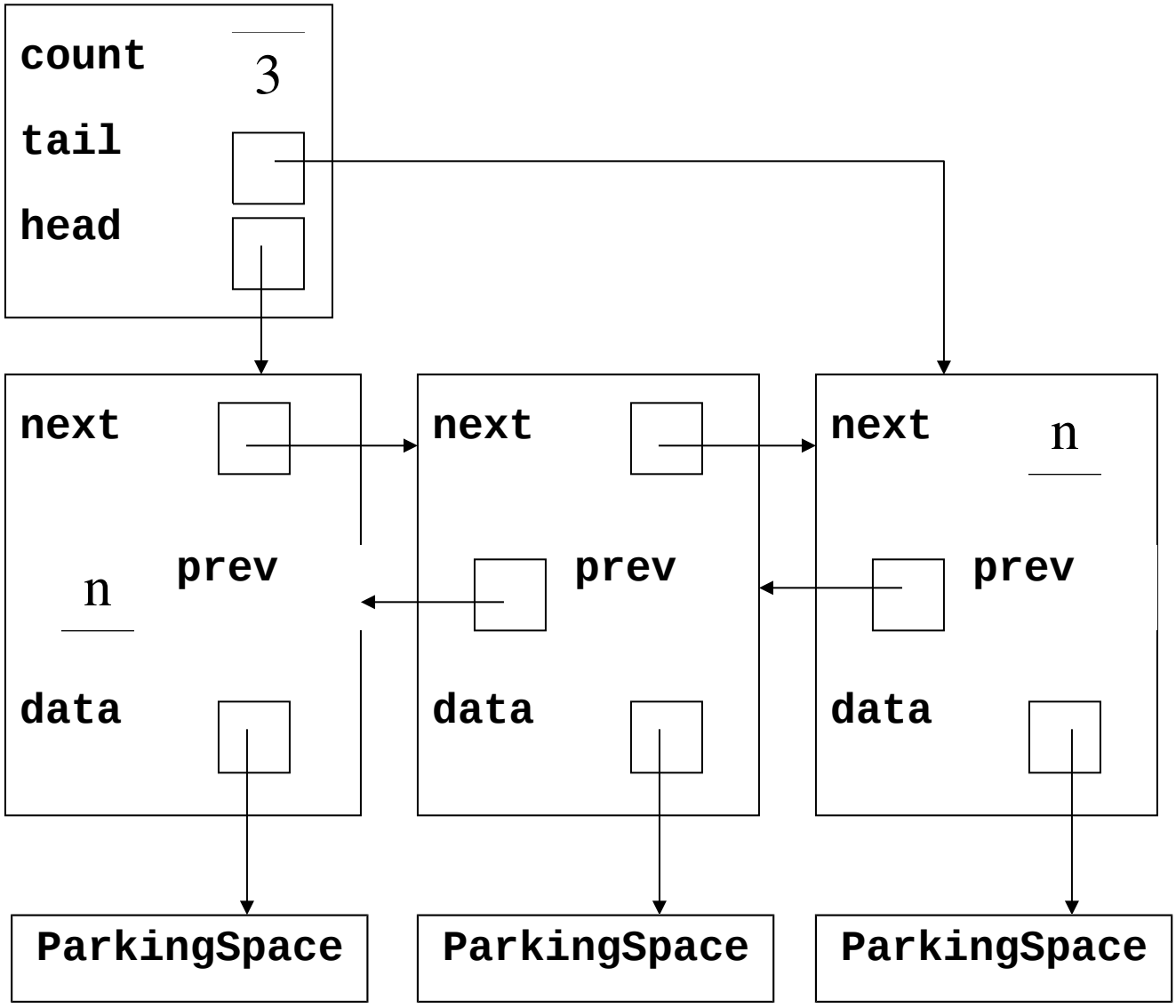
1. Declare and initialize a **ListElementDL** variable "**current**" to equal the **head** of the list.
2. While **current** is not **null** and the data stored in **current** is not equal to **data**, move **current** down the list.
3. If **current** is **null**, then **data** was not found, so return **null**. Otherwise, **data** was found, so do the following:
    a. If **current**'s **previous** field is not **null**, then make the element **previous** to **current** point forward around **current** by setting its **next** field to **current**'s **next** field. Otherwise, set **head** to be **current**'s **next** field.
    b. If **current**'s **next** field is not **null**, then make the element **next** to **current** point backward around **current** by setting its **previous** field to **current**'s **previous** field. Otherwise, set **tail** to be **current**'s **previous** field.
    c. Return the data stored in **current**.

int size() Return **count.**

boolean isEmpty() Return **true** if **count** is zero otherwise return **false.**

void clear() Set **head** and **tail** to **null**. Set **count** to zero.

String toString() Similar to implementation in **ListSL** class.


count
3
tail
head
next
prev
n
data
next
prev
data
next
n
prev
data
ParkingSpace
ParkingSpace
ParkingSpace