6. Linked Lists :: Implementation :: DList Class :: get() and getNodeAt()

The *get*() method:

`+get(pIndex: int): Integer <<throws IndexOutOfBoundsException>>`

returns the element stored in the *DList* at index *pIndex*. The method throws a *java.lang.IndexOutOfBoundsException* if *pIndex* is less than 0 or if *pIndex* is greater than or equal to the size of the list:

```
public Integer get(int pIndex) throws IndexOutOfBoundsException {
  ...
}
```

The way we are implementing the *DList* class—by permitting insertion, retrieval, and removal of elements via an index—means that it would be **very convenient** to have a method that will return a reference to the *Node* at a specific index. That is the purpose of:

`#getNodeAt(pIndex: int): Node <<throws IndexOutOfBoundsException>>`

which is called by *add*(), *remove*(), and *set*(). *getNodeAt*() throws an *IndexOutOfBoundsException* if *pIndex* is less than 0 or greater than or equal to the size of the list.

6. Linked Lists :: Implementation :: DList Class :: getNodeAt()

Here is the pseudocode for *getNodeAt*():

```
Method getNodeAt(In: pIndex) Returns Node Throws IndexOutOfBoundsException
  -- Check for pIndex out of bounds and throw exception if necessary.
  If pIndex is out of bounds Then
    Throw IndexOutOfBoundsException
  End If

  -- Since accessing the head and tail nodes is a common operation we check for
  -- those cases first.
  If pIndex = 0 Then
    Return reference to the head node
  ElseIf pIndex = getSize() - 1 Then
    Return reference to the tail node
  End If

  -- Otherwise, start at the node at index 1 and walk forward until the node at
  -- index pIndex is reached and then return it.
  node ← the node succeeding the head node
  For index ← 1 to pIndex - 1 Do
    Make node refer to the node succeeding node
  End For
  Return node
End Method getNodeAt
```

6. Linked Lists :: Implementation :: DList Class :: get() and getNodeAt()

Having implemented *getNodeAt*() which returns the *Node* at index *pIndex*, then implementing *get*() is quite simple:

```
// Returns the element at index pIndex. Thows IndexOutOfBoundsException if
// pIndex < 0 or pIndex >= mSize.
public Integer get(int pIndex) throws IndexOutOfBoundsException {
  return getNodeAt(pIndex).getData();
}
```

where *getNodeAt*() throws the appropriate exception when *pIndex* is out of bounds. If an exception is thrown by *getNodeAt*() then *get*() simply rethrows the exception. Therefore, we are ensured that the reference returned by *getNodeAt*(*pIndex*) will be nonnull and we will not encounter a *NullPointer Exception* when we call *getData*() on the *Node.*

6. Linked Lists :: Implementation :: DList Class :: getNodeAt() and set()

The *set*() method:

`+set(pIndex: int, pData: Integer): Integer <<throws IndexOutOfBoundsException>>`

changes the element at index *pIndex* to *pData* and returns the element that we replaced. For example,

```
DList list = new DList();
list.append(1);
list.append(2);
list.append(3);
list.append(4);
Integer x = list.set(2, 5);
System.out.println(list + "; x = " + x);
```

would display:

```
1 2 5 4 ; x = 3
```

By making use of *getNodeAt*(), *set*() is straightforward to implement and will be left as an exercise for the student.