5. Linked Lists :: Implementation :: DList Class :: add(), append(), and prepend()

A linked list is a **linear** data structure which means that the elements of the list are stored **sequentially** (one after the other) and therefore, the position of the elements within a list can be specified by an **index**. By convention, we shall number the indices starting at 0.

The *add*() method is called to **insert** a new element into a *DList*:

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

where *pIndex* is the index within the list where the new element is to be inserted. The method throws a *java.lang.IndexOutOfBoundsException* if *pIndex* is less than 0 or if *pIndex* is greater than the size of the list, i.e., the value returned by *getSize*():

```
public add(int pIndex , Integer pData) throws IndexOutOfBoundsException {
  if (pIndex < 0 || pIndex > getSize()) throw new IndexOutOfBoundsException();
  ...
}
```

Note that when *pIndex* is 0 we are inserting the new element before the head of the list, which we will define as a **prepend** operation:

When *pIndex* is equal to the size of the list we are inserting the new element after the tail of the list, which we will define as an **append** operation:

5. Linked Lists :: Implementation :: DList Class :: add(), append(), and prepend()

Therefore, for convenience—because appending and prepending is a very common list operation—we implement *append*() and *prepend*() as:

```
public void append(Integer pData) {
  add(getSize(), pData);
}

public void prepend(Integer pData) {
  add(0, pData);
}
```

For example:

```
DList list = new DList();
list.append(1);
list.append(2);
list.prepend(4);
list.prepend(5);
list.append(3);
list.prepend(6);
System.out.println(list);
```

Which would display: