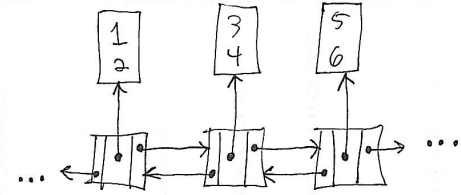
**Intrusive Linked Lists**

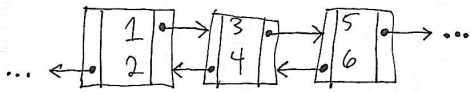
In a "normal" linked list implementation, each node *contains* a payload (data) value.

An *intrusive* linked list is one where the each node *is* a payload value. Intrusive lists are useful because they eliminate a layer of indirection between the node and data values.

E.g.: let's say that each data value consists of two integers. Here's how a normal linked list might store a sequence of pairs of integers:



In an intrusive linked list, we would define a special linked list node type that contained both the data (in this case, pair of integer values) and also the "link" fields (successor and predecessor references):



The intrusive linked list is potentially more time and memory efficient. No pointer must be followed to get from node to payload. Also, only one object is allocated per value stored in the list, rather than two.

Intrusive linked lists can be implemented in Java by parametizing the linked list with the node type, rather than the element (data) type.

Code sketch:

public class YLinkedListNodeBase<NodeType extends YLinkedListNodeBase<NodeType>> {

public NodeType prev;

public NodeType next;

public YLinkedListNodeBase() {

super();

}

}

public class YLinkedListNode<E, NodeType extends YLinkedListNodeBase<NodeType>>   
 extends YLinkedListNodeBase<NodeType> {

public E payload;

public YLinkedListNode() {

}

public YLinkedListNode(E payload) {

this.payload = payload;

}

}

public class YLinkedList<NodeType extends YLinkedListNodeBase<NodeType>> {

private NodeType sentinel;

public YLinkedList(Class<NodeType> nodeClass) {

try {

sentinel = nodeClass.newInstance();

} catch (Exception e) {

throw new IllegalArgumentException("Cannot instantiate sentinel", e);

}

sentinel.next = sentinel;

sentinel.prev = sentinel;

}

...

}