Dynamic Data Structures.
Linked Lists (Part 1).

Arrays can be used to represent sets of objects. However, arrays cannot grow or shrink in size: if another element is added to the set or removed from it – it becomes problematic.

Dynamic data structures:

Vectors - Java dynamic arrays. Can grow.

Linked Lists –structure of the following form:

```
  □ □ □ □
    ↘ ↘ ↘ ↘
```

Each box here represents an object. Each object knows its successor in the list.

Each box (called also list node) contains the object data as well as a link (reference in Java) to the next node in the list, i.e. the reference to the successor node. The last node does not have a successor; i.e. its reference to the next node is set to null.

The first node in the list is called the head node.

Why use Linked Lists to represent data sets?

Mainly because insertion and deletion of new nodes is less problematic than insertion/deletion in an array.

INSERTION: with arrays when run out of space need to allocate a bigger array and copy all elements to the new array – never happens in linked lists.

DELETION: in arrays blank cells use up memory – inefficient.

The drawback of linked list representation is the fact that in order to get to, for example, the 4-th object in the list one has to start from the head node and go through the first three following the link.

In an array to get an object one just needs to reference it by index.

This affects a number of algorithms for example, search in a linked list of n elements always takes O(n) in the worst case, even if the list is sorted, whereas in a sorted array BinarySearch runs in only O(lg n) time. An analog of BinarySearch cannot be implemented efficiently for LinkedLists, because in the linked list we cannot jump to the middle element in one operation, as we did in an array.

So linked lists are useful to represent highly dynamic sets, i.e. those that grow and shrink a lot, thus requiring many insertion/deletion operation, but for which fast search is not a very high priority.
While learning about the details of Linked Lists and their implementation in Java we’ll use the following example problem:

Implement a program that keeps track of orders in a pizza shop. The set of orders should be represented by a linked list, where each node is an object representing a pizza order. A node is added to the linked list when a new order is placed and removed from the list when an order gets delivered or the orderer cancels it.

Each order has a name of the orderer, their address, the type and the quantity of the ordered pizza pie.

Representing the nodes:
Each node is an object that in addition to the order’s attributes has an additional attribute that stores a reference to the next node in the list.

Example:

```java
public class PizzaOrderNode {
    // instance variables
    private String name;     // name of person placing the order
    private String address;  // address of person placing the order
    private String pizzaType; //
    private int quantity;     //

    private PizzaOrderNode next; // a link to the next node in the list

    // Constructor

    public PizzaOrderNode (String orderedBy, String oAddress,
                           String type, int howMany) {
        name = orderedBy;
        address = oAddress;
        pizzaType = type;
        quantity = howMany;

        next = null; // no successor node, so set the next to null
    }
}
```
To connect a node (referenced by pNodeOne) to another node (referenced by pNodeTwo), the instance variable next of pNodeOne’s object must be set to pNodeTwo.

The goal is:

Thus, the class definition need a method that sets the value of instance variable next to the object that is passed in as a parameter.

```
// mutator method setNext (PizzaOrderNode aNode)
// Sets the next field of the calling
// object to the object that is passed in as a parameter, aNode

public void setNext (PizzaOrderNode aNode)
{
    next = aNode;
}
```

Example: creating two objects of class PizzaOrderNode and linking them together by appending the second object to the first.

```java
PizzaOrderNode pNodeOne = new PizzaOrderNode ("Gary", "12 Main Str.", "cheese", 3);

PizzaOrderNode pNodeTwo = new PizzaOrderNode ("Clair", "3 Cambridge Ave.", " special", 1);

// append pNodeTwo to pNodeOne
pNodeOne.setNext(pNodeTwo);
```