ArrayList<Integer> a = new ArrayList<Integer>();

LinkedList<String> b = new LinkedList<String>();

a.add(5); // appends 5

a.add(4); // appends 4

a.get(1); // gets the element at index 1

**Array Lists**

Reads values in constant time

Slightly inefficient when inserting new values.

**Linked Lists**

Reading values require iteration through the list.

Efficient at inserting values into the list.

**Generic Typing**

A method not knowing the **exact class** of the object being parsed into it:

* Relying on **“common”** methods between classes

A Class not knowing the exact class of the object being stored inside it.

* Example with employees + more specific types of employees (board)

Example

* You want to code a  
    
  class Array() {   
   ArrayList();  
   public Array(int a) {  
   }  
  }  
    
  such that it can take in any type and store it as an array.

Example

* ArrayList<….> (how do you specify if you want to store integer, string etc.)
* This is done through Generic Typing.
* Allows calculations / methods / actions regardless of what type it is.

Set

* Collection of items of the same type (int, string … etc.)
* Contains unique elements (i.e. no duplicates
  + Somewhere in logic you must state that if user tries to add in same element, then it will fail the method.
* Homogenous

Sets – Operations

* **Union** – returns a new set with all unique elts in sets A and B
* **Intersection** – returns a new set containing the common elements of 2 sets
* **Equals** – checks whether two sets are equal or not – i.e. contain the same elts. Recall that if set A = set B:
  + **Size(Union(A,B)) = size(A) = size(B)**
  + And **size(Intersection(A,B)) = size(A) = size(B)**

Interfaces

* **Blueprint** of a certain “property” or characteristic a class can have.