Array

Declaring an array does not create it! No memory is allocated for individual array elements. This requires a separate creation step.

Declaration (2 ways)	Initialization
<u>**ONE**</u> `datatype[] arrayname1, arrayname2;` **Example:** `int[] myArray1, myArray2;` <u>**TWO**</u> `Datatype arrayname[];` **Example:** `int myArray1[], x, myArray2[];`	`arrayName = newdatatype[arraySize];` **Example:** `myList = new double[8];` **NOTE:** *The new keyword creates an object or array. The object or array is created in a location of memory called the heap. A reference (pointer) to the array is assigned to the variable.*

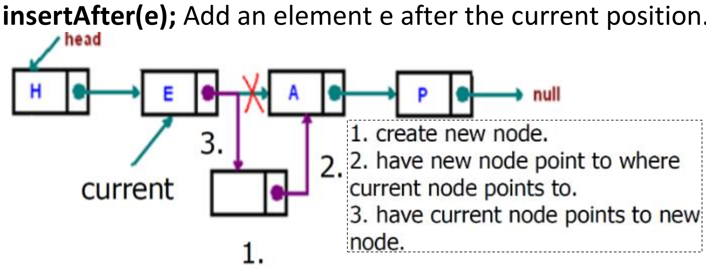
2D Array

```
int square[][];
```

```
public void printSquare() {
 for (int row = 0; row < square.length; row++) {
   for (int col = 0; col < square.length; col++) {
       System.out.printf("%3d", square[row][col]);
   }
   System.out.println();
 }
} // END OF printSquare METHOD
```

LINKED LIST

insertAfter(e); Add an element e after the current position.



current(); Returns the current element.

size(); Returns the number of elements on the list.

forward(); Move the current position forward one position.

```
public void forward() {
     Node tmp = current.getNext();
     if (tmp != null) current = tmp;
}
```

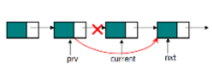
backward(); Move the current position backward one position.

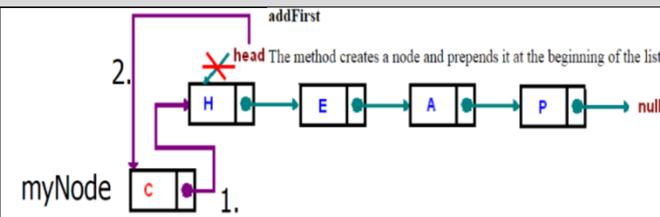
```
public void backwards(){
    if(head != current){
//Create a node to traverse the list in order to
// find the Node before the current one
        Node tmp = head;
// While the next node for tmp is not the current one
// and not the end of the list, step forward one node
        while((tmp.getNext() != current)&&(tmp.getNext()!=null)){
            tmp = tmp.getNext();
        }
        current = tmp;
    }
}
```

resetCurrent(); Reset the current position at the head element.

remove(e); Element e is removed from the list.

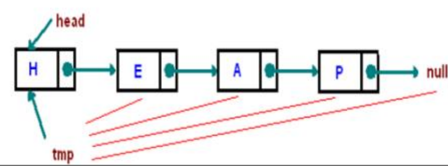
```
public void remove(Object o) {
 if (size!=0)     {
     //Node prev will point to the Node just before the node being removed
     Node prev = null;  Node tmp = head;
     while (tmp.getNext()!=null && tmp.getElement()!=o) {
       prev = tmp;
       tmp = tmp.getNext();
     }
     if (tmp.getElement()==o)   {
       current = current==tmp ? prev:current;
       prev.setNext(tmp.getNext()); // General condition
       size--;
     }
   }
}
```





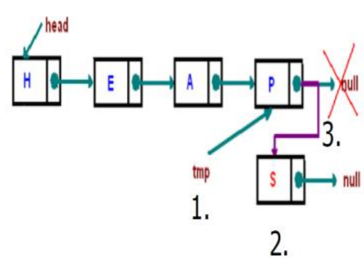
Traversing

Start with the head and access each node until you reach null. Do not change the head reference.

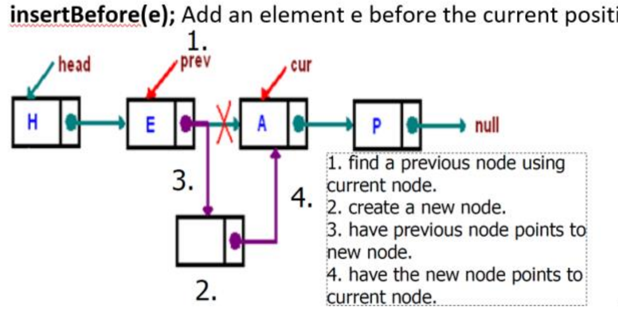


addLast

The method appends the node to the end of the list. This requires traversing, but make sure you stop at the last node



insertBefore(e); Add an element e before the current position.



Primitive Data Types (Byte, Short, Int, Double, Etc.)	Ordering of Operator Precedence & associativity
byte (8 bits), **short** (16 bits), **int**(32 bits), **long** (64 bits) **float** (32 bits), **double** (64 bits) **char** - Unicode! e.g., '\u12ab' (16 bits) **Boolean**(16 bits, true/false)	All are LEFT TO RIGHT, ***except*** Unary, Conditionals, and assignment operators **Unary:** +, -, ++, --, !
The 80/20 rules	**General optimization techniques**
❑ In general, ***80% percent of a program's execution time is spent executing 20% of the code.*** ❑ This means that a small part of the code is running most of the time, and the bigger part of the code is running seldom. ❑ 90%/10% for performance-hungry programs. ❑ 90 percent of a program's execution time is spent running 10 percent of the code. ❑ Spend your time optimizing the important 10/20% of your program. ❑ Optimize the common case even at the cost of making the uncommon case slower.	❑ **Strength reduction** ▪ Use the faster and cheaper version of an operation ▪ *E.g.* `x >> 2` *instead of* `x / 4   // Note: readability issue here!` `x << 1` *instead of* `x * 2` ❑ **Common sub expression elimination** ▪ Reuse results that are already computed and store them for use later, instead of re-computing them. ▪ *E.g.* `double x = d * (limit / max) * sx;` `double y = d * (limit / max) * sy;` `double depth = d * (limit / max);` `double x = depth * sx;` `double y = depth * sy;`