Lecture 15
CSE11 – Fall 2013
Arrays

Arrays

- A “collection” of objects, stored that can accessed by an index (also known as a subscript)
- Example
  - String [] sArray = new String[20];
  - defines sArray to have 20 slots.
- Each slot has either a String or is **null**
  - We call the objects stored in the slots: “elements”

Array Initializers

- int [] myArray = {1,2,3,4,5};
  - Creates and initializes and array of 5 ints;
- String [] words = {“please”, “excuse”, “my”, “dear”, “aunt”, “sally”};
  - Creates and initializes an array of 6 Strings

Declaration and new

Create an Array of Strings of dimension 20

```
String [] sArray = new String[20];
```

`sArray =`

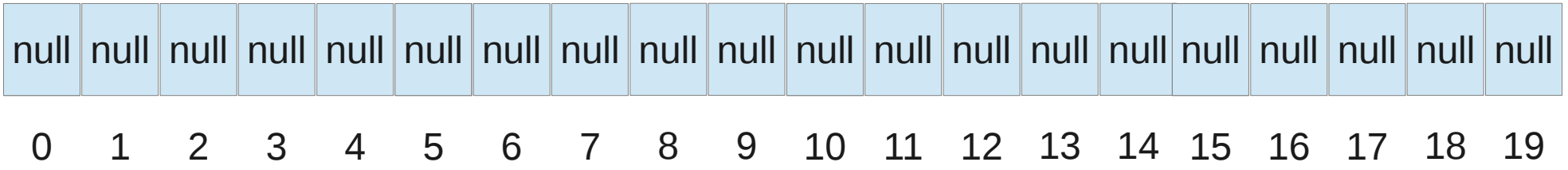
null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null	null

- The slots are “allocated” but contain no String objects.
- In general, we say that each element of the array is a “reference” to an object
- At initialization all elements of the array reference the **null** object.

The Array Indices (Java is 0-based indexing)

```
String [] sArray = new String[20];
```

`sArray =`

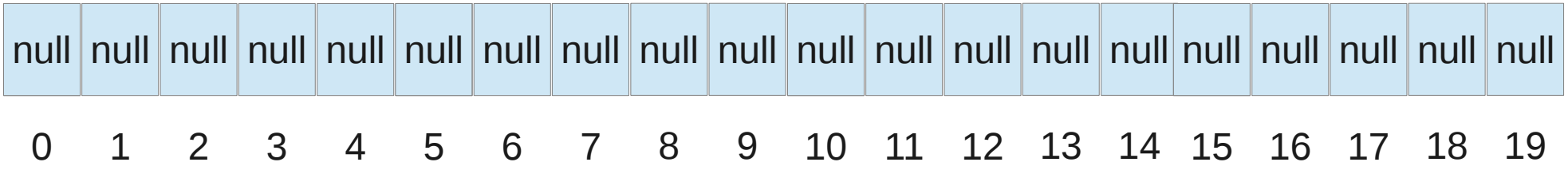


- The first element of the array is at index 0
- The size of the array (number of slots, whether empty (null) or full (have a valid reference) is available as `sArray.length`
- The last index is (sArray.length - 1).

Accessing an Element of the Array

```
String [] sArray = new String[20];
```

sArray =

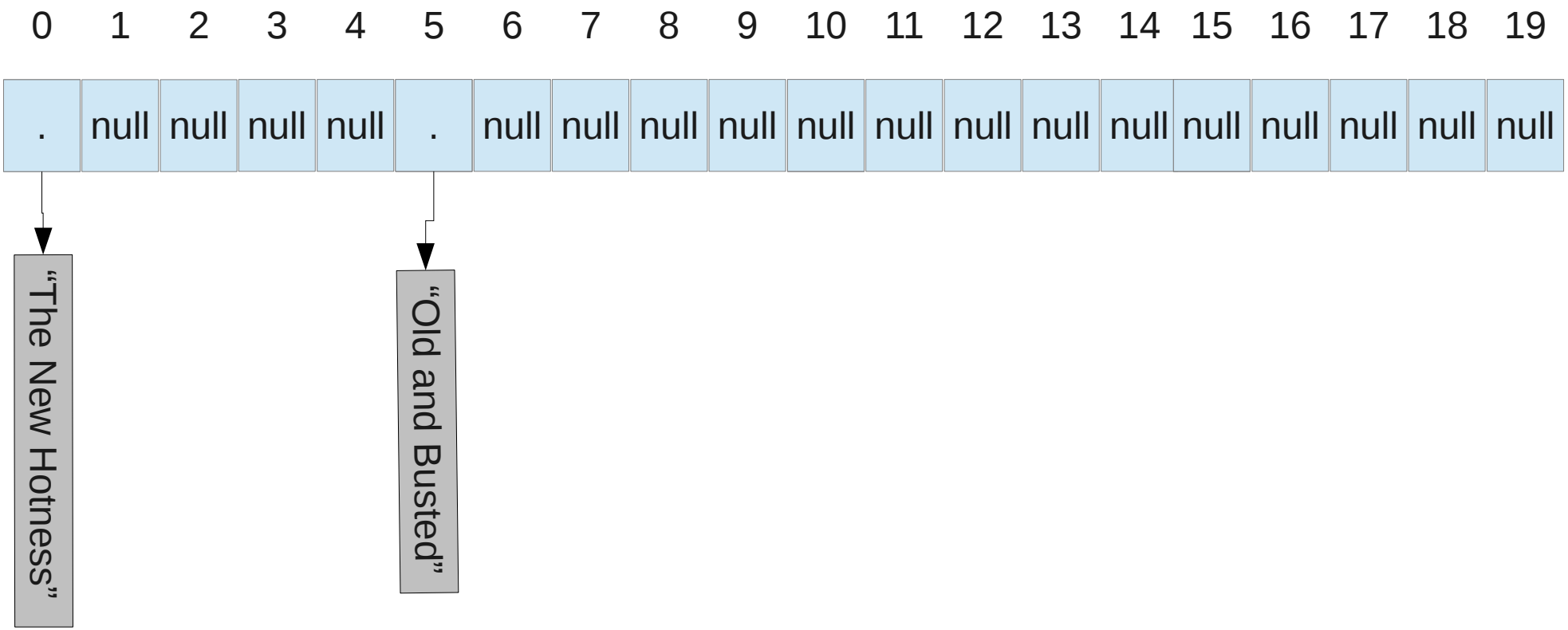



```
String  arrayMember = sArray[i];
```

- arrayMember is the “i+1”th element of the array
  - sArray[0] is the first element. sArray[6] is the 7th element
- It is a runtime error for index < 0
- it is runtime error for index >= array.length

Storing Data into an Array

```
String [] sArray = new String[20];
sArray[0] = new String("The New Hotness");
sArray[5] = new String("Old and Busted");
```



Different Kinds of Arrays

- Arrays are a *linear data structure* where all elements are of the same object type
- It is possible to have arrays of primitive types. The difference is the array is initialized with 0's instead of nulls.
  - int vector[3];
  - double coefficients[5];

new and arrays of primitive types

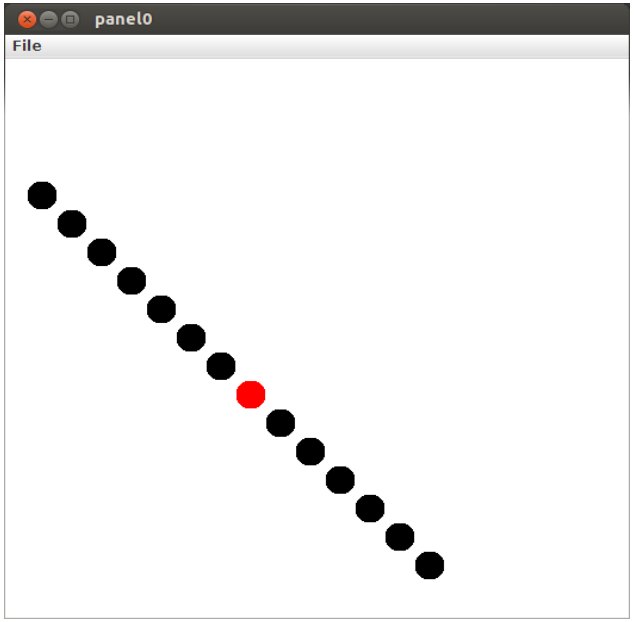
```
int size = 15;
FilledRectangle[] rectangles;
rectangles = new FilledRectangle[size];


// the following is correct, even though int is a
// primitive
int[] midtermScores;
midtermScores = new int[size];
```

Arrays that have different “types” of objects

- Elements of an Array
  - All the same class (or any extended version of that class)
    - e.g. WindowController [] = new WindowController[15];
      - Could have WindowController instances
      - And/Or class DiagonalLoop extends WindowController
      - And/or class myDiag extends DiagonalLoop
  - Instances that support an Interfaces
    - e.g. MyListeners[] = new ActionListener[100];
      - Could store ANY object (instance) that implments the ActionListener interface
      - MyListener[4].actionPerformed(evt) invokes the actionPerformed Method on the 5th listener
        -

Method Invocation with Array Members

- Modify Diagonal.java so that we could click and drag on any circle on a diagonal and move the entire array of circles
- And color the circle we clicked on red.



Code Fragments (DiagonalArray.java)

```
private FilledOval circles[];
private Location startPoint;
private FilledOval clicked;
private boolean dragging = false;

 public void onMouseClick(Location point) {
        instructions.hide();
        int ncircles;
        if (circles != null) return;

        ncircles = Math.min(canvas.getWidth(), canvas.getHeight())
                                /DIAMETER;
        circles = new FilledOval[ncircles];
        for (int i = 0; i < ncircles; i++)
                circles[i] = new FilledOval(i*DIAMETER,i*DIAMETER,
                        DIAMETER, DIAMETER, canvas);
}
```

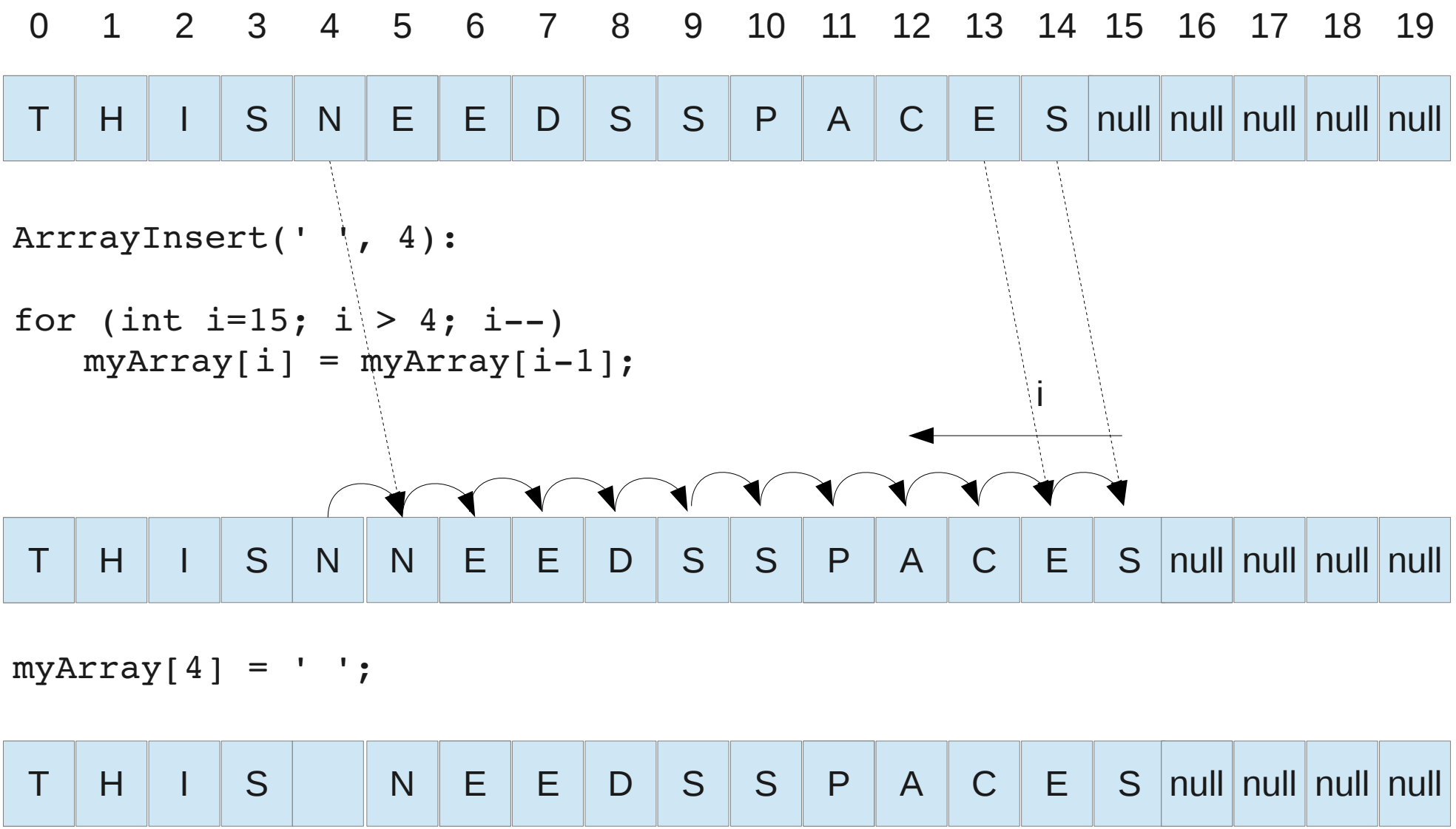
Code Fragments (DiagonalArray.java)

```
public void onMousePress(Location point)  {
        if (circles == null) return;
        // determine if mouse in any of the circles

        for(int i =0; i < circles.length; i++)
                if (circles[i].contains(point))
                {
                        dragging = true;
                        (clicked = circles[i]).setColor(Color.RED);
                }

        startPoint = point;
}
 public void onMouseDrag(Location point) {
        if (dragging) {
                double dx = point.getX() - startPoint.getX();
                double dy = point.getY() - startPoint.getY();
                // move all the circles
                for(int i =0; i < circles.length; i++)
                        circles[i].move(dx, dy);
                startPoint = point;
        }

}
public void onMouseRelease(Location point) {
        if (clicked != null)
                clicked.setColor(Color.BLACK);
        clicked = null;
        dragging = false;
}
```

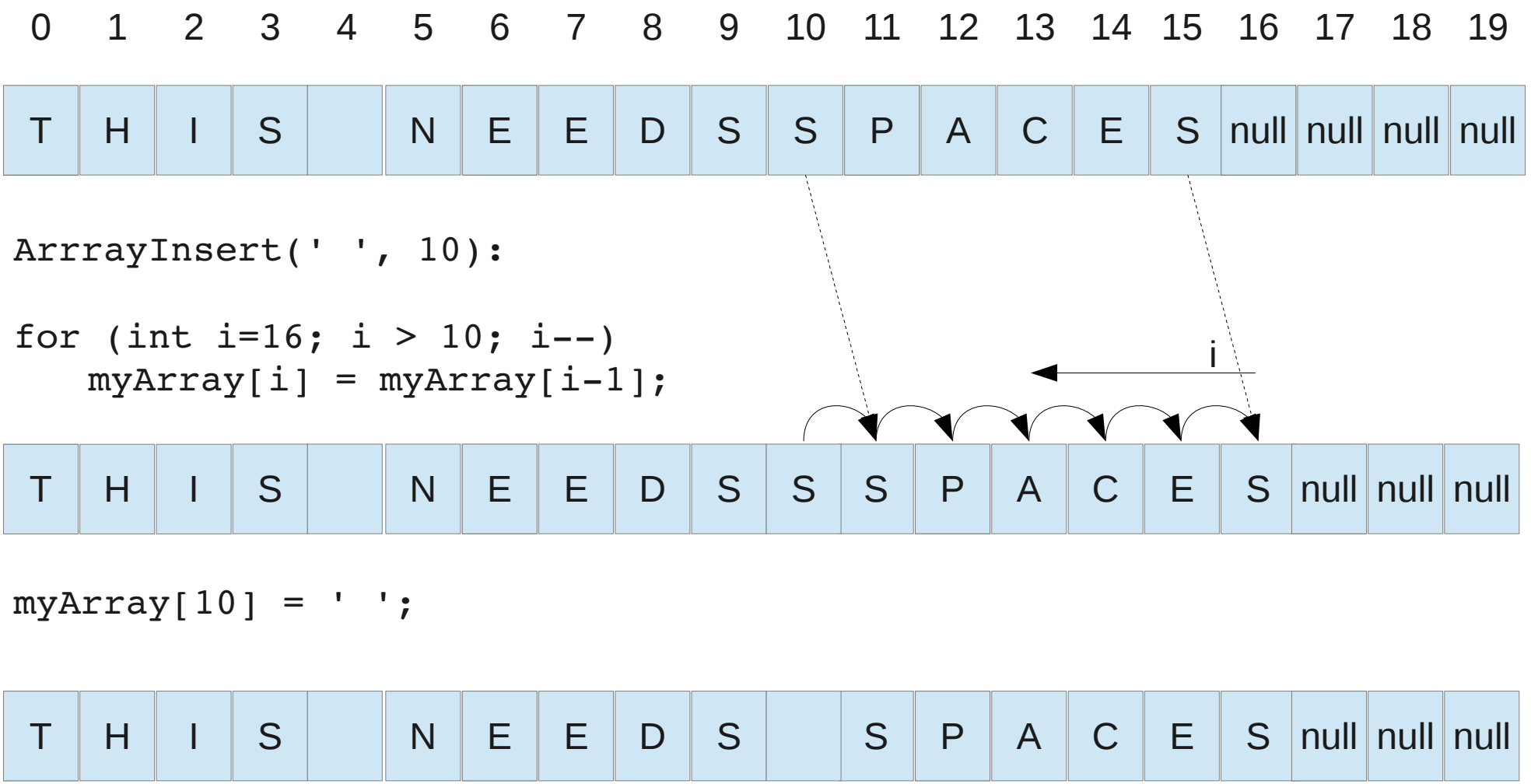
Common “operations” on Arrays

- There are NO native operations on arrays, but there are common operations
- Insert an element at a particular location
- Delete an element at a particular location
- Sort the contents of an array (complete sort is a later chapter)
- Find/remove duplicate entries
- Compress (remove empty entries)

Insert Into an Array

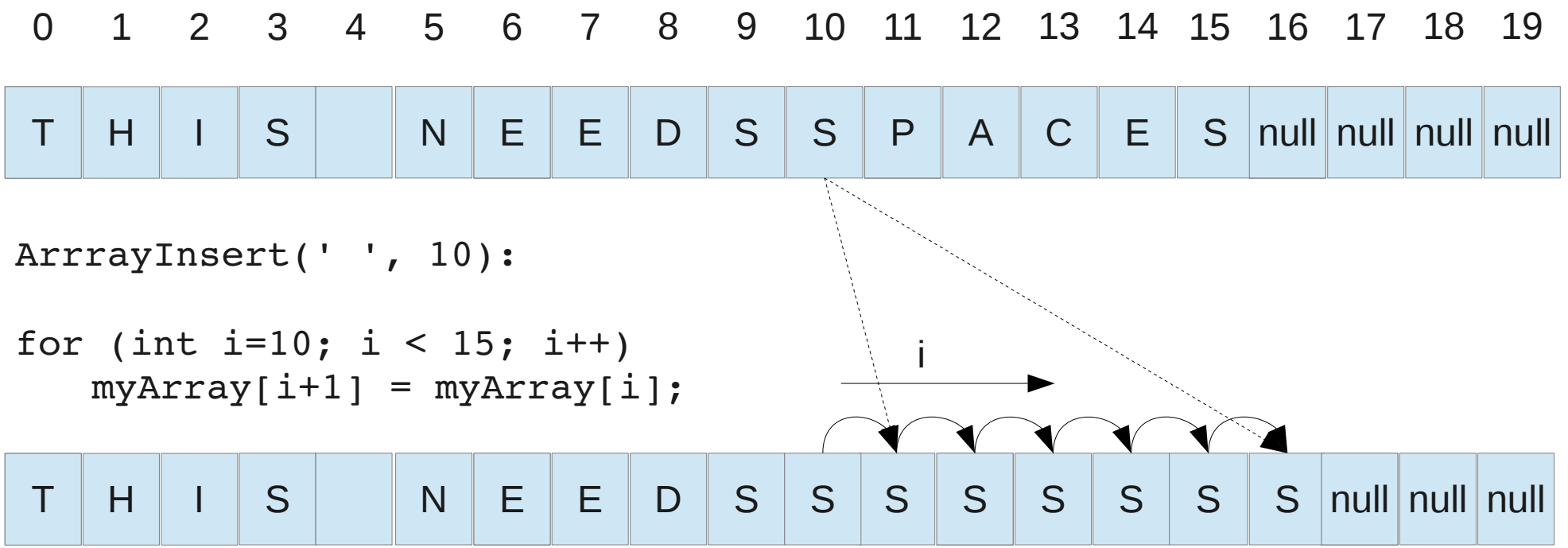


Insert Into an Array



Common Insertion Error

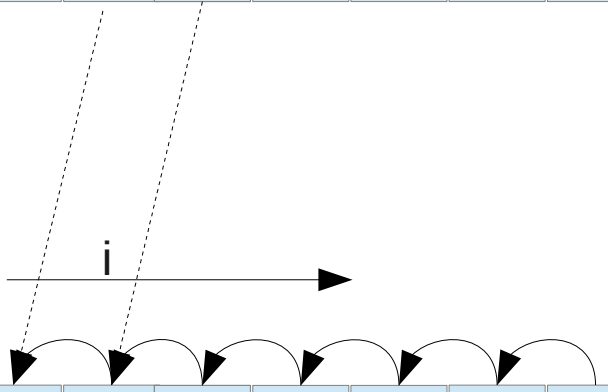
- Counting upwards instead of downwards, when moving original array elements to higher locations



Delete From an Array

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	H	I	S		N	E	E	D	S		A		D	D	E	L	E	T	E

```
ArrrayDelete(13):

for (int i=13; i < myArray.length - 1; i++)
    myArray[i] = myArray[i+1];
```



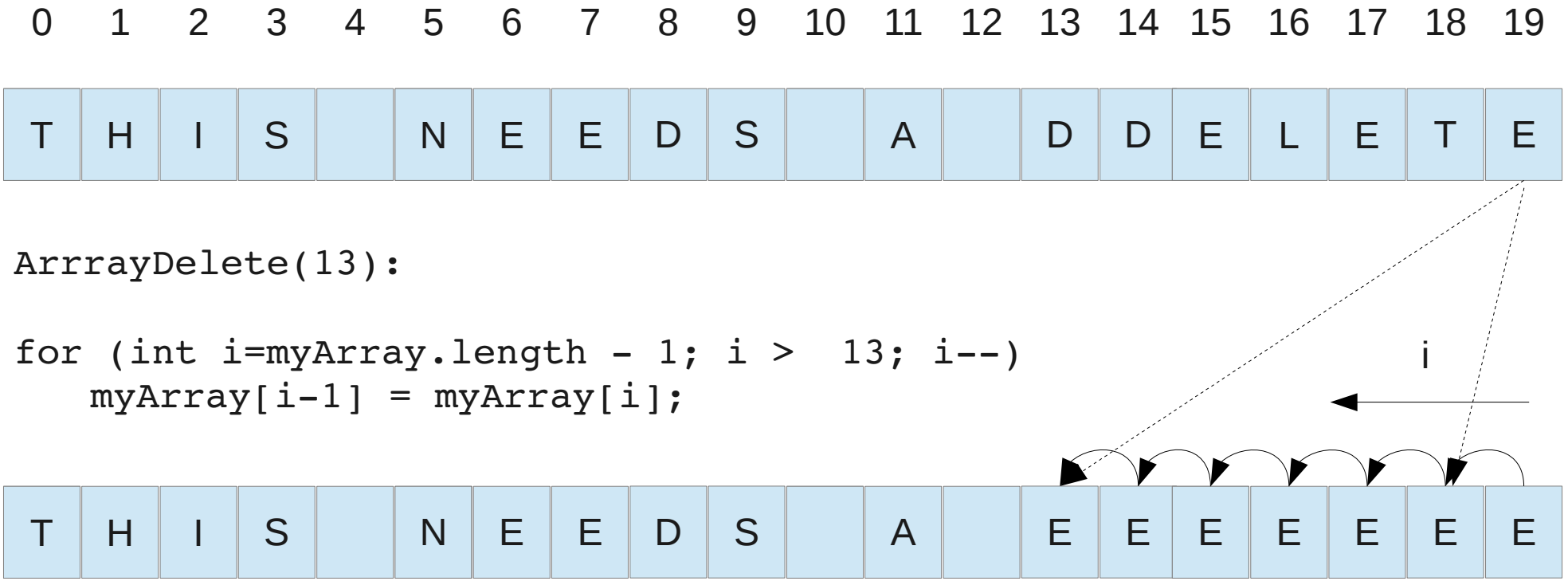
T	H	I	S		N	E	E	D	S		A		D	E	L	E	T	E	E

```
myArray[myArray.length - 1] = null;
```

T	H	I	S		N	E	E	D	S		A		D	E	L	E	T	E	null

Common Deletion Error

- Counting downwards instead of upward when moving original array elements to lower locations



length vs. number of elements

- Array.length – total number of available slots
- number of filled elements – your program must keep track!
- It is very common to get these confused in code.

Common Practice

- An array of length N has K <= N “used” elements
- The “used” elements (those with objects stored in them) are numbered from 0 .. K-1.
  - No “holes” in the array
- This is COMMON but not universal

Compaction using a Delete Method

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	H	I	S	null	N	E	E	D	S	null	C	O	M	P	A	C	T	null	null

```
myArray.nelem = 18;
for (j=0; j< myArray.nelem; j++)
{
    if (myArray[j] == null)
    {
        myArray.delete(j);
        myArray.nelem--;
    }
}
```

T	H	I	S	N	E	E	D	S	C	O	M	P	A	C	T	null	null	null	null

myArray.nelem = 16

THERE IS A SUBTLE LOGIC BUG IN THIS CODE. CAN YOU FIND IT (fails in a particular case)?

Removing Duplicates using Delete

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
T	H	I	S	null	N	E	E	D	S	null	D	E	D	U	P	null	null	null	null

```
myArray.nelem = 16;
for ( int j=0; j< myArray.nelem; j++)
{
    for (int k = j+1; k < myArray.nelem; k++)
    {
        if (myArray[j] == myArray[k])
        {
            myArray.delete(k);
           myArray.nelem--;
        ]
    }
}
```

T	H	I	S	null	N	E	D	U	P	null	null	null	null	null	null	null	null	null	null

myArray.nelem = 10

THERE IS A SUBTLE LOGIC BUG IN THIS CODE. CAN YOU FIND IT?

move using insert and delete

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L	E	A	S	E		M	O	V	E		P	U	P	null	null	null	null	null	null

```
move(int from, int to)
{
    tmp = myArray[from];
    myArray.delete[from];
    myArray.insert(tmp, to);
}
```

P	L	E	A	S	E		M	O	V	E		U	P	null	null	null	null	null	null

move using insert and delete

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
L	E	A	S	E		M	O	V	E		P	U	P	null	null	null	null	null	null

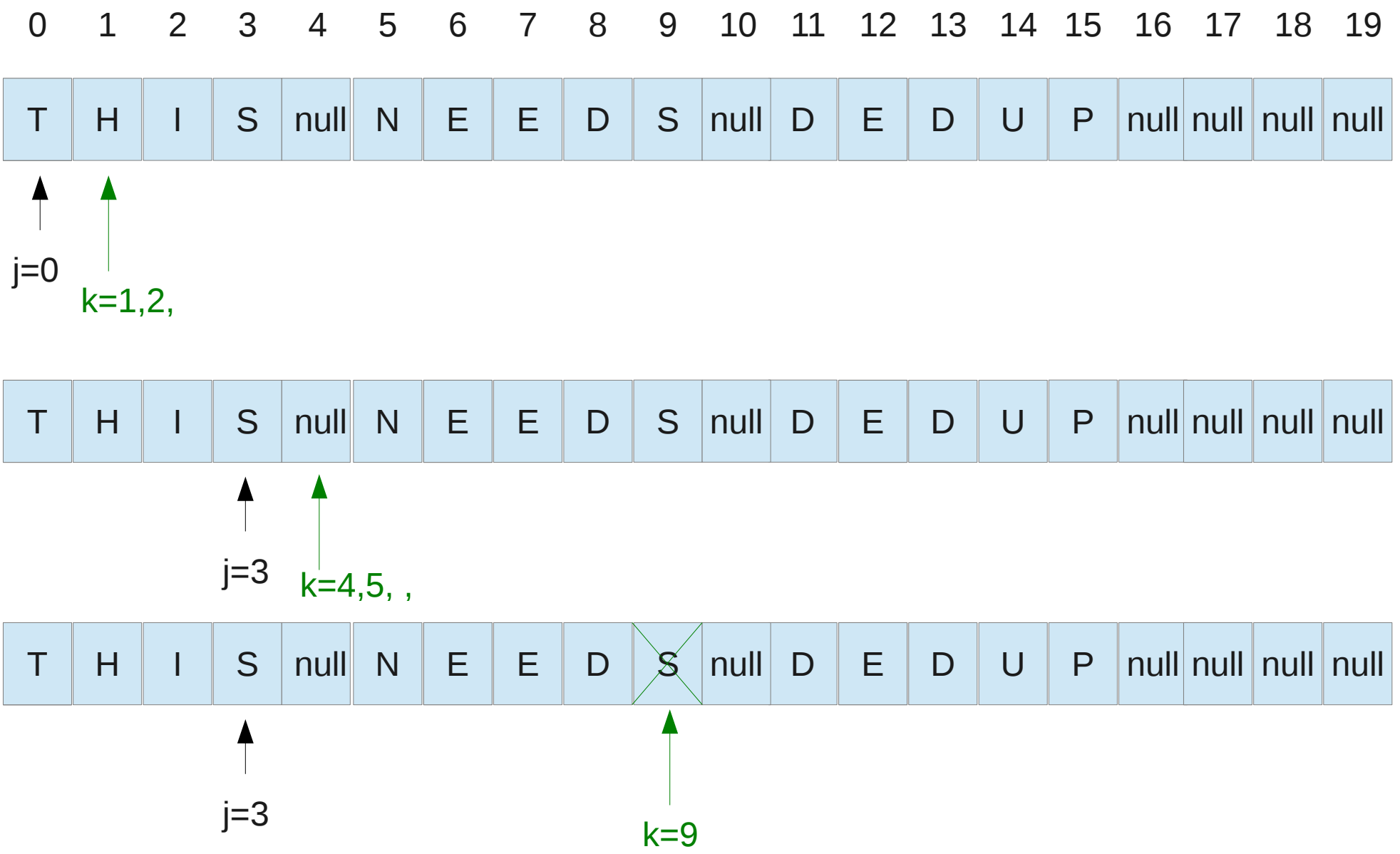
```
move(int from, int to)
{
    tmp = myArray[from];
    myArray.delete[from];
    myArray.insert(tmp, to);
}
```

P	L	E	A	S	E		M	O	V	E		U	P	null	null	null	null	null	null

When designing/building Array-base Methods

- ALWAYS check bounds at the entry of a method.
  - all arguments that are indices into the array
    - arg >= 0
    - arg < array.length
- The code the goes through an array is often called “walking” the array
  - Many array methods are step-by-step iteration
  - Often refer to index “pointers” that walk array

Removing Duplicates using Delete



The java.util.Arrays

- A utility library with the following capabilities
  - sort
  - binary search
  - fill
  - copyOf
  - copyOfRange
  - equals
  - toString