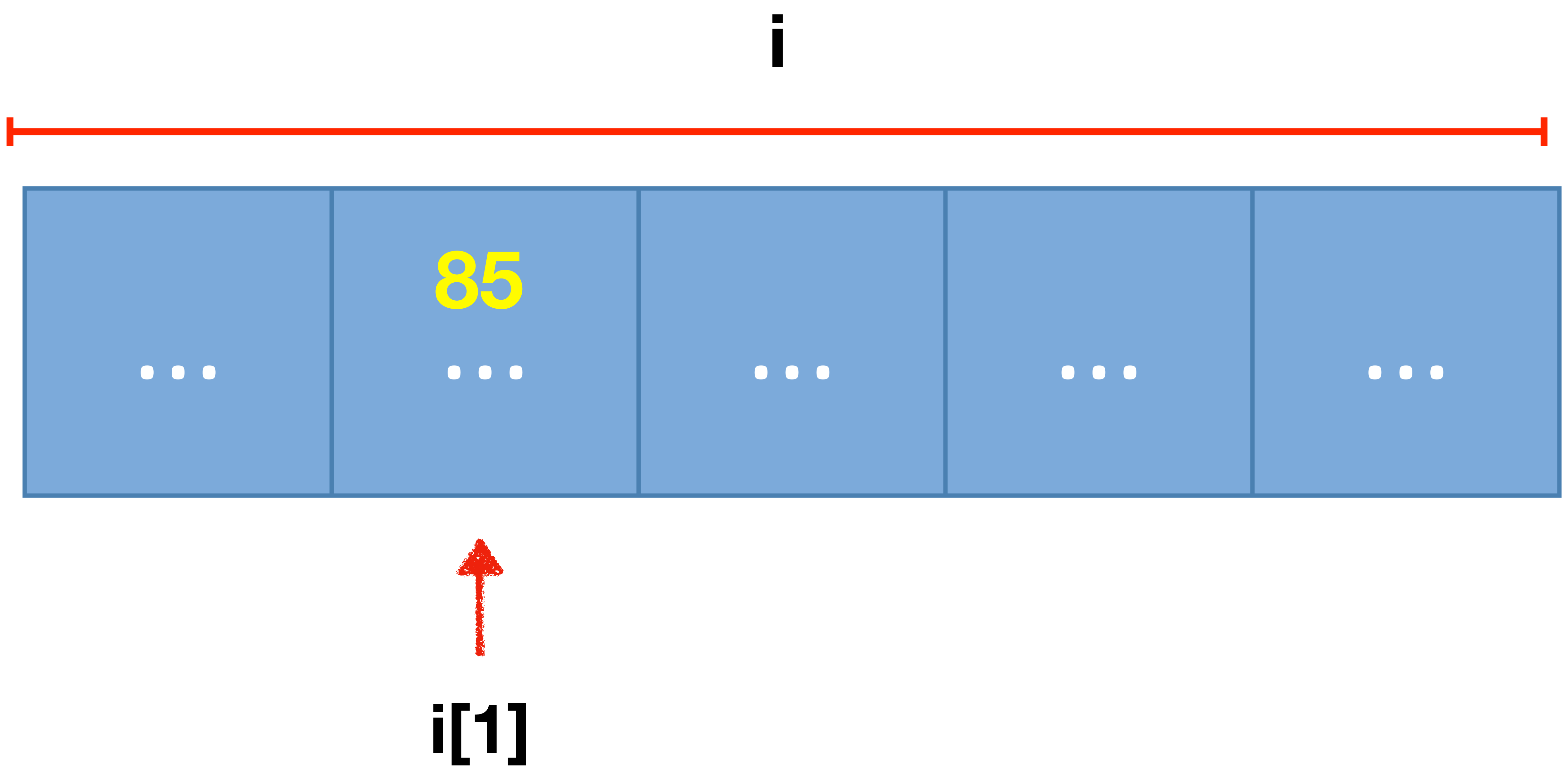
Programming Using Java

Session 8: Searching/Sorting

Arrays

```
class MyClass {
 static void main() {
  int i[] = new int[];
  i[1] = 85;
 }
}
```

- ordered group of elements of the same type
- fixed number of elements
- elements can be accessed by index (zero-based)



```
public class NotFoundException extends Exception {
   NotFoundException() {super(); }; NotFoundException(String s) { super(s);}
}
```

```
public class Search {

 public static int linearSearch(double arr[], double val) throws NotFoundException {

   int d = 0;
   while ((arr[d] != val) && (d < arr.length)) d++;
   if (d == arr.length) throw new NotFoundException();
   return d;
  }
}
```

hits the end of the array

value found

```
public class User {
   public static void main(String[] args) {
      double dArray[] = {2,4,6,8,10,12,14};
      double v = 10;
      try {int f = Search.linearSearch(dArray, v); } catch (NotFoundException e) {…}
  }
}
```

Linear Search

```
public class NotFoundException extends Exception {
   NotFoundException() {super(); }; NotFoundException(String s) { super(s);}
}
```

```
public class Search {
 public static int binarySearch(double arr[], double val) throws NotFoundException {
   int low = 0; int high = arr.length; int mid = (low + high + 1) / 2;
   do {
    if ( arr[mid] == val ) return mid;
    else if ( val < arr[mid] ) high = mid - 1;
    else low = mid + 1;
    mid = (low + high + 1) / 2;
   } while( low <= high );
   throw new NotFoundException();
}
```

value found

hits the end of the array

```
public class User {
   public static void main(String[] args) {
      double dArray[] = {2,4,6,8,10,12,14};
      double v = 10;
      try {int f = Search.binarySearch(dArray, v); } catch (NotFoundException e) {…}
  }
}
```

Binary Search

Sorting

- placing data elements in <u>some defined order</u>
    - integers are natural
    - strings???
- helps humans interpret and visualize information that can be inferred from the data elements
- helps computers detect patterns, isolate abnormal elements and classify individual elements actions usually include input, output or computation operations

```
public class Sort {
   public static double[] insertionSort( double[] arr ) {
    for(int i = 1; i <  arr.length; i++) {
      int j = i;
      double currentElement = arr[i];

      while(j > 0 &&  arr[j-1] > currentElement) {
        arr[j] = arr[j-1];
        j--;
      }
      arr[j] = currentElement;
     }
   return arr;
  }
}
```

beginning of unsorted array

find location in sorted array for element

place element in sorted array

```
public class User {
   public static void main(String[] args) {
      double dArray[] = {5,3,4,2,1};
      dArray = Sort.insertionSort(dArray);
    }
}
```

Insertion Sort

Classification	Operator	Informal Description
$O(1)$	"constant time"	computation effort not dependant on the number of elements in the data structure
$O(n)$	"linear time"	computation effort grows at rate proportional to the number of elements
$O(n^2)$	"quadratic time"	computation effort quadruples when the number of elements to be processed doubles
$O(\log_2 n)$	"logarithmic time"	computation effort grows by 1 when the number of elements to be processed doubles

Big O Notation

"…quantifying amount of work needed to solve problem…"