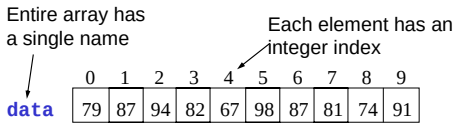
- Previous Lecture:
  - Overloading
  - Calling instance methods (and beware of null)
  - Review with Person class
- Today's Lecture:
  - 1-d array
  - Linear search
  - Binary search [in section]
  - Selection sort
- Reading:
  - Sec 8.6-8.8, 8.11

Arrays

- An array is an object
- An array is an ordered list of values (or objects)
- Each element is of the same type



An array of size N is indexed from 0 to N-1

Array declaration

```
type[] identifier;
```

Examples:

```
int[] counts;
double[] price;
boolean[] flip;
char[] vowel;
String[] names;
Interval[] series;
```


Array construction (instantiation)

```
new type[ size ]
```

must be an integer

Example:

```
new int[4]
```

Declaration & creation:

```
int limit= 4;
double[] price;
price= new double[limit];
```


Array declaration & construction

```
type[] identifier = new type[size];
```

Example:

```
int[] counts= new int[4];
```

Then values can be assigned into the cells, e.g.:

```
counts[0]= 6;  counts[2]= 9;
```


Array length and default values

Once created, an array has a fixed length, held in the array's constant called length:

```
int[] counts= new int[4];
System.out.println(counts.length);
// will print 4

System.out.println(counts[2]);
// Array components have default
// values.  Above statement will
// print 0
```

Array creation with initializer list

Create an array using an initializer list:

```
int[] x= new int[]{6,3,4,8};
```

Length of array is determined by length of the initializer list. Shortcut:

```
int[] x= {6,3,4,8};
```

Only when declaring & creating in same statement!

Index operator [ ]

***identifier*[*integer_expression*]**

Accesses an element of the array, e.g.:

```
int[] count= new int[101];
   // declaration & instantiation
count[70+9]= 98;
   // set count[79] to 98
int face= (int) (Math.random()*6);
count[face]= count[face] + 1;
count[face]++;
```


Elements in an array

If `count` is of type `int[]`, i.e., an array of `int`s, then the type of

`count[i]`

is `int` and `count[i]` can be used anywhere an `int` variable can be used

Type of count: int[]

Type of count[i]: int

Pattern for processing an array

```
// assume an array has been
// created and is referred to by
// variable A

for (int i=0; i<A.length; i++) {

  // perform some process
  // (on A[i])

}
```


Example

```
// Create an array of length 6
// with random numbers in the range
// of 5 to 9.  Calculate the sum.

double[] a= new double[6];
double sum= 0;  // sum so far
for (int j=0; j<a.length; j++) {
   a[j]= Math.random()*4+5;
   sum= sum + a[j];
}
```



```
// Linear Search:
// f is index of first occurrence of value z in array a
int f, k= 0;
while ( a[k]!=z  &&  k<a.length )
   k++;
if (k==a.length)
   f= -1; //signal for z not found
else
   f= k;
```

a. Correct
b. Incorrect: f is off by one
c. Incorrect: while condition is wrong
d. Incorrect: if condition is wrong

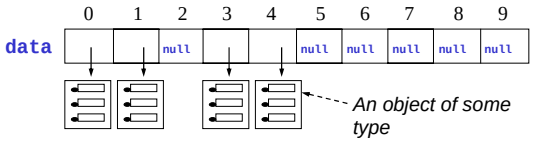
Sorting

- Arrange elements in a list in some order
- Must specify which order
- Sort “in-place”
- Many algorithms:
  - Select sort
  - Insertion sort
  - Bubble sort, ...


```
public static void selectSort(double[] a){


  // Loop from first to second last element
  // Index i: 1st cell in unsorted segment
  for (int i=0; i<a.length-1; i++){
      // Find index of min in unsorted segment









      // Swap i-th element with min



  }
}
```

Array of objects

- An array is an object
- Elements of an array can be object references
- Each element is of the same type



An array of size N is indexed from 0 to N-1

Creating an array of objects

Three steps:

1. Declare array reference variable
   `Interval[] series;`
2. Instantiate array of object references
   `series= new Interval[4];`
3. Instantiate individual objects
   `series[0]= new Interval(0,5);`
   `series[1]= new Interval(1,7);`

Many `Interval`s

- Class `ManyIntervals` is a client of class `Interval`.
- Create an array of `Interval` objects with random `base` and `width` values. Use integer values.
- Find the `Interval` with the highest endpoint.
- Search for the first `Interval` that has a specific endpoint value


```
class Interval {

  private double base;  // low end
  private double width; // interval width

  public Interval(double base, double w){
    this.base = base;
    width = w;
  }

  public double getEnd() { return base+width; }

  //other methods
}
```

```
public class ManyIntervals{

  public static void main(String[] args) {

    int n= 4; //number of Intervals to create
    int H= 5; //highest value for base, range
    int L= 1; //lowest value for base, range

    //Set of Intervals
    Interval[] set=









    //Find Interval with highest endpoint









    System.out.println("Interval with highest endpoint: " +         );


    //Find 1st Interval with endpoint 6
    int target= 6;









  } //method main

} //class ManyIntervals
```