Structured Data in Java
Additional Array Information

Array Initialization

```
type[] array_name = {value, value, … value};
```

– Example:

```
int[] numbers = {32, 17, 3, -21, 6, 77, 35, -10};
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
value	32	17	3	-21	6	77	35	-10

– Useful when you know the array values at compile-time
  - Example: An array to hold the number of days in each month, or an array to hold the names of the days of the week

– You don’t specify the size of the array, but rather the compiler figures it out by counting the values

Arrays as Parameters

- Declaration (similar to declaring an array variable):

```
public static type methodName(type[] name) {
```

  – Example:

```
public static double average(int[] arr) {
```

- Call:

```
methodName(arrayName);
```

  – Example:

```
int[] score = {97, 76, 82, 85, 91};
double ave = average(score);
```

Note: there is no use of square brackets [ ] at the call site

Array Parameter Example

```
public void process() {
    int[] score = {97, 76, 82, 85, 91};
    double ave = average(score);
    out.println("Average is: " + ave);
}

public static double average(int[] arr) {
    int total = 0;
    for (int i=0; i<arr.length; i++) {
        total = total + arr[i];
    }
    return (double) total / arr.length;
}
```

Methods that Return Arrays

- A Java method may return an array
- Specify an array as the return type:

`public static` *`type`*`[]` *methodName*`(`*parameters*`) {...}`

- To return the array value
  - Declare a local array or use an array parameter
  - Use that array identifier in the `return` statement
- Must assign the returned array to an appropriate array variable

Methods that Return Arrays

- Example:

```
//Returns a new array that has been mirrored.
//Example: [3, 8, 10, 4]=>[3, 8, 10, 4, 4, 10, 8, 3]
public static int[] mirror(int[] orig) {
    int[] tmp = new int[2*orig.length];
    for (int i=0; i<orig.length; i++) {
        tmp[i] = orig[i];
        tmp[tmp.length-i-1] = orig[i];
    }
    return tmp;
}
```

Using Methods that Return Arrays

```
type[] name = methodName(parameters);
```

- Example:

```
public void process() {
    int[] arr = {3, 8, 10, 4};
    int[] reflection = mirror(arr);
    ...;
}
```

- Note: no need to initialize `reflection` with a `'new'` operation
  - The `'new'` operation is done in the `mirror` method

The `Arrays` class

- The `Arrays` class has some static methods for manipulating arrays:

Method name	Description
`toString(`**arr**`)`	returns a string representing the array inside [ ], e.g. `"[7, 33, 51, 14]"`
`equals(`**arr1**`, `**arr2**`)`	returns `true` if the two arrays are equal, that is they contain the same elements in the same order
`fill(`**arr**`, `**val**`)`	sets every element in the array to have the specified value
`sort(`**arr**`)`	sorts the elements in the array into ascending order
`binarySearch(`**arr**`, `**val**`)`	returns the index of the given value in an array (< 0 if not found); the array must be sorted

- Must `import java.util.*;` to access the class

The `Arrays` class: Example

- Consider the problem of finding the median of a set of values
- Copy the array so that we do not modify the original
- Sort the copy
- Report the median value
  - Middle value of an odd length array
  - Average of two middle values of an even length array

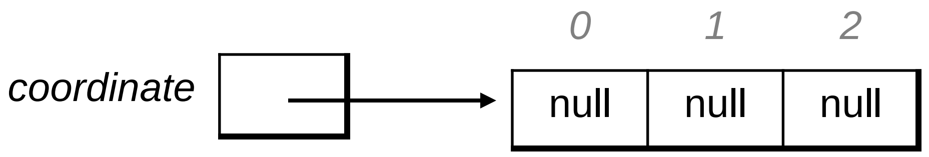
The `Arrays` class: Example

```
//Return the median value of an array of numbers
//without changing the parameter array
public static double median(int[] numbers) {
   int[] tmp = Arrays.copyOf(numbers, numbers.length);
   Arrays.sort(tmp);
   int mid = tmp.length/2;    // Note: int division
   if  (tmp.length%2 == 0) { // even length?
      return (tmp[mid-1]+tmp[mid])/2.0; //float division
   } else {
      return tmp[mid];
   }
}
```

Arrays of objects

- Recall: when you construct an array of primitive values like `int`s, the elements' values are all initialized to 0
- The elements of an array of objects are initialized to store a special *reference value* called `null`
  - **`null`** **:** A reference that does not refer to any object

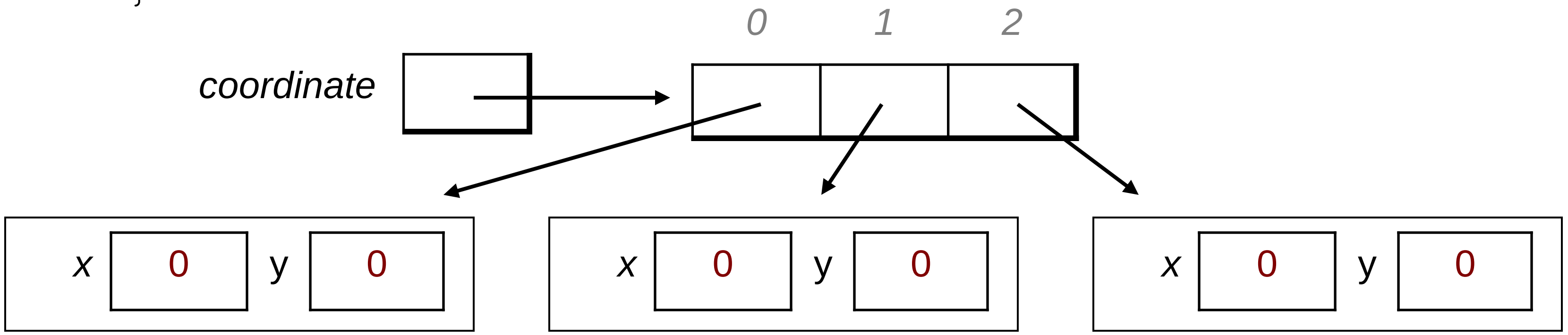
```
Point[] coordinate = new Point[3];
```



Two-step initialization

- Arrays of objects require a *two-step initialization*:
  1) create the array, where each element is initially `null`
  2) create a new object for each element of the array

```
Point[] coordinate = new Point[3];                // step 1
for (int i=0; i<coordinate.length; i++) {
    coordinate[i] = new Point(0, 0);              // step 2
}
```



Multi-Dimensional Arrays

- So far, we’ve only used one-dimensional arrays
- Java also allows multi-dimensional arrays
- Each pair of square brackets indicates another dimension of the array

```
int[][][] box = new int[3][5][2];
```