Arrays

- Aggregate data type
- Deal with items of same type
  - Lists of words
  - Numbers
- Analogies
  - Mailboxes in post office
  - CD racks with slots
- Simplifies naming
  - What if you needed to come up with unique name for each data item?
- Allows use of loops
- Required for many mathematical and statistical problems
- Multiple elements or cells

Using arrays

- Use subscript or index to access an element
  
  ```java
  x[5] = 20;
  System.out.println("Result is " + x[5]);
  ```
- First element is element 0, not 1!!!
- Often used in loops
  
  ```java
  int k = 0, sum = 0;
  while (k < 10)
  { 
    sum = sum + measurements[k];
    k = k + 1;
  }
  ```
- Note that subscript is an int variable, k

Creating Arrays

- Declaration
  ```java
  double weights[];
  ```
- Definition
  ```java
  weights = new double[50];
  ```
- Combine declaration and definition
  ```java
  double weights[] = new double[50];
  ```

```java
int num[] = new int[6];
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
</tbody>
</table>

Arrays & Loops

```java
num[0] = 0;
int k = 2;
while (k < num.length)
{
  num[k]
  k = k + 1;
}
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>21</th>
<th>4</th>
<th>9</th>
<th>16</th>
<th>25</th>
</tr>
</thead>
</table>

- Subscript range errors!!!
  - Java checks (many languages do not)
  - Costs & tradeoffs
Array Examples

- Sum up elements in a list (4 ways to do same thing)
  ```java
  int k = 0, sum = 0;
  while (k < 10) {
    sum = sum + data[k];
    k = k + 1;
  }
  int k = 1, sum = 0;
  while (k <= 10) {
    sum = sum + data[k - 1];
    k = k + 1;
  }
  ```
- Count occurrences of something
- Search for something
- Information retrieval

Array Example (another)

- Example: method to count number of A grades
  ```java
  public static int getAs(int[] grades) {
    int aCount = 0;
    for (int k = 0; k < grades.length; k++) {
      if (grades[k] >= 90)
        aCount++;
    }
    return aCount;
  }
  ```

Java Basics – Arrays

- Array Bounds
  - Index must in range
    - `0 <= k < grades.length`
  - Else get `ArrayIndexOutOfBoundsException`
- Short Circuit Evaluation
  - Allows expression of the form
    ```java
    if (k>=0 && k<grades.length && grades[k] >= 90) {
      aCount++;
    }
    ```
  - Safe from exception
  - **Order important!**
  - Holds for any chain of ANDed terms
  - Similar rules for chain of ORed terms

Java Basics – Arrays

- Creating Arrays (by example)
  ```java
  int[] counts = new int[101];
  String[] colors = new String[3];
  Counter[] tallies = new Counter[10];
  int[] codes;
  codes = new int[17];
  ```
- Using Initializer Lists
  ```java
  String[] colors = {"red", "green", "blue"};
  double[] shims = {1.0, 1.1, 1.3, 1.6, 1.9, 2.5};
  ```
- Initializing Object Arrays
  ```java
  for (int k = 0; k < tallies.length; k++) {
    tallies[k] = new Counter();
  }
  ```
Java Basics – Arrays

- Arrays are Objects
  - Behavior of arrays similar to other objects
  - Thus `grades.length` works

- Assignments (Warning!)
  - Since array identifiers are just references
    - Array assignment doesn't create new array!
  - Use `newArrayname = arrayname.clone();`
    - This works well for arrays of primitives
  - What happens for arrays of objects?

ArrayList

- Better to use than an array (very often)
- ArrayList
  - Can grow and shrink
  - Has methods for common tasks (see API)
  - Only holds objects
- Can’t have an ArrayList of int or double
  - There is a special Integer (an int that is an object) and Double (note the capital letters!) class

ArrayList (cont)

- Create an ArrayList
  `ArrayList<Integer> idlist = new ArrayList<Integer>();`
- Add an element to the ArrayList
  `idlist.add(8);`
- Modify kth element in an ArrayList
  `idlist.set(k, 9);`
- Sum the elements in the ArrayList
  ```
  // sum up integers in the ArrayList
  int sum = 0;
  for (Integer current : idlist) {
    sum += current;
  }
  ```

Convenient to use ArrayList, not array

- If you are given an array as a parameter
  - Copy values to an ArrayList
  - Then can work with the ArrayList
- If you need to return an array
  - Copy values from ArrayList to an array
- For Example, you’ll need to do both of these for APTs that use arrays.
Example: singleNumbers

- Given an integer array that could have duplicates, return an array that has only unique numbers from the original array.
- For example if the parameter array is: 5 5 8 5
- Then the array to return should be: 5 8

First convert array to ArrayList

```java
public int[] singleNumbers(int[] ids){
    // convert the array ids into an ArrayList idlist
    ArrayList<Integer> idlist = new ArrayList<Integer>();
    for (int k = 0; k < ids.length; k++) {
        idlist.add(ids[k]);
    }
    // Alternatively, loop can be written as
    for (int id : ids) {
        idlist.add(id);
    }
    
    Second, find unique numbers

    // create an ArrayList that will hold unique numbers
    ArrayList<Integer> singles = new ArrayList<Integer>();
    singles.add(idlist.get(0)); // first number is unique
    for (Integer current : idlist) {
        boolean isIn = false;
        for (Integer currentSingle : singles) {
            if (current.equals(currentSingle))
                isIn = true;
        }
        if (!isIn)
            singles.add(current);
    }
    
    Third, convert ArrayList to Array

    // convert ArrayList to array
    int[] answer = new int[singles.size()];
    int position = 0;
    for (Integer currentSingle : singles) {
        answer[position] = currentSingle;
        position++;
    }
    return answer;
```