Applications of Arrays

Some Sample Array Declarations

```
// array of 10 student grades (integers)
int grades[];
grades = new int[10];

// average temperatures for each month
double avgTemp[];
avgTemp = new double[12];

// e-mail list for 100 members
String mailList[] = new String[100];
```

Frequency Table

A common programming problem involves tracking:

(a) a limited number of options, with
(b) unknown, or unlimited, repetitions

For example, a "fair die" is a die which has equal probabilities of rolling any face (1 to 6). If simulating a fair die using a random number generator, how can we test this?

Frequency Table

With 6 possible rolls (1 to 6), we require an array with a length of 6. Each element will be used to count, or accumulate, the total rolls of each side.

```
int[] dieRolls = new int[6];
```

or

```
int[] dieRolls = {0, 0, 0, 0, 0, 0};
```

Note: By default, Java sets all integer values to zero. Good coding practice, however, would include code to initialize all starting values.

Frequency Table

```
int numRolls, roll;
int[] dieRolls = new int[6];

// how many test rolls?
numRolls = In.getInt();

for (int i = 0; i < numRolls; i++)
{
   // simulate 0-5, since arrays start
   // at zero, not one
   roll = (int)(6 * Math.random());
   dieRolls[roll] = dieRolls[roll] + 1;
}
```

Linking Data Across Arrays

One of the limitations of an array is the fact that it may only contain a single data type (e.g., integers or Strings).

Many applications of data involve the mixing of data types in a useful way. For example, student data may include a student ID (int), multiple strings (names, address), and grades (double).

Yet for possibly hundreds or thousands of students, arrays make sense.

Linking Data Across Arrays

For large sets of data, arrays make sense. When multiple data types are required, it is possible to use multiple arrays and link the data using the index of the array.

This requires each array to be the same length. In addition, the contents of each arrays must align according to the index. All index '0' data must align, all index '1' data, etc.

Linking Data Across Arrays

For example, consider the months of the year and the number of days in each. While there are other ways to track this, two linked arrays will work.

```
String[] monthName = {"Jan", "Feb", "Mar", ...};
int[] monthDays = {31, 28, 31, ...};
```

To use these arrays, we could write a program which prompts the user for a number of days and then outputs all months matching that number.

Linking Data Across Arrays

```
String[] monthName = {"Jan", "Feb", "Mar", ...};
int[] monthDays = {31, 28, 31, ...};

System.out.print("How many days? ");
int numDays = In.getInt();

for (int i = 0; i < monthName.length; i++)
{
   if (monthDays[i] == numDays)
   {
      System.out.println(monthName[i]);
   }
}
```

Comparing Arrays

Arrays are complex data structures, and that complexity means simple comparisons between arrays are not possible.

We have already seen an example of this complexity with strings. To compare a string, we are required to use a special command.

```
string1.equals(string2)
```

The equivalent command for arrays is:

```
Arrays.equal(array1, array2)
```

Comparing Arrays

While comparing arrays this way is easy, it also does not teach anything about array structures and programming.

You must be able to compare two arrays, element by element, by writing the code yourself.

For this course, any use of the "`Arrays.equals()`" command will be interpreted as you not knowing how to compare arrays, with appropriate deductions on assignments.