CSCE A201
Arrays

Say you want to keep an inventory of items. Based on what we have covered so far, you have to create a separate variable for each one:

```
int numItem1;
int numItem2;
int numItem3;
```

etc.

This works fine, but what if you had hundreds of items? It would be too much work to explicitly declare each one. It would be nice if we could programmatically access each separate variable. The construct that allows us to do this is called an **array**. We will first examine one-dimensional arrays, and then proceed to multi-dimensional arrays.

An array is a consecutive group of memory locations that all have the same name and the same type. To refer to a particular location, we specify the name and then the positive index into the array. The index is specified by square brackets, []. The first index is 0. The format for defining an array is:

```
Type[] varName = new Type[Size];
```

For example, let's say we define an array of type **byte** of size six:

```
byte[] a = new byte[6];
```

This allocates in the computer 6 bytes for the array: These bytes are stored somewhere in memory, let's say they are stored at memory address X, where let's ay X is at 1000:

Memory Address	**Array Index**	**Contents**
X, e.g. 1000	a[0]	0
X+1, e.g. 1001	a[1]	0
X+2, e.g. 1002	a[2]	0
X+3, e.g. 1003	a[3]	0
X+4, e.g. 1004	a[4]	0
X+5, e.g. 1005	a[5]	0

Initially the contents of the array variables are set to zero.

How does the computer know how much memory to allocate to each array element? It allocates enough memory to hold the size of the data type the array is defined for. Since Java allocates 4 bytes for an integer, and we define our array as:

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

This results in memory allocation:

Memory Address	Array Index	Contents
X, e.g. 1000	a[0]	0
X+4, e.g. 1004	a[1]	0
X+8, e.g. 1008	a[2]	0
X+12, e.g. 1012	a[3]	0
X+16, e.g. 1016	a[4]	0
X+20, e.g. 1020	a[5]	0

We can refer to the array like we have defined six variables. a[0] refers to the first variable in the array. Note that if we try to access a[6] in this case, then we will be trying to access memory location X+24. This will be exceeding the bounds of the array, and will generate a Java runtime error (programming languages that aren't interpreted generally don't catch this error, resulting in bugs where we may be accessing memory we aren't supposed to!).

Here are some simple ways we can access the array just like it was a variable:

```
a[3] = 54;                    // Stores 54 into array element 3
a[0] = a[3];                  // Copies 54 into array element 0
a[5] = a[2+1];                // Copies contents of a[3] to a[5]
i=5;
a[i]++;                       // Increment value in a[5]

for (i=0;i<6; i++)
      System.out.println(a[i]);     // Print each array element
```

Note that if we use:

```
System.out.println(a);
```

where a is an array, we do NOT print out each element. Instead the computer will print out the memory address where the array is stored. This is probably not what you want; to print out all array elements, we need to individually print each one in a loop. An array is actually a class object, and class objects are stored as a reference to the memory address where the object is located.

The flexible thing here is we can programmatically access each variable, based on the index, instead of hard-coding the access by hand.
To initialize an array, we could use a loop as above:

```
int[] a = new int[6];
int j;
for (j=0; j<6; j++)
     a[j]=99;
```

We can also use initializers for arrays. To define an array and initialize it at the same time, we can use curly braces { } :

```
int[] a = {1, 2 , 3, 4, 5, 6};
```

This sets a[0] to 1, a[1] to 2, a[2] to 3, a[3] to 4, a[4] to 5, and a[5] to 6. Since we are initializing the array with six elements, the compiler knows to make the size of the array six.

An equivalent way to initialize the same array is to use:

```
int[] a = new int[6];
a[0]=1; a[1]=2; a[2]=3; a[3]=4; a[4]=5; a[5]=6;
```

A good programming technique when using arrays is to define some constant for the array size. This has the benefit that if you ever need to change the size of the array, you can do it in one place. If you hardcoded a number like "6" into the program, then there may be many places you need to change:

```
public static final  int ARRAYSIZE = 6;
…
int[] a = new int[ARRAYSIZE], j;
for (j=0; j<ARRAYSIZE; j++)
{
     a[j]=j;
}
```

Let's look at a few example programs that use arrays. The first one inputs 10 numbers from the user and then calculates the average:

```
private static final int ARRAYSIZE = 10;
public static void main(String[] args)
{
      int[] n = new int[ARRAYSIZE];
      int i=0, total=0;
      Scanner keyboard = new Scanner(System.in);

      for (i=0; i<ARRAYSIZE; i++)
      {
            System.out.println("Enter a number. ");
            n[i] = keyboard.nextInt();
      }
      for (i=0; i<ARRAYSIZE; i++ )
      {
            total += n[i];
      }
      System.out.println("The average is : " + total/ARRAYSIZE);
}
```

In this example, we loop over the array twice. Once to input each value, and another time to generate the average. Note that we could compute the average while also entering the numbers, if we wished. What would happen if the first for loop had "<=" instead of "<" ?

Arrays Are Objects

An array is a type of object, and like other objects it has methods and instance variables. Some of these methods include "equals()", "clone()", and "toString()" which we will discuss later. For now, a useful variable is "length" which is equal to the size of the array. For example, in the previous case, we could use the following:

```
for (i=0; i < n.length; i++)
{
      . . .
}
```

Also, since arrays are objects, this means we can't use == to compare two arrays to see if they are equal. Instead, this will compare their memory addresses instead of the contents of the array.

To see if two arrays are equal, you must use something that compares every element, e.g.:

```
// First, assume the arrays are equal
if (a.length != b.length)
{
      // not equal
}
for (i=0; i < a.length; i++)
{
      if (a[i]!=b[i])
      {
            // not equal
      }
}
```

If you import `java.util.Arrays` then you may use `Arrays.equals(arr1,arr2)` to see if two arrays are identical (there is also a deepequals which determines if sub-objects in the arrays are also equal).

Also, we can't use the assignment operator to set one array equal to another if we wish to copy the contents of one array to another. The assignment:

```
b = a;
```

Will instead set variable b to the same memory address as a. For example, consider the following code snippet:

```
int[] a = new int[3];
int[] b = new int[3];

a[0]=1; a[1]=1; a[2]=1;
b[0]=0; b[1]=0; b[2]=0;
System.out.println(a[0] + " " + b[0]);    // 1 0
b = a;
System.out.println(a[0] + " " + b[0]);    // 1 1
a[0]=9;
System.out.println(a[0] + " " + b[0]);    // 9 9
```

The first two print statements behave as expected. However, the last print statement outputs that b[0] is equal to 9! This is because in setting "b=a" we are setting b to point to the same memory address where a is stored. Then, when we set "a[0]=9", when we dereference array b, we get back the new value for a.

To copy one array to another we should copy each element individually, e.g.:

```
for (i=0; i < b.length; i++)
{
     a[i]=b[i];
}
```

The clone( ) method can also be used to make an identical copy of an array, to get the same effect more easily:

```
b  = a.clone();
```

Array b is now set to a copy of array s.

Two-Dimensional Arrays

A two-dimensional array is a collection of data of the same type that is structured in two dimensions. Individual variables are accessed by their position within each dimension. You can think of a 2-D array as a table of a particular data type. The following example creates a 2-D array of type int:

```
int[][]  twoDimAry = new int[5][10];
```

twoDimAry is an array variable that has 5 rows and 10 columns. Each row and column entry is of type int. The following code fragment sets all the entries in twoDimAry to 0:

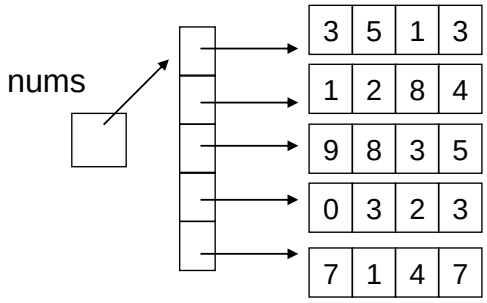
```
for (int column = 0; column < 10; column++)
    for(int row = 0; row < 5; row++)
        twoDimAry[row][column] = 0;
```

Processing a two-dimensional array variable requires two loops: one for the rows and one for the columns. If the outer loop is the index for the column, the array is processed by

column. If the outer loop is the index for the row, the array is processed by row. The preceding loop processes twoDimAry by columns.

Internally, Java really stores 2D arrays as an array of arrays. For example:

```
int [][] nums = new int[5][4];
```



This organization can be a bit inefficient, but allows for "ragged" arrays where the size of the second dimension varies from row to row.

Multidimensional Arrays

You have seen one-dimensional and two-dimensional arrays. In Java, arrays may have any number of dimensions. To process every item in a one-dimensional array, you need one loop. To process every item in a two-dimensional array, you need two loops. The pattern continues to any number of dimensions. To process every item in an n-dimensional array, you need n loops.

For example, if we wanted to declare an array of 3 dimensions, each with 10 elements, we could do so via;

```
int[][][] three_d_array = new int[10][10][10];
```

Array Exercise:

Write a program that inputs 10 integer, numeric grades into an array and finds the average and the max.

Array Example – Trivia Game

Let's write a program to play a simple trivia game. First we need a database of trivia questions. For simplicity, let's make our game have five questions. Each question has a correct answer and a point value. In the game we'll ask each question to the player and calculate the total score.

Our strategy will be to create three arrays, each with five elements. The first array will hold the questions, the second array will hold the answers, and the third array will hold the value. The arrays will be tied together by a common index. The structure looks like this:

Index = 0 Index = 1 Index = 2 Index = 3 Index = 4

Questions Array

Question 1	Question 2	Question 3	Question 4	Question 5

Answers Array

Answer 1	Answer 2	Answer 3	Answer 4	Answer 5

Values Array

Value 1	Value 2	Value 3	Value 4	Value 5

For example, question 3, the answer to question 3, and the point value for question 3 are all stored at index 2 in the respective arrays.

We can hard code these into three separate arrays. Later we'll see how to read them from a file.

```
import java.util.Scanner;

public class TriviaGame
{
	// Preview of upcoming concept,
	// this is a constant that is accessible
	// in the whole class
	private static int NUMQUESTIONS = 5;

	public static void main(String[] args)
	{
		// Arrays to hold the questions and answers
		String[] questions = new String[NUMQUESTIONS];
		String[] answers = new String[NUMQUESTIONS];

		// You can also declare the array as empty
		// then set it to new later in the code
		int[] values;
		values = new int[NUMQUESTIONS];

		// Manually copy in questions, answers, and values
```

```
            questions[0] = "The first Pokemon that Ash receives from" +
                                  " Professor Oak";
            answers[0] = "pikachu";
            values[0] = 1;
            questions[1] = "This Late Night Talk Show Host studied" +
                                    " CS at Albany College";
            answers[1] = "Jimmy Fallon";
            values[1] = 2;
            questions[2] = "This supermodel has a koding camp"+
                                    " for girls";
            answers[2] = "Karlie Kloss";
            values[2] = 2;
            questions[3] = "A mathematician's lost parrot";
            answers[3] = "polygon";
            values[3] = 3;
            questions[4] = "PT Barnum said 'This way to the _______'"+
                                    " to attract people to the exit.";
            answers[4] = "egress";
            values[4] = 1;

            // Print some data as a test to check the arrays
            System.out.println(questions[1] + " " +
                        answers[1] + " " + values[1]);
            System.out.println(questions[4] + " " +
                        answers[4] + " " + values[4]);
      }
}
```

For a real game we would probably have a lot more questions!

The game will simply ask all questions, outputs if the player is correct or not, and then output the total score. To keep track of these we can add some new variables:

```
      int score = 0;                // Overall score
      int questionNum = 0;          // Track if question 0 - 4
```

score is an integer that tracks our score. questionNum keeps track of which question we are asking, and will start at 0 and go up to 4, the last question.

We can start by writing a loop that asks each question and inputs an answer:

```
      // Ask each question
      Scanner keyboard = new Scanner(System.in);
      for (questionNum = 0; questionNum < questions.length;
           questionNum++)
      {
            System.out.println("Question " + (questionNum+1) + ": " +
                  questions[questionNum]);
            System.out.println("Your answer?");
            String answer = keyboard.nextLine();
      }
```

At this point we can test the program to make sure it is iterating through all of the questions correctly. Next we can check if the answer matches the real answer. If so, we add points to the score. Here is the final total program:

```
import java.util.Scanner;

public class TriviaGame
{
      // Preview of upcoming concept,
      // this is a constant that is accessible
      // in the whole class
      private static int NUMQUESTIONS = 5;

      public static void main(String[] args)
      {
       // Arrays to hold the questions and answers
       String[] questions = new String[NUMQUESTIONS];
       String[] answers = new String[NUMQUESTIONS];

       // You can also declare the array as empty
       // then set it to new later in the code
       int[] values;
       values = new int[NUMQUESTIONS];

       // Score and question
       int score = 0;
       int questionNum = 0;

       // Manually copy in questions, answers, and values
       questions[0] = "The first Pokemon that Ash receives from" +
                           " Professor Oak";
       answers[0] = "pikachu";
       values[0] = 1;
       questions[1] = "This Late Night Talk Show Host studied" +
                             " CS at Albany College";
       answers[1] = "Jimmy Fallon";
       values[1] = 2;
       questions[2] = "This supermodel has a koding camp"+
                             " for girls";
       answers[2] = "Karlie Kloss";
       values[2] = 2;
       questions[3] = "A mathematician's lost parrot";
       answers[3] = "polygon";
       values[3] = 3;
       questions[4] = "PT Barnum said 'This way to the _______'"+
                                  " to attract people to the exit.";
       answers[4] = "egress";
       values[4] = 1;

       // Ask each question
       Scanner keyboard = new Scanner(System.in);
       for (questionNum = 0; questionNum < questions.length;
           questionNum++)
       {
            System.out.println("Question " + (questionNum+1) + ": " +
```

```
                questions[questionNum]);
            System.out.println("Your answer?");
            String answer = keyboard.nextLine();

            // Check if answer is correct; if so, add to the score
            if (answer.equalsIgnoreCase(answers[questionNum]))
            {
                  score+=values[questionNum];
                  System.out.println("That's right! Your new score is "
                      + score);
            }
            else
            {
                  System.out.println("Sorry, the correct answer is " +
                              answers[questionNum]);
            }
       }
       // Output the final score
       System.out.println("The game is over! Your final score is " +
                           score);
      }
}
```