### **Arrays**

* Used to represent large collections of data
* Once created, arrays are of fixed size
* Can refer to elements of an array with indexing

### **Declaring Array Variables**

Array variable declarations are of the following form:

<data type>[] <array variable>;

for example

int[] numbers;

is an array of int's.

Note that declaring arrays does not allocate space for the array.

### **Creating Arrays**

Arrays are created and assigned to array variables as follows:

<array variable> = new <data type>[<size>];

For example

numbers = new int[10];

creates an array of ten int's.

Note that you can not change the size of an array after creation.

Array can be declared and created together as follows

int[] numbers = new int[10];

### **Initializing Arrays**

Arrays can be initialized by setting the elements as

numbers[0] = 2;

numbers[1] = 4;

numbers[2] = 6;

numbers[3] = 8;

numbers[4] = 10;

numbers[5] = 12;

numbers[6] = 14;

numbers[7] = 16;

numbers[8] = 18;

numbers[9] = 20;

### **Array Length and Default Values**

The following gives the size of the array

numbers.length

When first created, arrays contain default values, e.g. 0 for numbers, false for booleans, etc.

### **Indexed Variables**

An element of an array can be referred to as

<array variable>[<index>]

For example

numbers[3]

is the fourth element in the array numbers.

This is also referred to as an **indexed variable**.

Examples of use:

numbers[2] = numbers[0] + numbers[1];

or

for (int i = 0; i < numbers.length; i++) numbers[i] = 2\*(i+1);

### **Array Initializers**

Arrays can be also be declared, created and initialized in one statement as follows:

int[] numbers = {1, 5, 3, 6};

This is equivalent to:

int[] numbers = new int[4];

numbers[0] = 1;

numbers[1] = 5;

numbers[2] = 3;

numbers[3] = 6;

With this approach, declaration and initialization must be done in one statement.

### **Processing Arrays**

What is the best loop construct to use with arrays? While? Do While? For?

For loops are ideal for arrays as

* all elements are of the same data type
* the array is of a fixed length

Examples:

Total of all numbers

int total = 0;

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

total += numbers[i];

Maximum of all numbers

int max = numbers[0];

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

if (numbers[i] > max) max = numbers[i];

How would you find the index of the smaller element?

### **Foreach Loop**

There is special for construct available for stepping over each element of an array.

This is referred to as foreach, although you will still use the keyword "for".

The following prints each element of the array

for (int e : numbers)

System.out.println(e);

### **Copying Arrays**

Array assignments operate on references.

So, two arrays numbers1 and numbers2 can be in an assignment as follows:

int[] numbers1 = {1, 2, 3, 4};

int[] numbers2 = {11, 22, 33, 44};

numbers1 = numbers2;

But now both variable numbers1 and numbers2 refer to memory holding {11, 22, 33, 44}

As no variables are now pointing to the memory {1, 2, 3, 4}, it will be reclaimed (garbage collected) by Java.

There are two ways to copy arrays:

Loop and make a copy of the elements

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

numbers1[i] = numbers2[i];

Use System.arraycopy

System.arraycopy(numbers2, // source

0, // source start index

numbers1, // target

0, // target start index

numbers1.length); // length

### **Passing Arrays to Methods**

* Primitive data types (like int, double, boolean, etc.) are passed by value.
* When arrays are passed as arguments, a reference to the array is passed into the method.
* If you change the contents of the array in the method, the change will be visible outside as well
* Similarly, when a method returns an array, the reference to the array is returned.

### **Command Line Arguments**

We have seen String arrays in the main method:

public static void main(String[] args)

So when you call the program as

java MyProgram -i 56

In main, args[0] is "-i" and args[1] is "56".