|  |
| --- |
| **Week 2 – Arrays and ArrayLists** |
| This week’s topics:   * Arrays (fixed-size data structure) * ArrayLists (expandable data structure)   **Arrays**  An array is a group of variables (called elements) containing values that all have the same data type. Specific elements in the array are accessed by their index, beginning at zero.  **Creating an Array**  Size (how many elements we want)  Square brackets  Data Type  Name  **int**[] numbers = **new int**[10];  The above statement would create the array below. Each element in the array receives a default value (**0** for numeric, **false** for Booleans, **null** for Strings – we explore this later in the term).   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | | **Value** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   This array has 10 elements, accessible through indices 0-9. To “go into” a specific index and access its value, we provide the name of the array, and the index number in square brackets. E.g.:  // assign 12 to the 1st element (position/index 0)  numbers[0] = 12;  If we know the values we are going to populate an array with beforehand, we can create an array and immediately populate it with values (written between curly braces { } and separated by commas) using an **array initialiser**. E.g:  String[] daysOfTheWeek = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};  **Array Length**  Every array knows how many elements are stored within it through an associated length variable. This can be accessed using the syntax *arrayName*.length. e.g.  System.***out***.println(daysOfTheWeek.length); // outputs 7  **Array Bounds**  Every array has a bounds (indices we are able to access). This begins with the lower bound (index 0) and the upper bound (array.length – 1). Attempting to access an index that does not exist causes an exception (we can think of this as just an error for now) to occur. E.g. attempting to access the value at index 10 within the numbers array we previously created:  numbers[10] = 4;    Although the numbers array we created had 10 elements, these were accessible from indices 0-9, meaning numbers[10] is out-of-bounds.  **Iterating through** **Arrays**  Arrays can be iterated (looped) through, this can be useful if we want to perform some task involving the contents of an array (such as print each value in turn, or perhaps perform some calculation on each element).  // loop through the daysOfTheWeek array, printing each value along the way  **for** (**int** i = 0; i < daysOfTheWeek.length; i++) {  System.***out***.println(daysOfTheWeek[i]);  }  We can also loop backwards by starting the for loop counter off at the *arrayName.length* – 1 (referring to the last index of the array).  // loop through the array backwards, print each value  **for** (**int** i = daysOfTheWeek.length-1; i >=0; i++) {  System.***out***.println(daysOfTheWeek[i]);  }  **ArrayList**  Since arrays are fixed-size data structures, they cannot grow or shrink at runtime if you suddenly decide you need more space. Java provides a collection which can automatically grow/shrink during runtime, called the **ArrayList**.  An ArrayList can only hold objects (not primitives such as int, float). If we do want to store primitives, we can get around this by using their associated Wrapper classes (each primitive has a wrapper class). For example, if we wanted to create an ArrayLists of ints, we can do so like this:  ArrayList<Integer> numbers = **new** ArrayList<>();    We need to add an import statement to the top of our code in order to use the ArrayList class:  **import** java.util.ArrayList;  **ArrayList Methods**  There are many methods of the ArrayList class (meaning each ArrayList object also has them).   |  |  | | --- | --- | | **Method** | **Description / Example** | | add( item ) | Adds the **item** the the array list.  e.g. numbers.add(56); | | add( index, item ) | Add the **item** at the specific **index** | | remove ( index ) | Removes the **item**at the specific index | | set (index, item) | Updates the slot at **index** with new value **item** | | get ( index ) | Returns the itemat an **index** | | indexOf ( item ) | Returns the index of the first occurrence of **item** in the list (returns **-1** if not found) |   **For-each (Enhanced for) Loop**  An alternative for loop (called the for-each loop) is available in Java which makes it convenient to loop through a collection – whether the collection is an array or an ArrayList.  The for-each loop does not declare a counter variable, and is only capable of looping through a collection from the start, one item at a time. This can help to avoid potential ArrayIndexOutOfBound exceptions from occurring.  The syntax for the for-each loop is:   |  |  | | --- | --- | | **Syntax** | **for** (**type** variable : collection) {  statements  } | | **Example**  **(using numbers array)** | **for** (**int** number : numbers) {  sum = sum + number;  } |   For examples of how to use each of these loops with a class type, see next page. |

|  |  |
| --- | --- |
| Assuming a class called Dog (right), we can create multiple objects of this type and add these objects to an ArrayList. We can then loop through this ArrayList, fetching each dog using the **get** method, and do something with it (e.g. print out one of the object’s variables or call the object’s methods).  **import** java.util.ArrayList;  **public** **class** DogMain {  **public** **static** **void** main(String[] args) {    Dog dog1 = **new** Dog("Bert", 6);  Dog dog2 = **new** Dog("Charles", 3);  Dog dog3 = **new** Dog("Beethoven", 10);  ArrayList<Dog> dogs = **new** ArrayList<>();  dogs.add(dog1);  dogs.add(dog2);  dogs.add(dog3);    // traditional for loop  **for** (**int** i = 0; i < dogs.size(); i++) {  System.***out***.println("Dog name: " + dogs.get(i).name);  }  // enchanced for loop  // for every Dog in dogs arraylist - refer to the current dog as d  **for** (Dog d : dogs) {  System.***out***.println("Dog age: " + d.age);  }  }  } | **public** **class** Dog {  String name;  **int** age;    Dog(String name, int age) {  **this**.name = name;  **this**.age = age;  }    } |