Arrays and Strings

Objectives

The objectives of this practical session are to create and manipulate the following:

! Arrays of primitive variables

! Arrays of object references

Overview

There are four parts to this practical. In Part 1, you will create and use an array of integers. In Part 2, you will define a class called `Sort` with a class method to sort an array of integers into ascending order. In Part 3 you will create and display an array of `Person` objects. In Part 4 you will sort the `Person` objects by age and optionally by name.

Practical

Part 1. Creating and Using an Array of Primitives

1. Copy the provided sort package into your newly created IntelliJ project.

   The package `sort` contains the skeleton code of a class called `SortTest` with a `main()` method. The eventual objective of this class is to test the `Sort` class, but in this part of the practical you are simply going to use it to create and test an array of integers.

2. In the `main()` method, where indicated by the comments, create an array of ten elements of type `int`.

3. Use the class method `Math.random()` to fill the array with random values. Note that `Math.random()` returns a value between 0.0 and 1.0, so you'll need to multiply this number by some factor (e.g. 100) and cast the result to an `int`.

4. Display the initial contents of the array by amending the commented out `for` loop that uses `System.out.print()`. Save and execute ensuring a list of 10 unsorted random whole numbers in the range 0 to 99 appear.

Part 2. Sorting an Array of Primitives

1. Examine the class `Sort` and note that it contains the skeleton code of a class called `Sort` with a class method called `bSort()`. In this method, you'll need to write some code to sort the elements in the array into ascending order. As you may know, there are a number of different sorting algorithms that we

could use. While it is not very efficient with large arrays, the simplest one to understand is called the *bubble sort*. The bubble sort gets its name from the way large values tend to move towards the top of the array as multiple passes are made though the array. The number of passes required is equal to the number of elements in the array minus one.

To implement the bubble-sort algorithm requires two nested loops. Basically, you need to compare each element in the array with those above it. If the value of the element is greater than any of them, it must be swapped with the lesser one. Copy and paste sample code from `development\student\Arrays_and_Strings\sort\Temp.txt`. Incidentally, don't forget that you can find the size of an array in its `length` instance variable.

2. Save the project and fix any compiler errors.
3. In the `SortTest` class, after displaying the initial contents of the array, call the `bSort()` method of the `Sort` class to sort it. Then, amend the commented out `for` loop to display the new contents of the array.
4. Execute the `SortTest` class. It should now sort the array as intended.

   Congratulations! You have successfully created and manipulated arrays of primitives.

Part 3. Creating and Using an Array of Objects

1. Paste the folder `person` into your project.

   The package contains the skeleton code of a class called `Person` with a constructor, two instance methods, `getAge()` and `getName()`, and two *class* methods, `bSortByAge()` and `bSortByName()`.
2. Declare an instance variable of type `int` for the person's age and another variable of type `String` for the person's name.
3. Define a constructor to initialise both instance variables.
4. The package also contains the skeleton code of a class called `PersonTest`. In its `main()` method, declare a variable called 'persons' capable of referencing an array of `Person` objects. Then, create the array called 'persons' referencing as many `Person` objects as there are names in the supplied `testNames` String array.
5. Fill the array with `Person` objects. Each time you create a new `Person` object, you'll need to specify a name and age to the constructor. We have provided a selection of names and ages in the class variables `testNames & testAges`, and you may use these if you wish.
6. Display the initial contents of the array, i.e. the name and age of each `Person` object in two columns (you can use an appropriate number of tabs to separate the columns, but don't worry too much about the formatting). Don't forget that you'll need to use the `getName()` and `getAge()` methods of the `Person` class to retrieve a person's name and age, respectively.

7. Save the project and execute the `PersonTest` class. Ensure that it displays the names and ages correctly.

Part 4. Sorting an Array of Objects

1. In the `bSortByAge()` method of the `Person` class, use a bubble sort algorithm (similar to the one you wrote earlier) to sort the `Person` objects in the array into ascending order by age.
2. In the `PersonTest` class, after displaying the initial contents of the array, call your `bSortByAge()` method. Then, display the new contents of the array. (You could move the display code into a new class method called, say, `displayArray()`)
3. Save the project and execute the `Person` class. If it works, proceed to the next step.
4. Now it's time to sort the `Person` objects in the array by name. Instead of comparing the two `Person` objects age's you will compare their name's. Remember that class `String` has a very useful compareTo method that returns an `int`. See this chapters notes. In the `bSortByName()` method, simply copy the `bSortByAge()` method and change the `if` statement that does the comparison.
5. In the `PersonTest` class, replace the call to `bSortByAge()` by a call to `bSortByName()`.
6. Execute the `Person` class. If it works, take a break!

Optional

7. As an alternative to make your if statement in the `for` loop shorter write a `compareTo()` method in your `Person` class that performs a similar function to the `String` method `compareTo()`. This method should take a single argument, which is a reference to a `Person` object and return an `int` and only needs to be one line of code!
8. Change the if statement of the `for` loop to use your `Person` class `compareTo()` method.