Arrays

Part I

One Dimensional Arrays

1 Declaring arrays <=> creating arrays

Arrays are lists of variables of the same type stored contiguously in memory.

Declaration syntax:

```java
elementType [] arrayName;
```

This does not allocate memory for any items. It simply states that in the future the array may be created.

Creating array syntax:

```java
arrayName = new elementType[numberOfElements];
```

This allocates a block of memory large enough to store numberOfElements variables of type elementType.

Declaring and creating array in one step:

```java
elementType [] arrayName = new elementType [numberOfElements];
```

Example:

```java
int [] assignmentScores;
assignmentScores = new int [15];
```

Alternative way of declaring and creating arrays (all in one step):

```java
elementType [] arrayName = {comma-separated-list-of-elements};
```

Example:

```java
int [] assignmentScores = { 9, 10, 10, 0, 8, 7, 10, 10, 9, 9};
```

When array is created, all its elements are filled with zeros of the corresponding type.

The size of an array can be determined by using `arrayName.length`. This value cannot be changed once the array is created.
2 Accessing individual elements

Each element of an array can be accessed using the following syntax:

arrayName[index]

where index is \( 0 \leq index < \text{numberOfElements} \).

Examples:

```java
System.out.printf( "element at position %n is %f \n", i, array[i]);
array[17]= 26.116;
```

3 Processing arrays

**Initializing arrays** to random value, predefined value, value from a user, ...

```java
double [] numbers = new double [50];
for (int i = 0; i < numbers.length; i++){
    ...
    numbers[i] = value;
}
```

**Adding** all elements of an array

```java
double [] numbers = new double [50];
... //initialize the values in the array
double sum = 0.0;
for (int i = 0; i < numbers.length; i++){
    sum = sum + numbers[i];
}
```

**Finding smallest/largest**

Find the smallest value in the array of numbers and its first location (if there are multiple occurrences of the smallest value).

```java
double [] numbers = new double [50];
... //initialize the values in the array
double minValue = numbers[0];
int minIndex = 0;
for (int i = 1; i < numbers.length; i++){
    if (numbers[i] < minValue )
    {
        minValue = numbers[i];
        minIndex = i;
    }
```
Find the largest value in the array of numbers and its last location (if there are multiple occurrences of the largest value).

```java
double[] numbers = new double[50];
... // initialize the values in the array
double maxValue = numbers[0];
int maxIndex = 0;
for (int i = 1; i < numbers.length; i++){
    if (numbers[i] >= maxValue)
    {
        maxValue = numbers[i];
        maxIndex = i;
    }
}
```

4 Copying array references is not the same as copying the content of arrays

It is not possible to simply assign an array to another and get a copy. The following does not do what you might expect:

```java
int[] a1 = new double[50];
int[] a2 = a1;
```

After the above is executed, `a1` and `a2` contain references to the *same array*. Moreover, `a1` and `a2` refer to the same location in memory.

In order to copy the content of one array to another we need to copy them one element at a time. Their sizes should also match.

```java
int[] a1 = new double[50];
int[] a2 = new double[50];
for (int i = 0; i < a1.length; i++){
    a1[i] = a2[i];
}
```

5 Arrays and methods

As with ordinary variables arrays can be passed to methods as parameters and returned as a return value.

Passing arrays to methods

When you pass an array to a method, it is the reference variable that is passed. The significance of this is that the method cannot change what the name of the array points to, BUT it can modify the contents of the array itself.

Syntax:

```java
modifiers returnType methodName (arrayType [] arrayName ){
    method body
}
```
Example: What will be printed by the following code?

```java
public class TestArrayParameters {
    public static void main(String[] args) {
        int i; // loop counter variable
        double[] myNums = {1.1, 2.2, 3.3, 4.4, 5.5};

        for (i = 0; i < myNums.length; i++)
            System.out.print( myNums[i] + " , " );

        multiplyBy( myNumbers, 2.0 );

        for (i = 0; i < myNums.length; i++)
            System.out.print( myNums[i] + " , " );
    }

    public static void multiplyBy(double[] numbers, double multiplier) {
        for (int i = 0; i < numbers.length; i++) {
            numbers[i] *= multiplier;
        }
    }
}
```
Returning arrays from methods

When a method returns an array, the reference of the array is returned.

Syntax:

```java
modifiers returnType[] methodName ( list-of-parameters ){
    method body
}
```

Example:

```java
public class TestReturnedArray {

    public static void main (String [] args) {
        int [] myNumbers = createRandomIntArray(15);
        if (myNumbers.length != 15)
            System.err.println("Something went wrong!");
        for (int i = 0; i < myNumbers.length; i++)
            System.out.println(myNumbers[i]);
    }

    public static int [] createRandomIntArray(int size)
    {
        int [] randomNumbers = new int [size];
        for (int i = 0; i < randomNumbers.length; i++) {
            randomNumbers[i] = (int) (Math.random() *
                (Integer.MAX_VALUE - Integer.MIN_VALUE + 1))
                + Integer.MIN_VALUE;
        }
    }
}
```

6 Searching arrays

Linear search  Linear search algorithm searches a list of items (an array) for a specific item (a key). If a matching item is found, the return value indicates its location on the list (array index). If a matching item is not found, the return value should indicate failure, for example, an invalid index, such as -1. The linear search algorithm starts at the first item on the list and compares it to the key and then repeats these steps until the item is found or the end of the list is reached.

```java
public static int linearSearch (double [] list, double key)
{
    for (int i = 0; i < list.length; i++)
    {
        if (key == list[i])
            return i;
    }
    return -1;
}
```

If the array has \( n \) item:

- how many locations will be looked at in the worst case?
- how many locations will be looked at in the best case?
- how many locations will be looked at on average?

**Binary search**  If the list is stored in a sorted order, we can find the key, or determine that it is not in the list, in many fewer steps. Binary search algorithm searched a list of items for a specific key, but it discards a half of the remaining list at each step (this is only possible because of the assumption that the array is sorted).
public static int binarySearch ( double [] list, double key )
{
    int low = 0;
    int hight = list.length -1;
    while(high>=low) {
        int mid = (low + high)/2;
        if (key < list[mid])
            high=mid-1;
        else if (key == list[mid])
            return mid;
        else
            low=mid+1;
    }
    return -1;
}

If the array has n items:

- how many locations will be looked at in the worst case?
- how many locations will be looked at in the best case?
- how many locations will be looked at on average?
- how many operations does it take to sort the array first?

### 7 Sorting arrays

**Selection sort** Assume that you want to sort a list from smallest to largest. Selection sort is a sorting algorithm that starts by finding the smallest item on the list (remember the method of finding the min?) and swaps it with the first element of the list. Then it finds the smallest element in the remaining list (ignoring the first one) and swaps it with the second element on the list. It continues until the remaining unsorted part of the list is empty.

```java
public static void selectionSort(double[] list) {
    for (int i = 0; i < list.length - 1; i++) {
        // Find the minimum in the list[i..list.length-1]
        double currentMin = list[i];
        int currentMinIndex = i;
        for (int j = i + 1; j < list.length; j++) {
            if (currentMin > list[j]) {
                currentMin = list[j];
                currentMinIndex = j;
            }
        }
        // Swap list[i] with list[currentMinIndex] if necessary;
        if (currentMinIndex != i) {
            list[currentMinIndex] = list[i];
            list[i] = currentMin;
        }
    }
}
```

Notice that this algorithm does not introduce any new techniques. It simply repeats several time the find minimum element followed by swap of elements.
Insertion sort  Assume again that you want to sort a list from smallest to largest. Insertion sort algorithm repeatedly inserts an elements into a sorted sublist until the whole array is sorted.

```java
public static void insertionSort(double[] list) {
    for (int i = 1; i < list.length; i++) {
        /** insert list[i] into a sorted sublist list[0..i-1] so that list[0..i] is sorted. */
        double currentElement = list[i];
        int k;
        for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
            list[k + 1] = list[k];
        }
        // Insert the current element into list[k + 1]
        list[k + 1] = currentElement;
    }
}
```

How many operations does it take to sort an array?

Part II

Two-Dimensional Arrays

A two-dimensional array is an array of arrays.

8  Declaring and creating two-dimensional arrays

**Declaration** syntax:

```java
elementType [][] arrayName;
```

**Creating** array syntax:

1. each row of the same size

```java
arrayName = new elementType[numberOfRows][numberOfColumns];
```

2. ragged array

```java
arrayName = new elementType[numberOfRows][];
for (int i = 0; i < arrayName.length; i++) {
    arrayName[i] = new elementType[numberOfColsInRowI];
}
```

9  Accessing individual elements

Each element of an array can be accessed using its row and column index:

```java
arrayName[rowIndex][columnIndex];
```
10 Processing two-dimensional arrays

Nested loops (often for loops) are used for processing two-dimensional arrays.

**Initializing arrays** to random value, predefined value, value from a user, ...

```java
double [][] numbers = new double [50][150];
for (int row = 0; row < numbers.length; row++){
    for (int col = 0; col < numbers[row].length; col++){
        ...
        numbers[row][col] = value;
    }
}
```

**Printing values on a diagonal of a square two-dimensional array**

```java
double [][] numbers = new double [50][50];
... //initialize the values in the array
for (int row = 0; row < numbers.length; row++){
    //make sure the array is square
    if (numbers.length != numbers[row].length )
        System.err.println("Error: array is not square!");
    System.out.println( numbers[row][row]);
}
```

How would you:

- find smallest element in each row / each column?
- determine which row/column has the largest element in the array?
- sort each row/column of the array?