FINAL REPORT

Explanations

Java Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. To declare an array, define the variable type with square brackets:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, we can use an array literal - place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You access an array element by referring to the index number. This statement accesses the value of the first element in cars:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

Note: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Array Element

To change the value of a specific element, refer to the index number:

```
cars[0] = "Opel";
```

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Volvo
```

Array Length

To find out how many elements an array has, use the length property:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
// Outputs 4
```

Loop Through an Array

You can loop through the array elements with the for loop, and use the length property to specify how many times the loop should run. The following example outputs all elements in the cars array:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

Loop Through an Array with For-Each

There is also a "for-each" loop, which is used exclusively to loop through elements in arrays:

```
for (type variable : arrayname) {
  ...
}
```

The following example outputs all elements in the **cars** array, using a "for-each" loop:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

The example above can be read like this: for each String element (called i - as in index) in cars, print out the value of i. If you compare the for loop and for-each loop, you will see that the for-each method is easier to write, it does not require a counter (using the length property), and it is more readable.

Multidimensional Arrays

A multidimensional array is an array containing one or more arrays. To create a two-dimensional array, add each array within its own set of curly braces:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
```

myNumbers is now an array with two arrays as its elements. To access the elements of the myNumbers array, specify two indexes: one for the array, and one for the element inside that array. This example accesses the third element (2) in the second array (1) of myNumbers:

```
int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
int x = myNumbers[1][2];
System.out.println(x); // Outputs 7
```

We can also use a for loop inside another for loop to get the elements of a two-dimensional array (we still have to point to the two indexes):

```
public class MyClass {
  public static void main(String[] args) {
    int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };
    for (int i = 0; i < myNumbers.length; ++i) {
      for(int j = 0; j < myNumbers[i].length; ++j) {
        System.out.println(myNumbers[i][j]);
      }
    }
  }
}
```

Variable Arguments (Varargs)

In JDK 5, Java has included a feature that simplifies the creation of methods that need to take a variable number of arguments. This feature is called varargs and it is short-form for variable-length arguments. A method that takes a variable number of arguments is a varargs method. Prior to JDK 5, variable-length arguments could be handled two ways. One using overloaded method (one for each) and another put the arguments into an array, and then pass this array to the method. Both of them are potentially error-prone and require more code. The varargs feature offers a simpler, better option.

Java.util.Arrays.sort() Method

Description

The java.util.Arrays.sort(Object[] a, int fromIndex, int toIndex) method sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements. The range to be sorted extends from index fromIndex, inclusive, to index toIndex, exclusive.

Declaration

Following is the declaration for java.util.Arrays.sort() method

```
public static void sort(Object[] a, int fromIndex, int toIndex)
```

Parameters

- a − This is the array to be sorted.
- fromIndex − This is the index of the first element (inclusive) to be sorted.
- toIndex − This is the index of the last element (exclusive) to be sorted.

Return Value

This method does not return any value.

Exception

- IllegalArgumentException − if fromIndex > toIndex
- ArrayIndexOutOfBoundsException − if fromIndex < 0 or toIndex > a.length
- ClassCastException − if the array contains elements that are not mutually comparable (for example, strings and integers)

Java.util.Arrays.binarySearch() Method

Description

The java.util.Arrays.binarySearch(int[] a, int key) method searches the specified array of ints for the specified value using the binary search algorithm. The array must be sorted before making this call. If it is not sorted, the results are undefined.

Declaration

Following is the declaration for java.util.Arrays.binarySearch() method

```
public static int binarySearch(int[] a, int key)
```

Parameters

- a − This is the array to be searched.
- key − This is the value to be searched for.

Return Value

This method returns index of the search key, if it is contained in the array, else it returns (-(insertion point) - 1). The insertion point is the point at which the key would be inserted into the array: the index of the first element greater than the key, or a.length if all elements in the array are less than the specified key.

Exception

NA

Array Index Out Of Bounds Exception

Java supports creation and manipulation of arrays, as a data structure. The index of an array is an integer value that has value in interval [0, n-1], where n is the size of the array. If a request for a negative or an index greater than or equal to size of array is made, then the JAVA throws a ArrayIndexOutOfBounds Exception. This is unlike C/C++ where no index of bound check is done. The ArrayIndexOutOfBoundsException is a Runtime Exception thrown only at runtime. The Java Compiler does not check for this error during the compilation of a program.

Null Pointer Exception

NullPointerException is a RuntimeException. In Java, a special null value can be assigned to an object reference. NullPointerException is thrown when program attempts to use an object reference that has the null value.

These can be:
Invoking a method from a null object.

- Accessing or modifying a null object’s field.
- Taking the length of null, as if it were an array.
- Accessing or modifying the slots of null object, as if it were an array.
- Throwing null, as if it were a Throwable value.
- When you try to synchronize over a null object.

Why do we need the null value?
Null is a special value used in Java. It is mainly used to indicate that no value is assigned to a reference variable. One application of null is in implementing data structures like linked list and tree. Other applications include Null Object pattern and Singleton pattern. The Singleton pattern ensures that only one instance of a class is created and also, aims for providing a global point of access to the object.