# Lets Code!

## Fundamentals: Arrays

Instructor: Tariq Hook

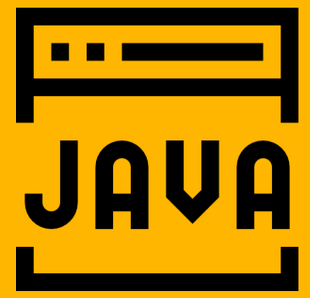
You can find me on github @code-rhino

# Key Terms

- Arrays
- ArrayLists
- Var Args
- Functional Decomposition

- Wrapper Classes
- For Each
- Anonymous Array
- Command-Line Params

# Arrays



# Arrays

An array is a data structure that stores a collection of values of the same type. You access each individual value through an integer index.

Declare an array variable by specifying the array type - which is the element type followed by [] - and the array variable name

```
int[] a;
```

Use the new operator to create the array.

```
int[] a = new int[100];
```

# Arrays

To find the number of elements of an array, use array.length

```
for (int i = 0; i < a.length; i++)
    System.out.println(a[i]);
```

Once you create an array, you canno t change its size

# The "for each" Loop

The enhanced for loop

```
for (variable : collection) statement
```

```
for (int element : a) System.out.println(element);
```

# Array Initializers and Anonymous Arrays

shortcut for creating an array object and supplying initial values at the same time

```
int[] smallPrimes = { 2, 3, 5, 7, 11, 13 };
```

anonymous array:

```
new int[] { 17, 19, 23, 29, 31, 37 }
```

# Array Initializers and Anonymous Arrays

```
smallPrimes = new int[] { 17, 19, 23, 29, 31, 37 };
//is shorthand for
int[] anonymous = { 17, 19, 23, 29, 31, 37 };

smallPrimes = anonymous;
```

# Array Copying

You can copy one array variable into another, but then both variables refer to the same array:

```
int[] luckyNumbers = smallPrimes;
luckyNumbers[5] = 12; // now smallPrimes[5] is also 12
```

If you actually want to copy all values of one array into a new array, you use the copyOf method in the Arrays class

```
int[] copiedLuckyNumbers = Arrays.copyOf(luckyNumbers, luckyNumbers.length);
```

# Command-Line Parameters

```
java Example foo bar baz 1 2 3
```

```java
public class Example {
  public static void main(String[] args) {
    for(String clArg : args) {
      System.out.println(clArg);
    }
  }
}
```

# Array Sorting

To sort an array of numbers, you can use one of the sort methods in the Arrays class:

```
int[] a = new int[10000];
...
Arrays.sort(a)
```

This method uses a tuned version of the QuickSort algorithm that is claimed to be very efficient on most data sets.

Arrays API

# Multidimensional Arrays

Multidimensional arrays use more than one index to access array elements

Declaring a two-dimensional array in Java

```
double[][] balances;
```

You cannot use the array until you initialize it.

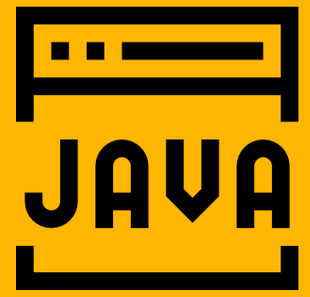
```
balances = new double[NYEARS][NRATES];
```

# Multidimensional Arrays

```
int[][] magicSquare =
{
    {16, 3, 2, 13},
    {5, 10, 11, 8},
    {9, 6, 7, 12},
    {4, 15, 14, 1}
};

magicSquare[i][j]
```

# ArrayLists



# ArrayLists

An ArrayList is a generic class that stores data like an array, but with dynamic sizing and methods.

```
ArrayList intArrList = new ArrayList();
```

```
ArrayList intArrList = new ArrayList<>();
```

# Wrapper Classes

Generic classes (like ArrayList) cannot take primitive types as it's type parameter. Luckily, Java has "wrapper classes" to fix this, and will "autobox" and "unbox" the values.

```
ArrayList< Integer > intArrList = new ArrayList< Integer >();
for(int i = 0; i < 5; i++) {
  intArrList.add(i);
}
```

# Wrapper Classes

- A wrapper class is, essentially, the Object representation of the primitive type.

- They also, sometimes, have helpful methods and class constants within them that the primitives don't.

- For example, Integer has a method to return the bit string of a number. Double has the `NEGATIVE_INFINITY`, `POSITIVE_INFINITY`, and `NaN` constants.

| primitive | Wrapper |
| --- | --- |
| int | Integer |
| byte | Byte |
| short | Short |
| long | Long |
| char | Character |
| float | Float |
| double | Double |
| boolean | Boolean |

# Array Lists

```
String[] arr = {"Foo", "Bar", "Baz"};
ArrayList<string> arrList = new ArrayList<>();
for(String word : arr) {
  arrList.add(word);
}
arrList.get(0);
arrList.set(2, "new one");
arrList.remove(1);
arrList.size();</string>
```

# Copying Arrays and ArrayLists

```
int[] arr1 = {1, 2, 3};
int[] arr2 = arr1;
```

This is just one array with two things referencing it.

You need **`Arrays.copyOf()`** to quickly do a copy of an array.

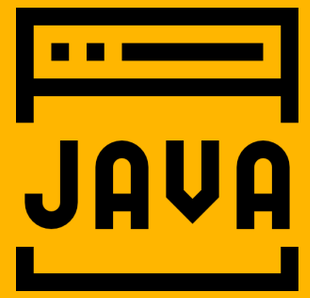
For arrayLists:

```
ArrayList copiedList = new ArrayList<>
(originalList);
```

# Arrays to Array List

You can also turn ArrayLists into arrays and vice-versa.

Look into the ArrayList and Arrays APIs to see what methods they have available.




# Functional Decomposition

# Functional Decomposition

```
public class Example {

  public static void main(String[] args) {

    int[] arr = {1, 2, 3, 4, 5};
    int[] doubledArr = new int[arr.length];

    for(int i = 0; i < arr.length; i++) {
      doubledArr[i] = arr[i] * 2;
    }

    int[] quadArr = new int[arr.length];

    for(int i = 0; i < arr.length; i++) {
      quadArr[i] = doubledArr[i] * 2;
    }

    for(int item : quadArr) {
      System.out.println(item);
    }
  }
}
```

# Functional Decomposition

```
public class Example {
  public static void main(String[] args) {
    int[] arr = {1, 2, 3, 4, 5};
    int[] doubledArr = doubleArr(arr);
    int[] quadArr = doubleArr(doubledArr);

    printArr(quadArr);
  }

  public static int[] doubleArr(int[] arr) {
    int[] doubled = new int[arr.length];
    for(int i = 0; i < arr.length; i++) {
      doubled[i] = arr[i] * 2;
    }
    return doubled;
  }

  public static void printArr(int[] arr) {
    for(int item : arr) {
      System.out.println(item);
    }
  }
}
```

# Var Args

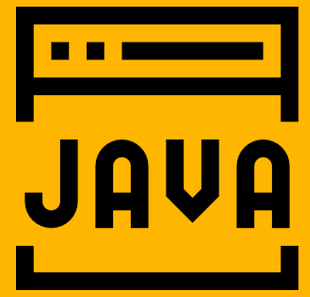
Essentially just an array and must be the last parameter in the function declaration.

```
public static int avg(int... values) {
  int sum = 0;
  for(int value : values) {
    sum += value;
  }
  return sum / values.length;
}
```

# Wrap Up

- Arrays
- ArrayLists
- Var Args
- Functional Decomposition

- Wrapper Classes
- For Each
- Anonymous Array
- Command-Line Params




# Keep Coding !!!

## Clean Code is Happy Code