POWERING POTENTIAL

# Arrays

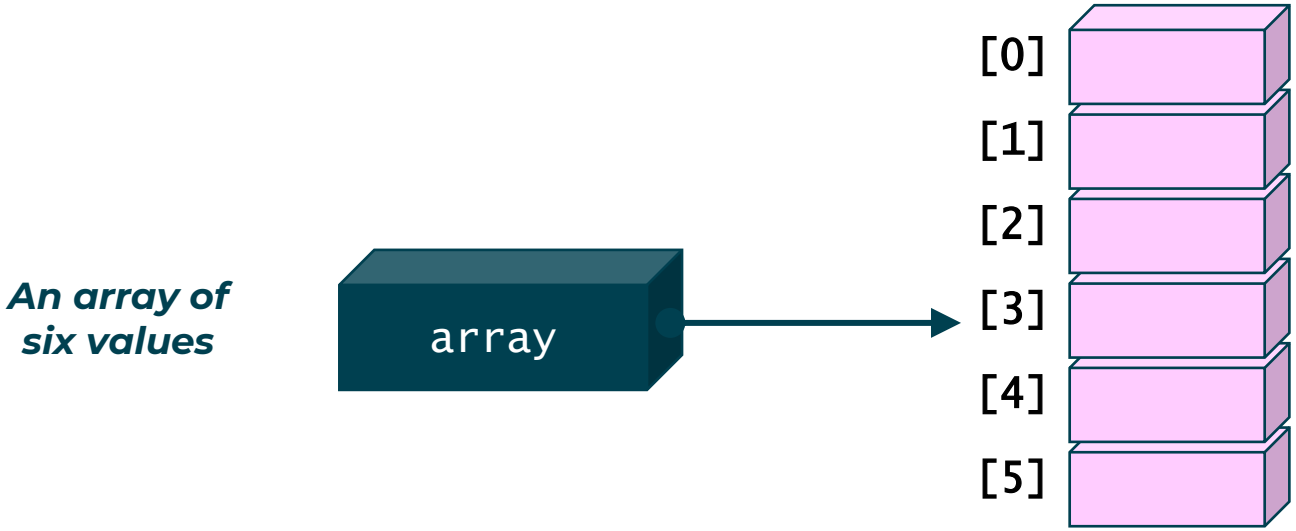
# CONTENTS

- **Objectives**
  - Appreciate the functionality offered by arrays
- **Contents**
  - Arrays
    - Declaring
    - Creating
    - Filling
    - Iterating over
    - Single Step initialisation
    - Arrays of arrays
- **Hands on Labs**

# Arrays

- **An array can store a collection of variables all of the same type**
  - Each element in the array can hold a single item (value or reference type)
  - Array elements are accessed by index number, e.g.  names[3]

- **Arrays are objects**
  - Must be created before they can be used
  - An array variable (of any type) is a reference type



# Steps for Creating an Array

1. **Declare a variable capable of referencing an array and the type that it holds**

```
int[] votes;
String[] names;
```

Note: empty Square brackets

2. **Create array of required length and assign it to variable – Array size is fixed**

```
votes = new int[3];
names  = new String[6];
```

An array of three zeros

An array of six nulls

- Value type elements will be initialised to 0 or false
- References will be initialised to null

# Single Step Initialisation

- **Arrays can be created and initialised at the same time**
  - Array length set automatically

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

separated by commas
All elements are of the same type

```
String[] names = { "Tom", "Jaz", "Kash" };
```

- **Class Arrays provides utility (static) 'array' methods**
  - `sort()`, `copyOf()`, `binarySearch()`

# Filling an Array

- **To access an array element use subscript [] notation using an integer**
  - The first element is [0]

```
votes = new int[3];
for( int i = 0; i < 3; i++ ) {
  votes[i] = ...;
}
```

Fill an 'int' array with int values

- **Indices are checked to ensure they are within range**
  - Use the length property to avoid run-time exceptions

```
String[] names = new String[6];
for( int i = 0; i < names.length; i++ ) {
  names[i] = ...;
}
```

Using the **length** property is good practice

# Iterating over an Array

- **Access array elements by iterating (looping) over them**

```
for( int i = 0; i < votes.length; i++ )
{
  process(votes[i]);
}
```

- **But often you just want to READ ALL of them**
  - Not worrying about “which one am I on”

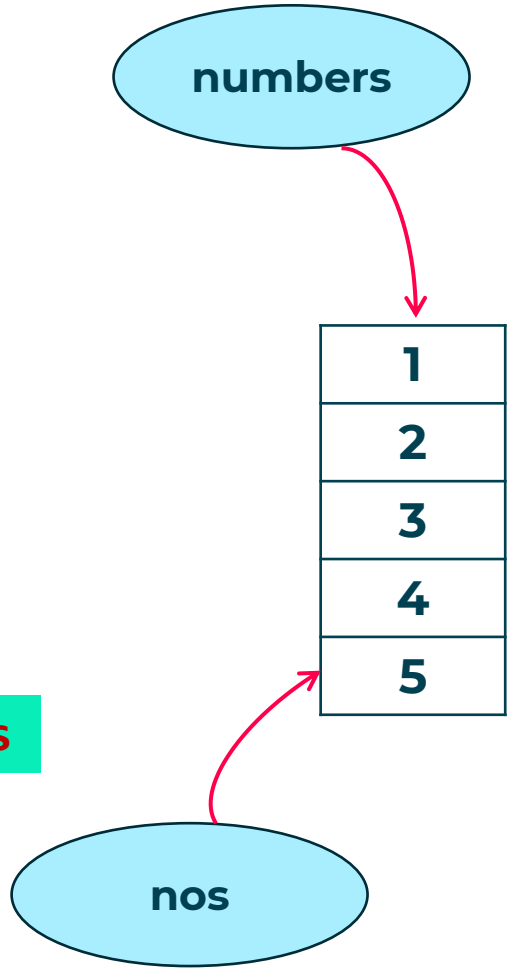
```
for ( int vote : votes )
{
  process(vote);
}
```

# Array as parameter

```
public static void main(String[] args) {
    int[] numbers = {1,2,3,4,5};
    incArray(numbers);
    for (int i : numbers) {
        print(i);
    }
}

private static void incArray(int[] nos)  {
    for (int i = 0; i < nos.length; i++) {
        nos[i]++;
    }
}
```

```
2
3
4
5
6
```




The address of the array numbers is copied to nos

# Quiz – which one displays a sum of int[]?

```
for(int i=0;i < nums.length;i++) {
  int sum = 0;
  sum += nums[i];
  print(sum);
}
```

1

```
int sum = 0;
for(int i=0;i<nums.length;i++) {
  sum += nums[i];
  print(sum);
}
```

4

```
for(int i=0;i < nums.length;i++) {
  int sum = 0;
  sum += nums[i];
}
print(sum);
```

2

```
int sum = 0;
for(int i=0;i< nums.length;i++) {
  sum += nums[i];
}
print(sum);
```

5

```
int sum;
for(int i=1;i<=nums.length;i++) {
  sum + nums[i];
}
print(sum);
```

3

```
int sum = 0;
for (int num : nums) {
  sum += num;
}
print(sum);
```

6

C#

```
int sum = 0;
foreach (int num in nums) {
  sum += num;
}
print(sum);
```

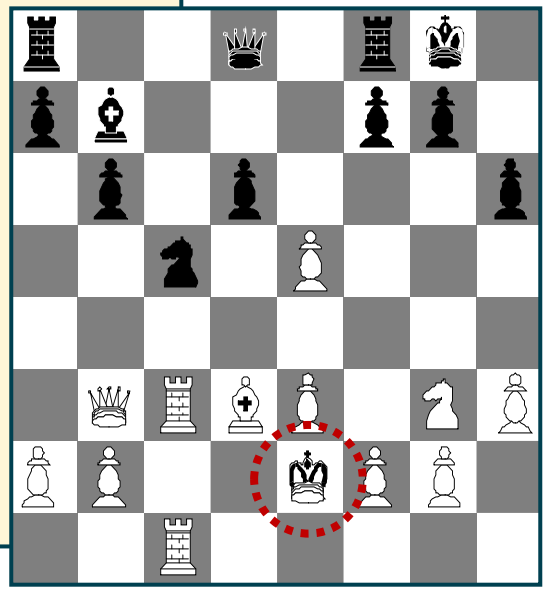
6

# Multi-dimensional array example

```
char EMPTY = 0;
char WKING = 1;
char WQUEEN = 2; // etc...

char[][] board = new char[8][8]; // array of arrays
board[4][1] = WKING;

for (int col = 0; col < board.length; col++) {
   for (int row = 0; row < board[col].length; row++) {
      if (board[col][row] == EMPTY) {
         // the square is empty
      }
      System.out.println(board[col][row]);
   }
}
```



# Variable Number of Parameters

**Variable length parameter lists use `Object...` (literally dot dot dot)**

```
public PrintStream printf(String format, Object... args ) {
   ...
}
```

```
public static void main(String[] args) {
    int x = 1, y = 2, z = 3;
    print(x);
    print(x,y);
    print(x,y,z);
}

static void print(Object... things) {
    for (Object x : things)
        System.out.println(x.toString());
}
```

Must be last parameter

# Review

- **Java supports arrays of objects or primitives**
  - Uses [ ] notation
  - Fixed size collection of elements
- **Arrays are objects themselves**
  - Allocate the array object
  - Assign the elements into the array
  - Use for or enhanced for to iterate over
- **Arrays of arrays are supported**
  - Rarely needed
- **Arrays are a simple ‘fixed size’ sort of collection – much more later**

# Hands On Labs

- Write code to find the sum, min, max and average of numbers in an array
- Implement the bubble sort algorithm

# Multi-dimensional Arrays – non rectangular

- **Non-rectangular grid of names** (jagged arrays)

Empty

```
String[][] names;
names = new String[3][];          // an array of 3 String[]
names[0] = new String[2];
names[1] = new String[6];
names[2] = new String[2];         // 2,6,2 grid of names
for( int row = 0; row < names.length; row++ ) {
 for( int col = 0; col < names[row].length; col++ ) {
  names[row][col] = “”;          // 10 empty names
 }
}
```

# Another example of using continue & break

```
int passMark = 12, passesReqd = 3;
boolean singlePass = false;
int[] examScores = { 15, 7, 3, 12, 15, 7, 9, 11};
for (int examMark : examScores ) {
  if ( examMark >= passMark ) {
    singlePass = true;
    print("Has passed something!");
    break;
  }
}

for (int examMark : examScores ) {
  if (examMark < passMark) {
    continue;
  }
  passesReqd--;
  // Other processing
}
print("Units still required " + Math.max(0,passesReqd) + "\n");
```