CSY2030
Systems Design & Development
Revision of Java 4

Overview of Lecture

- Today we will revise the following in Java:
  - Introduction to Arrays
  - Processing Array contents
  - Looping through Arrays

Introduction to Arrays

- Primitive variables are designed to hold only one value at a time.

  E.g. *int num = 5;*

- Arrays allow the creation of a collection of like values that are indexed by a counter.
- This allows you to store more than one value inside a single variable
- An array can store any type of data but only one type of data at a time.
- An array is a list of data elements.

Introduction to Arrays

- Arrays are declared using a type followed by two square brackets

```
String[] stringArray;
int[] intArray;
double[] doubleArray;
boolean[] booleanArray;
```

- Once an array has been created, it needs to be initiated with a size. This defines how many values the array can store

```
stringArray = new String[10];
intArray = new int[10];
doubleArray = new double[10];
booleanArray = new boolean[10];
```

Introduction to Arrays

- Initialisation can also be done on one line
  - E.g to create an array that can store 10 strings, you can use the code:

```
String[] stringArray = new String[10];
```

- Unlike integers and Strings, you must use the **new** keyword to create an array
- This is because arrays are not primitive types. All non-primitive types must be created using the **new** keyword

Introduction to Arrays

- Once the array has been created, you can assign values to each index

```
String[] stringArray = new String[10];

stringArray[3] = "Third index in the string array";
```

- Each index stores its own value

Introduction to Arrays

- After the value has been written to the array it can be used like any other variable

```
String[] stringArray = new String[10];

stringArray[3] = "Third index in the string array";

System.out.println(stringArray[3]);
```

- You can read or write a value to each index using the same square bracket syntax

Introduction to Arrays

- Note indexes start from zero.

```
String[] stringArray = new String[10];

stringArray[0] = "First index in the string array";

System.out.println(stringArray[0]);
```

- You can read or write a value to each index using. As such, a 10 element string array will have indexes from 0 to 9, trying to write to index 10 will cause an error:

```
String[] stringArray = new String[10];

stringArray[10] = "Tenth index in the string array";

System.out.println(stringArray[10]);
```

Introduction to Arrays

- When the array is created, an empty table of 10 rows is created

```
String[] stringArray = new String[10];
```

Index	Value
0	*null*
1	*null*
2	*null*
3	*null*
4	*null*
5	*null*
6	*null*
7	*null*
8	*null*
9	*null*

Introduction to Arrays

- When you write to an index, the value is stored in the table

```
String[] stringArray = new String[10];
stringArray[0] = "First index in the string array";
```

Index	Value
0	First index in the string array
1	*null*
2	*null*
3	*null*
4	*null*
5	*null*
6	*null*
7	*null*
8	*null*
9	*null*

Introduction to Arrays

- When you write to an index, the value is stored in the table

```
String[] stringArray = new String[10];
stringArray[0] = "First index in the string array";
stringArray[1] = "Second index in the string array";
```

Index	Value
0	First index in the string array
1	Second index in the string array
2	*null*
3	*null*
4	*null*
5	*null*
6	*null*
7	*null*
8	*null*
9	*null*

Creating arrays - Size

- The size of the array must be a non-negative integer
- It may be an integer, a constant or a variable

```
int size = 4;

String[] stringArray = new String[size];
```

- Once the array is created the size cannot be changed

Creating arrays - Size

- Using variables is useful if you don't know the number of elements you'll need at the time of writing the program.
- For instance, asking the user how many numbers they would like to store:

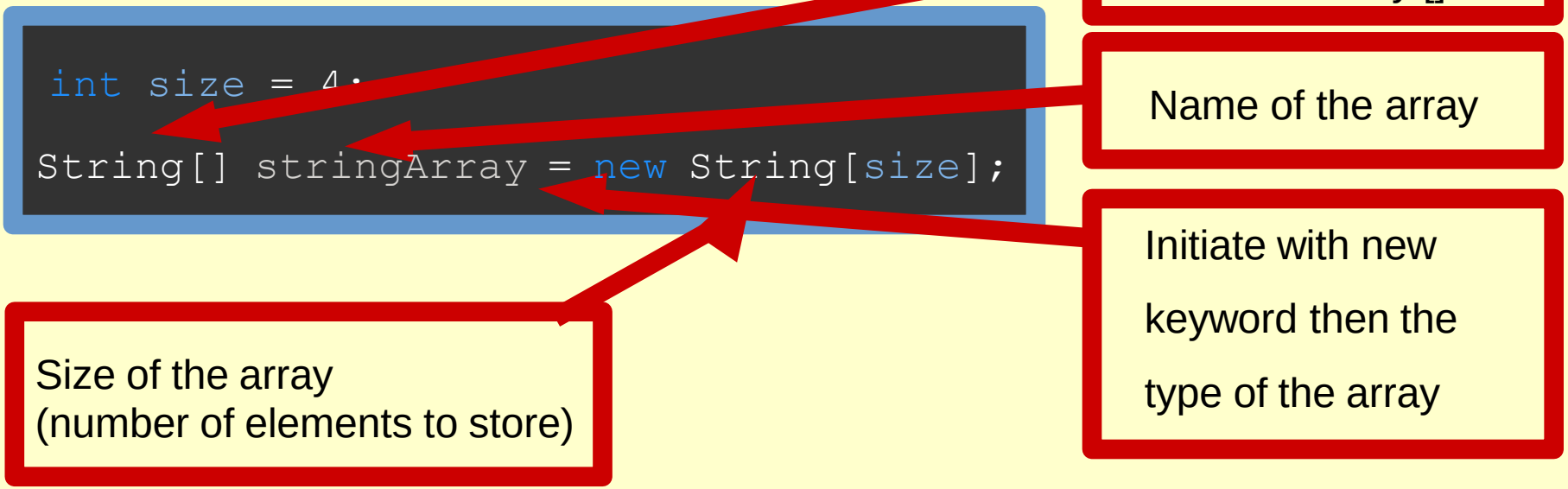
```
Scanner scan = new Scanner(System.in);
System.out.println("How many numbers would you like to store?");
int numItems = scan.nextInt();

int[] numbers = new int[numItems];

for (int i = 0; i < numItems; i++) {
        numbers[i] = scan.nextInt();
}
```

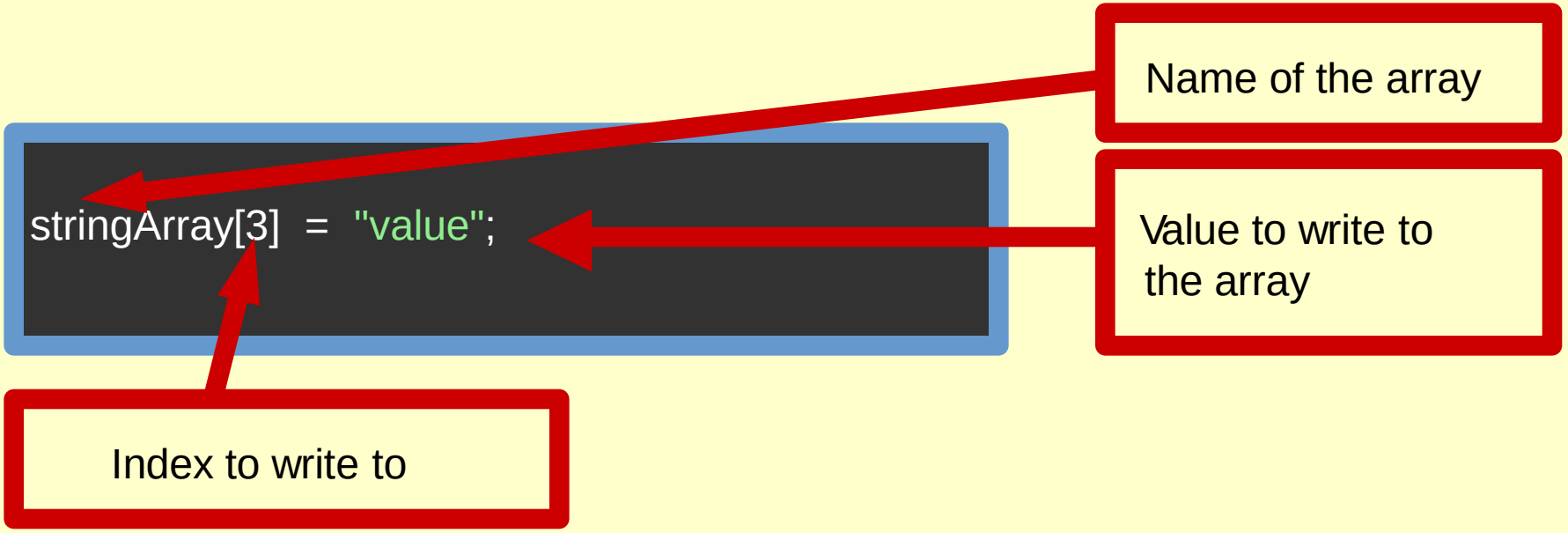
Array syntax - Recap

To create an array:



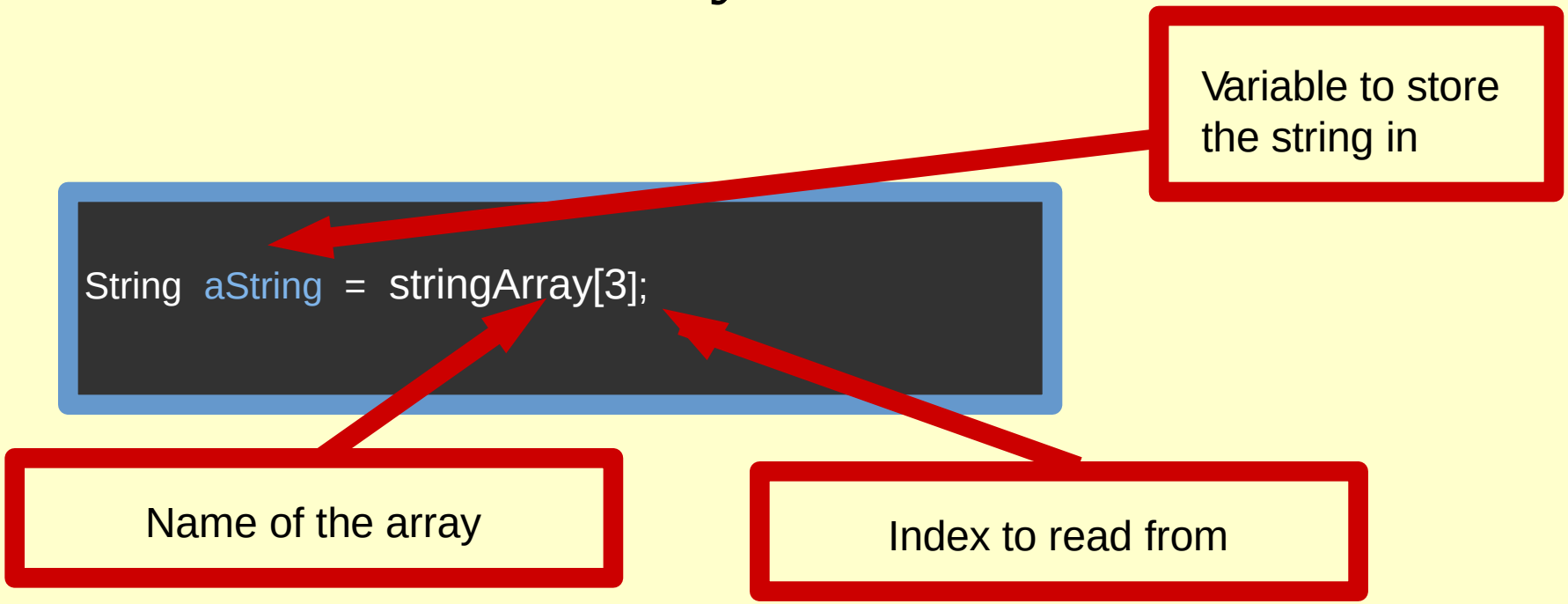
Array syntax - Recap

To write to an array:



Array syntax - Recap

To read from an array:



Array syntax - Recap

- You can treat a variable stored in an array like any other variable

```
int intArray[] = new int[3];

intArray[1] = 2;

intArray[2] = 5;

System.out.println(intArray[1]);

int total = intArray[1] +

intArray[2];

System.out.println(total);
```

Why use Arrays?

- Arrays are useful when you need to group a set of values together and use the in other parts of the program. Consider this code:

```
public class ArrayExample1 {

public static void main(String[] args) {
	int result = add(8, 4, 3);
	System.out.println(result);
}



public static int add(int num1, int num2, int num3) {
	int total = num1 + num2 + num3;
	return total;
}

}
```

- Problem: The add method needs to be passed three arguments

Why use Arrays?

- By using an array, the method header can be simplified

```
public class ArrayExample1 {

        public static void main(String[]        args) {
                int[] intArray = new int[3];
                intArray[0]          =    8;
                intArray[1]          =    4;
                intArray[2]          =    3;
                int result = add(intArray);

                System.out.println(result);
        }

        public static int add(int[] array) {
                int total = array[0] + array[1] + array[2];
                return total;
        }
  }
```

Array shorthand notation

- This code is quite a lot to type just to store 3 numbers

```
int[] intArray = new int[3];
intArray[0] = 8;
intArray[1] = 4;
intArray[2] = 3;
```

- Java allows a shorthand notation which is less to type but has the same result

```
int[] intArray = {8, 4, 3};
```

Array shorthand notation

- The shorthand notation can be used with any types and any number of elements
- If you know what the contents will be up front this saves you from counting the number

```
int[] numbers = {20, 5, 6};

String[] strings = {"This", "is", "an", "array", "of", "strings"};

double[] doubles = {0.7, 20.5, 8.6};

char[] chars = {'c', 'h', 'a', 'r'};
```

Why use arrays?

- This can be applied to the add method to make the code more concise

```
public class ArrayExample1 {

public static void main(String[] args) {
	int[] intArray = {8, 4, 3};
	Int result = add(intArray);
	System.out.println(result);
}


public static int add(int[] array) {
	int total = array[0] + array[1] + array[2];
	return total;
}
}
```

Array length

- The code for the add method assumes there are 3 values in the array

```
public static int add(int[] array) {
    int total = array[0] + array[1] + array[2];
    return total;
}
```

- If you call add() with a 2 element array it will break
- If you call add() with a 4 element array only the first 3 numbers will be added

```
int[] intArray1 = {8, 4};
int[] intArray2 = {8, 4, 3, 7};

int result1 = add(intArray1);
Int result2 = add(intArray2);
```

Array length

- Arrays have a .length property which can be used to find the size of the array

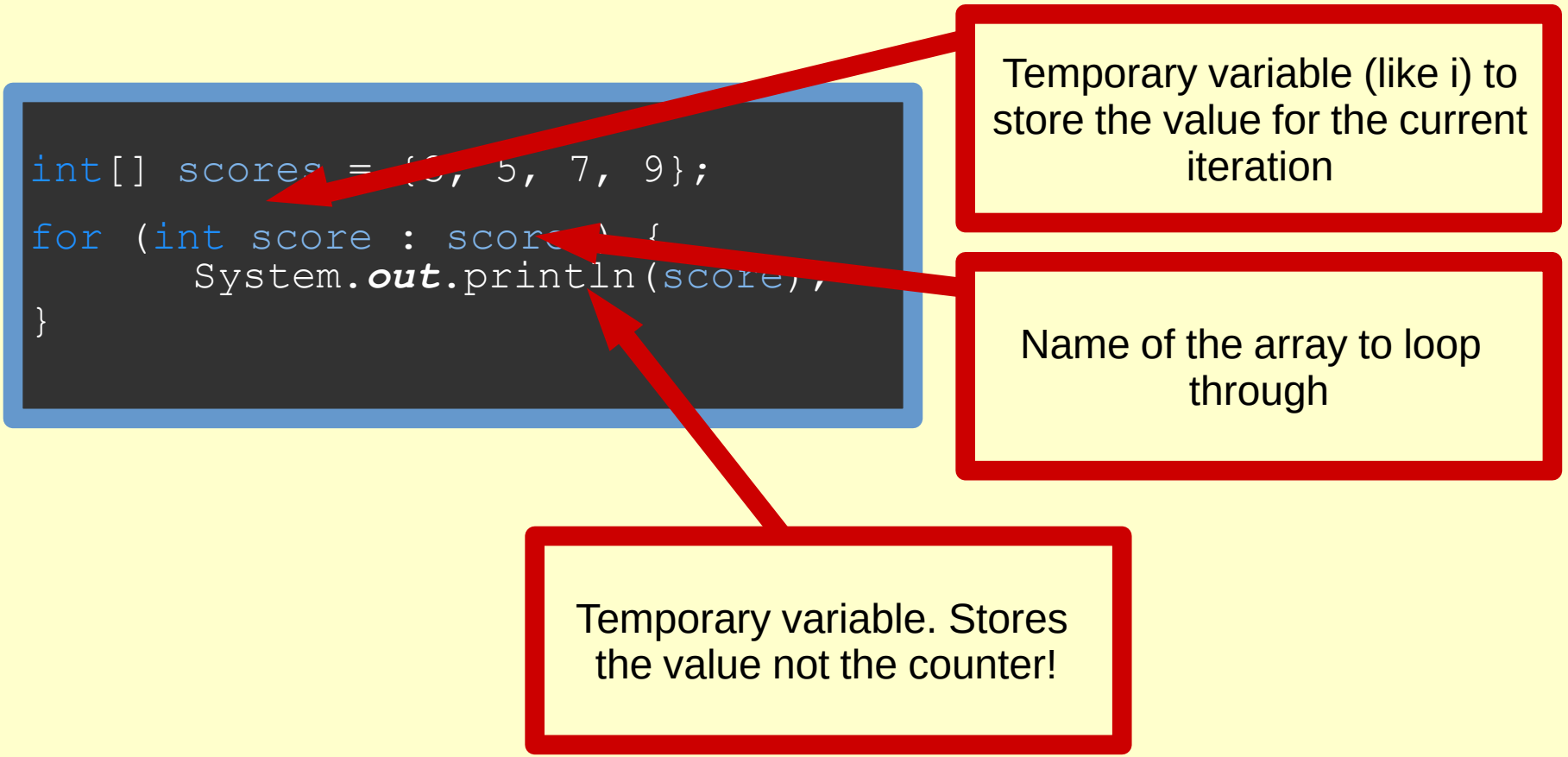
```
int[] intArray1 = {8, 4};
int[] intArray2 = {8, 4, 3, 7};

System.out.println(intArray1.length);
System.out.println(intArray2.length);
```

```
2
4
```

The enhanced for loop

- Java provides a special version of a for loop for looping through an array value



The enhanced for loop

- When using the enhanced for loop you don't have a counter (the i variable, usually)
- The enhanced for loop is quicker to type but is only useful when you don't need a counter

```
int[] scores = {3, 5, 7, 9};

for (int score : scores) {
        System.out.println(score);
}
```

Non-primitive variables

- When you assign a value to a variable using the *new* keyword it creates a new instance
- If you assign an array variable to another array, it will be a reference to the original array

```
int[] array1 = {4, 2, 3};
int[] array2 = array1;
```

This does not copy the array

Non-primitive variables

- Both arrays point to the same instance
- When one is changed, the changes will be reflected in both array variables

```
int[] array1 = {1, 2, 3};
int[] array2 = array1;
array1[1] = 9;
System.out.println(array1[1]);
System.out.println(array2[1]);
```

Prints:
9
9