Java Programming

CPT111 – Lecture 5

Erick Purwanto

Xi'an Jiaotong-Liverpool University
西交利物浦大学

CPT111 Java Programming

Lecture 5

For Loops, Arrays, Static Methods 1

Welcome!

- Welcome to Lecture 5 – For Loops, Arrays, and Static Methods 1
- In this lecture we are going to learn about:
  - For Loops
    - using counters to iterate through loops
    - using nested loops
  - Arrays
    - storing data
    - using different types of arrays
    - using arrays and loops
  - Static Methods 1
    - introducing static methods
    - methods and arrays

Flow Control

- Branching
  - If/If-Else Statement
  - Covered in Week 4

```
if(something is true) {
    // do something
}
```

- Condition Loops
  - While loop
  - Covered in Week 4

```
while(counter < 10) {
    // do something
    counter++;
}
```

- Counter Loops
  - For loop
  - This week !

```
for(int i = 0; i <= 10; i++) {
    // do something
}
```

Part 1: For Loop

- Previously we covered the While loop
- Good for handling an unknown number of loops
  - loop/repeat while a condition is not true
  - for example, counting a total
- Can be used for counting

Countdown with While

```
int countdown = 10; // initialize counter
while(countdown > 0) { // looping condition
    System.out.println(countdown);
    countdown = countdown - 1;  // update
}
System.out.println("Go!");
```

```
tput - temporary1 (r
run:
10
9
8
7
6
5
4
3
2
1
Go!
BUILD SUCCES
```

- Requires creating a variable before the loop
  - and updating the counter
- The scope of the variable goes beyond the loop

Countdown with For

```
for(int i = 10; i > 0; i--) {
    System.out.println(i);
}
System.out.println("Go!");
```

```
tput - temporary1 (r
run:
10
9
8
7
6
5
4
3
2
1
Go!
BUILD SUCCES
```

- Does the exact same thing with less code
- Everything (condition, counter initialization and update) handled in the for loop header
- If we know how many times we want to loop, best to use a for loop

The For Loop

```
for(int i = 0; i < 10; i++)
```

The initial value. Here, we start at 0.

The condition: continue to loop ***while***...

The increment: change i by this much each time we loop

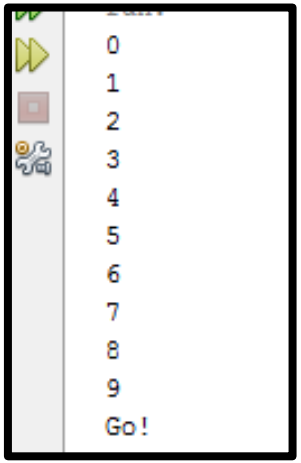
```
for(int i = 0; i < 10; i++)
```

The initial value. Here, we start at 0.

The for loop

Changing the initial value

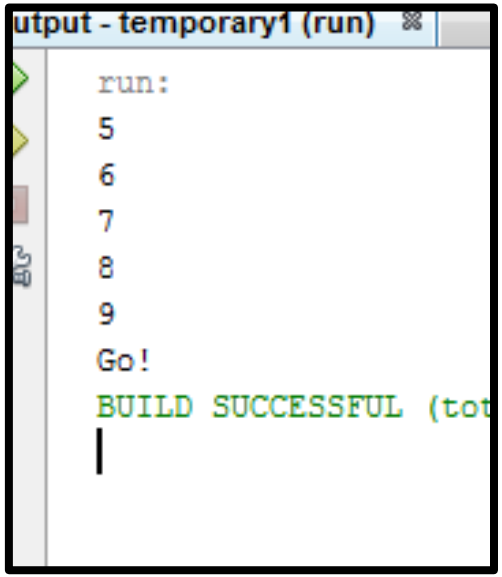
```
for(int i = 0; i < 10; i++) {
    System.out.println(i);
}
System.out.println("Go!");
```



- Counting up from 0 to 9
- Starting from 0

Changing the initial value

```
for(int i = 5; i < 10; i++) {
    System.out.println(i);
}
System.out.println("Go!");
```



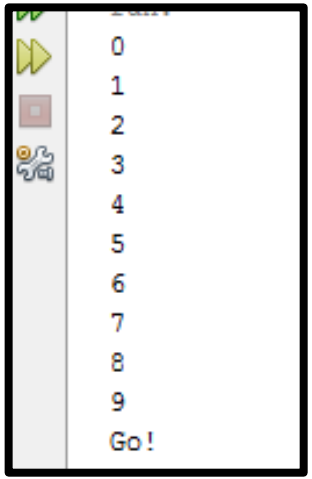
- Counting up from 5 to 9
- Starting from 5
- We can start our initial variable anywhere

```
for(int i = 0; i < 10; i++)
```

The condition:
continue to loop
while...

Changing the condition

```
for(int i = 0; i < 10; i++) {
    System.out.println(i);
}
System.out.println("Go!");
```



- Will loop while i <10
- Looping 10 times, starting from 0, finishing at 9

Changing the condition

```
for(int i = 0; i <= 5; i++) {
    System.out.println(i);
}
System.out.println("Go!");
```

```
utput - temporary1 (run)
run:
0
1
2
3
4
5
Go!
BUILD SUCCESSFUL
```

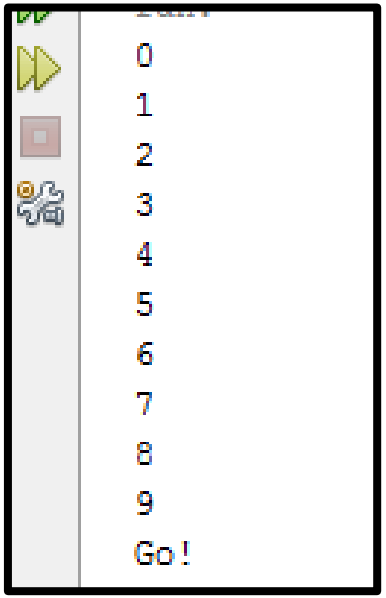
- Will loop while i <= 5
- Looping 6 times, starting from 0, finishing at 5
- Can use the comparison operators discussed in Week 4 such as < > =

```
for(int i = 0; i < 10; i++)
```

The increment: change i by this much each time we loop

Changing the increment

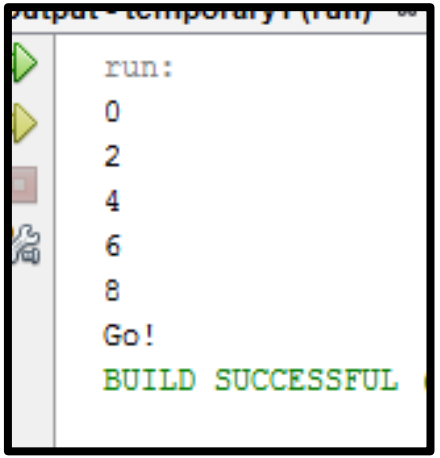
```
for(int i = 0; i < 10; i++) {
    System.out.println(i);
}
System.out.println("Go!");
```



- Counting up from 0 to 10
- Incrementing by 1 each time
- Adding 1 to the loop each time

Changing the increment

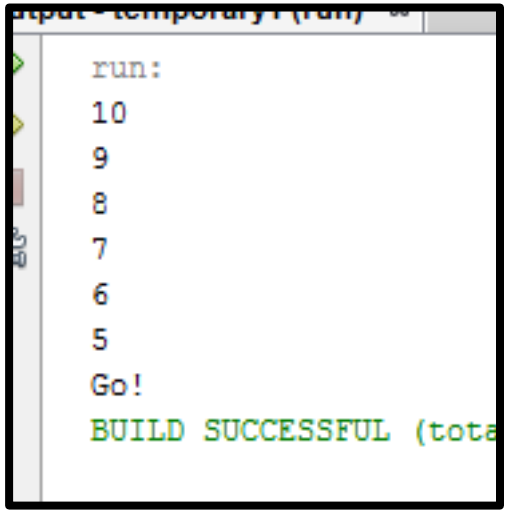
```
for(int i = 0; i < 10; i = i+2) {
    System.out.println(i);
}
System.out.println("Go!");
```



- Counting up from 0 to 10
- Incrementing by 2 each time
  - Adding 2 to the loop each time

Changing the increment

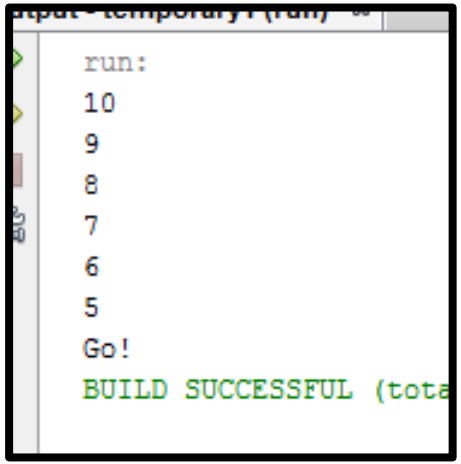
```
for(int i = 10; i >= 5; i--) {
    System.out.println(i);
}
System.out.println("Go!");
```



- Counting up from 10 to 5
- Decrementing by 1 each time
  - removing 1 from the loop each time
- We can count up or down by almost anything!

Scope of a Loop

```
for(int i = 10; i >= 5; i--) {
    System.out.println(i);
}
System.out.println("Go!");
```



- The i variable has a scope of only within the for loop
- Does not exist outside!

In-Class Quiz 1

- Which of the following loop iterates exactly 10 times?

  - ❑ `for(int i = 10; i < 20; i++)`
  - ❑ `for(int j = 30; j > 20; j--)`
  - ❑ `for(int k = 1; k <= 10; k++)`
  - ❑ `for(int l = 5; l < 25; l += 2)`

For Loop Summary

1. The loop is first initialised with a value, which can be any number
2. A condition is then checked. If this is met, then the content is processed
3. The next stage is to increment the loop
4. Next, the condition is checked again. If it is still met, then the content is processed again and then incremented
5. This process repeats until the condition is no longer met

In the example below, the variable i is initialised to be 1, the condition is that it is less than or equal to 10, and the increment is to add one (i = i +1)

```
for (int i = 1; i <= 10; i++) {
    System.out.print(i + " ");
}
```

Nested For Loops

- Nested For loops
  - similar to IF statements
  - can put a loop within another loop
    - important to consider the scope issue

Nested For Loops

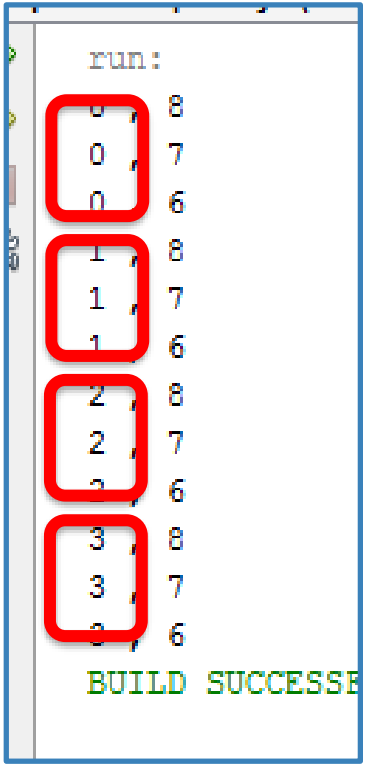
```
for(int i = 0; i <= 3; i++) {
    // loop the ith variable from 0 to 4
    for(int j = 8; j >= 6; j--) {
        // loop the jth variable from 8 to 6
        System.out.println(i + " , " + j);
    }
}
```

```
run:
0 , 8
0 , 7
0 , 6
1 , 8
1 , 7
1 , 6
2 , 8
2 , 7
2 , 6
3 , 8
3 , 7
3 , 6
BUILD SUCCESSF
```

- Two loops here
- First loop runs 4 times, from 0 to 3
- Second loop runs 3 times, from 8 to 6
- The inner loop is inside the outer loop

Nested For Loops

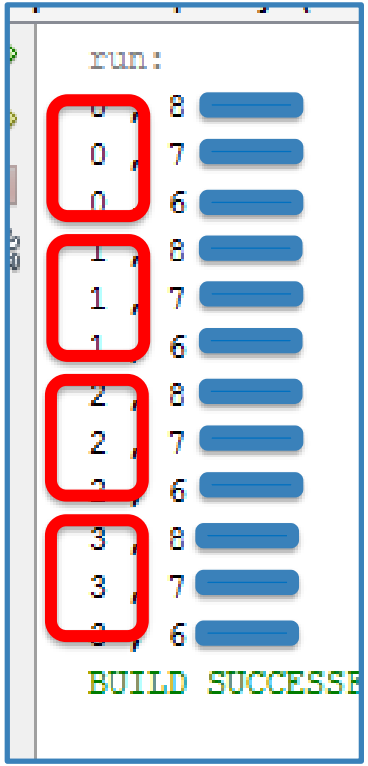
```
for(int i = 0; i <= 3; i++) {
    // loop the ith variable from 0 to 4
    for(int j = 8; j >= 6; j--) {
        // loop the jth variable from 8 to 6
        System.out.println(i + " , " + j);
    }
}
```



- First loop runs four times
- Inside this first loop,
  the second loop runs 3 times

Nested For Loops

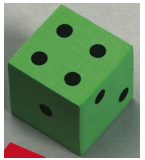
```
for(int i = 0; i <= 3; i++) {
    // loop the ith variable from 0 to 4
    for(int j = 8; j >= 6; j--) {
        // loop the jth variable from 8 to 6
        System.out.println(i + " , " + j);
    }
}
```



- Second loop (jth variable) runs inside 3 times inside each iteration of (ith variable)
- System.out is inside both loops
- Can use both i and j variables, as both are inside scope
- Loop runs 4 * 3 times = 12 times!

Another Example

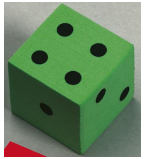
- Our dice problem in Week 3



```
double rand = ((Math.random() * 6) + 1);
int dice = (int) rand;
```

One Dice Roll

- Our dice problem…
- Good for displaying one value



```
double rand = ((Math.random() * 6) + 1);
int dice = (int) rand;
System.out.println("Result is " + dice);
```

Multiple Dice Rolls

- We can copy and paste...
- Never do this!
  - So much repeated code
- What if we wanted 200 dice?

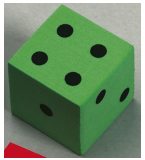
X 200

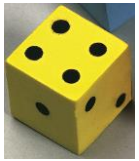
```
double rand = ((Math.random() * 6) + 1);
int dice = (int) rand;
System.out.println("Result is "  + dice);
rand = ((Math.random() * 6) + 1);
dice = (int) rand;
System.out.println("Result is " + dice);
rand = ((Math.random() * 6) + 1);
dice = (int) rand;
System.out.println("Result is " + dice);
rand = ((Math.random() * 6) + 1);
dice = (int) rand;
System.out.println("Result is " + dice);
rand = ((Math.random() * 6) + 1);
dice = (int) rand;
System.out.println("Result is " + dice);
```

Solution with For Loops

- If we want to generate several numbers, we can use a loop
- How many numbers do we want to generate? 4?

```
public static void main(String[] args) {
    for(int i = 0; i < 4; i++) {
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        System.out.println("Result is " + dice);
    }
}
```

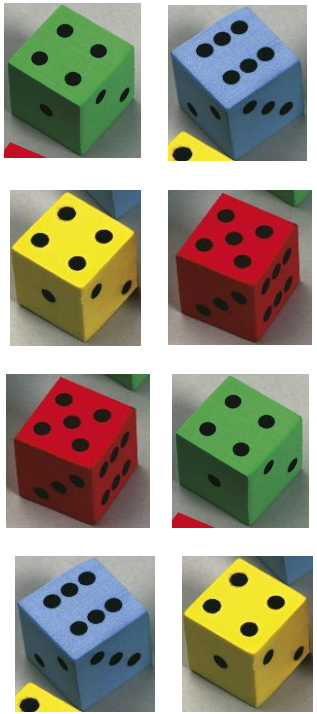




Solution with For Loops

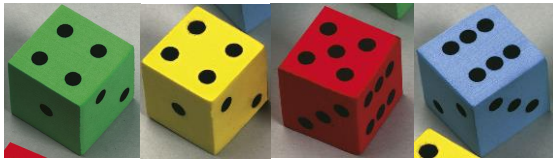
- Eight dice?
- Almost the same code!

```
public static void main(String[] args) {
    for(int i = 0; i < 8; i++) {
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        System.out.println("Result is " + dice);
    }
}
```



Storing Data

- For loops useful for iterating
  - But how can we store the output?
- Can we store data?
  - What if we want to store our dice results
  - Remember them outside the loop?
- We can <u>store our data in an array</u>!

[]

Part 2: Arrays

Index	Value
0	"The"
1	"cow"
2	"jumped"
3	"over"
4	"the"
5	"moon"

- In all programming languages, we often need to store a group or collection of data
- Arrays are just one way to store many data
- In Java, we can have:
  - Arrays of primitive data
  - Arrays of Objects
- This is an array of Objects of type String
- It has a length of 6

Arrays Visualised

An array is a continuous piece of memory, each piece big enough for a primitive or a pointer – the memory address of an Object.

Index	0	1	2	3	4
Value	100	200	300	400	500

```
int[] myArray = new int[5];
myArray[0] = 100;  myArray[1] = 200;  myArray[2] = 300;
myArray[3] = 400;  myArray[4] = 500;
```

- We can access (get) any value at any index with 1 call
- If our array has 1 million items, we can get array[999 999] in one step – FAST!
- If we want to search our array for an item using a ‘for’ loop, we have to iterate over the whole array

An array of primitives

```
int [] myArray;

myArray = new int[5];

myArray[0] = 100;
myArray[1] = 200;
myArray[2] = 300;
myArray[3] = 400;
myArray[4] = 500;
```

Declare a variable called myArray of type int[] (int array)

Get a new array object with memory for 5 ints, assign it to the variable myArray

Initialise each index or "slot" in the array with an int value. Each index is like a **variable** of **type** int.

- Arrays have a variable or a property called **length**
- Here `myArray.length` will return 5
- However, if you try
  `myArray[5] = 600;`
  you will get an error: `ArrayIndexOutOfBoundsException`

Arrays can be initialised when declared

```
int[] myArray = {100, 200, 300, 400, 500};
```

The same as:

```
int[] anArray;
anArray = new int[5];
anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
anArray[3] = 400;
anArray[4] = 500;
```

```
int[] anArray = new int[5];
anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
anArray[3] = 400;
anArray[4] = 500;
```

Index	Value
0	100
1	200
2	300
3	400
4	500

```
int[] anArray = new int[5];
anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
anArray[3] = 400;
anArray[4] = 500;
```

- You cannot change the size of an array after creating it.
- Trying to access an index which is equal to or greater than **`array.length`** will cause an exception (error)
- This is a very, very common cause of error, and it will cause your program to crash!

Array initialization

When created, arrays are automatically **initialized** with the default value of their type. E.g.

```
String[] s = new String[100];
// default values: null

boolean[] b = new boolean[4];
// default values: false

int[] i = new int[10];
// default values: 0
```

Array Types

An array can store almost anything, not just ints!

Index	0	1	2	3	4
Value	"a"	"b"	"c"	"d"	"e"

```
String [] myArray = {"a", "b", "c", "d", "e"};
```

Index	0	1	2
Value	true	false	true

```
boolean[] myArray = {true, false, true};
```

Changing an Array Value

Can change an array value easily, by searching for its index

```
String [] myArray = {"a", "b", "c", "d", "e"};
```

Index	0	1	2	3	4
Value	"a"	"b"	"c"	"d"	"e"

```
myArray[1] = "hello";
```

Index	0	1	2	3	4
Value	"a"	"hello"	"c"	"d"	"e"

```
myArray[4] = "bye";
```

Index	0	1	2	3	4
Value	"a"	"hello"	"c"	"d"	"bye"

Using an Array Value

Can use an array easily, by accessing its index

```
String [] myArray = {"a", "b", "c", "d", "e"};
```

Index	0	1	2	3	4
Value	"a"	"b"	"c"	"d"	"e"

```
System.out.println("Value is " + myArray[1]);
```

Value is b

Note: to access the first value in an array, we must access the 0th index!

```
myArray[0]
```

Array Lengths

Arrays can not be resized after creation
We can get the length using **myArray.length**.

Index	0	1	2	3	4
Value	"a"	"b"	"c"	"d"	"e"

```
String [] myArray = {"a", "b", "c", "d", "e"};
int len = myArray.length;
```

will return len = 5

Index	0	1	2
Value	true	false	true

```
boolean[] myArray = {true, false, true};
int len = myArray.length;
```

will return len = 3

For Loops and Arrays

- We can use for loops to search our arrays
- Used with arrays all the time!
- Think about how zero-index works with <length. Clever!

Index	0	1	2	3	4
Value	"a"	"b"	"c"	"d"	"e"

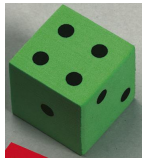
```
for (int i = 0; i < myArray.length; i++) {
    System.out.println(myArray[i]);
}
```

```
a
b
c
d
e
```

Looping Dice

- We used a for loop to roll our dice 4 times
- So we can display them
- What if we want to store our values?

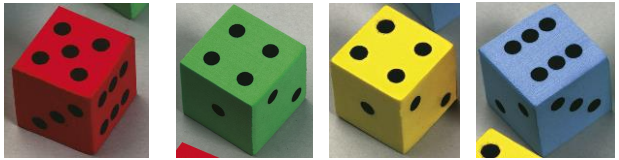
```
public static void main(String[] args) {
    for(int i = 0; i < 4; i++) {
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        System.out.println("Result is " + dice);
    }
}
```



Useful Information Storage - Dice

- We can use arrays to store information
- Create an array, to store 4 dice
- Create it outside the loop!

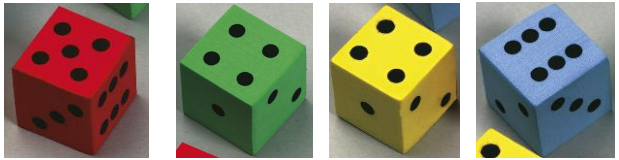
```
public static void main(String[] args) {
    int[] diceList = new int[4];
    for(int i = 0; i < 4; i++){
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        diceList[i] = dice;
        System.out.println("Result is " + dice);
    }
}
```



Useful Information Storage - Dice

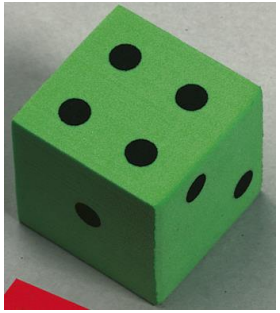
- Store the output in the appropriate place in the array
- We might want to do more with it
- What if we want to calculate the sum of all the dice rolls?

```
public static void main(String[] args) {
    int[] diceList = new int[4];
    for(int i = 0; i < 4; i++){
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        diceList[i] = dice;
        System.out.println("Result is " + dice);
    }
}
```



Using our Data

- Once we have our data in arrays, we can use it!
- Previous examples showed us displaying our data with system.outs
- But what if we want to do other things?
    - What is the average value?
    - What is the highest value?
    - What is the lowest value?
    - What is the sum total?



Dice – Find the Total

- Assume we created our array of diceList earlier
- Assume we assigned values using Random numbers
- Can use a loop to loop through the array
- Variable to store grand total must be OUTSIDE loop
- Loop will go through every item in array so can use diceList.length

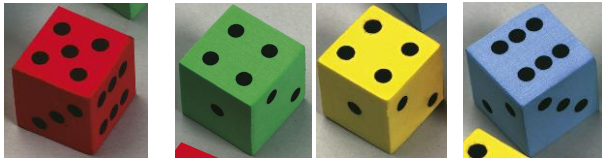
```
int total = 0;

for(int i = 0; i < diceList.length; i++)
```

Dice – Find the Total

- In each iteration, we store the total value, and add the value of the ith diceList array to the total

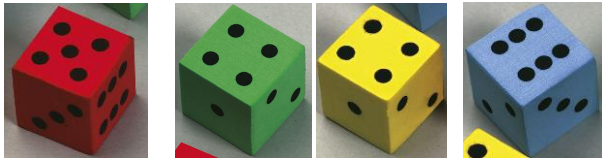
```
int total = 0;
for (int i = 0; i < diceList.length; i++) {
    total = total + diceList[i];
}
```



Dice – Find the Total

- Finally, report the total value

```
int total = 0;
for (int i = 0; i < diceList.length; i++) {
    total = total + diceList[i];
}
System.out.println("The total is " + total);
```



Our Dice Main Method

- Its getting quite long
- This is before we add any scanner inputs, or any other things, such as finding highest value
- How can we break our program up?

```java
public static void main(String[] args) {
    int[] diceList = new int[4];
    for (int i = 0; i < 4; i++) {
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        diceList[i] = dice;
        System.out.println("Result is " + dice);
    }
    int total = 0;
    for (int i = 0; i < diceList.length; i++) {
        total = total + diceList[0];
        System.out.println("THis dice value is " + diceList[0]);
    }
    System.out.println("The total is " + total);

}
```

Main Method

- So far you have written a lot of code
- All in the **main method**
- There is a LOT of code there
- Does many different things

```java
public static void main(String[] args) {
    int[] diceList = new int[4];
    for (int i = 0; i < 4; i++) {
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        diceList[i] = dice;
        System.out.println("Result is " + dice);
    }
    int total = 0;
    for (int i = 0; i < diceList.length; i++) {
        total = total + diceList[0];
        System.out.println("THis dice value is " + diceList[0]);
    }
    System.out.println("The total is " + total);

}
```

Part 3: Methods

- Long blocks of code very hard to read
- Many things happening
- May be a lot of repeated statements
  - You may have also have done a lot of copy and pasting
- Good practice is that a block of code should do ONE thing
- Minimises errors, increases readability

Methods

- We can create additional methods
- A method is a collection of statements grouped together
- We will discuss them in more depth next week
- Useful to split up repeated code, or divide code into different blocks

Main method

- **main** is a method
- It does not return a type (void)
- It has 1 parameter
- It is a special case, runs when program starts
- As code becomes more complex, should use multiple methods

```
public static void main(String[] args) {
    double numA = Math.random() * 10;
    double numB = Math.random() * 10;
    double ans = 3 * numA + numB - 1;
    System.out.println(ans);
}
```

Declaring a method

the first line is the **method header**

```
public static int addTwoNumbers() {
    int result = 3 + 5;
    return result;
}
```

- Method must have a name
  - (almost) any name you like
  - Should be meaningful
  - Should tell us what the method does

- Our methods are **public static**
  - also called **static method** or **function**
  - we will explain this in future weeks
  - for now, don't worry about this!

Declaring a method

Remember **curly brackets** for the **code block**

```
public static int addTwoNumbers() {
    int result = 3 + 5;
    return result;
}
```

- A method is a code block
- Need to use curly brackets to mark beginning and end

Declaring a method

Declare type in header

```
public static int addTwoNumbers() {
    int result = 3 + 5;
    return result;
}
```

- Method must return a type
  - What we want out of the method
  - `void` for nothing
  - `int` for an integer
  - `boolean` for a boolean

- Used in 2 places
  - Declare type in header
  - Return a value

Declaring a method

Declare type in header

```
public static void addTwoNumbers() {
    int result = 3 + 5;
    System.out.println(result);
    return;
}
```

- Can return nothing
  - Use void
  - Do not return anything

- Useful for displaying
  - otherwise, return the value to be printed by other methods!

Calling a method in main

- Program has two methods
- Main method, which calls the addTwoNumbers method
- addTwoNumbers method is run
- Returns a result

```
public static void main(String[] args) {
    int total = addTwoNumbers();
    System.out.println("Total is " + total);
}

 public static int addTwoNumbers(){
    int result = 3 + 5;
    return result;
}
```

Calling a method in main

- This time, runs the main method
- but calls the addTwoNumbers() method 3 times
- Can make it more complex, but saves on code writing
- For example, display result nicely

```
public static void main(String[] args) {
    int total = addTwoNumbers();
    total = total + addTwoNumbers();
    total = total + addTwoNumbers();
    System.out.println("Total is " + total);
}

 public static int addTwoNumbers(){
    int result = 3 + 5;
    return result;
}
```

Calling a method in main

- displayTotal method
- Some system.outs
- Formatting adjusted to be nicer
- It's a void method
  - No need for a return type
  - Just display
- Has one parameter

```
public static void main(String[] args) {
    int total = addTwoNumbers();
    displayTotal(total);
    total = total +addTwoNumbers();
    displayTotal(total);
    total = total +addTwoNumbers();
    displayTotal(total);
}
public static int addTwoNumbers(){
    int result = 3 + 5;
    return result;
}
public static void displayTotal(int input) {
    System.out.println("**************");
    System.out.println("*Total is " + input + "*");
    System.out.println("**************");
}
```

A better method

- How can we make more useful methods?
- Add parameters so we can customise it

- **Method parameters**
- What the method needs to run
- What does the method need?

- Declared in method header

Method Parameter

- Can have as many parameters as we want
- 2, 3, 4, … as many as you want!

```
public static int addTwoNumbers(int num1, int num2) {
    int result = num1 + num2;
    return result;
}
```

- The method applies within curly brackets
- Scope of the variables is within this code block only

- Can be of any type or name

Calling method with Parameters

```
public static void main(String[] args) {
    int total = addTwoNumbers(3, 5);
    displayTotal(total);
}
public static int addTwoNumbers(int num1, int num2) {
      int result = num1 + num2;
      return result;
}
```

Two integers passed in as arguments

Method has two parameters of type int

Calling a method multiple times

```
public static void main(String[] args) {
    int total = addTwoNumbers(3, 5);
    displayTotal(total);
    total = addTwoNumbers(8, 2);
    displayTotal(total);
}
public static int addTwoNumbers(int num1, int num2) {
      int result = num1 + num2;
      return result;
}
```

Same method called two times, with different arguments

Different totals each time

Parameters passed in

In-Class Quiz 2 – Local Variable

- What will be printed by this program?

```
public static int addTwoNumbers(int num1, int num2) {
    int result = num1 + num2;
    return result;
}

public static void main(String[] args) {
    int num1 = 2;
    int num2 = 5;
    int result = 10;
    addTwoNumbers(num1, num2);
    System.out.println(result);
}
```

- 2
- 5
- 7
- 10

Remember Our Dice Main Method

- Its getting quite long
- This is before we add any scanner inputs, or any other things, such as finding highest value
- How can we break our program up?

```java
public static void main(String[] args) {
    int[] diceList = new int[4];
    for (int i = 0; i < 4; i++) {
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        diceList[i] = dice;
        System.out.println("Result is " + dice);
    }
    int total = 0;
    for (int i = 0; i < diceList.length; i++) {
        total = total + diceList[0];
        System.out.println("THis dice value is " + diceList[0]);
    }
    System.out.println("The total is " + total);

}
```

Part 4: Method and Array

- Create an array in our main method
- Create a method to assign values to an array
  - Call it generateDice()
- Create another method findTotal() to find total

```java
public static void main(String[] args) {
    int[] diceList = new int[4];
    for (int i = 0; i < 4; i++) {
        double rand = ((Math.random() * 6) + 1);
        int dice = (int) rand;
        diceList[i] = dice;
        System.out.println("Result is " + dice);
    }
    int total = 0;
    for (int i = 0; i < diceList.length; i++) {
        total = total + diceList[0];
        System.out.println("THis dice value is " + diceList[0]);
    }
    System.out.println("The total is " + total);

}
```

Create generateDice() Method

- Create generateDice() method
- Receives an array parameter
- Other code is the same
- Returns the array

```
public static int[] generateDice(int[] input) {
    // Loop for length of array
    for (int i = 0; i < input.length; i++) {
        // Dice generation code
        double rand =
                  ((Math.random() * 6) + 1);
        int dice = (int) rand;
        input[i] = dice;
        System.out.println("Result is " +
                              dice);
    }
    // return the array
    return input;
}
```

Create findTotal() method

- Create findTotal() method
- Receives an array parameter
- Other code is the same
- Returns an integer, the total value

```
public static int findTotal(int[] input){
    int total = 0;
    for (int i = 0; i < input.length; i++) {
        total = total + input[i];
        System.out.println("This dice value
                                is " + input[i]);
    }
    return total;
}
```

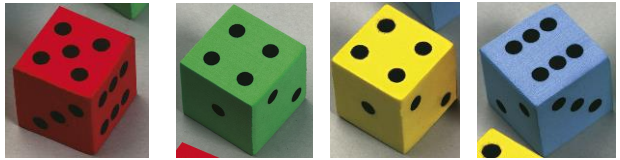
New main method

- Now much smaller
- Creates an empty array first
- Passes this into generateDice()
- Populated array is returned
- This is then passed into findTotal()
- An integer is returned, and displayed

```
public static void main(String[] args) {
    int[] diceList = new int[4];

    diceList = generateDice(diceList);

    int total = findTotal(diceList);
    System.out.println("The total is " +
                                    total);
}
```



Printing All values

- Create a method called `printInts()` ← **used in Labs and CW1 !**
- Receives an array as a parameter
- Runs a loop and displays output along the way
- Does not return anything as it is a void method

```
public static void printInts(int[] nums) {
    System.out.print("[");
    for(int i = 0; i < nums.length; i++) {
        if(i != nums.length-1)
            System.out.print(nums[i] + ", ");
        else
            System.out.print(nums[i]);
    }
    System.out.println("]");
}
```

Printing All values

- Call the printInts() method in the main method
- Very useful for Lab, CW1 this week!

```
public static void main(String[] args) {
    int[] diceList = new int[4];
    diceList = generateDice(diceList);

    printInts(diceList);
}
```

Output - Lecture4 (run)

```
run:
[6, 6, 1, 4]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Finding Max Value

- We loop (iterate) over arrays with a for loop, looking for current max

```
public static int findMax(int[] input) {
    // Set default max value as being the first value
    int max = input[0];
    // loop round array, starting from the second value
    for (int i = 1; i < input.length; i++) {
        // check each value, if greater than current max
        if (input[i] > max) {
            // set the max to be the new value
            max = input[i];
        }
    }
    // return the chosen max value
    return max;
}
```

Calling our method

- Call findMax() method in the main method:

```
public static void main(String[] args) {
    int[] diceList = new int[4];
    diceList = generateDice(diceList);
    printInts(diceList);
    int max = findMax(diceList);
    System.out.println("The max value is " + max);
}
```

Output - Lecture4 (run)

```
run:
[2, 1, 6, 5]
The max value is is 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

Finding Max Index

- We loop (iterate) over arrays with a for loop
- Almost entirely the same as before

```
public static int findMaxLoc(int[] input) {
    // Set default max value as being the first value
    int max = input[0];
    int maxLocation = 0;
    // loop round array starting from the second value
    for (int i = 1; i < input.length; i++) {
        // check each value, if greater than current max
        if (input[i] > max) {
            // set the max to be the new value and store its index
            max = input[i];
            maxLocation = i;
        }
    }
    // return the chosen max value's index
    return maxLocation;
}
```

Calling our method

- Call the findMaxLoc() method in the main method:

```
public static void main(String[] args) {
    int[] diceList = new int[4];
    diceList = generateDice(diceList);
    printInts(diceList);
    int max = findMax(diceList);
    System.out.println("The max value is " + max);
    int loc = findMaxLoc(diceList);
    System.out.println("The index value is " + loc);
}
```

Output - Lecture4 (run)

```
run:
[3, 2, 6, 3]
The max value is is 6
The index value is is 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Thank you for your attention !

- In this lecture, you have learned to:
  - loop using for loops and nested for loops
  - create array of primitives and objects, and operate on it
  - use static methods to do computation modularly
  - manipulating array using a method

- Please continue to Lab 5, and solve
  - Exercise #5.1 - #5.4 and
  - CW1 #5.1, #5.2