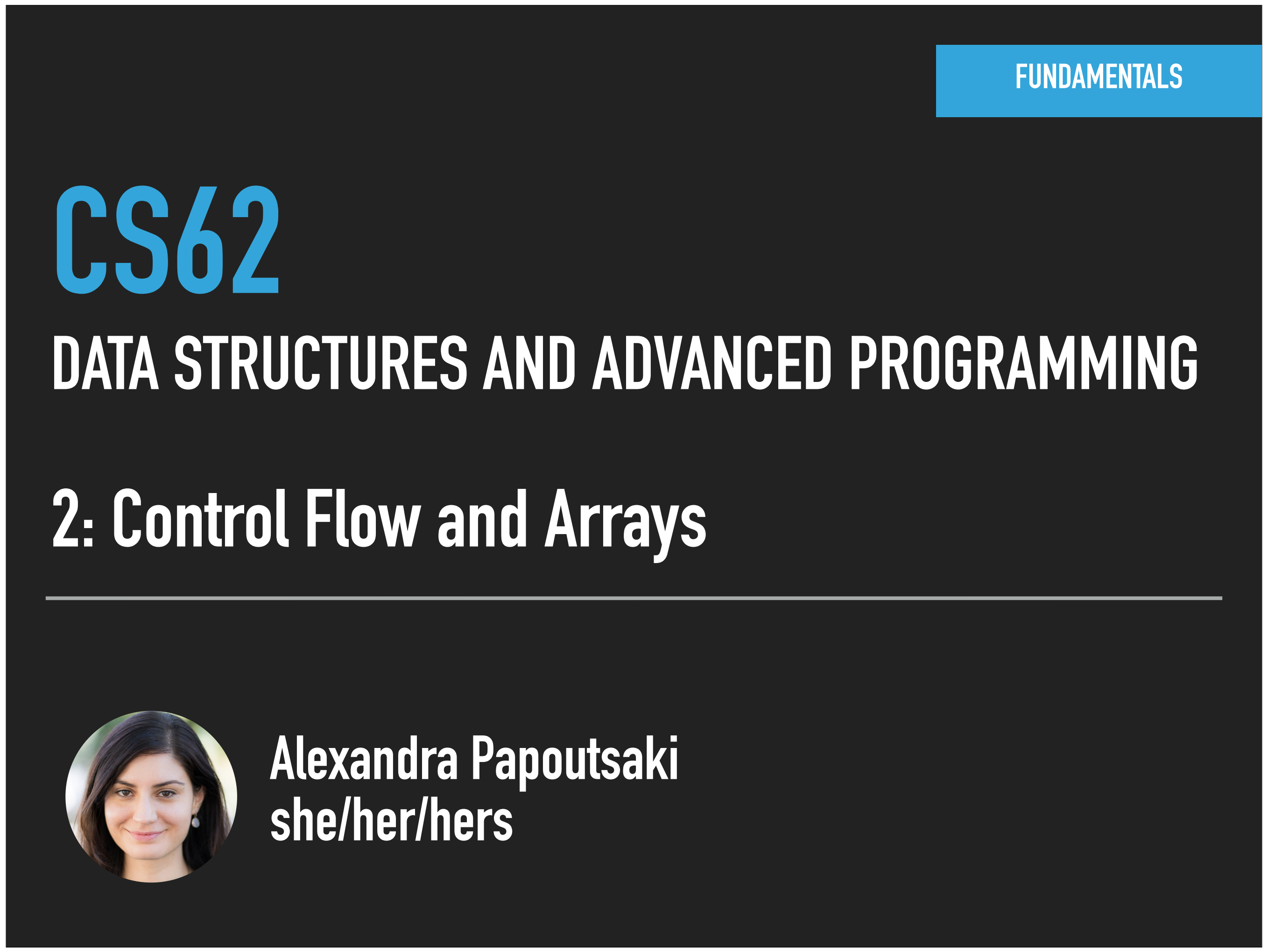


FUNDAMENTALS

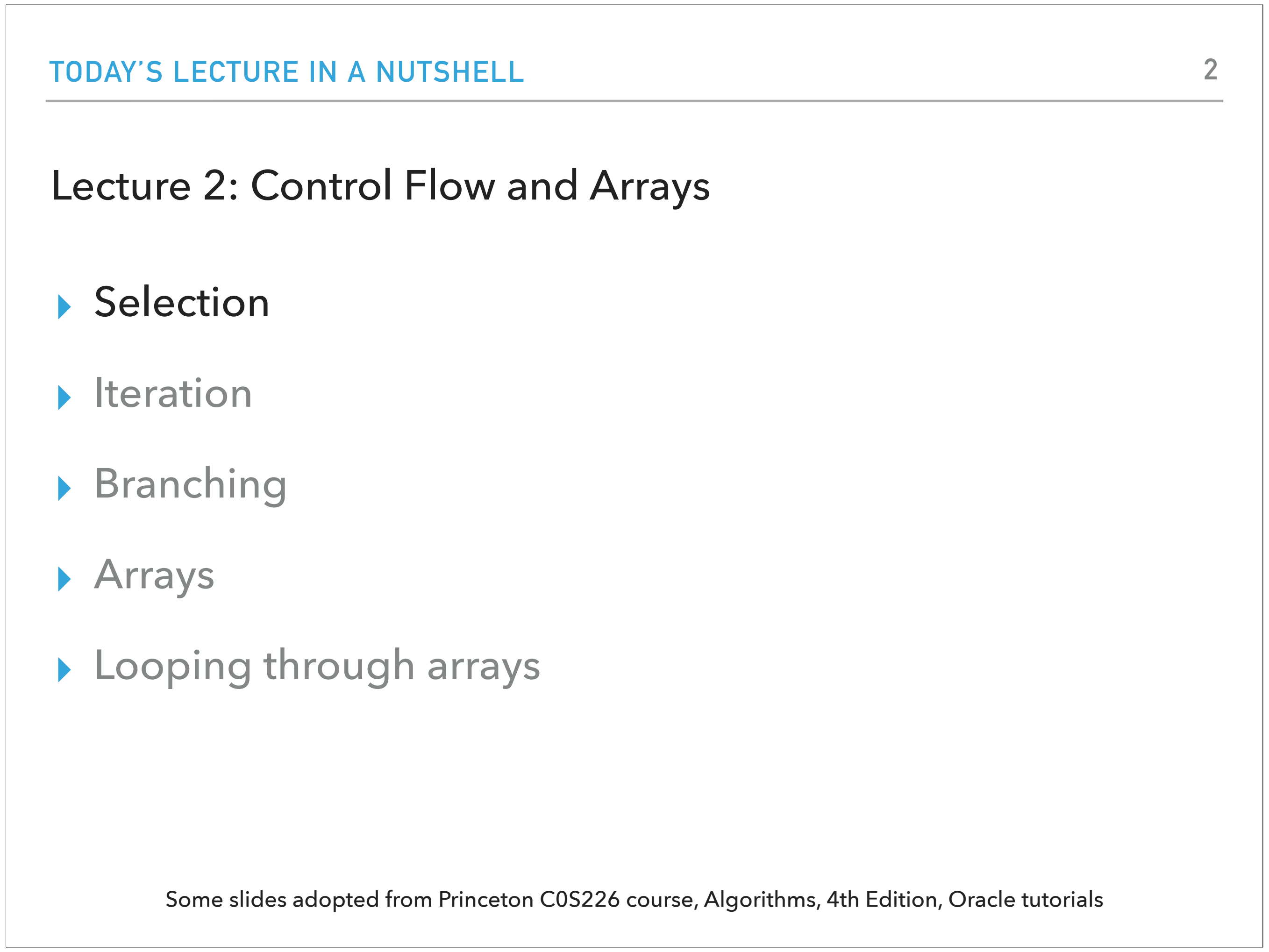
CS62

DATA STRUCTURES AND ADVANCED PROGRAMMING

2: Control Flow and Arrays

Alexandra Papoutsaki
she/her/hers

Today we will talk about control flow in our Java programs. The default mode of the order of execution of our programs is **sequential**, that is our code is executed line by line. Our class meeting will focus on two alternative control structures: **selection** and **iteration**, which break up the flow of execution by employing decision making, looping, and branching, enabling your program to conditionally execute particular blocks of code. You already have experience with both with if statements and for/while loops respectively. We'll spend our time seeing how to write these in Java and we will also encounter a few more options that it offers.



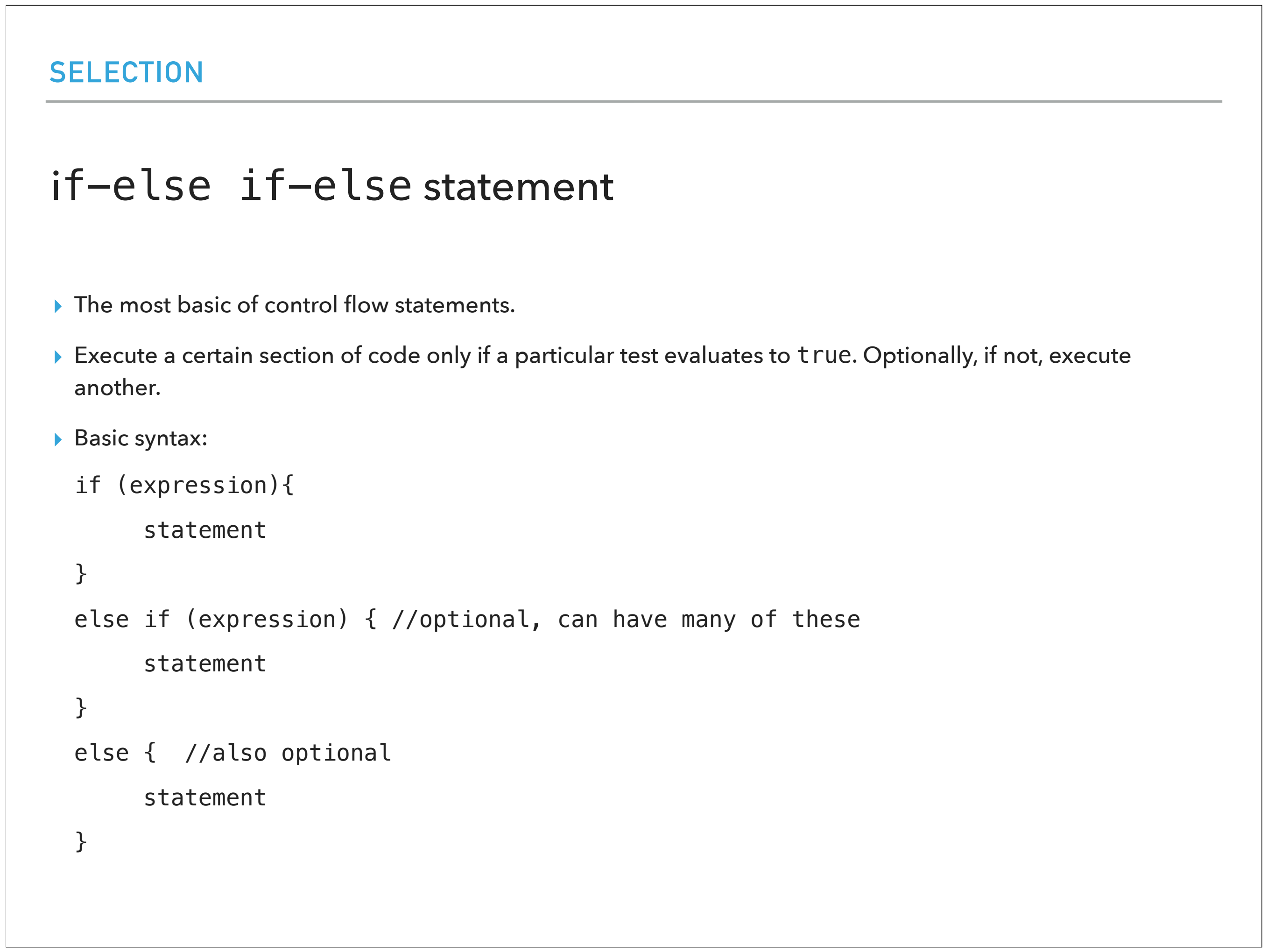
TODAY'S LECTURE IN A NUTSHELL

Lecture 2: Control Flow and Arrays

- Selection
- Iteration
- Branching
- Arrays
- Looping through arrays

Some slides adopted from Princeton C0S226 course, Algorithms, 4th Edition, Oracle tutorials

Let's start with selection control flow statements that will allow us to employ decision making.



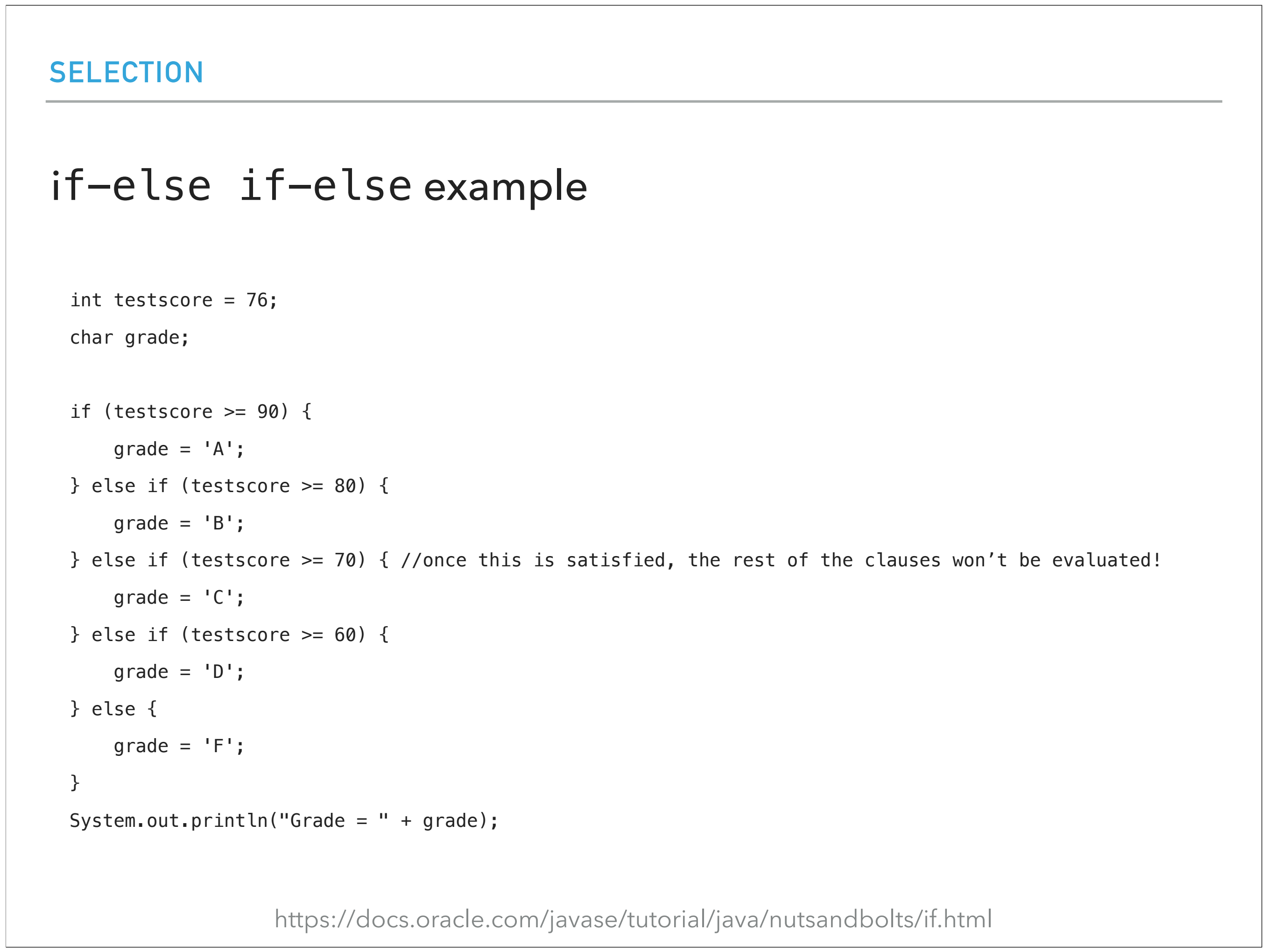
SELECTION

`if-else if-else` statement

- The most basic of control flow statements.
- Execute a certain section of code only if a particular test evaluates to `true`. Optionally, if not, execute another.
- Basic syntax:

```
if (expression){
     statement
}
else if (expression) { //optional, can have many of these
     statement
}
else {  //also optional
     statement
}
```

You are most likely already familiar with the If-else if-else statement. This is the most basic of all the control flow statements. It tells your program to execute a certain section of code only if a particular test evaluates to true. Optionally, you can include secondary paths of execution when an if clause evaluates to false. Don’t forget that the order of your if, else-if statements matters!



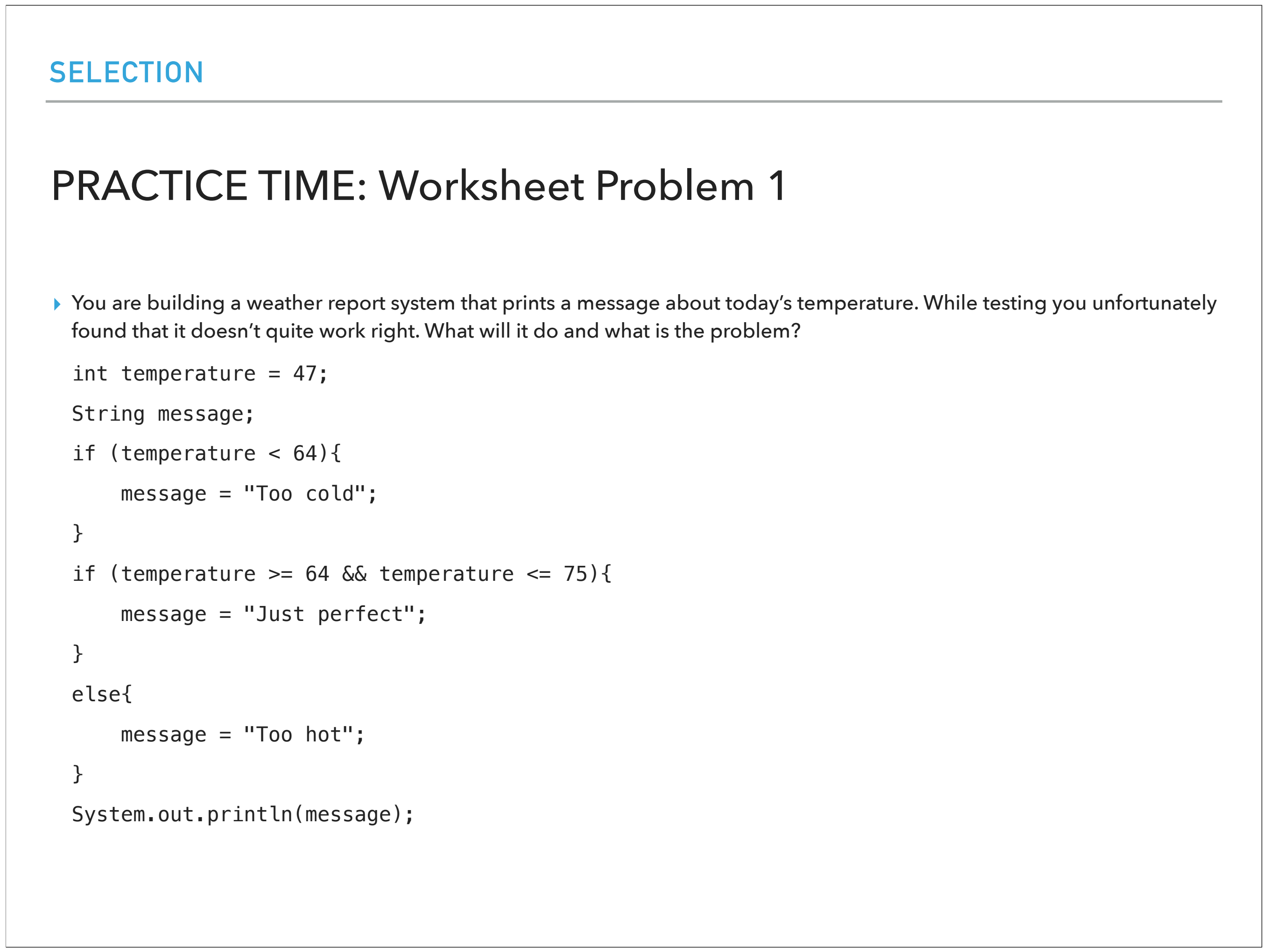
SELECTION

`if-else if-else` example

```
int testscore = 76;
char grade;

if (testscore >= 90) {
    grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) { //once this is satisfied, the rest of the clauses won’t be evaluated!
    grade = 'C';
} else if (testscore >= 60) {
    grade = 'D';
} else {
    grade = 'F';
}
System.out.println("Grade = " + grade);
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html

Here is an example of a program that given a testscore, in this case 76, it prints a messages with the final grade (no +/- in this system). The output from the program is:
Grade = C
You may have noticed that the value of testscore can satisfy more than one expression in the compound statement: 76 >= 70 and 76 >= 60. However, once a condition is satisfied, the appropriate statements are executed (grade = 'C';) and the remaining conditions are **not** evaluated.



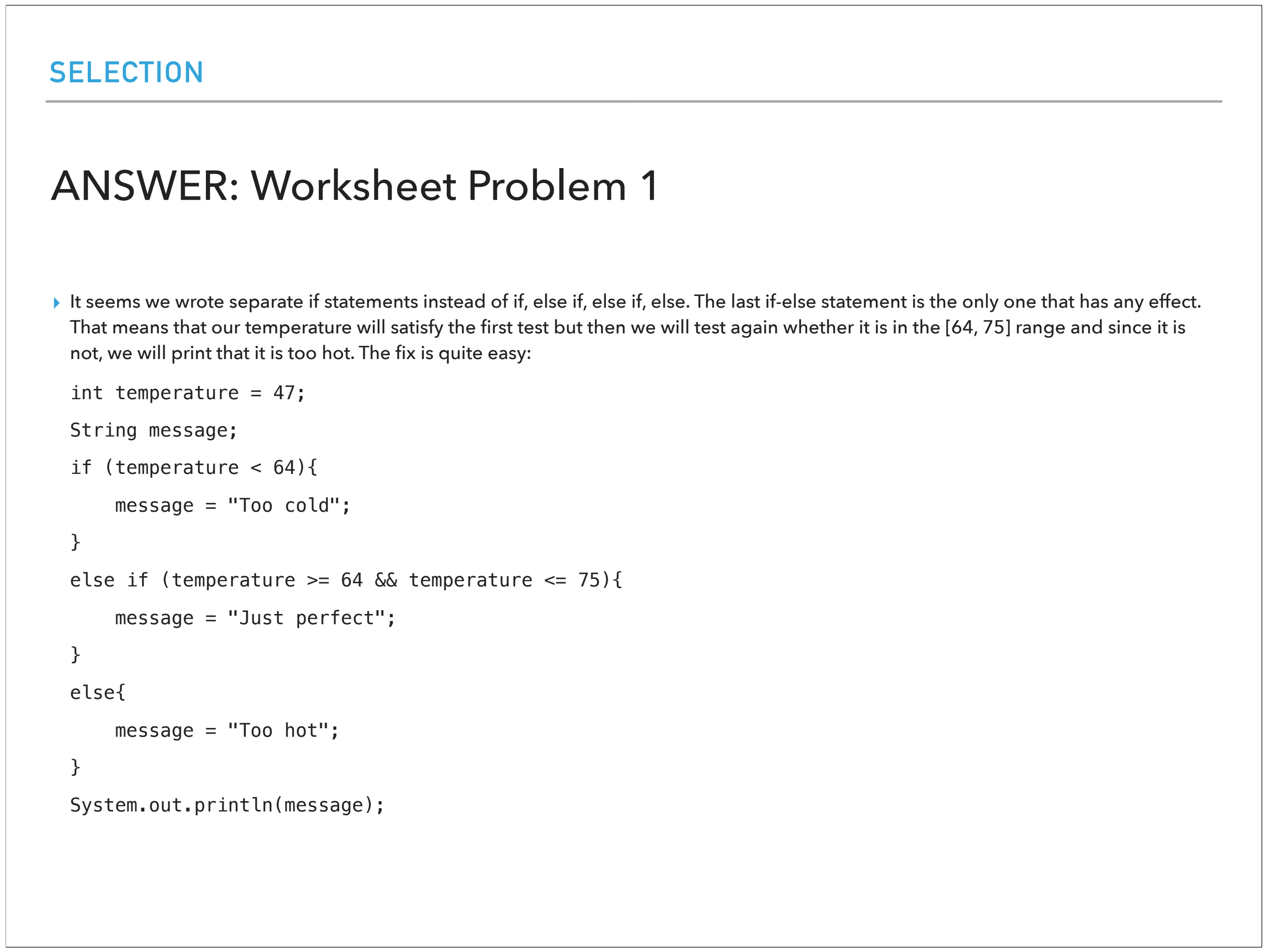
PRACTICE TIME: Worksheet Problem 1

- You are building a weather report system that prints a message about today's temperature. While testing you unfortunately found that it doesn't quite work right. What will it do and what is the problem?

```
int temperature = 47;
String message;
if (temperature < 64){
    message = "Too cold";
}
if (temperature >= 64 && temperature <= 75){
    message = "Just perfect";
}
else{
    message = "Too hot";
}
System.out.println(message);
```

Let's practice with if-else statements. You are building a weather report system that prints a message about today's temperature. While testing you unfortunately found that it doesn't quite work right. What will it do and what is the problem?

```
int temperature = 47;
String message;
if (temperature < 64){
   message = "Too cold";
}
if (temperature >= 64 && temperature <= 75){
   message = "Just perfect";
}
else{
   message = "Too hot";
}
System.out.println(message);
```



SELECTION

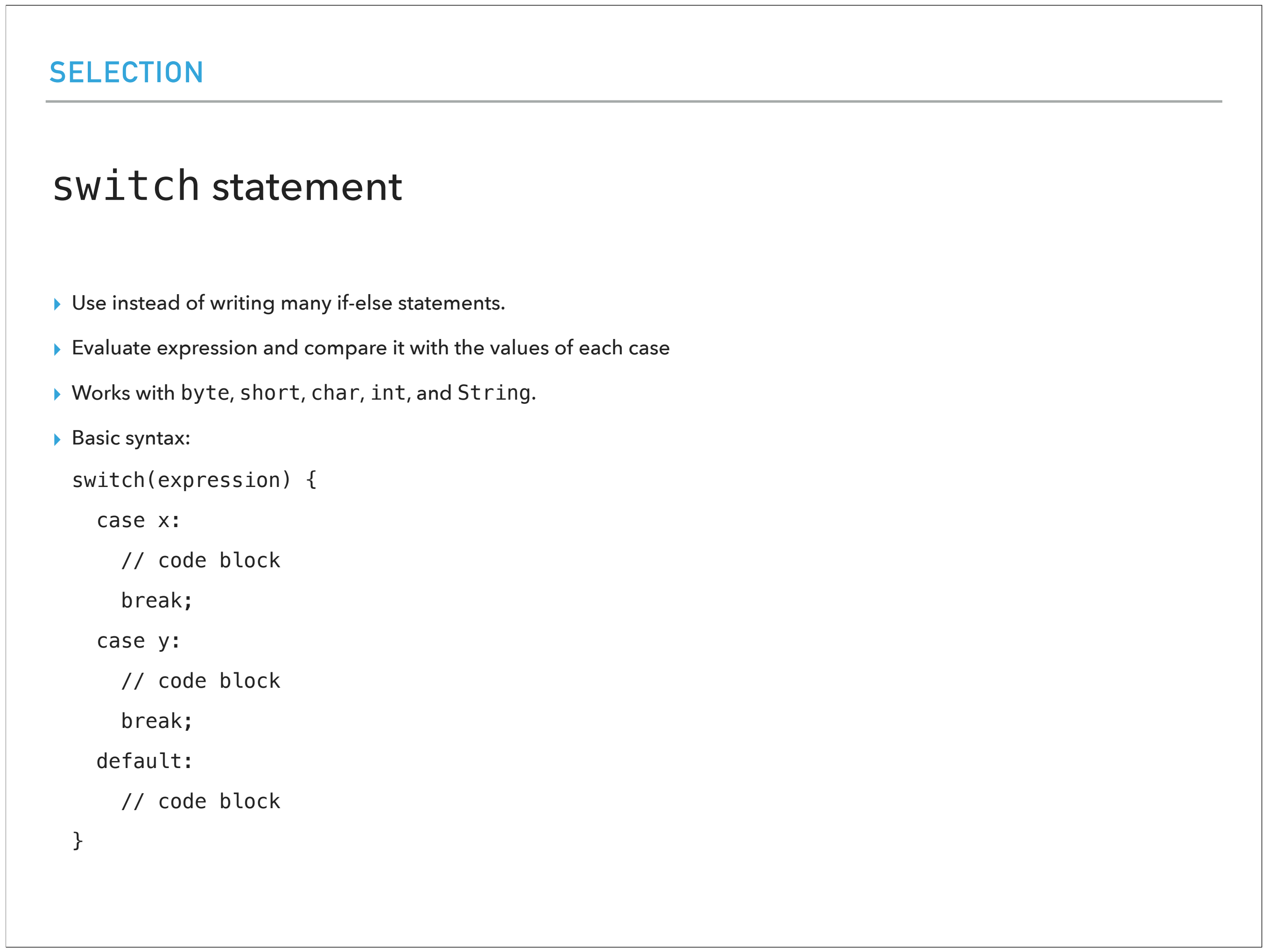
ANSWER: Worksheet Problem 1

- It seems we wrote separate if statements instead of if, else if, else if, else. The last if-else statement is the only one that has any effect. That means that our temperature will satisfy the first test but then we will test again whether it is in the [64, 75] range and since it is not, we will print that it is too hot. The fix is quite easy:

```
int temperature = 47;
String message;
if (temperature < 64){
    message = "Too cold";
}
else if (temperature >= 64 && temperature <= 75){
    message = "Just perfect";
}
else{
    message = "Too hot";
}
System.out.println(message);
```

It seems we wrote separate if statements instead of if, else if, else if, else. The last if-else statement is the only one that has any effect. That means that our temperature will satisfy the first test but then we will test again whether it is in the [64, 75] range and since it is not, we will print that it is too hot. The fix is quite easy:

```
int temperature = 47;
String message;
if (temperature < 64){
    message = "Too cold";
}
else if (temperature >= 64 && temperature <= 75){
    message = "Just perfect";
}
else{
    message = "Too hot";
}
System.out.println(message);
```



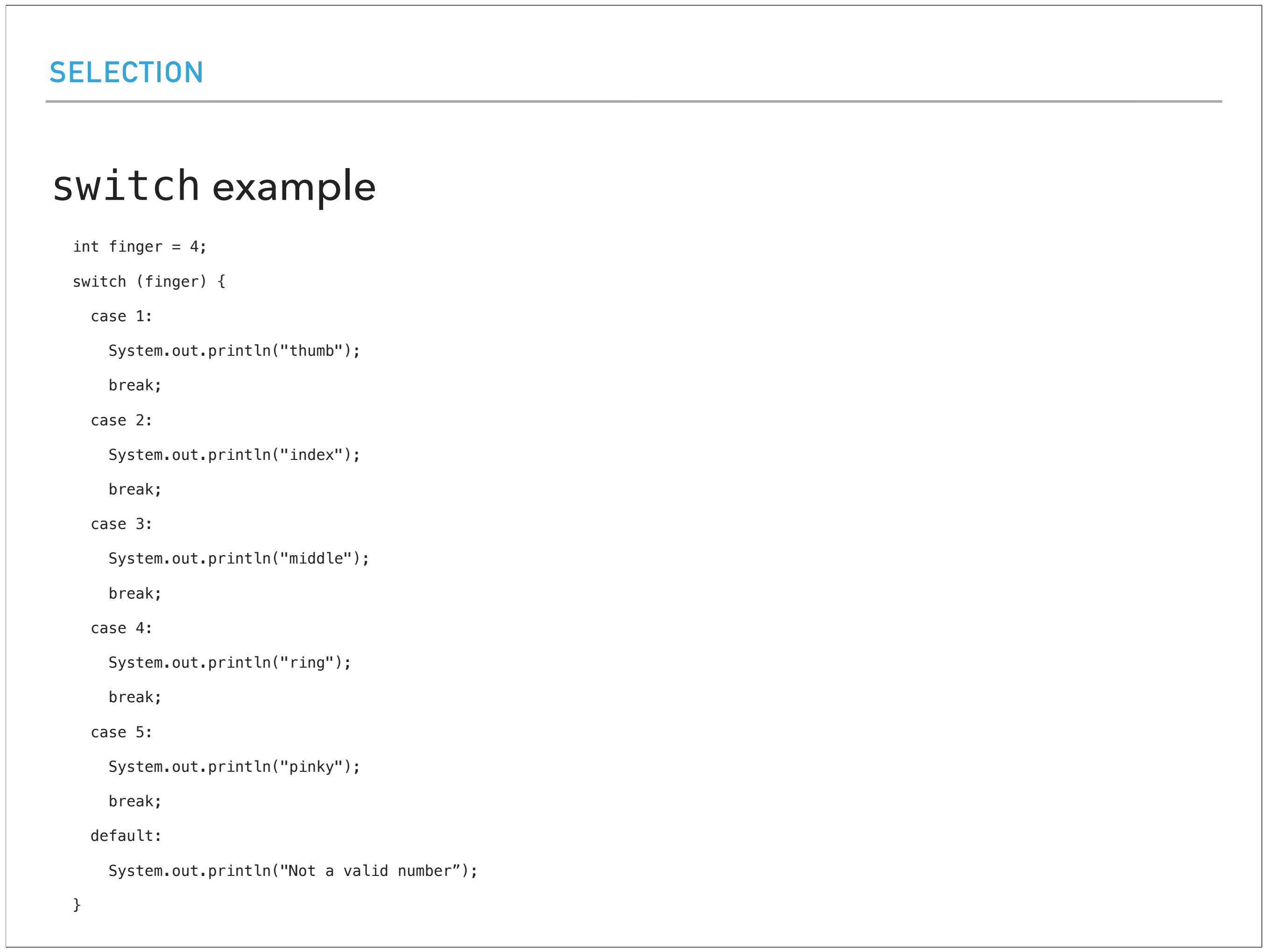
SELECTION

`switch` statement

- Use instead of writing many if-else statements.
- Evaluate expression and compare it with the values of each case
- Works with `byte`, `short`, `char`, `int`, and `String`.
- Basic syntax:

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Java has one more selection statement to control the flow of the program: switch. The switch statement can have a number of possible execution paths. A switch works with the byte, short, char, int primitive data types, and Strings. The idea is that we evaluate the expression within switch(expression) the value of the expression is compared with the values of each case. If there is a match, the associated block of code is executed. The break and default keywords are optional, and will talk about them later.


SELECTION

switch example

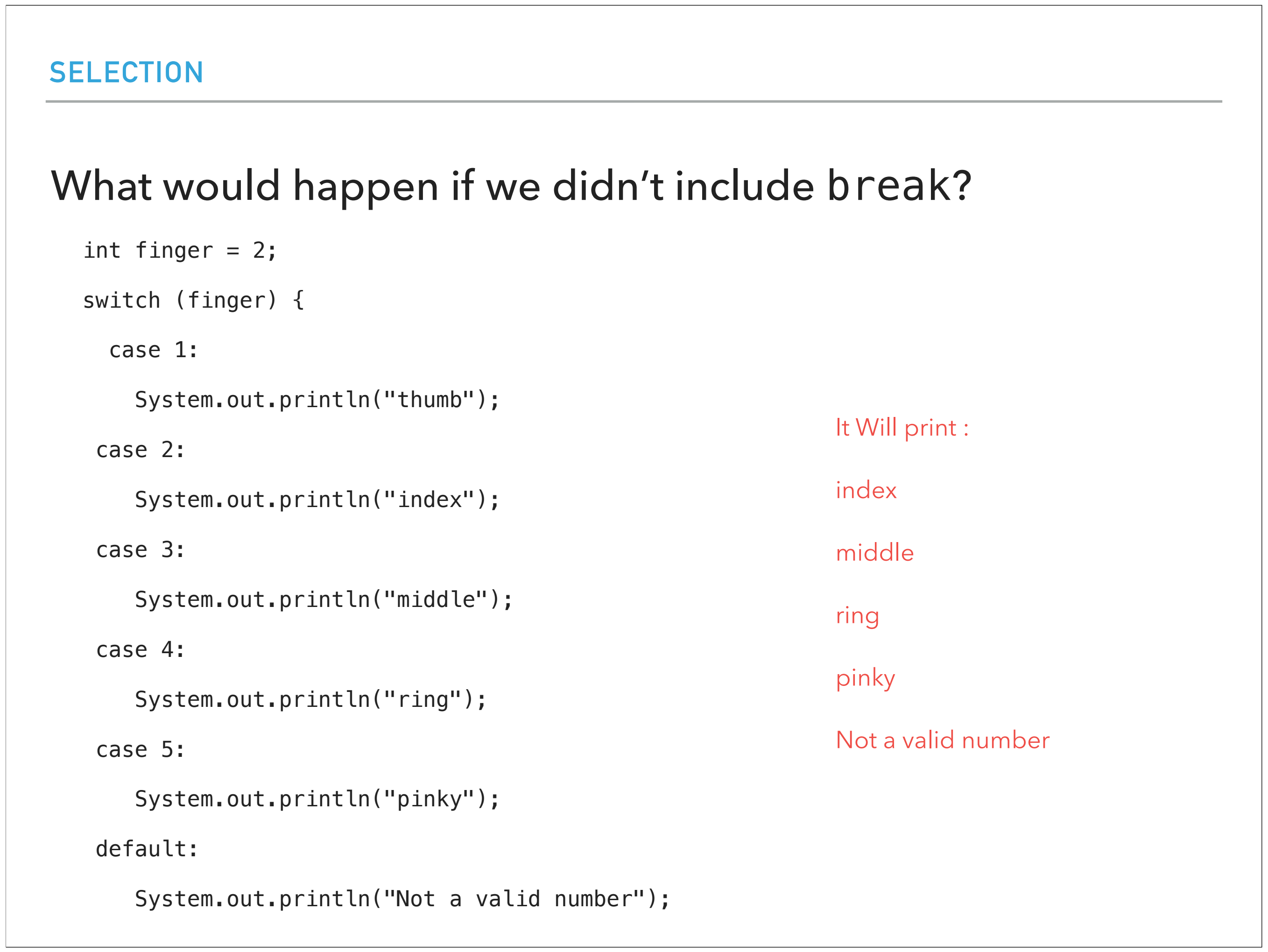
```
int finger = 4;
switch (finger) {
  case 1:
    System.out.println("thumb");
    break;
  case 2:
    System.out.println("index");
    break;
  case 3:
    System.out.println("middle");
    break;
  case 4:
    System.out.println("ring");
    break;
  case 5:
    System.out.println("pinky");
    break;
  default:
    System.out.println("Not a valid number”);
}
```

Here is an example of a program that uses a switch to assign a number 1…5 to one of the thumb, index, middle, ring, and pinky finger options.

`break` and `default`

- When Java reaches a `break` keyword, it breaks out of the switch block and does not execute the rest of the code.
    - You need to add a `break` statement otherwise you will go through all the remaining cases!
- The `default` keyword specifies what code to run if there is no case match.

When Java reaches a break keyword, it breaks out of the switch block and does not execute the rest of the code. You need to add a break statement otherwise the remaining cases will be evaluated!
The default keyword specifies what code to run if there is no case match. In this example, we have used the default case to address a finger number outside the valid range of 1-5.



SELECTION

What would happen if we didn't include break?

```
int finger = 2;
switch (finger) {
  case 1:
    System.out.println("thumb");
 case 2:
    System.out.println("index");
 case 3:
    System.out.println("middle");
 case 4:
    System.out.println("ring");
 case 5:
    System.out.println("pinky");
 default:
    System.out.println("Not a valid number");
```

It Will print :

index

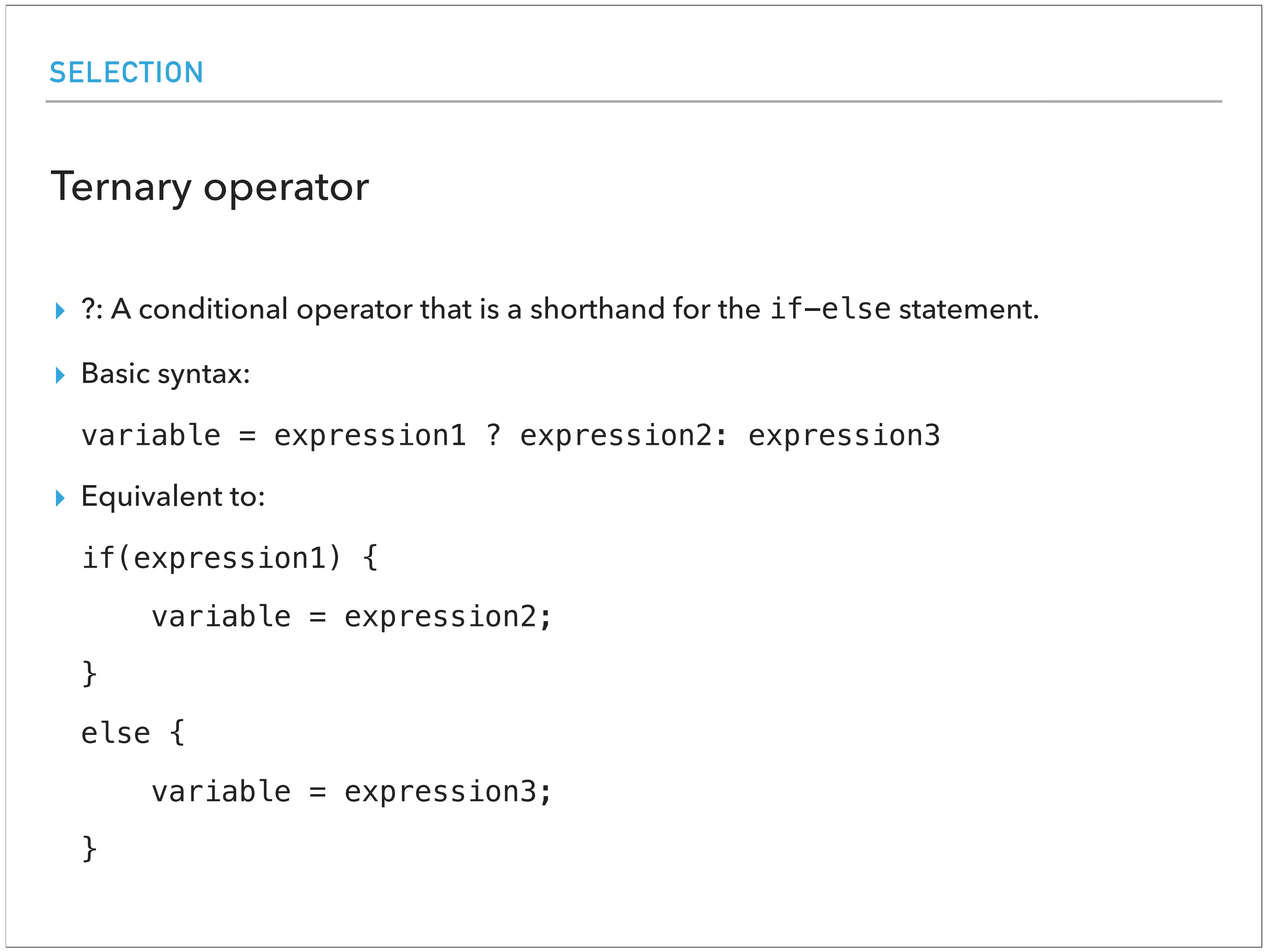
middle

ring

pinky

Not a valid number

Here is an example of the same program not including break statements. In this case, although our variable finger is 2, the program Will print:
index
middle
ring
pinky
Not a valid number
Definitely not what we intended!



SELECTION

Ternary operator

- ?: A conditional operator that is a shorthand for the `if-else` statement.
- Basic syntax:

```
variable = expression1 ? expression2: expression3
```

- Equivalent to:

```
if(expression1) {
    variable = expression2;
}
else {
    variable = expression3;
}
```

The last thing we will see about selection statements to control the flow of our program is the ternary operator ?: We will use it as a shorthand for the if-else statement.
The basic syntax is
variable = expression1 ? expression2: expression3
And it is equivalent to:

```
if(expression1) {
    variable = expression2;
}
else {
    variable = expression3;
}
```

Ternary operator example

```
int n1 = 32;
int n2 = 47;
int max;

// Largest among n1 and n2
max = (n1 > n2) ? n1 : n2;

// Print the largest number
System.out.println("Maximum is = " + max);
```

Here is a simple example of how to use the ternary operator to check which number is larger.

PRACTICE TIME - Worksheet Problem 2

- What will the following program print?

```
int n1 = 32;

int n2 = 47;

System.out.println((n1>n2) ? (n1+n2):(n1-n2));
```

Let's see if the ternary operator makes sense by practicing with this simple program.

ANSWER - Worksheet Problem 2

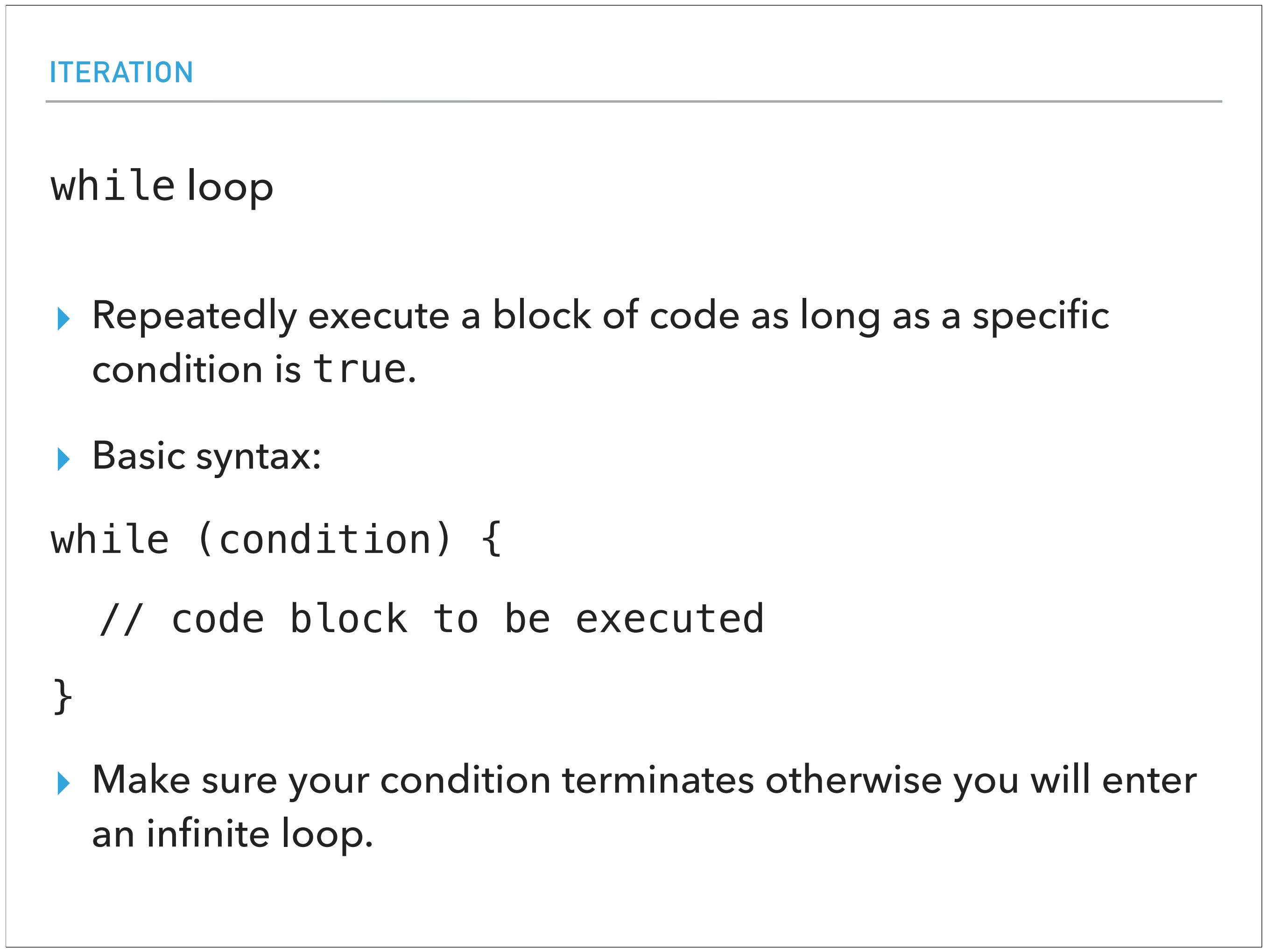
- Since the `n1>n2` expression evaluates to `false`, it will print −15 (i.e. `n1–n2`)

Since the n1>n2 expression evaluates to false, it will print -15 (i.e. n1-n2)

Lecture 2: Control Flow and Arrays

- Selection
- Iteration
- Branching
- Arrays
- Looping through arrays

Let's now see how we can control the flow of our programs with repetition or iteration, starting with the while loop.



ITERATION

while loop

- Repeatedly execute a block of code as long as a specific condition is `true`.
- Basic syntax:

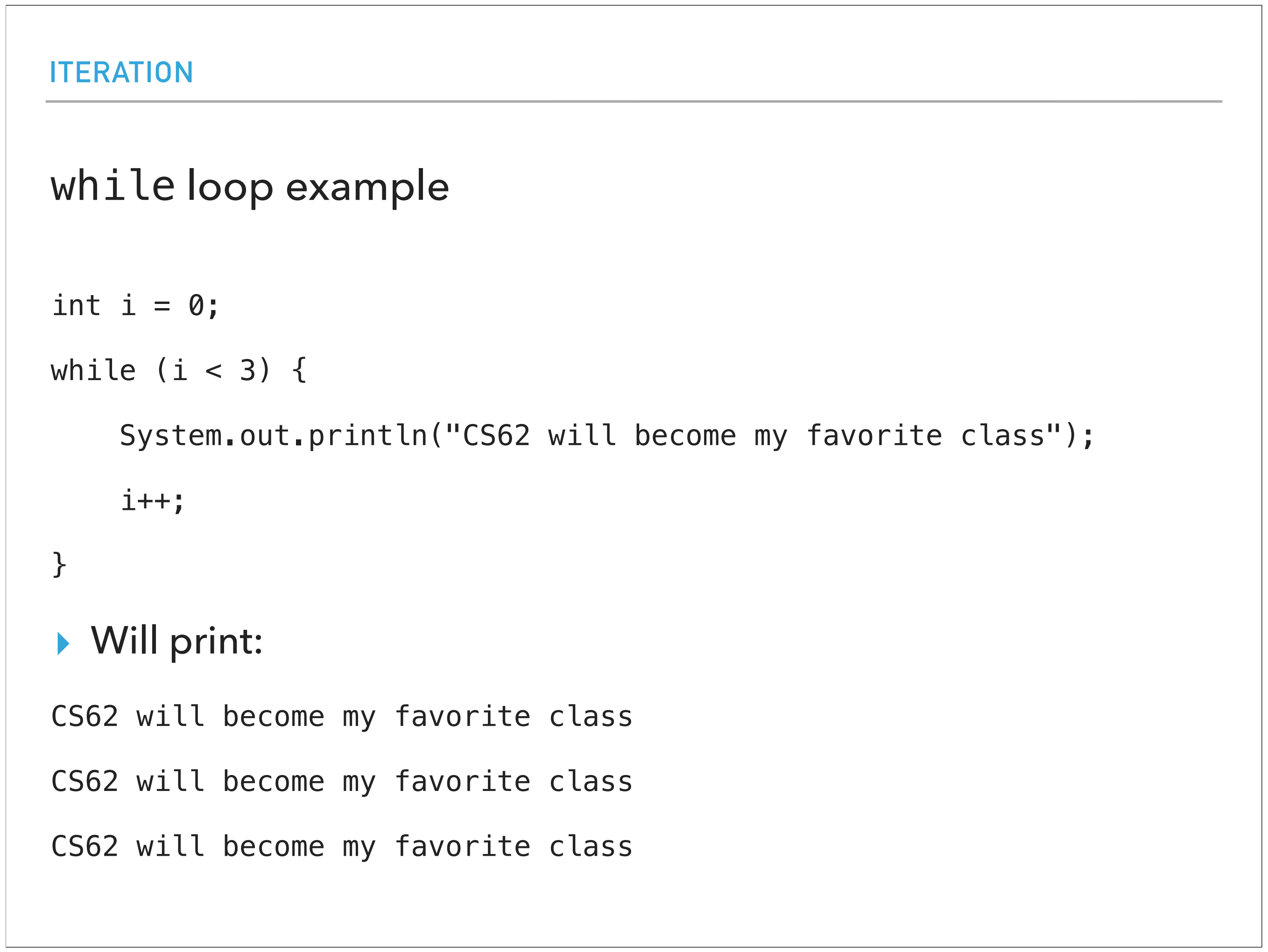
```
while (condition) {
  // code block to be executed
}
```

- Make sure your condition terminates otherwise you will enter an infinite loop.

The while loop enables us to repeatedly execute a block of code as long as a specific condition is true. Basic syntax:

```
while (condition) {
  // code block to be executed
}
```

Make sure your condition terminates otherwise you will enter an infinite loop.



`while` loop example

```
int i = 0;
while (i < 3) {
    System.out.println("CS62 will become my favorite class");
    i++;
}
```

- Will print:

```
CS62 will become my favorite class
CS62 will become my favorite class
CS62 will become my favorite class
```

Here is an example of a small program that prints the same message three times.

```
int i = 0;
while (i < 3) {
   System.out.println("CS62 will become my favorite class");
   i++;
}
```

Will print:

```
CS62 will become my favorite class
CS62 will become my favorite class
CS62 will become my favorite class
```

`do-while` loop

- Variant of `while` loop that will execute the block of code once, before it checks if the condition is `true`. It will then proceed as usual.
- Basic syntax:

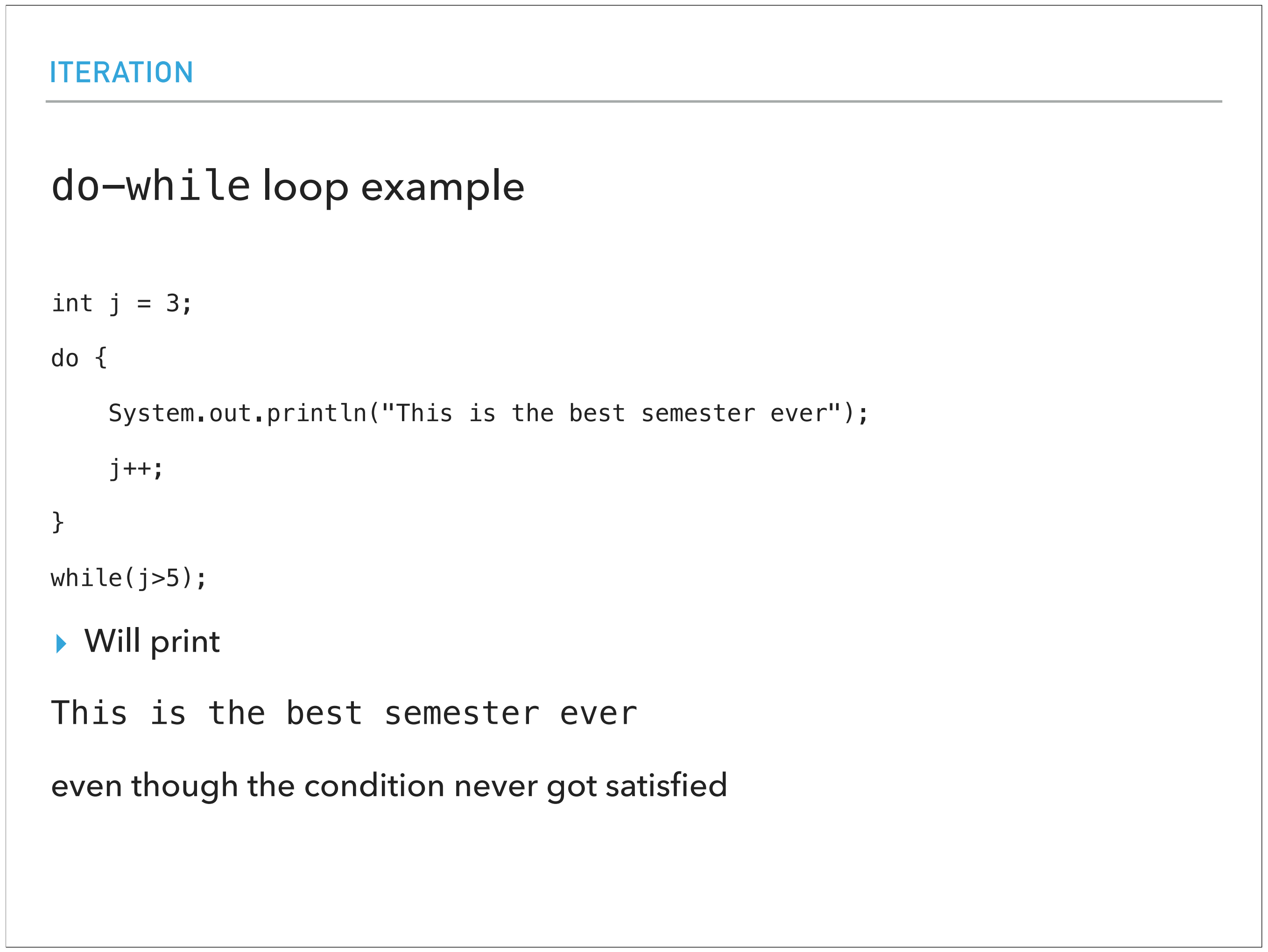
```
do {
  // code block to be executed
} while(condition);
```

- Make sure your condition terminates otherwise you will enter an infinite loop.

Java supports a variation of the while loop called the do-while loop which executes the block of code once, before it checks if the condition is true. It will then proceed as usual. Basic syntax:

```
do {
  // code block to be executed
} while(condition);
```

Make sure your condition terminates otherwise you will enter an infinite loop and don’t forget the semicolon at the end!



do-while loop example

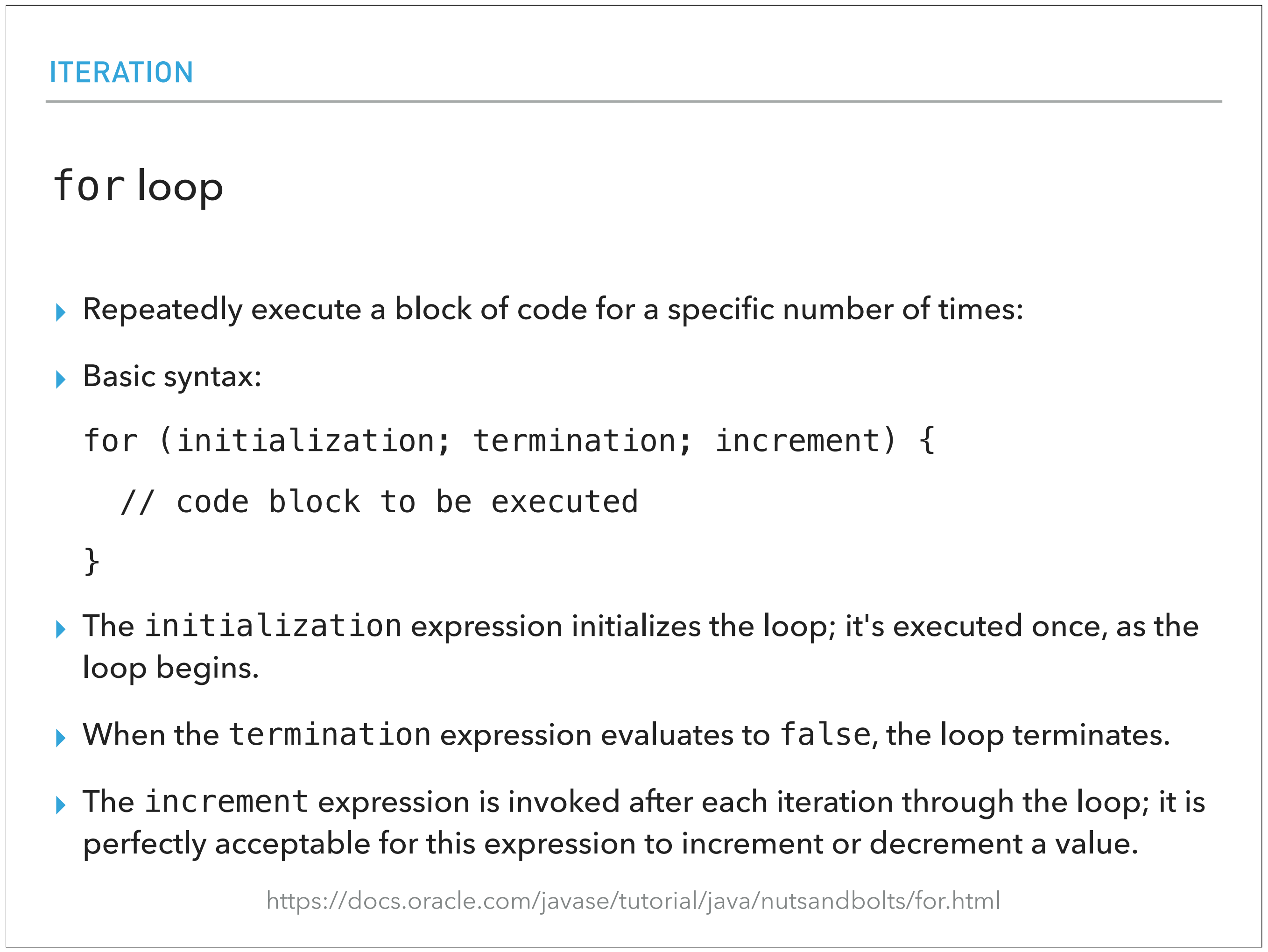
```
int j = 3;
do {
    System.out.println("This is the best semester ever");
    j++;
}
while(j>5);
```

- Will print

```
This is the best semester ever
```

even though the condition never got satisfied

In this example, the message will be printed once, even though the condition never got satisfied.



for loop

- Repeatedly execute a block of code for a specific number of times:
- Basic syntax:

```
for (initialization; termination; increment) {
  // code block to be executed
}
```

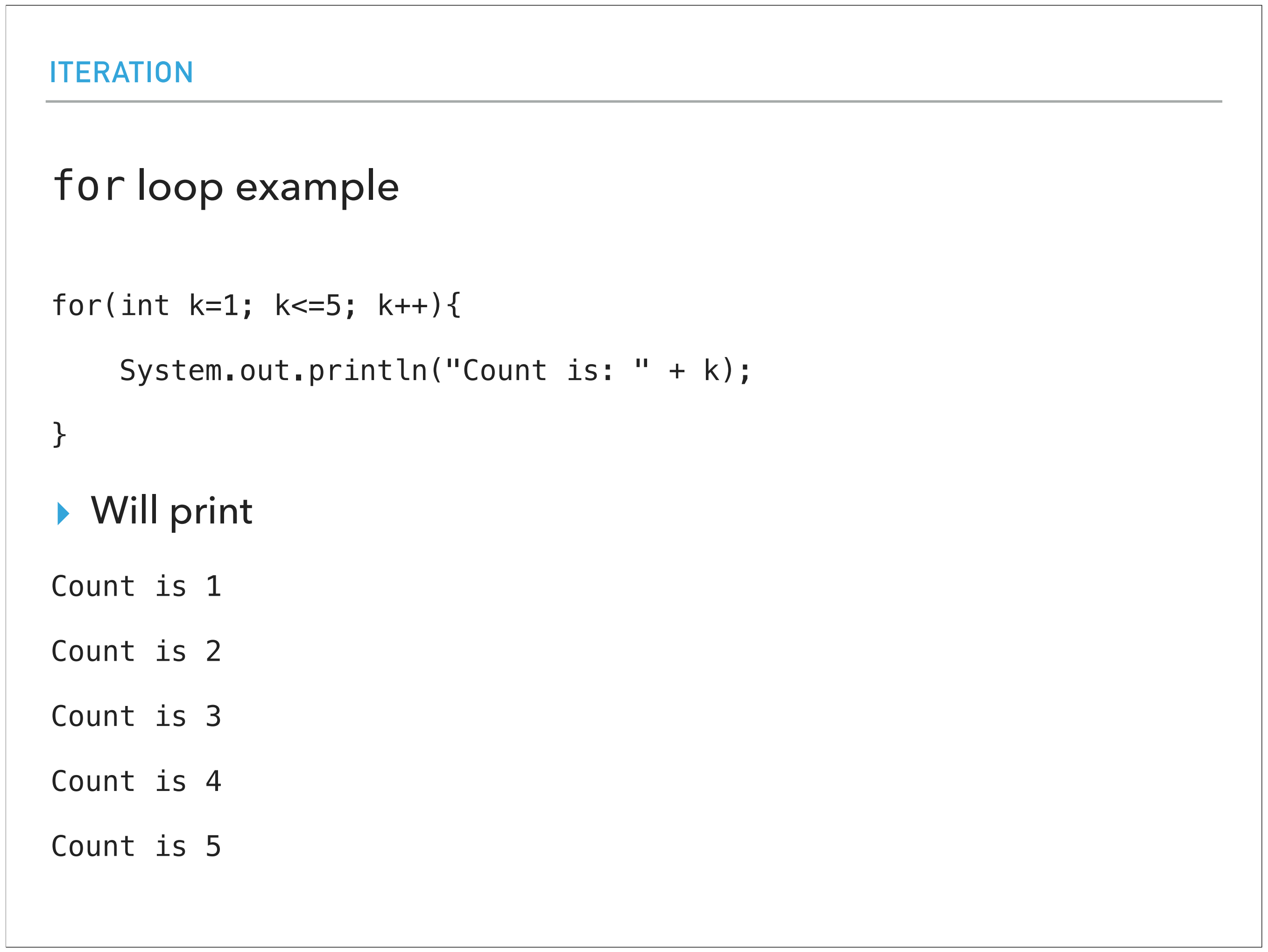
- The `initialization` expression initializes the loop; it's executed once, as the loop begins.
- When the `termination` expression evaluates to `false`, the loop terminates.
- The `increment` expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

Let's now talk about for loops, another iteration mechanism that we will use when we know how many times we want to execute a block of code. The general syntax is:

```
for (initialization; termination; increment) {
  // code block to be executed
}
```

The initialization expression initializes the loop; it's executed once, as the loop begins.
When the termination expression evaluates to false, the loop terminates.
The increment expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment or decrement a value.
Note that semicolons are only used after the first two expressions only.



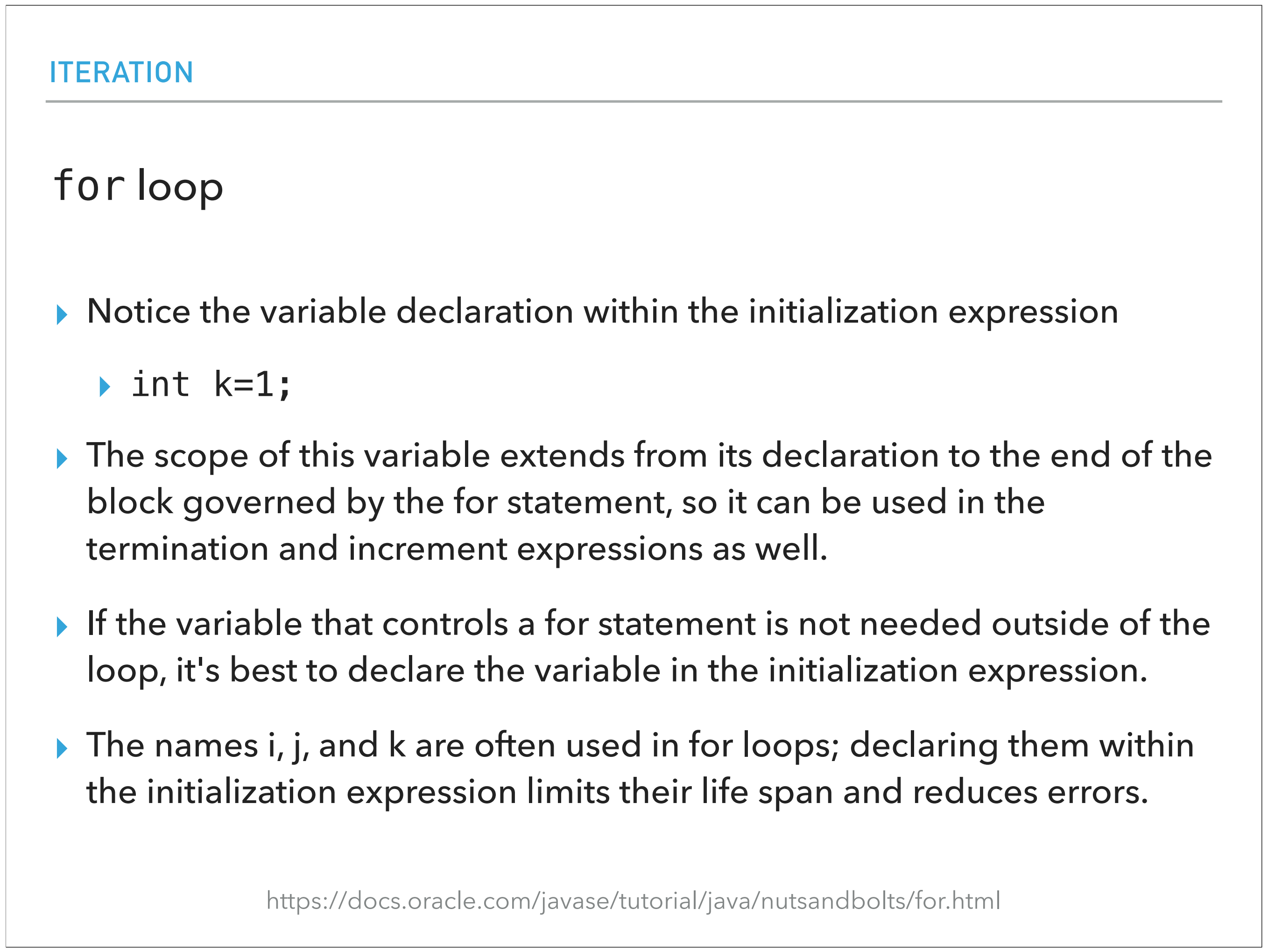
`for` loop example

```
for(int k=1; k<=5; k++){

    System.out.println("Count is: " + k);

}
```

- Will print

```
Count is 1

Count is 2

Count is 3

Count is 4

Count is 5
```

In this example, the message will be printed 5 times.

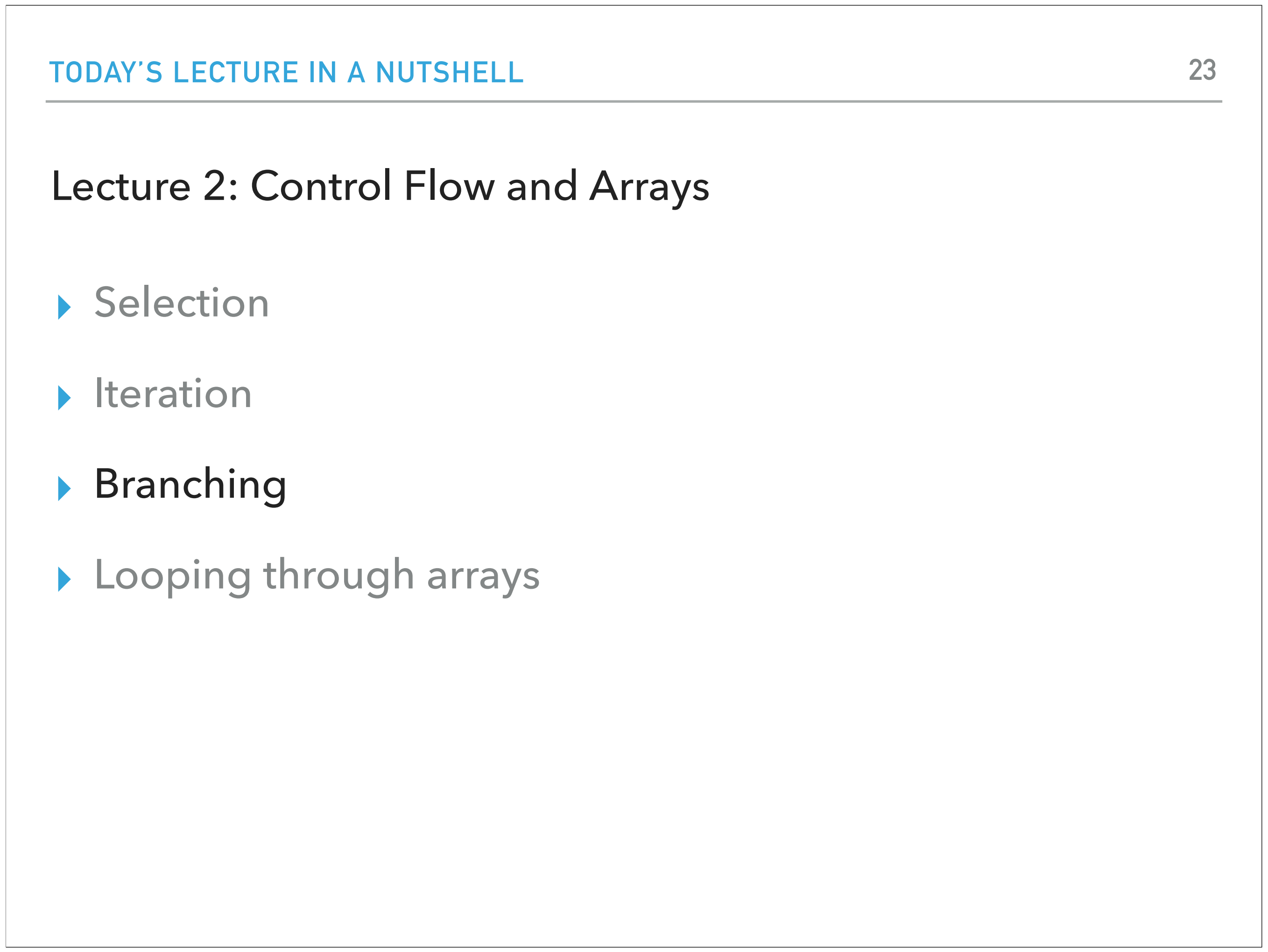


`for` loop

- Notice the variable declaration within the initialization expression
  - `int k=1;`
- The scope of this variable extends from its declaration to the end of the block governed by the for statement, so it can be used in the termination and increment expressions as well.
- If the variable that controls a for statement is not needed outside of the loop, it's best to declare the variable in the initialization expression.
- The names i, j, and k are often used in for loops; declaring them within the initialization expression limits their life span and reduces errors.

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

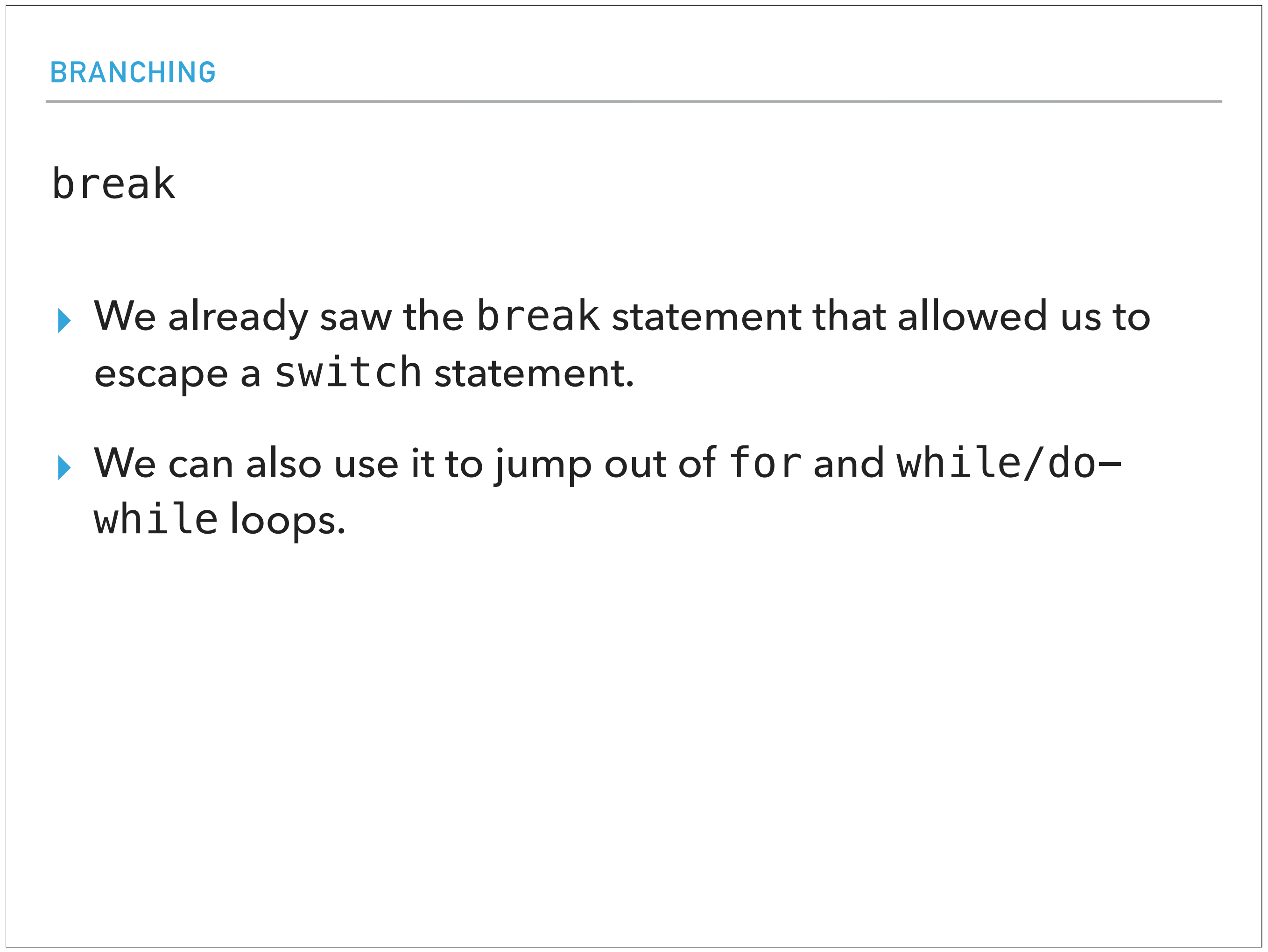
Notice how the code declares a variable within the initialization expression. The scope of this variable extends from its declaration to the end of the block governed by the for statement, so it can be used in the termination and increment expressions as well. If the variable that controls a for statement is not needed outside of the loop, it's best to declare the variable in the initialization expression. The names i, j, and k are often used to control for loops; declaring them within the initialization expression limits their life span and reduces errors.



Lecture 2: Control Flow and Arrays

- Selection
- Iteration
- Branching
- Looping through arrays

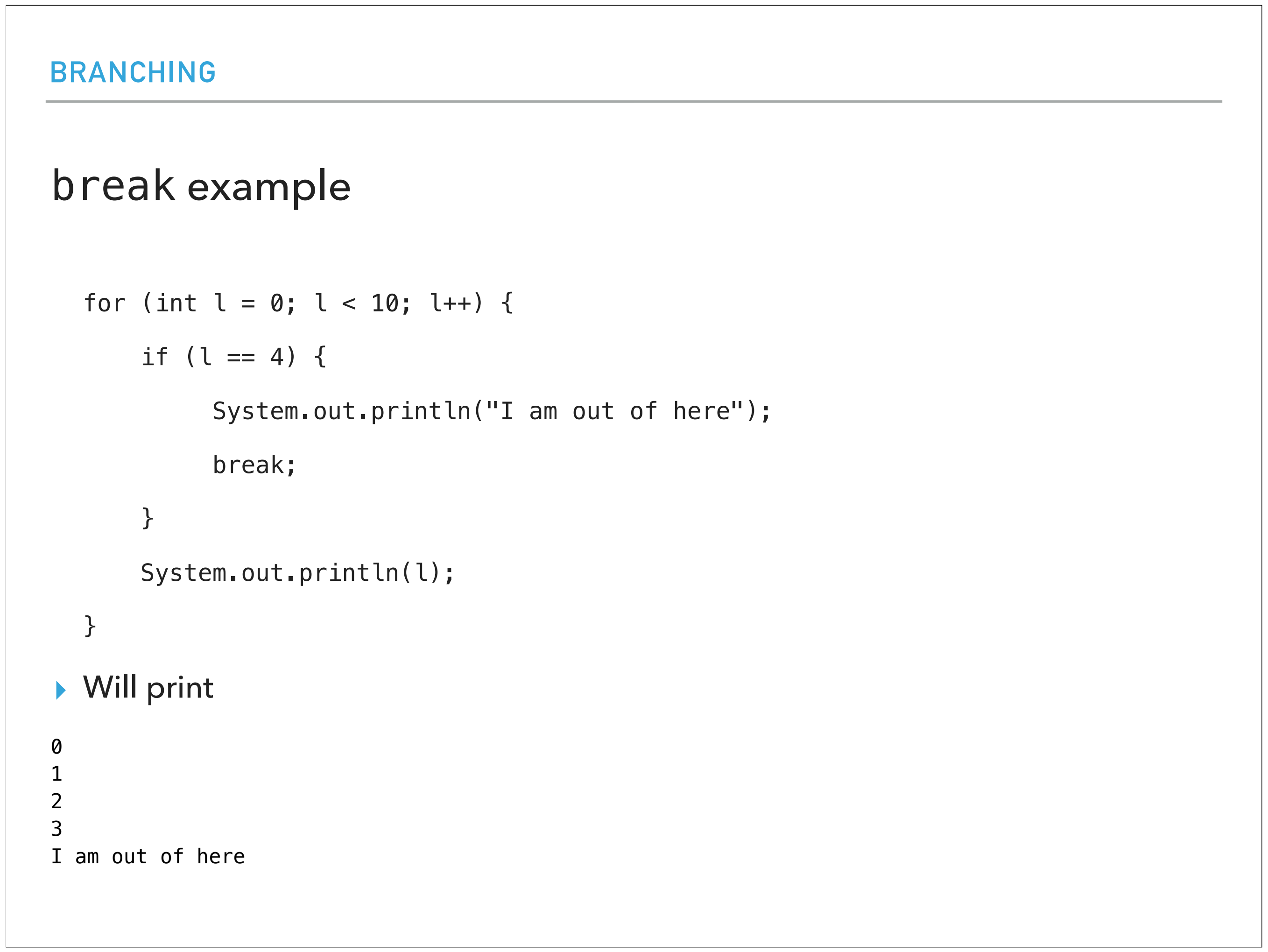
Let's talk now about two types of branching statements.


BRANCHING

`break`

- We already saw the `break` statement that allowed us to escape a `switch` statement.
- We can also use it to jump out of `for` and `while`/`do-while` loops.

The first one, break, should already be familiar; we saw that we can use it to escape a switch statement. But we can also use it to jump out of a for or a while loop.



break example

```
for (int l = 0; l < 10; l++) {
    if (l == 4) {
         System.out.println("I am out of here");
         break;
    }
    System.out.println(l);
}
```

- Will print

```
0
1
2
3
I am out of here
```

This program will print:

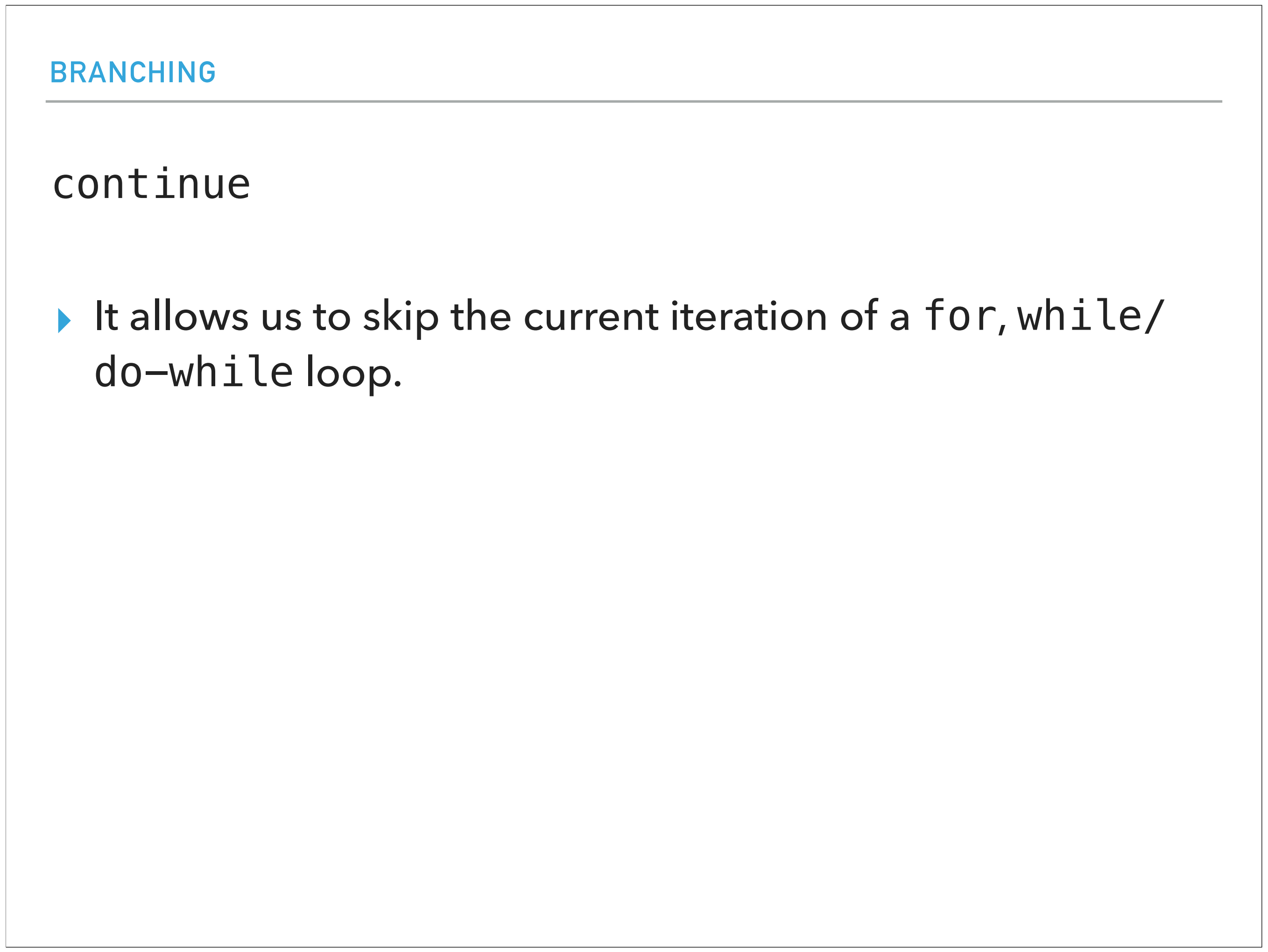
0

1

2

3

I am out of here

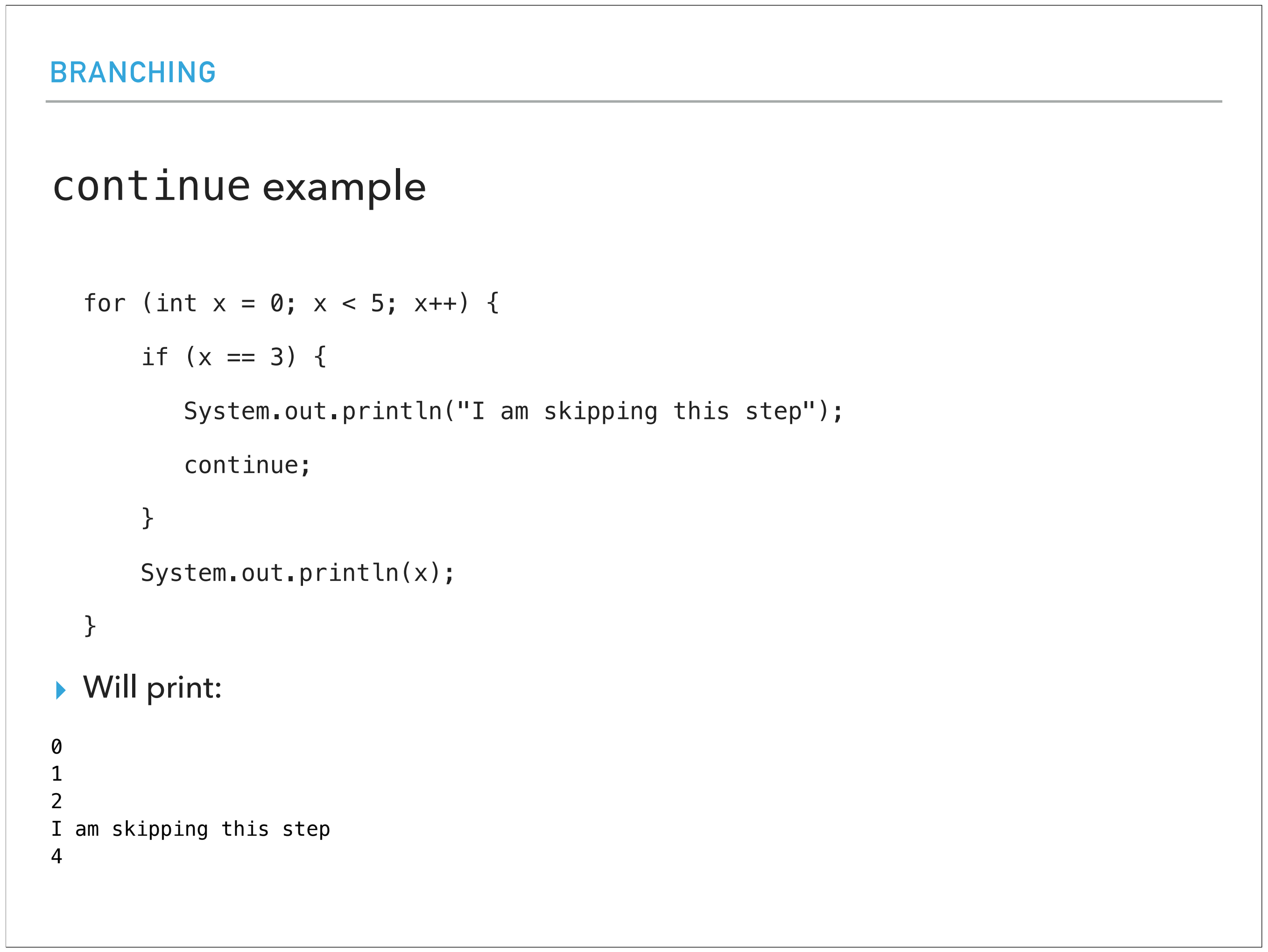


BRANCHING

`continue`

- It allows us to skip the current iteration of a `for`, `while`/`do-while` loop.

The second branching statement, continue, enables us to skip one iteration if a certain condition is met. We can use it again both for for loops and while loops.



`continue` example

```
for (int x = 0; x < 5; x++) {
    if (x == 3) {
       System.out.println("I am skipping this step");
       continue;
    }
    System.out.println(x);
}
```

- Will print:

```
0
1
2
I am skipping this step
4
```

This program will print:

0

1

2

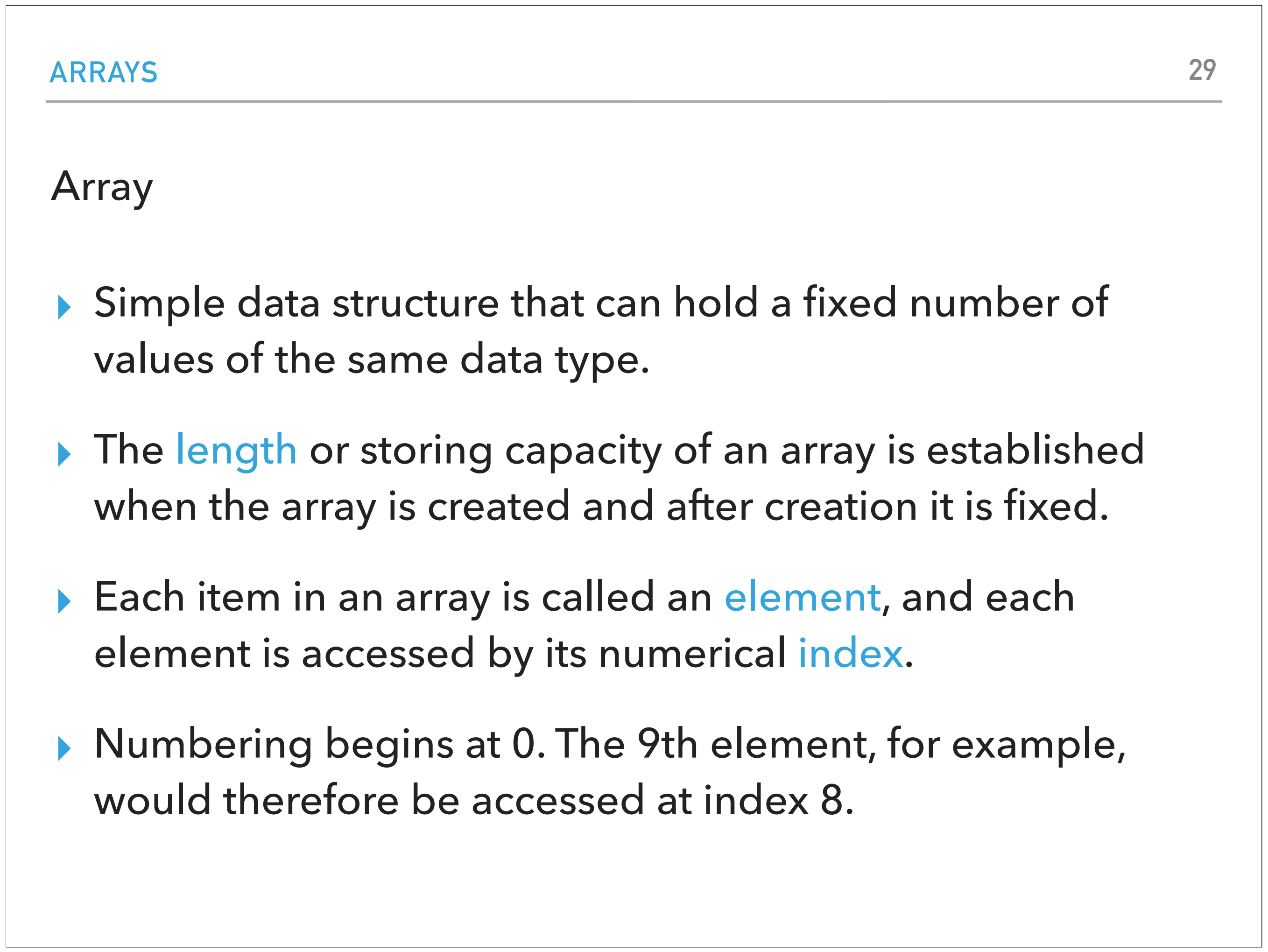
I am skipping this step

4

Lecture 2: Control Flow and Arrays

- Selection
- Iteration
- Branching
- Arrays
- Looping through arrays

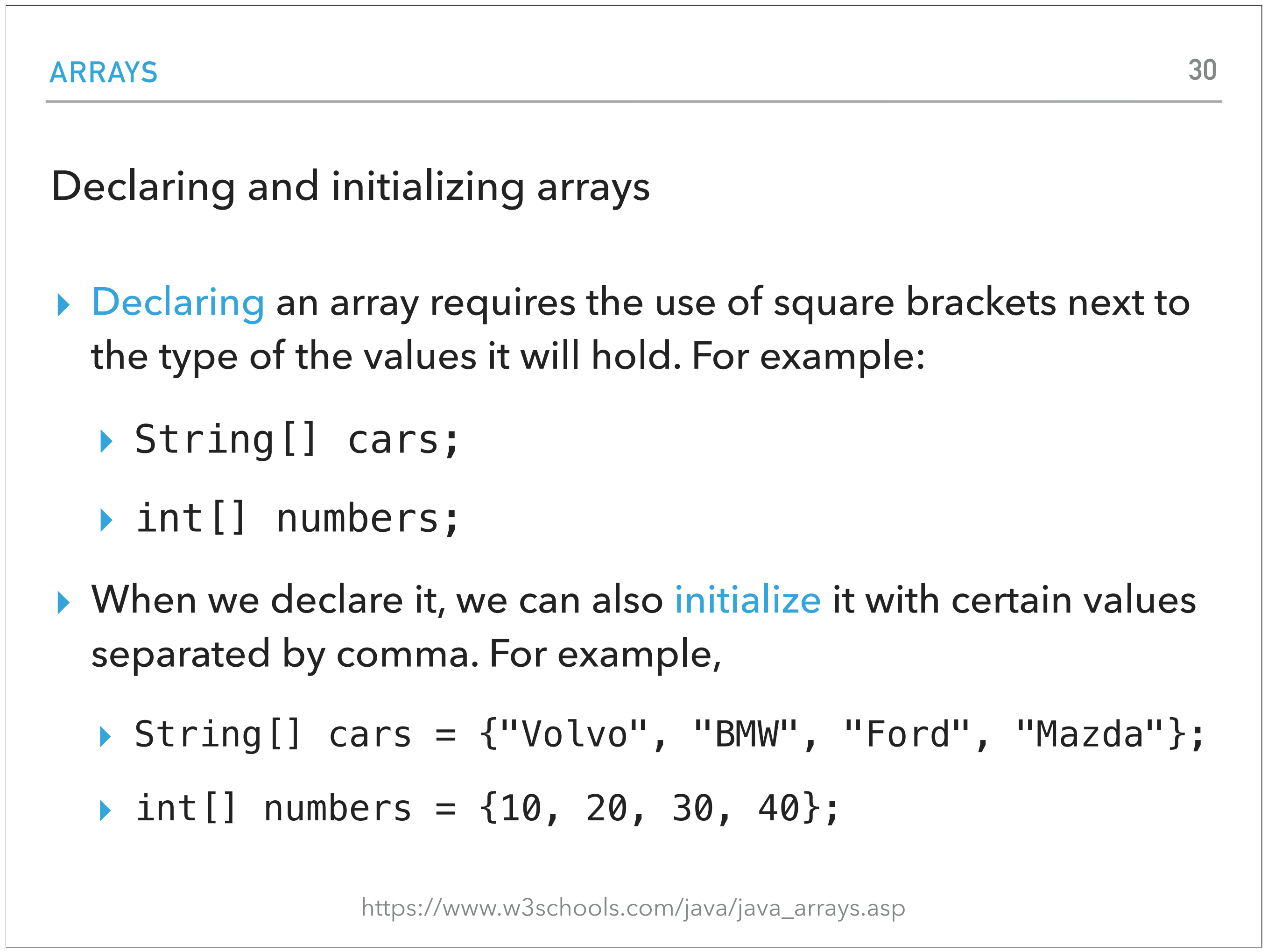
Now that we covered the basics of control flow, let's see our very first data structure, the array!



Array

- Simple data structure that can hold a fixed number of values of the same data type.
- The length or storing capacity of an array is established when the array is created and after creation it is fixed.
- Each item in an array is called an element, and each element is accessed by its numerical index.
- Numbering begins at 0. The 9th element, for example, would therefore be accessed at index 8.

An array is a simple data structure that Java supports so that we can hold in memory a fixed number values of the same data type. The length or storing capacity of an array is established when the array is created and after creation it is fixed. Each item in an array is called an element, and each element is accessed by its numerical index. Because we are computer scientists, numbering begins at 0. The 9th element, for example, would therefore be accessed at index 8.

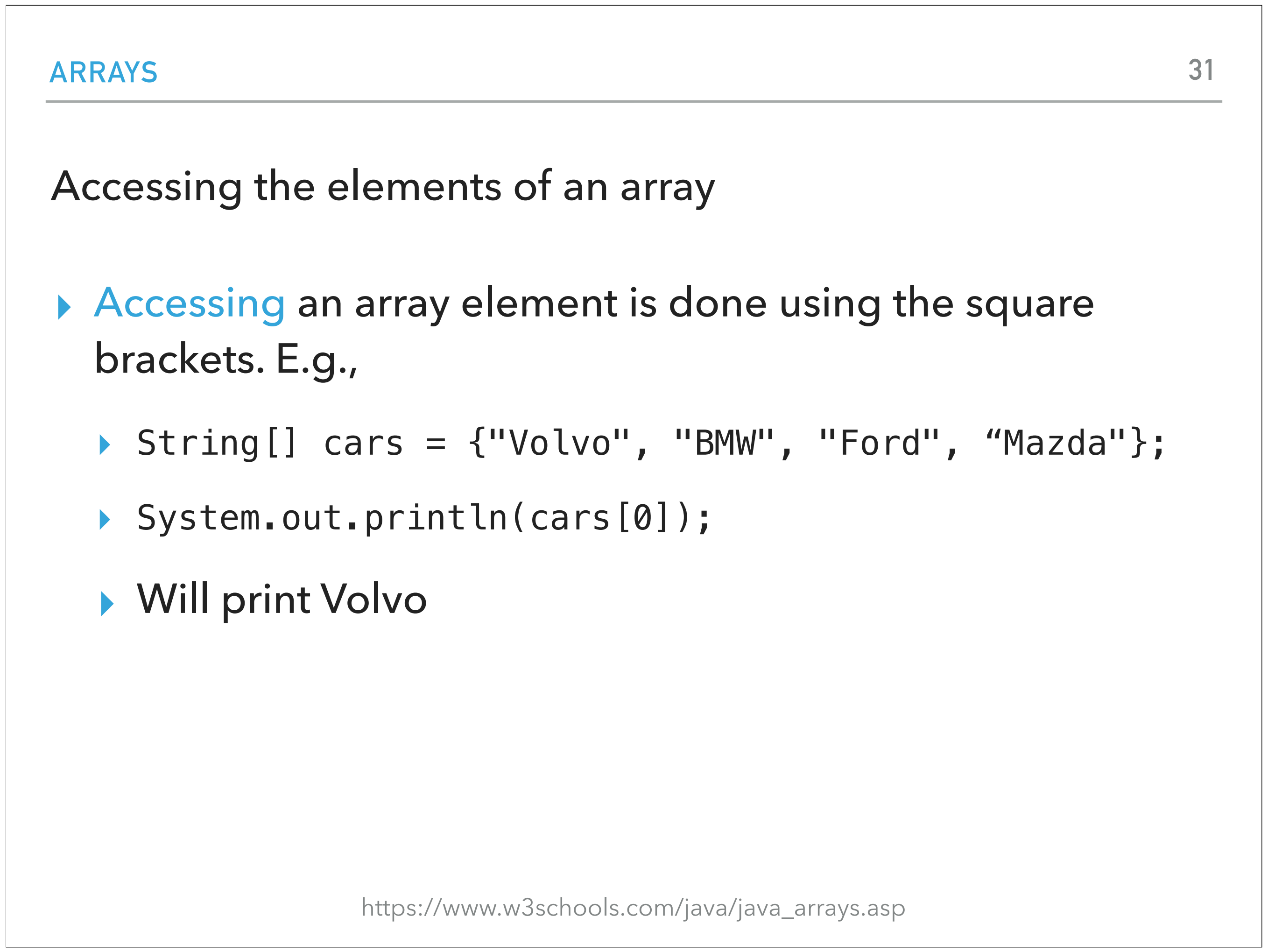


Declaring and initializing arrays

- Declaring an array requires the use of square brackets next to the type of the values it will hold. For example:
  - `String[] cars;`
  - `int[] numbers;`
- When we declare it, we can also initialize it with certain values separated by comma. For example,
  - `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};`
  - `int[] numbers = {10, 20, 30, 40};`

https://www.w3schools.com/java/java_arrays.asp

To use an array, we need to first declare the type of the values that it will hold (remember, they need to all be of the same data type!). We will use the square brackets to do so. For example, String[] cars; declares an array called cars that will hold strings and int[] numbers; declares an array that is called numbers and will hold integers. When we declare an array, we can also initialize it with its contents. The values it will contain will be separated by comma: String[] cars = {"Volvo", "BMW", "Ford", “Mazda”}; We could have also declared and initialized the array in one go, e.g., String[] cars = {"Volvo", "BMW", "Ford", "Mazda"}; or
int[] numbers = {10, 20, 30, 40};

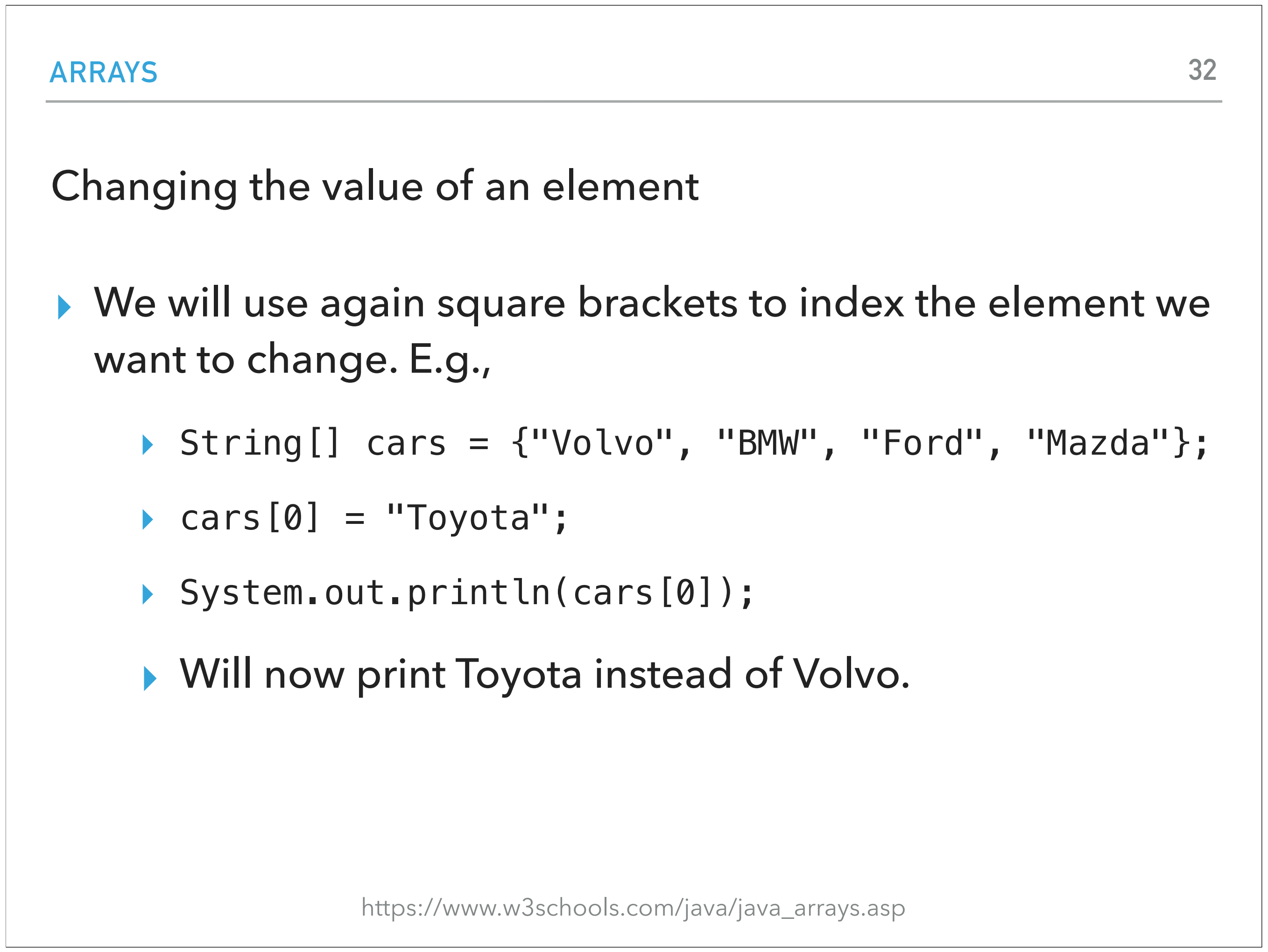


Accessing the elements of an array

- Accessing an array element is done using the square brackets. E.g.,
  - `String[] cars = {"Volvo", "BMW", "Ford", “Mazda"};`
  - `System.out.println(cars[0]);`
  - Will print Volvo

https://www.w3schools.com/java/java_arrays.asp

If you want to access an array element, you will again use the square brackets (remember we start counting at 0). For example, if we had String[] cars = {"Volvo", "BMW", "Ford", “Mazda"};
The statement System.out.println(cars[0]); Will print Volvo



Changing the value of an element

- We will use again square brackets to index the element we want to change. E.g.,
  - `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};`
  - `cars[0] = "Toyota";`
  - `System.out.println(cars[0]);`
  - Will now print Toyota instead of Volvo.

https://www.w3schools.com/java/java_arrays.asp

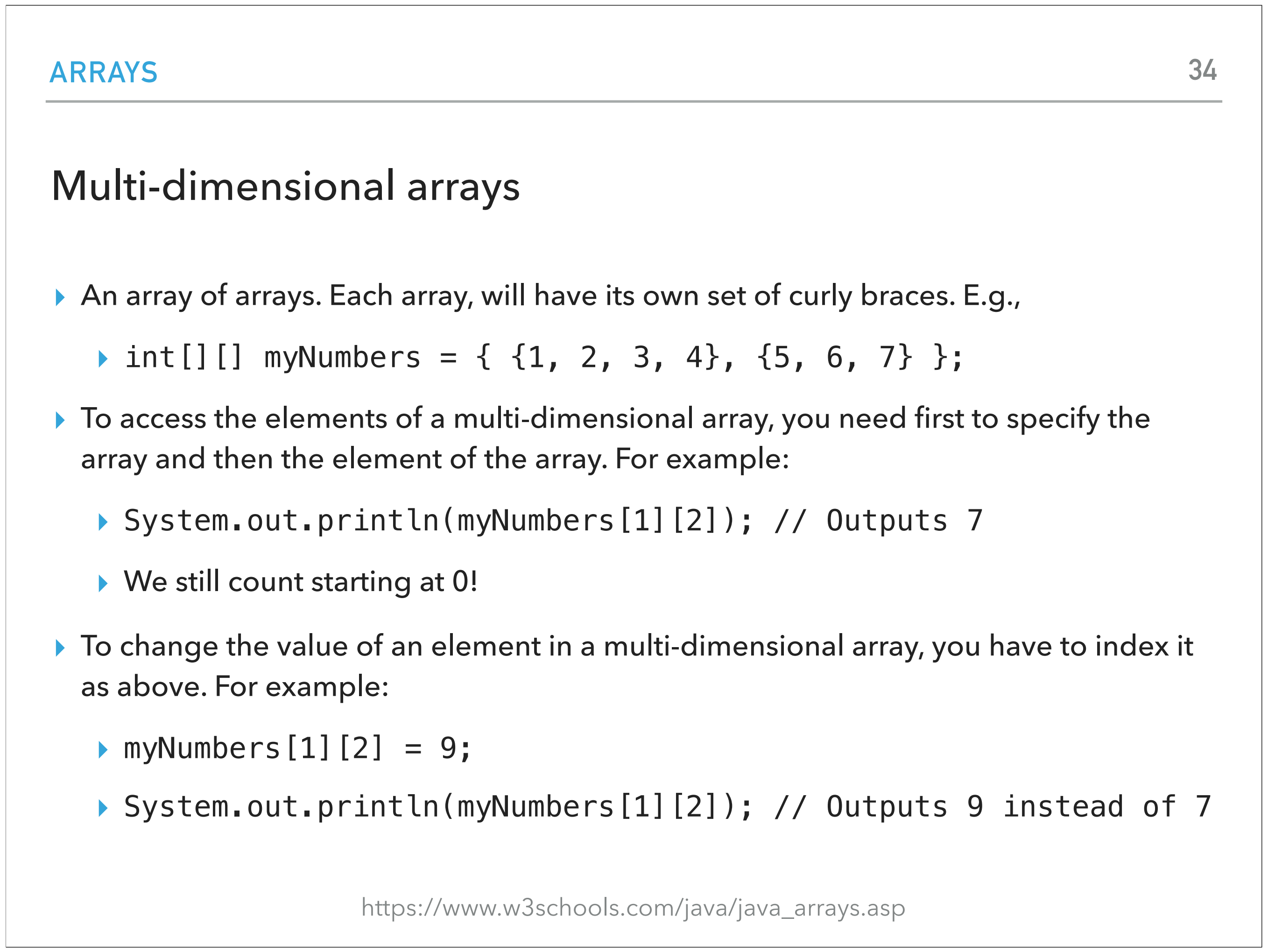
If you want to change the value of an element, you will use again the square brackets to index the element you want to change. For example, if you have cars[0] = "Toyota";
System.out.println(cars[0]);
Will now print Toyota instead of Volvo.

Array length

- We can determine the storing capacity of an array using the `length` property. E.g.,
  - `String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};`
  - `System.out.println(cars.length);`
  - Will print 4
- If you request an index that is either negative or larger than `length-1`, then you will get an [ArrayIndexOutOfBoundsException](https://www.w3schools.com/java/java_arrays.asp).

https://www.w3schools.com/java/java_arrays.asp

Finally, if you want to determine the storing capacity of an array, you will use the length property. String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
Will print 4
Note that if you request an index that is either negative or larger than length-1, then you will get an ArrayIndexOutOfBoundsException.



Multi-dimensional arrays

- An array of arrays. Each array, will have its own set of curly braces. E.g.,
  - `int[][] myNumbers = { {1, 2, 3, 4}, {5, 6, 7} };`
- To access the elements of a multi-dimensional array, you need first to specify the array and then the element of the array. For example:
  - `System.out.println(myNumbers[1][2]); // Outputs 7`
  - We still count starting at 0!
- To change the value of an element in a multi-dimensional array, you have to index it as above. For example:
  - `myNumbers[1][2] = 9;`
  - `System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7`

https://www.w3schools.com/java/java_arrays.asp

A multidimensional array is an array of arrays. Multidimensional arrays are useful when you want to store data as a tabular form. 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} };
To access the elements of a multi-dimensional array, you need first to specify the array and then the element of the array. For example:
System.out.println(myNumbers[1][2]);

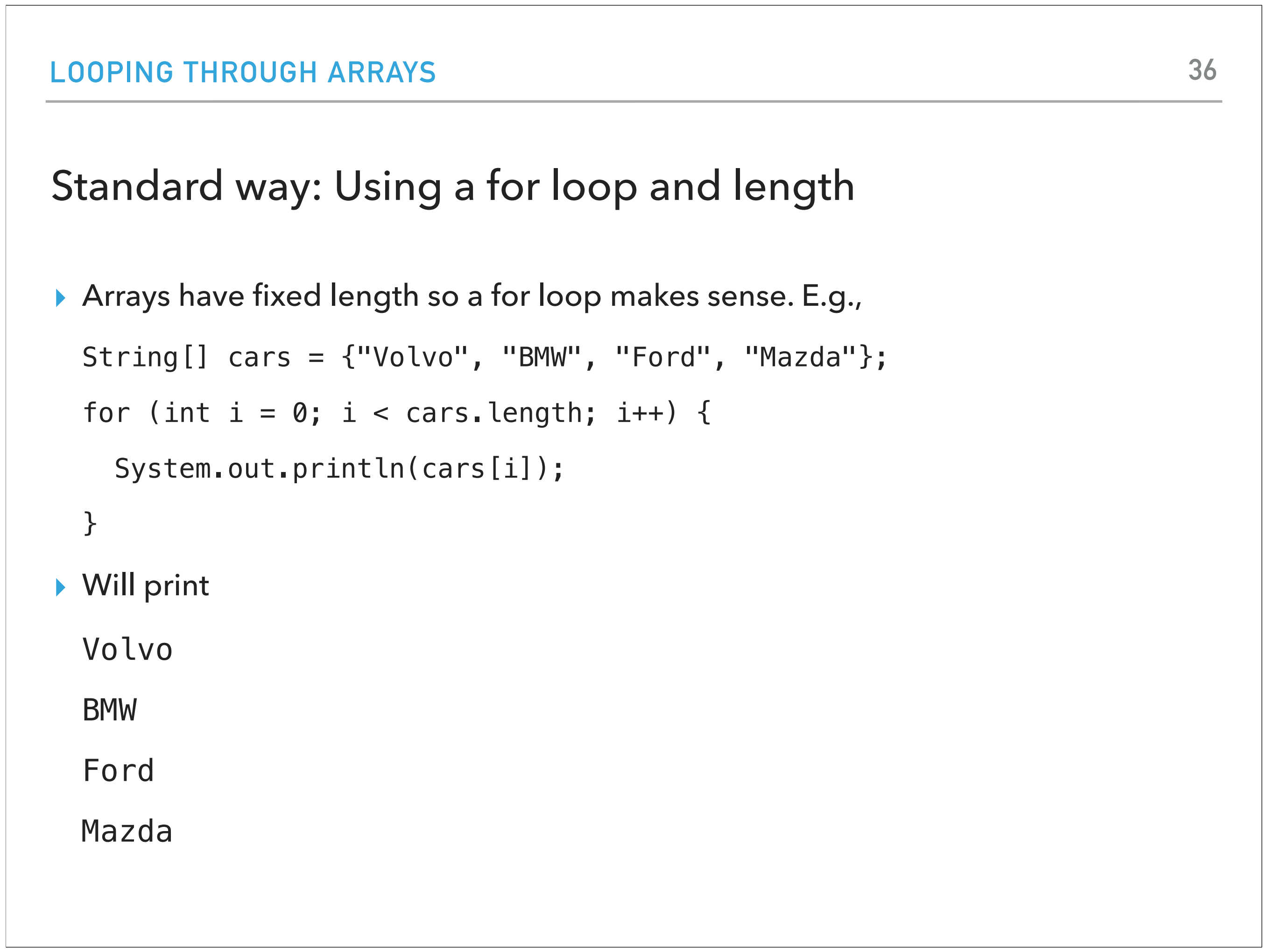
Will go to the second array and print its third element. We still count starting at 0!

To change the value of an element in a multi-dimensional array, you have to index it as above. For example:
myNumbers[1][2] = 9;
System.out.println(myNumbers[1][2]); // Outputs 9 instead of 7

Lecture 2: Control Flow and Arrays

- Selection
- Iteration
- Branching
- Arrays
- Looping through arrays

Let's see now we can loop through arrays. Remember, arrays are containers of fixed length that hold values of the same data type.



Standard way: Using a for loop and length

- Arrays have fixed length so a for loop makes sense. E.g.,

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

- Will print

```
Volvo
BMW
Ford
Mazda
```

The first and original Java-esque way of looping through an array is using a for loop. Since arrays have a fixed length, we know exactly how many iterations our for loop needs to make. For example, String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

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

Will print

```
Volvo
BMW
Ford
Mazda
```

For-each loop

- A new way of looping through arrays that doesn't need an iteration counter.
- Basic syntax:

```
for (type variableName : arrayName) {
     ...
}
```

- For example:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String car : cars) {
      System.out.println(car);
} //works same as before
```

There is also a more modern and Python-esque that Java now supports called a for-each loop and that doesn't need an iteration counter. Use it if that's not something you need.

Basic syntax:

```
for (type variableName : arrayName) {
   ...
}
```

For example:

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

PRACTICE TIME - Worksheet Problem 3

- Declare and initialize an array of strings with all the classes you are taking this semester.
    - Remember the word `class` is a reserved word, you cannot use it to name your variables.
- Write a for loop that loops through each class
- If a class is called “CSCI062” you need to print “CSCI062: This is the best class ever, no need to see more” and break the for loop.
    - We will use the equals method to compare equality among Strings.
    - e.g., someString.equals(someOtherString)
- Otherwise, if a class is called “CSCI101”, you need to print “CSCI101: New CS achievement unlocked” and continue to the next iteration.
- Otherwise, print the name of the class.

Let’s put everything together by practicing on our worksheet. Your task is to declare and initialize an array of strings with all the classes you are taking this semester.
Write a for loop that loops through each class
If a class is called “CSCI062” you need to print “CSCI062: This is the best class ever, no need to see more classes” and break the for loop.
We will use the equals method to compare equality among Strings.
e.g., someString.equals(someOtherString)
Otherwise, if a class is called “CSCI101”, you need to print “CSCI101: New CS achievement unlocked” and continue to the next iteration.
Otherwise, print the name of the class.

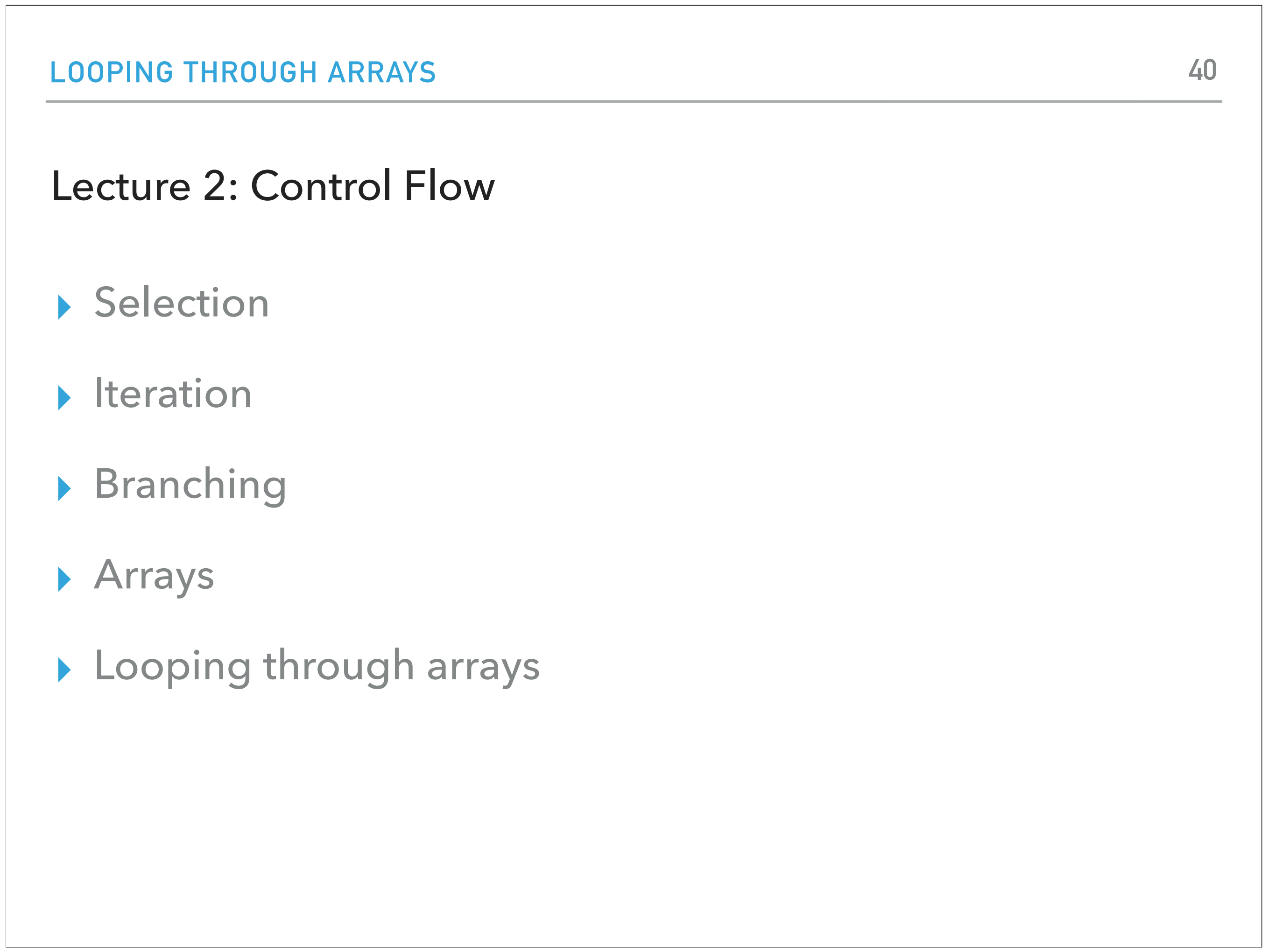
ANSWER - Worksheet Problem 3

- Here is my attempt. You could have also used a regular for loop instead of a for-each loop.

```
String[] classes = {"PHYS032", "CSCI101", "ANTH051", "CSCI062", "IMAG002"};
for(String myClass:classes){
    if(myClass.equals(”CSCI062”)){
         System.out.println("CSCI062: This is the best class ever, no need to see more");
         break;
    }
    else if(myClass.equals(“CSCI101”)){
        System.out.println("CSCI101: New CS achievement unlocked");
        continue;
    }
    System.out.println(myClass);
}
```

Here is my attempt. You could have also used a regular for loop instead of a for-each loop.

```
String[] classes = {"PHYS032", "CSCI101", "ANTH051", "CSCI062", "IMAG002"};
for(String myClass:classes){
    if(myClass.equals(”CSCI062”)){
         System.out.println("CSCI062: This is the best class ever, no need to see more");
         break;
    }
    else if(myClass.equals(”CSCI101”)){
        System.out.println("CSCI101: New CS achievement unlocked");
        continue;
    }
    System.out.println(myClass);
}
```



Lecture 2: Control Flow

- Selection
- Iteration
- Branching
- Arrays
- Looping through arrays

To summarize, today we talked about how to control the flow of our program beyond the standard sequential way. We saw selection statements, such as if-else if-else, and switch, iteration statements, such as while/do-while, and for loops, we learned about branching statements break and continue, and we also saw what arrays are and how to loop through arrays with a standard for loop and using the length property of an array or using a for-each loop.

Readings:

- Control flow: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/flow.html
- Arrays: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Code

- Lecture 2 code

Worksheet

- Lecture 2 worksheet

Here are some resources if you want to read more.