CS62
DATA STRUCTURES AND ADVANCED PROGRAMMING

2: Control Flow and Arrays

Alexandra Papoutsaki
she/her/hers
Lecture 2: Control Flow and Arrays

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

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

```java
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
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?

```java
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);
```
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:

```java
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);
```
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:

```java
switch(expression) {
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
```
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");
}
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.
What would happen if we didn’t include break?

```java
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
Ternary operator

?- A conditional operator that is a shorthand for the if–else statement.

Basic syntax:

\[
\text{variable} = \text{expression1} \ ? \ \text{expression2} : \ \text{expression3}
\]

Equivalent to:

```java
if(expression1) {
    variable = expression2;
} else {
    variable = expression3;
}
```
Ternary operator example

```java
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);
```
What will the following program print?

```java
int n1 = 32;
int n2 = 47;
System.out.println((n1>n2) ? (n1+n2):(n1-n2));
```
Since the \( n_1 > n_2 \) expression evaluates to \texttt{false}, it will print \(-15\) (i.e. \( n_1 - n_2 \)).
Lecture 2: Control Flow and Arrays

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

- Repeatedly execute a block of code as long as a specific condition is true.

- Basic syntax:

```java
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
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:

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

- Make sure your condition terminates otherwise you will enter an infinite loop.
do–while loop example

```java
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
for loop

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

- Basic syntax:

```java
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
for loop example

```java
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
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
Lecture 2: Control Flow and Arrays

- Selection
- Iteration
- Branching
- Looping through arrays
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.
break example

```java
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
continue

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

```java
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
Lecture 2: Control Flow and Arrays

- Selection
- Iteration
- Branching
- Arrays
- Looping through arrays
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.
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};`
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
Changing the value of an element

- We will use again square brackets to index the element we want to change. E.g.,
  
  ```java
  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
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
Multi-dimensional arrays

- An array of arrays. Each array will have its own set of curly braces. E.g.,
  ```java
  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:
  ```java
  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:
  ```java
  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
Standard way: Using a for loop and length

- Arrays have fixed length so a for loop makes sense. E.g.,
  ```java
  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:
  ```java
  for (type variableName : arrayName) {
      ...
  }
  ```
- For example:
  ```java
  String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
  for (String car : cars) {
      System.out.println(car);
  } //works same as before
  ```
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.
Here is my attempt. You could have also used a regular for loop instead of a for-each loop.

```java
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
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