**ITERATIVE CONTROL STRUCTURE**

**Loop**

* One of a computer’s most important attributes is the ability to perform repeatedly certain actions quickly, and loops are at the very heart of that.
* It is a repetition of a certain code segment/s in the program while the condition being evaluated remains to be true.
* It is used to iterate a part of the program repeatedly until specified condition is true.
* As soon as the condition becomes false, the loop automatically stops
* The while loop is considered a repeating “if-statement.

**Kinds of Loop:**

**While Loop**

* The DO/WHILE LOOP is a variant of the while loop.
* This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
* The Do-While loop is an exit-controlled type of loop.
* Example:
* int i = 0;

while (i <5) {

System.out.println(i);

I++;

}

**Do-While Loop**

* For a Do-While loop to work, the condition is not necessary to be met as this loop also works well for the first time even when the condition is not met.
* The compiler executor then enters the function execution block executes whatever is there within the block of statements, and then comes out to check the expression part where the condition is compared.
* If the condition is met, then the loop is reiterated; otherwise, the loop is exited from the block.
* The basic difference between the while and the do-while loop is that while the former one looks for the preconditions, the latter one targets the postconditions.
* Used when you want a code block to run repeatedly while the condition is met.
* A more compact format but more complicated version of a while loop.
* Example:
* int i = 0;

do {

System.out.println(i);

I++;

} while (i < 5);

**For Loop**

* **Steps**
* **Initializing Condition –** the variables to be used
* **Testing Condition –** In the test condition, one of the counter variable variables is checked whether it is greater than or less than a certain quantity.
* **Statement Execution –** in this phase, the print statement or the variable inside the for loop is executed making it easier to generate the output.
* **Incrementing/Decrementing Condition –** in this phase, the loop control variable or the counter variable is incremented by 1 generally to move the code forward.
* **Terminating the Loop –** when the condition doesn’t satisfy in the testing condition phase, the loop closes and doesn’t work anymore.
  + Example:
* for (int i = 0; i < 5; i++) {

System.out.println(i);

}

**JAVA ARRAYS**

**What is an Array?**

* Stores multiple data items (elements) of the same data type
* Placed in a contiguous block of memory divided into slots
* Allows storage and manipulates data efficiently
* Elements are the individual values in an array.

**Declaring Arrays**

* **Declare with Values**

datatype identifier[] = {val1, val2, val3, val4};

String names[] = {“David”, “Alenere”, “Jasper”};

A computer screen shot of a code

Description automatically generated

* **Declare without Values**

datatype identifier[] = new datatype[size];

String names[] = new String[4];

A computer screen shot of a code

Description automatically generated

* **Index** is a number that represents a position in a collection

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| VALUE | “David” | “Goliath” | “Paul” | “Mark” | “Matthew” |
| INDEX | 0 | 1 | 2 | 3 | 4 |

**Accessing Arrays**

* Reading Array Elements

Identifier[index];

Names[0];

* Assigning Array Element

Identifier[index] = value;

Names[0] = “Miles”;

**Reading Array Elements**

A computer screen shot of a code

Description automatically generated

**Assigning Array Element**

A computer code with text

Description automatically generated

**Assigning Array Element with User Input**

A screenshot of a computer screen

Description automatically generated

**2D JAVA ARRAYS**

**2D Array**

* It’s arrayed within an array.
* It basically looks like a table with rows and columns.
* Like 1D Arrays, it can only store one data type at a time.

**Declaring 2D Arrays**

* **Declare with Values**

A white background with black text

Description automatically generated

A computer screen shot of a computer code

Description automatically generated

* **Declare without Values**

A black text on a white background

Description automatically generated A screenshot of a computer

Description automatically generated

**Accessing 2D Array**

A table with numbers and letters

Description automatically generated

* **Read Values**

System.out.println(identifier [row][col]);

System.out.println(names[0][1]);

* **Write Values**

Identifier[row][col] = value;

Name[0][0] = “Ace”;

A computer screen shot of a program code

Description automatically generated

**Array Length**

* In order to get the number of elements in an array, the length field of an array can be used.
* The length field returns the size of the array and is written as follows:

arrayName.length

String [] cars = {“Volvo”, “BMW”, “Ford”, “Mazda”};

System.out.println(cars.length);

A computer screen shot of a program code

Description automatically generated

**JAVA METHODS**

**Methods**

* Also known as subprograms or functions.
* Block of code that runs independently after being called.
* Program by itself – has its own input, output, and processes.
* Divides the entire program into smaller portions.

**Types of Method**

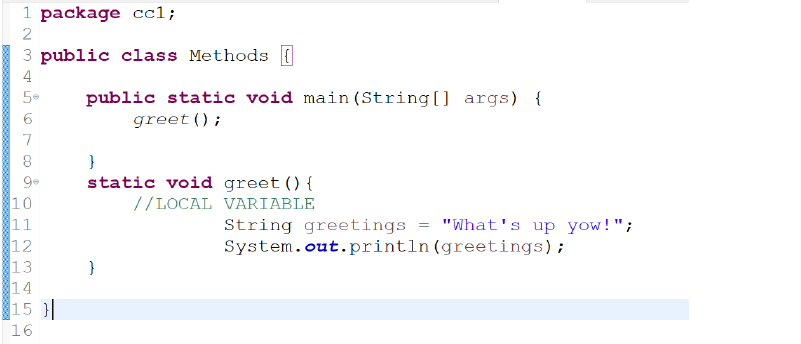
* **Static method**
  + Static is a keyword used to describe how objects are managed in memory.
  + It means that the static object belongs specifically to the class, instead of instances of that class.
  + **When should we use static methods?**
* They are helpful for operations not dependent on any particular class instance, such as mathematical calculations or checking for equality.
* **Static methods** can be convenient and help you write more concise code. However, they should be used sparingly and only when it makes sense.
* In general, **static methods** are best used for utility classes and singletons. Otherwise, it is usually better to create an object of the class and call instance methods on that object.

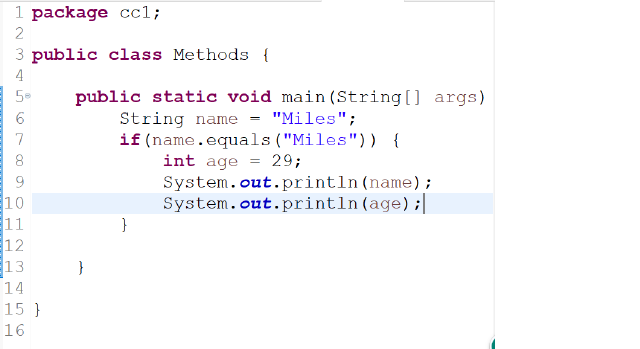
**Variable Scoping**

* **Global Variables**
* Variables declared within a class
* It can be accessed within the whole class



* **Local Variables**
* Variables declared inside a method, condition, loops, and any other block of code
* It can only be accessible within that block of code





* **Arguments/Parameters**
* A value that needs to be passed on a Method so that the method can use that value and perform various operations on it.
* **Methods w/ Arguments**
* modifiers returntype methodName(arguments) {

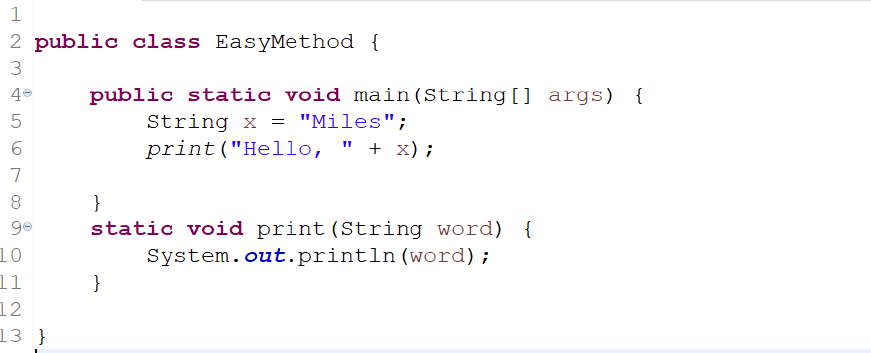
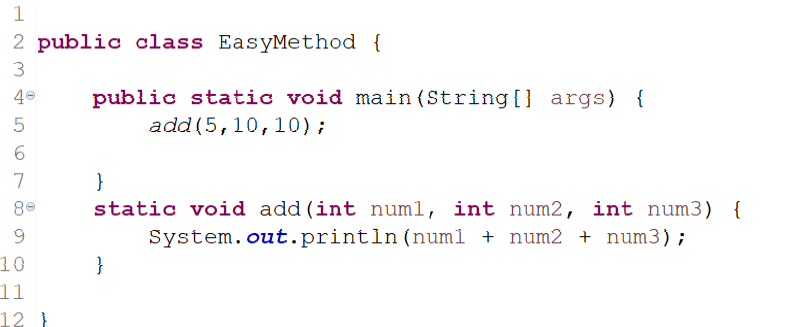
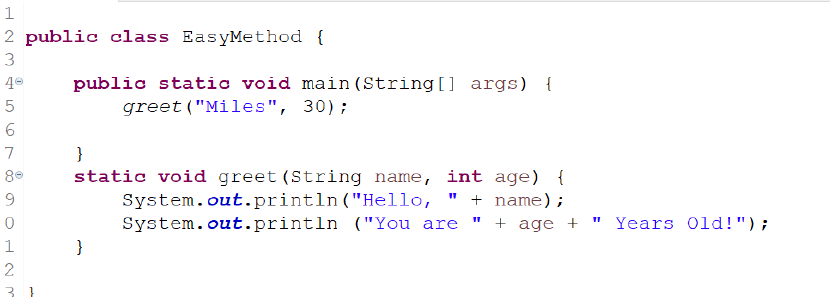
//Do anything here

}

* static void say (String word) {

System.out.println(word);

}

* 
* 
* 
* **Return method**
* **Return Keyword**
* Return keyword is used to return a value from the method.
* It is used when a method has a result.
* **Return Type**
* The type of value that will be returned.
* Return types are the same as the data types.
* Int: return integers

String: return strings

* **Methods w/ Return**
* Modifiers returntype methodName(arguments) {

//do anything here

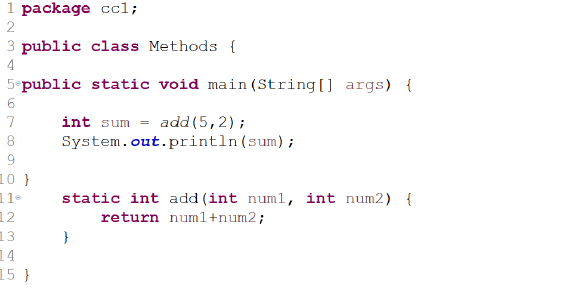
Return value;

}

* static int add(int num1, int num2) {

return num1 + num2;

}

* 
* **main() method**

import java.util.ArrayList;

public class WeatherAnalysisTool {

public static void main(String[] args) {

ArrayList<String> daysOfWeek = new ArrayList<>();

ArrayList<Integer> temperatures = new ArrayList<>();

// Populate the data

String[] dayNames = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"};

int[] tempValues = {22, 22, 22, 22, 23, 23, 23};

for (String day : dayNames) {

daysOfWeek.add(day);

}

for (int temp : tempValues) {

temperatures.add(temp);

}

// Convert array lists to 2D array

String[][] weatherData = new String[2][daysOfWeek.size()];

for (int i = 0; i < daysOfWeek.size(); i++) {

weatherData[0][i] = daysOfWeek.get(i);

weatherData[1][i] = String.valueOf(temperatures.get(i));

}

// Display the 2D array in a table form

System.out.println("Day | " + String.join(" | ", weatherData[0]));

System.out.println("Temperature | " + String.join(" | ", weatherData[1]));

System.out.println("--------------------------------------------");

// Find day with lowest and highest temperature using for-loops

String lowestDay = findDayWithLowestTemperature(weatherData);

String highestDay = findDayWithHighestTemperature(weatherData);

int lowestTemperature = Integer.parseInt(weatherData[1][findIndexForDay(weatherData, lowestDay)]);

int highestTemperature = Integer.parseInt(weatherData[1][findIndexForDay(weatherData, highestDay)]);

System.out.println("Day with the lowest temperature: " + lowestDay + " with a temperature of " + lowestTemperature + " °C");

System.out.println("Day with the highest temperature is " + highestDay + " with a temperature of " + highestTemperature + " °C");

}

public static String findDayWithLowestTemperature(String[][] weatherData) {

int minTemperature = Integer.MAX\_VALUE;

String minDay = "";

for (int i = 0; i < weatherData[0].length; i++) {

int temperature = Integer.parseInt(weatherData[1][i]);

if (temperature < minTemperature) {

minTemperature = temperature;

minDay = weatherData[0][i];

}

}

return minDay;

}

public static String findDayWithHighestTemperature(String[][] weatherData) {

int maxTemperature = Integer.MIN\_VALUE;

String maxDay = "";

for (int i = 0; i < weatherData[0].length; i++) {

int temperature = Integer.parseInt(weatherData[1][i]);

if (temperature > maxTemperature) {

maxTemperature = temperature;

maxDay = weatherData[0][i];

}

}

return maxDay;

}

public static int findIndexForDay(String[][] weatherData, String day) {

for (int i = 0; i < weatherData[0].length; i++) {

if (weatherData[0][i].equals(day)) {

return i;

}

}

return -1;

}

}