Methods

A method is a modular, reusable block of code that can be called throughout a program to complete a certain task.

```
/*
The following method is a public method
called findSum. The method takes in two
int parameters called int1 and int2. This
method returns an int value.
*/
public static int findSum(int num1, int
num2) {
  return num1 + num2;
}

public static void main(String[] args) {
  // Call the method with the arguments 3
and 4
  int sum = findSum(3,4);
  System.out.println(sum); // Prints: 7
}
```

Variable Types

Variables are used to name, store, and reference different types of data.
Primitive data types are predefined types of data and include `int` , `double` , `boolean` , and `char` .
Reference data types contain references to an object.
An example reference data type is `String` .

```
// int - stores whole numbers:
int num = 10;

// double - stores decimal numbers:
double dec = 4.99;

// boolean - stores true or false values:
boolean isTrue = true;

// char - stores a single character
value:
char firstLetter = 'A';

// String - stores multiple characters:
String message = "hello there";
```

Conditional Statements

In Java, conditional statements execute code based on the truth value of given `boolean` expressions.

```
boolean expression1 = false;
boolean expression2 = false;
boolean expression3 = true;

if (expression1) {
  System.out.println("The first
expression is true");
} else if (expression2) {
  System.out.println("The second
expression is true");
} else if (expression3) {
  System.out.println("The third
expression is true");
} else {
  System.out.println("All other
expressions were false");
}
// Prints: The third expression is true
```

Comparison and Logical Operators

Conditional operators and logical operators evaluate the relationship between values in order to determine a `true` or `false` value.

```
// Comparison Operators:
int a = 1;
int b = 5;
System.out.println(a > b); // Prints:
false
System.out.println(a < b); // Prints:
true
System.out.println(a >= 1); // Prints:
true
System.out.println(a + 4 <= b); //
Prints: true
System.out.println(a == 1); // Prints:
true
System.out.println(b != 5); // Prints:
false

// Logical Operators:
System.out.println(!true); // Prints:
false
System.out.println(!false); // Prints:
true

System.out.println(true && true); //
Prints: true
System.out.println(true && false); //
Prints: false
System.out.println(false && true); //
Prints: false
System.out.println(false && false); //
Prints: false

System.out.println(true || true); //
Prints: true
System.out.println(true || false); //
Prints: true
System.out.println(false || true); //
Prints: true
System.out.println(false || false); //
Prints: false
```

String Methods

Java's `String` class has many useful methods including:

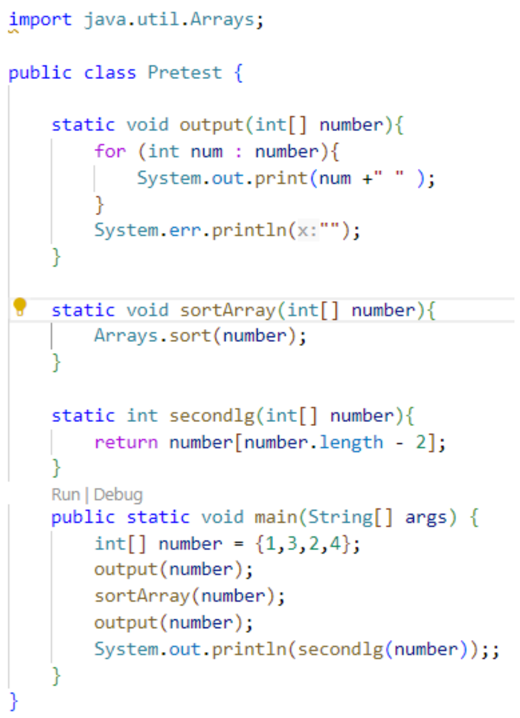
- `.length()` , which returns the length of the `String`
- `.concat()` , which concatenates two `String` s together
- `.equals()` , which checks for `String` equality
- `.indexOf()` , which returns the index of the first occurrence of a specified character
- `.charAt()` , which returns the character at a specified index
- `.substring()` , which extracts a substring

```
// Using the .length() method:
String str = "Hello World!";
System.out.println(str.length()); //
Prints: 12


// Using the .concat() method:
String name = "Code";
name = name.concat("cademy");
System.out.println(name); // Prints:
Codecademy


// Using the .equals() method:
String flavor1 = "Mango";
String flavor2 = "Matcha";
System.out.println(flavor1.equals(flavor2
)); // Prints: false


// Using the .indexOf() method:
String letters = "ABCDEFGHIJKLMN";
System.out.println(letters.indexOf("C"));
// Prints: 2


// Using the .charAt() method:
String currency = "Yen";
System.out.println(currency.charAt(2));
// Prints: n


// Using the .substring() method
String line = "It was the best of times,
it was the worst of times.";
System.out.println(line.substring(26));
// Prints: it was the worst of times.
System.out.println(line.substring(7,
24)); // Prints: the best of times
```

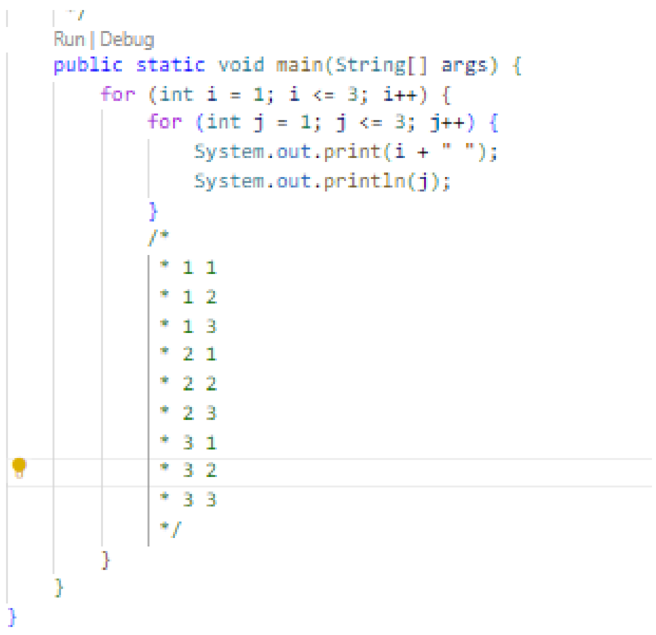
Loops

Java has four kinds of loops that rely on a `boolean` condition and continue to iterate until the condition is no longer true:

- `while` loops
- `do-while` loops
- `for` loops
- `for-each` loops



```
import java.util.Arrays;

public class Pretest {

    static void output(int[] number){
        for (int num : number){
            System.out.print(num +" " );
        }
        System.err.println(x:"");
    }

    static void sortArray(int[] number){
        Arrays.sort(number);
    }

    static int secondlg(int[] number){
        return number[number.length - 2];
    }
    Run | Debug
    public static void main(String[] args) {
        int[] number = {1,3,2,4};
        output(number);
        sortArray(number);
        output(number);
        System.out.println(secondlg(number));;
    }
}
```



```
     */
    Run | Debug
    public static void main(String[] args) {
        for (int i = 1; i <= 3; i++) {
            for (int j = 1; j <= 3; j++) {
                System.out.print(i + " ");
                System.out.println(j);
            }
            /*
             * 1 1
             * 1 2
             * 1 3
             * 2 1
             * 2 2
             * 2 3
             * 3 1
             * 3 2
             * 3 3
             */
        }
    }
}
```

```java
// An example of a while loop:
int x = 0;
while (x < 2) {
  System.out.println(x);
  x++;
} // Prints: 0 and 1


// An example of a do-while loop:
do {
  System.out.println("Impossible!");
} while (2 == 4); // Prints: Impossible!


// An example of a for loop:
for (int i = 0; i < 10; i++) {
        System.out.println(i);
} // Prints: 0 to 9, inclusive


// An example of a for-each loop:
String[] colors = {"Red", "Blue",
"Yellow"};
for (String c : colors) {
        System.out.println(c);
} // Prints: Red, Blue, and Yellow
```

codecademy

break and continue

Java has two keywords that help further control the number of iterations in a loop:

- `break` is used to exit, or break, a loop. Once `break` is executed, the loop will stop iterating.
- `continue` can be placed inside of a loop if we want to skip an iteration. If `continue` is executed, the current loop iteration will immediately end, and the next iteration will begin.

```
// An example of a break statement:
for (int i = 0; i < 10; i++) {
  System.out.println(i);
  if (i == 4) {
    break;
  }
} // Prints: 0 to 4, inclusive
```

```
// An example of a continue statement:
int[] numbers = {1, 2, 3, 4, 5};
for (int i = 0; i < numbers.length; i++)
{
  if (numbers[i] % 2 == 0) {
    continue;
  }
  System.out.println(numbers[i]);
} // Prints 1, 3, and 5
```

Print Share