Static Methods of the Math Class

Learn how to use static methods from the Math class to implement numerical calculations in Java.

The Math Class

Math can play a major role in our programs. It can become tedious to write out every equation we need in our code. Luckily, a lot of this work can be truncated using the `Math` class. The Java `Math` class is part of the `java.lang` package; it contains a variety of methods that can be used to perform numerical calculations in our programs. In this article, we'll discuss the different types of `Math` methods available to us and how to implement them in our Java programs.

The `Math` class also offers a good opportunity for us to learn about Static methods. We'll take a look at what static methods are and how they're different from non-static methods.

Calling Static Methods

Every method in the `Math` class is static. This means that we can call and use these methods without creating an object of the class. There are two main options for calling a static method.

Our first option is to append the dot operator to the class name followed by the method we want to execute. If we wanted to reference a method of the `Math` class, we would use `Math.NameOfMethodHere`. Let's see this in action with the `Math` class method `min()` which returns the smaller value of two given numbers:

```
class Numbers {
  public static void main(String[] args) {
    // Call method using the Class name, the dot operator, the method name, and arguments
    int smallerNumber = Math.min(3, 10);
    System.out.println(smallerNumber); // Prints: 3
  }
}
```

How is this any different from calling a non-static method? We don't need to create an object of the class in order to use the methods it contains. Let's see an example of a non-static method:

```
class Numbers {
  int firstNumber;
  int secondNumber;

  public Numbers (int num1, int num2) {
    firstNumber = num1;
    secondNumber = num2;
  }
```

```
  // non-static method
  public int returnSum() {
    return firstNumber + secondNumber;
  }

  public static void main(String[] args) {
    // Create an object
    Numbers myNumbers = new Numbers(2, 5);
    // Call a non-static method on object
    int sum = myNumbers.returnSum();
    System.out.println(sum); // Prints: 7
  }
}
```

In our code above, we had to create an object of type `Numbers` in order to use the non-static method `returnSum()`. With non-static methods, if we don't create an object of this class (or one of its subclasses), we do not have access to its methods. This isn't the case for static methods.

Our second option for calling a static method from the `Math` class is to import the class by adding `import static java.lang.Math.*;` to the top of our program. If we import the `Math` class, we can reference the method using only the method name like so:

```
import static java.lang.Math.*; // import Math class

class Numbers {
  public static void main(String[] args) {
    // Call method by using method name and arguments
    int smallerNumber = min(3, 10);
    System.out.println(smallerNumber); // Prints: 3
  }
}
```

Useful Methods

There are many useful methods from the `Math` class that can be implemented in our programs. For those of us taking the AP Computer Science A exam, the following methods and their descriptions will be available in the Java Quick Reference Guide:

int abs(int x)

Purpose: Returns the absolute value of an int value

The absolute value states how many numbers a value is away from `0`. The absolute value is always a positive number. For example, the absolute value of `-5` is `5` because it is `5` away from `0`. In Java, we can get the absolute value of a number like this:

```
System.out.println(Math.abs(5)); // Prints: 5
System.out.println(Math.abs(-5)); // Prints: 5
```

double abs(double x)

Purpose: Returns the absolute value of a double value

This is similar to the previous method, but this method takes in and returns a `double` type value:

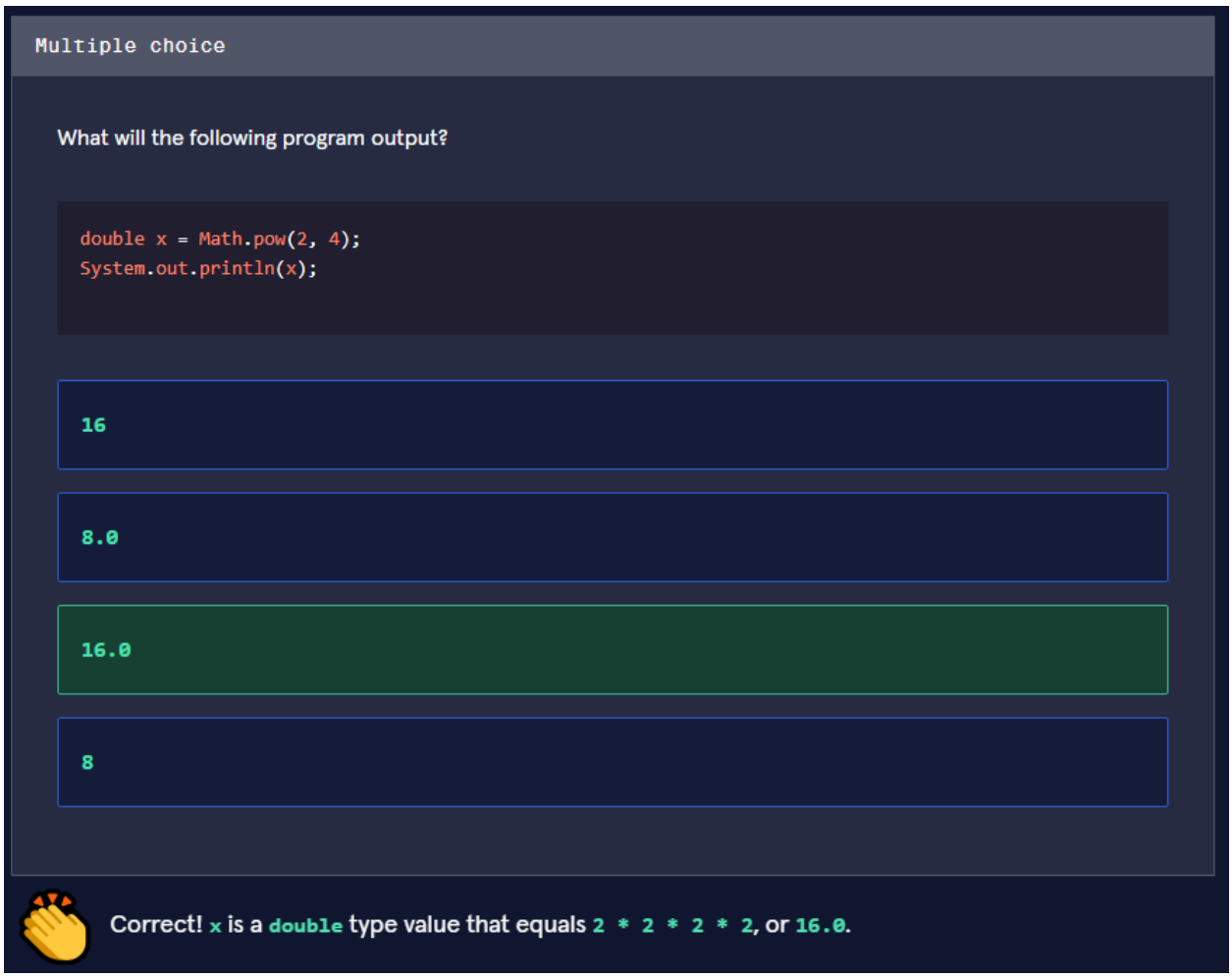
```
System.out.println(Math.abs(5.0)); // Prints: 5.0
System.out.println(Math.abs(-5.0)); // Prints: 5.0
```

double pow(double base, double exponent)

Purpose: Returns the value of the first parameter raised to the power of the second parameter.

The power, or exponent, describes how many times a number should be multiplied by itself. For example, `5` to the power of `3` is equivalent to `5 * 5 * 5`, or `125`. If we wanted to see this in Java, we could use `Math.pow()` like this:

```
double x = Math.pow(5, 3);
System.out.println(x); // Prints: 125.0
```



double sqrt(double x)

Purpose: Returns the positive square root of a double value

The square root of a number represents what value can be multiplied by itself in order to equal a specified value. For example, the square root of `49` is `7` because `7 * 7` is `49`. In java, getting the square root of a value looks like this:

```
double x = Math.sqrt(49);
System.out.println(x); // Prints: 7.0
double y = Math.sqrt(52);
System.out.println(y); // Prints: 7.211102550927978
```

double random()

Purpose: Returns a double value greater than or equal to `0.0` and less than `1.0`

Randomization is a great way to add probability to our programs. There are many ways to implement `Math.random()` in Java. Its default use case is to produce a random `double` value between `0.0` and `1.0`. For example:

```
System.out.println(Math.random());
System.out.println(Math.random());
System.out.println(Math.random());
```

The random values can change every time we run our program. An example output of the above program is the following:

```
0.8592007008856128
0.6120058754881421
0.48259656765819403
```

With some manipulation, we can use `Math.random()` to create a random `int` or `double` value within a predefined range.

For example, if we wanted a random `double` value between `0` and `10`, not including `10`, we would multiply `Math.random()` by `10`

```
// Random double value between 0 and 10, not including 10
double a = Math.random() * 10;
```

If we wanted a random `int` value between `0` and `9`, we would need to implement the `(int)` casting operator in our expression like so:

```
// Random int value between 0 and 9
int b = (int)(Math.random() * 10);
```

If we wanted our range to start at `1` and end at `10`, we would have to add `1` to the end of our previous expression:

```
// Random int value between 1 and 10
int c = (int)(Math.random() * 10) + 1;
```

Using addition also gives us the ability to start the range at any number. What if we wanted an `int` value in the range of `10` up to and including `20`? We would have to implement the algorithm `(Math.random() * (maxValue - minValue + 1)) + minValue`.

For example:

```
// Random int value between 10 and 20
int d = (int)(Math.random() * 11 ) + 10;
```

We multiply `Math.random()` by `11` because `20` (our max value) minus `10` (our minimum value) plus `1` is `11`. We add `+ 10` outside the parentheses so that our smallest value is guaranteed to be `10`.

Here's another way to think about this algorithm — the value that you multiply by defines the number of possible values you can get. The number that you add defines the starting value. So, for example, `(int)(Math.random() * 3 ) + 5;` will give you one of three random values starting at `5`. So this could give you `5`, `6`, or `7`.

Finally, be careful of off-by-one errors when using `Math.random()`. For example, you might write some code that you think generates a number between 1 and 10, but it actually generates a number between 1 and 9. Be sure to test your code frequently to spot logical errors like these!

Coding question

Run the following program to see the different kinds of random values we can generate using `Math.random()`.

Feel free to play around with these values to create your own range of randomized values.

```
class RandomNumbers {
  public static void main(String[] args) {
    // Random double value between 0 and 1
    System.out.println(Math.random());

    // Random double value between 0 and 9
    double a = Math.random() * 10;
    System.out.println(a);

    // Random int value between 0 and 9
    int b = (int)(Math.random() * 10);
    System.out.println(b);

    // Random int value between 1 and 10
    int c = (int)(Math.random() * 10) + 1;
    System.out.println(c);

    // Random int value between 10 and 20
    int d = (int)(Math.random() * 11 ) +
```

Run

Check answer

Multiple choice

Which option will produce a random `int` value in a range starting from `3` up to (and including) `8`?

```
int x = (int)(Math.random() * 11) + 3;
```

```
int x = (int)(Math.random() * 3) + 8;
```

```
int x = (int)(Math.random() * 6) + 3;
```

```
int x = (int)(Math.random() * 8) + 3;
```

Correct! Using the algorithm, `(Math.random() * (maxValue - minValue + 1)) + minValue`, the correct range is produced by the value `(int)(Math.random() * 6) + 3;`.

Additional Methods

The examples above aren't the only methods available in the `Math` class! To see all the methods offered by the `Math` class, check out the official documentation for the `Math` class.

Multiple choice

Which of the following options is not a method in the `Math` class?

Hint: You might need to check the `Math` class documentation.

```
tanh(double x)
```

```
negateExact(int a)
```

```
roundDown(float x)
```

```
hypot(double x, double y)
```

Correct! This is not a real method.

Conclusion

Great job reaching the end of this article. Let's recap what we learned:

- The `Math` class is part of the `java.lang` package and provides useful static methods for performing mathematical equations.
- To call these static methods, reference the class name + the dot operator + the method name. To only reference the method name, import the `Math` class into your program.
- For students taking the AP Computer Science A exam, several methods from the `Math` class will be available in a quick reference sheet.