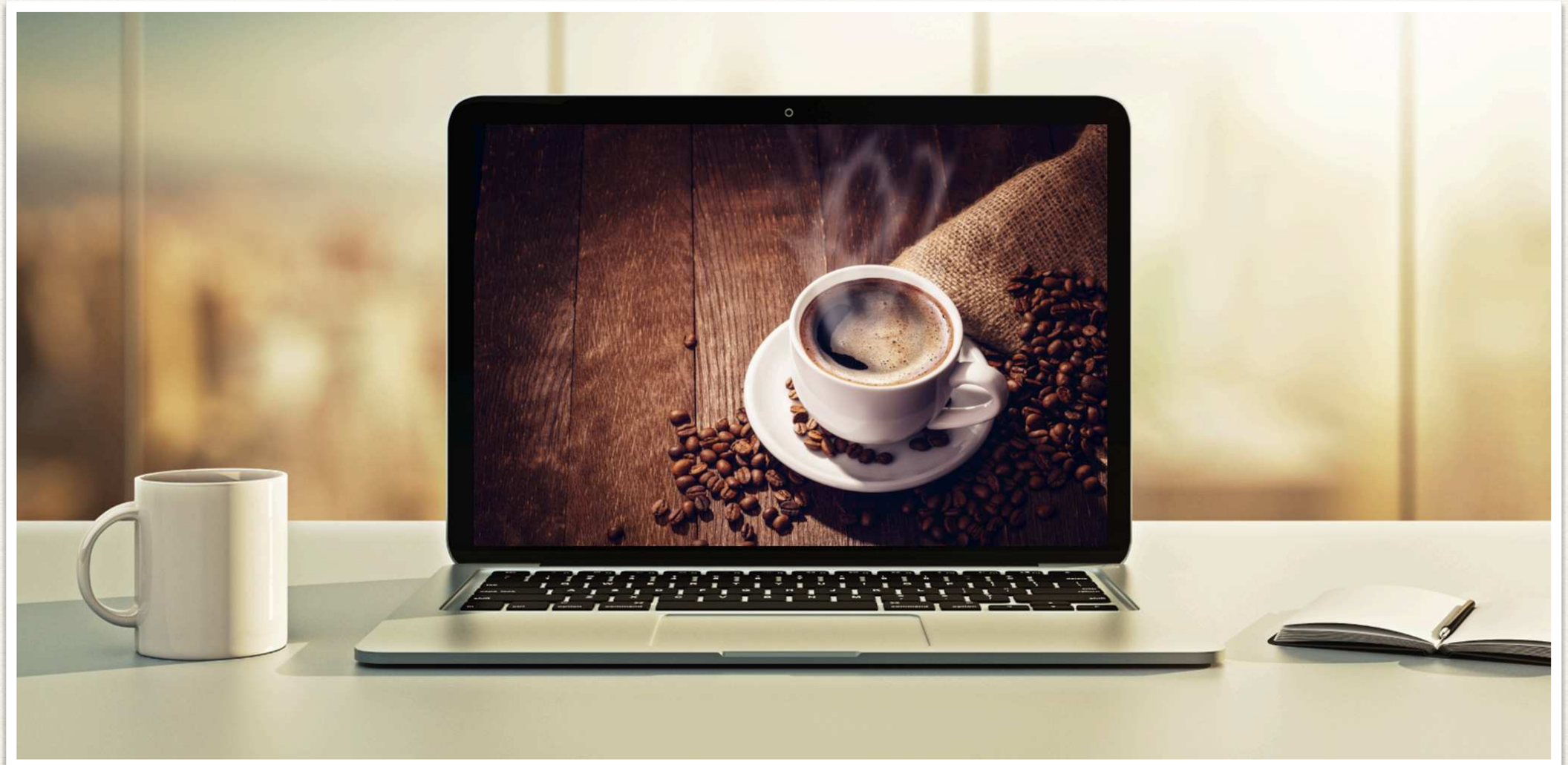
Methods



Methods

- A method is a block of code that is executed when you call it
- You can add your own custom code to a method
- Methods are useful for code reuse, readability and maintenance

Development Process

- Step 1: Define the method
- Step 2: Call the method

Basic Example

Reusable block of code

Step 1: Define the method

```
static void displayGreetings() {
    System.out.println("Hello world!");
    System.out.println("Welcome, welcome.");
    System.out.println("Please make yourself at home.");
}
```

Step 2: Call the method

```
public static void main(String[] args) {

    displayGreetings();

}
```

```
Hello world!
Welcome, welcome.
Please make yourself at home.
```

General Syntax

public, private, protected or *nothing*

static for class methods

primitive data types or class (predefined or custom)

Any method name

Any collection of parameters

```
[access modifier] [static] [return type] [method name]([params]) {

    // code to execute

}
```

code to execute

We will cover public, private, protected in detail later in the course

Applying Syntax to our example

public, private, protected or *nothing*

static for class methods

primitive data types or class (predefined or custom)

void: returns nothing

Any method name

Any collection of parameters

```
static void displayGreetings() {
    System.out.println("Hello world!");
    System.out.println("Welcome, welcome.");
    System.out.println("Please make yourself at home.");
}
```

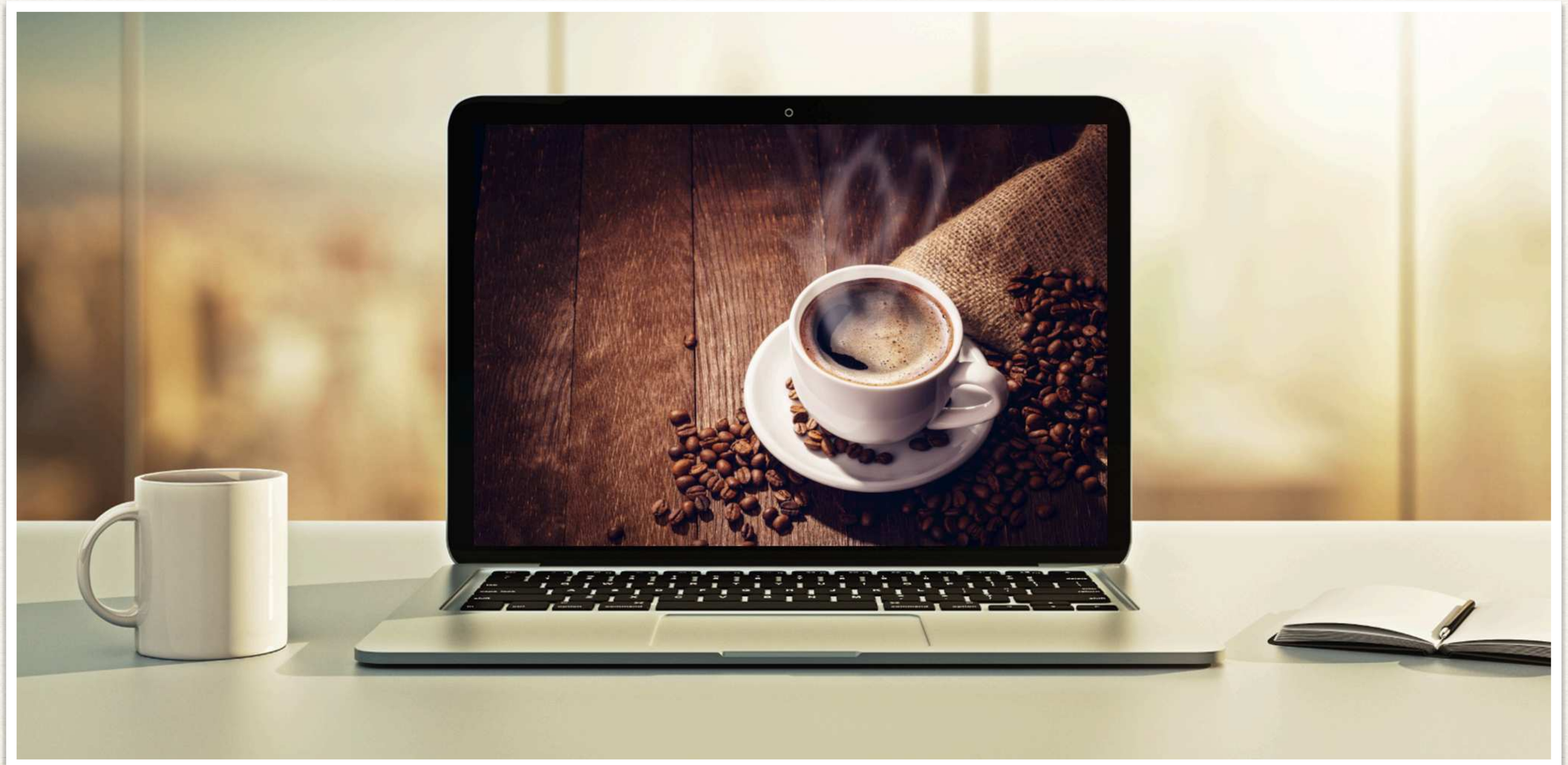
code to execute

Pulling it all together

```
public class MethodDemo {

    public static void main(String[] args) {

        // call the method
        displayGreetings();

    }

    // define the method
    static void displayGreetings() {
        System.out.println("Hello world!");
        System.out.println("Welcome, welcome.");
        System.out.println("Please make yourself at home.");
    }

}
```

```
Hello world!
Welcome, welcome.
Please make yourself at home.
```

Method Parameters and Method Overloading



Method Parameters

- We can pass parameters to a method
- The method can perform a task based on the input
  - `computeGradeAverage( … )`
  - `displayManyGreetings( … )`
  - `searchForCustomersWithLastName("Davis")`
- This promotes code reuse
  - Reuse the same method in different contexts based on the input / method parameters

Passing parameter to methods

<type> <param name>

Step 1: Define the method

```
static void displayManyGreetings(int count) {

    for (int i=1; i <= count; i++) {
        System.out.println("Hello world!");
        System.out.println("Welcome, welcome.");
        System.out.println("Please make yourself at home.");

        System.out.println();
    }
}
```

Step 2: Call the method

```
public static void main(String[] args) {

    displayManyGreetings(3);

}
```

Display our greeting 3 times

```
Hello world!
Welcome, welcome.
Please make yourself at home.

Hello world!
Welcome, welcome.
Please make yourself at home.

Hello world!
Welcome, welcome.
Please make yourself at home.
```

Pulling it all together

```java
public class MethodDemo {

    public static void main(String[] args) {

        // call the method
        displayManyGreetings(3);

    }

    static void displayManyGreetings(int count) {

        for (int i=1; i <= count; i++) {
            System.out.println("Hello world!");
            System.out.println("Welcome, welcome.");
            System.out.println("Please make yourself at home.");

            System.out.println();
        }
    }

    static void displayGreetings() {
        System.out.println("Hello world!");
        System.out.println("Welcome, welcome.");
        System.out.println("Please make yourself at home.");
    }

}
```

Display our greeting 3 times

```
Hello world!
Welcome, welcome.
Please make yourself at home.

Hello world!
Welcome, welcome.
Please make yourself at home.

Hello world!
Welcome, welcome.
Please make yourself at home.
```

Did you notice anything???

```
public class MethodDemo {

    public static void main(String[] args) {

        // call the method
        displayManyGreetings(3);

    }

    static void displayManyGreetings(int count) {

        for (int i=1; i <= count; i++) {
            System.out.println("Hello world!");
            System.out.println("Welcome, welcome.");
            System.out.println("Please make yourself at home.");

            System.out.println();
        }
    }

    static void displayGreetings() {
        System.out.println("Hello world!");
        System.out.println("Welcome, welcome.");
        System.out.println("Please make yourself at home.");
    }

}
```

Same code from method:
`displayGreetings()`

Refactor: Call existing method

```
public class MethodDemo {

    public static void main(String[] args) {

        // call the method
        displayManyGreetings(3);

    }

    static void displayManyGreetings(int count) {

        for (int i=1; i <= count; i++) {

            displayGreetings();

            System.out.println();
        }
    }

    static void displayGreetings() {
        System.out.println("Hello world!");
        System.out.println("Welcome, welcome.");
        System.out.println("Please make yourself at home.");
    }

}
```

Call existing method:
`displayGreetings()`

Method Overloading

- We can have multiple methods with the same name
- This is known as **method overloading**
- Simply provide different parameters
  - Different parameter types and number of parameters
- In our example, we can overload the method name: **`displayGreetings`**

Overloaded method example: `displayGreetings`

```
public class MethodDemo {

    public static void main(String[] args) {

        // call the method
        displayGreetings(3);

    }

    static void displayGreetings(int count) {

        for (int i=1; i <= count; i++) {

            displayGreetings();

            System.out.println();
        }
    }

    static void displayGreetings() {
        System.out.println("Hello world!");
        System.out.println("Welcome, welcome.");
        System.out.println("Please make yourself at home.");
    }

}
```

Call the `displayGreetings` method that has one parameter of type `int`

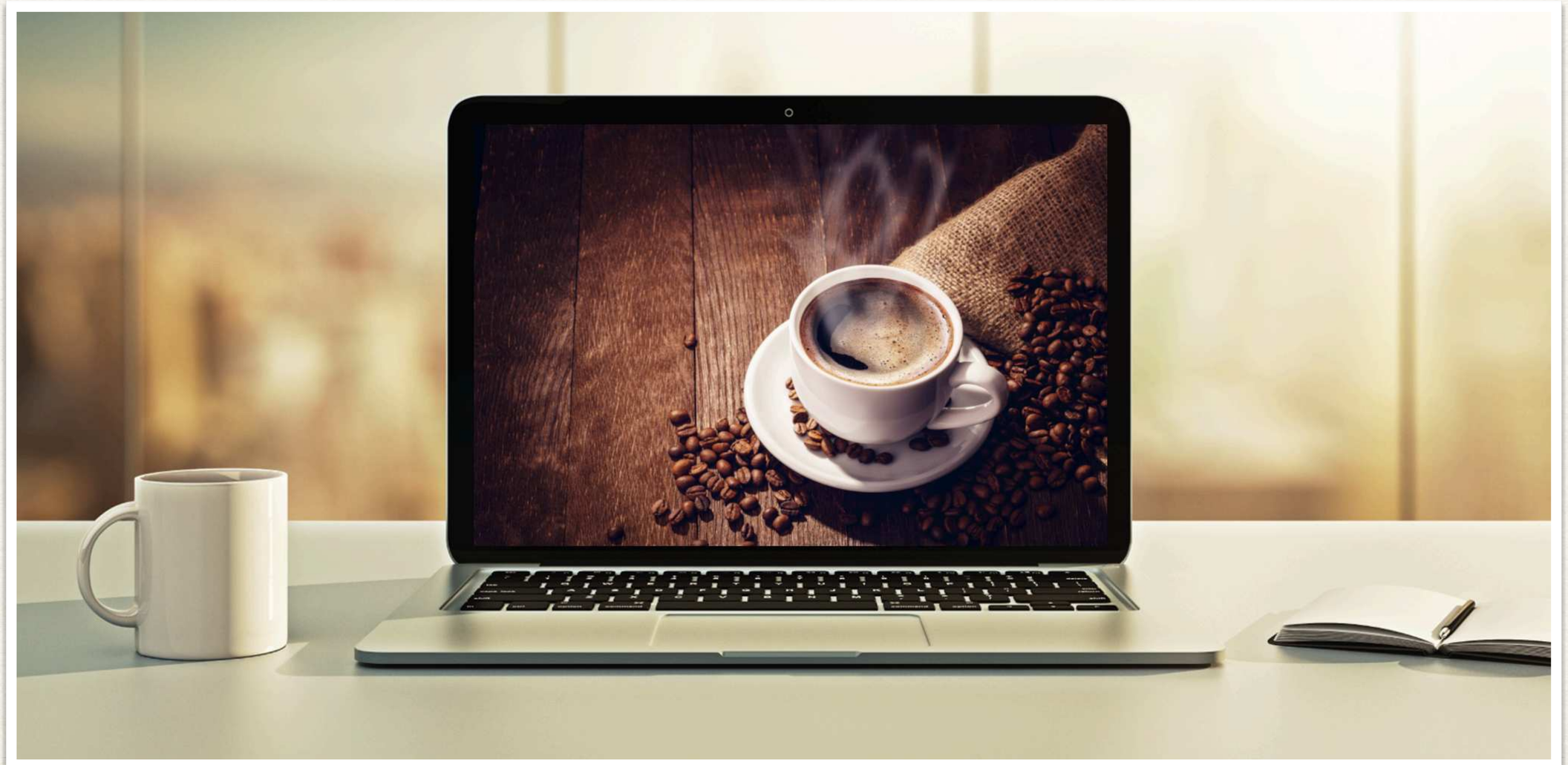
Both methods have the same name: `displayGreetings`

Differ in the parameter types and number of parameters

Java will match on the appropriate method and call accordingly

If there is an issue, you will have a compilation error

Methods: Return Value



Method Return Value

- A method can perform an operation based on the input and return a value
  - `double computeGradeAverage( … )`
  - `int getInventoryCountForProduct( … )`
  - `Customer searchForCustomerWithId( … )`

- Return type can be a
  - primitive: int, double, …
  - class: predefined or custom, …
  - …

Example: Summation

- Create a method to calculate a summation of a number
- A summation adds all numbers from 1 to a given number

- For example:
  - `summation(3) = 1+2+3 = 6`
  - `summation(5) = 1+2+3+4+5 = 15`

Example: Summation

Step 1: Define the method

```
static int summation(int num) {

    int result = 0;

    for (int i=1; i <= num; i++) {
        result += i;
    }

    return result;
}
```

A summation adds all numbers from 1 to a given number

`result = result + i;`

Return the result from the method

Step 2: Call the method

```
public static void main(String[] args) {

    int val = 5;
    int output = summation(val);
    System.out.println("Summation of " + val + " is " + output);

}
```

```
Summation of 5 is 15
```

Pulling it all together

```java
public class MethodReturnDataDemo {

    public static void main(String[] args) {

        int val = 5;
        int output = summation(val);
        System.out.println("Summation of " + val + " is " + output);

    }

    static int summation(int num) {

        int result = 0;

        for (int i=1; i <= num; i++) {
            result += i;
        }

        return result;
    }

}
```

```
Summation of 5 is 15
```

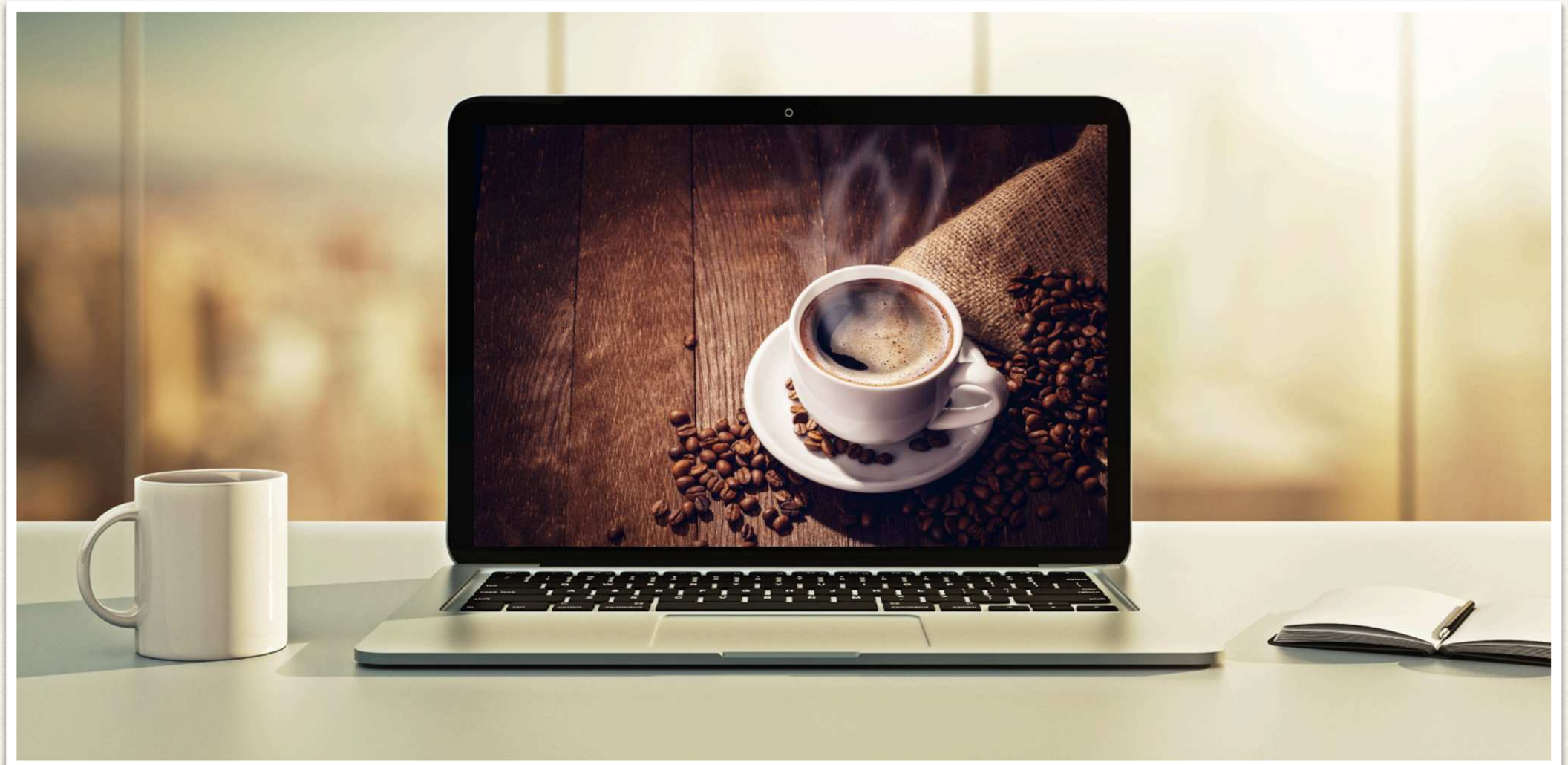
Refactor: Use a mathematical formula

```java
public class MethodReturnDataDemo {

    public static void main(String[] args) {

        int val = 5;
        int output = summation(val);
        System.out.println("Summation of " + val + " is " + output);

    }

    static int summation(int num) {

        int result = num * (num + 1) / 2;

        return result;
    }

}
```

$$summation(n) = \frac{n(n+1)}{2}$$

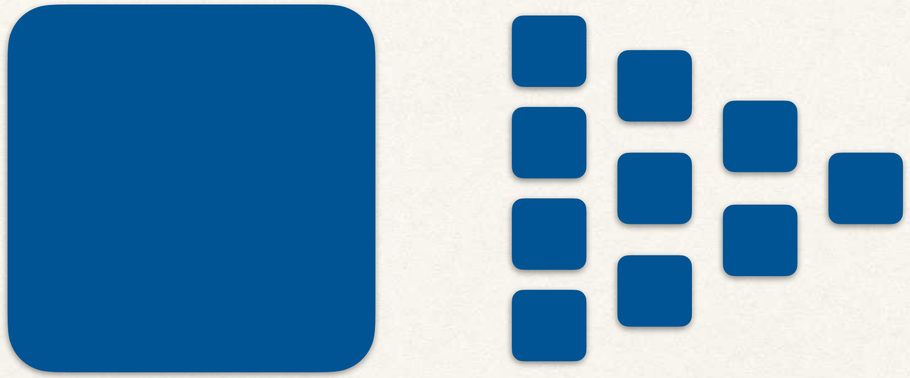
```
Summation of 5 is 15
```

Methods: Recursion



Recursion

- Recursion is an approach where a method calls itself to solve a problem
- Breaks a complex problem into smaller problems



Recursion: Use Cases

- Mathematical operations: factorial, fibonacci, …
- Sorting and searching algorithms: quicksort, mergesort, …
- Graphics: fractals, mandelbrot, …
- Artificial intelligence: supervised learning, natural language processing, …

Recursion: Key Components

- Base Case:
  - Condition that stops the recursion
  - Very important, without base case, code will run infinitely
- Recursive Case:
  - Break the problem into a smaller problem
  - Methods calls itself using modified parameters for smaller problem

Example: Factorial

- Create a method to calculate a factorial of a number
- A factorial multiples all numbers from 1 to a given number

$$n! = n * (n-1) * (n-2) * \ldots * 1$$

- For example:
  - `factorial(3) = 3*2*1 = 6`
  - `factorial(5) = 5*4*3*2*1 = 120`
  - `factorial(0) = 1`

Special case: 0! = 1

Factorial Approaches

- There are two approaches for computing the factorial
  - Iterative
  - Recursive

- We will focus on the recursive approach to demonstrate recursion

$$n! = n * (n - 1)!$$

Compute Factorial using Recursion

$$n! = n * (n-1)!$$

```
factorial(5) = 5 * factorial(4)
             = 5 * 4 * factorial(3)
             = 5 * 4 * 3 * factorial(2)
             = 5 * 4 * 3 * 2 * factorial(1)
             = 5 * 4 * 3 * 2 * 1 * factorial(0)
             = 5 * 4 * 3 * 2 * 1 * 1
             = 120
```

Recursive case

Recursive case

Recursive case

Recursive case

Base case

Example: Factorial

Step 1: Define the method

```
static int factorial(int num) {

    // base case: 0! = 1
    if (num == 0) {
        return 1;
    }
    else {
        // recursive case: num! = num * (num-1)!
        return num * factorial(num - 1);
    }
}
```

Step 2: Call the method

```
public static void main(String[] args) {

    int val = 5;
    int result = factorial(val);
    System.out.println("Factorial of " + val + " is " + result);

}
```

```
Factorial of 5 is 120
```

Pulling it all together

```java
public class RecursionDemo {

    public static void main(String[] args) {

        int val = 5;
        int result = factorial(val);
        System.out.println("Factorial of " + val + " is " + result);

    }

    static int factorial(int num) {

        // base case: 0! = 1
        if (num == 0) {
            return 1;
        }
        else {
            // recursive case: num! = num * (num-1)!
            return num * factorial(num - 1);
        }

    }

}
```

```
Factorial of 5 is 120
```