Classes & Methods

Brandon Krakowsky

Classes

Classes

- Everything in Java is object-oriented and class-based
  - This means you have to create *at least one class* to write a Java program

Classes

- Everything in Java is object-oriented and class-based
  - This means you have to create *at least one class* to write a Java program
- A class describes an object
  - It's like a *template* for a new kind of object
  - When you define a class, you're defining a new *data type*

Classes

- Everything in Java is object-oriented and class-based
  - This means you have to create *at least one class* to write a Java program
- A class describes an object
  - It's like a *template* for a new kind of object
  - When you define a class, you're defining a new *data type*
- To use the object, you create an *instance* of the class

Classes

- Everything in Java is object-oriented and class-based
  - This means you have to create *at least one class* to write a Java program
- A class describes an object
  - It's like a *template* for a new kind of object
  - When you define a class, you're defining a new *data type*
- To use the object, you create an *instance* of the class
- A class includes:
  - Fields (instance variables) that hold the data for each object
  - Constructors that describe how to create a new object instance of the class
  - Methods that describe the actions the object can perform

Classes

- Everything in Java is object-oriented and class-based
  - This means you have to create *at least one class* to write a Java program
- A class describes an object
  - It's like a *template* for a new kind of object
  - When you define a class, you're defining a new *data type*
- To use the object, you create an *instance* of the class
- A class includes:
  - Fields (instance variables) that hold the data for each object
  - Constructors that describe how to create a new object instance of the class
  - Methods that describe the actions the object can perform

We'll look at Methods today!

Defining a Class

- Here’s simple syntax for defining a sample class:

```
public class ClassName {
    // The fields (instance variables) of the object
    // The constructors for creating the object
    // The methods for communicating with the object
}
```

Defining a Class

- Here's simple syntax for defining a sample class:

```
public class ClassName {
    // The fields (instance variables) of the object
    // The constructors for creating the object
    // The methods for communicating with the object
}
```

- public is an *access modifier* that defines the visibility of the class
  - public means any other program in the Java project can use the class (i.e., create instances or call methods)
  - We'll talk about other *access modifiers* later in the course

Defining a Class

- Here's simple syntax for defining a sample class:

```
public class ClassName {
    // The fields (instance variables) of the object
    // The constructors for creating the object
    // The methods for communicating with the object
}
```

- public is an *access modifier* that defines the visibility of the class
  - public means any other program in the Java project can use the class (i.e., create instances or call methods)
  - We'll talk about other *access modifiers* later in the course
- Things in a class can be in any order

Methods

What is a Method?

- A method is a block of organized, reusable code that is used to perform a single, related action

What is a Method?

- A method is a block of organized, reusable code that is used to perform a single, related action
- A method is defined in a class
  - This allows you to use and communicate with the object defined by that class

What is a Method?

- A method is a block of organized, reusable code that is used to perform a single, related action
- A method is defined in a class
  - This allows you to use and communicate with the object defined by that class
- Java provides *built-in methods*
  - These are part of the core language, or imported package or class

What is a Method?

- A method is a block of organized, reusable code that is used to perform a single, related action
- A method is defined in a class
  - This allows you to use and communicate with the object defined by that class
- Java provides *built-in methods*
  - These are part of the core language, or imported package or class
- Java also allows you to define your own *user-defined methods*

Built-In Methods

- You’ve already been using built-in methods!
  - The *System.out.println()* method to print to the console

```
System.out.println(“Hello World!”);
```

Built-In Methods

- You’ve already been using built-in methods!

  - The *System.out.println()* method to print to the console

    ```
    System.out.println(“Hello World!”);
    ```

  - The *Math.pow()* method to calculate a number raised to the power of some other number

    ```
    double result = Math.pow(2, 3);
    System.out.println(result); //8.0
    ```

Built-In Methods

- You’ve already been using built-in methods!

  - The *System.out.println()* method to print to the console
```
System.out.println(“Hello World!”);
```

  - The *Math.pow()* method to calculate a number raised to the power of some other number
```
double result = Math.pow(2, 3);
System.out.println(result); //8.0
```

  - The *nextInt()* method of an imported *Scanner* object (class) to get user input of an int
```
System.out.println(“Enter an int: ”);
Scanner scan = new Scanner(System.in);
int myInt = scan.nextInt();
System.out.println(“Your number is: ” + myInt);
```

Built-In Methods

- There are lots of built-in methods. Here are some others:

  - The *Math.max()* method to return the maximum value between two numbers

```
int a = 10;
int b = 20;
System.out.println(Math.max(a, b)); //20
```

Built-In Methods

- There are lots of built-in methods. Here are some others:

  - The *Math.max()* method to return the maximum value between two numbers

```
int a = 10;
int b = 20;
System.out.println(Math.max(a, b)); //20
```

  - The *charAt()* method of a *String* object (class) to return a character by index position

```
String str1 = “hello world!”;
System.out.println(str1.charAt(1)); //e
```

Built-In Methods

- There are lots of built-in methods. Here are some others:

  - The *substring()* method of a *String* object (class) to extract characters, between two indices (positions), from a string

```
String str = “Welcome to the program!”;
System.out.println(str.substring(0, 7)); //Welcome
```

Built-In Methods

- There are lots of built-in methods. Here are some others:

  - The *substring()* method of a *String* object (class) to extract characters, between two indices (positions), from a string

```
String str = “Welcome to the program!”;
System.out.println(str.substring(0, 7)); //Welcome
```

  - The *equals* method of a *String* object (class) to compare two strings

```
String str1 = "hello!";
String str2 = “goodbye!”;
System.out.println(str1.equals(str2)); //false
```

User-Defined Methods

- Methods have conventions
    - Name a method based on what it does
    - Whitespace is not required, but important for readability
        - Method body “code blocks” (groups of statements) should be indented (4 spaces or tab)

User-Defined Methods

- Methods have conventions
  - Name a method based on what it does
  - Whitespace is not required, but important for readability
    - Method body “code blocks” (groups of statements) should be indented (4 spaces or tab)
- Sometimes a method takes an input
  - These are called *parameters*
  - When you call (or use) the method, you pass *arguments* to satisfy the *parameters*

User-Defined Methods

- Methods have conventions
  - Name a method based on what it does
  - Whitespace is not required, but important for readability
    - Method body "code blocks" (groups of statements) should be indented (4 spaces or tab)
- Sometimes a method takes an input
  - These are called *parameters*
  - When you call (or use) the method, you pass *arguments* to satisfy the *parameters*
- Sometimes a method produces an output
  - This is called the method's *return* value

Defining a Method in a Class

- You **always** define a method in a class
- The syntax for a method is:

```
return-type methodName(parameters) {
    // locally defined variables
    // code using parameters
    // optionally return a value
}
```

  - Parenthesis include optional *parameters*, treating them as variables
  - Methods optionally *return* a value

Defining a Method in a Class

- You **always** define a method in a class
- The syntax for a method is:

```
return-type methodName(parameters) {
    // locally defined variables
    // code using parameters
    // optionally return a value
}
```

  - Parenthesis include optional *parameters*, treating them as variables
  - Methods optionally *return* a value

- If a method DOES return a result, return-type is the data type of the result
  - You must use a return statement to exit the method with a result of the correct type

Defining a Method in a Class

- You **always** define a method in a class
- The syntax for a method is:

```
return-type methodName(parameters) {
    // locally defined variables
    // code using parameters
    // optionally return a value
}
```

  - Parenthesis include optional *parameters*, treating them as variables
  - Methods optionally *return* a value
- If a method DOES return a result, return-type is the data type of the result
  - You must use a return statement to exit the method with a result of the correct type
- If a method DOESN'T return a result, return-type is void
  - This indicates that a method doesn't return a value
  - In this case, you don't need to use a return statement to exit the method

Defining a Method in a Class

- Sample class definition with a method:

```
public class ClassName {

    // A method that calculates the square of a given x
    int square(int x) {
        int y = x * x; //calculate square of x
        return y; //return calculated square of x
    }
}
```

Using a Method in a Class

- To use a method in a class, you first create an instance of the class by using the keyword new
- Here's syntax to define a class and to create an instance:

```
public class ClassName {

    // A method that calculates the square of a given x
    int square(int x) {
        int y = x * x; //calculate square of x
        return y; //return calculated square of x
    }

    public static void main(String[] args) {
        //create instance of ClassName class
        ClassName c = new ClassName();
    }
}
```

Using a Method in a Class

- To use a method in a class, you first create an instance of the class by using the keyword new
- Here’s syntax to define a class and to create an instance:

```
public class ClassName {

    // A method that calculates the square of a given x
    int square(int x) {
        int y = x * x; //calculate square of x
        return y; //return calculated square of x
    }

    public static void main(String[] args) {
        //create instance of ClassName class
        ClassName c = new ClassName();
    }
}
```

- new creates a new instance of the object

Using a Method in a Class

- To call a method, use the instance of the class

```
public class ClassName {

    // A method that calculates the square of a given x
    int square(int x) {
        int y = x * x; //calculate square of x
        return y; //return calculated square of x
    }

    public static void main(String[] args) {
        //create instance of ClassName class
        ClassName c = new ClassName();

        //call square method using the instance of ClassName class
        c.square(4);
    }
}
```

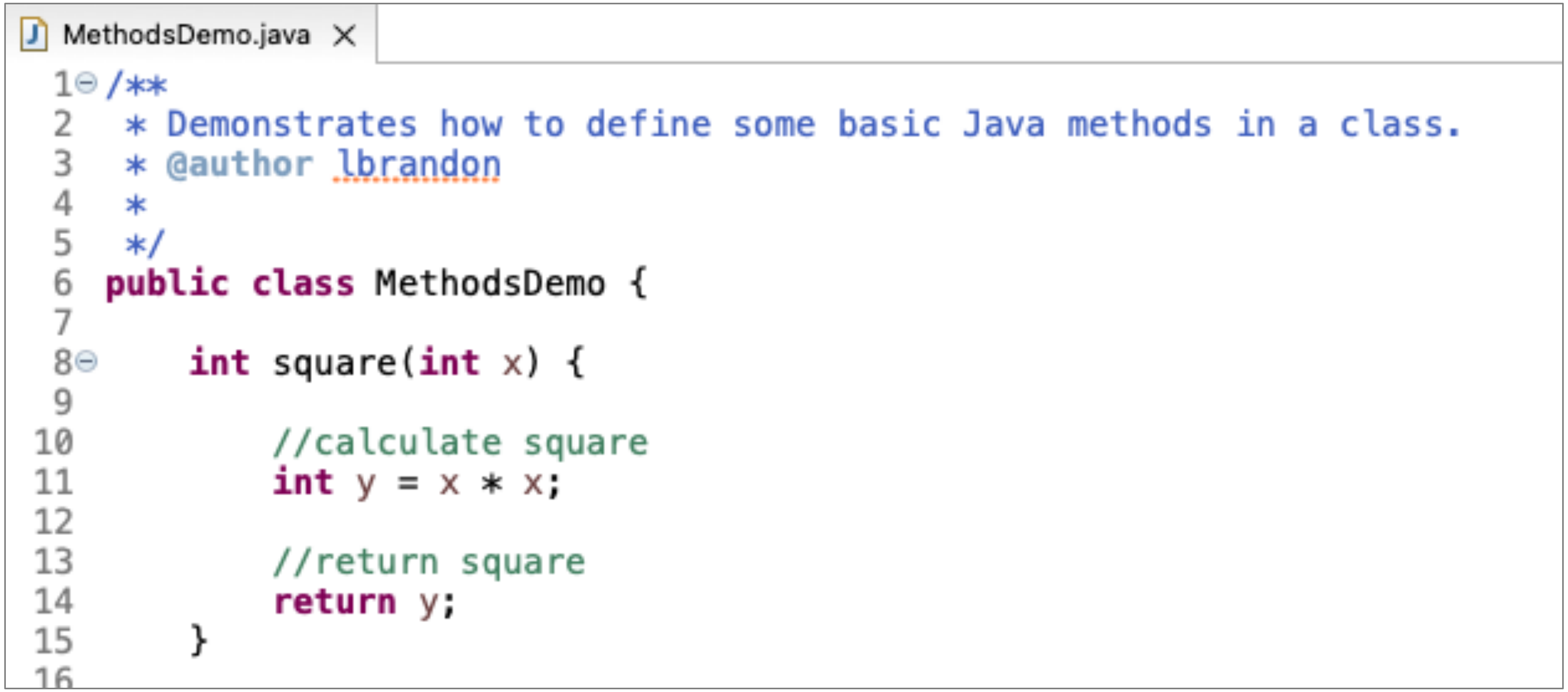
User-Defined Methods

- Let’s define a method *square*
  - It takes one int as a *parameter*
  - It *returns* the result of squaring that int

PennEngineering

User-Defined Methods

- Let’s define a method *square*
  - It takes one int as a *parameter*
  - It *returns* the result of squaring that int

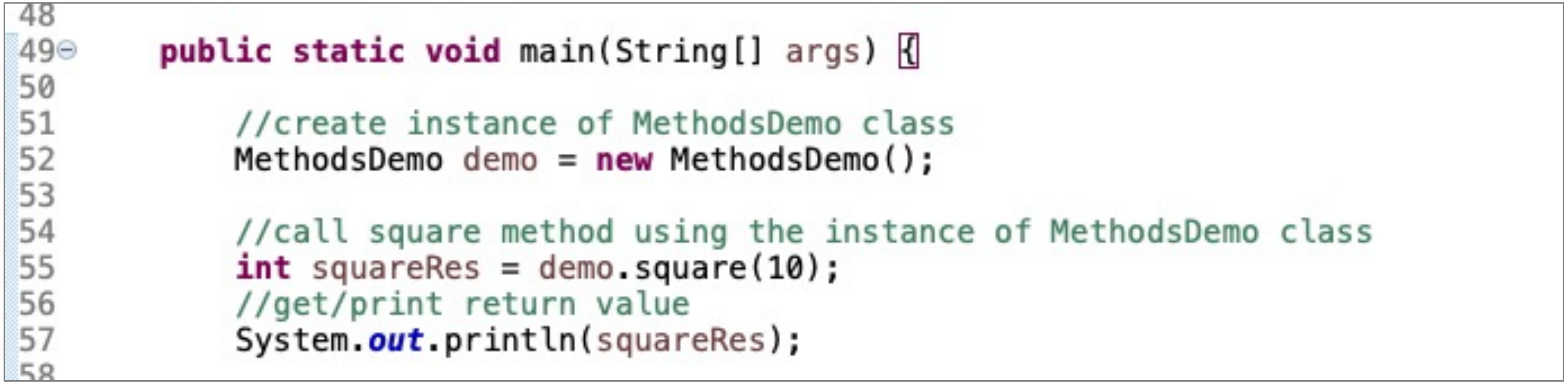


MethodsDemo.java

```java
/**
 * Demonstrates how to define some basic Java methods in a class.
 * @author lbrandon
 *
 */
public class MethodsDemo {

    int square(int x) {

        //calculate square
        int y = x * x;

        //return square
        return y;
    }

```

User-Defined Methods

- Now let’s use the method *square*
  - When we call it, we pass 10 as an *argument*
  - Then we store the *return* value in a squareRes variable and print it



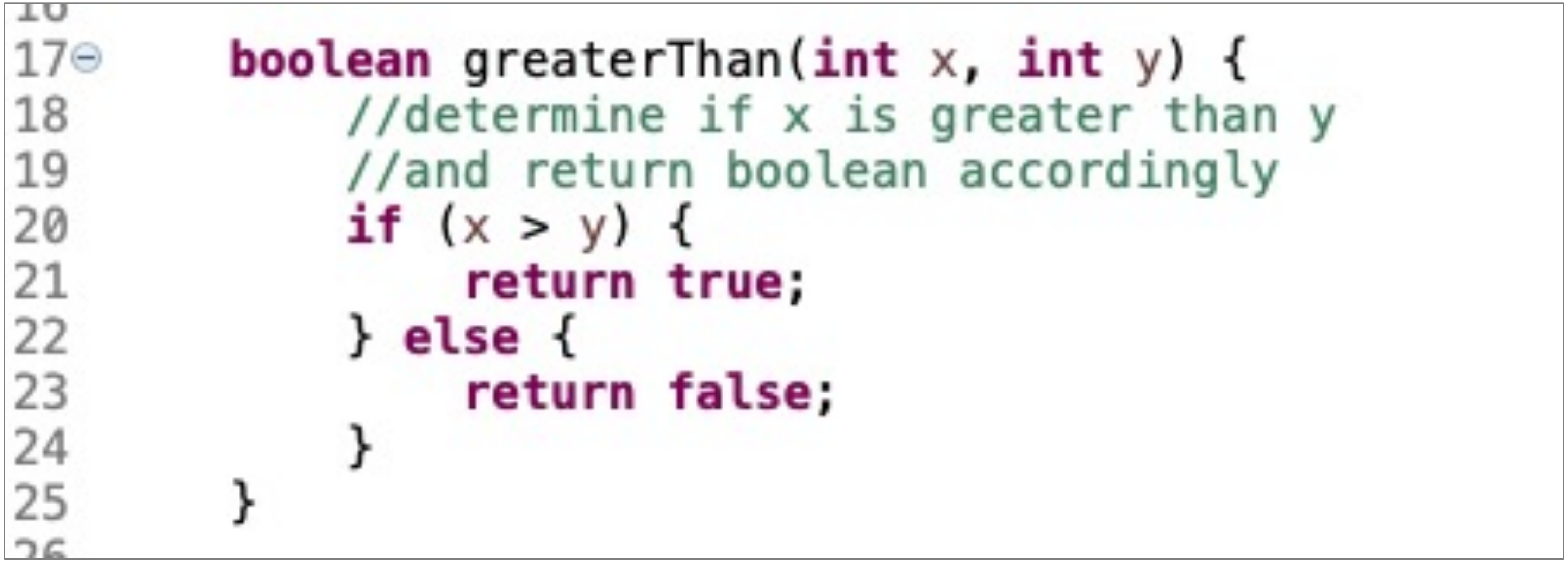
```
    public static void main(String[] args) {

        //create instance of MethodsDemo class
        MethodsDemo demo = new MethodsDemo();

        //call square method using the instance of MethodsDemo class
        int squareRes = demo.square(10);
        //get/print return value
        System.out.println(squareRes);
```

User-Defined Methods

- Let’s define a method *greaterThan*
  - It takes two ints as *parameters*
  - It *returns* true if the 1st *parameter* is greater than the 2nd *parameter*

User-Defined Methods

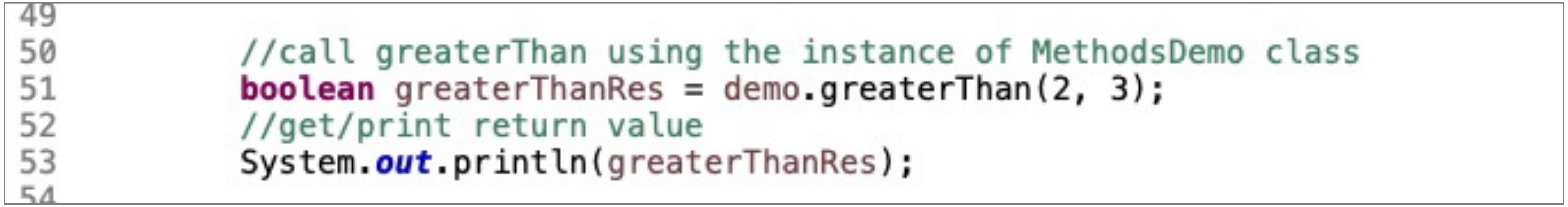
- Let’s define a method *greaterThan*
    - It takes two ints as *parameters*
    - It *returns* true if the 1st *parameter* is greater than the 2nd *parameter*



```
    boolean greaterThan(int x, int y) {
        //determine if x is greater than y
        //and return boolean accordingly
        if (x > y) {
            return true;
        } else {
            return false;
        }
    }
```

User-Defined Methods

- Now let’s use the method *greaterThan*
    - When we call it, we pass 2 and 3 as *arguments*
    - Then we store the *return* value in a greaterThanRes variable and print it



```
        //call greaterThan using the instance of MethodsDemo class
        boolean greaterThanRes = demo.greaterThan(2, 3);
        //get/print return value
        System.out.println(greaterThanRes);
```

Javadocs Review

- You can (and should) provide *Javadocs* (*Java documentation*) just *before* the definition of a method (or class)
  - *Javadocs* describe the operation of the method (or class)
  - For reference, this is the equivalent of a *docstring* inside of a Python function or class

Javadocs Review

- You can (and should) provide *Javadocs* (*Java documentation*) just *before* the definition of a method (or class)
  - *Javadocs* describe the operation of the method (or class)
  - For reference, this is the equivalent of a *docstring* inside of a Python function or class
- *Javadocs* are for someone who is using your method (or class) and wants to know "what it does" at a high level and/or "how to use it"

Javadocs Review

- You can (and should) provide *Javadocs* (*Java documentation*) just *before* the definition of a method (or class)
  - *Javadocs* describe the operation of the method (or class)
  - For reference, this is the equivalent of a *docstring* inside of a Python function or class
- *Javadocs* are for someone who is using your method (or class) and wants to know “what it does” at a high level and/or “how to use it”
- This is different from *comments*, which are for a programmer who might be reading your code and wants to know the details of “how it works”

Javadocs Review

- As a shortcut, you can type the following right above a method (or class)

  ```
  /**
  ```
  and then hit Enter

- It will add a javadoc block and you can fill in the rest

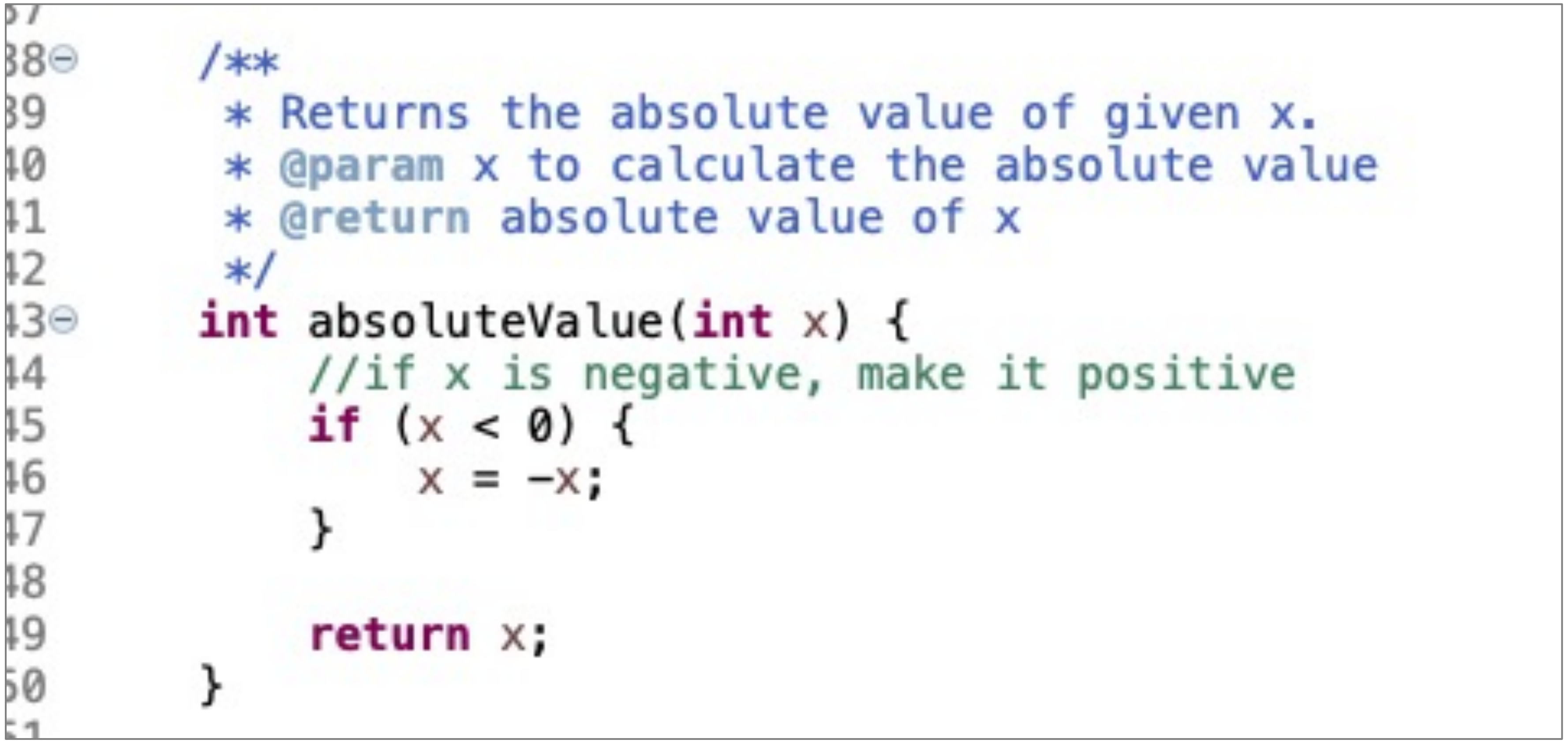
  ```
  /**
  * Returns the square of given x.
  * @param x to square
  * @return the square of x
  */
  int square(int x) {
      int y = x * x;
      return y;
  }
  ```

User-Defined Methods

- Define a method *absoluteValue*
  - It takes one int as a *parameter*
  - It *returns* the absolute value of that int
  - Make sure to add *Javadocs*!

User-Defined Methods

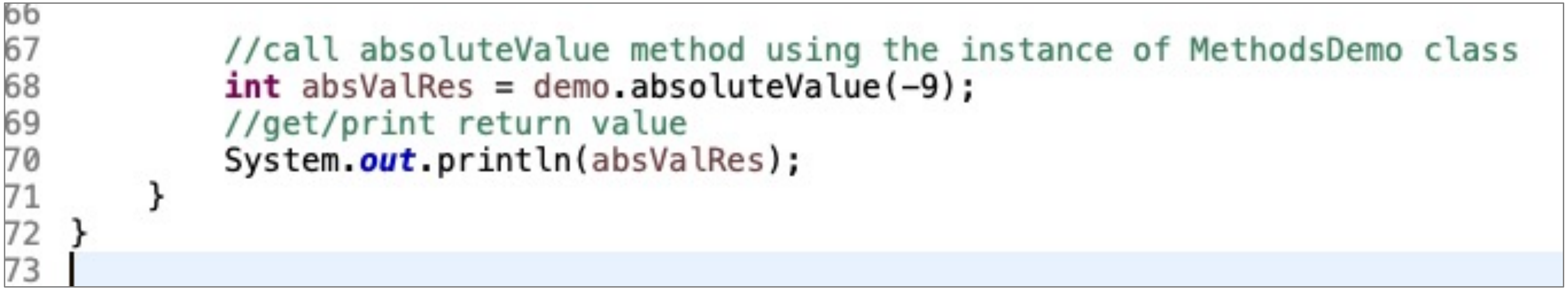
- Define a method *absoluteValue*
  - It takes one int as a *parameter*
  - It *returns* the absolute value of that int
  - Make sure to add *Javadocs*!



```
    /**
     * Returns the absolute value of given x.
     * @param x to calculate the absolute value
     * @return absolute value of x
     */
    int absoluteValue(int x) {
        //if x is negative, make it positive
        if (x < 0) {
            x = -x;
        }

        return x;
    }
```

User-Defined Methods

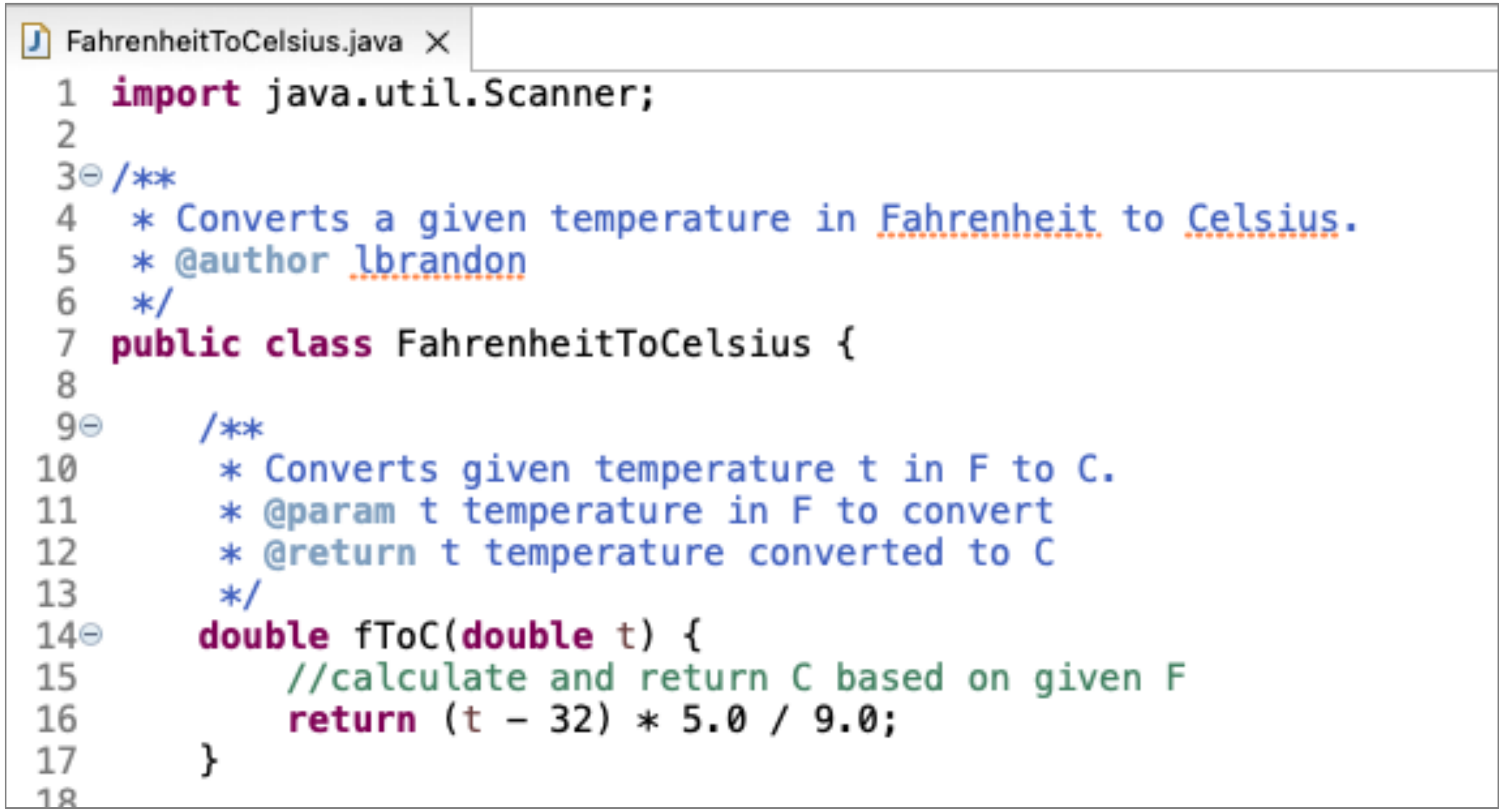
- Use the method *absoluteValue*
    - Pass -9 as an *argument*



```

        //call absoluteValue method using the instance of MethodsDemo class
        int absValRes = demo.absoluteValue(-9);
        //get/print return value
        System.out.println(absValRes);
    }
}

```

User-Defined Methods

- You can call one method from within another
- First, define a method *fToC* to convert Fahrenheit to Celsius

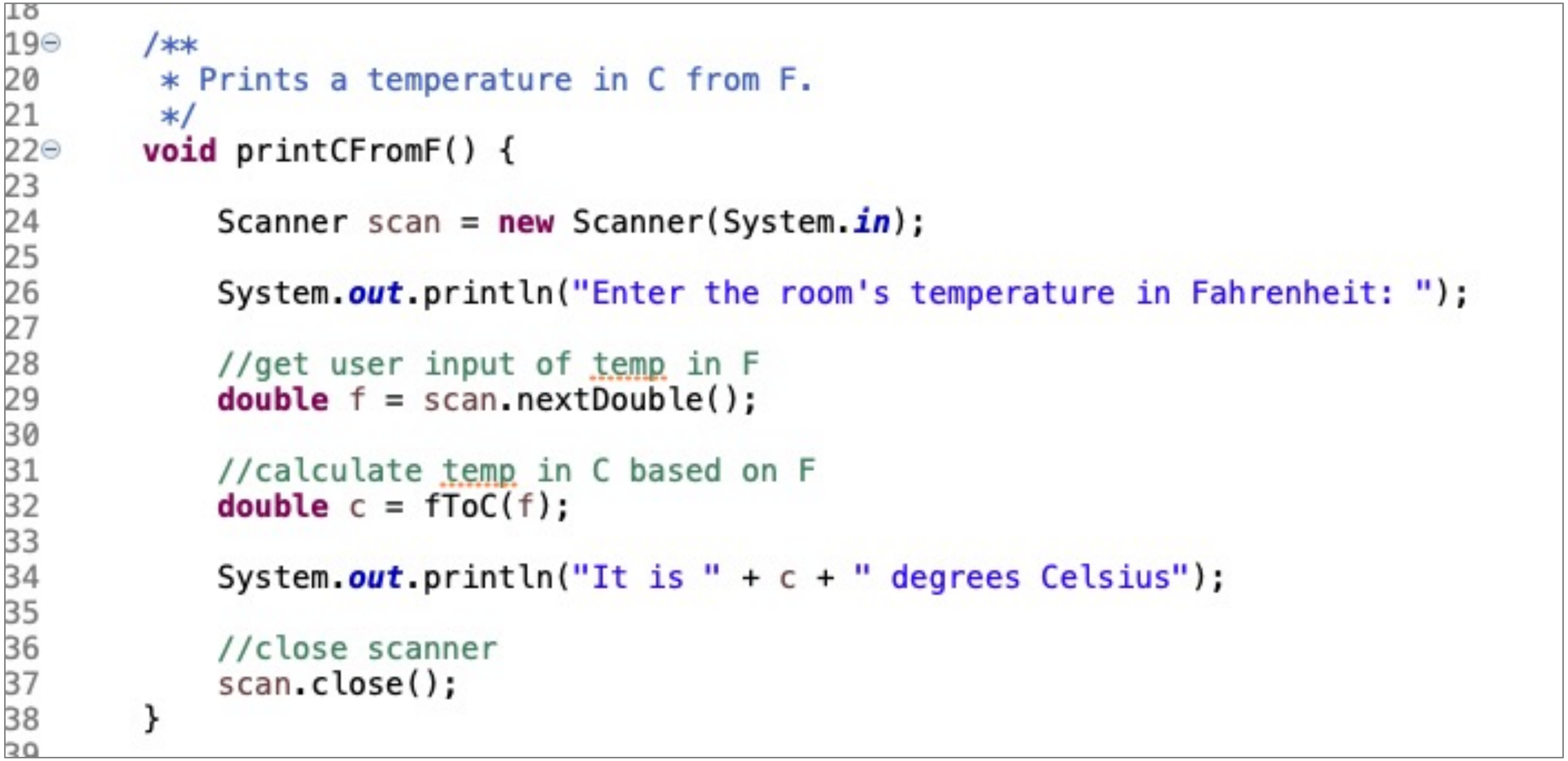


FahrenheitToCelsius.java

```
import java.util.Scanner;

/**
 * Converts a given temperature in Fahrenheit to Celsius.
 * @author lbrandon
 */
public class FahrenheitToCelsius {

    /**
     * Converts given temperature t in F to C.
     * @param t temperature in F to convert
     * @return t temperature converted to C
     */
    double fToC(double t) {
        //calculate and return C based on given F
        return (t - 32) * 5.0 / 9.0;
    }

```

User-Defined Methods

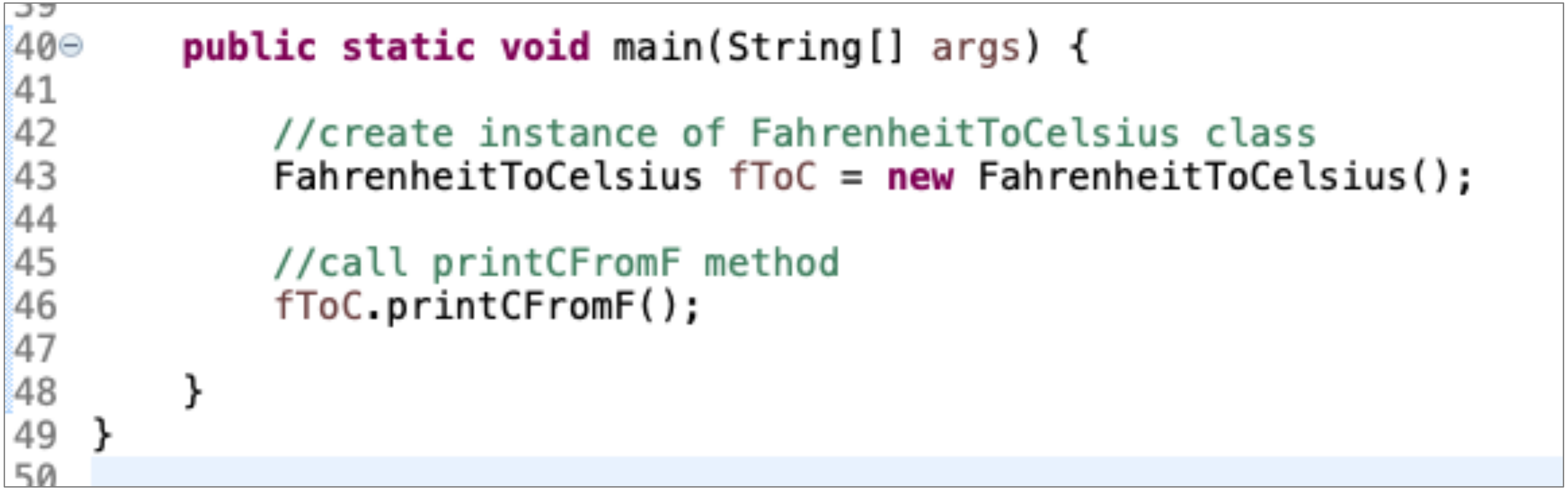
- Then, define a method *printCFromF* that will call *fToC* inside of it



```
    /**
     * Prints a temperature in C from F.
     */
    void printCFromF() {

        Scanner scan = new Scanner(System.in);

        System.out.println("Enter the room's temperature in Fahrenheit: ");

        //get user input of temp in F
        double f = scan.nextDouble();

        //calculate temp in C based on F
        double c = fToC(f);

        System.out.println("It is " + c + " degrees Celsius");

        //close scanner
        scan.close();
    }
```

User-Defined Methods

- Then, call the *printCFromF* method!



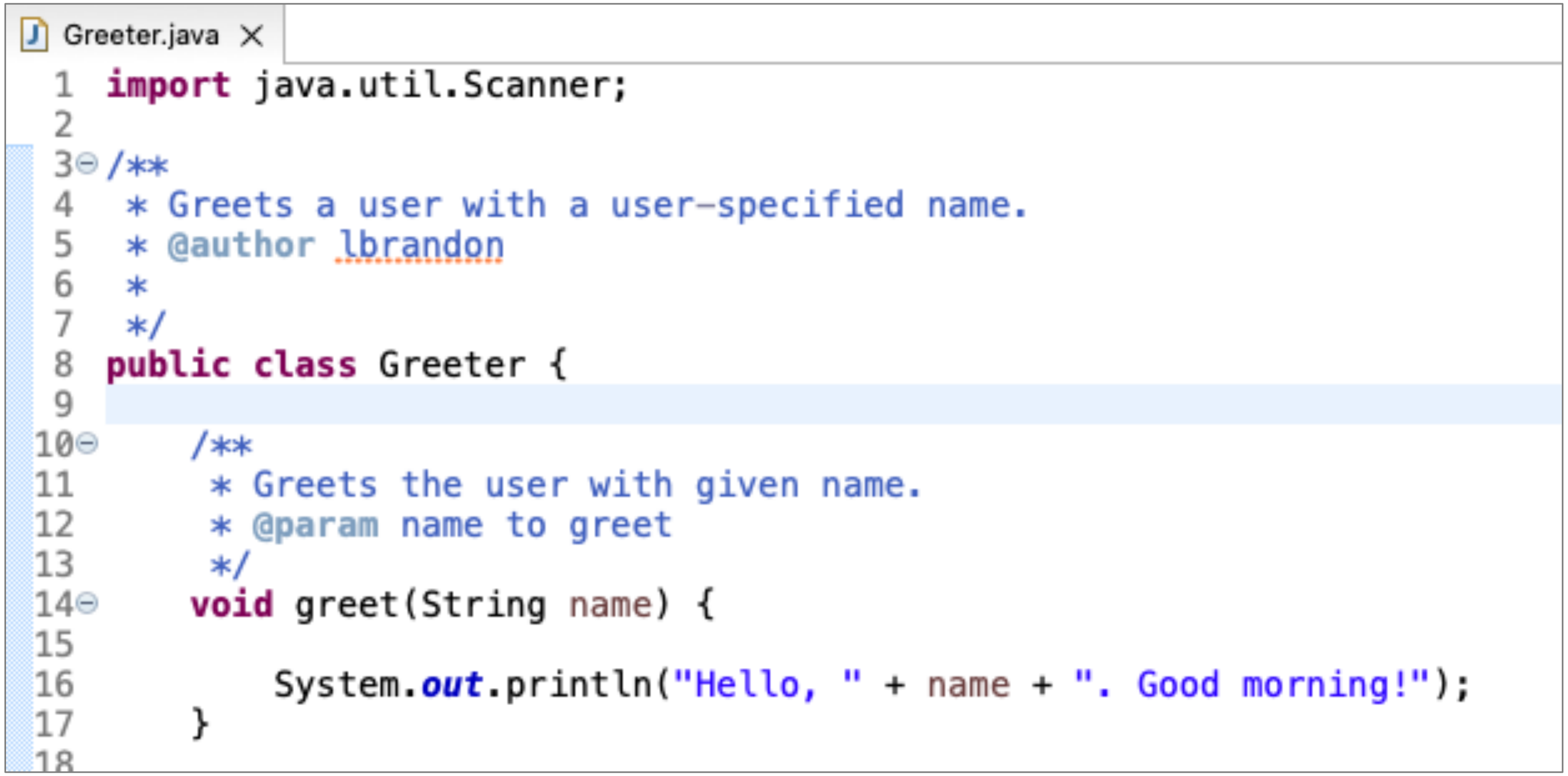
```
    public static void main(String[] args) {

        //create instance of FahrenheitToCelsius class
        FahrenheitToCelsius fToC = new FahrenheitToCelsius();

        //call printCFromF method
        fToC.printCFromF();

    }
}
```

User-Defined Methods

- Define a method *greet* that takes a user's name (as a string) as a *parameter* and greets them by printing something like "Hello, Karen. Good morning!"

User-Defined Methods

- Define a method *greet* that takes a user's name (as a string) as a *parameter* and greets them by printing something like "Hello, Karen. Good morning!"

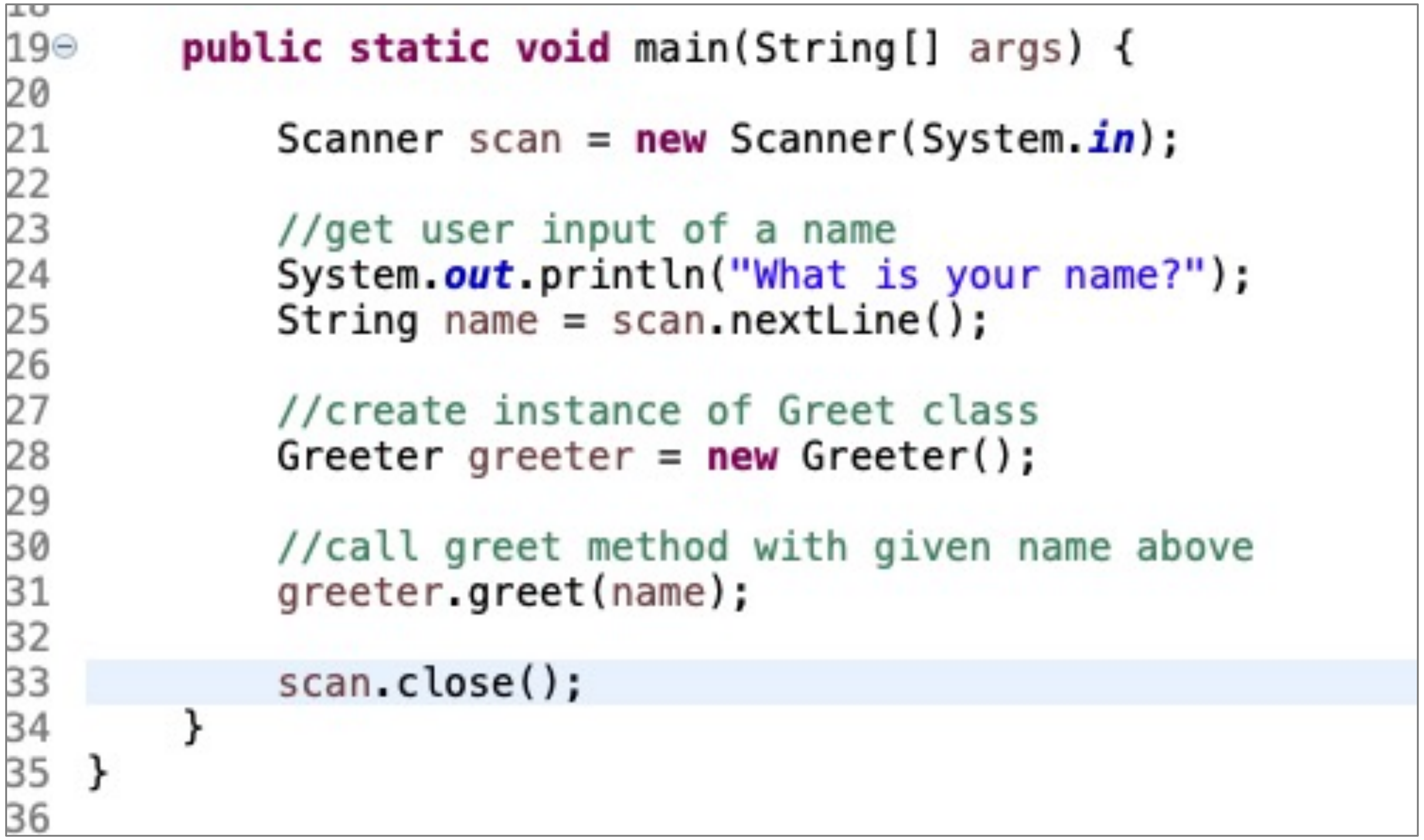


Greeter.java

```java
import java.util.Scanner;

/**
 * Greets a user with a user-specified name.
 * @author lbrandon
 *
 */
public class Greeter {

    /**
     * Greets the user with given name.
     * @param name to greet
     */
    void greet(String name) {

        System.out.println("Hello, " + name + ". Good morning!");
    }

```

User-Defined Methods

- Define a method *greet* that takes a user's name (as a string) as a *parameter* and greets them by printing something like "Hello, Karen. Good morning!"
- Get the user to input their name and pass that as an *argument* to the *greet* method



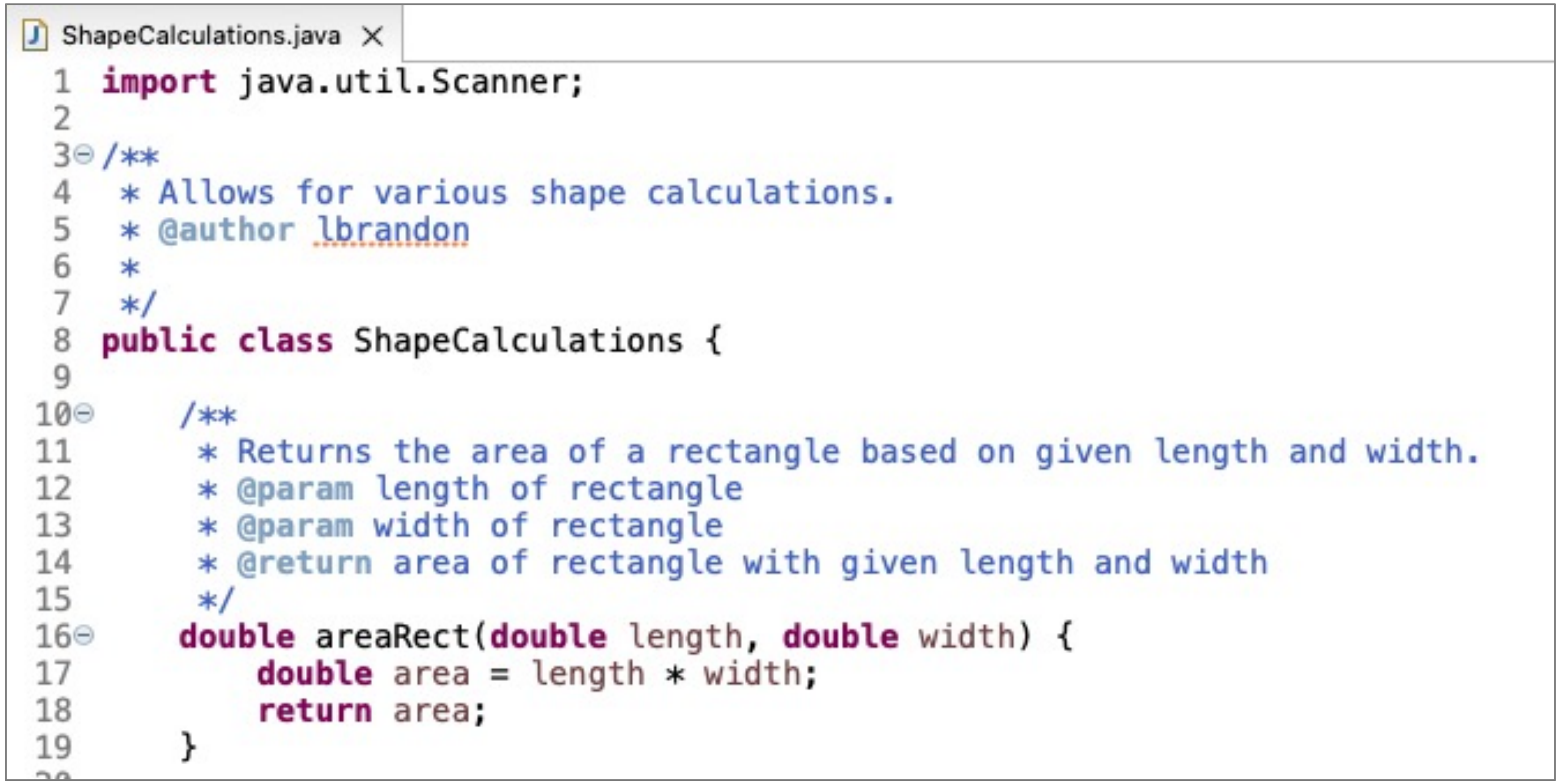
```
    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        //get user input of a name
        System.out.println("What is your name?");
        String name = scan.nextLine();

        //create instance of Greet class
        Greeter greeter = new Greeter();

        //call greet method with given name above
        greeter.greet(name);

        scan.close();
    }
}

```

User-Defined Methods

- Define a method *areaRect* that computes the area of a rectangle.
  - It takes two doubles (length, width) as *parameters*
  - It *returns* the area

User-Defined Methods

- Define a method *areaRect* that computes the area of a rectangle.
  - It takes two doubles (length, width) as *parameters*
  - It *returns* the area

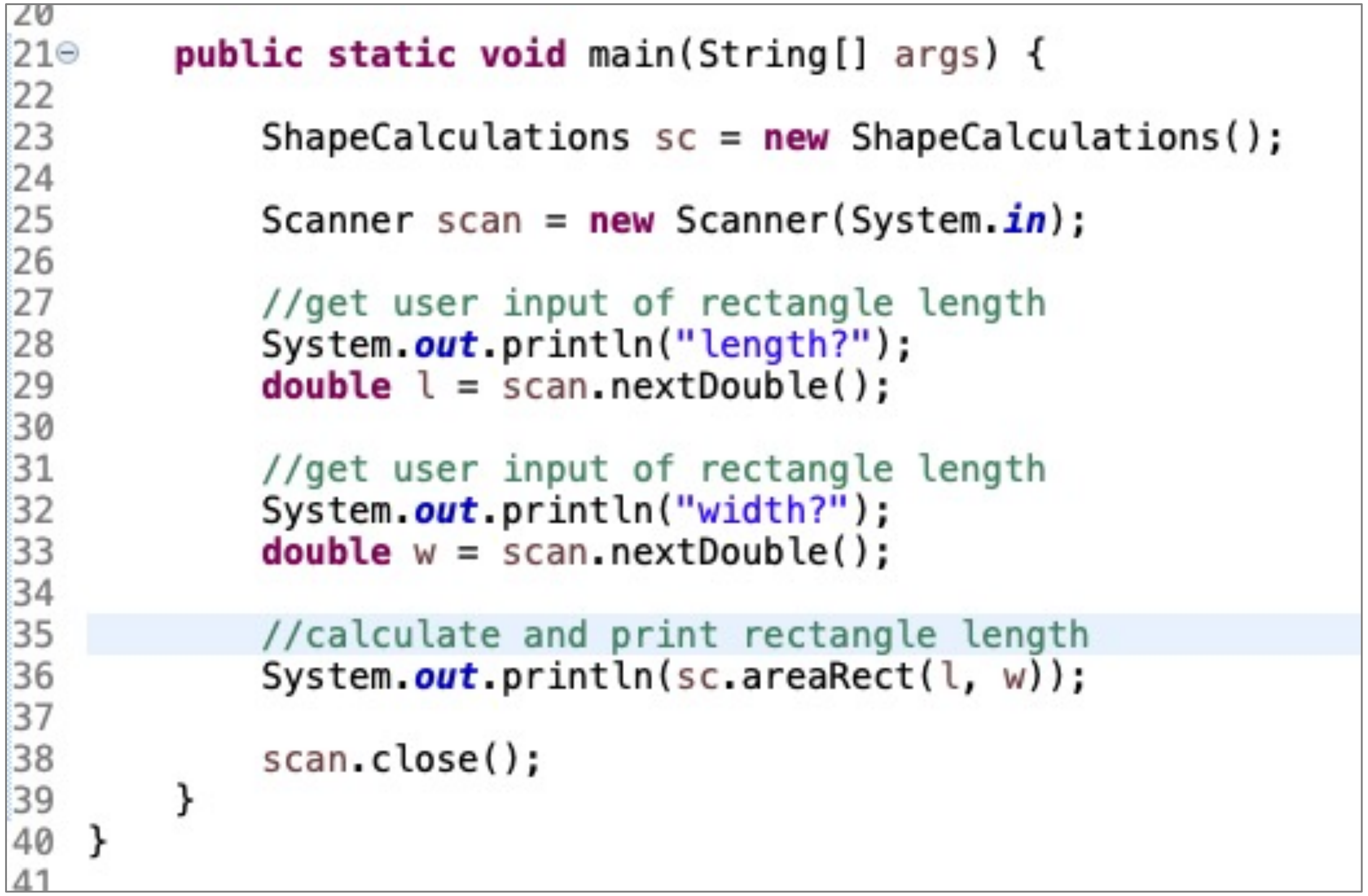


ShapeCalculations.java

```java
import java.util.Scanner;

/**
 * Allows for various shape calculations.
 * @author lbrandon
 *
 */
public class ShapeCalculations {

    /**
     * Returns the area of a rectangle based on given length and width.
     * @param length of rectangle
     * @param width of rectangle
     * @return area of rectangle with given length and width
     */
    double areaRect(double length, double width) {
        double area = length * width;
        return area;
    }
```

User-Defined Methods

- Get the user to input a *length* value and a *width* value, and pass those as *arguments* to the *areaRect* method



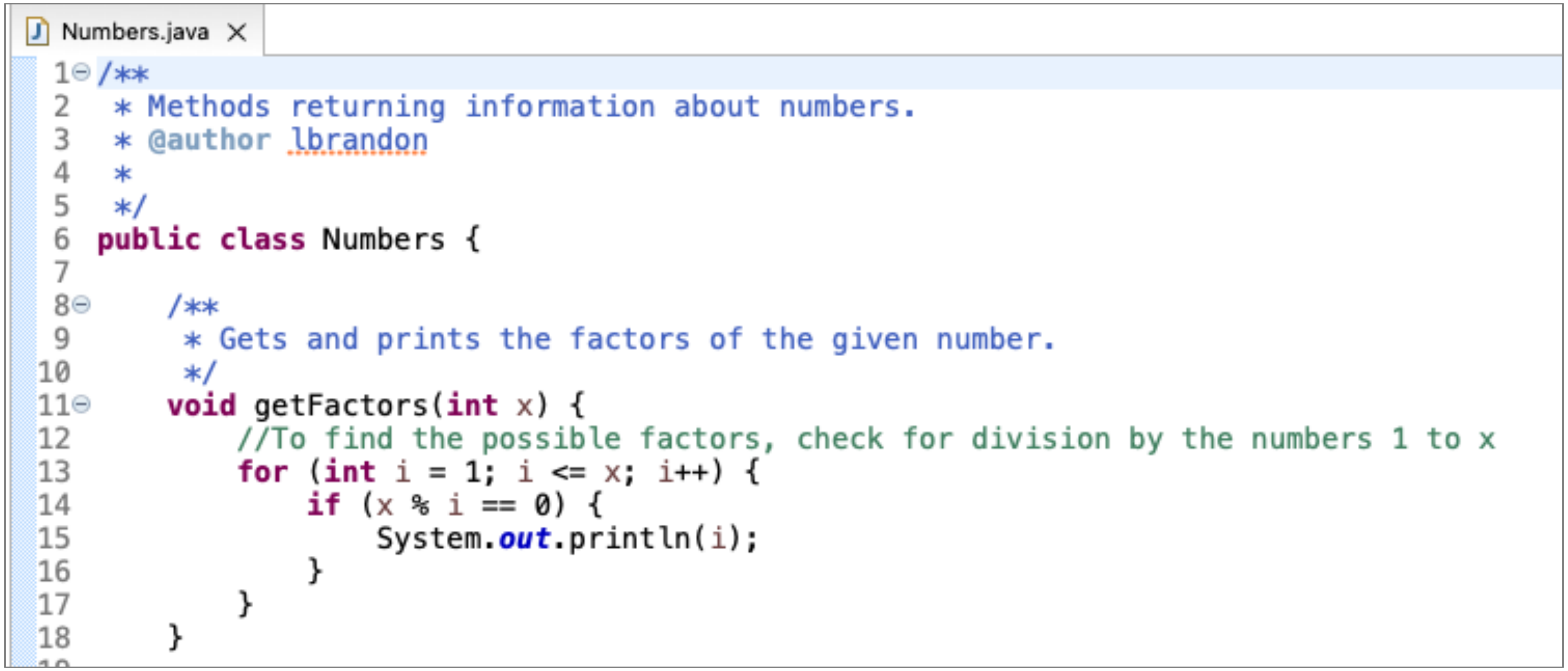
```
    public static void main(String[] args) {

        ShapeCalculations sc = new ShapeCalculations();

        Scanner scan = new Scanner(System.in);

        //get user input of rectangle length
        System.out.println("length?");
        double l = scan.nextDouble();

        //get user input of rectangle length
        System.out.println("width?");
        double w = scan.nextDouble();

        //calculate and print rectangle length
        System.out.println(sc.areaRect(l, w));

        scan.close();
    }
}
```

User-Defined Methods

- Define a method *getFactors* that takes an int as a *parameter* and prints the factors of that number
  - Basically, find the numbers between 1 and the given integer that divide the number evenly

User-Defined Methods

- Define a method *getFactors* that takes an int as a *parameter* and prints the factors of that number
    - Basically, find the numbers between 1 and the given integer that divide the number evenly

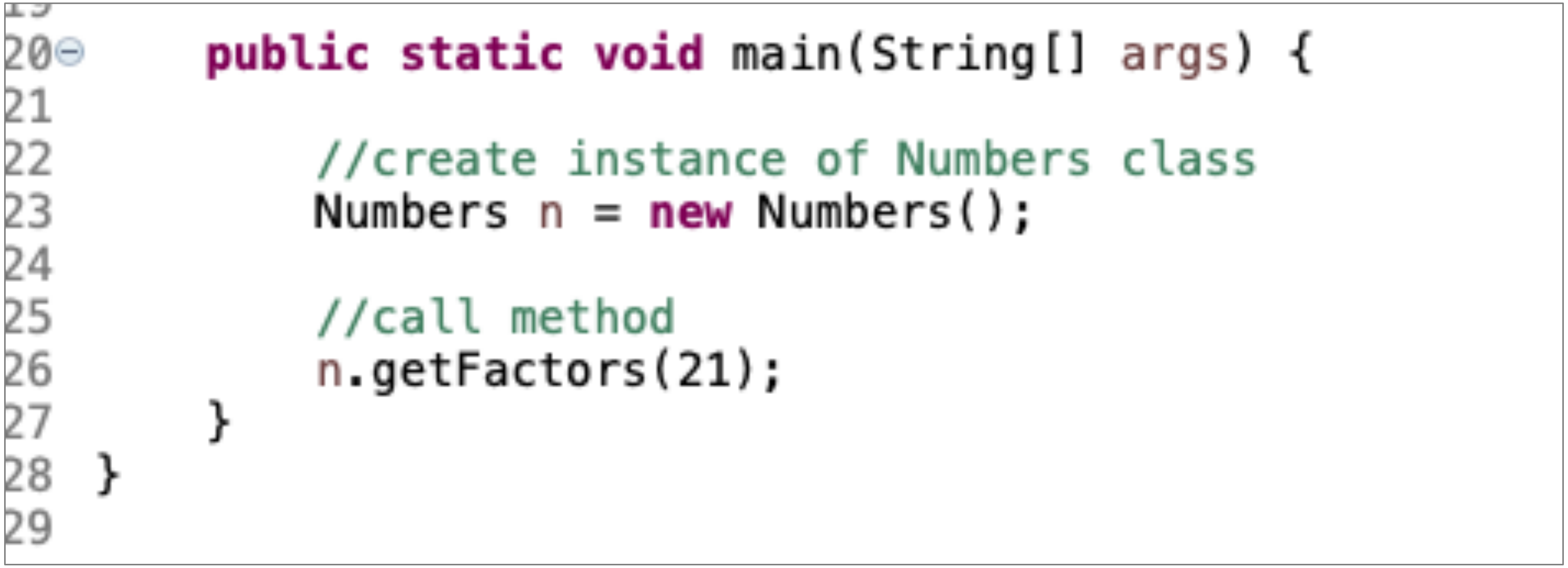


Numbers.java

```java
/**
 * Methods returning information about numbers.
 * @author lbrandon
 *
 */
public class Numbers {

    /**
     * Gets and prints the factors of the given number.
     */
    void getFactors(int x) {
        //To find the possible factors, check for division by the numbers 1 to x
        for (int i = 1; i <= x; i++) {
            if (x % i == 0) {
                System.out.println(i);
            }
        }
    }
```

User-Defined Methods

- Define a method *getFactors* that takes an int as a *parameter* and prints the factors of that number
    - Basically, find the numbers between 1 and the given integer that divide the number evenly



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

        //create instance of Numbers class
        Numbers n = new Numbers();

        //call method
        n.getFactors(21);
    }
}

```

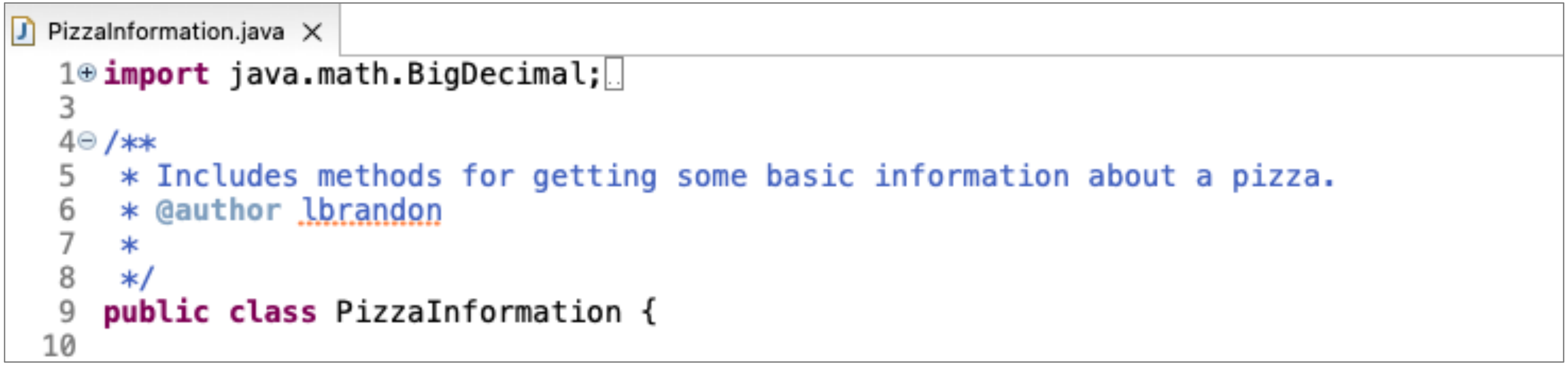
Pizza Information Program

Pizza Information

- Create a program that allows for getting (and printing) basic information about a pizza of a particular size
  - Create method(s) to calculate the area of a pizza with a given diameter
  - Create method(s) to calculate the total calories of a pizza with a given area

Pizza Information

- Create a program that allows for getting (and printing) basic information about a pizza of a particular size
  - Create method(s) to calculate the area of a pizza with a given diameter
  - Create method(s) to calculate the total calories of a pizza with a given area

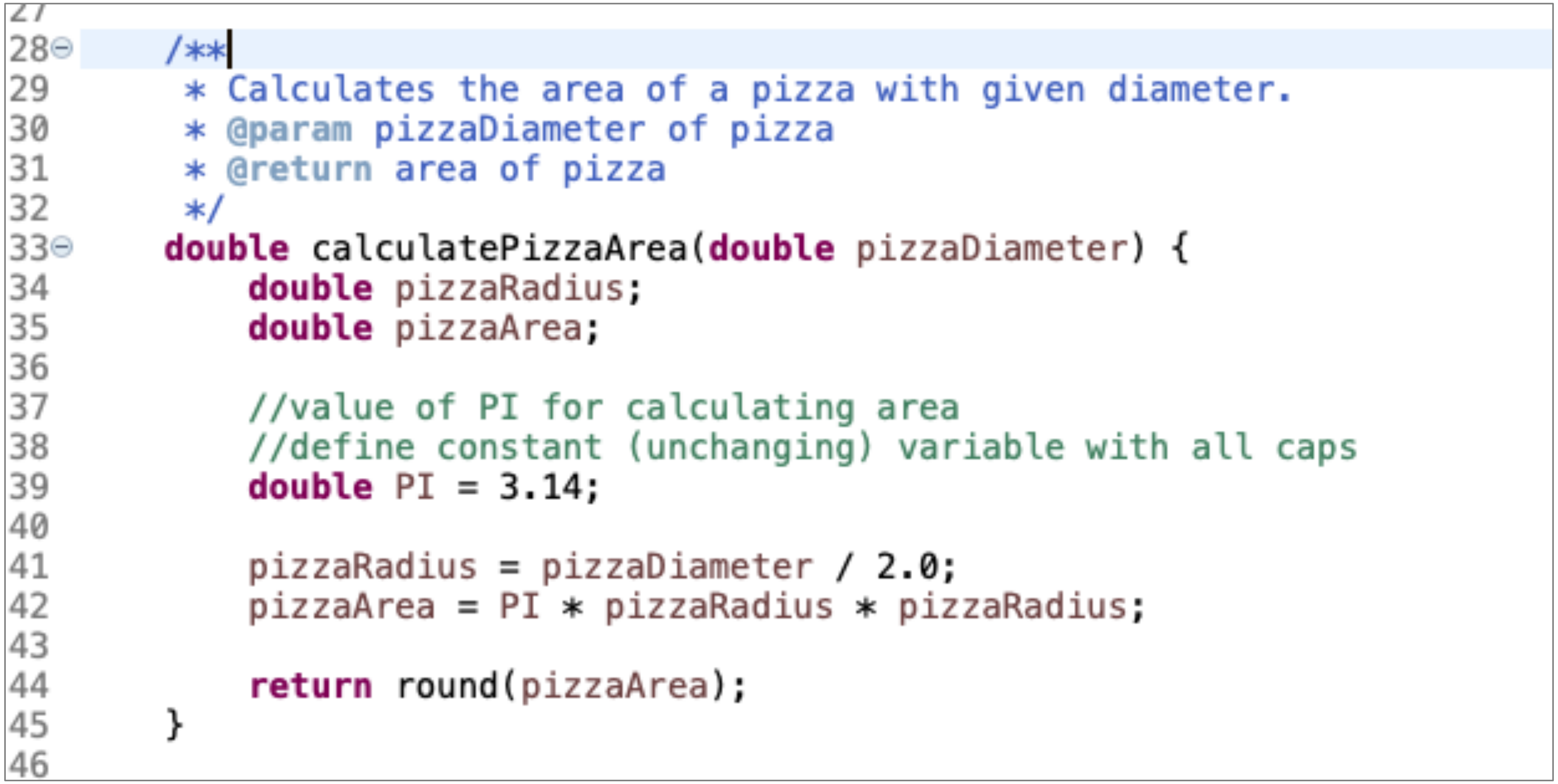


PizzaInformation.java

```java
import java.math.BigDecimal;

/**
 * Includes methods for getting some basic information about a pizza.
 * @author lbrandon
 *
 */
public class PizzaInformation {

```

Pizza Information

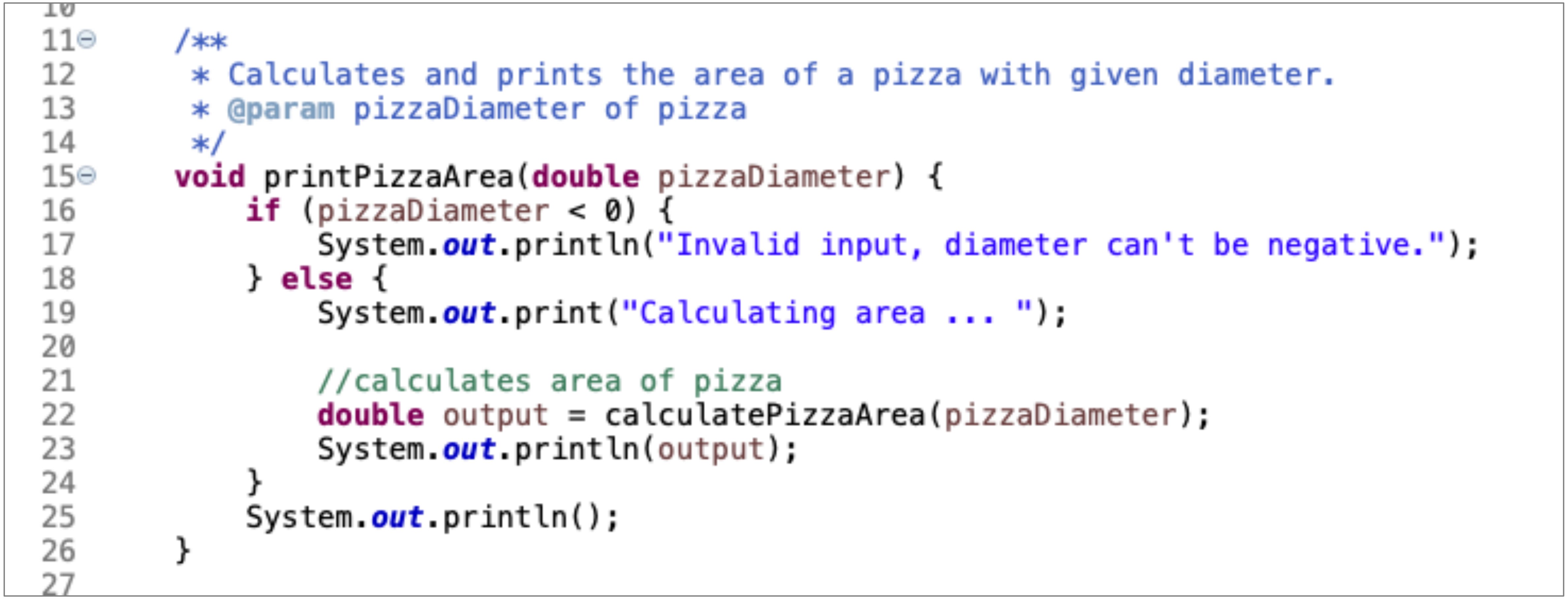
- Create *calculatePizzaArea* method



```
    /**
     * Calculates the area of a pizza with given diameter.
     * @param pizzaDiameter of pizza
     * @return area of pizza
     */
    double calculatePizzaArea(double pizzaDiameter) {
        double pizzaRadius;
        double pizzaArea;

        //value of PI for calculating area
        //define constant (unchanging) variable with all caps
        double PI = 3.14;

        pizzaRadius = pizzaDiameter / 2.0;
        pizzaArea = PI * pizzaRadius * pizzaRadius;

        return round(pizzaArea);
    }
```

Pizza Information

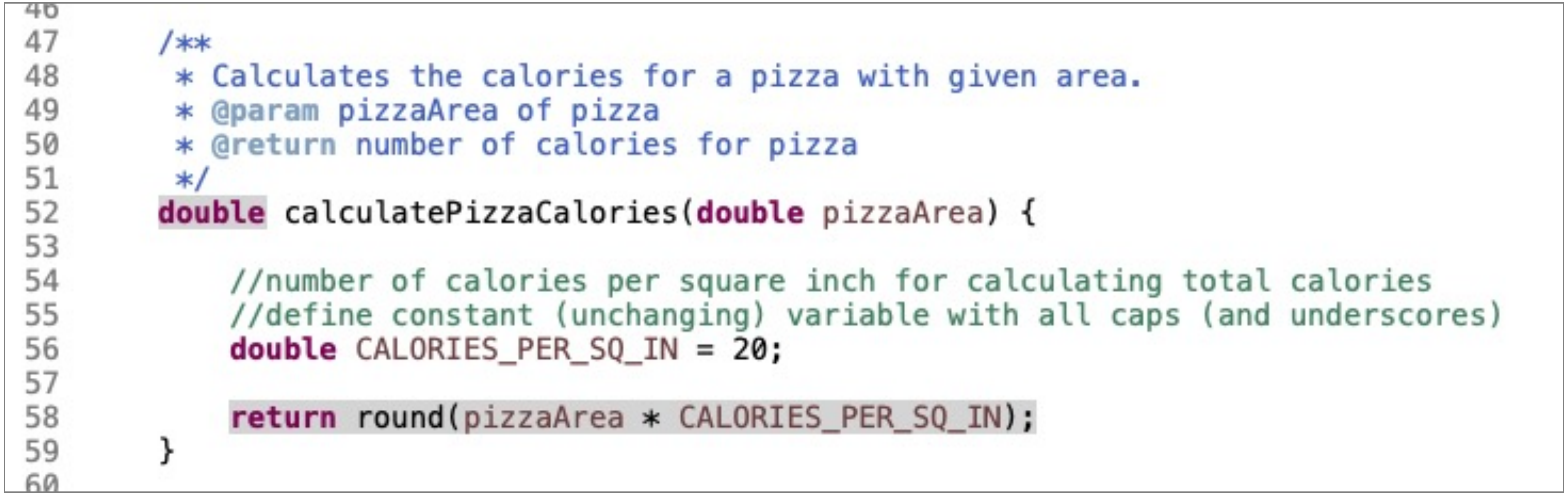
- Create *printPizzaArea* method



```
    /**
     * Calculates and prints the area of a pizza with given diameter.
     * @param pizzaDiameter of pizza
     */
    void printPizzaArea(double pizzaDiameter) {
        if (pizzaDiameter < 0) {
            System.out.println("Invalid input, diameter can't be negative.");
        } else {
            System.out.print("Calculating area ... ");

            //calculates area of pizza
            double output = calculatePizzaArea(pizzaDiameter);
            System.out.println(output);
        }
        System.out.println();
    }
```

Pizza Information

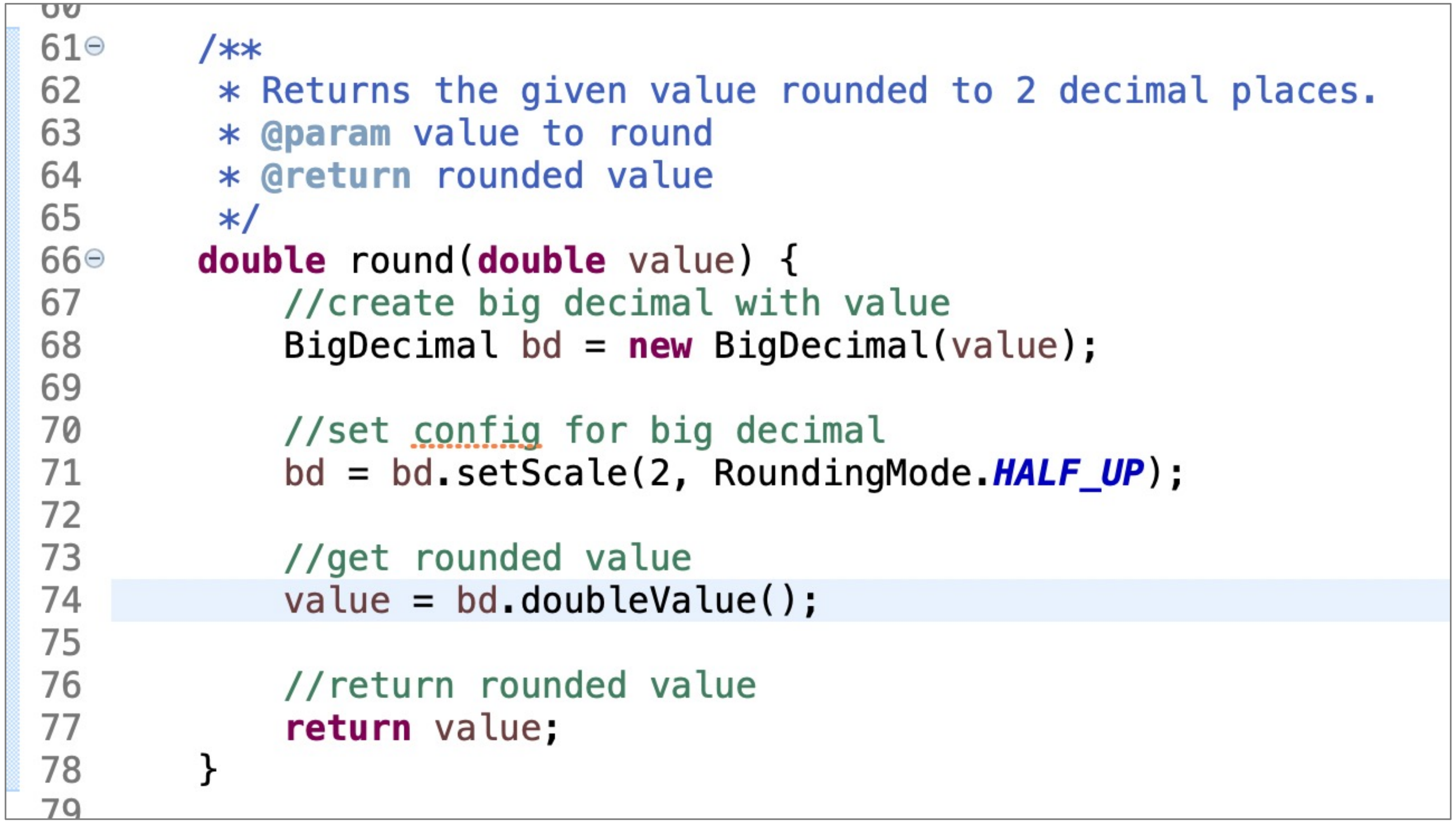
- Create *calculatePizzaCalories* method



```java
    /**
     * Calculates the calories for a pizza with given area.
     * @param pizzaArea of pizza
     * @return number of calories for pizza
     */
    double calculatePizzaCalories(double pizzaArea) {

        //number of calories per square inch for calculating total calories
        //define constant (unchanging) variable with all caps (and underscores)
        double CALORIES_PER_SQ_IN = 20;

        return round(pizzaArea * CALORIES_PER_SQ_IN);
    }
```

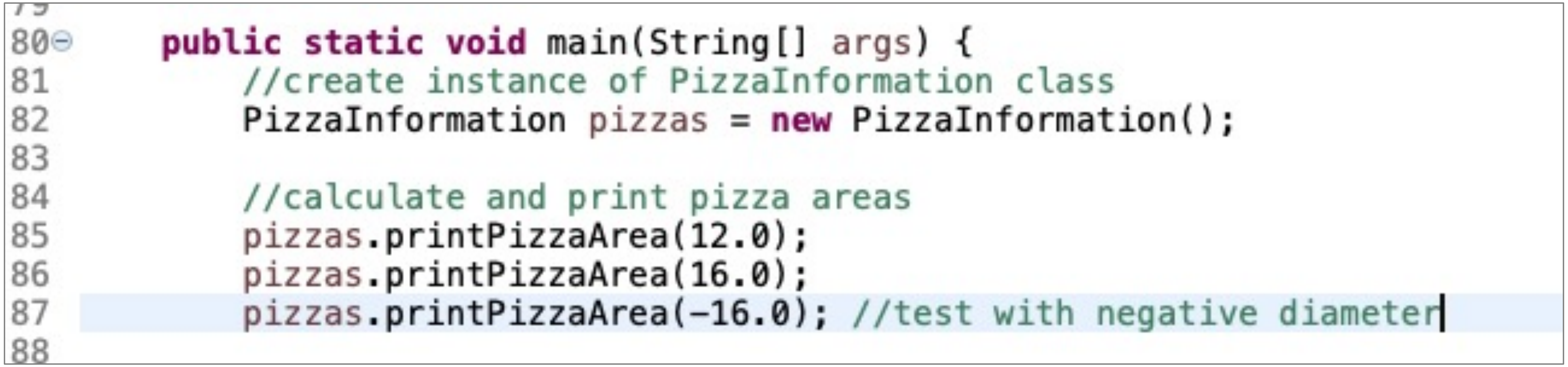
Pizza Information

- Create *round* method



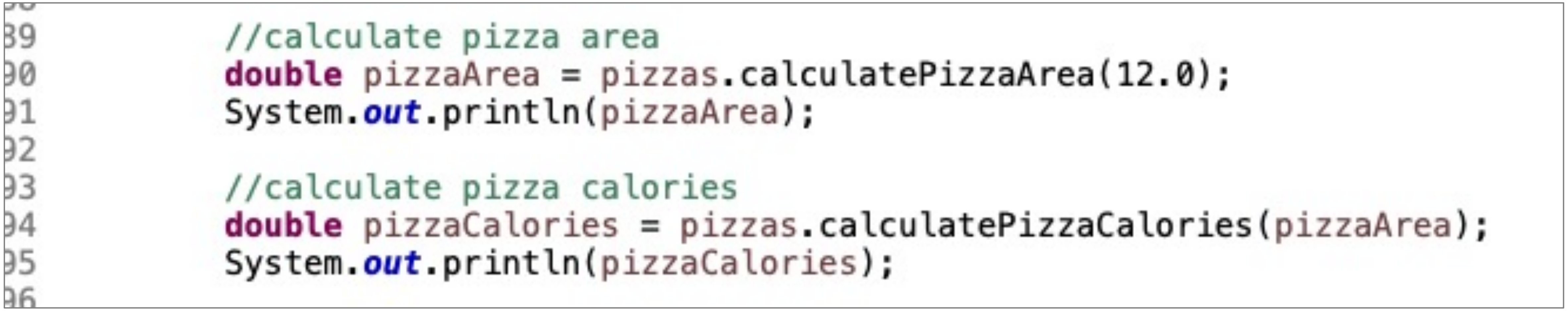
```
    /**
     * Returns the given value rounded to 2 decimal places.
     * @param value to round
     * @return rounded value
     */
    double round(double value) {
        //create big decimal with value
        BigDecimal bd = new BigDecimal(value);

        //set config for big decimal
        bd = bd.setScale(2, RoundingMode.HALF_UP);

        //get rounded value
        value = bd.doubleValue();

        //return rounded value
        return value;
    }
```

Pizza Information



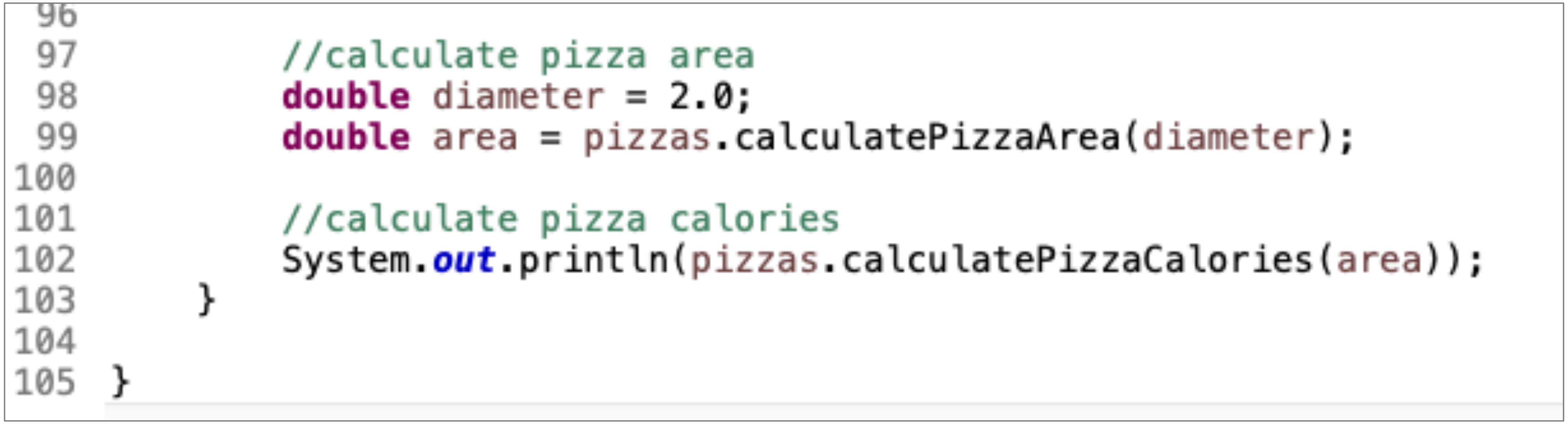
```
    public static void main(String[] args) {
        //create instance of PizzaInformation class
        PizzaInformation pizzas = new PizzaInformation();

        //calculate and print pizza areas
        pizzas.printPizzaArea(12.0);
        pizzas.printPizzaArea(16.0);
        pizzas.printPizzaArea(-16.0); //test with negative diameter

```

Pizza Information



```
        //calculate pizza area
        double pizzaArea = pizzas.calculatePizzaArea(12.0);
        System.out.println(pizzaArea);

        //calculate pizza calories
        double pizzaCalories = pizzas.calculatePizzaCalories(pizzaArea);
        System.out.println(pizzaCalories);
```

Pizza Information



```java

            //calculate pizza area
            double diameter = 2.0;
            double area = pizzas.calculatePizzaArea(diameter);

            //calculate pizza calories
            System.out.println(pizzas.calculatePizzaCalories(area));
        }

}
```

Homework 4

Homework 4

Will be assigned by tonight, Wednesday, September 21st at midnight and due Monday, October 3rd at midnight

- In this assignment, you will implement **a simplified version of a common card game called "21".** Blackjack is a better-known variant.
- The topics are:
  - Loops
  - Strings
  - Classes & Methods
- To complete the assignment:
  - Submit your completed *.java* file to Canvas