Java Basics Iteration 4

Previous Iteration	Wiki Home	Forums
Iteration 2	Wiki Home Page	Forums

Previous Iteration	tWiki Home
Iteration 3	tWiki Home Page

Table of Contents

Iteration Overview

During this iteration, students will learn about classes and instances in Java, in addition to completing a programming lab.

A note about the final demo: As you progress through the labs, save screenshots along the way to share with a Topic Advisor.

Classes in Java

We will continue to talk about program design and about how we can make our programs readable to other software developers. As you enter the workforce, there are often several software developers working on the same program, so it's essential to write code in a way that's easy to read and understand. A lot of times, I'll forget about code that I've written six months ago. And if it's written in a way where it's hard to understand, I have to take a significant amount of time to understand it before I can add functionality to it. We want to prevent that.

One way to make our program easier to read and understand is with classes. We've seen classes a little bit before because all of our code so far has been contained within the main class. But now we are going to dive a little deeper. Everything we see, hear and experience in everyday life can be represented in code. We've represented the on-repeat functionality of a music player. The final total that may be on a restaurant receipt and more. How well something is represented is up to us and we decide what to code. Although we've used functions and other tools to represent these in code, most of the time, classes act as blueprints for these things. A class is a user-defined blueprint that has a set of attributes and behaviors that define the item that it is supposed to represent.

For example, let's say we wanted to create a class or blueprint for a triangle. When we think of a triangle, it has a base, a height, and three sides with various lengths. These are attributes or properties of a triangle, so we can define them as that for our blueprint or class. Each of these attributes will define a data type and they will act as a variable in the blueprint. The base, height and sides of the triangle will have double values. Every triangle we create will have a base, height, and side lengths. The values may not be the same, but every triangle has these attributes, and these attributes are really variables that will hold a particular value for a given triangle.

A class can also have behavior. The behaviors are defined as functions that are related to the class. In the debugging chapter, we debugged some code that calculated the area of a triangle. We could turn that code into a function called findArea and add that as a behavior to our triangle class. The functionality would be the same, but it's nice to make it a function so we can reuse it as well as organize it by putting it in our Triangle class. Another name for a behavior is a method. We could say that findArea is a method of the Triangle class. We could also add a behavior or function to this class that decides if the triangle is an equilateral, isosceles or scalene triangle. An equilateral triangle has equal side lengths. An isosceles triangle has two side lengths that are equal, and a scalene triangle has all unique side lengths. With if statements and some comparisons with the side length attributes, we could implement a function called calculateTriangleType that returns the type of triangle. One could argue that the fact the triangle is equilateral, isosceles or scalene, is more of an attribute or property rather than something to be calculated. But this is just one programmatic design.

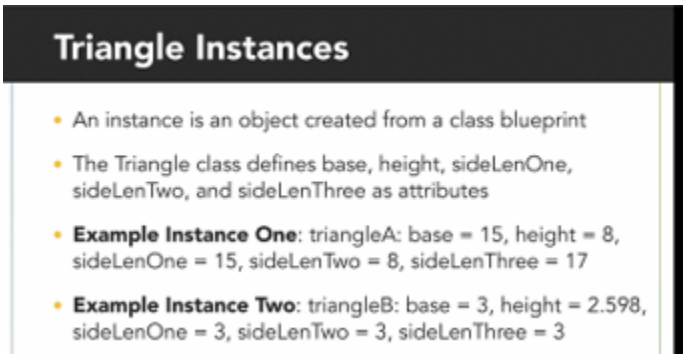
There are a bunch of different ways you could represent a triangle in code, it just depends on what functionality you want and what data you want to store. For the behaviors we've talked about so far, they have manipulated or used attributes of the class. findArea used the triangle's base and height and calculateTriangleType used the side lengths of the triangle. Although functions inside the class or behaviors often use a class's properties, they don't have to. We add the keyword static if the function does not use the properties of the class but still relates to the overall theme or idea of the class. We'll dive into the keyword static later on in this chapter.

Now that we understand classes conceptually, we can connect these blueprints to the execution of our programs.

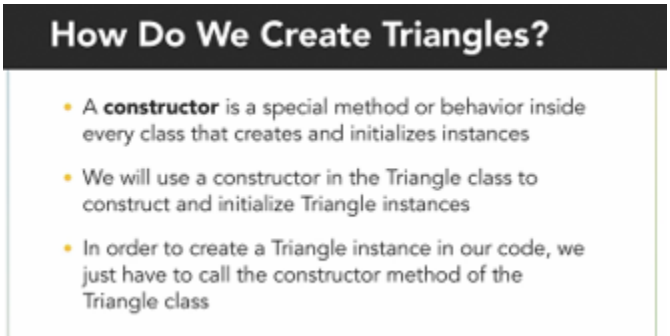
Constructors in Java

When we were talking about functions in an earlier chapter, we had to define a function in order to call it. Similarly, we have to define a blueprint or class in order to use it. In the last lesson, we defined the Triangle class or triangle blueprint. But we haven't created any triangles yet. We just defined the attributes and behavior of a triangle in a blueprint. In this lesson, we are going to learn how to create individual triangle instances in our program.

An instance is an object created from the class blueprint. Just like we make buildings from blueprints in real life, we use a class or blueprint to create objects or instances in code. Thinking back to our triangle class, a triangle has a base height and three side lengths. An instance of a triangle, say named triangleA, could have a base 15, height 8, and sides with lengths 15, 8, and 17. Another instance of a triangle could have base 3, height 2.598, and side lengths 3, 3, and 3. From our single Triangle class, we can create as many instances as we want, and each instance will have its own base, height, and side length attribute variables.



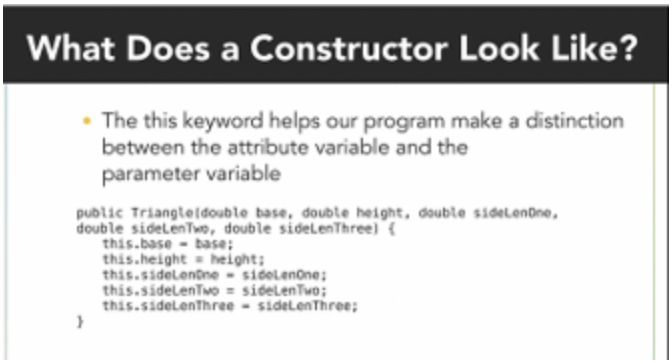
So how do we create these instances? We use a special method or behavior called a constructor. Every class has a constructor, and it's how we create and initialize each triangle we want to use in our program. It's how we can create triangleA with specific base, height, and side length values, and then construct triangleB with different attribute values. We call this special function inside the Triangle class.



What does a constructor look like? Since it's a function, it has a name, and that name is always the same as the name of the class, so in this case, triangle. Then we have open-close parentheses for the inputs to the function and brackets for the implementation of the function. Right now, they are empty. For the return type, the constructor always returns an instance of the class, in this case an instance of triangle. We don't have to state it in the definition of the method because it's already built into Java to expect that behavior. In the implementation, we won't be using the return keyword either because no matter what we do in the constructor, it will always return an instance of the class, in this case Triangle.

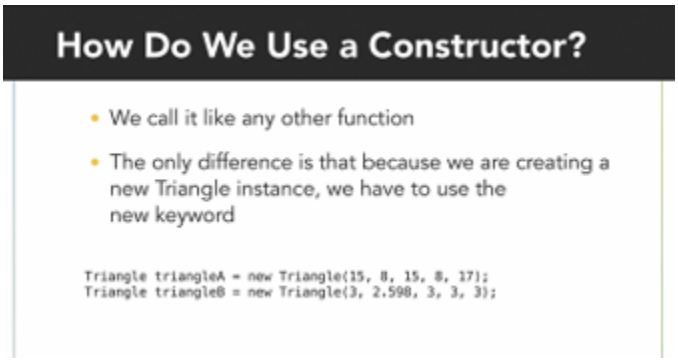
In constructing a Triangle instance or Triangle object we'll want to initialize its attributes with values. We'll want to give a value to the base, the height, and the side lengths. We could set these to any default value but ideally this constructor would be dynamic, and on-the-fly decide what values to give these attributes. To do this, we can add parameters representing each triangle attribute to this function. Just like our parameters to other functions, each parameter has a data type and a name. Each parameter does not have to have the same name as the attribute their data represents, but it is often the case that this is so. This might look a little intimidating but stick with me. This was tough for me too when I first got started with Java.

With parameters, we have access to the appropriate values we want to assign to each of the attributes. To access the attribute variable for the object we are trying to construct, we use the "this" keyword and the dot operator. The "this" keyword helps our program make a distinction between the attribute variable and the parameter variable. We use the word "this" because we are talking about an attribute of the triangle we are constructing rather than the parameter variable.

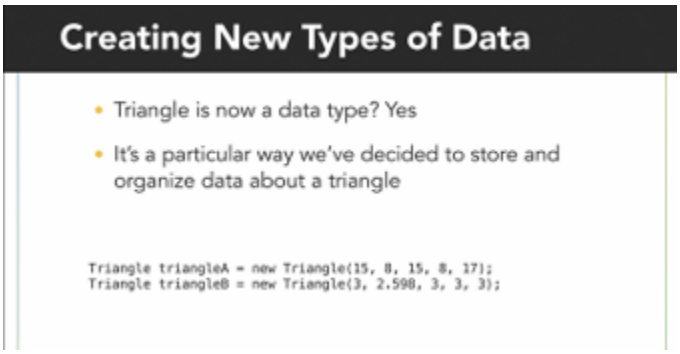


When we use the dot operator on the keyword "this," we can access the to-be-created instance's attribute variables. This allows us to access the instance we are creating's base, height, and side lengths so we can initialize them. With the code below, we are initializing the attributes of the new triangle with the values of the corresponding parameters. In the constructor, we initialize the triangle we are creating's base attribute with the value given in the base parameter. We also initialize the new triangle's height attribute with the value given in the height parameter and so on for each attribute of the triangle we are creating. And that's our constructor function defined. How do we use it? Just like any other function, we'll call it with argument values, and it will return a triangle. The only difference is that because we are creating a new triangle object or a new triangle instance we have to use the new keyword.

In this example, we create two variables with the data type Triangle. One triangle is named triangleA and the other is called triangleB. Then we use the new keyword and call the constructor with the appropriate attribute values. It returns a triangle that is now stored in the respective variable. The triangleA instance will have a base with the value 15, height with the value 8, side length one with 15, side length two with 8, and side length three with 17. The triangleB instance will have a base with the value 3, a height of 2.598, and all side lengths with value 3.



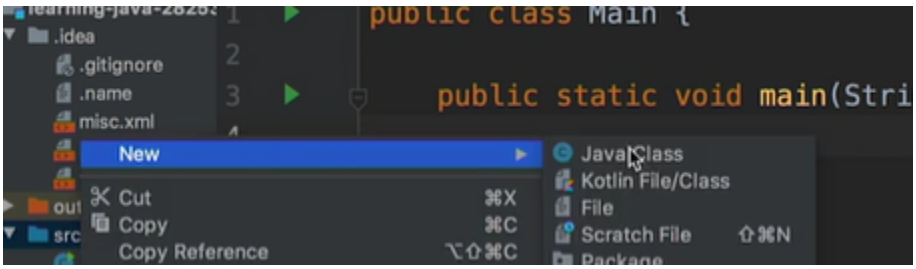
Now you might be thinking, "Triangle is now a data type?" And yes, it is. When we create a class, we are essentially creating a new way to store a group of data. Just think about all those double values we are storing. Our class is one particular way we've decided to store and organize data about a triangle. This means when we create a triangle, store it in a variable or perhaps even use it as a parameter, we must remember to use Triangle as our data type.



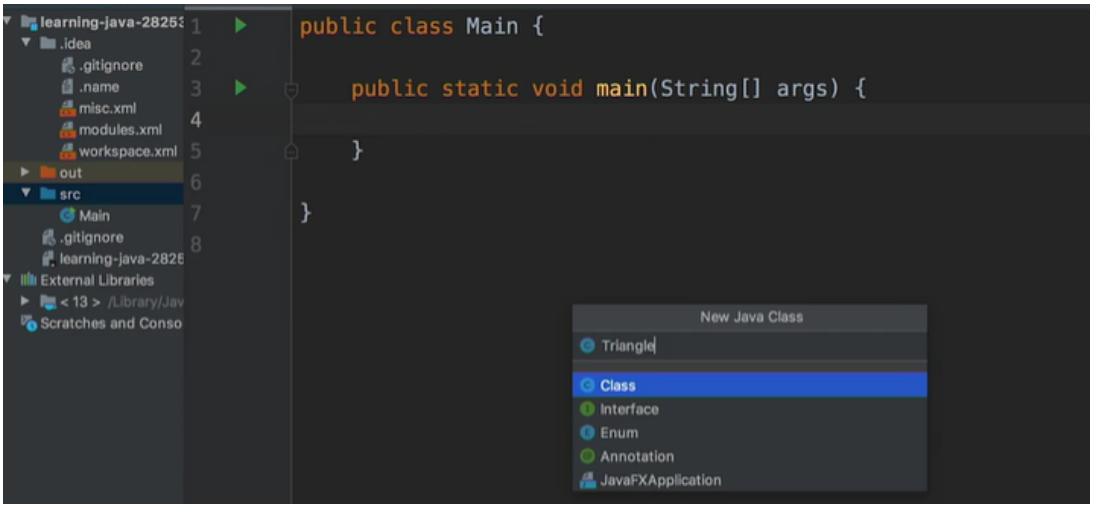
Let's try pulling all of this together by defining our first class in Java and creating instances from it using a constructor.

Creating a class in Java

With an understanding of classes, instances and the constructor, we can begin to write Java Code that helps us represent a triangle in a program. First, we need to write code that defines our triangle blueprint. And we can do that with the triangle class.



In Java, we'll create a new class file and name it triangle.

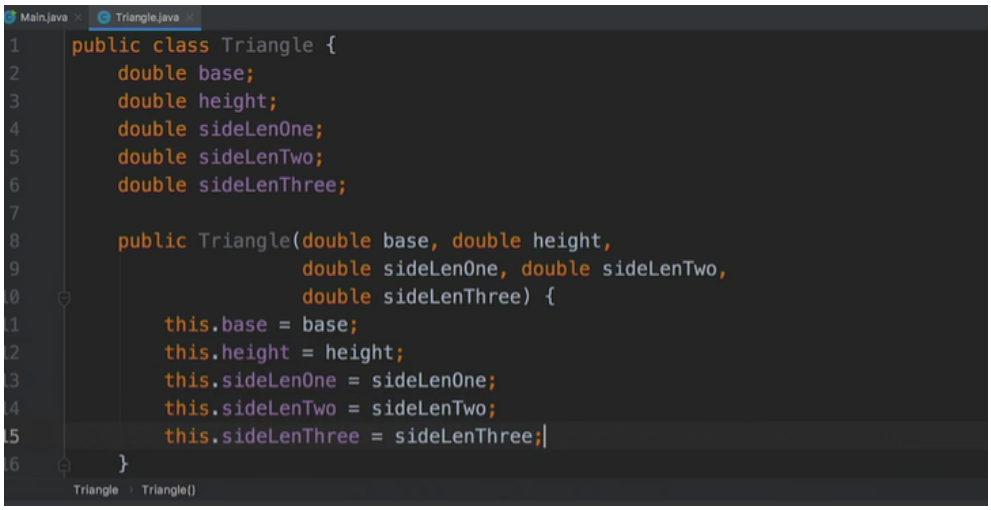


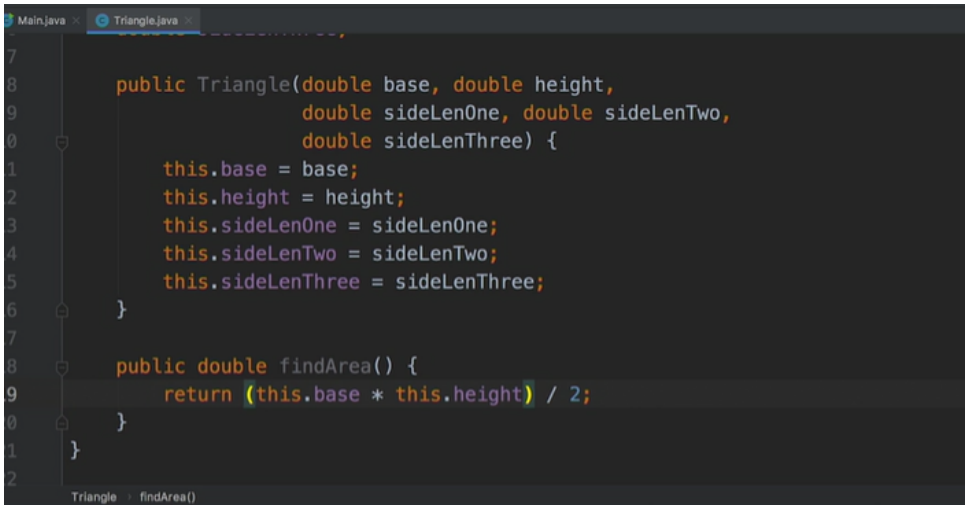
Inside the curly braces for our class, we'll need to add the attributes and behavior we want the class to have. In earlier lessons, we said that a triangle has a base, height and three different side links. We can add those inside of our class. We now have five attribute variables that we have created, but they do not have a value yet.

These attribute variables are also called instance variables because they are variables that each instance will have an individual value for. We could assign a default value to these instance variables, but as we discussed in the last lesson, usually, we want a constructor to initialize them. With the constructor, we can create a triangle instance with specific base, height and side length values. To create a constructor, we'll write "public, triangle", and then our inputs. Next, we'll use these inputs to assign values to our attributes. To access each attribute, we'll write, "this." And then the attribute's name. And then to assign it a value, the value of one of our inputs, we'll just use the name of that parameter.

You'll notice that as we use each attribute, it becomes highlighted, because now it's in use by the program. It's not just some attribute that's created and never used. If we think of this in relation to scope, the attributes are accessible throughout the triangle class. But the constructor's parameters are only accessible inside the constructor. Since the attribute in the constructor's parameter have the same name, we use this in the dock operator on the attributes to keep them separate. Here, we have a constructor that takes in unique values with parameters and assigns them to their corresponding attribute. The constructor is just one behavior or method that a class can have. But we can have as many methods as we want.

Method is just another name for a behavior or function that belongs to a class. In an earlier lesson, we talked about adding a function that finds the area of a triangle to this class. We can do just that. We'll call this function "find area" and it will return a double. We do need to have the return type here, because find area will not be a constructor. With the "this" keyword, we can get access to the appropriate instance variables based on height, so the function takes no inputs. Our formula for finding the area of a triangle, is base times height divided by two, so we'll return "this.base * this.height/two". We can access "this.base" and "this.height" here because the base and height attributes are created within the triangle class. With the addition of this last behavior, our class is defined. We could add attributes and behavior later on, but this is where our blueprint will end for now.

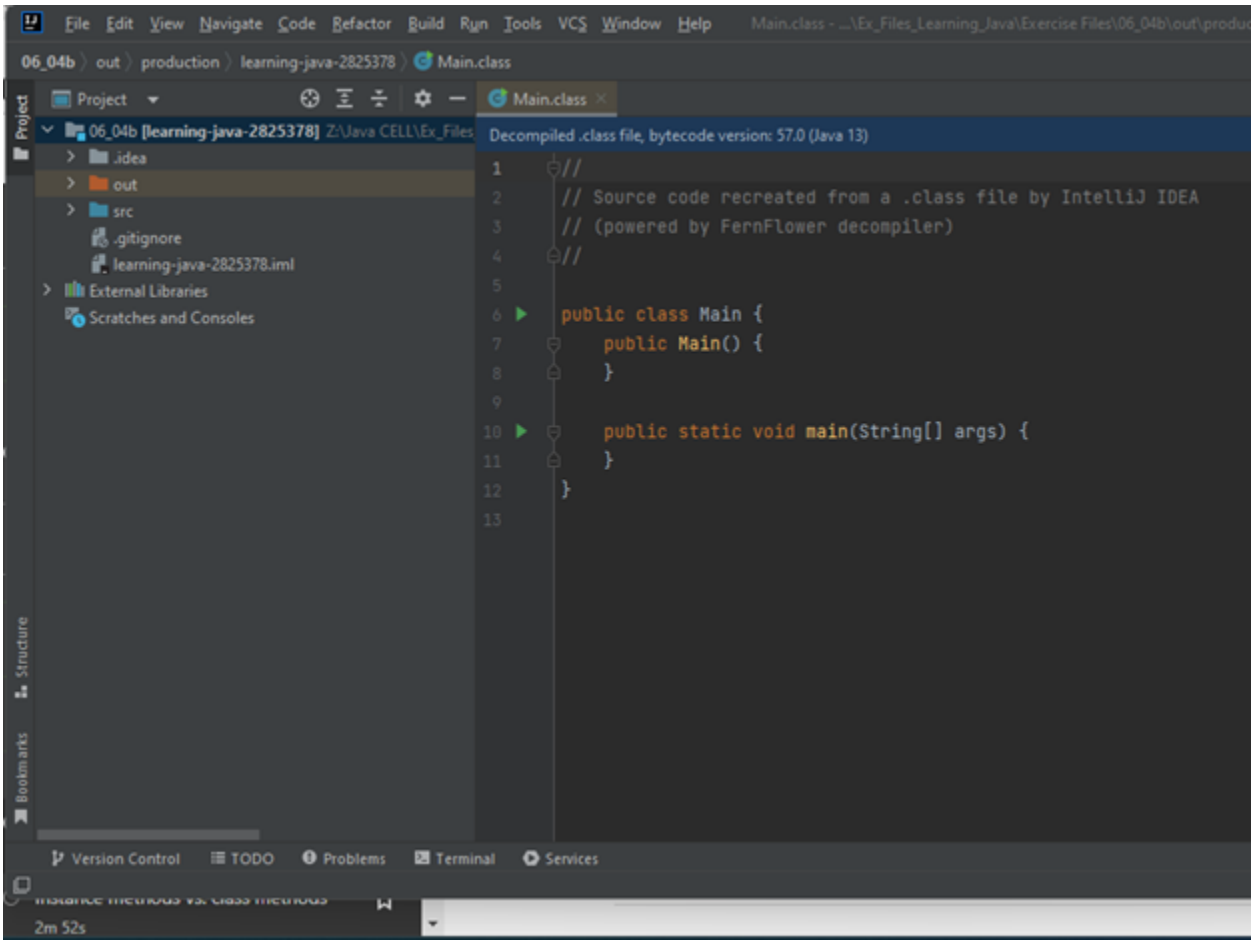




Next, we'll try creating some triangle instances with this blueprint.

Creating instances in Java

Let's make some triangle instances. Continuing on our program from the last lesson, we'll move back to our main class, with the main method, in order to do this. We could add a main method to our triangle class, and run that file instead of this file, but I find the separation of our blueprints and the code we are actually running makes it a bit easier to understand at first.


File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
06_04b out production learning-java-2825378 Main.class
Project
06_04b [learning-java-2825378] Z:\Java CELL\Ex_Files
.idea
out
src
.gitignore
learning-java-2825378.iml
External Libraries
Scratches and Consoles
Main.class
Decompiled .class file, bytecode version: 57.0 (Java 13)
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//
public class Main {
public Main() {
}
public static void main(String[] args) {
}
}
Version Control
TODO
Problems
Terminal
Services
2m 52s

```java
public class Triangle {
    2 usages
    double base;
    2 usages
    double height;
    1 usage
    double sideLenOne;
    1 usage
    double sideLenTwo;
    1 usage
    double sideLenThree;

    public Triangle(double base, double height,
                    double sideLenOne, double sideLenTwo,
                    double sideLenThree) {
        this.base = base;
        this.height = height;
        this.sideLenOne = sideLenOne;
        this.sideLenTwo = sideLenTwo;
        this.sideLenThree = sideLenThree;
    }

    public double findArea() { return (this.base * this.height) / 2; }
}
```

To create a triangle, we'll create a triangle variable and call the constructor method. We'll name the first triangle "triangle A" and give it the values 15, 8, 15, 8, and 17.

```java
ic class Main {

public static void main(String[] args) {
    Triangle triangleA = new Triangle( base: 15,  height: 8,  sideLenOne: 15,  sideLenTwo: 8,  sideLenThree
```

The second triangle we'll call triangle B and give it the values 3, 2.598, 3, 3, and 30.

```java
ic class Main {

public static void main(String[] args) {
    Triangle triangleA = new Triangle( base: 15,
    Triangle triangleB = new Triangle( base: 3,  height: 2.598,  sideLenOne: 3,  sideLenTwo: 3, sideLenThre
}
```

double base, double height, double sideLenOne, double sideLenTwo, **double sideLenThree**

```
Main {

tatic void main(String[] args) {
ngle triangleA = new Triangle( base: 15,  height: 8,  sideLenOne: 15,  sideLenTwo: 8,  sideLenThree: 17);
ngle triangleB = new Triangle( base: 3,  height: 2.598,  sideLenOne: 3,  sideLenTwo: 3, sideLenThree: 3)
```

Now we have two triangle instances. Let's add some breakpoints and walk through this program so we can see what's going on behind the scenes. We'll add a breakpoint just before triangle A is created, as well as a breakpoint before triangle B is created.

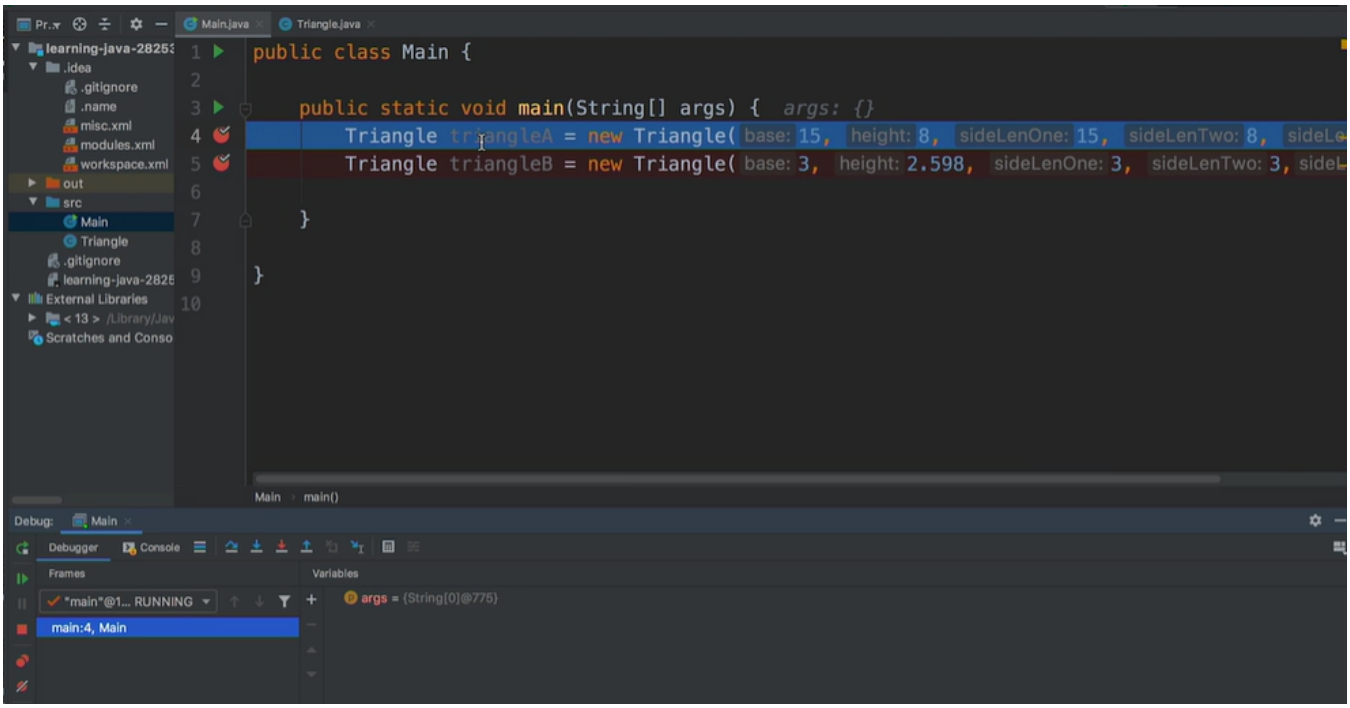
```
1  ic class Main {
2
3      public static void main(String[] args) {
4          Triangle triangleA = new Triangle( base: 15,  height: 8,  sideLenOne: 15,  sideLenTwo: 8,  sideLenThre
5          Triangle triangleB = new Triangle( base: 3,  height: 2.598,  sideLenOne: 3,  sideLenTwo: 3, sideLenTh
6
```

Going back to the triangle class, we'll also add a breakpoint on this first line of the constructor.

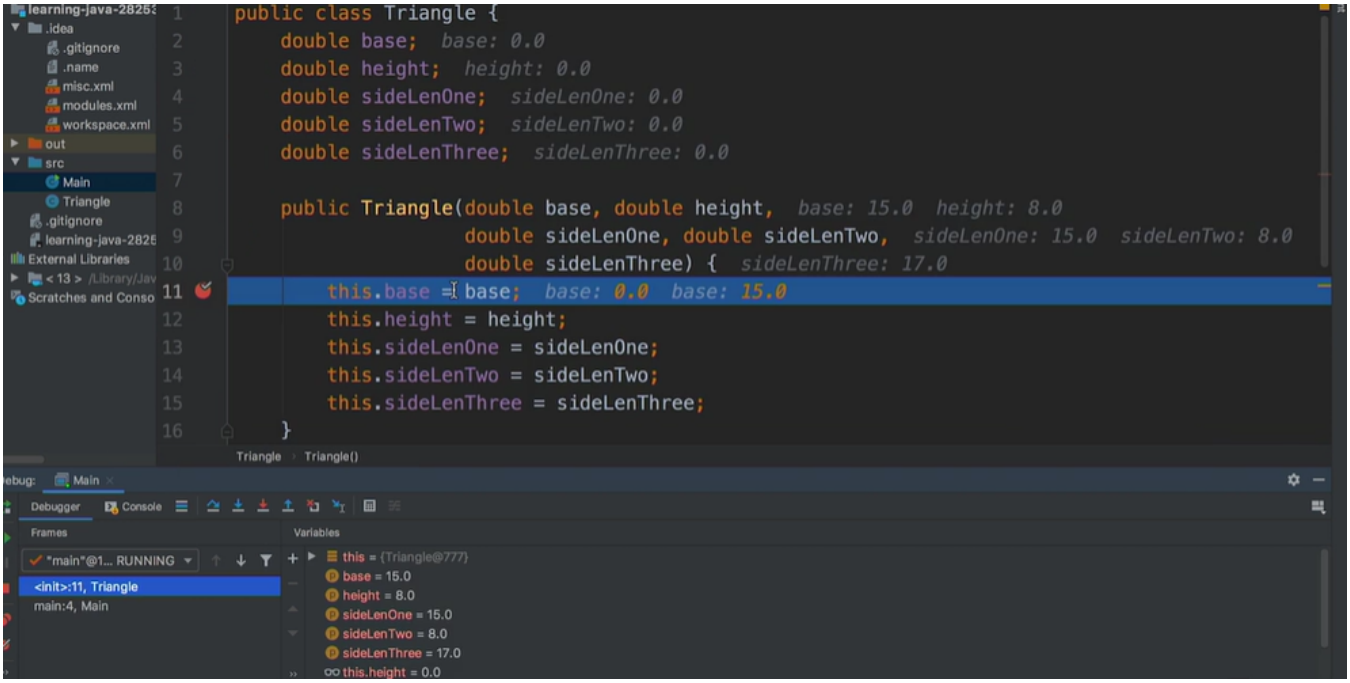
Main.java Triangle.java

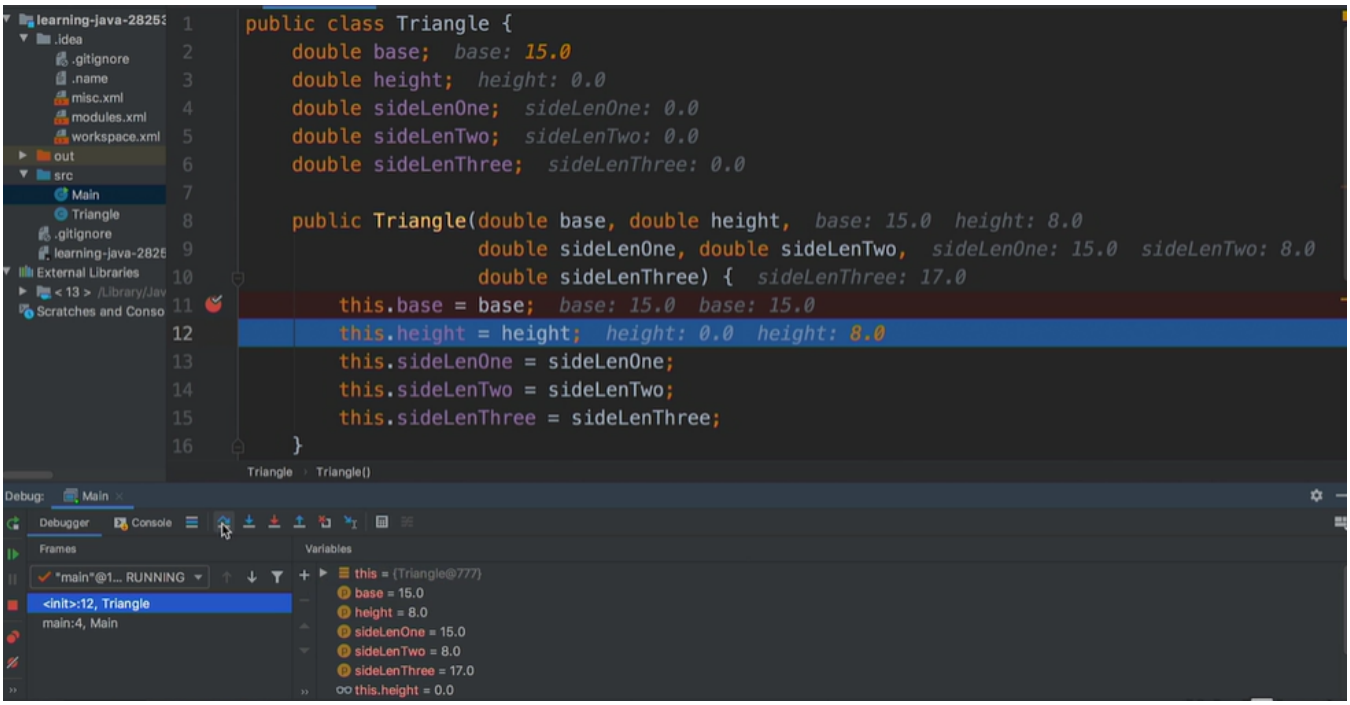
```
1   public class Triangle {
2       double base;
3       double height;
4       double sideLenOne;
5       double sideLenTwo;
6       double sideLenThree;
7
8       public Triangle(double base, double height,
9                       double sideLenOne, double sideLenTwo,
10                      double sideLenThree) {
11          this.base = base;
12          this.height = height;
13          this.sideLenOne = sideLenOne;
14          this.sideLenTwo = sideLenTwo;
15          this.sideLenThree = sideLenThree;
16      }
```

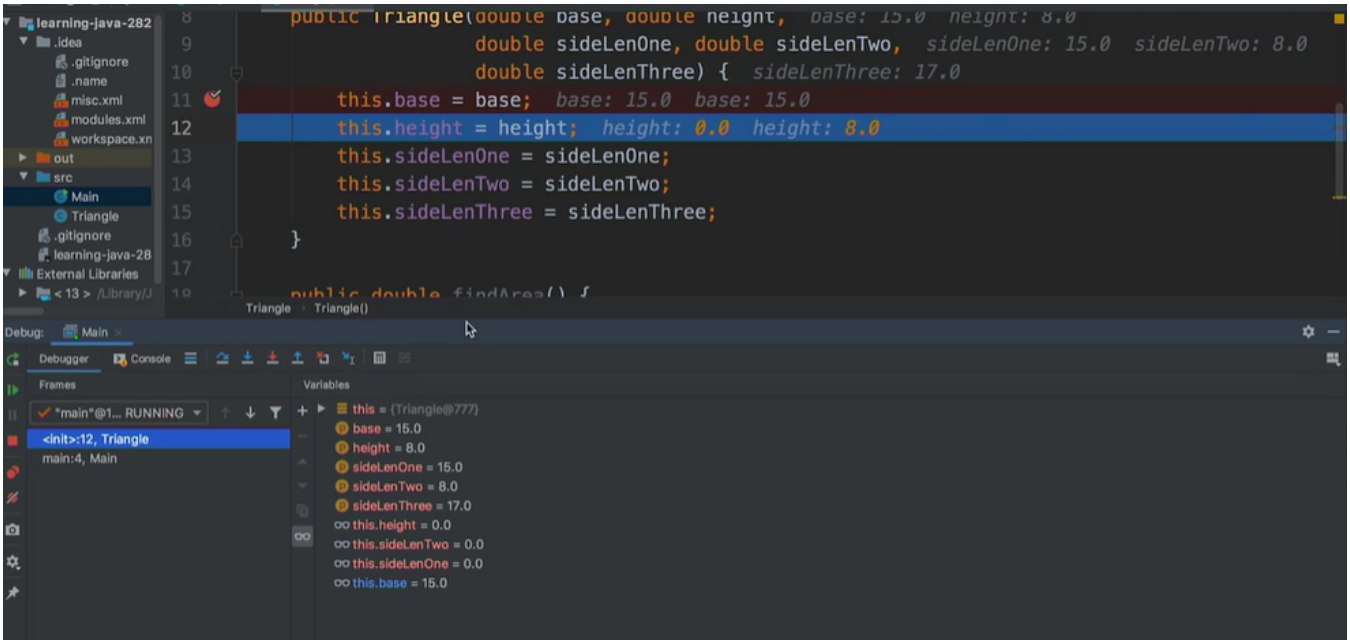
We'll be running our main class in debug mode. Our first stop is right before we create triangle A. We are passing 15 for the base, 8 for the height, 15, 8 and 7 for each of the side lines.



Let's hit Continue. Now we are in the constructor itself, where we are creating the triangle instance that will be stored in the triangle A variable. Our parameters have the values we passed, 15 for the base, 8 for the height, and so on. In the following lines of code, in the constructor, we will assign these values to the attributes of the triangle to be created. All of our attributes for the new triangle have defaulted to the value zero, as you can see here at the top.

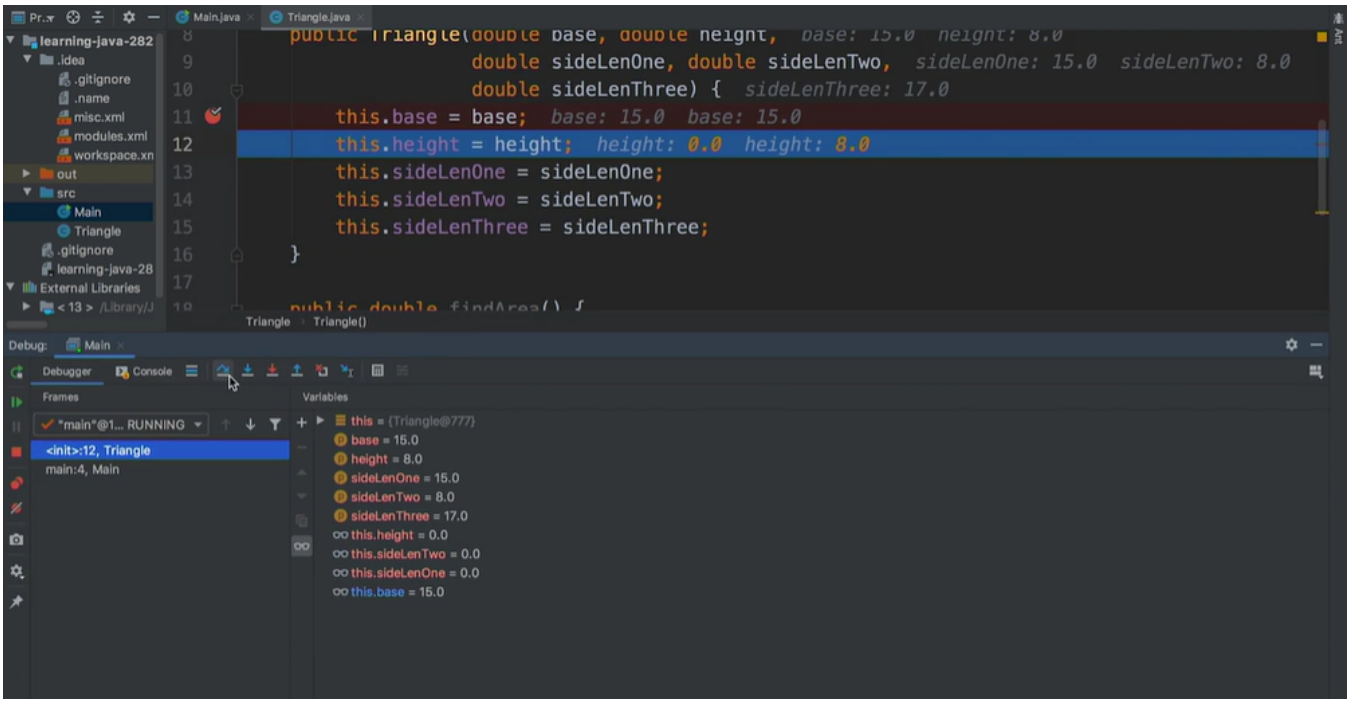


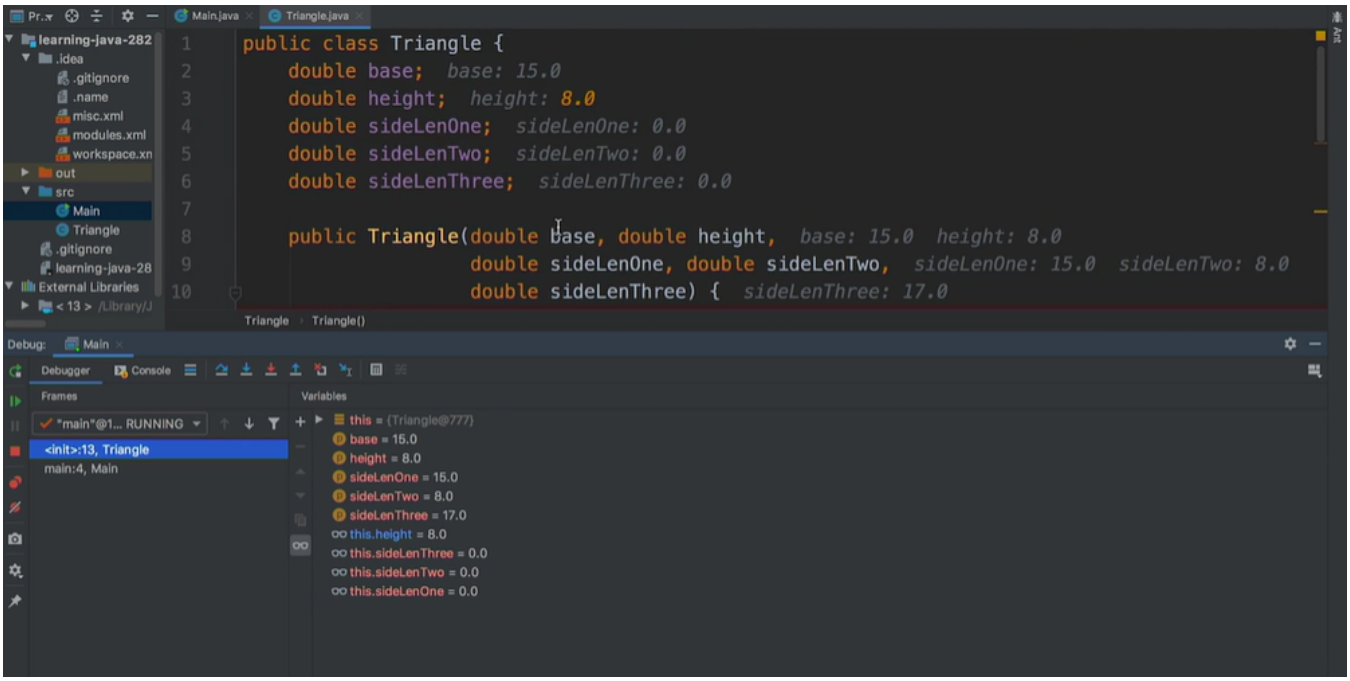




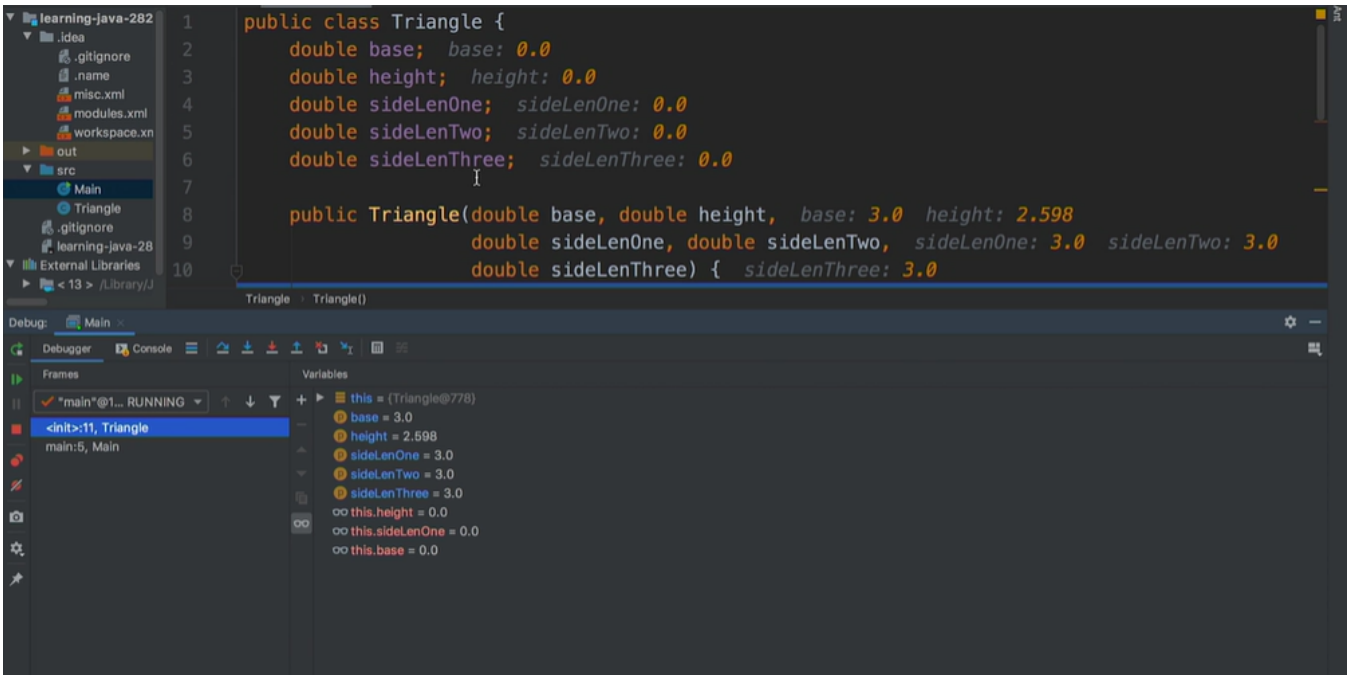
This is the magic of an IDE. With the constructor, we will initialize each of these attributes. Let's hit the Step Over button, and now the base attribute for the triangle to be stored in the triangle A variable has the value 15. We can see the new base attribute value highlighted at the top. In the variables window at the bottom, if we open up the pane, we can see that the base is initialized here as well.

The "this" keyword stands for the triangle we are constructing and initializing, so as we initialize it, the values within the triangle object we are creating should be initialized. Let's hit Step Over again. The height attribute for the new triangle A is now 8 and seven zero, and this will happen for each attribute we are initializing. We hit Step Over again.





The initialization for side length One is executed, and now the side length One attribute has the value 15. Hitting Step Over again, side length Two is initialized. Hitting Step over one last time, the value of side length three finally gets initialized and the triangle has been constructed. All that's left to do is to return from the constructor. We can do this by hitting Continue. Now we are on to creating our next triangle, triangle B. Let's hit Continue, and we hit that breakpoint inside the constructor again. Now we're constructing triangle B. This to-be-created triangle instance, like the other, has its default attribute values set to zero. They are not connected to the attribute values of triangle A, because different triangles should have different values for their base, height and side lines. Each instance created from a constructor is independent of each other.



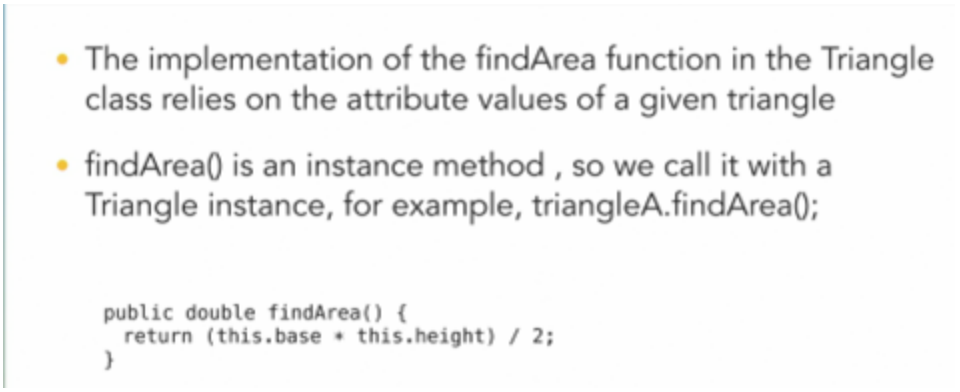
The attribute values of one triangle do not affect the other.

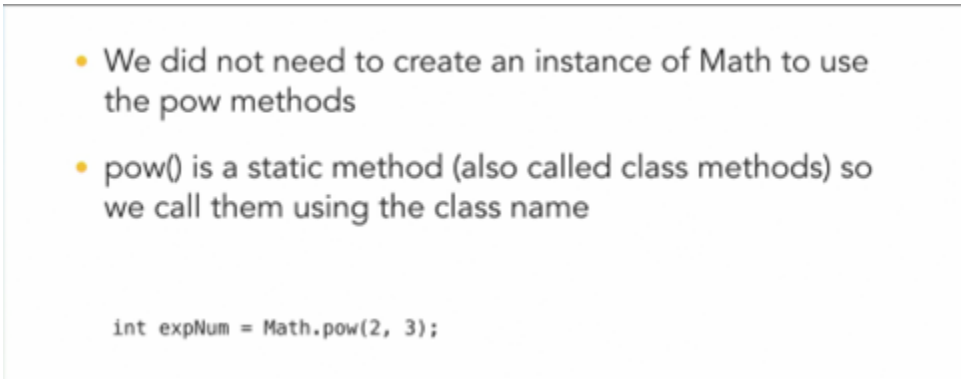
Let's hit Step Over a few times, and as we continue the rapid constructor, each attribute is getting initialized, this time for triangle B. Let's hit Continue, and now we have created two triangle instances, using a constructor from the triangle class. Our program not only knows how to represent triangles, but it has two triangles represented within it. So, what can we do with these new triangles? We added the behavior of find area to the triangle class in the last lesson, so we'll try using it on one of our triangles.

Instance methods vs. Class methods

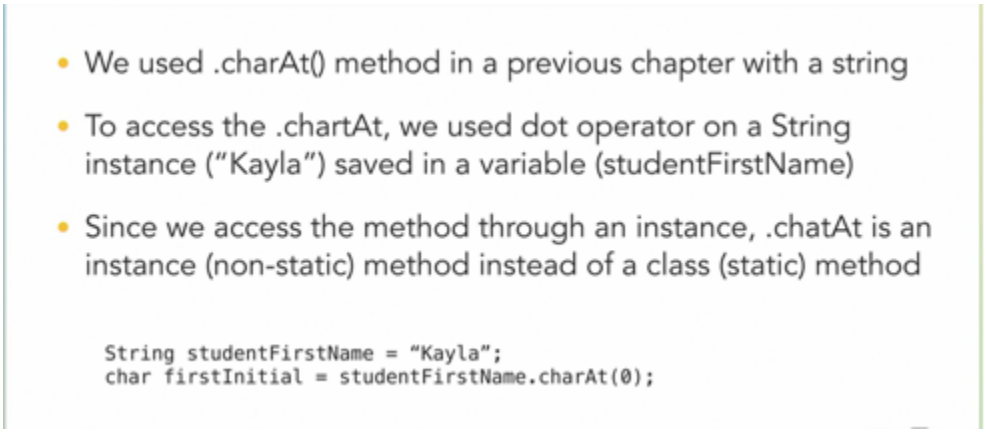
Let's try calculating the area of each triangle. We'll store the area of triangle A in a variable called triangleAArea, and it will be a double, because that's what the findArea function returns. To get access to the findArea function, we use the dot operator on the triangle A instance. So, we'll write triangleA. findArea.

Now you might be thinking, why didn't we write triangle.findArea? Isn't that what we did when we used pow on the math class with math.pow? And yes, this is where a lot of people get confused. The reason we did triangleA.findArea instead of triangle.findArea is because in order to find the area of a given triangle, you have to have a triangle instance. You can't calculate the area of a triangle that doesn't exist yet. The implementation of the findArea function relies on the attributes of a given triangle. The base might be 8 or 10 or 15. We don't know until the triangle is actually created. Because we have to have a triangle instance already created to use the findArea method, we can say findArea is an instance method. For math.pow, we did not need to create an instance of Math in order to use the pow function. We just accessed the class and used the function we wanted. That's because pow is not an instance method. It is a static method or class method because you do not need an instance of the class in order to use the function.



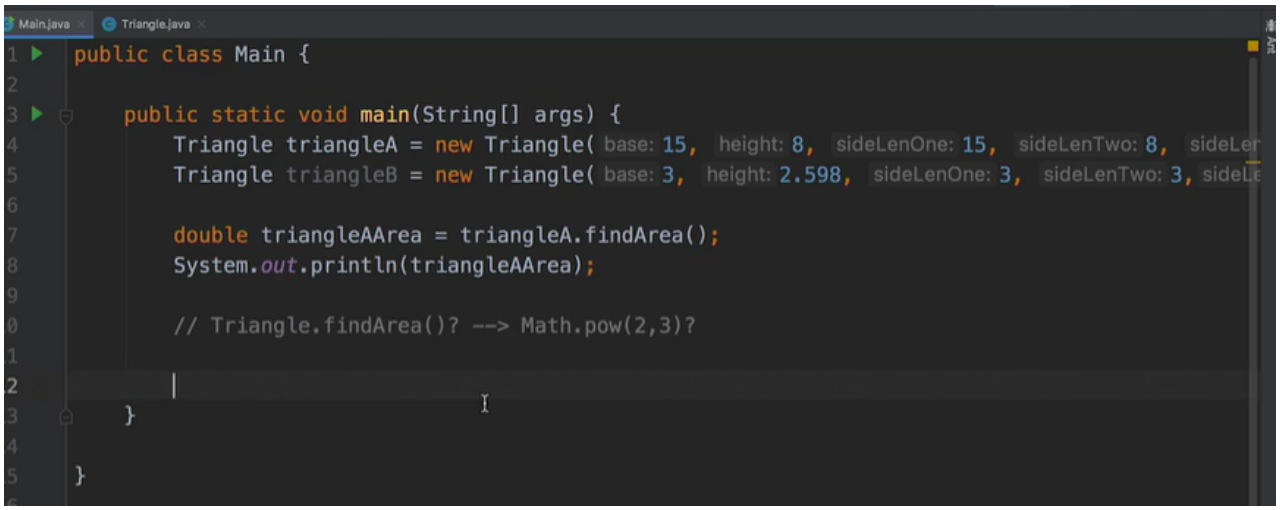


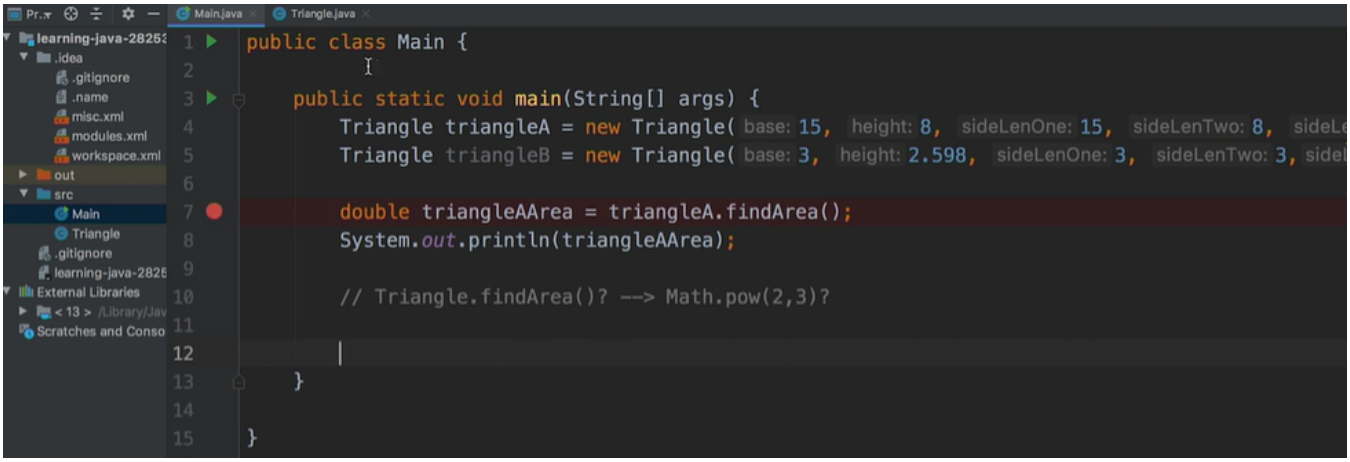
Because of this, instance methods are often referred to as non-static methods, since you have to create an instance in order to use them. You do not need an instance created to use a static method. You can just reference the class. In a previous iteration, we used .charAt() with a string, and we did need a string already created in order to use it. In our code, we wrote studentFirstName.charAt(0) and studentLastName.charAt(0), where the studentFirstName and the studentLastName both evaluated to string values. In both of these cases, we used the dot operator on an instance of a string in order to get access to the .charAt() method. Because we used the dot operator on an instance and not the class name, .charAt() is an instance method, and not a class method. Understanding the difference between instance methods and class methods is very important as you continue to learn Java.

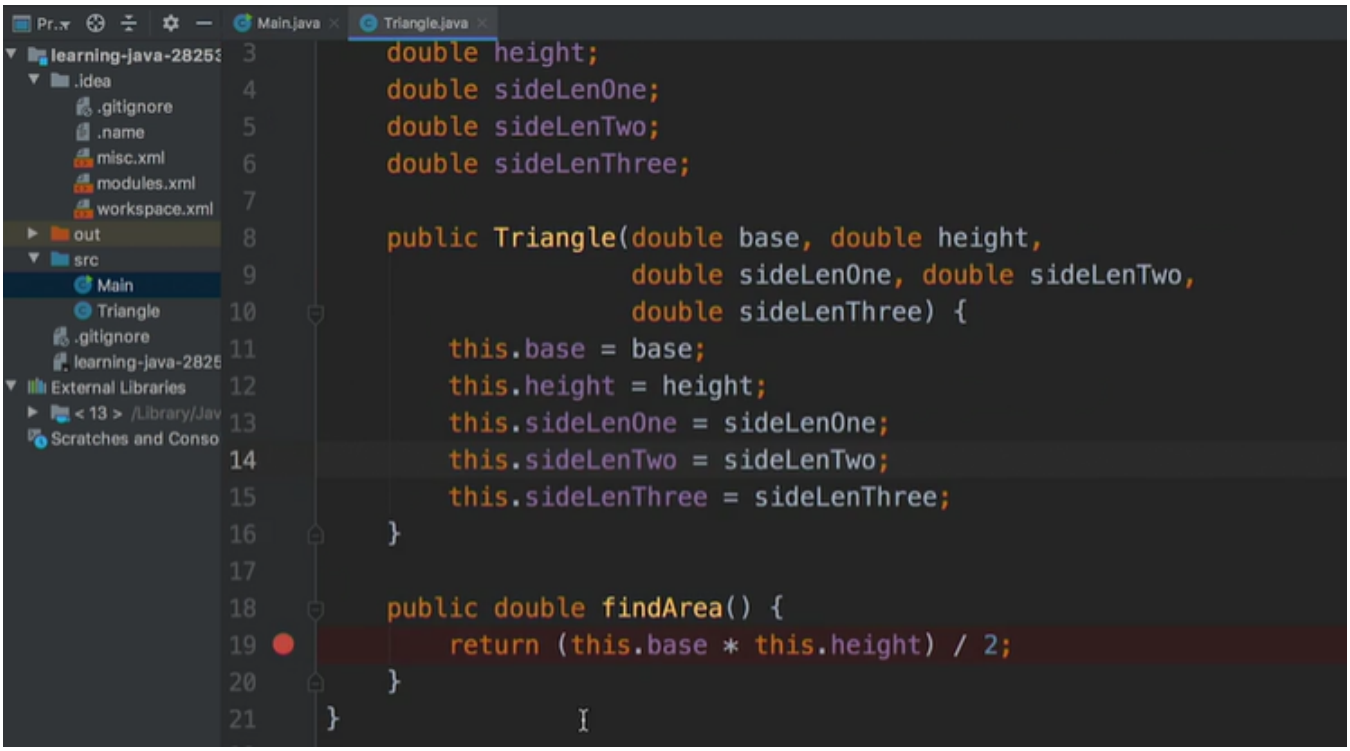


Using instance methods in Java

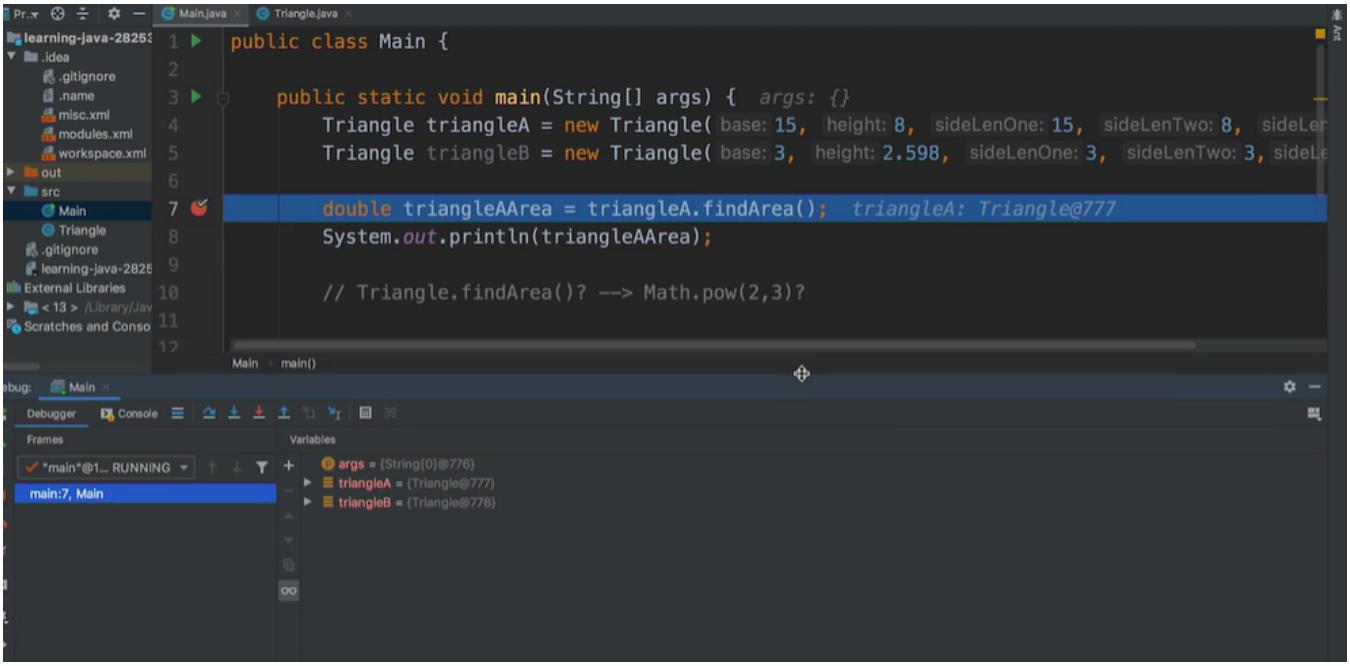
Let's get back to our triangle program. We've created two triangles, and we are using the dot operator on triangleA to access the findArea function, so that we can find the area of triangleA. Let's add a break point when we make the findArea function call, and then on the first line of the findArea implementation.

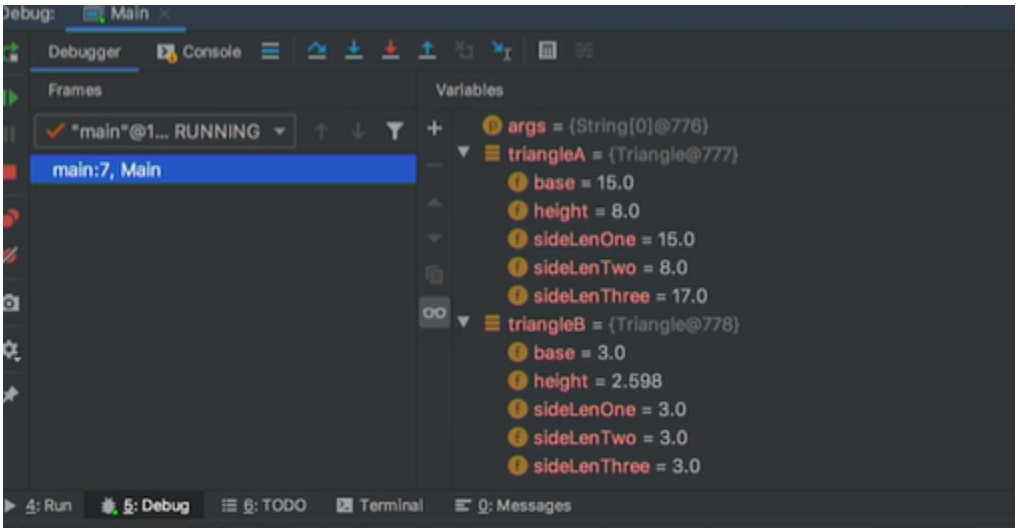

Main.java
Triangle.java
public class Main {
public static void main(String[] args) {
Triangle triangleA = new Triangle(base: 15, height: 8, sideLenOne: 15, sideLenTwo: 8,
Triangle triangleB = new Triangle(base: 3, height: 2.598, sideLenOne: 3, sideLenTwo: 3,
double triangleAArea = triangleA.findArea();
System.out.println(triangleAArea);
// Triangle.findArea()? --> Math.pow(2,3)?
}
}


Main.java
Triangle.java
learning-java-28253
.idea
.gitignore
.name
misc.xml
modules.xml
workspace.xml
out
src
Main
Triangle
.gitignore
learning-java-2825
External Libraries
Scratches and Conso
public class Main {
public static void main(String[] args) {
Triangle triangleA = new Triangle(base: 15, height: 8, sideLenOne: 15, sideLenTwo: 8,
Triangle triangleB = new Triangle(base: 3, height: 2.598, sideLenOne: 3, sideLenTwo: 3,
double triangleAArea = triangleA.findArea();
System.out.println(triangleAArea);
// Triangle.findArea()? --> Math.pow(2,3)?
}
}

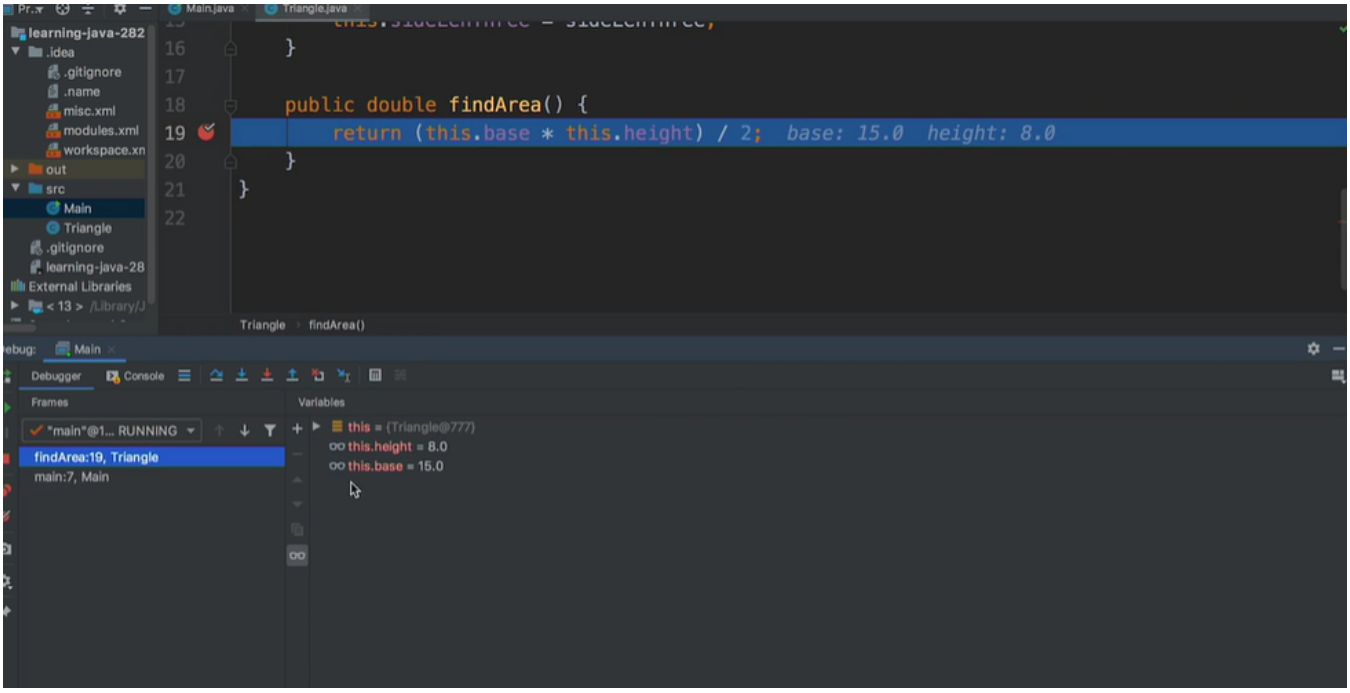


Now, let's run in debug mode. We are about to call the findArea method on triangleA. In the variables window, we can see that our triangleA instance is there, with its specific attribute values, and the triangleB instance is there, with its specific attribute values.

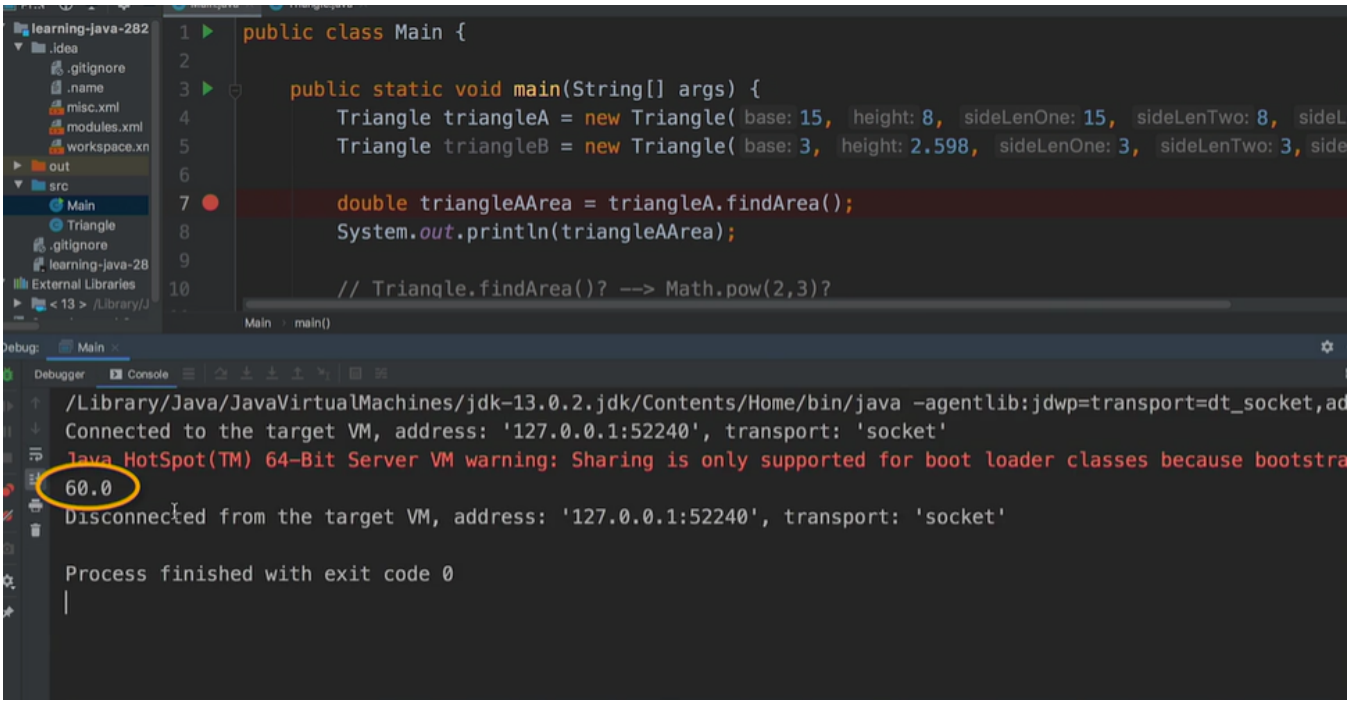




We'll hit continue, and now we find ourselves in the findArea implementation. For triangleA, the base attribute has the value 8, and the height attribute has the value 15. We used the "this" keyword to access triangleA's specific base and height values, and used the base and height divided by two formula to calculate triangleA's specific area.

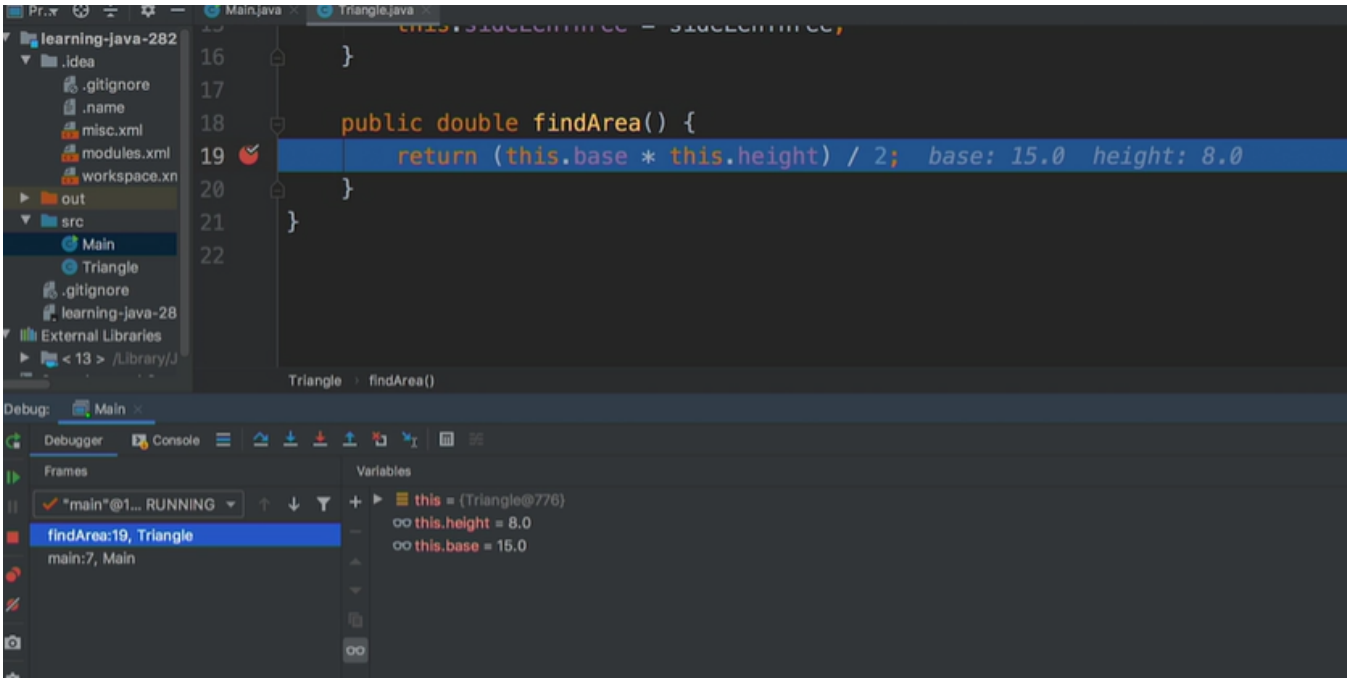


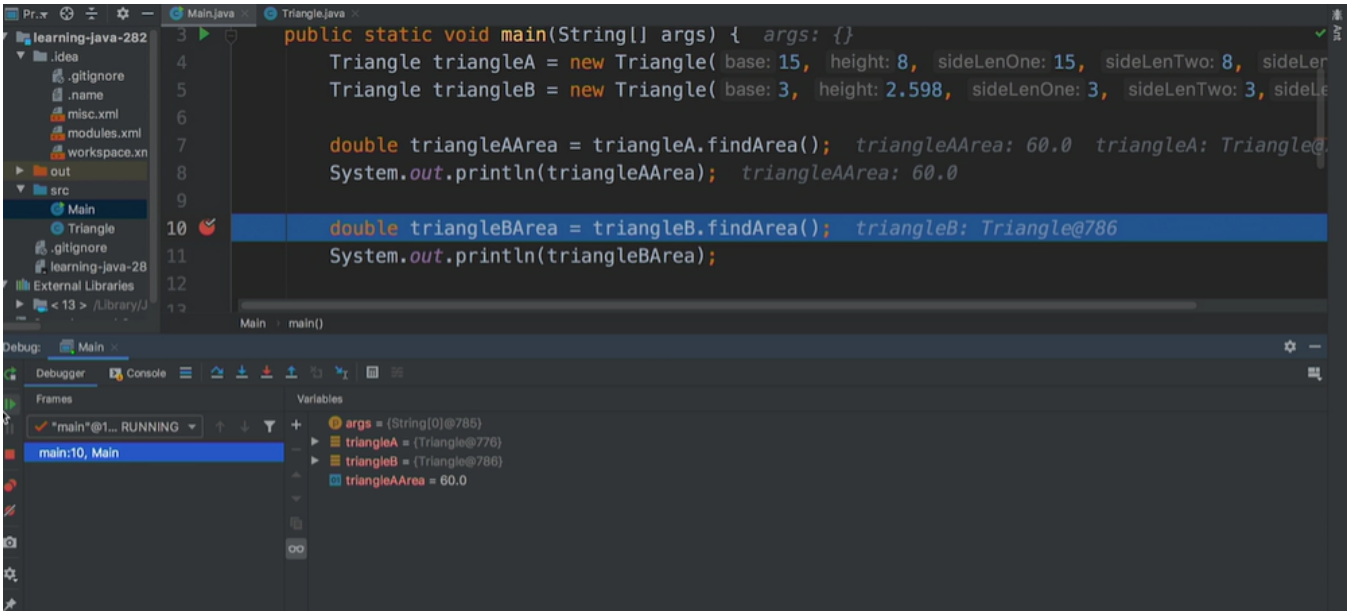
Hitting continue, and switching over to the console, we can see that triangleA's area is 60.



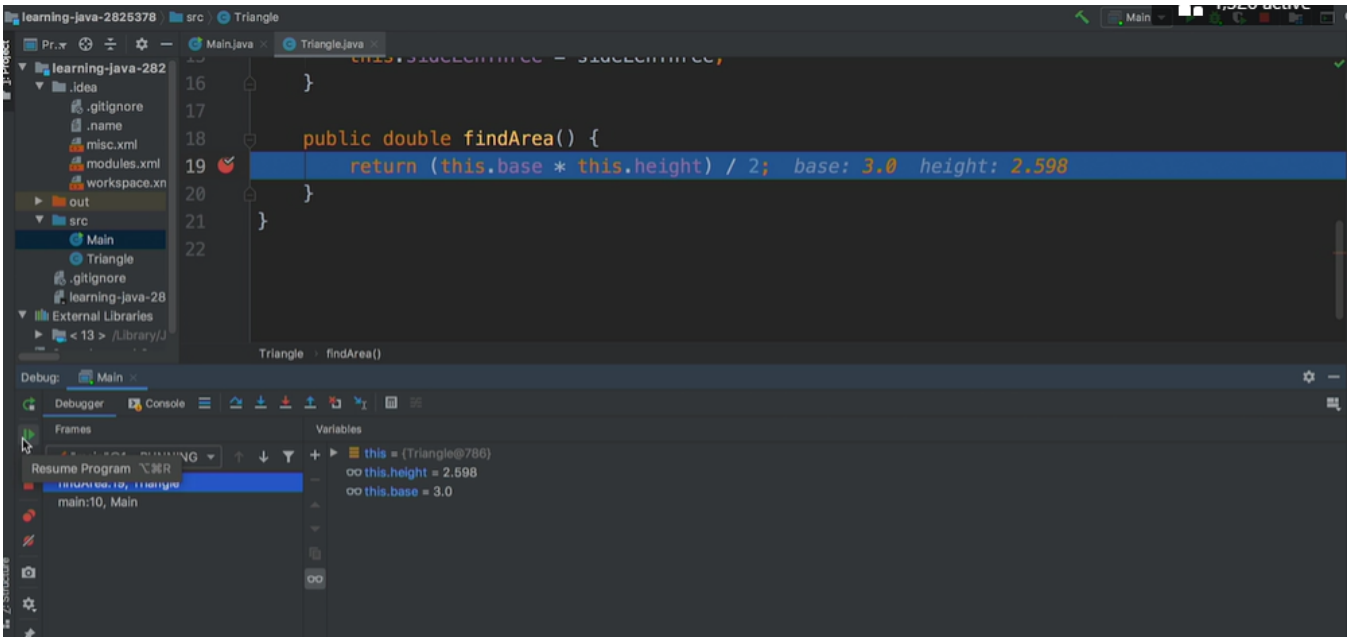
We can calculate triangleB's area as well. Of course, since triangleB has different attribute values for the base and height, we will get a different output for the area. So, let's add a double variable called triangleBArea, and assign it to the value, triangleB.findArea returns. Then we'll print out triangleBArea, with System.out.println, triangleBArea.

We'll remove the break point from the triangleA call, and move it to the triangleB call, and run this in debug mode. The first time we encounter findArea is with the call from triangleA.

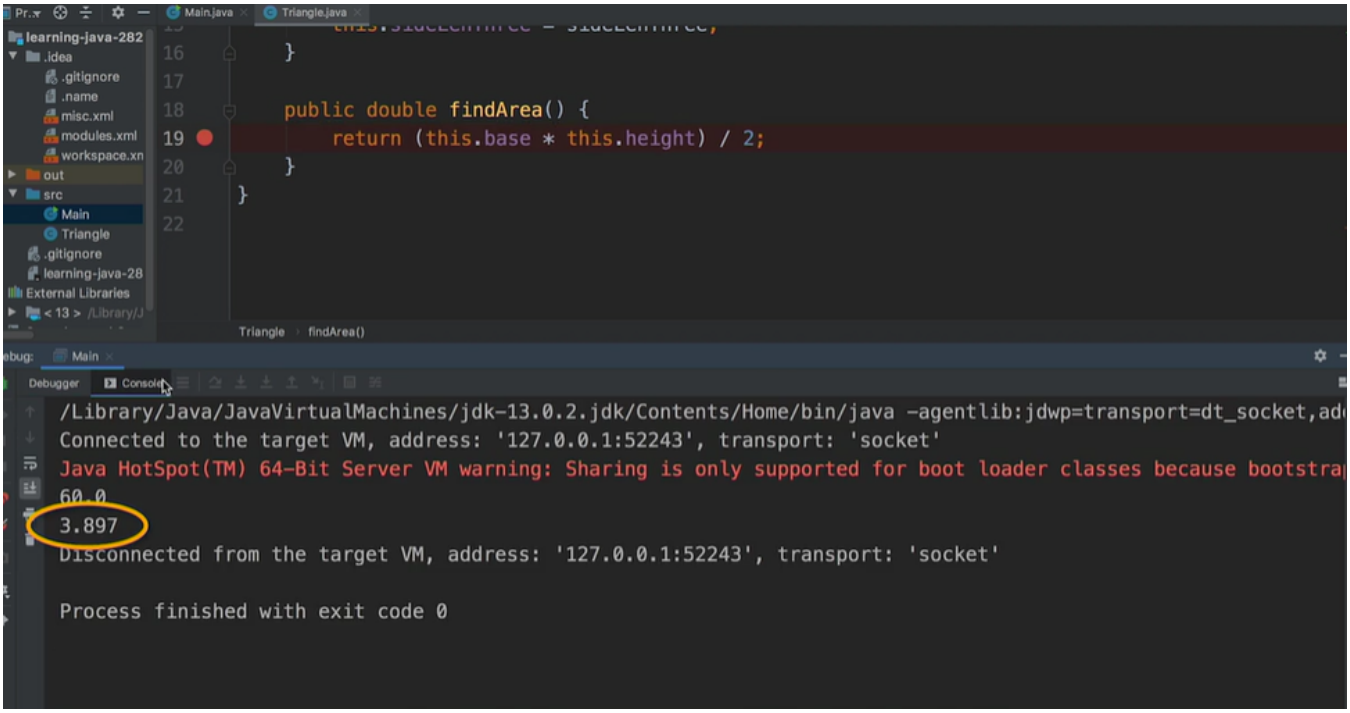




So that's why we see a base of 8 and a height of 15. If we hit continue, we'll stop right before triangleB calls findArea. Hitting continue again, we are back in the findArea function, and now we have triangleB's attribute values for the base and height.



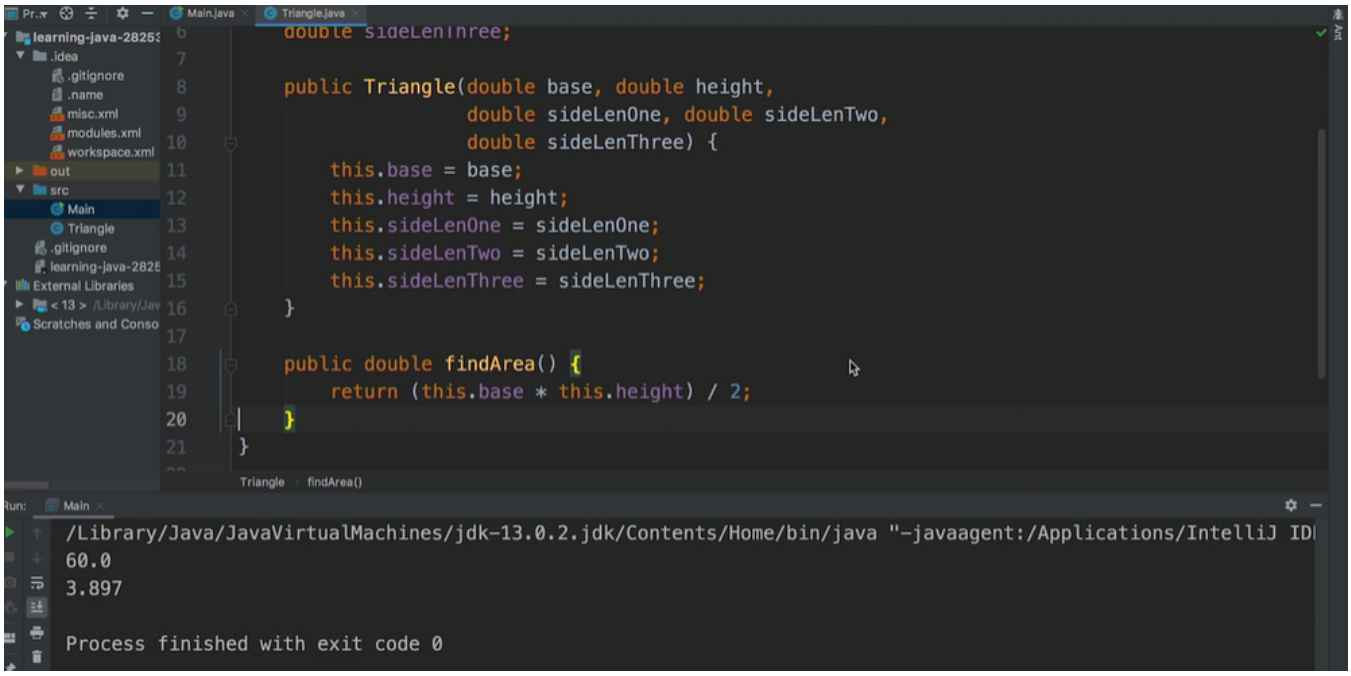
These are used to calculate the final area, and hitting continue, we'll switch back to the console, and the final area is 3.897.

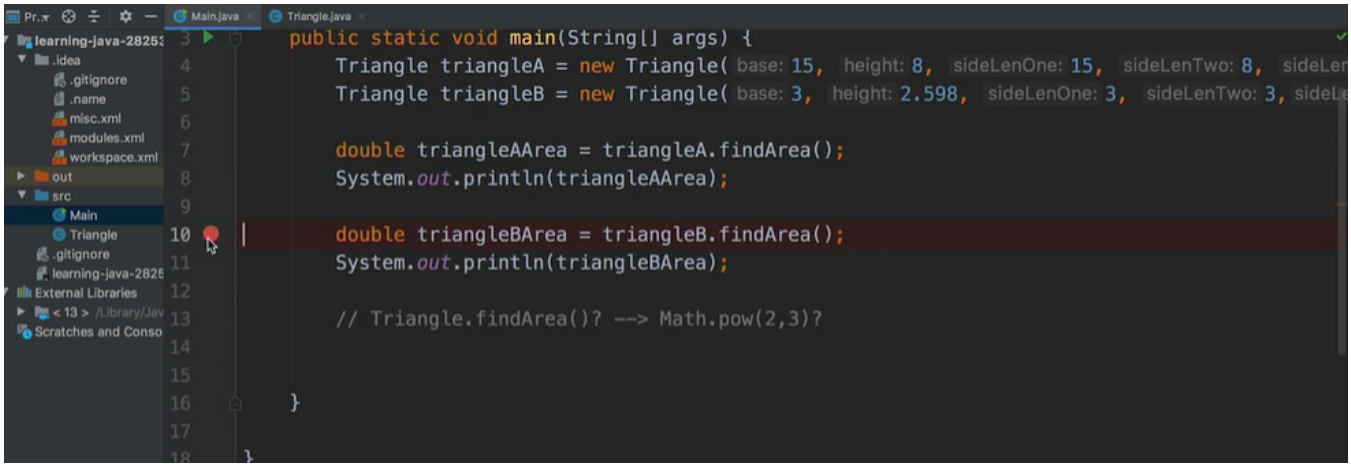


We just used our custom-built findArea instance method from the triangle class on two triangle instances.

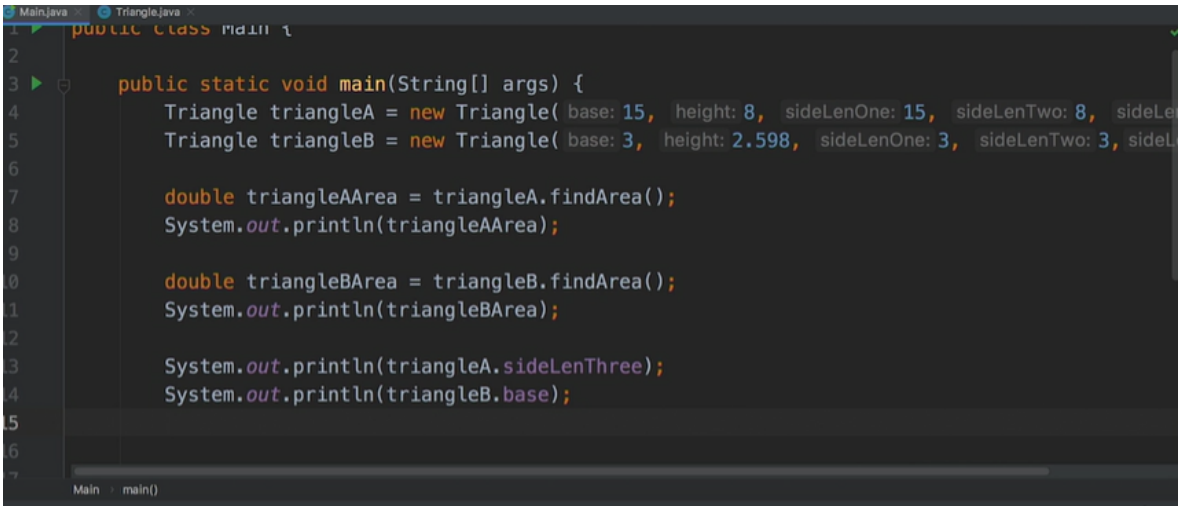
Instance and class variables in Java

With the findArea call, we've used the dot-operator to access a behavior or method from our Triangle class. What about attributes? So far, we have only accessed a triangle's instances values using the "this" keyword inside its class. But we can also access it outside its class. In our main method we could print out triangle A's, Sidelenthree with system.out.printLn, trianglea, sideLenThree. We use the .operator on trianglea, to access its instance variable.

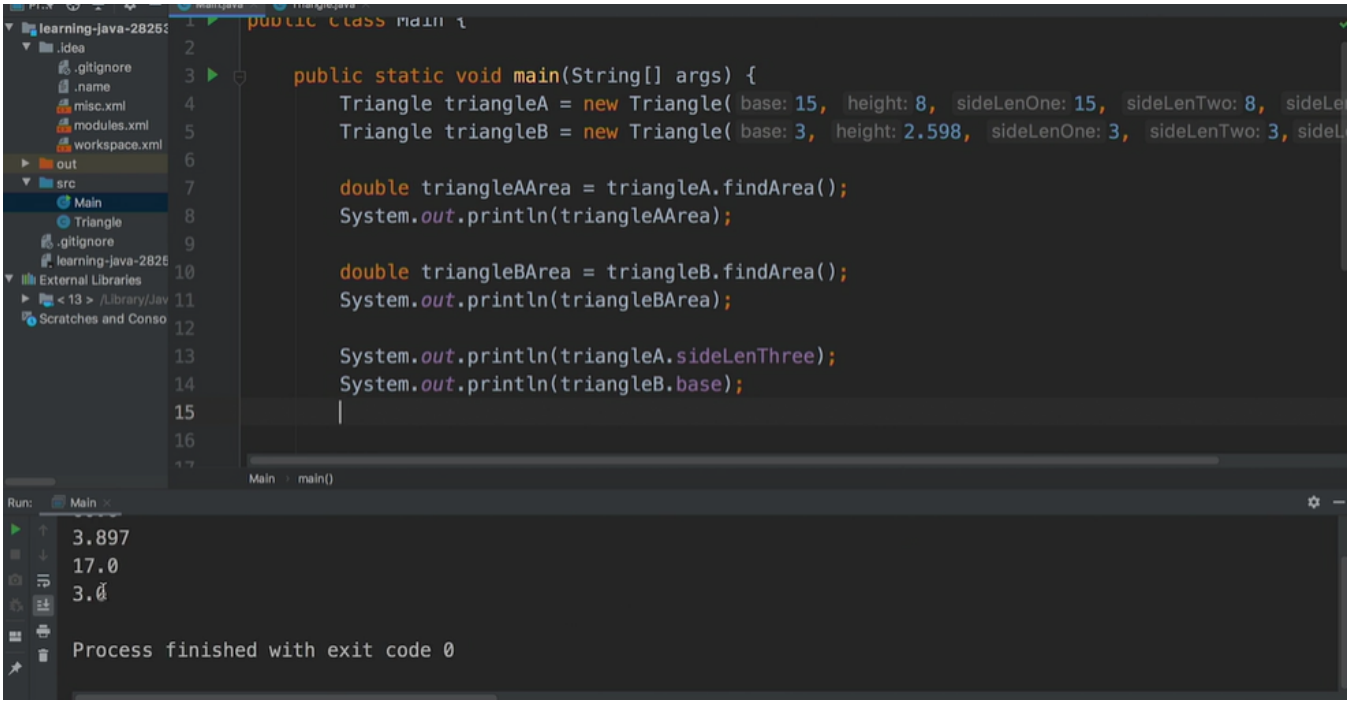




Or attribute variable, sidelenthree. We could also print out triangleb's base with system.out.println triangleb.base The rule here is that it is a function if it has parentheses and an attribute if it does not have parentheses. Sidelenthree and base are both attributes here.



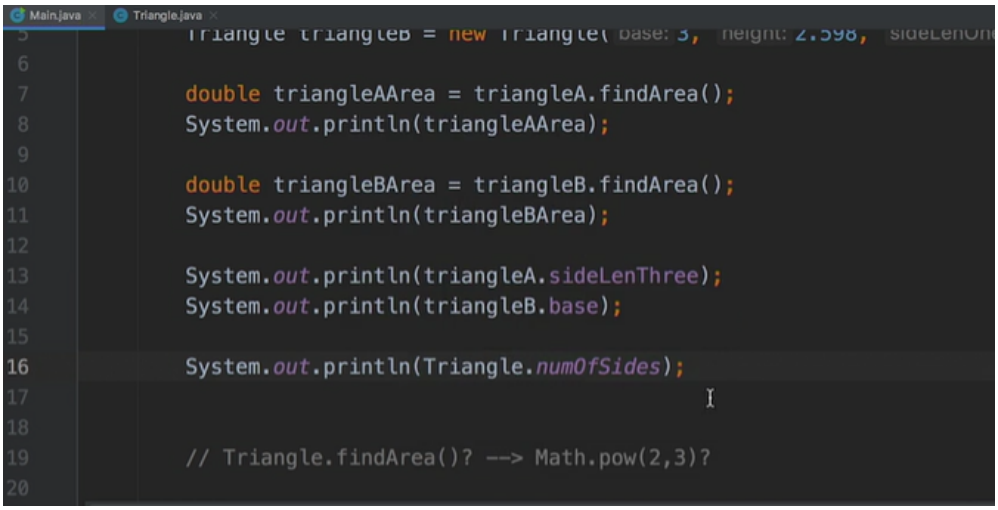
So, we do not add any parentheses. We are accessing a variable, not calling a function. Let's save this and run the program and see what we get in the output. The last two values 17 is the length of side three. And, three is the base of triangleB. Remember, both sidelenthree and base are instance variables. Both of these variables hold information about a specific triangle instance.



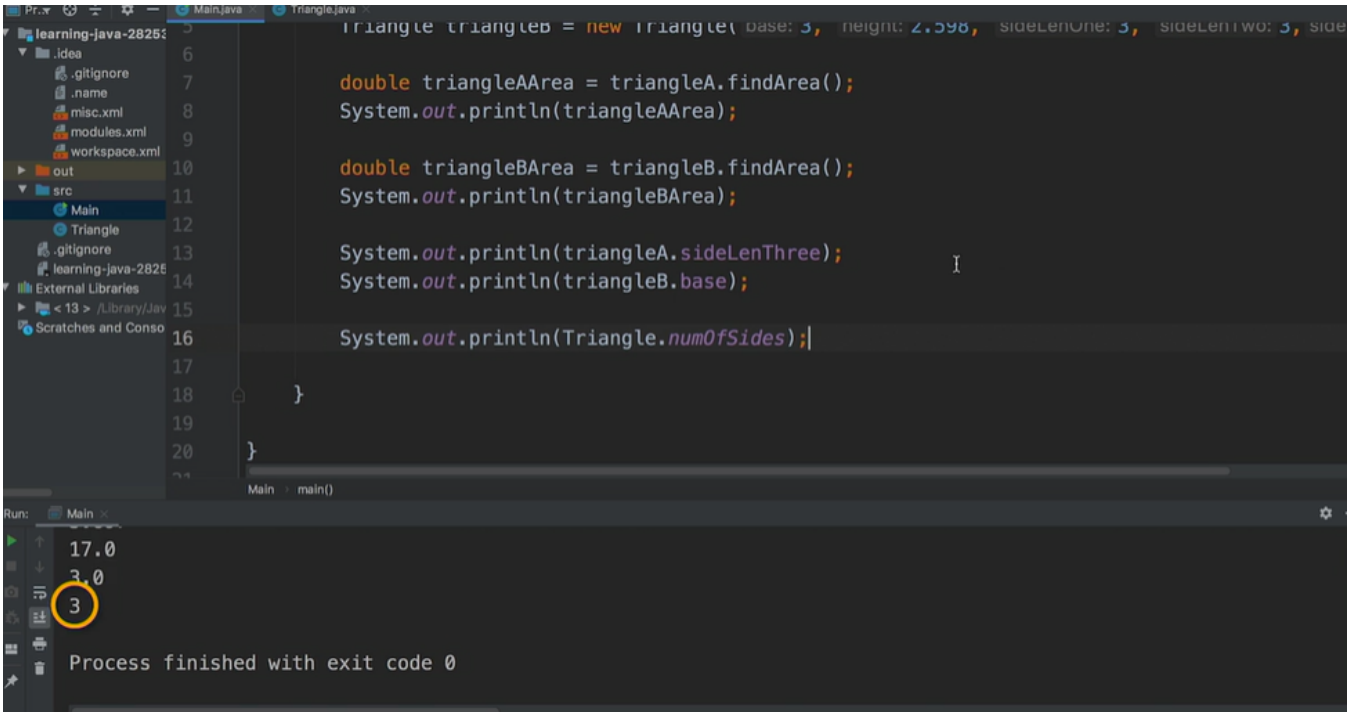
Just like we had static and non-static methods, we have static and non-static variables. Non-static variables are the instance variables. The base of the triangle, the height of the triangle, they do not stay static or the same between triangle instances. The values of base, height, and the rest of our instance variables change depending on the triangle. A static variable is something that will not change per instance. They hold the value for the whole class to use. For example, we might have a static variable that stores how many sides there are to a triangle. This is something that will not change per instance. But rather if you change it, it should effect every instance. Similar to how we accessed static behavior, or static methods we access a static variable using the class name. Since it does not change from instance to instance and belongs to the class rather than a single instance, a static variable is accessed with a . operator, using the class name. Let's try adding a static variable to our triangle class. Inside the triangle class, we'll add a static variable above our instance variables. This static variable will be an int, and we will call it numofsides. We will give it the value three.

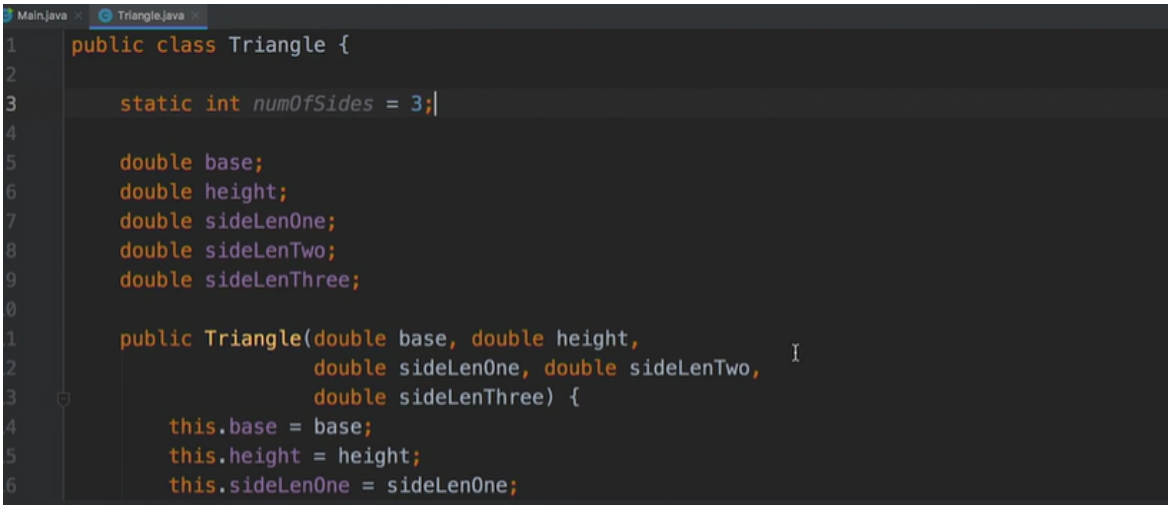
```java
public class Triangle {

    static int numOfSides = 3;

    double base;
    double height;
    double sideLenOne;
    double sideLenTwo;
    double sideLenThree;

    public Triangle(double base, double height,
                    double sideLenOne, double sideLenTwo,
                    double sideLenThree) {
        this.base = base;
        this.height = height;
        this.sideLenOne = sideLenOne;
```

Going back to our main class, we can access the static numofsides variable with the class name triangle, and the .operator. We will want to print it out to the console. So, we will write system.out.println, and then use the class name triangle to access the static variable, numofsides.



Let's run it. And we see three in our console. We have three sides to a triangle.

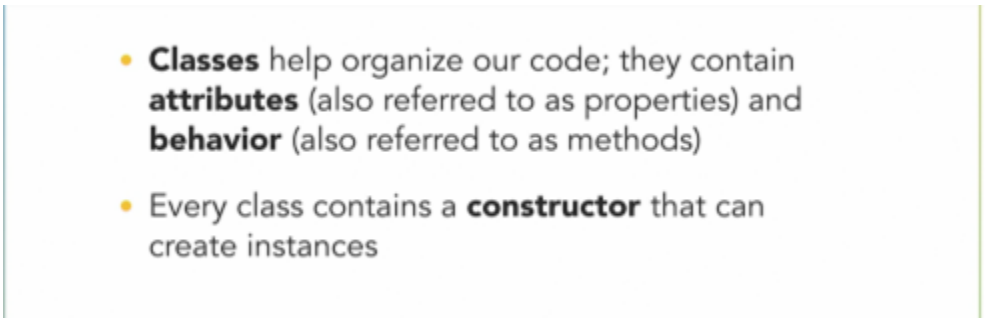


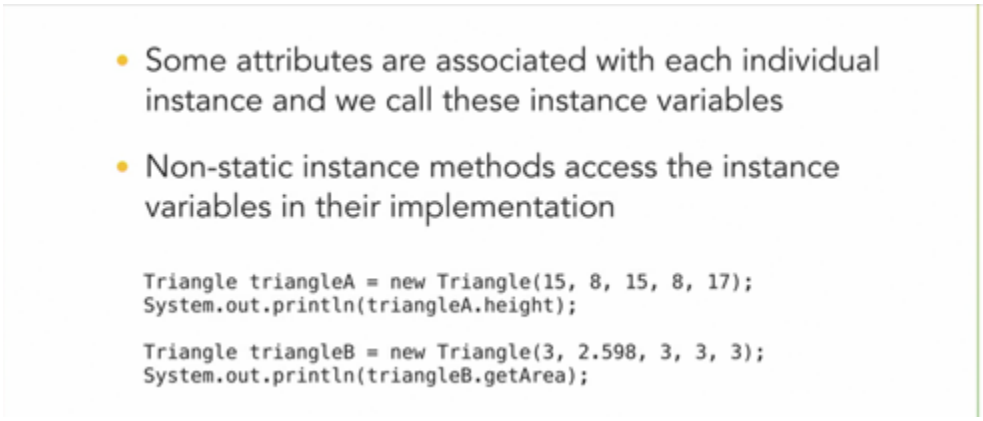


Now, this representation of triangle isn't perfect. Right now, we assume the values given to the constructor are valid. None of them are zero or negative. And they together meet the requirements for a triangle. This is something that you can add on later if you want to. But we are keeping this as simple as possible for now. We have introduced a lot of new concepts, so next we will take some time to review what we have learned.

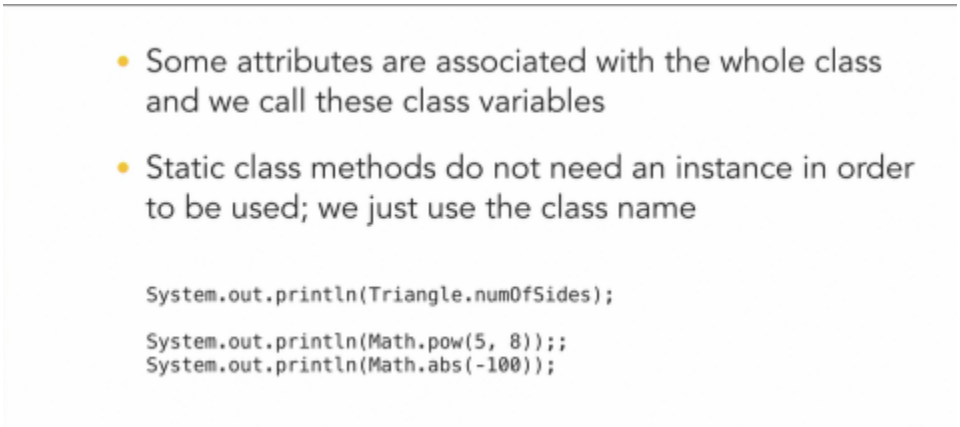
Review: Classes vs. instances

So, what are the big takeaways from this iteration? First, in Java, we can create classes to organize our code. In our previous example, we used a triangle class to organize data related to a triangle as well as how we can calculate things from triangle data. A class contains attributes and behavior, also called properties and methods. Every class also contains a constructor, which is a specific type of method that allows us to create instances of the class. Ultimately, a class is just a blueprint, and we have to use the constructor in order to create triangles in our code. Some of these attributes are associated with each individual instance, and we call these instance variables. These were the height, base, and side length attributes in our example. These attributes are accessed with the instance itself, and we use the dot operator on the instance to access instance variables. Along with instance variables, each instance has instance functions or non-static methods.





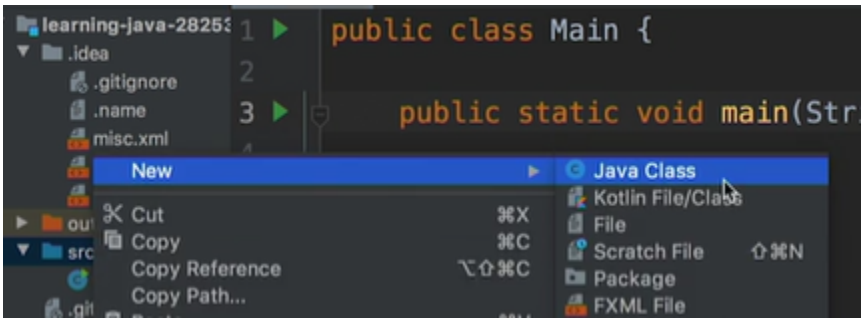
Similarly, these are accessed using the instance with the dot operator and parentheses. We also have class variables and class methods. These variables and methods are static, and we access them using the class name with the dot operator. To use static class variables and methods, we do not need to create an instance, we can just use the class name. This new vocabulary can sometimes be overwhelming, but when you are debugging, Google searching, and trying to figure out how to fix your code, these words often come up.



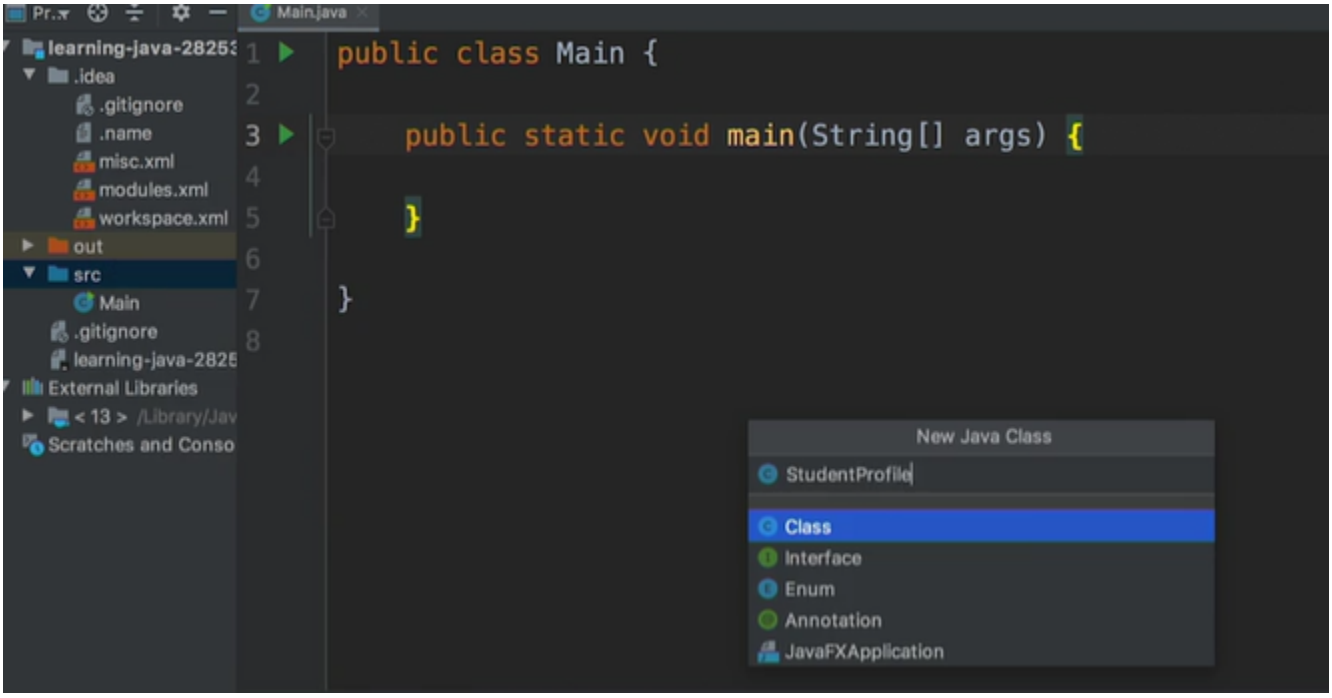
Understanding these concepts is a big hurdle in Java, and once you get it, everything starts to make more sense.

Sample solution: Student profile

Let's represent a student's profile in Java using a class. We'll create a new Java file



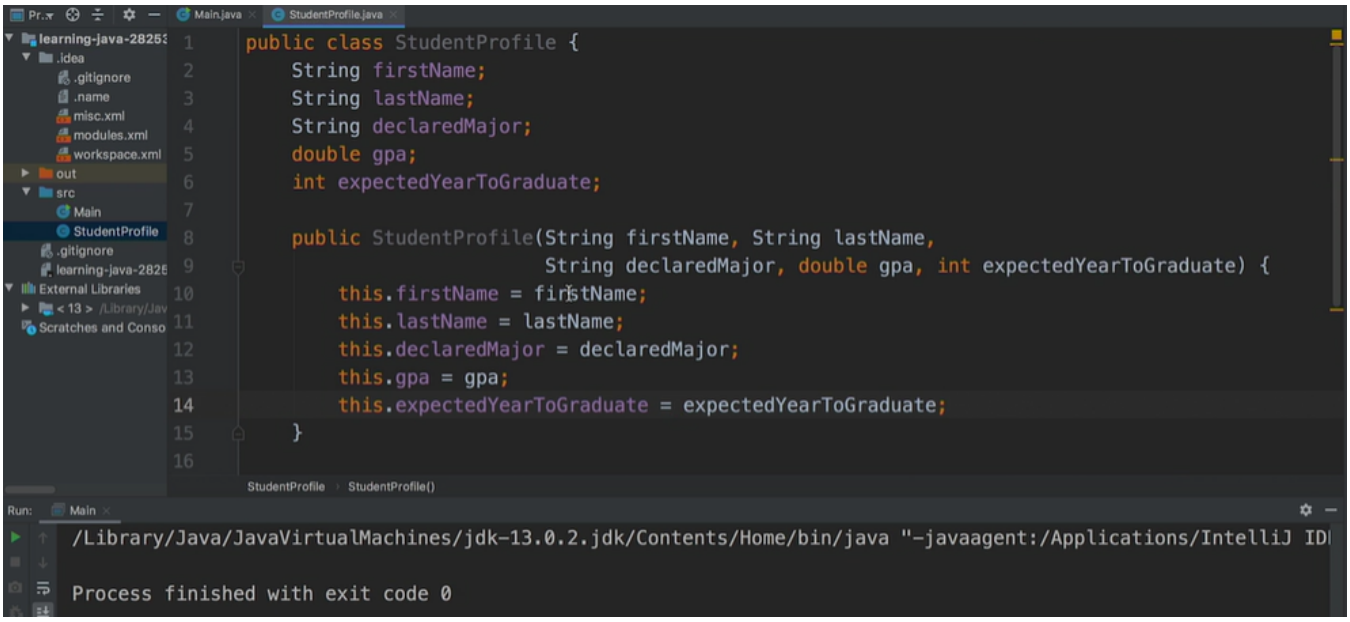
and call it student profile.



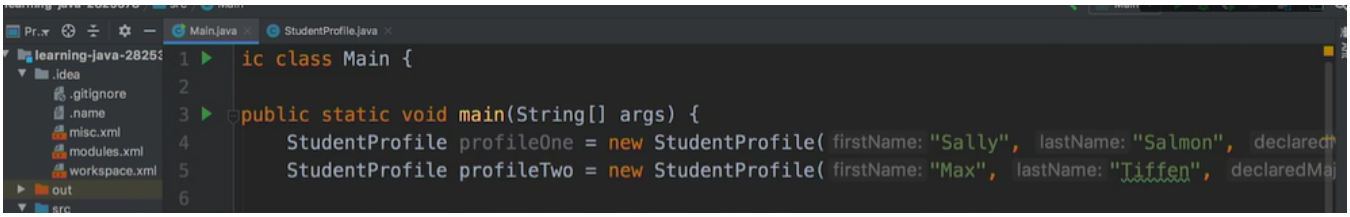
In this class, we'll add our attributes. Each attribute or instance variable will have a data type. So, we'll start with first name, which will be a string. GPA will be a double. And expected year to graduate will be an int. Next, we'll add our behavior or instance methods. The challenge is to add a function that increments the expected year to graduate instance variable. So, we'll write public and then the name of the function, Increment expected (see the sample java files to create the Java student Profiles in the 'Files' section of the Java Basics CELL MS Teams channel).

Graduation year- It will take no parameters. And for the return type, we could have it return the new expected year to graduate value or return nothing, returning void. In this case, we'll say void. In the implementation, we'll be setting the value of the instance variable expected year to graduate. To access this instance variable, we can use the keyword 'this' with the dot operator. For the assignment, we want to access the current value of expected year to graduate. And add one. We can do this by writing this dot expected year to graduate and then plus one. And that's our implementation.

Moving on to the constructor, we'll write public, the name of the class, student profile and in the parenthesis we'll add our parameters. Each of these parameters should map back to an attribute. So, we'll have one for first name, last name with the parameters in order; we can set each attribute to the value that their corresponding parameter is holding. To access the attribute, we use the 'this' keyword. Starting with first name, we'd write this dot first name. Then, we'll assign that the value of the parameter first name. Next, we'll do the same for last name. And that's our constructor.



Our student profile class is finished. Now, we'll go to our main class and create two student profile instances.



With the two profiles created, we'll call the increment expected graduation year function on profile two. Then, we'll print out the value of profile two's expected year to graduate instance variable.

```
ic class Main {

public static void main(String[] args) {
    StudentProfile profileOne = new StudentProfile( firstName: "Sally",  lastName: "Salmon",  declaredM
    StudentProfile profileTwo = new StudentProfile( firstName: "Max",  lastName: "Tiffen",  declaredMaj

    profileTwo.incrementExpectedGraduationYear();
    System.out.println(profileTwo.expectedYearToGraduate);
}
```

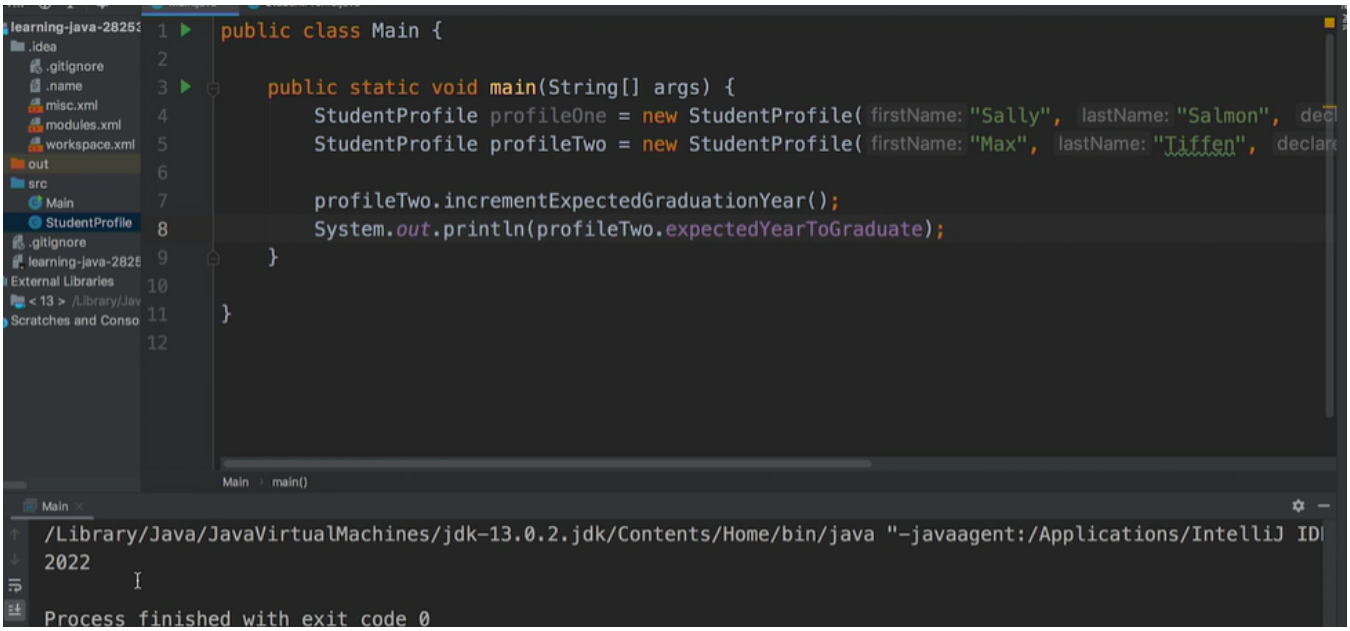
Let's run the program and see what happens. Profile two started out with a graduation year of 2021,

```
lly",  lastName: "Salmon",  declaredMajor: "Film",  gpa: 3.75,  expectedYearToGraduate: 2022);
x",  lastName: "Tiffen",  declaredMajor: "Computer Science",  gpa: 3.45,  expectedYearToGraduate: 2021);
```

we called increment expected graduation year on profile two.

```
public class Main {

    public static void main(String[] args) {
        StudentProfile profileOne = new StudentProfile( firstName: "Sally",  lastName: "Salmon",  de
        StudentProfile profileTwo = new StudentProfile( firstName: "Max",  lastName: "Tiffen",  decla

        profileTwo.incrementExpectedGraduationYear();
        System.out.println(profileTwo.expectedYearToGraduate);
    }

}
```

When we printed out profile two's expected year to graduate instance variable to the console, we get 2022.



This is exactly what we expect. For your convenience, the sample files for you to practice are located on the Java Basic MS team room under the files tab.

Congratulations. You finished the course. From creating your very first Java program, to understanding the fundamentals of classes and instances, you've learned a lot. From here you can use some of your Java foundations to build your own Java programs. You can also learn more on Learn computer programming | Online courses from JetBrains Academy where you can take both Java Beginners and Advanced courses.

Final Demo

Schedule a demo with the Topic Advisor (TA) to show the results of your labs.

Next Steps

- Congrats on completing CELL!
- You will also receive a Completion email with next steps and link to CELL experience survey
- Consider continuing to support others by collaborating with other learners and being active in MS Teams
- Use your new skills and consider enrolling in another CELL in which Java is a pre-requisite, such as ChatBots with Java or Git
- If you noticed any errors on this page, please let the CELL Program Team know by sending an email to the Technology Learning Team mailbox: g31488@att.com
- Check out the **Software Dev CLT** to view more learning options

Previous Iteration	tWiki Home
Iteration 3	tWiki Home Page

Previous Iteration	Wiki Home	Forums
Iteration 2	Wiki Home Page	Forums