Java Basics Iteration 2

Previous Iteration	Wiki Home	Forums	Next Iteration
Iteration 1	Wiki Home Page	Forums	Iteration 3

Previous Iteration	tWiki Home	Next Iteration
Iteration 1	tWiki Home Page	Iteration 3

Table of Contents

Iteration Overview

In this iteration, you will learn about data types, variables, strings, input/output, control flow, operators, and while loops. You will also create decision-making and multiple-choice programs.

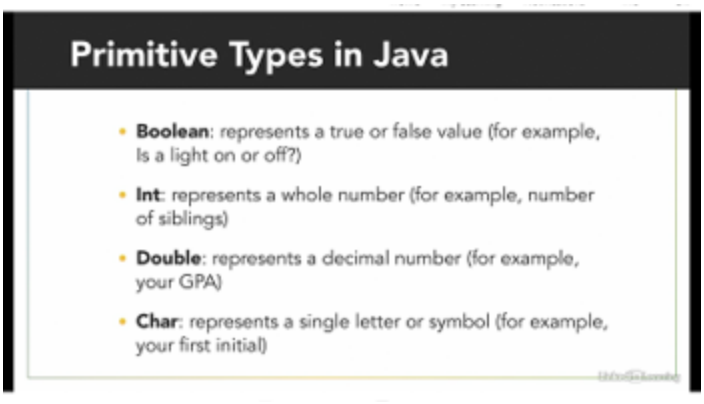
Save screenshots of your work along the way to show the Topic Advisor during your demo at the end of this iteration.

Primitive data types

In computer science, data is information that is stored or processed by a computer. While that might seem fairly abstract, there are many data points we use in everyday life. Your name, your age, the number of apples in your pantry, whether your kitchen light is on or off. These can all be considered pieces of data, and we represent pieces of data in code using data types. Similar to other programming languages, Java classifies different pieces of data with data types based on their value. For example, there's a data type for letters and symbols, and there are various data types for numbers. Ultimately, a data type provides a set of possible values, and if a piece of data is one of these values, it is classified as that specific type.

Java separates its data types into two main categories that are then broken down further into more distinct data types. One of the categories is called primitive data types which consist of the most basic data types in the Java language. The most important primitive data types include Boolean, int, double, and char. There are other primitive types that exist in Java, but these are the most foundational. The second category is called reference types, and we'll talk about reference types in a later lesson.

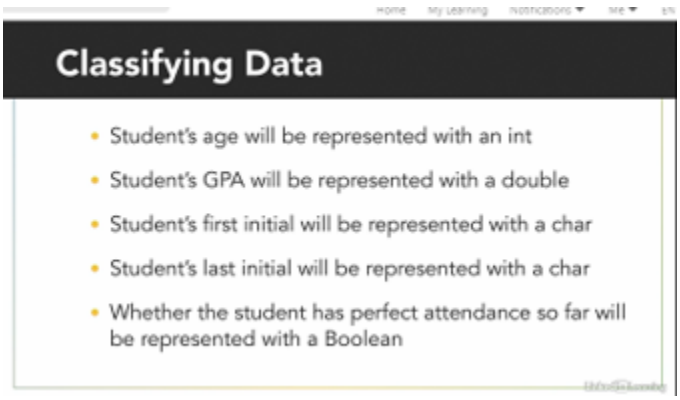
Diving deeper into these primitive types, the Boolean data type represents a true or false value. This means the data piece of whether the kitchen light is on or off could be represented as a Boolean in code. It might have the value true, or it might have the value false, but it can never be both. It must be either true or false. Int, short for integer, represents a whole number. The number of siblings you have could be represented in code as an int. You could have zero siblings, three siblings, 10 siblings, any whole number of siblings. A double represents a decimal number. With a double, you can represent your GPA, say 3.4, in Java. The last primitive type we'll talk about is a char which is short for character. A char represents a single letter or symbol. Some possible values for a char could be your first initial or last initial, your favorite letter, or the hashtag symbol. We classify these data types, Boolean, int, double, char, as primitive types because they are the basis in foundation for all other data types within the Java programming language.



Let's try using our primitive types in Java.

Data types and variables in Java

In this lesson, we'll be creating a few pieces of data that could represent attributes of a given high school student. Our program will have representations for a student's age, GPA, first initial, last initial, as well as if the student has had perfect attendance so far this year. Before we get to the code, let's assign a Java data type to each of these pieces of data an age could be represented in Java with either an int or a double depending on how exact we want to be. In this case, we'll be using an int to represent the age. Potential values for a student's GPA could be 3.4 or 2.7 or 4.0. These are all decimal values, so the data type for a student's GPA would be double. A students first initial and last initial are each a single character, so both of these would be represented by the data type char. A student can either have perfect attendance or not have perfect attendance, there's no in-between. So, has perfect attendance should be a Boolean. Let's try creating some Java code that represents these pieces of data.



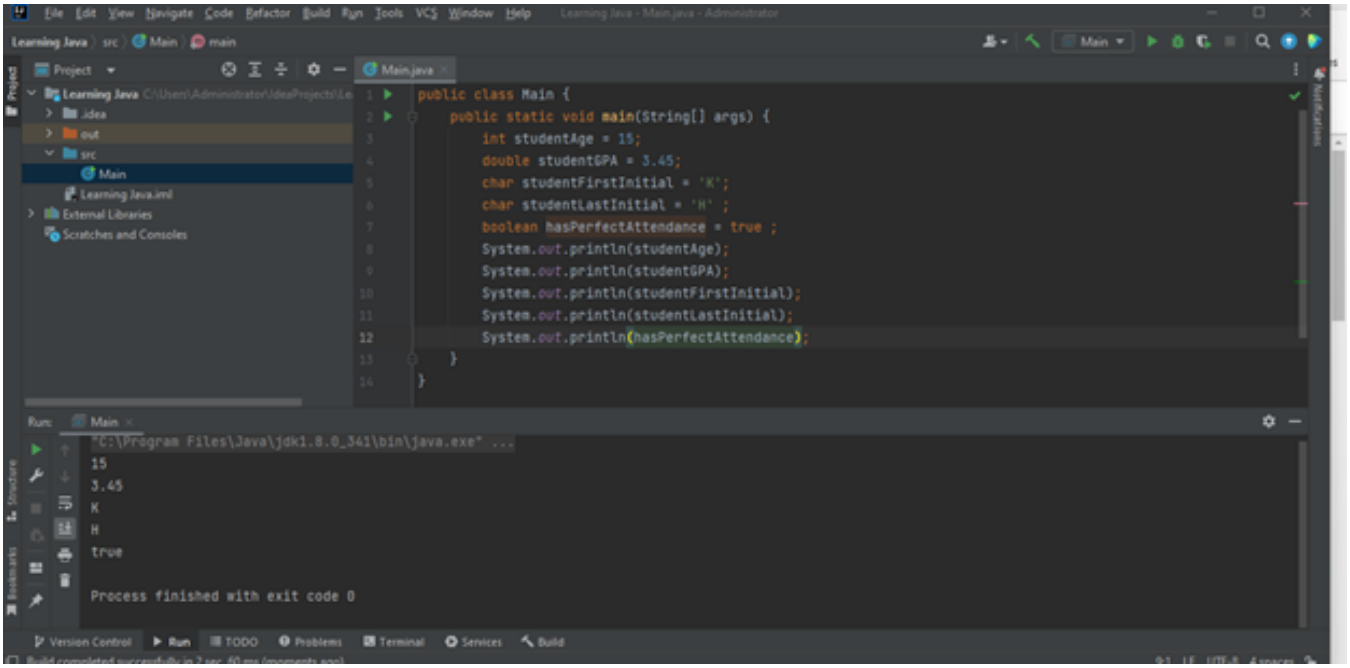
We could represent a student's age by just writing 15. Then we could represent a student's GPA with just 3.45. However, we get errors. What we've written are technically numbers and pieces of data, but in Java we save data in something called a variable. A variable is a way to label data and reference it later in your program. Instead of saying 15, we can create a label, say student age, and then assign a value to it, in this case, 15. We still get an error here, and that's because we need to add a data type to the variable. We can add int at the beginning of the line and then a semicolon at the end of the line to end the statement, and all of our errors are gone. Congrats, you just created your first variable in Java.

As you can see, a variable has a data type, label, and value. In this case, the data type is int, the label is student age, and the value is 15. The equal sign is an assignment operator that assigns the int variable with the label, student age, and the value 15. Let's create the next piece of data in Java, a student's GPA. We need to create another variable, and we decided before that the data type would be double. So, let's write double and give the variable the label studentGPA, and then we'll assign it the value 3.45 and add a semicolon to end the statement. Second variable, done. Three more to go.

Next we'll represent a student's first initial with the data type char studentFirstInitial as the label, and give it the value K. We'll do something similar for the students last initial with the data type char label studentLastInitial and give it the value H. For all of these variables, you could put a different value as long as it can still be represented by that given data type. For example, you couldn't put a letter or symbol as the value of student age, but you could put another whole number, say 17 instead of 15.

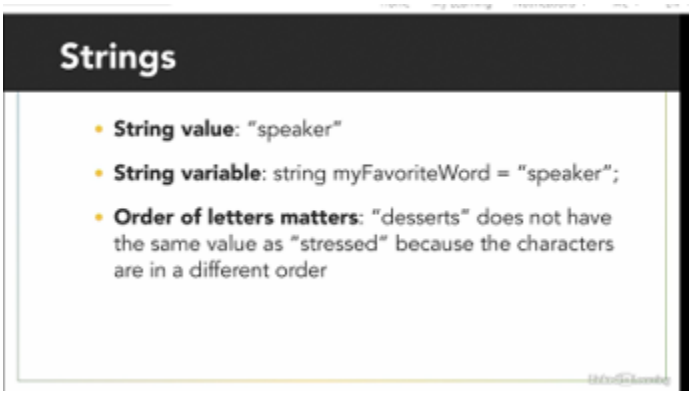
The last piece of data we want to represent is perfect attendance. Like the other variables, we'll write the data type, Boolean, the label, hasPerfectAttendence, and then give it the value true. You may notice that our variable names are grayed out, that's because although we've created a piece of data and represented it in Java, we don't use or reference it in our program. One way we could reference it is by printing out the value of the variable to the console. We'll write System.out.println, and instead of adding quotes, we'll just write the name of the variable. Since studentAge has the value 15, we'll print out 15 to the console. We'll save this and then click the play button, or you could left click and hit Run Main.main, and we get 15 in our console.

Let's try this with the other variables. We'll write System.out.println studentGPA System.out.println studentFirstInitial System.out.println the last initial and then the student has perfect attendance. We'll run this again and we get 15, 3.45, K, H, and true as expected in the console.



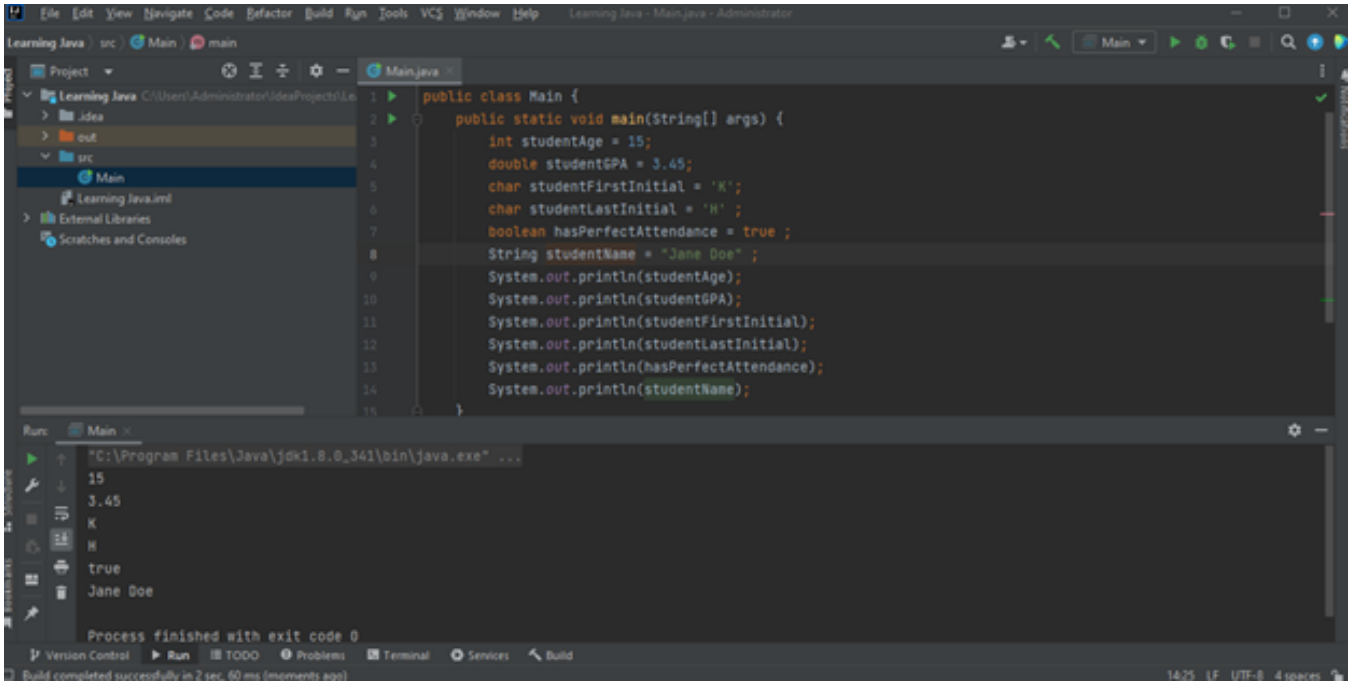
Strings in Java

We've learned about a few data types. These data types, int, Boolean, double, char, were primitive types. These are great for storing a whole number, true /false values, a single letter or symbol, but what if you wanted to store and reference some text, something that requires more than just a single character? In this lesson, we'll be looking at the second overarching category of Java data types called reference types. And our first reference type is called a string. A string is a sequence of ordered characters. Before, we could represent a single character with a char, but now we can represent a word or a person's name with the string data type. To create a string, we surrounded the series of characters with quotation marks. This represents a string value. To create a string variable, we just state the data type string, give the variable a label, and then use the assignment operator to assign the variable the string value. We should also note that the order of characters does matter.

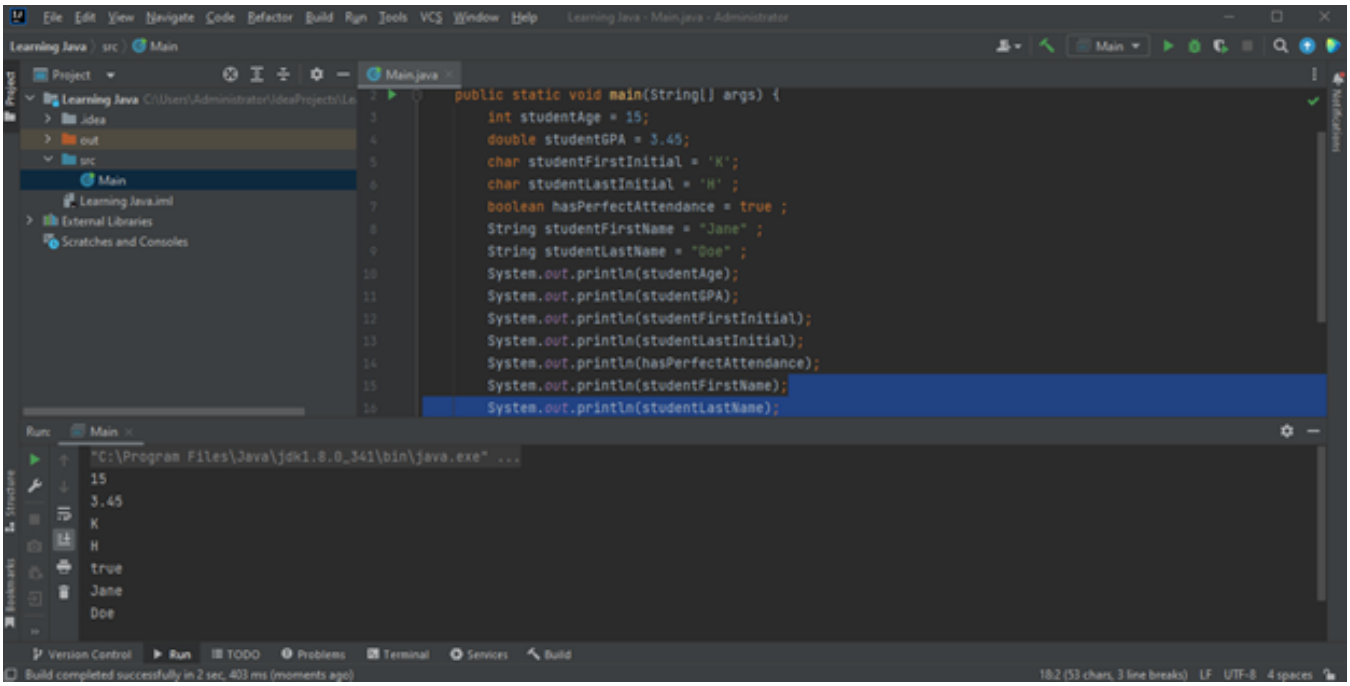


So, let's say we have a string with "desserts," and then we have a different string with "stressed." Although these two strings have the same letters, they would not be considered as having equal value, because the letters are in different orders. So, what makes a string a reference type? Unlike primitive types and the other data types we've looked at, a string is built out of characters. It is not a basic data type. It is created out of organizing char data pieces in a significant way. The deeper meaning behind reference types is not covered in this course, but it is highly recommend you check out Programming Foundations: Data Structures to find out more.

Now that we know about strings, let's create a string in Java. Using our program from the last lesson, we'll add a studentName variable that holds the student's name. We'll write String studentName, and in this case, the student's name will be Jane Doe. With the variable created, we'll print it out with System.out.println and the variable name studentName. We'll run this in the console.

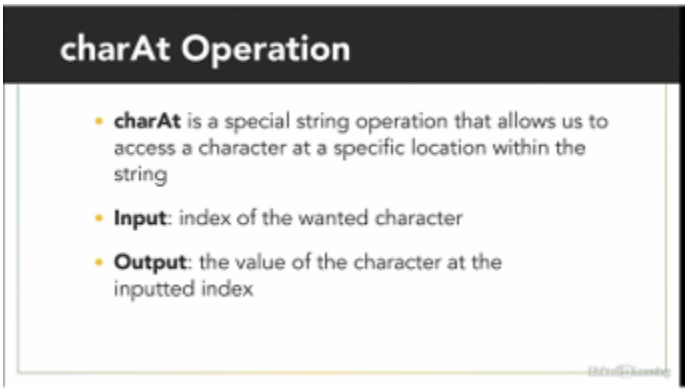


And here we see Jane Doe. Notice in the value of studentName, the student's first and last name is included. This is a design choice. Instead, we could create two separate strings, one holding the first name and another holding the last name. Let's go ahead and do that. Instead of studentName, we'll write studentFirstName, and we'll move to String studentLastName and be the value. And then this is red because studentName no longer exists, but studentFirstName does, so we'll print out the value of studentFirstName, as well as the value of studentLastName. We'll save this. Run it. And we see Jane and Doe in lines separate in the console.

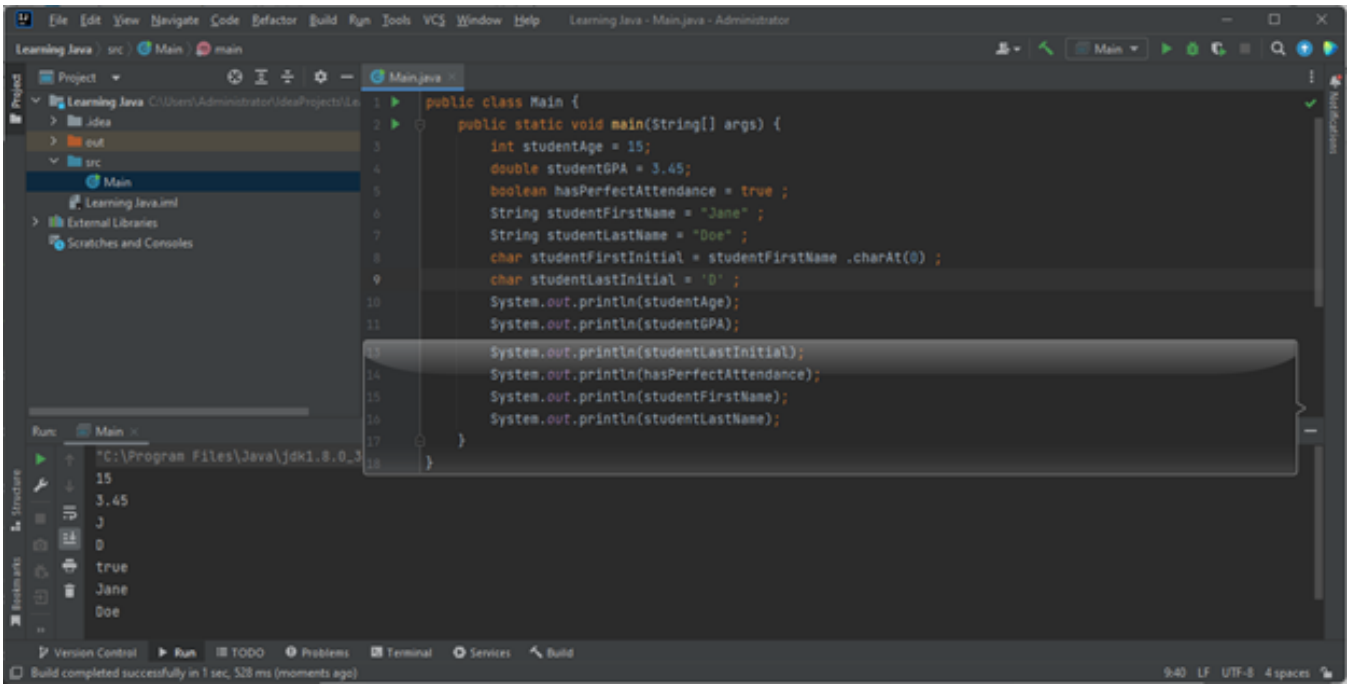


Using indexes with strings

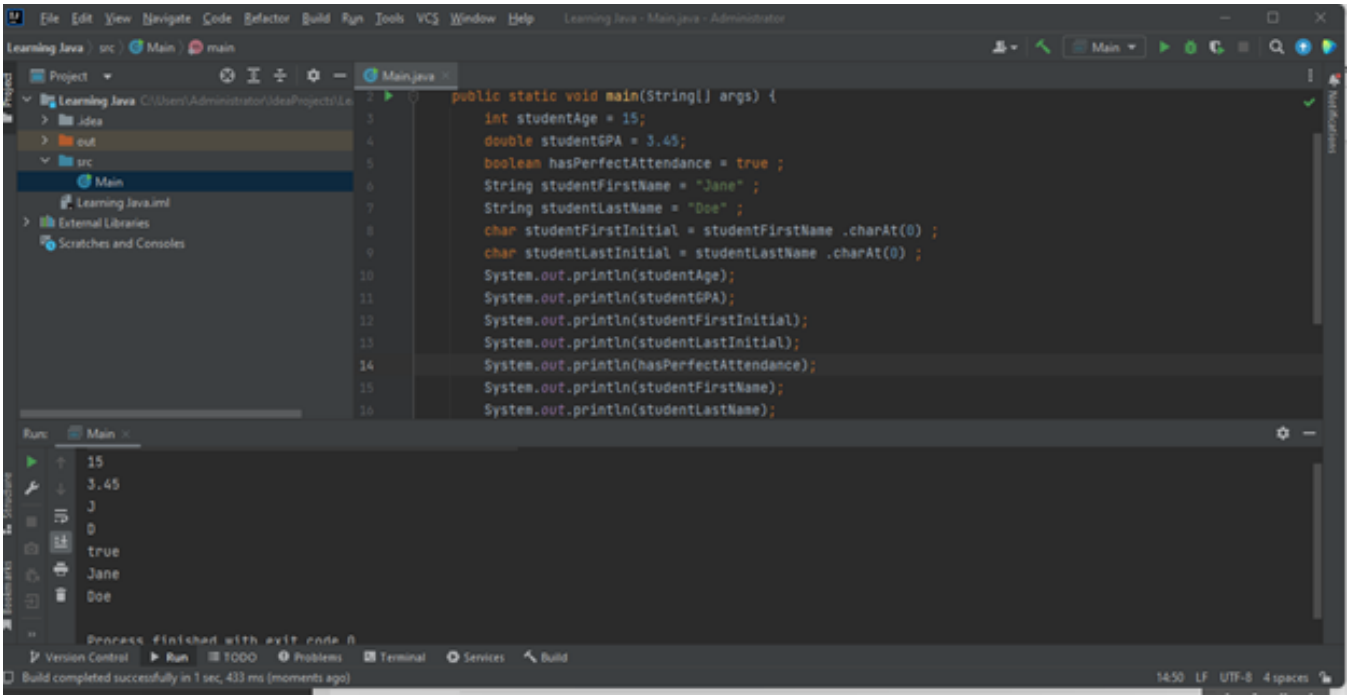
Now that we have the student's first and last name separated, we can calculate the student's first initial from the first name and last initial from the last name. All we need to do is somehow grab the first letter of the first name and the first letter of the last name to create each initial. Earlier we talked about how the ordering of the letters within a string matters. And this is one place where that comes into play. For the string Jane, we say J is at index or location zero. A is at index one, N is at index two, E is at index three. This means that in order to calculate the first initial of the first name, we'll want to access the character at index zero, or the zeroth slot in the string. To do this we can use a special string operator to access the letters inside the first name. To grab the first character of the first name, we can use an operation called charAt. CharAt allows us to access a character at a specific location within a string.



It takes one input, and that is the index or location of the character we want to access. Going back to the code, we can change the value of studentFirstInitial to using charAt on studentFirstName. We'll write studentFirstName and then to use the charAt operation, we'll write .charAt. Next we'll add the location we want to access within StudentFirstName which is zero for the first initial. We'll save this and run it.



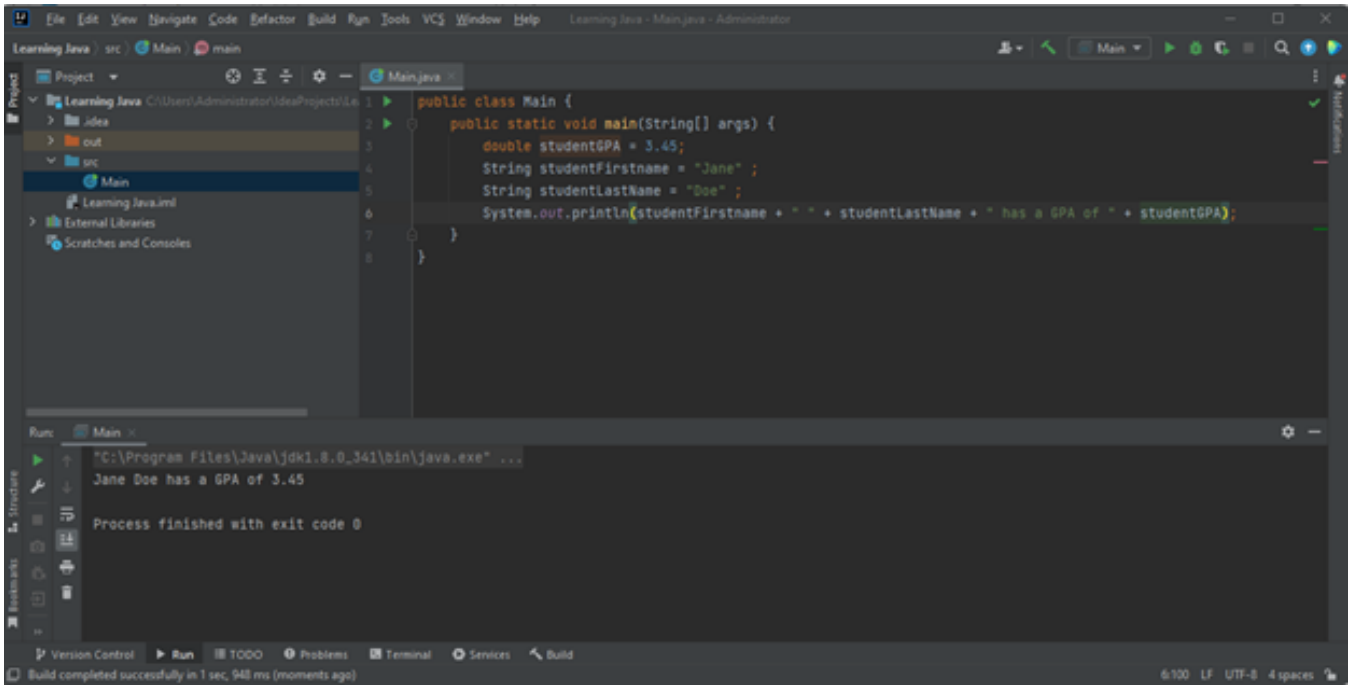
Scrolling down, we see studentFirstInitial is the second to last thing to be printed out, and we see J in our console. Exactly what we expect. Looking at the value of studentFirstInitial we write studentFirstName because that's the variable that holds the string we want to access the character from. We write dot because the operation we want to use is a string operation, and we get it for free with the string data type. Then we write charAt because this is the operation we want to use. And finally, we add our input which is zero because we want to access the character at the zeroth index. The output of this operation is J and that's what saved to the studentFirstInitial variable. Let's do the same for studentLastInitial. We'll write studentLastName because we want the first initial of the last name and use charAt to get that first initial of the last name with zero. We'll save this, run it and in our console, we get D as expected.



Great, we just calculated data from another data piece. A student's initials from a student's name. We also learned that the concept of a data type consists not only of the different values it can store, but also the different operations we can use with it. For example, our string data type has the charAt operation.

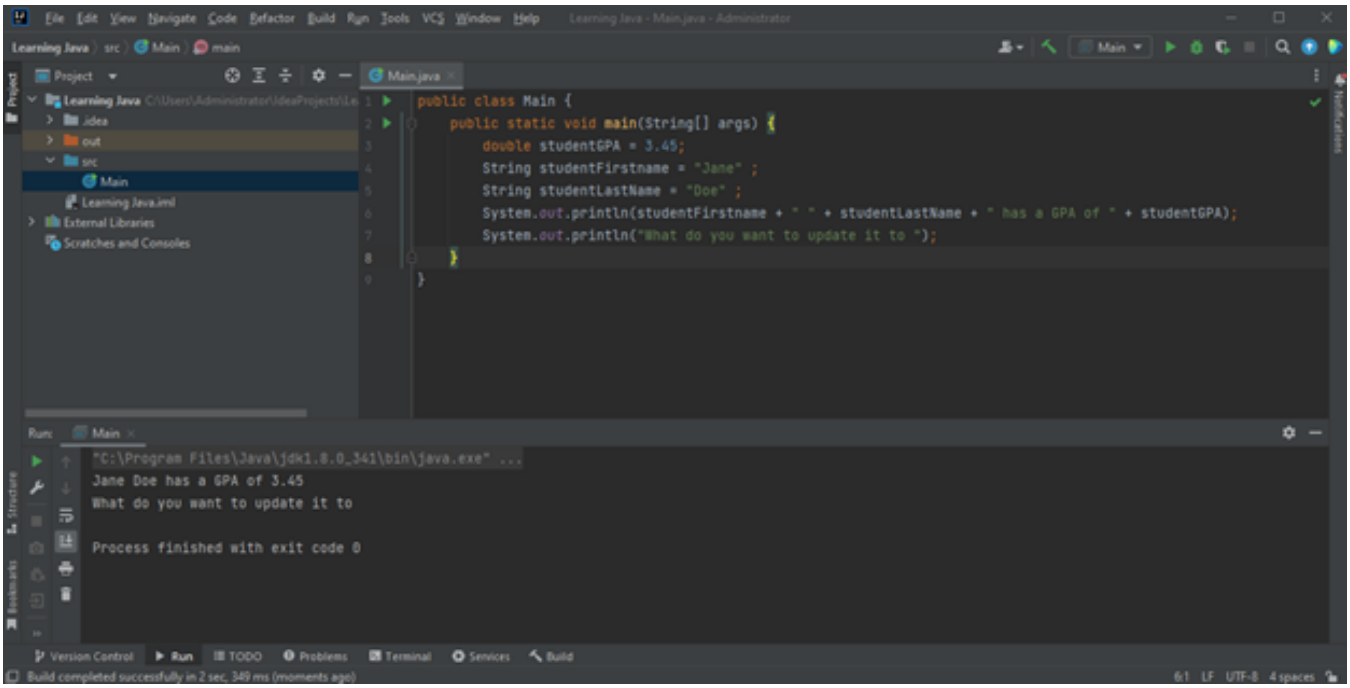
Concatenating strings in Java

So far, we've printed out the string, "Hello world," individually, and a few variables individually, but now we are going to learn how to print them together in the same line. To do this, we need to combine string values with variables that hold strings in a print statement. Using the example from the last lesson, we can print out a student's full name, and say that student has a GPA of X, with X being the student's current GPA. In the code, we'll write System.out.println, and then access the value of the student's first name, with studentFirstName. To add on the last name, we'll add a plus sign for concatenation, combining the value of these two strings. Then we'll type the variable that holds the student's last name, studentLastName. If we put the student's first and last name together like this, there will be no space between the first and last name when it is printed out to the user. We can add a space by adding a literal string, so a string that is not evaluated from a variable, with a space. We'll want to concatenate the other side, so we'll add a plus button between the space and studentLastName. For the next part of the print statement, we want to add, has a GPA of, and then the student's GPA. We'll tack on the literal string of "has a GPA of," with the plus sign, and then on the other side, we'll also concatenate the student's GPA. This value is stored in the studentGPA variable. We'll save this, run the program and in the console, we see the first and last name, Jane Doe, has a GPA of, and then the student's GPA, 3.45.



Input and output in Java

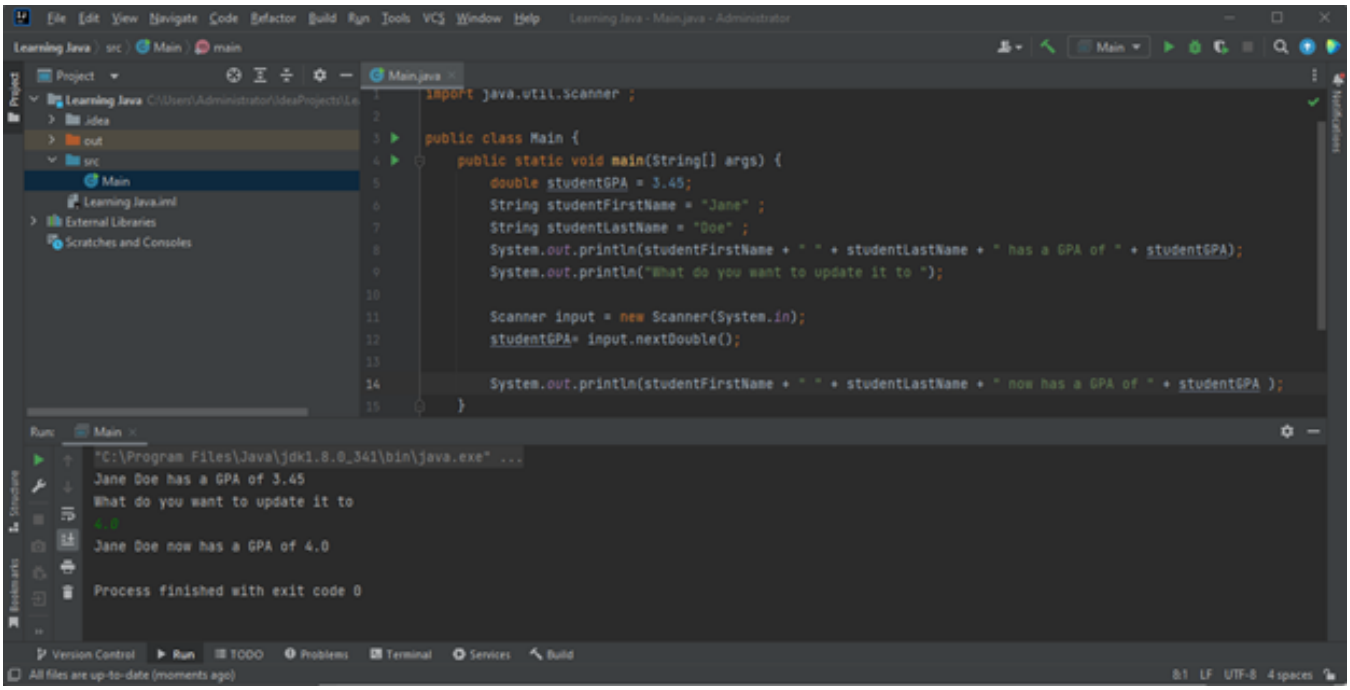
So far, we've learned how to output information to the user using System.out.println(). We've created several variables and printed out their values to the user as well as concatenated their values to String literals. In this lesson, we'll look at how we can create a program that allows the user to input information affecting the program's output. Thinking back to our student program, we might need to update a student's GPA. With some updates to the code, we can let the user dynamically change the value of a student's GPA. In the previous lesson, we wrote a print statement that tells a given student their GPA. Before having the user update their GPA, we should ask the user what value they want to update the GPA to and that's the print statement added here.



Next we need to get access to the user's input and update the value of the student GPA variable. To do this, we can create a Scanner with System.in and save the Scanner in the variable labeled input. This is the opposite of the System.out we have been using to print to the console. The Scanner has some built in operations that will allow us to retrieve input from the user. This means we'll write input = new Scanner (System.in). This creates a new Scanner that will scan System.in for the user's input. We get an error here because we need to add a data type for our variable. The data type here is Scanner. There are lots of data types in Java, but for now you can think of the Scanner as a tool that has operations that allow us to get input.

In order to use those operations, we have to create a Scanner first. We also need to import java.util.Scanner in order to use the Scanner. Think of it as a special data type that we need to import in order to use. With the Scanner created. We can use the Scanners next double operation to get the next double or decimal value that the user enters in the console window. Similar to how we used the chart at operation on Strings. We'll use the nextDouble operation on our input scanner. So, we'll write input.nextDouble, we'll want to save the value in our student GPA variable, so we'll add student GPA = to assign the user's inputted value to the student's GPA variable. Since we've already introduced and defined the student GPA variable before, we do not need to add the data type because Java already recognizes this variable as a double from its initial definition and it can only store double values.

Now we'll create a nice output String so the user can see that the student GPA has been updated. Let's run the program. Jane Doe has a GPA of 3.45. What do you want to update it to? We'll do a 4.0 (type 4.0 on console), Jane Doe now has a GPA of 4.0.

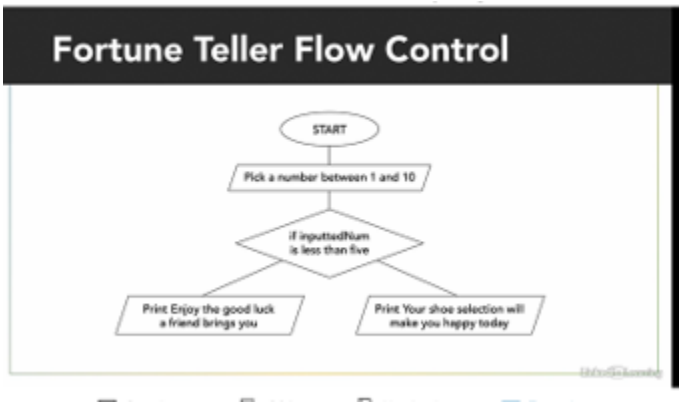


Mapping out program control flow

We've looked at how we can represent data in Java using data types and variables, as well as how to input and output data. We are going to add some decision-making logic, that will allow our programs to have different outcomes. We saw this a little bit in the last lesson, when we printed out whatever the user inputted, but we can create more sophisticated control flow, using conditions.

Let's break this down. What is a control flow? A program's control flow is the order in which the program's instructions are executed. All of the programs we've looked at so far execute one statement after the other. We created a variable, then printed the variable. Or we created one variable and then another variable and then printed them both. Each line of code was executed sequentially. For example, line one is executed before line two, line two is executed before line three and so on. We can manipulate which line of code is executed with special control flow statements and conditions. With these control flow statements, a line of code might never be executed. It might be executed once, or it might be executed multiple times. The conditions determine how many times, if at all, a given line of code is executed.

In this lesson, we'll take a look at the control flow of a virtual fortune teller program. With the fortune teller, the user will pick a number between 1 and 10. Depending on the number the user picks, the user will receive a fortune. Let's dive into this diagram.

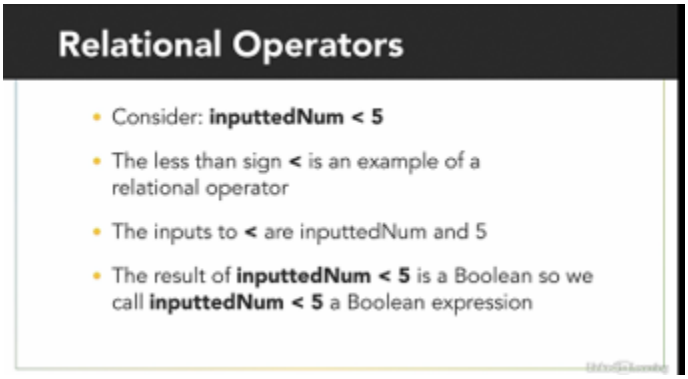


This maps out the control flow of our program and describes how we want our program to work. Notice everything is not sequential. We start at the start and the fortune teller, our program, will ask the user to pick a number between 1 and 10. The circle represents the start, and the parallelogram represents input or output. The diamond represents the decision, and that decision determines the choice and ultimately which print statement in fortune is run. Our one decision inside the diamond is asking the question, is inputtedNum less than five? This is our condition. Based on whether this is true or false, one of the true print statements will run. If the inputted condition is less than five, meaning the condition is true, the lower left lock is run. If the inputtedNum is not less than five, meaning it is equal to five or greater than five, the lower right block is run. Only one of the fortune print statements will be run. Not both or neither.

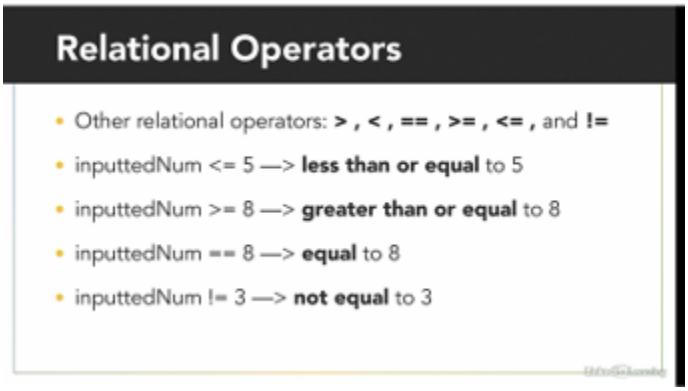
Now, we know how to program some of this. We know how to output, pick a number between 1 and 10. We also know how to get access to the users input and save it to a variable. We do not know how to implement or write Java code for a decision block. We call these decision blocks or control flow statements because they decide which code statements to run on each execution of our program. In our case, the decision block is asking the question, is the condition inputtedNum less than five. Depending on whether the condition is true or false, meaning if inputtedNum is less than five or not, a given code statement is run. Of course, right now, this is just a diagram. It's a representation of how we want our program to work in Java. Now that we have an idea on how we want our program to work, we can take this decision block in the diagram and break it down piece by piece, so that we can implement it in Java.

Operators in Java

For the decision block in our control flow, we have three main components. We have a condition, the code that gets run if the condition is true, and the code that gets run if the condition is false. To simplify our condition, we can just write out our condition with the less than sign instead of writing it out in English. This is closer to what it will look like in our code. Ultimately, the value of inputting them or the number that the user inputs, will determine which print statement is run, which is exactly what we want. We call the less than sign a relational operator. Its inputs are inputtedNum and five. And the relational operator, less than, states a comparison between these two inputs. The result of this overall comparison will evaluate to either true or false. Since the inputtedNum less than five comparison evaluates to a Boolean, we call it a Boolean expression. And a Boolean expression can be used as a condition for our decision blocks. Of course, less than is just one relational operator. There are many others we could use here instead. This is a list of other relational operators we can use in our conditions.



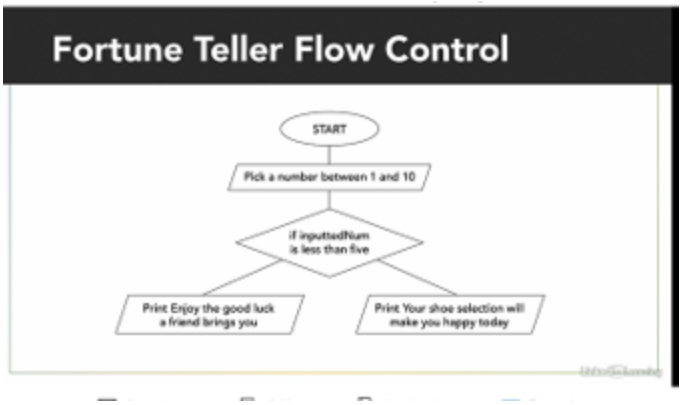
If we wanted to make our relational operator be less than or equal to instead of just less than, we can add an equal sign and it would test if the inputtedNum is less than or equal to five. Similarly, for the greater than sign, we would just add an equal sign to test if the inputtedNum is greater than or equal to eight. To check for equality, we would use double equal signs, and this would check if the inputtedNum is equal to eight. If it is, the statement would be true. If not, the statement would be false. To check for inequality, we use an exclamation point and an equal sign. In this case, the statement would be true as long as inputtedNum does not equal three. It would be false in the case that inputtedNum equals three.



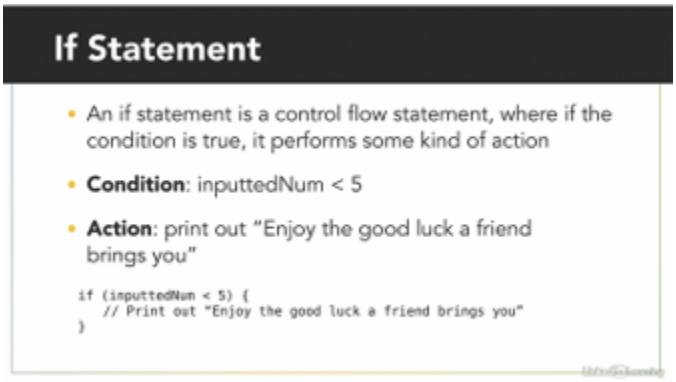
Ultimately, a condition in the decision block evaluates to true or false depending on some variable. That's what makes it a condition that can be evaluated during the program's execution and manipulate a program's control flow. Now that we have an idea of how decision-making works in our programs, let's try implementing the fortune teller logic in Java.

Decision-making with if in Java

Let's review our Fortune Teller program. There are three main parts to this program. First, we ask the user to pick a number between 1 and 10. Next, we output the user's fortune depending on which number is inputted.



Let's implement this program. In the code, we already have a print statement that asks the user to pick a number between 1 and 10. We also have a scanner that reads in the next int a user inputs with the nextInt operation. We save the output in a variable called inputtedNum. Now, we have to make a decision in our program. If the inputtedNum is less than five, the program should output the good luck fortune. If the inputtedNum is equal to or greater than five, the program should print out the shoe selection fortune. Before, we were just using blocks, but this is a specific type of control flow statement that starts with an if, so we call it an if statement. An if statement is a control flow statement, where if the condition is true, it performs some kind of action. In this case, our condition is inputtedNum less than five.

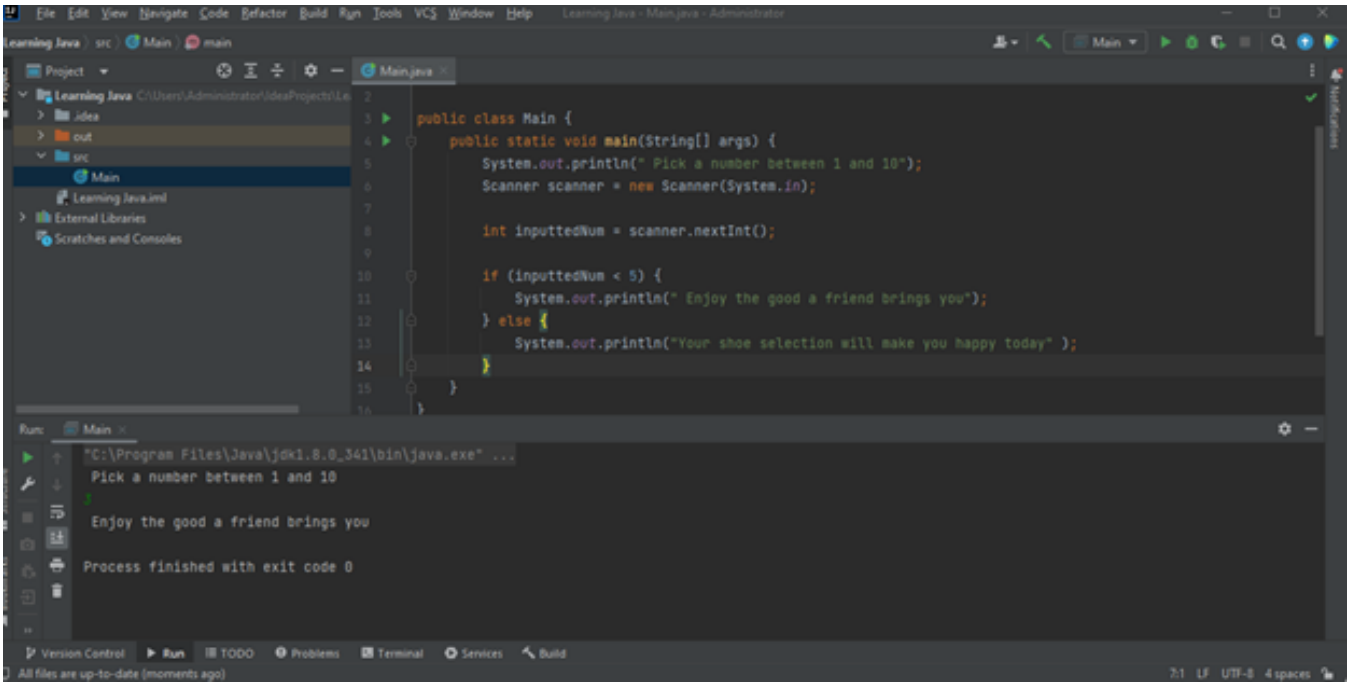


The condition will either be true or false. For our program, if an inputted number is less than five, meaning the condition is true, then we execute the code in the if block or in the curly brackets. In this case, if the inputtedNum is less than five, we would print out enjoy the good luck a friend brings you. This print statement would only be executed if the condition is true meaning inputtedNum is less than five.

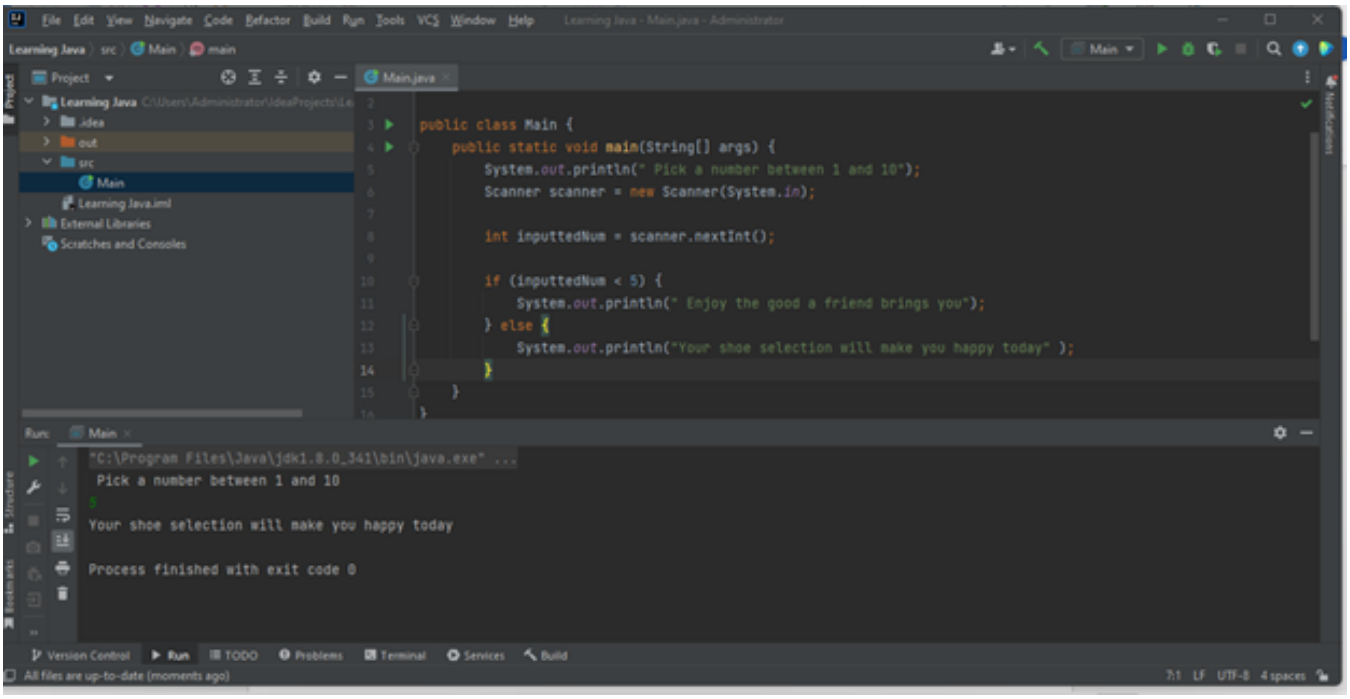
So, what if inputtedNum is not less than five? Then the condition evaluates to false. When we mapped out our program in an earlier lesson, we said that if inputtedNum was not less than five, then we wanted to print out the other fortune. To do this, we add an else statement to our if statement. The curly brackets with the else statement encompass the code that will be run if the condition is false. Similar to the if block, the code inside the else block only runs if the condition is false. We don't always have to have an else block associated with an if block, but we do in this case because we want to perform an action if the condition is not true.

Let's add an if statement to our code. We'll write if and then our condition, inputtedNum is less than five, we'll create our if block with the curly brackets and we'll write System.out.println and then we'll write the else block. And there we have our if else. Only the if block or the else block will be run in an execution of the program because the condition cannot be true and false at the same time. The condition can be true, or it can be false but only one fortune will be printed. This means that the code inside the if block doesn't know about the code inside the else block, and the code inside the else block doesn't know about the code inside the if block. This will become more important later one.

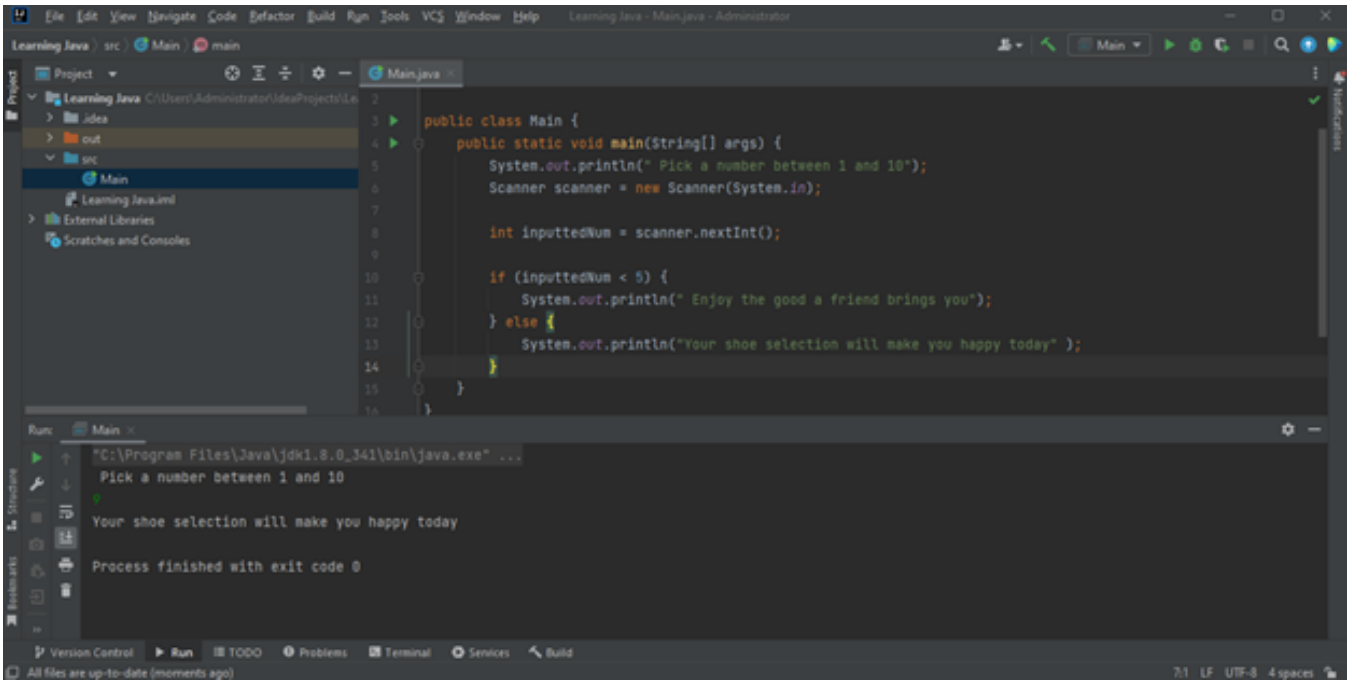
Let's run the program. We'll click the play button and pick a number between 1 and 10; we'll pick 3. We get the first fortune. Enjoy the good luck a friend brings you and that's because 3 is less than 5.



We'll run the program again and choose 5. In this case, five is not less than five, it's equal to five and so we get this second fortune, your shoe selection will make you very happy today.



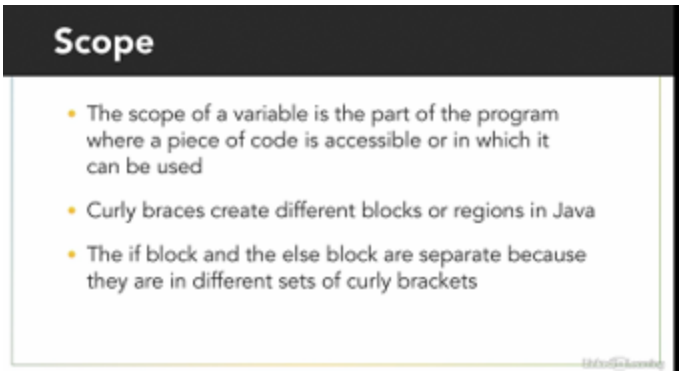
We'll try one more time. We'll put in 9 as our input, and we get that second fortune again because nine is not less than five.



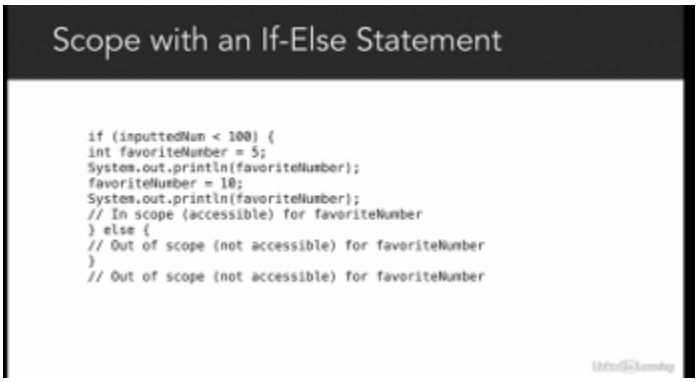
This condition is false. Next, we'll take a look at a concept called scope and show how it relates to if statements.

Understanding scope in Java

In the Fortune Teller program, during the program's execution, only the if block or the else block was executed. All of this had to do with the topic called scope. Scope refers to the region of the program where a piece of code is accessible or in which it can be used. Every time we use curly braces in Java, we are creating a block in the program. This is why the if block and the else block are separate. They are in different sets of curly brackets.



Why does this matter? Let's say an int variable named favoriteNumber, with the value, five, was created in the if block. This variable scope is within the block in which it was created. That's where the variable is accessible and can be used. It can be referenced or its value can be changed anywhere inside the if block. However, since it was not created in the curly braces of the else block, it cannot be used in the else block, because the else block is outside this variable's scope.

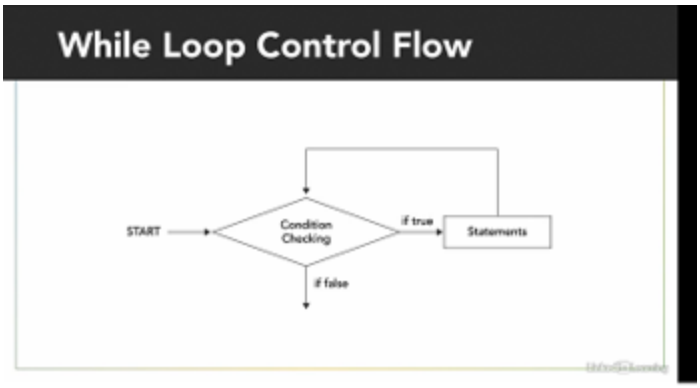


Now, let's say we created a string variable called favoriteFood outside the if-else block in the main function and gave it the value, pizza. Since favoriteFood is created outside the if-else block in the parent set of parentheses, this variable can be accessed and assigned a new value within both the if and else block. The favoriteFood scope, or where favoriteFood can be accessed, is anywhere within the curly braces it was created in.

Let's take a look at some other types of control flow statements.

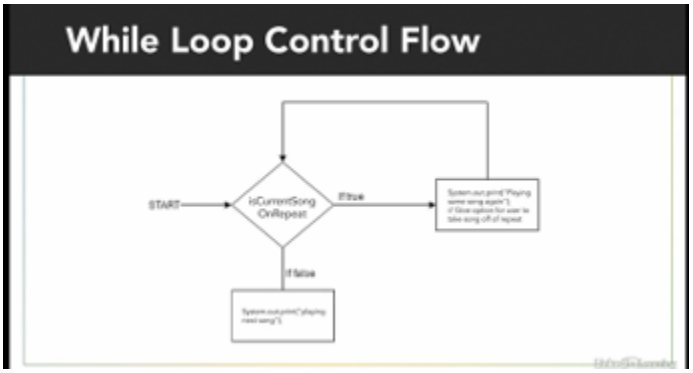
While loops (conceptually)

If statements aren't the only way we can add decision making to our programs. We also have loops. A loop is similar to an if statement, except that it allows code to be executed repeatedly, based on a Boolean condition, instead of just one time. There are several different types of loops in Java, but the one we'll be focusing on in this course is the while loop. A while loop looks like this.



We start at the start, and then follow the arrow to check a condition. If that condition evaluates to true, we execute a series of statements, and then check the condition again. If that condition is false, we exit the loop, and continue to the rest of the code.

On your smartphone, you've probably listened to music, and if you find a song you like, you put it on repeat. When a song is on repeat, it plays over and over again until you take it off repeat. We can represent this type of functionality in a Java program. Filling in the condition of the while loop, we can say our condition is: is current song on repeat? For the statements to execute, if the condition is true, we can print, "Playing same song on repeat," and give an option for the user to take the song off of repeat. If the condition is false, we can print, "Playing next song." Now, if is current song on repeat is never true, then playing same song again will never be printed. We'll never repeat the song. If the user never takes the song off repeat, then the condition will always be true, and playing same song again will be continuously printed. That's why we call it a while loop. While the condition is true, keep running these given statements.



Now that's while loop. Let's try implementing it.

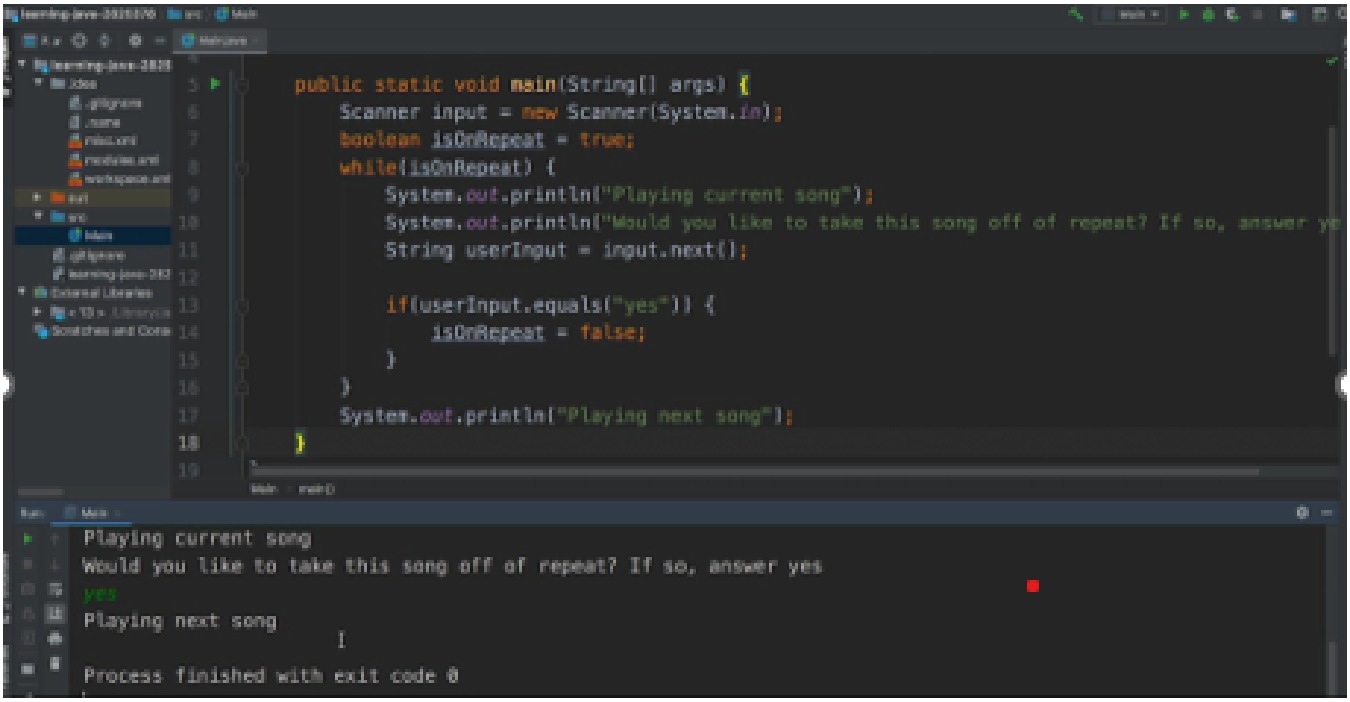
While loops in Java

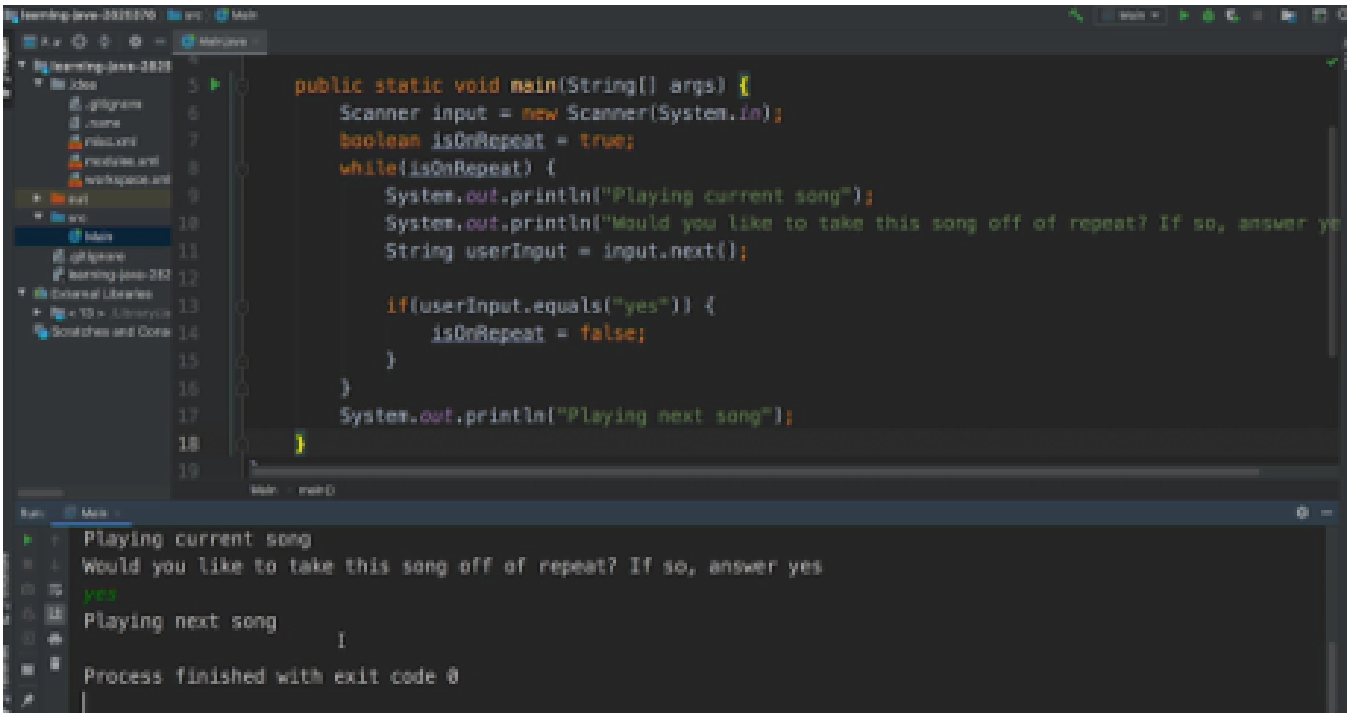
Let's create a while loop in Java. In the previous lesson, we diagrammed a while loop to represent a song playing on repeat. With the diagram, while the current song is on repeat, we play the song again. If the current song is not on repeat when we check the while loop condition, we exit the loop and play the next song. If the user never takes the song off of repeat, then the current song will play forever. We only exit the loop if the user takes the song off of repeat.

Let's jump to the code. The first thing we do is create a scanner to get us set up for user input. We also have a Boolean variable that represents if the current song is on repeat, and it equals true. Now, for the while loop. We want to say while our song is on repeat, meaning isOnRepeat is true, we want to run a series of statements. To do this, we'll use the keyword while and then our condition in parentheses. Then we'll use open/close curly brackets to create the while block for the statements we want to run while the condition is true.

Let's add those statements. We'll start by printing out that the current song is playing. On the next line, we'll ask the user if they want to take the song off of repeat. With these print statements done, we'll get access to the user's input with input.next and save it in a variable called userInput. Then, we'll use an if statement to check if the user's answer is yes, meaning we should change isOnRepeat to false. For the condition we'll write userInput.equals("yes"). Equals is a string operation we can use to check if the user input has the exact value yes. In the curly braces, we'll set isOnRepeat to false.

All right, we're almost done. We just need to add a print statement for when the program exits the loop that says, "Playing next song". So, after the while block we'll write System.out.println("Playing next song"). We'll save this and run the program. Playing current song. Would you like to take this song off of repeat? If so, answer yes. We'll say yes, we want to move on to the next song and it says, "Playing next song". When we went into the while loop, userInput did equal yes and so isOnRepeat was set to false, this means on the next check, isOnRepeat was false and we proceeded to play the next song. Let's run it again. Do we want to take the current song off of repeat? We'll say no and play the current song again. Do we want to take it off repeat now? No. In this case, isOnRepeat remains true, the input is never equaling yes and so isOnRepeat is never set to false. We could say no again. And now we'll finally say yes and that will set isOnRepeat to false and will play the next song.





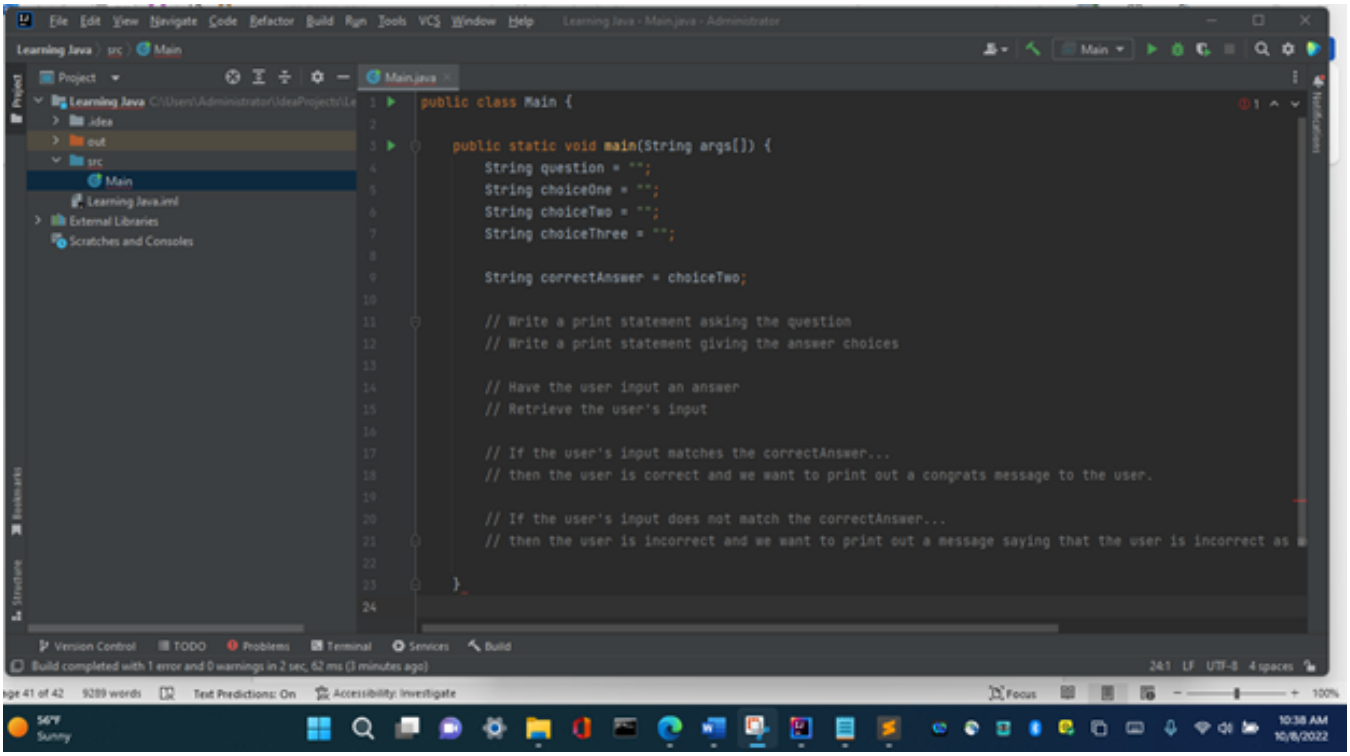
Challenge: Multiple choice

Let's practice control flow skills with this coding challenge. Be sure to use the tools you've learned about so far, including if statements, while loops, variables, and input-output. For this challenge, you will implement a single multiple-choice question in Java. You will come up with a question as well as three possible answer choices. One of these choices should be the correct answer. Using your knowledge of control flow and input and output, you will ask the user the question, as well as provide the possible answer choices. The user will respond with one of these choices. If the user is correct, we will print out a congratulations message. If the user is incorrect, we will print out that the user is incorrect, as well as what the correct answer choice is.

We've set up a few things for you in the code, so let's walk through what we have so far. First, we have four string variables. The first should contain the question. You will have to write in the question value. The next three contain answer choices. You will write in the values of the answers, as well. One of these choices will be the correct answer, and you should set that on line 10. The correct answer does not have to be choice two. It can be whatever you'd like.

You may have noticed these grayed-out notes below. These are called comments. Whenever we want to make a note in our code just to the developers viewing the code, we use a comment. They are not evaluated or used by our program. We create a comment with two backslashes, and then write whatever note we would like to share.

Let's get back to the challenge. Next, we'll write a print statement asking the user and then providing the answer choices to the user. Then you'll retrieve the user's input and check if it matches the correct answer. If the user is correct, print out a congrats message. If the user is incorrect, print out that the user is incorrect as well as what the correct answer choice is.

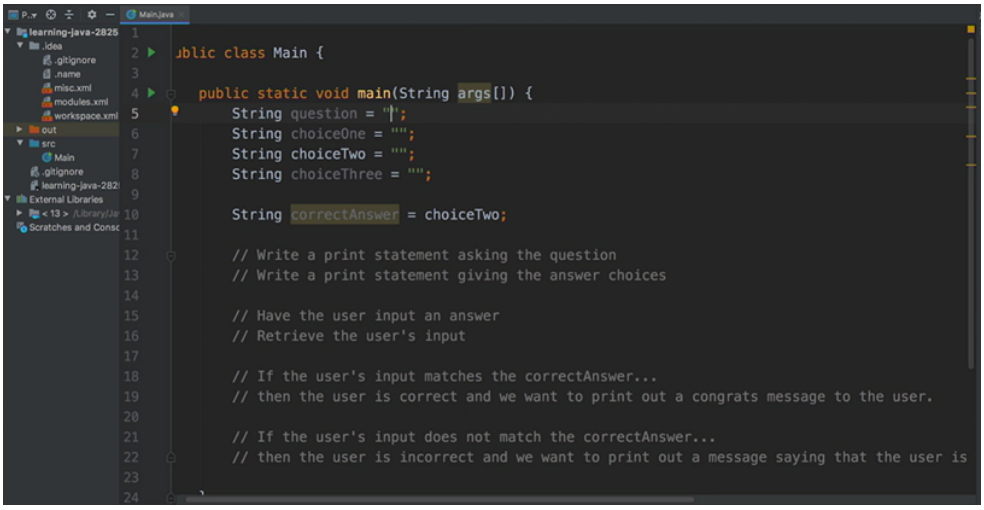


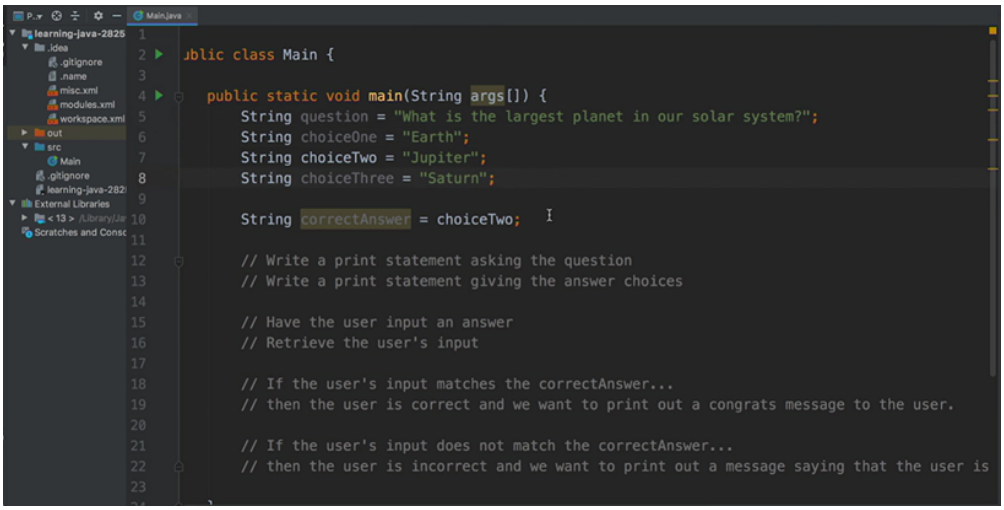
Sample solution: Multiple choice

Let's walk through how to create a multiple-choice program in Java. From the previous lesson we started off with this template.

```
public class Main {

    public static void main(String args[]) {
        String question = "";
        String choiceOne = "";
        String choiceTwo = "";
        String choiceThree = "";

        String correctAnswer = choiceTwo;

        // Write a print statement asking the question
        // Write a print statement giving the answer choices

        // Have the user input an answer
        // Retrieve the user's input

        // If the user's input matches the correctAnswer...
        // then the user is correct, and we want to print out a congrats message to the user.

        // If the user's input does not match the correctAnswer...
        // then the user is incorrect, and we want to print out a message saying that the user is incorrect as
well as what the correct choice was.
```

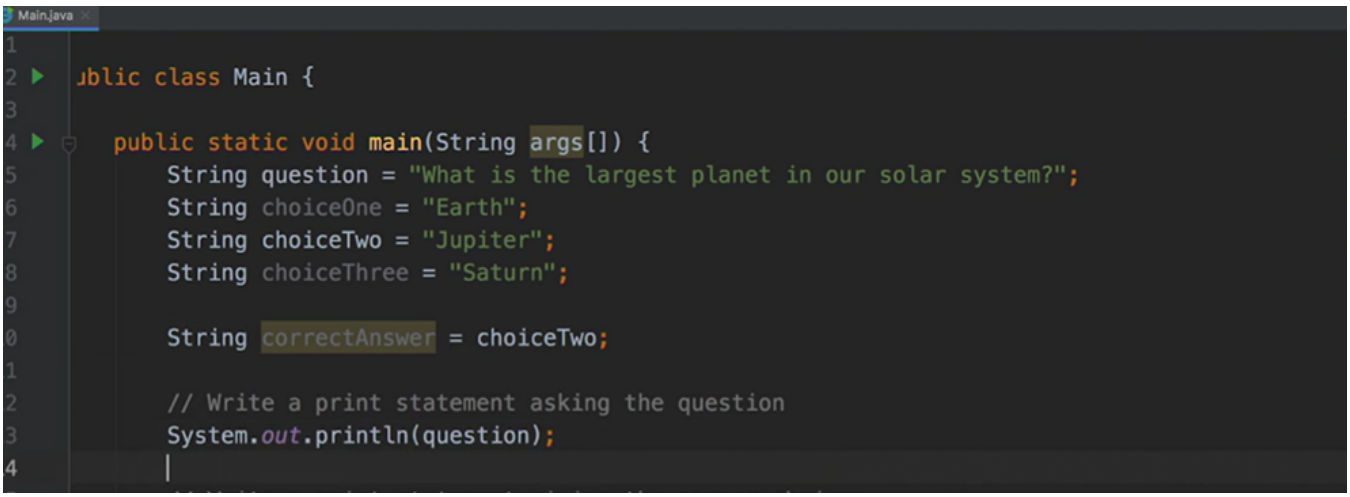
The first thing we need to do is come up with our question and answer choices. Our question will be what the largest planet in our solar system is. The answer choices will be Earth, Jupiter, and Saturn. We set the correct answer choice to be equal to choice two because Jupiter is the largest planet in the solar system. I could make choice three have the value Jupiter and then choice two have the value Saturn, but I would need to remember to update the correct answer to choice three instead of choice two.





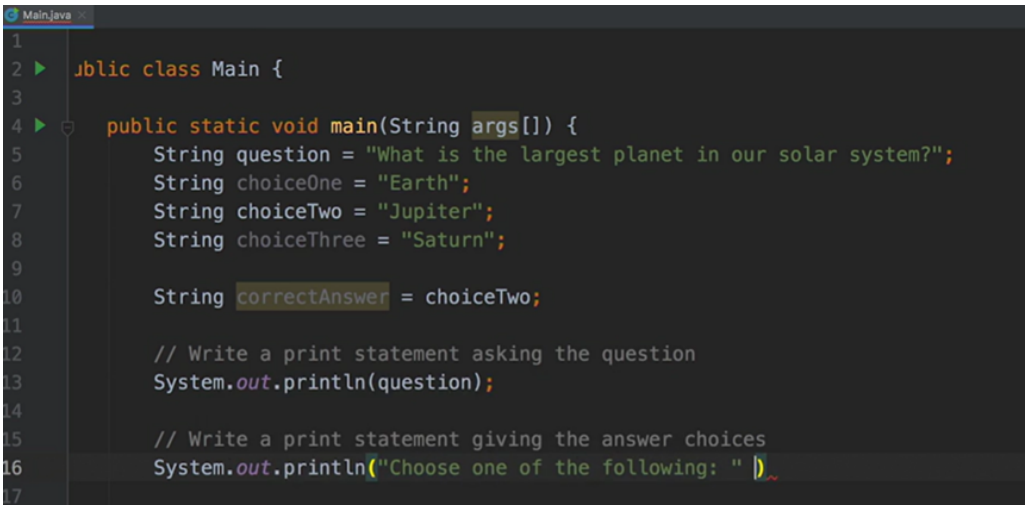
```
ublic class Main {

    public static void main(String args[]) {
        String question = "What is the largest planet in our solar system?";
        String choiceOne = "Earth";
        String choiceTwo = "Jupiter";
        String choiceThree = "Saturn";

        String correctAnswer = choiceTwo;

        // Write a print statement asking the question
        // Write a print statement giving the answer choices

        // Have the user input an answer
        // Retrieve the user's input

        // If the user's input matches the correctAnswer...
        // then the user is correct and we want to print out a congrats message to the user.

        // If the user's input does not match the correctAnswer...
        // then the user is incorrect and we want to print out a message saying that the user is
```

Next we'll ask our question.



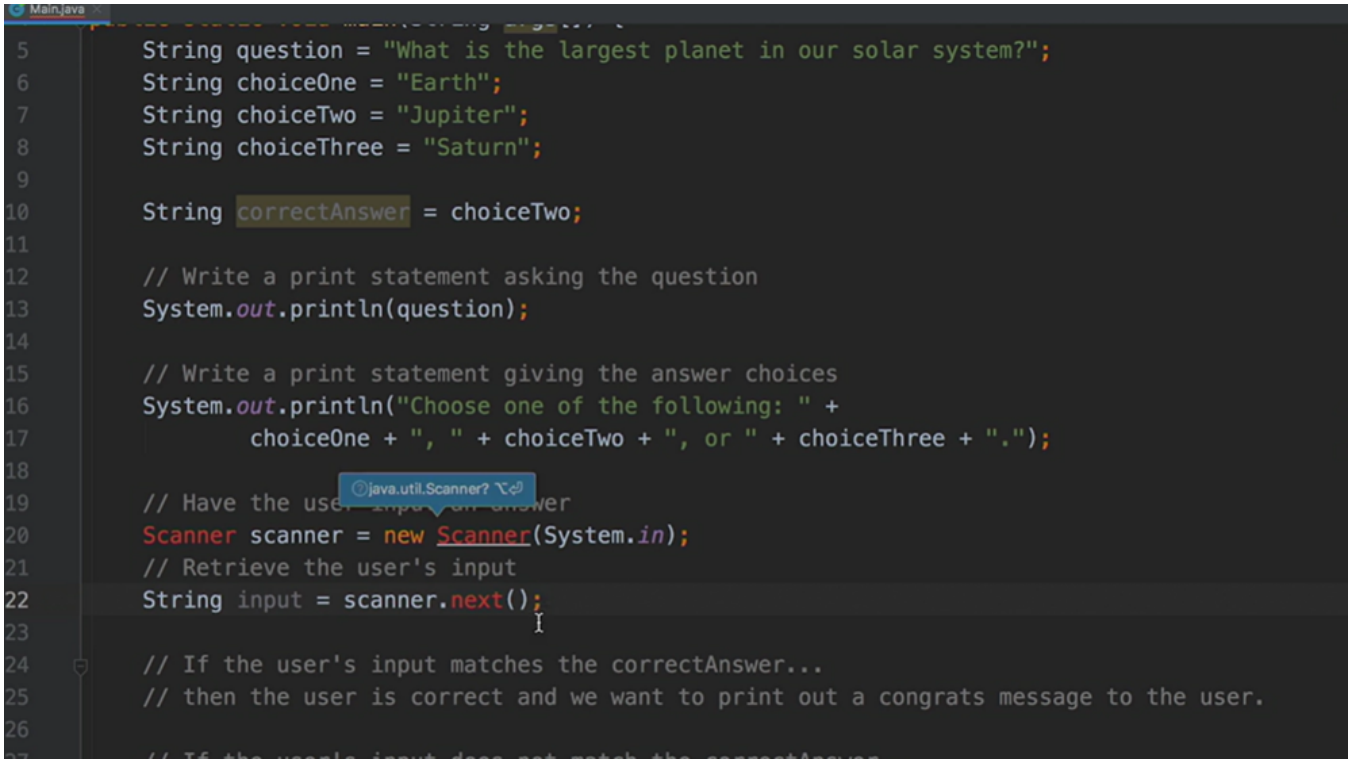
```
ublic class Main {

    public static void main(String args[]) {
        String question = "What is the largest planet in our solar system?";
        String choiceOne = "Earth";
        String choiceTwo = "Jupiter";
        String choiceThree = "Saturn";

        String correctAnswer = choiceTwo;

        // Write a print statement asking the question
        System.out.println(question);
```

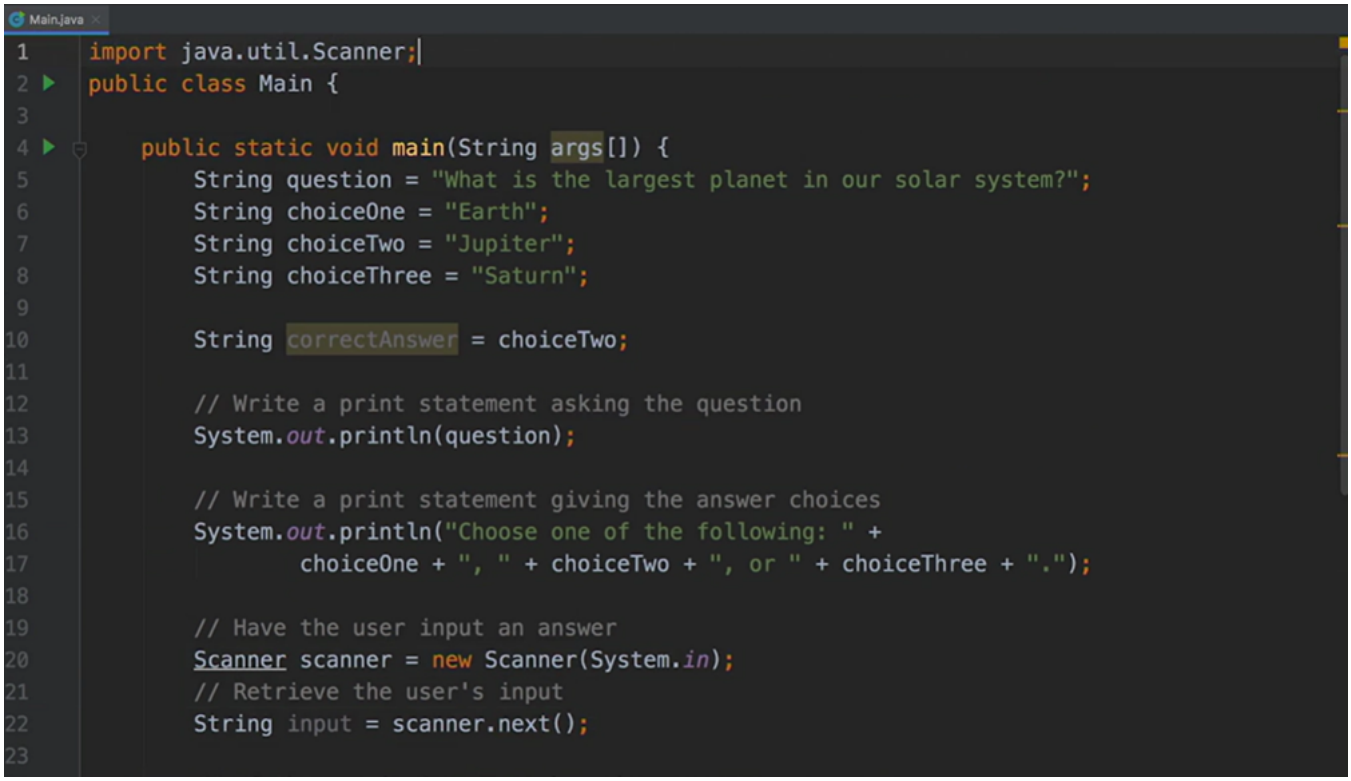
Then we'll give the user some answer choices.



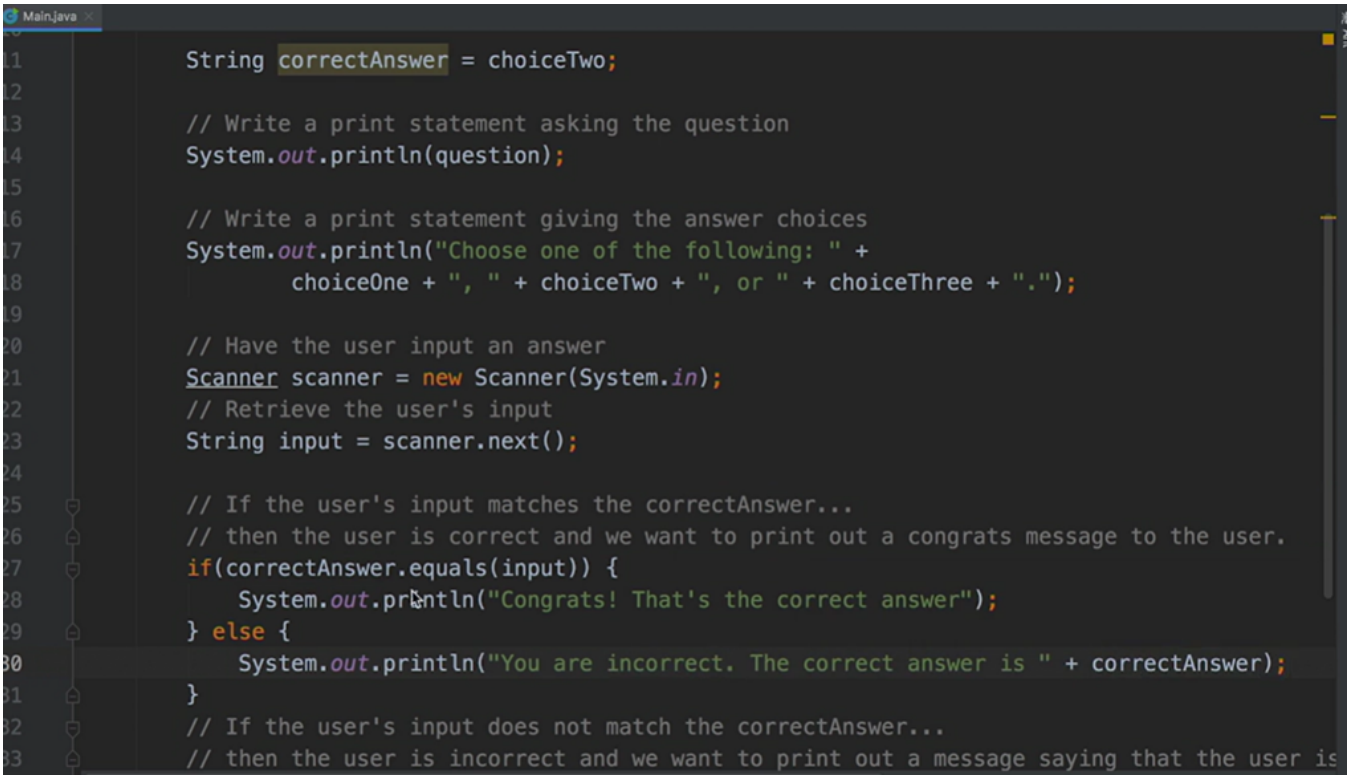
```
ublic class Main {

    public static void main(String args[]) {
        String question = "What is the largest planet in our solar system?";
        String choiceOne = "Earth";
        String choiceTwo = "Jupiter";
        String choiceThree = "Saturn";

        String correctAnswer = choiceTwo;

        // Write a print statement asking the question
        System.out.println(question);

        // Write a print statement giving the answer choices
        System.out.println("Choose one of the following: " )
```

To retrieve the users input we create a scanner using system.in and you use the .next operation.

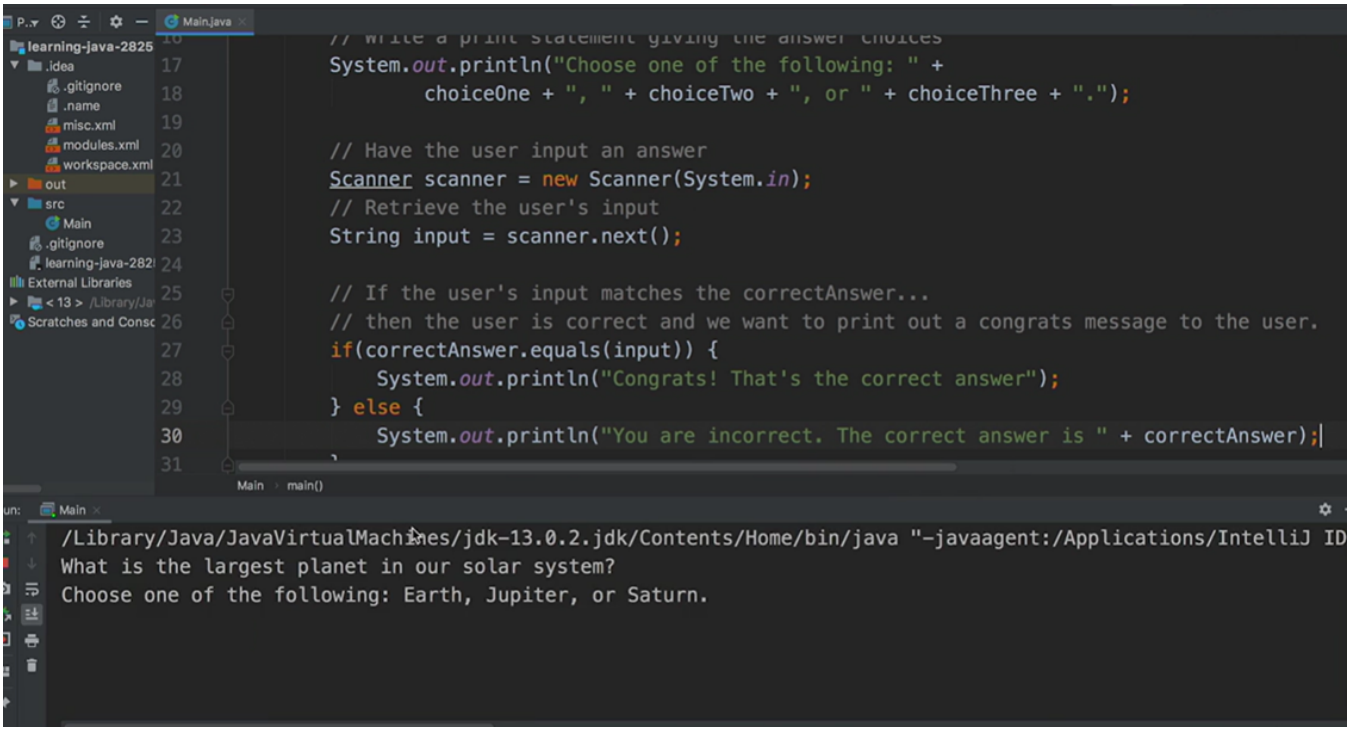




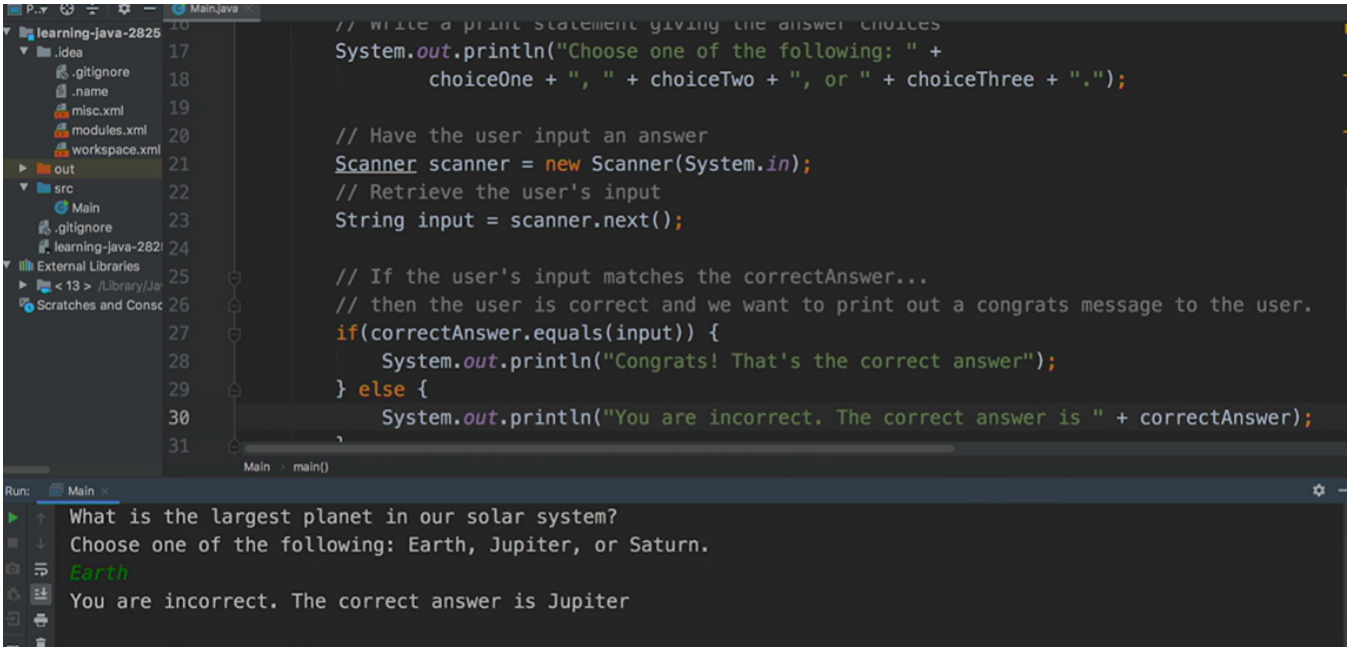
Now for the control flow portion. We can use an if statement to check if the users input matches the correct answer. If the condition is true we'll print out a congrats message. If the user is not correct then the user must have inputted something that is not equal to the value of the correct answer. We can add an else to our if statement so that a message stating the user is incorrect is printed in this case.

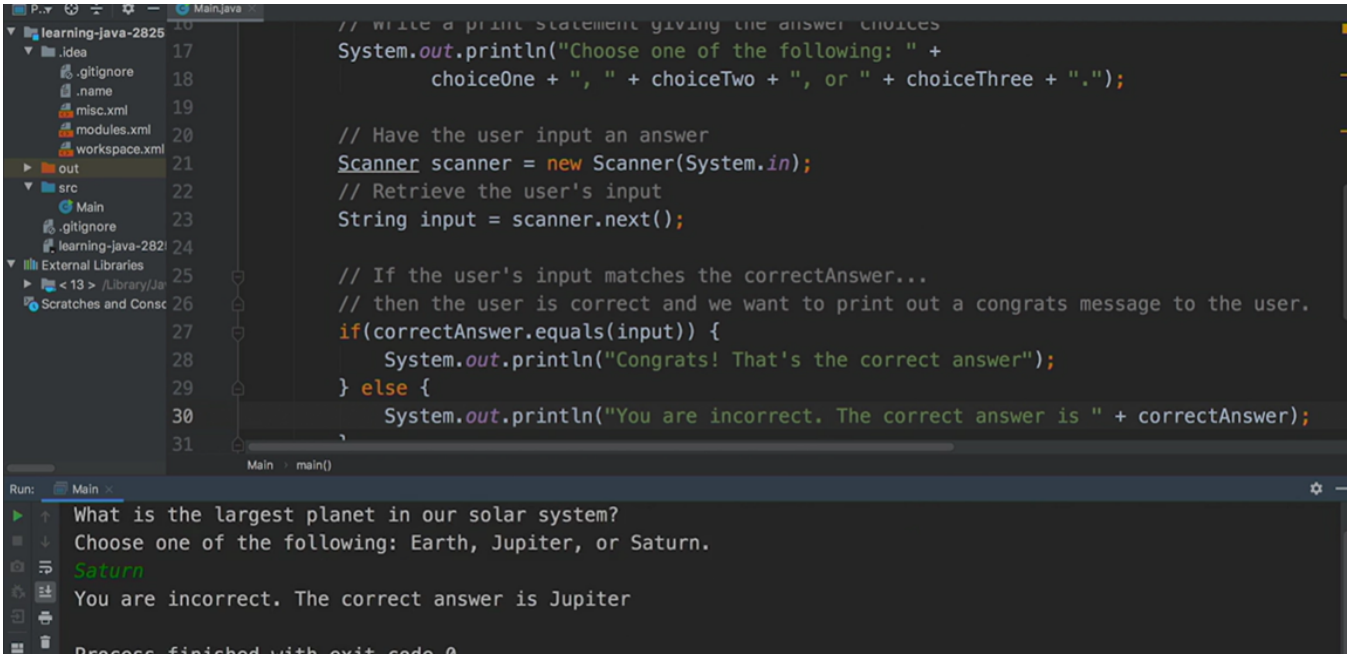


All right, let's try running the program. What is the largest planet in our solar system? Choose one of the following, Earth, Jupiter, or Saturn. Let's try Earth, you are incorrect.

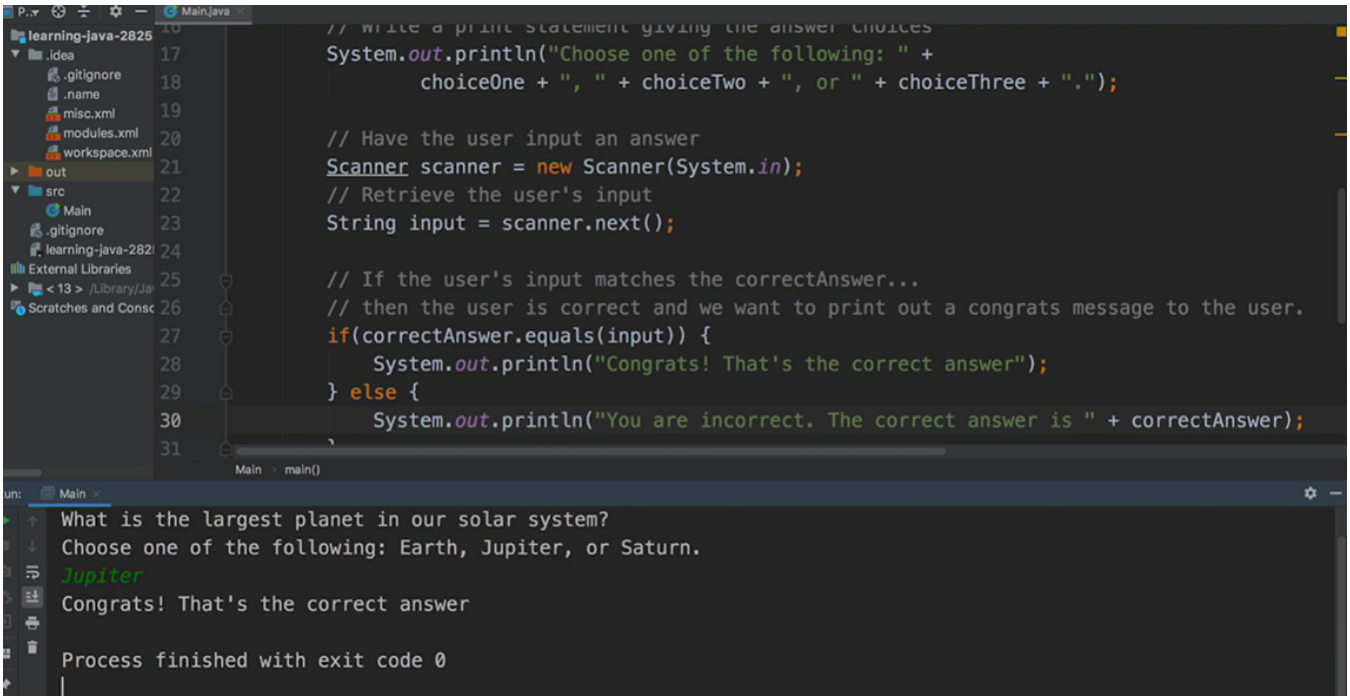


The correct answer is Jupiter. Let's try Saturn. That is also incorrect.

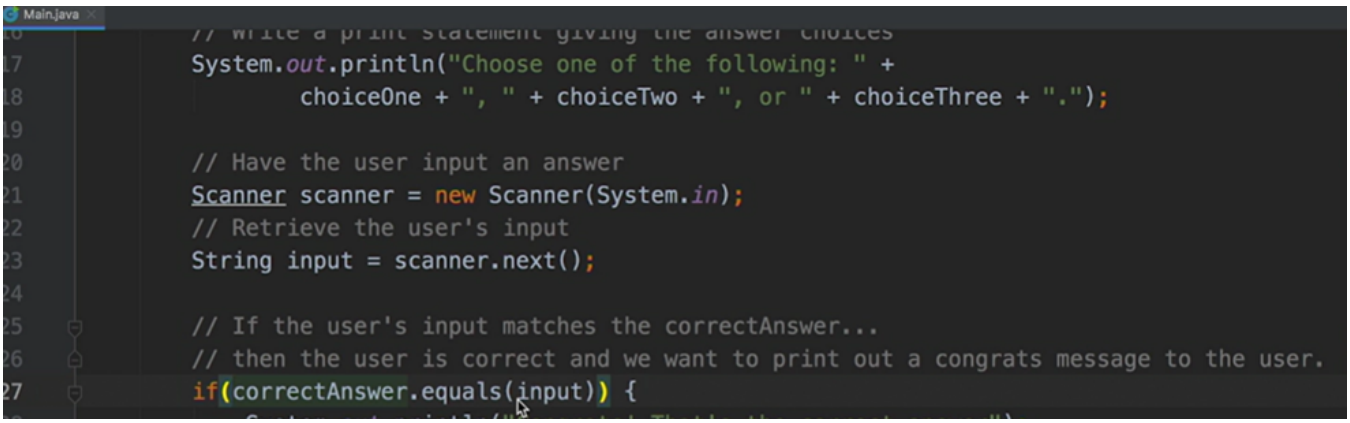


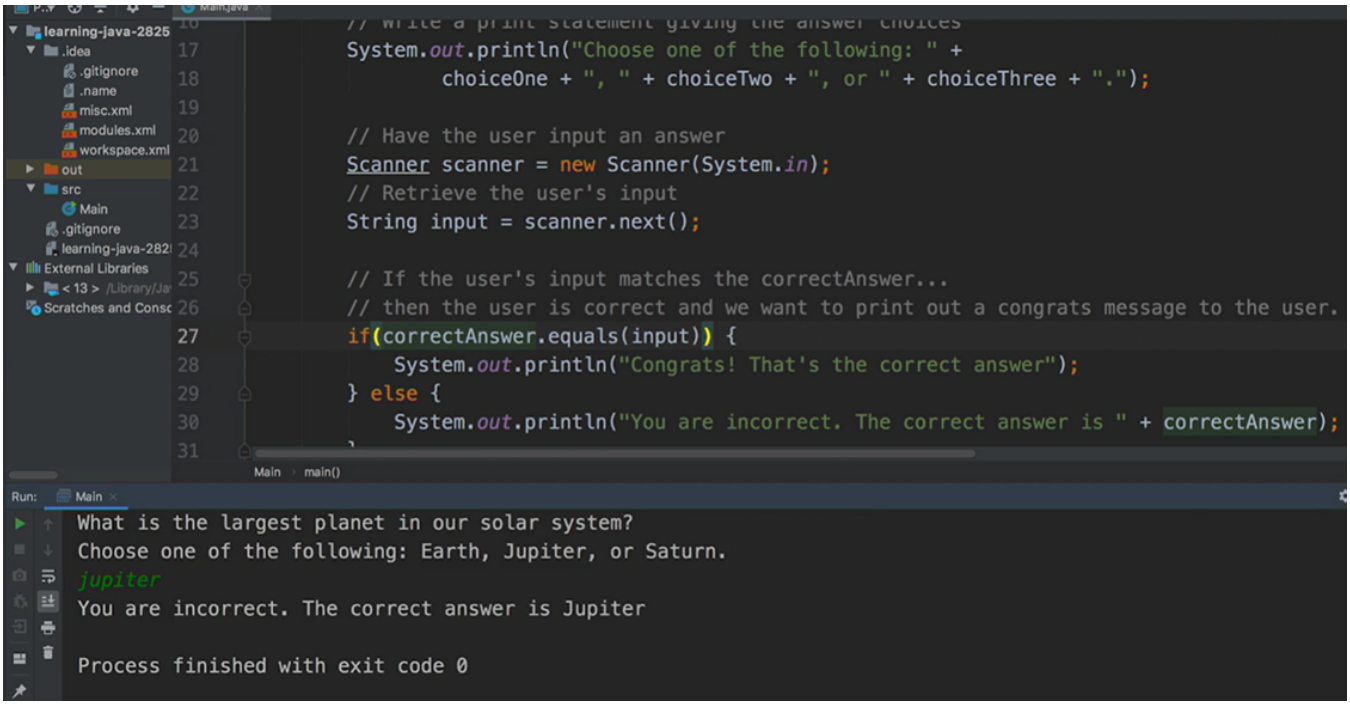


We'll try one more with Jupiter. That is the correct answer, congrats.



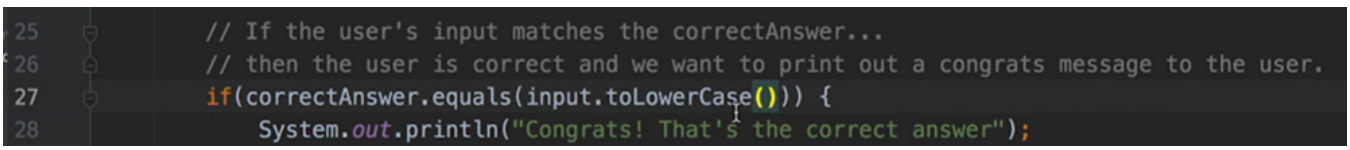
The if statement is the main reason that we got different outputs depending on our input. In the condition we check if the correct answer Jupiter equals whatever the user, we, have inputted. Only when our input equals Jupiter we get this correct answer message. Now what happens if I input Jupiter but with a lowercase J? We get that we're incorrect, but the correct answer is what we've stated. This happens because .equals is case-sensitive.





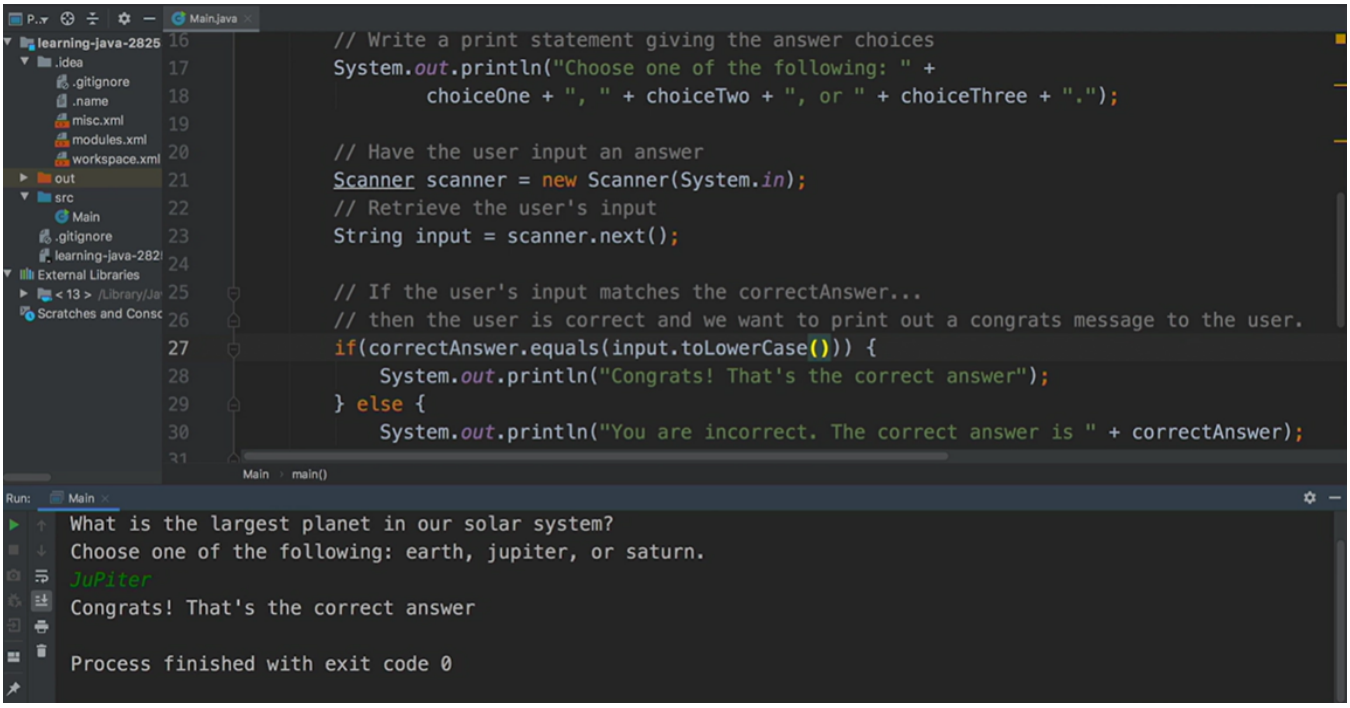
That means if the casing of our input does not equal the casing of correct answer this condition will be false. We don't want our condition to be case-sensitive. We can fix it in this case by changing all of our answer choices to having lowercase first letters making them fully lowercase. We run the program again, a full lowercase Jupiter will work but other versions of it will not work.

To make it work for the other versions we need to convert our input to lowercase. We can fix this by using a string operation called toLowerCase(). This will convert the users input to lowercase. This way it doesn't matter what caps we use to enter Jupiter. All of these versions will work.



Now how can we add onto this program? Let's say we wanted to add the ability for the user to try again and again until they get the right answer. To do this we would need to use a while loop. This would allow the user to keep guessing until their answer was correct. We could also add more multiple-choice questions. With what we know so far, adding additional questions would require us to add a significant amount of code. If you want to create a more efficient quiz with several multiple-choice questions, you'll need to learn about data structures, which is a more advanced topic that's not covered in this course. In working on this challenge, you may have run into some tough errors, and that's okay.

Final sample code



In the next iteration, we'll look at how we can debug our programs by finding some common coding errors and fixing them.

Demo

Now you are ready to schedule your 1x1 demo with the TA. You may show the TA screenshots of your work.

Next Steps

- Schedule a demo with a Topic Advisor (TA).
- Check your CELL group schedule in SharePoint
- Continue to collaborate with others and be active in MS Teams.
- 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
- Navigate to Iteration 3 and continue your CELL journey.

Previous Iteration	tWiki Home	Next Iteration
Iteration 1	tWiki Home Page	Iteration 3

Previous Iteration	Wiki Home	Forums	Next Iteration