Forming an Algorithm

by Sophia

WHAT'S COVERED

In this lesson, you will review various problems and find patterns to help build a solution to those problems, prior to writing code. Specifically, this lesson covers:

1. Introduction to Algorithms

When building a program to solve a problem, it is essential to understand the step-by-step process or processes for solving the problem. The logical step-by-step plan that is used to solve that problem is found in an **algorithm**. When designing an algorithm, it is important to consider different ways to solve a problem. This will help determine the best way to solve that problem. As you become more comfortable with a programming language, you will discover better ways to optimize a solution.

THINK ABOUT IT

Before writing an algorithm, it is important to think about the big picture for the algorithm. It should be written with the final goal of solving the problem in mind. Additionally, there are other aspects of the algorithm that need to be considered. These include consideration for how often this program is meant to run, the deadline for creating the program, and the complexity of the problem. For example, if a program is only meant to run once, it would likely be acceptable if it were not fully optimized. However, if a program is meant to run millions of times a day, the optimization of a solution is crucial. Think about a mobile app that you open up on a regular basis. Would you continue to use that app if it took five minutes for the app to even load? Most would probably not and look for an alternative solution.

Once the bigger picture is determined, the next step is to think about the individual stages and smaller steps. The problem will need to be broken down into these components to prepare for writing the algorithm. There are basic criteria that should be considered in every situation. These criteria involve items related to the programming mindset.

STEP BY STEP

1. Determine what inputs are involved in solving the problem and where they are coming from. This should provide a starting point to establishing the problem.

2. Determine what the outputs of the problem should be. More than just the individual items that we output, the focus should be on the end result that the program is designed to achieve. This focus could be a single line of output on the screen, a file, an email, or a whole report that is to be generated.

3. Determine what the order of the steps in the process should be. The order, as discussed, is important because it helps define the actions that need to occur in the order they need to occur.

4. Determine what types of decisions need to be made within the program. Think back to the prior tutorial. Sophie needed to bring different items depending on the day and the weather forecast. These types of decisions add a layer of choice and options in the process based on the input and other variables.

5. Consider if there are any parts of the problem that need to be, or could be, repeated.

IN CONTEXT
Think about a person going to an ATM to withdraw money. The person is presented with a menu of transaction options to choose from. Once a transaction is complete, the customer is presented with the menu again. The customer can complete as many transactions as needed before the program ends. Rather than having to exit out of the program completely after a single transaction, the menu is presented over and over again until a selection is made to exit out of the program.

TERM TO KNOW

Algorithm
A logical step-by-step plan that we need to build for solving a problem.

2. Computer Guessing Game

In the following example program, a simple guessing game allows the user to guess a random whole number between 1 and 100, inclusively. The user will be able to keep guessing until the correct number is found. After each guess, the program will tell the user if the guess is too high or too low. Once the user has guessed the number correctly, the program will tell the user how many times it took the user to guess, and exit the program.

Analyzing this program begins with defining the input elements for the program. In this case, the program requires a random number that is generated between 1 and 100. This can be referred to as `randomNumber`. It is also important to read and store the input from the user for the guess. This can be called `guess`. Lastly, a counter is required to track the number of guesses that the user needed to guess the number correctly. This variable can be referred to as `numberOfGuesses`.

STEP BY STEP

1. Determine what the output at the end of the program should be. This should be the `numberOfGuesses` printed to the screen once the user has guessed the number correctly.

2. Consider the presence of any conditions that should be considered, as these affect the flow of the steps. There are conditions in this program. The program needs to check if the guess is higher than the `randomNumber`. If it is, there is a need to inform the user. If the guess is lower than the `randomNumber`, the program should advise the user. If the guess is equal to the `randomNumber`, the program should end after outputting the appropriate message to the user.

3. Consider any repeating elements. In this case, one repeating element increases the value of `guessCount` after each entry. The program will also need to keep running the checks and output the appropriate response to the user until the guess is equal to the `randomNumber`.

4. Review what the code would look like. Do not try to decipher the code below at first. These functions will be discussed at a later point when discussing what functions do and how to write lines of code. For now, determine if the logic can be followed and if the comments are able to be read as identified by the hashtags.

HINT

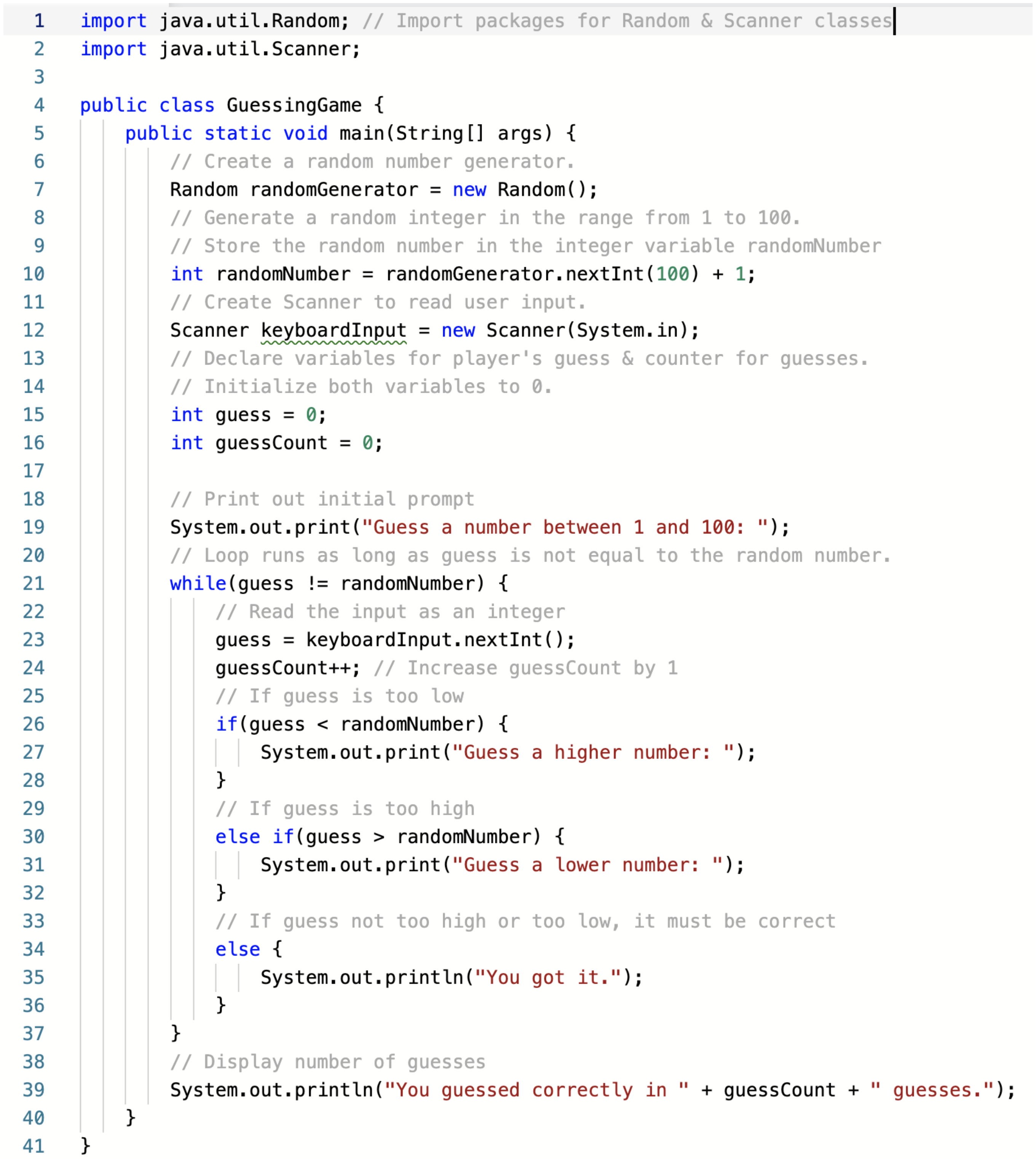
The `//` (double forward slash) symbols are used in Java code to identify a **comment**. Commented lines of code are ignored by the compiler and do not affect the behavior of the code. They are also hidden from the user when the program is run. A programmer can choose to add comments throughout the code to explain how the program or piece of code works. These hidden comments make it easier for the reader to identify the intention of that section of code. Keep in mind that some programs are hundreds of lines long or more, so commenting is always a best practice. We will talk more about commenting and its use in a later tutorial.

For this code example, **we will bold only the actual compiled code that is running**.

```
// Import packages for Random & Scanner classes
import java.util.Random;
import java.util.Scanner;

public class GuessingGame {

  public static void main(String[] args) {
    // Create a random number generator.
    Random randomGenerator = new Random();
```

```
        // Generate a random integer in the range from 1 to 100.
        // Store the random number in the integer variable randomNumber
        int randomNumber = randomGenerator.nextInt(100) + 1;
        // Create Scanner to read user input.
        Scanner keyboardInput = new Scanner(System.in);

        // Declare variables for player's guess & counter for guesses.
        // Initialize both variables to 0.
        int guess = 0;
        int guessCount = 0;

        // Print out initial prompt
        System.out.print("Guess a number between 1 and 100: ");
        // Loop runs as long as guess is not equal to the random number.
        while(guess != randomNumber) {
          // Read the input as an integer
          guess = keyboardInput.nextInt();
          guessCount++; // Increase guessCount by 1
          // If guess is too low
          if(guess < randomNumber) {
            System.out.print("Guess a higher number: ");
          }
          // If guess is too high
          else if(guess > randomNumber) {
            System.out.print("Guess a lower number: ");
          }
          // If guess not too high or too low, it must be correct
          else {
            System.out.println("You got it.");
          }
        }
        // Display number of guesses
        System.out.println("You guessed correctly in " + guessCount + " guesses.");
      }
    }
```

The following is a screenshot of the Guessing Game code in an editor:



```java
import java.util.Random; // Import packages for Random & Scanner classes
import java.util.Scanner;

public class GuessingGame {
    public static void main(String[] args) {
        // Create a random number generator.
        Random randomGenerator = new Random();
        // Generate a random integer in the range from 1 to 100.
        // Store the random number in the integer variable randomNumber
        int randomNumber = randomGenerator.nextInt(100) + 1;
        // Create Scanner to read user input.
        Scanner keyboardInput = new Scanner(System.in);
        // Declare variables for player's guess & counter for guesses.
        // Initialize both variables to 0.
        int guess = 0;
        int guessCount = 0;

        // Print out initial prompt
        System.out.print("Guess a number between 1 and 100: ");
        // Loop runs as long as guess is not equal to the random number.
        while(guess != randomNumber) {
            // Read the input as an integer
            guess = keyboardInput.nextInt();
            guessCount++; // Increase guessCount by 1
            // If guess is too low
            if(guess < randomNumber) {
                System.out.print("Guess a higher number: ");
            }
            // If guess is too high
            else if(guess > randomNumber) {
                System.out.print("Guess a lower number: ");
            }
            // If guess not too high or too low, it must be correct
            else {
                System.out.println("You got it.");
            }
        }
        // Display number of guesses
        System.out.println("You guessed correctly in " + guessCount + " guesses.");
    }
}
```

HINT

The course uses a browser-based code editor tool, an IDE, to write and run your code. You will learn more about the IDE in the next lesson.

IN CONTEXT

Wraparounds are unavoidable on some lines due to your personal device's display. This should not cause an issue when copy/paste is used to enter the code samples into the IDE or other code editors. However, please don't let this deter you from manually typing in the code! There is one issue in the previous code listing that would be good to avoid while typing out the code. While a line of code in Java can be split before or after a plus sign, a line can't legally be split after an opening quotation mark. Breaking long lines to the right of a plus sign is a fairly straightforward rule, but other line breaks may require consultation. While you are learning, it may work best to keep comment lines short and break them early when in doubt. Consider the following:

```
    // Display number of guesses
    System.out.println("You guessed correctly in " + guessCount + "
guesses.");
    }
```

The line that starts with System.out.println needs to split after the + but before the ":

```
    // Display number of guesses
    System.out.println("You guessed correctly in " + guessCount +
" guesses.");
    }
```

TERM TO KNOW

Comments
The // (double forward slash) is used in Java code to identify a comment. Commented lines of code are ignored by the compiler during run time. A programmer adds comments throughout the code to explain how the program or function works.

3. Algorithm to Guess the Guessing Game

DID YOU KNOW

There is an algorithm that can be used to guess a specific number in 7 guesses or less each time. A user could get lucky on a guess as well, but in this algorithm, it will be consistent.

Let's take a look at the steps involved in planning this program.

STEP BY STEP

1. On the first guess, a user will want to choose the middle number, which is 50. This will eliminate 50% of all of the possible numbers, as it will either be higher or lower than that amount. If it is the correct value, great, but a user had a 1 in 100 chance on that number.

2. Depending on if it is higher or lower, a user will select the middle value between that and the end or beginning of the range. So, if 50 is too low, a user guesses 75 since that's the value midway between 50 and 100. By doing so, the user is again eliminating 50% of all of the numbers that are left.

3. The user will keep doing that each time, and with each guess, the user eliminates 50% of all of the remaining numbers until the user has the final number.

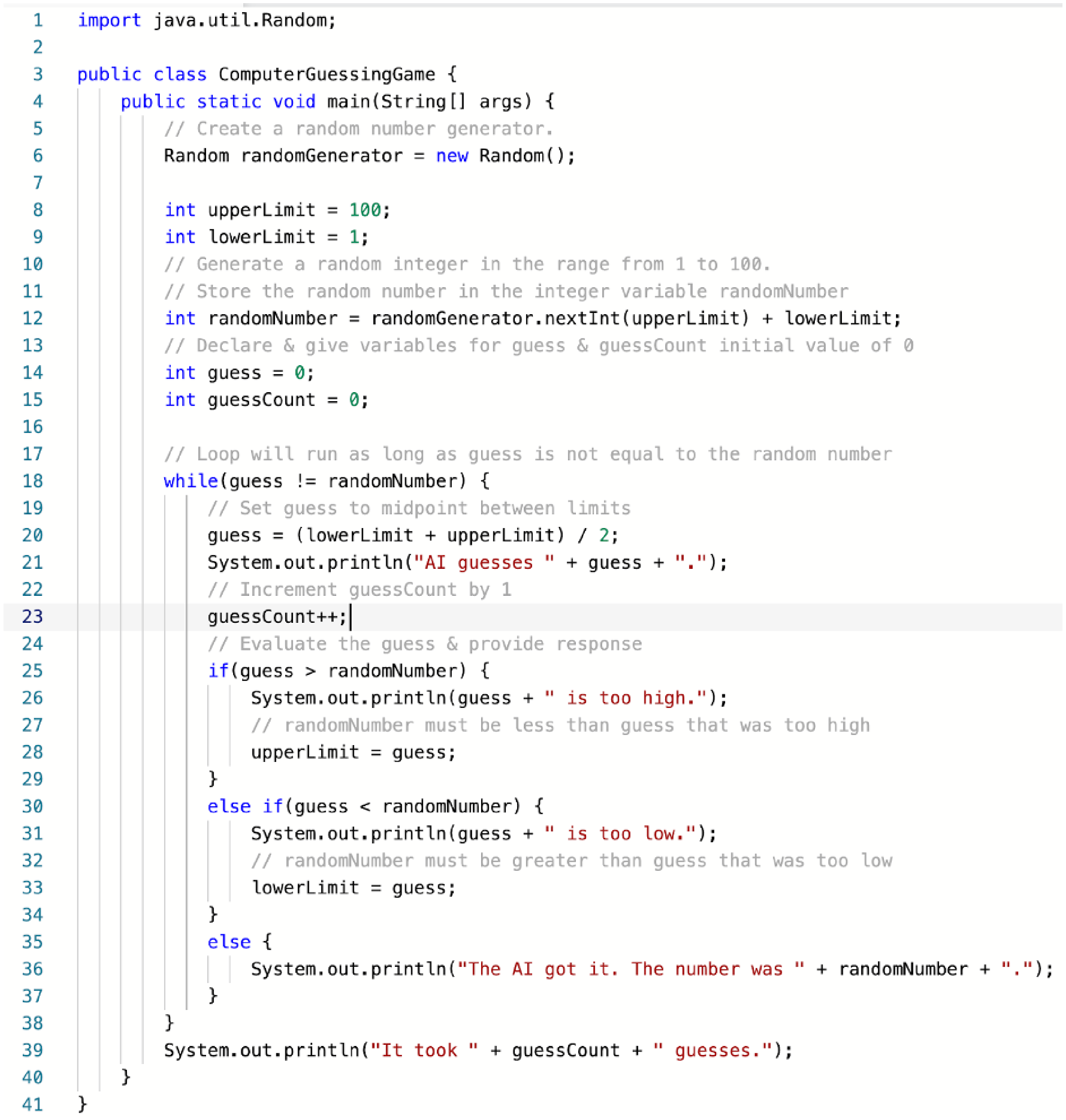
Here's a sample run demonstrating this algorithm that can be tracked:

```
Guess a number between 1 and 100: 50
Guess a higher number: 75
Guess a lower number: 63
Guess a lower number: 56
Guess a higher number: 59
Guess a lower number: 58
Guess a lower number: 57
You got it.
You guessed correctly in 7 guesses.
```

To provide a clearer picture of how the formula or process for guessing the number works, let's remove the "complication" of human guesses and have the computer itself apply the algorithm. The Java program below includes code that carries out the steps in the algorithm using relatively simple arithmetic. The computer will reliably perform the same steps each time to solve the problem without regard to the number of "guesses" it is making or a sense of having won or lost. Any "intelligence" or "cleverness" is the result of applying the simple steps of narrowing down the range of numbers for each guess.

```
import java.util.Random;

public class ComputerGuessingGame {

    public static void main(String[] args) {
        // Create a random number generator.
        Random randomGenerator = new Random();

        int upperLimit = 100;
        int lowerLimit = 1;
        // Generate a random integer in the range from 1 to 100.
        // Store the random number in the integer variable randomNumber
        int randomNumber = randomGenerator.nextInt(upperLimit) + lowerLimit;
        // Declare & give variables for guess & guessCount initial value of 0
        int guess = 0;
        int guessCount = 0;
        // Loop will run as long as guess is not equal to the random number
        while(guess != randomNumber) {
            // Set guess to midpoint between limits
            guess = (lowerLimit + upperLimit) / 2;
            System.out.println("AI guesses " + guess + ".");
            // Increment guessCount by 1
            guessCount++;

            // Evaluate the guess & provide response
            if(guess > randomNumber) {
                System.out.println(guess + " is too high.");
                // randomNumber must be less than guess that was too high
                upperLimit = guess;
            }
            else if(guess < randomNumber) {
                System.out.println(guess + " is too low.");
                // randomNumber must be greater than guess that was too low
                lowerLimit = guess;
             }
            else {
                // Need to print out randomNumber in response so user can see what
                // it was
                System.out.println("The AI got it. The number was " + randomNumber + ".");
            }
        }
        System.out.println("It took " + guessCount + " guesses.");
    }
}
```

The following is a screenshot of the ComputerGuessingGame code with AI algorithm added in as editor.



```java
import java.util.Random;

public class ComputerGuessingGame {
    public static void main(String[] args) {
        // Create a random number generator.
        Random randomGenerator = new Random();

        int upperLimit = 100;
        int lowerLimit = 1;
        // Generate a random integer in the range from 1 to 100.
        // Store the random number in the integer variable randomNumber
        int randomNumber = randomGenerator.nextInt(upperLimit) + lowerLimit;
        // Declare & give variables for guess & guessCount initial value of 0
        int guess = 0;
        int guessCount = 0;

        // Loop will run as long as guess is not equal to the random number
        while(guess != randomNumber) {
            // Set guess to midpoint between limits
            guess = (lowerLimit + upperLimit) / 2;
            System.out.println("AI guesses " + guess + ".");
            // Increment guessCount by 1
            guessCount++;
            // Evaluate the guess & provide response
            if(guess > randomNumber) {
                System.out.println(guess + " is too high.");
                // randomNumber must be less than guess that was too high
                upperLimit = guess;
            }
            else if(guess < randomNumber) {
                System.out.println(guess + " is too low.");
                // randomNumber must be greater than guess that was too low
                lowerLimit = guess;
            }
            else {
                System.out.println("The AI got it. The number was " + randomNumber + ".");
            }
        }
        System.out.println("It took " + guessCount + " guesses.");
    }
}
```

EXAMPLE

Here is a sample of the results when running the program after 7 guesses:

```
AI guesses 50.
50 is too low.
AI guesses 75.
75 is too low.
AI guesses 87.
87 is too low.
```

```
AI guesses 93.
93 is too low.
AI guesses 96.
96 is too low.
AI guesses 98.
98 is too low.
AI guesses 99.
The AI got it. The number was 99.
It took 7 guesses.
```

EXAMPLE

Here are the results when running the ComputerGuessingGame’s input and output after 4 guesses.:

```
AI guesses 50.
50 is too high.
AI guesses 25.
25 is too low.
AI guesses 37.
37 is too low.
AI guesses 43.
The AI got it. The number was 43.
It took 4 guesses.
```

EXAMPLE

Here are the results when running the ComputerGuessingGames input and output after 2 guesses:

```
AI guesses 50.
50 is too low.
AI guesses 75.
The AI got it. The number was 75.
It took 2 guesses.
```

REFLECT

While there were some lucky guesses along the way, the algorithm did function as expected.

SUMMARY

In this lesson, you were **introduced to algorithms** as a logical step-by-step plan that is built to create a solution to a problem. You also reviewed conditional and repeatable elements in the sample **Computer Guessing Game** code. Finally, you learned how to use basic guessing games and enhance them by using an algorithm (AI) that can **guess the guessing game** by producing consistent tries. Finally, you

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

learned that it is not the underlying code that matters at this point. The logical steps behind the algorithm are most important at this point.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source py4e.com/html3/

TERMS TO KNOW

Algorithm
A logical step-by-step plan that we need to build for solving a problem.

Comments
The // (double forward slash) is used in Java code to identify a comment. Commented lines of code are ignored by the compiler during run time. A programmer adds comments throughout the code to explain how the program or function works.

© 2024 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.