Translating the Code

by Sophia

WHAT'S COVERED

In this lesson, you will learn about the initial steps to translate pseudocode to code. Specifically, this lesson covers:

1. Coding Framework

Looking back at the pseudocode that you created for the demonstration program, you first defined the output the program should produce. Now you will need to think about the key pieces of functionality. The overall game is made up of one or more dice rolls. The DiceRoll class represents a roll of a pair of dice. It needs to generate two random integers in the range from 1 to 6. In Java (like many other programming languages), this involves using a random number generator. This requires a bit of setup which we will demonstrate below. The class needs to keep track of the values of the two dice and also needs to report the sum of the dice. A comment outline for the DiceRoll class might look like this:

EXAMPLE

```
// Attribute to hold the rolls of the 2 dice
// Attribute to hold the sum of the 2 dice
// Attribute for the random number generator from the Random class
```

As previously noted, you would need a single roll of a pair of dice, and the values will not change, nor will the sum of the dice change. This means that you can use the `DiceRoll()` constructor to generate the numbers and get the sum. An outline for the constructor using comments might look like this:

EXAMPLE

```
// Create the random number generator
// Get the random number for the 1st die and store it
// Get the random number for the 2nd die and store it
// Add the values and store
```

Using these comments as the scaffolding, the DiceRoll class ends up like this (with a couple of additional comments to document the code) :

```
import java.util.Random;  // Needed for Random

public class DiceRoll {
  // The values of the 2 dice (an array of int)
  private final int[] die = new int[2];
  // Attribute to hold the sum of the dice
  private int sum;
  // Attribute for random number generator instance
  private Random randomNumberGenerator;

  public DiceRoll() {
    // Create the random number generator
    randomNumberGenerator = new Random();
    // Get the random number for 1st die in range from 1 to 6 & store in array
    die[0] = randomNumberGenerator.nextInt(6) + 1;
    // Get the random number for the 2nd die in range from 1 to 6 & store in array
    die[1] = randomNumberGenerator.nextInt(6) + 1;
    // Add values of dice and store
    sum = die[0] + die[1];
  }

  // Accessor method to get sum of dice
  public int getSum() { return sum; }

  // String representation of the dice roll
  public String toString() {
    return "Dice: " + die[0] + "  " + die[1] + "  Sum = " + sum;
  }
}
```

TRY IT

Directions: Try adding these comments in a file called DiceRoll.java. Then add the lines of code that implement the steps outlined in the comments. The DiceRoll class won't run on its own, so compile it using javac DiceRoll.java first.

REFLECT

This will reveal if there are syntax errors or other problems that prevent it from building. Remember that if javac is successful, there won't be any messages in the Shell.

The `Game` class represents the results from a collection of dice rolls that result in a win or loss for the user. The `Game` class also keeps track of the player's name and the date and time at the start of the game.

Here is a list of comments corresponding to the attributes needed in this class:

EXAMPLE

```
// Player's name
// The date & time
// Collection for dice rolls
// Value of "point" if 1st roll is not a win or loss
// The result of the game ("Win" or "Lose")
```

Since it is best to have the code in the driver class handle the input and output, the player's name will be passed to the `Game()` constructor as a parameter.

The comment "outline" of the `Game()` constructor might look like this:

EXAMPLE

```
// Assign player's name to the attribute for the player's name
// Set the attribute for the date/time to the current date/time
// Create a collection to hold the DiceRoll objects that make up the game
```

The central action in the Game class will go in the `play()` method.

A set of notes about the process for this method is captured by comments like these:

EXAMPLE

```
// Call the constructor for the DiceRoll class to get a roll of the dice
// Store the roll in the collection of rolls for the game
// Check the sum of the roll
  // If the sum is 2, 3, or 12, the result is set to "Lose"
    // Set value so that rolls do not continue
  // If the sum is 7 or 11, the result is set to "Win"
    // Set value so that rolls do not continue
  // Other sum for the roll is the value of the "point"
```

```
      // Keep rolling the dice
      // If sum of roll is 7, set status to "Lose"
         Set value so that rolls do not continue
      // If sum roll is point, set status to "Win"
         Set value so that rolls do not continue

   // When the play() method is done, it returns the collection of DiceRoll objects

   // The Game class will need appropriate accessor ("getter") methods:
   // get the game result ("Win" or "Lose")
   // get the number of dice rolls for the game
   // get the name of the player
   // get the date/time of the game
```

Using these comments as a framework, here is the implementation of the `Game` class. Since the collection of dice rolls in a game will vary and may contain duplicates, the code uses an `ArrayList<DiceRoll>` to hold the rolls of the dice.

```
import java.time.LocalDateTime;  // Needed for date & time
import java.util.ArrayList;  // Needed for ArrayList to hold

public class Game {
  private String playerName = "";
  private LocalDateTime gameDateTime;
  // ArrayList to hold dice rolls for player
  private ArrayList<DiceRoll> diceRolls;

  private int point = 0;
  private boolean keepRolling = true;
  // String to hold game result msg: "Win" or "Lose"
  private String gameResult = "";

  public Game(String playerName) {
    gameDateTime = LocalDateTime.now();
    this.playerName = playerName;
    diceRolls = new ArrayList<DiceRoll>();
  }

  public ArrayList<DiceRoll> play() {
    // 1st roll for player
    DiceRoll roll = new DiceRoll();
    diceRolls.add(roll);
    int sum = roll.getSum();

    // Check for "craps" & resulting loss
    if(sum == 2 || sum == 3 || sum == 12) {
      gameResult = "Lose";
      keepRolling = false;
    }

    // Check for immediate win
    else if(sum == 7 || sum == 11) {
      gameResult = "Win";
      keepRolling = false;
    }
    else {
      // Sum of dice is now player's "point"
      point = roll.getSum();
    }

    // Loop for subsequent rolls
    while(keepRolling) {
      roll = new DiceRoll();
      diceRolls.add(roll);
      if(roll.getSum() == 7) {
        gameResult = "Lose";
        keepRolling = false;
      }
      else if(roll.getSum() == point) {
        gameResult = "Win";
        keepRolling = false;
      }
    }
    return diceRolls;
  }

  public String getResult() {
    return gameResult;
  }

  // Get the number of dice rolls for game
  public int getRollCount() {
    // Number of DiceRoll objects in the ArrayList is number of rolls
    return diceRolls.size();
  }
  public String getPlayerName() {
    return playerName;
  }

  public LocalDateTime getGameDateTime() {
    return gameDateTime;
  }
}
```

TRY IT

Directions: Add the comments shown above to a file named Game.java. Then add the lines of code that implement the steps outlined by the comments. As you might expect, the Game class won't run on its own. You will need to compile it using javac Game.java first.

REFLECT

Running the program will reveal if there are syntax errors or other problems that prevent it from building. Remember that if javac is successful, there won't be any messages in the Shell.

With the `Game` class (composed of `DiceRoll` objects) in place, we can outline the process that the code in the application's `main()` will work through. The driver class is the `Craps` class, since it represents a craps session for a player that is made up of one or more games. This class will need to track statistics, handle user input, provide output, and do the logging to a file. Here are some comments to note the key variables needed in the `main()` method. These are variables in the body of the main() method, not class-level attributes:

EXAMPLE

```
// win count
// loss count
// roll count
// log file
// Scanner for reading user input
// player's name
// Game object
// ArrayList for rolls in game
```

Now, here are the key steps in the process that the code in main() will work though:

EXAMPLE

```
// Create log file
// Declare Scanner to read from System.in
// Prompt for input of player name
// Read the player's name and store it

// Run the sample game
    // Create Game object
    // Call the play() method and store result in ArrayList
    // Loop through the ArrayList to print out dice rolls in the game
    // Print the result (Win/Lose) for sample game

// Prompt for number of games player wants to play
// Read input as integer
// Track if number entered is valid
    // Repeat prompt/input if entry is not valid

// Loop to play requested number of games
    // Create a new Game object
    // Call the play() method & store result in ArrayList
    // Loop through ArrayList to print out dice rolls
    // If getResult() returns "Win" add 1 to win count
    // Else add 1 to loss count
    // Format date and time for output
    // Assemble String with statistics
    // Calculate percentage of wins by dividing win count by number of games played
    // Print statistics data to Shell
    // Write statistics data to log file
```

TRY IT

Directions: Add the comments above from the demonstration program and follow along with the writing of the code.

Next, it can be helpful to identify the variables that we are planning to use in our program and start to define them in Java.

KEY CONCEPT

Generally, it's a good idea to initialize numeric values to 0 or its intended value and strings to an empty string or their intended values. This way, we won't be surprised by incorrect values being displayed.

TRY IT

Directions: Start by coding with the `main()` method first:

```
int winCount = 0;
int lossCount = 0;
```

```
// Total number of dice rolls for game
int rollCount = 0;

// log file
File logFile = new File("game.log.txt");
// Scanner for reading user input
Scanner input = new Scanner(System.in);
// String variable for player name
String playerName  = ""; // Later gets value from input.nextLine();
```

TRY IT

Directions: Then, start working on the other parts of the program:

```
// Prompt for user name
// Read the input as a String
// Play sample game so player can see how it works
// Display result from the sample game
// Prompt for number of games player wants to play
//  Read input as integer
// Track if number entered is valid
    // Repeat prompt/input if entry is not valid

// Loop to play requested number of games
    // Create a new Game object
    // Call the play() method & store result in ArrayList
    // Loop through ArrayList to print out dice rolls
    // If getResult() returns "Win" add 1 to win count
    // Else add 1 to loss count
    // Format date and time for output
    // Assemble String with statistics
    // Calculate percentage of wins by dividing win count by number of games played
    // Print statistics data to Shell
    // Write statistics data to log file
```

REFLECT

Notice we have a method called `play()`. This method is provided by the `Game` class, so the code in `main()` will need to instantiate a `Game` object to use to call this method. While this step is specific to an object-oriented language and may not always be reflected in pseudocode, the pseudocode above does reflect this step.

TRY IT

Directions: Next, let's code the first input that we need the user to enter.

EXAMPLE

```
// Prompt for user name
System.out.print("Enter Player Name: ");
// Read the player's name and store it
playerName = input.nextLine();
```

The next step in the coding process is to play the sample game by constructing a Game object, calling its play() method, loop through the ArrayList that play() returns to show the dice rolls, and then display the result. This ends up being a bit more complex than the initial pseudocode indicates, but it's not uncommon to find that more steps (or smaller steps) are needed as the code gets written.

EXAMPLE

```
// Run sample game
System.out.println("\nRunning Sample Game: ");
Game game = new Game(playerName);
// ArrayList to track roles of dice in a game
ArrayList rolls = game.play();
// Loop through DiceRoll objects in ArrayList & display
for(DiceRoll roll : rolls) {
   // println() will automatically call DiceRoll object's toString() method
   System.out.println("\t" + roll);
}
// Call Game object's accessor methods to get result (win/lose) and # of rolls for game
System.out.println("\nResult of Sample Game: " + game.getResult() + " in " +
      game.getRollCount() + " roll(s)\n");
// Add an extra blank line in output for spacing
System.out.println();
```

Just as some steps in the pseudocode may break down into multiple steps when the code is written, it is also possible that not all comments will translate into separate statements. The pseudocode and comments derived from the pseudocode are meant to be guides and not to dictate the code exactly.

You will see the rest of the code for the main() method in the next lesson. It is possible, though, to take the code for main() written so far and test it.

TRY IT

Directions: Type the following code into a file named Craps.java. Remember that the DiceRoll and Game classes need to be compiled using javac first in order to use those classes in this code. Don't forget to add the import statements for the different library classes.

```java
import java.io.File;
import java.util.Scanner;
import java.util.ArrayList;

public class Craps {
  public static void main(String[] args) {
    int winCount = 0;
    int lossCount = 0;
    // Total number of dice rolls for game
    int rollCount = 0;

    // log file
    File logFile = new File("game.log.txt");

    // Scanner for reading user inputCra
    Scanner input = new Scanner(System.in);
    // String variable for player name
    String playerName  = ""; // Later gets value from input.nextLine();
    // Prompt for user name
    System.out.print("Enter Player Name: ");
    // Read the player's name and store it
    playerName = input.nextLine();

    // Run sample game
    System.out.println("\nRunning Sample Game: ");
    Game game = new Game(playerName);
    // ArrayList to track roles of dice in a game
    ArrayList<DiceRoll> rolls = game.play();
    // Loop through DiceRoll objects in ArrayList & display
    for(DiceRoll roll : rolls) {
      // println() will automatically call DiceRoll object's toString() method
      System.out.println("\t" + roll);
    }
    // Call Game object's accessor methods to get result (win/lose) and # of rolls for game
    System.out.println("\nResult of Sample Game: " + game.getResult() + " in " +
           game.getRollCount() + " roll(s)\n");
    // Add an extra blank line in output for spacing
    System.out.println();
  }
}
```
The result of running this code should look something like this (though of course, the rolls will be random and vary in number):

```
~/.../main/java$ javac DiceRoll.java
~/.../main/java$ javac Game.java
~/.../main/java$ java Craps.java
Enter Player Name: Sophia

Running Sample Game:
    Dice: 6  2  Sum = 8
    Dice: 3  2  Sum = 5
    Dice: 5  1  Sum = 6
    Dice: 4  2  Sum = 6
    Dice: 3  1  Sum = 4
    Dice: 4  5  Sum = 9
    Dice: 6  6  Sum = 12
    Dice: 3  5  Sum = 8

Result of Sample Game: Win in 8 roll(s)

~/.../main/java$
```

Core functionality will be expanded on in the upcoming lesson.

---

2. Guided Brainstorming

The goal of these initial steps is for us to understand how to break down our pseudocode into smaller pieces so that you can implement them part by part. It’s a good idea to identify all of the variables needed first. As that is completed, you can then move on to breaking down any functions or methods that we have. In this step, you would use the `pass` statement to create a function or method without fully implementing the functions.

As you start to write our code, begin to remove comments for logic that you have fully implemented. Also, consider the bigger picture. Beyond the pure logic of the program, think about the program that you plan to build.

Remember the Drink Order program? You could describe its process like the following pseudocode.

EXAMPLE

```
// Ask if user wants 1) water, 2) coffee, or 3) tea
// If drink choice is 1 (water)
    // Add "water" to output string
    // Ask to enter 1) hot or 2) cold
    // If hot, add "hot" to output string
    // else if cold, add "cold" to the output string
```

```
        // Ask if user would like ice
        // If response is 'Y' or 'y', add "with ice" to output string
        // Treat any other response as no - no further action
// Else if choice is 2 (coffee)
    // Add "coffee" to the output string
    // Ask if user would like decaf (Y/N)
    // If 'Y' or 'y', add "decaf" to output string
    // Treat other input as 'N' - do nothing
    // Ask if user would like 1) milk, 2) cream, or 3) none
    // If choice is 1 (milk), add "milk" to the output string
    // else if choice is 2 (cream), add "cream" to output string
    // Else, any other choice is ignored
    // Ask if user would like sugar (Y/N)
    // If response is 'Y' or 'y', add "sugar" to output string
    // Else, do nothing
// Else if choice is 3 (tea)
    // Add "tea" to output string
    // Ask user about tea type: 1) black, 2) green
    // If tea type is 1, add "black" to output string
    // Else if tea type is 2, add "green" to output string
    // Else treat any other response as black & add "black" to output string
// Else not a valid drink selection - print message
// Print out final drink choice with options
```

Here is the pseudocode translated into a series of comments using a double slash (//) and breaking some steps down a bit more:

EXAMPLE

```
// Ask if user wants 1) water, 2) coffee, or 3) tea
// If drink choice is 1 (water)
    // Add "water" to output string
    // Ask to enter 1) hot or 2) cold
    // If hot,
        // add "hot" to output string
    // else if cold,
        // add "cold" to the output string
        // Ask if user would like ice
        // If response is 'Y' or 'y',
            // add "with ice" to output string
        // Treat any other response as no - no further action
// Else if choice is 2 (coffee)
```

```
    // Add "coffee" to the output string
    // Ask if user would like decaf (Y/N)
    // If 'Y' or 'y',
        // add "decaf" to output string
    // Treat other input as 'N' - do nothing
    // Ask if user would like 1) milk, 2) cream, or 3) none
    // If choice is 1 (milk),
        // add "milk" to the output string
    // else if choice is 2 (cream),
        // add "cream" to output string
    // Else, any other choice is ignored
    // Ask if user would like sugar (Y/N)
    // If response is 'Y' or 'y',
        // add "sugar" to output string
    // Else, do nothing
// Else if choice is 3 (tea)
    // Add "tea" to output string
    // Ask user about tea type: 1) black, 2) green
    // If tea type is 1,
        // add "black" to output string
    // Else if tea type is 2,
        // add "green" to output string
    // Else treat any other response as black
        // add "black" to output string
// Else not a valid drink selection - print message
// Print out final drink choice with options
```

For the Drink Order program, we'll implement the functionality that's in the pseudocode in the next lesson.

TRY IT

Directions: This lesson is meant to just prep your program for conversion to code. It always helps to just comment your pseudocode from the start and start to fill in sections of the code that you're comfortable with in the same steps that we took. Now is a good time to convert your pseudocode into comments for your project.

SUMMARY

In this lesson, you examined the **coding framework**, as you started to translate the pseudocode into code. This is a great starting point since the lines of pseudocode can act as both our foundation of what to code as well as any starting comments. You started to code our demonstration program by identifying the variables that we are planning to use in the program. You also coded some initial functions and

expected output. Then, in the **Guided Brainstorming** section, you converted the pseudocode of the drink order program into comments to set that example up as well.

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/