Testing as You Go

by Sophia

WHAT'S COVERED

In this lesson, you will learn about testing as we progress. Specifically, this lesson covers:

1. Checking for Errors

Errors, bugs, and exceptions are always there in any program that we develop. These errors and issues can cause all types of problems with the program. As you work through the code, it's a good idea to test the program as much as you can. You will want to try to do everything that you can to break it and then find ways to help protect it from breaking in the future.

EXAMPLE Some examples of things we should test and debug are:

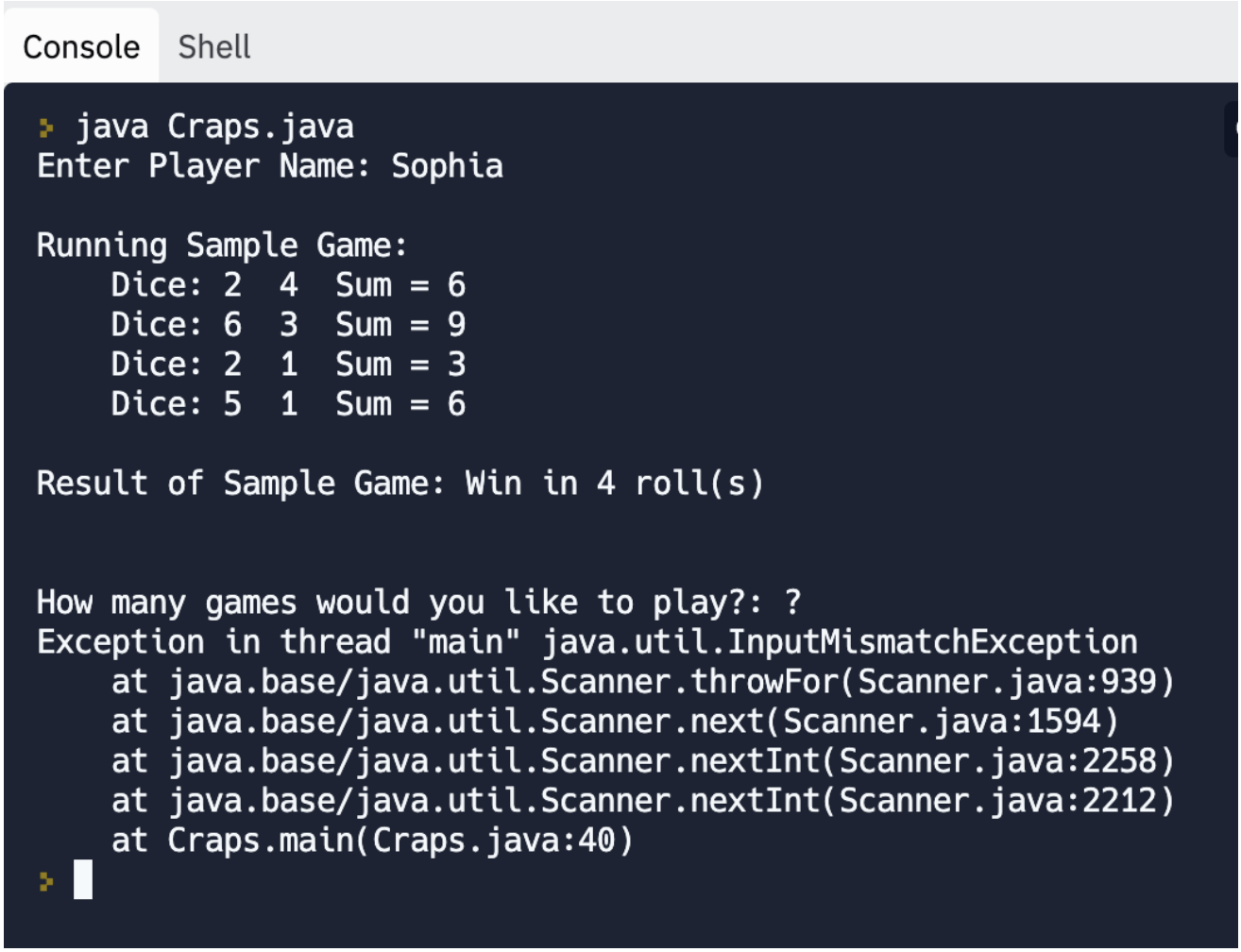
1. Invalid inputs—if the program asks the user to enter in an integer, try to enter something else, like a character.
2. Edge and corner cases for conditional statements.
3. Uppercase and lowercase inputs for conditional statements compared to strings.
4. Complete testing of end user use, or the program, from start to finish.

There may be times while debugging code when you cannot find a solution for an error. If so, you can move on temporarily, and you may simply add the code into a comment area and test it again later on.
In the sample program, the only required input is for the number of games the player would like to play. The user is prompted to enter a number. What happens if you enter a character other than a digit? The code in the relevant section of the application's main() could just try to read the number of games to play, like the code below.

EXAMPLE This is a simplified version of the code shown in the previous lesson that handles input using a more naive approach:

```
int numberOfGamesToPlay = 0;
System.out.print("How many games would you like to play?: ");
// Read input as an integer
numberOfGamesToPlay = input.nextInt();
System.out.println(); // Add blank line in output
```

If the Craps.java code is run using just this way of handling input and the user enters a character that is not a valid digit, the result looks like this:



TRY IT

Directions: Try running the Craps.java program from the previous lesson with the "simplified" approach to input shown above. While entering a valid number won't cause any problems, see what happens if you enter a letter or other symbol instead of a number.

EXAMPLE Take a look again at how the code presented in the previous unit actually handles this input:

```
// Track if input of requested # of games to play is valid
boolean inputValid = false;
int numberOfGamesToPlay = 0;

// Assume invalid input of number until it proves to be correct
while(!inputValid) {
   try {
      System.out.print("How many games would you like to play?: ");
      numberOfGamesToPlay = input.nextInt();
      // If previous statement doesn't throw exception, input valid
      inputValid = true;
   }
```

```
    catch(InputMismatchException ex) {
        System.out.println("Not a valid number.");
        // Clear out input buffer
        input.nextLine();
    }
}
System.out.println(); // Add blank line in output
```

While this version is longer and a bit more complex, it will provide for a better user experience.

In the code above for the input of the number of games to play, a boolean variable,`inputValid`, is declared and initialized to false. The while loop is then set up to keep running as long as `inputValid` is not true. The value of this variable is set to true after the Scanner has successfully read an integer. If the user does not enter a valid integer, the call to nextInt() throws an exception (an `InputMismatchException`) and control switches to the catch block, so the statement that sets `inputValid` to true does not run. This means that the loop will keep going until the user makes a valid entry. This is a common pattern used with user input in Java. This approach has the advantage of avoiding a program crash due to invalid input.

TRY IT

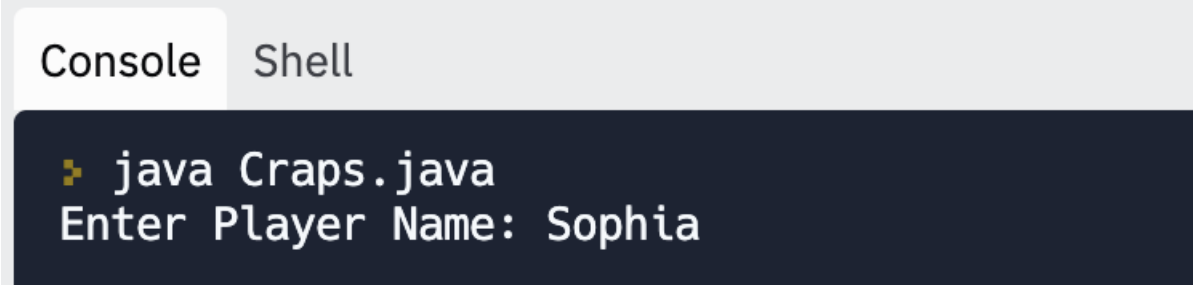
Directions: Try the original code for Craps.java with the improved input handling shown above. To make sure that you have the code entered correctly, here is a complete listing of the Craps.java file (as covered in the previous lesson) with the input section of the code in bold:

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.StandardOpenOption;
import java.util.Scanner;
import java.util.ArrayList;
import java.text.DecimalFormat;
import java.time.format.DateTimeFormatter;
import java.util.InputMismatchException;

public class Craps {
  public static void main(String[] args) {
    // Variables for stats
    int winCount = 0;
    int lossCount = 0;
    int rollCount = 0;

    // Log file for player statistics
    File logFile = new File("game.log.txt");

    // Scanner to read player name & desired number of games
    Scanner input = new Scanner(System.in);
    System.out.print("Enter Player Name: ");
```

```
String playerName = input.nextLine();

// Run sample game first
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();
for(DiceRoll roll : rolls) {
  System.out.println("\t" + roll);
}
System.out.println("\nResult of Sample Game: " + game.getResult() + " in " +
      game.getRollCount() + " roll(s)\n");
System.out.println();

// Track if input of requested # of games to play is valid
boolean inputValid = false;
int numberOfGamesToPlay = 0;

// Assume invalid input of number until it proves to be correct
while(!inputValid) {
  try {
    System.out.print("How many games would you like to play?: ");
    numberOfGamesToPlay = input.nextInt();
    // If previous statement doesn't throw exception, input valid
    inputValid = true;
  }
  catch(InputMismatchException ex) {
    System.out.println("Not a valid number.");
    // Clear out input to remove \n
    input.nextLine();
  }
}
System.out.println(); // Add blank line in output

// Loop to play desired number of games
for(int i = 0; i < numberOfGamesToPlay; i++) {
  game = new Game(playerName);
  // Need to use parentheses so value is calculated before concatenation
  System.out.println("Game " + (i + 1) + ":");
  rolls = game.play();
  rollCount += game.getRollCount();
  for(DiceRoll roll : rolls) {
    System.out.println("\t" + roll);
  }
  if(game.getResult().equals("Win")) {
    winCount++;
  }
```

```
      else {
        lossCount++;
      }
      System.out.println("\nResult: " + game.getResult() + " in " +
          game.getRollCount() + " roll(s)\n");
    }

    // Compose String with summary statistics
    String summary = "Statistics for " + game.getPlayerName() + ": " + "\n";
    String gameDateTimeFormatString = "MM/dd/YYYY HH:mm:ss\n";
    DateTimeFormatter dateTimeFormatter =
DateTimeFormatter.ofPattern(gameDateTimeFormatString);
    summary += "Game Date & Time: ";
    summary += dateTimeFormatter.format(game.getGameDateTime());
    summary += "# of Games: " + numberOfGamesToPlay + "\n";
    summary += "# of Dice Rolls: " + rollCount + "\n";
    summary += "# of Wins: " + winCount + "\n";
    summary += "# of Losses: " + lossCount + "\n";
    DecimalFormat percentageFormat = new DecimalFormat("0.00%");
    double winPercentage = (double) winCount / numberOfGamesToPlay;
    summary += "% Wins: " + percentageFormat.format(winPercentage) + "\n\n";

    // Display statistics on screen
    System.out.println(summary);

    // Write statistics to log file
    try {
      // Create log file if it doesn't exist and then append to it.
      Files.writeString(logFile.toPath(), summary, StandardOpenOption.CREATE,
              StandardOpenOption.APPEND);
    }
    catch(IOException ex) {
      System.out.println("Error writing to log file: " + ex.getMessage());
    }
   }
}
```

Running this code should produce output like the following when an incorrect value is entered for the number of games to play:



```
Running Sample Game:
    Dice: 1  1  Sum = 2

Result of Sample Game: Lose in 1 roll(s)


How many games would you like to play?: ?
Not a valid number.
How many games would you like to play?: X
Not a valid number.
How many games would you like to play?: 2

Game 1:
    Dice: 3  3  Sum = 6
    Dice: 3  5  Sum = 8
    Dice: 3  3  Sum = 6

Result: Win in 3 roll(s)

Game 2:
    Dice: 3  1  Sum = 4
    Dice: 5  2  Sum = 7

Result: Lose in 2 roll(s)

Statistics for Sophia:
Game Date & Time: 07/10/2022 19:41:09
# of Games: 2
# of Dice Rolls: 5
# of Wins: 1
# of Losses: 1
% Wins: 50.00%
```

REFLECT

The key constructs here are the while loop that keeps running until the user enters a valid number and the try/catch blocks. If the call to nextInt() fails due to invalid input, the program routes to the catch block immediately, so the boolean variable is not set to true, which means the loop keeps running. The use of try

and catch avoids an ugly error message (and crash), and the loop makes sure the user is prompted for a number until a valid entry is made.

1a. Debugging

If you have errors or issues that you can't figure out, remember you can temporarily add System.out.println() statements to display the values of variables and make sure they are being set and updated as expected. If it is hard to figure out how far the program has gotten before an error occurs, it can be helpful to add some outputs (using System.out.println()) with messages like "OK to line 50", etc. Just don't forget to remove these extra debugging statements when the problem has been fixed.

HINT

Adding brief comments like // DEBUG above or after these temporary statements can help you find and remove them later.

---

2. Adding the Fourth Journal Entry

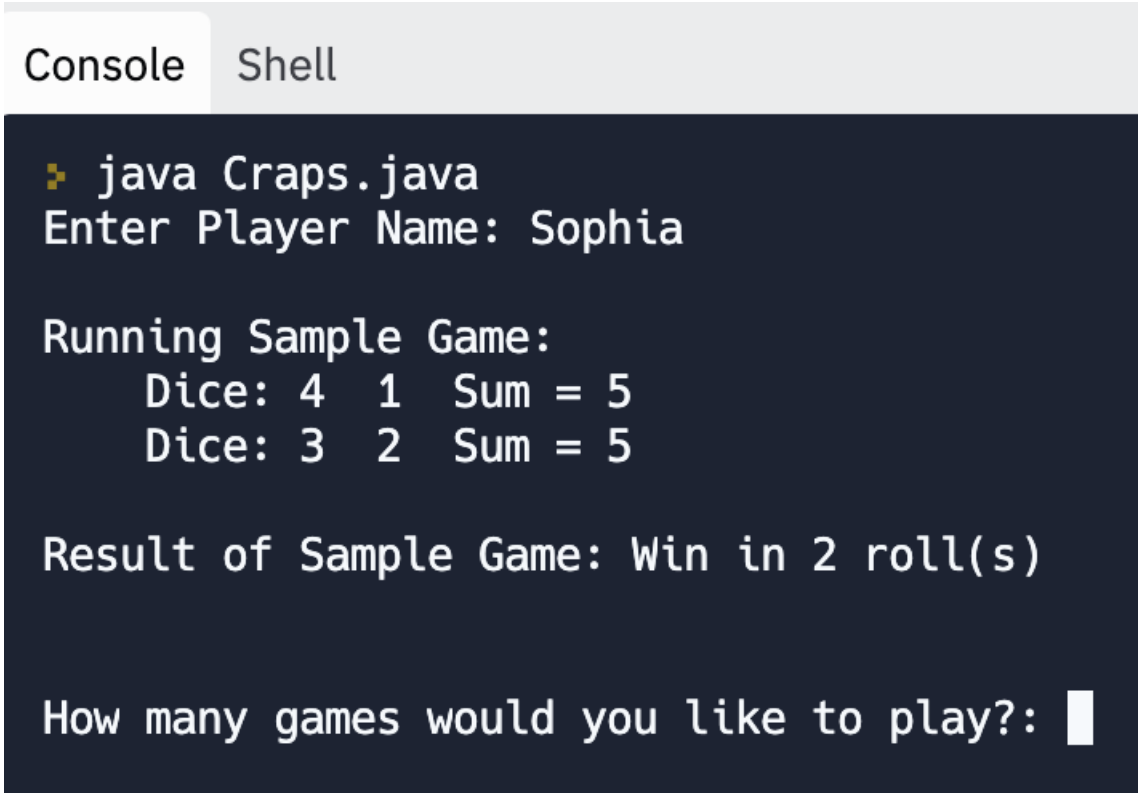
THINK ABOUT IT

At this point, we are ready to add the next journal entry (Part 4) for the Touchstone. The testing of your program is something you want to not only perform, but also show. You should document what you did for it and show the fixes that you made to resolve the problems that occurred.

What we would not want to add for our journal entry are the details of what was wrong without explaining what the issues were and how we fixed the issues. For example, this would be a bad entry for Part 4.

2a. Bad Example of Journal Entry for Part 4

Here is my test of the craps game.



HINT

A better entry for Part 4 would explain what the errors and issues were that came up and what was done to fix them. Simply looking at this output statement, you can't identify what the issue was specifically.

2b. Good Example of Journal Entry for Part 4

After changing my initial code in the main program to check for invalid user entries by adding the try and catch blocks, the program no longer crashed, but more was needed so that it would prompt the user to enter a valid number of games to play. The revised code uses a while loop so that the user is prompted for input of the number of games until a valid number is entered.

---

3. Guided Brainstorming

You will notice that as you program and test, you will likely change the code as you go. Doing so will ensure that the program works correctly every step of the way and simplifies debugging. Every program will be different, so it is important that you're thinking about the problem.

Let's look at our Drink Order program that we've been working on:

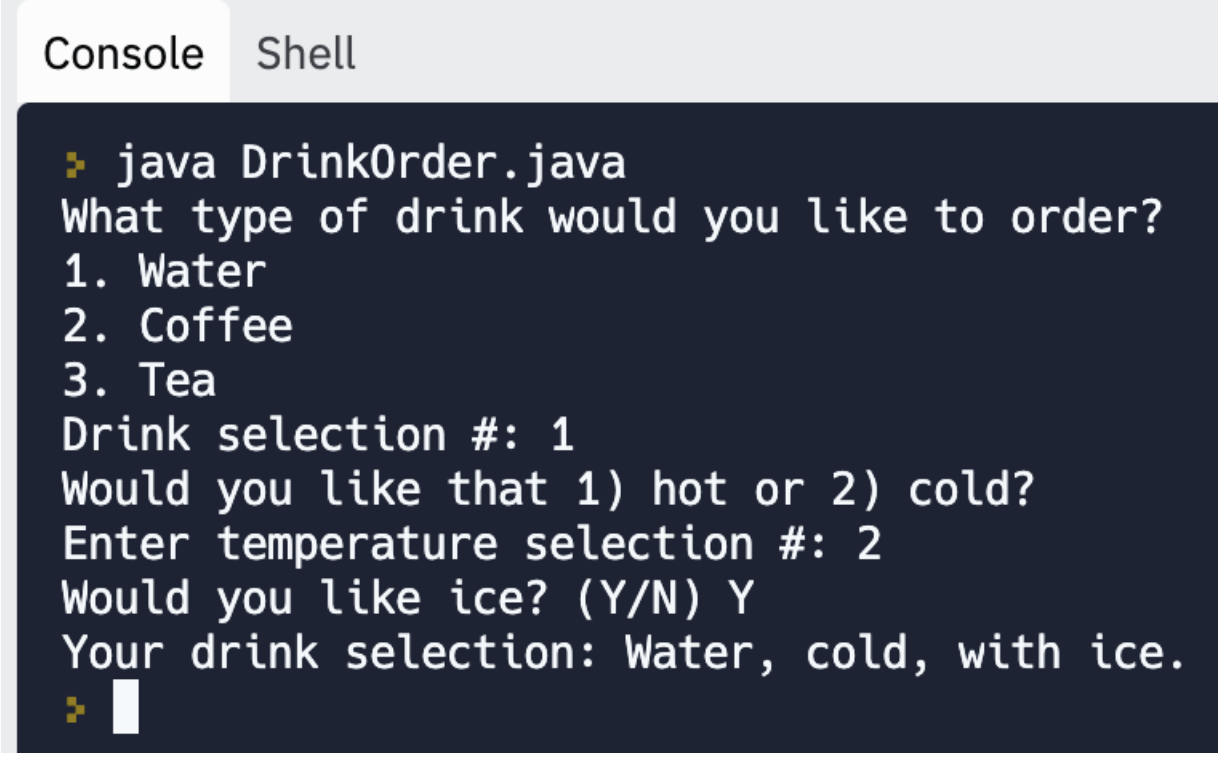
```
import java.util.Scanner;

public class DrinkOrder {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.println("What type of drink would you like to order?");
    // Note use of new line \n to print 3 lines with 1 statement.
```

```
System.out.println("1. Water\n2. Coffee\n3. Tea");
System.out.print("Drink selection #: ");
// String variable to hold drink details
String drinkDetails = "No drink chosen.";
int choice = input.nextInt();
// Remove \n left in input to avoid problems with later inputs
input.nextLine();
if(choice == 1) {
  drinkDetails = "Water";
  System.out.println("Would you like that 1) hot or 2) cold?");
  System.out.print("Enter temperature selection #: ");
  choice = input.nextInt();
  // Remove new line left in input stream to avoid problems with later inputs
  input.nextLine();
  if(choice == 1) {
    drinkDetails += ", hot";
  }
  else if(choice == 2) {
    drinkDetails += ", cold";
    System.out.print("Would you like ice? (Y/N) ");
    // Read input as a String
    String response = input.nextLine();
    // Extract 1st char
    char yesNo = response.charAt(0);
    // Y or y is yes, anything else interpreted as no
    if(yesNo == 'Y' || yesNo == 'y') {
      drinkDetails += ", with ice";
    }
  }
  else {
    System.out.println("Not a valid temperature selection.");
  }
}
else if(choice == 2) {
  drinkDetails = "Coffee";
  System.out.print("Would you like decaf? (Y/N): ");
  String decafResponse = input.nextLine();
  char decafYesNo = decafResponse.charAt(0);
  if(decafYesNo == 'Y' || decafYesNo == 'y') {
    drinkDetails += ", decaf";
  }
  System.out.println("Would you like 1) milk, 2) cream, or 3) none?");
  System.out.print("Enter choice #: ");
  int milkCreamChoice = input.nextInt();
  // Remove new line left in input stream to avoid problems with later inputs
  input.nextLine();
```

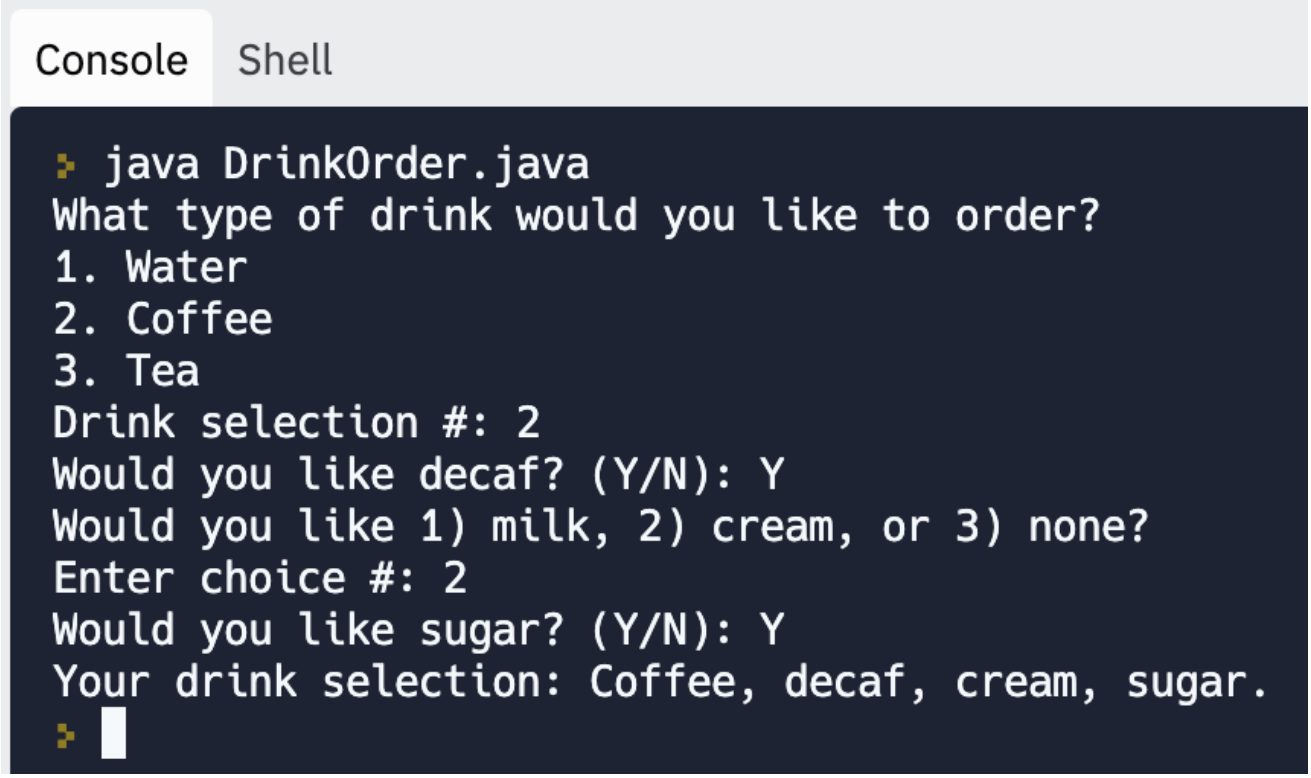
```
      if(milkCreamChoice == 1) {
        drinkDetails += ", milk";
      }
      else if(milkCreamChoice == 2) {
        drinkDetails += ", cream";
      }
      System.out.print("Would you like sugar? (Y/N): ");
      String sugarResponse = input.nextLine();
      char sugar = sugarResponse.charAt(0);
      if(sugar == 'Y' || sugar == 'y') {
        drinkDetails += ", sugar";
      }
    }
    else if(choice == 3) {
      drinkDetails = "Tea";
      System.out.print("Type of tea: 1) Black or 2) Green: ");
      int teaChoice = input.nextInt();
      // Remove \n left in input to avoid problems with later inputs
      input.nextLine();
      if(teaChoice == 1) {
        drinkDetails += ", black";
      }
      else if(teaChoice == 2) {
        drinkDetails += ", green";
      }
      else {
        // Invalid selection - assume black tea
        drinkDetails += ", black";
        System.out.println("Not a valid choice. Assuming black tea.");
      }
    }
    else {
      System.out.println("Sorry, not a valid drink selection.");
    }

    // Print out final drink selection
    System.out.println("Your drink selection: " + drinkDetails + ".");
  }
}
```

You should test each of the branches and scenarios of the valid values between the water, coffee, and tea to ensure it works as expected.

↷ EXAMPLE Let's try entering a 1 when choosing the beverage to see what happens:

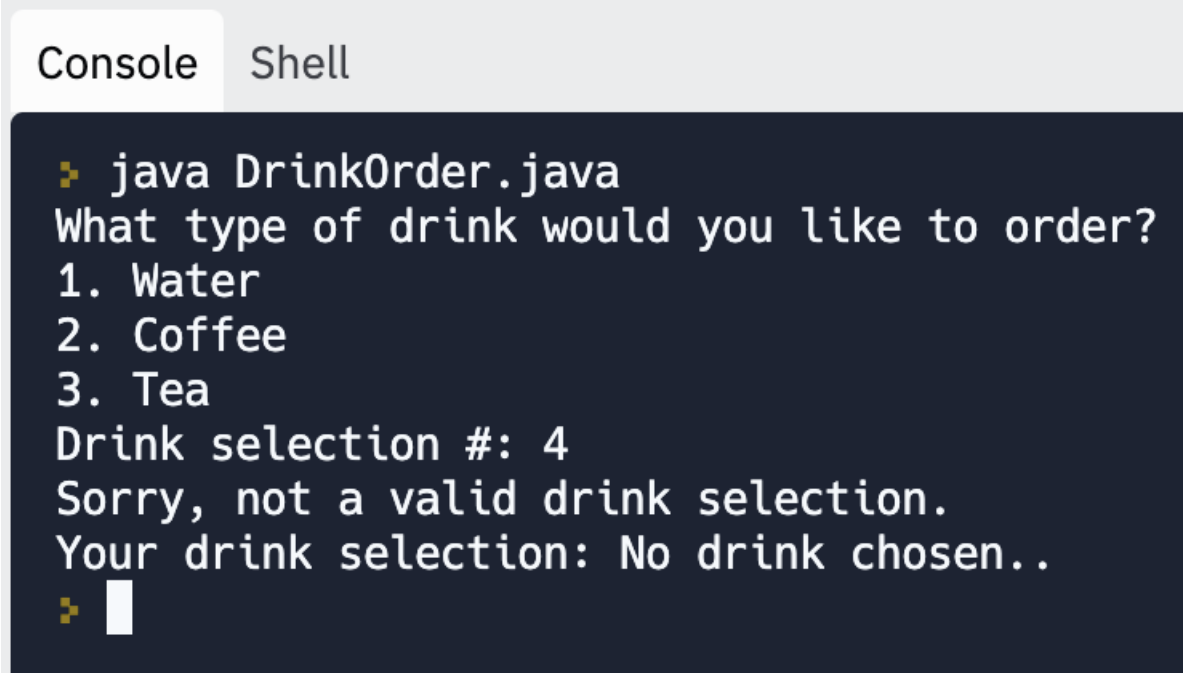


↗ EXAMPLE Let's try entering a 2 when choosing the beverage:



This is fine so far, but what happens if the user makes an invalid selection?

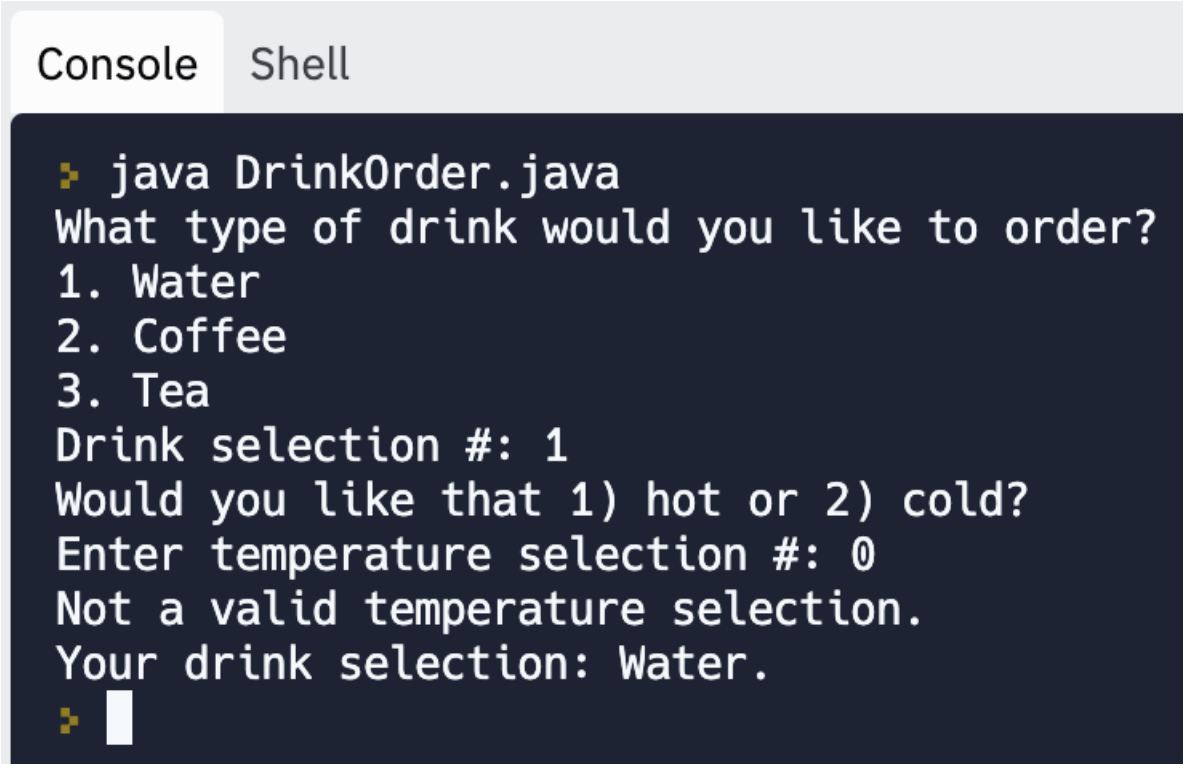
↗ EXAMPLE Let's try entering a 4 when choosing the beverage:



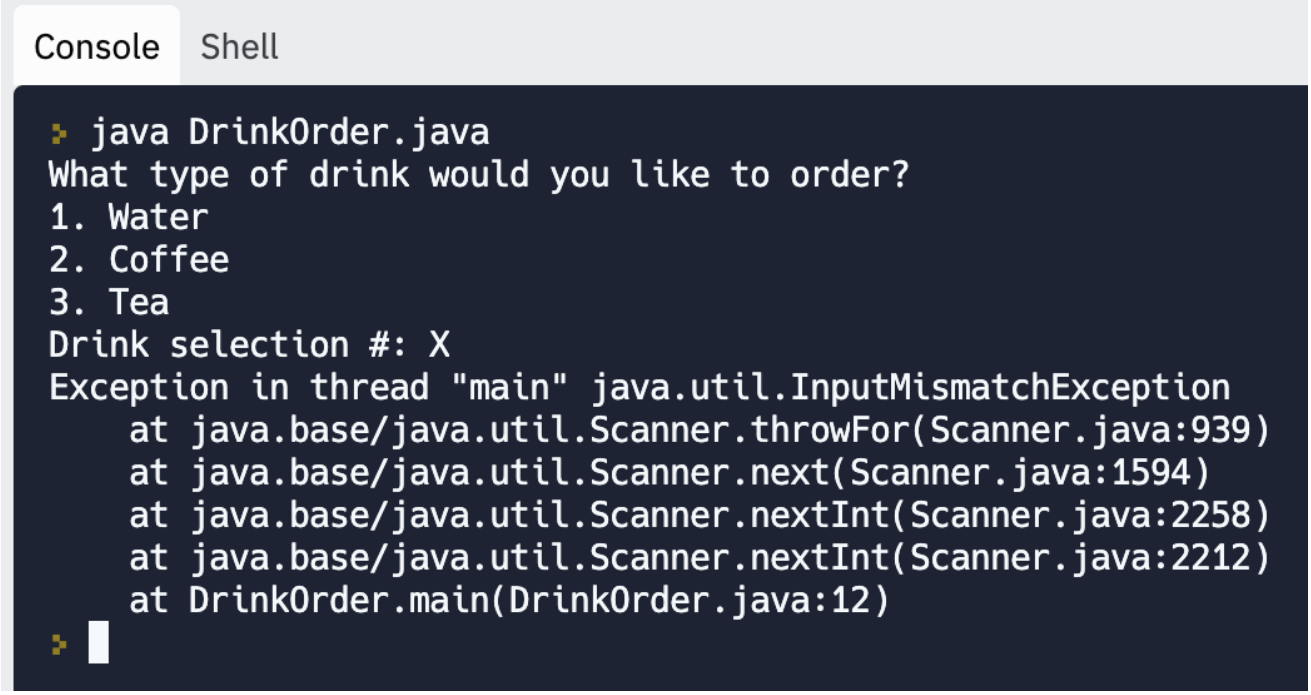
The program ends with an indication that the entry was not a valid drink selection. This is okay, but here is a place where introducing a loop to run until the user makes a valid selection could provide a better user experience (instead of requiring the person to start the program again).

Something similar happens if the person makes an incorrect entry later in the program.

↗ EXAMPLE Let's try entering the letter "O" when choosing the temperature selection:



If the user enters a non-digit when a numeric choice is expected, the program will crash (as we have seen before):



As with the craps game, it would be a good idea to use a combination of a loop around try and catch blocks to deal with problems reading the input. Here is a revised start of the code that handles invalid input similar to the solution for the number of games to play for the craps program. This is not a complete listing—just the start to show how the changes could be made.

↷ EXAMPLE

```
import java.util.InputMismatchException;
```

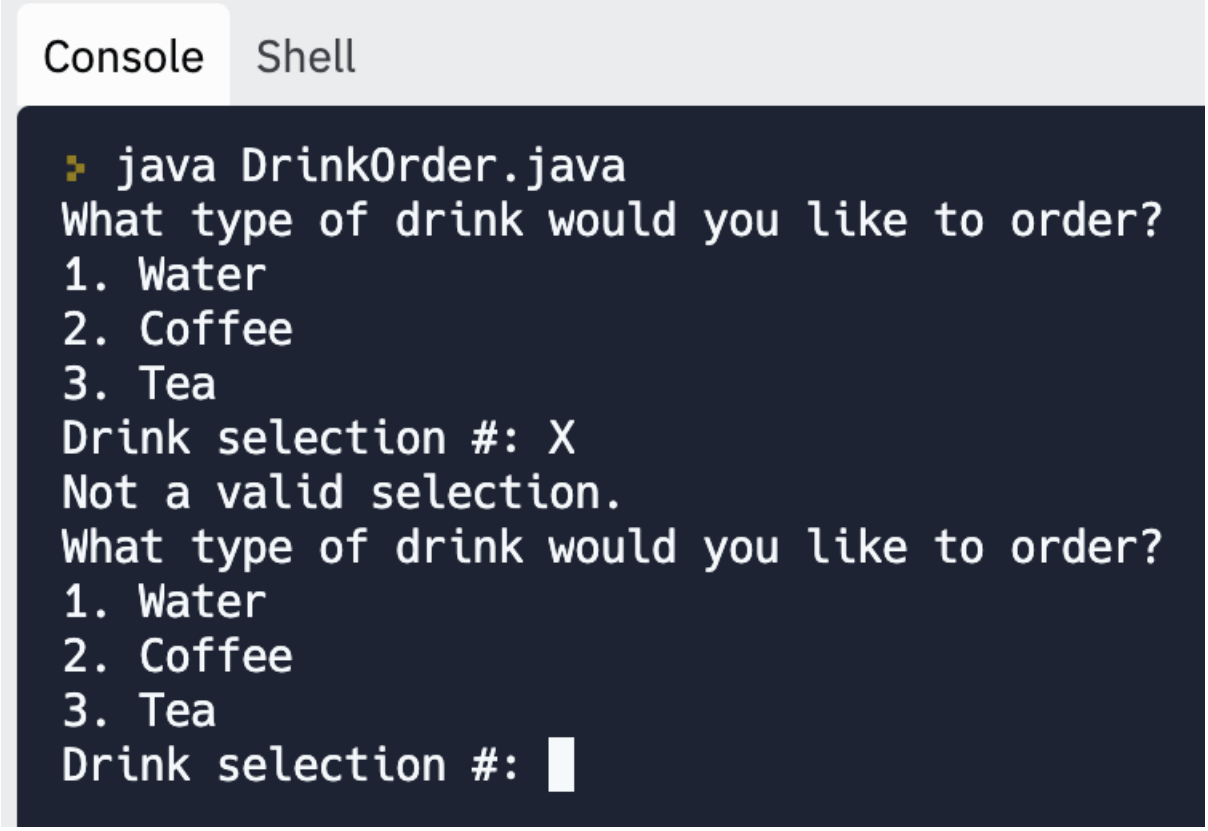
The version below includes `import java.util.InputMismatchException;` so the program can catch an `InputMismatchException` when `nextInt()` is called.

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class DrinkOrder {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean validInput = false;
    // String variable to hold drink details
    // Need to declare before start of loop to avoid scope problems
    String drinkDetails = "No drink chosen.";
    // Note need to declare choice variable before loop
    // to avoid problems with scope.
    int choice = 0;
    while(!validInput) {
      System.out.println("What type of drink would you like to order?");
      // Note use of new line \n to print 3 lines with 1 statement.
      System.out.println("1. Water\n2. Coffee\n3. Tea");
```

```
      System.out.print("Drink selection #: ");
      try {
        choice = input.nextInt();
        validInput = true;
      }
      catch(InputMismatchException ex) {
        System.out.println("Not a valid selection.");
      }
      // Remove \n left in input to avoid problems with later inputs
      input.nextLine();
    }
  }
}
```

The program now prompts the user again, if the input is not a valid digit.

↗ EXAMPLE Let's try entering the letter "X" at the drink selection prompt:



Now the program lets the user know that our drink selection was invalid. The code still does not allow another attempt if the number entered is out of range, though.

THINK ABOUT IT

What happens if you enter the number 4 as your drink selection? You might want to think about how you could modify the code to handle numeric user entry errors like we did for letter entry errors.

TRY IT

Directions: Needless to say, you should be testing your code as you are writing, but there are a lot of logical issues that can occur, especially with mistakes that may not be caught right away. It's easier to test cases that are supposed to work as intended versus ones that may break the code. For example, if the program is asking

for a number, we could test with a letter, or if a program is asking for numbers 1 through 10, we enter in 0, 11, or -5. Take the time to ensure that you have tested those edge cases as well as the completely incorrect cases. Review the example of a good entry for Part 4 in the Example Java Journal Submission document and add your entry for Part 4 to your Java Journal.

SUMMARY

In this lesson, you tested the demonstration program by **checking for errors**. You first checked for invalid user inputs and needed to add some exception handling. Then, you noticed that the program did not repeat the input request after an invalid value. You needed to add a loop to continue the input request until a valid value was added. Remember that you can temporarily add extra **debugging** statements if you have errors or issues that you can't figure out; just don't forget to remove them later when the problem has been fixed. You then had an opportunity to see both a **bad example and a good example** for our **fourth journal entry**. The good example had both what you were testing for and how you solved the need. In the **Guided Brainstorming** section, you identified other examples of good test cases to ensure the program is working the way it is intended to.

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/**