Fundamental Programming Structures in Java
IFT 194: HW 2

Brandon Doyle
bdoyle5@asu.edu
1215232174

Dr. Usha Jagannathan
Usha.Jagannathan@asu.edu

July 14, 2018

Section 2.1

2.1

The difference between the expressions `4`, `4.0`, `'4'`, and `"4"` is the type. For instance, the first is an `int`, the second may be a `double` or `float`, the third is a `char`, and the fourth is a `String` object.

2.3

We next examine the output of the following code fragment.

```
System.out.print("Here we go!");
System.out.println("12345");
System.out.print("Test this if you are not sure.");
System.out.print("Another.");
System.out.println();
System.out.println("All done.");
```

We expect this to output the following.

```
Here we go!12345
Test this if you are not sure.Another.
All done.
```

This is purely a result of using `println` as opposed to just `print`. Moreover, `println` can be called without any arguments.

2.4

We're asked to determine what is wrong with the following statement.

```
System.out.println("To be or not to be, that is the question.");
```

I honestly don't think there could be anything wrong with this statement, unless it's pointing at the fact that we may not continue strings on more than one line (as it may be formatted in the text). In that case, it is required that we use concatenation as

```
System.out.println("To be or not to be, that"
    + " is the question");
```

2.8

We are asked to determine which value is contained in the primitive `double` variable `depth` after the following lines execute.

View the source of this document on GitHub.

```
depth = 2.4;
depth = 20 - depth * 4;
depth = depth / 5;
```

These statements evaluate to

$$\frac{20 - 2.4 \times 4}{5} = \boxed{2.08.}$$

2.11

```
int iResult, num1 = 25, num2 = 40, num3 = 17, num4 = 5;
double fResult, val1 = 17.0, val2 = 12.78;
```

Given the above declarations, we're tasked with determining the result of several statements.

a. `iResult = num1 / num4` will store the integer 5 in `iResult`.
b. `fResult = num1 / num4` will store the double 5.0 in `fResult`.
c. `iResult = num3 / num4` will store the integer 3 in `iResult`.
d. `fResult = num3 / num4` will store the double 3.0 in `fResult`.
e. `fResult = val1 / num4` will store the double 3.4 in `fResult`.
f. `fResult = val1 / val2` will store the double 1.33... in `fResult`.
g. `iResult = num1 / num2` will store the integer 0 in `iResult`.
h. `fResult = (double)num1 / num2` will store the double 0.625 in `fResult`.
i. `fResult = num1 / (double)num2` will store the double 0.625 in `fResult`.
j. `fResult = (double)(num1 / num2)` will store the double 0.0 in `fResult`.
k. `iResult = (int)(val1 / num4)` will store the integer `3` in `iResult`.
l. `fResult = (int)(val1 / num4)` will store the double `3.0` in `fResult`.
m. `fResult = (int)((double)num1 / num2)` will store the double 0.0 in `fResult`.
n. `iResult = num3 % num4` will store the integer 2 in `iResult`.
o. `iResult = num2 % num3` will store the integer 6 in `iResult`.
p. `iResult = num3 % num2` will store the integer 17 in `iResult`.
q. `iResult = num2 % num4` will store the integer 0 in `iResult`.

2.12

We're tasked with stating the order in which the following expressions will be evaluated by providing a number beneath each operator.

a.
```
a - b - c - d
    1   2   3
```
b.
```
a - b + c - d
    1   2   3
```
c.
```
a + b / c / d
    3   1   2
```

d.
```
a + b / c * d
    3   1   2
```

e.
```
a / b * c * d
    1   2   3
```

f.
```
a % b / c * d
    1   2   3
```

g.
```
a % b % c % d
    1   2   3
```

h.
```
a - (b - c) - d
    2    1    3
```

i.
```
(a - (b - c)) - d
     2    1     3
```

j.
```
a - ((b - c) - d)
    3     1    2
```

k.
```
a % (b % c) * d * e
    2    1    3   4
```

l.
```
a + (b - c) * d - e
    3    1    2   4
```

m.
```
(a + b) * c + d * e
     1    2   4   3
```

Section 2.2

5.2

We are tasked with determining what the issue may be with the `if-else` blocks in the following source.

```
package hw_2;

public class Fragment1
{
    final static int MAX = 15;

    public static void main(String[] args)
    {
        int total = 15, sum = 16;

        if (total == MAX)
            if (total < sum)
                System.out.println("Total == MAX and < sum.");
        else
            System.out.println("Total is not equal to MAX.");
    }

}
```

Figure 1: Fragment1.java.

The problem is actually a well-documented feature of C-like languages: because C and Java ignore whitespace, indentation, despite being important for readability, is also ignored by the lexer. This means that Java has no way of knowing if we meant

```
if (X)
    if (Y)
        <block>
else
    <another block>
```

or

```
if (X)
    if (Y)
        <block>
    else
        <another block>
```

By default, Java appears to pair this `else`-statement with the nested `if`, which can be seen by setting `sum = 3` in my example in Figure 1 (which produces an incorrect result).

5.3

We are asked if the following code segment is valid if part of an otherwise valid program.

```
...
if (length = MIN_LENGTH)
    System.out.println("The length is minimal.");
...
```

This is of course invalid due to the assignment operator in the `if`-statement's condition instead of `==`.

5.9

We're given the following code segment.

```
package hw_2;

public class Loop
{
    public static void main(String[] args)
    {
        int count = 50;
        while (count >= 0)
        {
            System.out.println(count);
            count++;
        }
    }
}
```

Figure 2: Loop.java.

The problem is that it's an infinite loop. Three changes we could make to fix this are

1. `count` could be initialized to a negative integer;

2. the `while`-loop's condition could be changed to `(count < x)`, where $x > 50$;

3. the increment `count++` could be changed to a decrement `count--`.

5.10

We're tasked with writing a `while`-loop that verifies that the user enters a positive integer value. See Figure 3 for my solution

```java
package hw_2;

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

public class PositiveInput
{
    public static void main(String[] args)
    {
        int input = 0;

        try (var scnr = new Scanner(System.in)) {
            // My while-loop
            while (true)
            {
                System.out.print("Please enter a positive integer: ");
                try {
                    input = scnr.nextInt();
                    if (input > 0)
                        break;
                    else
                        System.out.println("*** Error: please enter a positive integer");
                } catch (InputMismatchException ex) {
                    System.out.println("*** Error: please enter an integer");
                    scnr.next();
                }
            }
        }

        System.out.println(String.format("You entered: %d", input));
    }
}
```

Figure 3: PositiveInput.java.

PP 5.4

We're asked to write a simple implementation of the game Hi-Lo. Please see Figure 4 for my solution. Also, it seems I can break this program by typing a line with whitespace. I'm actively thinking and searching for a way to fix this.

6.1

We're asked to determine how many iterations the following `for`-loops will execute.

a. `for (int i = 0; i < 20; ++i) {}` – this loop will iterate 20 times.
b. `for (int i = 0; i <= 20; ++i) {}` – this loop will iterate 21 times.
c. `for (int i = 5; i < 20; ++i) {}` – this loop will iterate 15 times.

d. `for (int i = 20; i > 0; --i) {}` – this loop will iterate 20 times.
e. `for (int i = 1; i < 20; i += 2) {}` – this loop will iterate 10 times.
f. `for (int i = 1; i < 20; i *= 2) {}` – this loop will iterate 5 times.

6.4

We're asked to transform the following `while`-loop into an equivalent `do-while`-loop.

```
package hw_2;

public class While
{
    public static void main(String[] args)
    {
        int num = 1;
        while (num < 20)
        {
            num++;
            System.out.println(num);
        }
    }
}
```

Figure 5: While.java.

I arrive at the following solution.

```
package hw_2;

public class DoWhile
{
    public static void main(String[] args)
    {
        int num = 1;
        do {
            num++;
            System.out.println(num);
        } while (num < 20);
    }
}
```

Figure 6: DoWhile.java.

6.8

We're asked to write a `for`-loop that will print the multiples of 3 from 300 down to 3. See Figure 7 for my solution.

6.9

We're asked to write a program that will accept 10 integers from a user and print the largest. See Figure 8 for my solution.

```java
package hw_2;

public class Multiples
{
    public static void main(String[] args)
    {
        for (int i = 300; i >= 3; System.out.println(i), i -= 3);
    }
}
```

Figure 7: Multiples.java.

```java
package hw_2;

import java.util.Scanner;
import java.util.InputMismatchException;
import java.util.Arrays;

public class Largest
{
    /**
     * Accept 10 integers from a user and print the largest back to the console.
     *
     * @param args Not used.
     */
    public static void main(String[] args)
    {
        try (var scnr = new Scanner(System.in)) {
            int inputIntegers[] = new int[10];

            for (int i = 0; i < 10; ++i)
            {
                while (true)
                {
                    System.out.print(String.format("Enter integer %d: ", i + 1));
                    try {
                        inputIntegers[i] = scnr.nextInt();
                        break;
                    } catch (InputMismatchException ex) {
                        System.out.println("*** Error: Please enter an integer");
                        scnr.next();
                    }
                }
            }

            // Use a stream, available since Java 8
            int largest = Arrays.stream(inputIntegers).max().getAsInt();

            /* Equivalent to the following
             *
             * int largest = 0;
             * for (int i = 0; i < inputIntegers.length; ++i)
             *  if (inputIntegers[i]> largest)
             *      largest = inputIntegers[i];
             */

            System.out.println("The largest integer you input was: " + largest);
        }
    }
}
```

Figure 8: Loop.java.

PP 6.3

We're asked to write a program that will output a 12×12 multiplication table to the console. Please see Figure 9 for my solution.

```java
package hw_2;

public class MultiplicationTable
{
    public static final int SIZE = 12;

    /**
     * Print a multiplication table to the console.
     *
     * @param args  Not used.
     */
    public static void main(String[] args)
    {
        for (int i = 1; i <= SIZE; ++i)
        {
            for (int j = 1; j <= SIZE; ++j)
                System.out.print(i * j + "\t");
            System.out.println("\n");
        }
    }
}
```

Figure 9: MultiplicationTable.java.

Section 2.3

8.1

We are tasked with determining which of the following are valid (primitive) array declarations.

a. `int primes = { 2, 3, 4, 5, 7, 11 };` – this is not a valid array declaration because it's missing brackets after the identifier (and 4 is not prime).
b. `float elapsedTimes[] = { 11.47, 12.04, 11.72, 13.88 };` – this is *almost* a valid declaration, with the exception that the arrays contents are implicitly doubles, whereas we'd like to use floats (half the size).
c. `int[] scores = int[30];` – this is incorrect because it's missing the `new` keyword to instantiate the object.
d. `int[] primes = new { 2, 3, 5, 7, 11 };` – this is again invalid syntactically, because the `new` keyword is unnecessary when we're explicitly stating what we want the array to contain.
e. `int[] scores = new int[30];` – this is a valid declaration and a new array object is created as a result.
f. `char grades[] = { 'a', 'b', 'c', 'd', 'f' };` – this is, perhaps surprisingly, valid. Apparently, there is a slight difference between pairing the brackets with the type as opposed to the identifier, but I would rather pair them with the type.
g. `char[] grades = new char[];` – this is invalid because we must declare the initial size of the array object in the latter pair of brackets.

8.4

We're tasked with determing the problems that can arise within the loop in Figure 10. I immediately noticed that it's indexed at 1, which is problem because arrays are indexed at 0. Hence, `i` should start at 0 and proceed until `i < numbers.length` is violated.

```java
package hw_2;

public class LoopProblems
{

    public static void main(String[] args)
    {
        int[] numbers = {3, 2, 3, 6, 9, 10, 12, 32, 3, 12, 6};
        for (int i = 1; i <= numbers.length; ++i)
            System.out.println(numbers[i]);
    }

}
```

Figure 10: LoopProblems.java.

8.5

In this problem we're asked to write array declarations to represent students' names for a class of 25 students and students' test grades for a class of 40. See my solutions in Figure 11.

```java
package hw_2;

public class ArrayDeclarations
{
    public static void main(String[] args)
    {
        // a.
        String[] studentsNames = new String[25];

        // b.
        int[] testGrades = new int[40];
    }
}
```

Figure 11: ArrayDeclarations.java

8.6

We're asked to write code that initializes an entire array to the constant `INITIAL`. See Figure 12 for my solution.

```java
package hw_2;

import java.util.Arrays;

public class Initialization
{
    public static final int INITIAL = 5;

    public static void main(String[] args)
    {
        int[] arr = new int[30];

        // Initialize all elements to INITIAL value.
        for (int i = 0; i < arr.length; ++i)
            arr[i] = INITIAL;
    }
}
```

Figure 12: Initialization.java.

8.7

Here we're asked to print all the elements of an array called `names` backward. See Figure 13 for my solution.

```java
package hw_2;

import java.util.Random;

public class Backwards
{
    public static void main(String[] args)
    {
        var generator = new Random();

        int[] arr = new int[30];

        // Initialize array
        for (int i = 0; i < arr.length; ++i)
            arr[i] = generator.nextInt(arr.length);

        // Print contents forward
        for (int i = 0; i < arr.length; ++i)
            System.out.print(arr[i]+ " ");
        System.out.println();

        // Print contents backward
        for (int i = arr.length - 1; i >= 0; --i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }
}
```

Figure 13: Backwards.java.

```java
package hw_2;

import java.util.Random;
import java.util.Scanner;
import java.util.InputMismatchException;

public class HiLo
{
    final public static int MAX = 100;

    /**
     * Play Hi-Lo.
     *
     * TODO: Make input with spaces safe.
     *
     * @param args  Not used.
     */
    public static void main(String[] args)
    {
        var generator = new Random();
        int randomNumber = generator.nextInt(MAX);
        int totalGuesses = 0;
        int currentGuess = 0;
        String keepPlaying = "y";

        System.out.println("Welcome to Hi-Lo!");

        try (var scnr = new Scanner(System.in)) {
            while (!keepPlaying.matches("^[^Yy].*"))
            {
                while (true)
                {
                    currentGuess = getNextGuess(scnr);

                    if (currentGuess == -1) {
                        // Exit the program
                        keepPlaying = "n";
                        break;
                    }

                    if (currentGuess == randomNumber) {
                        System.out.println("\n    CORRECT!");
                        totalGuesses++;
                        break;
                    } else if (currentGuess < randomNumber) {
                        System.out.println("\n    Too low!");
                    } else {
                        System.out.println("\n    Too high!");
                    }

                    totalGuesses++;
                }

                System.out.println("\nTotal number of guesses: " + totalGuesses);

                totalGuesses = 0;

                if (currentGuess != -1) {
                    System.out.print("\nWould you like to keep playing? [y|n]: ");
                    keepPlaying = scnr.next();
                }
            }
        }

        System.out.println("\n*** Exiting");
    }

    /**
     * Get user input for a guess to the game.
     *
     * @return The user's input, which may range from -1 to MAX.
     */
    public static int getNextGuess(Scanner scnr)
    {
```

```
        int input = 0;

        while (true)
        {
            System.out.print(String.format("\nPlease enter an integer between 0 and"
                                           + " %d (or -1 to exit): ", MAX));
            try {
                input = scnr.nextInt();

                // Check bounds
                if (input <= MAX && input >= -1)
                    break;
                else
                    System.out.println(
                        "*** Error: Please enter an integer between 0 and " + MAX
                    );
            } catch (InputMismatchException ex) {
                System.out.println("*** Error: Please enter an integer");
                scnr.next();
            }
        }

        return input;
    }
}
```

Figure 4: HiLo.java