Notes: Primitive Data and Definite Loops

Literals / Values

- int literal: number without a decimal, e.g., -7, 0, 103
- double literal: number with a decimal, e.g., -7.0, 0.2, 103.5
- String literal: characters surrounded by quotes, e.g., "hello world"

Data types

- Three main data types (you will learn more later)

Data type	**Description**
int	integers, pos, zero, neg, up to 2^31-1
double	floating point numbers (real), pos, zero, neg, up to 10^308
String	series of text characters

Operators

- **Java precedence**: when operators of the same precedence appear next to each other, they are evaluated left to right.

Priority	**Operation**
1	parens
2	unary operations, **casting**
3	multiplication, division, mod
4	addition, subtraction, string concatenation
5	less than, less than or equal to, greater than, greater than or equal to
6	equal to, not equal to

- **Integer division**: When dividing integers, all information after the decimal point is lost. This is called truncating.

```
System.out.println( 6 / 2 );      //results in 3
System.out.println( 6 / 2.0 );    //results in 3.0
System.out.println( 6.0 / 2 );    //results in 3.0
System.out.println( 13 / 2 );     //results in 6
System.out.println( 13 / 2.0 );   //results in 6.5
```

- **Casting**: You can force Java to change a data type. If you cast a double to an int, you _always round down_ (truncate).

```
System.out.println( (double) 47 );        //results in 47.0
System.out.println( (double) 47 / 2 );    //results in 23.5
System.out.println( (double)(47 / 2) );   //results in 23.0 bc the paren's ha
ppen first
System.out.println( (int) 3.2 );          //results in 3
System.out.println( (int) 3.9 );          //results in 3
```

Variables

- We need to declare variables with a type and a name before they can be used
- Once a variable is declared you cannot redeclare it (with a type)

Assignment statements

- `=` is the assignment operator, it has nothing to do with equality
- Assignment statements should be read right-to-left

```
// 1 is stored in the int variable named x
int x = 1;

// 1 is added to the current value of x (in this case 1), resulting in 2
// and that is stored back into x
x = x + 1;

// "hello" is stored in the String variable named greeting
String greeting = "hello";
```

For Loops

- Canonical Example

```
for(int i = 0; i < 10; i++) {
    System.out.println(i);
}
```

- Structure

```
for(initialization; test; update) {
    body
}
```

The order of execution of a for-loop is:

1. initialization
2. check the test condition
3. if the test condition is `true`, execute the statements inside the body of the loop
4. execute the update
5. repeat steps 2 and 3 until the condition is `false`

- Nested Loop Example

```
for(int i = 1; i <= 3; i++) {
    for(int j = 1; j <= i; j++) {
        System.out.print(j);
    }
    System.out.println();
}
```

Generally the outer loop of two nested for-loops controls the number of rows of output while the inner loop controls the output on a single line. The code above produces:

```
1
12
123
```

Notes: Definite Loops, Constants

Constants

- Should be at the top of your program
- Start with `public static final`
- Can be used throughout the program and are considered "global variables"

Nest Loops

- With ASCII Art drawing using nested loops:
    - Though for loops generally start with 0 (for example when we learn Arrays and ArrayLists in the future), in ASCII art, we often start with 1
    - The outer loop generally controls the number of line of output
    - The inner loops should have `System.out.print` statements inside, to ensure that Characters are printed next to each other
    - At the very end of the outer loop's body, there is usually a `System.out.println()` so that the next iteration of the outer loop prints on the next line of output

```
/* output:
    ......
    ......
    ......
*/
public static void dotBox() {
      // controls number of lines of output
      for(int line = 1; line <= 3; line++) {
         // controls number of columns of output per line
         for(int dot = 1; dot <= 6; dot++) {
            System.out.print(".");
         }
         System.out.println();
      }
   }
```

Case Study: Hourglass Figure

From the textbook:

```
// This produces the top half of the hourglass figure
/* output:
    +------+
    |\..../|
    | \../ |
    |  \/  |
    |  /\  |
    | /..\ |
    |/....\|
    +------+
*/
public class DrawFigure2 {
    public static final int SUB_HEIGHT = 4;

    public static void main(String[] args) {
        drawLine();
        drawTop();
        drawBottom();
        drawLine();
    }

    // Produces a solid line
    public static void drawLine() {
        System.out.print("+");
        for (int i = 1; i <= (2 * SUB_HEIGHT); i++) {
            System.out.print("-");
        }
        System.out.println("+");
    }

    public static void drawTop() {
        for (int line = 1; line <= SUB_HEIGHT; line++) {
            System.out.print("|");
            for (int i = 1; i <= (line - 1); i++) {
                System.out.print(" ");
            }
            System.out.print("\\");
            for (int i = 1; i <= (2 * SUB_HEIGHT - 2 * line); i++) {
                System.out.print(".");
            }
            System.out.print("/");
            for (int i = 1; i <= (line - 1); i++) {
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }

    // This produces the bottom half of the hourglass figure
    public static void drawBottom() {
        for (int line = 1; line <= SUB_HEIGHT; line++) {
            System.out.print("|");
            for (int i = 1; i <= (SUB_HEIGHT - line); i++) {
                System.out.print(" ");
            }
            System.out.print("/");
            for (int i = 1; i <= 2 * (line - 1); i++) {
                System.out.print(".");
            }
            System.out.print("\\");
            for (int i = 1; i <= (SUB_HEIGHT - line); i++) {
                System.out.print(" ");
            }
            System.out.println("|");
        }
    }
}
```