CS 2300 Week 03 Lab

Elementary Control Structures

0. Reminders.

Refer to the Week 02 Lab for information about using Intellij IDEA.

All labs are submitted to Blackboard and graded online.

1. Keyboard Shortcuts.

Most editors and IDE's have **keyboard shortcuts** – key combinations you can use that are faster than mousing, clicking, and typing in stuff. Some of these are more or less standard in many or all applications. For example, `ctrl + C`, `ctrl + X`, `ctrl + V` for copy, cut, and paste, which go back to the WordStar program in the 80's, work in most applications.

Intellij IDEA has an extensive set of Keyboard Shortcuts. You can check these out at `Help / Keyboard Shortcuts.pdf`.

Intellij IDEA also will accept other standard sets of keyboard shortcuts. The NetBeans IDE has many very useful shortcuts, so it is a good idea to have these included in IDEA, as follows:

1.1. Enter `Ctrl + Alt + S` to open the Settings/Preferences dialog.

1.2. Select **Keymap** in the list on the left edge.

1.3. At the top of the main part of the screen is a box that might say **Windows** or **MacOS** in it. Click on this box to get a pulldown menu. Select **NetBeans** and click **OK**. Now all the NetBeans keyboard shortcuts will work!

1.4. Two very useful NetBeans shortcuts are **sout** and **soutv**.

1.5. To use sout, start on a new line and type `sout` and then hit the `Tab` key. This will cause the sout to expand into:

```
System.out.println();
```

This is very useful, because you need this statement all the time. Once you have this, you can put something inside the parenthesis, like this:

```
System.out.println("Wow! this is cool!");
```

which will be printed to the console when the program runs.

1.6. The `soutv` shortcut is similar, but prints out whatever variable has been most recently defined. This is a huge help when testing your program to be sure it is working correctly.

2. Start small. Do one thing at a time.

2.1. Create a minimal "Hello World" program the same way as last week, only this time have the program write `Lab 03: Elementary Control Structures` rather than `Hello World!` This will serve as a title for the program.

It is also a good habit to put some output at the end. For example, you can put in the statement

```
System.out.println("\n\nGoodbye.\n\n");
```

at the end of your program. Then put everything else in between the Title line and the Goodbye line. Note that there are a couple of `\n` symbols included in the string. That is the magic symbol for **new line**. This results in a blank line being printed. It is often a good idea to put bland lines in your output to keep everything from being crammed together.

2.2. Run the program. When you run the program, an output window should open and show you the output, as well as how long it took to run. If it does not work, fix it.

2.3. When it works correctly, go on and add some more stuff.

3. Control Structures

The reminder of the lab adds code to demonstrate how **control structures** work. These will be further discussed in class.

The code goes **inside** the `main` method, between the Title line and the Goodbye line.

4. Expressions.

Control structures are governed by expressions, which can have the value **true** or **false**. Add the following code, and figure out why each expression is true or is false:

```
System.out.println("\nExpression Values");

System.out.println("4 < 10:     " + ( 4 < 10 ));
System.out.println("4 < 4:      " + ( 4 < 4  ));
System.out.println("4 <= 4:     " + ( 4 <= 4 ));
System.out.println("4 == 4:     " + ( 4 == 4 ));
System.out.println("4 != 4:     " + ( 4 != 4 ));
System.out.println("4 % 2 == 0: " + ( 4 % 2 == 0 ));
System.out.println("true:       " + true);
System.out.println("not true:   " + ! true);
```

This example has a slightly confusing aspect. In the first six statements, an extra pair of parentheses -- `( e < 10 )` -- were needed because otherwise the `+` operator gets confused. The last two were not ambiguous so parentheses were not needed.

To illustrate this point, add the following code after the previous code.

```
System.out.println("\nTwo meanings for + operator");
```

```
    System.out.println("concatenation: " + 4 + 4);
    System.out.println("addition:      " + (4 + 4));
```

The code, `"con.." + e + e` has two + operators. The compiler must figure out if the two + signs mean addition or concatenation. How does it decide? In the first line, the two + signs are dealt with from left to right. For the left + sign, the compiler looks to the left of the + and sees a string. Therefore it treats + as concatenation, and pastes a `4` onto the end of the string. The it deals with the right + sign, and pastes another `4` onto the end, resulting in `44`.

For the code, `"add.." + (e + e)` there is a pair of parentheses, which always overrule precedence decision, so first right + sign adds the `4 + 4` to get `8`. Then the left + sign pastes the `8` onto the end of the string.

5. The `for` loop.

Repetition can be done with a **for loop**, a **while loop**, or a **do – while loop**. Add the following simple for loop.

```
System.out.println();
for (int i = 1; i < 10; ++i){
    System.out.println(i + i);
}
```

Run the program. What does it do?

The `System.out.println()` at the top skips a line so the output is not crammed onto the previous output. The brief statement `++i` **increments** the value in `i`. That means add `1` to `i`. This is a very common operation and is done a lot.

6. The `if – else` structure.

Selection often takes the form of an **if – else** statement like this:

```
if ( expression ) {
    do one thing
} else {
    do another thing
}
```

7. Loop with selection.

Next we will write a loop with an if – else statement inside it to check if numbers are even or odd.

7.1. Add the following:

```
System.out.println();
for (int i = 0; i < 20; ++i){
    if (i % 2 == 0){
        System.out.println(i + " is even.");
    } else {
        System.out.println(i + " is odd.");
    }
}
```

7.2. If you are having trouble getting the formatting right, mark the section, and hit **alt + shift + f** to format it. Or, you can select the entire file (with **ctrl + a**), and format the entire thing.

7.3. This code uses a **for loop** for repetition, to write 20 lines of output.

7.7. This code uses an **if** – **else statement**, to do one thing, or do the other thing.

7.5. This code uses the **mod operator**, **%**, to control the if – else statement. The mod operator returns the remainder when you do integer division. For example, 123 % 10 is equal to 3, because that is the remainder when you divide 123 by 10. In this case we have `i % 2`, which is 0 if `i` is an even number, and 1 if `i` is an odd number.

7.6. The code also uses the **test for equality operator**, **==**. This operator is true if both things are the same, and false otherwise. Important note: you can only use the `==` to test for equality with primitive data types (such as `int`). For classes you must use the **equals** method like this: `bigInt1.equals(bigInt2)`.

8. Application of loops.

Now add the following code that adds up the numbers from 1 to 20.

Whenever you are adding up a bunch of things (a very common task!), you need an **accumulator** (a variable to add stuff to). In this case, the accumulator is called `total`, because that is a good name for it.

```
System.out.println();
int total = 0;
for (int i = 1; i <= 20; ++i){
    total += i;
}
System.out.println("total is " + total);
```

The line

```
total += i;
```

could also have been written as

```
total = total + i;
```

The first way might look funny at first, but it is better.

Run this program, and see what the total is.

9. Submit your lab.

When everything works, submit the program to Blackboard.

See instructions for Labs 01 and 02 if you need a reminder on submitting.