Lab Questions
Big Java, Late Objects / Java for Everyone, 2e
Chapter Number: 3 Decisions

1) The `if` statement is used to implement a decision. The simplest form of an `if` statement has two parts: a condition and a body. If the *condition* is true, the *body* of the statement is executed. The body of the `if` statement consists of a statement block.

Consider the following code:

```
if (n > 10)
{
   System.out.print("*****");
}
if (n > 7)
{
   System.out.print("****");
}
if (n > 4)
{
   System.out.print("***");
}
if (n > 1)
{
   System.out.print("**");
}
System.out.println("*");
```

How many * will be printed when the code is executed

a) with n = 6 ?
b) with n = 20 ?
c) with n = 2 ?
d) with n = –1 ?

2.1) An alternate form for an `if` statement has multiple parts: a *condition* that evaluates to true or false, a *statement* that is executed if the condition is true, the word *else*, and finally a *statement* that is executed when the condition is false. Each statement can be a *simple statement* consisting of a single Java instruction, a compound statement (such as another `if` statement) or a block statement (matching braces `{}` that surround one or more Java statements). We suggest using the brace notation in every case. Consider the code below that prompts the user to input a value for `x` and for `y`. It then prints the smallest value contained in the variables `x` and `y`.

```
import java.util.*;

public class SmallestInt
{
   public static void main(String[] args)
```



```
   {
      Scanner scan = new Scanner(System.in);
      System.out.println("Enter a value for x:");
      int x = scan.nextInt();
      System.out.println("Enter a value for y:");
      int y = scan.nextInt();
      if (x <= y)
      {
         System.out.println("The smallest value was " + x);
      }
      else
      {
         System.out.println("The smallest value was " + y);
      }
   }
}
```

Modify the code above so that it prompts the user to enter a third value for a variable z. Rewrite the logic so that the program prints out the smallest value contained in x, y, and z.

2.2) The code below is a more efficient solution to the problem in Lab 3.2.1.

```
import java.util.*;

public class SmallestInt
{
   public static void main(String[] args)
   {
      Scanner scan = new Scanner(System.in);

      System.out.println("Enter a value for x:");
      int x = scan.nextInt();
      int smallest = x;   // x is the smallest value so far

      System.out.println("Enter a value for y:");
      int y = scan.nextInt();
      if (y < smallest)
      {
         smallest = y;   // Update smallest if necessary
      }
      System.out.println("Enter a value for z:");
      int z = scan.nextInt();
      if (z < smallest)
      {
         smallest = z;   // Update smallest if necessary
      }

      System.out.println("The smallest value was " + smallest);
   }
}
```

Modify the code so that it prompts the user for four integers (w, x, y, and z) and prints the smallest value contained in those variables. How hard would it be to modify the version of the

program you wrote in Lab 3.2.1 (or that lab's suggested solution) to solve the four-variable problem?

3) In the code below, the `if` statement evaluates the condition `x < 10` and assigns the variable `color` either the value `"red"` or `"blue"`. The condition is first examined and the corresponding alternative is taken. The strategy in this code is to wait until we know exactly which alternative to take before assigning a color.

```
String color = "";
if (x < 10)
{
   color = "red";
}
else
{
   color = "blue";
}
```

Often an alternate strategy (let's call it "act first, decide later") can be used to simplify the logic. If the actions in the true and false statements are reversible, we can go ahead and execute the false (or true) statement and then code an `if` statement that determines whether that action was correct. If the action was incorrect we can reverse it. We solve the problem posed above using this alternative strategy:

```
String color = "blue";
if (x < 10)
{
   color = "red";
}
```

We "act first" by assuming blue is the right color to assign to the variable `color`. We correct it if that was wrong. The logic is simpler and involves coding one less alternative in the `if` statement.

Rewrite the code above again, but this time start by setting the color variable to "red". How does that change the condition?

4) The relational operators in Java are `==`, `!=`, `<`, `>`, `<=`, and `>=`.

Assume x and y are integers. Using relational operators, formulate the following conditions in Java:

a) x is positive
b) x is zero or negative
c) x is at least 10
d) x is less than 10
e) x and y are both zero
f) x is even

5) To compare strings in Java you can't simply use the relational operators. You need to use methods of the `String` class: `equals()`, `compareTo()`, and `substring()`. Assume the following code:

```
String word1 = "catalog";
String word2 = "cat";
```

Write the following conditions in Java:

1) `word1` is lexicographically greater than `"aaa"`
2) `word1` is lexicographically equal to `word2`
3) `word1` is lexicographically less than `word2`
4) `word1` and `word2` have the same three-letter prefix

6) Copy and run the following program. Explain how the program compares the two strings. How can you modify the program so that `str2` and `str3` are equal when they are compared?

```
public class StringEqual
{
   public static void main(String[] args)
   {
      String str1 = "abcd";
      String str2 = "abcdefg";
      String str3 = str1 + "efg";
      System.out.println("str2 = " + str2);
      System.out.println("str3 = " + str3);
      if (str2 == str3)
      {
         System.out.println("The strings are equal");
      }
      else
      {
         System.out.println("The strings are not equal");
      }
   }
}
```

7) Remember the childhood game "Rock, Paper, Scissors"? It is a two-player game in which each person simultaneously chooses either rock, paper, or scissors. Rock beats scissors but loses to paper, paper beats rock but loses to scissors, and scissors beats paper but loses to rock. The following code prompts player 1 and player 2 to each enter a string: rock, paper, or scissors. Finish the code by adding nested `if` statements to appropriately report "Player 1 wins", "Player 2 wins", or "It is a tie."

```
import java.util.Scanner;
public class RockPaperScissors
{
```



```
    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.println("Player 1: Choose rock, scissors, or paper:");
        String player1 = scan.next().toLowerCase();
        System.out.println("Player 2: Choose rock, scissors, or paper:");
        String player2 = scan.next().toLowerCase();
        (your code goes here…)
    }
}
```

8) The program in Lab 3.7 was heavily nested with `if` statements, and it can be difficult to follow the logic of any program that is heavily nested in this way. By constructing complex conditions with the `&&` operator, it is possible to simplify the code and remove some of the `else` alternatives. Rewrite the program in Lab 3.7 using complex conditions and omitting the `else` construct.

9) Draw a flowchart for a program that reads two integers and prints the smaller number.

10.1) Write a program that prompts the user to enter three strings. Compare the `String` objects lexicographically and print the middle-valued string. For example, if the three strings were "abcd", "wxyz", and "pqrs", the program would print "pqrs". Limit yourself to simple, nested `if` statements that don't use the Boolean operators `&&` or `||`. Be sure and test your code by giving it input data that tests every path through your code. Make a list of values for `str1`, `str2`, and `str3` that would thoroughly test the code.

10.2) Rewrite the program solution for Lab 3.10.1 using the Boolean operator `&&` to simplify the logical structure.

11) Programmers take many visual cues from the indenting in a program, so it is imperative that the indentation we provide reflects the logic of the program.

Consider the program below, which is extremely difficult to read because it is so badly indented. Take the program and indent it properly so that the indents reflect the logical structure of the program.

Some programmers adopt a style for `if` statements in which each alternative is always represented as a block of code surrounded by `{}`. There are a couple of advantages to this style:

• the braces clearly indicate the true and false alternatives, and
• the program is easier to maintain if you need to add more lines within one of the alternatives in the future.

After indenting the following code, add `{}` to all the alternatives.

```
public class BadIfs
{
   public static void main(String[] args)
   {
      int x = 9;
      int y = 3;
      int z = 7;
      if (x < y){System.out.println("aaa"); if (x < z)
System.out.println("bbb"); } else
System.out.println("ccc");System.out.println("ddd");  if (y > z)if (z > x)
System.out.println("eee");
else System.out.println("fff");  else
System.out.println("ggg");  }
}
```

Here is the output of the program if you run it as listed:

```
ccc
ddd
ggg
```

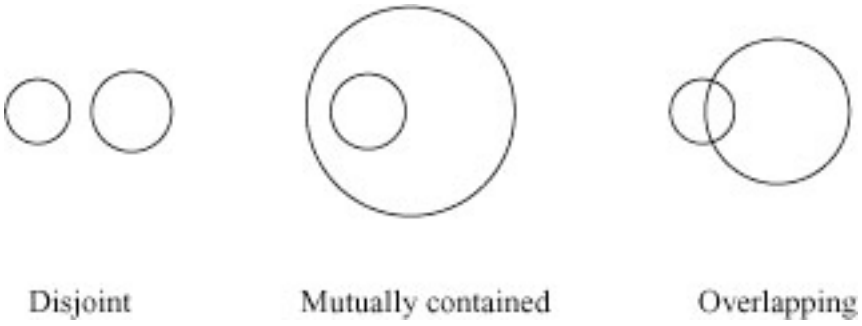
Make sure that the program still produces the same output when you have indented it properly.

12.1) Build and run the following program. What happens when the two points have the same *x*-coordinate?

```
import java.util.Scanner;

public class Slope
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.print("Input x coordinate of the first point: ");
      double xcoord1 = in.nextDouble();

      System.out.print("Input y coordinate of the first point: ");
      double ycoord1 = in.nextDouble();

      System.out.print("Input x coordinate of the second point: ");
      double xcoord2 = in.nextDouble();

      System.out.print("Input y coordinate of the second point: ");
      double ycoord2 = in.nextDouble();

      double slope = (ycoord2 - ycoord1) / (xcoord2 – xcoord1);
      System.out.println("The slope of the line is " + slope);
   }
}
```


12.2) Correct and rebuild the slope program to disallow a vertical line (denominator = 0).

13) Complete the following code to test whether two circles, each having a user-defined radius and a fixed center point lying along the same horizontal line, are disjoint, overlapping, or mutually contained.



Disjoint Mutually contained Overlapping

```
public class CircleOverlap
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.print("Input the radius of the first circle: ");
      double radius1 = in.nextDouble();
      double xcenter1 = 0;
      double ycenter1 = 0;
      System.out.print("Input the radius of the second circle: ");
      double radius2 = in.nextDouble();
      double xcenter2 = 40;
      double ycenter2 = 0;

      // Your work goes here
   }
}
```

14.1) Java has three logical operators, `&&`, `||`, and `!`. Using these operators, express the following:

x and y are both positive or neither of them is positive.

14.2) Using Java's three logical operators, `&&`, `||`, and `!`, formulate the following conditions on the date given by the variables `day` and `month` of type `int`.

a) The date is in the second quarter of the year.
b) The date is the last day of the month. (Assume February always has 28 days.)

c) The date is before April 15.

15) According to the following program, what color results when using the following inputs?

a) `Y N Y`
b) `Y Y N`
c) `N N N`

```
public class ColorMixer
{
   public static void main(String[] args)
   {
      String mixture = "";
      boolean red = false;
      boolean green = false;
      boolean blue = false;

      Scanner in = new Scanner(System.in);
      System.out.print("Include red in mixture? (Y/N) ");
      String input = in.next();
      if (input.toUpperCase().equals("Y"))
      {
         red = true;
      }

      System.out.print("Include green in mixture? (Y/N) ");
      input = in.next();
      if (input.toUpperCase().equals("Y"))
      {
         green = true;
      }

      System.out.print("Include blue in mixture? (Y/N) ");
      input = in.next();
      if (input.toUpperCase().equals("Y"))
      {
         blue = true;
      }

      if (!red && !blue && !green)
      {
         mixture = "BLACK";
      }
      else if (!red && !blue)
      {
         mixture = "GREEN";
      }
      else if (red)
      {
         if (green || blue)
         {
            if (green && blue)
            {
               mixture = "BLACK";
```



```
            }
            else if (green)
            {
               mixture = "YELLOW";
            }
            else
            {
               mixture = "PURPLE";
            }
         }
         else
         {
            mixture = "BLACK";
         }
      }
      else
      {
         if (!green)
         {
            mixture = "BLUE";
         }
         else
         {
            mixture = "WHITE";
         }
      }

      System.out.println("Your mixture is " + mixture);
   }
}
```

