Chapter 5 Name: _____
BJP 3rd Ed per: _____

A fencepost loop executes a "loop-and-a-half" by executing part of a loop's body once before the loop begins.

A sentinel loop is a kind of fencepost loop that repeatedly processes input until it is passed a particular value, but does not process the special value.

The `boolean` primitive type represents logical values of either `true` or `false`. Boolean expressions are used as tests in `if` statements and loops. Boolean expressions can use relational operators such as `<` or `!=` as well as logical operators such as `&&` or `!`.

Complex Boolean tests with logical operators such as `&&` or `||` are evaluated lazily: If the overall result is clear from evaluating the first part of the expression, later parts are not evaluated. This is called short-circuited evaluation.

`Boolean` variables (sometimes called "flags") can store Boolean values and can be used as loop tests.

A complex Boolean expression can be negated using a set of rules known as De Morgan's laws, in which each sub-expression is negated and all AND and OR operations are swapped.

A robust program checks for errors in user input. Better robustness can be achieved by looping and reprompting the user to enter input when he or she types bad input. The `Scanner` class has methods like `hasNextInt` that you can use to "look ahead" for valid input.

Assertions are logical statements about a particular point in a program. Assertions are useful for proving properties about how a program will execute. Two useful types of assertions are preconditions and postconditions, which are claims about what will be true before and after a method executes.

Self-Check Problems

Section 5.1: The `while` Loop

1. For each of the following `while` loops, state how many times the loop will execute its body. Remember that "zero," "infinity," and "unknown" are legal answers. Also, what is the output of the code in each case?

a.
```
int x = 1;
while (x < 100) {
    System.out.print(x + " ");
    x += 10;
}
```
10
1 11 21 31 41 51 61 71 81 91

b.
```
int max = 10;
while (max < 10) {
    System.out.println("count down: " + max);
    max--;
}
```
0

c.
```
int x = 250;
while (x % 3 != 0) {
    System.out.println(x);
}
```
∞
250 250 250 250 250 ...

d.
```
int x = 2;
while (x < 200) {
    System.out.print(x + " ");
    x *= x;
}
```
7
2 4 8 16 32 64 128

e.
```
String word = "a";
while (word.length() < 10) {
    word = "b" + word + "b";
}
System.out.println(word);
```
5
bbbbbabbbbb

f.
```
int x = 100;
while (x > 0) {
    System.out.println(x / 10);
    x = x / 2;
}
```
8
10 5 2 1 0 0 0 0

2. Convert each of the following `for` loops into an equivalent `while` loop:

a.
```
for (int n = 1; n <= max; n++) {
    System.out.println(n);
}
```
int n = 1;
while (n <= max) SOPln(n++);

b.
```
int total = 25;
for (int number = 1; number <= (total / 2); number++) {
    total = total - number;
    System.out.println(total + " " + number);
}
```
int total = 25, number = 1;
while (number <= total/2) SOPln((total -= number) + " " + number++);

c.
```
for (int i = 1; i <= 2; i++) {
    for (int j = 1; j <= 3; j++) {
        for (int k = 1; k <= 4; k++) {
            System.out.print("*");
        }
        System.out.print("!");
    }
    System.out.println();
}
```
int i = 1;
while (i <= 2) {
int j = 1;
while (j <= 3) {
int k = 1;
while (k <= 4) { SOP("*"); k++; }
SOP("!"); j++;
}
SOPln(); i++;
}

d.
```
int number = 4;
for (int count = 1; count <= number; count++) {
    System.out.println(number);
    number = number / 2;
}
```
int count = 1;
while (count <= number) {
SOPln(number);
number = number / 2;
count++;
}

3. Consider the following method:

```
public static void mystery(int x) {
    int y = 1;
    int z = 0;
    while (2 * y <= x) {
        y = y * 2;
```

```
        z++;
    }
    System.out.println(y + " " + z);
}
```

For each of the following calls, indicate the output that the preceding method produces:

```
mystery(1);  // 1 0
mystery(6);  // 4 2
mystery(19); // 16 4
mystery(39); // 32 5
mystery(74); // 64 6
```

$z = \lfloor \log_2 x \rfloor$

$y = 2^z$

4. Consider the following method:

```
public static void mystery(int x) {
    int y = 0;
    while (x % 2 == 0) {
        y++;
        x = x / 2;
    }
    System.out.println(x + " " + y);
}
```

For each of the following calls, indicate the output that the preceding method produces:

```
mystery(19); // 19 0
mystery(42); // 21 1
mystery(48); // 6 3
mystery(40); // 10 2
mystery(64); // 2 5
```

5. Consider the following code:

```
Random rand = new Random();
int a = rand.nextInt(100);      // [0, 100)
int b = rand.nextInt(20) + 50;  // [50, 70)
int c = rand.nextInt(20 + 50);  // [0, 70)
int d = rand.nextInt(100) - 20; // [-20, 80)
int e = rand.nextInt(10) * 4;   // [0, 40)
```

What range of values can each variable (a, b, c, d, and e) have?

6. Write code that generates a random integer between 0 and 10 inclusive. `rand.nextInt(11)`

7. Write code that generates a random odd integer (not divisible by 2) between 50 and 99 inclusive. `rand.nextInt(25) * 2 + 51`

8. For each of the `do/while` loops that follow, state the number of times that the loop will execute its body. Remember that "zero," "infinity," and "unknown" are legal answers. Also, what is the output of the code in each case?

No, it's not

a.
```
int x = 1;
do {
```

```
        System.out.print(x + " ");
        x = x + 10;
    } while (x < 100);
```

1 11 21 31 41 51 61 71 81 91
10

```
b. int max = 10;
   do {
        System.out.println("count down: " + max);
        max--;
   } while (max < 10);
```

count down: 10
count down: 9
count down: 8
⋮
Infinite

```
c. int x = 250;
   do {
        System.out.println(x);
   } while (x % 3 != 0);
```

1 250

```
d. int x = 100;
   do {
        System.out.println(x);
        x = x / 2;
   } while (x % 2 == 0);
```

100 50
2

```
e. int x = 2;
   do {
        System.out.print(x + " ");
        x *= x;
   } while (x < 200);
```

2 4 8 16 32 64 128
7

```
f. String word = "a";
   do {
        word = "b" + word + "b";
   } while (word.length() < 10);
   System.out.println(word);
```

bbbbabbbbb
5

```
g. int x = 100;
   do {
        System.out.println(x / 10);
        x = x / 2;
   } while (x > 0);
```

10 5 2 1
4

```
h. String str = "/\\";
   do {
        str += str;
   } while (str.length() < 10);
   System.out.println(str);
```

/\/\/\/\
5

9. Write a do/while loop that repeatedly prints a certain message until the user tells the program to stop. The do/while is appropriate because the message should always be printed at least one time, even if the user types n after the first message appears. The message to be printed is as follows:

```
She sells seashells by the seashore.
Do you want to hear it again? y
She sells seashells by the seashore.
Do you want to hear it again? y
```

```
do {
    SOP("She sells seashells by the seashore");
    SOP("Do you want to hear it again?");
}
while (!console.next().equals("n"))
```

```
She sells seashells by the seashore.
Do you want to hear it again? n
```

10. Write a method called `zeroDigits` that accepts an integer parameter and returns the number of digits in the number that have the value `0`. For example, the call `zeroDigits(5024036)` should return `2`, and `zeroDigits(743)` should return `0`. The call `zeroDigits(0)` should return `1`. (We suggest you use a `do/while` loop in your solution.)

11. Write a `do/while` loop that repeatedly prints random numbers between `0` and `1000` until a number above `900` is printed. At least one line of output should always be printed, even if the first random number is above `900`. Here is a sample execution:

```
Random number: 235
Random number: 15
Random number: 810
Random number: 147
Random number: 915
```

Section 5.2: Fencepost Algorithms

12. Consider the flawed method `printLetters` that follows, which accepts a `String` as its parameter and attempts to print the letters of the `String`, separated by dashes. For example, the call of `printLetters("Rabbit")` should print `R-a-b-b-i-t`. The following code is incorrect:

```
public static void printLetters(String text) {
    for (int i = 0; i < text.length(); i++) {
        System.out.print(text.charAt(i) + "-");
    }
    System.out.println(); // to end the line of output
}
```

What is wrong with the code? How can it be corrected to produce the desired behavior?

13. Write a sentinel loop that repeatedly prompts the user to enter a number and, once the number `-1` is typed, displays the maximum and minimum numbers that the user entered. Here is a sample dialogue:

```
Type a number (or -1 to stop): 5
Type a number (or -1 to stop): 2
Type a number (or -1 to stop): 17
Type a number (or -1 to stop): 8
Type a number (or -1 to stop): -1
Maximum was 17
Minimum was 2
```

If `-1` is the first number typed, no maximum or minimum should be printed. In this case, the dialogue would look like this:

```
Type a number (or -1 to stop): -1
```

Section 5.3: The boolean Type

14. Consider the following variable declarations:

```
int x = 27;
int y = -1;
```

```
int x = 27;
int y = -1;
int z = 32;
boolean b = false;
```

What is the value of each of the following Boolean expressions?

a. `!b`
b. `b || true`
c. `(x > y) && (y > z)`
d. `(x == y) || (x <= z)`
e. `!(x % 2 == 0)`
f. `(x % 2 != 0) && b`
g. `b && !b`
h. `b || !b`
i. `(x < y) == b`
j. `!(x / 2 == 13) || b || (z * 3 == 96)`
k. `(z < x) == false`
l. `!((x > 0) && (y < 0))`

```
public static boolean isVowel (char ch) {
    return "aeiou".indexOf(Character.toLowerCase(ch)) != -1;
}
```

15. Write a method called `isVowel` that accepts a character as input and returns `true` if that character is a vowel (a, e, i, o, or u). For an extra challenge, make your method case-insensitive.

16. The following code attempts to examine a number and return whether that number is prime (i.e., has no factors other than 1 and itself). A flag named `prime` is used. However, the Boolean logic is not implemented correctly, so the method does not always return the correct answer. In what cases does the method report an incorrect answer? How can the code be changed so that it will always return a correct result?

```
public static boolean isPrime(int n) {
    boolean prime = true;
    for (int i = 2; i < n; i++) {
        if (n % i == 0) {
            prime = false;
        } else {
            prime = true;
        }
    }
    return prime;
}
```

17. The following code attempts to examine a `String` and return whether it contains a given letter. A flag named `found` is used. However, the Boolean logic is not implemented correctly, so the method does not always return the correct answer. In what cases does the method report an incorrect answer? How can the code be changed so that it will always return a correct result?

```
public static boolean contains(String str, char ch) {
    boolean found = false;
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == ch)
    return true;
```

```
    } else {
        found = false;
    }
  }
  return found;
}
```

18. Using "Boolean Zen," write an improved version of the following method, which returns whether the given `String` starts and ends with the same character:

```
public static boolean startEndSame(String str) {
  return str.charAt(0) == str.charAt(str.length() - 1);
}
```

19. Using "Boolean Zen," write an improved version of the following method, which returns whether the given number of cents would require any pennies (as opposed to being an amount that could be made exactly using coins other than pennies):

```
public static boolean hasPennies(int cents) {
  boolean nickelsOnly = (cents % 5 == 0);
  if (nickelsOnly == true) {
    return false;
  } else {
    return true;
  }
}
```

20. Consider the following method:

```
public static int mystery(int x, int y) {
  while (x != 0 && y != 0) {
    if (x < y) {
      y -= x;
    } else {
      x -= y;
    }
  }
  return x + y;
}
```

For each of the following calls, indicate the value that is returned:

```
mystery(3, 3)    3
mystery(5, 3)    1
mystery(2, 6)    2
```

```
mystery(12, 18)
mystery(30, 75)
```

21. The following code is a slightly modified version of actual code that was in the Microsoft Zune music player in 2008. The code attempts to calculate today's date by determining how many years and days have passed since 1980. Assume the existence of methods for getting the total number of days since 1980 and for determining whether a given year is a leap year:

```
int days = getTotalDaysSince1980();
year = 1980;
while (days > 365) { // subtract out years
    if (isLeapYear(year)) {
        if (days > 366) {
            days -= 366;
            year += 1;
        }
    } else {
        days -= 365;
        year += 1;
    }
}
```

Thousands of Zune players locked up on January 1, 2009, the first day after the end of a leap year since the Zune was released. (Microsoft quickly released a patch to fix the problem.) What is the problem with the preceding code, and in what cases will it exhibit incorrect behavior? How can it be fixed?

22. Consider the following variable declarations:

```
int x = 27;
int y = -1;
int z = 32;
boolean b = false;
```

Write a new Boolean expression that is the negation of each of the following Boolean expressions. Use De Morgan's laws rather than simply writing a `!` at the beginning of each entire expression.

a. `b`
b. `(x > y) && (y > z)`
c. `(x == y) || (x <= z)`
d. `(x % 2 != 0) && b`
e. `(x / 2 == 13) || b || (z * 3 == 96)`
f. `(z < x) && (z > y || x >= y)`

Section 5.4: User Errors

23. The following code is not robust against invalid user input. Describe how to change the code so that it will not proceed until the user has entered a valid age and grade point average (GPA). Assume that any `int` is a legal age and that any `double` is a legal GPA.

```
Scanner console = new Scanner(System.in);
System.out.print("Type your age: ");
int age = console.nextInt();
```

```
System.out.print("Type your GPA: ");
double gpa = console.nextDouble();
```

For an added challenge, modify the code so that it rejects invalid ages (for example, numbers less than 0) and GPAs (say, numbers less than 0.0 or greater than 4.0).

24. Consider the following code:

```
Scanner console = new Scanner(System.in);
System.out.print("Type something for me! ");

if (console.hasNextInt()) {
    int number = console.nextInt();
    System.out.println("Your IQ is " + number);
} else if (console.hasNext()) {
    String token = console.next();
    System.out.println("Your name is " + token);
}
```

What is the output when the user types the following values?

a. Jane
b. 56
c. 56.2

25. Write a piece of code that prompts the user for a number and then prints a different message depending on whether the number was an integer or a real number. Here are two sample dialogues:

```
Type a number: 42.5
You typed the real number 42.5

Type a number: 3
You typed the integer 3
```

26. Write code that prompts for three integers, averages them, and prints the average. Make your code robust against invalid input. (You may want to use the `getInt` method discussed in this chapter.)

Section 5.5: Assertions and Program Logic

27. Identify the various assertions in the following code as being always true, never true, or sometimes true and sometimes false at various points in program execution. The comments indicate the points of interest:

```
public static int mystery(Scanner console, int x) {
    int y = console.nextInt();
    int count = 0;

    // Point A
    while (y < x) {
        // Point B
        if (y == 0) {
```

```
            count++;
            // Point C
        }
        y = console.nextInt();
        // Point D
    }

    // Point E
    return count;
}
```

Categorize each assertion at each point with ALWAYS, NEVER, or SOMETIMES:

	y < x	y == 0	count > 0
Point A	S	S	N
Point B	A	S	S
Point C	A	A	A
Point D	S	S	A
Point E	N	S	S

28. Identify the various assertions in the following code as being always true, never true, or sometimes true and sometimes false at various points in program execution. The comments indicate the points of interest:

```
public static int mystery(int n) {
    Random r = new Random();
    int a = r.nextInt(3) + 1;
    int b = 2;
    // Point A
    while (n > b) {
        // Point B
        b = b + a;
        if (a > 1) {
            n--;
            // Point C
            a = r.nextInt(b) + 1;
        } else {
            a = b + 1;

            // Point D
        }
    }
    // Point E
    return n;
}
```

Categorize each assertion at each point with ALWAYS, NEVER, or SOMETIMES:

	`n > b`	`a > 1`	`b > a`
Point A	S	S	S
Point B	A	S	S
Point C	S	A	A
Point D	S	A	A
Point E	N	S	S

29. Identify the various assertions in the following code as being always true, never true, or sometimes true and sometimes false at various points in program execution. The comments indicate the points of interest:

```
public static int mystery(Scanner console) {
    int prev = 0;
    int count = 0;
    int next = console.nextInt();
    // Point A
    while (next != 0) {
        // Point B
        if (next == prev) {
            // Point C
            count++;
        }
        prev = next;
        next = console.nextInt();
        // Point D
    }
    // Point E
    return count;
}
```

Categorize each assertion at each point with ALWAYS, NEVER, or SOMETIMES:

	`next == 0`	`prev == 0`	`next == prev`
Point A	S	A	S
Point B	N	S	S
Point C	N	S	A
Point D	S	N	S
Point E	A	S	S