Introduction to Java
Notes Packet #3

Name: ____

Objective: By the completion of this packet, students should be able to write loops.

The Modulus Operator.

The modulus operator (a.k.a. the remainder operator) is the percent sign (%). It is used to find the remainder of a division problem. For example:

```
int x = 14 % 5;      // 4
int y = 24 % 6;      // 0
int z = 8 % 10;      // 8
```

In some circumstances, the modulus operator turns out to be very useful. For example, if you need to know if one number is evenly divisible by another, use this code:

```
if ( num1 % num2 == 0 )      // then num2 is a factor of num1
```

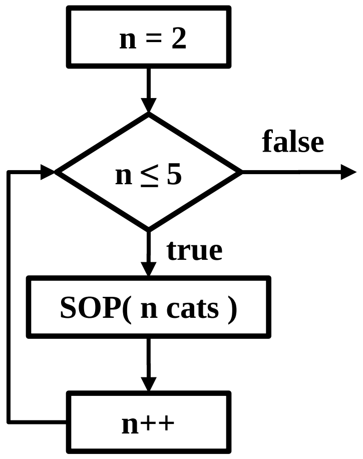
Example 1. Suppose a store sells soft pretzels for 50 cents each and $5 for a dozen. The code below calculates the cost on *n* pretzels.

```
int n;          // n represents the number of pretzels being bought
// code that assigns n a value
double cost = 5 * ( n / 12) + 0.5 * ( n % 12 );
```

While Loops.

A while loop is a control structure that allows you to write code that is executed repeatedly as long as some condition is true.

Java Code	**Flowchart**

`int n = 2;`
 	
`while( n <= 5 ) {`
`    System.out.println(n + "cats");`
`    n++;`
`}`
	

Every pass through the body of a loop is called an **iteration**.

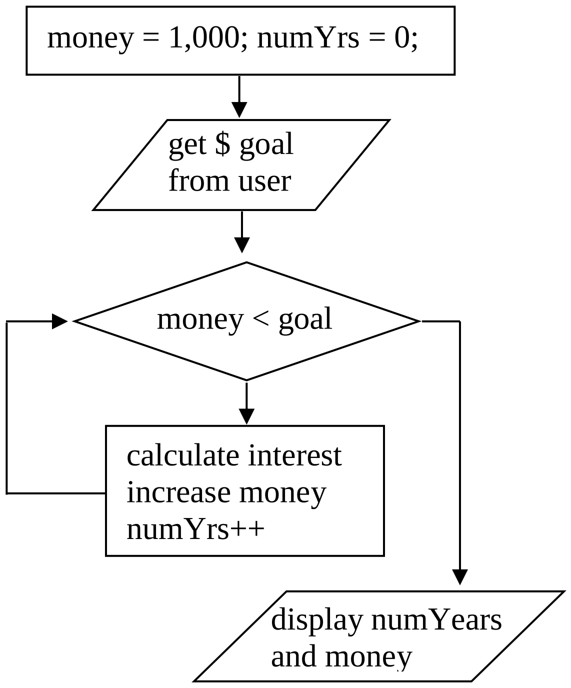
```
double money = 1000.0;
int numYears = 0;

Scanner get = new Scanner( System.in );
System.out.print( "Enter your goal: " );
double goal = get.nextDouble();

while ( money < goal ) {
      double interest = 0.1*money;
      money += interest;
      numYears++;
}

System.out.println( numYears + " years ");
System.out.println( money + " dollars ");
```

You have $1,000 and a 10% interest rate. Calculate how many years must pass until your money has reached (or surpassed) some goal



money = 1,000; numYrs = 0;
get $ goal from user
money < goal
calculate interest increase money numYrs++
display numYears and money

There are two basic types of while loops: task/event-oriented and count-oriented.

Task/Event-Oriented While Loop. This kind of loop continues until some task is completed or some event occurs. For example:

```
Scanner read = new Scanner( System.in );
int num = 0;
int count = 0;
int total = 0;
while ( num >= 0 ) {
                System.out.print("Enter a number ");
                num = read.nextInt();
                if ( num >= 0 ) {
                        total = total + num;
                        count++;
                }
}
System.out.println( "The " + count + " numbers add up to " + total )  ;
```

Keep looping while num is not negative.

As long as num is not negative, add the number to the total and increase count by 1.

If the user enters 3, 5, and -2, what is displayed? **The 2 numbers add up to 8**

Count-Oriented While Loop. This kind of loop continues for a specific number of **iterations** and then stops. For example:

```
int n = 1;              n is a “counter” because it keeps count of the iterations.
while ( n <= 5 ) {      Keep looping while this is true
    System.out.println("Hello");   Body of the loop
    n++;                The last statement changes the counter
}
```

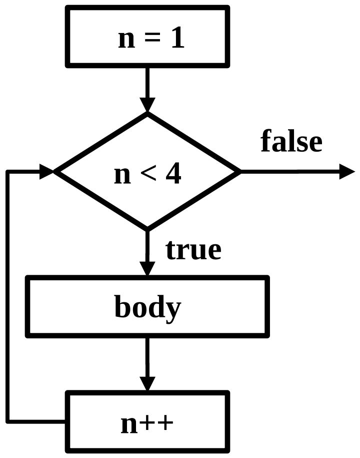
A count-oriented loop may count forward or backward. It may count in steps of 1 or any other value.

For Loops. A for-loop is typically used as an alternative to a count-oriented while loop.

The first statement in a for-loop contains three statements separated by semicolons:

1. **Initialize, and usually declare, the counter (which is usually an int).**
2. **Boolean expression involving the counter; keep looping while true.**
3. **Update the counter. This is executed at the end of each iteration.**

Java Code	**Flowchart**

`for ( int n = 1; n < 4; n++) {`
`      // the body of the loop`
`}`
	

Example 2. What does this loop display?	**Example 3**. What does this loop display?
5, 6, 7, 8,	**25, 18, 11,**

`for ( int k = 5; k <= 8; k++ ){`
`        System.out.print( k + ", " );`
`}`
	`for ( int n = 25; n >= 10; n = n - 7 ){`
	`        System.out.print( n + ", " );`
	`}`