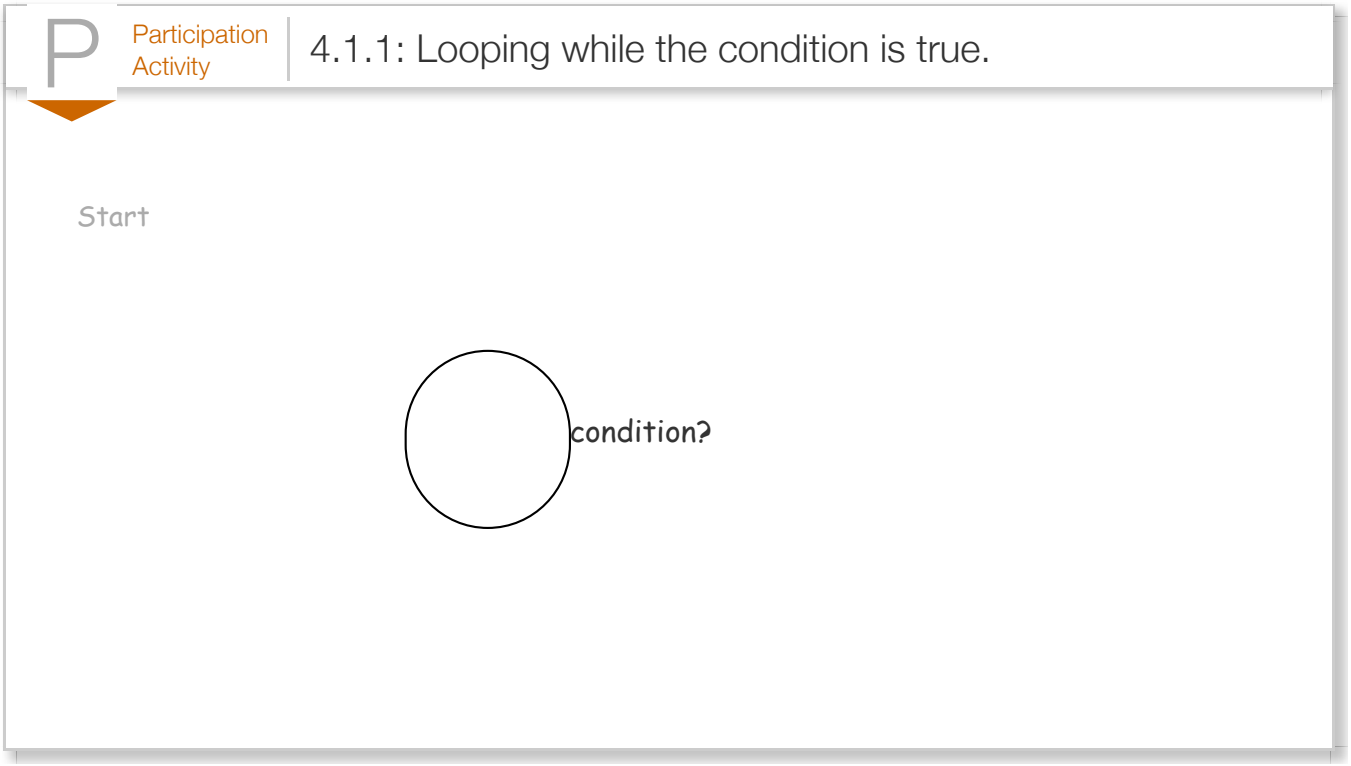
Chapter 4 - Loops

Section 4.1 - Loops

Participation Activity

4.1.1: Looping while the condition is true.



Some behaviors should be repeated over and over, like a racecar driving around a track. A ***loop*** is a construct that repeatedly executes specific code as long as some condition is true.

Participation Activity

4.1.2: Loop basics.

Which loop condition achieves the given racetrack driving goal?

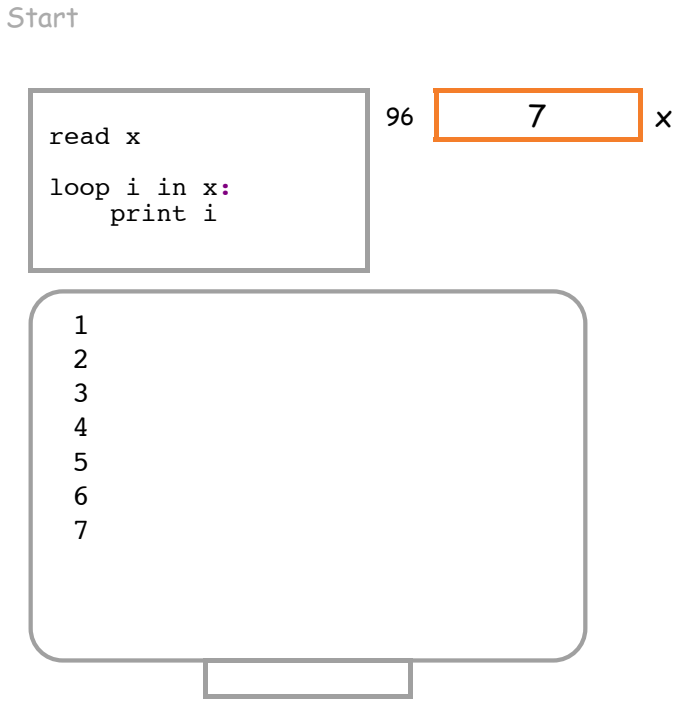
#	Question	Your answer
1	Loop as long as it is sunny.	It is sunny.
		It is not sunny.
2	Loop as long as it is not raining.	It is raining.
		It is not raining.
3	Loop 3 times.	Number of completed laps is 0 or greater.
		Number of completed laps is less than 3.
		Number of completed laps equals 3.
4	Loop while the car's fuel tank is at least 20% full.	Fuel tank is at 20%.
		Fuel tank is 20% or more.
		Fuel tank is less than 20%.

The above describes a common kind of loop known as a *while* loop.

Below is a loop (in no particular language) that prints a value a specified number of times.

Participation Activity

4.1.3: Loop a given number of times.



Section 4.2 - While loops

A ***while loop*** is a program construct that executes a list of sub-statements repeatedly as long as the loop's expression evaluates to true.

Construct 4.2.1: While loop statement general form.

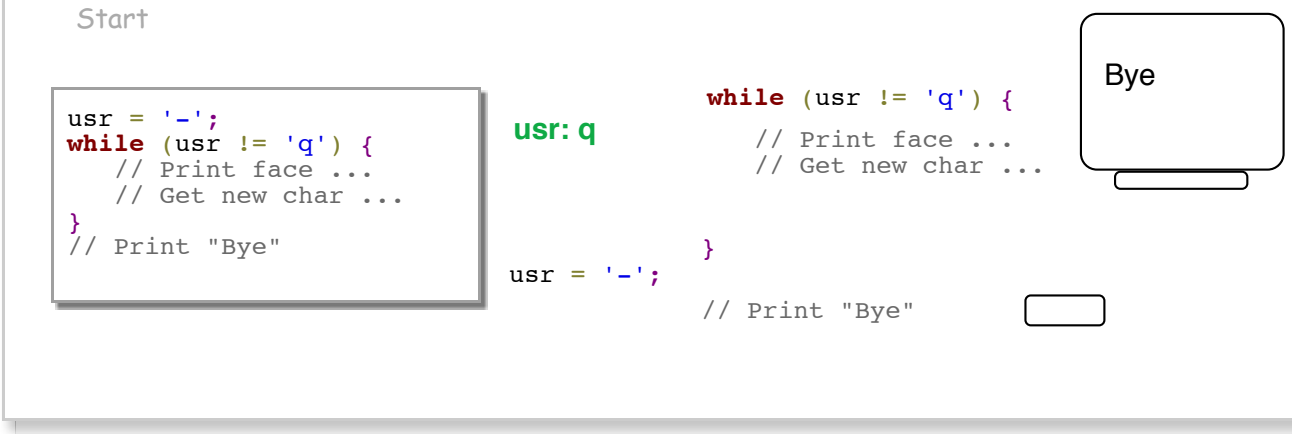
```
while (expression) { // Loop expression
    // Loop body: Sub-statements that execute if the
    // expression evaluated to true
}
// Statements that execute after the expression evaluates to false
```

When execution reaches the while loop statement, the expression is evaluated. If true, execution proceeds into the sub-statements inside the braces, known as the ***loop body***. At the loop body's

end, execution goes back to the while loop statement start. The expression is again evaluated, and if true, execution again proceeds into the loop body. But if false, execution instead proceeds past the closing brace. Each execution of the loop body is called an ***iteration***, and looping is also called *iterating*.

Participation Activity

4.2.1: While loop.



Participation Activity

4.2.2: Basic while loops.

How many times will the loop body execute?

#	Question	Your answer

1	`x = 3;`
	`while (x >= 1) {`
	`  // Do something`
	`  x = x - 1;`
	`}`	

2	Assume user would enter 'n', then 'n', then 'y'.
	`// Get userChar from user here`
	`while (userChar != 'n') {`
	`  // Do something`
	`  // Get userChar from user here`
	`}`	

3	Assume user would enter 'a', then 'b', then 'n'.
	`// Get userChar from user here`
	`while (userChar != 'n') {`
	`  // Do something`
	`  //Get userChar from user here`
	`}`	

The following example uses the statement `while (userChar != 'q') { }` to allow a user to end a face-drawing program by entering the character q:

Figure 4.2.1: While loop example: Face-printing program that ends when user enters 'q'.

```
import java.util.Scanner;

public class FacePrint {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      char userChar = '-'; // User-entered character
      String usrInput = "";

      while (userChar != 'q') {
         // Print face
         System.out.println("" + userChar + " " + userChar);
         System.out.println(" " + userChar);
         System.out.println("" + userChar + "" + userChar + "" + us

         // Get user character
         System.out.print("\nEnter a character ('q' to quit): ");
         usrInput = scnr.next();
         userChar = usrInput.charAt(0); // Get the first char in th
         System.out.println("");
      }

      System.out.println("Goodbye.");

      return;
   }
}
```

```
- -
 -
---

Enter a characte

a a
 a
aaa

Enter a characte

x x
 x
xxx

Enter a characte

Goodbye.
```

The Scanner does not directly support reading a single character. The above program first reads a string from the user input using `usrInput = scnr.next();`. The first character within that string is then stored into userChar using `userChar = usrInput.charAt(0);`.

Once execution enters the loop body, execution continues to the body's end even if the expression becomes false midway through.

Participation Activity

4.2.3: Loop expressions.

Use a *single operator* in each expression, and the most straightforward translation of the stated goal into an expression.

#	Question	Your answer
1	Iterate while x is less-than 100.	`while ( ) { /* Loop body statements go here * }`
2	Iterate while x is greater than or equal to 0.	`while ( ) { // Loop body }`
3	Iterate while c equals 'g'.	`while ( ) { // Loop body }`
4	Iterate while c is not equal to 'x'.	`while ( ) { // Loop body }`
5	Iterate *until* c equals 'z' (tricky; think carefully).	`while ( ) { // Loop body }`

Below is a simple loop example, which separately prints each digit of an integer, showing each iteration.

Participation Activity

4.2.4: While loop step-by-step

Start

```
...
// Read num from user ...

   // Print each digit
   while (num > 0) {
      // Print  num % 10 ...
      num = num / 10;
   }
```

Iteration	num	Output
	902	
1		2
	90	
2		0
	9	
3		9
	0	

Printing out every digit of a user-entered number using the % and // operators.

Below is another loop example. The program asks the user to enter a year, and then prints the approximate number of a person's ancestors who were alive for each generation leading back to that year, with the loop computing powers of 2 along the way.

Figure 4.2.2: While loop example: Ancestors printing program..

```
import java.util.Scanner;

public class AncestorsPrinter {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int YEARS_PER_GEN = 20; // Approx. years per generation
      int userYear = 0;             // User input
      int consYear = 0;             // Year being considered
      int numAnc = 0;               // Approx. ancestors in considered

      System.out.print("Enter a past year (neg. for B.C.): ");
      userYear = scnr.nextInt();

      consYear = 2020;
      numAnc = 2;

      while (consYear >= userYear) {
         System.out.println("Ancestors in " + consYear + ": " + numAnc

         numAnc = 2 * numAnc;                  // Each ancestor had two
         consYear = consYear - YEARS_PER_GEN; // Go back 1 generation
      }

      return;
   }
}
```

```
Enter a past
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in

...

Enter a past
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
Ancestors in
```

Each iteration prints a line with the year and the ancestors in that year. (Note: the numbers are large due to not considering breeding among distant relatives, but nevertheless a person has many ancestors).

The program checks for `consYear >= userYear` rather than for `consYear != userYear`, because consYear might be decreased past userYear without equaling it, causing an infinite loop, printing years well past 1950. An ***infinite loop*** is a loop that will always execute (i.e., execute infinitely) because the loop's expression always evaluates to true. A common error is to accidentally create an infinite loop due to assuming equality will be reached. Good practice is to include greater-than or less-than along with equality in a loop expression.

Another common error is to use the assignment operator = rather than the equality operator == in a loop expression, resulting in a compilation error.

A program with an infinite loop may print excessively, or just seem to stall. On some systems, the user can halt execution by pressing Control-C on the command prompt, or by selecting Stop (or Pause) from within an IDE.

Participation Activity

4.2.5: While loop iterations.

What will the following code output? (For an infinite loop, type "IL")

#	Question	Your answer

1	`int x = 0;`
	`while (x > 0) {`
	`  System.out.print(x + " ");`
	`  x = x - 1;`
	`}`
	`System.out.print("Bye");`	

2	`int x = 5;`
	`int y = 18;`
	`while (y >= x) {`
	`  System.out.print(y + " ");`
	`  y = y - x;`
	`}`	

3	`int x = 10;`
	`while (x != 3) {`
	`  System.out.print(x + " ");`
	`  x = x / 2;`
	`}`	

4	`int x = 0;`
	`while (x <= 5) {`
	`  System.out.print(x + " ");`
	`}`	

Participation Activity

4.2.6: Range of data types.

Computing in loops can easily exceed a variable's range. Execute the ancestors program below with the given input of 1300. What do you observe around year 1400? Recall that an int variable can usually only represent up to about 2 billion. Try changing the definition of numAnc from type int to long, and then see how distant of a year you can enter before observing incorrect output.

```
1
2  import java.util.Scanner;
3
4  public class AncestorsPrinter {
5     public static void main (String [] args) {
6        Scanner scnr = new Scanner(System.in);
7        final int YEARS_PER_GEN = 20; // Approx. years per
8        int userYear = 0;             // User input
9        int consYear = 0;             // Year being consid
10       int numAnc = 0;               // Approx. ancestors
11
12       System.out.print("Enter a past year (neg. for B.C.
13       userYear = scnr.nextInt();
14
15       consYear = 2020;
16       numAnc = 2;
17       while (consYear >= userYear) {
18          System.out.println("Ancestors in " + consYear +
19
```

```
1300
```

Run

Challenge Activity

4.2.1: Enter the output for the while loop.

Start

Enter the output of the following program.

```
public class whileLoopOutput {
   public static void main (String [] args) {
      int g = 0;

      while (g <= 3) {
         System.out.print(g);
         g = g + 1;
      }

      return;
   }
}
```

0123

1	2	3	4	5

Check Next

Challenge Activity

4.2.2: Basic while loop with user input.

Write an expression that executes the loop body as long as the user enters a non-negative number.

Note: These activities may test code with different test values. This activity will perform three tests, wi
of 5, 2, -1, then with userNum initially 0 and user input of -17, then with userNum initially -1. See Hov

Also note: If the submitted code has an infinite loop, the system will stop running the code after a few
never reached." The system doesn't print the test case that caused the reported message.

```
1  import java.util.Scanner;
2
3  public class NonNegativeLooper {
4     public static void main (String [] args) {
5        Scanner scnr = new Scanner(System.in);
6        int userNum = 0;
7
8        userNum = 9;
9        while (/* Your solution goes here  */) {
10          System.out.println("Body");
11          userNum = scnr.nextInt();
12       }
13       System.out.println("Done.");
14
15       return;
16    }
17 }
```

Run

Challenge Activity

4.2.3: Basic while loop expression.

Write a while loop that prints userNum divided by 2 (integer division) until reaching 1. Follow each nur
userNum = 20:

```
20 10 5 2 1
```

Note: These activities may test code with different test values. This activity will perform four tests, witl
= 1, then with userNum = 0, then with userNum = -1. See How to Use zyBooks.

Also note: If the submitted code has an infinite loop, the system will stop running the code after a few
never reached." The system doesn't print the test case that caused the reported message.

```
1  import java.util.Scanner;
2
3  public class DivideByTwoLoop {
4     public static void main (String [] args) {
5        int userNum = 0;
6
7        userNum = 20;
8
9        /* Your solution goes here  */
10
11       System.out.println("");
12
13       return;
14    }
15 }
```

Run

Section 4.3 - More while examples

The following is an example of using a loop to compute a mathematical quantity. The program computes the greatest common divisor (GCD) among two user-entered integers numA and numB,

using Euclid's algorithm: If numA > numB, set numA to numA - numB, else set numB to numB - numA. These steps are repeated until numA equals numB, at which point numA and numB each equal the GCD.

Figure 4.3.1: While loop example: GCD program.

```
import java.util.Scanner;

// Output GCD of user-input numA and numB

public class GCDCalc {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int numA = 0; // User input
      int numB = 0; // User input

      System.out.print("Enter first positive integer:
      numA = scnr.nextInt();

      System.out.print("Enter second positive integer
      numB = scnr.nextInt();

      while (numA != numB) { // Euclid's algorithm
         if (numB > numA) {
            numB = numB - numA;
         }
         else {
            numA = numA - numB;
         }
      }

      System.out.println("GCD is: " + numA);

      return;
   }
}
```

```
Enter first positive integer:
Enter second positive integer
GCD is: 1

...

Enter first positive integer:
Enter second positive integer
GCD is: 5

...

Enter first positive integer:
Enter second positive integer
GCD is: 33

...

Enter first positive integer:
Enter second positive integer
GCD is: 500
```

Participation Activity

4.3.1: GCD program.

Refer to the GCD code provided in the previous figure. Assume user input of numA = 15 and numB = 10.

#	Question	Your answer
1	For the GCD program, what is the value of numA *before* the first loop iteration?	
2	What is the value of numB *after* the first iteration of the while loop?	
3	What is numB after the second iteration of the while loop?	
4	How many loop iterations will the algorithm execute?	

Below is a program that has a "conversation" with the user, asking the user to type something and then (randomly) printing one of four possible responses until the user enters "Goodbye":

Figure 4.3.2: While loop example: Conversation program.

```
import java.util.Scanner;

/* Program that has a conversation with the user. Uses a switch statement
   and a random number (sort of) to mix up the program's responses. */

public class Conversation {

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int randNum0_3 = 0;   // Random number 0 to 3
      String userText = ""; // User input

      System.out.print("Tell me something about yourself. ");
```

```
        System.out.println("You can type \"Goodbye\" at anytime to quit.\n");
        System.out.print("> ");

        userText = scnr.nextLine();

        while (!userText.equals("Goodbye")) {
           randNum0_3 = userText.length() % 4; // "Random" num. %4 ensures 0-3
           switch (randNum0_3) {
              case 0:
                 System.out.println("\nPlease explain further.\n");
                 System.out.print("> ");
                 break;

              case 1:
                 System.out.println("\nWhy do you say: \"" + userText + "\"?\n");
                 System.out.print("> ");
                 break;

              case 2:
                 System.out.println("\nI don't think that's right.\n");
                 System.out.print("> ");
                 break;

              case 3:
                 System.out.println("\nWhat else can you share?\n");
                 System.out.print("> ");
                 break;

              default:
                 System.out.println("\nUh-oh, something went wrong. Try again.\n");
           }

           userText = scnr.nextLine();
        }

        System.out.println("\nIt was nice talking with you. Goodbye.\n");

        return;
     }
  }
```

```
Tell me something about yourself. You can type "Goodbye" at anytime to quit.

> I'm 26 years old.

Why do you say: "I'm 26 years old."?

> Well, I was born 26 years ago.

I don't think that's right.

> I am sure it is correct.

Please explain further.

> Goodbye

It was nice talking with you. Goodbye.
```

The loop checks whether userText is "Goodbye"; if not, the loop body executes. The loop body generates a "random" number between 0 and 3, by getting the length of the user's text (which is sort of random) and mod'ing by 4. The loop body then prints one of four messages, using a switch statement (if you haven't studied switch, think of switch like an if-else statement).

Participation Activity

4.3.2: Conversation program.

#	Question	Your answer
1	What will be printed if the user types "Ouch"?	
2	What will be printed if the user types "Bye"?	
3	Which switch branch will execute if the user types "Goodbye"? Valid answers are branch 0, 1, 2, 3, or none.	
4	How many loop iterations will execute if the user plans to type "I'm hungry", "You are weird", "Goodbye", and "I like you".	

Challenge Activity

4.3.1: Bidding example.

Write an expression that continues to bid until the user enters 'n'.

```
4  public class AutoBidder {
5     public static void main (String [] args) {
6        Scanner scnr = new Scanner(System.in);
7        Random randGen = new Random();
8        char keepGoing = '-';
9        int nextBid = 0;
10
11       randGen.setSeed(5);
12       while (/* Your solution goes here  */) {
13          nextBid = nextBid + (randGen.nextInt(10) + 1);
14          System.out.println("I'll bid $" + nextBid + "!");
15          System.out.print("Continue bidding? ");
16          keepGoing = scnr.next().charAt(0);
17       }
18       System.out.println("");
19
20       return;
21    }
22 }
```

Run

Challenge Activity

4.3.2: While loop: Insect growth.

Given positive integer numInsects, write a while loop that prints that number doubled without reachin space. After the loop, print a newline. Ex: If numInsects = 8, print:

```
8 16 32 64
```

```
1  import java.util.Scanner;
2
3  public class InsectGrowth {
4     public static void main (String [] args) {
5        int numInsects = 0;
6
7        numInsects = 8; // Must be >= 1
8
9        /* Your solution goes here  */
10
11       return;
12    }
13 }
```

Run

Section 4.4 - Counting

Commonly, a loop should iterate a specific number of times, such as 10 times. A ***loop variable*** counts the number of iterations of a loop. To iterate N times using an integer loop variable i, a while loop[Note_whileloops] with the following form is used:

Construct 4.4.1: Loop variable to iterate N times.

```
// Iterating N times using loop variable i
i = 1;
while (i <= N) {
   // Loop body
   i = i + 1;
}
```

For example, the following program outputs the amount of money in a savings account each year for the user-entered number of years, with $10,000 initial savings and 5% yearly interest:

Figure 4.4.1: While loop that counts iterations: Savings interest program.

```
import java.util.Scanner;

public class SavingsInterestCalc {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int INIT_SAVINGS = 10000;     // Initial savings
      final double INTEREST_RATE = 0.05; // Interest rate
      int userYears = 0;                  // User input of number of
      int i = 0;                          // Loop variable
      double currSavings = 0.0;           // Savings with interest

      System.out.println("Initial savings of $" + INIT_SAVINGS);
      System.out.println("at " + INTEREST_RATE + " yearly interest.\

      System.out.print("Enter years: ");
      userYears = scnr.nextInt();

      currSavings = INIT_SAVINGS;
      i = 1;
      while (i <= userYears) {
         System.out.println(" Savings in year " + i
                            + ": $" + currSavings);
         currSavings = currSavings + (currSavings * INTEREST_RATE);

         i = i + 1;
      }

      System.out.println();

      return;
   }
}
```

```
Initial savings
at 0.05 yearly

Enter years: 5
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea

...

Initial savings
at 0.05 yearly

Enter years: 15
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
 Savings in yea
```

The statements that cause iteration to occur userYears times are highlighted.

A common error is to forget to include the loop variable update (`i = i + 1`) at the end of the loop,

causing an unintended infinite loop.

Participation Activity

4.4.1: Basic while loop parts.

Use <= in each loop expression.

#	Question	Your answer

1	Loop iterates 10 times.	`i = 1;`
		`while (` ____ `) {`
		`  // Loop body`
		`  i = i + 1;`
		`}`

2	Loop iterates 2 times.	`i = 1;`
		`while (` ____ `) {`
		`  // Loop body`
		`  i = i + 1;`
		`}`

3	Loop iterates 8 times. NOTE the initial value of i.	`i = 0;`
		`while (` ____ `) {`
		`  // Loop body`
		`  i = i + 1;`
		`}`

Counting down is also common, such as counting from 5 to 1, as below.

Figure 4.4.2: While loop with variable that counts down.

```
i = 5;
while (i >= 1) {
   // Loop body
   i = i - 1;
}
```

The loop body executes when i is 5, 4, 3, 2, and 1, but does not execute when i reaches 0.

Counting is sometimes done by steps greater than 1, such as a loop that prints even values from 0 to 100 (0, 2, 4, 6, ..., 98, 100), as below.

Figure 4.4.3: Loop variable increased by 2.

```
i = 0;
while (i <= 100) {
   // Loop body
   i = i + 2;
}
```

Note that the loop variable update is `i = i + 2;` rather than `i = i + 1;`

Creating the loop variable initialization, expression, and loop variable update to achieve specific goals is an important skill.

Participation Activity

4.4.2: Loop to print presidential election years.

Modify the program to print the U.S. presidential election years since 1792 to present day, knowing such elections occur every 4 years. Don't forget to use <= rather than == to help avoid an infinite loop.

```
1
2  public class ElectionYears {
3     public static void main(String[] args) {
4        int electYear = 0;
5
6        electYear = 1792;
7
8        // FIXME: Put the following in a while loop
9        System.out.println(electYear);
10
11    }
12 }
13
```

Run

Participation Activity

4.4.3: More counting with while loops.

Complete the following.

#	Question	Your answer

1	Loop iterates with i being the odd integers from 0 to 9.	`i = 1;`
		`while (i <= 9) {`
		`  // Loop body`
		`  i = [ ];`
		`}`

2	Loop iterates with i being multiples of 5 from 0 to 1000 (inclusive).	`i = 0;`
		`while (i <= 1000) {`
		`  // Loop body`
		`  i = [ ];`
		`}`

3	Loop iterates from 212 to 32 (inclusive).	`i = 212;`
		`while (i >= 32) {`
		`  // Loop body`
		`  i = [ ];`
		`}`

4	Loop iterates from -100 to 31 (inclusive).	`i = -100;`
		`while (i [ ] 32) {`
		`  /* Loop body statements go here */`
		`  i = i + 1;`
		`}`

Participation Activity

4.4.4: Loop simulator.

The following tool allows you to enter values for a loop's parts, and then executes the loop. Using the tool, try to solve each listed problem individually.

1. 0 to 100,000 by 5000s (so 0, 5000, 10000, ...).
2. -19 to 19 by 1s.
3. 10 to -10 by 1s.
4. Multiples of 3 between 0 and 100
5. Powers of 2 from 1 to 256 (so 1, 2, 4, 8, ...).
6. Come up with your own challenges.

```
int i = ;
while (i ) {
   System.out.print(i + " ");
   i = i ;
}
```

Run code

Output is: `Awaiting your input...`

Participation Activity

4.4.5: Calculate a factorial.

Write a program that lets a user enter N and that outputs N! (meaning N*(N-1)*(N-2)*...*2*1). Hint: Initialize a variable totalValue to N, and use a loop variable i that counts from N-1 down to 1.

```
1
2 import java.util.Scanner;
3
4 public class ElectionYears {
5    public static void main(String[] args) {
6       Scanner scnr = new Scanner(System.in);
7       int totalVal = 0;
8       int userInt = 0;
9
10      // FIXME: Ask user to input an integer, store in u
11
12      totalVal = userInt;
13      // FIXME: Add while loop that counts down to 1, up
14
15      System.out.println(userInt + "! is " + totalVal);
16   }
17 }
18
```

```
5
```

Run

Because i = i + 1 is so common in programs, the programming language provides a shorthand version ***++i***. The ++ is known as the ***increment operator***. A loop can thus be written as follows.

Construct 4.4.2: Loop with increment operator.

```
i = 1;
while (i <= N) {
   // Loop body
   ++i;
}
```

No space is necessary between the ++ and the i. A common error by new programmers is to use i = ++i instead of just ++i. The former works but is strange and unnecessary.

Likewise, the ***decrement operator***, as in --i, is equivalent to i = i - 1.

Sidenote: C++'s name stems from the ++ operator, suggesting C++ is an increment or improvement

over its C language predecessor.

The increment/decrement operators can appear in *prefix* form (++i or--i) or *postfix* form (i++ or i--). The distinction is relevant when used in a larger expression, as in x < i++. The prefix form first increments the variable, then uses the incremented value in the expression. The postfix form first uses the current variable value in the expression, and then increments the variable. We do not recommend use of the increment/decrement operators in larger expressions, and thus only use the prefix form, which some say is safer for beginner programmers in case they accidentally type i = ++i, which works as expected, whereas i = i++ does not.

Participation Activity

4.4.6: Increment/decrement operators.

#	Question	Your answer

1	What is the final value of i?
	`i = 0;`
	`++i;`
	`++i;`	

2	Replace the loop variable update statement by using the decrement operator.
	`i = 9;`
	`while (i > 0) {`
	`  // Loop body`
	`  i = i - 1;`
	`}`
		i = 9;
		while (i > 0) {
		  // Loop body
		  ;
		}

Challenge Activity

4.4.1: While loop: Print 1 to N.

Write a while loop that prints 1 to userNum, using the variable i. Follow each number (even the last or positive. Ex: userNum = 4 prints:

```
1 2 3 4
```

```
1  import java.util.Scanner;
2
3  public class CountDown {
4     public static void main (String [] args) {
5        int userNum = 0;
6        int i = 0;
7
8        userNum = 4;     // Assume positive
9
10       /* Your solution goes here  */
11
12       System.out.println("");
13
14       return;
15    }
16 }
```

Run

Challenge Activity

4.4.2: Printing output using a counter.

Re-type the following and run, note incorrect behavior. Then fix errors in the code, which should print

```
while (numPrinted != numStars) {
   System.out.print("*");
}
```

```
1  import java.util.Scanner;
2
3  public class StarPrinter {
4     public static void main (String [] args) {
5        int numStars = 0;
6        int numPrinted = 0;
7
8        numStars = 12;
9        numPrinted = 1;
10
11       /* Your solution goes here  */
12
13       System.out.println("");
14
15       return;
16    }
17 }
```

Run

(*Note_whileloops) *(To instructors): Focus is placed on mastering basic looping using while loops, before introducing for loops. Also, looping N times is initially done using 1 to <= N rather than 0 to < N due to being more intuitive to new programmers and less prone to error, the latter being commonplace as a consequence of arrays being numbered starting at 0.*

Section 4.5 - For loops

Counting in loops is so common that the language supports a loop type for that purpose. A ***for loop***

statement collects three parts—the loop variable initialization, loop expression, and loop variable update —all at the top of the loop, thus enhancing code readability reducing errors like forgetting to update the loop variable.

Construct 4.5.1: For loop.

```
for (initialExpression; conditionExpression; updateExpression) {
  // Loop body: Sub-statements to execute if the
  // conditionExpression evaluates to true */
}
// Statements to execute after the expression evaluates to false
```

A while loop and its equivalent for loop are shown below. Clearly, while loops are sufficient, but a for loop is a widely-used programming convenience.

Participation Activity 4.5.1: While/for loop correspondence.

Start

```
i = 0;
while (i <= 99) {
   // Loop body statements
   ++i;
}
```

```
for (i = 0; i <= 99; ++i) {
   // Loop body statements
}
```

```
i = 0;
while ( i <= 99 ) {
    // Loop body statements
    ++i;
}
```

```
for ( i = 0; i <= 99; ++i ) {
    // Loop body statements
}
```

Note that the for loop's third part (++i above) does *not* end with a semicolon.

Participation Activity

4.5.2: For loops.

Complete the for loop to achieve the goal. Use prefix increment (++i) or decrement (--i) where appropriate.

#	Question	Your answer

1	Iterate for i from 0 to 9.	`for (i = 0; i <= 9; ) {`
		`// Loop body`
		`}`

2	Iterate for numCars from 1 to 500. Note the variable is numCars (not i).	`for ( numCars <= 500; ++numCars) {`
		`// Loop body`
		`}`

3	Iterate for i from 99 down to 0. Compare with 0.	`for (i = 99; --i) {`
		`// Loop body`
		`}`

4	Iterate for i from 0 to 20 by 2s (0, 2, 4, ...). Use i = ??, NOT ++i.	`for (i = 0; i <= 20; ) {`
		`// Loop body`
		`}`

5	Iterate for i from -10 to 10. Compare with 10.	`for ( ) {`
		`// Loop body`
		`}`

Table 4.5.1: Choosing between while and for loops: General guidelines (not strict rules though).

for	Use when the number of iterations is computable before entering the loop, as when counting down from X to 0, printing a character N times, etc.
while	Use when the number of iterations is not computable before entering the loop, as when iterating until a user enters a particular character.

Participation Activity

4.5.3: While loops and for loops.

Choose the most appropriate loop type.

#	Question	Your answer
1	Iterate as long as user-entered char c is not 'q'.	while
		for
2	Iterate until the values of x and y are equal, where x and y are changed in the loop body.	while
		for
3	Iterate 100 times.	while
		for

Good practice is to use a for loop's parts to count the necessary loop iterations, with nothing added or omitted. The following loop examples should be avoided, if possible.

Figure 4.5.1: Avoid these for loop variations.

```
// initialExpression not related to counting iterations; move r = rand() before loop
for (i = 0, r = rand(); i < 5; ++i) {
   // Loop body
}

// updateExpression not related to counting iterations; move r = r + 2 into loop body
for (i = 0; i < 5; ++i, r = r + 2) {
   // Loop body
}
```

Participation Activity 4.5.4: For loop variations.

#	Question	Your answer
1	Each of the above for loop variations yields a syntax error.	True
		False
2	Even though the above for loop variations may execute correctly, they are generally considered bad style.	True
		False

A common error is to also have a ++i; statement in the loop body, causing the loop variable to be updated twice per iteration.

Figure 4.5.2: Common error: loop variable updated twice.

```
// Loop variable updated twice per iteration
for (i = 0; i < 5; ++i) {
   // Loop body
   ++i; // Oops
}
```

Participation Activity

4.5.5: For loop double increment.

#	Question	Your answer
1	Putting ++i at the end of a for loop body, in addition to in the updateExpression part, yields a syntax error.	True
		False

Challenge Activity

4.5.1: Enter the output for the for loop.

Start

Enter the output of the following program.

```
public class forLoopOutput {
   public static void main (String [] args) {
      int i = 0;

      for (i = 0; i <= 4; ++i) {
         System.out.print(i);
      }

      return;
   }
}
```

01234

1	2	3	4	5

Check Next

Challenge Activity

4.5.2: For loop: Print 1 to N.

Write a for loop that prints: 1 2 .. userNum. Print a space after each number, including after the last n

```
1 2 3 4
```

```
1  import java.util.Scanner;
2
3  public class CountToNum {
4     public static void main (String [] args) {
5        int userNum = 0;
6        int i = 0;
7
8        userNum = 4;
9
10       /* Your solution goes here  */
11
12       System.out.println("");
13
14       return;
15    }
16 }
```

Run

Challenge Activity

4.5.3: For loop: Print N to 0.

Write code that prints: userNum ... 2 1 Blastoff! Your code should contain a for loop. Print a newline a
Ex: userNum = 3 outputs:

```
3
2
1
Blastoff!
```

```
1  import java.util.Scanner;
2
3  public class CountdownRocket {
4     public static void main (String [] args) {
5        int userNum = 0;
6        int i = 0;
7
8        userNum = 3;
9
10       /* Your solution goes here  */
11
12       return;
13    }
14 }
```

Run

Section 4.6 - Nested loops

A ***nested loop*** is a loop that appears in the body of another loop. The nested loops are commonly referred to as the ***inner loop*** and ***outer loop***.

Nested loops have various uses. One use is to generate all combinations of some items. For example,

the following program generates all two-letter .com Internet domain names.

Figure 4.6.1: Nested loops example: Two-letter domain name printing program.

```
import java.util.Scanner;

public class DomainNamePrinter {

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String usrInput = "?";
      char letter1 = '?';
      char letter2 = '?';

      System.out.print("Enter any key to begin: ");
      usrInput = scnr.next(); // Unused; just to start the

      System.out.println("Two-letter domain names:");

      letter1 = 'a';
      while (letter1 <= 'z') {
         letter2 = 'a';
         while (letter2 <= 'z') {
            System.out.println("" + letter1 + "" + letter2
            ++letter2;
         }
         ++letter1;
      }

      return;
   }
}
```

```
Enter any key to begin: a

Two-letter domain names:
aa.com
ab.com
ac.com
ad.com
ae.com
af.com
ag.com
ah.com
ai.com
aj.com
ak.com
al.com
am.com
an.com
ao.com
ap.com
aq.com
ar.com
as.com
at.com
au.com
av.com
aw.com
ax.com
ay.com
az.com
ba.com
bb.com
bc.com
bd.com
be.com

...

zw.com
zx.com
zy.com
zz.com
```

Note that the program makes use of ascending characters being encoded as ascending numbers, e.g., 'a' is 97, 'b' is 98, etc., so assigning 'a' to letter1 and then incrementing yields 'b'.

(Forget about buying a two-letter domain name: They are all taken, and each sells for several hundred thousand or millions of dollars. Source: dnjournal.com, 2012).

Participation Activity

4.6.1: Two character dotcom domain names.

Modify the program to include two-character .com names where the second character can be a letter or a number, as in a2.com. Hint: Add a second loop, following the `while (letter2 <= 'z')` loop, to handle numbers.

```
1
2  import java.util.Scanner;
3
4  public class DomainNamePrinter {
5     public static void main (String [] args) {
6        Scanner scnr = new Scanner(System.in);
7        char letter1 = '?';
8        char letter2 = '?';
9
10       System.out.println("Two-letter domain names:");
11
12       letter1 = 'a';
13       while (letter1 <= 'z') {
14          letter2 = 'a';
15          while (letter2 <= 'z') {
16             System.out.println("" + letter1 +
17                                "" + letter2 + ".com");
18             ++letter2;
19          }
```

Run

Below is a nested loop example that graphically depicts an integer's magnitude by using asterisks, creating a "histogram." The inner loop is a for loop that handles the printing of the asterisks. The outer loop is a while loop that handles executing until a negative number is entered.

Figure 4.6.2: Nested loop example: Histogram.

```
import java.util.Scanner;

public class IntHistogram {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int numAsterisk = 0;  // Number of asterisks to print
      int i = 0;            // Loop counter

      numAsterisk = 0;
      while (numAsterisk >= 0) {
         System.out.print("Enter an integer (negative to qu
         numAsterisk = scnr.nextInt();

         if (numAsterisk >= 0) {
            System.out.println("Depicted graphically:");
            for (i = 1; i <= numAsterisk; ++i) {
               System.out.print("*");
            }
            System.out.println("\n");
         }
      }

      System.out.println("Goodbye.");

      return;
   }
}
```

```
Enter an integer (negat
Depicted graphically:
*********

Enter an integer (negat
Depicted graphically:
**********************

Enter an integer (negat
Depicted graphically:
**********************

Enter an integer (negat
Goodbye.
```

Participation Activity

4.6.2: Nested loops: Inner loop execution.

#	Question	Your answer

1	Given the following code, how many times will the inner loop body execute?
	`int row = 0;`
	`int col = 0;`
	`for(row = 0; row < 2; row = row + 1) {`
	`  for(col = 0; col < 3; col = col + 1) {`
	`  // Inner loop body`
	`  }`
	`}`	

2	Given the following code, how many times will the inner loop body execute?
	`char letter1 = '?';`
	`char letter2 = '?';`
	
	`letter1 = 'a';`
	`while (letter1 <= 'f') {`
	`  letter2 = 'c';`
	`  while (letter2 <= 'f') {`
	`  // Inner loop body`
	`  ++letter2;`
	`  }`
	`  ++letter1;`
	`}`	

Participation Activity

4.6.3: Nested loops: What is the output.

#	Question	Your answer

1	What is output by the following code?
	`int row = 0;`
	`int col = 0;`
	`for(row = 2; row <= 3; row = row + 1) {`
	`  for(col = 0; col <= 1; col = col + 1) {`
	`  System.out.print("" + row + col + " ");`
	`  }`
	`}`	

2	What is output by the following code?
	`char letter1 = '?';`
	`char letter2 = '?';`
	
	`letter1 = 'y';`
	`while (letter1 <= 'z') {`
	`  letter2 = 'a';`
	`  while (letter2 <= 'c') {`
	`  System.out.print("" + letter1`
	`  + letter2 + " ");`
	`  ++letter2;`
	`  }`
	`  ++letter1;`
	`}`	

Challenge Activity

4.6.1: Nested loops: Indent text.

Print numbers 0, 1, 2, ..., userNum as shown, with each number indented by that number of spaces. leading spaces, then the number, and then a newline. Hint: Use i and j as loop variables (initialize i an spaces like spaces after the printed number. Ex: userNum = 3 prints:

```
0
 1
  2
   3
```

```
1  public class NestedLoop {
2     public static void main (String [] args) {
3        int userNum  = 0;
4        int i = 0;
5        int j = 0;
6
7        /* Your solution goes here  */
8
9        return;
10    }
11 }
```

Run

Challenge Activity

4.6.2: Nested loops: Print seats.

Given numRows and numCols, print a list of all seats in a theater. Rows are numbered, columns lette after each seat, including after the last. Use separate print statements to print the row and column. E prints:

```
1A 1B 1C 2A 2B 2C
```

```
1 public class NestedLoops {
2    public static void main (String [] args) {
3       int numRows = 2;
4       int numCols = 3;
5
6       // Note: You'll need to define more variables
7
8       /* Your solution goes here  */
9
10      System.out.println("");
11
12      return;
13   }
14 }
```

Run

Section 4.7 - Developing programs incrementally

Creating correct programs can be hard. Following a good programming process helps. What many new programmers do, but shouldn't, is write the entire program, compile it, and run it—hoping it works. Debugging such a program can be difficult because there may be many distinct bugs.

Experienced programmers develop programs ***incrementally***, meaning they create a simple program version, and then growing the program little-by-little into successively more-complete versions.

The following program allows the user to enter a phone number that includes letters. Such letters appear on phone keypads along with numbers, enabling phone numbers like 1-555-HOLIDAY. The program converts a phone number having numbers/letters into one having numbers only.

The first program version simply prints each string element, to ensure the loop iterates properly.

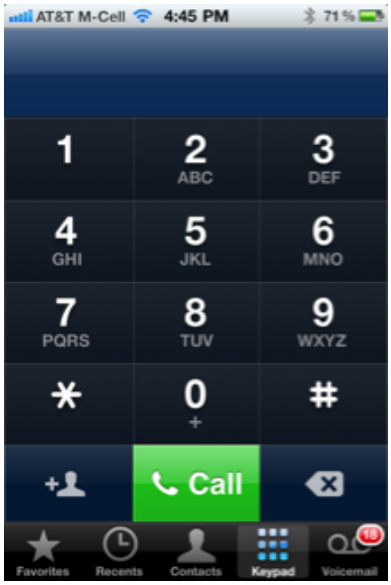


Figure 4.7.1: Incremental program development.

```
import java.util.Scanner;

public class PhoneNumberDecoder {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String phoneStr = ""; // User input: Phone number string
      int i = 0;            // Current index in phone number string

      System.out.print("Enter number: ");
      phoneStr = scnr.next();

      for (i = 0; i < phoneStr.length(); ++i) { // For each element
         System.out.println("Element " + i + " is: " + phoneStr.cha
      }

      return;
   }
}
```

```
Enter number: 1-
Element 0 is: 1
Element 1 is: -
Element 2 is: 5
Element 3 is: 5
Element 4 is: 5
Element 5 is: -
Element 6 is: H
Element 7 is: O
Element 8 is: L
Element 9 is: I
Element 10 is: I
Element 11 is: I
Element 12 is: Y
```

The second program version outputs any number elements, outputing '?' for non-number elements. A ***FIXME comment*** is commonly used to indicate program parts to be fixed or added, as above. Some editor tools automatically highlight the FIXME comment to attract the programmer's attention.

Figure 4.7.2: Second version echoes numbers, and has FIXME comment.

```
import java.util.Scanner;

public class PhoneNumberDecoder {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String phoneStr = "";  // User input: Phone number string
      int i = 0;             // Current index in phone number string
      char currChar = '_';   // Current char in phone number string

      System.out.print("Enter phone number: ");
      phoneStr = scnr.next();

      System.out.print("Numbers only: ");

      for (i = 0; i < phoneStr.length(); ++i) { // For each ele
         currChar = phoneStr.charAt(i);
         if ((currChar >= '0') && (currChar <= '9')) {
            System.out.print(currChar); // Print character as is
         }
         // FIXME: Add else-if branches for letters and hyphen
         else {
            System.out.print('?');
         }
      }

      System.out.println("");
      return;
   }
```

```
Enter phone number:
Numbers only: 1?555
```

The third version completes the else-if branch for the letters A-C (lowercase and uppercase), per a standard phone keypad. The program also modifies the if branch to echo a hyphen in addition to numbers.

Figure 4.7.3: Third version echoes hyphens too, and handles first three letters.

```
import java.util.Scanner;

public class PhoneNumberDecoder {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      String phoneStr = "";  // User input: Phone number string
      int i = 0;             // Current index in phone number string
      char currChar = '_';   // Current char in phone number string

      System.out.print("Enter phone number: ");
      phoneStr = scnr.next();

      System.out.print("Numbers only: ");

      for (i = 0; i < phoneStr.length(); ++i) { // For each element
         currChar = phoneStr.charAt(i);
         if (((currChar >= '0') && (currChar <= '9')) || (currChar ==
            System.out.print(currChar); // Print character as is
         }
         else if (((currChar >= 'a') && (currChar <= 'c'))
                  || ((currChar >= 'A') && (currChar <= 'C'))) {
            System.out.print('2');
         }
         // FIXME: Add remaining else-if branches
         else {
            System.out.print('?');
         }
      }

      System.out.println("");
      return;
   }
}
```

```
Enter phone nu
Numbers only:
```

The fourth version can be created by filling in the if-else branches similarly for other letters. We added more instructions too. Code is not shown below, but sample input/output is provided.

Figure 4.7.4: Fourth and final version sample input/output.

```
Enter phone number (letters/- OK, no spaces): 1-555-HOLIDAY
Numbers only: 1-555-4654329

...

Enter phone number (letters/- OK, no spaces): 1-555-holiday
Numbers only: 1-555-4654329

...

Enter phone number (letters/- OK, no spaces): 999-9999
Numbers only: 999-9999

...

Enter phone number (letters/- OK, no spaces): 9876zywx%$#@
Numbers only: 98769999????
```

Participation Activity

4.7.1: Incremental programming.

Complete the program by providing the additional if-else branches for decoding other letters in a phone number. Try incrementally writing the program by adding one "else if" branch at a time, testing that each added branch works as intended.

```
1
2  import java.util.Scanner;
3
4  public class PhoneNumberDecoder {
5     public static void main(String[] args) {
6        Scanner scnr = new Scanner(System.in);
7        String phoneStr = ""; // User input: Phone number
8        int i = 0;            // Current index in phone nu
9        char currChar = '_';  // Current char in phone num
10
11       System.out.println("Enter phone number: ");
12       phoneStr = scnr.next();
13
14       System.out.print("Numbers only: ");
15
16       for (i = 0; i < phoneStr.length(); ++i) { // For e
17          currChar = phoneStr.charAt(i);
18          if (((currChar >= '0') && (currChar <= '9')) ||
19             System.out.print(currChar); // Print charact
```

1-800-555-HOLIDAY

Run

Participation Activity

4.7.2: Incremental programming.

#	Question	Your answer
1	A good programming process is to write the entire program, then incrementally remove bugs one at a time.	True
		False
2	Expert programmers need not develop programs incrementally.	True
		False
3	Incremental programming may help reduce the number of errors in a program.	True
		False
4	FIXME comments provide a way for a programmer to remember what needs to be added.	True
		False
5	Once a program is complete, one would expect to see several FIXME comments.	True
		False

Section 4.8 - Break and continue

A ***break statement*** in a loop causes an immediate exit of the loop. A break statement can sometimes yield a loop that is easier to understand.

Figure 4.8.1: Break statement: Meal finder program.

```
import java.util.Scanner;

public class MealSolver {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int EMPANADA_COST = 3;
      final int TACO_COST     = 4;

      int userMoney    = 0;
      int numTacos     = 0;
      int numEmpanadas = 0;
      int mealCost     = 0;
      int maxEmpanadas = 0;
      int maxTacos     = 0;

      System.out.print("Enter money for meal: ");
      userMoney = scnr.nextInt();

      maxEmpanadas = userMoney / EMPANADA_COST;
      maxTacos     = userMoney / TACO_COST;

      for (numTacos = 0; numTacos <= maxTacos; ++numTacos) {
         for (numEmpanadas = 0; numEmpanadas <= maxEmpanadas; ++numEmpanadas) {

            mealCost = (numEmpanadas * EMPANADA_COST) + (numTacos * TACO_COST);

            // Find first meal option that exactly matches user money
            if (mealCost == userMoney) {
               break;
            }
         }

         // Find first meal option that exactly matches user money
         if (mealCost == userMoney) {
            break;
         }
      }

      if (mealCost == userMoney) {
         System.out.println("$" + mealCost + " buys " + numEmpanadas
                 + " empanadas and " + numTacos
                 + " tacos without change.");
      }
      else {
         System.out.println("You cannot buy a meal without having "
                 + "change left over.");
      }

      return;
   }
}
```

```
Enter money for meal: 20
$20 buys 4 empanadas and 2 tacos without change.

...

Enter money for meal: 31
```

```
$31 buys 9 empanadas and 1 tacos without change.
```

The nested for loops generate all possible meal options for the number of empanadas and tacos that can be purchased. The inner loop body calculates the cost of the current meal option. If equal to the user's money, the search is over, so the break statement immediately exits the inner loop. The outer loop body also checks if equal, and if so that break statement exits the outer loop.

The program could be written without break statements, but the loops' condition expressions would be more complex and the program would require additional code, perhaps being harder to understand.

Participation Activity

4.8.1: Break statements.

Given the following while loop, what is the value assigned to variable z for the given values of variables a, b and c?

```
mult = 0;
while (a < 10) {
   mult = b * a;
   if (mult > c) {
      break;
   }
   a = a + 1;
}
z = a;
```

#	Question	Your answer
1	a = 1, b = 1, c = 0	
2	a = 4, b = 5, c = 20	

A ***continue statement*** in a loop causes an immediate jump to the loop condition check. A continue statement can sometimes improve the readability of a loop. The example below extends the previous meal finder program to find meal options for which the total number of items purchased is evenly divisible by the number of diners. The program also outputs all possible meal options, instead of just reporting the first meal option found.

Figure 4.8.2: Continue statement: Meal finder program that ensures items purchased is evenly divisible by the number of diners.

```
import java.util.Scanner;

public class MealSolverMultipleDiners {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      final int EMPANADA_COST = 3;
      final int TACO_COST     = 4;

      int userMoney    = 0;
      int numTacos     = 0;
      int numEmpanadas = 0;
      int mealCost     = 0;
      int maxEmpanadas = 0;
      int maxTacos     = 0;
      int numOptions   = 0;
      int numDiners    = 0;

      System.out.print("Enter money for meal: ");
      userMoney = scnr.nextInt();

      System.out.print("How many people are eating: ");
      numDiners = scnr.nextInt();

      maxEmpanadas = userMoney / EMPANADA_COST;
      maxTacos     = userMoney / TACO_COST;

      for (numTacos = 0; numTacos <= maxTacos; ++numTacos) {
         for (numEmpanadas = 0; numEmpanadas <= maxEmpanadas; ++numEmpanadas) {

            // Total items  must be equally divisible by number of diners
            if (((numTacos + numEmpanadas) % numDiners) != 0) {
               continue;
            }

            mealCost = (numEmpanadas * EMPANADA_COST) + (numTacos * TACO_COST);

            if (mealCost == userMoney) {
               System.out.println("$" + mealCost + " buys " + numEmpanadas
                       + " empanadas and " + numTacos
                       + " tacos without change.");
               numOptions += 1;
            }
         }
      }

      if (numOptions == 0) {
         System.out.println("You cannot buy a meal without having "
                 + "change left over.");
      }

      return;
   }
}
```

```
Enter money for meal: 60
How many people are eating: 3
```

```
$60 buys 12 empanadas and 6 tacos without change.
$60 buys 0 empanadas and 15 tacos without change.

...

Enter money for meal: 54
How many people are eating: 2
$54 buys 18 empanadas and 0 tacos without change.
$54 buys 10 empanadas and 6 tacos without change.
$54 buys 2 empanadas and 12 tacos without change.
```

The nested loops generate all possible combinations of tacos and empanadas. If the total number of tacos and empanadas is not exactly divisible by the number of diners (e.g., `(( numTacos + numEmpanadas) % numDiners ) != 0 )`, the continue statement proceeds to the next iteration, thus causing incrementing of numEmpanadas and checking of the loop condition.

Break and continue statements can avoid excessive indenting/nesting within a loop. But they could be easily overlooked, and should be used sparingly, when their use is clear to the reader.

Participation Activity

4.8.2: Continue.

Given:

```
for (i = 0; i < 5; ++i) {
   if (i < 10) {
      continue;
   }
   <Print i>
}
```

#	Question	Your answer
1	The loop will print at least some output.	True
		False
2	The loop will iterate only once.	True
		False

Challenge Activity

4.8.1: Simon says.

"Simon Says" is a memory game where "Simon" outputs a sequence of 10 characters (R, G, B, Y) ar
Create a for loop that compares the two strings starting from index 0. For each match, add one poin
the loop using a break statement. Ex: The following patterns yield a userScore of 4:

```
simonPattern: R, R, G, B, R, Y, Y, B, G, Y
userPattern:  R, R, G, B, B, R, Y, B, G, Y
```

```
2
3  public class SimonSays {
4     public static void main (String [] args) {
5        String simonPattern = "";
6        String userPattern = "";
7        int userScore = 0;
8        int i = 0;
9
10       userScore = 0;
11       simonPattern = "RRGBRYYBGY";
12       userPattern  = "RRGBBRYBGY";
13
14       /* Your solution goes here  */
15
16       System.out.println("userScore: " + userScore);
17
18       return;
19    }
20 }
```

Run

Section 4.9 - Enumerations

Some variables only need store a small set of named values. For example, a variable representing a traffic light need only store values named GREEN, YELLOW, or RED. An ***enumeration type*** defines a name for a new type and possible values for that type.

Construct 4.9.1: Enumeration type.

```
public enum identifier {enumerator1, enumerator2,  ...}
```

The items within the braces ("enumerators") are named constants. Those constants are not assigned a specific numeric value, but instead must be referred to by the defined names. An enumeration defines a new data type that can be used like the built-in types int, char, etc.

Figure 4.9.1: Enumeration example.

```
import java.util.Scanner;

/* Manual controller for traffic light */
public class TrafficLightControl {
   // enum type definition occurs outside the main method
   public enum LightState {RED, GREEN, YELLOW, DONE}

   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      LightState lightVal = LightState.RED;
      String userCmd = "-";

      System.out.println("User commands: n (next), r (red), q (quit).\n");

      lightVal = LightState.RED;
      while (lightVal != LightState.DONE) {

         if (lightVal == LightState.GREEN) {
            System.out.print("Green light  ");
            userCmd = scnr.next();
            if (userCmd.equals("n")) { // Next
               lightVal = LightState.YELLOW;
            }
         }
         else if (lightVal == LightState.YELLOW) {
            System.out.print("Yellow light  ");
            userCmd = scnr.next();
            if (userCmd.equals("n")) { // Next
               lightVal = LightState.RED;
            }
         }
         else if (lightVal == LightState.RED) {
            System.out.print("Red light  ");
            userCmd = scnr.next();
            if (userCmd.equals("n")) { // Next
               lightVal = LightState.GREEN;
            }
         }

         if (userCmd.equals("r")) { // Force immediate red
            lightVal = LightState.RED;
         }
         else if (userCmd.equals("q")) { // Quit
            lightVal = LightState.DONE;
         }
      }

      System.out.println("Quit program.");
      return;
   }
}
```

```
User commands:

Red light  n
Green light  n
Yellow light
Red light  n
Green light  n
Red light  n
Green light  n
Yellow light
Red light  q
Quit program.
```

The program declares a new enumeration type named LightState. The program then defines a new variable lightVal of that type. The loop updates lightVal based on the user's input.

The example illustrate the idea of a ***state machine*** that is sometimes used in programs, especially programs that interact with physical objects, wherein the program moves among particular situations ("states") depending on input; see Wikipedia: State machine.

A programmer must include both the enumeration type and the enumerator within that type, as in `lightVal = LightState.RED;`. A common error is to omit the enumeration type in an expression. For example, the statement `lightVal = RED;` results in a compilation error.

Different enumerated types may use some of the same enumerators. For example, the above program might also declare `public enum Warnings {GREEN, ORANGE, RED}`. The enumeration values are then accessed as Warnings.RED and LightState.RED.

One might ask why the light variable wasn't simply defined as a string, and then compared with strings "GREEN", "RED", and "YELLOW". Enumerations are safer. If using a string, an assignment like `light = "ORANGE"` would not yield a compiler error, even though ORANGE is not a valid light color. Likewise, `light == "YELOW"` would not yield a compiler error, even though YELLOW is misspelled.

One could instead define final variables for strings like `final String LS_GREEN = "GREEN";` or even integer values like `final int LS_GREEN = 0;` and then use those constants in the code, but an enumeration is clearer, requires less code, and is less prone to error.

Participation Activity

4.9.1: Enumerations.

#	Question	Your answer
1	Define a new public enumeration type named HvacStatus with three named values HvacOff, AcOn, FurnaceOn, in that order.	
2	Define a variable of the enumeration type HvacStatus named systemStatus.	
3	Assign the value AcOn to the variable systemStatus.	

C Challenge Activity

4.9.1: Enumerations: Grocery items.

Print either "Fruit", "Drink", or "Unknown" (followed by a newline) depending on the value of userItem. newline) if the value of userItem does not match any of the defined options. For example, if userItem

```
Fruit
```

```
1  import java.util.Scanner;
2
3  public class GrocerySorter {
4     public enum GroceryItem {GR_APPLES, GR_BANANAS, GR_JUICE, GR_WATER};
5        public static void main (String [] args) {
6
7        GroceryItem userItem = GroceryItem.GR_APPLES;
8
9        /* Your solution goes here  */
10
11       return;
12    }
13 }
```

Run

Section 4.10 - Java example: Salary calculation with loops

P Participation Activity

4.10.1: Calculate adjusted salary and tax with deductions: Using loops.

A program may execute the same computations repeatedly.

The program below repeatedly asks the user to enter an annual salary, stopping when the user enters 0 or less. For each annual salary, the program determines the tax rate and computes the tax to pay.

1. Run the program below with annual salaries of 40000, 90000, and then 0.
2. Modify the program to use a while loop inside the given while loop. The new inner loop should repeatedly ask the user to enter a salary deduction, stopping when the user enters a 0 or less. The deductions are summed and then subtracted from the annual income, giving an adjusted gross income. The tax rate is then calculated from the adjusted gross income.
3. Run the program with the following input: 40000, 7000, 2000, 0, and 0. Note that the 7000 and 2000 are deductions.

Reset

```
1
2 import java.util.Scanner;
3
4 public class IncomeTax {
5    public static void main (String [] args) {
6       Scanner scnr = new Scanner(System.in);
7       final String SALARY_PROMPT = "\nEnter annual salary (0 to exit): ";
8       int    annualSalary    = 0;
9       int    deduction       = 0;
10      int    totalDeductions = 0;
11      double taxRate         = 0.0;
12      int    taxToPay        = 0;
13
14      System.out.println(SALARY_PROMPT);
15      annualSalary = scnr.nextInt();
16
17      while (annualSalary > 0) {
18
19         // FIXME: Add a while loop to gather deductions. Use the variables
```

```
40000
90000
0
```

Run

A solution to the above problem follows. The input consists of three sets of annual salaries and deductions.

Participation Activity

4.10.2: Calculate adjusted salary and tax with deductions: Using loops (solution).

Reset

```
1
2 import java.util.Scanner;
3
4 public class IncomeTax {
5    public static void main (String [] args) {
6       Scanner scnr = new Scanner(System.in);
7       final String PROMPT_SALARY = "\nEnter annual salary (0 to exit): ";
8       final String PROMPT_DEDUCTION = "Enter a deduction (0 to end deductions): ";
9       int    annualSalary    = 0;
10      int    oneDeduction    = 0;
11      int    totalDeductions = 0;
12      int    adjustedSalary  = 0;
13      double taxRate         = 0.0;
14      int    taxToPay        = 0;
15
16      System.out.println(PROMPT_SALARY);
17      annualSalary = scnr.nextInt();
18
19      while (annualSalary > 0) {
```

```
40000 3000 6000 0
90000 5000 0
60000 2000 1000 1450 0
```

Run

Participation Activity

4.10.3: Create an annual income and tax table.

A tax table shows three columns: an annual salary, the tax rate, and the tax amount to pay. The program below shows most of the code needed to calculate a tax table.

1. Run the program below and note the results.
2. Alter the program to use a for loop to print a tax table of annual income, tax rate, and tax to pay. Use starting and ending annual salaries of 40000 and 60000, respectively, and a salary increment of 5000.
3. Run the program again and note the results. You should have five rows in the tax table.
4. Alter the program to add user prompts and read the starting and ending annual incomes from user input.
5. Run the program again using 40000 and 60000, respectively, and the same salary increment of 5000. You should have the same results as before.
6. Alter the program to ask the user for the increment to use in addition to the starting and ending annual salaries.
7. Run the program again using an increment of 2500. Are the entries for 40000, 45000, 50000, 55000 and 60000 the same as before?

Reset

```
import java.util.Scanner;

public class IncomeTax {
   public static void main (String [] args) {
      final int INCOME_INCREMENT = 5000;

      Scanner scnr = new Scanner(System.in);
      int    annualSalary         = 0;
      double taxRate              = 0.0;
      int    taxToPay             = 0;
      int    startingAnnualSalary = 0;  // FIXME: Change the starting salary to 4000
      int    endingAnnualSalary   = 0;  // FIXME: Change the ending salary to 60000

      // FIXME: Use a for loop to calculate the tax for each entry in the table.
      // Hint: the initialization clause is annualSalary = startingAnnualSalary

         // Determine the tax rate from the annual salary
         if (annualSalary <= 0) {
            taxRate = 0.0;
```

40000 60000 5000

Run

A solution to the above problem follows.

Participation Activity

4.10.4: Create an annual income and tax table (solution).

Reset

```
1
2  import java.util.Scanner;
3
4  public class IncomeTax {
5     public static void main (String [] args) {
6        Scanner scnr = new Scanner(System.in);
7        int    annualSalary         = 0;
8        double taxRate              = 0.0;
9        int    taxToPay             = 0;
10       int    startingAnnualSalary = 0;
11       int    endingAnnualSalary   = 0;
12       int    incomeIncrement      = 0;
13
14       System.out.println("Enter first annual salary for the table: ");
15       startingAnnualSalary = scnr.nextInt();
16       System.out.println("Enter last annual salary for the table: ");
17       endingAnnualSalary = scnr.nextInt();
18       System.out.println("Enter the increment for the table: ");
19       incomeIncrement = scnr.nextInt();
```

40000 60000 2500

Run

Section 4.11 - Java example: Domain name validation with

Participation Activity

4.11.1: Validate domain names.

A ***top-level domain*** (TLD) name is the last part of an Internet domain name like .com in example.com. A ***core generic top-level domain*** (core gTLD) is a TLD that is either .com, .net, .org, or .info. A ***second-level domain*** is a single name that precedes a TLD as in apple in apple.com

The following program uses a loop to repeatedly prompt for a domain name, and indicates whether that domain name consists of a second-level domain followed by a core gTLD. An example of a valid domain name for this program is apple.com. An invalid domain name for this program is support.apple.com because the name contains two periods. The program ends when the user presses just the Enter key in response to a prompt.

1. Run the program and enter domain names to validate. Note that even valid input is flagged as invalid.
2. Change the program to validate a domain name. A valid domain name for this program has a second-level domain followed by a core gTLD. Run the program again.

Reset

```
1  import java.util.Scanner;
2
3  public class CoreGtldValidation {
4
5     public static void main (String [ ] args) {
6        Scanner scnr = new Scanner(System.in);
7        String inputName   = "";
8        String searchName  = "";
9        String coreGtld1   = ".com";
10       String coreGtld2   = ".net";
11       String coreGtld3   = ".org";
12       String coreGtld4   = ".info";
13       String theTld      = "";
14       boolean isCoreGtld = false;
15       // FIXME: Add variable periodCounter to count periods in a domain name
16       int periodPosition = 0; // Position of the period in the domain name
17
18       int i = 0;
19
```

```
apple.com
APPLE.COM
apple.comm
le.info
```

Run

A solution for the above problem follows.

Participation Activity

4.11.2: Validate domain names (solution).

Reset

```
1  import java.util.Scanner;
2
3  public class CoreGtldValidation_Solution {
4     public static void main (String [ ] args) {
5        Scanner scnr = new Scanner(System.in);
6        String inputName   = "";
7        String searchName  = "";
8        String coreGtld1   = ".com";
9        String coreGtld2   = ".net";
10       String coreGtld3   = ".org";
11       String coreGtld4   = ".info";
12       String theTld      = "";
13       boolean isCoreGtld = false;
14       int periodCounter  = 0;
15       int periodPosition = 0;
16       int i = 0;
17
18       System.out.println("\nEnter the next domain name (<Enter> to exit): ");
19       inputName  = scnr.nextLine();
```

apple.com
APPLE.COM
apple.comm
le.info

Run