Topic 15
Indefinite Loops - While Loops

"If you cannot grok [understand] the overall structure of a program while taking a shower [e.g., with no external memory aids], you are not ready to code it."

-Rich Pattis

Based on slides for Building Java Programs by Reges/Stepp, found at http://faculty.washington.edu/stepp/book/

Types of loops

- **definite loop**: A loop that executes a known number of times.
  - The for loops we have seen so far are definite loops. We often use language like "repeat _ times" or "for each of these things".
  - Examples:
    - Repeat these statements 10 times.
    - Repeat these statements *k* times.
    - Repeat these statements for each odd number between 5 and 27.
- **indefinite loop**: A loop where it is not easily determined in advance how many times it will execute.
  - Indefinite loops often keep looping as long as a condition is true, or until a condition becomes false.
  - Examples:
    - Repeat these statements until the user types a valid integer.
    - Repeat these statements while the number *n* is not prime.
    - Repeat these statements until a factor of *n* is found.
    - Flip a coin until you get 10 flips in a row of the same result

The while loop statement

- The *while loop* is a new loop statement that is well suited to writing indefinite loops.
- The while loop, general syntax:

```
while (<condition>) {
    <statement(s)> ;
}
```

  - Example:

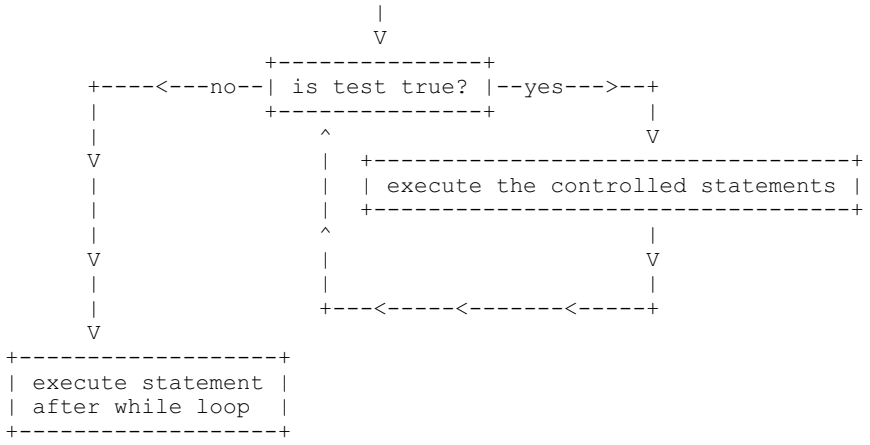
```
int number = 1;
while (number <= 200) {
    System.out.print(number + " ");
    number *= 2;
}
```

  - OUTPUT:

```
1 2 4 8 16 32 64 128
```


While loop flow chart

- The execution of a while loop can be depicted as the following:



Example while loop

- A loop that finds and prints the first factor of a number (other than 1):

```
Scanner console = new Scanner(System.in);
System.out.print("Type a number: ");
int number = console.nextInt();
int factor = 2;
while (number % factor != 0) {
    factor++;
}
System.out.println("First factor: " + factor);
```

- OUTPUT:

```
Type a number: 49
First factor: 7
```


Equivalence of for,while loops

- Any for loop of the following form:

```
for (<initialization>; <condition>; <update>) {
    <statement(s)> ;
}
```

can be replaced by a while loop of the following form:

```
<initialization>;
while (<condition>) {
    <statement(s)> ;
    <update> ;
}
```


for/while loop example

- What while loop is essentially equivalent to the following for loop?

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

- ANSWER:

```
int i = 1;
while (i <= 10) {
    System.out.println("Hi there");
    i++;
}
```


While loop problem

- Write a piece of Java code that uses a while loop to repeatedly prompt the user to type a number until the user types a non-negative number, then square it.
  - Expected output:

```
Type a non-negative integer: -5
Invalid number, try again: -1
Invalid number, try again: 11
11 squared is 121
```

- Solution:

```
System.out.print("Type a non-negative integer: ");
int number = console.nextInt();
while (number < 0) {
    System.out.print("Invalid number, try again: ");
    number = console.nextInt();
}
int square = number * number;
System.out.println(number + " squared is " + square);
```

Square Root

- Recall Heron's method for calculating square roots
- problem: Find sqrt(n)
- Algorithm:
  1. Make a guess at the solution. (x_1)
  2. $x_2 = (x_1 + (n / x_1)) / 2$
  3. Repeat for $x_3, x_4, x_5, \ldots$

Write a Java program that implements Heron's method to find the square root of 133,579 using 20 iterations of the algorithm.

Square Root

- Why 20 iterations?
  Is that enough?
  Too many?

```
public static double squareRoot(double num){
      double result = num / 2;
      for(int i = 1; i <= 20; i++){
            result = (result + (num / result)) / 2.0;
      }
      return result;
}
```


Square Root

- Rewrite square root using a while loop

- Make initial guess
- refine results while result squared is not equal to num

Square Root

- First Attempt

```
public static double squareRoot2(double num){
       double result = num / 2;
       while( result * result != num){
           result = ( result + (num / result)) / 2.0;
       }
       return result;
}
```

- Problem.
  - Recall that variables use a finite amount of memory and are subject to round off and precision errors
- Will get stuck in an infinite loop
- Define a tolerance and accept results that meet that tolerance

Sentinel Loops

- Sentinel: a value that signals the end of user input
- Sentinel loop: a loop that keeps repeating until the sentinel value is found
- Problem:
  - Write a program to read in ints from the user until they enter -1 to quit.
  - Print out the sum and average of the numbers entered

Example Sentinel Program

```
Enter an int (-1 to quit): 12
Enter an int (-1 to quit): 37
Enter an int (-1 to quit): 42
Enter an int (-1 to quit): 25
Enter an int (-1 to quit): 12
Enter an int (-1 to quit): 99
Enter an int (-1 to quit): -1
Sum of 6 numbers is 227
Average of 6 numbers is 37.833333333333336
```


Sentinel Program – First Attempt

- initialize sum, count of numbers, and number
- while number isn't sentinel value
  - read in a num
  - add it to sum
  - increment count of numbers
- print out sum and average

Sentinel Program – First Attempt

```
public static void main(String[] args){
     Scanner key = new Scanner(System.in);
     int sum = 0;
     int count = 0;
     int number = 0; // anything but -1
     while( number != -1 ){
         System.out.print("Enter an int (-1 to quit): ");
         number = key.nextInt();
         sum += number;
         count++;
     }
     System.out.println( "Sum of " + count
             + " numbers is " + sum );
     System.out.println( "Average of " + count
             + " numbers is " + (1.0 * sum / count));
}
```

Sentinel Program – First Attempt

- Output

```
Enter an int (-1 to quit): 12
Enter an int (-1 to quit): 37
Enter an int (-1 to quit): 42
Enter an int (-1 to quit): 25
Enter an int (-1 to quit): 12
Enter an int (-1 to quit): 99
Enter an int (-1 to quit): -1
Sum of 7 numbers is 226
Average of 7 numbers is 32.285714285714285
```


Sentinel Loop

- What is the problem?
  - A compiler error?
  - A runtime error?
  - A logic error?
- We are adding the sentinel to the sum and counting it as a number
- We need to read N numbers (including the sentinel value) but only want to use the first N – 1
- A fencepost problem!

Sentinel Loop – Second Attempt

```
public static void main(String[] args){
    Scanner key = new Scanner(System.in);
    int sum = 0;
    int count = 0;
    System.out.print("Enter an int (-1 to quit): ");
    int number = key.nextInt();
    while( number != -1 ){
        sum += number;
        count++;
        System.out.print("Enter an int (-1 to quit): ");
        number = key.nextInt();
    }
    System.out.println( "Sum of " + count
                              + " numbers is " + sum );
    System.out.println( "Average of " + count
              + " numbers is " + (1.0 * sum / count));
}
```


Sentinel Loop

- Adding num to sum and incrementing count moved to top of the loop
- Should add an if to ensure program does not divide by 0
- Add a constant for the Sentinel to make program more readable

Sentinel Loop – Final Version

```
public static final int SENTINEL = -1;

public static void main(String[] args){
    Scanner key = new Scanner(System.in);
    int sum = 0;
    int count = 0;
    System.out.print("Enter an int ("
                     + SENTINEL + " to quit): ");
    int number = key.nextInt();
    while( number != SENTINEL ){
        sum += number;
        count++;
        System.out.print("Enter an int (-1 to quit): ");
        number = key.nextInt();
    }
    System.out.println( "Sum of " + count
                        + " numbers is " + sum );
    if( count > 0 )
        System.out.println( "Average of " + count
                + " numbers is " + (1.0 * sum / count));
    else
        System.out.println( "Cannot compute average of 0 terms.");
}
```


Type boolean

- **`boolean`**: Primitive type to represent logical values.
  - A `boolean` variable can hold one of two values: `true` or `false`.
  - All the ***<condition>***s we have used in our `if` statements and `for` loops have been `boolean` literal values.
  - It is legal to create `boolean` variables, pass `boolean` parameters, return `boolean` values from methods, ...

`boolean` Examples

```
int x = 7;
boolean test1 = true;
boolean test2 = (x < 10);       // true
boolean test3 = (x % 2 == 0);   // false
if (test2)
     System.out.println("under 10");

int wins = 4;
boolean manyWins = wins >= 8;
boolean beatCAL = true;
if( manyWins && beatCAL)
    System.out.println("A great season!!!");
else if( manyWins )
    System.out.println("Good year, but no ax.");
else if( beatCAL )
    System.out.println("At least we have the ax.");
else
   System.out.println("Maybe I should become a UT fan.");
```


Review - Logical operators && || !

- Boolean expressions can be joined together with the following *logical operators*:

Operator	Description	Example	Result
`&&`	and	(9 != 6) && (2 < 3)	true
`||`	or	(2 == 3) || (-1 < 5)	true
!	not	!(7 > 0)	false

- The following 'truth tables' show the effect of each operator on any boolean values p and q:

p	q	p && q	p || q
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

p	!p
true	false
false	true

Methods that return boolean

- There are several methods in Java that return boolean values.
  - A call to one of these methods can be used as a ***<condition>*** on a for loop, while loop, or if statement.
  - Examples:

```
Scanner console = new Scanner(System.in);
System.out.print("Type your age or name: ");
if (console.hasNextInt()) {
    int age = console.nextInt();
    System.out.println("You are " + age + " years old.");
} else {
    String line = console.nextLine();
    if (line.startsWith("Dr.")) {
        System.out.println("Will you marry me?");
    }
}
```


Testing for valid user input

- A `Scanner` object has methods that can be used to test whether the upcoming input token is of a given type:

Method	Description
`hasNext()`	Whether or not the next token can be read as a `String` *(always true for console input)*
`hasNextInt()`	Whether or not the next token can be read as an `int`
`hasNextDouble()`	Whether or not the next token can be read as a `double`
`hasNextLine()`	Whether or not the next line of input can be read as a `String` *(always true for console input)*

- Each of these methods waits for the user to type input tokens and press Enter, then reports a true or false answer.
  - The hasNext and hasNextLine methods are not useful until we learn how to read input from files in Chapter 6.

Scanner condition example

- The `Scanner`'s `hasNext___` methods are very useful for testing whether the user typed the right kind of token for our program to use, before we read it (and potentially crash!).
- We will use them more when read data from files instead of the keyboard

Scanner condition Example Code

```
Scanner console = new Scanner(System.in);
System.out.print("How old are you? ");
if (console.hasNextInt()) {
    int age = console.nextInt();
    System.out.println("Retire in " + (65 - age) + " years.");
} else {
    System.out.println("You did not type an integer.");
}

System.out.print("Type 10 numbers: ");
for (int i = 1; i <= 10; i++) {
    if (console.hasNextInt()) {
        System.out.println("Integer: " + console.nextInt());
    } else if (console.hasNextDouble()) {
        System.out.println("Real number: " + console.nextDouble());
    }
}
```

"Boolean Zen"

- Methods that return a `boolean` result sometimes have an `if/else` statement:

```
public static boolean bothOdd(int n1, int n2) {
    if (n1 % 2 != 0 && n2 % 2 != 0) {
        return true;
    } else {
        return false;
    }
}
```


"Boolean Zen"

- ... but the `if/else` is sometimes unnecessary.
  - The `if/else`'s condition is itself a `boolean` expression its value is exactly what you want to return!!!

```
public static boolean bothOdd(int n1, int n2)
    return (n1 % 2 != 0 && n2 % 2 != 0);
}
```


"Boolean Zen" template

- Replace:

```
public static boolean <name>(<parameters>) {
    if (<condition>) {
        return true;
    } else {
        return false;
    }
}
```

- with:

```
public static boolean <name>(<parameters>) {
    return <condition>;
}
```


Boolean practice problem

- Write a program that compares two words typed by the user to see whether they "rhyme" (end with the same last two letters) and/or alliterate (begin with the same letter).
  - Use methods with return values to tell whether two words rhyme and/or alliterate.
  - Example:

```
Type two words: car STAR
They rhyme!
```

(run #2)

```
Type two words: bare bear
They alliterate!
```

(run #3)

```
Type two words: sell shell
They alliterate!
They rhyme!
```

Boolean practice problem

- Write a program that reads two numbers from the user and tells whether they are relatively prime (have no common factors).
  - Examples:

```
Type two numbers: 9 16
9 and 16 are relatively prime
```

(run #2)

```
Type two numbers: 7 21
7 and 21 are not relatively prime
7 is a factor of 7 and 21
```