Introduction to Computer Programming

Lecture #7 - Conditional Loops

The Problem with Counting Loops

- Many jobs involving the computer require repetition, and that this can be implemented using loops.
- Counting loops allows us to perform a statement or a block of statements a certain number of times.
- The problem is that we do not always know exactly how many times to perform the statements in a loop in every situation.
The Problem with Counting Loops (continued)

- Let’s take another look at our payroll program:
  - We do not always know how payroll records that we have.
  - It isn’t very convenient to have to count the records, especially if it’s a big number.
  - Wouldn’t it be better if we could keep going until we enter some special value to tell the computer to stop?

Conditional Loops

- Conditional loops allow us to do this.
- *Conditional* loops keep repeating as long as some *condition* is true (or until some condition *becomes* true).
- Steps in solving a problem that involve *while, until, as long as* indicate a conditional loop.
While Loops

- The most common form of conditional loops are **while** loops.
- In Java, they have the form:
  ```java
  while (condition)
      statement;
  or
  while(condition) {
      statements
  }
  ```

A simple example - *Keep Asking*

```java
import java.util.Scanner;

class KeepAsking {
    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        int number = keyb.nextInt();

        // Get your first number
        System.out.println
            ("Hi there. Pick a positive integer");
        number = keyb.nextInt();
    }
}
```
The test average program revisited

• Let’s take another look at our program to calculate test average:

• Write a program that can calculate a test average and grade for any number of tests. The program will finish when the user types in a grade of -1.
Sentinel Value

- Often conditional loops continue until some special value is encountered in the input which effectively tells the program to stop running the loop. This is called a **sentinel value** because it is the value for which we are watching.
- -1 is the sentinel value in the GPA algorithm’s main loop

The *TestAverage* Algorithm

1. As long as there are more grades, add them to the total
2. Divide the total by the number of courses
3. Print the corresponding letter grade
Refining The *TestAverage* Algorithm

1. As long as there are more grades, add them to the total
2. Divide the total by the number of courses
3. Print the corresponding letter grade

1.1 Get the first grade
1.2 As long as the grade is not -1, add it to the total and get the next grade

1.2.1 while the grade is not -1:
1.2.2 Add it to the total
1.2.3 Get the next grade
1.2.4 Add one to the number of courses

3. Divide the total by the number of courses
4. Print the corresponding letter grade
Refining The TestAverage Algorithm

1.1 Get the first grade
1.2.1 while the grade is not -1:
1.2.2 Add it to the total
1.2.3 Get the next grade
1.2.4 Add one to the number of courses
2. Divide the total by the number of courses
3. Print the corresponding letter grade

```java
System.out.println("What grade did you get on your first test ?");
System.out.println("Enter -1 to end");
thisGrade = keyb.nextInt();
```

```
Refining The TestAverage Algorithm

System.out.println("What grade did you get on your first test ?");
System.out.println("Enter -1 to end");
thisGrade = keyb.nextInt();
1.2.1 while the grade is not -1:
1.2.2 Add it to the total
1.2.3 Get the next grade
1.2.4 Add one to the number of courses
2. Divide the total by the number of courses
3. Print the corresponding letter grade
```

```
numTests++;
```
Refining the TestAverage Algorithm

```java
System.out.println("What grade did you get on your first test?");
System.out.println("Enter -1 to end");
thisGrade = keyb.nextInt();
1.2.2 while the grade is not -1:
1.2.2 Add it to the total
numTests++;
2. Divide the total by the number of courses
3. Print the corresponding letter grade

System.out.println("What grade did you get on this test?");
thisGrade = keyb.nextInt();
```

Refining the TestAverage Algorithm

```java
System.out.println("What grade did you get on your first test?");
System.out.println("Enter -1 to end");
thisGrade = keyb.nextInt();
1.2.2 while the grade is not -1:
1.2.2 Add it to the total
numTests++;
2. Divide the total by the number of courses
3. Print the corresponding letter grade

total = total + thisGrade;
```
Refining The *TestAverage* Algorithm

System.out.println
("What grade did you get on your first test ?");
System.out.println("Enter -1 to end");
thisGrade = keyb.nextInt();

1.2.1 while the grade is not -1:
    total = total + thisGrade;
    System.out.println
    ("What grade did you get on this test ?");
    thisGrade = keyb.nextInt();
    numTests++;
2. Divide the total by the number of courses
3. Print the corresponding letter grade

while (thisGrade != sentinelGrade) {
    
    testAverage = total/numTests;
}
Refining The TestAverage Algorithm

... ...
while (thisGrade != sentinelGrade) {
    total = total + thisGrade;
    System.out.println
        ("What grade did you get on this test 
            ");
    thisGrade = keyb.nextInt();
    numTests++;
}
testAverage = total/numTests;

3. Print the corresponding letter grade
if (testAverage >= 90)    courseGrade = 'A';
else if (testAverage >= 80) courseGrade = 'B';
else if (testAverage >= 70) courseGrade = 'C';
else if (testAverage >= 60) courseGrade = 'D';
else    courseGrade = 'F';

The TestAverage Program

import java.util.Scanner;

public class CalcGrade {
    // Calculates the average test grade and
    // converts it to a letter grade assuming that
    // A is a 90 average, B is an 80 average and so
    // on.

    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        final int sentinelGrade = -1;
        int thisGrade, numTests = 0, total, thisGrade;
        float testAverage;
        char courseGrade;

        // Initially, the total is 0
        total = 0;
// Get the first grade
System.out.println("What grade did you get on your first test ?");
System.out.println("Enter -1 to end");
thisGrade = keyb.nextInt();

// Add up the test grades
while (thisGrade != sentinelGrade) {
  // Make sure that the grades are valid percentages
  if (thisGrade > 100)
    System.out.println("This is not a valid test grade.");
  else if (thisGrade >= 0) {
    total = total + thisGrade;
    numTests++;
  } else
    System.out.println("This is not a valid test grade.");

  System.out.println("What grade did you get on this test ?");
  thisGrade = keyb.nextInt();
}

// Find the average
testAverage = total/numTests;

// Find the letter grade corresponding to the average
if (testAverage >= 90)
  courseGrade = 'A';
else if (testAverage >= 80)
  courseGrade = 'B';
else if (testAverage >= 70)
  courseGrade = 'C';
else if (testAverage >= 60)
  courseGrade = 'D';
else
  courseGrade = 'F';

// Print the results.
System.out.println("Your test average is ", testAverage);
System.out.println("Your grade will be ", courseGrade);
}
Payroll program revisited

- Let’s revisit the payroll program.
- Instead of counting up the payroll records so we can count the number of times we go through the loop, why not use some sentinel value in the last entry to tell the program when we’re finished?
- Since no one will ever make $0.00 per hour, we’ll use a pay rate of 0 as our sentinel value in the revised payroll program.

Our Initial Payroll Algorithm

1. Display instructions for the user
2. Keep processing payroll records as long as there are more.
Refining the Payroll Algorithm

1. Display instructions for the user
2. Keep processing payroll records as long as there are more.
   2.1 Get the first pay rate
   2.2 while The Rate is positive:
   2.3 Process payroll record
   2.4 Get the next pay rate
   2.5 Print final count

while (rate > 0.0) {
    // Payroll processing logic
}
Refining the Payroll Algorithm

1. Display instructions for the user
2.1 Get the first pay rate
2.2 while (rate > 0.0) {
   2.3 Process payroll record
   2.4 Get the next pay rate
}
2.5 Print final count

2.3.1 Calculate gross pay
2.3.2 Print gross pay
2.3.3 Increment record count

Refining the Payroll Algorithm

1. Display instructions for the user
2.1 Get the first pay rate
   while (rate > 0.0) {
      2.3.1 Calculate gross pay
      2.3.2 Print gross pay
      2.3.3 Increment record count
      2.4 Get the next pay rate
   }
2.5 Print final count

2.3.1.1 Get hours worked
2.3.1.2 Calculate gross pay
Refining the Payroll Algorithm

1. Display instructions for the user
2.1 Get the first pay rate
   while (rate > 0.0) {
      2.3.1.1 Get hours worked
      2.3.1.2 Calculate gross pay
      2.3.2 Print gross pay
      2.3.3 Increment record count
      2.4 Get the next pay rate
   }
2.5 Print final count

System.out.println
   "Enter the hours worked ?");
   hours = keyb.nextInt();

if (hours > 40)
   pay = 40*rate + 1.5*rate*(hours-40);
else
   pay = rate * hours;
Refining the Payroll Algorithm

1. Display instructions for the user
2.1 Get the first pay rate
   
   while (rate > 0.0) {
       System.out.println
           ("Enter the hours worked ?");
       hours = keyb.nextInt();
       if (hours > 40)
           pay = 40*rate + 1.5*rate*(hours-40);
       else
           pay = rate * hours;
       System.out.printf
           ("Gross pay is $%.2f\n\n", pay);

   2.3.3 Increment record count
2.4 Get the next pay rate
   }

   ...

   numPeople++;
Refining the Payroll Algorithm

1. Display instructions for the user
2.1 Get the first pay rate

while (rate > 0.0) {
    System.out.println
        ("Enter the hours worked ?");
    hours = keyb.nextInt();
    if (hours > 40)
        pay = 40*rate + 1.5*rate*(hours-40);
    else
        pay = rate * hours;
    System.out.printf
        ("Gross pay is $%4.2f
\n", pay);
    numPeople++;
    System.out.println
        ("What is the pay rate for the "
            + "next employee ? ");
    rate =keyb.nextDouble();
}
2.5 Print the final count

System.out.println
    ("To stop the program, enter a pay rate"
        + " of zero or less \n\n");
Refining the Payroll Algorithm

System.out.println("To stop the program, enter a pay rate" + " of zero or less \n\n");

2.1 Get the first pay rate
while (rate > 0.0) {
  ...
}

2.5 Print the final count

System.out.println("What is rate of pay for the employee ? ");
rate = keyb.nextDouble();

System.out.println("There were " + numPeople + " payroll records.");
The Payroll Program

import java.util.Scanner;

public class Payroll2  {
    // Processes a payroll for a given number of
    // employees. The user indicates that (s)he is
    // finished by entering a pay rate that is zero
    // or negative.
    public static void main(String[] args)  {
        Scanner keyb = new Scanner(System.in);
        int     numPeople = 0;
        double  hours, rate, pay;

        // Display instructions
        System.out.println
            ("To stop the program, enter a pay rate"
             + " of zero or less \n\n");

        // Ask the user for the first employee's
        // pay rate
        System.out.println
            ("What is rate of pay for the employee  ? ");
        rate = keyb.nextDouble();

        // Calculate gross salary for everyone on the
        // payroll
        while (rate > 0.0) {
            // Enter the hours worked
            System.out.println
                ("Enter the hours worked ?");
            hours = keyb.nextInt();

            // Calculate and print the pay.
            // If hours exceed 40, pay time and a half
            if (hours > 40)
                pay = 40*rate + 1.5*rate*(hours-40);
            else
                pay = rate * hours;

            numPeople++;

            // Ask the user for the next employee's
            // pay rate
            System.out.println
                ("What is rate of pay for the employee  ? ");
            rate = keyb.nextDouble();
        }

        System.out.println(
            "The payroll was calculated for "+numPeople + " employees.");
    }
}
System.out.printf
        ("Gross pay is $%4.2f\n\n", pay);
    numPeople++;

    // Get the pay rate for the next employee
    System.out.println
        ("What is the pay rate for the "
        + "next employee ? ");
    rate =keyb.nextDouble();
} System.out.println("There were " + numPeople
        + " payroll records.");
System.out.println("Payroll is finished.");
    }
}

Compound Interest program
revisited

- Our earlier program showed how much interest is compounded over a given number of years.
- Let’s see how long your money would have to earn interest to reach a million dollars
Redesigning Our Compound Interest Program

Input – Input deposit
Output – The final year and exact balance
Other information
   New Principle = (1 + Interest Rate) * Old Principle

Initial Algorithm:
• Set the initial principle at $24
• For every year since 1625, add 5% interest to the principle until the principle reaches $1,000,000 and print the principle every twenty years and when it reaches $1 million
• Print the values for the final year

Refining The Compound Interest Algorithm

1. Set the initial principle at $24

2. For every year since 1625, add 5% interest to the principle until the principle reaches $1,000,000 and print the principle every twenty years and when it reaches $1 million

3. Print the values for the final year

<table>
<thead>
<tr>
<th>2.1 Set Year to 1625</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2 While Principle &lt; $1,000,000</td>
</tr>
<tr>
<td>2.3 Add 5% Interest to the Principle</td>
</tr>
<tr>
<td>2.4 If the Year % 20 = 5</td>
</tr>
<tr>
<td>2.5 then print the principle</td>
</tr>
</tbody>
</table>
Refining The Compound Interest Algorithm

1. Set the initial principle at $24
2.1 Set Year to 1625
2.2 While Principle < $1,000,000
2.3 Add 5% Interest to the Principle
2.4 If the Year % 20 = 5
2.5 then print the principle
3. Print the values for the final year

while (principle < target) {
    interest = rate * principle;
    principle = principle + interest;
}

Refining The Compound Interest Algorithm

1. Set the initial principle at $24
2.1 Set Year to 1625
2.2 While Principle < $1,000,000
2.3 Add 5% Interest to the Principle
2.4 If the Year % 20 = 5
2.5 then print the principle
3. Print the values for the final year

while (principle < target) {
    interest = rate * principle;
    principle = principle + interest;
}
Refining The Compound Interest Algorithm

1. Set the initial principle at $24

2.1 Set Year to 1625

   while (principle < target) {
      interest = rate * principle;
      principle = principle + interest;

2.4 If the Year % 20 = 5

2.5 then print the principle

   }

3. Print the values for the final year

   if (year%20 == 5)
   System.out.printf
   ("year = %4d\tinterest = %13.2f"
   + "\tprinciple = %15.2f\n",
    year, interest, principle);

year++;
Refining The Compound Interest Algorithm

1. Set the initial principle at $24
2.1 Set Year to 1625
... ...
}
System.out.printf("year = %4d\tinterest = %13.2f"
+ "\tprinciple = %15.2f\n",
year, interest, principle);

principle = 24;

year = 1625;
import java.util.Scanner;

public class Interest4 {
    // Calculate the interest that the Canarsie Indians
    // could have accrued if they had deposited the $24
    // in a bank account at 5% interest.
    public static void main(String[] args) {
        int year = 1625;
        final double rate = 0.05, target = 1000000;
        double interest, principle;

        // Set the initial principle at $24
        principle = 24;

        // For every year since 1625, add 5% interest
        // to the principle until the principle
        // reaches $1,000,000
        // Print the principle every twenty years
        // and when it reaches $1 million
        // There has to be two fixed places for the
        // principle
        interest = 0;
        while (principle < target) {
            interest = rate * principle;
            principle = principle + interest;

            if (year%20 == 5)
                // Use 15 places for printing the principle
                System.out.printf
                ("year = %4d\tinterest = %13.2f"
                 + "\tprinciple = %15.2f\n",
                 year, interest, principle);
            year++;
        }
    }
}
Magic Number Problem

- The magic number game involves guessing a number and with each wrong guess, the player is told “too high” or “too low”. The goal is to guess the number in the smallest number of tries.
- We need a method for having the computer pick a number at random for the player to guess.
- We will need to learn about how to use “library functions” to provide us with the magic number.
Designing the Magic Number Algorithm

**Input** – The player’s guess(es)

**Output** – A clue (“too high” or “too low”) and the number of guesses that it took.

**Initial Algorithm**

1. Use the random number function to pick a number
   - Let the player make a guess
   - As long as the player hasn’t guessed the number, give the appropriate clue and let him/her guess again.
   - Print the number of tries

**Refining the Magic Number Algorithm**

1. 1. Use the random number function to pick a number
2. Let the player make a guess
3. As long as the player hasn’t guessed the number, give the appropriate clue and let him/her guess again.
4. Print the number of tries

3.1 While the guess isn’t the magic number:
3.2 Give the appropriate clue
3.3 Increment the count of tries
3.4 Let the player make another guess.
Refining the Magic Number Algorithm

1. Use the random number function to pick a number
2. Let the player make a guess
3.1 While the guess isn’t the magic number:
   3.2 Give the appropriate clue
   3.3 Increment the count of tries
   3.4 Let the player make another guess.
4. Print the number of tries

   3.2.1 if the number is too high, say so..
   3.2.2 Else say that the number is too low.

while (guess != magic) {

1. Use the random number function to pick a number
2. Let the player make a guess
   ```java
   while (guess != magic) {
       // If the number is too high, say so.
       if (guess > magic)
           System.out.println(".. Wrong .. Too high\n");
       else
           System.out.println(".. Wrong .. Too low\n");
   }
3. Increment the count of tries
4. Let the player make another guess.
3.4 Let the player make another guess.
   }
4. Print the number of tries
   ```

```java
if (guess > magic)
    System.out.println(".. Wrong .. Too high\n");
else
    System.out.println(".. Wrong .. Too low\n");
```
1. Use the random number function to pick a number
2. Let the player make a guess
   while (guess != magic) {
     if (guess > magic)
       System.out.println(".. Wrong .. Too high\n");
     else
       System.out.println(".. Wrong .. Too low\n");
     tries++;
   }
3. Let the player make another guess.
4. Print the number of tries

   System.out.println("Guess ?");
   guess = keyb.nextInt();

4. Print the number of tries

   System.out.println("You took " + tries + " guesses\n");
1. Use the random number function to pick a number
2. Let the player make a guess
   
   ```java
   while (guess != magic) {
   if (guess > magic)
       System.out.println(".. Wrong .. Too high\n");
   else
       System.out.println(".. Wrong .. Too low\n");
   tries++;
   System.out.println("Guess ?");
   guess = keyb.nextInt();
   }
   System.out.println("** Right!! ** ");
   System.out.println(magic
   + " is the magic number\n");
   System.out.println("You took " + tries
   + " guesses\n");
   ```

   System.out.println("Guess ?");
   guess = keyb.nextInt();

---

**import** and Standard Classes (rand)

- It is frequently helpful to be able to use software routines that have already been written for common tasks.
- **System.out.println** and **keyb.nextInt()** are examples of this.
- **import** allows us to access entire libraries of routines that are part of one or more **classes**
- When we write:
  ```java
  import java.util.Scanner;
  ```
  We are telling the Java compiler where to find the definitions of the Scanner class.
import and Standard classes (Random)

• To use the random number function, we need to include
  
  ```java
  import java.util.*;
  ```

• This tells the computer that java.util contains
subdirectories with class definitions that it will need to use.

• A class is similar to a data type but it can be defined by a
programmer or may come as a standard part of the
programming language. Classes need to be initialized
before use:
  
  ```java
  Scanner keyb = new Scanner(System.in);
  ```

• The name of the random number function that we want is
nextInt() – it is part of the object that we will declare
called newRandomNumber.

The Magic Number Program

```java
import java.util.*;

public class MagicNumber {

  // The magic number game has the user trying to
  // guess which number between 1 and 100 the
  // computer has picked
  public static void main(String[] args) {
    Scanner keyb = new Scanner(System.in);
    Random newRandomNumber = new Random();
    int magic, guess;
    int tries = 1;

    // Use the random number function to pick a
    // number
    magic = newRandomNumber.nextInt(100) + 1;
  }
}
```
// Let the user make a guess
System.out.println("Guess ?");
guess = keyb.nextInt();

while (guess != magic) {

    // Otherwise tell him whether it's too high
    // or too low
    else if (guess > magic)
    System.out.println
        (".. Wrong .. Too high\n");
    else
    System.out.println(".. Wrong .. Too low\n");

    // Let the user make another guess
    System.out.println("Guess ?");
guess = keyb.nextInt();
tries++;}

// If the user won, tell him/her
System.out.println("** Right!! **");
System.out.println(magic + " is the magic number\n");

    // Tell the user how many guesses it took
    System.out.println("You took " + tries + " guesses\n");
}
Declaring Boolean Constants

- If we want to work with true and false we can work with boolean variables.
- We can write:
  ```java
  boolean married = true;
  ...
  ...
  if (married)
      System.out.println("The employee is married\n");
  ```

! operator

- Sometimes we want to test to see if a condition is not true.
- We can do this by using the not operator, which is written as !:
  ```java
  if (!married)
      System.out.println("Do you" + " want to bring a" + " date? ");
  ```
& & and | | Operators

- Sometimes there may be two or more conditions to consider. For this reason we have the & & (AND) and | | (OR) operators.
- If we declare
  - boolean p, q;
  - ...
- Both p and q must be true for p & & q to be true.
- p | | q is true unless both p and q are false.

do.. while loops

- You may have noticed that we asked the user twice for same information - the number (s)he is guessing.
- Some loops really require that the condition be at the end - not at the beginning.
- In Java, we have the do.. while loop, whose syntax is:
  ```java
do {
    statement(s)
} (condition)
```
Revisiting the magic number program

- The main loop in the magic number program becomes:
  
  ```java
  do {
      System.out.println("Guess ?");
      guess = keyb.nextInt();

      // Otherwise tell him whether it's too high
      // or too low
      if (guess > magic)
          System.out.println(".. Wrong .. Too high\n");
      else
          System.out.println(".. Wrong .. Too low\n");

      // Try one more time...
      tries++;
  } while (guess != magic);
  ```
Nim

- The game Nim starts out with seven sticks on the table.
- Each player takes turns picking up 1, 2 or 3 sticks and cannot pass.
- Whoever picks up the last stick loses (the other player wins).

The Nim Problem

- **Input**
  - The number of sticks the player is picking up
- **Output**
  - The number of sticks on the table
  - Who won (the player or the computer)
- **Other Information**
  - Whoever leaves 5 sticks for the other player can always win if they make the right followup move:
    - If the other player takes 1, you pick up 3
    - If the other player takes 2, you pick up 2
    - If the other player takes 3, you pick up 1
Designing The Nim Algorithm

1. Print the instructions
2. Set the number of stick at 7 and initialize other values
3. Find out if the user wants to go first or second
4. If the user goes second, have the computer take two sticks and the user goes second, have the computer take two sticks.
5. As long as there is no winner, keep playing

5.1 While there is no winner
5.2 Find out how many sticks the user is taking
5.3 Make sure it’s a valid choice
5.4 Pick up the appropriate number of sticks.
5.5 Take the sticks off the table
5.6 See if there is a winner
1. Print the instructions
2. Set the number of sticks at 7 and initialize other values
3. Find out if the user wants to go first or second
4. If the user goes second, have the computer take two sticks and the user goes second, have the computer take two sticks.
5.1 While there is no winner
5.2 Find out how many sticks the user is taking
5.3 Make sure it’s a valid choice
5.4 Pick up the appropriate number of sticks.
5.5 Take the sticks off the table
5.6 See if there is a winner

5.3.1 Make sure the user picked 1, 2, or 3 sticks
5.3.2 Make sure that user didn’t take more sticks than are on the table

if (sticksLeft == 6 || sticksLeft == 5 || sticksLeft == 2)
    reply = 1;
else if (sticksLeft == 4)
    reply = 3;
else if (sticksLeft == 3)
    reply = 2;
5.1 While there is no winner

if (sticksLeft == 6 || sticksLeft == 5 || sticksLeft == 2)
    reply = 1;

5.5 Take the sticks off the table

else if (sticksLeft == 1) {
    System.out.println("Congratulations! You won!");
    winner = true;
}
else if (sticksLeft == 0) {
    System.out.println("Sorry, the computer has won - you have lost");
    winner = true;
}

else if (sticksLeft == 1) {
    System.out.println("You won!");
    winner = true;
}
else if (sticksLeft == 0) {
    System.out.println("Sorry, the computer has won you have lost");
    winner = true;
}

else if (sticksLeft == 1) {
    System.out.println("Congratulations! You won!");
    winner = true;
}
else if (sticksLeft == 0) {
    System.out.println("Sorry, the computer has won you have lost");
    winner = true;
}

checking for valid input

- Very often, people will enter data that is not valid.
- An example of this is a negative number of hours worked, a name that contains numbers, a ten-digit zip code, etc.
- It is very important to learn how to check for bad data and what to do when you encounter it.
import java.util.Scanner;

public class Nim {
    // A program for the class game of Nim
    public static void main(String[] args) {
        Scanner keyb = new Scanner(System.in);
        int sticksLeft, pickUp, reply;
        boolean winner, move;
        char answer;
        String answerString = new String();

        // Print the instructions
        System.out.println(("There are seven (7) sticks on the table.");
        System.out.println("Each player can pick up one, two , or "+"three sticks");
    }
}
System.out.println
("in a given turn. A player cannot "+ "pick up more than");
System.out.println
("three stick nor can a player pass.\n");

// Initialize values
sticksLeft = 7;
pickUp = 0;
reply = 0;
winner = false;
answer = ' ';
// If the user goes first, tell him how many
// sticks are on the table
else
    System.out.println("There are 
               + sticksLeft
               + " on the table.");

// As long as there is no winner, keep playing
while (!winner) {
    move = false;
    // How many sticks is the user taking
    while (!move) {
        System.out.println("How many sticks do 
               + "you wish to pick up?");
        pickUp = keyb.nextInt();

        // Make sure its 1, 2 or 3
        if (pickUp < 1 || pickUp > 3)
            System.out.println(pickUp
                                + " is not a legal number of sticks");

        // Make sure that there are
        // enough sticks on the table
        else if (pickUp > sticksLeft)
            System.out.println("There are not 
                                + pickUp + " sticks on the table");
        else
            move = true;
    }

    // Take the sticks off the table
    sticksLeft = sticksLeft - pickUp;
// Plan the computer's next move
if (sticksLeft == 6 || sticksLeft == 5
    || sticksLeft == 2)
    reply = 1;
else if (sticksLeft == 4)
    reply = 3;
else if (sticksLeft == 3)
    reply = 2;
// See if the user won
else if (sticksLeft == 1) {
    System.out.println
        ("Congratulations! You won!");
    winner = true;
}

// See if the user lost
else if (sticksLeft == 0) {
    System.out.println
        ("Sorry, the computer has won "
            + "- you have lost...");
    winner = true;
}
// If neither happened, get ready
// for the next move
if (!winner) {
    sticksLeft = sticksLeft - reply;
    System.out.println
        ("The computer picked up " + reply
        + " sticks.");
    System.out.println
        ("There are now " + sticksLeft
        + " sticks left on the table.\n\n\n\n");
}