Lecture 3: Applications of Looping and Conditional Execution

1 Monte Carlo Simulations

(from Wikipedia, retrieved on Feb. 9, 2014) Monte Carlo methods (or Monte Carlo experiments) are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results; typically one runs simulations many times over in order to obtain the distribution of an unknown probabilistic entity. The name comes from the resemblance of the technique to the act of playing and recording your results in a real gambling casino. They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to obtain a closed-form expression, or infeasible to apply a deterministic algorithm.

1.1 Computing the Value of $\pi$

One can try to approximate the value of $\pi$ using geometry.

Assume that the side of the square above is 1 unit length. The area of the square is then $1 \times 1 = 1$.

What is the area of the red part (a quarter of a disk)? The whole disk would have an area of $\pi r^2$, since we only have a quarter of it, the area is $\pi r^2 / 4 = \pi / 4$.

The ratio of the area of the quarter disk to the area of the whole square is $\pi / 4$.

How can this be used to approximate the value of $\pi$?
Simulation If we could randomly generate points within the whole square, we can count how many of them fall within the quarter disk (how?) and then divide that number by the total number of points. If we repeat the experiment enough times, the ration of these two numbers should be getting closer and closer to $\pi/4$. Then $\pi$ can be approximated by multiplying the ratio by 4.

Source Code See examples Computing Pi.java and Computing Pi_GUI.java in the lecture notes.

1.2 Computing the Chances of Hitting the Bullseye in a Dart Competition

Here is a dart board. We want to figure out what the chances are of hitting the bullseye (small disk in the center) and what the chances are of hitting each of the rings.

We can employ a very similar method as in the previous examples. If we simulate random dart throws (i.e., generating random points that fall within the square board) then we can simply count how many of them fall within each region.

Note: this example is not completely realistic: it does not take into account the skill of the player and it assumes that every single dart hits the board.

Source Code See examples BullseyeChance.java and BullseyeChance_GUI.java in the lecture notes.

2 Representing Numbers in Binary

You might have learned how to convert numbers from decimal to binary and vice versa in some previous course. Do you still remember how it is done?

You most likely did not learn the method that we are about to look at for converting from decimal to binary.
In this method we will take advantage of the fact that numbers on the computer are actually represented using binary. A short primitive data type uses 16 bits to represent integers. Each of those bits is either zero or one. When you assign a numerical value to a variable, lets say short \( x = 5 \) the 5 in decimal has to be converted to binary before it is stored. When you try to, for example, display the value of \( x \), it is converted from binary to decimal and then displayed. This is done because most humans, even the computer programmers, do not really think in binary numbers.

There are bit operators in Java that allow you to operate directly on the bit representation of variables (see Appendix G in the book).

### 2.1 Using a mask

You can ”get” a specific bit of a number by applying a binary and operator: & (single ampersand as opposed to the double ampersand). If you AND a number NUM with another number that you know has only one non-zero bit (which numbers are those?) then the result indicates something about the original number NUM.

**Example:**

\[
\begin{array}{c}
0000011100110011 & 0000111001110101 \\
\& 0000000000000001 & 0000000000000001 \\
\hline
0000000000000001 & 0000000000000000 \\
0000011100110011 & 0000111001110101 \\
\& 0000000000000100 & 0000000000000100 \\
\hline
0000000000000000 & 0000000000000000 \\
\end{array}
\]

### 2.2 Shifting bits

You can shift the bits of the number right or left. This is equivalent to dividing or multiplying by two. The shifting operators are >> (right shift) and << (left shift).

**Example:**

\[
\begin{array}{c}
0000011100110011 << 1 = 0000111001100110 \\
0000011100110011 << 3 = 0011100110011000 \\
0000011100110011 >> 1 = 0000011100110011 \\
0000011100110011 >> 3 = 0000000111001101 \\
0000011100110011 >> 3 = 00000000011100110 \\
\end{array}
\]

### 2.3 Using binary operators to convert decimal numbers to binary

There are many ways of converting decimal numbers to binary using bit operators. Here is one:
Algorithm for extracting bits from a decimal integer.

Repeat as many times as there are bits (16 in case of primitive type short):

- use the mask of decimal number 1 (it is 0000000000000001 in binary) to extract the rightmost bit from your number
  - (number & 1) is 1 if the rightmost bit is 1
  - (number & 1) is 0 if the rightmost bit is 0
- shift the number one bit to the right
  - number = number >> 1;

You have to make sure to “collect” all the zeros and ones from each iteration of the loop in a correct order. Can you think of another way of using bit operators to achieve the same goal?