**Class Notes 3**

Today's agenda:
1. Review RNG and Number Cube program
2. Magic 8 Ball
3. Game Logic and Artificial Intelligence
4. Rock, Paper, Scissors

**RNG**

Random Number Generation on a computer is never truly random, but only *pseudorandom.* Most algorithms that computers have to generate these numbers depend on a **seed value**, a number that constantly changes and is used to introduce that important degree of variability into the RNG (random number generator).

Java's syntax for using the RNG is:

```
import java.util.Random;

Random gen = new Random();

// RNG returns an integer between 0 and 9
int newNum = gen.nextInt( 10 );

// to get a number within a specific integer range
// use this specific algorithm
int min = 3;
int max = 12;
int randomNum = gen.nextInt((max-min) + 1) + min;
```

*Why do I have to go through this process? Why can't I just hardcode the numbers directly into the line with the generator?*

*Why do we want to be more abstract vs. concrete at times like this?*

1. It is easier to change the value of variables that have been declared at the beginning of the code rather than chasing down every single hardcoded value in a program.

2. We want to be able to refer to some value using its symbolic name vs. trying to catch its current value in memory when it is needed.

**Magic 8 Ball Toy**



There is a toy (entirely for entertainment purposes) that purports to give you advice/answers to yes and no questions called the Magic 8 Ball. Shaking the ball causes a 20-sided number cube inside, suspended in a semi-viscous solution, to rotate. At some point, one of the sides will cling to a transparent window at the bottom of the toy. The result is a yes, no, or maybe answer.

Your objective for this programming assignment is to create a program that simulates the output of a Magic 8 Ball. **Prompt** the user for a yes/no question, then proceed to give them a random response that corresponds to a yes, no, or maybe. You do not need to code 20 different responses, maybe only do 4-8 in total.

A sample run of the program might look like this:

```
Magic 8 Ball!
Enter a yes/no question: Am I beautiful?
Definitely not.

Magic 8 Ball!
Enter a yes/no question: Am I intelligent?
Signs point to yes.

Magic 8 Ball!
Enter a yes/no question: Is there hope for humanity?
Unclear at this time.
```

After the program is complete, run it multiple times until all possible outcomes have occurred. This is called **testing**, and it is a critical part of assuring that software has quality! In testing, we have to see that every possible kind of result that the program is capable of presenting is demonstrated at least once.

### Switch

The switch statement is a special kind of conditional structure that operates using an integer as its condition.

```
switch( integer )
{
    case 0: // code
            break;
    case 1: // code
            break;
    case 2: // code
            break;
    .
    .
    .
    default: // code if no other cases match
}
```

Switch is sometimes considered "poorly implemented" and is not a desired feature for most modern languages. In Java, it is being *deprecated* or

removed from the language, however as it works in C and C++, it is good to know that it is an option. In some cases, like this one, it actually is easier to understand and maintain than some massive if-else-if statement.

**Probability and Game Logic**

Using computers to model events that have statistically variable outcomes is an important part of *simulations*. Simulations let us test possible results for real-world applications in a virtual setting that has no real costs aside from running the program.

One popular game that is played frequently by humans is called **Rock, Paper, Scissors**. It is a zero-sum game that has random outcomes when played and thus is useful for decision making when discourse has come to a standstill.

If you are unfamiliar with the game, please visit: **www.worldrps.com**

**Programming Project: RPS**

Write a program that allows a human player to play a single round of RPS against a computer opponent.

Before writing any code, first write out the *algorithm*. An algorithm is the steps one takes to solve a problem. A computer program simply follows these steps in sequential order to achieve a solution. Write the steps of the algorithm for a single game of RPS now:

1. Prompt the user for their throw.
2. Announce the player's throw.
3. Randomly generate the computer's throw.
4. Announce the computer's throw.
5. Compare the results and declare a winner.

From these five steps, you should then generate code comments that match the steps of the algorithm.

Use // for a single line code comment.

Use /* and */ to enclose multi-line comments.

Please create a new class and main function to hold the RPS code, import Scanner and Random, instantiate the scanner and random objects, and then place the five code comments for the RPS algorithm in your main function.

**Data Representation**

At this point, we need to consider: how are we going to represent the three different possible throws?

*Should we ask people to type in the name of the throw they want to make?*

Spelling can be an issue...scissors tends to be executed with errors.

*Is there another way to represent the throws?*

We could use R, P, and S as Strings or char.
We could use integers 1, 2, and 3.

Since our RNG already comes up with integers, it might be best to stick with the same data type!

**Handling Errors in User Input**

*What should we do if our user types in a number less than 1 or greater than 3? What if they accidentally type in a letter?*

We can validate the input, and if it doesn't fit the range of acceptable inputs, we can modify it to suit the range.

## Loops

It is very frustrating to have to test RPS by having to run it over and over again manually. We can automate the process of repeating things in a programming language using a structure called a **loop**.

There are two kinds of loops in Java, each with a primary version and a secondary (and less commonly used) version.

### While

The while loop is best used when one does not know how many times they wish to repeat (or *iterate*) a block of code. For example, the software that controls the temperature in a building will probably keep the air conditioner or heater running until a target temperature is achieved. We do not know how long that will take, because that depends on variables that the system does not know or cannot control.

```
while( condition )
{
    // code block
}
```

A while loop will repeat the block of code as long as the condition is true. When it is run, first the condition is checked, and if true, the block of code will execute. It the returns to the condition to see if it is still true. If so, it repeats the block of code once again. This cycle continues until the condition becomes false and the loop ends.

### For

The second kind of loop is called the **for** loop, and it is best used when you want to iterate a specific number of times. A for loop statement has three parts:

1. loop control variable (LCV) - keeps track of how many times we have looped the block of code
2. condition - determines if we keep looping or not

3. increment/decrement - increases or decreases the LCV

```
for( int i = 0; i < 10; i++ )
{
    System.out.print( i + " " );
}
```

**output:**
**0 1 2 3 4 5 6 7 8 9**

For the incrementer, we can use a numeric shorthand in C like languages:
`i++ means the same thing as i = i + 1`
`i-- means the same thing as i = i - 1`

Now you understand how the programming language C++ got its name. It literally means "C + 1" because it added objects to the original C language. Of course, the languages B and A came before C...more proof that programmers are not very creative when it comes to naming things, but it is very logical.

C-like languages:
C++

Java (Sun Microsystems, now Oracle)
Kotlin (Google/Android)

Objective-C (NeXTSTEP, now Apple)
Swift (Apple)

C# (Microsoft)

**Programming Exercises**

1. Write a for loop that prints the numbers from 1-20 inclusive in sequential order.

2. Write a for loop that prints the numbers from 10 to 1 inclusive in descending order.

**Loop Equivalency**

It is technically possible to write any for loop as a while loop, and vice versa.

Re-write problems 1 and 2 using a while loop. You will need to create a **counter** variable to track the number of iterations, as a while loop does not create one by default.

**Infinite Loops**

A loop that fails to stop running is called an *infinite loop*. While writing one could possibly be deliberate, we consider infinite loops to be errors.

If you accidentally write an infinite loop and your Java program locks up, you need to shut down the program. In BlueJ you can stop the Java program from running by right-clicking on the bottom activity bar on the right side of the status bar, and selecting *Reset Java Virtual Machine*. Anytime you have a program that gets out of control, use this option to prevent it from sucking up memory, processor time, etc.

**Homework**

1. Read Chapter 6 (Loops), specifically sections 6.1 through 6.4. Note that we have not yet covered the two alternate forms of while and for in lecture yet. Try some of the programming examples or programming problems at the end of the chapter if you are not totally confident with your loop-writing abilities.

2. Modify your RPS code to allow the user to specify how many games they would like to play against the computer. Then, keep track of how many player wins, computer wins, and ties there are, and when the looping is completed, declare a "big winner" for the match. *Hint: where do you put variables so that the program can always get to them?*