Table of Contents

1 In these Sushi Cards you're going to make a simple game using one of the most popular programming languages in the world: Java. Minecraft was originally built in Java, as was Gmail. Just like a lot of programmers today, Java was the first language I learned.

2 The game you're going to create will be something you might have played before: You'll be tossing a "coin" and guessing which way it's going to land.

3 Since you usually need to install a bunch of software to do Java programming on your computer, you're going to use an online Java development tool for these cards, to make things easier. So go to dojo.soy/java-dev

4 This tool is called **repl.it**. You'll need to create an account (or sign in with one you already have!), but once you've got one you can save your code and even come back to it from another computer!

5 Once you're signed in and setup, click on the **New Session** button and type `Java` into the box that appears. Select the first result.

6 The screen that opens in front of you has a section on the left to write your code and a section on the right where you can see it run. Go to next card and you can write your first program!

1 Every Java program needs to start with some special code. For now, let's call them "magic words". I'll explain them in later Sushi Cards. Type this in:

```
class Main {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

2 Now run the code (click on the **run** button!) and see what happens in the section on the right!

3 That's your first Java program! A lot of that code is those magic words, but the bit inside the second set of curly braces (`{ }`) is what's making it do something. This bit is called the **main method**. The code there now tells Java to print whatever's in the the double quotes (`" "`) to the screen. It's how your game will talk to its player!

```
System.out.println("Hello World!");
```

4 Notice a couple of things about that line of code: the way the editor highlights different parts (the message text and the command to Java) in different colours. Remove a quote (don't forget to put it back!) and notice how the colours change! Also, notice how the line ends with a semicolon (`;`). That tells Java the line is over. Without it, Java will get really confused!

5 Of course, while "Hello World!" is the classic way to start your first computer program, it's not going to be much use in this game! Change it to "Guess heads (h) or tails (t)!" and run it again to test it. Now you're ready to get the player guessing!

```
System.out.println("Guess heads (h) or tails (t)!");
```

1 So, you're asking the player for a guess. How do you tell Java to listen for that guess? You'll need to add a few bits to the code to make it happen. First, you're going to use something called `Scanner` which you'll need to `import`. Importing code is how you tell Java you're going to use something more than the very basics, and lets it load it so it's there when you ask for it. So, add this line at the very start of your code:

```
import java.util.Scanner;
```

2 Next, you need to create a `Scanner` inside your **main method** and give it a label you can use to refer to it later. I picked `user_input` as my label, since I'm going to use the scanner to get user input! Add the following line at the very start of your **main method**:

```
Scanner user_input = new Scanner(System.in);
```

It tells Java to create a label for a `Scanner` called `user_input` and attach it to a `new Scanner` that looks at `System.in`, which is where things your users type into your program end up! This process of creating the `Scanner` and giving it a label is called **declaring** it. You'll be **declaring** lots of things as you create this game!

3 Now you need to user your `Scanner` to collect and store your player's input! After your print line, where you tell the player to guess, add the following code:

```
String user_guess = user_input.next();
```

What this does is **declare** a new `String` called `user_guess`, which is a piece of text, and tells `user_input` to collect the next text the player types and store it in that `String`.

4 Now time to check that you're getting the player's input! Add another line, just after the last one, that prints out what the user typed:

```
System.out.println("You guessed: "+user_guess);
```

Notice how you can stick two `Strings` together with a `+`. Now, run your code and type in a guess. See what happens!

1 Next, you need to actually toss a coin! This is a lot like the last card: you'll need to do some importing and declaring.

What you're going to use as a "coin" is a random number generator—a piece of code that's in Java already which picks numbers with an equal chance of choosing any number between 0 and some higher number that you tell it. Of course, since a coin only has two sides, you'll be picking between 0 and 1. Start out with the `import` line, just after your previous `import` line:

```
import java.util.Random;
```

2 Now, just like with the `Scanner`, you'll need to create a `Random` you can use. At the start of your **main method** add this line:

```
Random coin = new Random();
```

3 Now you need to actually toss the coin and store that result somewhere! Just like with the **Scanner** you're going to do both of those in one line, like this:

```
int toss = coin.nextInt(2);
```

Notice a couple of things here:

- The number that you gave Java is 2. Java includes the lower number (0) but not the higher number (2) when picking a number, so the numbers between 0 and 2 are 0 and 1.
- While you put the result from the `Scanner` into a `String`, you store the result from this `Random` in an `int`, which is short for **Integer**. An integer is a whole number, so this `Random` won't ever give you fractions or decimals. You can make one that will if you need to though!

4 Go ahead and print out the result of your coin toss to check it's working. Run the program a few times to see that the number really is random, then delete the line that prints out the result. You'll add something back later, but you want it to say "heads" or "tails" not "1" or "0"!

1 So now you have a random coin, and it lands on either heads (1) or tails (0) every time the program runs. In order to have it tell the player which way the coin lands, you need to translate those numbers into words. You can do this using **conditional** or `if` statements. Something like "*If* the coin value is 1 *then* print 'heads', otherwise print 'tails'".

2 In Java an `if` statement has a set of brackets after it in which you put your **condition**. Java reads the **condition** and decides if it's **true** or **false**. If it's **true**, it runs the code in the curly braces (`{ }`). If it's **false** it skips over that code and runs whatever comes after it.

3 So, to start with, get your code to print out that the coin landed on heads when you get a 1, or tails for a 0 by adding this bit of code after you've got that random `int`:

```
if (toss == 1) {
    System.out.println("The coin landed on heads.");
}

if (toss == 0) {
    System.out.println("The coin landed on tails.");
}
```

Notice that you use two equals signs `==` to check if the values on either side of them are the same.

4 Now you need to take the user's guess and turn it into a number, so you can compare it to `toss`. You use an **if statement** to check *if* the user's guess was a *h* and set a new **variable**, an `int` called `guess_number`, to *1* if it is. Then do the same for *t* and *0*. Add this to the end of the code in your **main method**.

```
int guess_number;
if (user_guess.equals("h")) {
    guess_number = 1;
}

if (user_guess.equals("t")) {
    guess_number = 0;
}
```

Notice that to compare **Strings** you have to use `name.equals(thing_to_compare_to)`. The thing in brackets can be the label for another **String** or it can just be a **String** of text, like in the example above. Also, notice that `guess_number` is created without giving it a value. While you've been creating and giving values to **variables** at the same time so far, you can do it separately and it's helpful to do so when the value depends on another value. Finally, notice that you only need to use `int` when *creating* the **variable**.

5

Ok, now it's time to tell the player if they guessed correctly! Check their guess against the toss and tell them if they won or lost!

Notice that **not equal** is `!=` in Java.

```
 if (toss == guess_number) {
     System.out.println("You guessed right!");
}

if (toss != guess_number) {
       System.out.println("You guessed wrong! Game over!");
}
```

Now play your game! On the next card, you'll see how to keep playing as long as you keep guessing correctly!

1 One of the things that makes computers really powerful is that you can get them to run the same code over and over again. Forever, if you really want to. This is done with **loops**, which are pieces of code that run until a certain **condition** isn't true (if it's never true, the loop will never run!).

2 You're going to use a `while` loop here, which will do the same task over and over, *while* some condition is true. In this case, *while* the player's last guess was right. You're going to need to store a `true` or `false` value that tells you whether that's the case. These are called **boolean** values, and are stored in a **variable** type of the same name. So, up at top of your **main method**, where you're creating all your variables, add one more like this:

```
boolean user_right = true;
```

The value has to start as **true** or the **loop** you're creating will never run!

3 Now, just after all of the variables at the start of your **main method** are created, add your `while` **loop** like this:

```
while (user_right) {


}
```

4 Select all of the code below the loop and drag it *into* the loop. Try running it. It should keep going forever, whether the player wins or loses (since you never set `user_right` to false!) and you'll need to use the stop button to end the program.

5 Breaking out of the loop and ending the program is easy. You just need to add a line to the **if statement** that handles the player guessing incorrectly, like this:

```
if (toss != guess_number) {
        System.out.println("You guessed wrong! Game over!");
        user_right = false;
}
```

6 That's it! You've got a complete game! Now run it, test it and try to break it! There are a bunch of things you can do as a player that this game can't handle properly! You'll see how to take care of some of them in the Intermediate **Java** Sushi Cards!