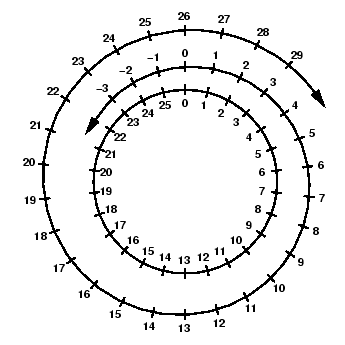
CS Lab 11

**Primitives, Casting, and Modular Arithmetic**

# Part 1:

**Filename**: MakeChange.java

Using modular arithmetic, please create a method to determine the   
proper way to break an amount of money into change.

String makeChange(double money)

Should return a String with the optimal number of the following, separated by  
spaces:

Hundreds, fifties, twenties, tens, fives, singles, quarters, dimes, nickels, and pennies.

So, makeChange(157.83)

Should return: “1 1 0 0 1 2 3 0 1 3”

Here are some mod outputs that suggest how to accomplish this:

493.54 %100

93.54 %50

43.54 %20

3.54 %10

3.54 %5

3.54 %1

0.54 %.25

0.04 %.10

0.04 %.05

0.04 %.01

0.00

**HINT:** *$100 bills are a special case, and you won’t be able to take the same modular approach as you can with all of the other bills and change. This is because there may be more than 100 $100 bills. (By contrast, there would never be more than 10 $10 bills). We have another process that can easily derive 10000 from 1000007 and 100 – can you think of what this is? And we have yet another process to convert 1000007.93 to 1000007.*

**HINT 2:** *0.01 is not representable in binary, so that number gets rounded inside your computer. It could be something like 0.01000000004 or something like 0.0099999999996! If you are finding your program shorts you by a penny about half of the time, this could be going on. What could you add to your money amount to prevent a problem like this?*

# Part 2:

**Filename**: Encryption.java

Here’s a simple form of key-based encryption.

The characters A-Z (capital letters only, no punctuation or spaces) have the char values 65-90, so the text “HELLOBOY” would have the values:

H E L L O B O Y

72 69 76 76 79 66 79 89

An *encryption key* is a piece of information necessary to encrypt some information, and a *decryption key* is information needed to decrypt it. The ENIGMA system in WWII was a German system of encoding that the allies cracked, allowing the Allies to read the messages passed back and forth by the Germans. Winston Churchill credited this with the Allied victory in the war.

The German system involved a simple rotating cypher, but with a twist. In a rotating cypher, each letter is mapped to another letter. In the simplest case, A would be mapped to B, B to C, C to D… Z to A. So, HELLO would become IFMMP. It sounds like a simple OriginalLetter + 1 formula, but because of the Z -> A mapping, the formula for such a cypher would involve modular arithmetic:

NewLetter = (OriginalLetter + 1) % 26.

Of course, the alphabet doesn’t start at 0 – it starts at 65. So, we must subtract 65 first, perform our mod, and then add the 65 back in. So, the new formula becomes

Newletter = ((OriginalLetter + 1 - 65) % 26) + 65

In this simple system, 1 is the key for both encrypting and decrypting the text. Of course, with a variable key, we could make this more complex. So, HELLO with a key of 1 would be IFMMP, a key of 2 would be JGNNQ, and a key of 3 would be KHOOR. A key of 26 would wrap back around to the original letters, HELLO.



**Figure 1**- Enigma machine

Remember that we promised that ENIGMA had a twist? The twist of the ENIGMA machine was that there were actually 26 different rotating cyphers, and a new cypher was chosen for each letter based on the output of the previous letter. Ouch!

We can imitate a simplified version of this system by adding a sum of the previous letter and the previous cypher key. So, assume that the first key is 1. H would become I. H has the ascii value 72, so for the next letter (E, ASCII 69), we would use 72 + 1 (our previous key value) as the new key. So, (69+(***72+1***)-65)%26 +65 = 77 % 26 + 65, which would be 25 + 65 = 90, letter Z.

The next letter, L (76), would have the previous key plus the previous letter, so **73+69**:

(76+(**73+69**)-65)%26 + 65 = 88, which would be ‘X’.

Under this cypher, HELLO would become IZXVW.

Your assignment: Make a Cypher class that will take an initial key and a String that is 8 characters long. Your job is to create two methods:

public String encrypt(int key, String text)

public String decrypt(int key, String text)

*Here are some values you can use to test your methods:*

encrypt (1, "HELLOBOY") should return IZXVWKLW, so decrypt (1, "IZXVWKLW") should return HELLOBOY

encrypt (14, "ARGUMENT") should return OSLSRIIO, so decrypt (14, "OSLSRIIO") should return ARGUMENT.

encrypt(7, "AARDVARK") 🡪 HUYOWJNK

encrypt(9, "PROGRAMS") 🡪 YCDWANMR

encrypt(11, "ZOOPHYTE") 🡪 KLMOITZQ

encrypt(1, "ZOOPHYTE") 🡪 ABCEYJPG

If the String coming in does not conform to the specification, the result will be a little strange:

encrypt(11, "Eats egg") 🡪 PIUFLIHG, but decrypt(11, "PIUFLIHG") 🡪 EGZYTKMM.

A few hints:

1. I gave you the formula for encrypt, but you will have to figure out decrypt on your own.
2. Encrypt and decrypt must be given the same key to work right.
3. Mod with negative numbers doesn’t work right in Java, but you can get a mathematically proper a % b with the formula ((a % b) + b) % b

# Part 3: (Optional Code Golf Challenge!)

**Filename**: C.java

Get the same output as part 1. There are no restrictions on what actions you can take!