# Make Up Problem Set
# Total Points: 30

**Choose 3 of the 5 problems below for full credit. See Java Grading Guide Sheet for grading/submission information. Partial credit will be given. (10 points each)**

1. The ultimate math game: Four is Cosmic. This simple number pattern has stumped many mathematicians. The pattern goes as follows: one hundred is ten, ten is three, three is five, five is four, and four is cosmic. See the pattern? The next term in the sequence is the number of letters in the current term. Write a Java program, given an initial term, trace the pattern until it terminates at "4 is cosmic". Do not consider spaces or commas part of a number's character count. There should also be no "and " in the spellings of any numbers. The user will enter an integer between `0 <= x <= 1000000.` Output one step of each pattern on each line, ultimately printing out "`four is cosmic`" after each terminates. Refer to the sample output below.

**Sample Runs (2):**

```
Enter a number (0- 1000000): 100

one hundred is ten
ten is three
three is five
five is four
four is cosmic


Enter a number (0- 1000000): 1

one is three
three is five
five is four
four is cosmic
```

Name the program: `CosmicNumbersXX.java`, where XX are your initials.

2. The Federal Bureau of Investigation (FBI) has recently changed its Universal Control Numbers (UCN) for identifying individuals who are in the FBI's fingerprint database to an eight-digit base 27 value with a ninth check digit. The digits used are:

```
0123456789ACDEFHJKLMNPRTVWX
```

Some letters are not used because of possible confusion with other digits:

```
B->8, G->C, I->1, O->0, Q->0, S->5, U->V, Y->V, Z->2
```

The check digit is computed as:

```
(2*D1 + 4*D2 + 5*D3 + 7*D4 + 8*D5 + 10*D6 + 11*D7 + 13*D8) mod 27
```

Where $D_n$ is the $n^{th}$ digit from the left. This choice of check digit detects any single digit error and any error transposing an adjacent pair of the original eight digits. Write a Java program to parse a UCN input by a user. The program should accept decimal digits and any capital letter as digits. If any of the confusing letters appear in the input, replace them with the corresponding valid digit as listed above. The program should compute the correct check digit and compare it to the entered check digit. The input is rejected if they do not match otherwise the decimal (base 10) value corresponding to the first eight digits is returned. The user will input a single line containing 9 decimal digits or capital (alphabetic) characters. The output will be one line consisting of the UCN number entered followed by the string `"Invalid"` (without the quotes) or the decimal value corresponding to the first eight digits. Refer to the sample output below.

**Sample Runs (2):**

```
Enter the UCN: 12345678A

12345678A- 11280469652


Enter the UCN: 12435678A

12435678A- Invalid
```

Name the program: `FBINumbersXX.java`, where `XX` are your initials.

3. Write an OO Java application to play the card game *Go Fish!* -- human against computer.

https://en.wikipedia.org/wiki/Go_Fish

The game starts by shuffling the deck and dealing 5 cards to each player. The players then alternate turns. On each player's turn, the game should ask him/her for the rank of a card to ask for. If the other player has any cards of that rank, they are transferred to the current player's hand. If the other player does not have any cards of that rank, the current player must "Go Fish" by drawing. If, at the end of the turn, a player has four cards of the same rank, they form a book, and are removed from the player's hand. The game ends when the deck is empty: the player with the greatest number of books wins.

The program should indicate when a player completes a book of four cards of the same rank: It's player 2's turn. The game ends when there are no more cards in the deck. The program should determine which player has more books and announce the winner of the game.

**Implementation:**

Some classes that will be needed are specified below and additional classes can be used. It is up to the student to figure out what the responsibilities (methods) of each class will be. Those classes are:

- `Fish`(object) - to represent the entire game. Creating a `Fish` object should start the game.
- `Card`-simulates one playing card (a suit and value)
- `Deck`-simulate a deck of 52 cards
- `Player` (object) - to represent a single player.

Also, no object should directly examine or alter the variables of any other object; all access should be done by talking to (calling methods of) the other object. The student should figure out what the various classes and objects should be and what their instance variables and their methods should be. There is no single "correct" design; any reasonable design will do. The better the design, the easier the programming will be. Use the objects defined to play a game of *Go Fish!* The computer player must play legally, but does not need to play well. Try to make the computer player "smart," so the human player does not always win. Use appropriate controls (labels, buttons, etc.). in your application. Card deck provided by your instructor in a file named `deck.zip`. Place all classes into one file. Output should be user friendly.

Name the program: `OOGoFishXX.java`, where XX are your initials.

4. Write a Java FX GUI application that draws a random colored stop sign. Make sure the program has an outer white stripe. Set the title to `Stop Sign`, and center the stage on the screen. Place all classes into one file. Do not use an image.


Name the program: `GUIStopXX.java`, where XX are your initials.

5. Write a Java program that uses recursion to determine one's age. The user will enter the current year and the year they were born (birth year) from the keyboard. The program will output the current age of the user. Check positive and zero cases only. A recursive function must be used for full credit. No recursive helper functions may be used as well. Output should be user friendly.

Name the program: `TestRecAgeXX.java`, where XX are your initials.