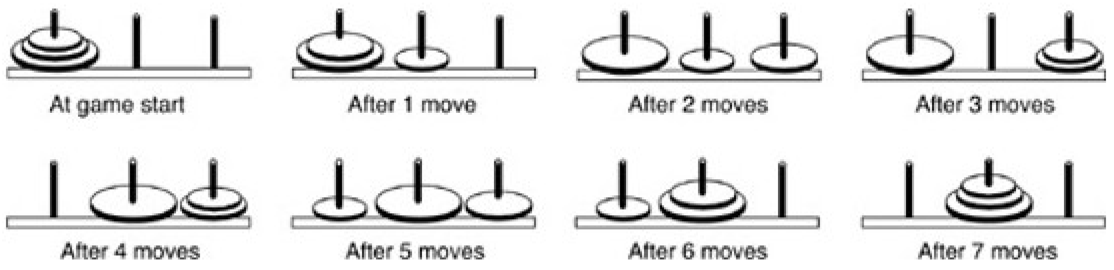
Ring Stack game

You are to create the game for playing the Ring Stack game. Your program will not play the game itself, but will allow the user to play the game.

This is how the game works:
There are 3 pegs. On the first peg are some number of rings. Each ring is a unique diameter. At the start of the game, all the rings are on the first peg and they are stacked in order of the largest diameter on the bottom to the smallest diameter on the top.

The object of the game is to get all of the rings on to peg number 2 stacked in the same order, that is from the largest diameter ring on the bottom to the smallest diameter ring on the top.



The rules for playing the game are thus:
1. Only one ring may be moved at a time.
2. A larger ring may never be stacked on top of a smaller ring at any time.
3. Only the top ring on a stack may be moved.
4. A ring can be moved to any peg, so long as rule number 2 is not violated.
5. A ring must be moved to a different peg than the one it is currently on.

We will number the rings, and the ring number will represent the diameter of the ring in inches. So, ring number 1 will be 1 inch in diameter, ring number 2 will be 2 inches in diameter, ring number 3 will be 3 inches in diameter, etc.

When I play the game I should see some representation of the 3 pegs and the rings that are on them. So if I am playing the game with 5 rings, at the start of the game I should see something like this:

```
peg 1   peg 2   peg 3

  1
  2
  3
  4
  5
```

There should be a question asking if I want to continue. If so then there should be a question asking me which peg to move the top item from and which peg to move it to. Then, after entering that information, I should see the resulting status of the pegs and rings:

```
peg 1   peg 2   peg 3

  2               1
  3
  4
  5
```

There should be a question asking if I want to continue. If so then there should be a question asking me which peg to move the top item from and which peg to move it to. Then, after entering that information, I should see the resulting status of the pegs and rings:

```
peg 1   peg 2   peg 3

  3       2       1
  4
  5
```

There should be a question asking if I want to continue. If so then there should be a question asking me which peg to move the top item from and which peg to move it to. Then, after entering that information, I should see the resulting status of the pegs and rings:

```
peg 1   peg 2   peg 3

  3       1
  4       2
  5
```

And so on until the game is finished. The game will end when I no longer want to make a move.

You will create three classes, each in a separate file: a class named Peg, a class named Towers, and a class named TowersGame.

The Peg class will keep track of the rings stacked on a peg. The peg may hold up to 64 rings, with each ring having its own diameter. The class's methods should include:

a) a constructor that places n rings on the peg (where n may be as large as 64), and these n rings have diameters from n inches on the bottom to one inch on the top;
b) an accessor method that returns the number of rings on the peg;
c) an accessor method that returns the diameter of the topmost ring,
d) a method that adds a new ring to the top (with the diameter of the ring as a parameter to the method);
e) a method that removes the topmost ring.

Make sure that all methods have appropriate preconditions to guarantee that the rule about ring sizes is enforced. Create private instance variables as needed and appropriate.

Note: All methods should have preconditions and postconditions. Preconditions are conditions that must be met before the method is called. These are not conditions that are validated inside the method. Postconditions are conditions that are true after the method has finished executing assuming the preconditions were met.

The Towers class must keep track of the status of all three pegs. You might use an array of three pegs, where each peg is an object of the Peg class. The towers class methods are:

Towers(int numRings)
Precondition: 1 <= numRings <= 64
Postcondition: The towers have been initialized with numRings on the first peg and no rings on the other two pegs. The diameters of the first peg's rings are from one inch (on the top) to numRings inches (on the bottom).

int countRings(int pegNumber)
Precondition: pegNumber is 1, 2, or 3
Postcondition: The return value is the number of rings on the specified peg.

int getTopDiameter(int pegNumber)
Precondition: pegNumber is 1, 2, or 3
Postcondition: If countRings(pegNumber) > 0, then the return value is the diameter of the top ring on the specified peg; otherwise, the return value is zero.

void move(int startPeg, int endPeg)
Precondition: stargPeg is a pegnumber (1, 2, or 3), and countRings(startPeg) > 0; endPeg is a different peg number (not equal to startPeg), and if endPeg has at least one ring, then getTopDiameter(startPeg) is less than getTopDiameter(endPeg).
Postcondition: The top ring has been moved from startPeg to endPeg.

void displayTowers()
Precondition: the game has started
Postcondition: a display indicating each peg and the stack of rings on each peg has been displayed with the smallest ring shown on top.

The TowersGame class will contain the main method for the program which will contain the game loop. The main method will ask the user how many rings does he/she wish to play with. The main method will contain a loop that will loop as long as the user wishes to keep making moves. There will also be a method to validate that the game rules have not been violated in each move:

boolean validMove(Towers game, int startPeg, int endPeg)
Precondition: game is a reference to the Towers object; startPeg contains the peg number that the ring is being moved from; endPeg contains the peg number that the ring is being moved to.
Postcondition: It has been determined whether or not the appropriate game rules have been violated. If any of them have, an appropriate error message is displayed and false has been returned. If no rules have been violated, true has been returned.

Do not use recursion in this assignment.