The YuleMachine 🎄

Xmas themed coding challenges

A person sitting at a computer

Description automatically generated

Here are 24 Xmas themed coding challenges, which begin with very simple challenges, and finish with some where more advanced algorithms have to be coded to solve the tasks.

Along with 24 empty methods, which you need to fill with merry holiday code, there is a suite of unit tests, which have to pass in order for the exercises to have been completed:

A screenshot of a computer program

Description automatically generated

# Introduction to the challenge

## Initial code

You will find a Visual Studio solution at <https://github.com/xnafan/YuleCoding> with two projects, YuleCoding.Code and YuleCoding.Tests. They are described in further detail here:

## YuleCoding.Code

This project contains three classes, YuleMachine, YuleMachineSolved and Program:

**YuleMachine**

Is a class with 24 methods which all throw a NotImplementedException.

For example:

|  |
| --- |
| **public bool IsItChristmasDay(int dayOfDecember) {  throw new NotImplementedException(); }** |

Your task is to replace the exception with an implementation that will make the tests pass.

**YuleMachineSolved**

This class contains suggested solutions to the exercises.

**Program**

Contains the Main() method for running two of the last exercises where you draw using ASCII art:

|  |  |
| --- | --- |
| a sky filled with snowflakes | and a Christmas tree 😊 |

**Note:** to test your code, *run the tests*, NOT the program.   
The program is only there to display the results of the last two exercises 😉

## YuleCoding.Tests

## Tests

The solution includes a complete NUnit test-suite which will fail initially, but as you implement the methods, they will go from red to green. Remember the Visual Studio test-shortcuts:

* Run current test CTRL+R -> CTRL+T
* Debug current test CTRL+R -> T
* Run all tests CTRL+R -> CTRL+A
* Debug all tests CTRL+R -> A

## Helper classes

Some of the exercises use model classes like **Gift, Person, Recipe** or **Santa**.  
These classes are all to be found in the [**YuleCoding.Code.HelperClasses**](https://github.com/xnafan/YuleCoding/tree/master/YuleCoding.Code/HelperClasses) folder and namespace.

## Hints and solutions

Each exercise has three hints below it with white text on white background. If you want assistance in solving an exercise, you can select the text and change its color to black (Word shortcut: CTRL+SPACE). This way you can try as much as possible for yourself instead of getting the complete solution. Note that you may also look in the unit tests, to try and figure out how to code an implementation, but the suggested solution may have been used in the unit tests for verification, so that way you may end up with more help than you wanted.

If you want to run the solution, to see the last two exercises print snow and a Christmas tree on the console, just instantiate the YuleMachineSolved instead of the YuleMachine in the Program.Main() method.

### Suggested approach

Start with the first few exercises to warm up and get the hang of the structure, testing, etc. and then go for those exercises you think sounds most fun/challenging/christmassy/whatever-you-feel-like 😄🎄

It is quite a challenge to solve them all, and some of them are quite tricky, so if you get stuck, maybe try another exercise and then come back to the one that was giving you problems.

1. **IsItChristmasEve Method**

Write a method that determines if a given day in December is Christmas Day. This method should take an integer representing the day of December and return true if it's the 24th (Christmas Eve), and false otherwise.

*Hints*:

1. Think about a simple comparison operation to check equality.
2. Use a return statement that directly checks if the input matches 24.
3. Your method should return true when dayOfDecember equals 24 and false otherwise.
4. **ChristmasGreeting Method**   
   Create a function that returns a personalized Christmas greeting. The method should take a name as input and return a string that says, "Merry Christmas, [Name]".

*Hints*:

1. You can concatenate strings using + or use string interpolation.
2. Ensure your output format includes "Merry Christmas," followed by the input name.
3. Use string interpolation or concatenation to join "Merry Christmas, " with the provided name.
4. **IsItChristmasEveYet Method**   
   Implement a method that tells you if it's Christmas yet. This method should take an integer representing the day of December and return "Not yet!" for days before the 24th, "Yes!" for the 24th, and "It's over!" for any day after. Make sure you return those exact strings – case sensitive, for the tests to succeed.

*Hints*:

1. Consider using a switch statement or if-else conditions.
2. Ensure you have separate conditions for days before, on, and after the 24th.
3. Use a switch statement with cases that handle the three possible outcomes.
4. **ChristmasPoemChecker Method**   
   Write a method that checks if a given string is a festive Christmas poem. The method should return true if the poem contains the words "christmas," "code," and "snow" (case-insensitive), and false otherwise.

*Hints*:

1. Use Contains to check if a word exists in the string.
2. Make sure to account for case insensitivity when searching for words.
3. Use StringComparison.OrdinalIgnoreCase with Contains to ensure case doesn't matter.
4. **HowManySantas Method**   
   Create a function that counts how many times the word "Santa" appears in a given string. The method should return an integer representing the number of occurrences.

*Hints*:

1. Use Split to separate the string into parts.
2. Count the number of times "Santa" appears by checking the length of the resulting array.
3. Subtract 1 from the length of the split array to get the exact count of "Santa" occurrences.
4. **WhosPositive Method**Implement a method that takes a collection of integers and returns how many of the numbers are positive (positive - and ready for the holidays 😄)

*Hints*:

1. Use Count to filter and count items in the list.
2. Check if each number is greater than zero when counting.
3. Use a lambda expression inside Count to simplify the condition.
4. **FestiveMaker Method**   
   Code a method that takes a string and returns a new string where each space is replaced with a Christmas tree emoji (🎄). Copy and paste this tree emoji for use in the return value.

*Hints*:

1. Use Replace to modify the text.
2. Specify the space character as the first argument in Replace and the emoji as the second.
3. Replace each space in the input string with "🎄" and return the result.
4. **EvenFestiverMaker Method**   
   Create a method that takes a string and replaces certain words with festive alternatives.   
   Replace "rain" with "snow", "sad" with "jolly", and "bill" with "gift" to ensure that the holidays are more joyful 😊

*Hints*:

1. Use multiple Replace methods sequentially.
2. Ensure each Replace call targets a specific word and its replacement.
3. Chain Replace calls to transform "rain" to "snow," "sad" to "jolly," and "bill" to "gift."
4. **NiceAndNaughtyLists Method**   
   Write a method that takes a collection of [Person](https://github.com/xnafan/YuleCoding/blob/master/YuleCoding.Code/HelperClasses/Person.cs) objects and separates them into "Nice" and "Naughty" lists based on their NiceOrNaughtyStatus.   
   Return a dictionary with two keys for "Nice" and "Naughty" and the lists with corresponding, separated Nice or Naughty people as the values.

*Hints*:

1. Use Where to filter the collection based on conditions.
2. Create lists for "Nice" and "Naughty" by checking the NiceOrNaughtyStatus.
3. Populate a dictionary with two keys: one for "Nice" and one for "Naughty," each containing the corresponding lists.
4. **WhatsThePriceOfChristmas Method**   
   Implement a method that calculates the total price of a list of [Gift](https://github.com/xnafan/YuleCoding/blob/master/YuleCoding.Code/HelperClasses/Gift.cs) objects.   
   The method should return the sum of the Price property of all gift objects.

*Hints*:

1. Use Sum to add up the prices of all gifts.
2. Make sure to access the Price property of each Gift.
3. Call Sum directly on the collection with a lambda expression that extracts Price.
4. **IsChristmasTooExpensive**  
   Create a method that checks if Santa's gift expenses exceed the maximum budget allowed. This method is very similar to the previous WhatsThePriceOfChristmas, and you may call the functionality in that method to solve this one.

*Hints:*

1. Start by calculating the total price of the gifts using a method like Sum.
2. Compare the total sum against the maxBudget parameter.
3. Return true if the total exceeds the budget and false otherwise.
4. **CalculateSantaLetterNiceness**   
   Develop a method that calculates the "niceness" score of a letter to Santa Claus based on how many times certain words appear in the text.

For every “please” add 2 to the niceness score.

For every “thank you” add 3 to the niceness score.

*Hints:*

1. Use Split to count the occurrences of words like "please" and "thank you".
2. Assign specific scores to these words and sum them up.
3. Lowercase the letter text before counting to ensure case insensitivity.
4. **HowLongTimeToBakeCookies**   
   Write a method that calculates the total time needed to bake cookies, considering setup and cleanup.  
   The method receives a list of [Recipe](https://github.com/xnafan/YuleCoding/blob/master/YuleCoding.Code/HelperClasses/Recipe.cs) objects which has a TimeToMake property with the minutes needed for that recipe. You need to add 10 minutes *between* each recipe, to set up the kitchen for the next recipe and 30 minutes at the end to clean up the kitchen.

*Hints:*

1. Sum the time for each recipe and add the setup and cleanup times.
2. Calculate the additional time required between recipes.
3. Don’t forget to account for a fixed cleanup time at the end.
4. **HowFarHaveTheReinDeerTravelled**   
   Create a method that sums up the total distance the reindeer have traveled.  
   The method will receive a list of [Mood](https://github.com/xnafan/YuleCoding/blob/master/YuleCoding.Code/HelperClasses/Mood.cs) values taken from Santa’s diary.

Based on the moods, you have to estimate how far they have travelled. Here are the values for the Mood enum:

*Sleeping = 0, Drowsy = 1, Fresh = 2, Happy = 3, Drunk = -1, Hungover = -2*

*Hints:*

1. Use Sum on the collection to calculate the total.
2. Ensure that distances is an enumerable collection of integers.
3. Use casting if the input is not directly an integer list.
4. **CalculateAmountOfGiftWrappingPaperNecessary**  
   Calculate the absolute minimal amount of wrapping paper needed to cover a gift with given dimensions.

*Hints:*

1. Apply the formula for the surface area of a box.
2. Sum the surface areas of all pairs of sides.
3. Use Height\*Width + Height \* Depth + Width \* Depth to find the three types of surfaces and multiply it by two, since there are two identical surfaces for each type.
4. **SortGiftsByToAndFrom**   
   Sort a list of [gifts](https://github.com/xnafan/YuleCoding/blob/master/YuleCoding.Code/HelperClasses/Gift.cs) alphabetically first by recipient and then by sender, so multiple gifts to the same recipient (the “To:” on the gift card) are positioned after each other and sorted by the giver (the “From:” on the gift card.

*Hints:*

1. Search online for LINQ methods for sorting.
2. Use OrderBy and ThenBy
3. Return the sorted list as an IEnumerable<Gift>.
4. **SortSantasReindeer**   
   This method ensures that Santa's reindeer (*Dasher, Dancer, Prancer, Vixen, Comet, Cupid, Donner, Blitzen, Rudolph*) are pulling Santa’s sleigh in an orderly fashion, by sorting them alphabetically. If Rudolph is on the list for that day, ensure Rudolph is first in the list. He takes the job of leading the flock very seriously 😉.

*Hints:*

1. Use the LINQ expression OrderBy for general sorting.
2. Check if "Rudolph" is present and handle it separately.
3. If Rudolph is present, insert him at the start of the list after sorting, and make sure to remove him from the position he was in previously, so he doesn’t show up in two places.
4. **DaysTillNextChristmas**   
   Create a method that calculates the number of days until the the next Christmas eve. Don’t worry about the time of day – just use whole days.

If the date is December 21st, then you should return 3

If the date is December 24th, then you should return 0

If the date is December 31st, then you should calculate the number of days till next December 24th.

*Hints:*

1. Use DateTime.Now to get the current date. Look at the DateOnly class, as this might simplify things.
2. Check if the current date is before or after December 24th and find out which 24th you should use for the calculation.
3. Calculate the difference in days between today and December 24th of this or next year. You can use the DateOnly.DayNumber to get the number of days since January 1st year 1 and use this for the calculation by subtracting today from next christmas
4. **ValidateSantasRouteReturnsToStart**   
   Ensure that Santa's route in any given city (not on the north pole or south pole) returns him to the starting point using directions. You receive a list of his movements North, South, East and West in the form of a List of char variables ‘N’, ‘S’, ‘E’, ‘W’.   
   Santa has returned to his starting point if
   1. there are an equal number of movements East and West and an equal number of movements North and South.
   2. The route is empty (has no directions)

You must throw an ArgumentException if an invalid direction (char that isn’t N,S,E,W) is in the list of directions passed to the method.

*Hints:*

1. Count how many times each direction appears.
2. Ensure the number of 'N' matches 'S' and 'E' matches 'W'.
3. Throw an exception for any unrecognized characters.
4. **DrawChristmasRaffleWinners**   
   Select a set number of winners randomly from a list of participants. You receive a list of names and how many winners should be selected. You must return that number of winners’ names with no duplicates.

You must throw an ArgumentException if

* The list of names is empty
* The list of names contains fewer than the number of winners to draw *or*
* The number of winners is less than 1

*Hints:*

1. Use Random.Shared.Next to pick random names from the list.
2. Ensure winners are unique by checking against the winners list.
3. Continue drawing until the required number of winners is reached, using a while loop.
4. **ChristmasCounting**  
   Implement a method that returns specific holiday greetings based on the number passed to it.

* If divisible by only 3, return "Ho".
* If divisible by only 5, return "HoHo".
* If the number is divisible by both 3 and 5, return "Happy Holidays!".
* If neither, return the number as a string.
* If the number is outside the range of 1 to 24, return "BAH, Humbug!".

*Hints:*

1. Check divisibility using the modulus operator (%).
2. Return early if both conditions for divisibility by 3 and 5 are true.
3. In your code make the checks from most specific (divisible by both 3 and 5) to least specific, so any if-else structure doesn’t terminate on the first match.
4. **CreateSecretSantaList**  
   Create a method that generates a Secret Santa pairing list. Each participant is assigned another participant as their recipient to give a gift to, ensuring a continuous loop where the last participant gives to the first.

The return value is a Dictionary<string, string> where each key holds the name of the giver and each value holds the name of the recipient.

If for example the argument sent to the method is {“Anna”, “Bob”, “Claire”}, a valid solution would be: {{“Anna”, “Bob”}, {“Bob”, “Claire”}, {“Claire”, “Anna”}}, so Anna would be Bob’s secret santa, Bob would be Claire’s and Claire would be Anna’s.

Additionally: throw an ArgumentException if there are fewer than two participants in the list you receive.

*Hints:*

1. Iterate through the list and create the pairs out of each name and the next one.
2. Make the last participant wrap around to the first to complete the circle.
3. Store the pairs in a Dictionary where the key is the giver and the value is the receiver.
4. **CreateAsciiSnow**  
   Develop a method that produces a string representation of snow on a grid of a given width and height, with snowflakes spaced at a specified distance. Throw an ArgumentException if width or height is less than 1.  
   The parameter distanceBetweenSnowFlakes is how often a character should be an asterisk. If the value is 3, every third character in the return string should be an asterisk, the remaining characters should be spaces. Make a newline character at the end of each line.

**Example:**

Given the parameters: width=25, height=15, distanceBetweenSnowFlakes=3, you should return a string like this with 25 characters in each line (and then a newline character), 15 lines and two spaces between each snowflake, so every third character is a snowflake.  
  
“\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \*

\* \* \* \* \* \* \* \* \*”

*Hints:*

1. Use a StringBuilder to efficiently construct the string.
2. Place snowflakes at positions where the index modulo distanceBetweenSnowFlakes equals 0. Modulo operations can be performed using the % operator:   
   bool drawSnowflake = characterCounter % distanceBetweenSnowFlakes;
3. Append newline characters after each completed row to format the grid correctly.
4. **CreateChristmasTree**  
   Create a method that returns a string representation of a Christmas tree of a given height. Each level of the tree should be centered and grow in width by two more characters than the previous level above it (top level: 1 asterisk, second level down: 3 asterisks, etc.). The base is a single vertical line (the pipe symbol: “|” ) which is added below and doesn’t count towards the height of the tree.

**Note:** *there are no spaces to the right of the tree. There is a newline character right after the rightmost asterisk on every line.*

Throw an ArgumentException if the height is less than 1.

**Example**

Calling the method with height 10 should return a tree with 10 lines that make up the tree and one additional line with the base of the tree like this

“ \*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

|”

All asterisks that make up the tree are centered using spaces to the left in the lines.

Calling the method with height=3 should return a tree like this:

“ \*

\*\*\*

\*\*\*\*\*

|”

*Hints:*

1. Use nested loops: one for the levels and one for constructing each level.
2. Add spaces to center-align each level.
3. Append the trunk after constructing all levels, using spaces for alignment.

🎄 Merry Xmas! / Happy Holidays! 🎄

Kind regards - Jakob Farian Krarup ([jfk@ucn.dk](mailto:jfk@ucn.dk))