Dongzhu Tan

Reflections on "Clean Code"

Naming (Chapter 2)

Name and explanation	Reflection and rules from Clean Code
WeatherDataFetcher Class for fetching weather data	**Class Names Should Be Noun or Noun Phrase** '**Fetcher**' acts as a compound noun where the verb acts as an adjective describing the kind of noun. So I think it is better to change the name to **'WeatherDataService',** that represents services rather than actions.
main Main function for module	**Use Intention-Revealing Names** The name could be more descriptive to reflect its specific action in the weather module. For a module, this can lead to confusion when imported by another application—such as with import { main } from *"./weather-module/src/app.js"*. To improve clarity and convey the specific action of the function, renaming it to '**fetchAndCalculateWeatherData()'** could be more descriptive and clear.
countAverageHumidity Method to calculate average humidity	**Method Names** The method name acts as a verb phrase, describing the action to calculate average humidity, which follows the recommendation for method naming in Clean Code. **Avoid Disinformation** I made a mistake here, it would be better to change this to **'calculateAverageHumidity'** to describe that the method performs calculations not for counting.
getCoordinates Method to retrieve coordinates	**Use Intention-Revealing Names** The method name is descriptive and reveals its intention—get the coordinates for the city/country that users ask for, it is easy to understand what the method does without needing additional comments. Så, changing it to **'retrieveCityCooperatives()'** would

	be better and clearer.
convertKelvinToCelsius Converts temperature from Kelvin to Celsius	**Method Names** The method name acts as a verb, describing the action taken by the method, which follows the recommendation for method naming in Clean Code.

Reflection:

After reviewing my code with the ideas from "Clean Code" I realize that while my intentions were clear, there's room for refinement to enhance understanding and maintainability.

1.In order to comply with "**Class Names Should Be Noun or Noun Phrase**" I'll transition from ‘WeatherDataFetcher’ to ‘WeatherDataService’ to emphasize the service-oriented nature of the class rather than an act of fetching.

2.In the spirit of "**Use Intention-Revealing Names**" I'll rename ‘getCoordinates()’ to ‘retrieveCityCoordinates()’ to better convey that the purpose of this method is not just to get any coordinates, it is specifically used to get the coordinates of a city.

3.For the ‘main()’ function, I'll employ **"Use Intention-Revealing Names"** by renaming it to ‘fetchAndCalculateWeatherData()’ . The idea is to avoid using the generic term 'main' , which can be ambiguous outside the context of the module, and instead clearly states the function's purpose — to calculate weather statistics.

4.Lastly, aligning with **"Avoid Disinformation"** once more, I'll change ‘countAverageHumidity()’ to ‘calculateAverageHumidity()’ to describe that the method performs a calculation not counting. "Calculating" might involve deriving statistical values, applying formulas, or performing algorithmic operations, while "counting" might simply mean determining the number of elements in an array or list, or counting the number of occurrences of a particular item.

Upon reflection, I recognize that names in my code, although functional, could be improved to follow "Clean Code" principles more closely. This would not only aid other programmers in understanding the code but also assist in future maintenance and enhancements.

Functions (Chapter 3)

Method name and link or code	Number of rows (not ws)	Reflektion
countAverageHumidity ()	10	**Do one thing** The method does only one thing - calculate average humidity. **Function Arguments** The method has only one argument (monadic) - the input total humidity at forecast for the next 40 days.
countAverageHumidity () **countMaximumRainfall ()** **countAverageWindSpeed ()** **countAverageTemperature ()**	79	**Don't Repeat Yourself** These functions inevitably repeat themselves - when calculating the average.
const maxRainfall = await weatherModule.countMaximumRainfall(rainfall)	1	**Use Descriptive Names** This function describes the visual representation of inputting 'rainfall' into '**countMaximumRainfall()**' that then calculates the maximum amount of rainfall.
const weatherModule = new WeatherModule(temperaturesInKelvin, humidities, windSpeeds, rainfall)	1	**Argument Objects** The constructor of the WeatherModule class is designed to accept four parameters, each of which is an object containing an array of data points for temperatures, humidities, wind speeds, and rainfall.

getCoordinates (country, city) {
 	
 try {
  //…
  } catch(error) {
   console.error()
 }
 	
}
	59	**Exception Handling**
		The function uses exceptions to indicate errors, which is better than returning error codes. This helps in separating error handling from the main logic.

Reflection:

Final summary, I think my code compares with the specification requirements for ‘Functions’ in Chapter 3 to a large extent.

However, considering that this is a simple model, some of my code has duplicates, such as the methods **"countAverageHumidity()", "countMaximumRainfall()" , "countAverageWindSpeed()", and "countAverageTemperature()"** represent situations where the *"Don't Repeat Yourself" (DRY)* principle seems to conflict with the functionality of the code. Each method here is responsible for calculating different indicators, but they share similar calculation patterns, especially the calculation of average and maximum values.

My doubt is: does this essentially involve duplication? Or is duplication here not a violation of the DRY principle, rather, it is a natural consequence of the problem domain, where each function computes different data. The commonality of their structure emphasizes a unified approach to data processing across different datasets, which is a positive feature in this regard, enhancing readability and maintainability.