Critical Thinking Assignment – Week 6

Todd Toler

Course Number (CSC505) – Principles of Software Development

Colorado State University – Global Campus

Dr. Steven A. Evans Sr.

February 20th, 2024

Check Writer Python

```python
# Level 1 - Highest level abstraction
def convert_amount_to_words(amount):
    """
    Convert a numeric dollar amount to words as would appear on a check.
    Example: 1234.56 -> "ONE THOUSAND TWO HUNDRED THIRTY-FOUR AND 56/100"
    """
    dollars = int(amount)
    cents = int(round((amount - dollars) * 100))
    return f"{convert_dollars_to_words(dollars)} AND {cents:02d}/100"

# Level 2 - Breaking down the dollar conversion
def convert_dollars_to_words(dollars):
    """
    Convert the dollar portion to words, handling different magnitude ranges.
    """
    if dollars == 0:
        return "ZERO"

    # Break into groups of thousands
    billions = dollars // 1000000000
    millions = (dollars % 1000000000) // 1000000
    thousands = (dollars % 1000000) // 1000
    remainder = dollars % 1000

    result = []

    if billions:
        result.append(f"{convert_hundreds_to_words(billions)} BILLION")
    if millions:
        result.append(f"{convert_hundreds_to_words(millions)} MILLION")
    if thousands:
        result.append(f"{convert_hundreds_to_words(thousands)} THOUSAND")
    if remainder or not result:
        result.append(convert_hundreds_to_words(remainder))

    return " ".join(result)

# Level 3 - Detailed conversion of numbers to words
def convert_hundreds_to_words(number):
    """
    Convert a number less than 1000 to words.
    """
    units = ["", "ONE", "TWO", "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "EIGHT", "NINE"]
    teens = ["TEN", "ELEVEN", "TWELVE", "THIRTEEN", "FOURTEEN", "FIFTEEN",
             "SIXTEEN", "SEVENTEEN", "EIGHTEEN", "NINETEEN"]
    tens = ["", "", "TWENTY", "THIRTY", "FORTY", "FIFTY", "SIXTY", "SEVENTY",
            "EIGHTY", "NINETY"]

    if number == 0:
        return ""

    result = []

    # Handle hundreds
    if number >= 100:
        result.append(f"{units[number // 100]} HUNDRED")
        number %= 100

```

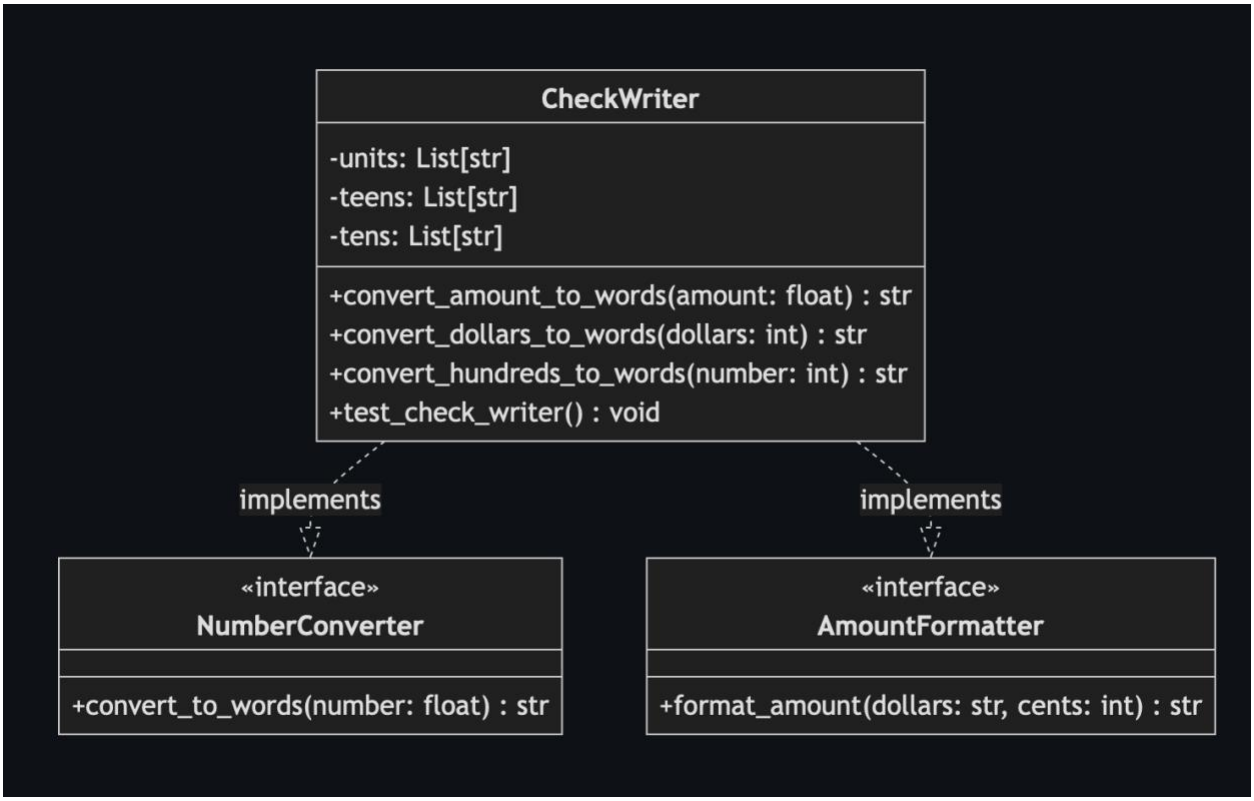
```
    # Handle tens and ones
    if number >= 20:
        ten_word = tens[number // 10]
        one_word = units[number % 10]
        if one_word:
            result.append(f"{ten_word}-{one_word}")
        else:
            result.append(ten_word)
    elif number >= 10:
        result.append(teens[number - 10])
    elif number > 0:
        result.append(units[number])

    return " ".join(result)

# Example usage and testing
def test_check_writer():
    """
    Test the check writer with various amounts.
    """
```

```
    test_cases = [
        0.00,
        1.23,
        45.67,
        100.00,
        1234.56,
        1000000.00,
        1234567.89,
        1000000000.00
    ]

    for amount in test_cases:
        print(f"\nAmount: ${amount:.2f}")
        print(f"In words: {convert_amount_to_words(amount)}")

if __name__ == "__main__":
    test_check_writer()
```

Code Execution

```
> /Users/toddtoler/github/school/csc505/venv/bin/python /Users/toddtoler/github/school/csc505/Week6/checkwriter.py

Amount: $0.00
In words: ZERO AND 00/100

Amount: $1.23
In words: ONE AND 23/100

Amount: $45.67
In words: FORTY-FIVE AND 67/100

Amount: $100.00
In words: ONE HUNDRED AND 00/100

Amount: $1234.56
In words: ONE THOUSAND TWO HUNDRED THIRTY-FOUR AND 56/100

Amount: $1000000.00
In words: ONE MILLION AND 00/100

Amount: $1234567.89
In words: ONE MILLION TWO HUNDRED THIRTY-FOUR THOUSAND FIVE HUNDRED SIXTY-SEVEN AND 89/100

Amount: $1000000000.00
In words: ONE BILLION AND 00/100
```

UML



Code Explanation

The check writer implementation demonstrates a sophisticated application of stepwise refinement and modular programming principles in handling financial text conversion. As noted by Pressman and Maxim (2021), stepwise refinement allows developers to "decompose a macro problem into successively smaller units" until reaching a level where the solution becomes straightforward to implement.

The program employs a three-tier abstraction hierarchy to convert numerical values into their written equivalents. At its core, the implementation utilizes string manipulation and mathematical operations to break down complex numbers into manageable components. This approach aligns with Martin's (2018) principle of single responsibility, where "a class should have only one reason to change," as demonstrated by the distinct methods handling different numerical ranges.

The first tier handles the overall conversion process, separating dollars from cents. The second tier manages the magnitude ranges (billions, millions, thousands), while the third tier handles the detailed conversion of numbers less than 1000 into words. This hierarchical structure implements what's known as separation of concerns, making the code both maintainable and extensible.

Particularly noteworthy is the program's handling of edge cases, such as zero values and teen numbers (11-19), which require special processing. The implementation also maintains consistency in output formatting, ensuring that all monetary amounts are represented in a standardized format suitable for financial documents.

Code GitHub

All code and picture assets are available at this GitHub

https://github.com/msfttoler/csc505/tree/main/Week6

References:

Martin, R. C. (2018). Clean architecture: A craftsman's guide to software structure and design (2nd ed.). Prentice Hall.

Pressman, R. S., & Maxim, B. R. (2021). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill Education.