Name : JAY PATEL (RedID :824672518)

```
import sys
import unittest

# Implemented Flyweight factory that returns Flyweight character object
# using __sizeof__ method (which holds the method to calculate memory
# hold by data member
class CharacterBuildingFactory():

    def __init__(self):
        self.character_array = {}

    def __sizeof__(self):
        size_of_array = 0
        for character in self.character_array:
            size_of_array += sys.getsizeof(character)
            size_of_array += sys.getsizeof(self.character_array[character])
        return size_of_array

    # character will reuse itself, rather than creating new one.
    def get_character(self, character):
        letter = None

        if character in self.character_array:
            letter = self.character_array[character]
        else:
            letter = Character(character)
            self.character_array[character] = letter

        return letter

    def get_character_array(self):
        return self.character_array

# this class makes character an object to be used further in the code
class Character():
    def __init__(self, character):
        self.character = character

    def __sizeof__(self):
        size_of_array = 0
        size_of_array += sys.getsizeof(self.char)
        return size_of_array

    # set the character value
    def set_character(self, character):
        self.character = character

    # return the character value
    def get_character(self):
        return self.character

# this returns the FlyWeight Character Object
class Font():
    def __init__(self, word, size_of_array, mode):
        self.word = word
        self.size_of_array = size_of_array
        self.mode = mode

    def __sizeof__(self):
        size_of_array = 0
        size_of_array += sys.getsizeof(self.word)
        size_of_array += sys.getsizeof(self.size_of_array)
        size_of_array += sys.getsizeof(self.mode)
        return size_of_array

    def get_word(self):
        return self.word

    def set_word(self, word):
        self.word = word

    def get_size_of_array(self):
        return self.size_of_array

    def set_size_of_array(self, size_of_array):
        self.size_of_array = size_of_array

    def get_mode(self):
        return self.mode

    def set_mode(self, mode):
        self.mode = mode


class FontCharacter():
    def __init__(self, character, font):
        self.character = character
        self.font = font

    def __sizeof__(self):
        size_of_array = 0
        size_of_array += sys.getsizeof(self.character)
        size_of_array += sys.getsizeof(self.font)
        return size_of_array

    # set the value of character
    def set_character(self, character):
        self.character = character

    def get_character(self):
        return self.character

# checks the implementation of the FlyWeight Pattern
# Runarray stores font object for each index
class RunArray():
    def __init__(self):
        self.fonts = {}

    def __sizeof__(self):
        size_of_array = 0
        for character in self.fonts:
            size_of_array += sys.getsizeof(character)
            size_of_array += sys.getsizeof(self.fonts[character])
        return size_of_array

    #supports different functionality supports modification object
    def add(self, first_character, count, font):
        last_character = first_character + count
        self.delete_existing(first_character, last_character)
        self.add_index(first_character, last_character, font)

    def append(self, count, font):
        first_character = 0
        for font in self.fonts:
            for index in self.fonts[font]:
                if index[1] > first_character:
                    first_character = index[1]
        self.add_index(first_character, first_character + count, font)

    def add_index(self, first_character, last_character, font):
        if font not in self.fonts:
            self.fonts[font] = [[first_character, last_character]]
        else:
            self.fonts[font].append([first_character, last_character])

    def delete_existing(self, first_character, last_character):
        count = 0
        for font in self.fonts:
            for index in self.fonts[font]:
                if index[0] <= first_character or index[1] >= last_character:
                    if index[1] < last_character:
                        count = first_character - index[0]
                        if count > 0:
                            self.add_index(index[0], first_character - 1, font)
```

```python
                    elif index[0] > first_character:
                        count = index[1] - last_character
                        if count > 0:
                            self.add_index(last_character + 1, index[1], font)
                    else:
                        count = first_character - index[0]
                        if count > 0:
                            self.add_index(index[0], first_character - 1, font)
                self.fonts[font].remove(index)

    # returns font object for character at given index
    def get_font(self, index):
        for font in self.fonts:
            for char in self.fonts[font]:
                if char[0] <= index and char[1] >= index:
                    return font
        return None


# It tests all responsible function for implementing the Flyweight Pattern
class TestFlyWeight(unittest.TestCase):

    def compare_memory_size(self):
        input = "CS 635 Advanced Object-Oriented Design & Programming\n" \
                + "Fall Semester, 2018\n" + "Doc 17 Mediator, Flyweight " \
                + ", Facade, Demeter, Active Object\n" + "Nov 19, 2019\n" \
                + "Copyright ©, All rights reserved. 2019 SDSU & Roger Whitney,"\
                + " 5500 Campanile Drive, San" + "Diego, CA 92182-7700 USA. "\
                + " OpenContent (http://www.opencontent.org/opl.shtml) license"\
                +" defines the copyright on this document."

        runs = RunArray()
        runs.add(0, 120, Font("Arial", 18, "Underlined"))
        runs.add(121, 256, Font("Verdana", 12, "Italian"))
        runs.append(256, Font("Times New Roman", 12, "Bold"))

        char_factory = CharacterBuildingFactory()
        chars = []
        flyweight_size = 0
        for index in range(len(input)):
            chars.append( char_factory.get_character(input[index]))
            flyweight_size += sys.getsizeof(chars[index])

        chars = []
        non_flyweight_size = 0
        for index in range(len(input)):
            char = Character(input[index])
            font = None
            if index < 120:
                font = Font("Arial", 18, "Underlined")
            elif index < 256:
                font = Font("Verdana", 12, "Italian")
            else:
                font = Font("Times New Roman", 12, "Bold")
            chars.append( FontCharacter(input[index], font) )
            non_flyweight_size += sys.getsizeof(chars[index])

        self.assertLess(flyweight_size, non_flyweight_size)


"""
Test Program Starts
"""
def main():
    # UnitTest start
    unittest.main()


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