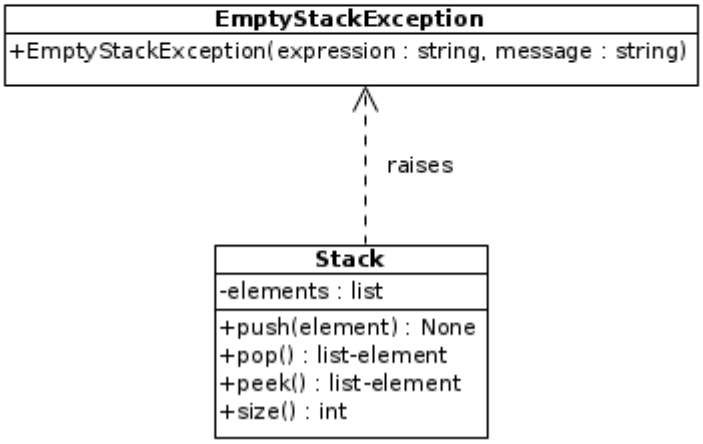
Aufgabe 1: Implementierung einer Stack Datenstruktur Klasse

Schreiben Sie ein Python Klasse, das die Datenstruktur Stack gemäß folgendem UML Klassendiagramm abbildet. Ein Stack funktioniert nach dem LIFO Prinzip und besitzt neben Konstruktor und Destruktor die folgenden Methoden:

- `push(element)`: Element auf dem Stack ablegen
- `pop()`: Letztes Element auf dem Stack zurückgeben und vom Stack entfernen
- `peek()`: Letztes Element zurückgeben, aber nicht vom Stack entfernen
- `size()`: Größe des Stacks (Anzahl Elemente) zurückgeben

Die Methoden `pop()` und `peek()` können eine eigens definierte `EmptyStackException` auslösen. Dazu muss die Exception zuerst implementiert werden. Dazu muss der Konstruktor mit den folgenden Parametern umgesetzt werden:

- `expression`: Ursache, die die Exception ausgelöst hat
- `message`: Erläuterung zu der Exception



Konzeptskizze: Machen Sie sich vor der Implementierung Gedanken zu Ihrem Konzept, wie Sie die Aufgabe lösen möchten!

Zur Unterstützung können Sie die Unit Tests auf der folgenden Seite verwenden.

```
# Tests für Stack
class StackTest(unittest.TestCase):

    def setUp(self):
        self.s1 = Stack()

    def tearDown(self):
        del self.s1

    # Test auf leeren Stack
    def test_empty(self):
        self.assertEqual(0, self.s1.size())

    # Test auf Exception
    def test_pop_empty(self):
        with self.assertRaises(EmptyStackException) as ese:
            result = self.s1.pop()
            self.assertEqual('pop() on empty Stack', ese.expression)
            self.assertEqual('Stack is empty', ese.message)

    # Test auf hinzugefügte Elemente
    def test_push_elements(self):
        self.s1.push('first element')
        self.assertEqual(1, self.s1.size())

        element = 'second element'
        self.s1.push(element)
        self.assertEqual(2, self.s1.size())

        last_element = self.s1.peek()
        self.assertEqual(element, last_element)

    # Test auf hinzugefügte und entfernte Elemente
    def test_pop_elements(self):
        numbers = 1024
        for i in range(numbers):
            self.s1.push(i)
            self.assertEqual(i//2+1, self.s1.size())
            if i % 2 == 0:
                self.s1.pop()
                self.assertEqual(i/2, self.s1.size())

        self.assertEqual(numbers//2, self.s1.size())
        for i in range(numbers//2):
            self.s1.pop()
        self.assertEqual(0, self.s1.size())
```

Aufgabe 2: Vererbung

Bearbeiten Sie das Beispiel zur Vererbung in Python unter
https://www.learnbyexample.org/python-inheritance/.

Beantworten Sie anschließend die folgenden Fragen:

1) Wie kann eine Methode in Python überschrieben werden?
2) Wofür wird das Überschreiben einer Methode benötigt?
3) Warum scheitert der Methodenaufruf setSpeed(25) auf dem Objekt von Vehicle?
4) Was bedeutet Mehrfachvererbung?
5) Was könnte ein Problem der Mehrfachvererbung sein?