```
class Random:
    def __init__(self,seed):
        self.seed = seed

    def next(self):
        self.seed = (((7**5)*self.seed)%((2**31)-1))
        return self.seed

    def choose(self, limit):
        self.randVal = self.next() % limit
        return self.randVal

class Rule:
    classCount = 1
    def __init__(self, left, right): #constructor
        self.left = left
        self.right = right
        self.count = Rule.classCount
        Rule.classCount += 1

    def __repr__(self):
        outString = ""
        outString += str(self.count) + " "
        outString += self.left + " -> "
        for i in self.right:
            outString += i + " "
        return outString

class Grammar:
    def __init__(self,seed):
        self.generated = Random(seed)
        self.rules = {}

    def rule(self, left, right):
        if left in self.rules.keys():
            self.rules[left] += (Rule(left,right),)
        else:
            self.rules[left] = (Rule(left,right),)

    def generate(self):
        string = ""
        keys = self.rules.keys()
        if 'Start' in keys:
            return self.generating(('Start',))
        else:
            raise Exception("Cannot generate strings without a rule
            for \"Start\".")
```

```python

    def generating(self, strings):
        result = ''
        for str in strings:
                if str in self.rules.keys():
                    myTuple = self.select(str)
                    result = result + self.generating(myTuple)
                else:
                    result = result + str + ' '
        return result

    def select(self, left):
        ruleTuple = self.rules[left]
        total = 0
        for rule in ruleTuple:
            total = total + rule.count
        index = self.generated.choose(total)
        i = 0
        while i < len(ruleTuple):
            rule = ruleTuple[i]
            index = index - rule.count
            if index <= 0:
                chosen = rule
                i = len(ruleTuple)
            i += 1
        for rule in ruleTuple:
            if rule != chosen:
                rule.count = rule.count + 1
        return chosen.right


G = Grammar(420) # As a consequence of the seed sometimes the
G.rule('Noun',('cat',))
G.rule('Noun', ('boy',))
G.rule('Noun', ('dog',))
G.rule('Noun', ('girl',))
G.rule('Verb', ('bit',))
G.rule('Verb', ('chased',))
G.rule('Verb', ('kissed',))
G.rule('Phrase', ('the', 'Noun', 'Verb', 'the', 'Noun'))
G.rule('Story', ('Phrase',))
G.rule('Story', ('Phrase', 'and', 'Phrase'))
G.rule('Story', ('Phrase', 'but', 'Phrase'))
G.rule('Start', ('Story', '.'))
print(G.generate())
# the dog kissed the boy .
```

```
 93
 94  G = Grammar(1234) # As a consequence of the seed sometimes the
 95  G.rule('Start', ('Story', '.'))
 96  print(G.generate())
 97  # Story .
 98
 99  G = Grammar(69420) # As a consequence of the seed sometimes the
100  G.rule('Noun',('cat',))
101  G.rule('Verb', ('bit',))
102  G.rule('Phrase', ('the', 'Noun', 'Verb', 'the', 'Noun'))
103  G.rule('Story', ('Phrase', 'but', 'Phrase'))
104  G.rule('Start', ('Story', '.'))
105  print(G.generate())
106  # the cat bit the cat but the cat bit the cat .
107
108
109  G = Grammar(2345234)
110  G.rule('Noun',('cat',))
111  G.rule('Noun', ('boy',))
112  G.rule('Noun', ('dog',))
113  G.rule('Noun', ('girl',))
114  G.rule('Verb', ('bit',))
115  G.rule('Verb', ('chased',))
116  G.rule('Verb', ('kissed',))
117  G.rule('Phrase', ('the', 'Noun', 'Verb', 'the', 'Noun'))
118  G.rule('Story', ('Phrase',))
119  G.rule('Story', ('Phrase', 'and', 'Phrase'))
120  G.rule('Story', ('Phrase', 'but', 'Phrase'))
121  print(G.generate())
122  # Traceback (most recent call last):
123  #   File "Project1.py", line 121, in <module>
124  #     print(G.generate())
125  #   File "Project1.py", line 46, in generate
126  #     raise Exception("Cannot generate strings without a rule for \"Start\".")
127  # Exception: Cannot generate strings without a rule for "Start".
128
```