Creating a New Type

We're going to explore classes by writing a register class.

In the example, we will use Canadian currency including the Loonie ($1) and the Toonie ($2).

Example

Consider this program:

```
# A cash register with 5 loonies, 5 toonies, 5 fives, 5 tens, and 5 twenties,
# for a total of $190.
register = CashRegister(5, 5, 5, 5, 5)
print(register.get_total())

register.add(3, 'toonies')
register.remove(2, 'twenties')

print(register.get_total())
```

The code above:

- creates a cash register with 5 loonies, 5 toonies, 5 fives, 5 tens, and 5 twenties,
- prints the total amount ($190) in the cash register,
- adds 3 toonies ($2 coins) to the cash register,
- removes 2 twenties ($20 bills) from the cash register, and
- prints the total amount ($156) in the cash register.

If you try to run the program, an error occurs because there is no `CashRegister` class yet:

```
Traceback (most recent call last):
  File "cash_register.py", line 4, in
    register = CashRegister(5, 5, 5, 5, 5)
  NameError: name 'CashRegister' is not defined
```

Defining Class `CashRegister`

The first line of the class definition is:

```
class CashRegister:
```

Method `__init__`

The first method that we'll write is the constructor, method `__init__`, which is called to initialize an object. Since it is a method, by convention, the first parameter is `self`. In this case, `self` refers to the `CashRegister` object that is being initialized. We'll also pass in the number of loonies, toonies, fives, tens and twenty dollar bills.

```
class CashRegister:
    """A cash register."""

    def __init__(self, loonies, toonies, fives, tens, twenties):
        """ (CashRegister, int, int, int, int, int) -> NoneType

        A CashRegister with loonies, toonies, fives, tens, and twenties.

        >>> register = CashRegister(5, 5, 5, 5, 5)
        >>> register.loonies
        5
        >>> register.toonies
        5
        >>> register.fives
        5
        >>> register.tens
        5
        >>> register.twenties
        5
        """

        self.loonies = loonies
        self.toonies = toonies
        self.fives = fives
        self.tens = tens
        self.twenties = twenties
```

Each of the assignment statements in method `__init__` creates an *instance variable* that belongs to the `CashRegister` object. For example, one of the object's instance variables is seen on the left-hand side of this assignment statement and it is named `loonies`:
`self.loonies = loonies`.

The following function call creates a new `CashRegister` object, initializes the object by calling the constructor, and then stores the object's memory address in `register`:

```
register = CashRegister(5, 5, 5, 5, 5)
```

Examining `__init__` Results

Now, when the program above is executed, a different error occurs:

```
Traceback (most recent call last):
  File "cash_register.py", line 33, in
    print(register.get_total())
  AttributeError: 'CashRegister' object has no attribute 'get_total'
```

The traceback tells us that we have successfully created a `CashRegister` object, but that it has no method `get_total`.

If we examine `register`, we see that it exists and that it contains the memory address of an object:

```
>>> register
<__main__.CashRegister object at 0x101d79cd0>
```

Further, we can examine the values of instance variables for the `CashRegister` object that `register` refers to:

```
>>> register.loonies
5
```

```
>>> register.twenties
5
```

We can create a second instance called `register2`:

```
>>> register2 = CashRegister(2, 3, 4, 6, 7)
```

Variable `register2` refers to a different `CashRegister` object than `register`, and that object has different values in its instance variables:

```
>>> register2.twenties
7
>>> register.twenties
5
```

Method `get_total`

Let's define `get_total`. Like all methods, its first parameter is `self`. Here is the implementation:

```
def get_total(self):
    """ (CashRegister) -> int

    Return the total amount of cash in the register.

    >>> register = CashRegister(5, 5, 5, 5, 5)
    >>> register.get_total()
    190
    """

    return self.loonies + self.toonies * 2 + self.fives * 5 + \
    self.tens * 10 + self.twenties * 20
```

When the program above is executed, we get the following results:

```
190

Traceback (most recent call last):
  File "cash_register.py", line 48, in
    register.add(3, 'toonies')
  AttributeError: 'CashRegister' object has no attribute 'add'
```

The line `190` results from our first call on `get_total()`. This is followed by an error that occurs because there is no `add` method in `CashRegister`.

Method `add`

Method `add` has three parameters: `self` (a `CashRegister` object), the number of items to add, and the denomination:

```
def add(self, count, denomination):
    """ (CashRegister, int, str) -> NoneType

    Add count items of denomination to the register.
    denomination is one of 'loonies', 'toonies',
    'fives', 'tens', and 'twenties'.

    >>> register = CashRegister(5, 5, 5, 5, 5)
```

```
    >>> register.add(2, 'toonies')
    >>> register.toonies
    7
    >>> register.add(1, 'tens')
    >>> register.tens
    6
    """

    if denomination == 'loonies':
        self.loonies += count
    elif denomination == 'toonies':
        self.toonies += count
    elif denomination == 'fives':
        self.fives += count
    elif denomination == 'tens':
        self.tens += count
    elif denomination == 'twenties':
        self.twenties += count
```

The `remove` Method

This method looks remarkably similar to method `add`:

```
def remove(self, count, denomination):
    """ (CashRegister, int, str) -> NoneType

    Remove count items of denomination from the register.
    denomination is one of 'loonies', 'toonies',
    'fives', 'tens', and 'twenties'.

    >>> register = CashRegister(5, 5, 5, 5, 5)
    >>> register.remove(2, 'toonies')
    >>> register.toonies
    3
    >>> register.remove(1, 'tens')
    >>> register.tens
    4
    """

    if denomination == 'loonies':
        self.loonies -= count
    elif denomination == 'toonies':
        self.toonies -= count
    elif denomination == 'fives':
        self.fives -= count
    elif denomination == 'tens':
        self.tens -= count
    elif denomination == 'twenties':
        self.twenties -= count
```

The complete class

We have finishing defining our new type, and the program runs without error:

```
190
156
```