# Lecture 8.1 : Object-oriented programming: More instance methods

## Adding yet another instance method

- Let's write an instance method that modifies the `Time` instance it is invoked on.
- Our new method increments a time by adding to it another time (it does not return anything).
- If we print the `Time` object before and after invoking the method on it we should find that it differs by the amount of time specified in the second parameter.
- Here is our first attempt at writing such a method:

```
class Time(object):

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def time_to_seconds(self):
        return self.hour*60*60 + self.minute*60 + self.second

    def is_later_than(self, other):
        return self.time_to_seconds() > other.time_to_seconds()

    def plus(self, other):
        return seconds_to_time(
            self.time_to_seconds() + other.time_to_seconds())

    def increment(self, other):
        z = self.plus(other)
        self = z

    def __str__(self):
        return 'The time is {:02d}:{:02d}:{:02d}'.format(
            self.hour, self.minute, self.second)

def seconds_to_time(s):
    minute, second = divmod(s, 60)
    hour, minute = divmod(minute, 60)
    overflow, hour = divmod(hour, 24)
    return Time(hour, minute, second)
```

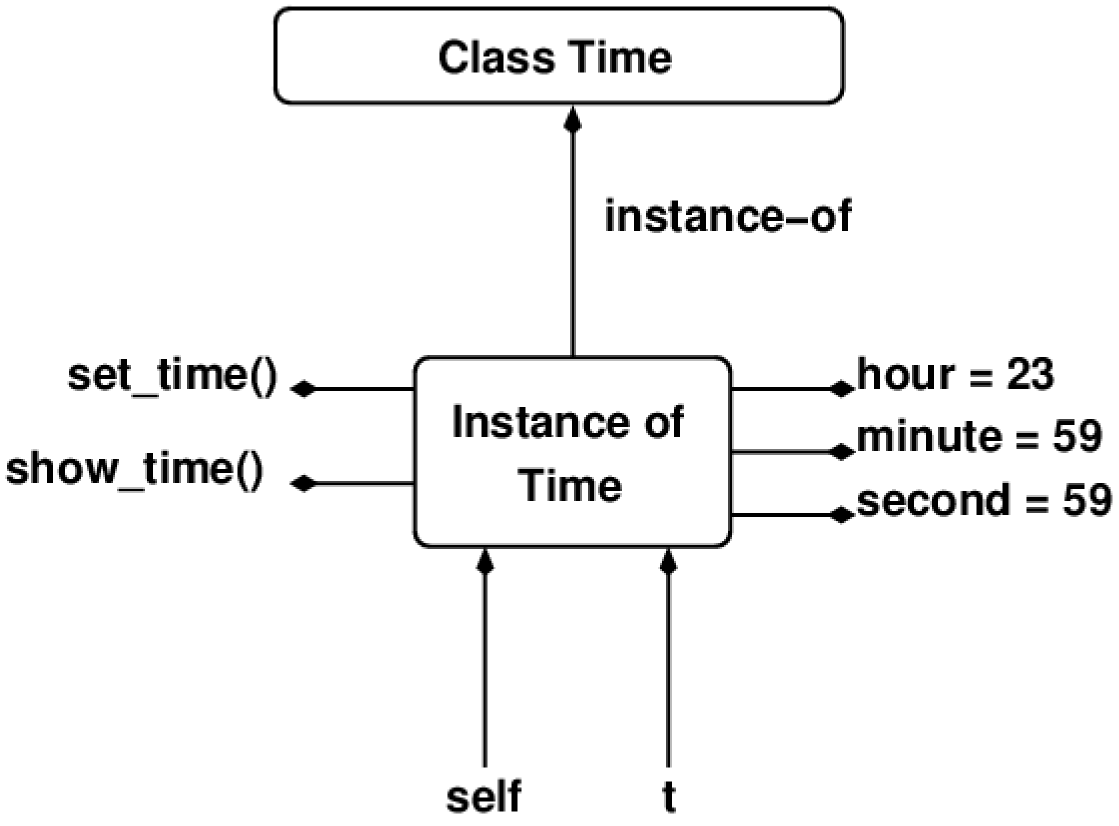
- We can see what this new method is trying to do: We pass to it a `Time` to be incremented in `self` and in `other` we pass by how much we want `self` to be incremented.
- The method adds the two times together (by calling the instance method `plus()` which handles any wraparound issues) to produce a new `Time` object `t`.
- Finally we *overwrite* `self` with a reference to this new `Time` object `t`. Will this new method work? Well let's try it and see.

```
t = Time(23, 59, 59)
i = Time(0, 0, 1)
```

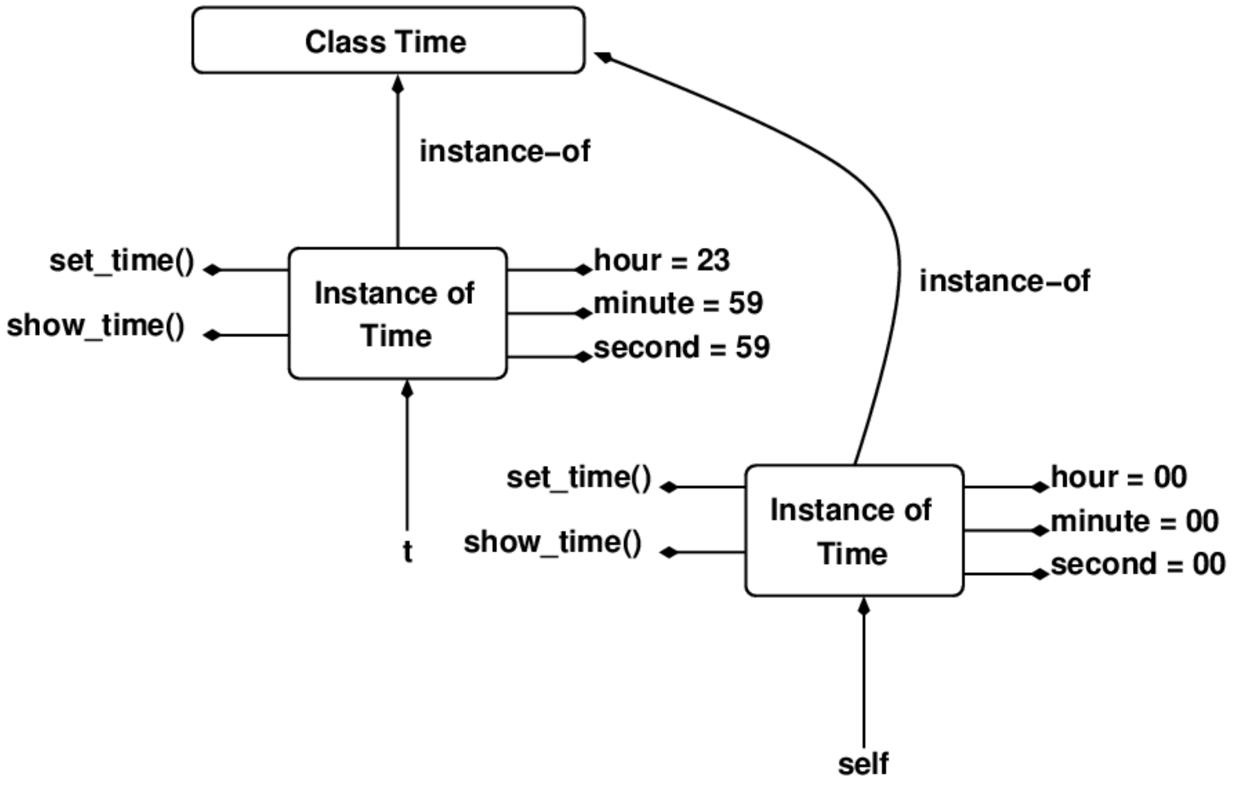
```
t.increment(i)
print(t)
```

```
The time is 23:59:59
```

- Well that's disappointing! What is going on? Why is `t` unchanged after invoking the `increment()` method? `t` should now be `00:00:00` but our method has had no effect on it.
- The following diagram represents the situation on entering the `increment()` method:



- This diagram represents the situation on leaving the `increment()` method:



- When `increment()` is invoked, `self` becomes a copy of `t`.
- Thus `self` and `t` both reference the same object.
- When the method executes `self = z`, however, `self` is overwritten to point to a new `Time` object `z`.
- Note however that `t` *still points to the original object* and this object remains unchanged. Thus when we print it we get back the original time.
- To update the `t` object via the `increment()` method we must write *through* `self` in order to update the object that both `t` and `self` point to.
- What we cannot do is *overwrite* `self` because doing so will cause a new object to be created (one that is unrelated to `t`).
- Below we write *through* `self` to update its attributes and in so doing we update the attributes of `t` (since `self` and `t` are aliases for the same object):

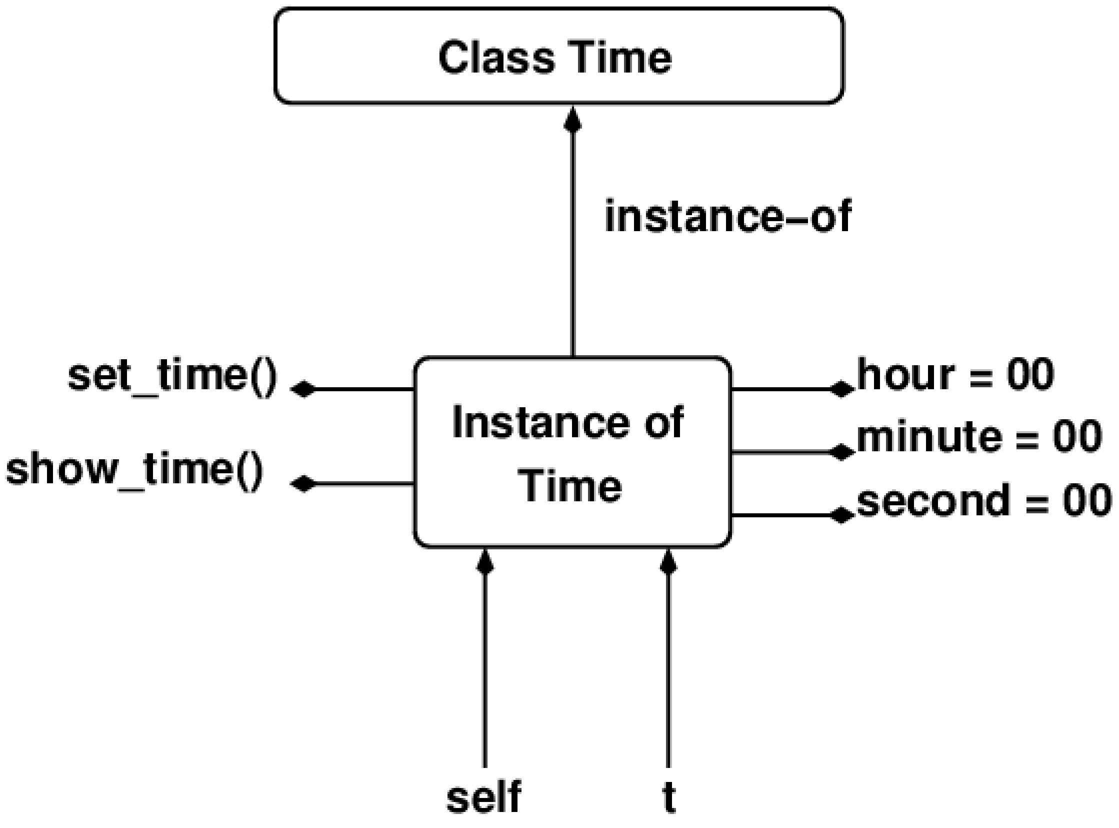
```python
class Time(object):

    def __init__(self, hour=0, minute=0, second=0):
        self.hour = hour
        self.minute = minute
        self.second = second

    def time_to_seconds(self):
        return self.hour*60*60 + self.minute*60 + self.second

    def is_later_than(self, other):
        return self.time_to_seconds() > other.time_to_seconds()

    def plus(self, other):
        return seconds_to_time(
            self.time_to_seconds() + other.time_to_seconds())

    def increment(self, other):
        z = self.plus(other)
        self.hour, self.minute, self.second = z.hour, z.minute, z.second
    def __str__(self):
        return 'The time is {:02d}:{:02d}:{:02d}'.format(
            self.hour, self.minute, self.second)

def seconds_to_time(s):
    minute, second = divmod(s, 60)
    hour, minute = divmod(minute, 60)
    overflow, hour = divmod(hour, 24)
    return Time(hour, minute, second)
```

- Let's verify this version works as intended.

```
t = Time(23, 59, 59)
i = Time(0, 0, 1)
t.increment(i)
print(t)
```

```
The time is 00:00:00
```

- That's more like it!
- We can represent this version with the following diagram:


Class Time
instance–of
set_time()
show_time()
Instance of Time
hour = 00
minute = 00
second = 00
self
t