Mini-Lecture 19

Designing Classes

Invariants

- Properties of an attribute that must be true
- Works like a precondition:
  - If invariant satisfied, object works properly
  - If not satisfied, object is "corrupted"
- **Examples**:
  - Point3 class: all attributes must be floats
  - RGB class: all attributes must be ints in 0..255
- Purpose of the **class specification**

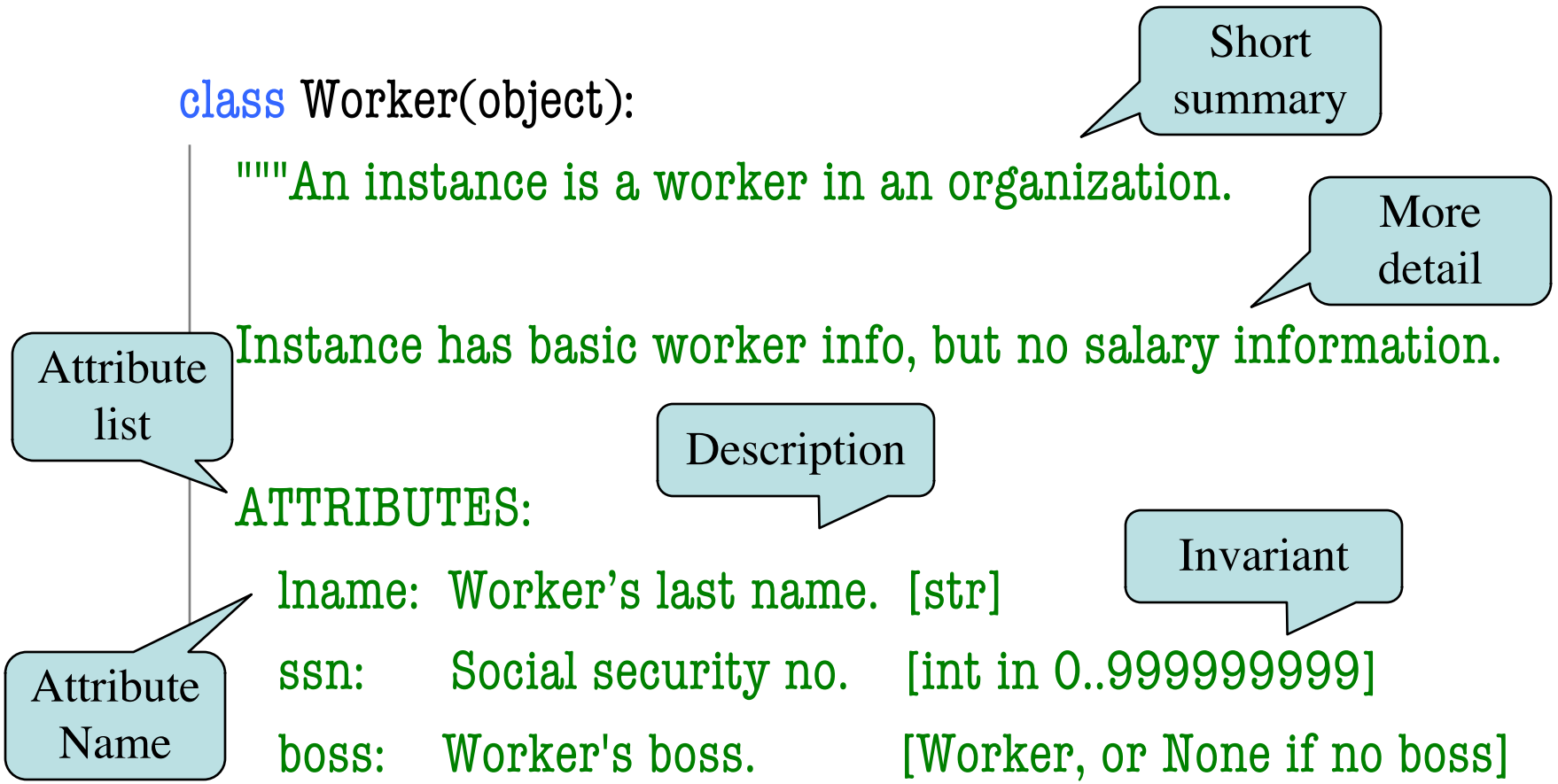
The Class Specification

```
class Worker(object):
    """An instance is a worker in an organization.

    Instance has basic worker info, but no salary information.

    ATTRIBUTES:
      lname:  Worker's last name.  [str]
      ssn:    Social security no.  [int in 0..999999999]
      boss:   Worker's boss.       [Worker, or None if no boss]
```

The Class Specification



Initializing the Attributes of an Object (Folder)

- Creating a new Worker is a multi-step process:
  - `w = Worker()` ← Instance is empty
  - `w.lname = 'White'`
  - ...
- Want to use something like

  `w = Worker('White', 1234, None)`
  - Create a new Worker **and** assign attributes
  - lname to 'White', ssn to 1234, and boss to None
- Need a **custom constructor**

Special Method: __init__

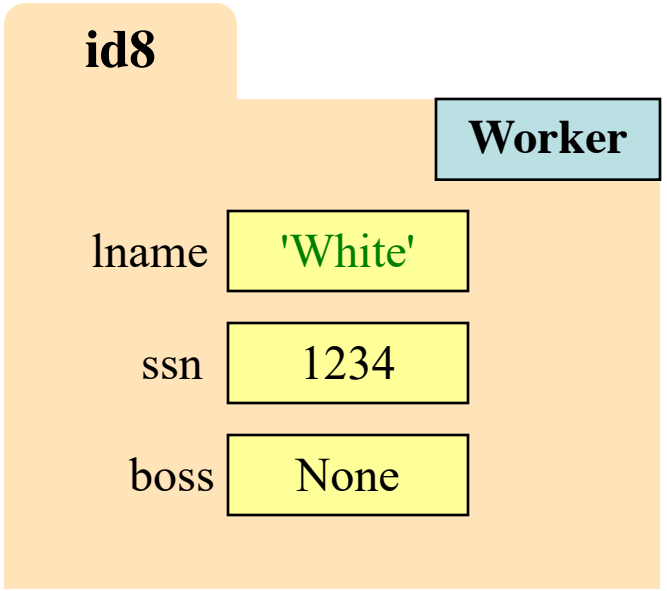
```
w = Worker('Obama', 1234, None)
```

```
def __init__(self, n, s, b):
    """Initializer: creates a Worker

    Has last name n, SSN s, and boss b

    Precondition: n a string, s an int in
    range 0..999999999, and b either
    a Worker or None.
    self.lname = n
    self.ssn  = s
    self.boss = b
```

Called by the constructor



Special Method: __init__

two underscores

```
w = Worker('Obama', 1234, None)
```

don't forget self

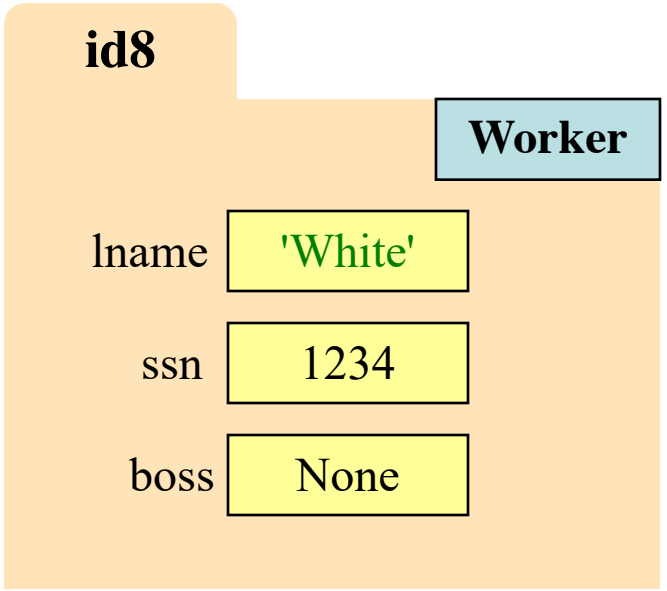
```
def __init__(self, n, s, b):
    """Initializer: creates a Worker

    Has last name n, SSN s, and boss b

    Precondition: n a string, s an int in
    range 0..999999999, and b either
    a Worker or None.
    """
    self.lname = n
    self.ssn  = s
    self.boss = b
```

use self to assign attributes

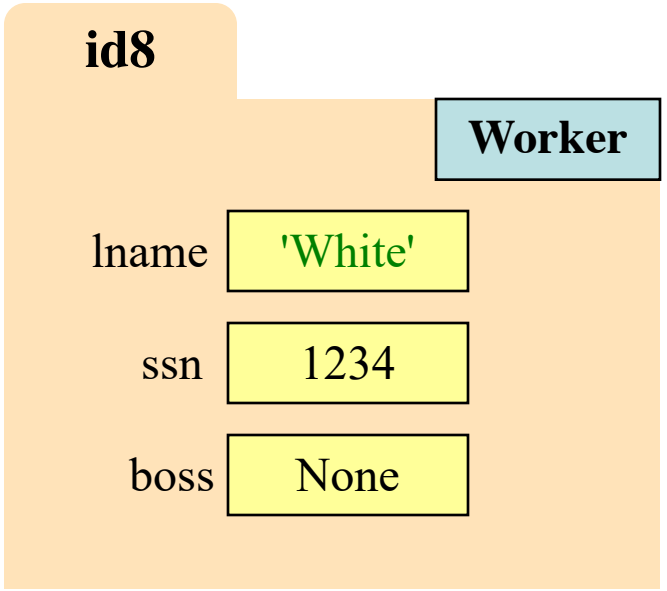
Called by the constructor



Evaluating a Constructor Expression

```
Worker('Obama', 1234, None)
```

1. Creates a new object (folder) of the class Worker
   - Instance is initially empty
2. Puts the folder into heap space
3. Executes the method __init__
   - Passes folder name to self
   - Passes other arguments in order
   - Executes the (assignment) commands in initializer body
4. Returns the object (folder) name



Designing a Class

1. Think about what values you want in the set
   - What are the attributes? What values can they have?
2. Think about what operations you want
   - This often influences the previous question

- To make (1) precise: write a *class invariant*
  - Statement we promise to keep true **after every method call**
- To make (2) precise: write *method specifications*
  - Statement of what method does/what it expects (preconditions)
- Write your code to make these statements true!

Planning out a Class

```
class Time(object):
    """Instances represent times of day.
    Instance Attributes:
        hour: hour of day [int in 0..23]
        min:  minute of hour [int in 0..59]"""

    def __init__(self, hour, min):
        """The time hour:min.
        Pre: hour in 0..23; min in 0..59"""

    def increment(self, hours, mins):
        """Move this time <hours> hours
        and <mins> minutes into the future.
        Pre: hours is int >= 0; mins in 0..59"""

    def isPM(self):
        """Returns: this time is noon or later."""
```

Class Invariant

States what attributes are present and what values they can have.

A statement that will always be true of any Time instance.

Method Specification

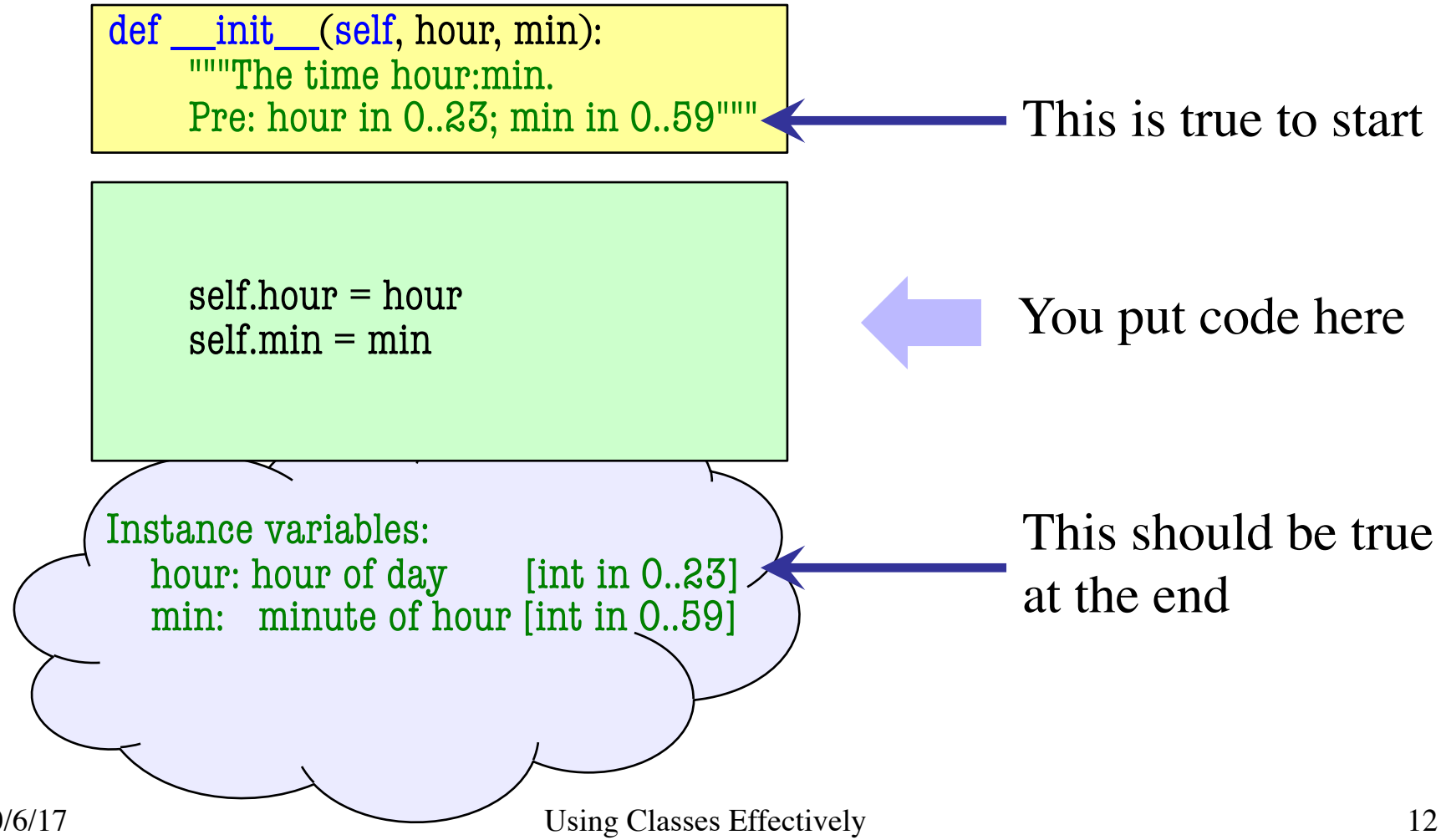
States what the method does.

Gives preconditions stating what is assumed true of the arguments.

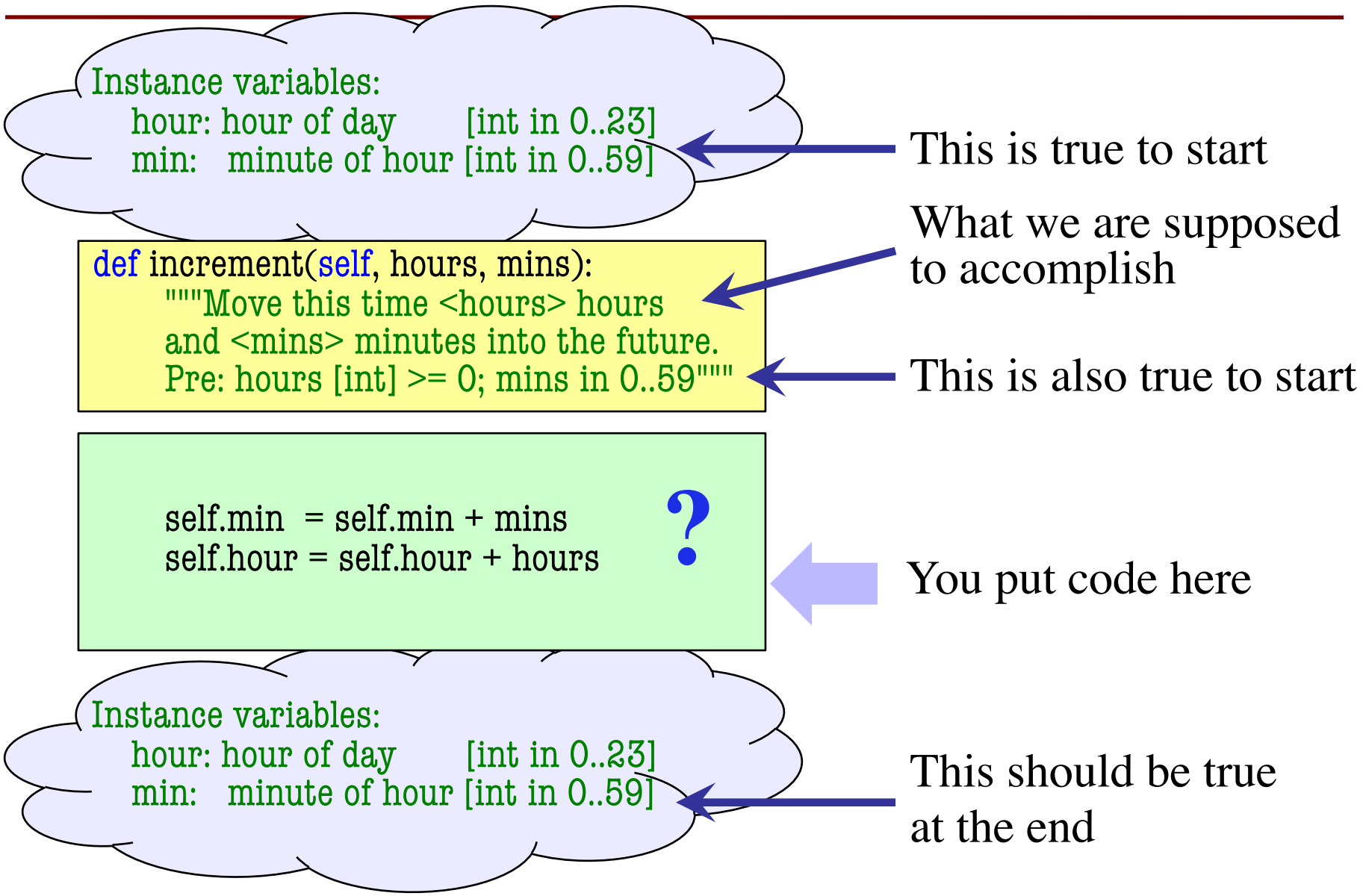
Implementing a Class

- All that remains is to fill in the methods. (All?!)
- When implementing methods:
  1. Assume preconditions are true
  2. Assume class invariant is true to start
  3. Ensure method specification is fulfilled
  4. Ensure class invariant is true when done
- Later, when using the class:
  - When calling methods, ensure preconditions are true
  - If attributes are altered, ensure class invariant is true

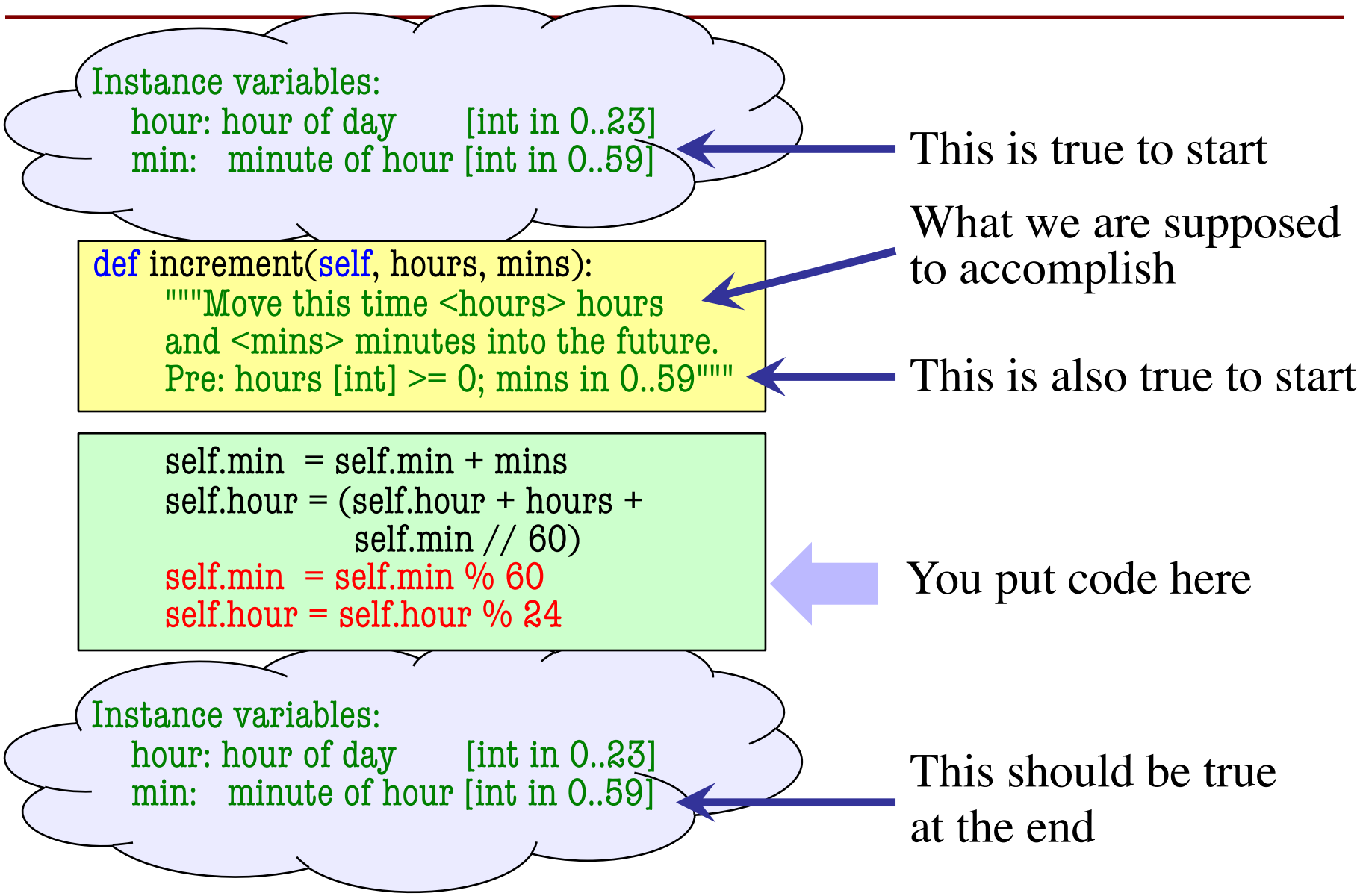
Implementing an Initializer



Implementing a Method



Implementing a Method



Role of Invariants and Preconditions

- They both serve two purposes
  - Help you think through your plans in a disciplined way
  - Communicate to the user* how they are allowed to use the class
- Provide the *interface* of the class
  - interface btw two programmers
  - interface btw parts of an app
- Important concept for making large software systems
  - Will return to this idea later

in•ter•face ˈintərˌfās noun

1. a point where two systems, subjects, organizations, etc., meet and interact : the interface between accountancy & the law.
   - *chiefly Physics* a surface forming a common boundary between two portions of matter or space, e.g., between two immiscible liquids : the surface tension of a liquid at its air/liquid interface.
2. *Computing* a device or program enab-ling a user to communicate with a computer.
   - a device or program for connecting two items of hardware or software so that they can be operated jointly or communicate with each other.

—The Oxford American Dictionary

*** …who might well be you!**