How to use an observer template

CS 5010 Program Design Paradigms
"Bootcamp"

Lesson 1.4

Learning Objectives

- By the end of this lesson, you should be able to use an observer template to write a function definition that examines a piece of data.

What does it mean to “examine” a piece of data?

- If the data is compound data, this means extracting its fields.
- If the data is itemization data, this means determining which variant the data is.
- If the data is mixed data, this means determining which variant the data is, and then extracting its fields, if any.
- Every data definition includes a template that shows how this examination process is to be organized.
- Writing a function using structural decomposition is accomplished by filling in the blanks in the template.
  - Definition of "filling in the blank" to come in Slide 11.

From Template to Function Definition

Recipe for Using a Template
1. Make a copy of the template and uncomment it
2. Fill in the function name and add more arguments if needed
3. For design strategy, write: “Use template for <data def> on <vble>,” where <data def> is the kind of data you are taking apart, and <vble> is the variable whose value you are looking at.
4. Use the template as a pattern for your function, using the items in the inventory.

Example: **book-receipts**

```
;; book-receipts : Book NonNegInt -> NonNegInt
;; GIVEN: a Book and the number of copies sold
;; RETURNS: the total receipts from the sales of the
;; given book. Ignores the number of copies on hand.
;; EXAMPLE:
;; (book-receipts
;;   (make-book "Felleisen" "HtdP2" 13 2795) 100)
;; = 279500
```

To do this, we'll need to look inside the Book to see its price, so we'll use the Book template

1. Make a copy of the template and uncomment it

```
(define (book-fn b)
  (...
    (book-author b)
    (book-title b)
    (book-on-hand b)
    (book-price b)))
```

2. Fill in the function name and add more arguments if needed

```
(define (book-receipts b sales)
  (...
    (book-author b)
    (book-title b)
    (book-on-hand b)
    (book-price b)))
```

3. Write down the strategy

```
;; STRATEGY: Use template for Book on b.
(define (book-receipts b sales)
  (...
    (book-author b)
    (book-title b)
    (book-on-hand b)
    (book-price b)))
```

This is what you'll write for the Design Strategy

The inventory of quantities we can use to construct the answer.

4. Fill in the blanks in the template

```
;; STRATEGY: Use template for Book on b.
(define (book-receipts b sales)
  (* (book-price b) sales))
```

Things we didn't use:

```
    (book-author b)
    (book-title b)
    (book-on-hand b)
```

That's OK!

We said:
The observer template (or just the template, for short) gives a skeleton for functions that examine or use the data. We can fill in the ... with any expression, using any or all of the expressions in the inventory.

The template is a starting point for your function definition; you don't have to follow it rigidly. But the closer you stay to the template, the less trouble you can get into.

Example: next state of traffic light

```
;; DATA DEFINITION:
;; a TrafficLightState (TLState) is one of
;; -- "red"
;; -- "yellow"
;; -- "green"
;; INTERPRETATION: self-evident
```

Contract, Purpose Statement, and Examples

```
;; next-state : TLState -> TLState
;; GIVEN: a TLState
;; RETURNS: the TLState that follows the given TLState
;; EXAMPLES:
;; (next-state "red") = "green"
;; (next-state "yellow") = "red"
;; (next-state "green") = "yellow"
```

1. Make a copy of the template and uncomment it

```
(define (tls-fn state)
 (cond
   [(string=? state "red")    ...]
   [(string=? state "yellow") ...]
   [(string=? state "green")  ...]))
```

2. Fill in the function name and add more arguments if needed

```
(define (next-state state)
 (cond
   [(string=? state "red")    ...]
   [(string=? state "yellow") ...]
   [(string=? state "green")  ...]))
```

3. Fill in the strategy

```
;; STRATEGY: Use template for TLState on state

(define (next-state state)
 (cond
   [(string=? state "red")    ...]
   [(string=? state "yellow") ...]
   [(string=? state "green")  ...]))
```

4. Fill in the blanks

```
;; STRATEGY: Use template for TLState on state

(define (next-state state)
 (cond
   [(string=? state "red")    ...]
   [(string=? state "yellow") ...]
   [(string=? state "green")  ...]))
```

What is the answer for “red”?

4. Fill in the blanks

```
;; STRATEGY: Use template for TLState on state

(define (next-state state)
 (cond
   [(string=? state "red")    "green"]
   [(string=? state "yellow") ...]
   [(string=? state "green")  ...]))
```

What is the answer for “red”?

Answer (from examples): “green”

4. Fill in the blanks

```
;; STRATEGY: Use template for TLState on state

(define (next-state state)
 (cond
   [(string=? state "red")    "green"]
   [(string=? state "yellow") "red"]
   [(string=? state "green")  ...]))
```

What is the answer for "yellow"?

Answer (from examples): "red"

4. Fill in the blanks

```
;; STRATEGY: Use template for TLState on state

(define (next-state state)
 (cond
   [(string=? state "red")    "green"]
   [(string=? state "yellow") "red"]
   [(string=? state "green")  "yellow"]))
```

What is the answer for "green"?

Answer (from examples): "yellow"

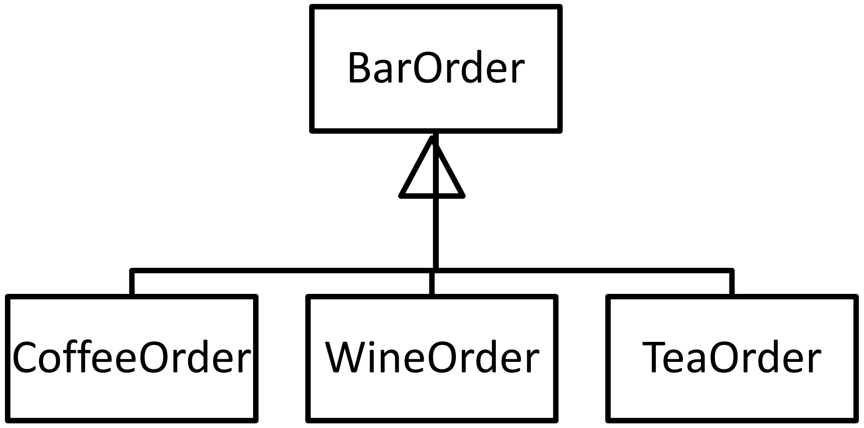
Working with other kinds of data

- We've seen how to use templates for compound data and itemization data
- Mixed data works the same way.
- Copy the template, uncomment it, and fill in the missing pieces. That's it!
- If you've thought hard enough about your function, filling in the blanks is easy.

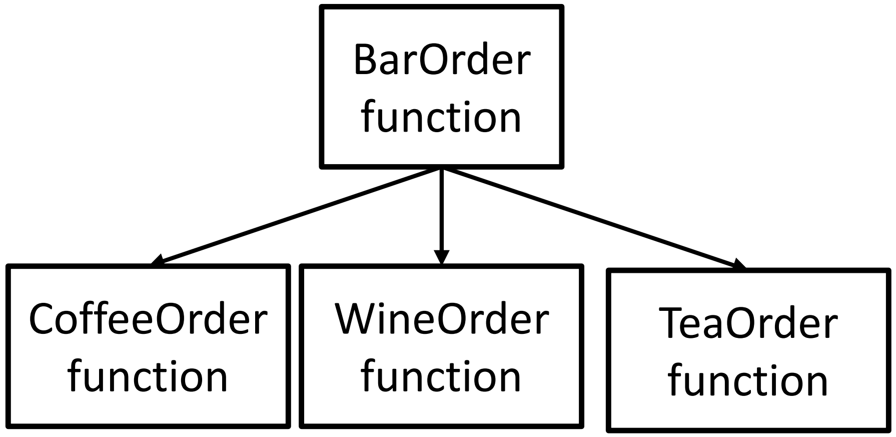
What can you put in the blanks?

- We said: Fill in the blanks in the template by combining the arguments and the values of the fields using simpler functions.
- This means :
  - You don't have to use all of the fields
  - You can use a field twice
  - You don't have to use the fields "in order"
- But it has to be simple.

Remember: The Shape of the Program Follows the Shape of the Data



Data Hierarchy (the open triangle means “OR”)



Call Tree (the arrow goes from caller to callee)

Remember: the shape of the program follows the shape of the data

Next Steps

- Study 01-4-1-book-receipts.rkt and 01-4-2-traffic-light.rkt in the Examples folder.
  - Be sure to finish the **previous-state** example in 01-4-2-traffic-light.rkt
- If you have questions or comments about this lesson, post them on the discussion board.
- Do the Guided Practices
- Go on to the next lesson.