Destructor Templates

CS 5010 Program Design Paradigms
“Bootcamp”

Lesson 1.4

Learning Objectives for This Lesson

- By the time you finish this lesson, you should be able to:
  - explain what a destructor template is
  - write destructor templates for typical data.

DDR Step 5: Destructor Template

- The destructor template (or just the template, for short) gives a skeleton for functions that examine or use the data.
- Once you write the template, writing the function is just a matter of filling in the blanks.
- This step is a little more complicated than the preceding ones, so we have a recipe for this, too!

The template recipe

Question	Answer
1. Does the data definition distinguish among different subclasses of data?	Write a cond with a clause for each subclasses.
2. How do the subclasses differ from each other?	Use the differences to formulate a condition per clause.
3. Do any of the clauses deal with structured values?	If so, add appropriate selector expressions to the clause.
4. Do any of the fields contain compound or mixed data?	If the value of a field is a *foo*, add a call to a *foo-fn* to use it.

Let's see where we are

The Function Design Recipe
1. Data Design
2. Contract and Purpose Statement
3. Examples and Tests
4. Design Strategy
5. Function Definition
6. Program Review

The Data Design Recipe
1. What information needs to be represented in your program? What kind of information is each piece?
2. Struct Definitions
3. Constructor Template
4. Interpretation
5. Destructor Template
6. Examples
7. Review

Question	Answer
1. Does the data definition distinguish among different subclasses of data?	Write a cond with a clause for each subclasses.
2. How do the subclasses differ from each other?	Use the differences to formulate a condition per clause.
3. Do any of the clauses deal with structured values?	If so, add appropriate selector expressions to the clause.
4. Do any of the fields contain compound or mixed data?	If the value of a field is a foo, add a call to a foo-fn to use it.

In this lesson

Question	Answer
1. Does the data definition distinguish among different subclasses of data?	Write a cond with a clause for each subclasses.
2. How do the subclasses differ from each other?	Use the differences to formulate a condition per clause.
3. Do any of the clauses deal with structured values?	If so, add appropriate selector expressions to the clause.
4. Do any of the fields contain compound or mixed data?	If the value of a field is a foo, add a call to a foo-fn to use it.

Lesson Outline

- In this lesson, we'll learn how to apply the template recipe to itemization, compound, and mixed data.
- We’ll start with mixed data, and then see how to work out the special cases of compound and itemization data.
- Let’s start with the BarOrder example. We’ll follow the template recipe.

Data Definition for mixed data: example

```
(define-struct coffee (size type milk?))
(define-struct wine (vineyard year))
(define-struct tea (size type))

;; A BarOrder is one of
;; -- (make-coffee Size Type Boolean)
;;  INTERP:
;;   size is the size of cup desired
;;   type is the origin of the coffee
;;   milk? tells whether milk is desired.
;; -- (make-wine Vineyard Year)
;;  INTERP:
;;   vineyard is the origin of the grapes
;;   year is the year of harvest
;; -- (make-tea Size String)
;;  INTERP:
;;   size is the size of cup desired
;;   type is the type of tea (as a string)
```

The structure definitions

Presumably Size and Type are data types defined elsewhere.

Here it's clear what the alternatives mean, so all we need to provide is the interpretation of each field in each alternative.

Presumably Vineyard is also a data type defined elsewhere.

Writing the template for BarOrder

```
;; bo-fn : BarOrder -> ??
(define (bo-fn order) ...)
```

Start by writing a template for the contract and the beginning of a function definition

Writing the template for BarOrder

```
;; bo-fn : BarOrder -> ??
(define (bo-fn order)
  (cond
    [... ...]



    [... ...]


    [... ...]))
```

1. Write a **cond** with as many alternatives as the data definition has.

Writing the template for BarOrder

```
;; bo-fn : BarOrder -> ??
(define (bo-fn order)
  (cond
    [(coffee? order) ...]


    [(wine? order) ...]


    [(tea? order) ...]))
```

2. Add predicates that distinguish the different cases

Writing the template for BarOrder

```
;; bo-fn : BarOrder -> ??
(define (bo-fn order)
  (cond
    [(coffee? order) (...
                          (coffee-size order)
                          (coffee-type order)
                          (coffee-milk? order))]
    [(wine? order) (...
                        (wine-vineyard order)
                        (wine-year order))]
    [(tea? order) (...
                       (tea-size order)
                       (tea-type order))]))
```

3. Add selectors to extract the values of the fields.

source file: 01-2-template-examples.rkt

What is the destructor template good for?

- The destructor template (or just the template, for short) gives a skeleton for functions that examine or use the data.
- The values after the ... give us an inventory of the values we can use on the right-hand side of the **cond**.

How to write a template for compound data

- Just like the one for mixed data, but you don’t need a **cond**.
- Here’s an example:

Template for compound data

```
(define-struct book (author title on-hand price))

;; A Book is a
;;  (make-book String String NonNegInt NonNegInt)
;; Interpretation:
;;   author is the author's name
;;   title is the title
;;   on-hand is the number of copies on hand
;;   price is the price in USD*100

;; book-fn : Book -> ??
(define (book-fn b)
  (...
    (book-author b)
    (book-title b)
    (book-on-hand b)
    (book-price b)))
```

The structure definition

The constructor template

The interpretation of each field

1. No subclasses, so no cond.
2. The selector functions give you the pieces of data that you can calculate with.

source file: 01-2-template-examples.rkt

Template for Itemization Data

- No selectors, just a **cond**
- Here’s a simple example:

```
;; A Size is one of
;; -- "small"
;; -- "medium"
;; -- "large"

;; size-fn : Size -> ??
(define (size-fn s)
  (cond
    [(string=? s "small") ...]
    [(string=? s "medium") ...]
    [(string=? s "large") ...]))
```

Summary

- You should now be able to write destructor templates for itemization, compound, and mixed data.

Next Steps

- Study the files 01-2-template-examples.rkt in the examples folder.
- Do Guided Practice 1.2.
- If you have questions about this lesson, ask them on the Discussion Board
- Do the Guided Practices
- Go on to the next lesson