Lecture 2: Functional vs. Object-Oriented Programming

May 10th, 2022

What We've Covered So Far:

In Fundies 1 - Functional design

In Fundies 2 - Started Object-Oriented

The Problem

Given a book or article, I want the citation for it in one of two styles:

- APA
- MLA

The Functional Approach

- Start with data definitions

```
 ;; A Publication is one of:
;;  -- (make-book [String String String String Number])
;;  -- (make-article String String String Number Number Number)

(define-struct book [title author publisher location year])
(define-struct article [title author journal volume issue year])

;; Examples:
(define gray.v1
 (make-book "Beasts of Prey" "Ayana Gray" "G.P. Putnam" "New York" 2021))

(define lubin.v1
  (make-article "How statically-type functional programmers write code" "Justin Lubin" "PACMPL" 5 3 2021))
```

- Define a template

```
 ;; process-publication: Publication -> ???
;; Template for processing Publications
(define (process-publication: pub)
  (cond
    [(book? pub)
     (... (book-title pub) ... (book-author pub) ... (book-publisher pub) ...
          (book-location pub) ... (book-year pub) ...)]
    [(article? pub)
     (... (article-title pub) ... (article-author pub) ...
          (article-journal pub) ... (article-volume pub) ...
          (article-issue pub) ... (article-year pub) ...)]))
```

- How do we cite?
- Write a function to format a string
  - The function acts on the data, the data does NOT own the function

```
(define (cite-mla pub)
   (cond
     [(book? pub)
       (format "~a. ~a ~a: ~a, ~a."
         (book-author pub)
         (book-title pub)
         (book-location pub)
         (book-publisher pub)
         (book-year pub))]
     [(article? pub)
       (format "~a. \"~a.\" ~a ~a.~a (~a)"
          (article-author pub)
          (article-title pub)
          (article-journal pub)
          (article-volume pub)
          (article-issue pub)
          (article-year pub))])))
```

- Now for APA, copy-paste and change some things

```
(define (cite-apa pub)
   (cond
     [(book? pub)
       (format "~a ~(a), ~a. ~a: ~a."
         (book-author pub)
         (book-year pub)
         (book-title pub)
         (book-location pub)
         (book-publisher pub))]
     [(article? pub)
       (format "~a (~a). ~a ~a, ~a (~a)"
          (article-author pub)
          (article-year pub)
          (article-title pub)
          (article-journal pub)
          (article-volume pub)
          (article-issue pub))])))
```

- What happens when we add new data?
    - Add a new type: website
    - We have to add a new `cond` case for websites to every function we have
        - Extending our code is cumbersome
- What happens when we add new functionality?
    - We can easily add Chicago, just write a new function

> ### The Functional Design Style
>
> Pro: Adding new behavior is easy
> Con: Adding new data is pervasive

The Object-Oriented Approach

- Consider defining a Publication as a function

```
;; A Publication is a [CitationStyle -> String]
;; where a CitationStyle is one of:
;;  - "apa"
;;  - "mla"

;; This is the template for creating a publication
(define (create-publication args ...)
    (lambda (style)
        (cond
          [(string=? style "apa" (... args ...))]
          [(string=? style "mla") (... args ...)])))
```

- Define a type of data that can cite itself like a book
    - Note the language "can cite itself"

```
;; new-book: String String String String Number -> Publication
;; To construct a new book.
(define (new-book title author publisher location year)
  (lambda (style)
    (cond
      [(string=? style "apa")
       (format "~a (~a). ~a. ~a: ~a."
               author year title location publisher)]
      [(string=? style "mla")
       (format "~a. ~a. ~a: ~a, ~a."
               author title location publisher year)])))
```

- Examples:

```
(define gray.v2 (create-books "Beasts of Prey" "Ayana Grey" "G. P. Putnam" "NY" 2021))

;; Cite gray.v2
(gray.v2 "mla")
;; or
(gray.v2 "apa")
```

- Focusing on what we can do as opposed to what we can act on
- In this style, we can add new data really easily
    - Want to create webpage? Just write a new function
- However, adding a new citation style is cumbersome

> ### The Object-Oriented Design Style
>
> What can they do → then consider the data
> Pro: Adding new data types is easy
> Con: Adding new operations is pervasive

What Does This Look Like in Java?

- Java explicitly defines classes and interfaces, etc.
- Let's take the same classes we wrote, but in Java

1. Define the `Publication` interface - this defines our operations for all publications
    - Answers the question "what can a publication do?"
        - A Publication can cite itself in APA or MLA style
    - Note that `Publication` is public - we want it to be used outside the interface

```
/**
 * Specifies operations for formatting citations for different publications
 */

/*
 * public is an ACCESS MODIFIER
 * An access modifier tells someone where they can use something
 *
 * public tells us that it can be used anywhere
 */

public interface Publication {
  /**
   * Formats a citation in APA style
   * @return the formatted citation
   */
  String citeApa();

  /**
   * Formats a citation in MLA style
   * @return the formatted citation
   */
  String citeMla();
}
```

2. Create the `Book` class - defines the `Publication` operations for a book

```
/**
 * The {@code Book} class represents the bibliographic info for books.
 */
public class Book implements Publication {
  // private - can only be accessed from this class
  //  - INFORMATION HIDING

  // final - cannot be changed once initialized

  private final String title, author, publisher, location;
  private final int year;

  // Constructor for a Book

  /**
   * Constructs a {@code Book} object.
   *
   * @param title the title of this book
   * @param author the author of this book
   * @param publisher the publisher of this book
   * @param location the location of the publisher of this book
   * @param year the year this book was published
   */

  // the constructor is public because we want clients to be able to create books
  public Book(String title, String author, String publisher, String location, int year) {
    this.title = title;
    this.author = author;
    this.publisher = publisher;
    this.location = location;
    this.year = year;
  }
}
```

3. Define operations for a `Book`
    - A `Book` can cite itself in MLA and APA

```
// in the Book class

public class Book implements Publication {
  // ... above code ...

  // implement interface methods
  //  - citeApa();
  //  - citeMla();

  @Override // annotation
  // don't need JavaDoc since it does the same thing described in the interface
  public String citeApa() {
    return this.author + " (" + this.year + "), " + this.title + ". " + this.location + ": " + this.publisher + ".";
  }

  @Override // annotation
  // don't need JavaDoc since it does the same thing described in the interface
  public String citeMla() {
    return this.author + ". " + this.title + ". " + this.location + ": " + this.publisher + ", " + this.year + ".";
  }
}
```

4. Write tests for the `Book` class
    - Create a new file called BookTest using IntelliJ features
    - In OOD, tests are separate from our code

```

// import JUnit for testing
import org.junit.Before;
import org.junit.Test;

// import JUnit's assert functions
import static org.junit.Assert.*;

public class BookTest {
  // create a new example book
  Publication rushdie;

  // @Before notes that this method should be run before every test
  @Before
  public void init() {
    this.rushdie = new Book("Midnight's Children", "Salman Rushdie", "Jonathan Cape", "London", 1980);
  }

  @Test
  public void testCiteApa() {
    // assertEquals goes in the order: expected, actual
    assertEquals("Salman Rushdie (1980). Midnight's Children. "
                    + "London: Jonathan Cape.",
            rushdie.citeApa());
  }

  @Test
  public void testCiteMla() {
    assertEquals("Salman Rushdie. Midnight's Children. London: "
                    + "Jonathan Cape, 1980.",
            rushdie.citeMla());
  }
}
```

JavaDoc

- Allows us to generate documentation
- In IntelliJ - go to Tools → Generate JavaDoc