UMass Boston Computer Science

CS450 High Level Languages (section 2)

How To Design ... OO Programs?

Monday, December 11, 2023
Logistics

• HW 9 in
  • due: Sun 12/10 11:59 pm EST

• HW 10 out (Shapes!)
  • due: Sun 12/17 11:59 pm EST
A Simple OO Example: Shapes

interface Shape
Image render();

class Circle
Num radius;
Color col;

Image render() {
    return circ-img (radius, col);
}

class Rectangle
Num width; Num height;
Color col;

Image render() {
    return rect-img (width, height, col);
}
A Simple OO Example: Terminology

**Interface / abstract class**

```java
interface Shape {
    Image render();
}
```

**(abstract) method** (concrete class implements)

```java
public class Circle {
    Num radius;
    Color col;

    Image render() {
        return circ-img (radius, col);
    }
}
```

**(abstract) method** (concrete class implements)

```java
public class Rectangle {
    Num width;
    Num height;
    Color col;

    Image render() {
        return rect-img (width, height, col);
    }
}
```

**(concrete) class**

**class Circle**

**(compound) Data definition!** fields

**class Rectangle**

**(compound) Data definition!** fields

**(concrete) class**

**(concrete) method**

implementation

**(concrete) method**

implementation
CS450 vs OO Comparison

CS 450 Design Recipe
• Compound data (struct) have fields, separate fns process data

OO Programming
• Compound data (class) group fields and methods together!
A Simple OO Example: Compare to CS450

(interface) Data definition

interface Shape

  Image render();

  function signature

(class) Data definition item

class Circle

  Num radius;
  Color col;

  (compound) Data definition

  Image render() {
    return circ-img (radius, col);
  }

  function implementation
  (one cond clause) for
  Shape data

(class) Data definition item

class Rectangle

  Num width;
  Num height;
  Color col;

  (compound) Data definition

  Image render() {
    return rect-img (width, height, col);
  }

  function implementation
  (one cond clause) for
  Shape data
CS450 vs OO Comparison

**CS 450 Design Recipe**
- Compound data (struct) have fields, separate fns process data
- **Itemization** DataDefs explicitly defined

**OO Programming**
- Compound data (class) group fields and methods together!
- **Itemization** DataDefs implied by interface / class definitions
CS450 vs OO Comparison

CS 450 Design Recipe

- **Compound data** *(struct)* have fields, *separate* fns process data
- **Itemization** Data Defs *explicitly* defined
- **Functions** organized by the kind of data they process!

OO Programming

- **Compound data** *(class)* group fields and methods *together*!
- **Itemization** Data Defs *implied* by interface / class definitions
- **Methods** organized by the kind of data they process!
A Simple OO Example: Compare to CS450

```plaintext
interface Shape
Image render();

class Circle
Num radius;
Color col;

Image render() {
  return circ-img(radius, col);
}

class Rectangle
(struct rect [w h col])
Num width;
Num height;
Color col;

Image render() {
  return rect-img (width, height, col);
}

;; A Shape is one of:
;; - (rect Num Num Color)
;; interp: fields are width, height, color
;; - (circ Num Color)
;; interp: fields are radius and color
;; Represents a shape to be drawn on a canvas

;; A Shape is one of:
;; - (rect Num Num Color)
;; interp: fields are width, height, color
;; - (circ Num Color)
;; interp: fields are radius and color
;; Represents a shape to be drawn on a canvas
```

```
(class Circle)

Num radius;
Color col;

Image render() {
  return circ-img(radius, col);
}

```

```
(class Rectangle)
(struct rect [w h col])
Num width;
Num height;
Color col;

Image render() {
  return rect-img (width, height, col);
}

```

```
;; render: Shape -> Image
(define (render sh)
  (cond
    [(rect? sh) (render-rect sh)]
    [(circ? sh) (render-circ sh)])))
```

```
"abstract" implementation
```

```
"concrete" implementations
```

```
method "dispatch"
```

```
"abstract" implementation
```

```
"concrete" implementations
```
## CS450 vs OO Comparison

### CS 450 Design Recipe
- **Compound data** *(struct)* have fields, separate fns process data
- **Itemization** Data Defs *explicitly* defined
- **Functions** organized by the kind of data they process!
- **Explicit** itemization **dispatch** *(cond)*

### OO Programming
- **Compound data** *(class)* group fields and methods **together**!
- **Itemization** Data Defs *implied* by interface / class definitions
- **Methods** organized by the kind of data they process!
- **Implicit** itemization **dispatch**
A Simple OO Example: Constructors

```java
interface Shape
    Image render();

Circle c = Circle(10, blue);
Image img = c.render();

class Circle
    Num radius;    Color col;
    // …
    Circle(r, c) {
        radius = r;
        col = c;
    }

class Rectangle
    Num width;    Num height;    Color col;
    // …
    Rectangle(w, h, c) {
        width = w;
        height = h;
        col = c
    }

Q: Where are method implementations for an obj instance “stored”?

A: It’s another (hidden) field (see “method table”!)
```
CS450 vs OO Comparison

CS 450 Design Recipe

- **Compound data** (struct) have fields, separate fns process data
- **Itemization** Data Defs explicitly defined
- **Functions** organized by the kind of data they process!
- **Explicit itemization dispatch** (cond)
- **Struct Constructor** explicitly includes method defs ???

OO Programming

- **Compound data** (class) group fields and methods together!
- **Itemization** Data Defs implied by interface / class definitions
- **Methods** organized by the kind of data they process!
- **Implicit itemization dispatch**
- **Object Constructor** implicitly includes method defs
OO-style Constructors ... with structs!

Shape "dispatch" function

```
;; render : Shape -> Image
(define (render sh)
  (cond
   [(rect? sh) (render-rect sh)]
   [(circ? sh) (render-circ sh)]))
```

Q: Where are method implementations for an obj instance “stored”?  
A: It’s another (hidden) field!

Shape "interface" definition

```
(struct Shape [render-method])
(struct circ Shape [r col])
(struct rect Shape [w h col])
```

Shape constructors

```
(define (mk-circ r col
           [circ-render-fn render-circ])
  (circ circ-render-fn r col)

(define (mk-rect w h col
                [rect-render-fn render-rect])
  (rect rect-render-fn w h col)
```
CS450 vs OO Comparison

CS 450 Design Recipe

• **Compound data** *(struct)* has *(possibly function)* fields!

• **Itemization** Data Defs explicitly defined

• **Functions** organized by the kind of data they process!

• **Explicit itemization dispatch** *(cond)*

• **Struct Constructor** explicitly includes method defs

OO Programming

• **Compound data** *(class)* group fields and methods together!

• **Itemization** Data Defs implied by interface / class definitions

• **Methods** organized by the kind of data they process!

• **Implicit itemization dispatch**

• **Object Constructor** implicitly includes method defs
OO-Style Dispatch ... with structs!

450-style “dispatch” function

```lisp
;; render : Shape -> Image
(define (render sh)
  (cond
    [(rect? sh) (render-rect sh)]
    [(circ? sh) (render-circ sh)])))
```

Shape OO-style “interface and “class” definitions

```lisp
(struct Shape [render-method])
(struct circ Shape [r col])
(struct rect Shape [w h col])
```

Q: But object itself must be argument to methods?

A: In OO langs, it’s a hidden argument (see “this”!)

```lisp
Circle c = Circle( 10, blue );
Image img = c.render(); // equiv to render(c)!
```

OO-Style “dispatch”

```lisp
;; render : Shape -> Image
(define (render sh)
  ((shape-render-method sh) sh))
```

```lisp
(define c (mk-circ 10 “blue”))
(define img (render c))
```
OO-Style Dispatch ... with structs!

450-style “dispatch” function

```scheme
;; render : Shape -> Image
(define (render sh)
  (cond
    [(rect? sh) (render-rect sh)]
    [(circ? sh) (render-circ sh)]))
```

OO-Style “dispatch”

```scheme
;; render : Shape -> Image
(define (render sh)
  ((shape-render-method sh) sh))
```

```scheme
;; render-circ : Circle -> Image
(define (render-circ this)
  (match-define (circ r col) this)
  (circle r “solid” col)); 2htdp/image fn
```

```scheme
;; render-rect : Rectangle -> Image
(define (render-rect this)
  (match-define (rect w h col) this)
  (rectangle w h “solid” col)); 2htdp/image fn
```
CS450 vs OO Comparison

CS 450 Design Recipe

• Compound data (struct) has (possibly function) fields!
• Itemization Data Defs explicitly defined
• Functions organized by the kind of data they process!
• Explicit itemization dispatch (cond)
• Constructor explicitly includes method defs
• Data to process is explicit arg

OO Programming

• Compound data (class) group fields and methods together!
• Itemization Data Defs implied by interface / class definitions
• Methods organized by the kind of data they process!
• Implicit itemization dispatch
• Constructor implicitly includes method defs
• Data to process (“this”) is implicit arg
How to Design ... OO-Style Programs

• For **Itemization Data Definition**
  1. List Item Data Defs (and other prev data def parts)
  2. Specify required methods
  3. Define “abstract” struct *(with # fields = # of methods)*
  4. Define explicit dispatch function(s) (one per method)

```plaintext
;; A Shape is one of:
;; - Rectangle
;; - Circle
;; interp: Represents shape to draw on a canvas
;; Required methods:
;; - render : Shape -> Image
```

```plaintext
(struct Shape [render-meth])
```

```plaintext
;; render : Shape -> Image
(define (render sh) ((shape-render-meth sh) sh))
```
How to Design ... OO-Style Programs

• For Itemization Data Definition
  1. List Item Data Defs (and other prev data def parts)
  2. Specify required methods
  3. Define “abstract” struct (with # fields = # of methods)
  4. Define explicit dispatch function(s) (one per method)

• For each item:
  1. Define separate Data def
  2. Define a struct, as substruct of “abstract” struct
  3. Define required methods
  4. Define constructor that includes method implementations
How to Design ... OO-Style Programs

Data Definition: Defs (and other Defs) are added, named methods are "methods" of the "struct" struct (which dispatch functions)

- For each item:
  1. Define separate Data def
  2. Define a struct, as substruct of "abstract" struct (which includes methods)
  3. Define required methods
  4. Define constructor that includes methods

;; A Rectangle is a:
;; (rect width : Num
;;   height : Num
;;   color : Color)

;; A Circle is a:
;; (circ radius : Num
;;   color : Color)

;; constructors create shape “objects”
(define (mk-rect w h col)
  [render-meth render-rect])
(define (mk-circ r col)
  [render-meth render-circ])

;; render-rect : Rectangle -> Image
(define (render-rect this)
  (match-define (rect w h col) this)
  (rectangle w h “solid” col)); 2htdp/image fn

;; render-circ : Circle -> Image
(define (render-circ this)
  (match-define (circ r col) this)
  (circle r “solid” col)); 2htdp/image fn

(struct rect Shape [w h col])
(struct circ Shape [r col])
HW 10 Preview: Drawing Shapes!

Use **Racket** to create an OO-Style shape drawing app.

Supports
- Rectangles!
- Circles!
In-class Coding 12/11: start hw10

Use RACKET to create an OO-Style shape drawing app.

Circle
Num r;
Posn center; // center coordinate

// places circle into given canvas
Image place-shape( Image canvas ) {
    return
    place-image( circ-img( r, … ),
        … center
        … canvas );
}

Rectangle
Num w; Num h;
Posn offset; // topleft corner

// places rect into given canvas
Image place-shape( Image canvas ) {
    return
    place-image ( rect-img(w, h, …)
        … offset
        … canvas);
}
Submit your in-class work to github