Introduction to Functional Classes in CS1

Marco T. Morazán

Seton Hall University

Introduction to Functional Classes in CS1

Introduction

- Last year at TFPIE

Design of Classes I

Marco T. Morazán
Seton Hall University
morazanm@shu.edu

The use of functional programming languages in the first programming course at many universities is well-established and effective. Invariably, however, students must progress to study object-oriented programming. This article presents how the first steps of this transition have been successfully implemented at Seton Hall University. The developed methodology builds on the students' experience with type-based design acquired in their previous introduction to programming courses. The transition is made smooth by explicitly showing students that the design lessons they have internalized are relevant in object-oriented programming. This allows for new abstractions offered by object-oriented programming languages to be more easily taught and used by students. Empirical evidence collected from students in the course suggests that the approach developed is effective and that the transition is smooth.

1 Introduction

- Designed based introductory courses to programming, inspired by the approach put forth in the textboo

Introduction to Functional Classes in CS1

Introduction

- Design-based introduction to programming courses are well-established
- Students are exposed to first-class functions: values

Introduction to Functional Classes in CS1

Introduction

- Design-based introduction to programming courses are well-established
- Students are exposed to first-class functions: values
-

```
;; (X → Y) (Y → Z) → (X → Z)
;; Purpose: Return a function for (f ∘ g)(x)
(define (compose-f-g g f)
  (λ (an-x) (f (g an-x))))

(define add2 (compose-f-g add1 add1))
(define id   (compose-f-g add1 sub1))

(check-expect (add2 3)  5)
(check-expect (id 10)  10)
```

Introduction to Functional Classes in CS1

Introduction

- The other side of the coin is rarely explored to any depth: data as functions

Introduction to Functional Classes in CS1

Introduction

- The other side of the coin is rarely explored to any depth: data as functions
- Missed opportunity to introduce object-oriented concepts

Introduction to Functional Classes in CS1

Introduction

- The other side of the coin is rarely explored to any depth: data as functions
- Missed opportunity to introduce object-oriented concepts
- Such an introduction is important
  - Facilitates the transition to object-oriented programming (OOP) and design
  - Appeases other instructors

Introduction to Functional Classes in CS1

Introduction

- The other side of the coin is rarely explored to any depth: data as functions
- Missed opportunity to introduce object-oriented concepts
- Such an introduction is important
  - Facilitates the transition to object-oriented programming (OOP) and design
  - Appeases other instructors
- Main idea: a function implements an interface using message-passing
  - Start with compound data of finite size that does not have variety
  - Move onto implementing union types including compound data of arbitrary size.

Introduction to Functional Classes in CS1

Related Work

- OOP textbooks: emphasize that classes and objects combine code and data

Introduction to Functional Classes in CS1

Related Work

- OOP textbooks: emphasize that classes and objects combine code and data
  - An object is a structure for incorporating data and the procedures for working with that data

Introduction to Functional Classes in CS1

Related Work

- OOP textbooks: emphasize that classes and objects combine code and data
  - An object is a structure for incorporating data and the procedures for working with that data
  - A class is a group of objects that share common state and behavior

Introduction to Functional Classes in CS1

Related Work

- OOP textbooks: emphasize that classes and objects combine code and data
  - An object is a structure for incorporating data and the procedures for working with that data
  - A class is a group of objects that share common state and behavior
  - Every object is an instance of a class, which serves as the type of the object and as a blueprint, defining the data which the object stores and the methods for accessing and modifying that data

Introduction to Functional Classes in CS1

Related Work

- OOP textbooks: emphasize that classes and objects combine code and data
  - An object is a structure for incorporating data and the procedures for working with that data
  - A class is a group of objects that share common state and behavior
  - Every object is an instance of a class, which serves as the type of the object and as a blueprint, defining the data which the object stores and the methods for accessing and modifying that data
  - Objects are used to combine data with procedures that operate on that data

Introduction to Functional Classes in CS1

Related Work

- OOP textbooks: emphasize that classes and objects combine code and data
  - An object is a structure for incorporating data and the procedures for working with that data
  - A class is a group of objects that share common state and behavior
  - Every object is an instance of a class, which serves as the type of the object and as a blueprint, defining the data which the object stores and the methods for accessing and modifying that data
  - Objects are used to combine data with procedures that operate on that data
- Show examples to illustrate how to use classes and how to create objects

Introduction to Functional Classes in CS1

Related Work

- OOP textbooks: emphasize that classes and objects combine code and data
  - An object is a structure for incorporating data and the procedures for working with that data
  - A class is a group of objects that share common state and behavior
  - Every object is an instance of a class, which serves as the type of the object and as a blueprint, defining the data which the object stores and the methods for accessing and modifying that data
  - Objects are used to combine data with procedures that operate on that data
- Show examples to illustrate how to use classes and how to create objects
- The unstated assumption is that beginning students absorb design principles by studying code samples

Introduction to Functional Classes in CS1

Related Work

- The unpublished textbook `How to Design Classes` is closely coupled with a `DSL` known as `ProfessorJ`

Introduction to Functional Classes in CS1

Related Work

- The unpublished textbook `How to Design Classes` is closely coupled with a `DSL` known as `ProfessorJ`
- Starts with classes as similar to a structure

Introduction to Functional Classes in CS1

Related Work

- The unpublished textbook `How to Design Classes` is closely coupled with a `DSL` known as `ProfessorJ`
- Starts with classes as similar to a structure
- An object–an instance of the type defined by the class

Introduction to Functional Classes in CS1

Related Work

- The unpublished textbook `How to Design Classes` is closely coupled with a `DSL` known as `ProfessorJ`
- Starts with classes as similar to a structure
- An object–an instance of the type defined by the class
- The design of a class starts by understanding how data is to be represented

Introduction to Functional Classes in CS1

Related Work

- The unpublished textbook `How to Design Classes` is closely coupled with a `DSL` known as `ProfessorJ`
- Starts with classes as similar to a structure
- An object–an instance of the type defined by the class
- The design of a class starts by understanding how data is to be represented
- Union types are introduced to motivate the need for interfaces

Introduction to Functional Classes in CS1

Related Work

- The unpublished textbook `How to Design Classes` is closely coupled with a `DSL` known as `ProfessorJ`
- Starts with classes as similar to a structure
- An object–an instance of the type defined by the class
- The design of a class starts by understanding how data is to be represented
- Union types are introduced to motivate the need for interfaces
- We use the `Racket` student languages, a specialized `DSL` is not required

Introduction to Functional Classes in CS1

Related Work

- A hybrid approach proposes the use of several `DSLs` embedded in `Racket` before making the full transition into `Java` in a *second semester* course

Introduction to Functional Classes in CS1

Related Work

- A hybrid approach proposes the use of several `DSLs` embedded in `Racket` before making the full transition into `Java` in a *second semester* course
- Relies heavily of the design recipe to provide a framework for program design and discussion

Introduction to Functional Classes in CS1

Related Work

- A hybrid approach proposes the use of several `DSLs` embedded in `Racket` before making the full transition into `Java` in a *second semester* course
- Relies heavily of the design recipe to provide a framework for program design and discussion
- Union types and design based on structural recursion are introduced as requiring a distinct class for each subtype

Introduction to Functional Classes in CS1

Related Work

- A hybrid approach proposes the use of several `DSLs` embedded in `Racket` before making the full transition into `Java` in a *second semester* course
- Relies heavily of the design recipe to provide a framework for program design and discussion
- Union types and design based on structural recursion are introduced as requiring a distinct class for each subtype
- Transition to `Java` occurs in the middle of the semester

Introduction to Functional Classes in CS1

Related Work

- A hybrid approach proposes the use of several `DSLs` embedded in `Racket` before making the full transition into `Java` in a *second semester* course
- Relies heavily of the design recipe to provide a framework for program design and discussion
- Union types and design based on structural recursion are introduced as requiring a distinct class for each subtype
- Transition to `Java` occurs in the middle of the semester
- Our work is not part of an `OOP` course
- Introduce students to object-oriented abstractions within the context of a first programming course using a program by design methodology

Introduction to Functional Classes in CS1

Student Background

- Use `HtDP`
  - Functions
  - Compound data: finite and arbitrary size
  - Functional abstraction: λ and curried functions

Introduction to Functional Classes in CS1

Student Background

```
(define (scale-by-2 L) (map (λ (x) (* 2 x)) L))
(define (scale-by-5 L) (map (λ (x) (* 5 x)) L))
(define (scale-by-3 L) (map (λ (x) (* 3 x)) L))
```

Introduction to Functional Classes in CS1

Student Background

```
(define (scale-by-2 L) (map (λ (x) (* 2 x)) L))
(define (scale-by-5 L) (map (λ (x) (* 5 x)) L))
(define (scale-by-3 L) (map (λ (x) (* 3 x)) L))

;; number → ((listof number) → (listof number))
(define (make-list-scaler sc)
  (λ (L) (map (λ (x) (* sc x)) L)))
```

Introduction to Functional Classes in CS1

Student Background

```
(define (scale-by-2 L) (map (λ (x) (* 2 x)) L))
(define (scale-by-5 L) (map (λ (x) (* 5 x)) L))
(define (scale-by-3 L) (map (λ (x) (* 3 x)) L))

;; number → ((listof number) → (listof number))
(define (make-list-scaler sc)
  (λ (L) (map (λ (x) (* sc x)) L)))
```

- `make-list-scaler` is
  - a curried function
  - a *class* that returns `objects` that scale a list of numbers by a given scalar

Introduction to Functional Classes in CS1

Student Background

```
(define (scale-by-2 L) (map (λ (x) (* 2 x)) L))
(define (scale-by-5 L) (map (λ (x) (* 5 x)) L))
(define (scale-by-3 L) (map (λ (x) (* 3 x)) L))

;; number → ((listof number) → (listof number))
(define (make-list-scaler sc)
  (λ (L) (map (λ (x) (* sc x)) L)))
```

- `make-list-scaler` is
  - a curried function
  - a *class* that returns `objects` that scale a list of numbers by a given scalar
- Connection is not automatically made by students
- Result is twofold:
  - Students gain more interest in the techniques being taught
  - Students obtain an understanding of some `OOP` abstractions

Introduction to Functional Classes in CS1

Structures as Interfaces

- Data definition defines a type
- States nothing about the valid type operations

Introduction to Functional Classes in CS1

Structures as Interfaces

- Data definition defines a type
- States nothing about the valid type operations
- The valid operations are defined in an *interface*

Introduction to Functional Classes in CS1

Structures as Interfaces

```
;; A 3Dposn is a structure:
;;   (make-3Dposn number number number)
(define-struct 3Dposn (x y z))

;; 3Dposn → number
;; Purpose: Return distance to the origin of given 3Dposn
(define (dist-origin a-3dposn)
  (sqrt (+ (sqr (3Dposn-x a-3dposn))
           (sqr (3Dposn-y a-3dposn))
           (sqr (3Dposn-z a-3dposn)))))
```

Introduction to Functional Classes in CS1

Structures as Interfaces

```
;; A 3Dposn is a structure:
;;   (make-3Dposn number number number)
(define-struct 3Dposn (x y z))

;; 3Dposn → number
;; Purpose: Return distance to the origin of given 3Dposn
(define (dist-origin a-3dposn)
  (sqrt (+ (sqr (3Dposn-x a-3dposn))
           (sqr (3Dposn-y a-3dposn))
           (sqr (3Dposn-z a-3dposn)))))
```

3Dposn interface:

- Request `x`: `number`
- Request `y`: `number`
- Request `z`: `number`
- Request `distance`: `number`

Introduction to Functional Classes in CS1

Structures as Interfaces

- Explicitly relate data and operations

Introduction to Functional Classes in CS1

Structures as Interfaces

- Explicitly relate data and operations
- Related suggests encapsulation

Introduction to Functional Classes in CS1

Structures as Interfaces

- Explicitly relate data and operations
- Related suggests encapsulation
- Functionality provided via *message-passing*

Introduction to Functional Classes in CS1

Structures as Interfaces

- Explicitly relate data and operations
- Related suggests encapsulation
- Functionality provided via *message-passing*
- An interface is implemented by a constructor function
  - called a *class*
  - Returns a message-processing function
  - It's a curried function

Introduction to Functional Classes in CS1

Structures as Interfaces

- Explicitly relate data and operations
- Related suggests encapsulation
- Functionality provided via *message-passing*
- An interface is implemented by a constructor function
  - called a *class*
  - Returns a message-processing function
  - It's a curried function
- An interface must specify the messages used to request a service
- Refine the `3Dposn`:

```
'getx: number
'gety: number
'getz: number
 'd2o: number
```

Introduction to Functional Classes in CS1

Structures and Interfaces

```
(require 2htdp/abstraction)

;; number number number → 3Dposn  Purpose: Return a 3Dposn object
(define (make-3Dposn x y z)
(local [;; 3Dposn → number Purpose: Return distance to origin
        (define (dist-origin x y z) (sqrt (+ (sqr x) (sqr y) (sqr z))))
        ;; message → 3Dposn service throws error  Purpose: Manage messages
        (define (manager m)
          (match m
            ['getx x]
            ['gety y]
            ['getz z]
            ['d2o  (dist-origin x y z)]
            [else (error (string-append "Unknown message to 3Dposn: "
                                        (symbol->string m)))]))]
  manager))
```

Introduction to Functional Classes in CS1

Structures and Interfaces

```
(require 2htdp/abstraction)

;; number number number → 3Dposn  Purpose: Return a 3Dposn object
(define (make-3Dposn x y z)
(local [;; 3Dposn → number Purpose: Return distance to origin
        (define (dist-origin x y z) (sqrt (+ (sqr x) (sqr y) (sqr z))))
        ;; message → 3Dposn service throws error  Purpose: Manage messages
        (define (manager m)
          (match m
            ['getx x]
            ['gety y]
            ['getz z]
            ['d2o  (dist-origin x y z)]
            [else (error (string-append "Unknown message to 3Dposn: "
                                        (symbol->string m)))]))]
  manager))

(define (3Dposn-x a-3dposn) (a-3dposn 'getx))

(define (3Dposn-y a-3dposn) (a-3dposn 'gety))

(define (3Dposn-z a-3dposn) (a-3dposn 'getz))

(define (dist-origin a-3dposn) (a-3dposn 'd20))
```

Introduction to Functional Classes in CS1

Services that Require More Input

- Add a service that requires more input: Return a function

Introduction to Functional Classes in CS1

Services that Require More Input

- Add a service that requires more input: Return a function

Introduction to Functional Classes in CS1

Services that Require More Input

- Add a service that requires more input: Return a function
- Add distance to

```
'getx: number
'gety: number
'getz: number
 'd2o: number
 'd2p: 3Dposn → number
```

Introduction to Functional Classes in CS1

Services that Require More Input

- Add a service that requires more input: Return a function
- Add distance to

```
'getx: number
'gety: number
'getz: number
 'd2o: number
 'd2p: 3Dposn → number
```

-
```
;; message → 3Dposn service throws error  Purpose: ...
(define (manager m)
  (match m
    ['getx x]
    ['gety y]
    ['getz z]
    ['d2o  (dist-origin x y z)]
    ['d2p  distance]
    [else (error ...)]))
```

Introduction to Functional Classes in CS1

Services that Require More Input

- Add a service that requires more input: Return a function
- Add distance to

```
'getx: number
'gety: number
'getz: number
 'd2o: number
 'd2p: 3Dposn → number
```

-
```
;; message → 3Dposn service throws error  Purpose: ...
(define (manager m)
  (match m
    ['getx x]
    ['gety y]
    ['getz z]
    ['d2o  (dist-origin x y z)]
    ['d2p  distance]
    [else (error ...)]))
```

-
```
;; 3Dpson → number  Purpose: Compute distance from this to given 3Dposn
(define (distance a-3dposn)
  (sqrt (+ (sqr (- x (3Dposn-x a-3dposn)))
           (sqr (- y (3Dposn-y a-3dposn)))
           (sqr (- z (3Dposn-z a-3dposn))))))
```

Introduction to Functional Classes in CS1

Services that Require More Input

- Add a service that requires more input: Return a function
- Add distance to

```
'getx: number
'gety: number
'getz: number
 'd2o: number
 'd2p: 3Dposn → number
```

-
```
;; message → 3Dposn service throws error  Purpose: ...
(define (manager m)
  (match m
    ['getx x]
    ['gety y]
    ['getz z]
    ['d2o  (dist-origin x y z)]
    ['d2p  distance]
    [else (error ...)]))
```

-
```
;; 3Dpson → number  Purpose: Compute distance from this to given 3Dposn
(define (distance a-3dposn)
  (sqrt (+ (sqr (- x (3Dposn-x a-3dposn)))
           (sqr (- y (3Dposn-y a-3dposn)))
           (sqr (- z (3Dposn-z a-3dposn))))))
```

-
```
(define (3Dposn-distance p1 p2) ((p1 'd2p) p2))
```

Introduction to Functional Classes in CS1

A Design Recipe for Interfaces

1. Identify the values that must be stored and the services that must be provided.

Introduction to Functional Classes in CS1

A Design Recipe for Interfaces

1. Identify the values that must be stored and the services that must be provided.
2. Develop an interface data definition and a data definition for messages.

Introduction to Functional Classes in CS1

A Design Recipe for Interfaces

1. Identify the values that must be stored and the services that must be provided.
2. Develop an interface data definition and a data definition for messages.
3. Develop a function template for the class that consumes the values that must be stored and whose body is a `local`-expression returning the message-processing function.

Introduction to Functional Classes in CS1

A Design Recipe for Interfaces

1. Identify the values that must be stored and the services that must be provided.
2. Develop an interface data definition and a data definition for messages.
3. Develop a function template for the class that consumes the values that must be stored and whose body is a `local`-expression returning the message-processing function.
4. Specialize the signature, purpose, class header, and message-processing function.

Introduction to Functional Classes in CS1

A Design Recipe for Interfaces

1. Identify the values that must be stored and the services that must be provided.
2. Develop an interface data definition and a data definition for messages.
3. Develop a function template for the class that consumes the values that must be stored and whose body is a `local`-expression returning the message-processing function.
4. Specialize the signature, purpose, class header, and message-processing function.
5. Write and make local the auxiliary functions needed by the message-passing function.

Introduction to Functional Classes in CS1

A Design Recipe for Interfaces

1. Identify the values that must be stored and the services that must be provided.
2. Develop an interface data definition and a data definition for messages.
3. Develop a function template for the class that consumes the values that must be stored and whose body is a `local`-expression returning the message-processing function.
4. Specialize the signature, purpose, class header, and message-processing function.
5. Write and make local the auxiliary functions needed by the message-passing function.
6. Write and test a wrapper function for each service.

Introduction to Functional Classes in CS1

Implementing Union Types

```
;; A square is a structure
;;     (make-sq number symbol symbol)
;; with a length, a mode, and a color.
(define-struct sq (length mode color))

;; A rectangle is a structure
;;     (make-rect number number symbol symbol)
;; with a width, length, a mode, and a color.
(define-struct rect (width length mode color))

;; A circle is a structure
;;     (make-circ number symbol symbol)
;; with a length, a mode, and a color.
(define-struct circ (radius mode color))

;; A geometric shape, gs, is either:
;;  1. sq
;;  2. rect
;;  3. circ
```

Introduction to Functional Classes in CS1

Implementing Union Types

- Designing interfaces for a union type requires individually reasoning about each subtype
- Not a huge intellectual leap for students

Introduction to Functional Classes in CS1

Step 1: Values and Services

- Three classes are needed: one for each subtype
  - The `sq` class needs to store a number and two symbols
  - The `rect` class needs to stores two numbers and two symbols
  - The `circ` class needs to store a number and two symbols

Introduction to Functional Classes in CS1

Step 1: Values and Services

- Three classes are needed: one for each subtype
  - The `sq` class needs to store a number and two symbols
  - The `rect` class needs to stores two numbers and two symbols
  - The `circ` class needs to store a number and two symbols
- The new step is defining the behavior for a union type
  - Determine if the gs ia a sq
  - Determine if the gs ia a rect
  - Determine if the gs ia a circ
  - Compute the gs's area
  - Determine if the area of this gs is larger than a given gs

Introduction to Functional Classes in CS1

Step 2: Interface and Message Definitions

```
;; A gs is an interface offering:
;;     'is-sq?:  Boolean
;;   'is-rect?:  Boolean
;;   'is-circ?:  Boolean
;;       'area:  number
;;    'bigger?:  gs → Boolean
```

Introduction to Functional Classes in CS1

Step 3: Class Function Template

```
;; ... → gs
;; Purpose: Return a gs object
(define (class-for-gs ...)
  (local
      :
    [ :
     ;; gs → Boolean
     ;; Purpose: Determine if this' area is larger than the given gs' area
     (define (is-this-bigger? a-gs) ...)

     ;; message → service throws error
     ;; Purpose: Provide service for the given message
     (define (manager m)
       (match m
         ['is-sq? ...]
         ['is-rect?  ...]
         ['is-circ?   ...]
         ['area   ...]
         ['bigger?    ...]
         [else (error (format "Unknown gs service requested: ~s" m))]))]
 manager))
```

Introduction to Functional Classes in CS1

Steps 4 and 5: Class Function Template Specialization

```
(require 2htdp/abstraction)
;; number symbol symbol → gs
;; Purpose: Return a sq object
(define (sq length mode color)
  (local                                  Do the same for every subtype
    [;; gs → Boolean
     ;; Purpose: Determine if this' area > given gs' area
     (define (is-this-bigger? a-gs) (> (sqr length) (gs-area a-gs)))

     ;; message → service throws error
     ;; Purpose: Provide service for the given message
     (define (manager m)
       (match m
         ['get-length length]
         ['get-mode   mode]
         ['get-color  color]
         ['is-sq?     #true]
         ['is-rect?   #false]
         ['is-circ?   #false]
         ['area       (sqr length)]
         ['bigger?    is-this-bigger?]
         [else
          (error (format "Unknown gs service requested: ~s" m))]))]
    manager))
```

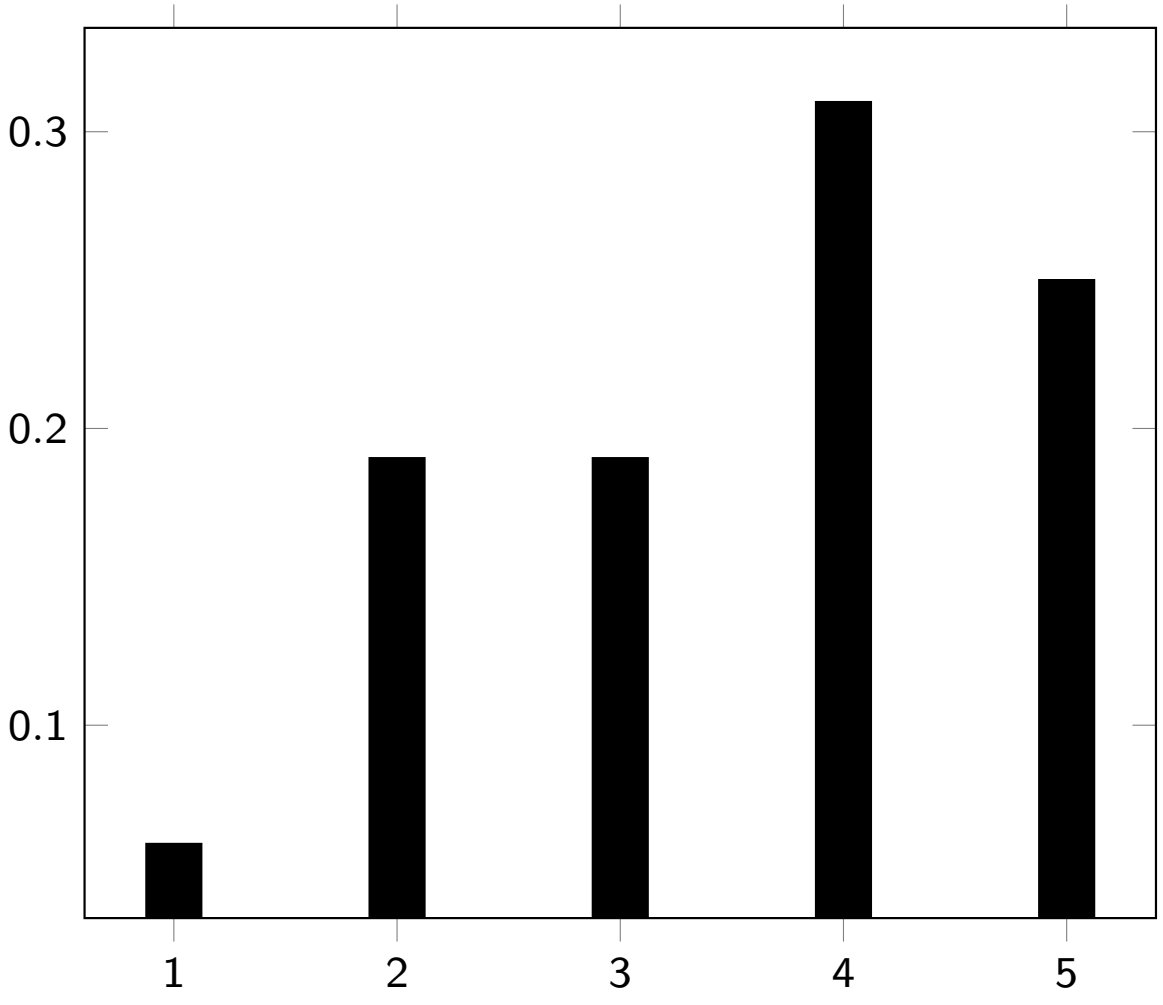
Introduction to Functional Classes in CS1

Step 6: Wrapper Functions and Tests

```
(define SQR1  (sq   5 'outline 'green))
(define RECT1 (rect 2 3 'solid 'blue))
(define CIRC1 (circ 7 'outline 'red))

;; gs → Boolean  Purpose: Determine if given gs is a sq
(define (gs-sq? a-gs) (a-gs 'is-sq?))

;; gs → Boolean  Purpose: Determine if given gs is a rect
(define (gs-rect? a-gs) (a-gs 'is-rect?))

;; gs → Boolean  Purpose: Determine if given gs is a circ
(define (gs-circ? a-gs) (a-gs 'is-circ?))

;; gs → Boolean  Purpose: Compute area of given gs
(define (gs-area a-gs) (a-gs 'area))

;; gs gs → Boolean Purpose: Determine if first given gs > second given gs
(define (gs-bigger? this that) ((this 'bigger?) that))

;; Tests for gs interface
(check-expect (gs-sq? SQR1)    #true)
(check-expect (gs-sq? RECT1)   #false)
(check-expect (gs-sq? CIRC1)   #false)
      :
```

Introduction to Functional Classes in CS1

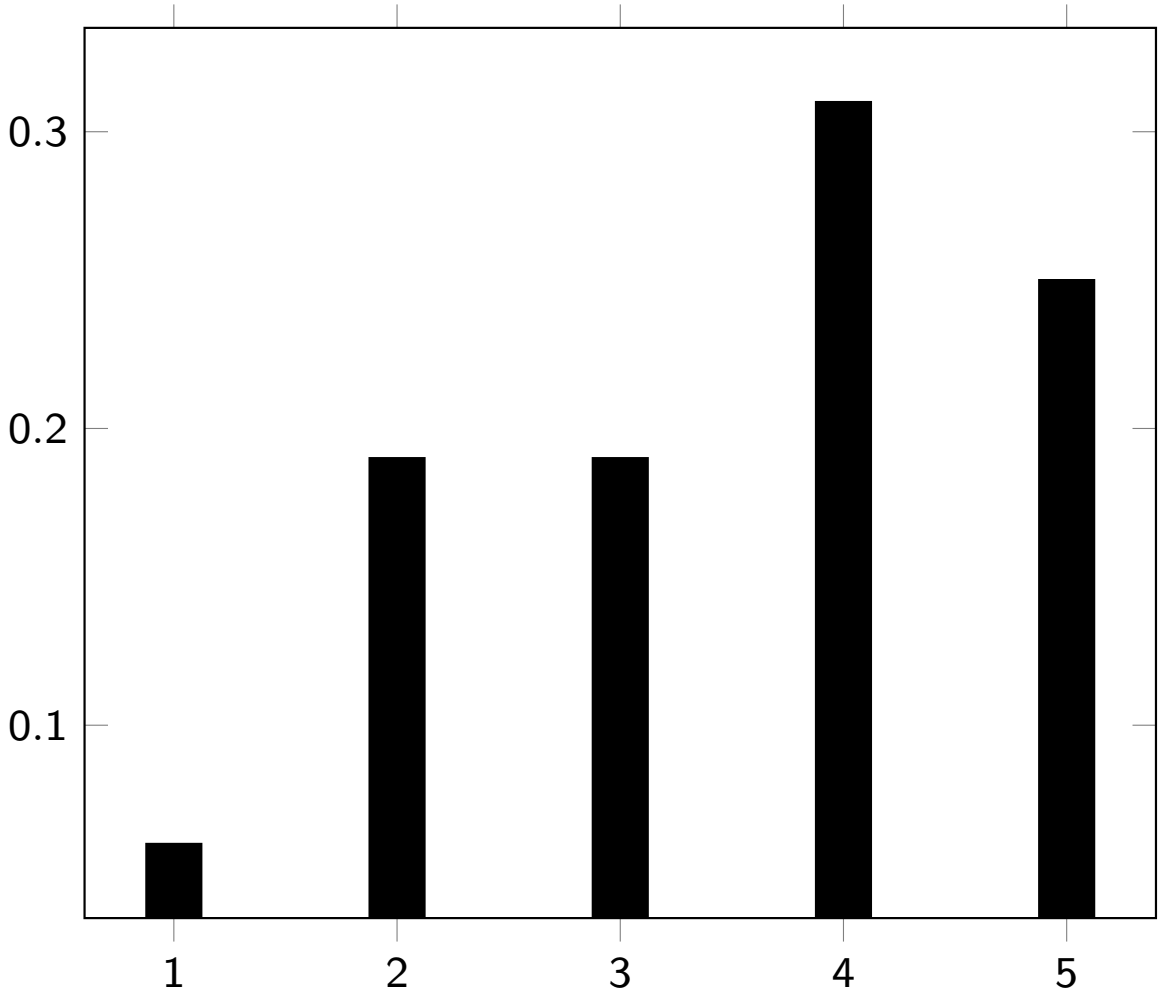
Student Feedback



- How effective How to Design Programs → design classes?

Introduction to Functional Classes in CS1

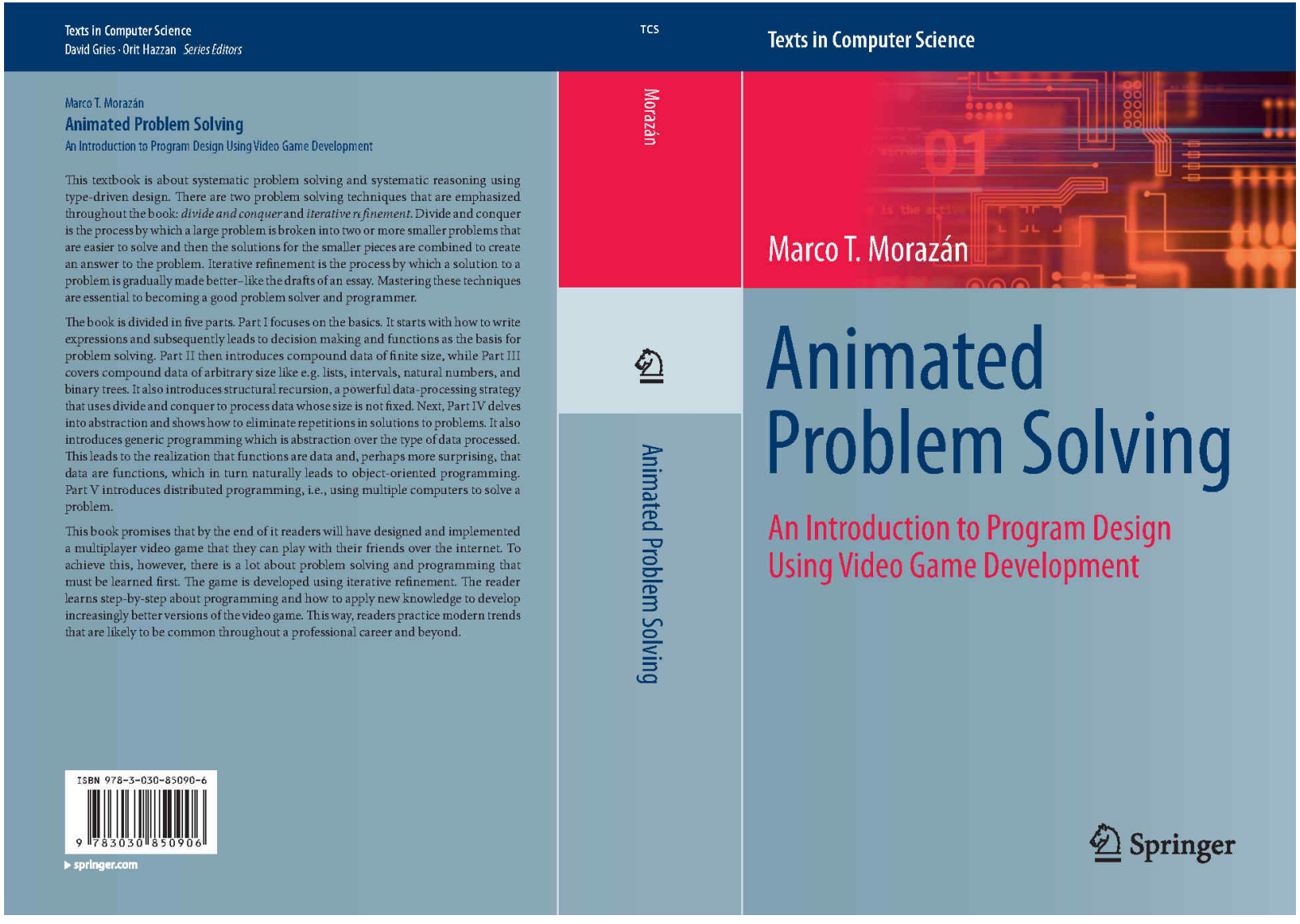
Student Feedback



- How effective How to Design Programs → design classes?
- Students expressed appreciation for classes before learning Java
- Students that arrived with prior OOP experience from high school
  - Expressed feeling frustrated with the approach at the beginning
  - Now feel they understand OOP much better

Introduction to Functional Classes in CS1

Questions



Questions?