CODECRAFT AUGUST 2023

OBJECT-ORIENTED VS FUNCTIONAL - WHAT'S THE DIFFERENCE?

WHAT ARE WE GOING TO DO?

- Look at two programming paradigms
- Build a model to reflect a problem
- Solve a problem using the model
- Reimplement the model in a functional style

WHAT ARE PROGRAMMING PARADIGMS?

Ways to classify programming languages based on their features

- Imperative
  - Procedural
  - Object-Oriented
- Declarative
  - Functional
  - Logic
  - Reactive

https://en.wikipedia.org/wiki/Programming_paradigm

HIGH-LEVEL DIFFERENCES

- Object-Oriented
  - Object-Oriented applications are a “Web of collaborating objects”
  - Behaviour of an object is defined in terms of which messages it sends, the results returned in response to receiving messages and the state mutated by those messages
- Functional
  - *It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures - Alan Perlis*
  - Compose applications built upon a simple set of data-structures e.g. maps, lists, sets
  - State is not implicitly mutated

OBJECT-ORIENTED LANGUAGES

This is not an exhaustive list

- Smalltalk
- C++
- Java
- Python
- Ruby
- JavaScript
- TypeScript

FUNCTIONAL LANGUAGES

This is not an exhaustive list

- Common Lisp
- Haskell
- OCaml
- Clojure
- Elixir

MULTI-PARADIGM LANGUAGES

This is not an exhaustive list

- C++
- Java
- JavaScript
- C#
- Scala
- Common Lisp
- Kotlin
- PHP
- Python
- TypeScript

WAREHOUSE PICKING

DESIGNING WITH OBJECTS!

EXERCISE 1

- Come up with a model of the Warehouse
- Start by thinking about the containers in the model
- What elements can you see in the picture?
- What information do you need to model each of the elements?
- Is there any information missing?

10 minutes

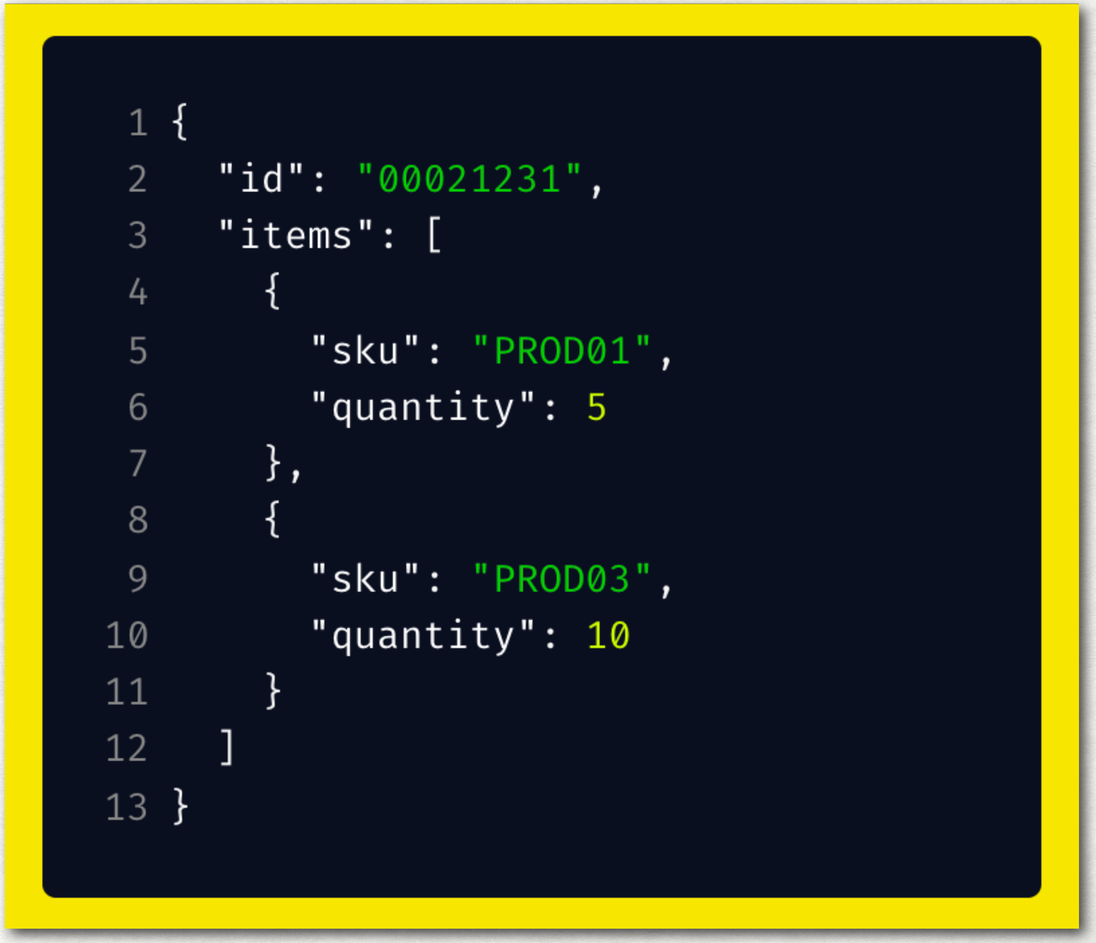
DEFINING A WAREHOUSE

This is some raw data for your model.

```
{
  "products": [
    {
      "id": "PROD01",
      "name": "Grommet"
    }
  ],
  "aisles": [
    {
      "id": "1",
      "bins": [
        {
          "id": "1",
          "contains": {
            "sku": "PROD01",
            "quantity": 50
          }
        },
        {
          "id": "2",
          "contains": {
            "sku": "PROD02",
            "quantity": 5
          }
        }
      ]
    }
  ]
}
```

DEFINING AN ORDER

An order is a set of product codes (SKUs) and the quantity and an ID



```
{
  "id": "00021231",
  "items": [
    {
      "sku": "PROD01",
      "quantity": 5
    },
    {
      "sku": "PROD03",
      "quantity": 10
    }
  ]
}
```

DESIGNING WITH OBJECTS!

- Create a pick list for an order in a warehouse!
- Identify the bins to pick the items from
  - If you're stuck, maybe start with a message like pickListForOrder(order)
- Produce a list of aisle/bins and the number of items to pick from each bin
- You don't need to come up with an optimal order to pick the items in
  - You can choose to only use bins with sufficient items to fulfil the order
- What state should the model be in after you've created the pick list?
  - What sort of things can go wrong with this?

30 minutes

DESIGNING WITH MESSAGES

Objects communicate by sending messages

Think about what messages your model sends between it's "web of collaborating objects"

SOMETHING GOES WRONG

Picker ends their shift unexpectedly - trips and falls and has to leave

How do the the pick list and stock counts differ?

The number of items in the bin isn’t correct!

DESIGN REFLECTION

- What messages are you sending between Objects?
- How does your inventory get updated?
  - What happens if someone else asks for a pick list while another picker is operating on a list?
  - What are the tradeoffs of your design?

FUNCTIONAL PROGRAMMING

DONEC QUIS NUNC

- *It is better to have 100 functions operate on one data structure than to have 10 functions operate on 10 data structures - Alan Perlis*
- Compose applications built upon a simple set of data-structures e.g. maps, lists, sets

```
{
  "products": [
    {
      "id": "PROD01",
      "name": "Grommet"
    }
  ],
  "aisles": [
    {
      "id": "1",
      "bins": [
        {
          "id": "1",
          "contains": {
            "sku": "PROD01",
            "quantity": 50
          }
        },
        {
          "id": "2",
          "contains": {
            "sku": "PROD02",
            "quantity": 5
          }
        }
      ]
    }
  ]
}
```

DESIGNING WITH FUNCTIONS

WHAT'S THE DIFFERENCE?

- Functions accept the data that they will operate on - "referential transparency"
- Functions can accept other functions for filtering and calculating - "high order functions"
- Functions accept data and return **new** data - "persistent data" - doesn't mean "stored to persistent storage"

DESIGNING WITH FUNCTIONS

- Create a pick list for an order in a warehouse!
- Identify the bins to pick the items from
  - If you're stuck, maybe start with a function like findBinWithProduct(aisle, product, quantity)
- Produce a list of aisle/bins and the number of items to pick from each bin
- You don't need to come up with an optimal order to pick the items in
  - You can choose to only use bins with sufficient items to fulfil the order
- What state should the model be in after you've created the pick list?

30 minutes

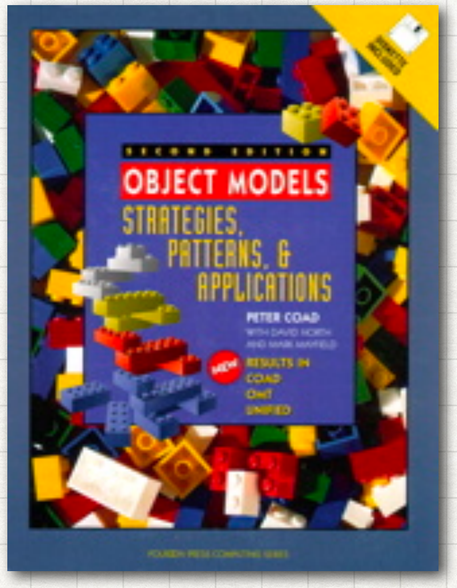
DESIGN REFLECTION

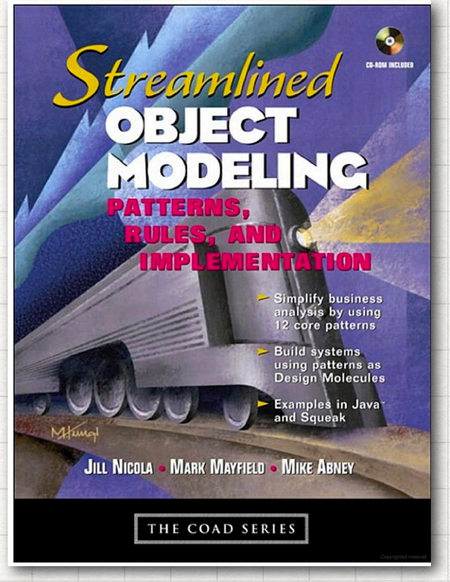
- How reusable are your functions?
- How does your inventory get updated?
  - What happens if someone else asks for a pick list while another picker is operating on a list?
  - What are the tradeoffs of your design?

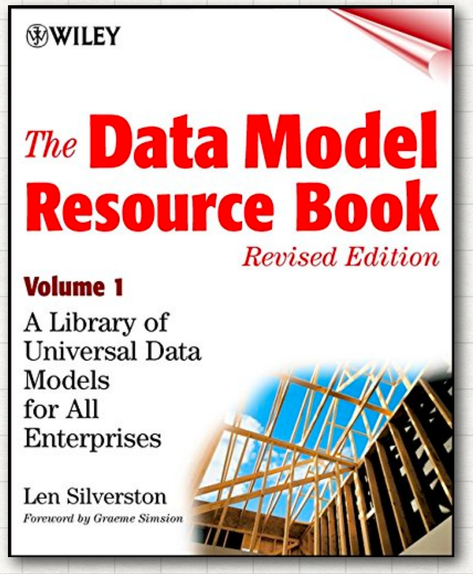
ONE FUNCTION MANY USES

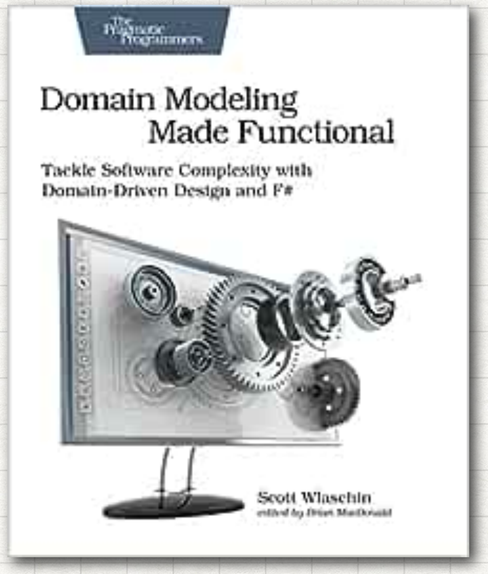
```
interface AisleBin {
    aisleId: string;
    bin: Bin;
}

type binPredicate = (bin: Bin) => boolean;

/**
 * Find matching bins in this warehouse
 * @param warehouse
 * @param pred
 * @returns
 */
export function findWarehouseBins(warehouse: Warehouse, pred: binPredicate): AisleBin[]
{   return warehouse.aisles.flatMap(
        (aisle: Aisle) => {
            return aisle.bins.filter(pred)
              .map((bin: Bin) => {
                return {
                    aisleId: aisle.id,
                    bin: bin,
                };
            });
        });
};
```

THANK YOU!


SECOND EDITION
OBJECT MODELS
STRATEGIES, PATTERNS, & APPLICATIONS
PETER COAD
WITH DAVID NORTH AND MARK MAYFIELD
NEW
RESULTS IN
COAD
OMT
UNIFIED


Streamlined
OBJECT MODELING
PATTERNS, RULES, AND IMPLEMENTATION
CD-ROM INCLUDED
Simplify business analysis by using 12 core patterns
Build systems using patterns as Design Molecules
Examples in Java and Squeak
JILL NICOLA • MARK MAYFIELD • MIKE ABNEY
THE COAD SERIES


WILEY
The Data Model Resource Book
Revised Edition
Volume 1
A Library of Universal Data Models for All Enterprises
Len Silverston
Foreword by Graeme Simsion


The Pragmatic Programmers
Domain Modeling Made Functional
Tackle Software Complexity with Domain-Driven Design and F#
Scott Wlaschin
edited by Brian MacDonald