Domain Models: Classes

Curt Clifton

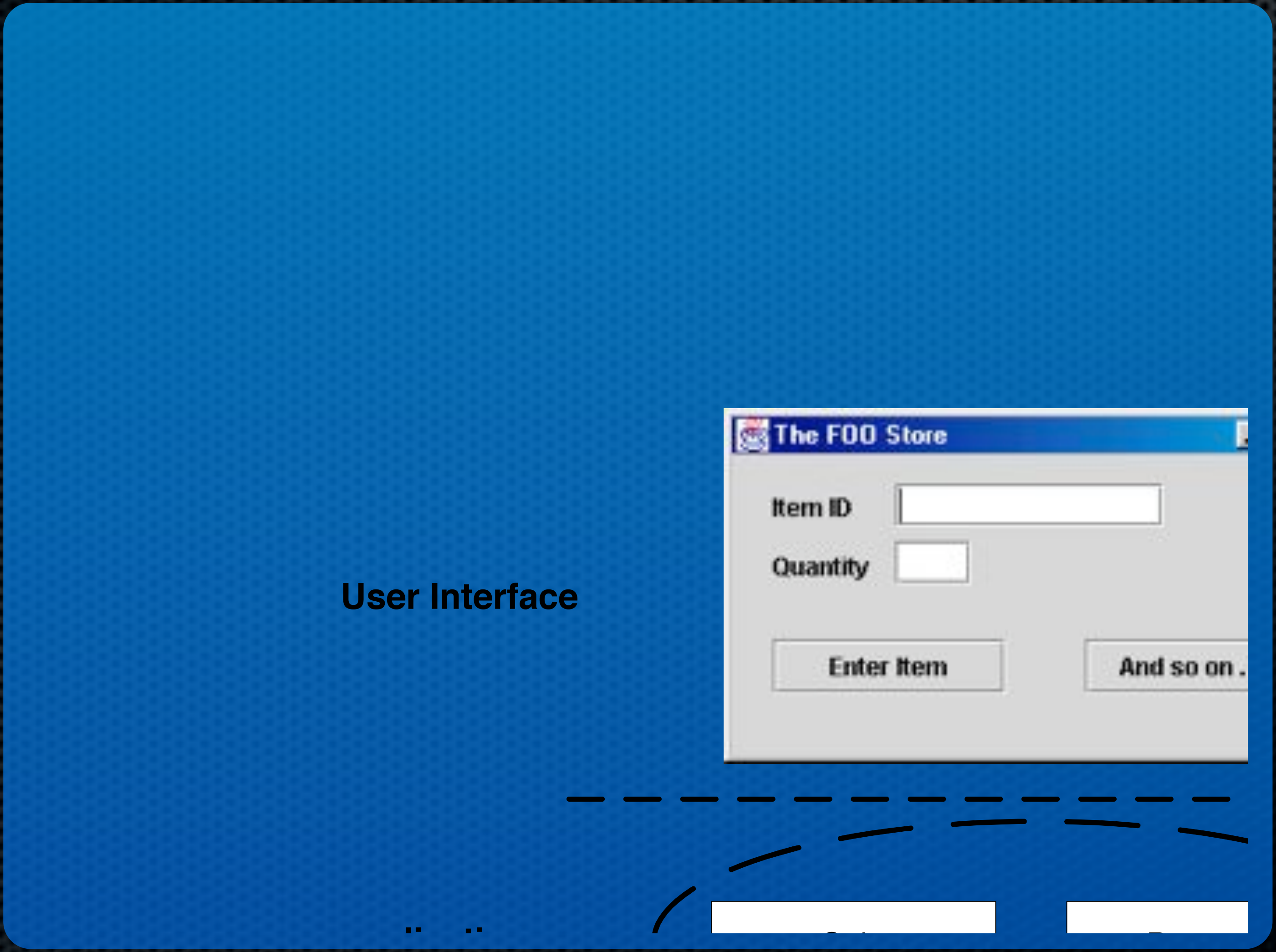
Rose-Hulman Institute of Technology

Q1

Running Case Studies

- NextGen Point of Sale (POS) System
- Monopoly Simulator

Focused on Application Logic Layer



Domain Model

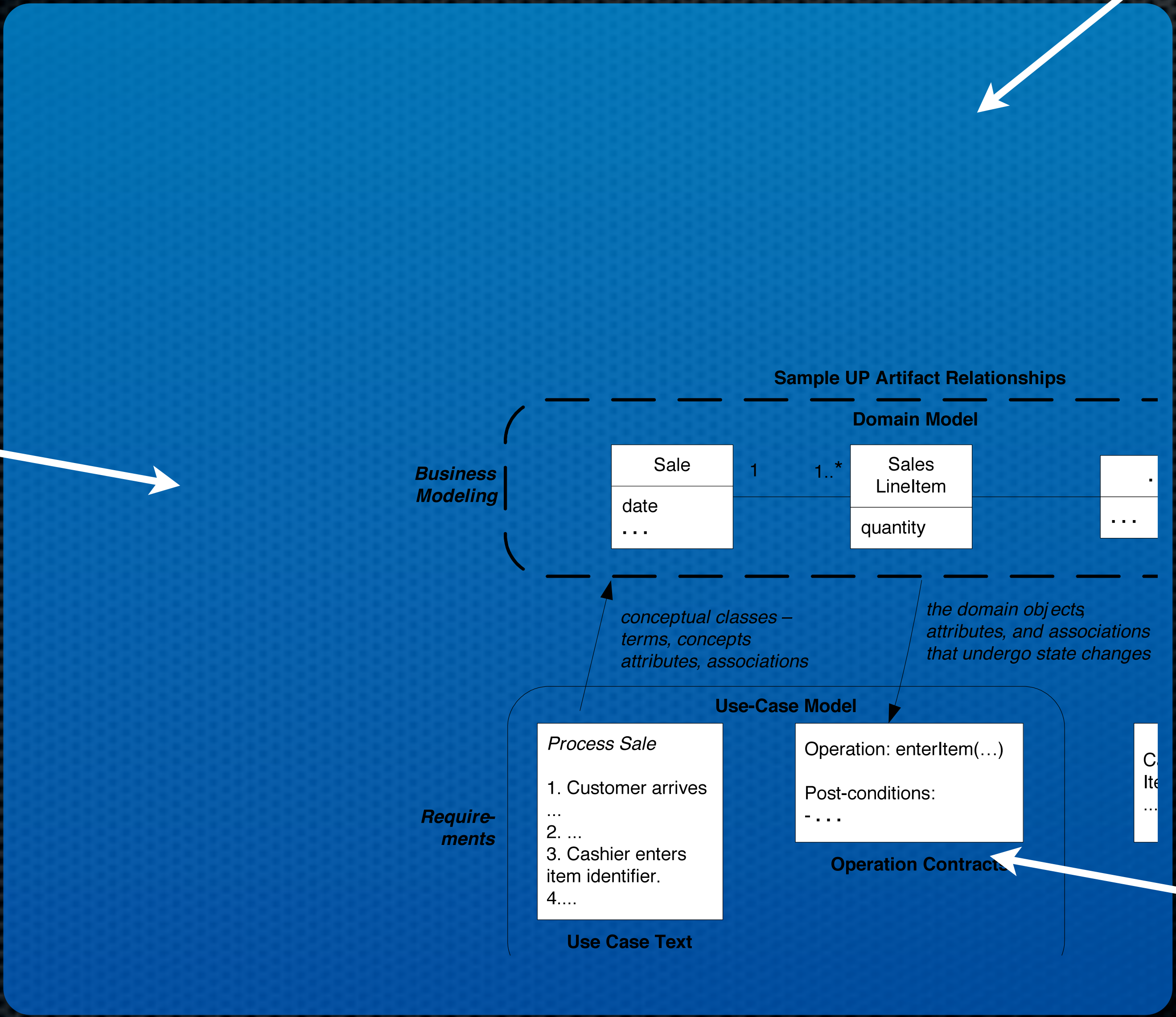
- Most important model in OO **analysis**
- Illustrates **noteworthy** concepts in a domain
- Source of inspiration for designing **some** software objects
- Basic notation is trivial, but it takes practice to build a **useful** model

Where we're going...

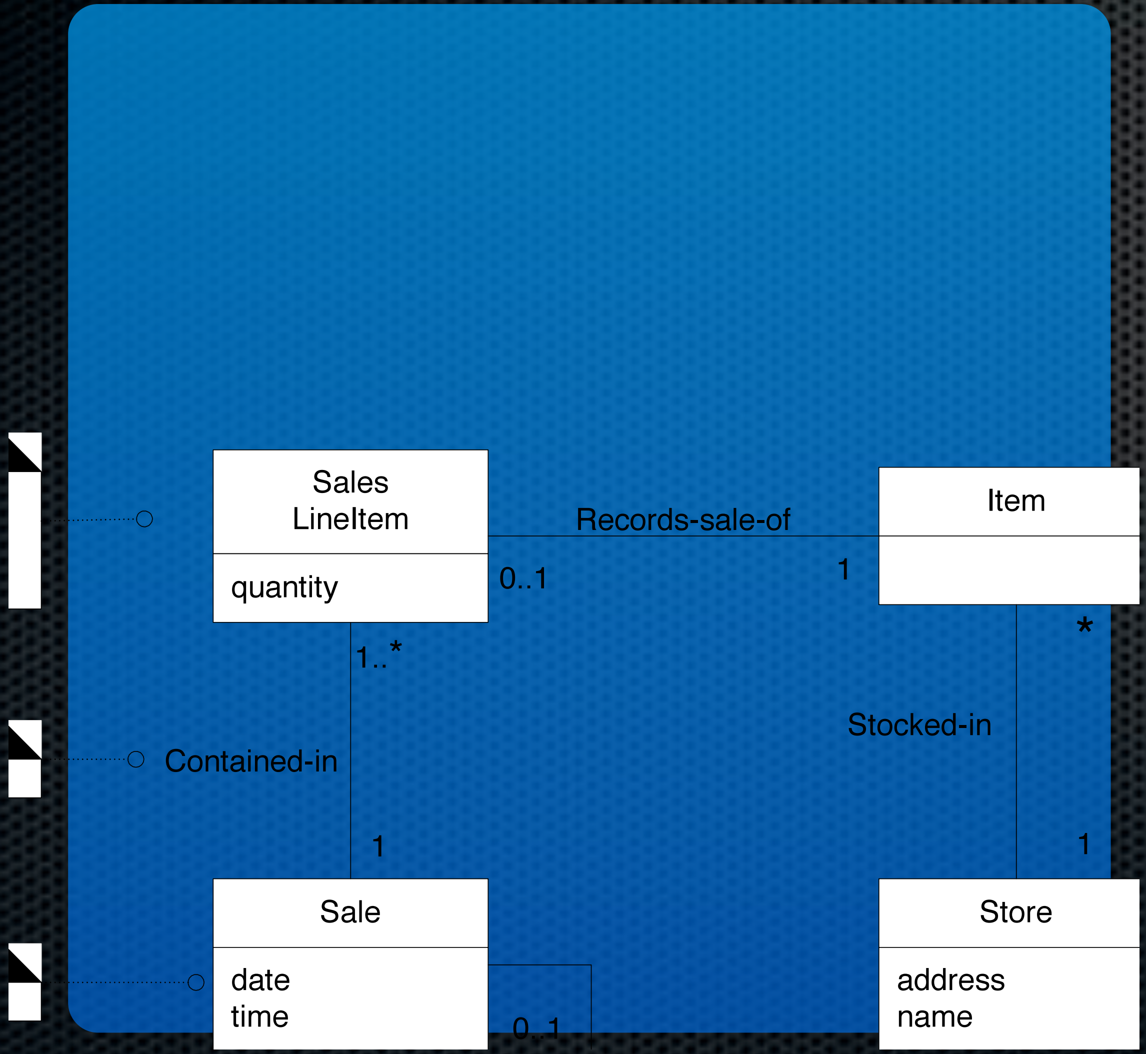
Domain Model

Use Cases

Design Model



Example POS Domain Model



An **abstraction** of the conceptual classes

Q2,3

Conflicting Demands

- Want **rich set of conceptual classes** to support understanding and communication
- Want a **short time investment**

Analysis
Paralysis

What is a Domain Model?

- **Visual representation** of conceptual classes and their relationships
- Focuses on **one domain**
- Illustrated using UML class diagrams **without operations**

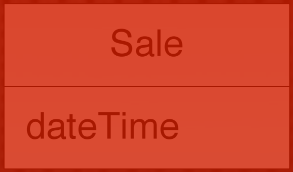
Q4,5

Pitfalls

Designing too Soon

Good

Avoid



visualization of a r
the domain of inte

it is a *not* a pictur

Confusing Terms

- Domain **model** vs. domain **layer**
- Domain layer typically refers to a part of a software solution that simulates the real-world domain

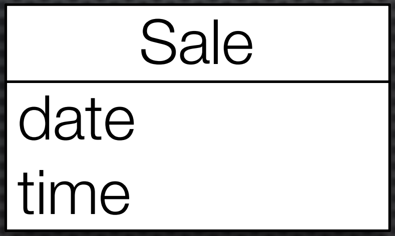
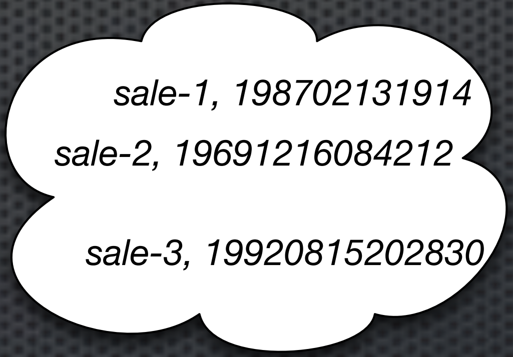
Confusion with Databases

- Domain model ≠ data model
- Data models:
  - Only show persistent data
  - Exclude classes that don’t have attributes
- Domain models may include:
  - External actors, transient data, any real-world classes

Conceptual Classes

- A conceptual class is an idea, thing, or object
- Formally, a conceptual class can be represented as:
  - a **symbol**,
  - its **intension**, or
  - its **extension**

Conceptual Class, Formally

	Example	Not Just an OO Idea
Symbol		$\mathbb{N}$
Intension	"A sale represents the event of a purchase transaction. It has a date and time."	$\{x \in \mathbb{Z} \mid x \geq 0\}$
Extension	 sale-1, 198702131914 sale-2, 19691216084212 sale-3, 19920815202830	$\{0, 1, 2, 3, \ldots\}$

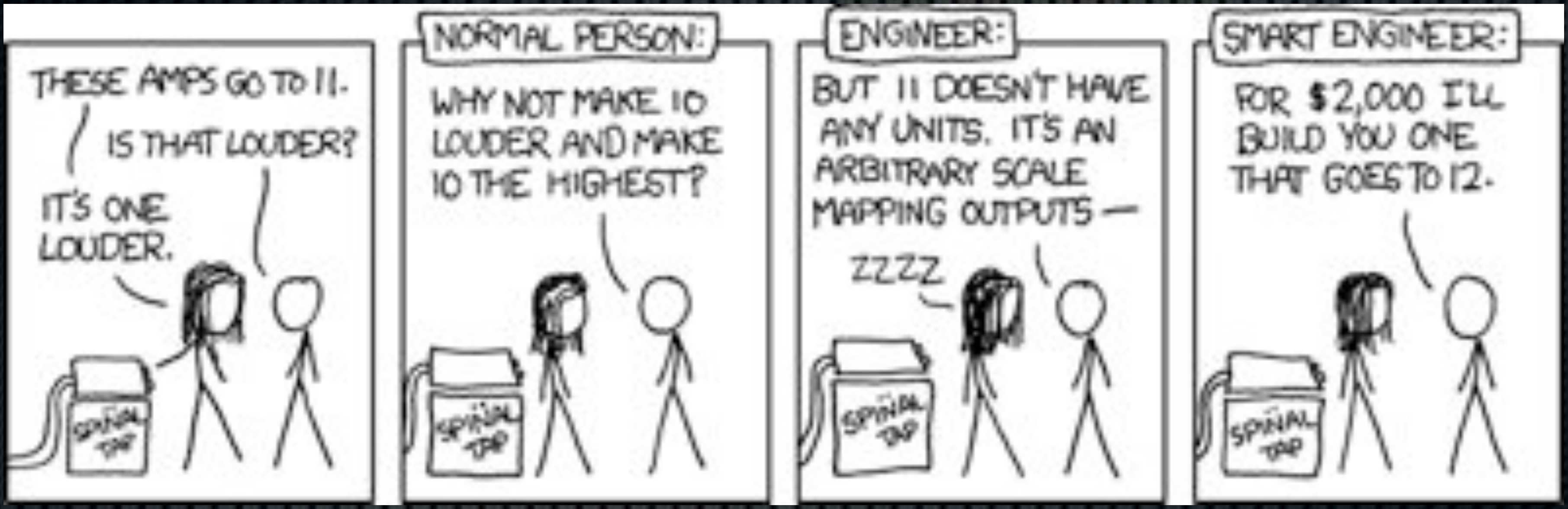
Why Create a Domain Model?

- Names from **domain model** move into the **domain layer** in the software
- Goal: lower **representational gap**
- Helps us:
  - Understand the software
  - Maintain the software

How?

Q6

It's important to understand your customer's domain…


THESE AMPS GO TO 11.
IS THAT LOUDER?
IT'S ONE LOUDER.
SPINAL TAP
NORMAL PERSON:
WHY NOT MAKE 10 LOUDER AND MAKE 10 THE HIGHEST?
ENGINEER:
BUT 11 DOESN'T HAVE ANY UNITS. IT'S AN ARBITRARY SCALE MAPPING OUTPUTS —
ZZZZ
SMART ENGINEER:
FOR $2,000 I'LL BUILD YOU ONE THAT GOES TO 12.

http://xkcd.com/670/

Wow, that's less than $200 per … uh …
That's a good deal!

How to Create a Domain Model

Bounded by the current requirements

1. Find the conceptual classes
2. Draw them as classes in a UML class diagram
3. Add associations and attributes (but not operations)

Q7

Strategies to Find Conceptual Classes

- Reuse or modify existing models
- Identify noun phrases; linguistic analysis
- Use a category list

Q8

Category Lists for Conceptual Classes

Conceptual Class Category	POS Examples
Business transactions *Here's where the $ is!*	Sale, Payment
Physical objects *Important for control systems, simulations*	Item, Register
Containers of things	Store, Aisle, Bin
...	...

Some Guidelines

Modeling the Unreal World

- Some domains are inherently abstract
  - Telecommunications
  - Server Management
  - Log File Analysis
- Guideline: **listen** carefully **to** the vocabulary and concepts used by **the domain experts**

Common Mistake

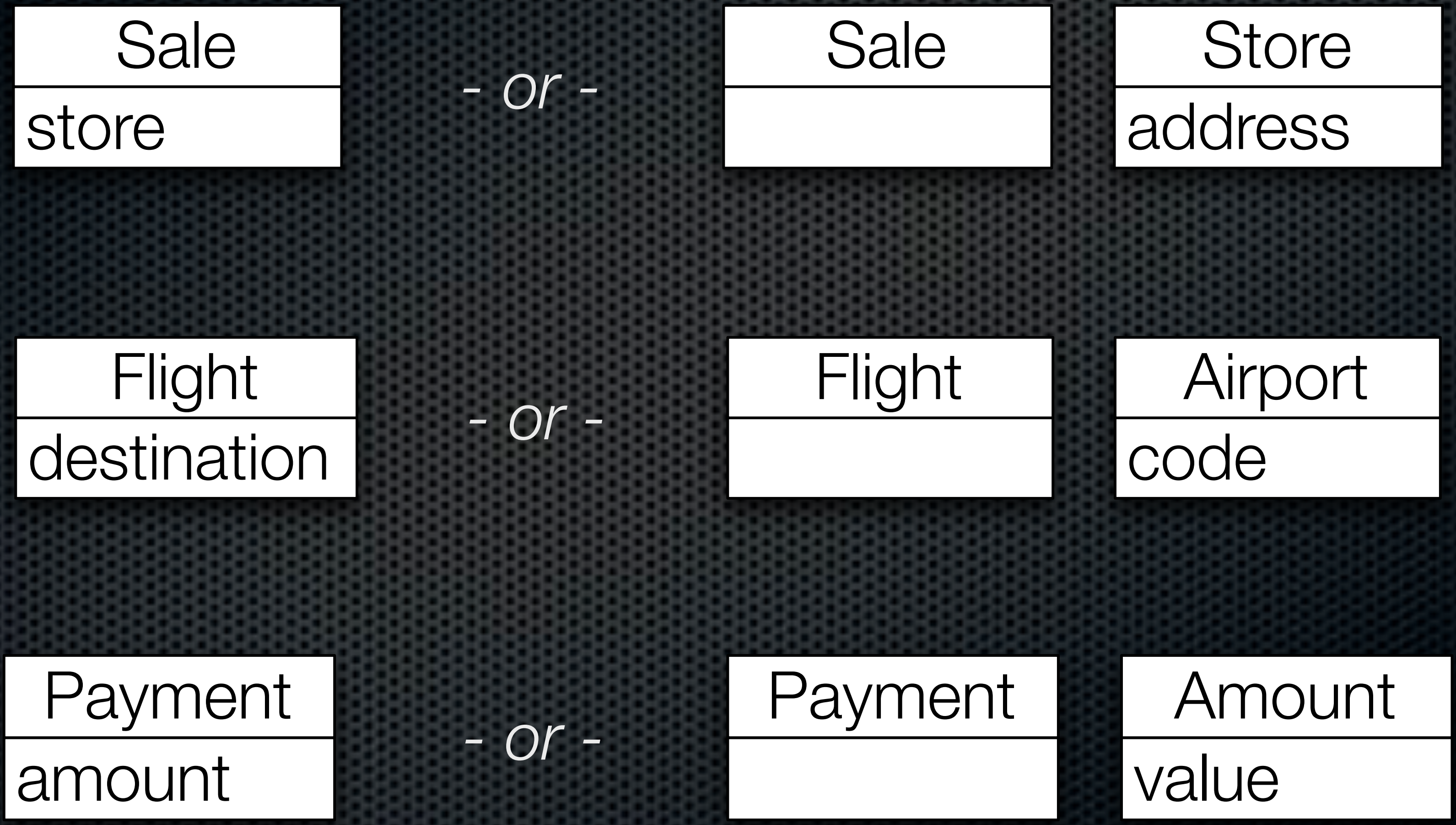
Your programmer's intuition helps here

int, double

String

- Sending an attribute to do a conceptual classes job
- Guideline: if some "attribute" isn't a **number** or **text** in the real world, then it probably should be a conceptual class not an attribute
- Examples...

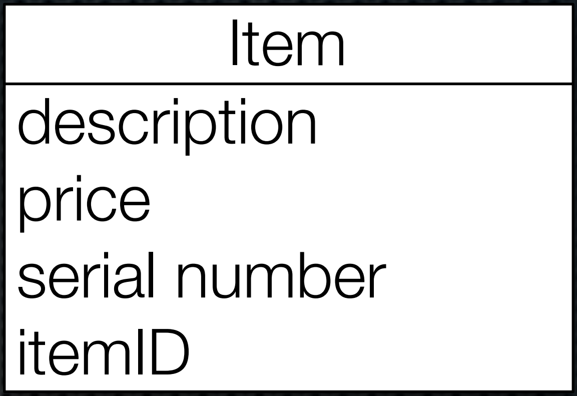
Attribute or Class?



Description Classes

- A **description class** contains information that describes something else, e.g., *ProductDescription*
- Example…

Consider…

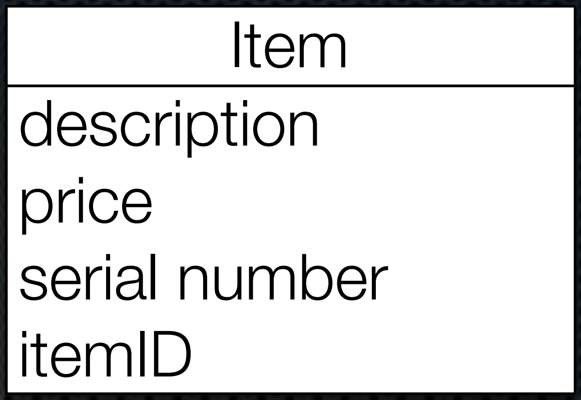


- Assume an *Item* instance represents a physical item in a store
- Item data only recorded within *Item* instances
- When a real-world item is solid, we remove the software *Item* from a collection and it’s garbage collected

Amps that go to 11 are sold out!

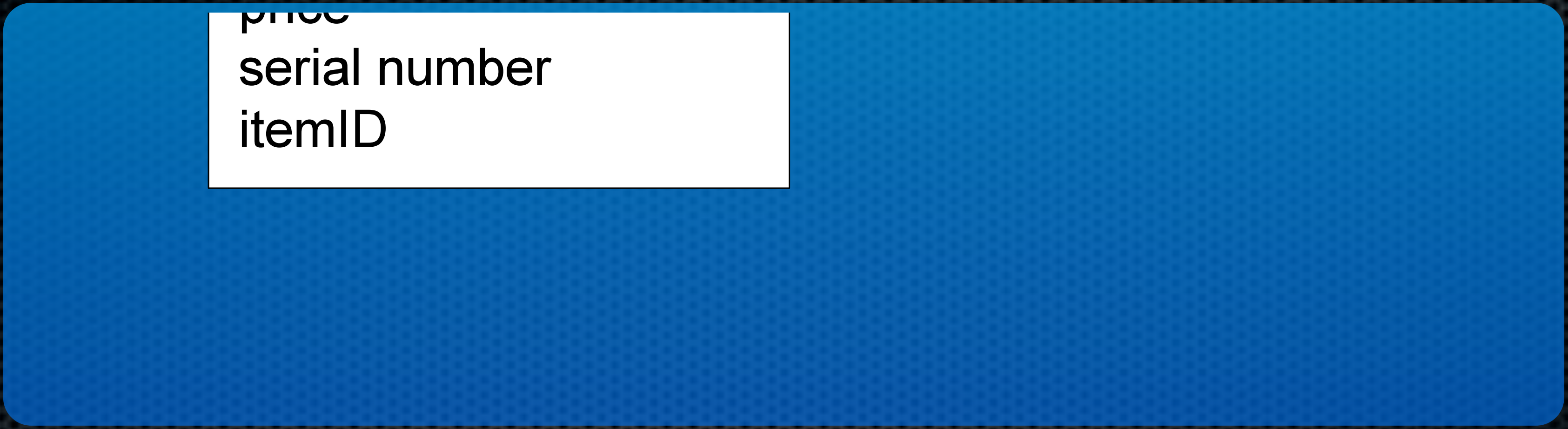
How much for an Amp that goes to 11?

Problems



- Lose memory of the price, etc., if no *Item* instances remain in the system
- Duplicate data
  - Wasted space
  - Error-prone

Solution: Use Description Class



- When information must be retained independent of existence of instances of the described item
- When deleting the described item could result in info. loss
- When it reduces redundant information