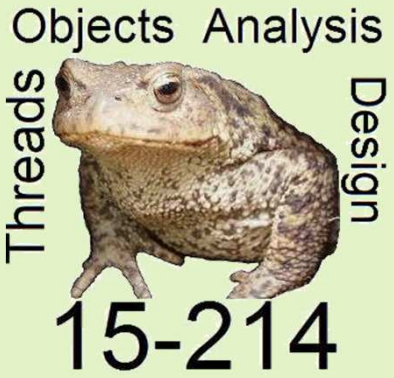


Principles of Software Construction: Objects, Design and Concurrency

Conceptual Modeling in Design

15-214

toad

Fall 2013

Jonathan Aldrich Charlie Garrod

School of Computer Science

institute for SOFTWARE RESEARCH

With slides from Klaus Ostermann

The four course themes

- Threads and Concurrency
  - Concurrency is a crucial system abstraction
  - E.g., background computing while responding to users
  - Concurrency is necessary for performance
  - Multicore processors and distributed computing
  - Our focus: application-level concurrency
  - Cf. functional parallelism (150, 210) and systems concurrency (213)
- Object-oriented programming
  - For flexible designs and reusable code
  - A primary paradigm in industry – basis for modern frameworks
  - Focus on Java – used in industry, some upper-division courses
- Analysis and Modeling
  - Practical specification techniques and verification tools
  - Address challenges of threading, correct library usage, etc.
- Design
  - Proposing and evaluating alternatives
  - Modularity, information hiding, and planning for change
  - Patterns: well-known solutions to design problems

Learning Goals

- What is software design?
- Conceptual Modeling
  - able to model a domain and their relationships
- Design Goals (Modularity, …)
  - able to critique designs, discuss tradeoffs
- Design Considerations with GRASP Principles
  - justify designs
  - toolkit to perform design decisions

Goal of Software Design

- For each desired program behavior there are infinitely many programs that have this behavior
  - What are the differences between the variants?
  - Which variant should we choose?
- Since we usually have to synthesize rather than choose the solution…
  - How can we design a variant that has the desired properties?

Tradeoffs

```
void sort(int[] list, String order) {
   …
  boolean mustswap;
  if (order.equals("up")) {
    mustswap = list[i] < list[j];
  } else if (order.equals("down")) {
    mustswap = list[i] > list[j];
  }
  …
}
```

```
void sort(int[] list, Comparator cmp) {
   …
  boolean mustswap;
  mustswap = cmp.compare(list[i], list[j]);
  …
}
interface Comparator {
  boolean compare(int i, int j);
}
class UpComparator implements Comparator {
  boolean compare(int I, int j) { return i<j; }}

class DownComparator implements Comparator {
  boolean compare(int I, int j) { return i>j; }}
```

Quality of a Software Design

- How can we measure the internal quality of a software design?
  - Extensibility, Maintainability, Understandability, Readability, …
  - Robustness to change
  - Low Coupling & High Cohesion
  - Reusability
  - Testability
  - => **modularity**
- …as opposed to external quality
  - Correctness: Valid implementation of requirements
  - Ease of Use
  - Resource consumption
  - Legal issues, political issues, …

"**Software engineering** is the branch of computer science that creates practical, cost-effective solutions to computing and information processing problems, preferentially by applying scientific knowledge, developing software systems in the service of mankind.
Software engineering entails making **decisions** under constraints of limited time, knowledge, and resources. […]

> Engineering quality resides in engineering judgment. […]
>
> Quality of the software product depends on the engineer's faithfulness to the engineered artifact. […]
>
> Engineering requires reconciling conflicting constraints. […]
>
> Engineering skills improve as a result of careful systematic reflection on experience. […]
>
> Costs and time constraints matter, not just capability. […]

Software Engineering for the 21st Century: A basis for rethinking the curriculum Manifesto, CMU-ISRI-05-108

Conceptual Design

Conceptual Modeling / Domain Models

- Find the concepts in the problem domain
  - Real-world abstractions, not necessarily software objects
- Establish a common vocabulary
- Common documentation, big pictutre
- For communication!
- Often using UML class diagrams as (informal) notation
- Starting point for finding classes

Running Example

Running Example

- **Point of sale** (**POS**) or **checkout** is the place where a retail transaction is completed. It is the point at which a customer makes a payment to a merchant in exchange for goods or services. At the point of sale the merchant would use any of a range of possible methods to calculate the amount owing - such as a manual system, weighing machines, scanners or an electronic cash register. The merchant will usually provide hardware and options for use by the customer to make payment. The merchant will also normally issue a receipt for the transaction.
- For small and medium-sized retailers, the POS will be customized by retail industry as different industries have different needs. For example, a grocery or candy store will need a scale at the point of sale, while bars and restaurants will need to customize the item sold when a customer has a special meal or drink request. The modern point of sale will also include advanced functionalities to cater to different verticals, such as inventory, CRM, financials, warehousing, and so on, all built into the POS software. Prior to the modern POS, all of these functions were done independently and required the manual re-keying of information, which resulted in a lot of errors.

http://en.wikipedia.org/wiki/Point_of_sale

Read description carefully, look for nouns and verbs

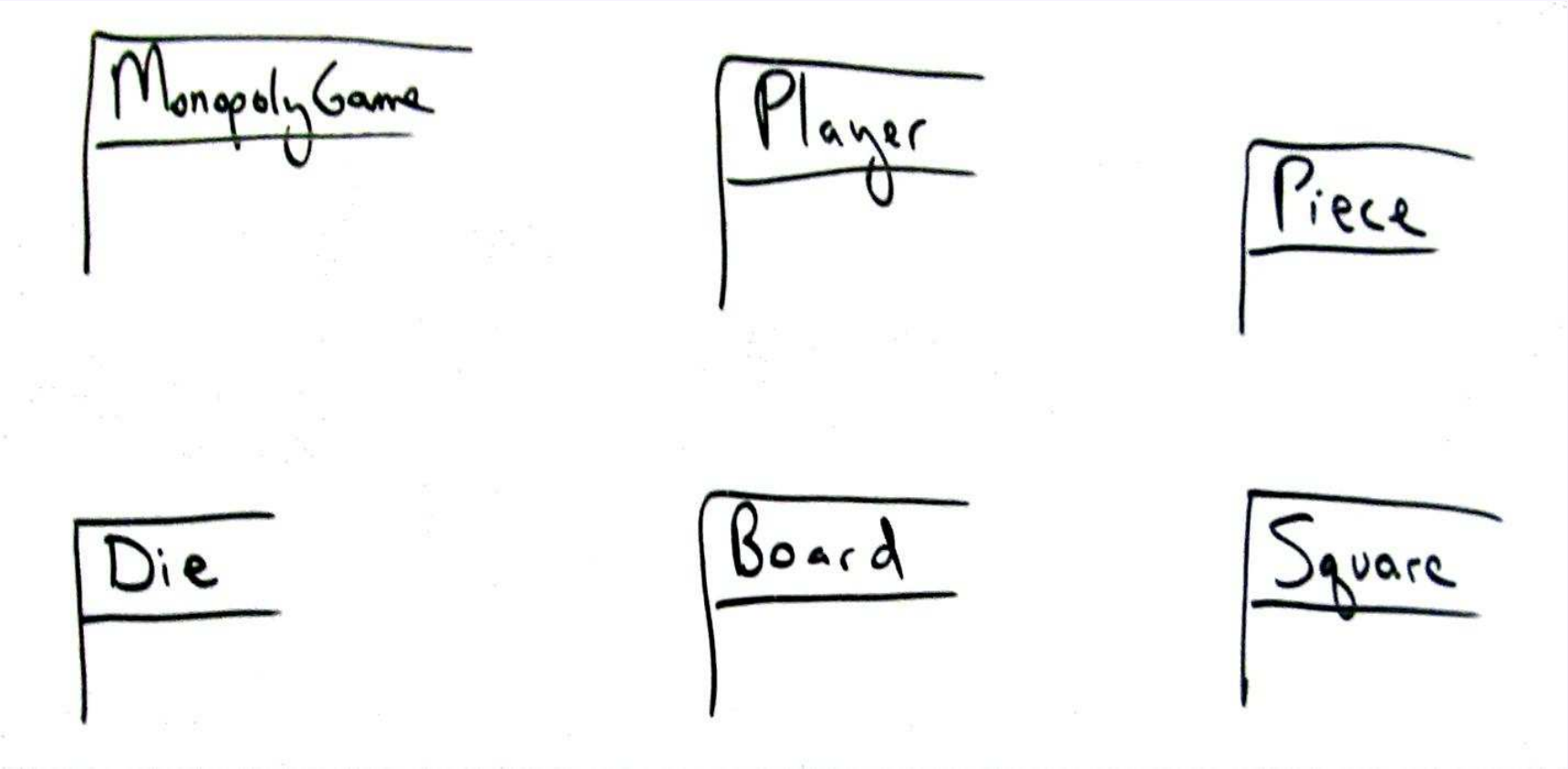
- **Point of sale** (**POS**) or **checkout** is the place where a retail transaction is *completed*. It is the point at which a customer makes a payment to a merchant in *exchange* for goods or services. At the point of sale the merchant would use any of a range of possible methods to *calculate* the amount owing - such as a manual system, weighing machines, scanners or an electronic cash register. The merchant will usually provide hardware and options for use by the customer to *make* payment. The merchant will also normally *issue* a receipt for the transaction.
- For small and medium-sized retailers, the POS will be customized by retail industry as different industries have different needs. For example, a grocery or candy store will need a scale at the point of sale, while bars and restaurants will need to *customize* the item sold when a customer has a special meal or drink request. The modern point of sale will also include advanced functionalities to cater to different verticals, such as inventory, CRM, financials, warehousing, and so on, all built into the POS software. Prior to the modern POS, all of these functions were done independently and required the manual re-keying of information, which resulted in a lot of errors.

http://en.wikipedia.org/wiki/Point_of_sale

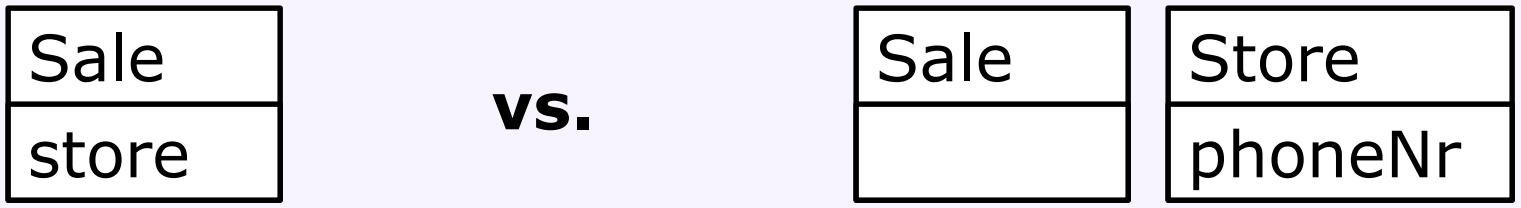
First Steps toward a Domain Model

Register	Item	Store	Sale
Sales LineItem	Cashier	Customer	Ledger
Cash Payment	Product Catalog	Product Description	

Domain Model

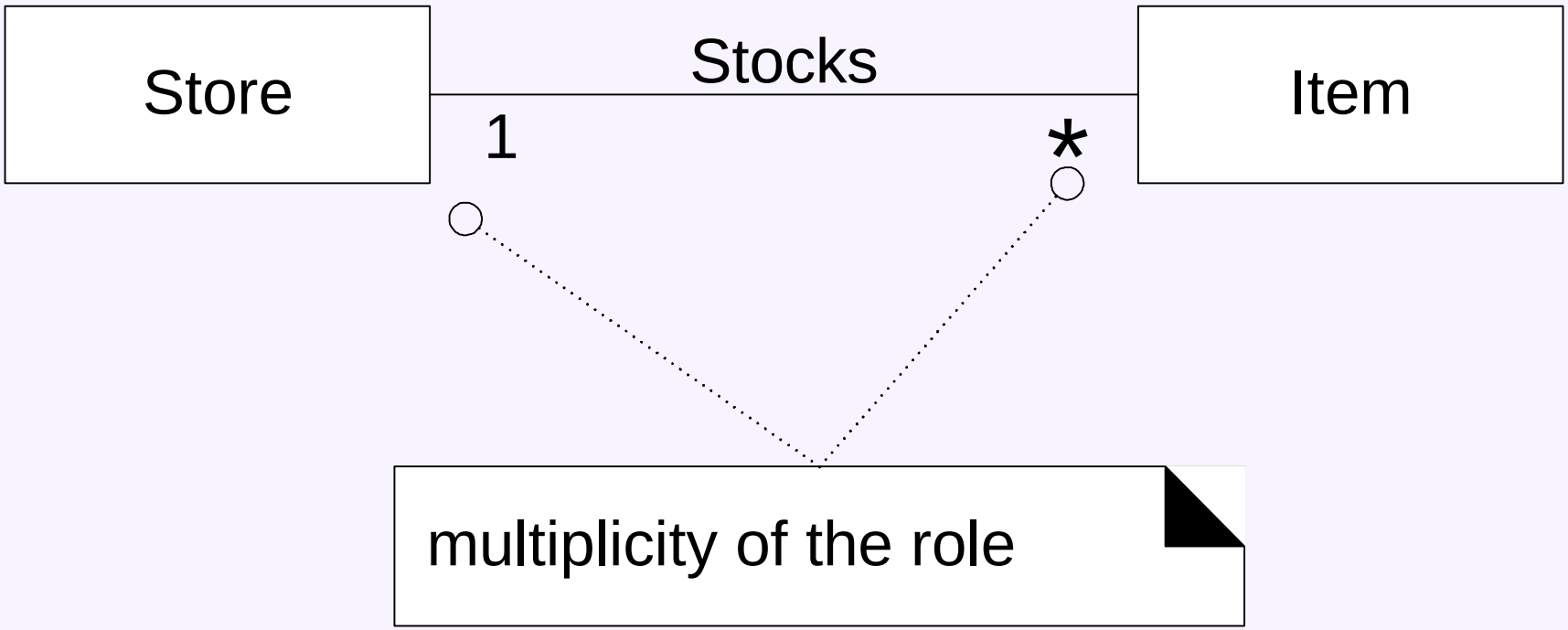


Classes vs. Attributes



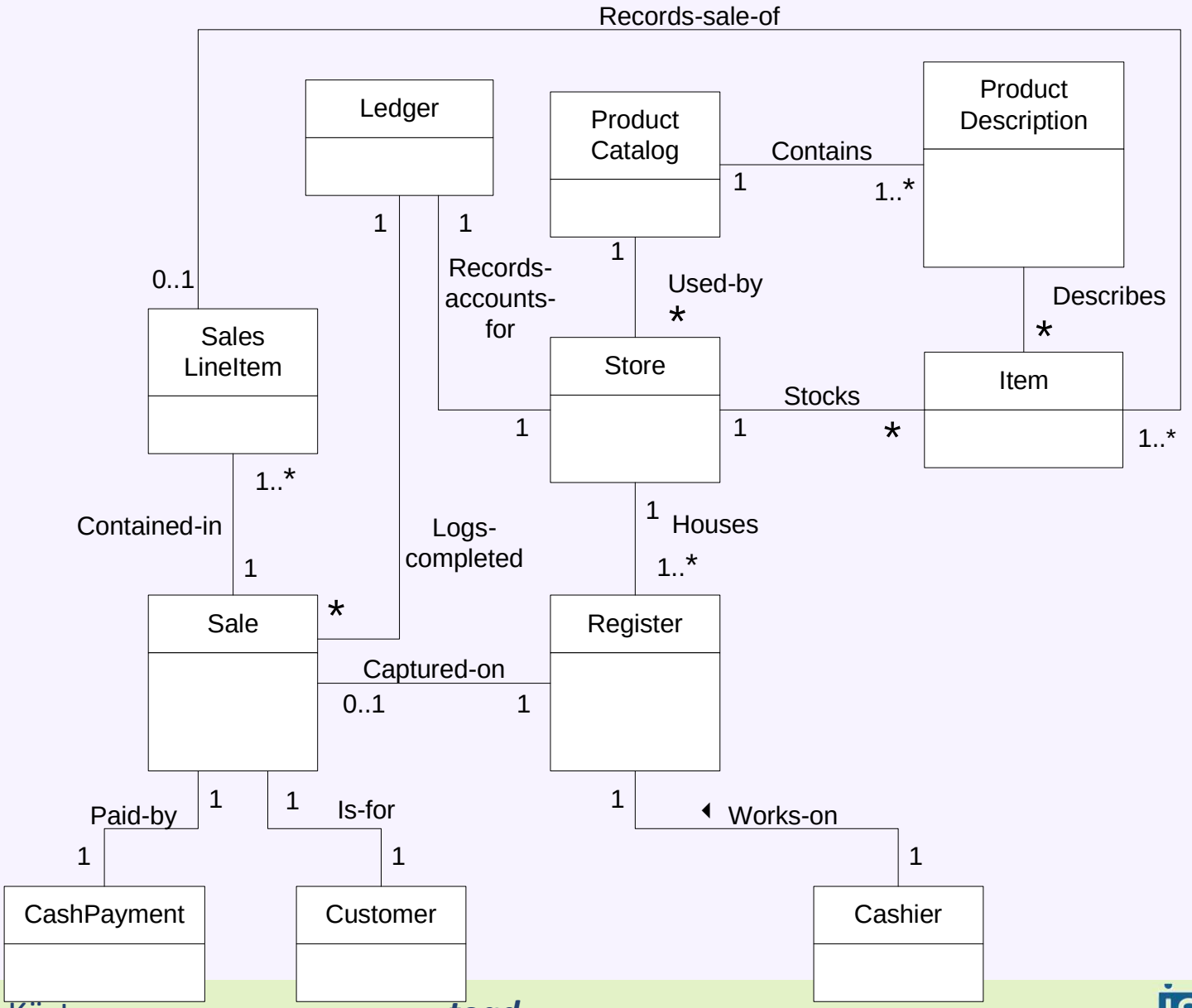
- "If we do not think of some conceptual class X as text or a number in the real world, it's probably a conceptual class, not an attribute"

Associations



- When do we care about a relationship between two objects? (in the real world)

Domain Model (excerpt)



Note

- Focus on concepts, not software classes, not data
  - ideas, things, objects
  - Give it a name, define it and give examples (symbol, intension, extension)
  - Add glossary
  - Some might be implemented as classes, other might not
- There are many choices
- Agree on a single vocabulary
- The model will never be perfectly **correct**
  - that's okay
  - start with a partial model, model what's needed
  - extend with additional information later
  - communicate changes clearly
  - otherwise danger of "analysis paralysis"

Three perspectives of class diagrams

- Conceptual: Draw a diagram that represents the concepts in the domain under study
  - Little or no regard for the software that might implement it
- Specification: Describing the interfaces of the software, not the implementation
  - Often confused in OO since classes combine both interfaces and implementation
- Implementation: Diagram describes actual implementation classes

Understanding the intended perspective is crucial to drawing and reading class diagrams, even though the lines between them are not sharp