Principles of Software Construction: Objects, Design, and Concurrency

Design: From Systems to Objects

toad

Fall 2014

Jonathan Aldrich Charlie Garrod

School of Computer Science

institute for SOFTWARE RESEARCH

Administrivia

- Homework 1 due at 11:59pm tonight
- Homework 2 coming tomorrow

Key concepts from Tuesday

Review: Inheritance Self-Evaluation Questions

1. Concept: what is the purpose of inheritance?
2. Practice: what does the following code print?

```
class Foo {
   public void frob() {
      System.out.print('A');
   }
   public void baz() {
      System.out.print('E');
      frob();
      System.out.print('S');
   }
}
```

```
public class Bar extends Foo {
   public void frob() {
      System.out.print('U');
   }
   public void baz() {
      System.out.print('R');
      super.baz();
      System.out.print('E');
   }
   public static void main(String args[]) {
      Foo foo = new Bar();
      foo.baz();
   }
}
```

Requirements and Design Overview

- Requirements Engineering
  - Requirements Elicitation *(see 15-313)*
    - Functional Requirements (often as *Use Cases*)
    - Quality Attributes (often as *Quality Attribute Scenarios*)
  - (Object-Oriented) Requirements Analysis
    - Domain Modeling
  - System Specification
    - System Sequence Diagrams
    - Behavioral Contracts
- (Object-Oriented) Software Design
  - Architectural Design *(mostly in 15-313)*
  - Responsibility Assignment
    - Object sequence diagrams
    - Object model (class diagrams)
  - Method specifications / code contracts

Today's Lecture: Learning Goals

- Know the steps involved in specifying a software system and refining that system specification into an object design
- Describe the dynamics of a use case with a System Sequence Diagram
- Specify system operations with behavioral contracts
- Derive interaction diagrams and an object model by assigning responsibilities

Requirements and Design Overview

- Requirements Engineering
  - Requirements Elicitation *(see 15-313)*
    - Functional Requirements (often as *Use Cases*)
    - Quality Attributes (often as *Quality Attribute Scenarios*)
  - (Object-Oriented) Requirements Analysis
    - Domain Modeling
  - System Specification
    - System Sequence Diagrams
    - Behavioral Contracts
- (Object-Oriented) Software Design
  - Architectural Design *(mostly in 15-313)*
  - Responsibility Assignment
    - Object sequence diagrams
    - Object model (class diagrams)
  - Method specifications / code contracts

System Specification

Goal: understand (and specify) what the system should do

- System Sequence Diagram
  - What are the operations on the system, performed by the user?
    - Examples: "run the program with input I", "Select menu command C"
  - In what order do they occur?
    - Must compile a file before running it

- Behavioral Contracts
  - Under what conditions can each interaction take place?
    - A file must typecheck correctly before you can compile it
  - What is the result of the interaction?
    - a .class file is created
    - The .class file is named after the class in the source code
    - …

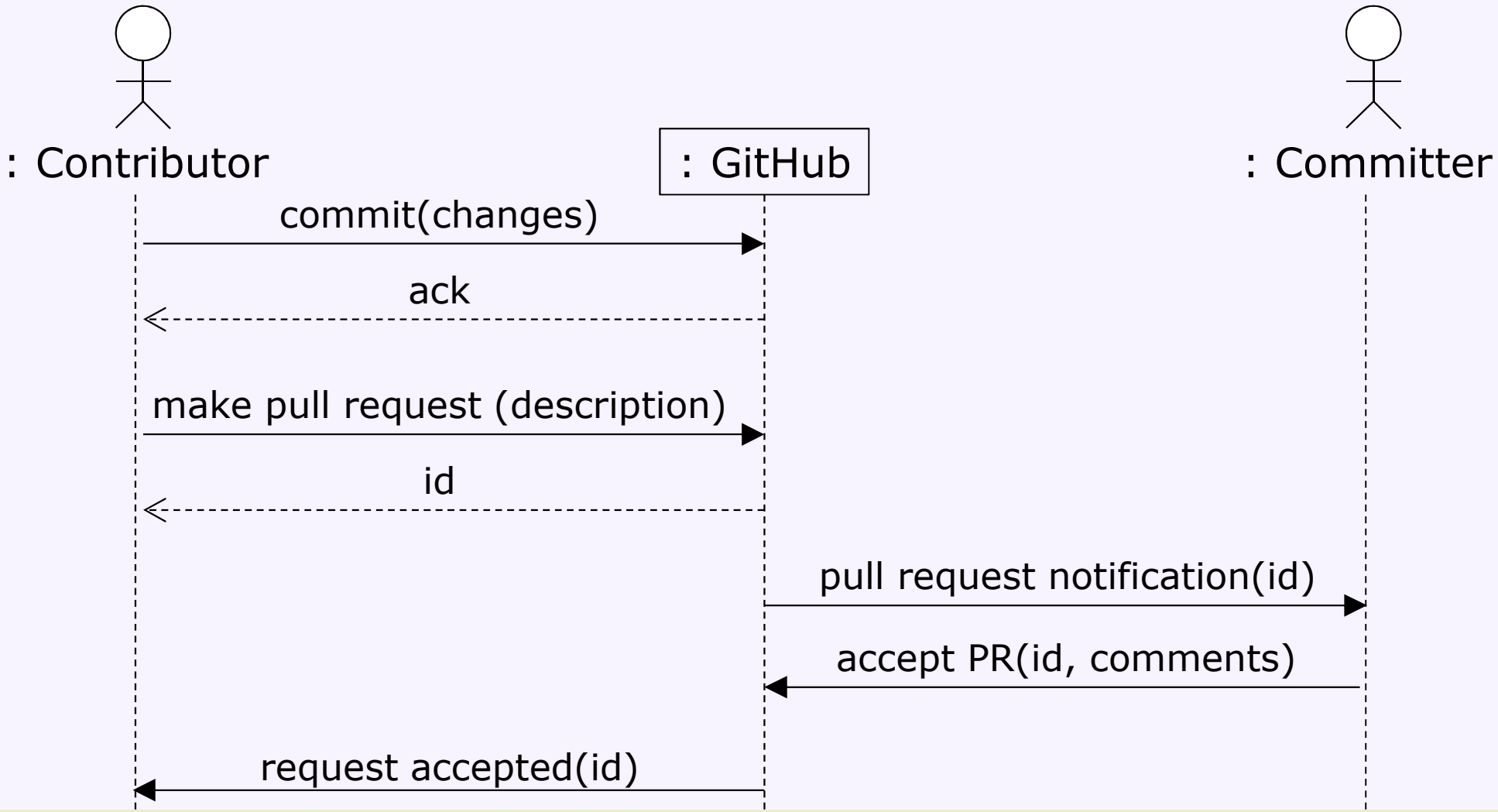
System Sequence Diagrams

- A **System Sequence Diagram** is a picture that shows, for one scenario of use, who interacts with the system, and the sequence of events that occur on the system's boundary
- e.g. use case: A commit and pull request using GitHub (aside: distributed VCS for large projects)

  1. Contributor modifies source code, commits changes to her own fork of the GitHub repository.
  2. GitHub applies commit to the fork's history and acknowledges the commit.
  3. Contributor initiates a pull request, describing the changes she has committed.
  4. GitHub acknowledges the pull request and returns an ID for the pull request, and notifies the Committer of the pull request.
  5. Committer inspects Contributor's work and accepts the pull request into the main repository, notifying GitHub of the accepted pull request and any comments associated with the acceptance.
  6. GitHub notifies the Contributor that her work was accepted.

System Sequence Diagrams

- A **System Sequence Diagram** is a picture that shows, for one scenario of use, who interacts with the system, and the sequence of events that occur on the system's boundary

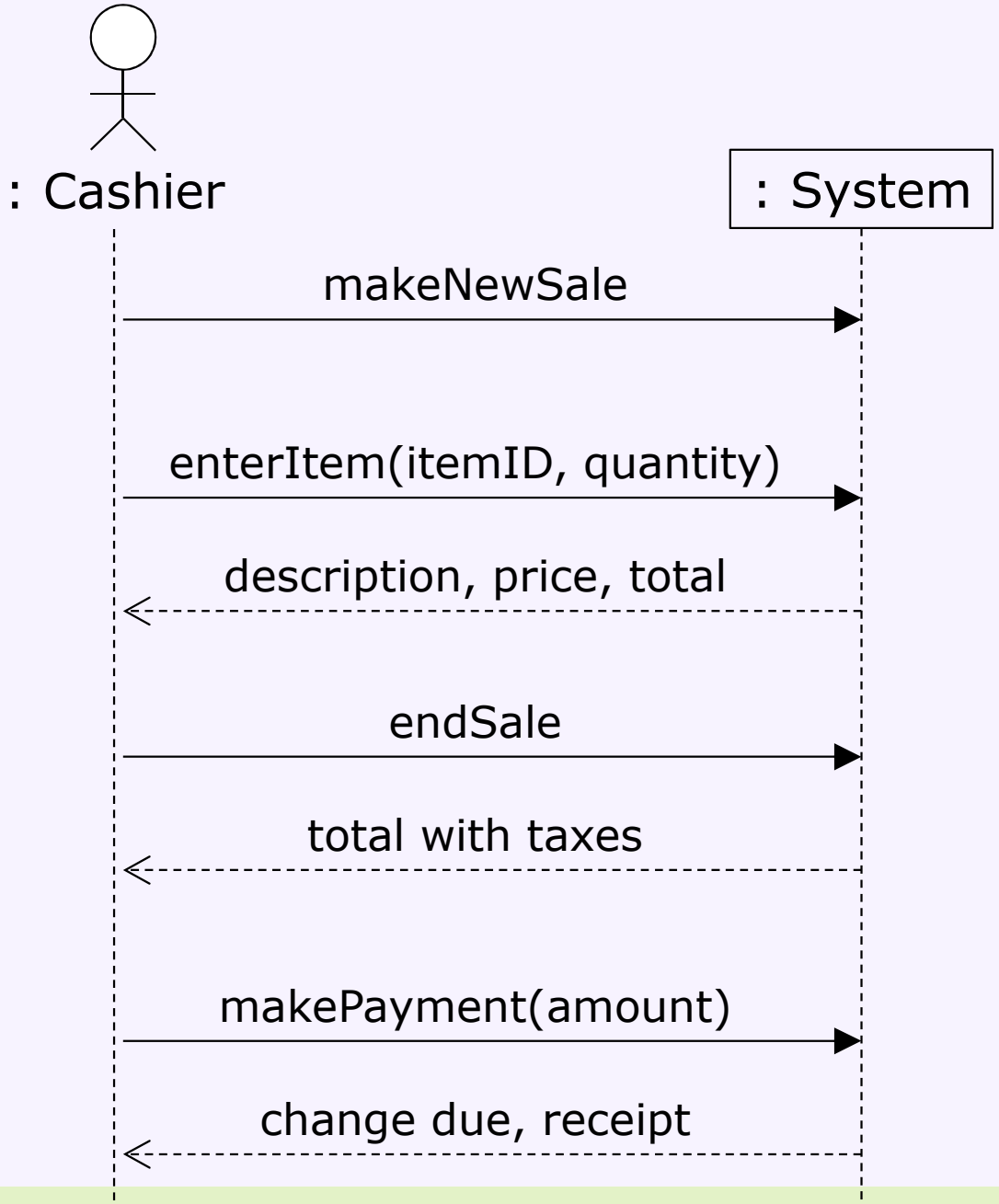


Sequence Diagram for the Point of Sale Example

- Exercise (paper): write a System Sequence Diagram for the Point of Sale system

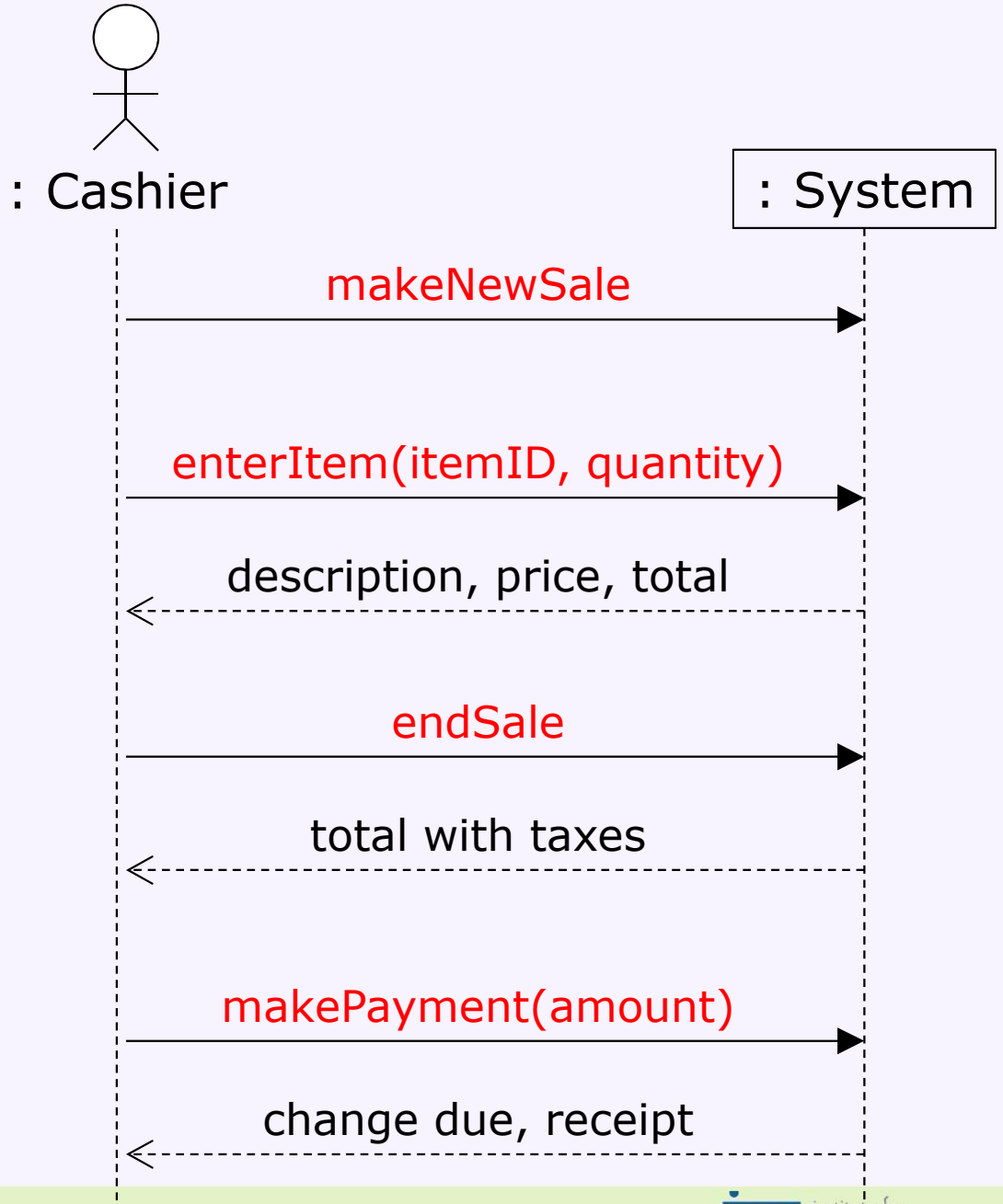
- Use Case: Successful Customer Checkout
  1. Customer arrives at POS checkout with goods to purchase
  2. Cashier starts a new sale
  3. Cashier enters item identifier and quantity
  4. System records sale line item and presents item description, price, and running total
  5. Cashier repeats steps 3-4 until all goods have been entered
  6. System presents total with taxes calculated
  7. Cashier tells customer the total, and asks for payment
  8. Customer pays and System provides change and a receipt

Sequence Diagram for the Point of Sale Example



Behavioral Contracts: What do These Operations Do?

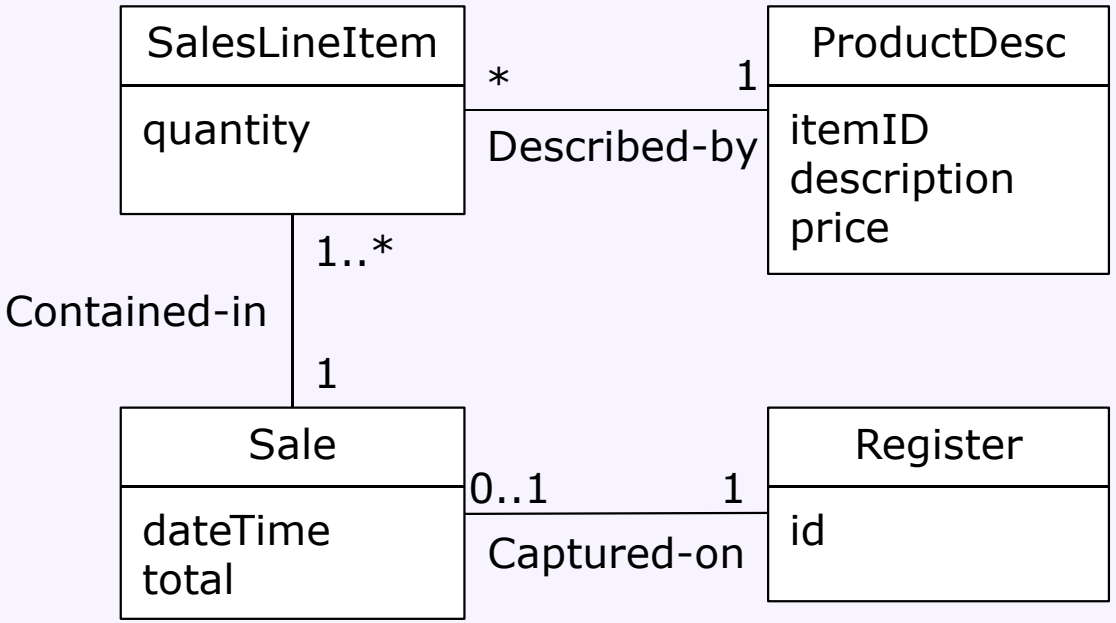
- To design, we need a spec
  - Preconditions
  - Postconditions
- We can write a *behavioral contract*
  - Like a pre-/post-condition specification for code
  - Often written in natural language
  - Focused on system interfaces
    - may or may not be methods



Example Point of Sale Contract

Operation: makeNewSale()
Preconditions: There is not currently a sale in progress

Postconditions: - A Sale instance s was created
- s was associated with a Register



A Point of Sale Contract

- Contract structure
  - Operation name, parameters
  - Requirement or use case this is a part of *(discussed in 15-313)*
  - Preconditions
  - Postconditions

Operation:	makeNewSale()
Preconditions:	There is not currently a sale in progress
Postconditions:	- A Sale instance s was created - s was associated with a Register

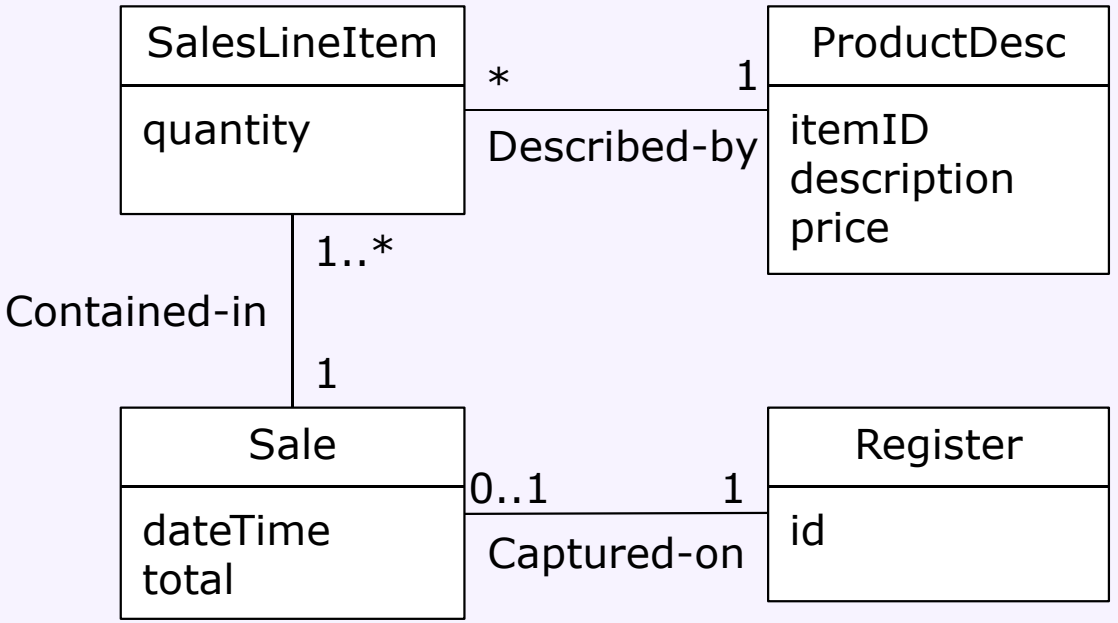
- Which contracts to write?
  - Operations that are complex or subtle
  - Operations that not everyone understands
  - Simple/obvious operations are often not given contracts in practice

- Writing postconditions
  - Written in past tense (a *post*-condition)
  - Describe changes to domain model
    - Instance creation and deletion
    - Attribute modification
    - Associations formed and broken
      - Easy to forget associations when creating objects!

Exercise (paper): Write a Point of Sale Contract

Operation: enterItem(itemID : ItemID, quantity : integer)
Preconditions:

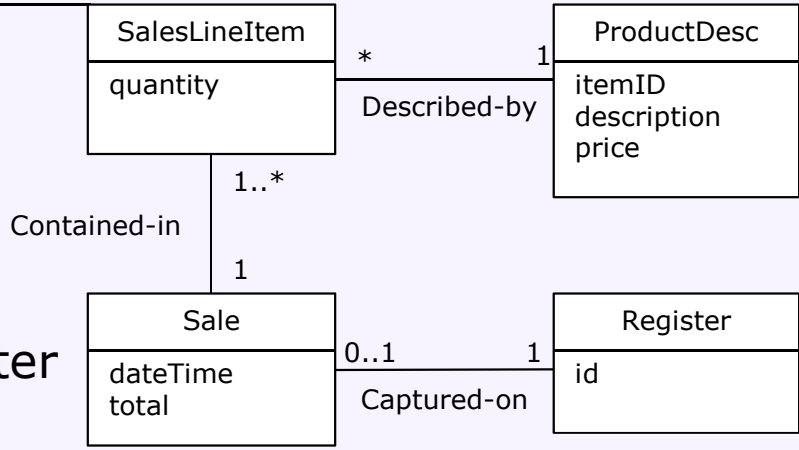
Postconditions:



Example Point of Sale Contracts

Operation: makeNewSale()
Preconditions: There is not currently a sale in progress

Postconditions:
- A Sale instance s was created
- s was associated with a Register



Operation: enterItem(itemID : ItemID, quantity : integer)
Preconditions: There is a sale s in progress

Postconditions:
- A SalesLineItem instance sli was created
- sli was associated with the sale s
- sli.quantity became quantity
- sli was associated with a ProjectDescription, based on itemID match

Requirements and Design Overview

- Requirements Engineering
  - Requirements Elicitation *(see 15-313)*
    - Functional Requirements (often as *Use Cases*)
    - Quality Attributes (often as *Quality Attribute Scenarios*)
  - (Object-Oriented) Requirements Analysis
    - Domain Modeling
  - System Specification
    - System Sequence Diagrams
    - Behavioral Contracts
- (Object-Oriented) Software Design
  - Architectural Design *(mostly in 15-313)*
  - Responsibility Assignment
    - Object sequence diagrams
    - Object model (class diagrams)
  - Method specifications / code contracts

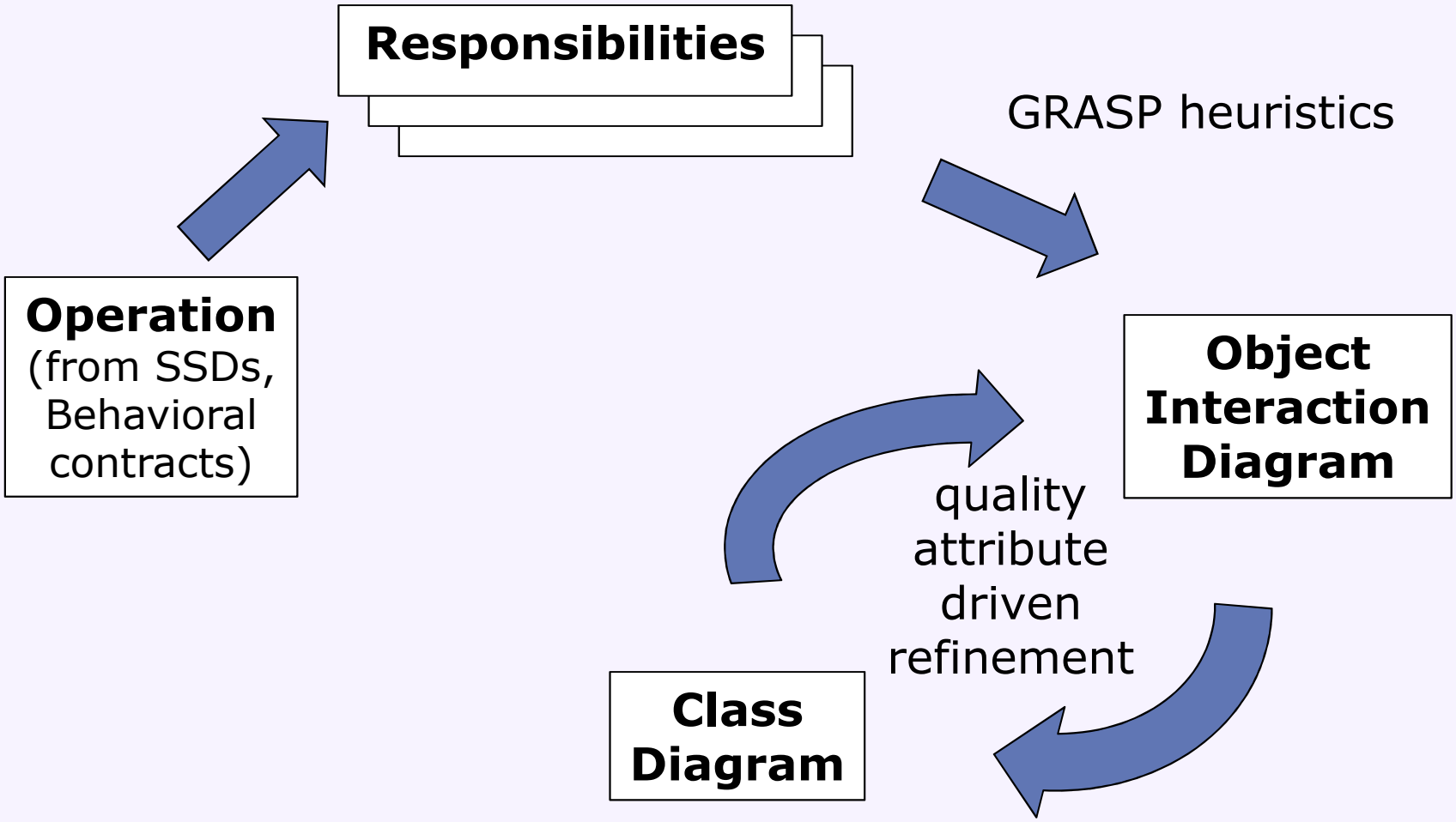
Responsibilities

- Responsibilities are *obligations that an object has to fulfill*
- Two types of responsibilities:
  - knowing
  - doing
- Doing responsibilities of an object include:
  - doing something itself, such as creating an object or doing a calculation
  - initiating action in other objects
  - controlling and coordinating activities in other objects
- Knowing responsibilities of an object include:
  - knowing about private encapsulated data
  - knowing about related objects
  - knowing about things it can derive or calculate

Responsibility Assignment

- Consider the system's operations
  - Found in System Sequence Diagrams and Behavioral Contracts
- Divide each system operation into responsibilities
  - Knowing, doing, etc.
- Assign each responsibility to an object
  - Knowing
    - Domain-related "knowing" responsibilities already in domain model
    - Implementation-related "knowing" responsibilities will be new
  - Doing
    - Become methods in the implementation
- Allocate object responsibilities to classes
  - Two objects may have the same class → merge responsibilities
  - A class may have a superclass → divide responsibilities
- How to assign
  - Drive by quality attributes
  - General Responsibility Assignment Software Patterns provide heuristics (GRASP patterns for short)

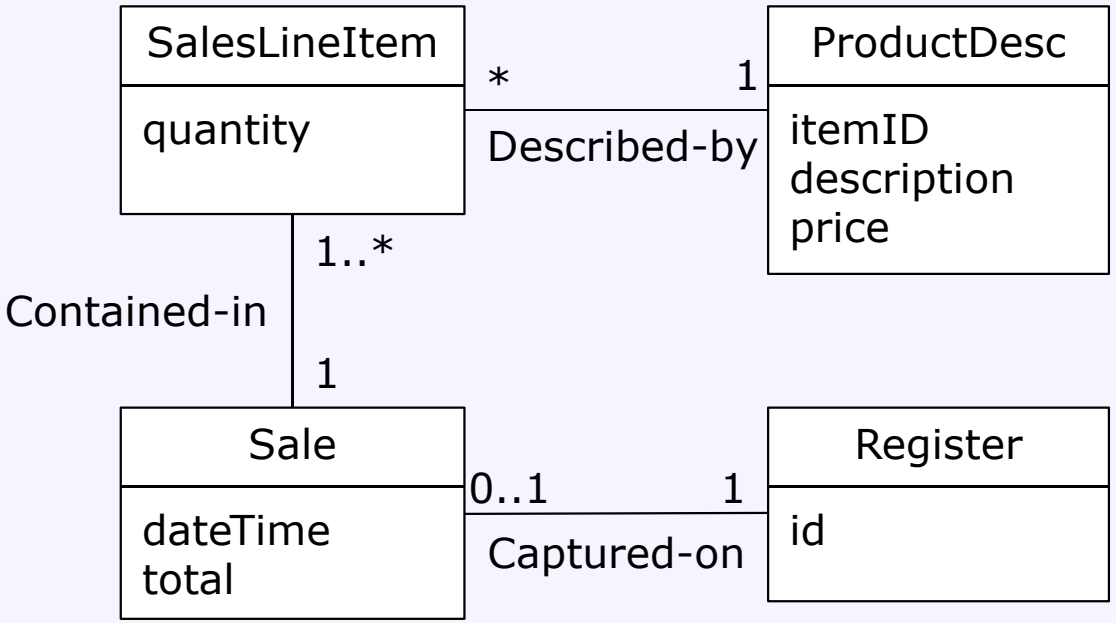
Responsibility Assignment, Graphically



Example Point of Sale Contract

Operation: makeNewSale()
Preconditions: There is not currently a sale in progress

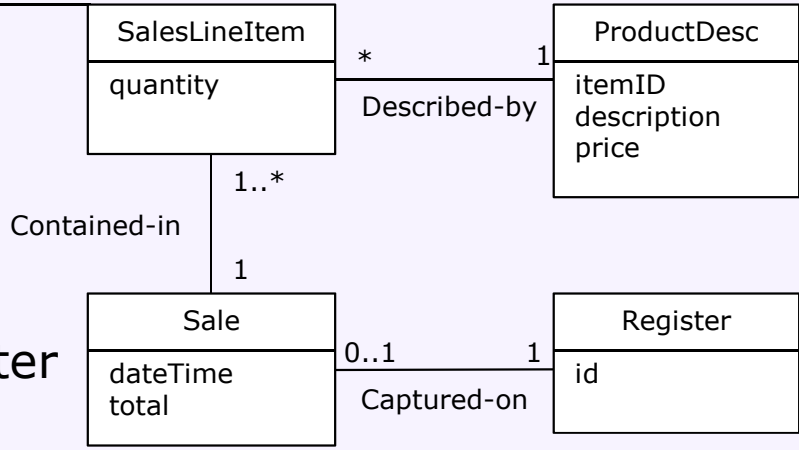
Postconditions: - A Sale instance s was created
- s was associated with a Register



Example Point of Sale Contracts

Operation: makeNewSale()
Preconditions: There is not currently a sale in progress

Postconditions:
- A Sale instance s was created
- s was associated with a Register



Operation: enterItem(itemID : ItemID, quantity : integer)
Preconditions: There is a sale s underway

Postconditions:
- A SalesLineItem instance sli was created
- sli was associated with the sale s
- sli.quantity became quantity
- sli was associated with a ProjectDescription, based on itemID match

What Happens to the Domain Model?

- Methods are added
- New classes, attributes, and associations are discovered
- Attributes are given types
- Associations are given field names and direction
- New inheritance relationships are created for flexibility and reuse
- Methods and fields are divided among superclass and subclasses

Toad's Take-Home Messages

Wait, don't get up yet!

- System specification identifies what the system is to do
  - System sequence diagrams for each scenario (use case)
  - Behavioral contracts for each operation
    - specify pre- and post-conditions in terms of domain model
- Responsibility Assignment produces a concrete design
  - Object interaction diagrams
  - Object model (a class diagram)
- Object model vs. Domain model
  - Methods are added
  - New classes, fields, associations are added for implementations
  - Fields and associations are further specified
  - New inheritance relations are added for reuse
- How do we know how to best assign responsibilities?
  - Coming soon!