Chapter 10

System sequence diagrams

A System Sequence Diagram

system as black box

the name could be "NextGenPOS" but "System" keeps it simple

the ":" and underline imply an instance, and are explained in a later chapter on sequence diagram notation in the UML

external actor to system

Process Sale Scenario

: Cashier

:System

makeNewSale

a UML loop **interaction frame**, with a boolean **guard** expression

loop   [more items]

enterItem(itemID, quantity)

description, total

endSale

a message with parameters

it is an abstraction representing the system event of entering the payment data by some mechanism

return value(s) associated with the previous message

an abstraction that ignores presentation and medium

the return line is optional if nothing is returned

total with taxes
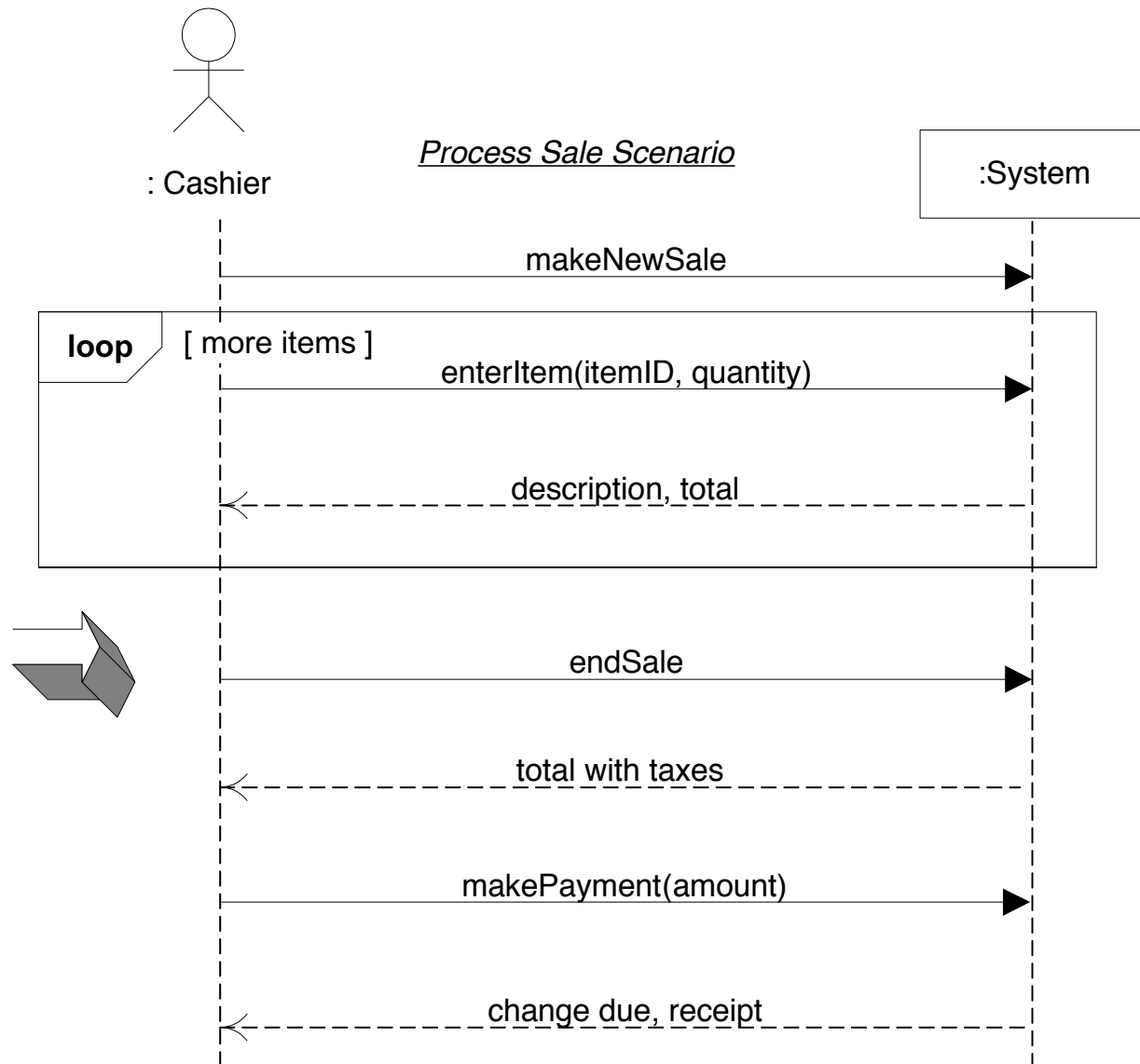
makePayment(amount)

change due, receipt

System Sequence Diagrams

- Simple diagrams that illustrate input and output events to the system
- Serve as input to
  - operation contracts
  - object design

to be covered later

- SSDs are ONE particular use of UML Sequence diagrams
- System treated as black box
  - Interactions with external actors detailed
- Guideline:
  - One SSD per "main success scenario" of a use case
  - Frequent or complex alternative scenarios

SSD for main success scenario of the ProcessSale Use Case

Process Sale Scenario

: Cashier

:System
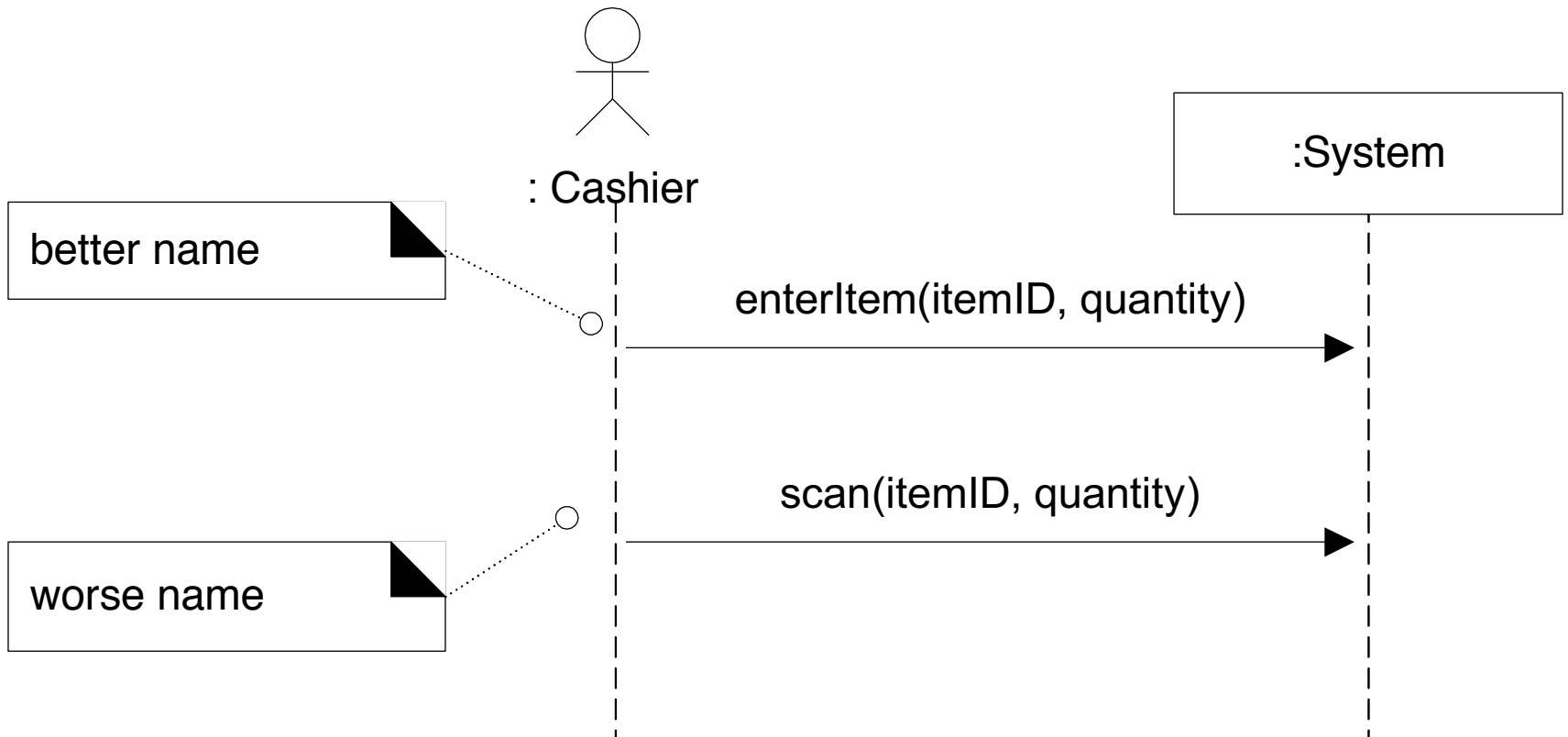
Simple cash-only *Process Sale* scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...

makeNewSale

loop [more items]

enterItem(itemID, quantity)

description, total

endSale

total with taxes

makePayment(amount)
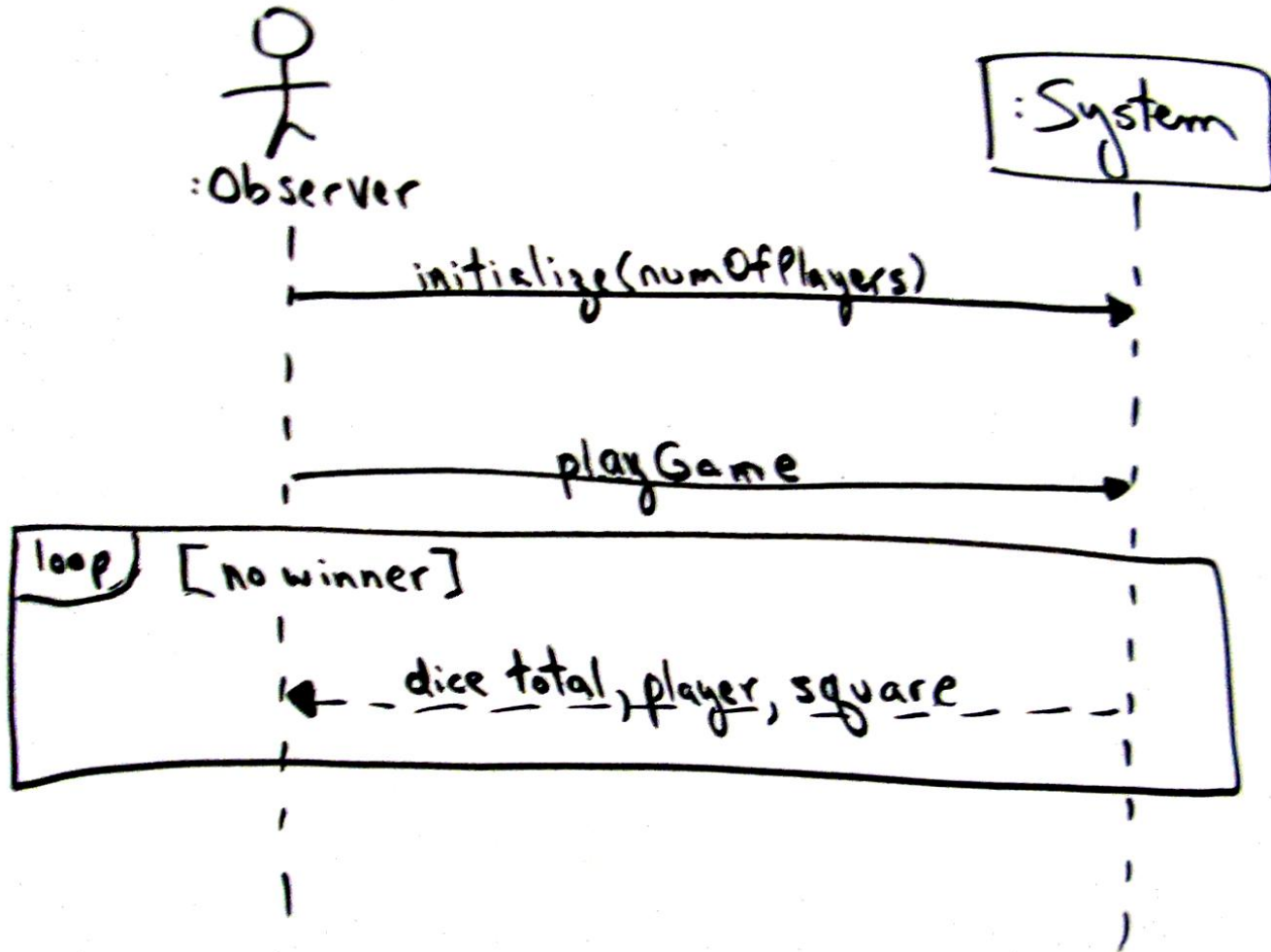
change due, receipt

Why draw an SSD?

- Isn't all of this already in the use case narrative?
  - Input and output events along with their arguments (associated information) are made explicit
- Put detailed explanations of certain event names, arguments in the Glossary
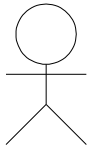
Naming Events

better name

: Cashier

:System

enterItem(itemID, quantity)

scan(itemID, quantity)

worse name

Monopoly SSD example

Chapter 11

Operation Contracts

Process Sale Scenario

: Cashier

:System

makeNewSale()

loop [more items]

enterItem(itemID, quantity)

description, total

endSale()

total with taxes

makePayment(amount)

change due, receipt

these input system events invoke *system operations*

the system event *enterItem* invokes a system operation called enterItem and so forth

this is the same as in object-oriented programming when we say the message *foo* invokes the method (handling operation) *foo*

- Operation contracts: Detailed description of changes to objects in domain model in response to events
  - Use pre- and post- conditions
- What is the difference from a use-case narrative?
  - There the pre- and post- conditions are described for the entire use case
  - Here pre- and post- conditions are described per system event

Example contract

Contract CO2: enterItem

Operation: Cross References:
Preconditions:

enterItem(itemID : ItemID, quantity : integer) Use Cases: Process Sale There is a sale underway.

Postconditions:

- A SalesLineItem instance sli was created (instance creation).
- sli was associated with the current Sale (association formed).
-sli.quantity became quantity (attribute modification).
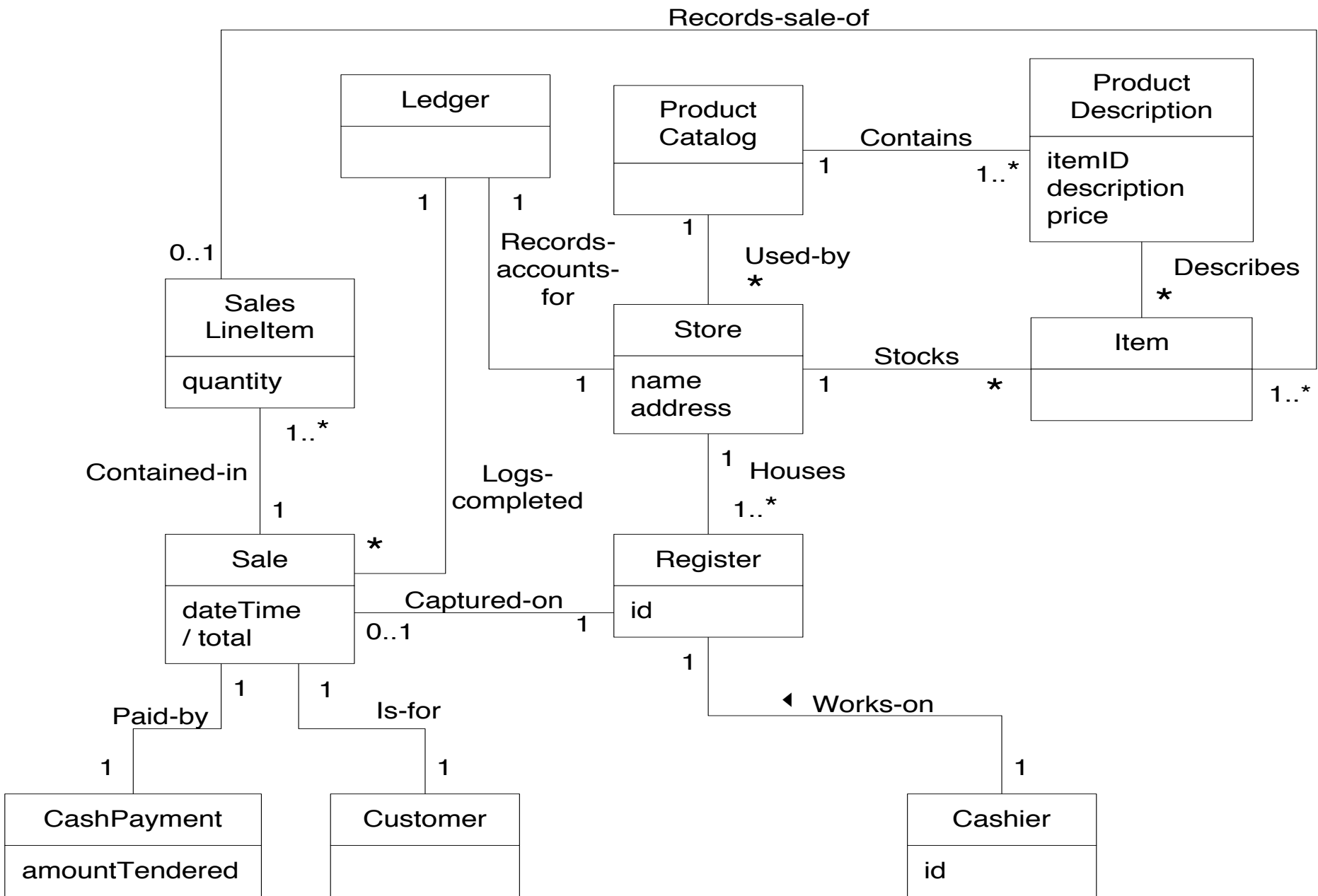- sli was associated with a ProductSpecification, based on itemID match (association formed).

Sections of a contract

Operation: Cross   Name of operation, and parameters

References: (optional) Use cases this operation can occur within

Preconditions: Noteworthy *assumptions* about the state of the system or objects in the Domain Model before execution of the operation. These will not be tested within the logic of this operation, are assumed to be true, and are non-trivial assumptions the reader should know were made.

Postconditions: -The state of objects in the Domain Model after completion of the operation. Discussed in detail in a following section.

Postconditions

- For important, non-trivial system events, describe changes in the state of the objects in the domain model
- Changes include
  - instances created
  - associations (UML links) formed or broken
  - attributes changed
- May result in updates to the domain model
  - Additions of new attributes and associations

POS Domain Model Example:

Records-sale-of

Ledger

Product Catalog —— Contains —— Product Description: itemID, description, price

1 ... 1..*

Records-accounts-for

Used-by *

Describes *

Sales LineItem: quantity

0..1

1..*

Contained-in

Store: name, address

Stocks

Item

1 1 * 1..*

Logs-completed

Houses

1 1..*

Sale: dateTime, / total

* Captured-on

Register: id

1 0..1 1

1

Works-on

Paid-by 1 1 Is-for

CashPayment: amountTendered

Customer

Cashier: id

1 1 1

Example contract

Contract CO4: makePayment

Operation: Cross References: Preconditions:	makePayment(amount: Money) Use Cases: Process Sale There is a sale underway.
Postconditions:	- A Payment instance p was created (instance creation). - p.amountTendered became amount (attribute modification). - p was associated with the current Sale (association formed). - The current Sale was associated with the Store (association formed); (to add it to the historical log of completed sales)