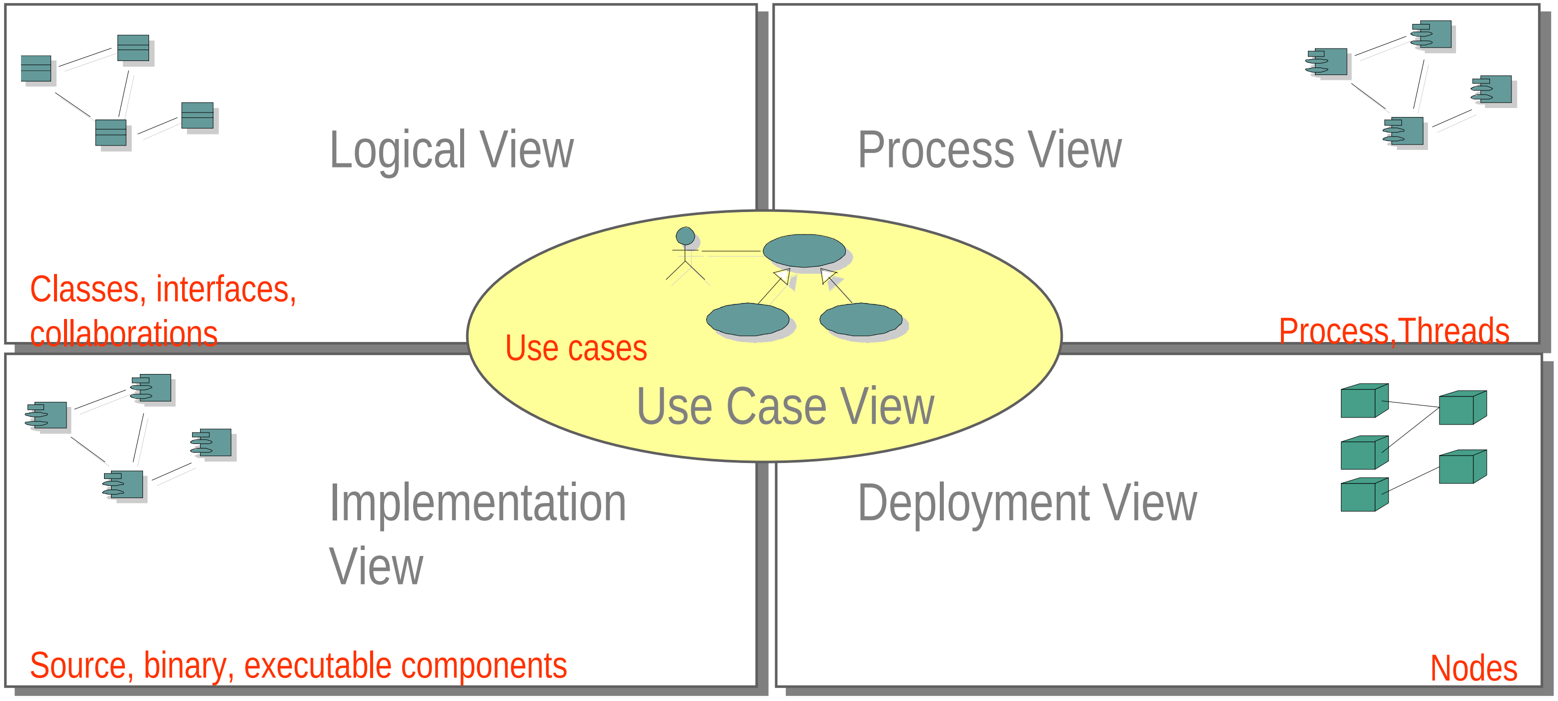
Logical View

1. Overview
2. Static Structures
3. Interactions
4. Dynamic Behavior
5. Example: Logical View for the ATM



1. Overview

-The purpose of the logical view is to ***specify the functional requirements of the system***. The main artifact of the logical view is the design model:

- The ***design model*** gives a concrete description of the functional behavior of the system. It is derived from the analysis model.
    - The ***analysis model*** gives an abstract description of the system behavior based on the use case model.

- In general only the design model is maintained in the logical view, since the analysis model provides a rough sketch, which is later refined into design artifacts.

Design Model

-The design model consists of collaborating classes, organized into subsystems.

-Artifacts involved in the design model may include:

- ***class***, ***interaction***, and ***state*** diagrams
- the ***subsystems and their interfaces***

2. Static Structures

Notion of Class

☞a ***description of a group of objects*** with:

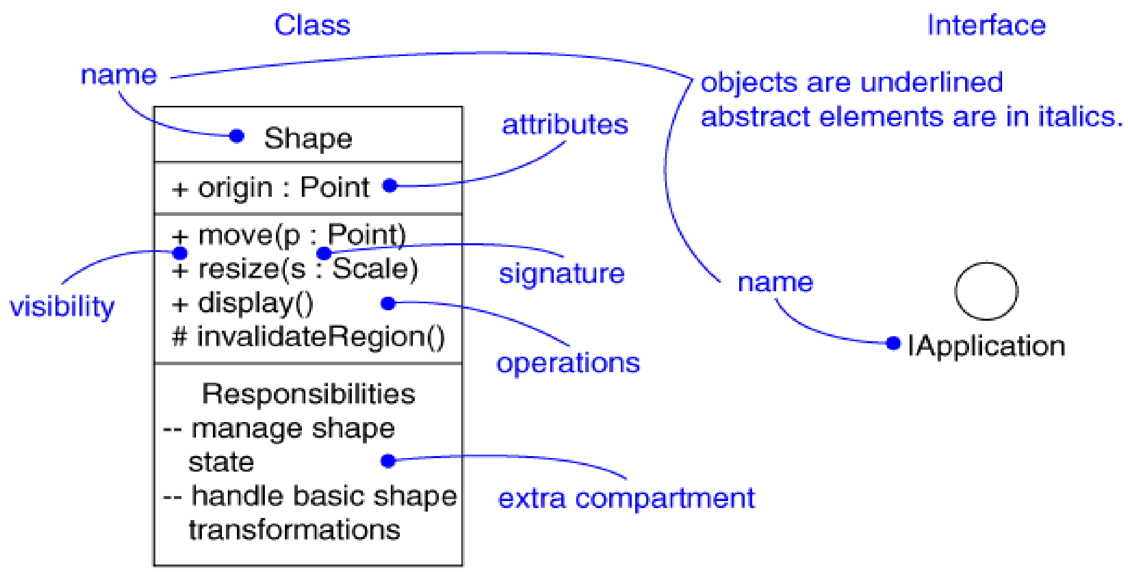
- common properties (***attributes***),
- common behavior (***operations***),
- common ***relationships*** to other objects, and common semantics.

☞ in the UML classes are represented as compartmentalized rectangles:

- top compartment contains the name of the class
- middle compartment contains the structure of the class (attributes)
- bottom compartment contains the behavior of the class (operations)

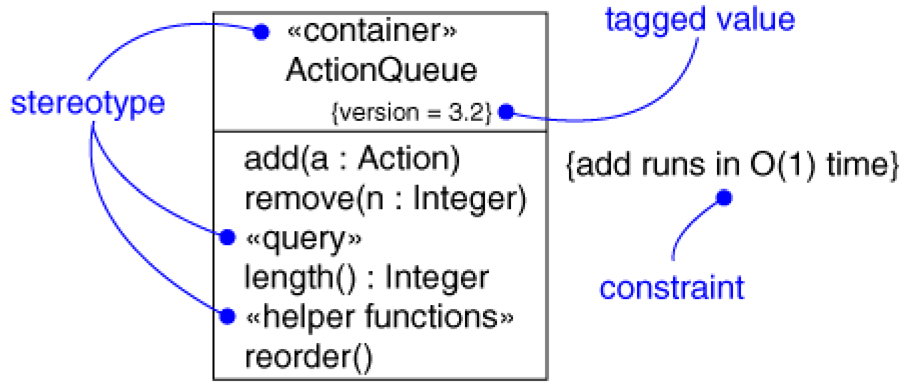
<u>**Visibility:**</u>

+	*public*
#	*protected*
-	*private*



Extensibility Mechanisms

- Stereotype
- Tagged value
- Constraint



Notion of Stereotype

- provides the capability to ***create a new kind of modeling element***.
- we can create new kinds of classes by defining stereotypes for classes.
- the stereotype for a class is shown below the class name enclosed in guillemets (<< >>).
- examples of class stereotypes: ***exception, utility etc.***

Boundary, Entity, and Control Classes

☞The ***Rational Unified Process*** advocates for finding the classes for a system by looking for *boundary*, *control*, and *entity* classes.

Entity classes:

- model information and associated behavior that is ***generally long lived***
- may ***reflect a real-world entity***, or may be needed to perform tasks internal to the system
- are ***application independent***: may be used in more than one application.

Boundary classes:

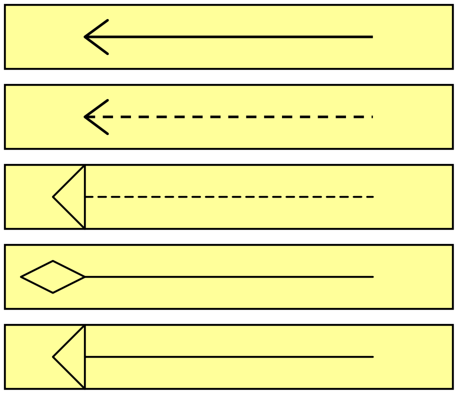
- handle the ***communication between the system surroundings and the inside*** of the system
- can provide the interface to a user or another system

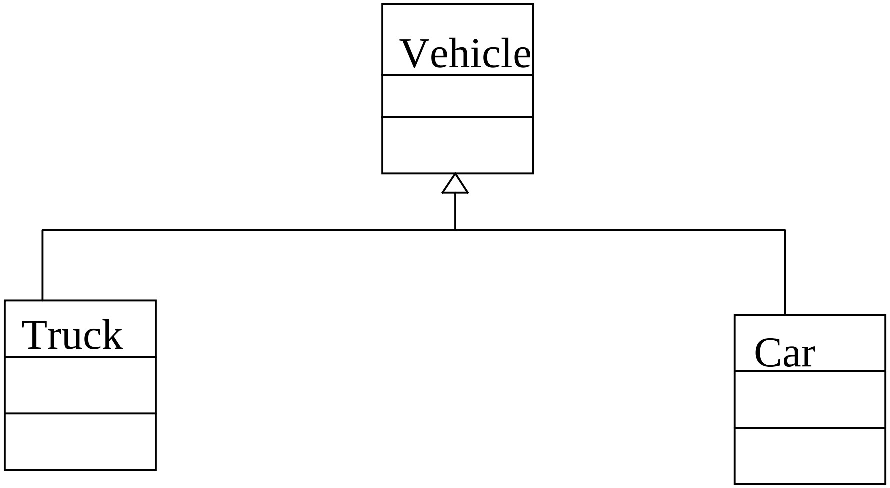
Control classes:

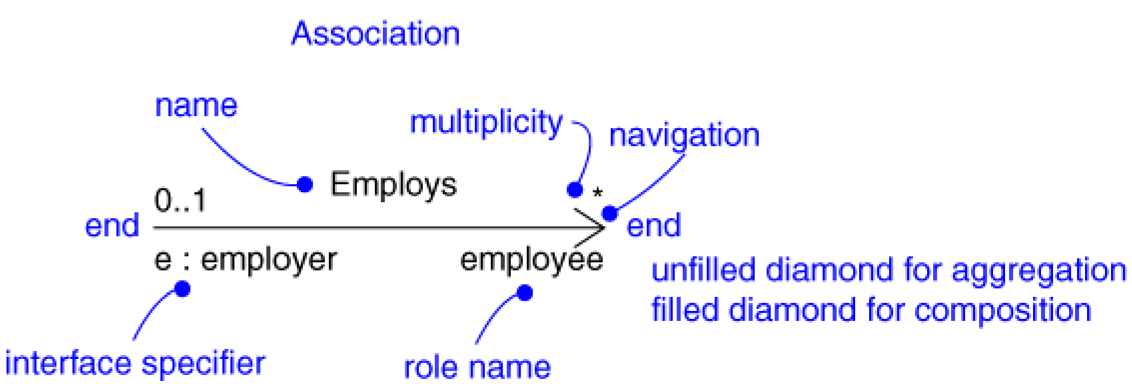
- model ***sequencing behavior*** specific to one or more use cases.
- typically are ***application-dependent*** classes.

Relationships

- ☞Provide the conduit for object interaction
- ☞Several kinds of relationships:
  - ***Association***
  - ***Dependency***
  - ***Realization***
  - ***Aggregation***
  - ***Inheritance***



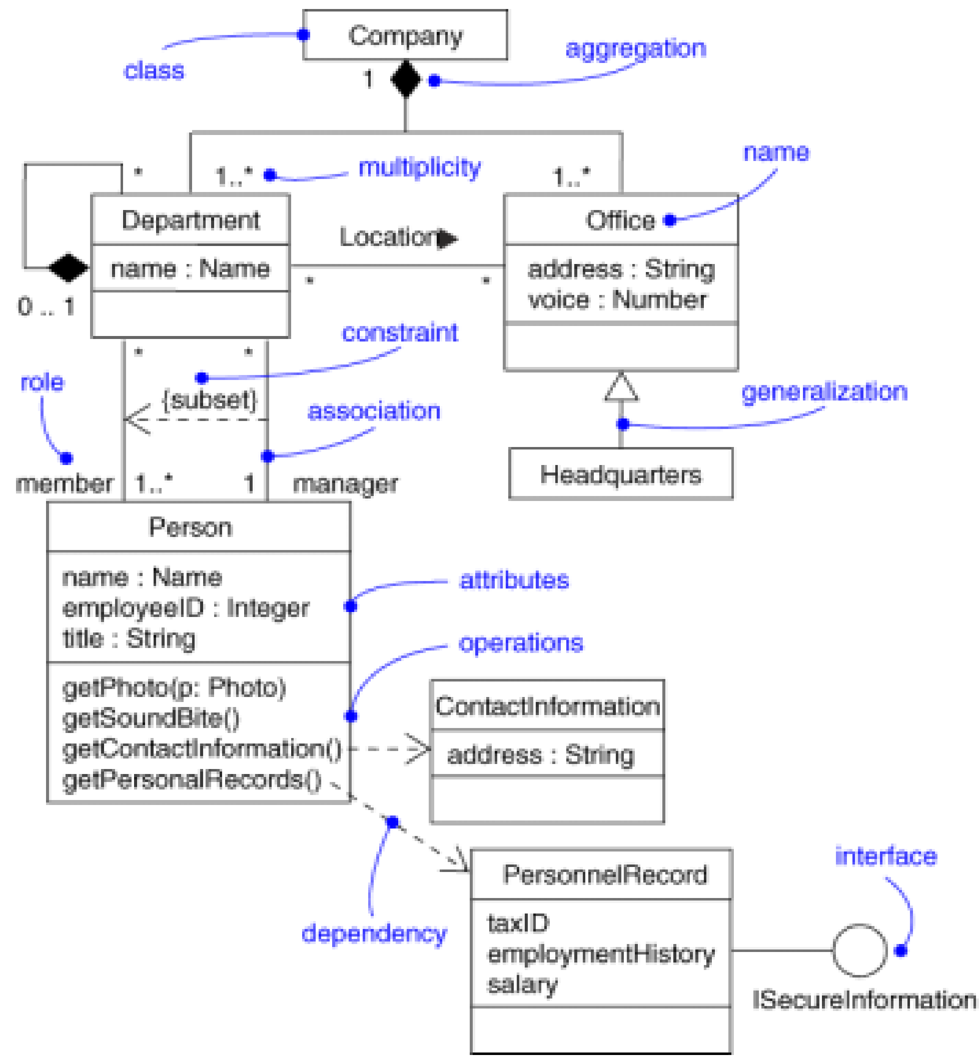




Class Diagram

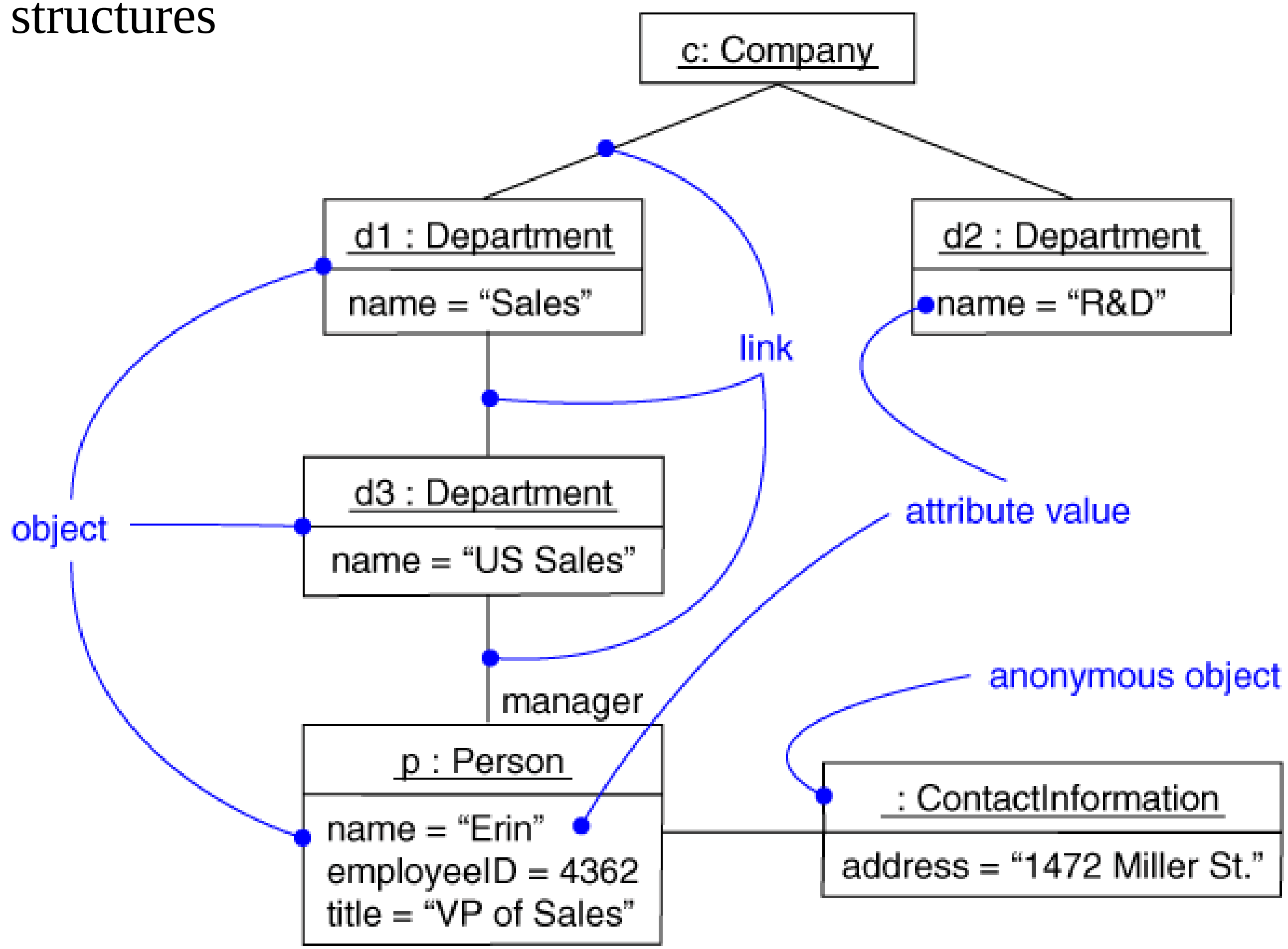
☞Purpose

- Provide a picture or view of some or all the ***classes/interfaces in the model***
- Static design view of the system



Object Diagram

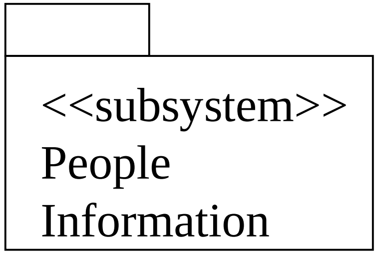
- Shows a ***set of objects*** and ***their relationships*** at a point in time
- Shows ***instances*** and ***links***
- Built during analysis and design (address the static design view)
- Purpose
  - •Illustrate data/object structures
  - •Specify snapshots



Static Structure Diagrams

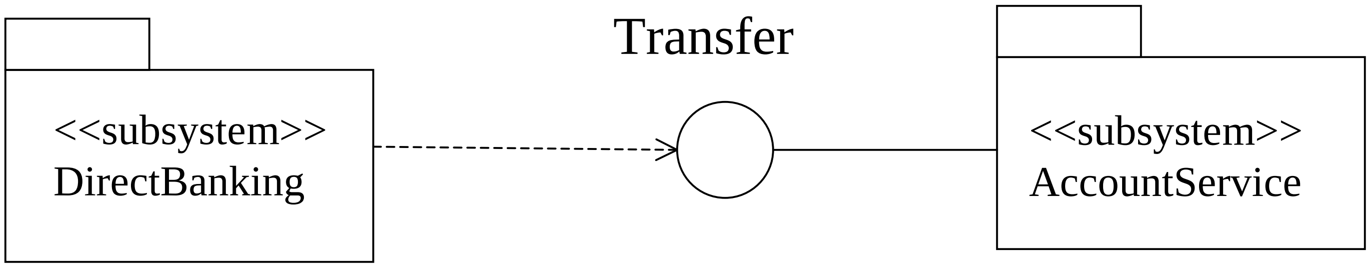
☞***Subsystem:*** Independent unit of functionality that consists of a collection of related classes and/or other subsystems.

- Offer interfaces and uses interfaces provided by other subsystems.
- In the UML, subsystems are represented as folders/***packages***:



☞***Dependency Relationships***: *provides* and *uses* relationships

- ***Uses*** relationship, shown as a dashed arrow to the used interface.
- ***Provides*** relationship, shown as a straight line to the provided interface.
- Subsystem A is dependent on subsystem B implies that one or more classes in A initiates communication with one or more public classes in B: A is called the ***client*** and B the ***supplier***.



3. Interactions

Use Case Realization

☞the functionality of a use case is defined by describing the scenarios involved.

- a scenario is an instance of a use case: it is one path through the flow of events for the use case.
- ***each use case is a web of scenarios***: primary scenarios (the normal flow for the use case) and secondary scenarios (the what-if logic of the use case).
- scenarios help identify the objects, the classes, and the object interactions needed to carry out a piece of the functionality specified by the use case.

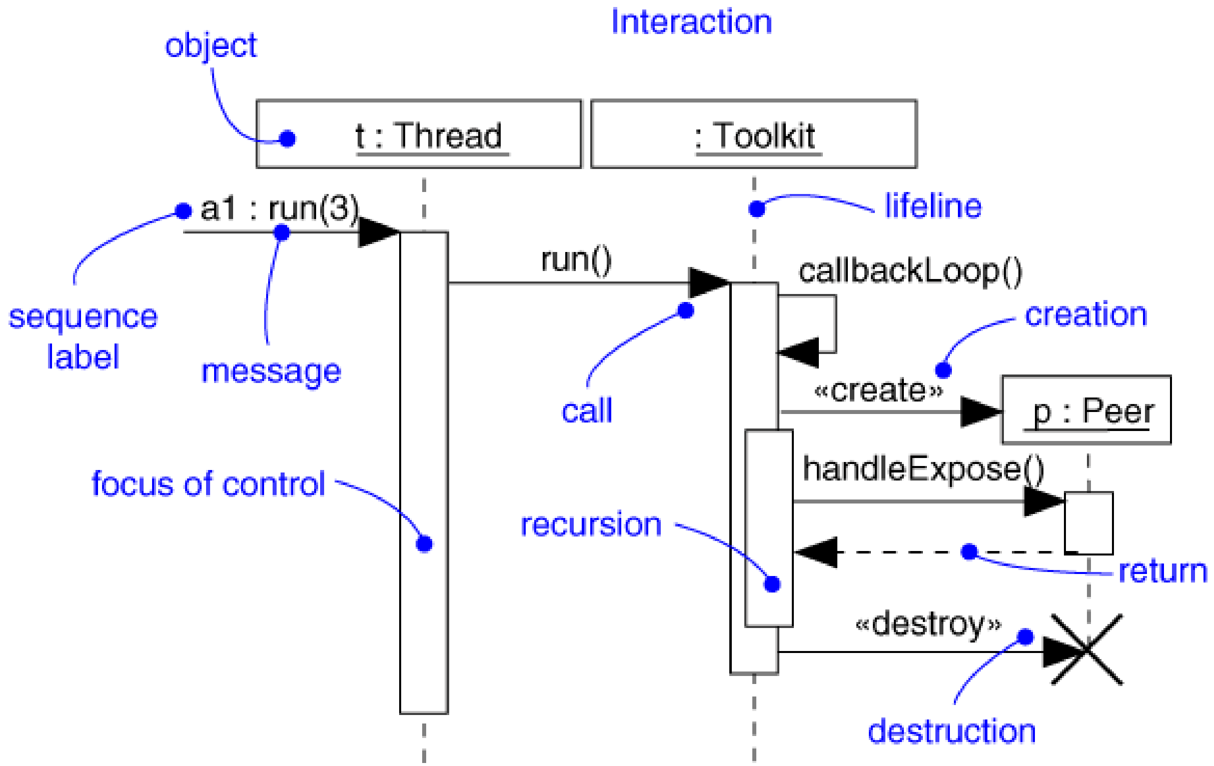
☞the flow of events for a use case is captured in text, whereas scenarios are captured in interaction diagrams.

☞two types of interaction diagrams:

- ***sequence diagrams***
- ***collaboration diagrams***

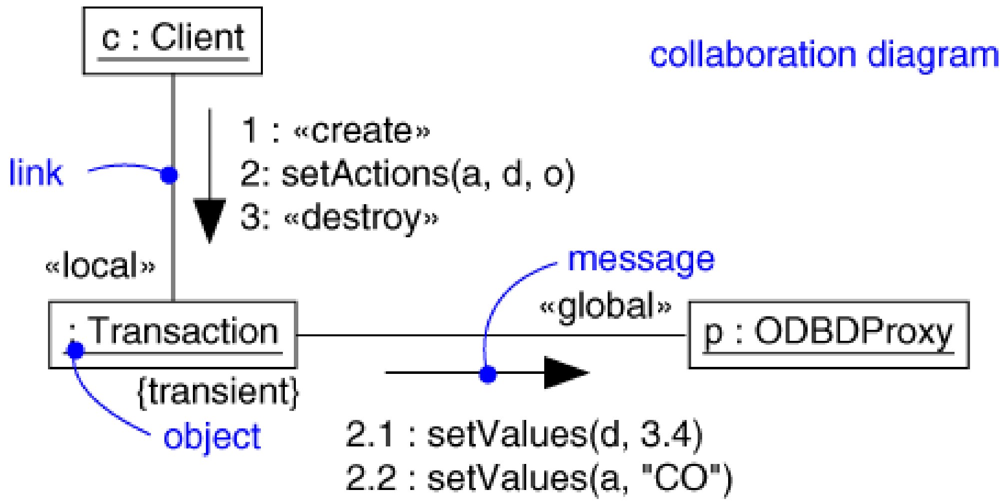
Sequence Diagram

- Shows object interactions ***arranged in time sequence***
- Purpose
  - –Model flow of control
  - –Illustrate typical scenarios
- Depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.



Collaboration Diagram

- Shows object interactions organized around the objects and their links to each other (Arranged to ***emphasize structural organization***)
- Purpose
  - –Model flow of control
  - –Illustrate coordination of object structure and control
- Alternate way to describe a scenario



- A collaboration diagram contains:
  - -objects drawn as rectangles
  - -links between objects shown as lines connecting the linked objects
  - -messages shown as text and an arrow that points from the client to the supplier.

4. Dynamic Behavior

Statechart Diagram

- ☞Use cases and scenarios provide a way to describe system behavior, that is the interaction between objects in the system.

- ☞A state transition diagram allows the modeling of the behavior inside a single object.

  - It ***shows the events or messages*** that cause a ***transition*** from ***one state to another***, and the actions that result from a state change.

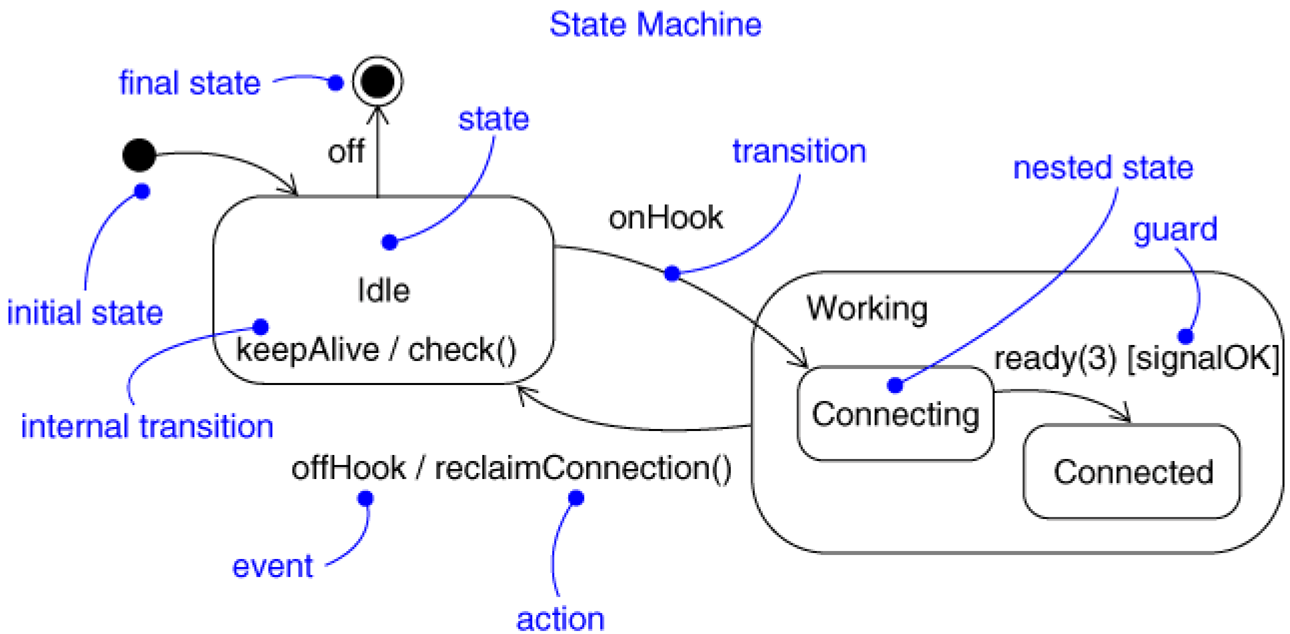
  - It is ***created only for classes with significant dynamic behavior***, like control classes.

☞ **State**:

•a condition during the life of an object when it ***satisfies some condition, performs some action***, or ***waits for an event***

•found by examining the attributes and links defined for the object

•represented as a rectangle with rounded corners



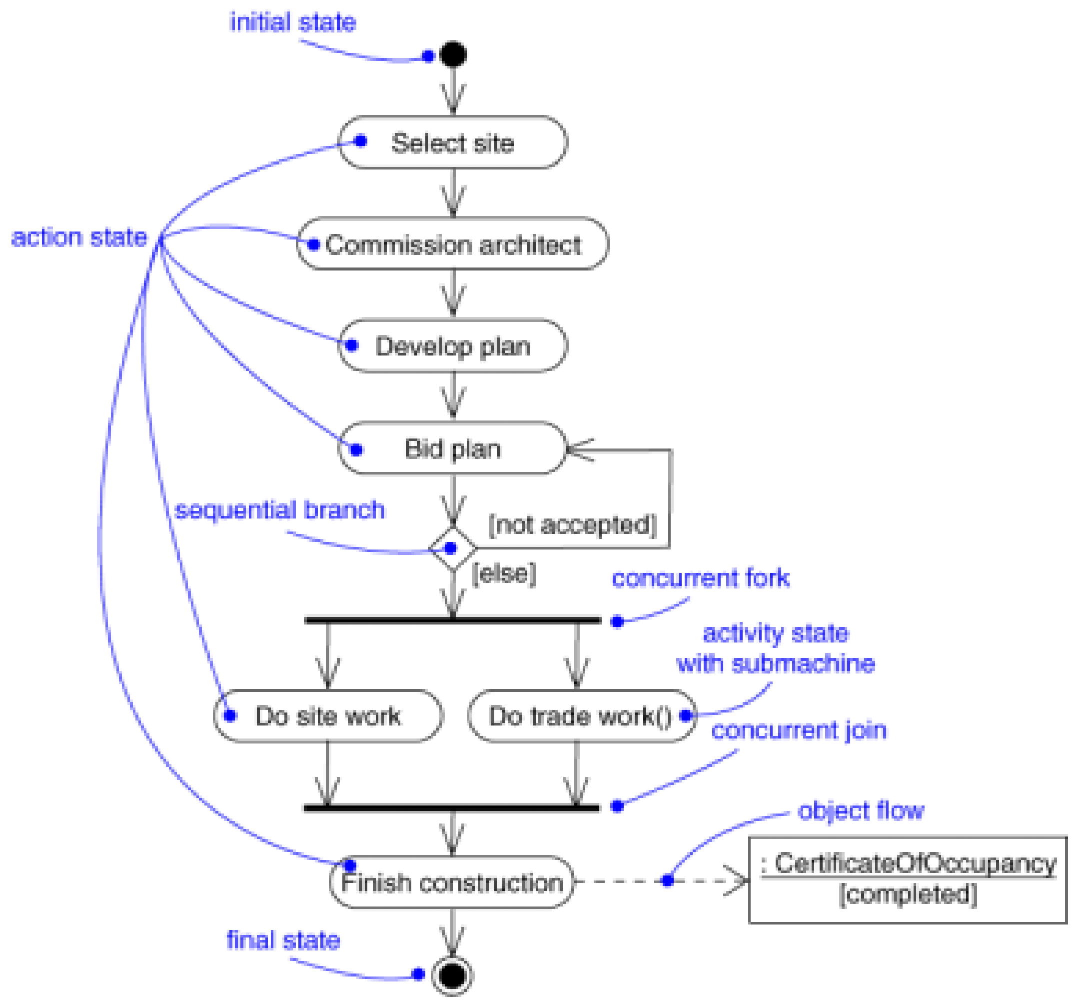
☞ **Transitions**:

•represents a change from an originating state to a successor state (that may be the same as the originating state).

•may have an action and/or a guard condition associated with it, and may also trigger an event.

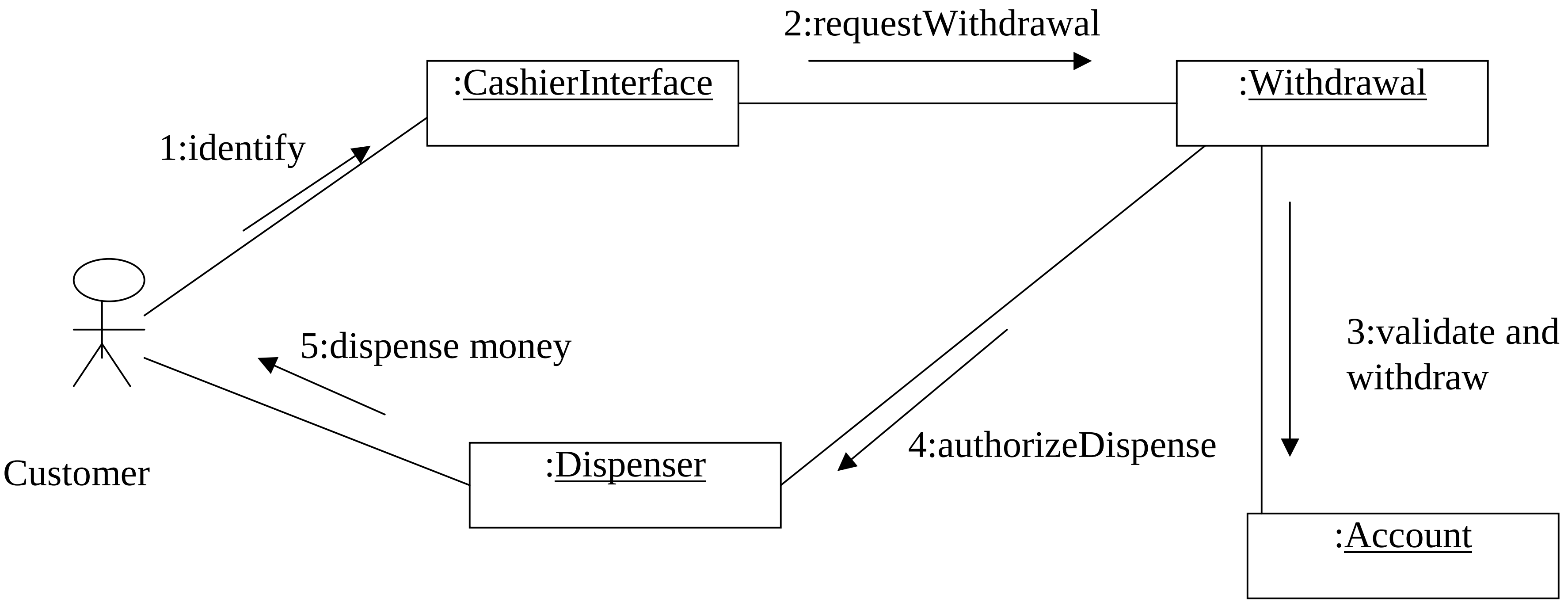
Activity Diagram

- Captures dynamic behavior (activity-oriented)
- Behavior that occurs within the state is called an **activity**: starts when the state is entered and either completes or is interrupted by an outgoing transition.
- Purpose
  - -Model business workflow
  - -Model operations

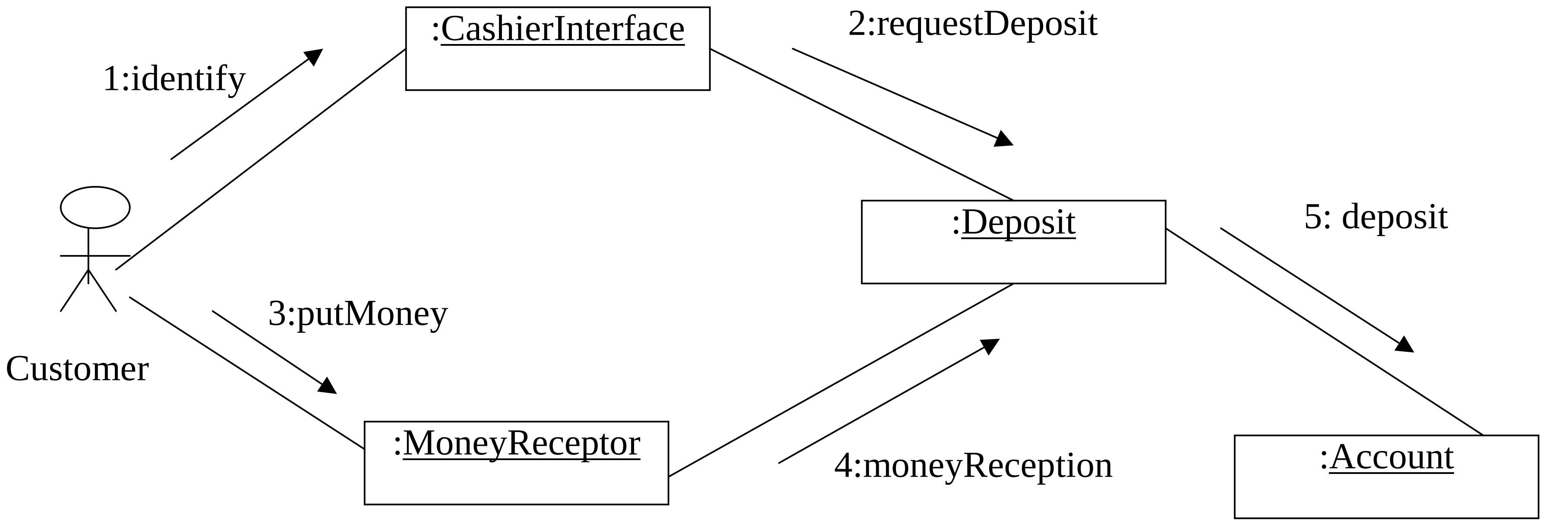


5. Example: Logical View for the ATM

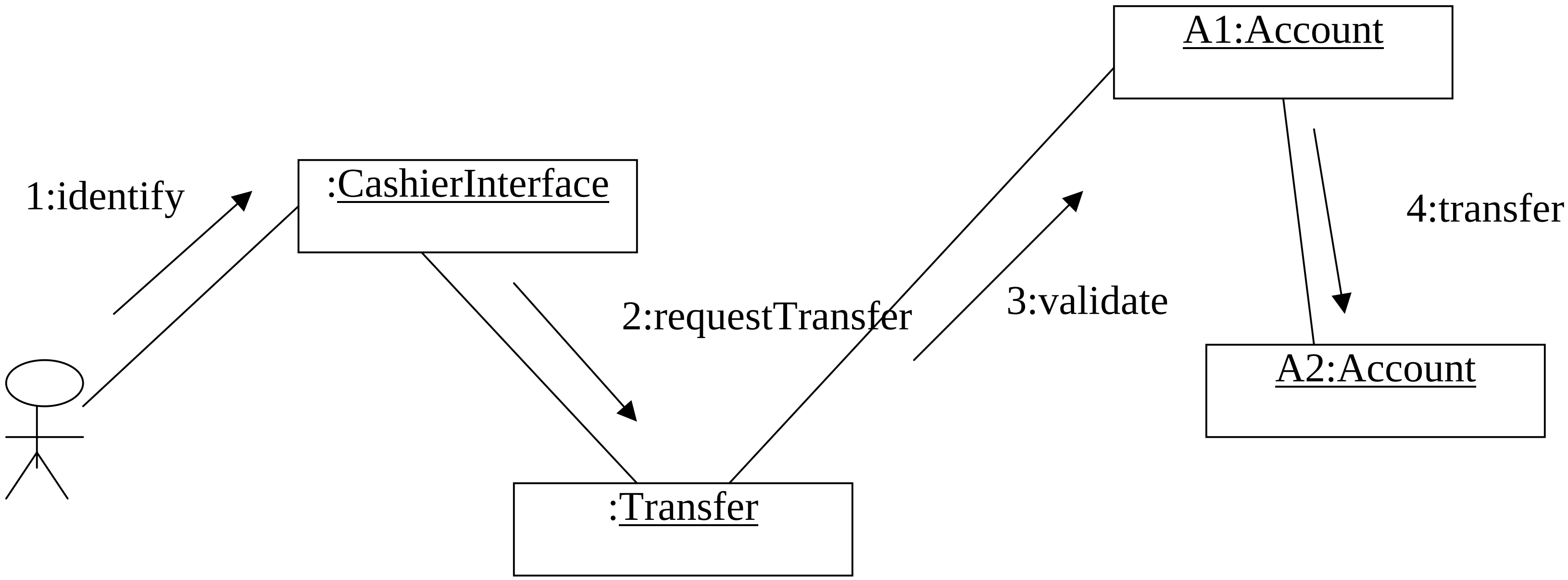
Withdraw Money Use case



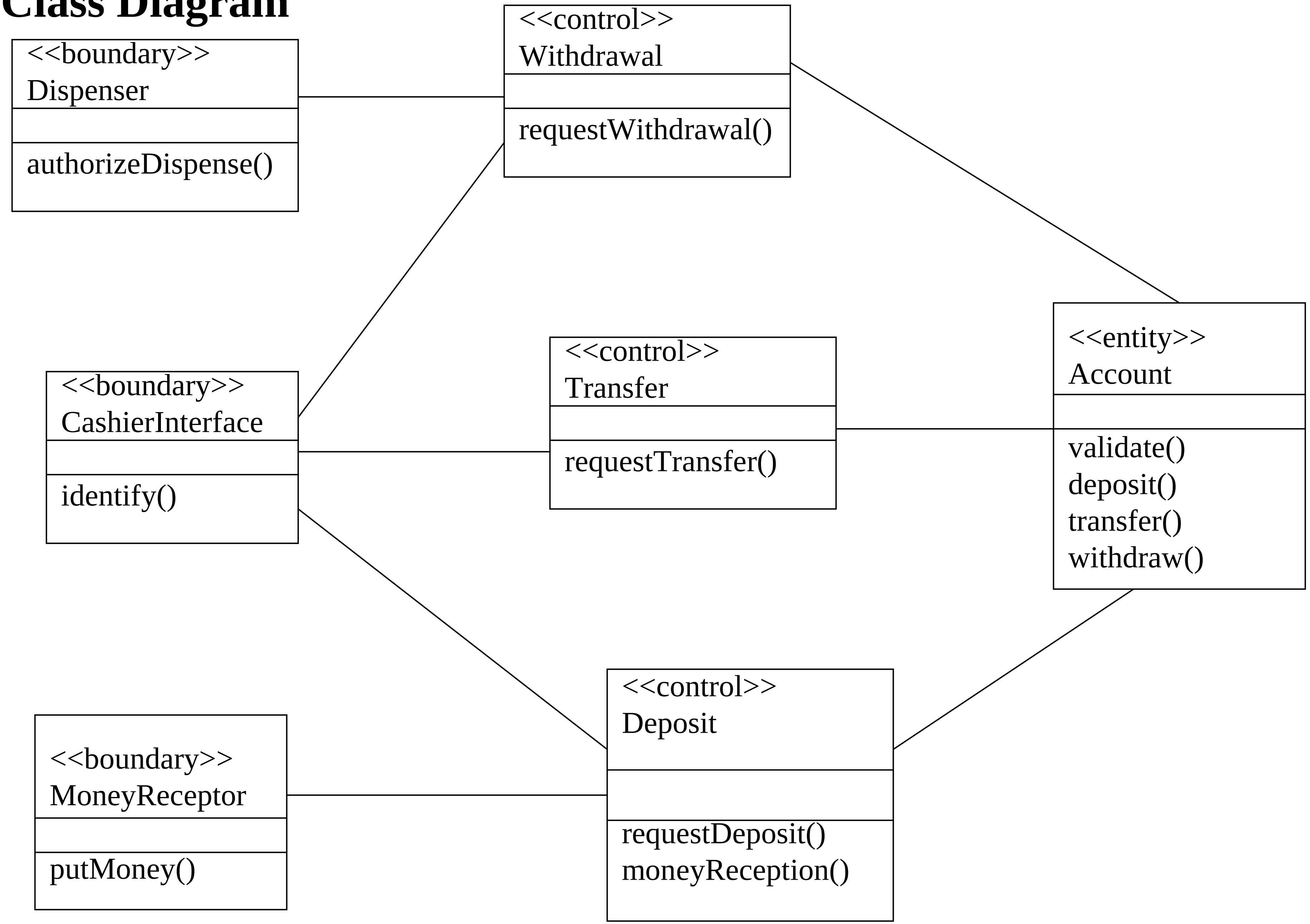
Deposit Use Case



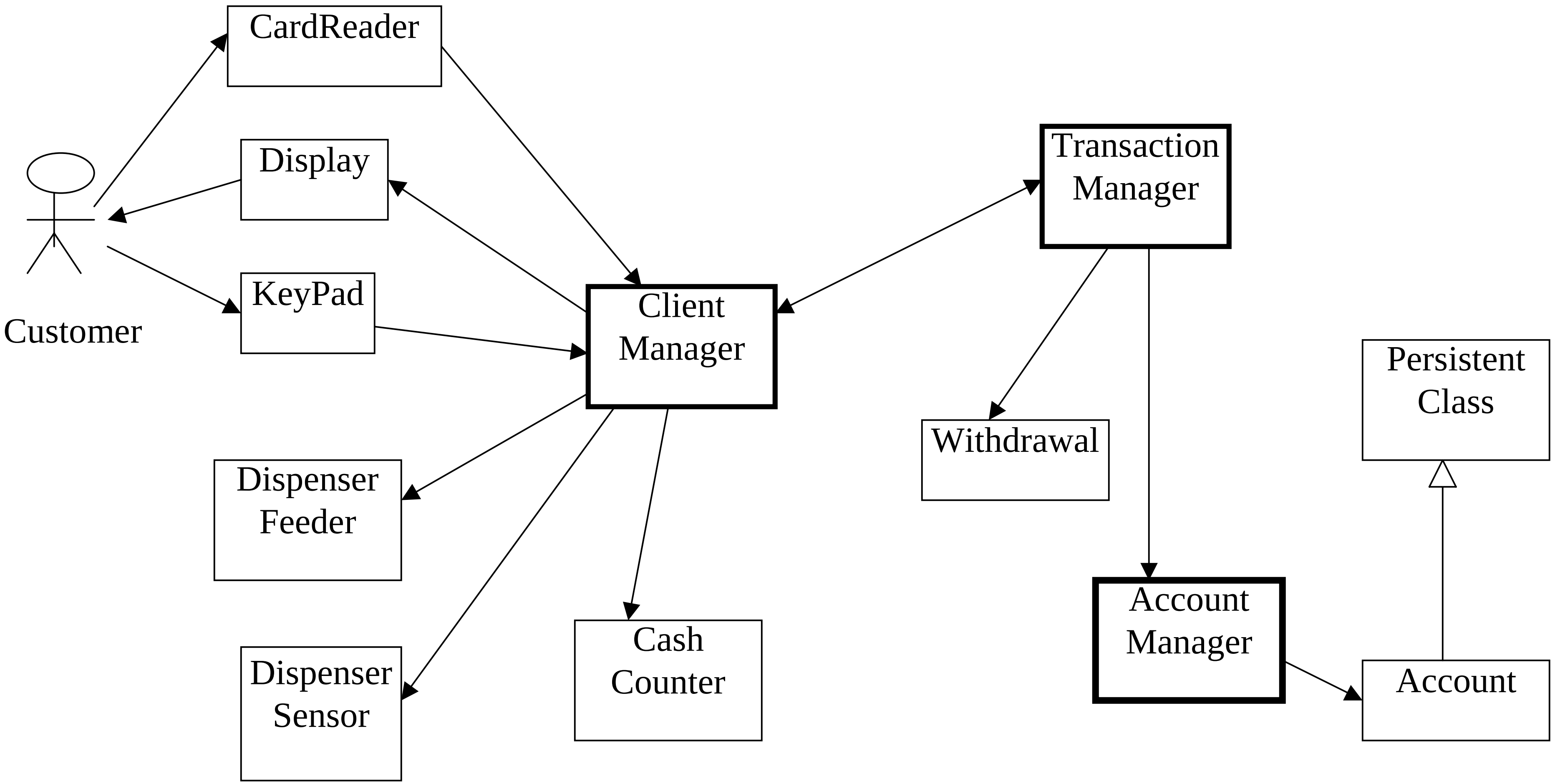
Transfer Use Case



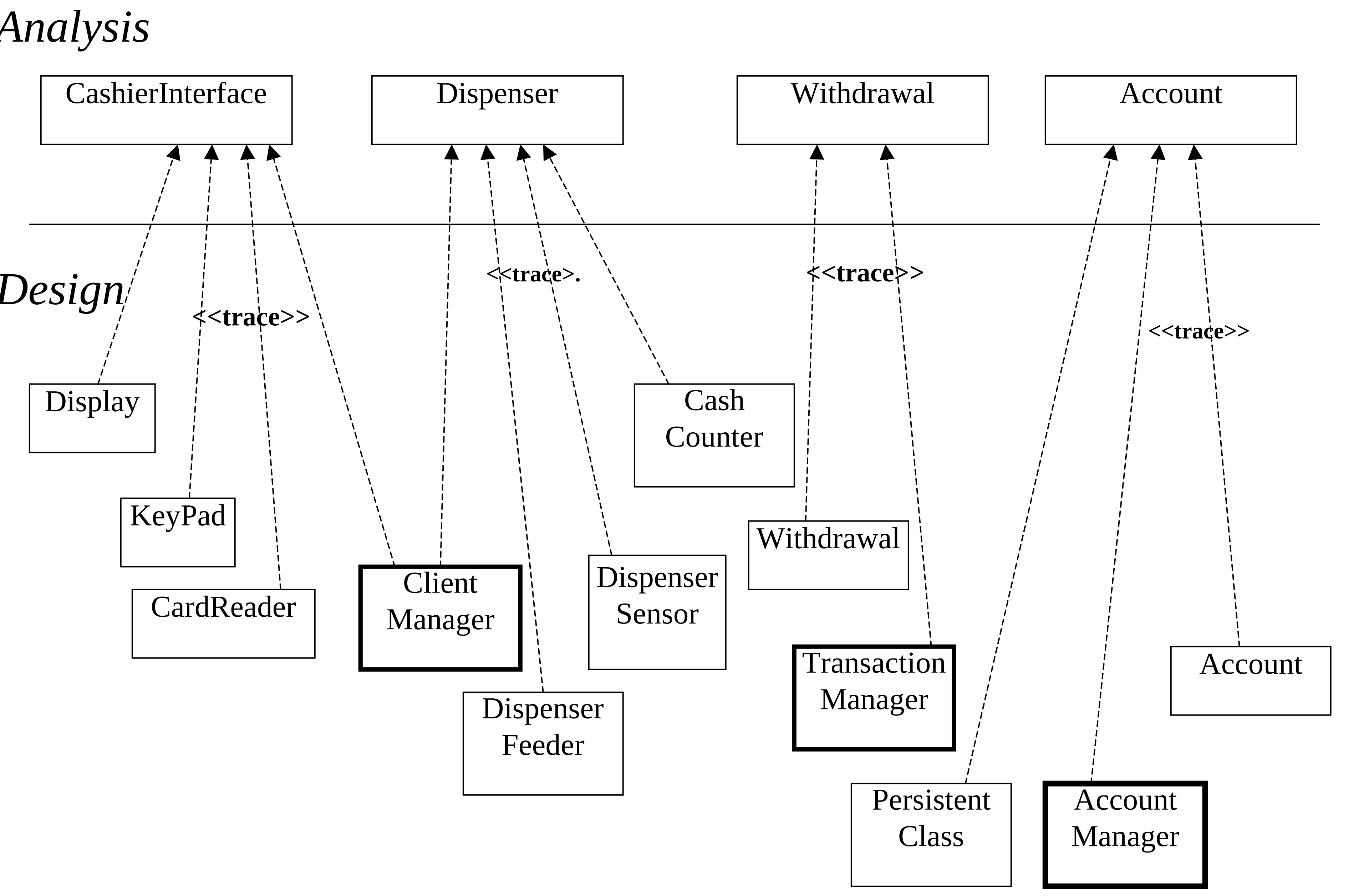
Class Diagram



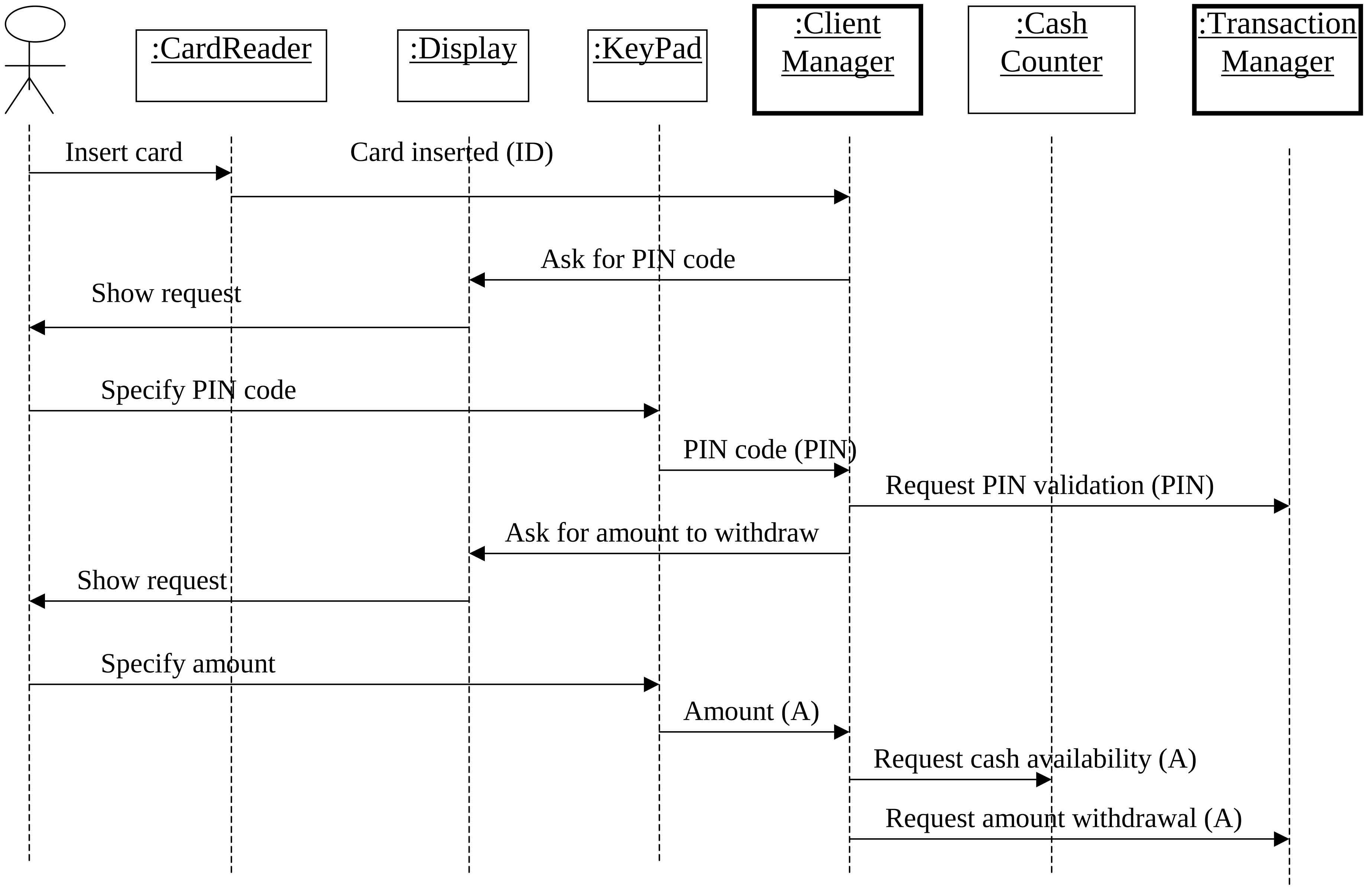
(Refined) Class diagram providing a view of the classes involved in withdraw Money use case (design model)



Traceability (Withdraw use case)

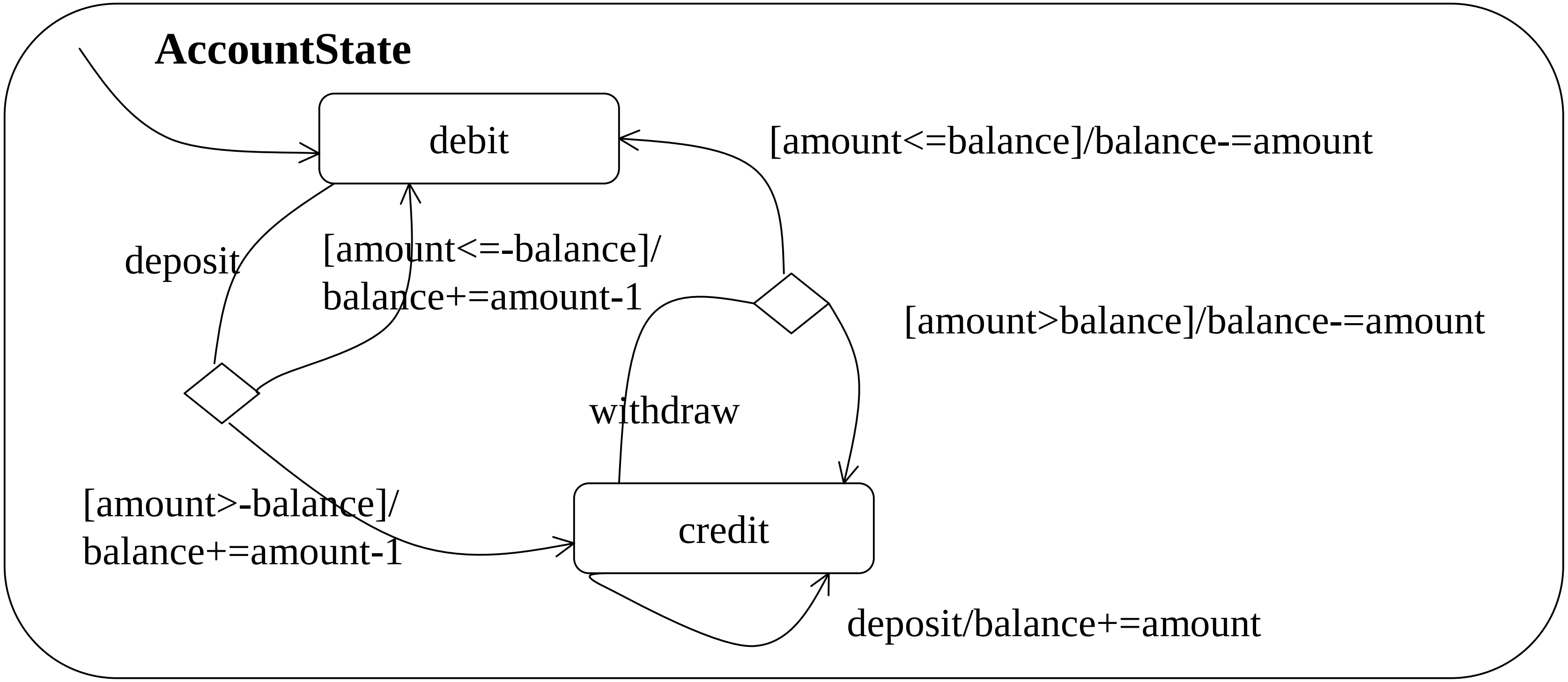


A Scenario of the Withdraw Money Use Case (Design Model)



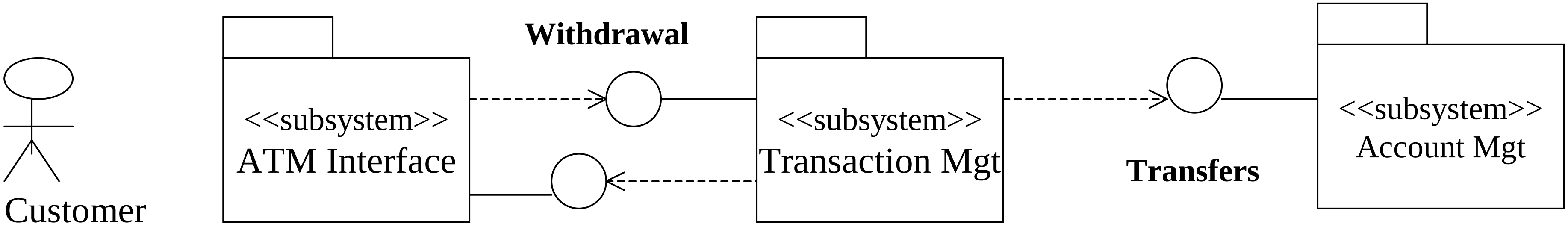
• • •

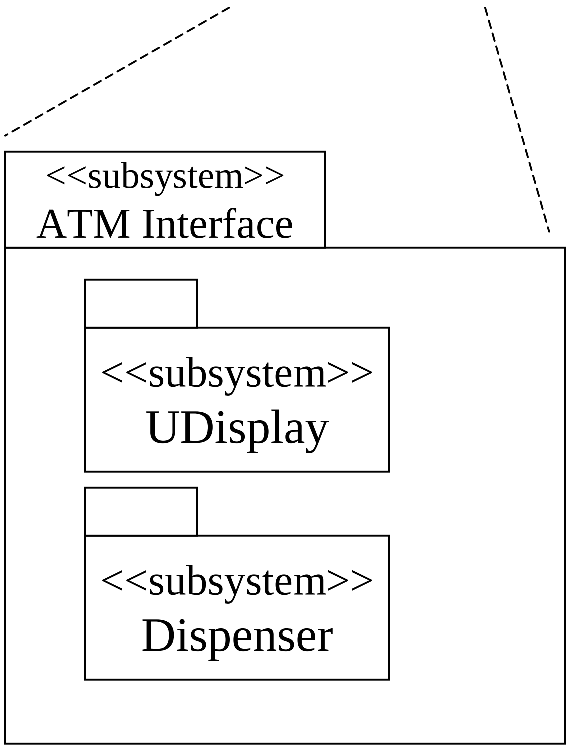
Statechart Modeling Dynamic Behavior of Account Class




```
public class Account {
  private int balance;
  public void deposit (int amount) {
    if (balance > 0) balance = balance + amount;
    else balance = balance + amount – 1; // transaction fee
   }
  public void withdraw (amount) {
    if (balance>0) balance = balance – amount;
  }}
```

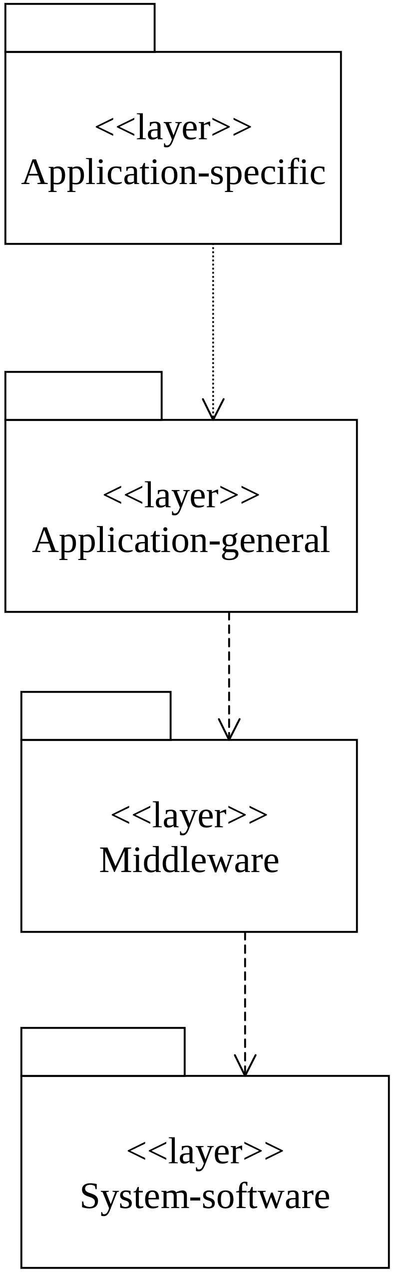
Static Structure Diagram



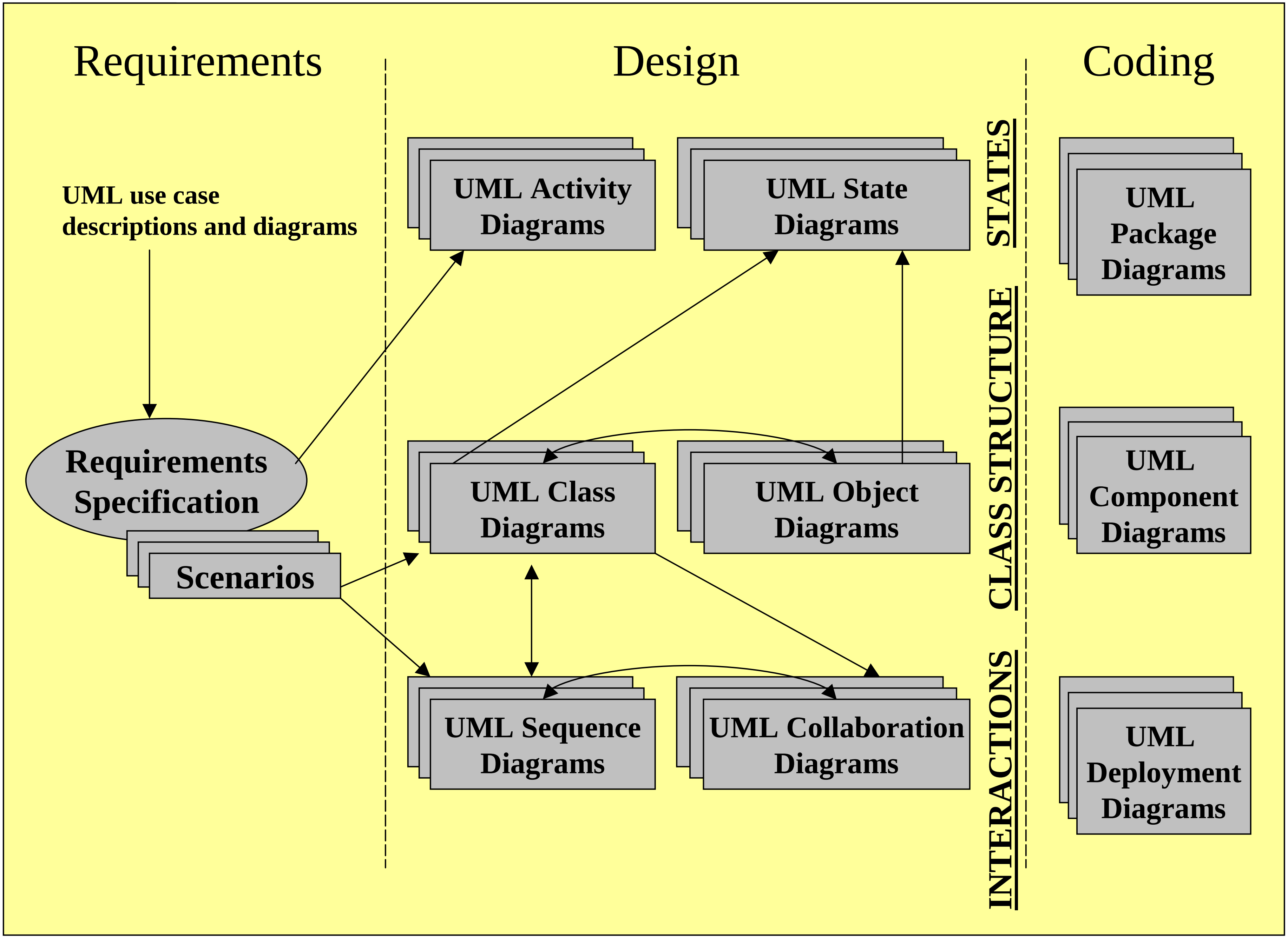


Classes	**Subsystems**
CardReader, Display, KeyPad, ClientMgr	UDisplay/ATM Interface
DispenserFeeder, DispenserSensor, CashCounter	Dispenser/ATM Interface
Withdrawal, TransactionMgr	TransactionMgt
Account, PersistentClass, AccountMgr	AccountMgt

Structuring Using Layer Architectural Pattern



Subsystems	**Layers**
ATM Interface	Application-specific
Transaction Mgt, Account Mgt	Application-general
	Middleware
	System-software


Requirements
Design
Coding
UML use case descriptions and diagrams
Requirements Specification
Scenarios
UML Activity Diagrams
UML State Diagrams
UML Class Diagrams
UML Object Diagrams
UML Sequence Diagrams
UML Collaboration Diagrams
STATES
CLASS STRUCTURE
INTERACTIONS
UML Package Diagrams
UML Component Diagrams
UML Deployment Diagrams