MANCHESTER 1824

The University of Manchester

COMP23420 Lecture 5

Structural Modelling: System Classes

Kung-Kiu Lau

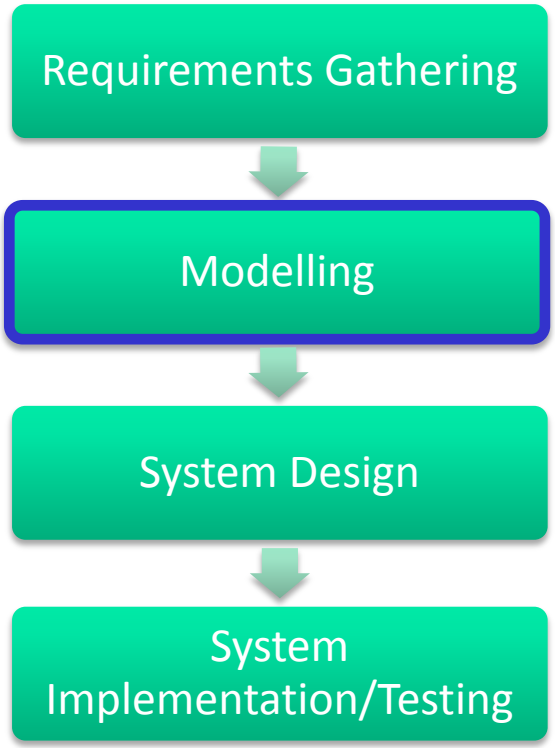
kung-kiu@cs.man.ac.uk

Office: Kilburn 2.68

Overview

- Where we are in the development process
- Refining domain classes into system classes
- Some guidelines and techniques
- Workshop 3: Structural Modelling for HTV

Where we are in the Development Process



Functional Model
- Activity Diagram
- Use Case Diagram

Structural Model
- Domain Model (Classes)
- System Classes

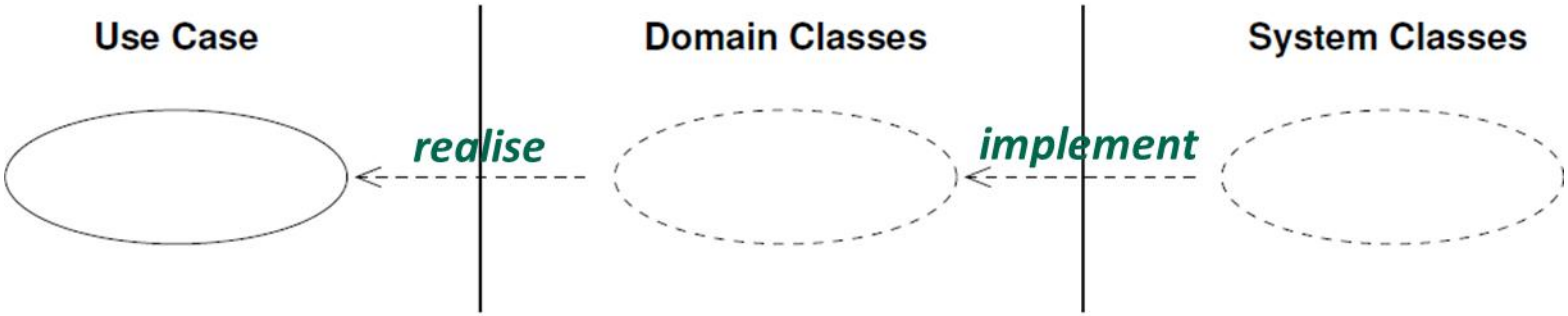
Behavioural Model
- Sequence Diagrams
- Communication Diagrams
- State Machine Diagrams

We have the domain model, which is our initial structural model; and now we refine the initial structural model, by refining domain classes into system classes.

Implementing Use Case Realisations

Domain classes are conceptual.

To implement use case realisations, we need to use software classes (system classes).

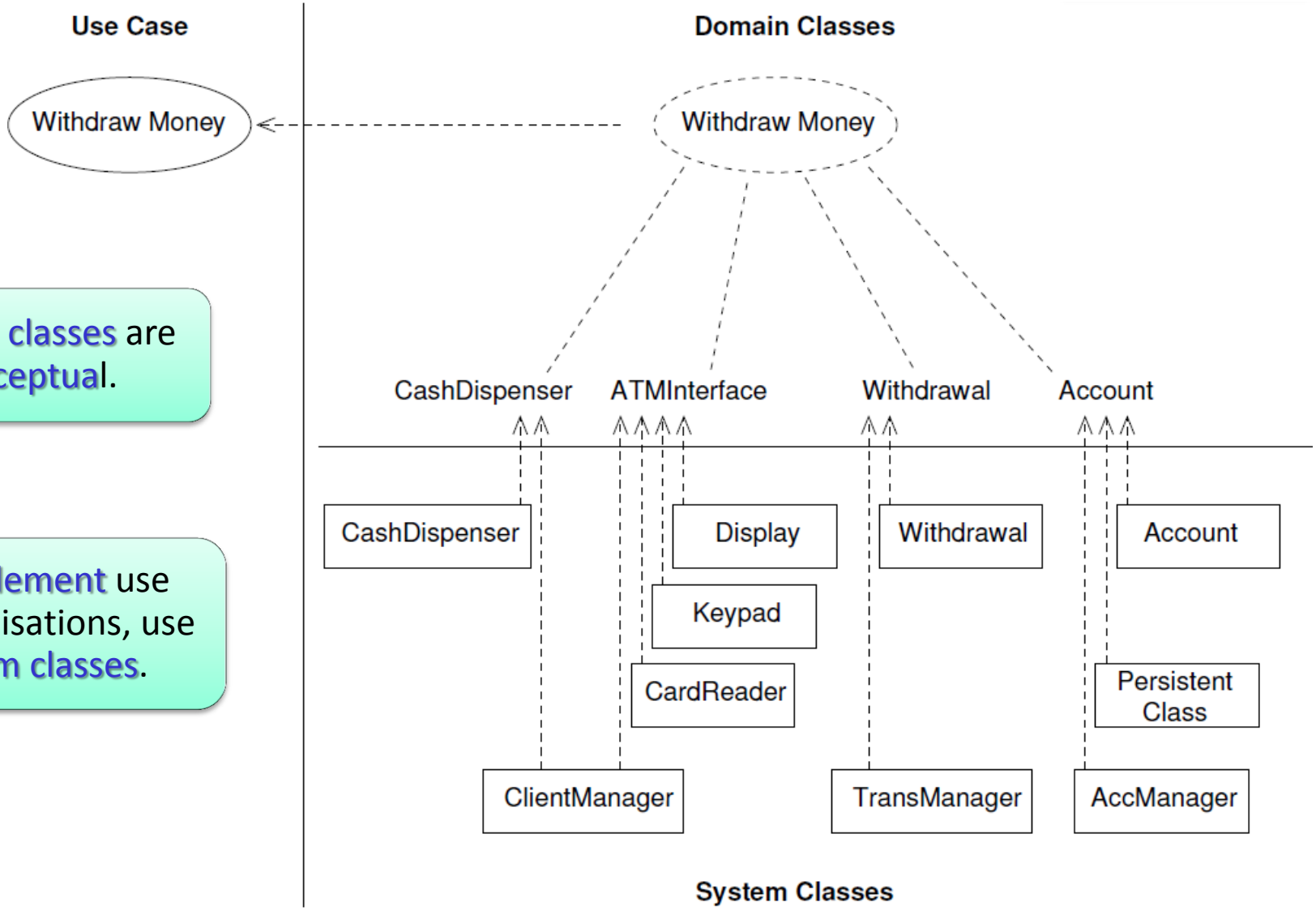


Use case realisation by domain classes and system classes.

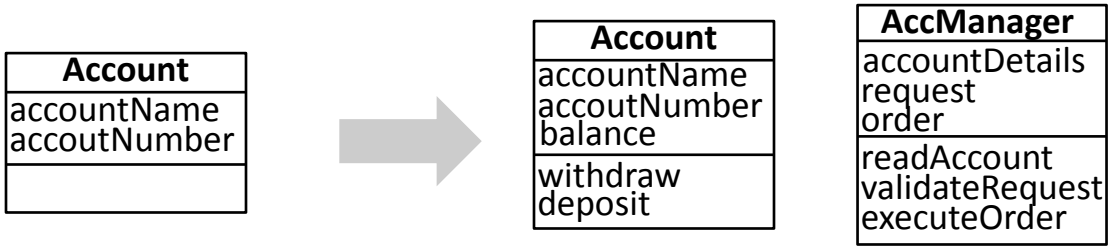
ATM Example

Domain classes are conceptual.

To implement use case realisations, use system classes.



From Domain Classes to System Classes



Domain classes collaborate to realise (all) use cases

So it suffices to implement domain classes (and their collaborations)

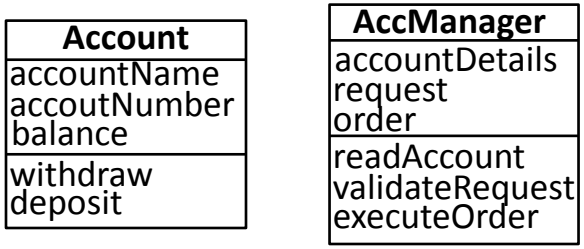
However, domain classes do not represent software artefacts

So need to turn domain classes into software classes that can be used to implement domain classes and hence the system

We refine domain classes into system classes

This refinement is a design process that kicks off the design of the system (that is why system classes are also called design classes)

System Classes



System classes are created from domain classes

They are designed for the implementation environment

They show software objects, not domain objects

They include operations as well as attributes

They are usually more detailed than domain classes

Their class diagrams include software-oriented things such as types and visibilities

We should deal with a small number of classes at a time, since each is described in more detail.

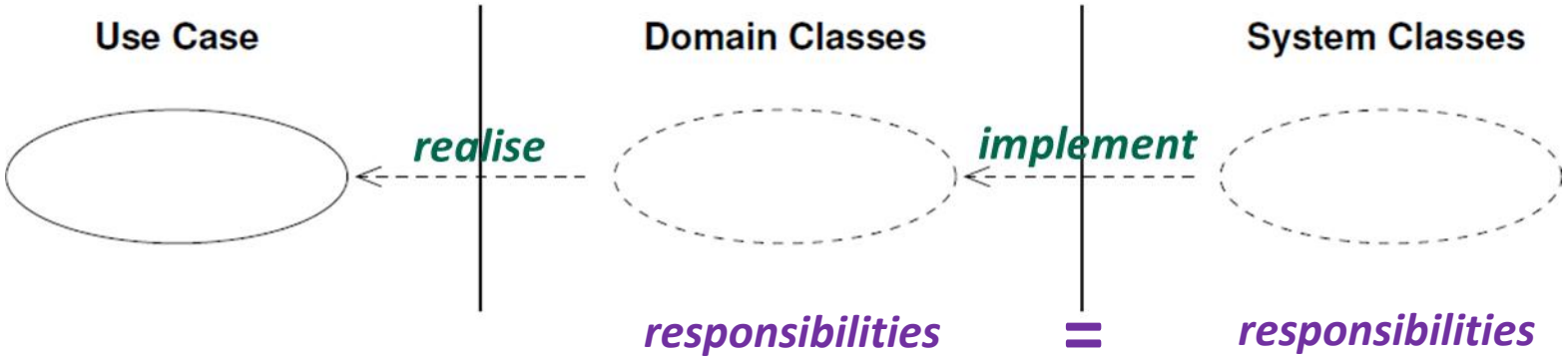
How to Identify System Classes

Domain classes often inspire system classes (``lowered representational gap'') but they don't usually correspond 1-1

Refining domain classes into system classes is a design process, so requires design skills, that follow good design principles

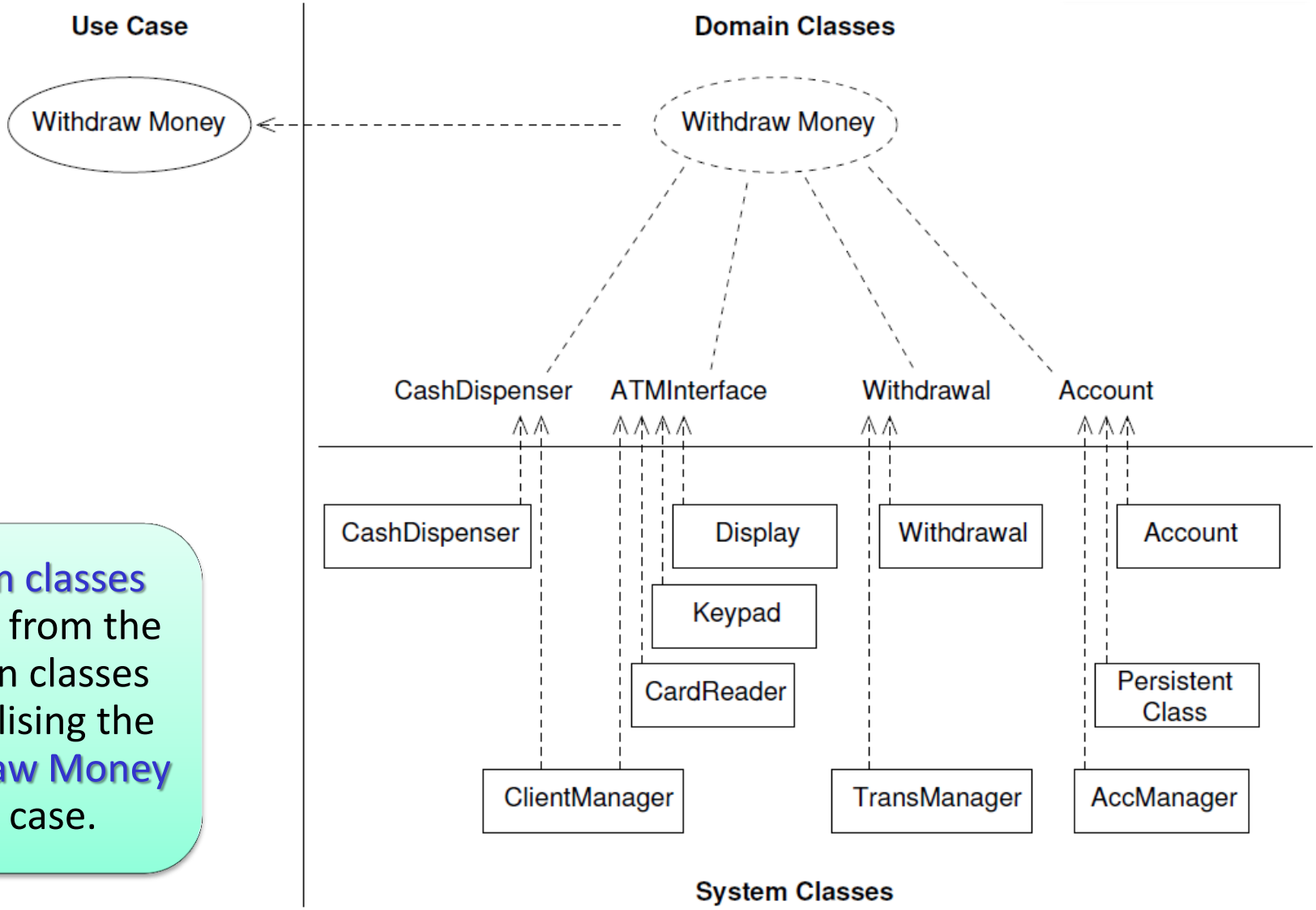
The key skill required is assigning responsibilities to software classes

The assignment must cover the responsibilities of the corresponding domain classes in each use case realisation:



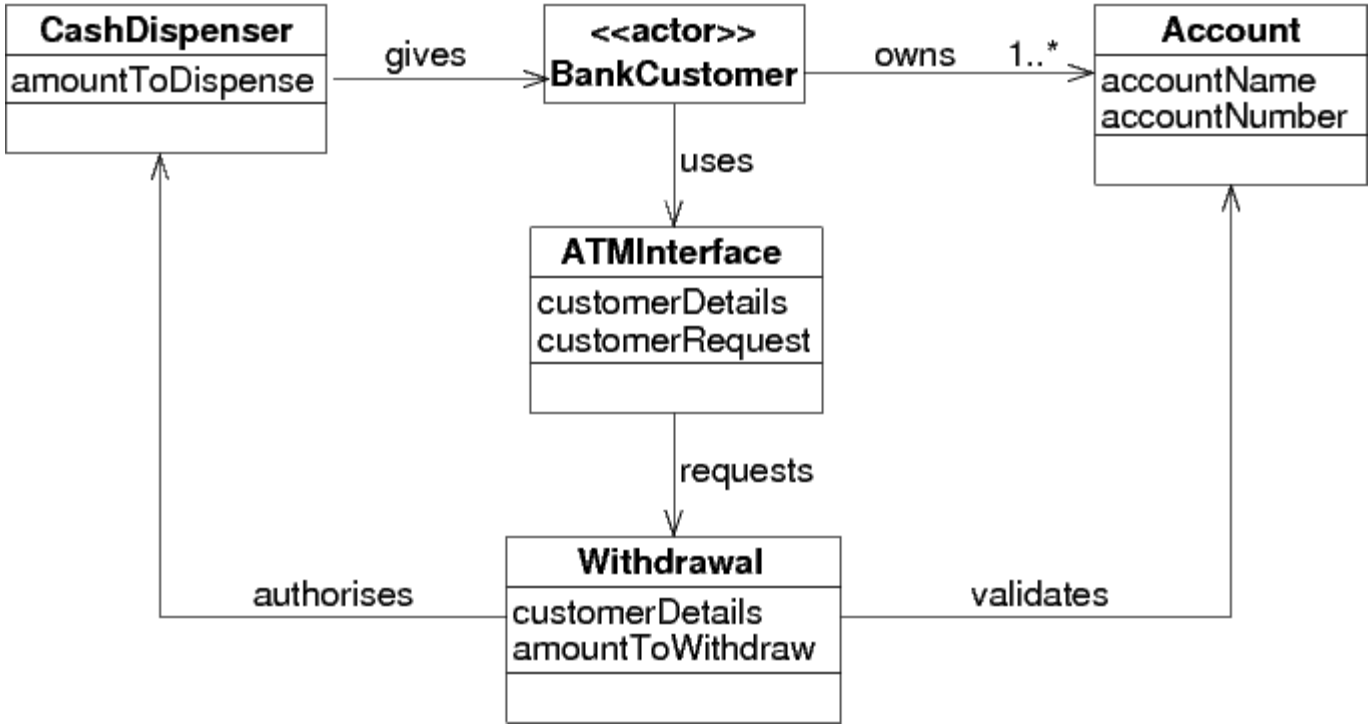
Use case realisation by domain classes and system classes.

ATM: System Classes for Withdraw Money

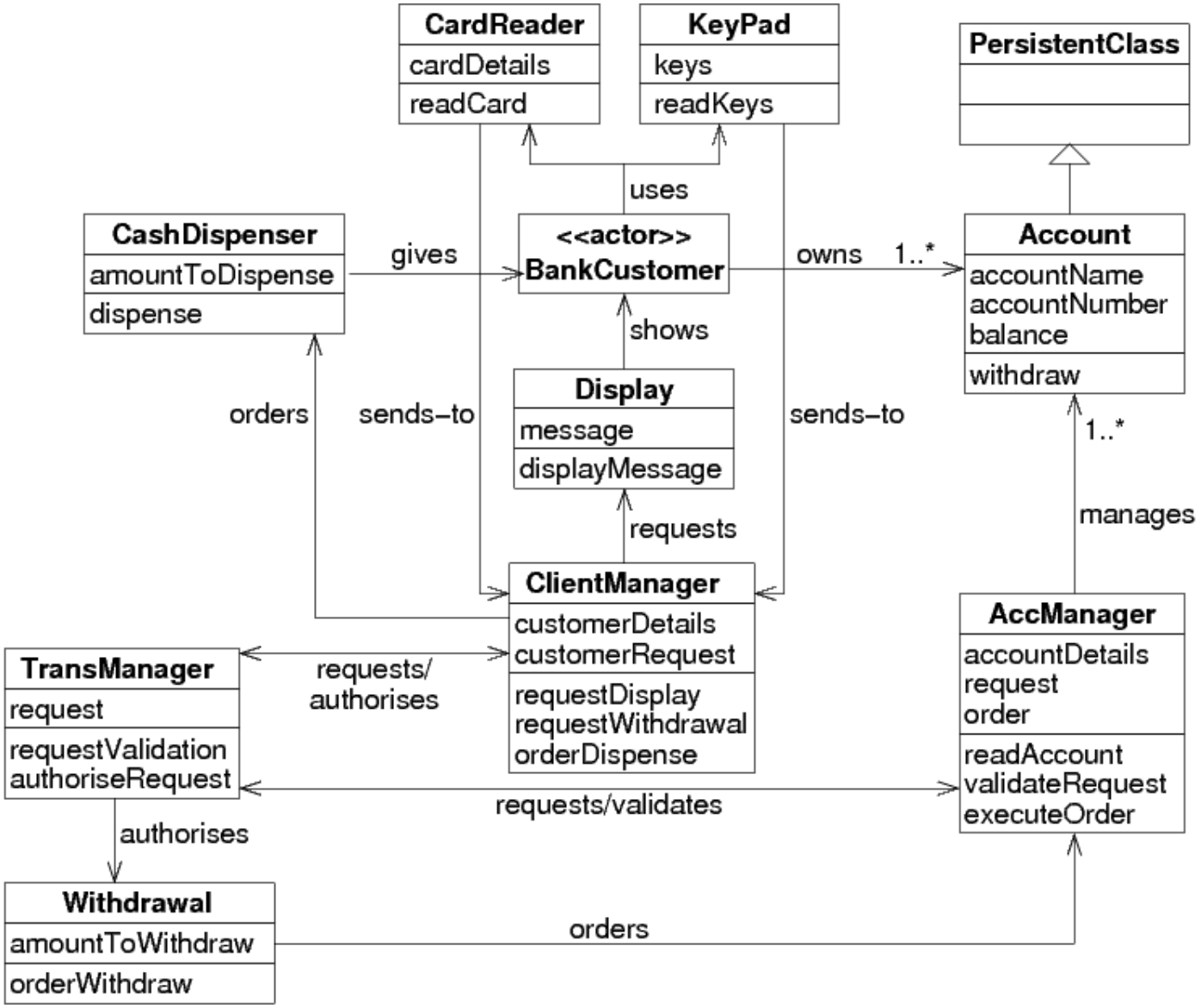


System classes refined from the domain classes for realising the Withdraw Money use case.

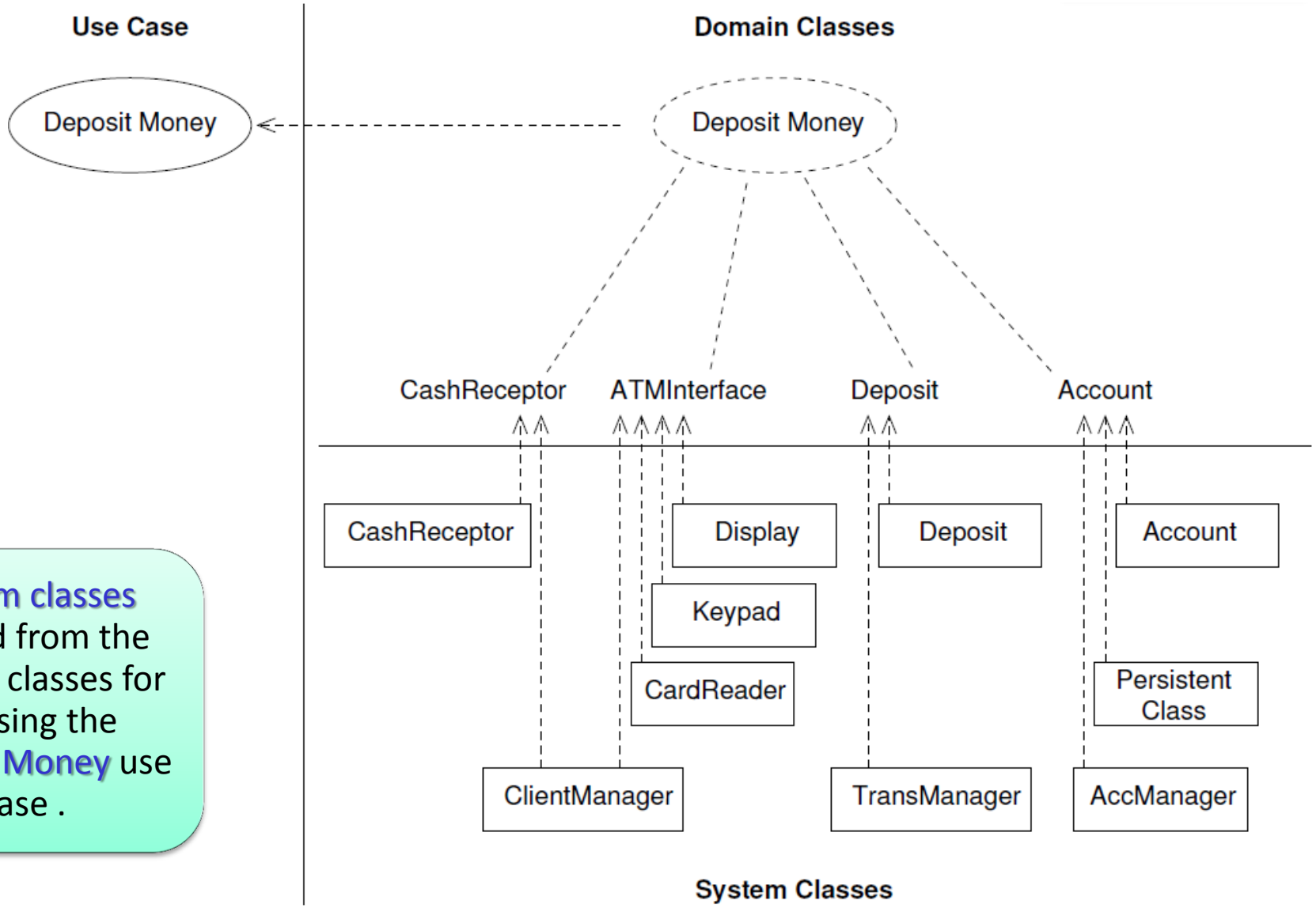
ATM: Domain Classes for Withdraw Money



ATM: System Classes for Withdraw Money

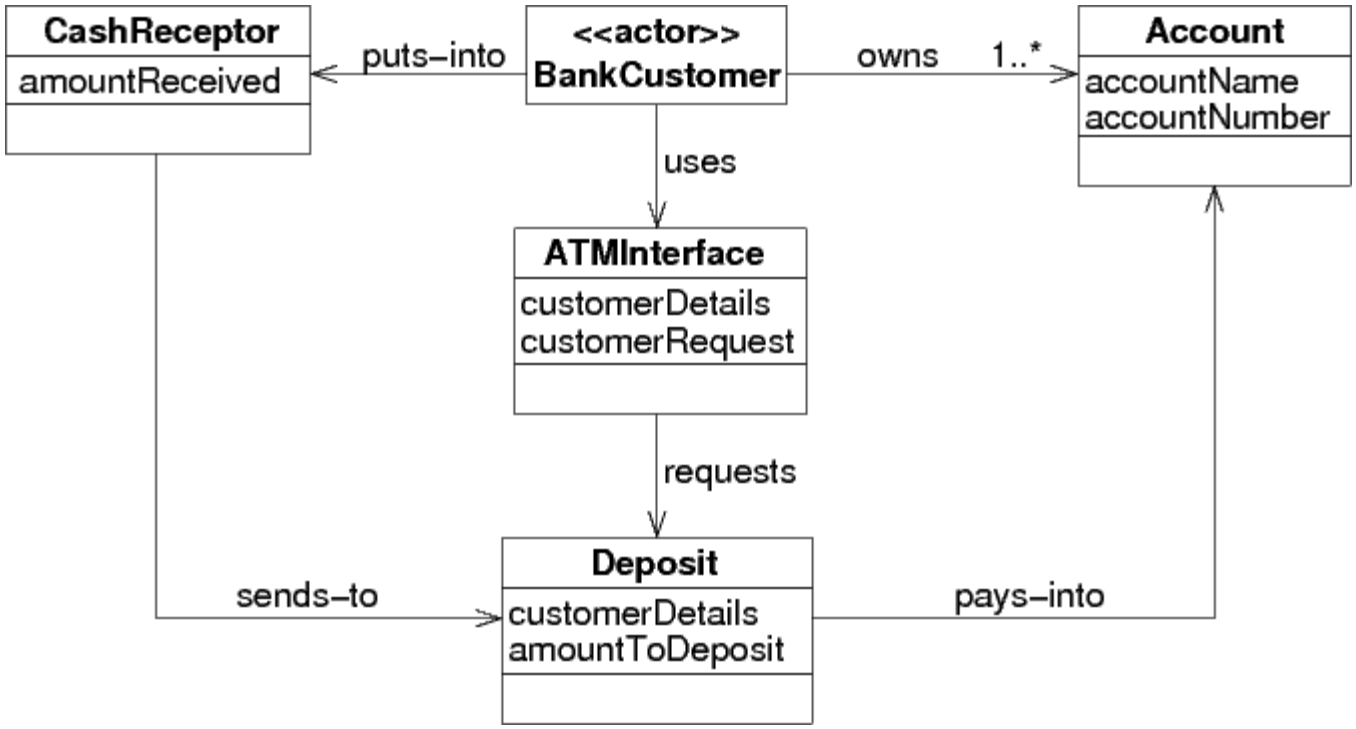


ATM: System Classes for Deposit Money

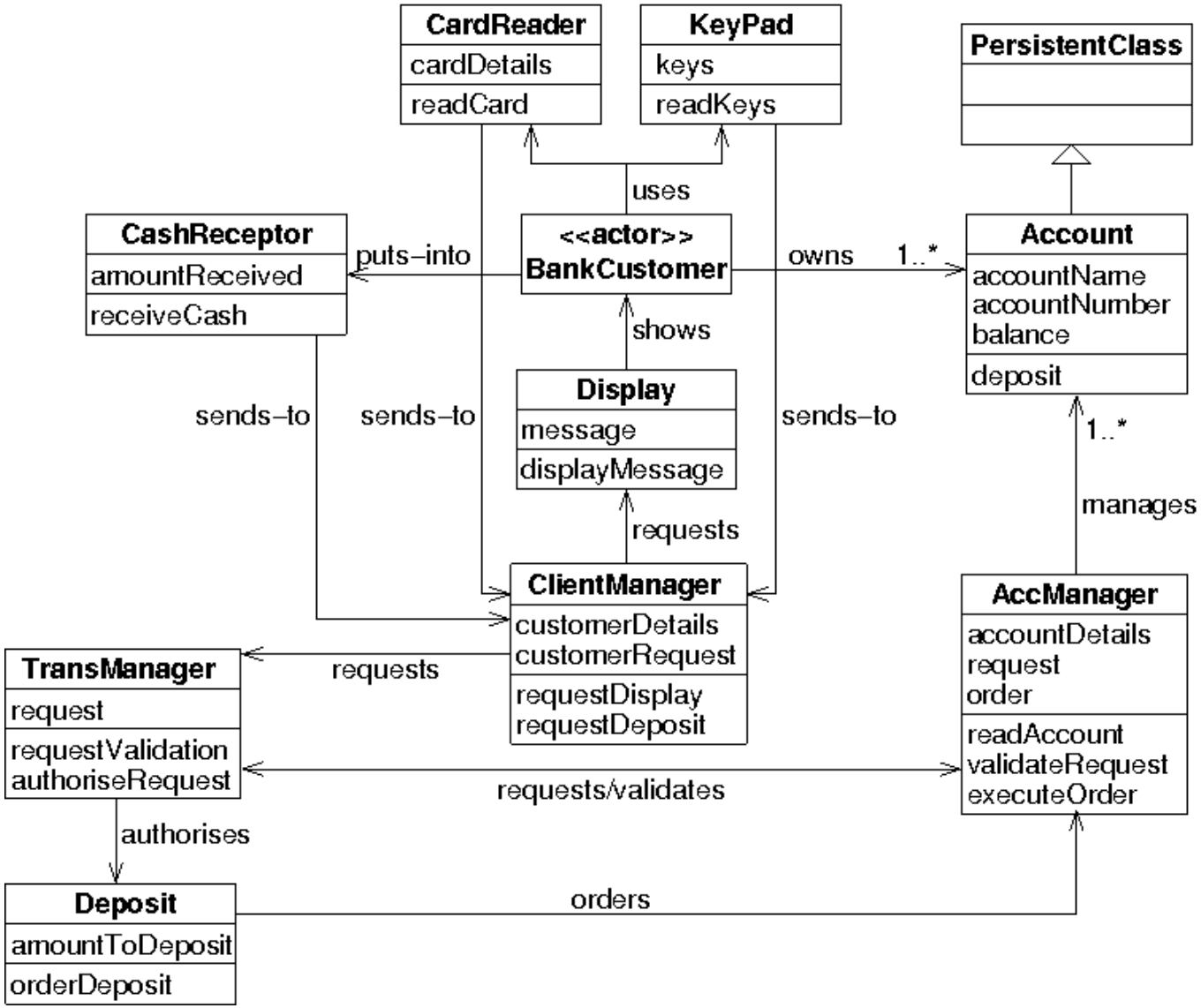


System classes refined from the domain classes for realising the Deposit Money use case .

ATM: Domain Classes for Deposit Money



ATM: System Classes for Deposit Money



The Complete System Class Diagram

Just as the domain model is the aggregation of

- domain classes
- their relationships

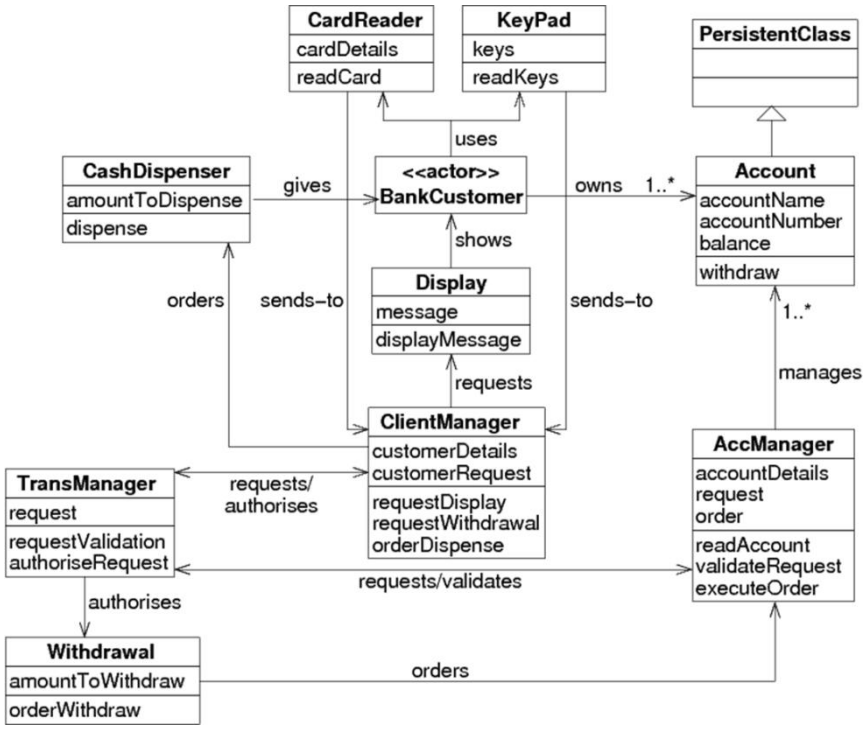
for all use case realisations ...

... so the complete system class diagram is the aggregation of

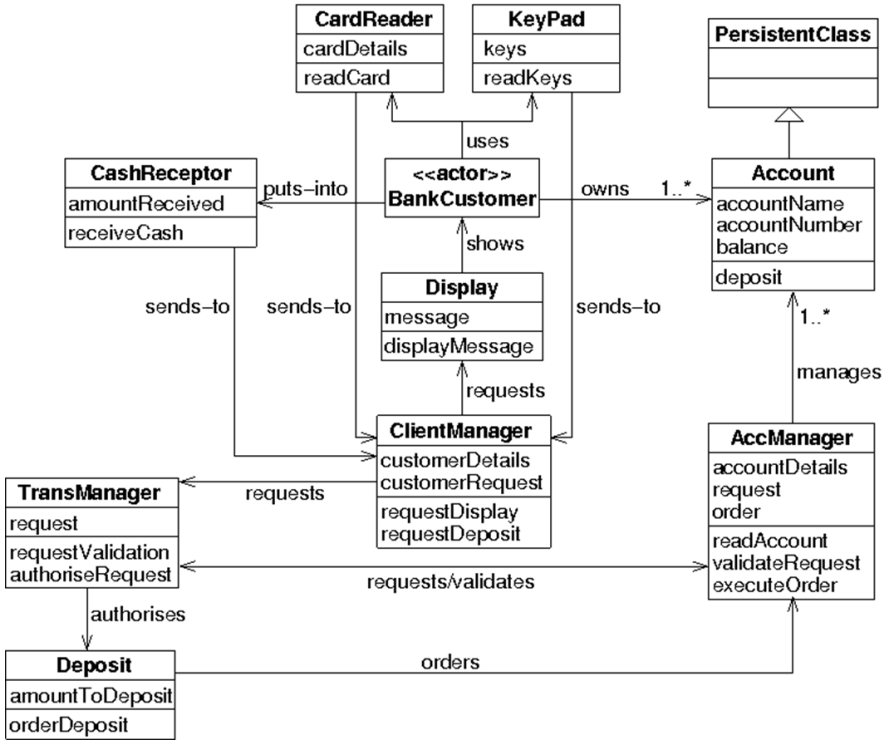
- system classes
- their relationships

for all use case realisations.

ATM: System Classes

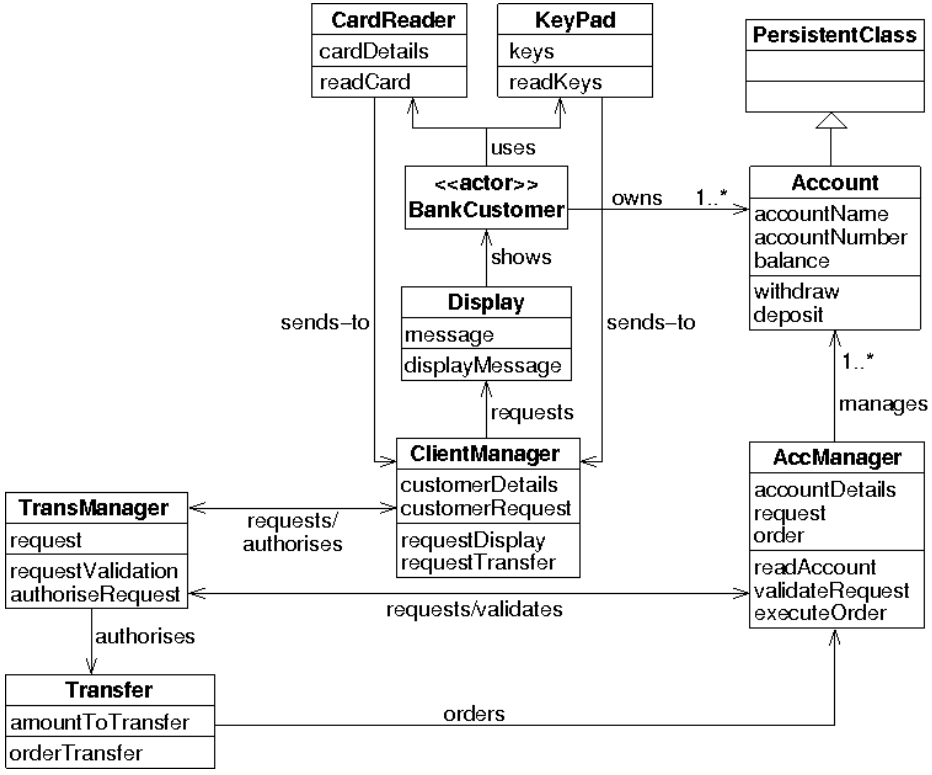


System classes for realising **Withdraw Money**

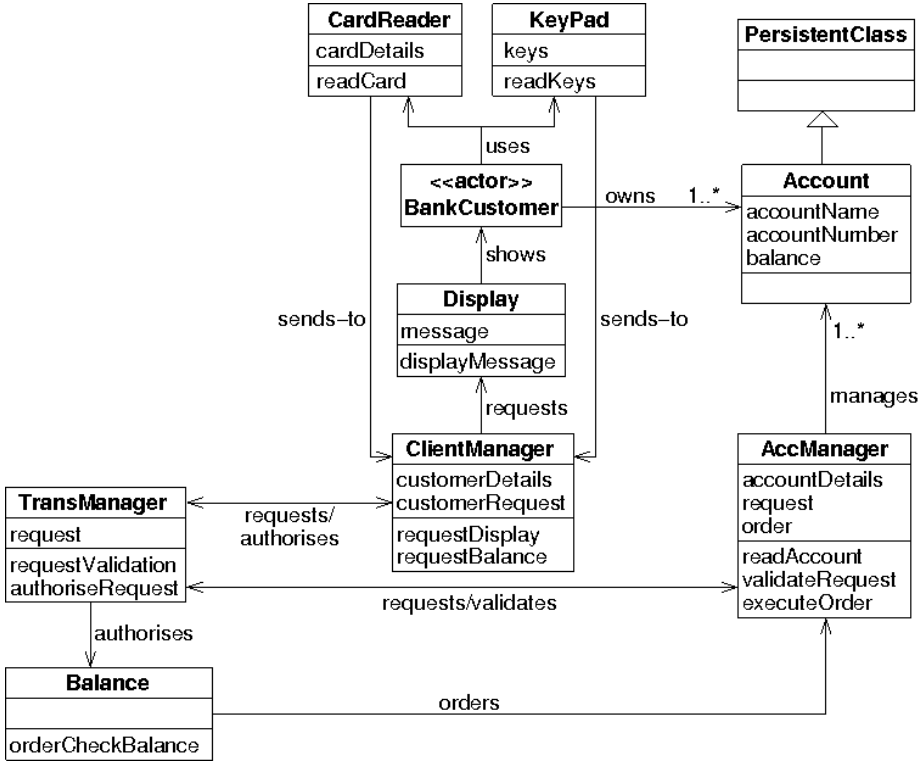


System classes for realising **Deposit Money**

ATM: System Classes



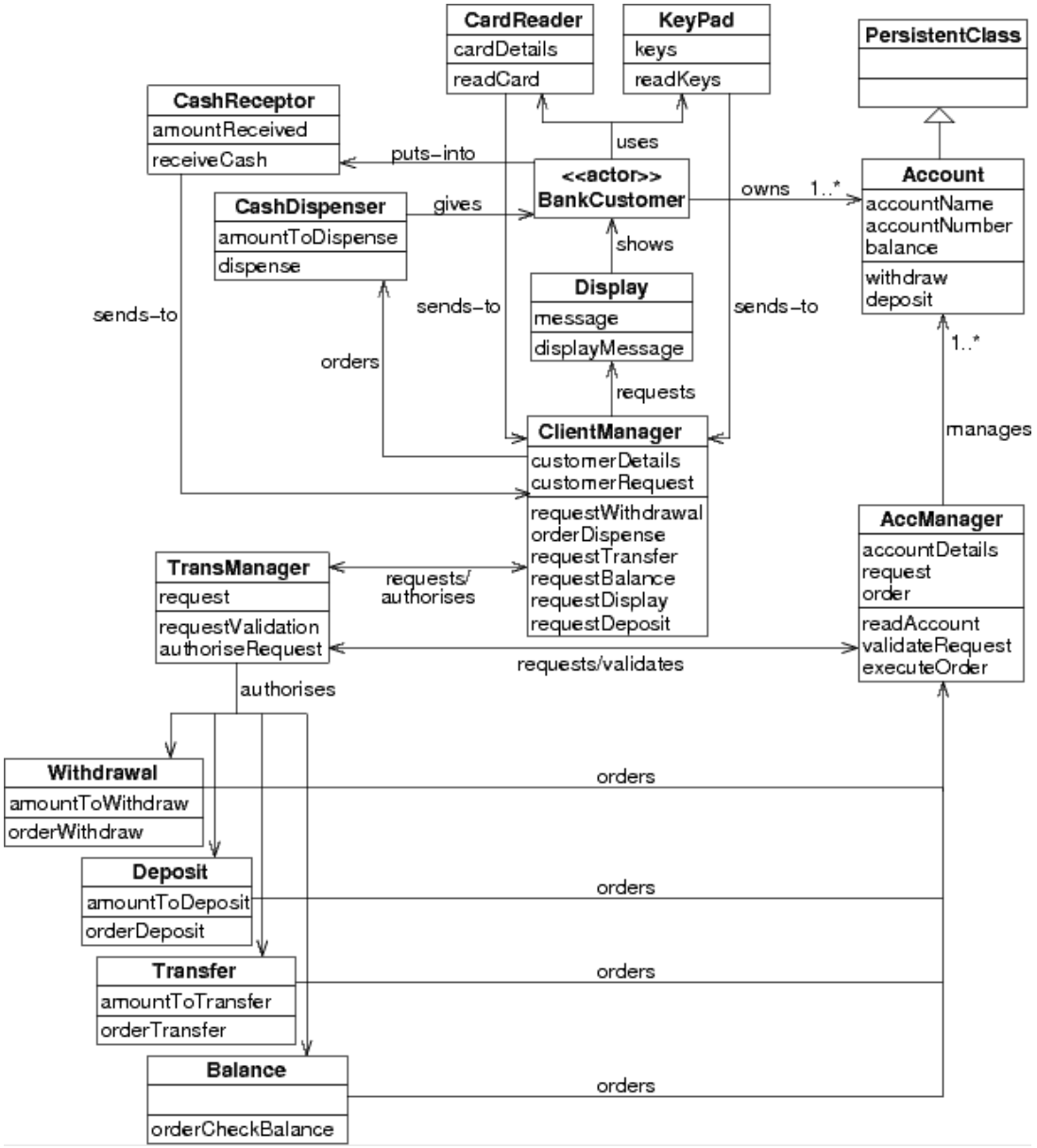
System classes for realising **Transfer Money**



System classes for realising **Check Balance**

ATM: The Complete System Class Diagram

Complete system class diagram (for all use case realisations)



Guidelines for Good Design

High cohesion

ensure that a class represents a single well-defined entity, e.g. Bus or Driver but not BusAndDriver

Low coupling

ensure that a class interacts with as few other classes as reasonably possible

Some coupling is essential (and therefore unavoidable) as the classes in a program must work together – the trick is to avoid spaghetti-type links between classes.

Some Techniques: (Re)Factoring

Create modules that account for similarities and differences between units of interest

Create new classes

- Generalization
- Aggregation
- Composition

Abstraction

Refinement

Some Techniques: Partitions and Collaborations

- Create smaller units (subsystems or partitions) or larger units
- Group units that collaborate
- Units or partitions may collaborate
- Objects in the same partition have high cohesion

Some Techniques: Layers

Organise system (classes) into layers

Separate application logic from user interface logic

Adopt Model-View-Controller (MVC) architecture

Typical layers:

- Foundation
- Problem domain
- Data Management
- Human-computer interaction
- Physical architecture

Some Techniques: Design Patterns

Design patterns are good designs that have been accumulated over many years of experience

A design pattern is a reusable design that can be customised to many recurring problems

Not really for a first course on Software Engineering …

… see next semester

Summary

Software design, which starts with system classes (or design classes), is probably the hardest skill, as well as the most important, in Software Engineering

The key skill is to find the right system classes and assign responsibilities to them (so that use case realisations are covered)

There are principles and guidelines for good design, but design is hard to teach/learn except by practice and experience

In the System Design phase of the development process, design gets closer to code, and design guidelines and techniques become even more important

MANCHESTER 1824

The University of Manchester

Workshop 3: Structural Modelling for HTV

- Identify domain classes
- Draw class diagrams for domain classes
- Refine domain classes into system classes
- Draw class diagrams for system classes

Bring:

- Laptops
  - For working
- USB sticks
  - For submission (feedback on Moodle later)
- Use case diagrams for HTV