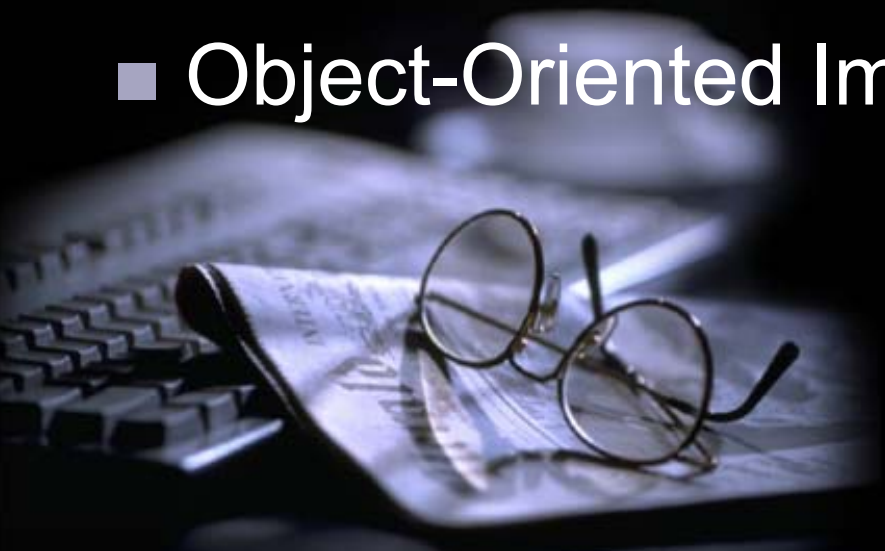
Object-Oriented Analysis and Design Methodology

Ir. I Gede Made Karma, MT
Email : igmkarma@yahoo.com

Contents

- An Introduction to the Object-Orientation
- An Introduction to the Object-Oriented Methodology
- Object-Oriented Notation Guide
- Object-Oriented Analysis and Design
- Object-Oriented Implementation



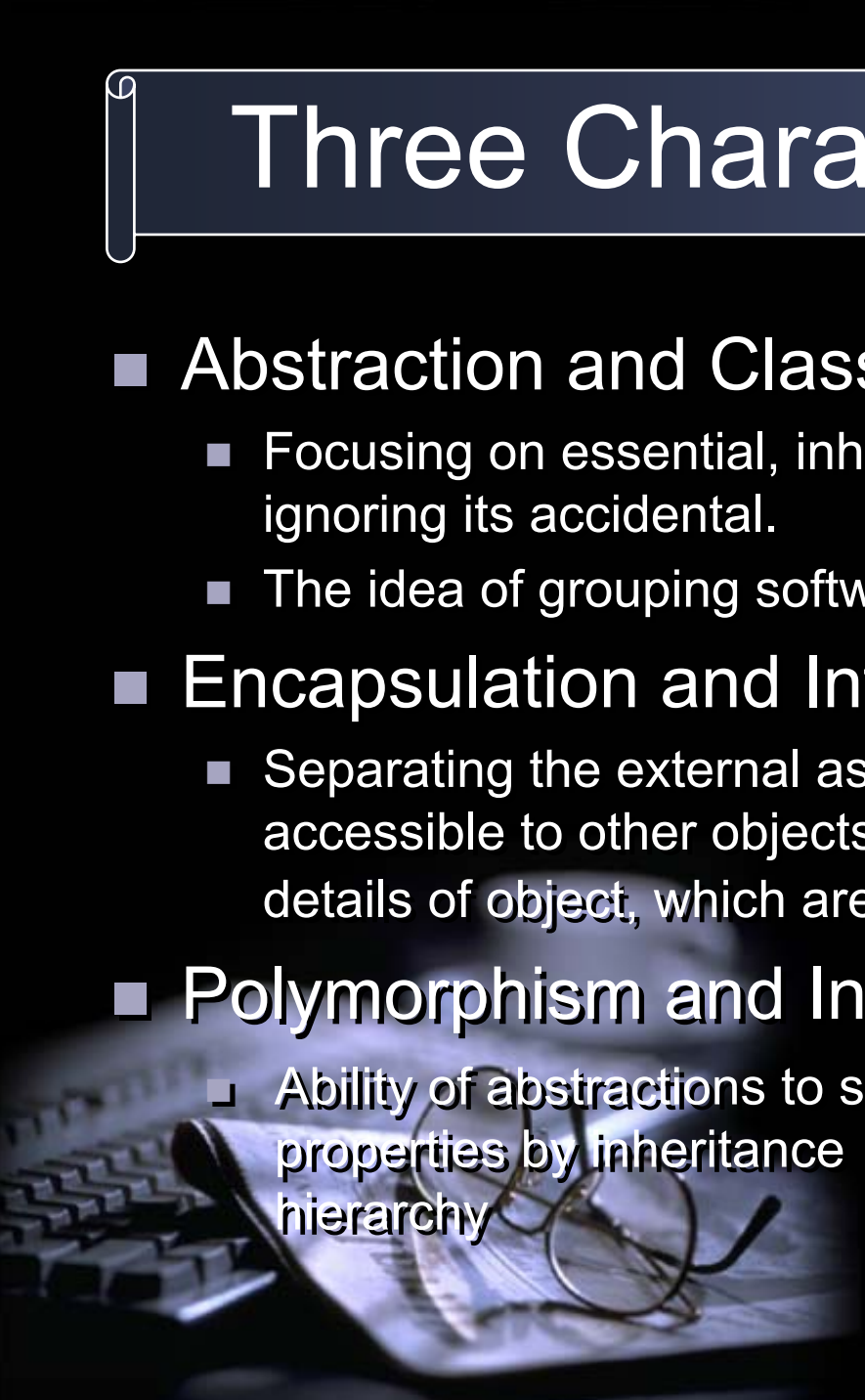
An Introduction to the Object-Orientation

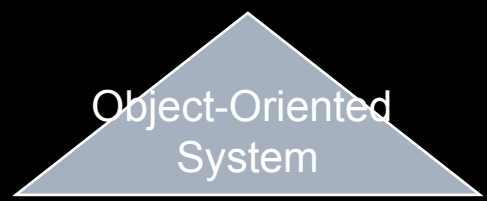
What is Object-Orientation

- A new technology based on objects and classes
- A way of thingking to organizing software as a collection of discrete objects that incorporate both data structure and behaviour
- An abstraction of the real world based on objects and their interactions with other objects

Three Characteristics of OO

- Abstraction and Classification :
  - Focusing on essential, inherent aspects of an entity and ignoring its accidental.
  - The idea of grouping software ideas into classes of things
- Encapsulation and Information Hiding :
  - Separating the external aspects of an object, which are accessible to other objects, from the internal implementation details of object, which are hidden from other objects
- Polymorphism and Inheritance :
  - Ability of abstractions to share properties by inheritance hierarchy





Object and Classes

- Object
  - An object is a thing or concept. It can be a real-world thing or concept, or an abstraction of a thing or concept expressed as a software representation.
  - An object has state (attributes) and behavior (method)
  - Individual objects, also called instances, have identity and are distinct things, and can be distinguished from other objects.
- Classes
  - A class is a description of a collection of objects with common attributes and behavior.
  - In practice, the definition or specification of a class includes the definitions of the attributes comprising the state, the methods implementing the behavior, and how to handle creation and destruction of an object.

An Introduction to the Object-Oriented Methodology

What Are Analysis and Design For

- Testing a physical entity before building system
- Communicating with Customers
- Visualization
- Reduction of Complexity

Various Type of Methodologies

- Shlaer/Mellor Method [Shlaer-1988]
- Coad/Yourdon Method [Coad-1991]
- Booch Method [Booch-1991]
- OMT Method [Rumbaugh-1991]
- Wirfs-Brock Method [Wirfs-Brock-1990]
- OOSE Objectory Method [Jacobson-1992]
- UML (Unified Modeling Language) [UML-1997]

Development Process

Object-Oriented **Analysis**

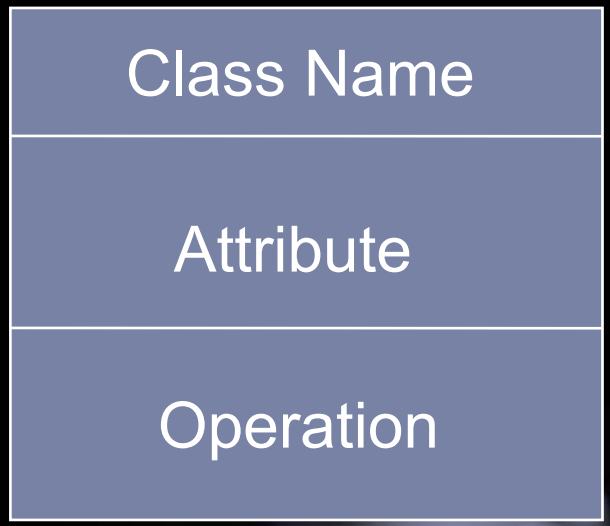
Object-Oriented **Design**

Object-Oriented **Implementation**

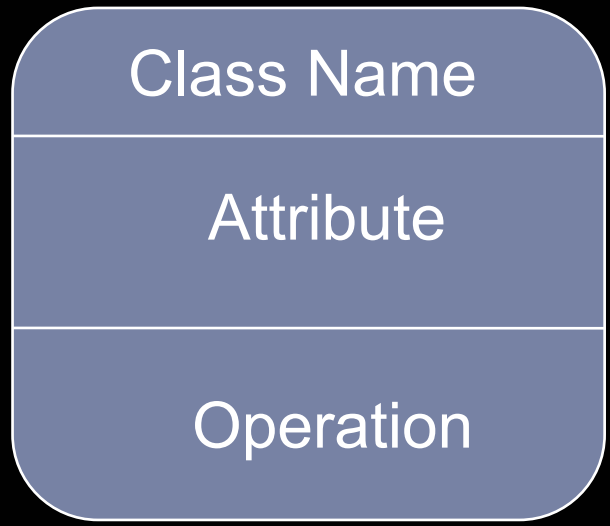
Object-Oriented Notation Guide

Class and Object

- Class

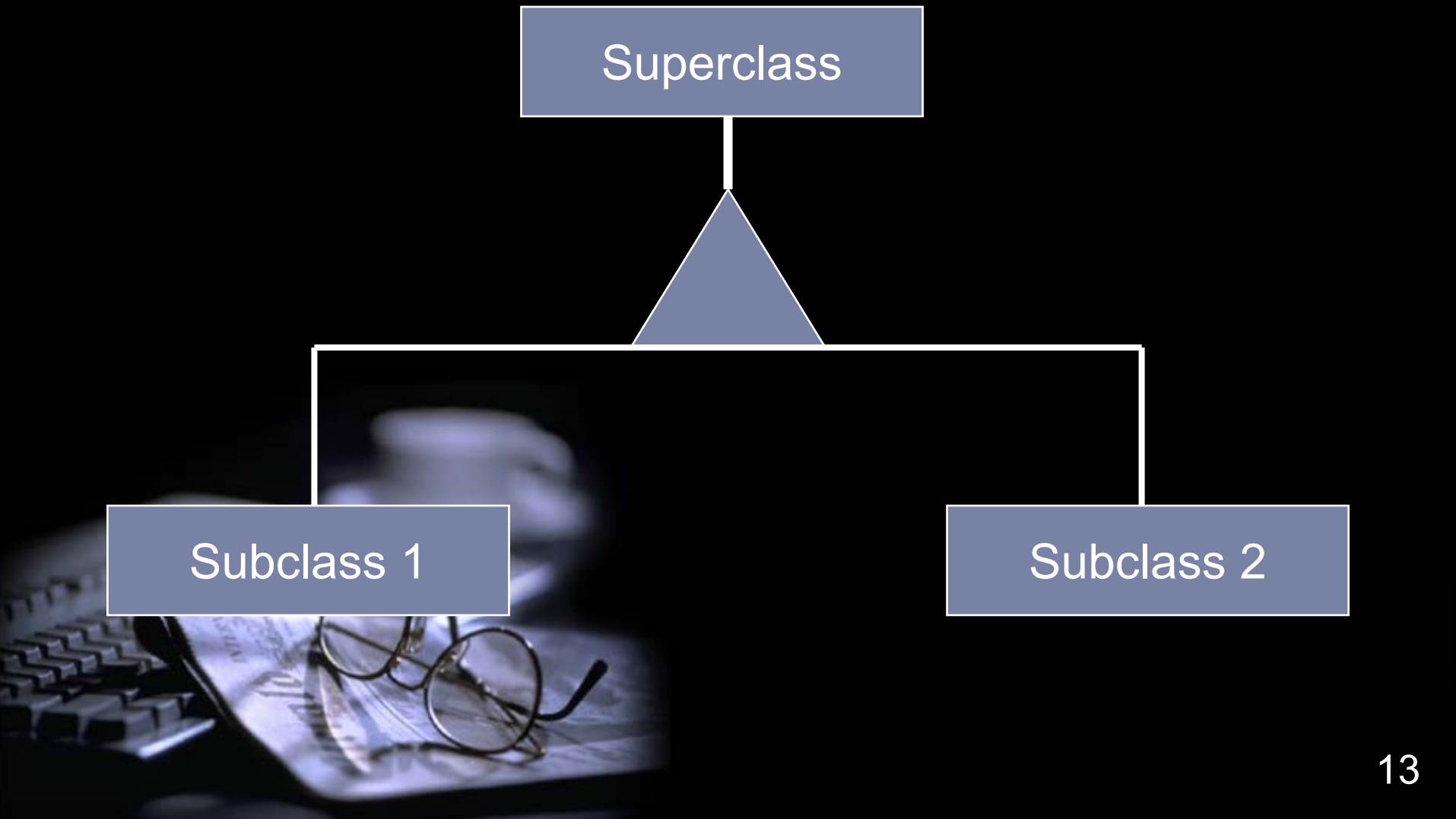


- Object Instances



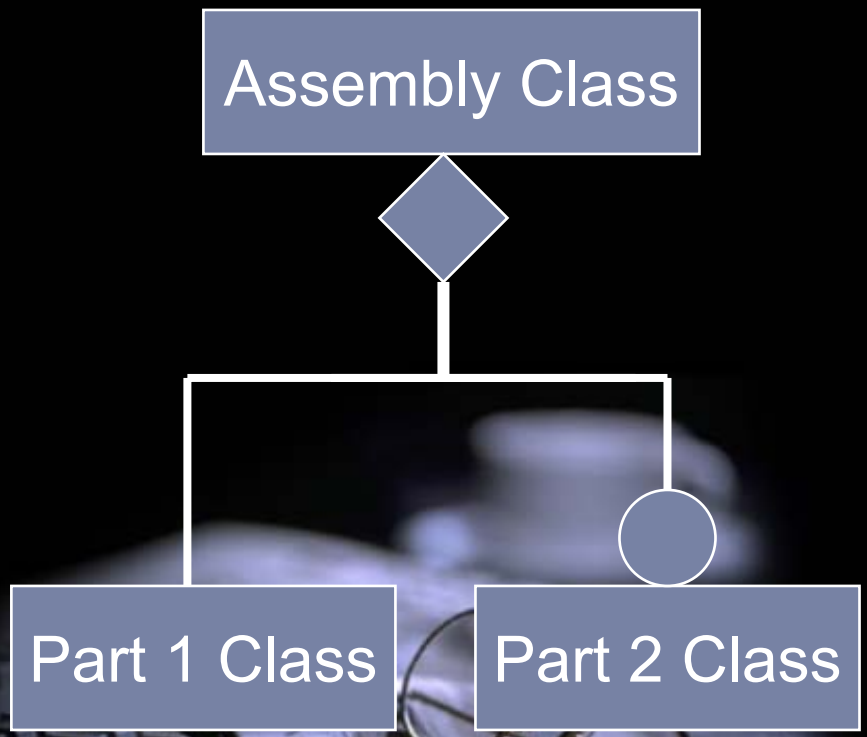
- Instantiation Relationship

Generalization and Inheritance

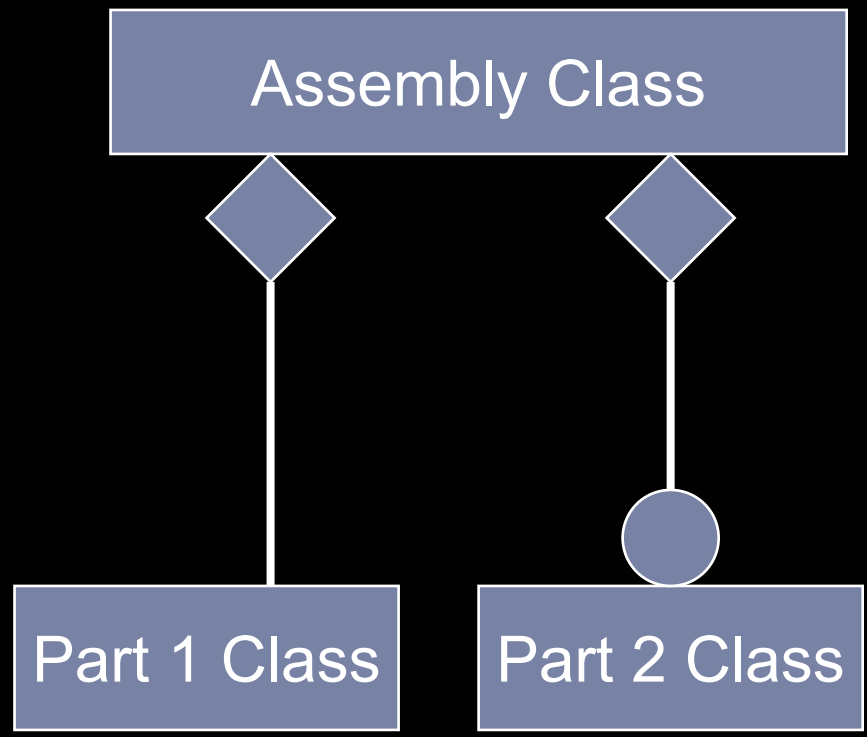


Aggregation

- Aggregation 1



- Aggregation 2



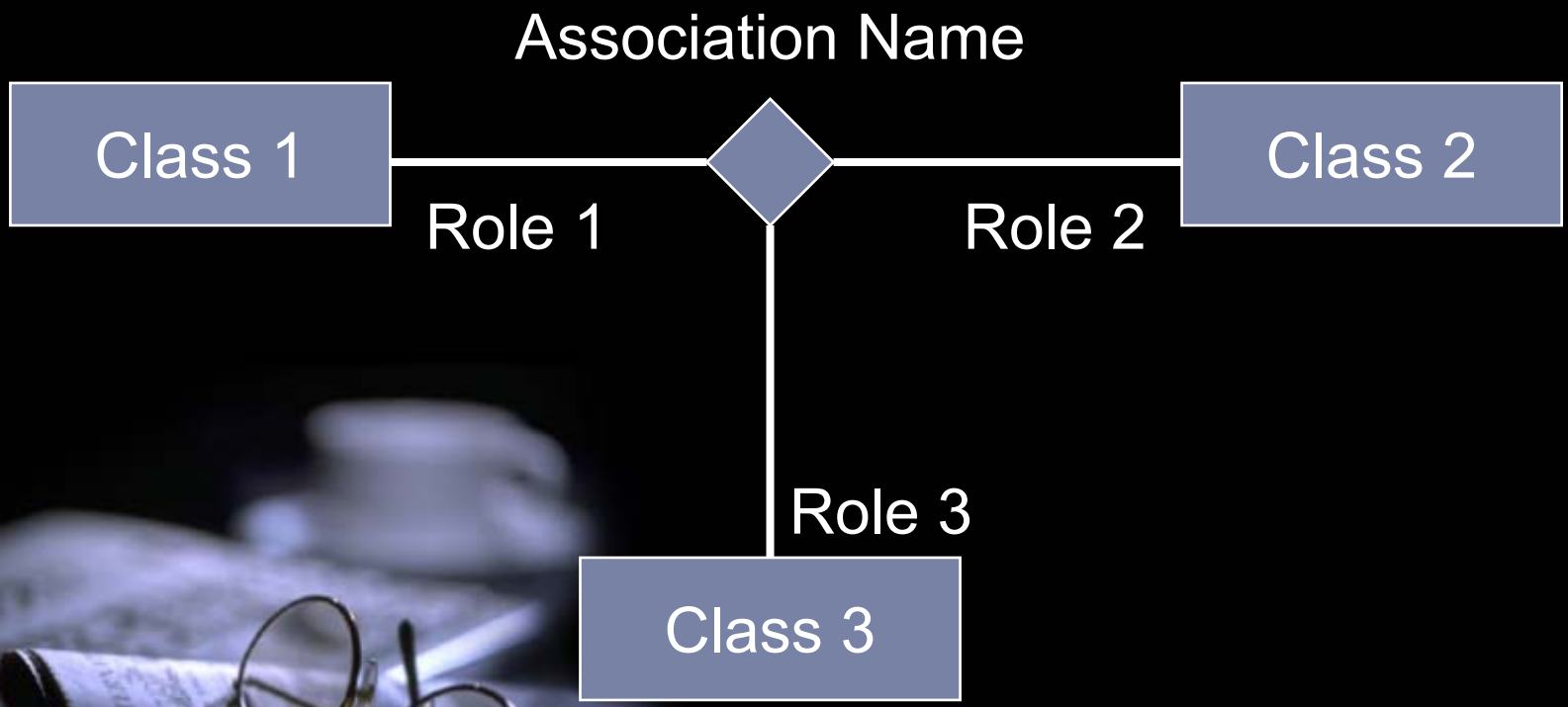
Association

- Association

- Qualified Association

- Multiplicity of Associations

Ternary Association



Object-Oriented Analysis and Design

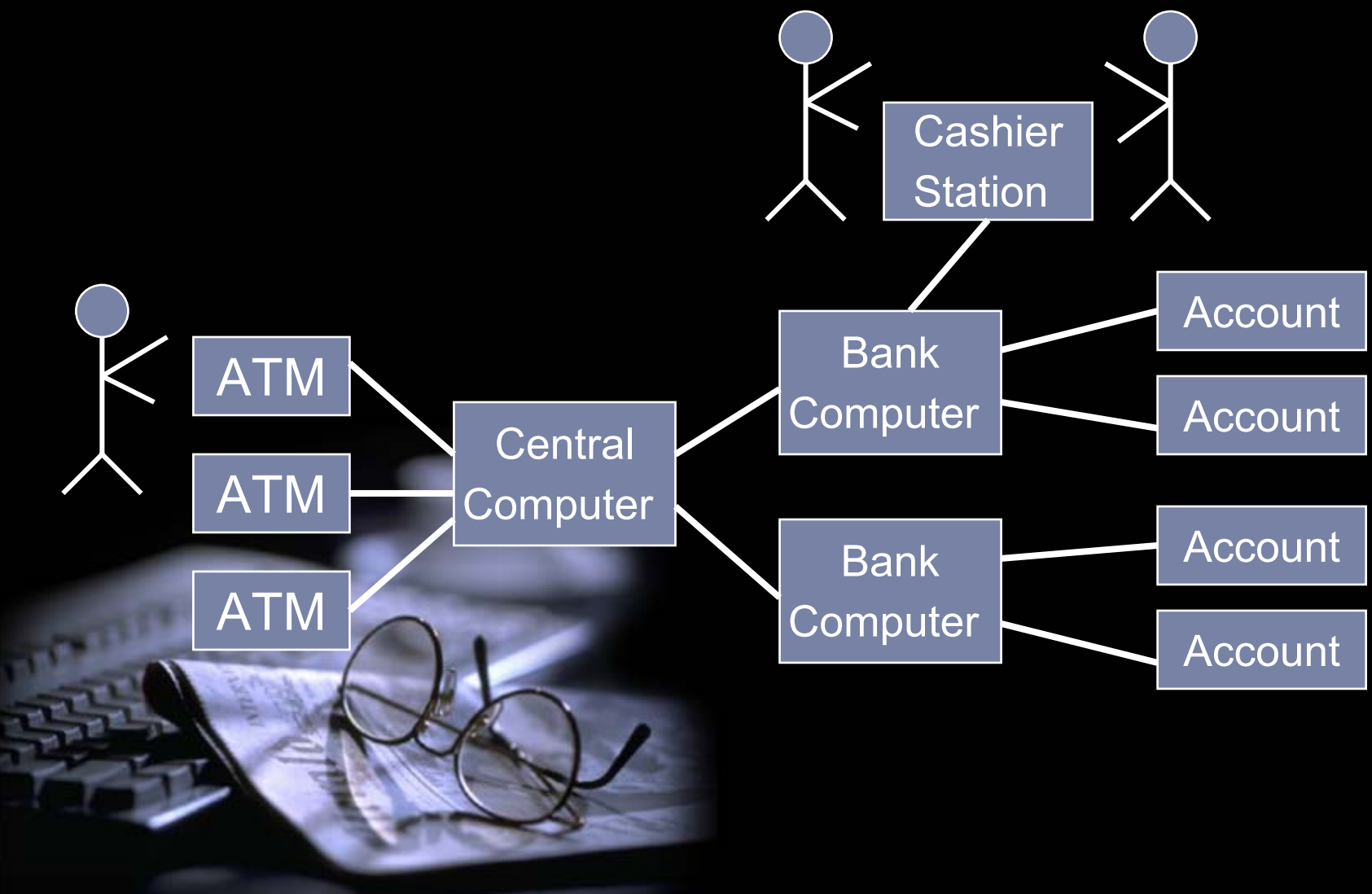
Analysis and Design Process

- Problem Statement
- System Architecture
- Object Modeling
  - Identifying Object Classes
  - Preparing a Data Dictionary for Classes
  - Identifying Associations
  - Identifying Attributes
  - Refining with Inheritance
  - Grouping Classes into Modules
- Dynamic Modeling
- Functional Modeling

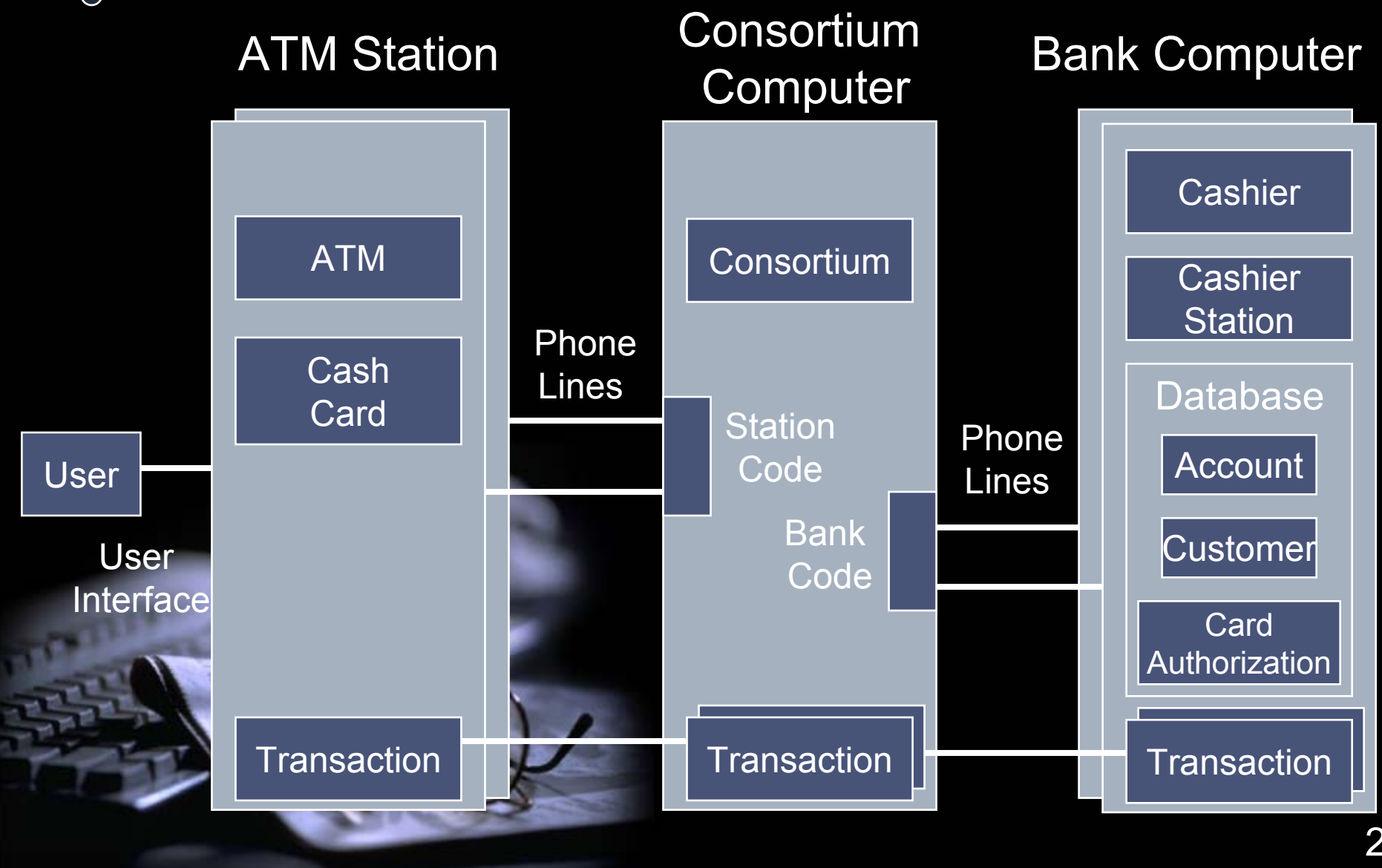
Problem Statement

- Requirements Statement
  - Problem Scope
  - What is needed
  - Application Context
  - Assumptions
  - Performance Needs

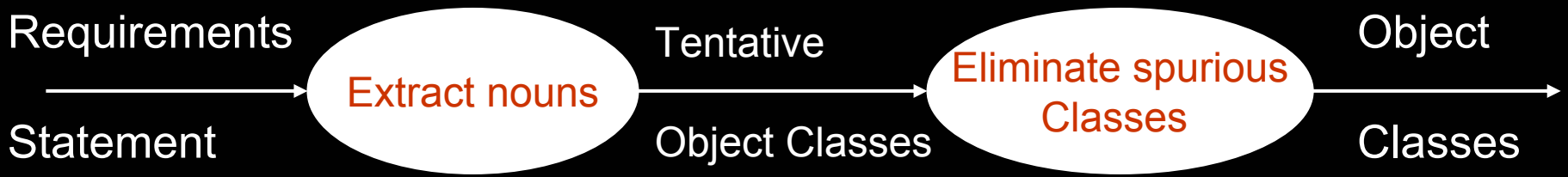
Example : ATM Network

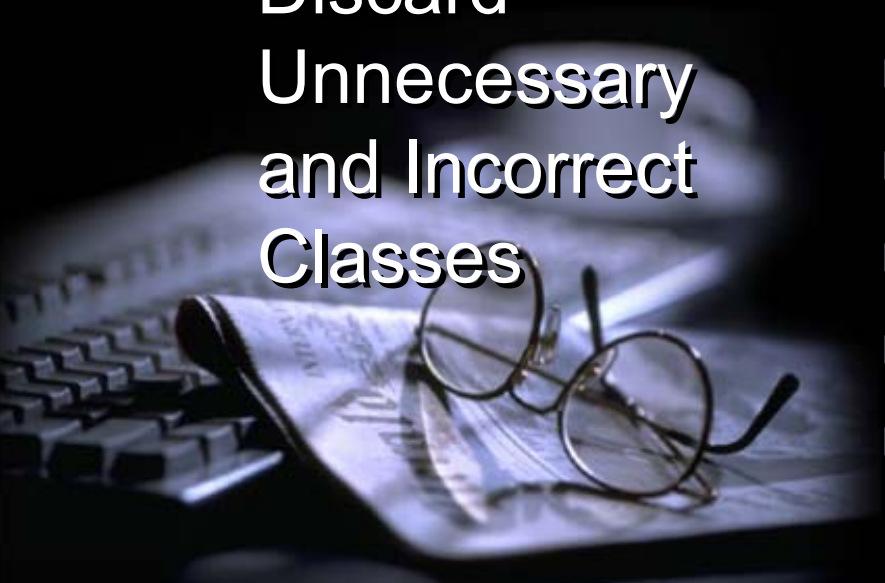


System Architecture



Identifying Object Classes





Discard Unnecessary and Incorrect Classes

- Redundant classes
- Irrelevant classes
- Vague classes
- Attributes
- Operations
- Roles
- Implementation constructs

Example: IOC for ATM Network

Bad Classes

Vague

System	Banking Network
Security Provision	Record Keeping Provision

Attribute

Receipt	Cash
Transaction Data	Account Data

Implementation

Access	Software
Transaction Log	Comm Line

Redundant

User

Irrelevant

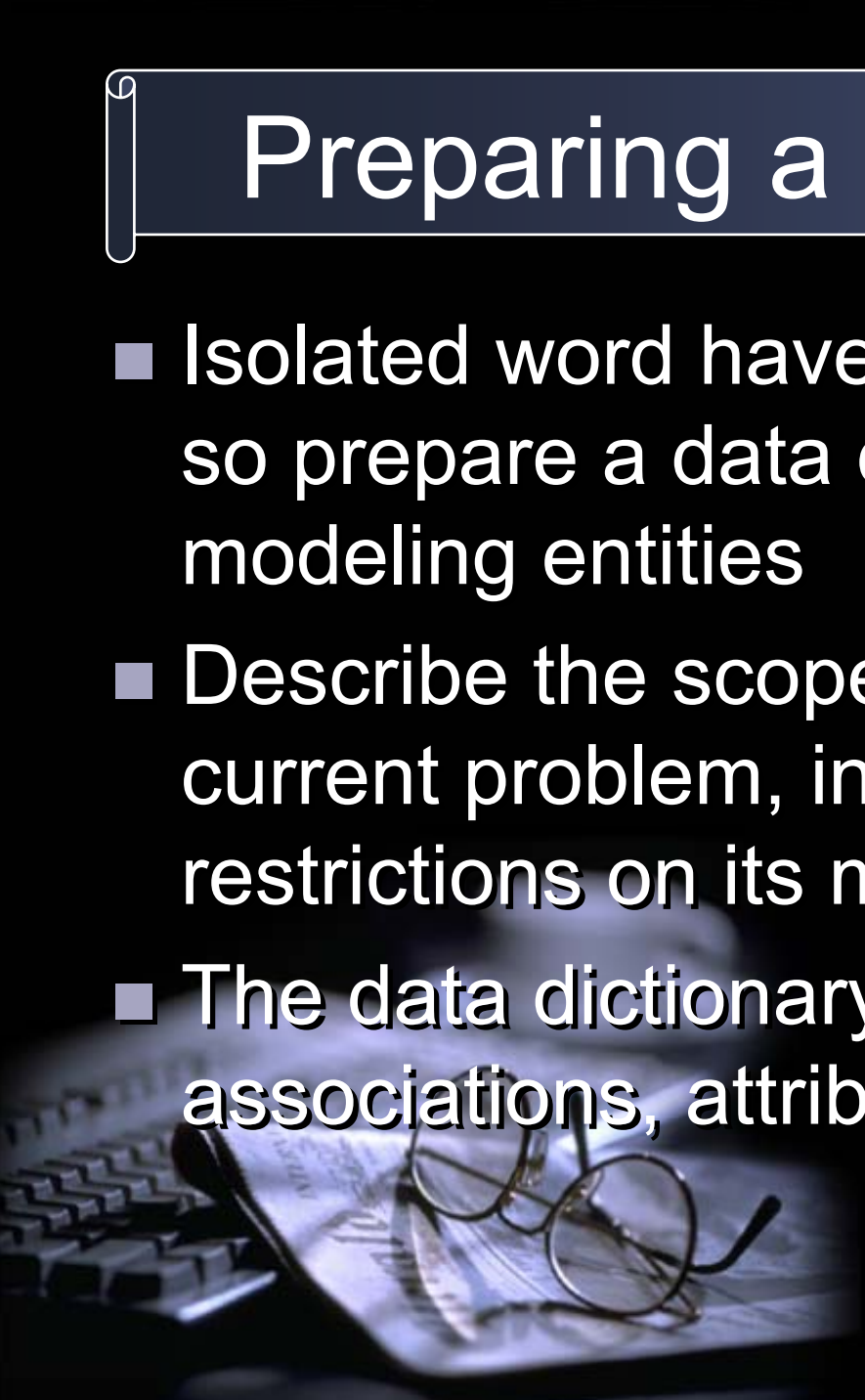
Cost

Good Classes

Account	ATM	Bank	Consortium	Customer	Cashier
Cashier Station	Central Computer	Bank Computer	Cash Card	Transaction	

Preparing a Data Dictionary

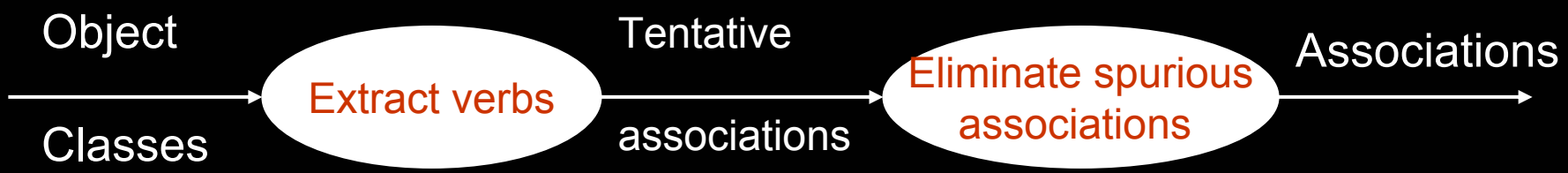
- Isolated word have many interpretations, so prepare a data dictionary for all modeling entities
- Describe the scope of the class within the current problem, including assumptions or restrictions on its membership or use
- The data dictionary also describes associations, attributes, and operation

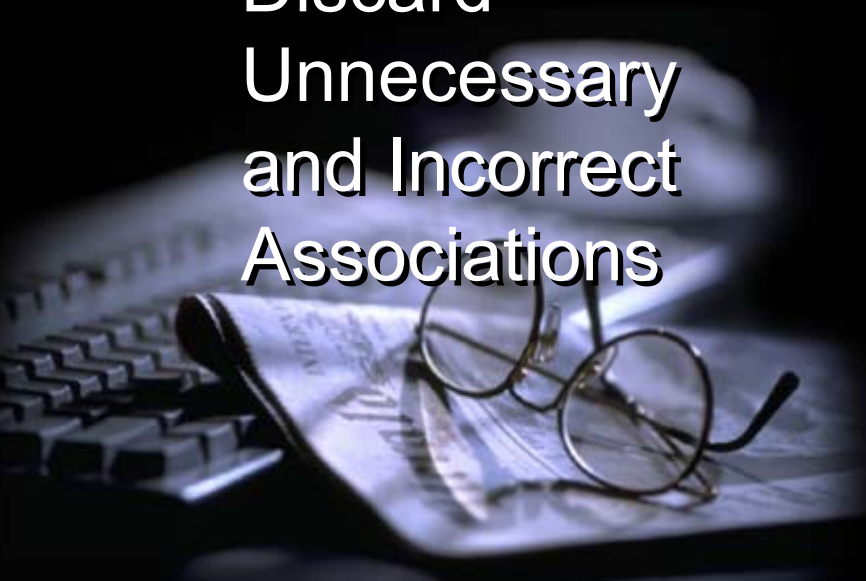


Example: DD for ATM Network

- *Account* : a single account in a bank against which transactions can be applied. Account may be of various types, at least checking or savings. A customer can hold more than one account.
- *Bank* : A financial institution that holds accounts for customers and that issues cash cards authorizing access to accounts over the ATM network.
- *ATM* : …
- *Bank Computer* : …
- *Cash Card* : …
- *Cashier* : ...
- etc.

Identifying Associations

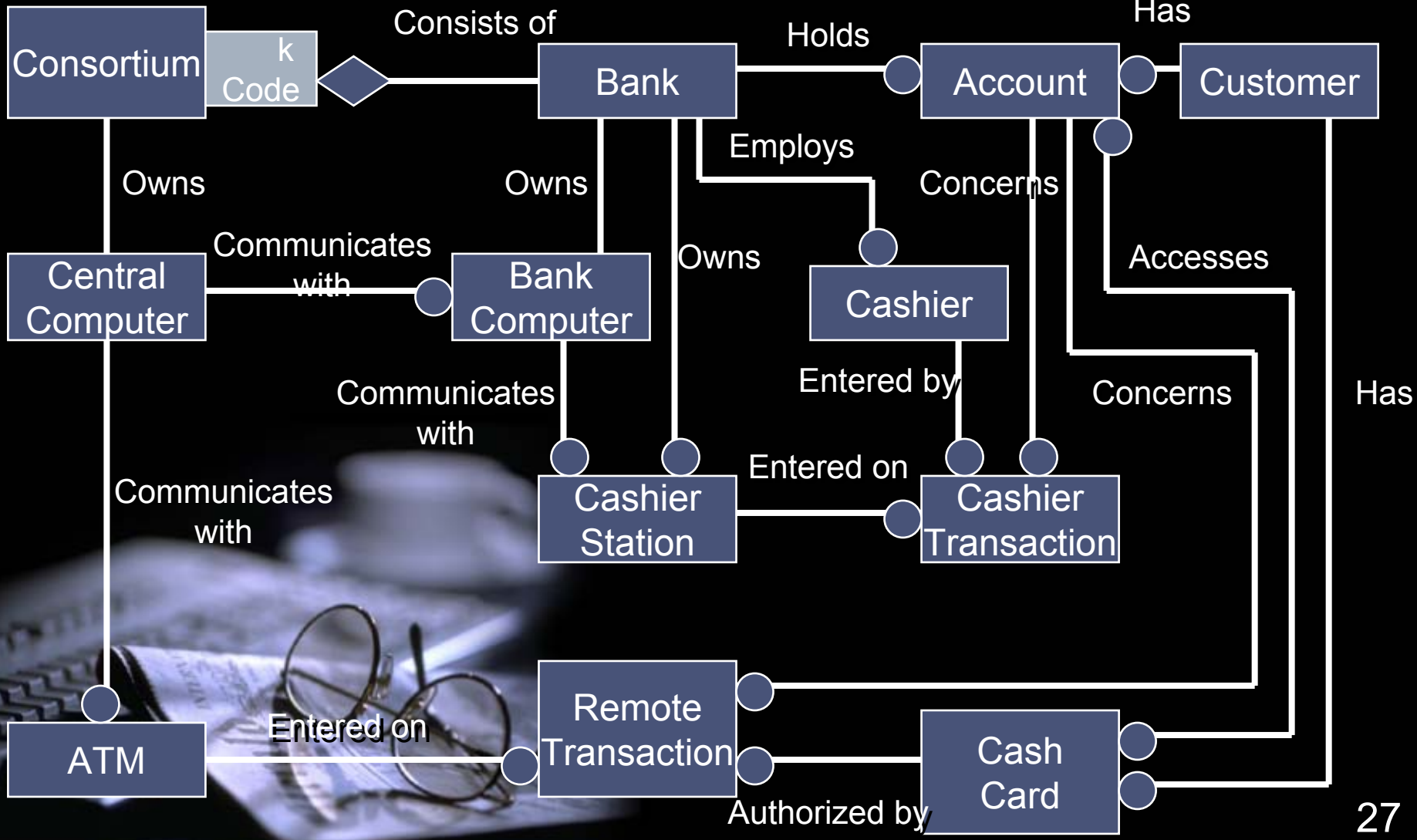




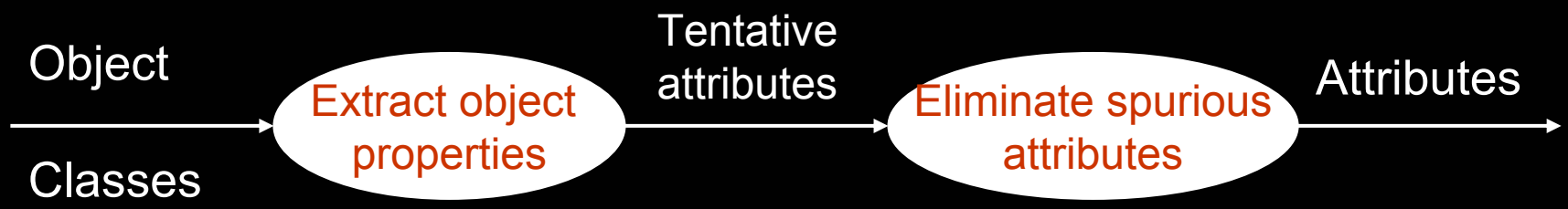
Discard Unnecessary and Incorrect Associations

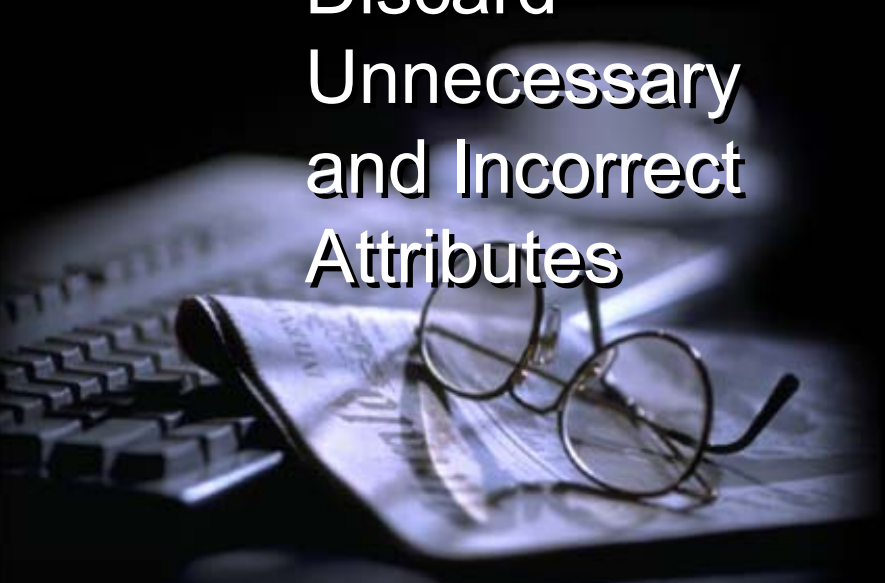
- Associations between eliminated classes
- Irrelevant or implementation associations
- Actions
- Ternary associations
- Derived associations
- Misnamed associations
- Multiplicity

Example: IAs for ATM Network



Identifying Attributes

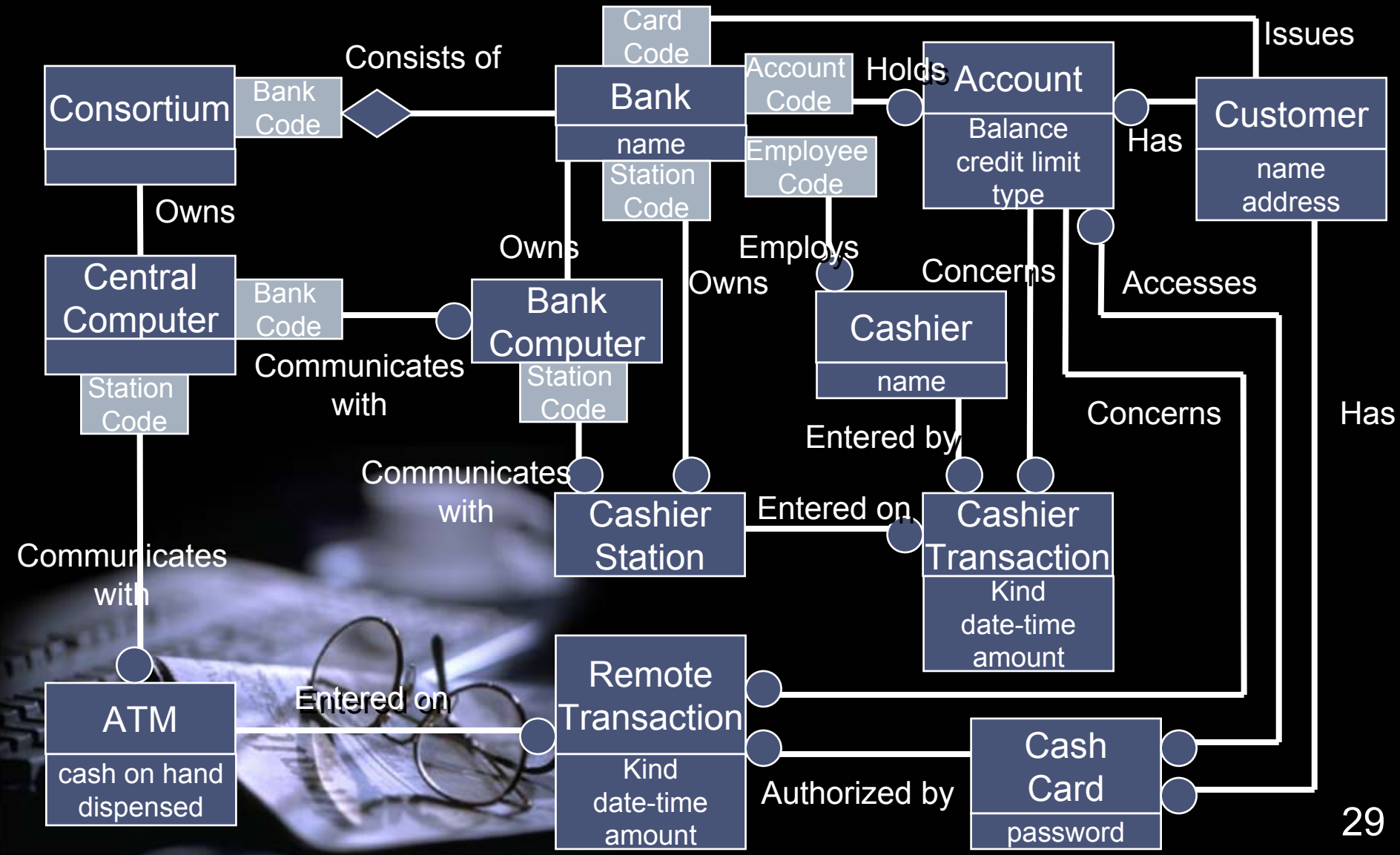




Discard Unnecessary and Incorrect Attributes

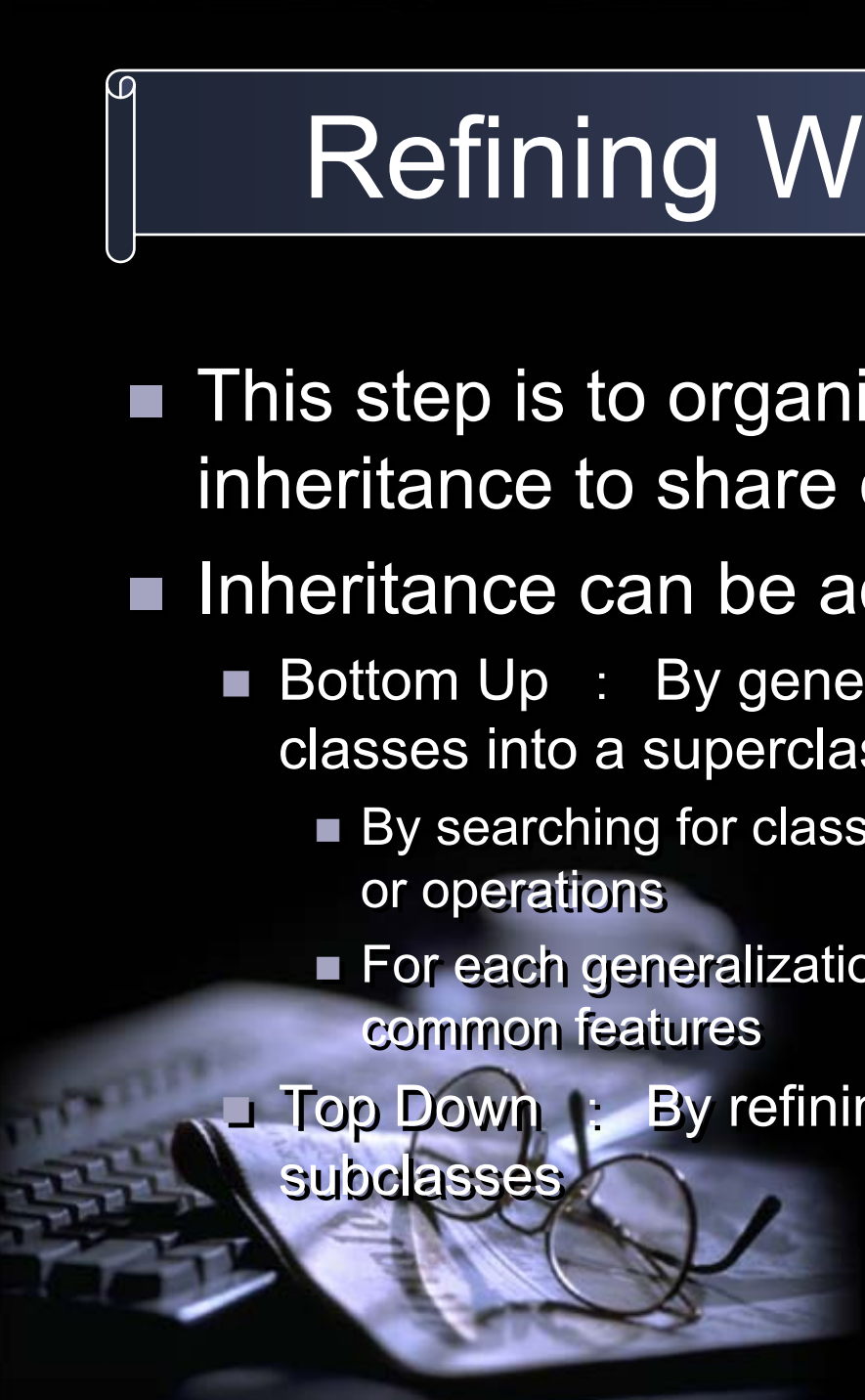
- Objects
- Qualifiers
- Names
- Identifiers
- Link attributes
- Internal values
- Fine detail
- Discordant attributes

Example: IAT for ATM Network

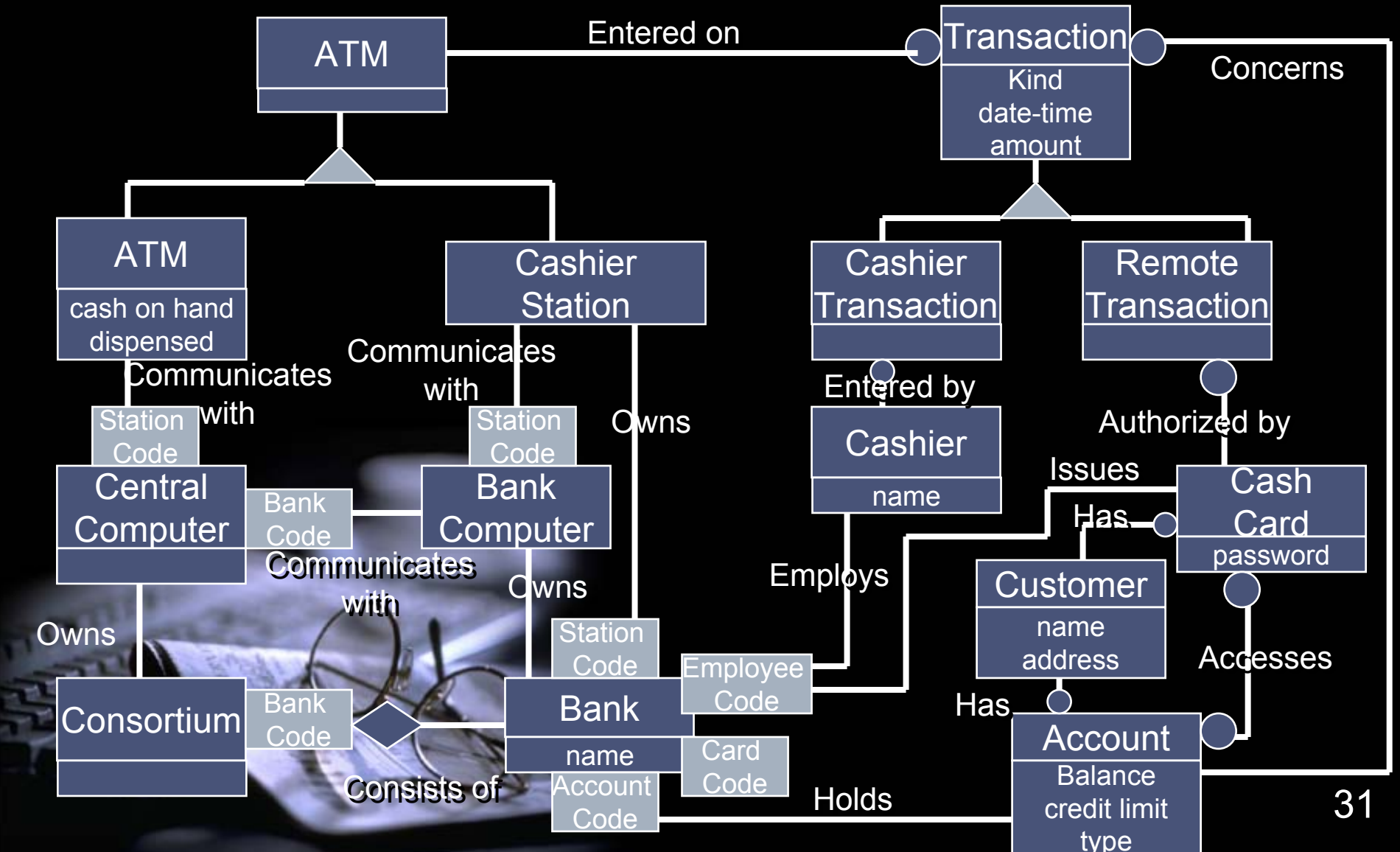


Refining With Inheritance

- This step is to organize classes by using inheritance to share common structure
- Inheritance can be added in two directions :
  - Bottom Up : By generalizing common aspect of existing classes into a superclasses
    - By searching for classes with similar attributes, associations, or operations
    - For each generalization, define a superclass to share common features
  - Top Down : By refining existing classes into specialized subclasses



Example: RWI for ATM Network



Grouping Classes into Modules

- A module is a set of classes that captures some logical subset of entire model
- For example: a model of computer operating system might contain modules for process control, device control, file maintenance, and memory management

Example: GCIM for ATM Network

- Tellers: Cashier, Entry Station, Cashier Station, ATM
- Account: Account, Cash Card, Card Authorization, Customer, Transaction, Update, Cashier Transaction, Remote Transaction
- Banks: Consortium, Bank

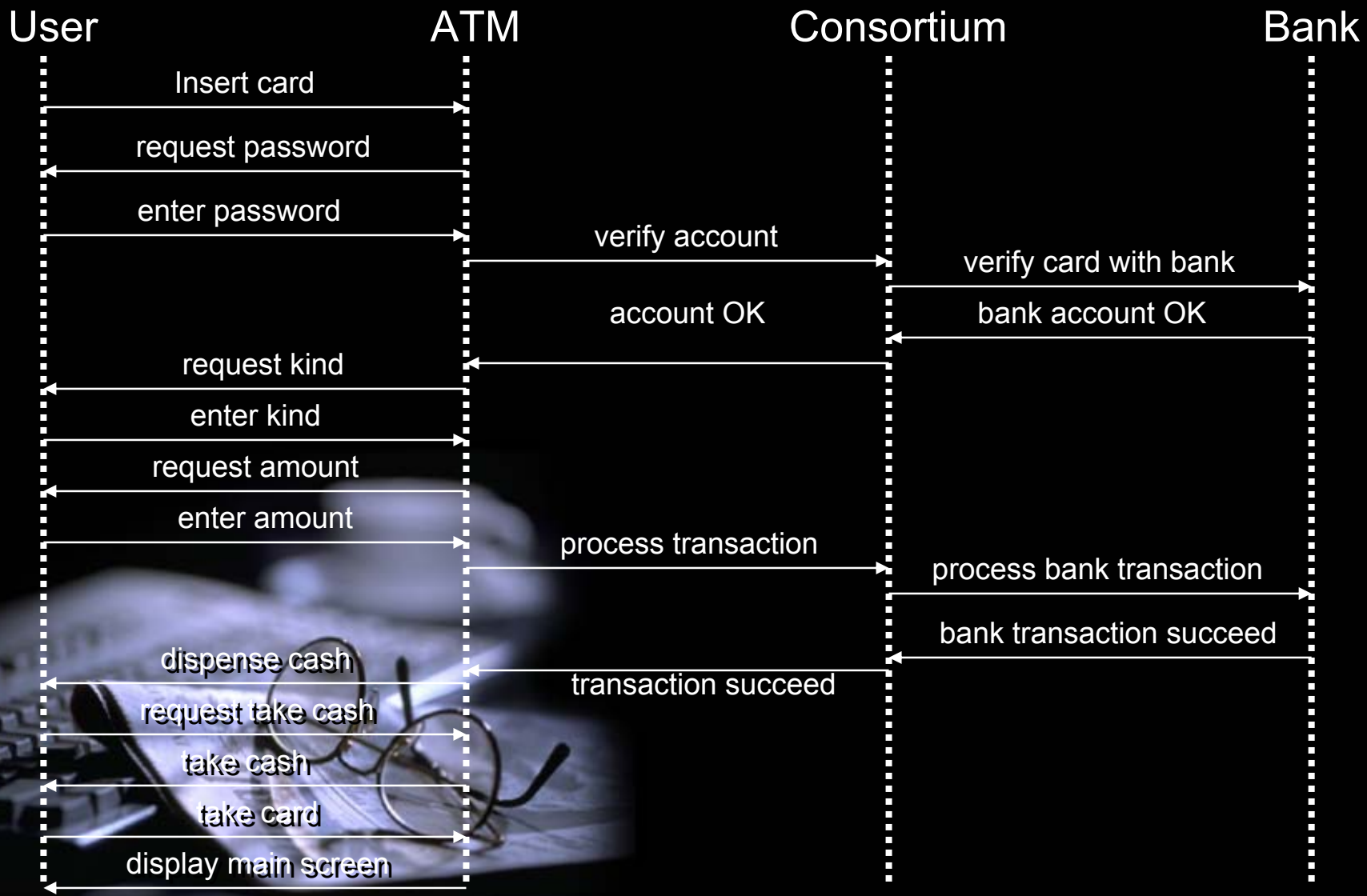
Dynamic Model

- The dynamic model shows the time-dependent behavior of the system and the objects in it.
- Begin dynamic analysis by looking for event, externally visible stimuli and responses.
- The dynamic model is important for interactive systems, but insignificant for purely static data repository, such as database.

Dynamic Model

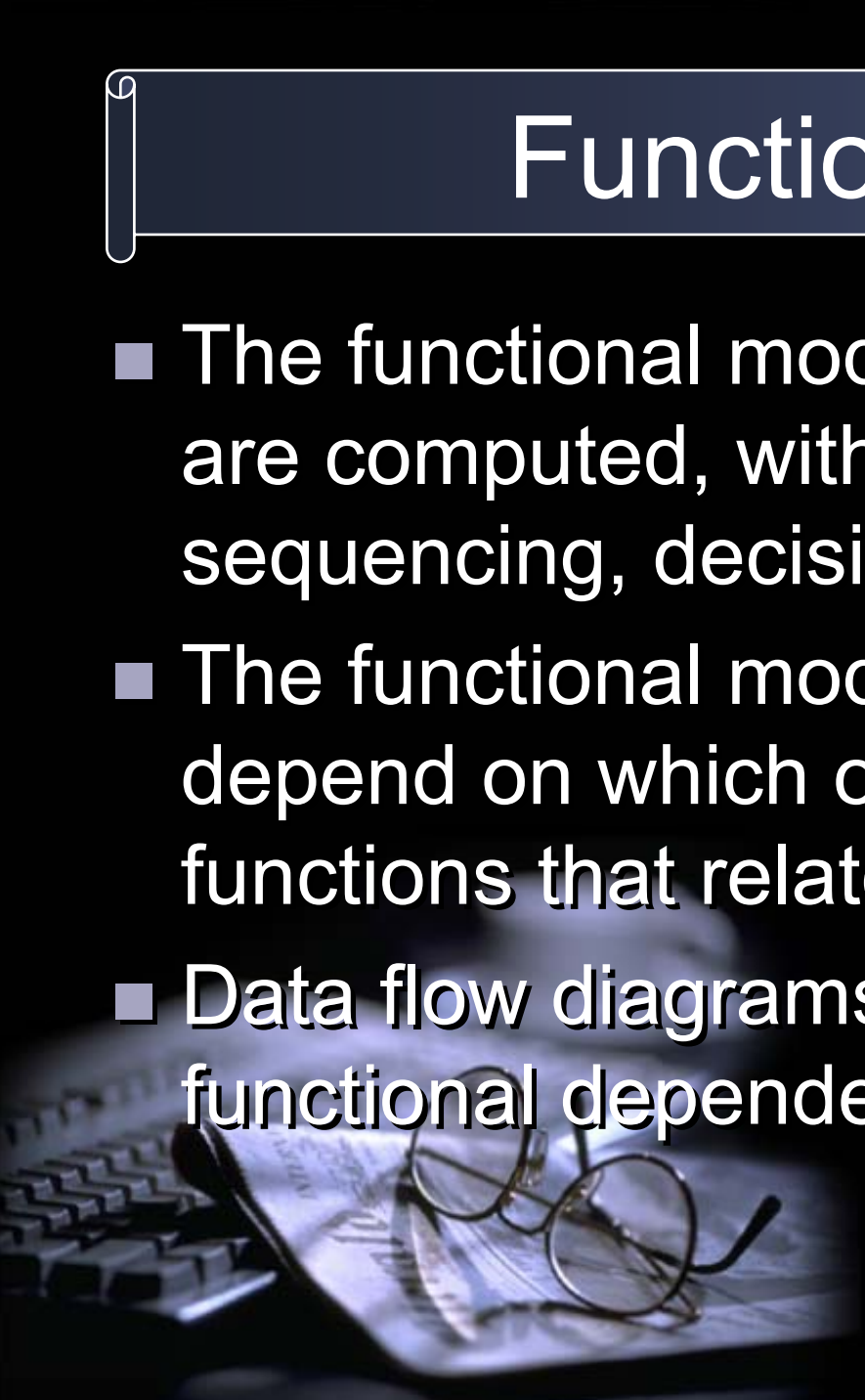
- The following steps are performed in constructing a dynamic model
  - Prepare scenarios of typical interaction sequences
  - Identify events between objects
  - Prepare an event trace for each scenario
  - Build a state diagram
  - Match events between objects to verify consistency

Example: DM for ATM Network

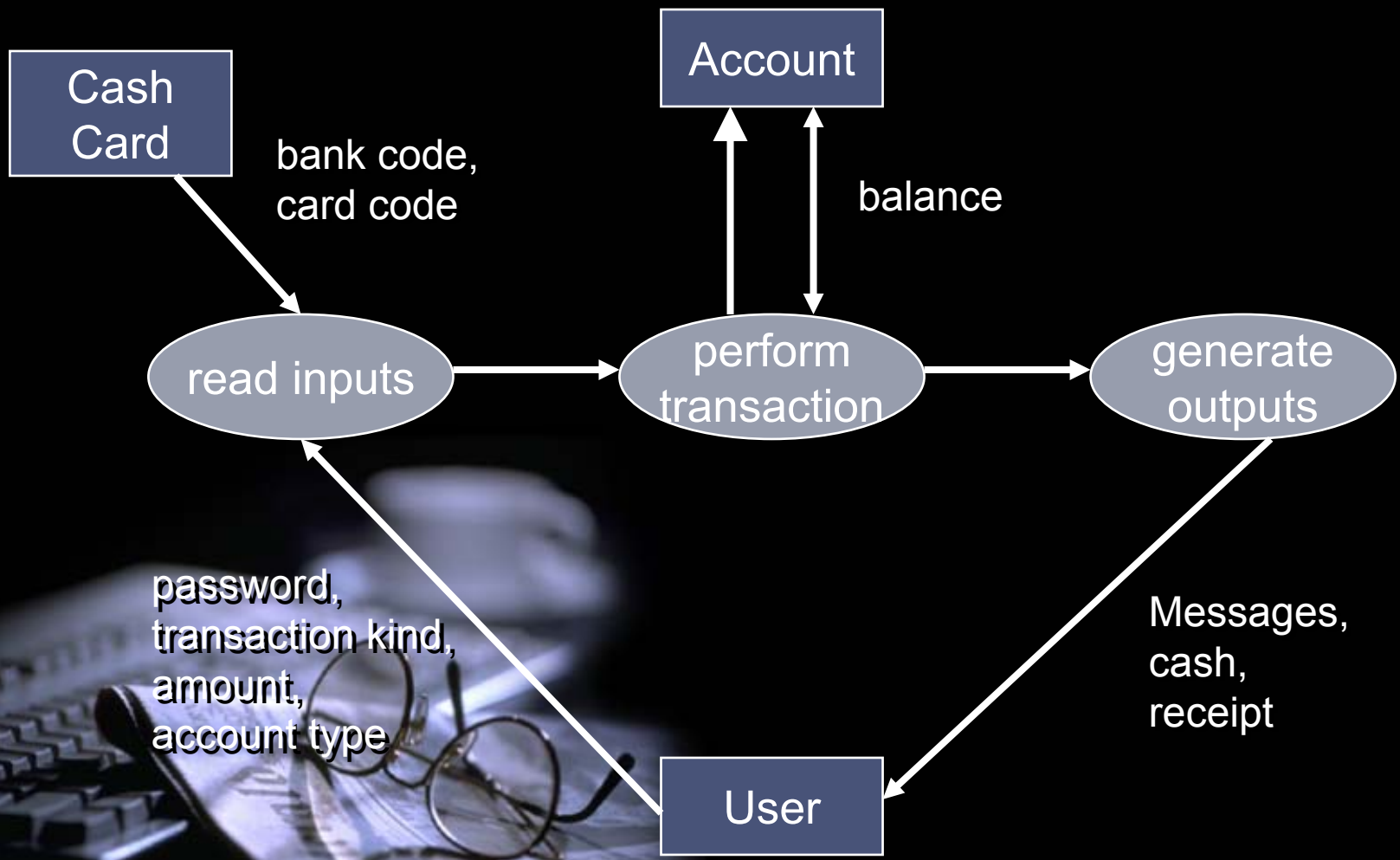


Functional Model

- The functional model shows how values are computed, without regard for sequencing, decisions, or object structure
- The functional model shows which values depend on which other values and the functions that relate them
- Data flow diagrams are useful for showing functional dependencies



Example: FM for ATM Network



Object-Oriented Implementation

Implementation Process

- Class Definition
- Creating Objects
- Calling Operations
- Using Inheritance
- Implementing Association

References -1-

- **[Booch-1991]** Grady Booch, *Object-Oriented Analysis and Design with Application*, Benjamin/Cummings, 1991.
- **[Booch-1999]** Grady Booch, James Rumbaugh, and Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
- **[Coad-1991]** Peter Coad and Edward Yourdon, *Object-Oriented Analysis*, Yourdon Press, 1991.
- **[Jacobson-1992]** Ivar Jacobson, Magnus Christerson, Patrik Jonson, and Gunnar Overgaard, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, 1992.

References -2-

- **[Jacobson-1999]** Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.
- **[Rumbaugh-1991]** James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenson, *Object-Oriented Modeling and Design*, Prentice Hall, 1991.
- **[Rumbaugh-1999]** James Rumbaugh, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.
- **[Shlaer-1988]** Sally Shlaer and Stephen J. Mellor, *Object-Oriented System Analysis: Modeling the World in Data*, Yourdon Press, 1988.

References -3-

- **[UML-1999]** *Unified Modeling Language Specification*, Object Management Group, www.omg.org, 1999.
- **[Wirfs-Brock-1990]** Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener, *Designing Object-Oriented Software*, Prentice Hall, 1990.