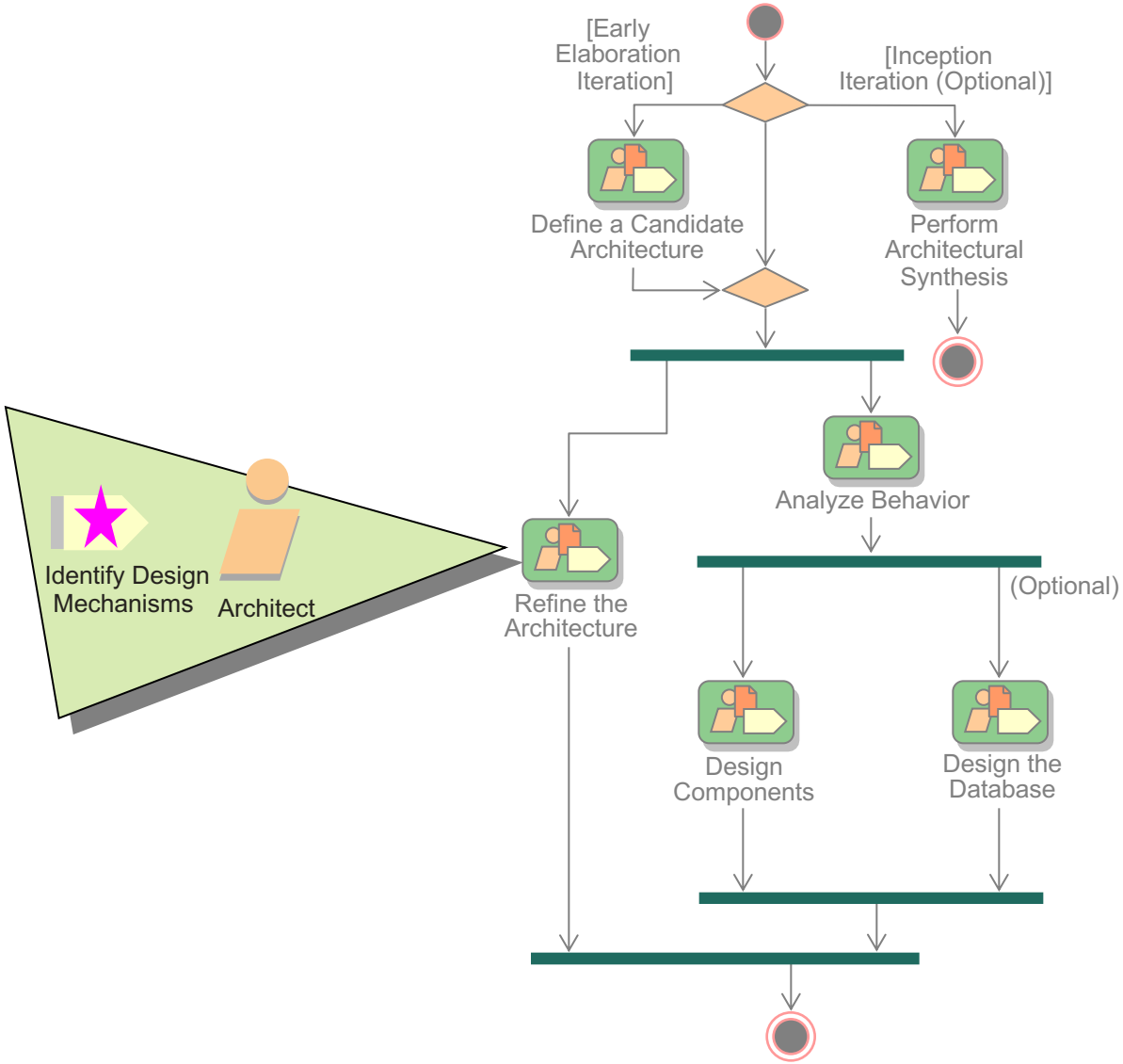
Software analysis and design

Module 12: Identify Design Mechanisms

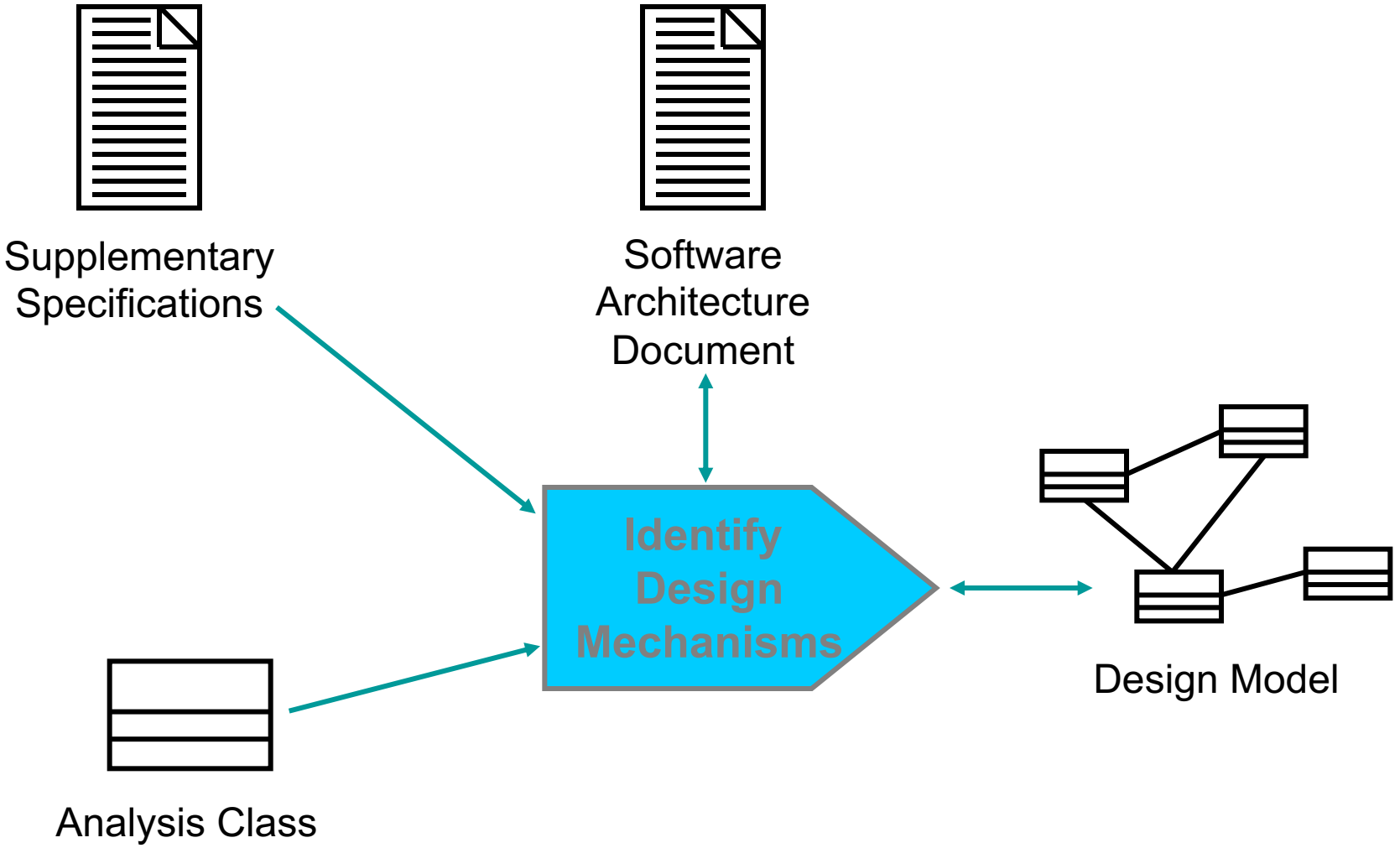
Objectives: Identify Design Mechanisms

- Define the purpose of the Identify Design Mechanisms activity and explain when in the lifecycle it is performed
- Explain what design and implementation mechanisms are and how they map from Analysis mechanisms
- Describe some key mechanisms that will be utilized in the case study

Identify Design Mechanisms in Context



Identify Design Mechanisms Overview



Identify Design Mechanisms: Steps

- Categorize clients of analysis mechanisms
- Document architectural mechanisms

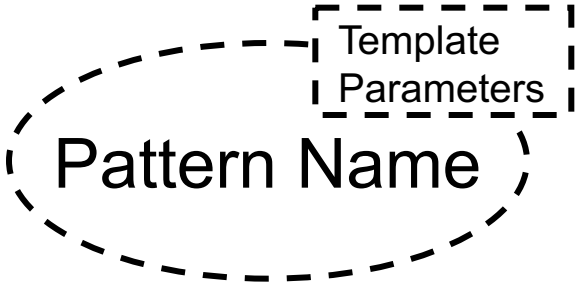
Review: Patterns and Frameworks

- Pattern
  - Provides a common solution to a common problem in a context
- Analysis/Design Pattern
  - Provides a solution to a narrowly scoped technical problem
  - Provides a fragment of a solution, or a piece of the puzzle
- Framework
  - Defines the general approach to solving the problem
  - Provides a skeletal solution, whose details may be analysis/design patterns

What Is a Design Pattern?

- A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context.

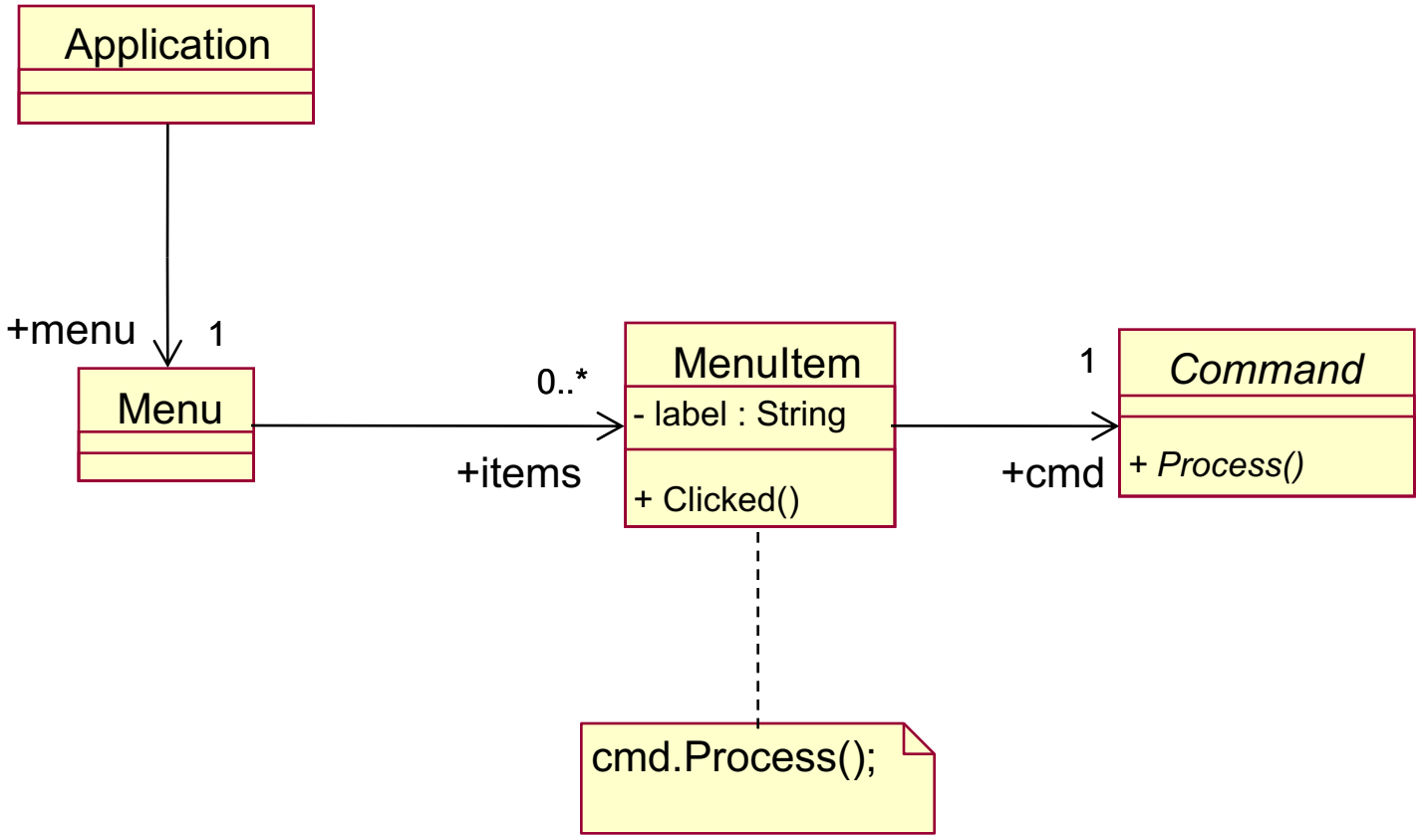
Erich Gamma et al. 1994. *Design Patterns—Elements of Reusable Object-Oriented Software*



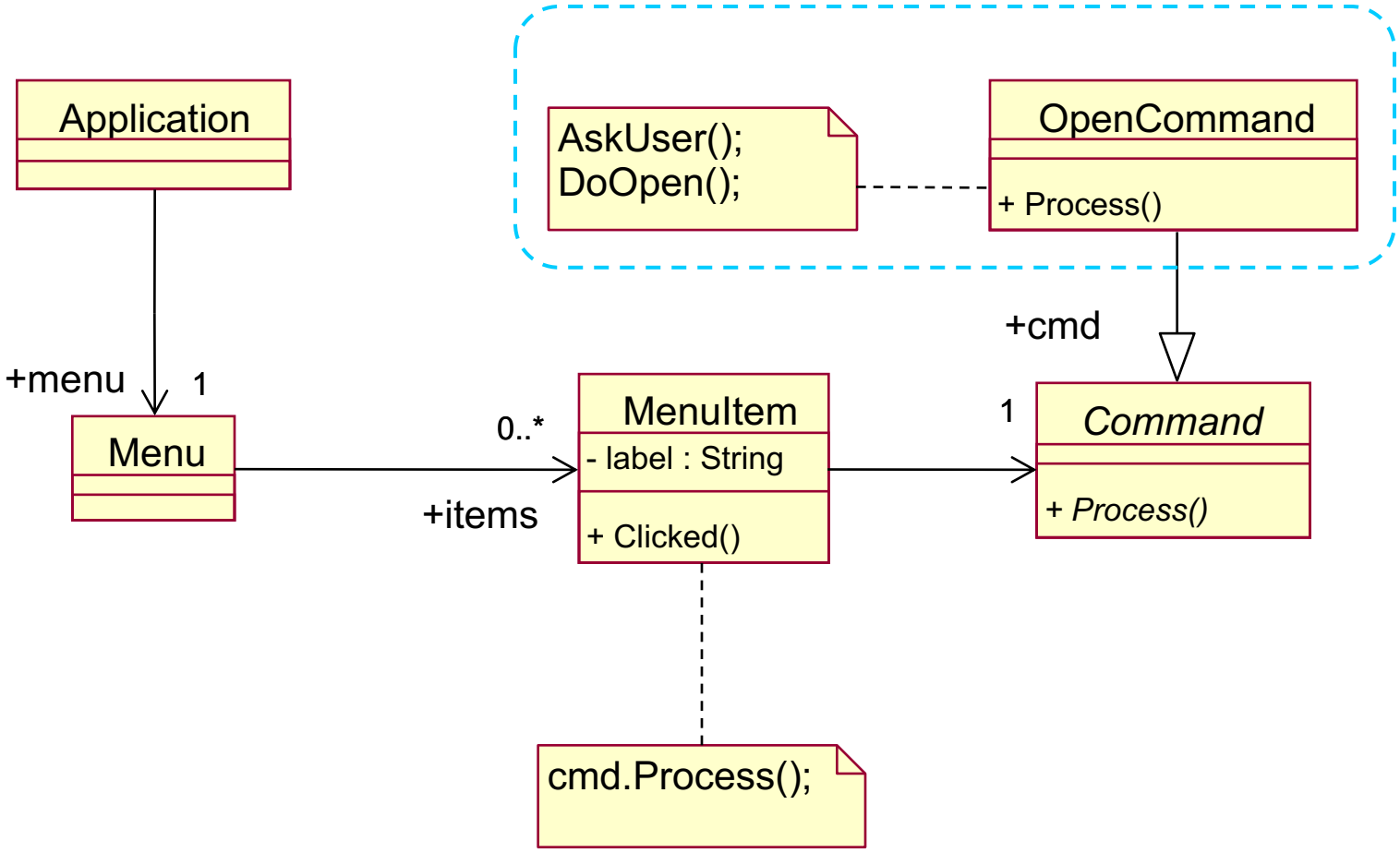
Examples of Pattern Usage

Pattern	Example
Command (behavioral pattern)	Issue a request to an object without knowing anything about the operation requested or the receiver of the request: for example, the response to a menu item, an undo request, the processing of a time-out
Abstract factory (creational pattern)	Create GUI objects (buttons, scrollbars, windows, etc.) independent of the underlying OS: the application can be easily ported to different environments
Proxy (structural pattern)	Handle distributed objects in a way that is transparent to the client objects (*remote proxy*) Load a large graphical object or any entity object "costly" to create/initialize only when needed (*on demand*) and in a transparent way (*virtual proxy*)
Observer (behavioral pattern)	When the state of an object changes, the dependent objects are notified. The changed object is independent of the observers. Note: The MVC architectural pattern is an extension of the Observer design pattern

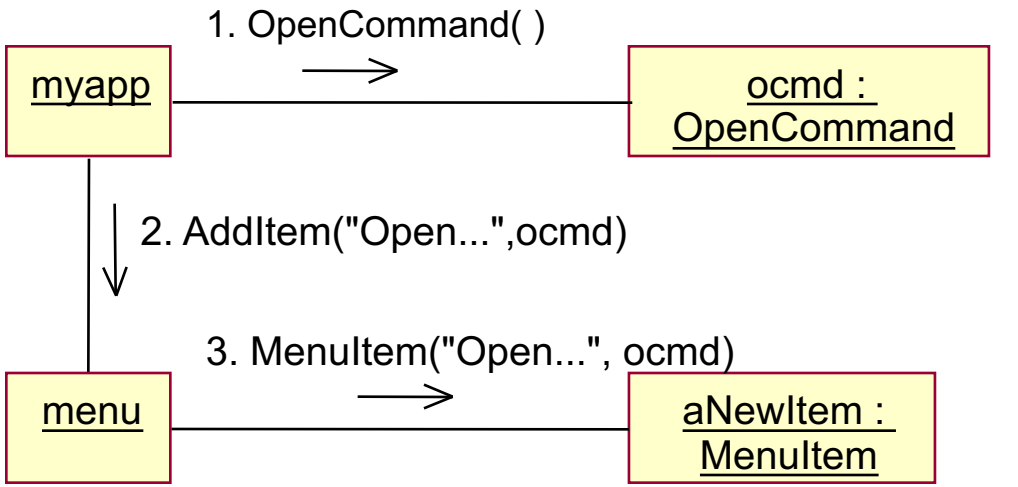
Detailing the Command Pattern



Detailing the Command Pattern (continued)

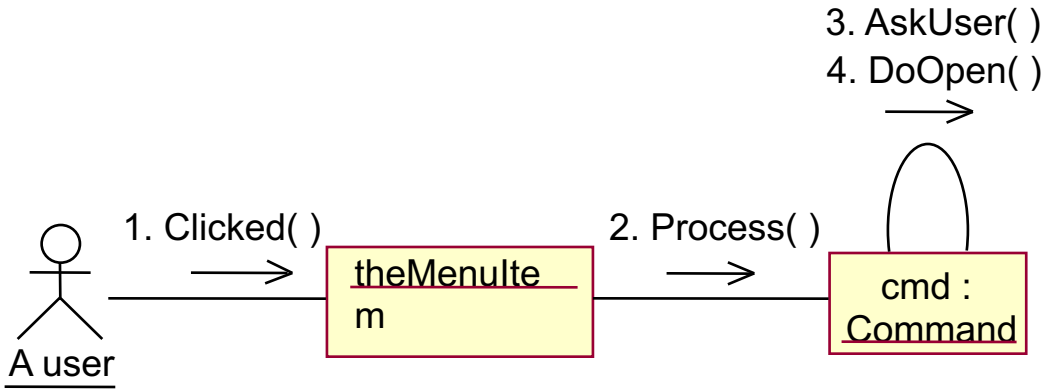


Detailing the Command Pattern (continued)

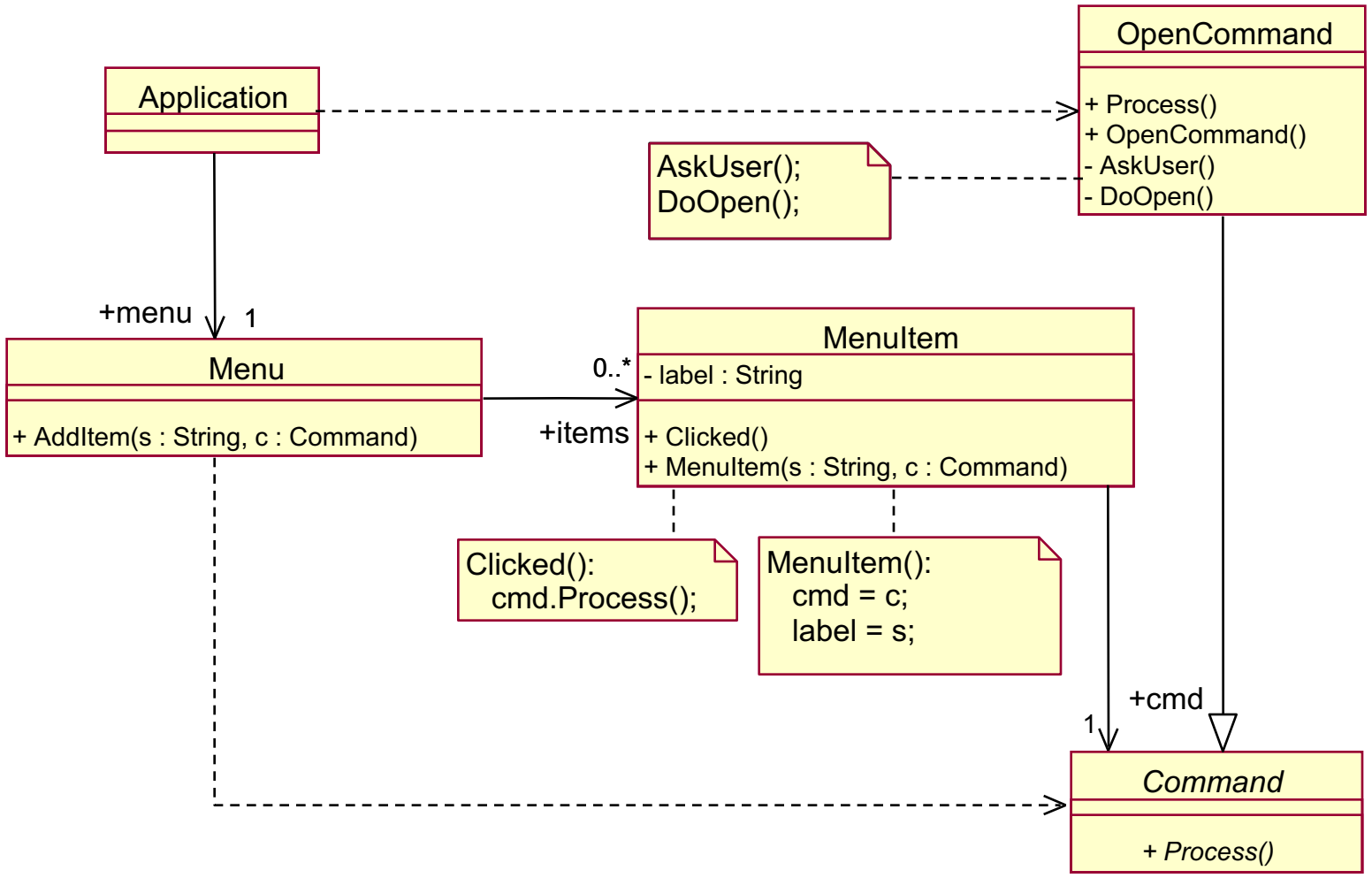


Initialization

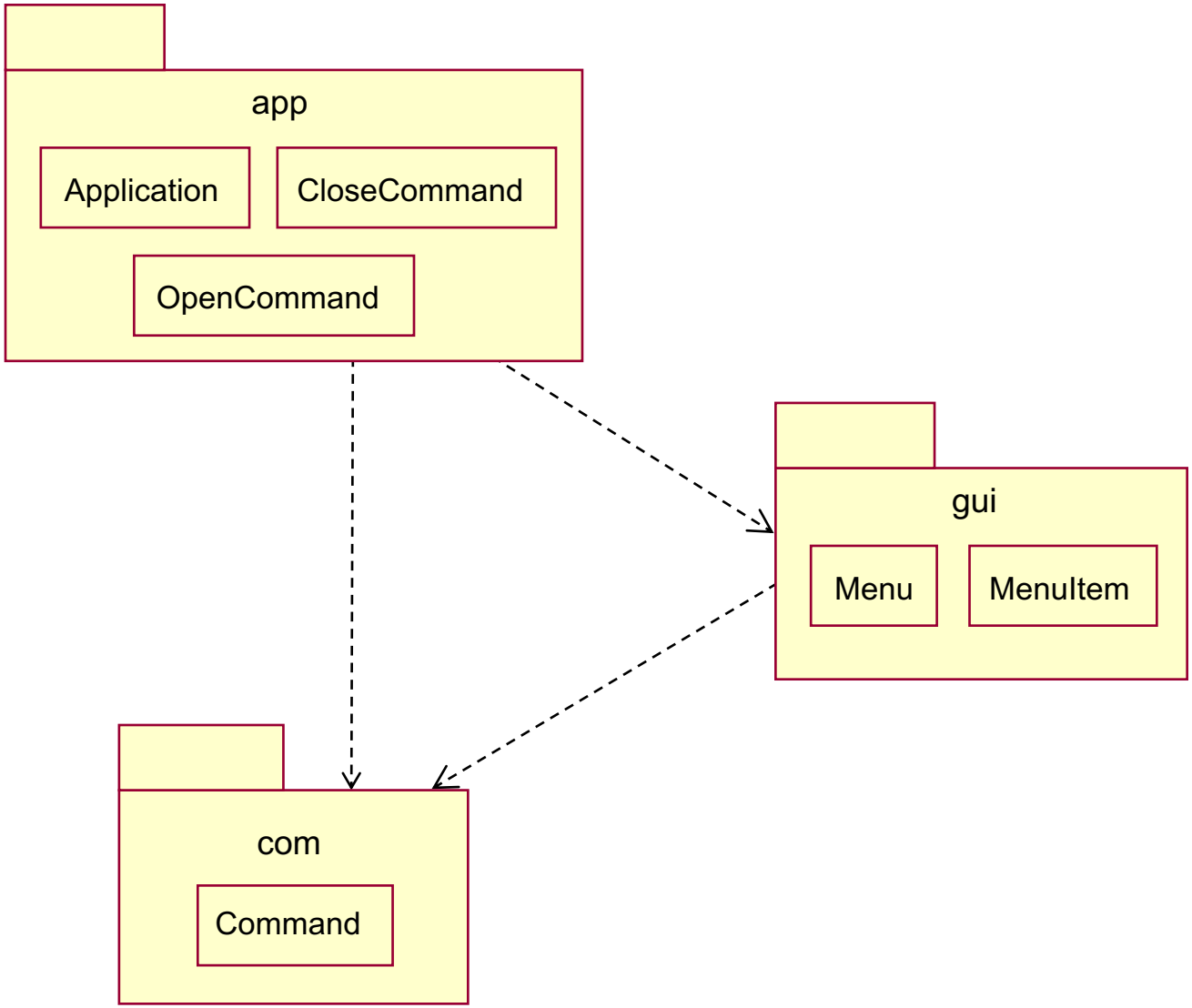
The user selects the Open… *menu item*



Detailing the Command Pattern (continued)

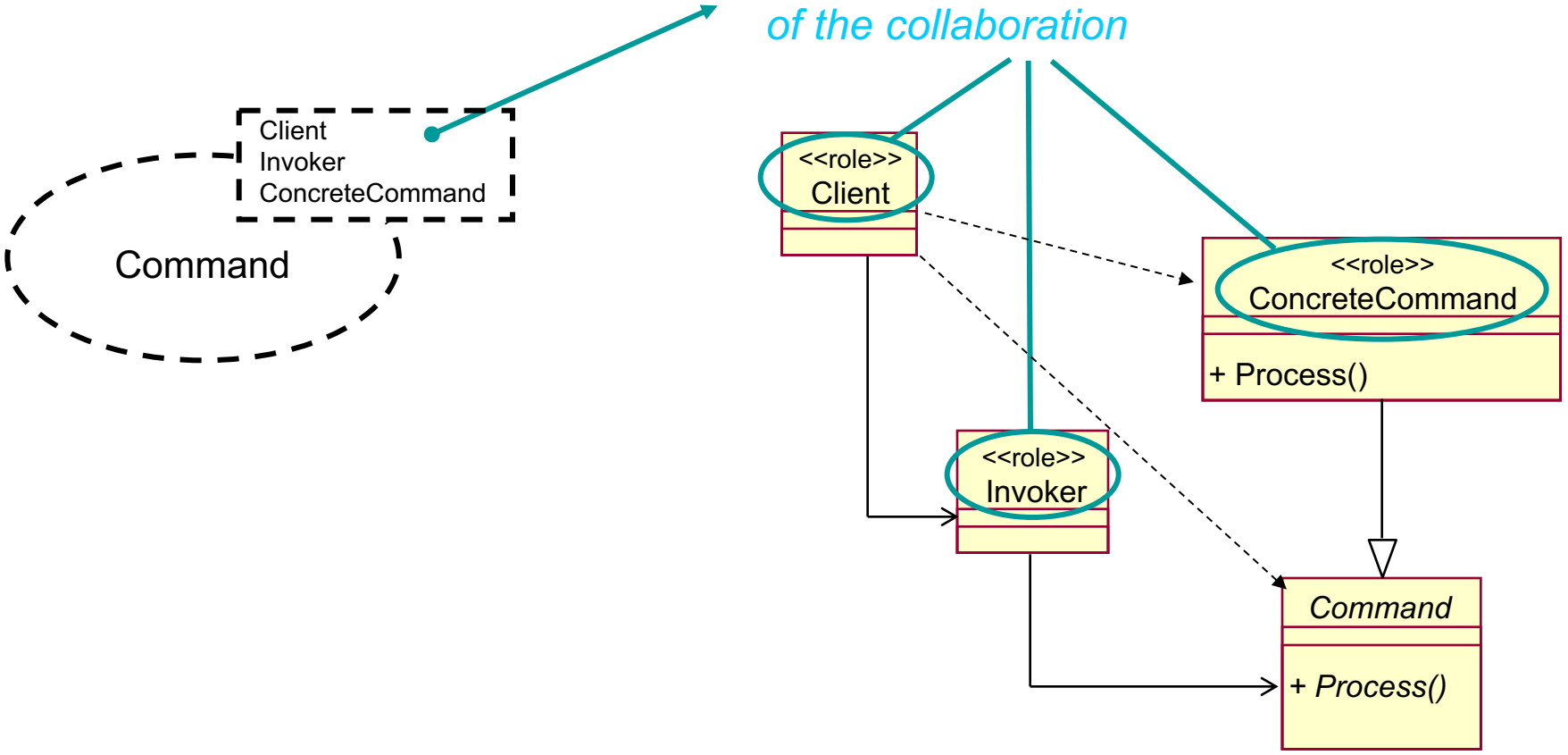


Detailing the Command Pattern (continued)



Representing Design Patterns in UML

- A design pattern is a parameterized collaboration:



Describing Analysis Mechanisms

- Collect all analysis mechanisms in a list
- Draw a map of the client classes to the analysis mechanisms

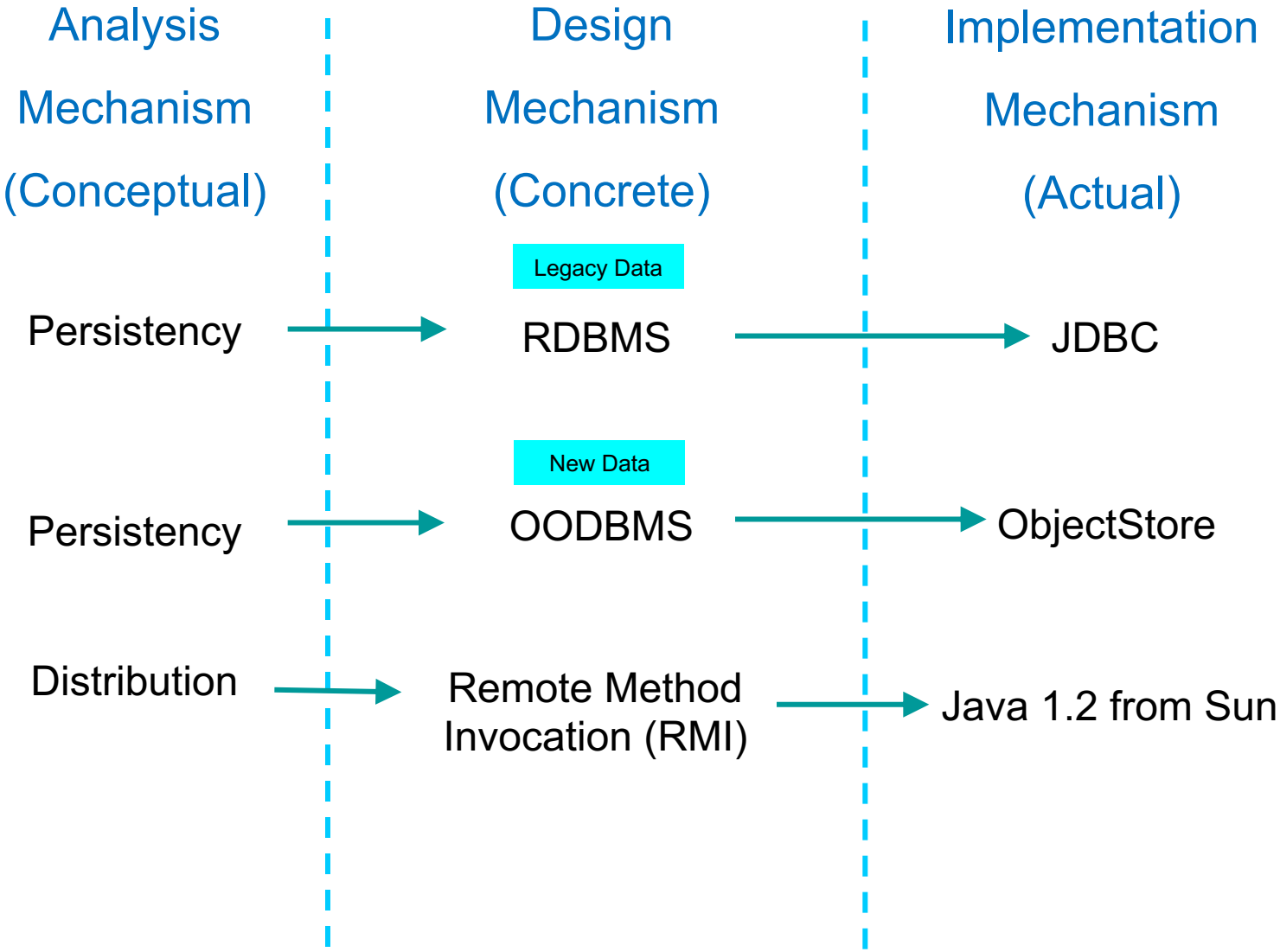
Analysis Class	Analysis Mechanism(s)
Student	Persistence, Security
Schedule	Persistence, Security
CourseOffering	Persistence, Legacy Interface
Course	Persistence, Legacy Interface
RegistrationController	Distribution

- Identify characteristics of the Analysis mechanisms

Identify Design Mechanisms: Steps

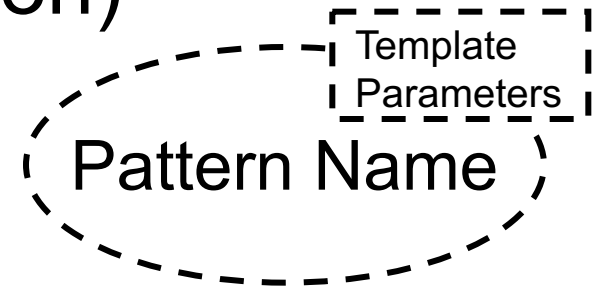
- Categorize clients of analysis mechanisms
- Documenting architectural mechanisms

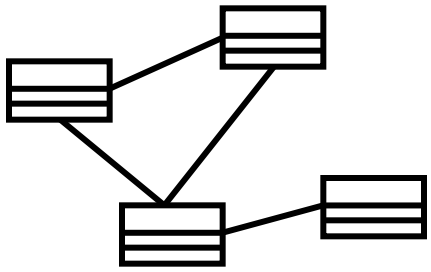
Design and Implementation Mechanisms



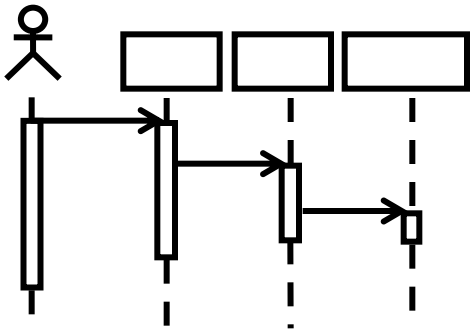
Review: Documenting Architectural Mechanisms

- Architectural mechanisms can be treated as patterns (i.e., parameterized collaboration)





Structural Aspect

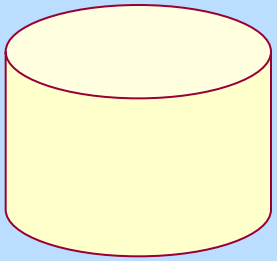


Behavioral Aspect

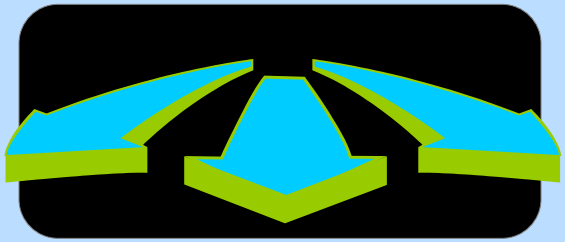
Documented in Design Guidelines

Review: Course Registration Analysis Mechanisms

Persistence

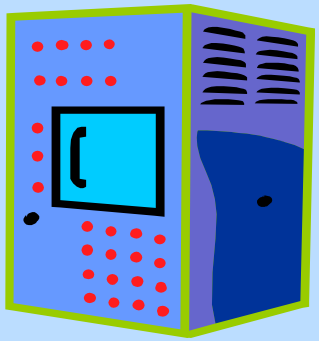


Distribution



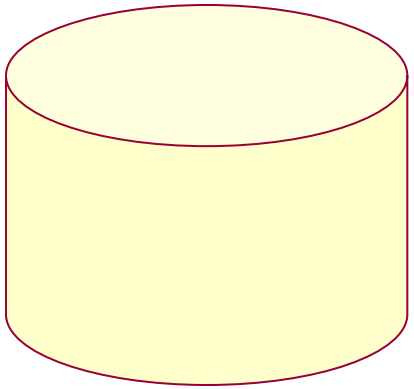
Security

Legacy Interface



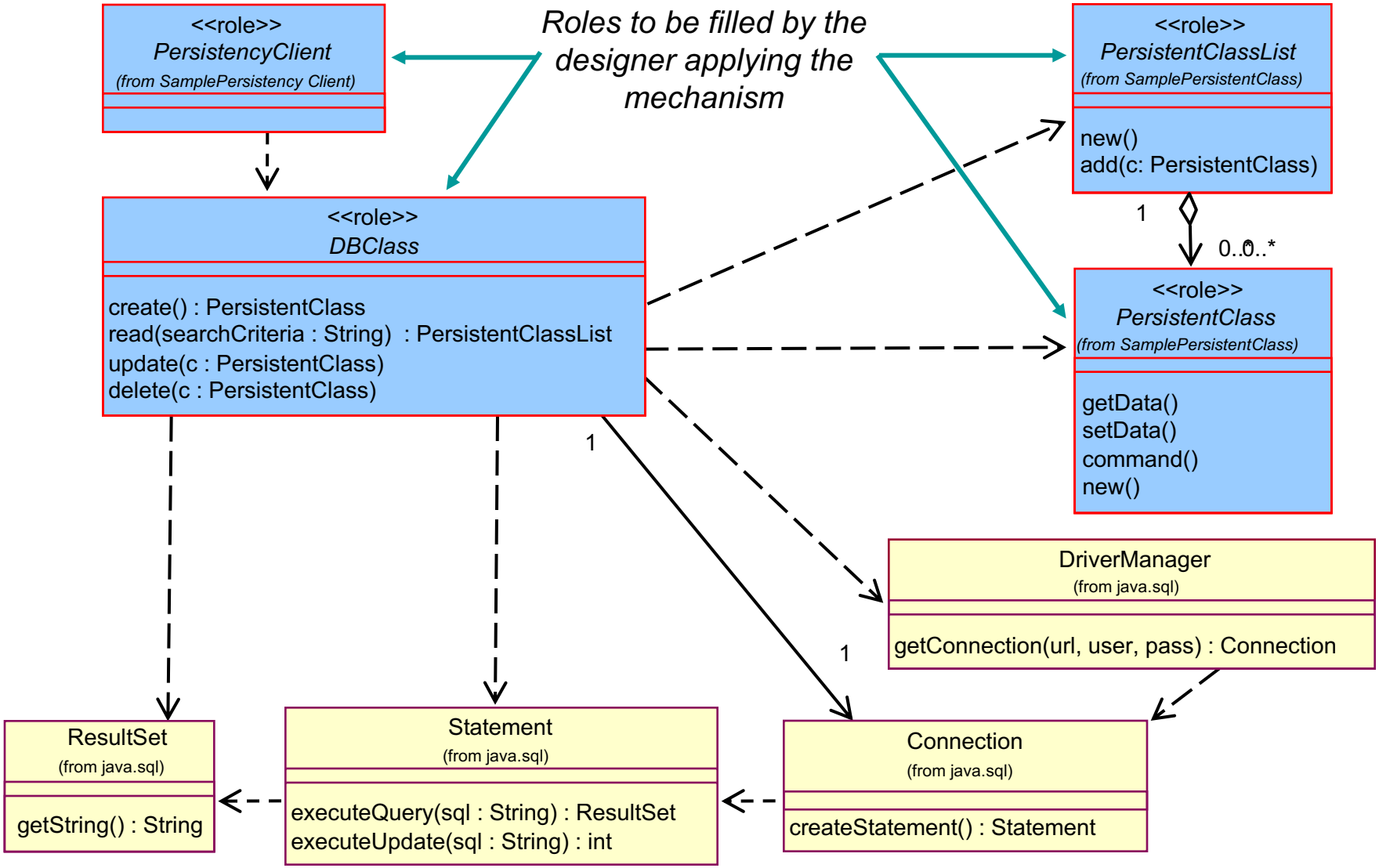
Mechanism: Persistency: RDBMS: JDBC

- Persistence characteristics:
  - Granularity
  - Volume
  - Duration
  - Access mechanism
  - Access frequency (creation/deletion, update, read)
  - Reliability

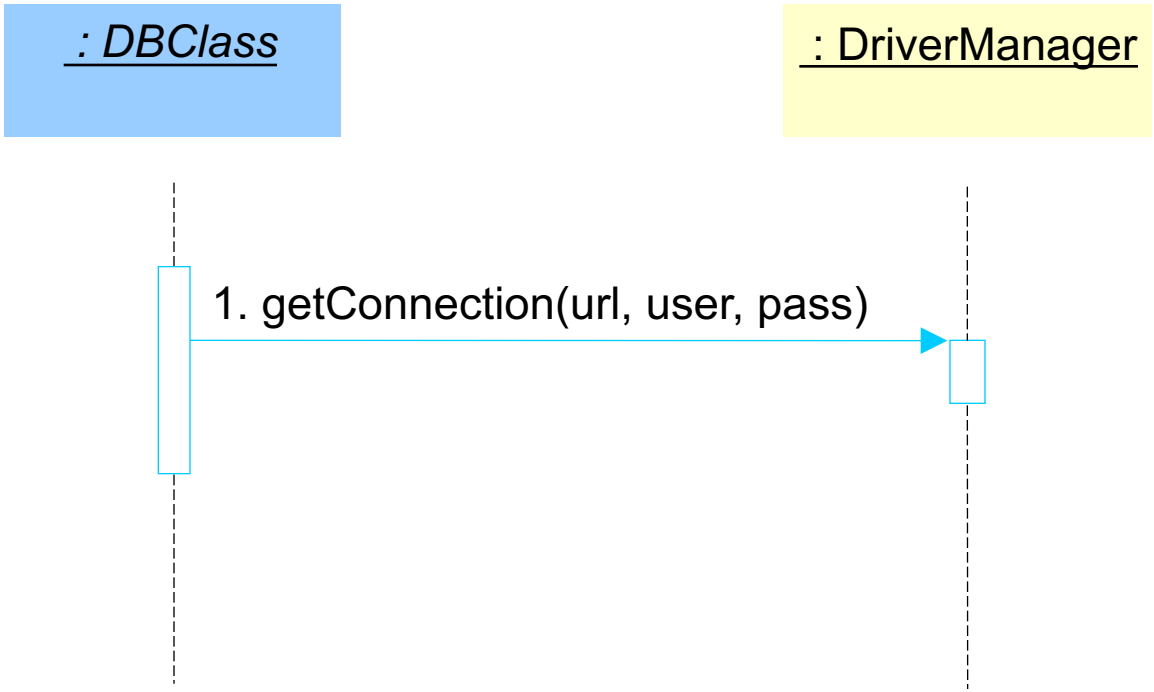


Note: JDBC is the standard Java API for talking to a SQL database.

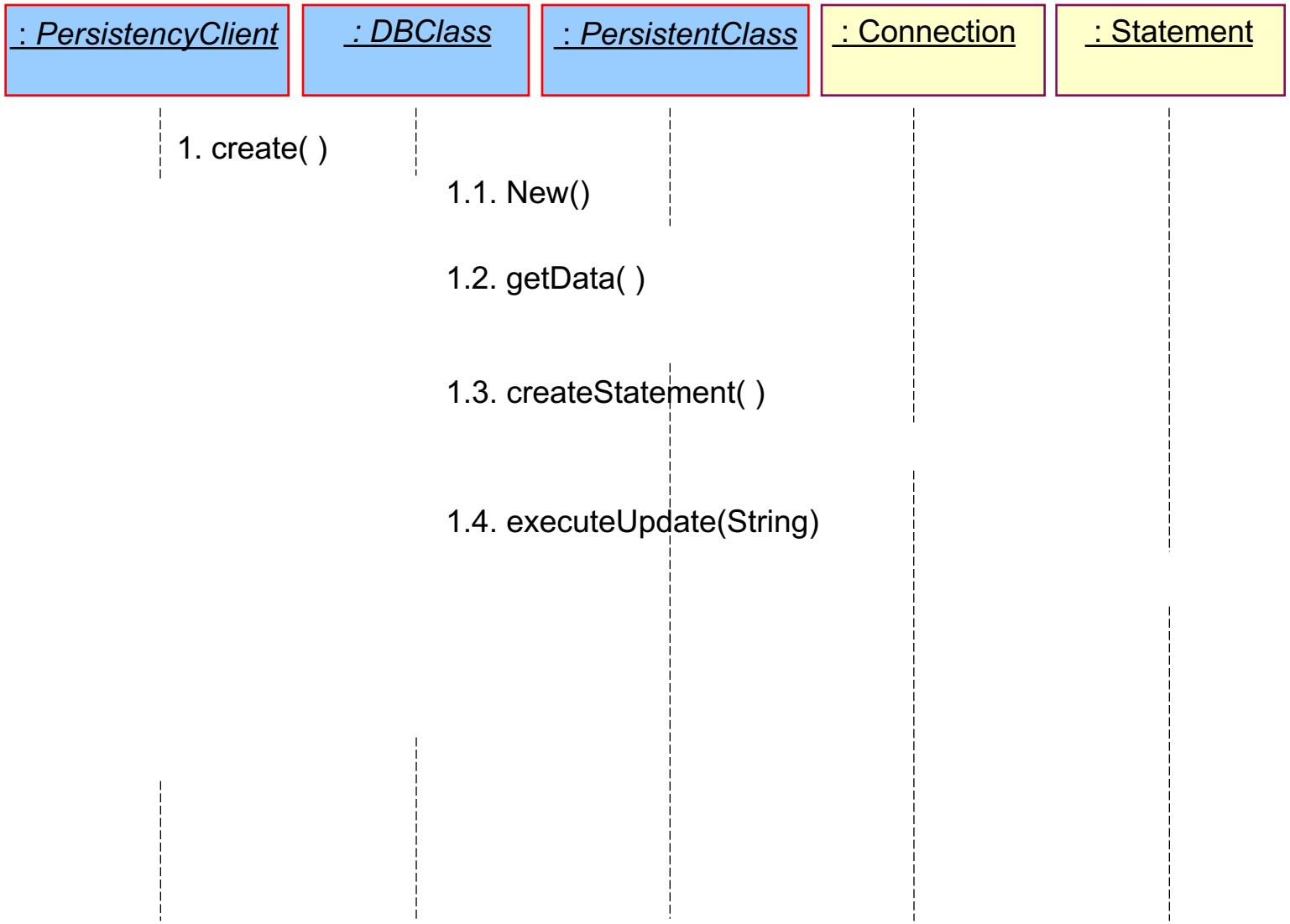
Example: Persistency: RDBMS: JDBC



Example: Persistency: RDBMS: JDBC: Initialize

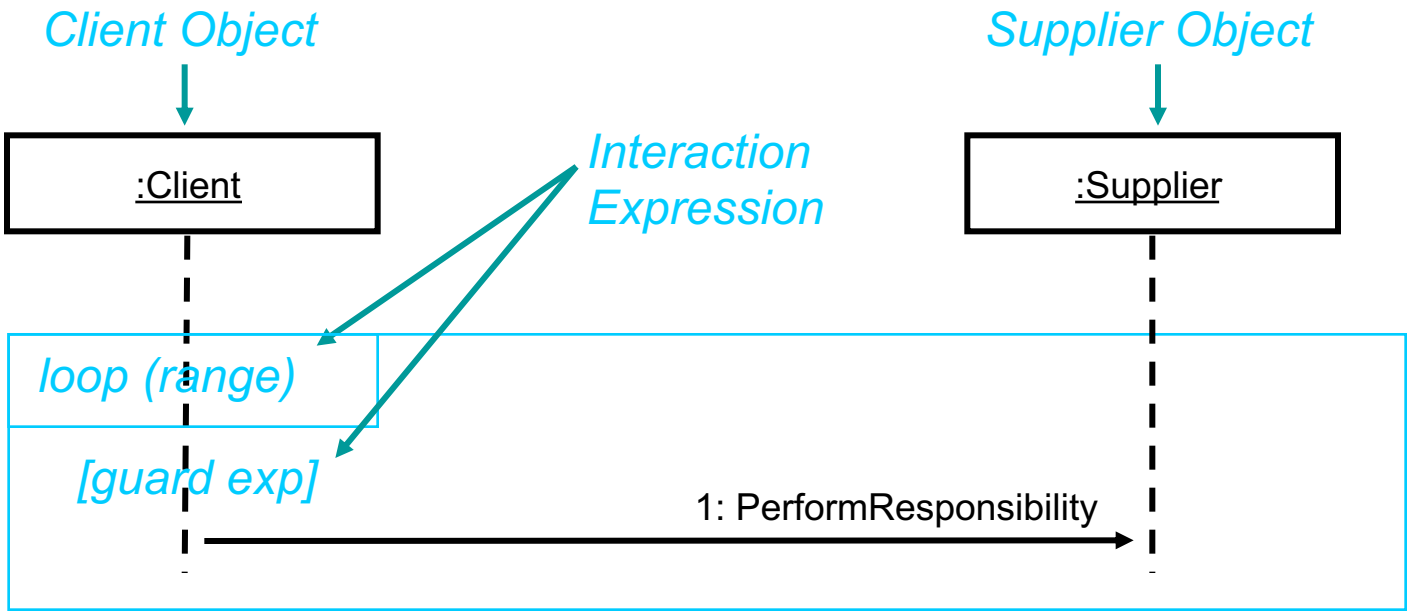


Example: Persistency: RDBMS: JDBC: Create

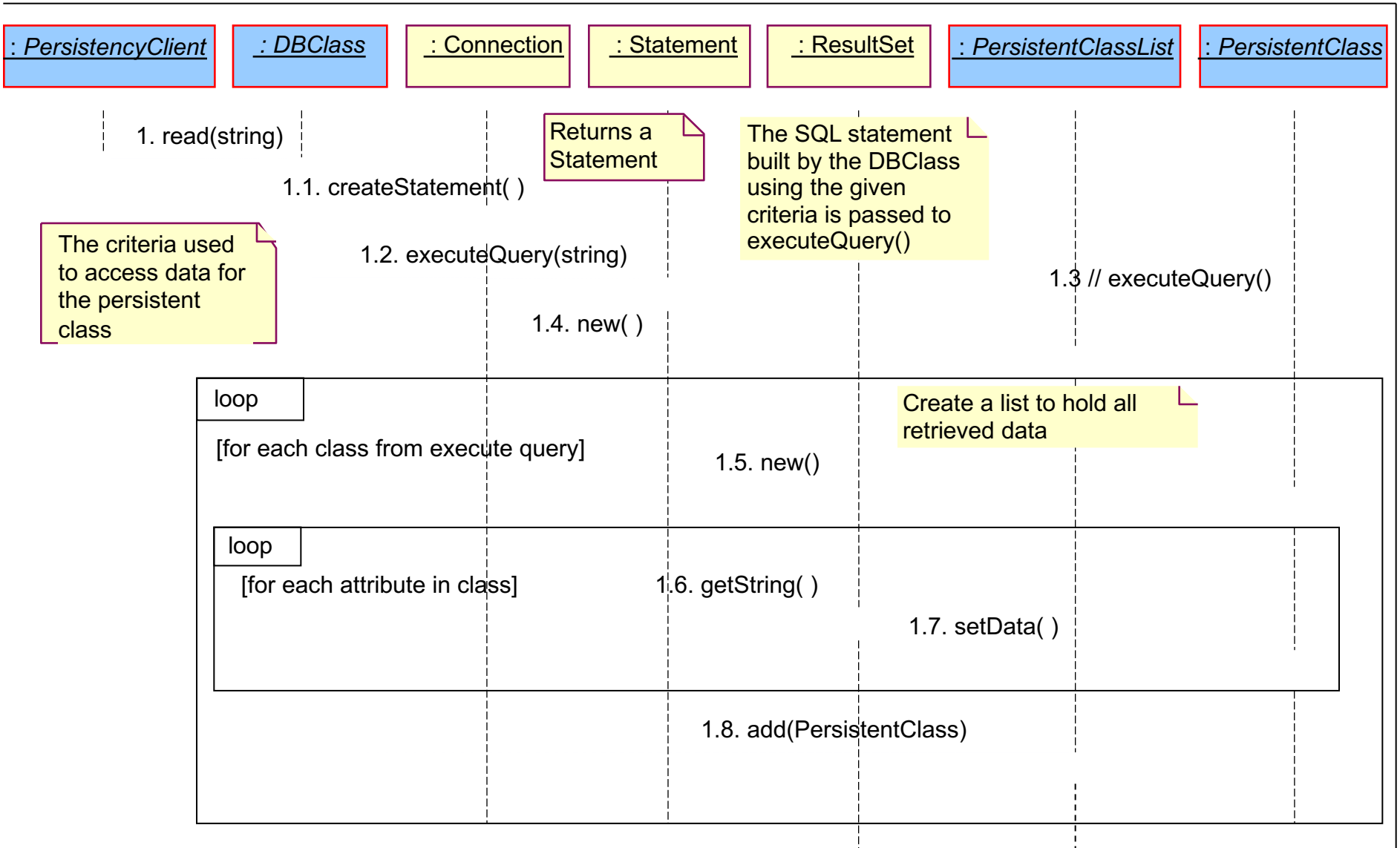


What is an Interaction Expression?

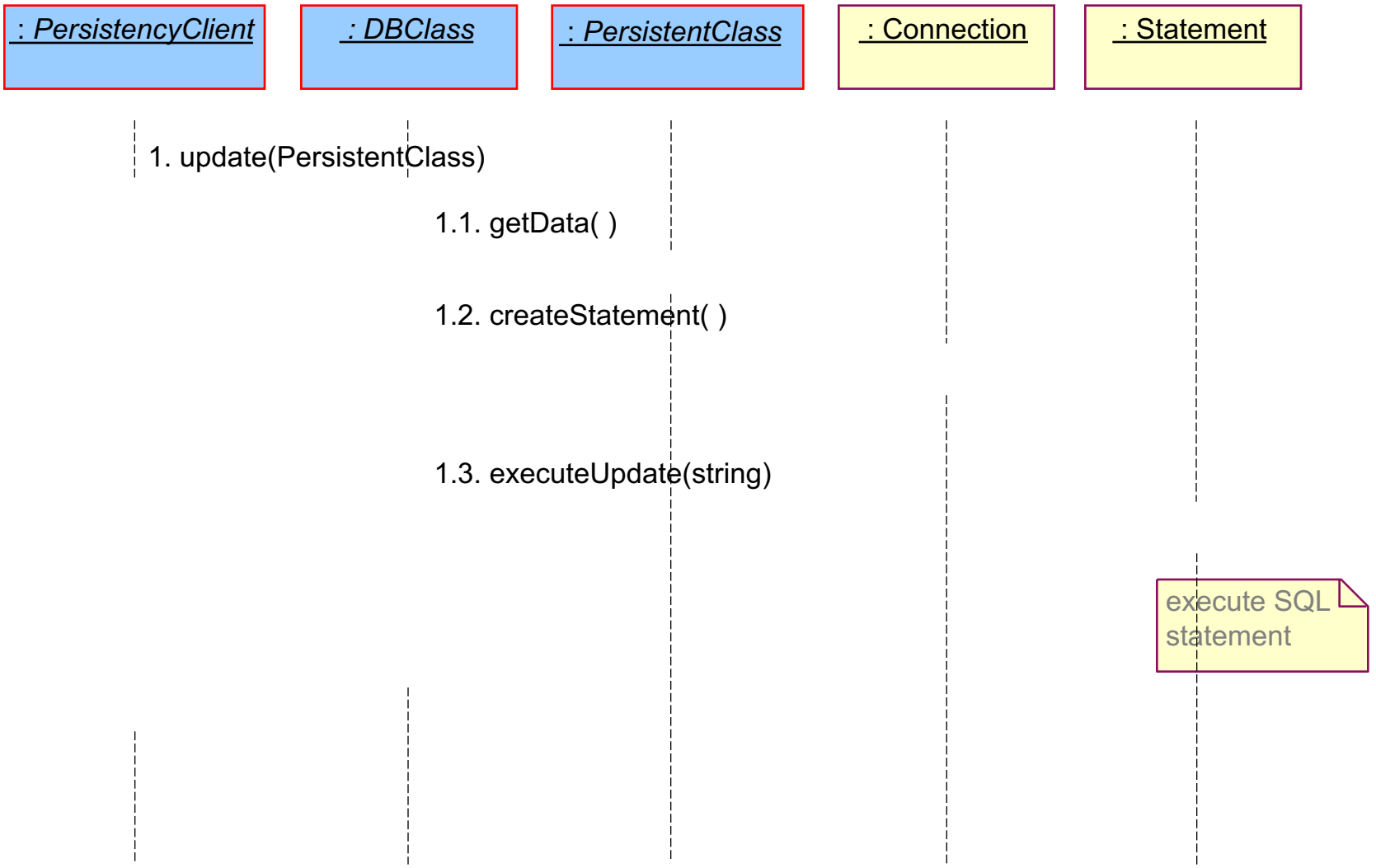
- A specification of the range of number of iterations of a loop.
  - Range can be specified with minimum and maximum values
  - A guard condition, enclosed in square brackets, can be included on a lifeline.



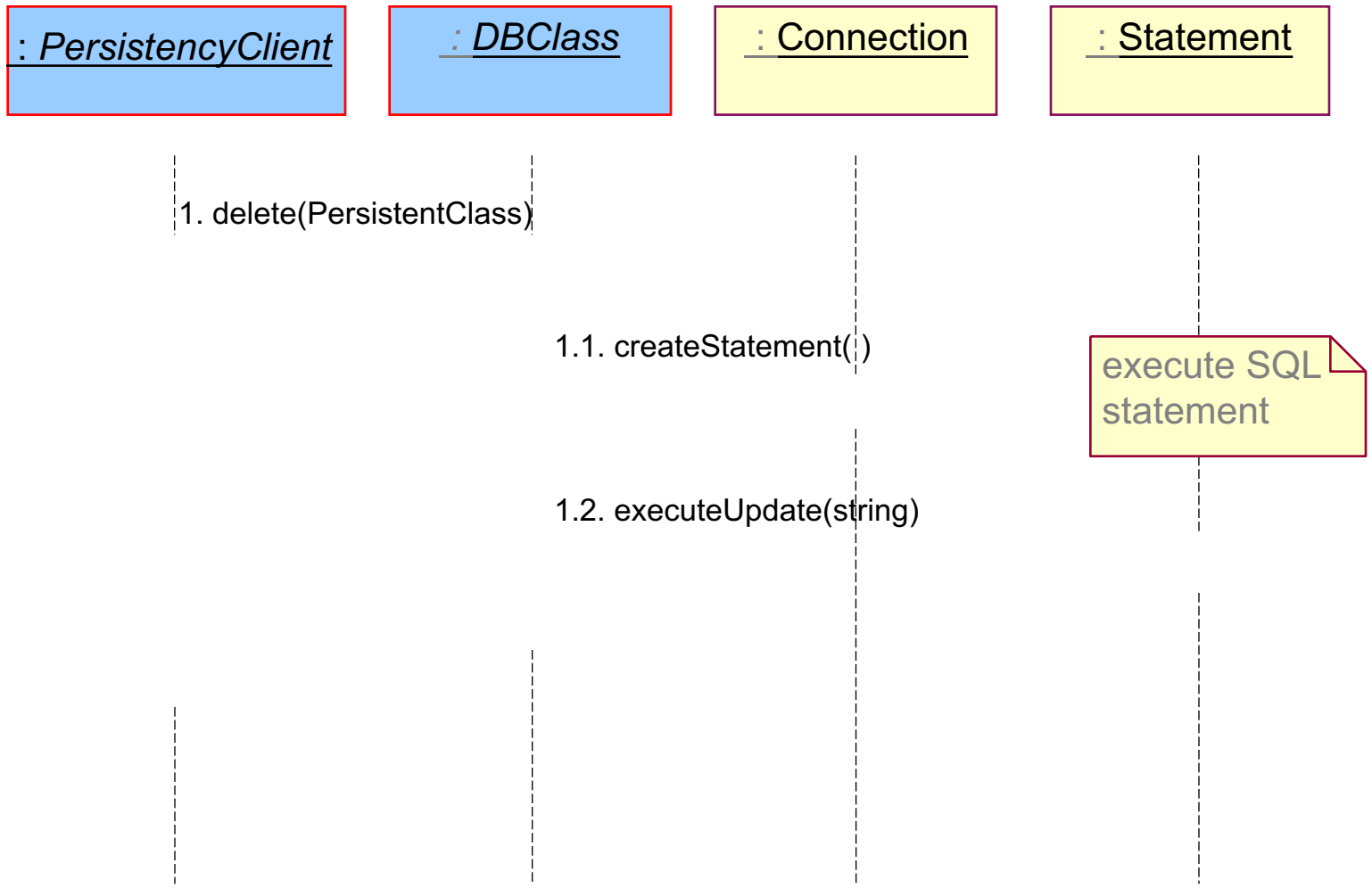
Example : Persistency: RDBMS: JDBC: Read



Example: Persistency: RDBMS: JDBC: Update



Example: Persistency: RDBMS: JDBC: Delete

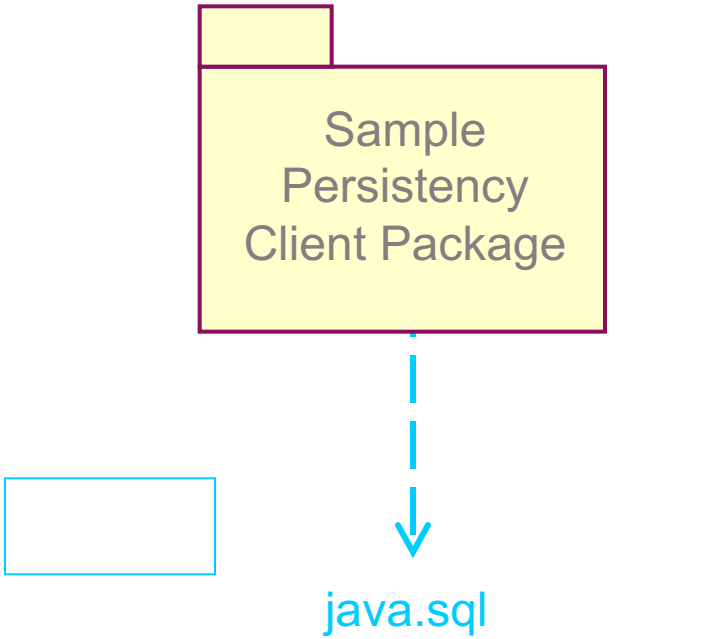


Incorporating JDBC: Steps

1. Provide access to the class libraries needed to implement JDBC
   - *Provide java.sql package*
2. Create the necessary DBClasses
   - Assign one DBClass per persistent class
3. Incorporate DBClasses into the design
   - Allocate to package/layer
   - Add relationships from persistency clients
4. Create/Update interaction diagrams that describe:
   - Database initialization
   - Persistent class access: Create, Read, Update, Delete

Deferred

Example: Incorporating JDBC



Review: Identify Design Mechanisms

- What does an analysis mechanism do?
- What is a pattern? What makes a framework different from a pattern?
- Why should you categorize analysis mechanisms? Identify the steps.