HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

SOICT

School of Information and Communication Technology

ONE LOVE. ONE FUTURE.

TRƯỜNG ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

IT3180 – Introduction to Software Engineering

14 – Models for Program Design

ONE LOVE. ONE FUTURE.

Approaches for Program Development

Heavyweight Approach

- Program design and coding are separated
- The design uses class models and other models to specify the program in detail, before beginning the code

Lightweight Approach

- Program design and coding are intertwinned
- The development is iterative

Mixed Approach

- Outline design is created using models, with details worked out iteratively during work

Program Design

The task of Program Design is to represent the software architecture in the form that can be implemented as one or more executable programs

Given a system architecture, the program design specifies:

- Programs, components, packages, classes, class hierarchies, etc.
- Interfaces, protocols (which may be not part of system architecture)
- Algorithms, data structures, security mechanism, operational procedures

UML Models

Models used for requirements

- **Use case diagram** shows a set of use cases and actors and their relationships

Models used for system architecture:

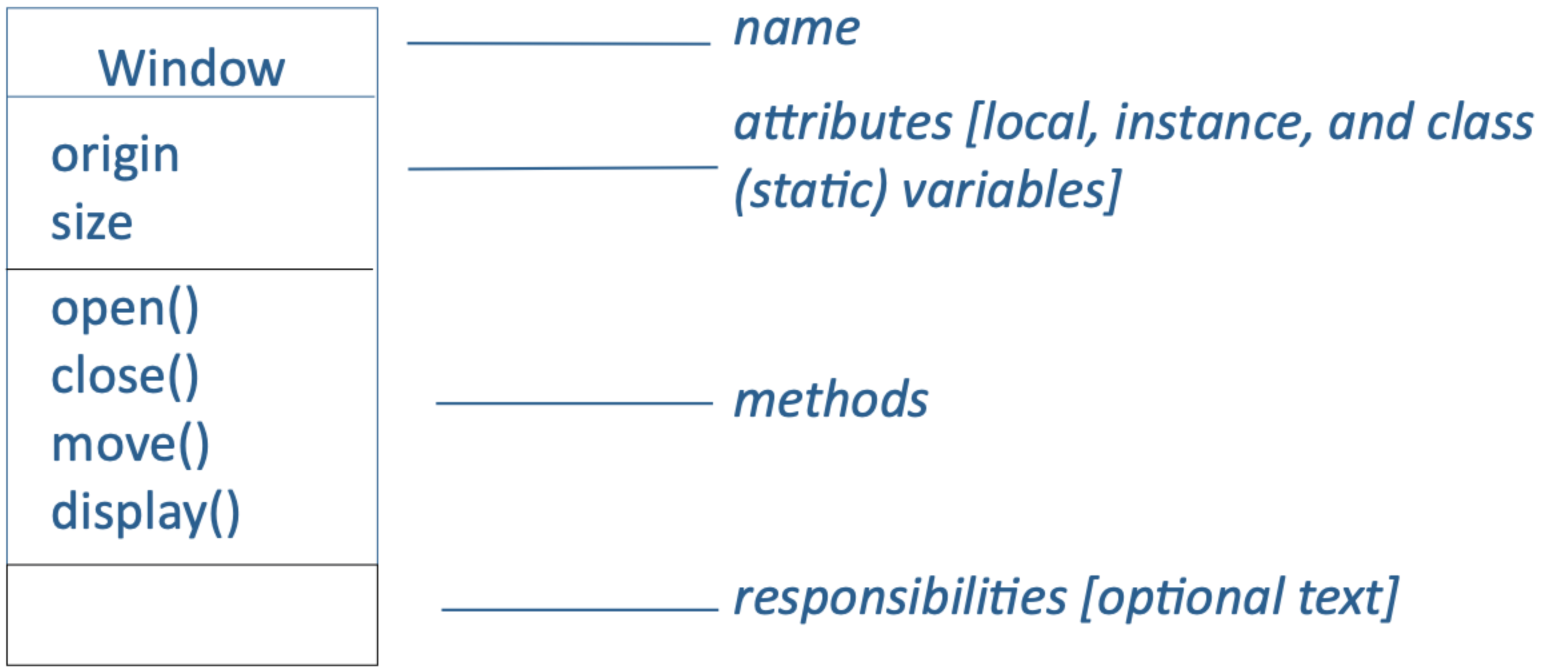
- **Component diagram** shows the organization and dependencies among a set of components
- **Deployment diagram** shows the configuration of processing nodes and the components living on them

Models used for program design

- **Class diagram** shows a set of classes, interfaces, and collaborations with their relationships
- **Object Diagram** or **Sequence Diagram** show a set of objects and their relationships

Class Diagram

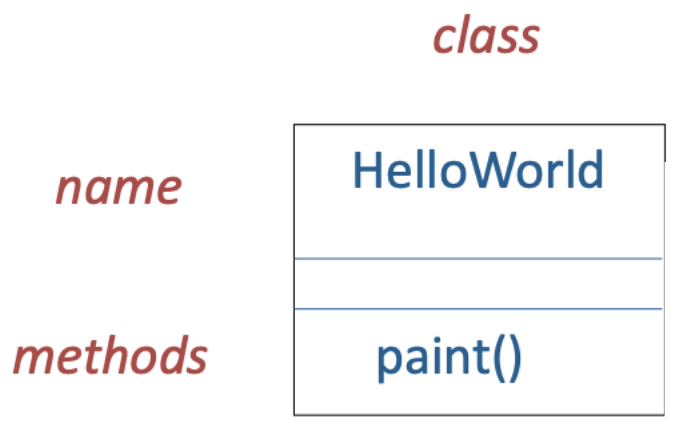
A **class** is a **description** of a set of objects that share the **same attributes methods**, **relationships** and **semantics**


Window
origin
size
open()
close()
move()
display()
name
attributes [local, instance, and class (static) variables]
methods
responsibilities [optional text]

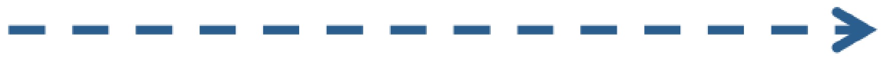
Example: The “Hello World” Applet

```
import java.awt.Graphics;
class HelloWorld extends java.applet.Applet {
    public void paint (Graphics g) {
        g.drawString ("Hello, World!", 10, 20);
    }
}
```

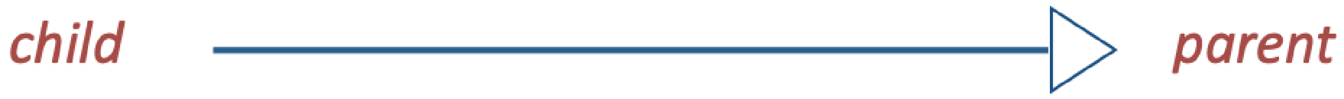
Example (2) – The Hello World Class



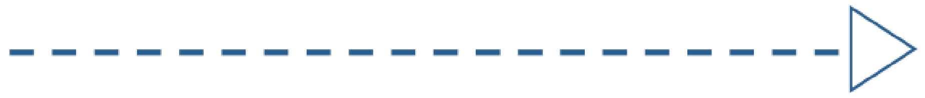
Notation: Relationships



A **dependency** is a semantic relationship between two things in which a change to one may effect the semantics of the other.

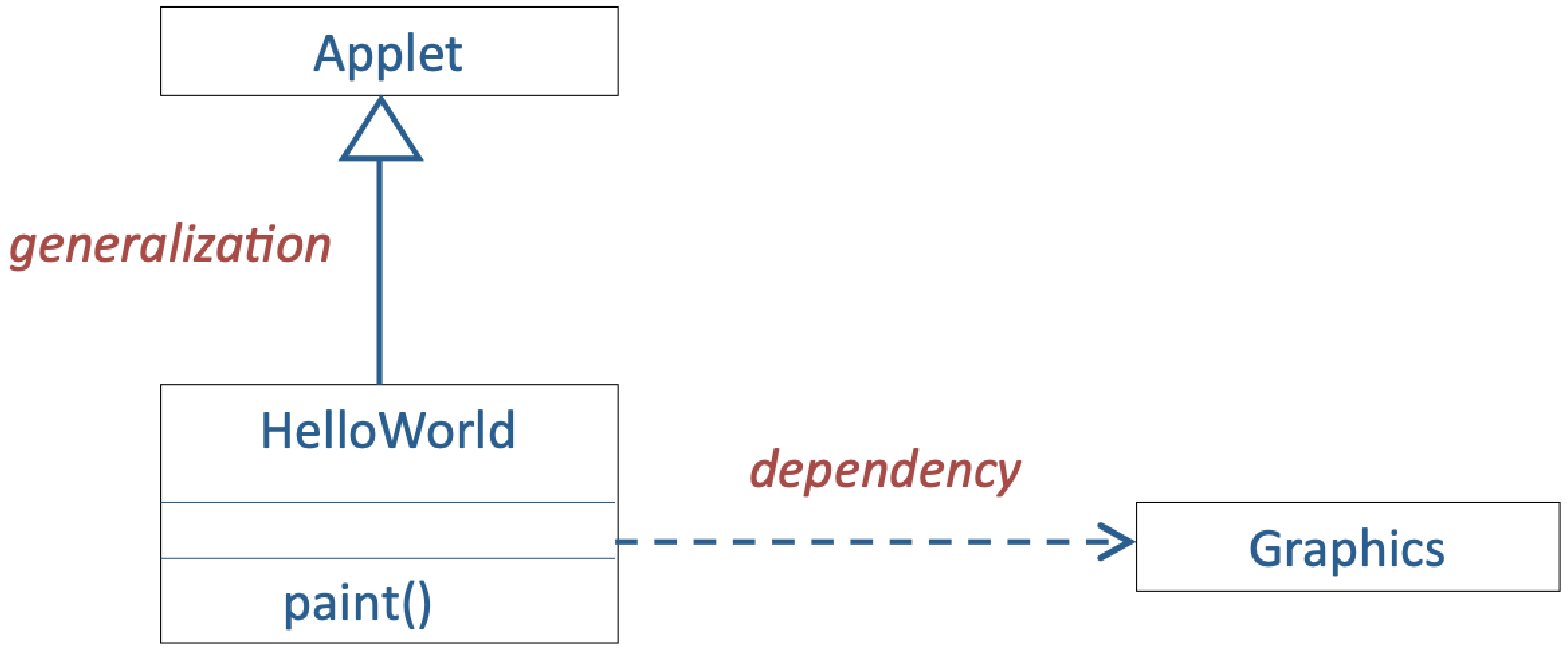


A **generalization** is a relationship is which objects of the specialized element (child) are substitutable for objects of the generalized element (parent).



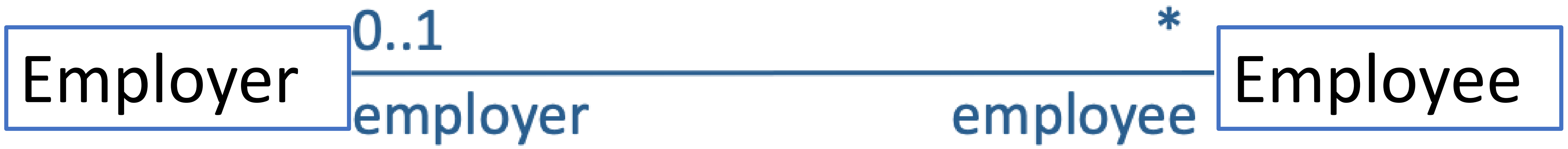
A **realization** is a semantic relationship between classifiers, wherein one classifier specifies a contract that another classifier guarantees to carry out.

Example (3) – The Hello World Class Diagram



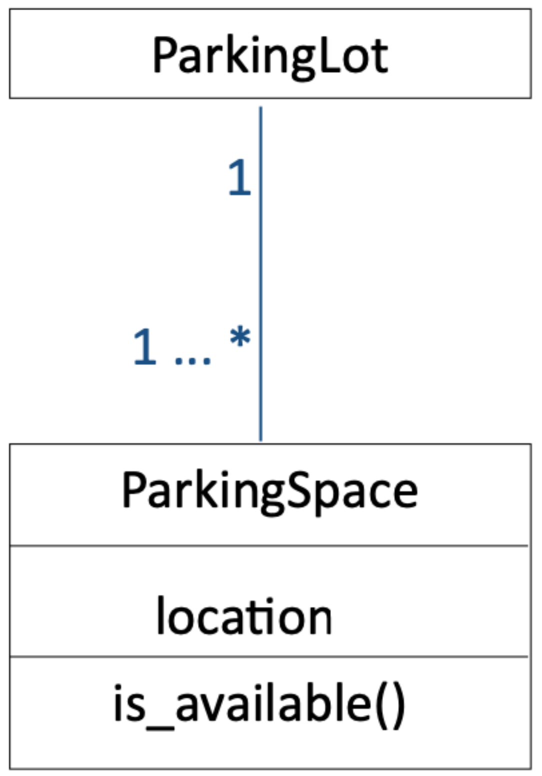
Notation: Association

An **association** is a structural relationship that describes a set of **links**, a link being a **connection among objects**



Notation: Association

A Parking Lot associates to one or many Parking Spaces



Which classes to be used?

- Given a case study, how do you decide what classes to use?

Step 1: Identify a set of candidate classes that represent the system design

- What terms do the users and developers use to describe the system? These terms are candidates for classes
- Is each candidate class crispy defined?
- For each class, what is its set of responsibilities? Are the responsibilities evenly balanced among the classes?
- What attributes and methods does each class need to carry out its responsibilities?

Which classes to be used? (2)

Step 2: Modify the set of classes

Goals:

- **Improve the clarity of design**
  - If the purpose of each class is clear, with easily understood methods and relationships, developers are likely to write simple code, which future maintainers can understand and modify
- **Increase the coherence within classes and lower the coupling between classes**
  - Aim for high cohesion within classes and weak coupling between them

Application Classes and Solution Classes

A good design is often a **combination** of **application classes** and **solution classes**

- Application classes represent **application concepts**
  - **Noun identification** is an effective technique to generate candidate application classes
- Solution classes represent **system concepts**
  - For example, user interface objects, databases, etc.

Noun Identification

Case study: A library example

- *The library contains books and journals. It may have several copies of a given book. Some of the books are reserved for short-term loans only. All others may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a 5me, but members of staff may borrow up to 12 items at one 5me. Only members of staff may borrow journals.*
- *The system must keep track of when books and journals are borrowed and returned, to enforce the rules.*

Noun Identification

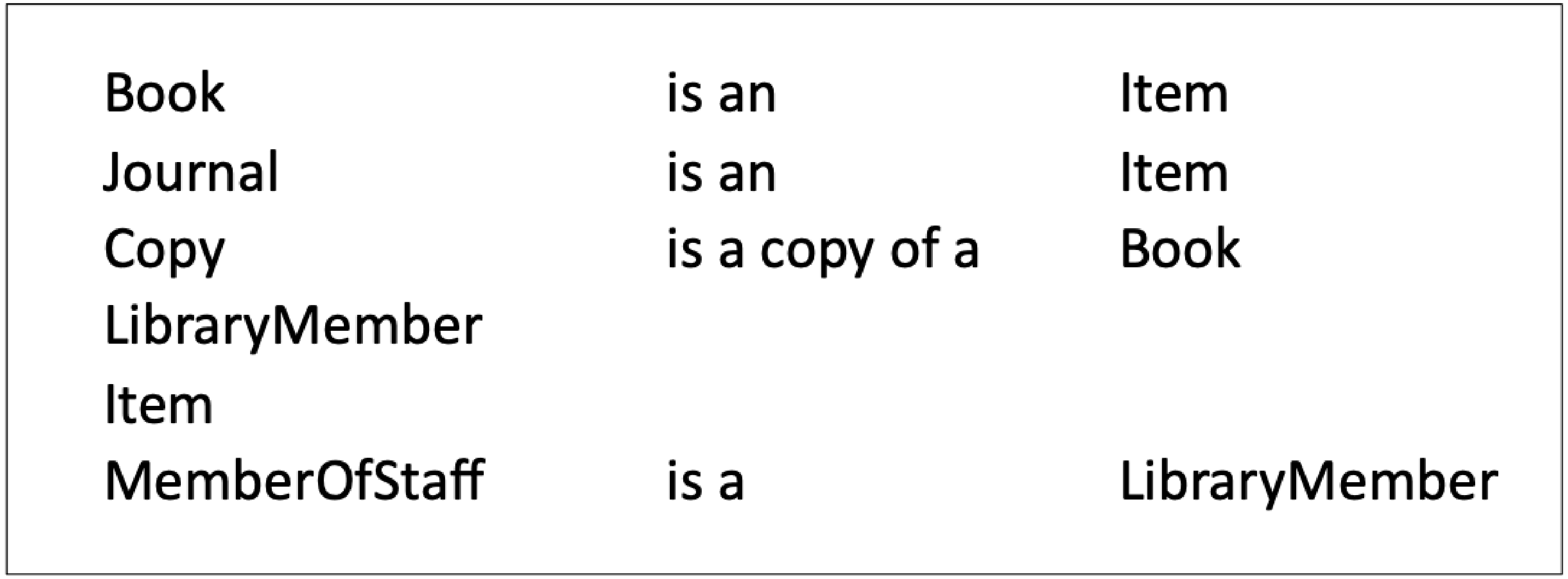
Case study: A library example

- *The **library** contains **books** and **journals**. It may have several **copies** of a given book. Some of the books are reserved for **short-term loans** only. All others may be borrowed by any **library member** for three **weeks**. **Members of the library** can normally borrow up to six **items** at a 5me, but **members of staff** may borrow up to 12 items at one 5me. Only members of staff may borrow journals.*
- *The **system** must keep track of when books and journals are borrowed and returned, to enforce the **rules**.*

Candidate Classes

Noun	Comments	Candidate
Library	*the name of the system*	
Book		
Journal		
Copy		
ShortTermLoan	*event*	
LibraryMember		
Week	*measure*	
MemberOfLibrary	*repeat of LibraryMember*	
Item	*book or journal*	
Time	*abstract term*	
MemberOfStaff		
System	*general term*	
Rule	*general term*	

Relation between classes



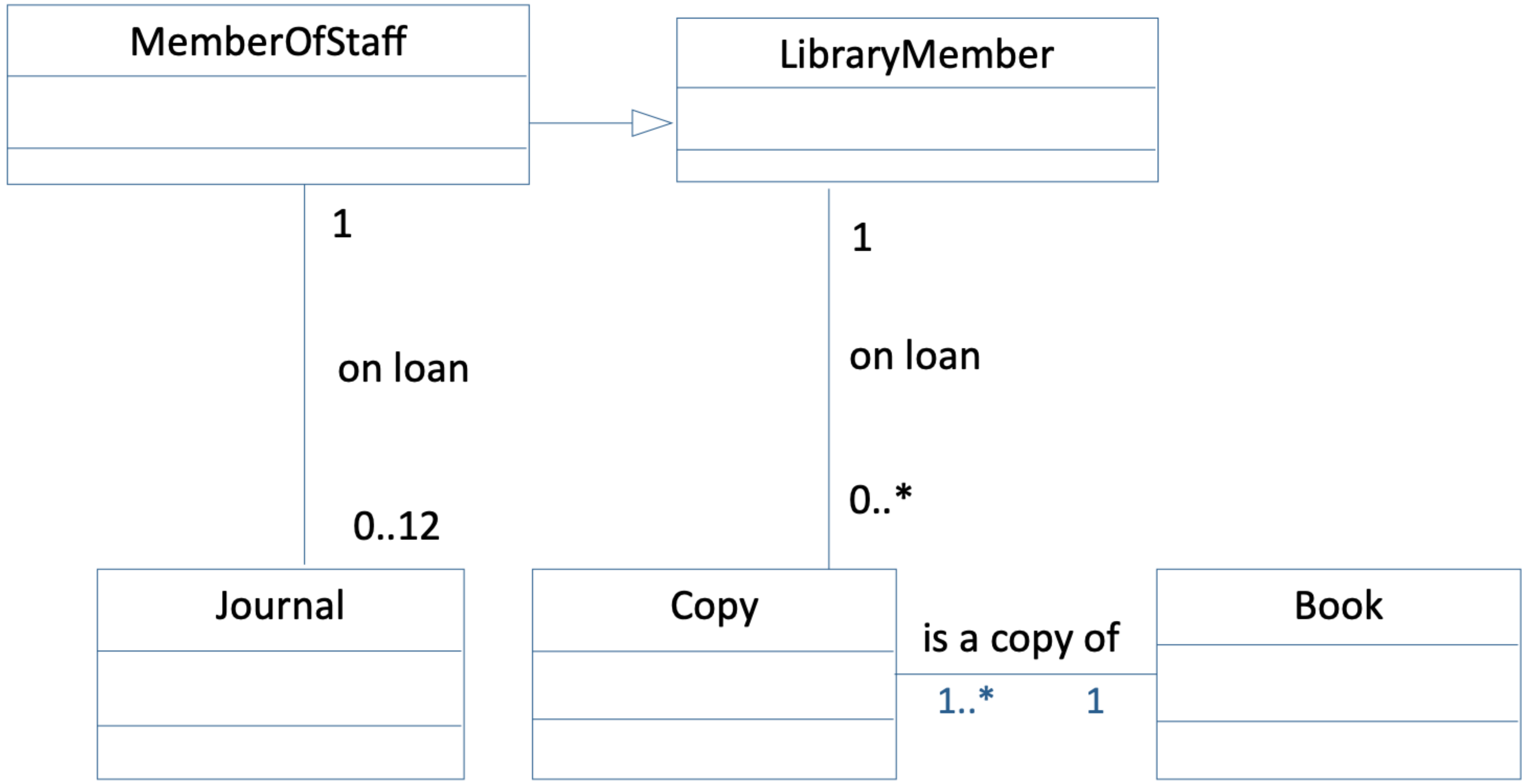
Book	is an	Item
Journal	is an	Item
Copy	is a copy of a	Book
LibraryMember		
Item		
MemberOfStaff	is a	LibraryMember

Methods

LibraryMember	borrows	Copy
LibraryMember	returns	Copy
MemberOfStaff	borrows	Journal
MemberOfStaff	returns	Journal

Item not needed yet.

Class Diagram (1)

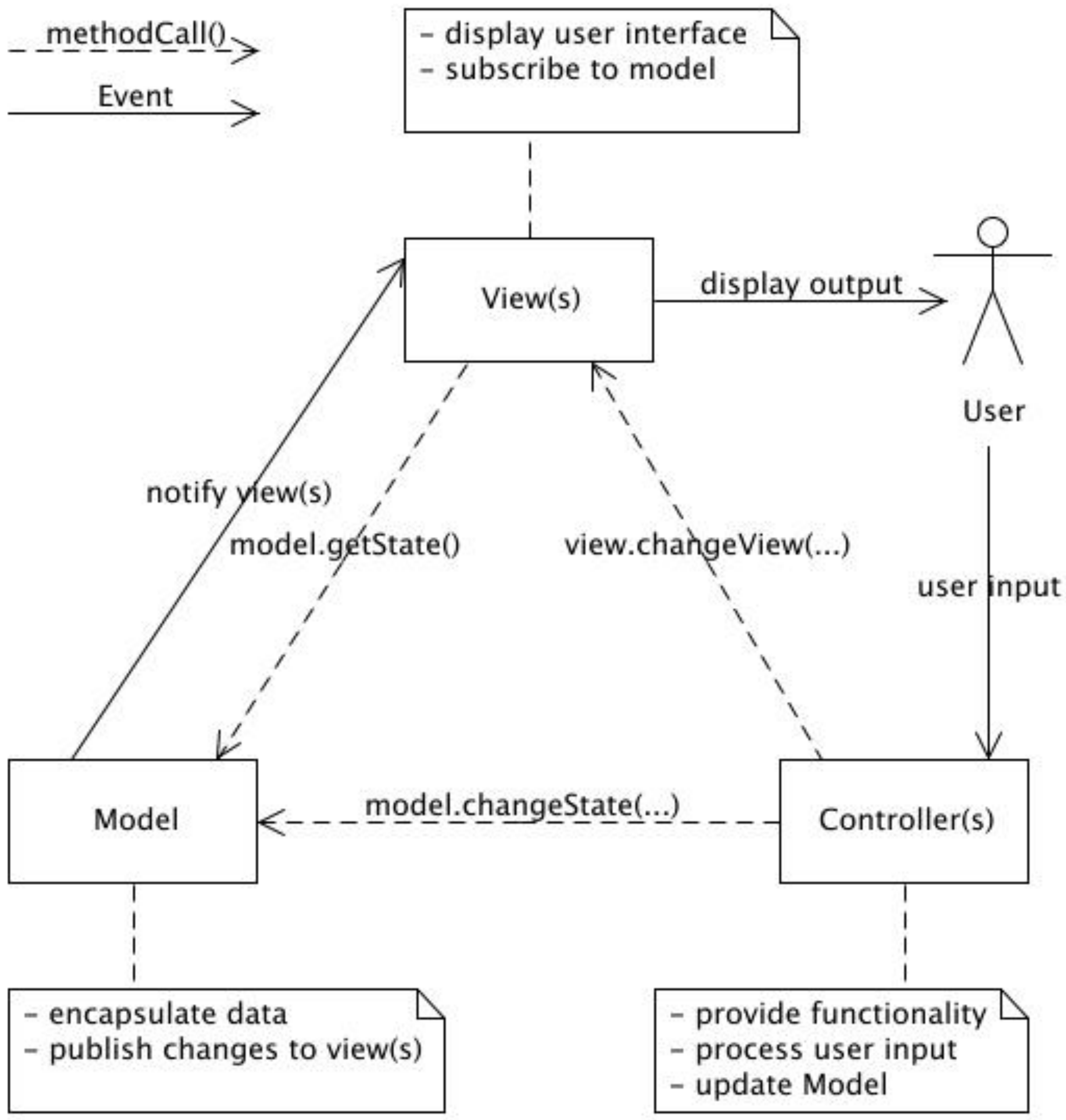


From Candidate Classes to Completed Design

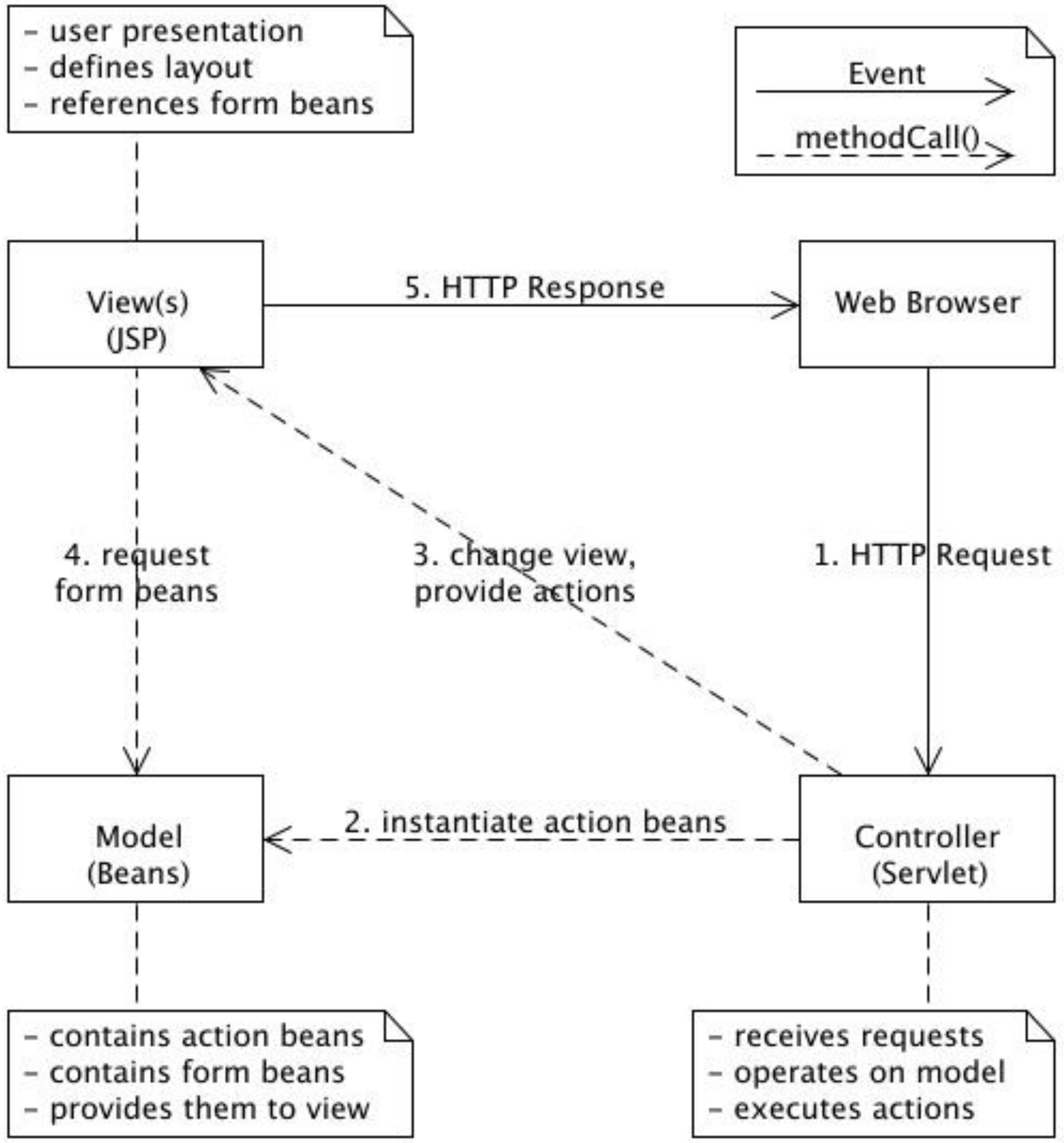
Methods used to move to final design

- **Reuse**
  - Wherever possible use existing components, or class libraries. They may need extensions.
- **Restructuring**
  - Change the design to improve understandability, maintainability, etc.
  - Techniques include merging similar classes, splitting complex classes, etc.
- **Optimization**
  - Ensure that the system meets anticipated performance requirements, e.g., by changed algorithms or restructuring
- **Completion**
  - Fill all gaps, specify interfaces, etc.

MVC Class Model



MVC Model for JSP Servlet

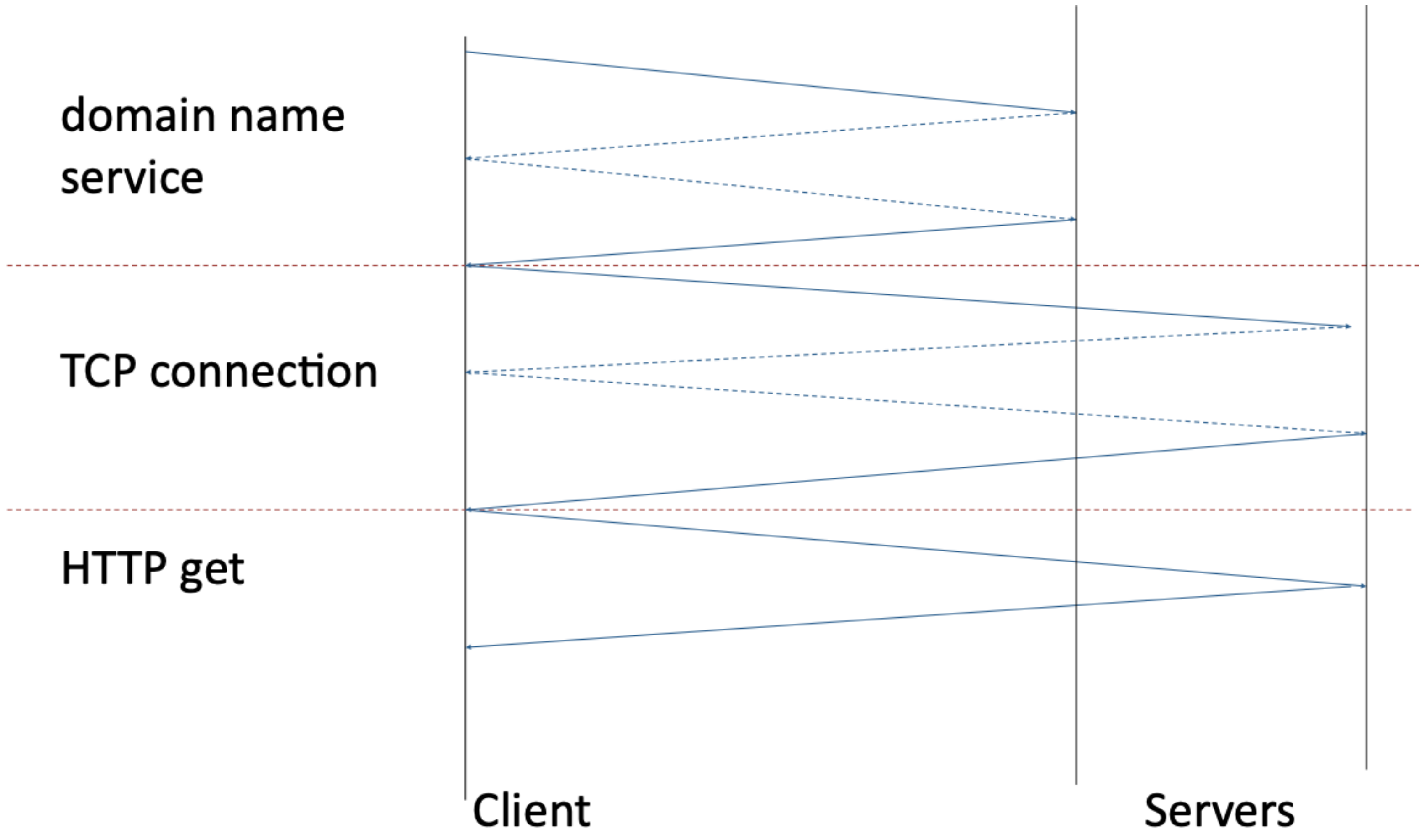


Modelling dynamic aspects of the system

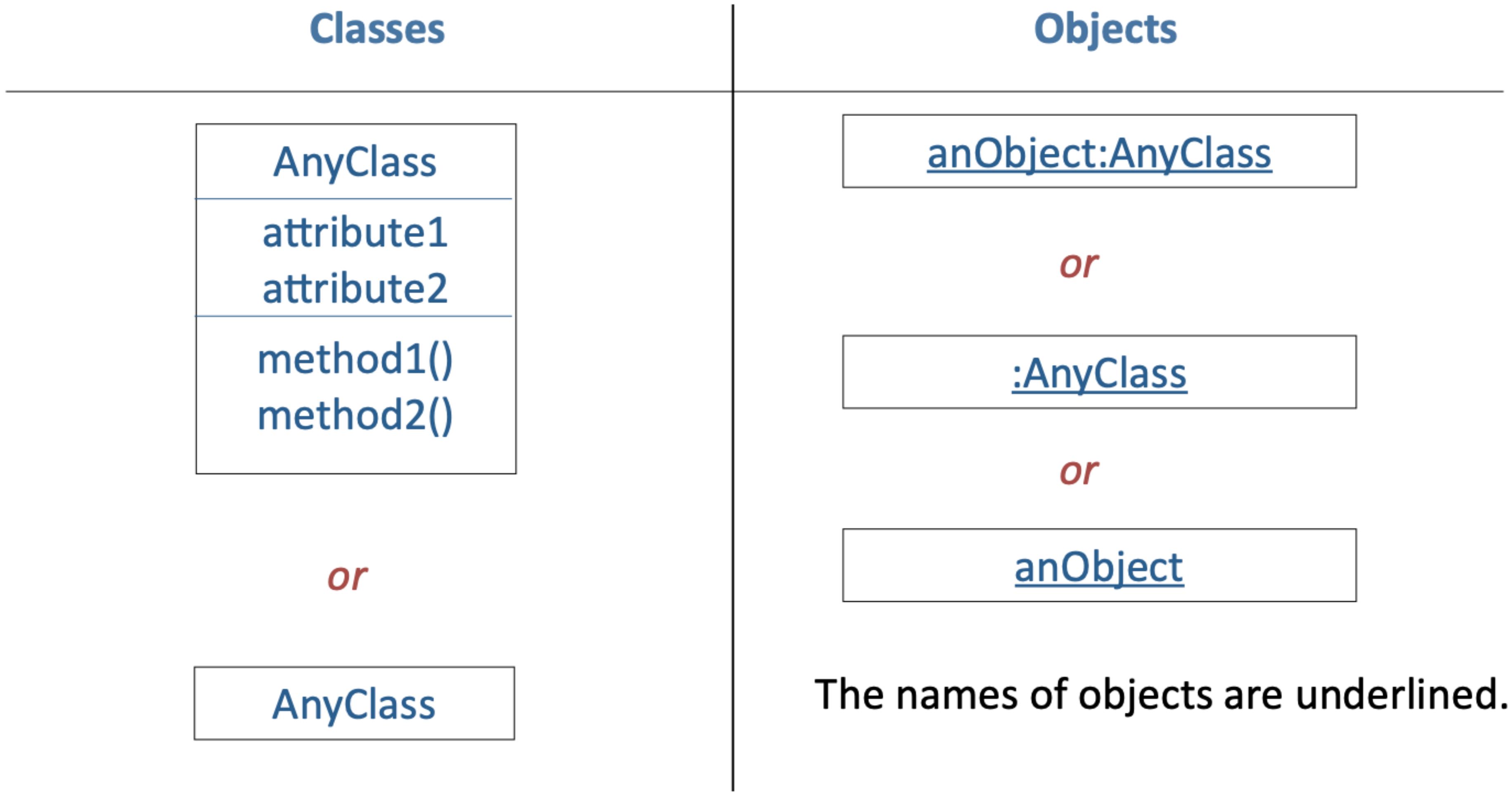
- **Interaction** diagrams
- Show set of objects and their relationships including messages that may be dispatched among them
- **Sequence diagram**: time ordering of messages

Interaction

Example: execution of an HTTP get command

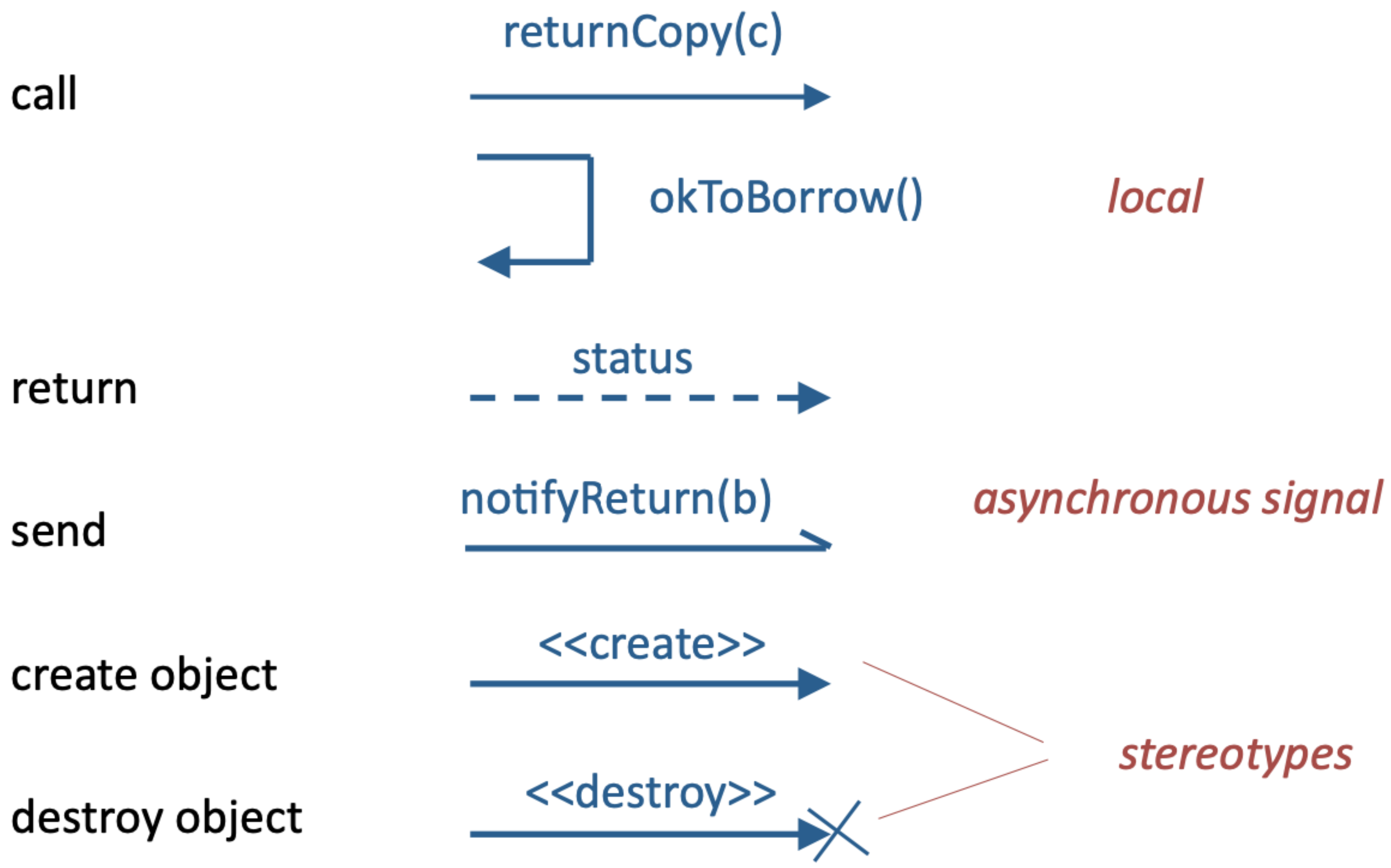


UML Notations for Class and Object

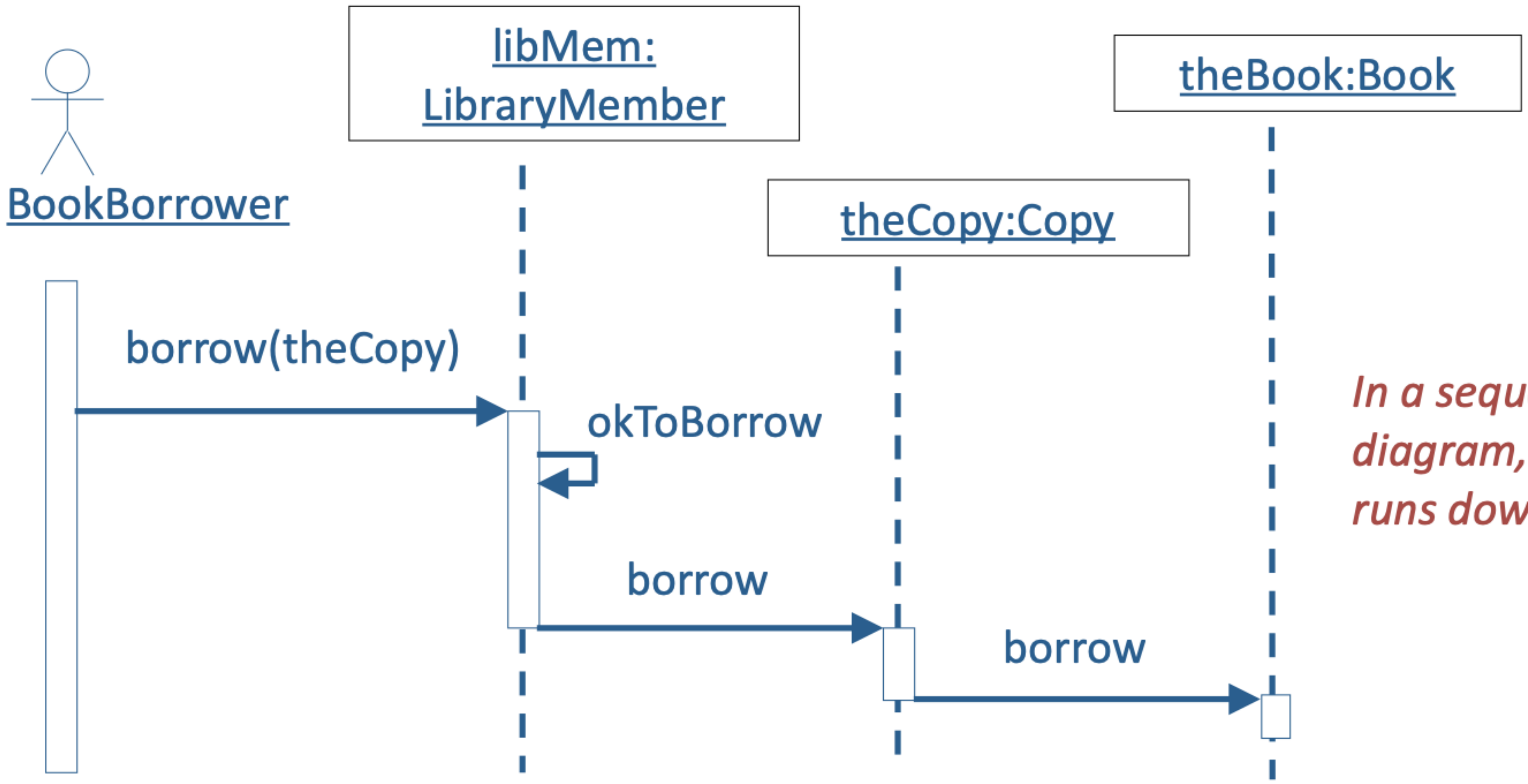


The names of objects are underlined.

Actions on Objects



Sequence Diagram: Borrow a copy of a book



In a sequence diagram, time runs downwards

Exercise: Sequence Diagram for Online Shopping

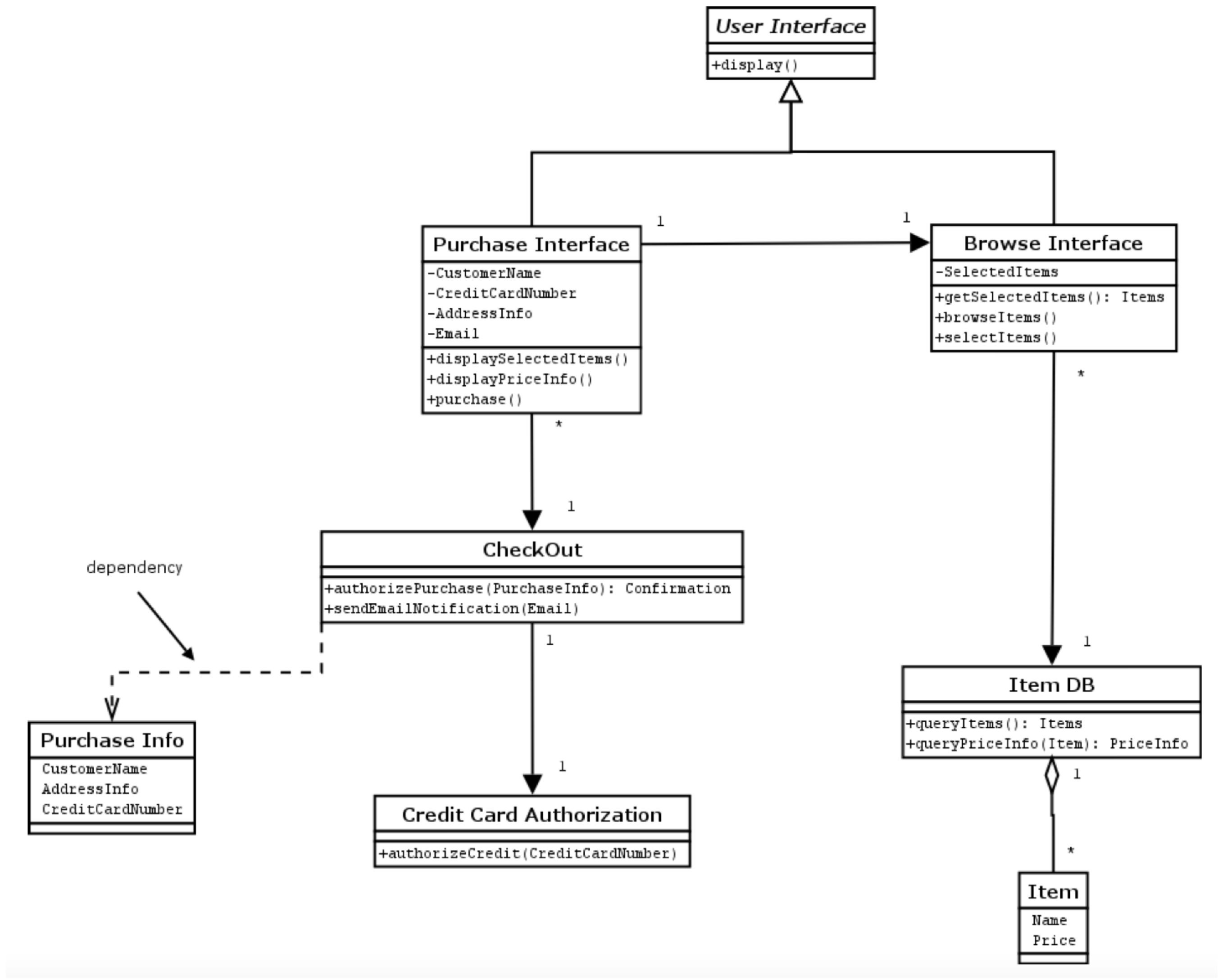
Use Case Meta-data for Online Shopping

1. Customer browses through catalog and select items to buy
2. Customer goes to checkout
3. Customer fills out shipping information
4. System presents full pricing information, including shipping information
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

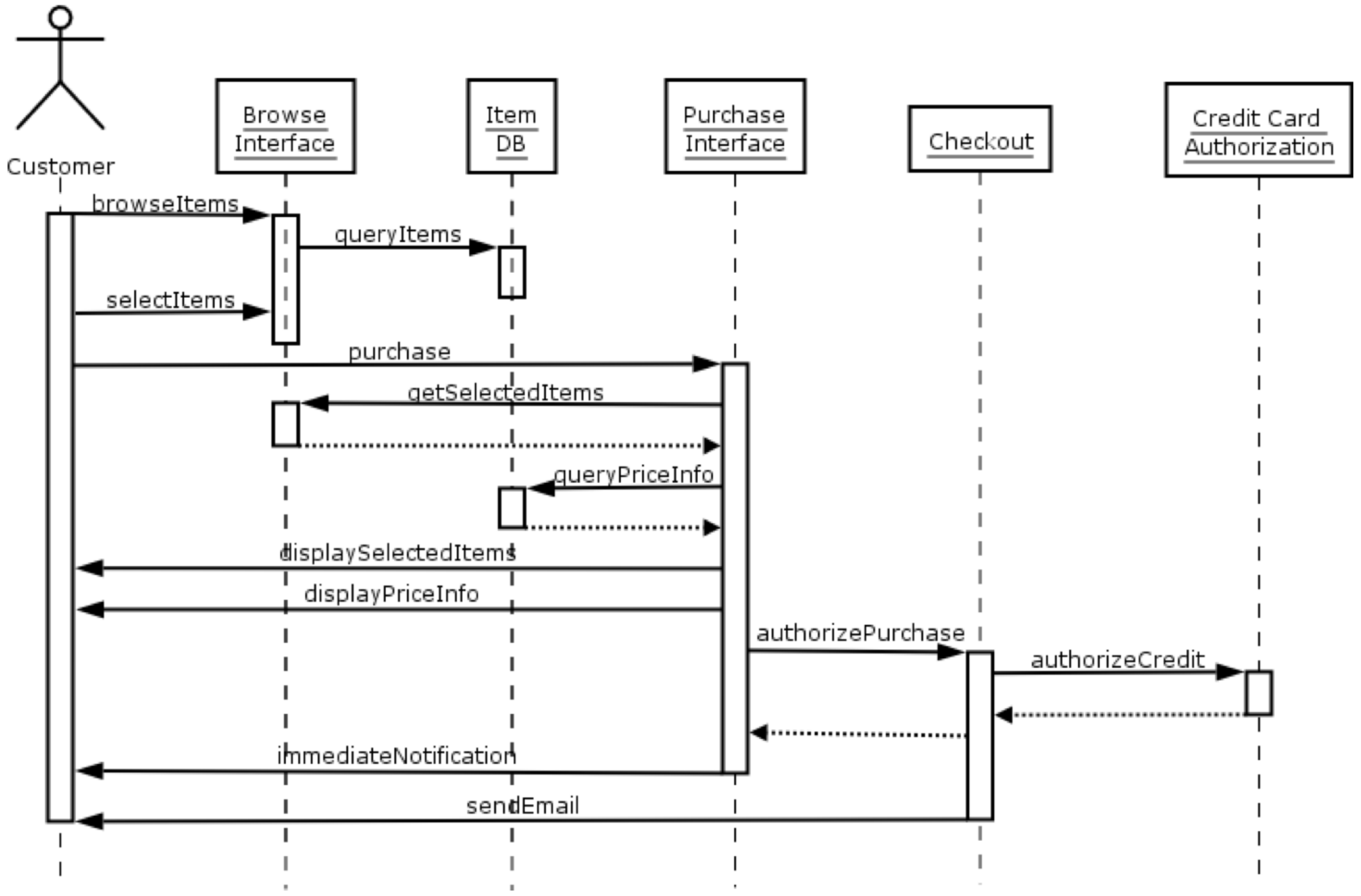
Exercise (2): Identify candidate classes for this scenario

- Application classes:
  - Customer
  - Catalog
  - Item/Product
  - Order
  - Purchased Item
- System classes:
  - User Interface
  - Purchase Interface
  - Browse Interface
  - Check Out
  - Credit Card Authorization
  - Item, Purchased Item

Class Diagram



Sequence Diagram



13 – Models for Program Design

(end of lecture)

ONE LOVE. ONE FUTURE.