Software Engineering I

Dr. Elham Mahmoudzadeh

Isfahan University of Technology

mahmoudzadeh@iut.ac.ir

202۱

Chapter ۶
Behavioral modeling(I)

Steps(I)

1. Preparing proposal
2. Requirements determination
   - User story
3. Abstract Business Process Modelling
4. Analysis
   - Functional Modelling
   - Structural Modelling
   - Behavioral Modelling

Steps(II)

5. Design
   - ➢ Optimization
   - ➢ Database Management
   - ➢ User Interface
   - ➢ Physical Architecture

Behavioral model

- Describe the internal dynamic aspects of an information system that supports the business processes in an organization.
- During analysis, behavioral models describe what the internal logic of the processes is without specifying how the processes are to be implemented.
- Later, in the design and implementation phases, the detailed design of the operations contained in the object is fully specified.

Inputs

- Use business process and functional models to describe the functional or external behavioral view of an information system.
- Use structural models to depict the internal structural or static view of an information system.

Types of behavioral models

- Behavioral models used to represent the underlying details of a business process portrayed by a use-case model, for example in UML, interaction diagrams (sequence and communication).
  - Interaction diagrams allow the analyst to model the distribution of the behavior of the system over the actors and objects in the system.
- Behavioral model is used to represent the changes that occur in the underlying data, for example in UML, behavioral state machines.

Focus on *what* the dynamic view of the evolving system is

and

not on *how* the dynamic aspect of the system will be implemented

Primary purposes of behavioral models

- Is to show how the underlying objects in a problem domain will work together to form a *collaboration* to support each of the *use cases*.
- Whereas structural models represent the objects and the relationships between them, behavioral models depict the internal view of the business process that a use case describes.
- The process can be shown by the interaction that takes place between the objects that collaborate to support a use case through the use of interaction (sequence and communication) diagrams.
- It is also possible to show the effect that the set of use cases that make up the system has on the objects in the system through the use of behavioral state machines.

Behavioral modeling

- Is an iterative process that iterates not only over the individual behavioral models but also over the functional and structural models.
- As the behavioral models are created, it is not unusual to make changes to the functional and structural models.

Objects, Operations, and Messages

- An object is an instantiation of a *class,* or thing about which we want to capture information.
  - If we were building an appointment system for a doctor’s office, classes might include doctor, patient, and appointment. The specific patients is Jim Maloney is considered objects—i.e., *instances* of the patient class.
- Each object has *attributes* that describe information about the object, such as a patient’s name.
- Each object also has *behaviors.* At this point in the development of the evolving system, the behaviors are described by *operations.* An operation is nothing more than an action that an object can perform.
  - For example, an appointment object can probably schedule a new appointment, delete an appointment, and locate the next available appointment.

Objects, Operations, and Messages(Cnt’d)

- Each object also can send and receive messages. *Messages* are information sent to objects to tell an object to execute one of its behaviors. Essentially, a message is a function or procedure call from one object to another object.
  - For example, if a patient is new to the doctor’s office, the system sends an insert message to the application. The patient object receives the instruction (the message) and does what it needs to do to insert the new patient into the system (the behavior).

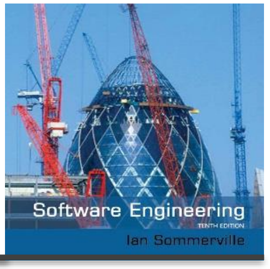
Sequence Diagram

- Is one of two types of interaction diagrams.
- They illustrate the objects that participate in a use case and the messages that pass between them over time for *one* use case.
- A sequence diagram is a *dynamic model* that shows the explicit sequence of messages that are passed between objects in a defined interaction.
- Because sequence diagrams emphasize the time-based ordering of the activity that takes place among a set of objects, they are very helpful for understanding real-time specifications and complex use cases.

Sequence diagram(Cnt’d)

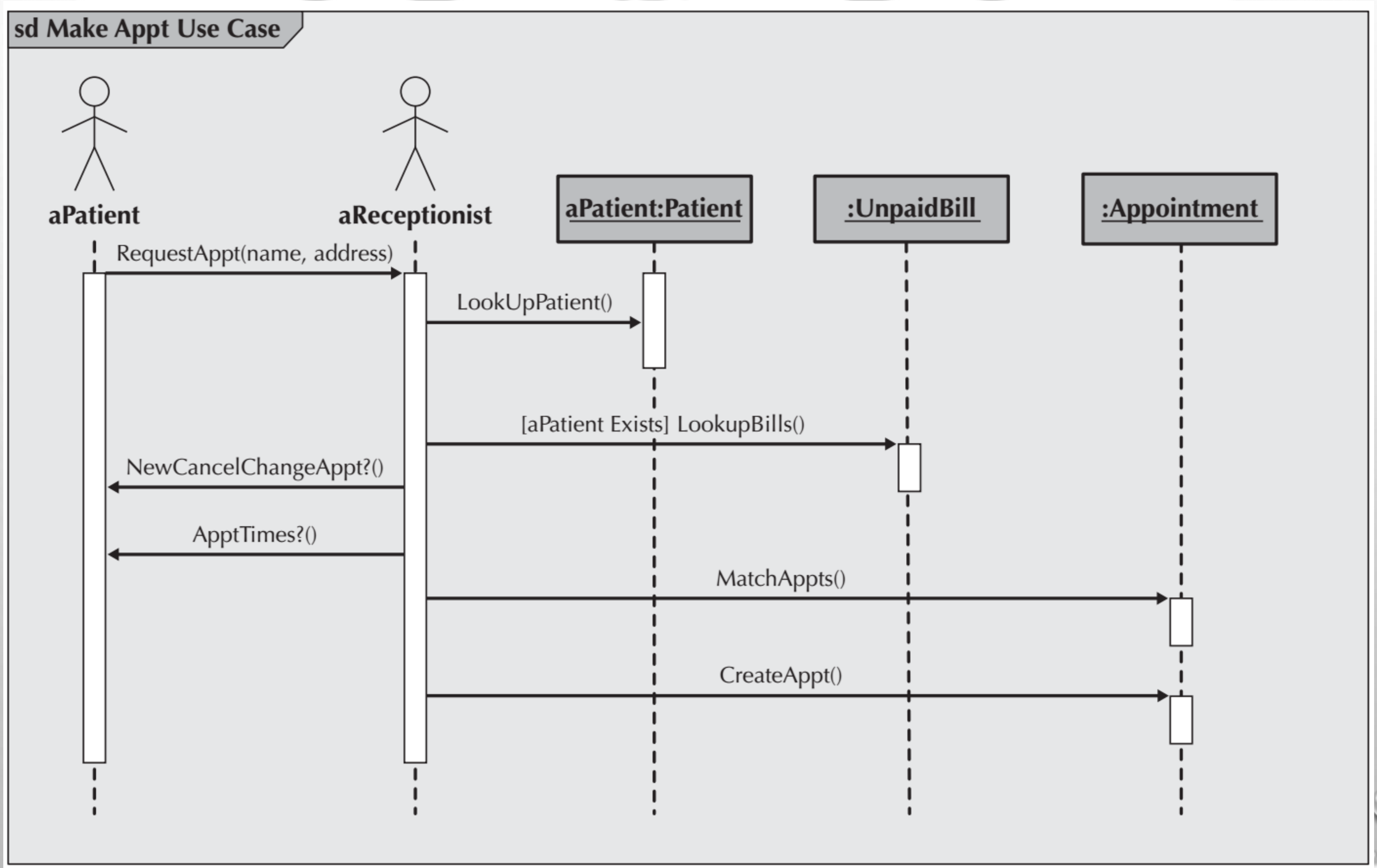
- The sequence diagram can be a *generic sequence diagram* that shows all possible scenarios for a use case, but usually each analyst develops a set of *instance sequence diagrams*, each of which depicts a single *scenario* within the use case.
- If you are interested in understanding the flow of control of a scenario by time, you should use a sequence diagram to depict this information.
- The diagrams are used throughout the analysis and design phases.
- However, the design diagrams are very implementation specific, often including database objects or specific user interface components as the objects.

Sequence diagrams

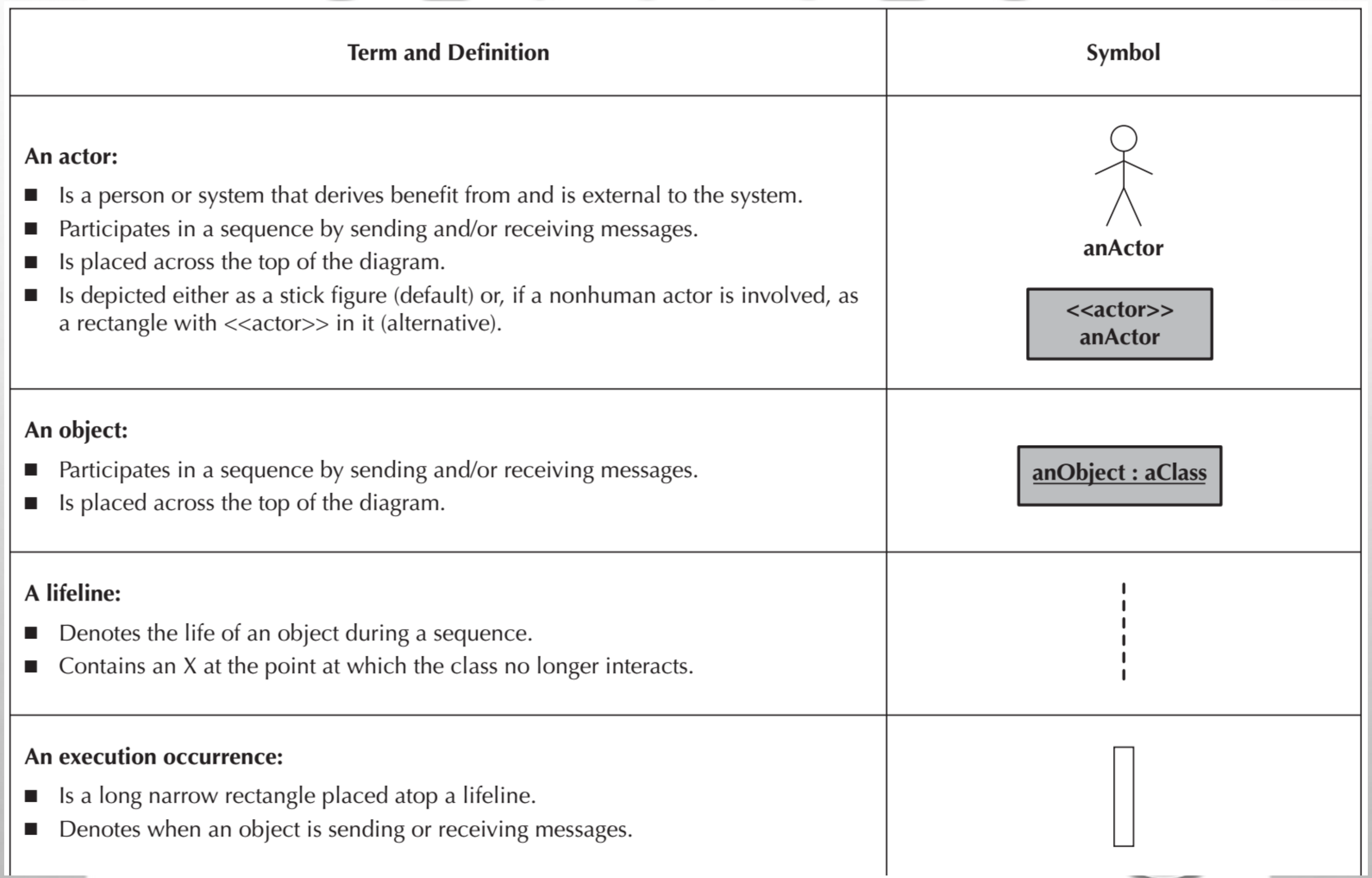


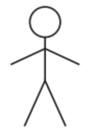
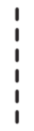
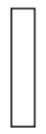
- ✧ Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
- ✧ A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- ✧ The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- ✧ Interactions between objects are indicated by annotated arrows.

An instance sequence diagram



Elements of a Sequence Diagram

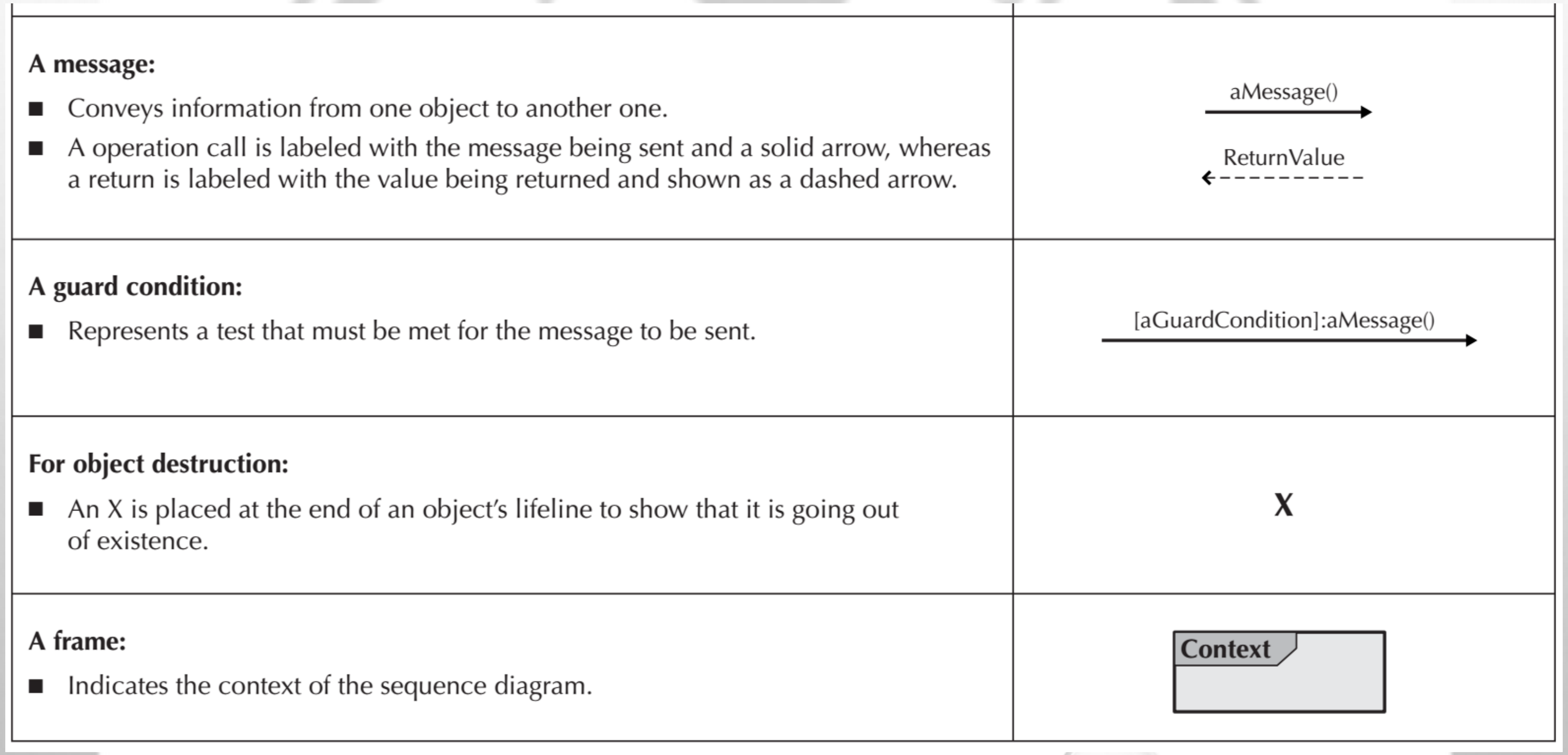


Term and Definition	Symbol
An actor: ■ Is a person or system that derives benefit from and is external to the system. ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram. ■ Is depicted either as a stick figure (default) or, if a nonhuman actor is involved, as a rectangle with <<actor>> in it (alternative).	 **anActor** **<<actor>> anActor**
An object: ■ Participates in a sequence by sending and/or receiving messages. ■ Is placed across the top of the diagram.	**<u>anObject : aClass</u>**
A lifeline: ■ Denotes the life of an object during a sequence. ■ Contains an X at the point at which the class no longer interacts.	
An execution occurrence: ■ Is a long narrow rectangle placed atop a lifeline. ■ Denotes when an object is sending or receiving messages.	

کل نکته فیلم sequence diagram:

انگار روی کلاس هاست ولی داینامیک
باز کردن یک فرایند نحوی عملیاتی شدن اون فرایند از طریق ارتباط بین کلاس هاست
لاجیک درونی رو بررسی میکنیم
ورودی:
اون فرایند و اکتیویتی دیاگرام و یوزکیس دیاگرام و کلاس دیاگرام
ایا استیت برنامه توی مسیرهای مختلف برای ما فرق میکنه؟ --> چرخه زندگی اون ابجکت به تصویر کشیده میشه
اینجا می خوایم بدونیم این فرایند چیه و نحوی ارتباط کلاس ها با هم اصلا وارد فاز پیاده سازی نمیشه اینجا فقط داریم تحلیل می کنیم
objects میشه کلاس هامون
مستطیل ایستاده ینی توی این زمان فعاله
اگر یک نمونه از اون کلاس باشه می نویسیم ولی اگر یک کلاس دیگه باشه با : اولش + اسم کلاس شروع میکنیم (این کلاسها باید حتما جز کلاس های توی کلاس دیاگرام باشه)
روی این مسیج ها اون تابعی از اون کلاس که باید فراخوانی بشه رو می نویسیم
اگر پارامتر باید ارسال بشه به بعدی توی پرانتز می ذاریم و باید نوشته بشه و شرط توی براکت

Elements of a sequence diagram(Cnt’d)



A message: ■ Conveys information from one object to another one. ■ A operation call is labeled with the message being sent and a solid arrow, whereas a return is labeled with the value being returned and shown as a dashed arrow.	aMessage() $\longrightarrow$ ReturnValue $\dashleftarrow$
A guard condition: ■ Represents a test that must be met for the message to be sent.	[aGuardCondition]:aMessage() $\longrightarrow$
For object destruction: ■ An X is placed at the end of an object’s lifeline to show that it is going out of existence.	**X**
A frame: ■ Indicates the context of the sequence diagram.	Context

Elements of a sequence diagram(Cnt’d)

- *Actors* and *objects* that participate in the sequence are placed across the top of the diagram using actor symbols from the use-case diagram. For each of the objects, the name of the class of which they are an instance is given after the object’s name.
- *A dotted line* runs vertically below each actor and object to denote the *lifeline* of the actors and objects over time. Sometimes an object creates a *temporary object;* in this case, an X is placed at the end of the lifeline at the point where the object is destroyed. When objects continue to exist in the system after they are used in the sequence diagram, then the lifeline continues to the bottom of the diagram.
- A thin rectangular box, called the *execution occurrence*, is overlaid onto the lifeline to show when the classes are sending and receiving messages.

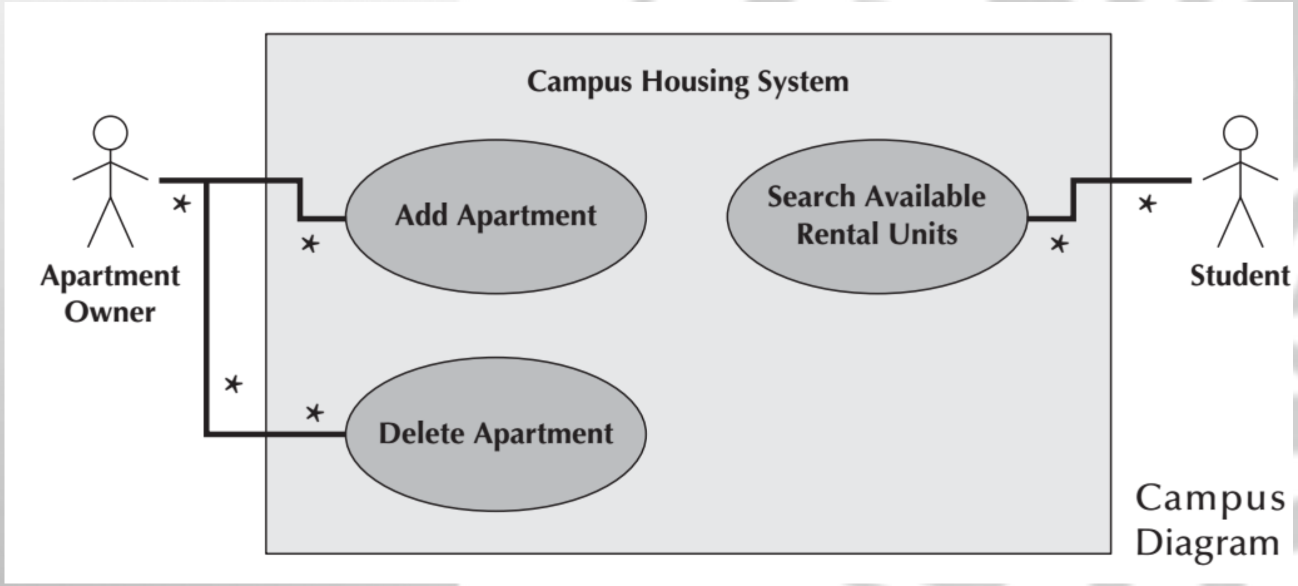
Elements of a sequence diagram(Cnt’d)

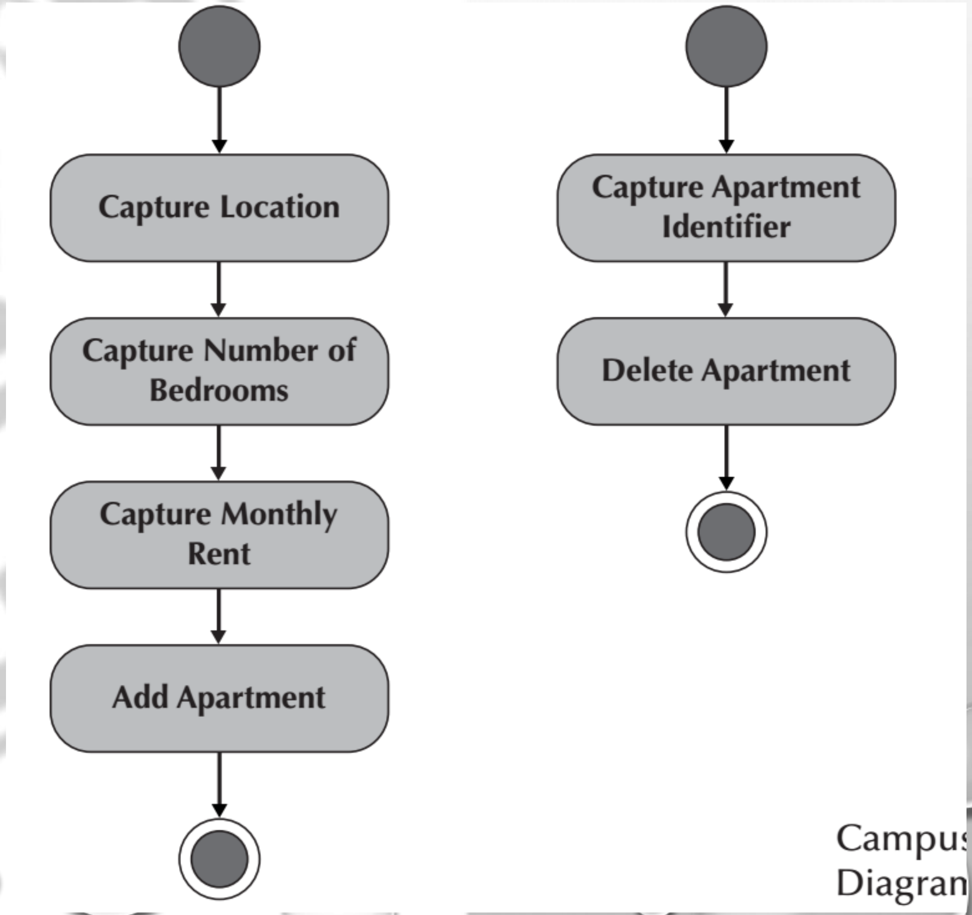
- A *message* is a communication between objects that conveys information with the expectation that activity will ensue.
- Two types of messages: operation call and return. *Operation call messages* passed between objects are shown using solid lines connecting two objects with an arrow on the line showing which way the message is being passed. Argument values for the message are placed in parentheses next to the message’s name. The order of messages goes from the top to the bottom of the page, so messages located higher on the diagram represent messages that occur earlier on in the sequence, versus the lower messages that occur later. A *return message* is depicted as a dashed line with an arrow on the end of the line portraying the direction of the return. The information being returned is used to label the arrow.

Elements of a sequence diagram(Cnt'd)

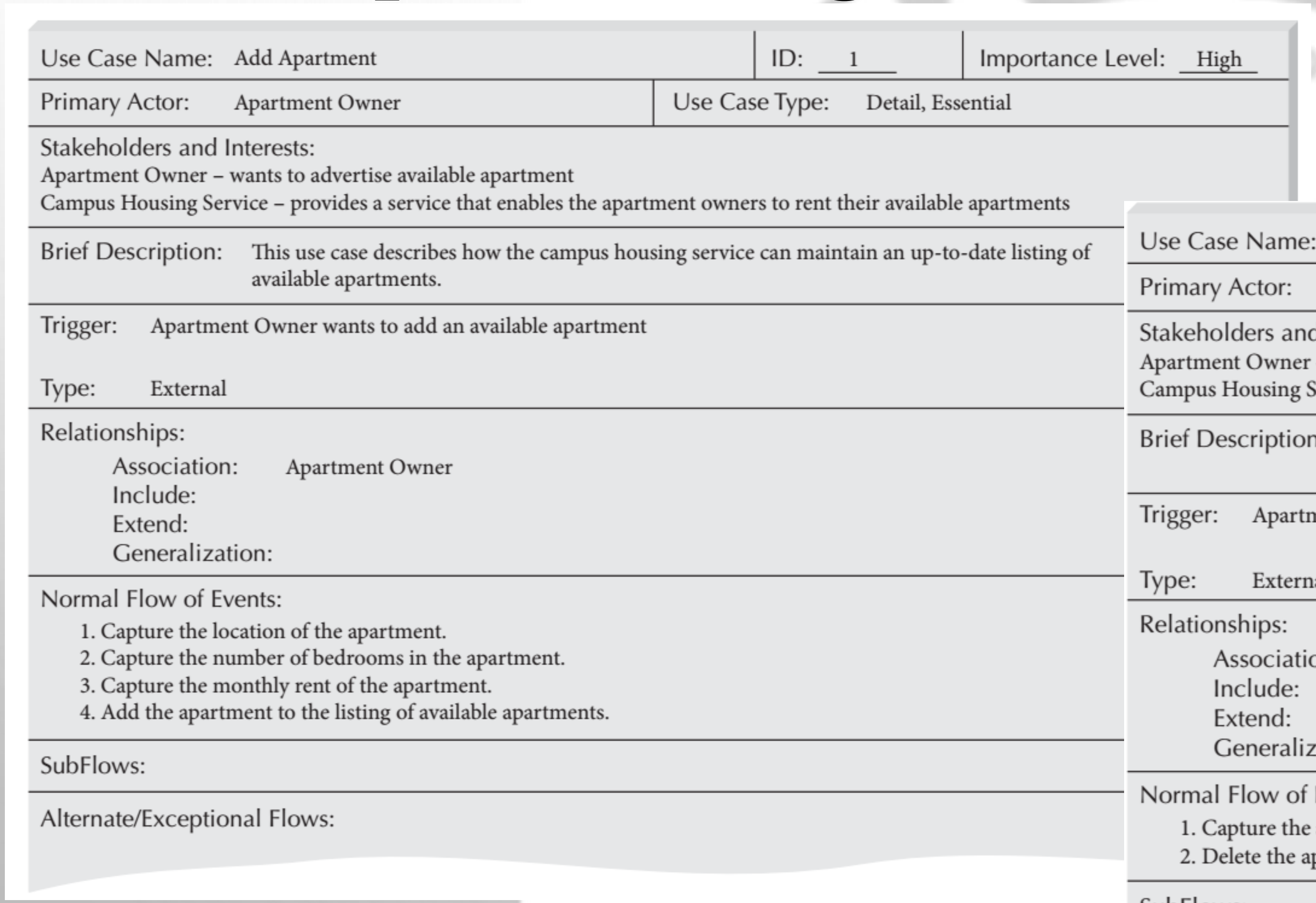
- At times a message is sent only if a *condition* is met. In those cases, the condition is placed between a set of brackets, []. The condition is placed in front of the message name.
- An object can send a message to itself. This is known as *self-delegation*.

Campus Housing Service Functional Models





Campus Housing Service Functional Models(Cnt’d)



Use Case Name: Add Apartment | ID: 1 | Importance Level: High

Primary Actor: Apartment Owner | Use Case Type: Detail, Essential

Stakeholders and Interests:
Apartment Owner – wants to advertise available apartment
Campus Housing Service – provides a service that enables the apartment owners to rent their available apartments

Brief Description: This use case describes how the campus housing service can maintain an up-to-date listing of available apartments.

Trigger: Apartment Owner wants to add an available apartment

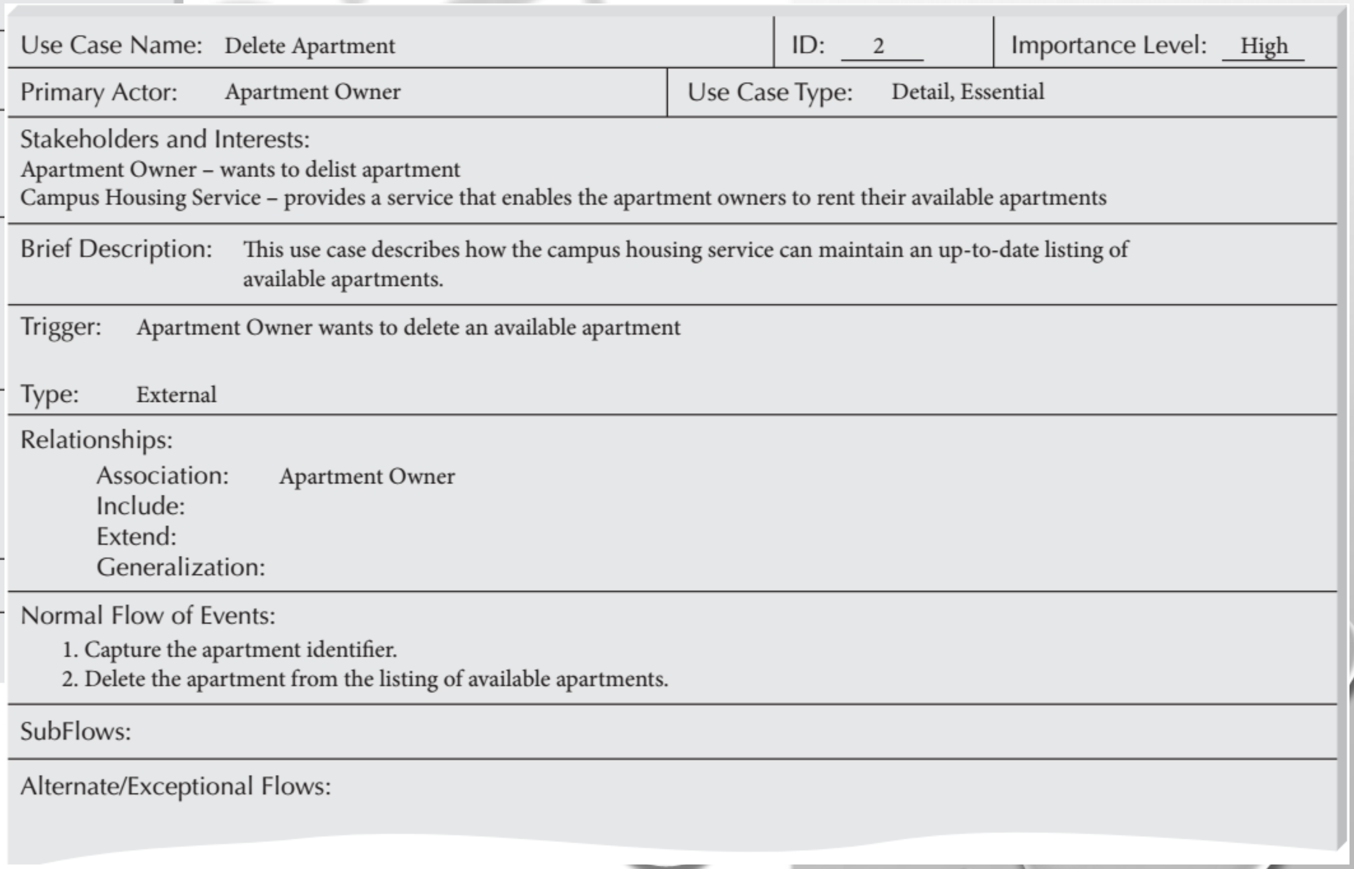
Type: External

Relationships:
Association: Apartment Owner
Include:
Extend:
Generalization:

Normal Flow of Events:
1. Capture the location of the apartment.
2. Capture the number of bedrooms in the apartment.
3. Capture the monthly rent of the apartment.
4. Add the apartment to the listing of available apartments.

SubFlows:

Alternate/Exceptional Flows:



Use Case Name: Delete Apartment | ID: 2 | Importance Level: High

Primary Actor: Apartment Owner | Use Case Type: Detail, Essential

Stakeholders and Interests:
Apartment Owner – wants to delist apartment
Campus Housing Service – provides a service that enables the apartment owners to rent their available apartments

Brief Description: This use case describes how the campus housing service can maintain an up-to-date listing of available apartments.

Trigger: Apartment Owner wants to delete an available apartment

Type: External

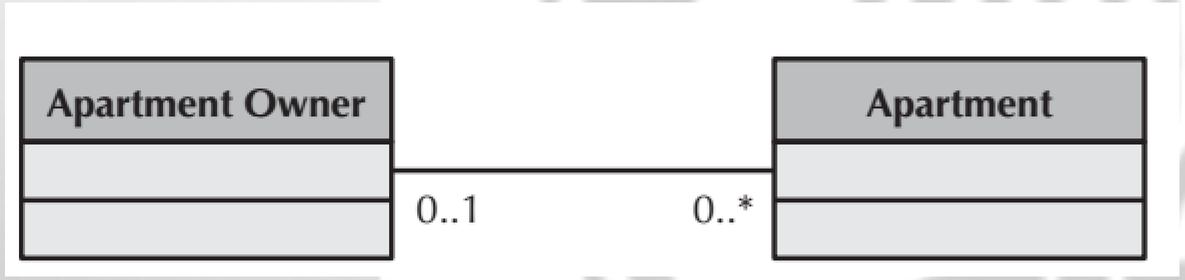
Relationships:
Association: Apartment Owner
Include:
Extend:
Generalization:

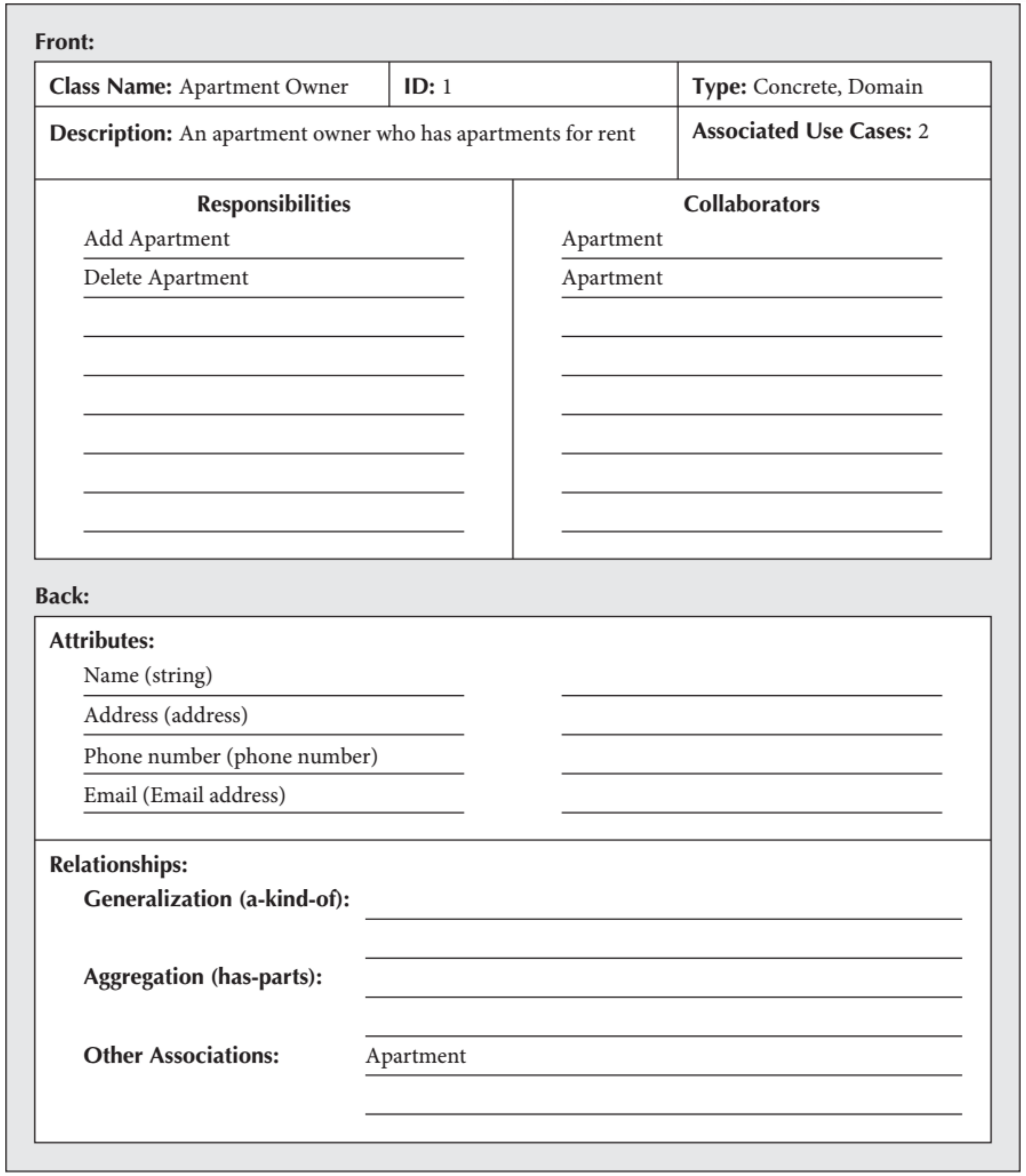
Normal Flow of Events:
1. Capture the apartment identifier.
2. Delete the apartment from the listing of available apartments.

SubFlows:

Alternate/Exceptional Flows:

Campus Housing Service structural Models





Front:

Class Name: Apartment Owner	**ID:** 1	**Type:** Concrete, Domain
Description: An apartment owner who has apartments for rent		**Associated Use Cases:** 2

Responsibilities	Collaborators
Add Apartment	Apartment
Delete Apartment	Apartment

Back:

Attributes:

Name (string)

Address (address)

Phone number (phone number)

Email (Email address)

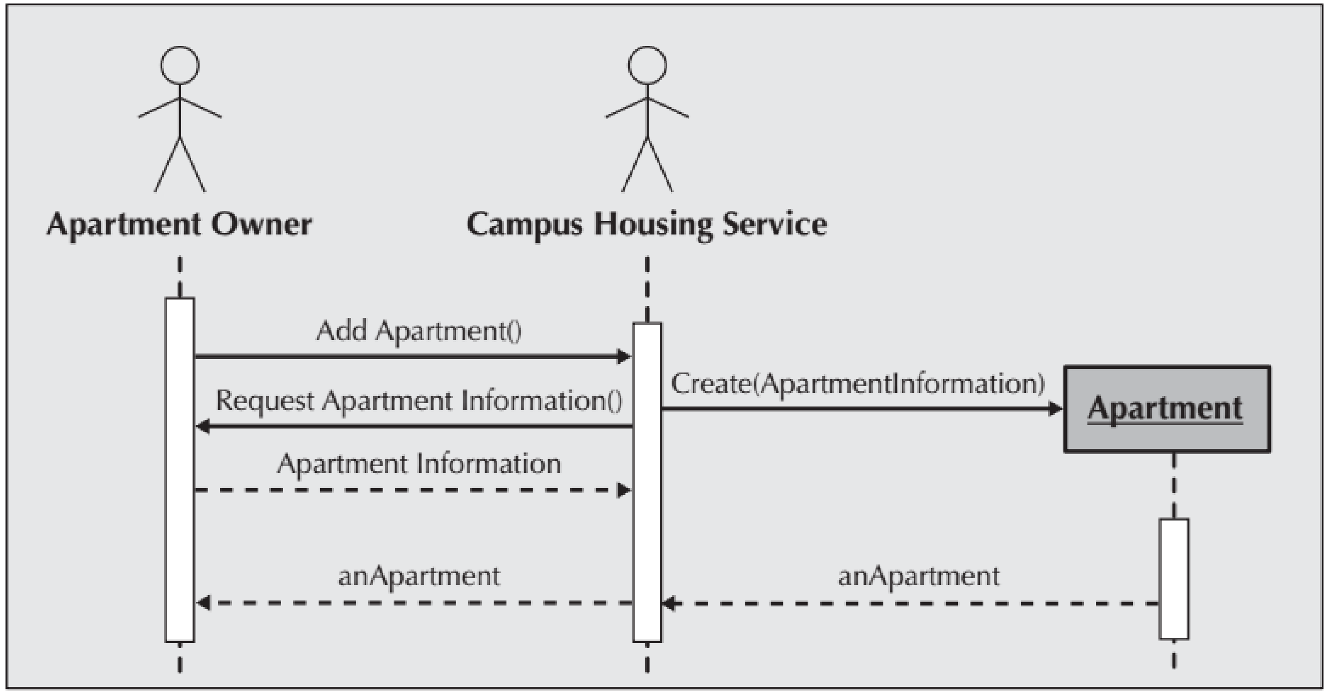
Relationships:

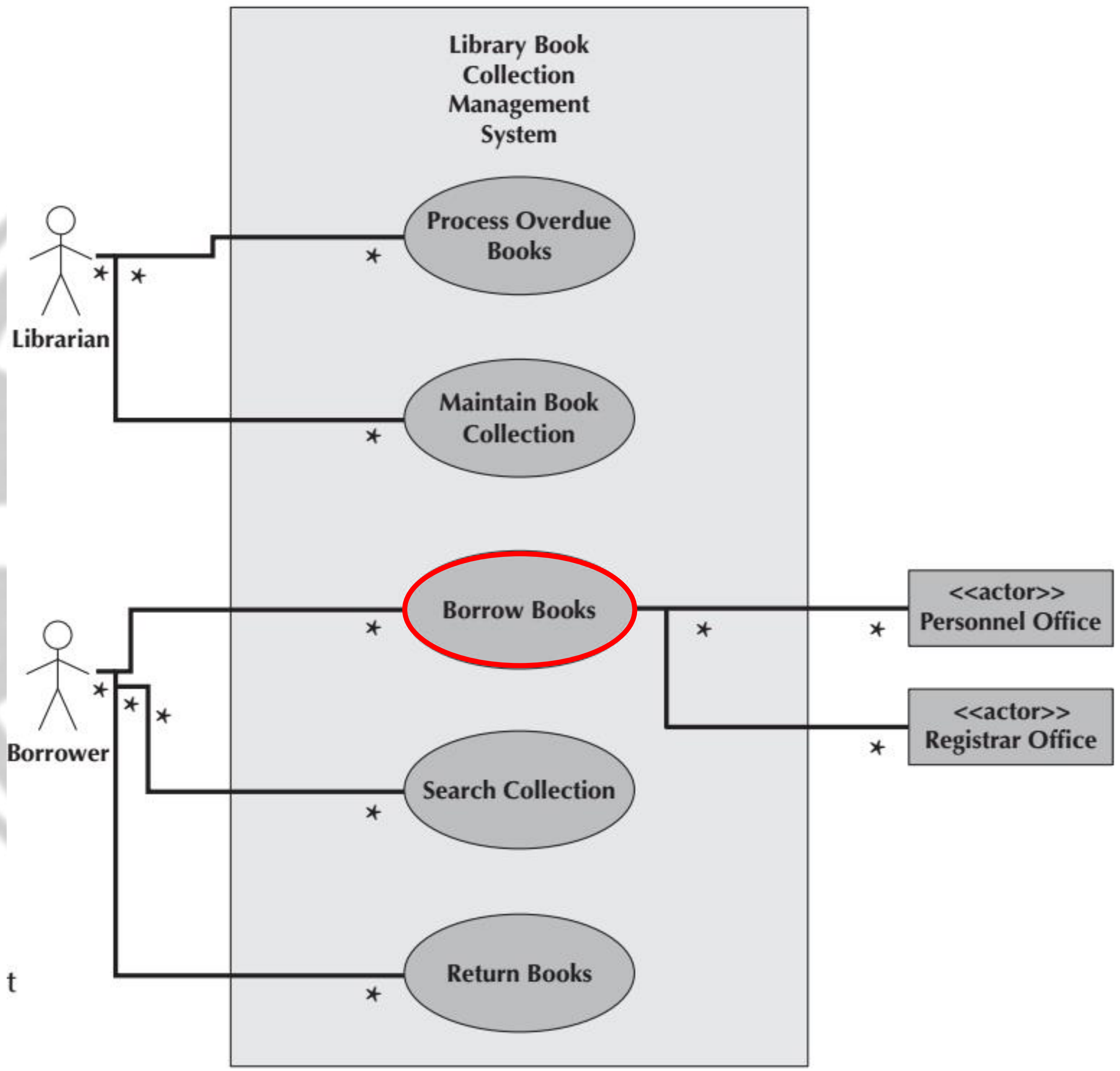
Generalization (a-kind-of):

Aggregation (has-parts):

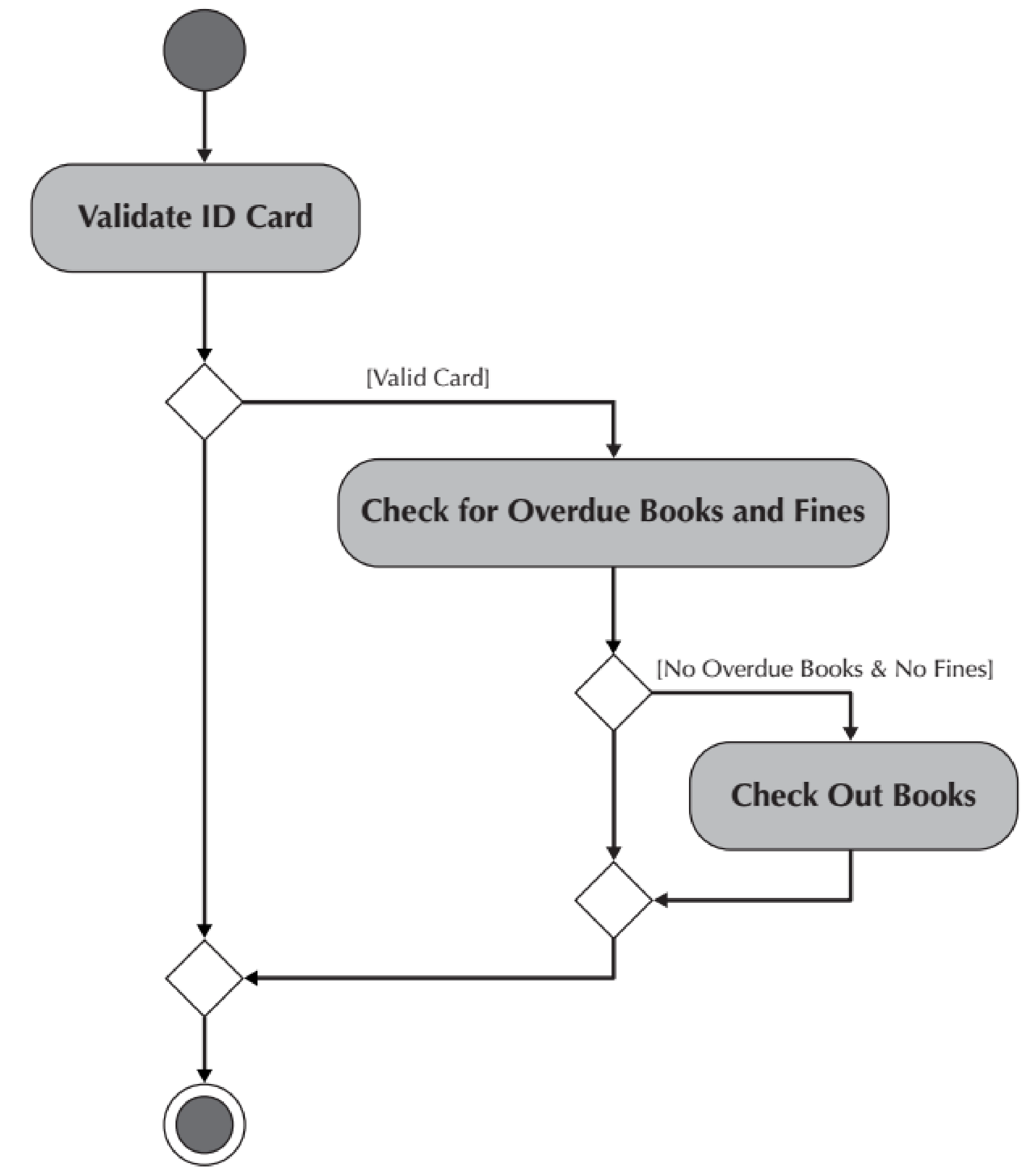
Other Associations: Apartment

Sequence Diagram for the Add Apartment Use Case

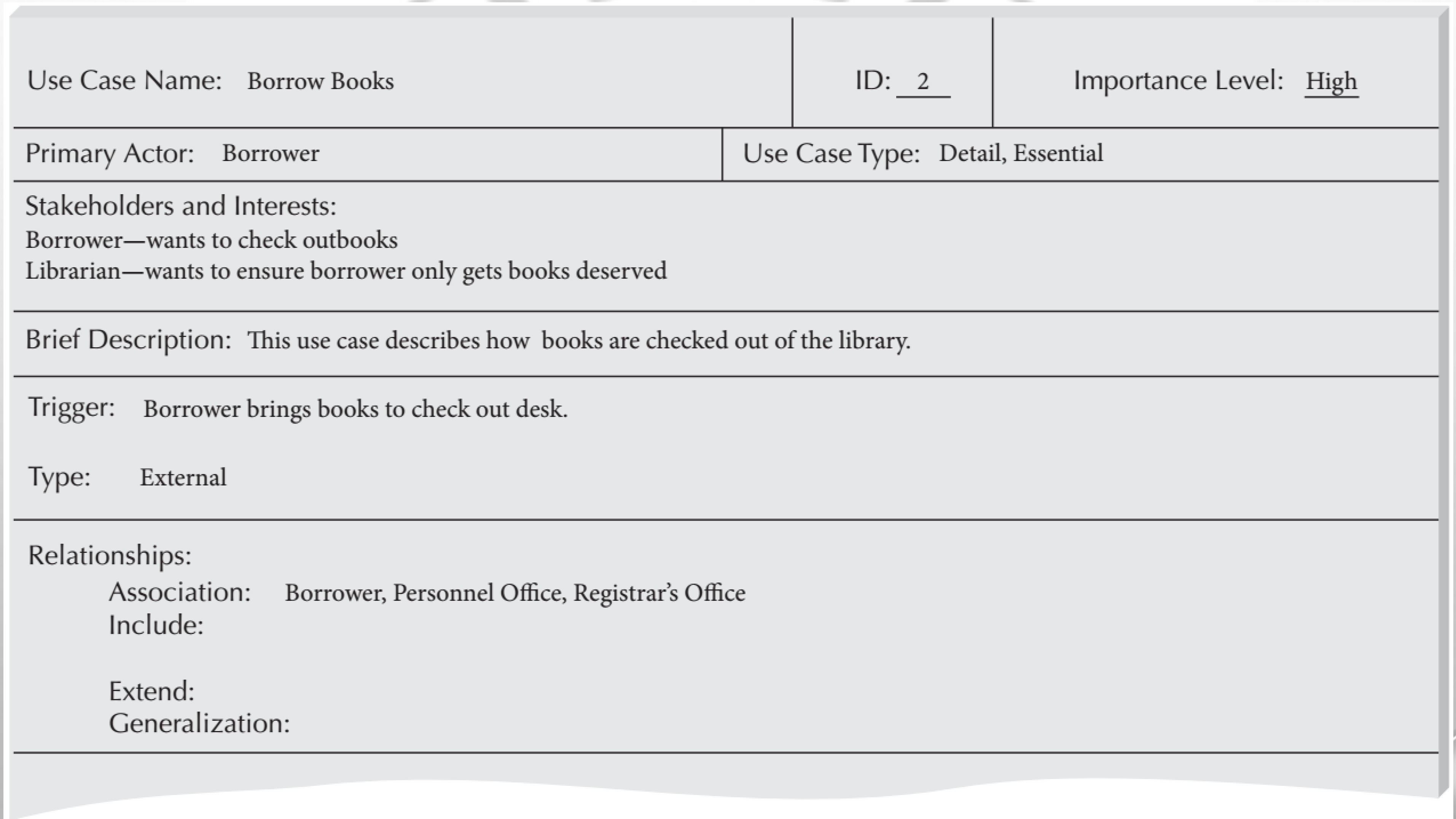



Library Book Collection Management System
Librarian
Borrower
Process Overdue Books
Maintain Book Collection
Borrow Books
Search Collection
Return Books
<<actor>> Personnel Office
<<actor>> Registrar Office
*

t


Validate ID Card
[Valid Card]
Check for Overdue Books and Fines
[No Overdue Books & No Fines]
Check Out Books

Overview Description for the Borrow Books Use Case



Use Case Name: Borrow Books	ID: 2	Importance Level: High
Primary Actor: Borrower	Use Case Type: Detail, Essential	

Stakeholders and Interests:
Borrower—wants to check outbooks
Librarian—wants to ensure borrower only gets books deserved

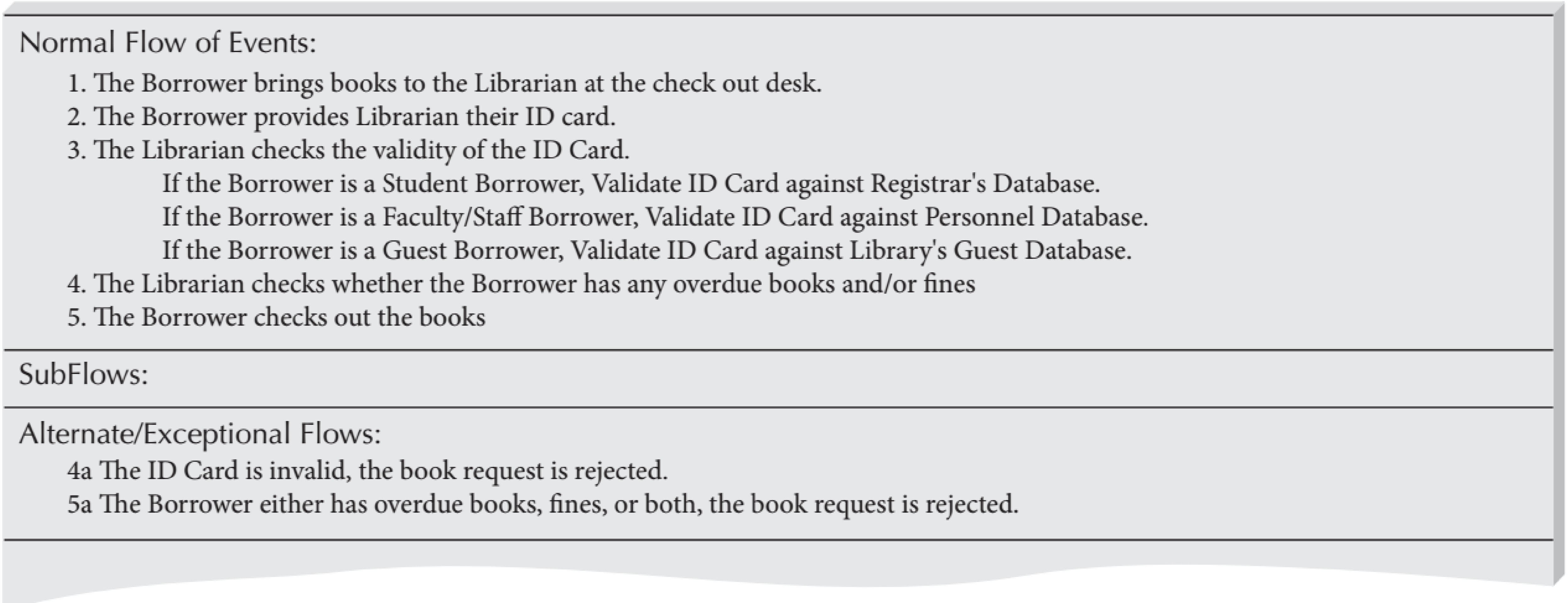
Brief Description: This use case describes how books are checked out of the library.

Trigger: Borrower brings books to check out desk.

Type: External

Relationships:
- Association: Borrower, Personnel Office, Registrar's Office
- Include:
- Extend:
- Generalization:

Flow Descriptions for the Borrow Books Use Case



Normal Flow of Events:

1. The Borrower brings books to the Librarian at the check out desk.
2. The Borrower provides Librarian their ID card.
3. The Librarian checks the validity of the ID Card.
    - If the Borrower is a Student Borrower, Validate ID Card against Registrar's Database.
    - If the Borrower is a Faculty/Staff Borrower, Validate ID Card against Personnel Database.
    - If the Borrower is a Guest Borrower, Validate ID Card against Library's Guest Database.
4. The Librarian checks whether the Borrower has any overdue books and/or fines
5. The Borrower checks out the books

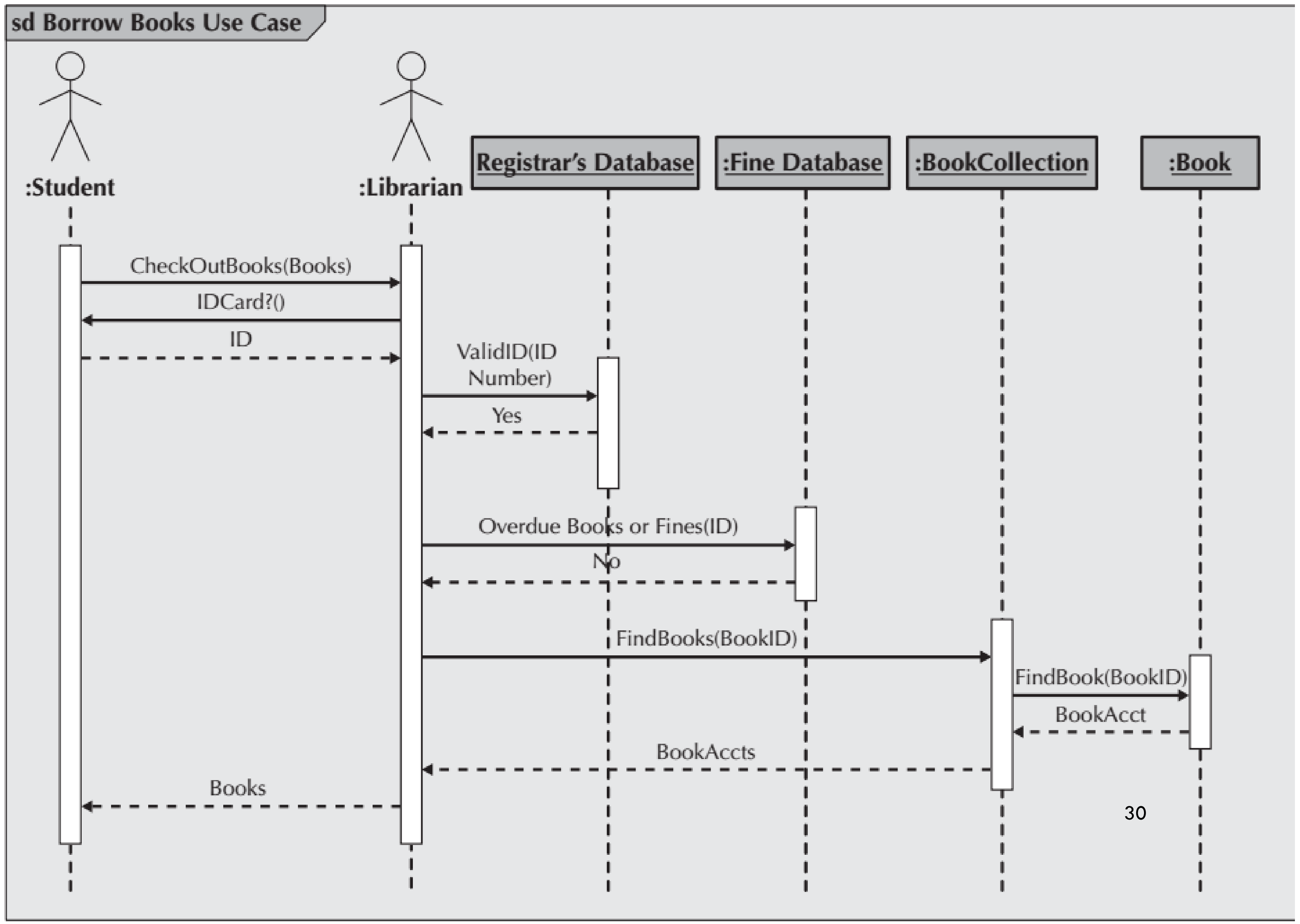
SubFlows:

Alternate/Exceptional Flows:

4a The ID Card is invalid, the book request is rejected.

5a The Borrower either has overdue books, fines, or both, the book request is rejected.

Sequence Diagram of the **Borrow Books Use Case** for Students with a Valid ID and No Overdue Books or Fines



Reference

- **Dennis, Wixon, Tegarden**, **“System Analysis and Design, An Object Oriented Approach with UML”, 5th Edition, 2015.**