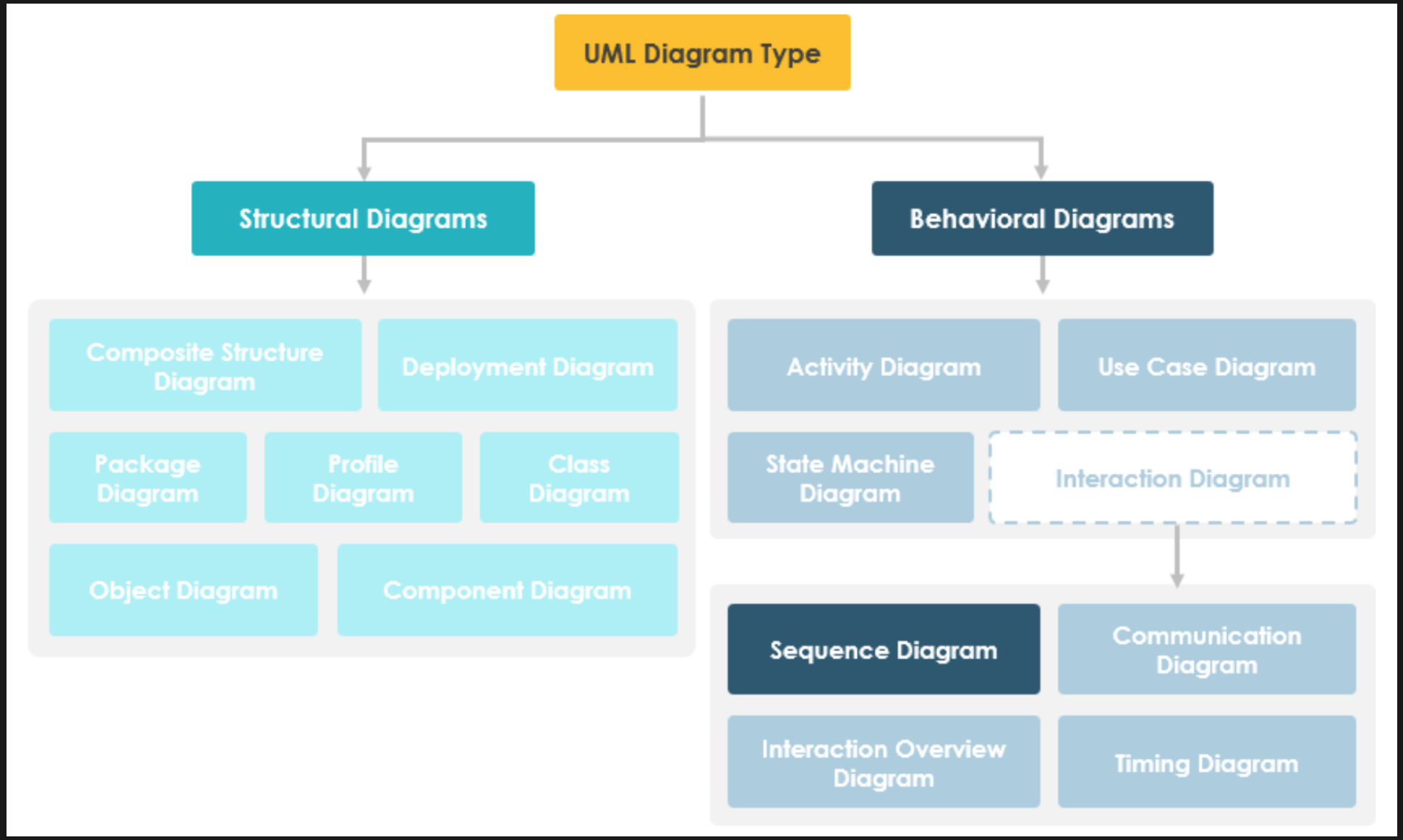
# DIAGRAMAS DE SECUENCIA

# DIAGRAMAS DE SECUENCIA



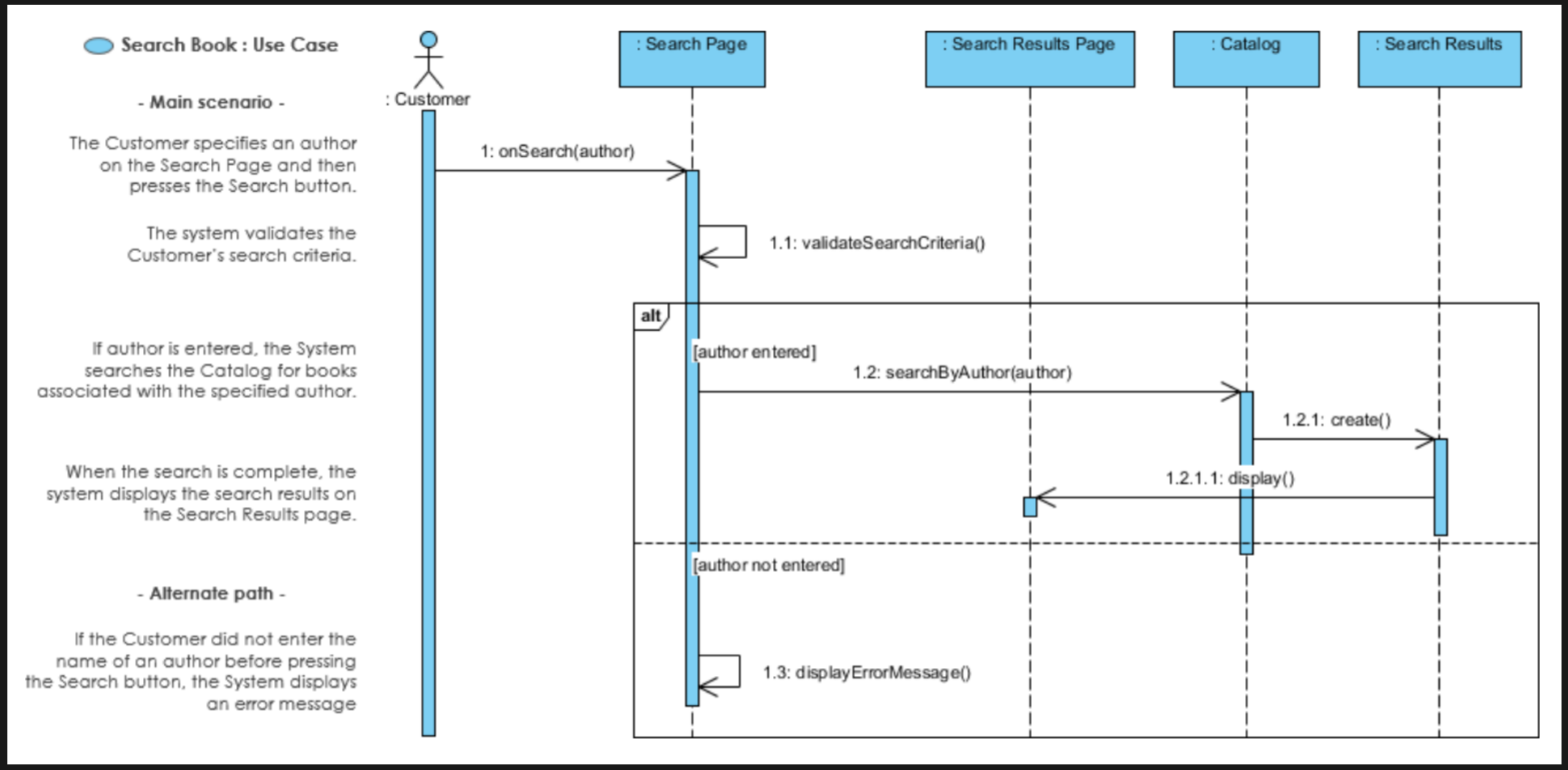
# DIAGRAMAS DE SECUENCIA

Es un diagrama de interacción que muestra cómo ocurren las operaciones entre objetos

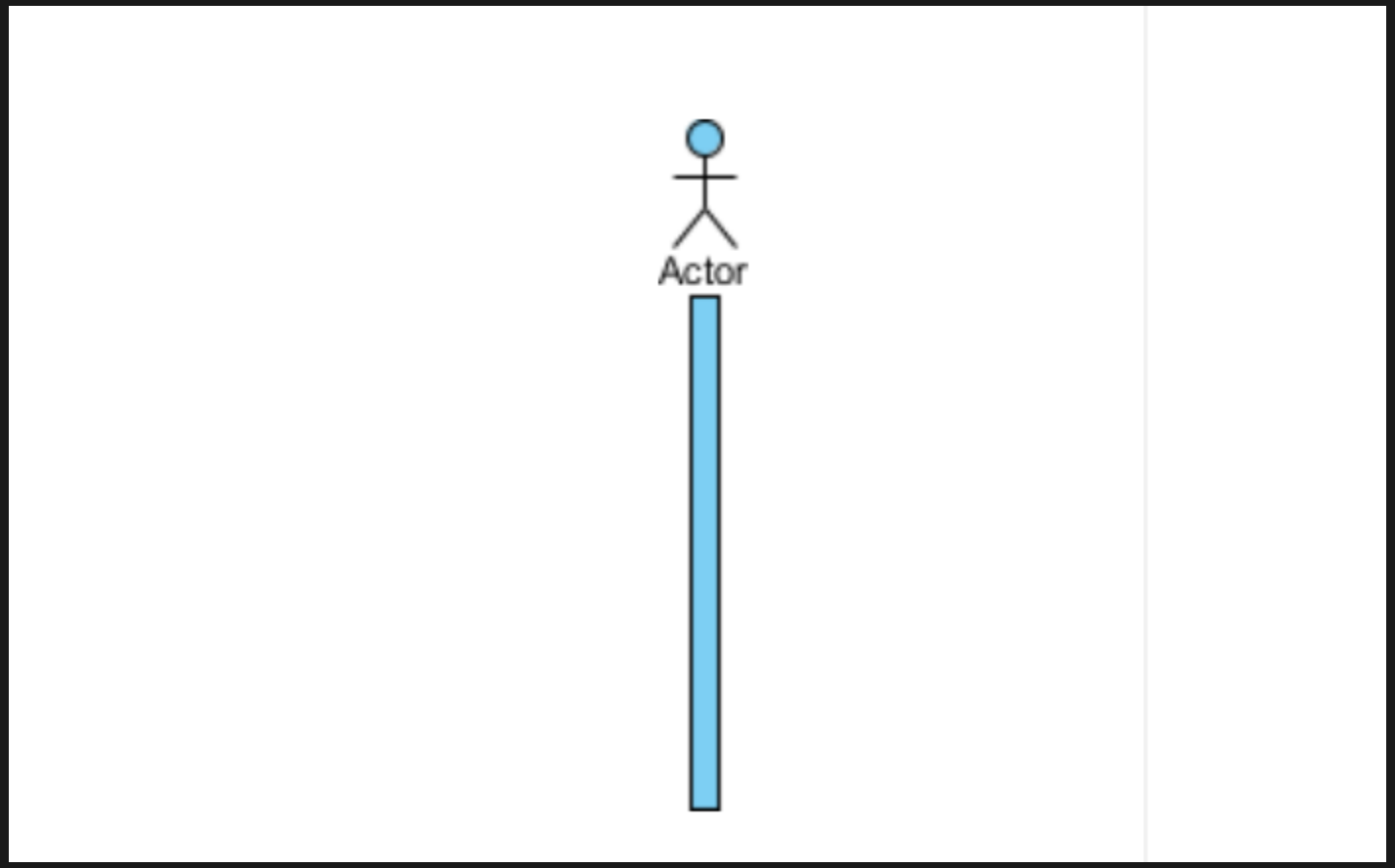
# DIAGRAMAS DE SECUENCIA

- Objetos
- Actores
- Línea de Vida
- Mensajes

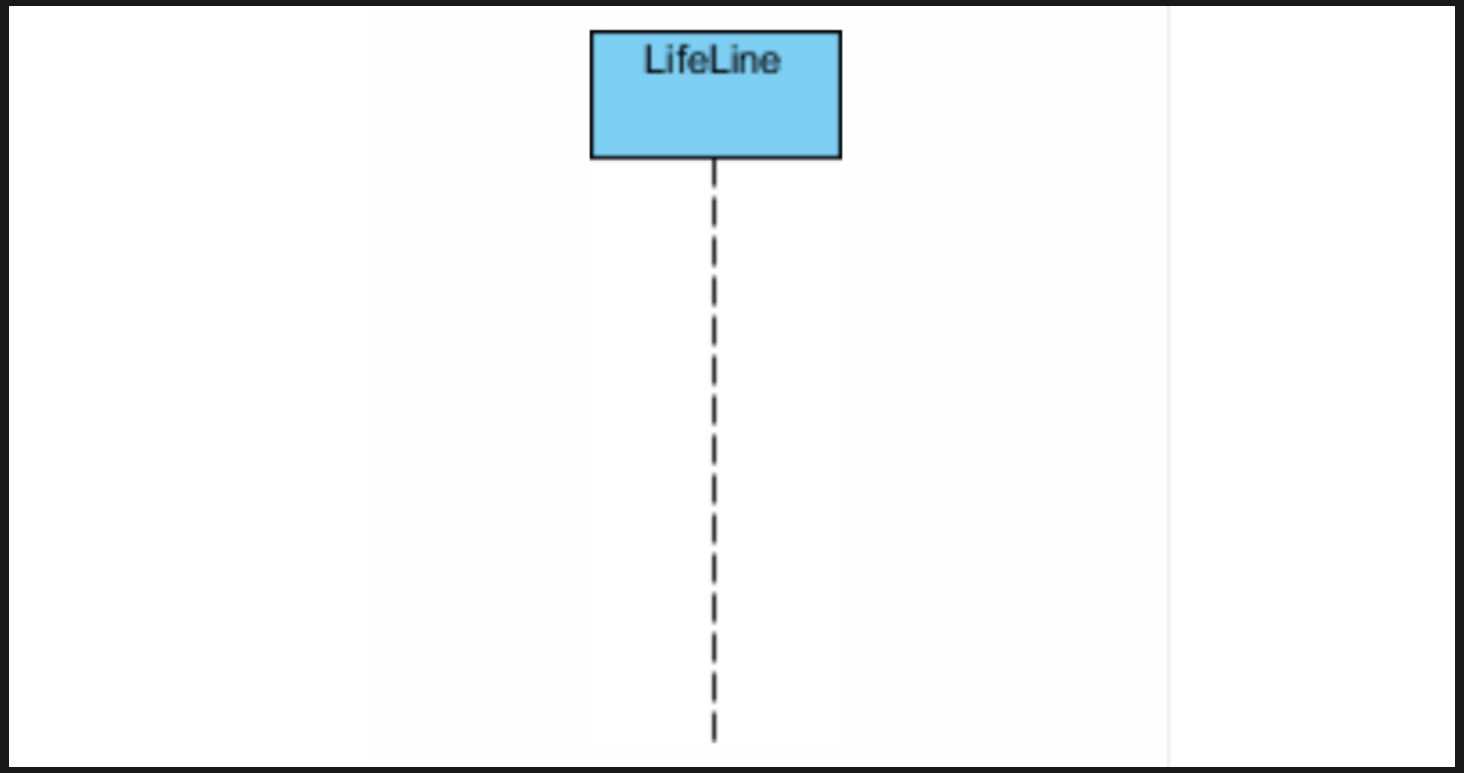
# DIAGRAMAS DE SECUENCIA



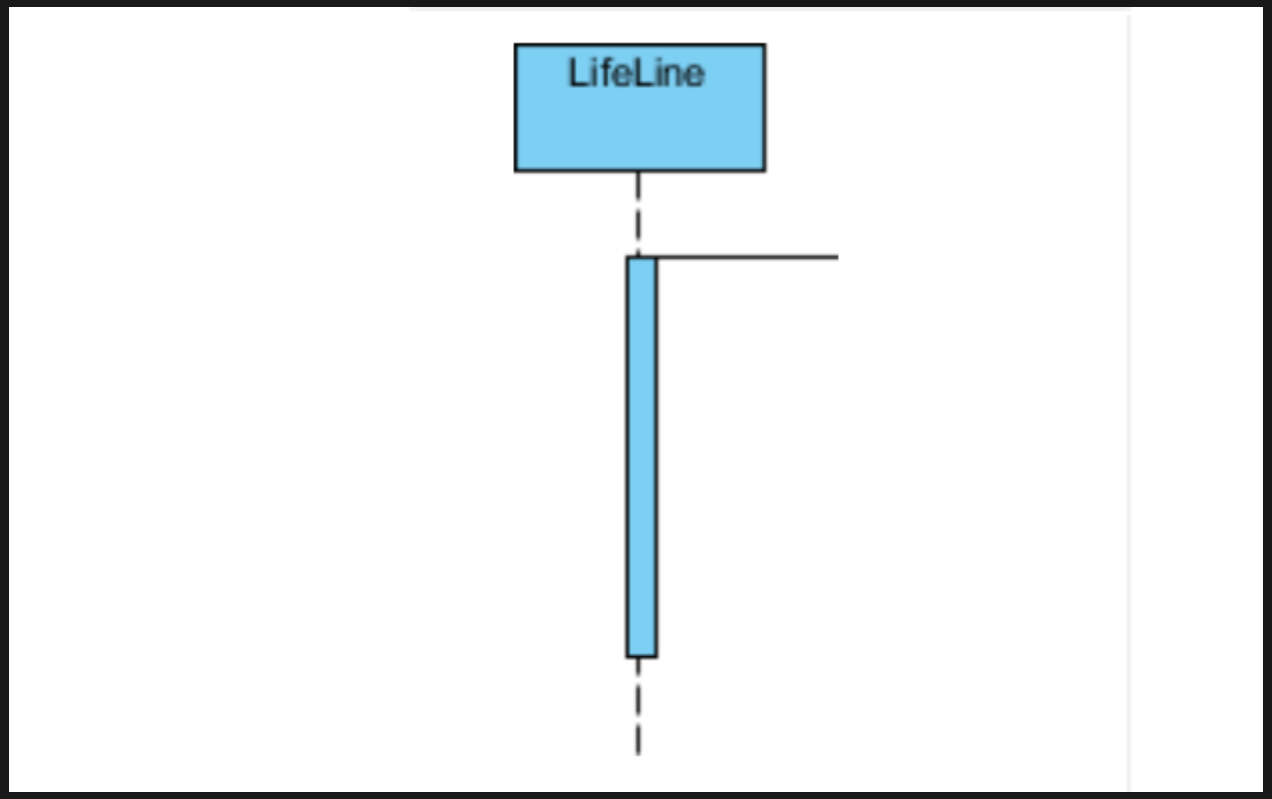
# ACTOR



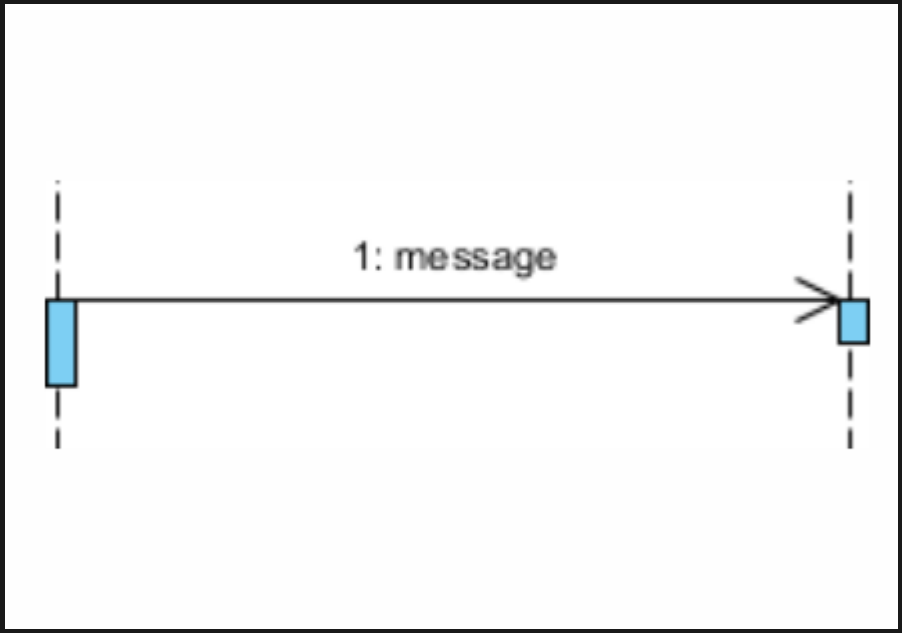
# LÍNEA DE VIDA



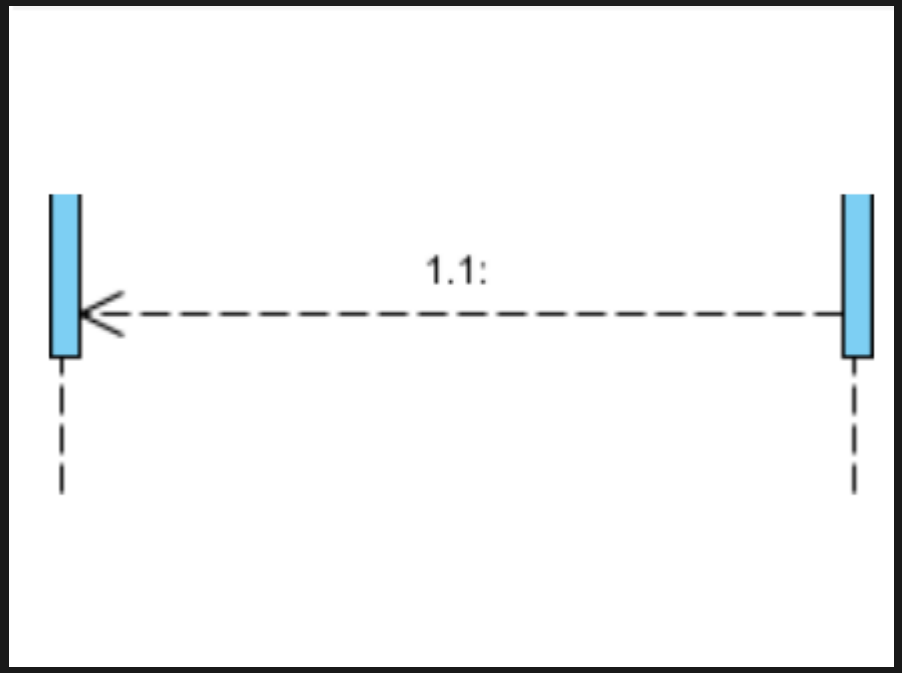
# ACTIVACIONES



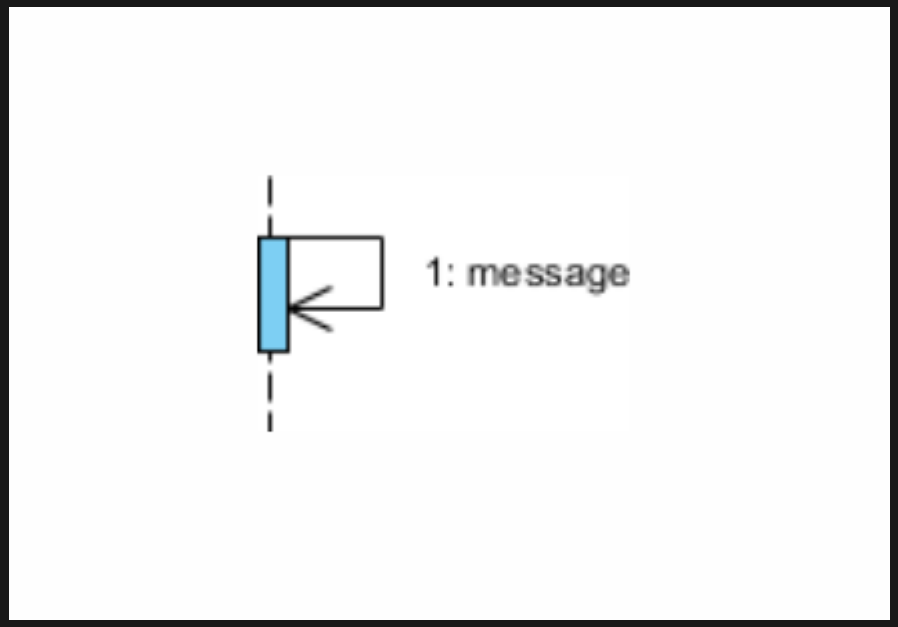
# MENSAJES



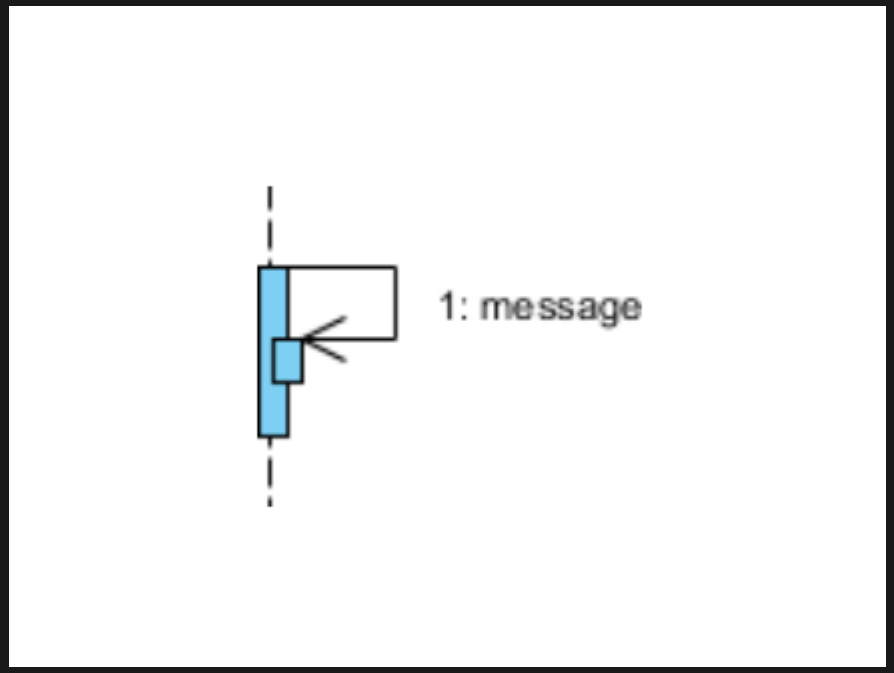
# MENSAJES DE RETORNO



# AUTO MENSAJES

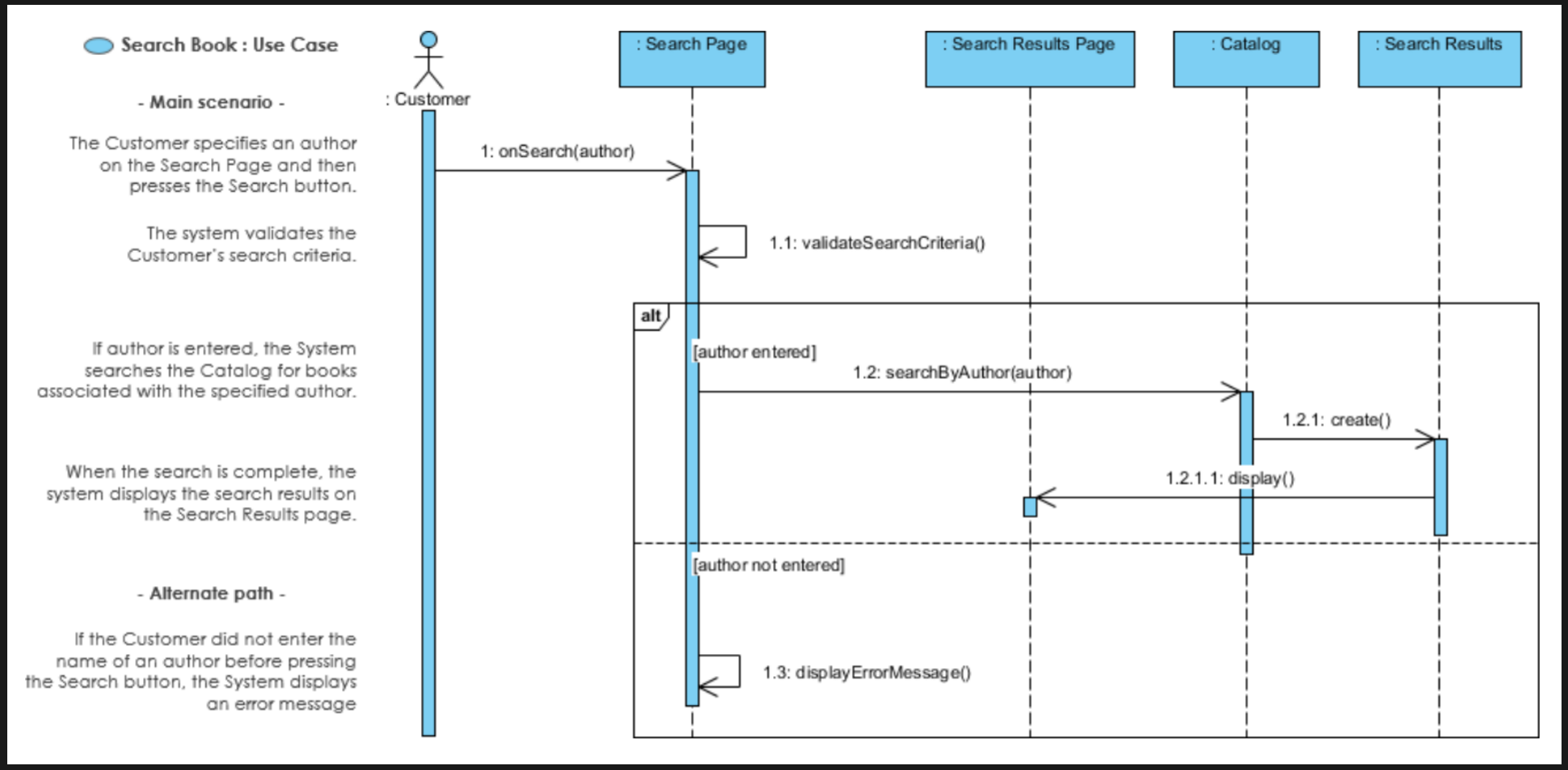


# MENSAJES RECURSIVOS

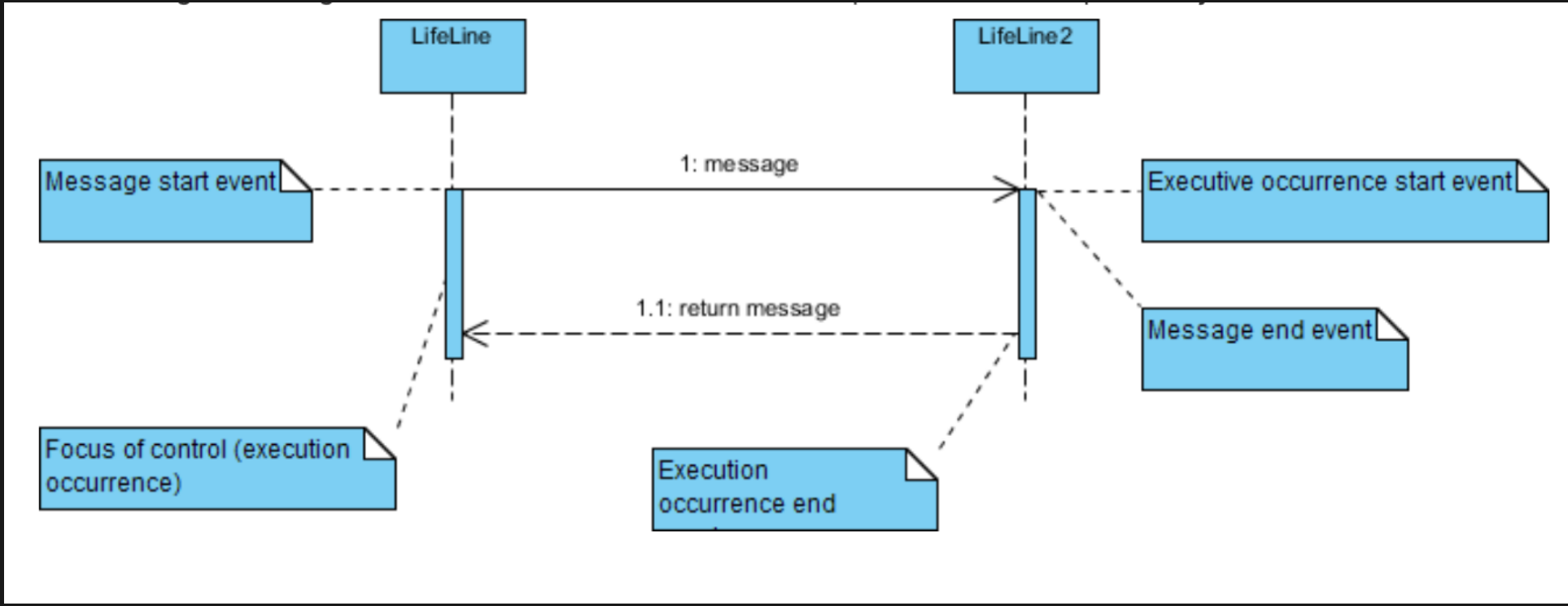


# NOTA

# DIAGRAMAS DE SECUENCIA



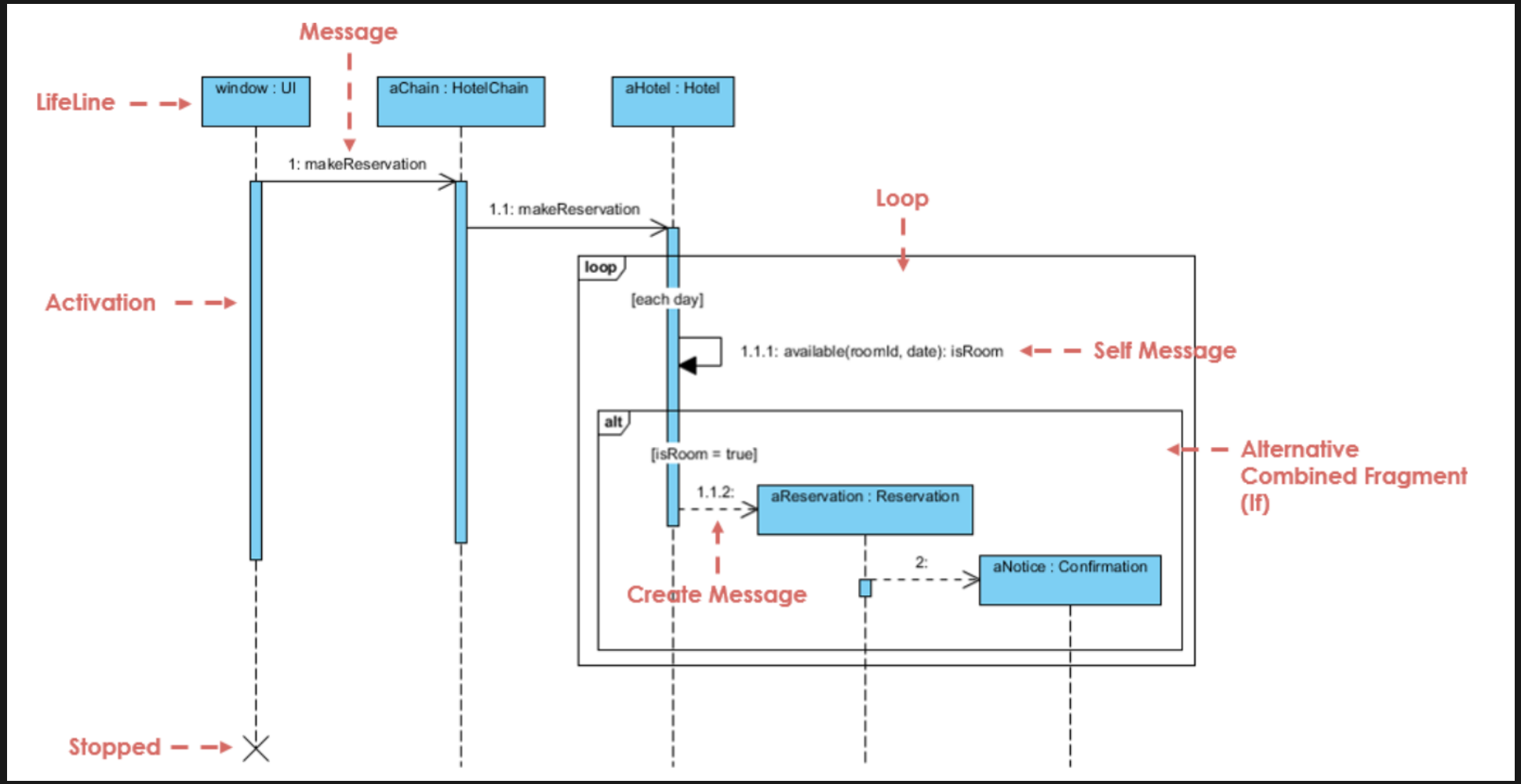
# EJEMPLOS


LifeLine
LifeLine2
Message start event
1: message
Executive occurrence start event
1.1: return message
Message end event
Focus of control (execution occurrence)
Execution occurrence end

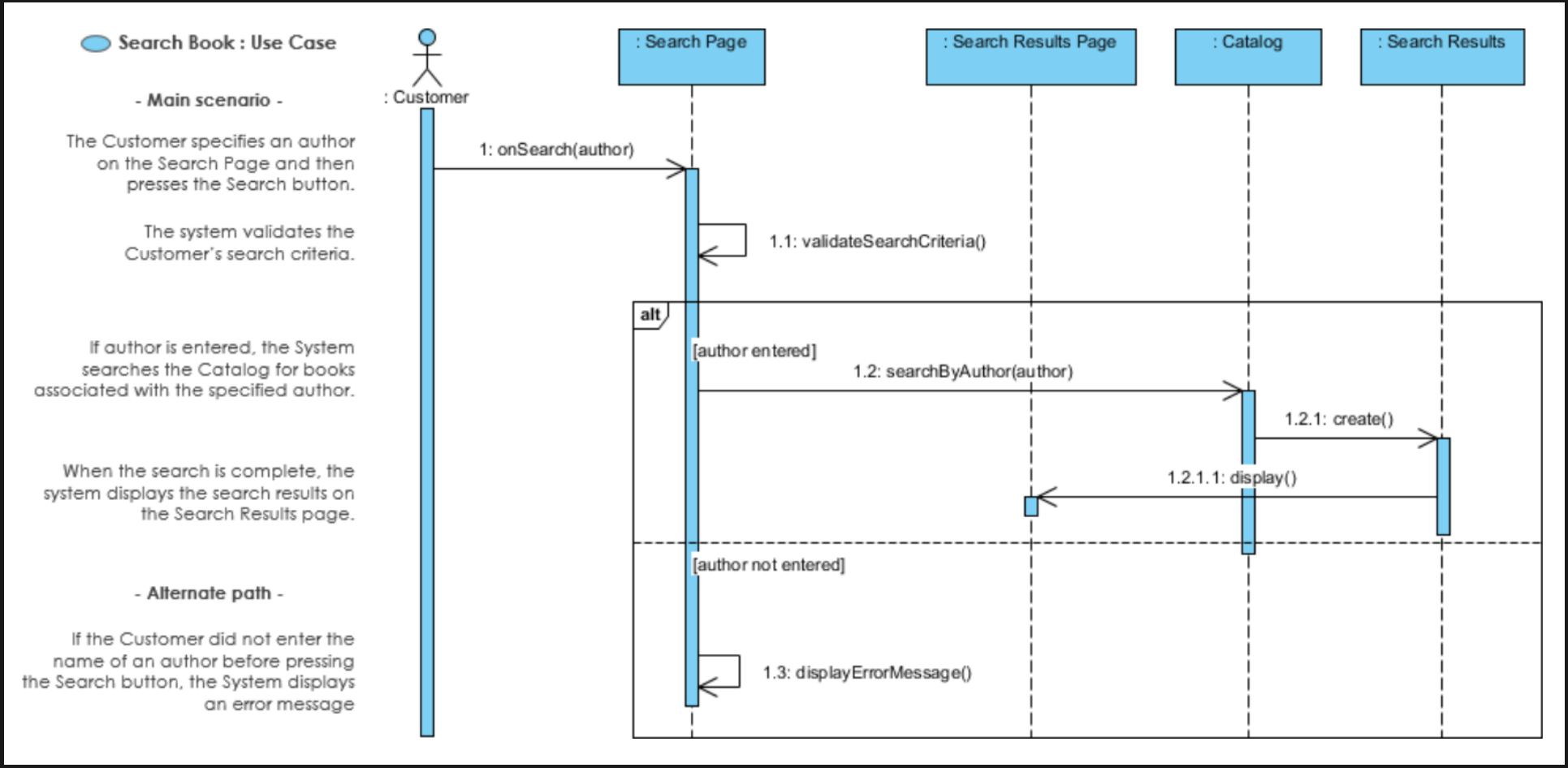
# FRAGMENTOS

| | |
|---|---|
| **alt** | Alternative multiple fragments: only the one whose condition is true will execute. |
| **opt** | Optional: the fragment executes only if the supplied condition is true. Equivalent to an alt only with one trace. |
| **par** | Parallel: each fragment is run in parallel. |
| **loop** | Loop: the fragment may execute multiple times, and the guard indicates the basis of iteration. |
| **region** | Critical region: the fragment can have only one thread executing it at once. |
| **neg** | Negative: the fragment shows an invalid interaction. |
| **ref** | Reference: refers to an interaction defined on another diagram. The frame is drawn to cover the lifelines involved in the interaction. You can define parameters and a return value. |
| **sd** | Sequence diagram: used to surround an entire sequence diagram. |

# FRAGMENTOS


Message
LifeLine
window : UI
aChain : HotelChain
aHotel : Hotel
1: makeReservation
1.1: makeReservation
Loop
Activation
loop
[each day]
1.1.1: available(roomId, date): isRoom
Self Message
alt
[isRoom = true]
Alternative
Combined Fragment
(If)
1.1.2:
aReservation : Reservation
2:
aNotice : Confirmation
Create Message
Stopped

# FRAGMENTOS


Search Book : Use Case
- Main scenario -
The Customer specifies an author on the Search Page and then presses the Search button.
The system validates the Customer's search criteria.
If author is entered, the System searches the Catalog for books associated with the specified author.
When the search is complete, the system displays the search results on the Search Results page.
- Alternate path -
If the Customer did not enter the name of an author before pressing the Search button, the System displays an error message
: Customer
: Search Page
: Search Results Page
: Catalog
: Search Results
1: onSearch(author)
1.1: validateSearchCriteria()
alt
[author entered]
1.2: searchByAuthor(author)
1.2.1: create()
1.2.1.1: display()
[author not entered]
1.3: displayErrorMessage()

# EJERCICIO

Diseñar un Diagrama de Secuencia para el juego del laberinto

```java
public class LabyrinthGame {

  public Labyrinth makeLabyrinth() {
    Labyrinth labyrinth = new Labyrinth();
    Room h1 = new Room(1);
    Room h2 = new Room(2);
    Door door = new Door(1,2)

    labyrinth.addRoom(h1);
    labyrinth.addRoom(h2);

    h1.setSide("North", new Wall() );
    h1.setSide("South", new Wall() );
    h1.setSide("East", new Wall() );
    h1.setSide("West", door);

    return labyrinth;
  }

}
```

```java
public class LabyrinthGame {

  public Labyrinth makeAutomaticLabyrinth() {
    Labyrinth labyrinth = new Labyrinth();
    Room room;

    for (int i = 0; x< 10; i++){
      room = new Room();

      if(i == 2 || i == 5 ){
        room.addWindow(new Window());
      }

      labyrinth.addRoom(room);

    }
    return labyrinth;
  }

}
```

```java
public class MainGame {

  public static void main( String args[] ) {
    LabyrinthGame labyrinthGame = new LabyrinthGame();

    labyrinthGame.makeLabyrinth();
    labyrinthGame.makeAutomaticLabyrinth();

  }

}
```