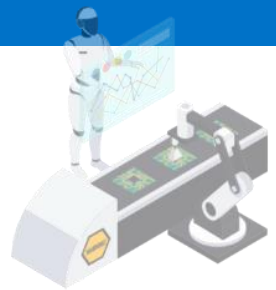
12강. 상호작용 다이어그램

컴퓨터과학과 김희천 교수

목차



Chapter. 1

상호작용 다이어그램 개요

1. 상호작용 다이어그램

상호작용 다이어그램

- 구성 요소 사이의 상호작용을 보여줌
- 유스케이스를 수행하기 위해 객체들이 어떻게 상호작용 하는지를 표현

✚ 대표적인 상호작용 다이어그램은 시퀀스 다이어그램과 통신 다이어그램

✕ 보통 1개 유스케이스에서 객체 간 상호작용을 다이어그램으로 표현하여 유스케이스를 상세히 함

✚ 시퀀스 다이어그램과 통신 다이어그램은 서로 1:1 변환이 가능함

Chapter. 2

시퀀스 다이어그램

1. 시퀀스 다이어그램 (1/2)

✚ 시스템의 구성 요소들이 어떻게 상호작용하는 하는가를 시각화

- 목표 시스템의 논리를 명세화
- 특정 객체들이 활성화되어 동작하고, 다른 객체들을 호출하는 순서를 보여줌
- 시간 흐름과 순서에 따른 시스템 동작을 표현

✕ 통신 다이어그램은 메시지 흐름보다는 상호작용에 참여하는 객체들 간의 관계를 파악할 때 효과적임

- 시퀀스 다이어그램은 울타리 형태이며 통신 다이어그램은 네트워크 형태

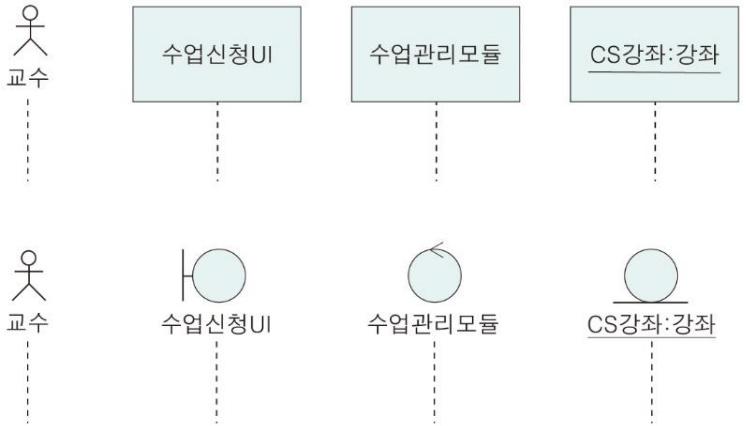
1. 시퀀스 다이어그램 (2/2)

시퀀스 다이어그램을 사용하는 이유

- 동적 행위를 표현하는 모델 가운데 가장 많이 활용됨
- 모든 유스케이스에 대해서 시퀀스 다이어그램을 작성할 필요는 없음
- 시스템의 동작 흐름을 파악하고 빠뜨린 객체나 메시지가 없는지 검증하기 위함

2. 참여 요소와 생명선 (1/2)

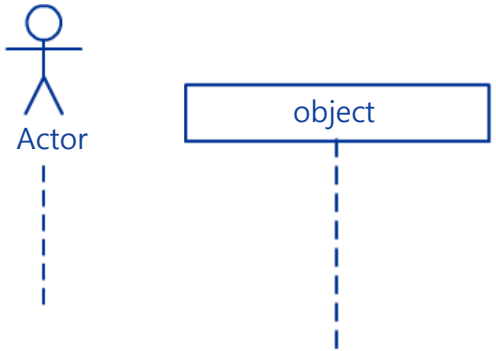
- **참여 요소는 메시지를 주고받는 주체**
  - 객체, 서브시스템, 외부 시스템, 하드웨어 등
  - 사각형으로 표시되며, 다이어그램의 상단에 적당한 간격을 두고 수평으로 배치됨
- **참여 요소는 액터, 경계 객체, 제어 객체, 엔터티 객체 순으로 나타남**



2. 참여 요소와 생명선 (2/2)

✚ 생명선

- 참여 요소에서 아래쪽으로 향하는 점선
- 상호작용의 순서를 보여주기 위한 시간 축
- 위에서 아래 방향으로 시간이 흘러감
- 길이가 시간 간격을 의미하는 것은 아님



✕ 참여 요소의 이름 표기법과 예

- **이름[선택자] : 클래스이름** ref 상호작용다이어그램
- :Student - Student 클래스의 익명 객체
- admin:Administrator - Administrator 클래스의 admin 객체
- :RegistrationSystem ref reg_detail - RegistrationSystem은 서브시스템이며 이것의 자세한 작업은 reg_detail 이라는 이름의 상호작용 다이어그램에 나옴

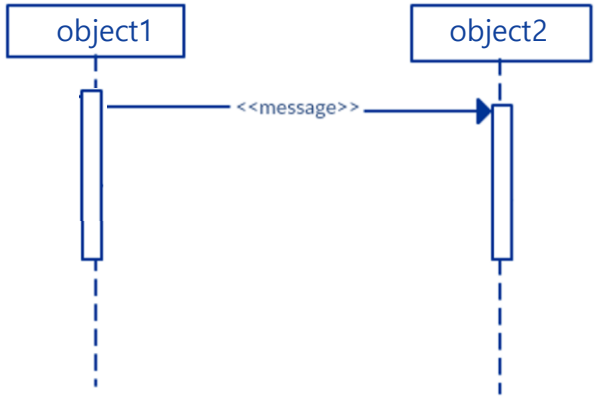
3. 메시지 (1/2)

메시지 전송 또는 시그널이 전송되는 것을 이벤트 발생이라 함

- 화살표로 표시
- 이벤트 발생은 상호작용이 있음을 의미
- 메시지 전송은 호출자가 수신자에게 메소드 실행을 요청하는 이벤트를 발생시키는 것
- 메시지의 실행 순서는 위에서 아래로 진행됨

활성화 막대

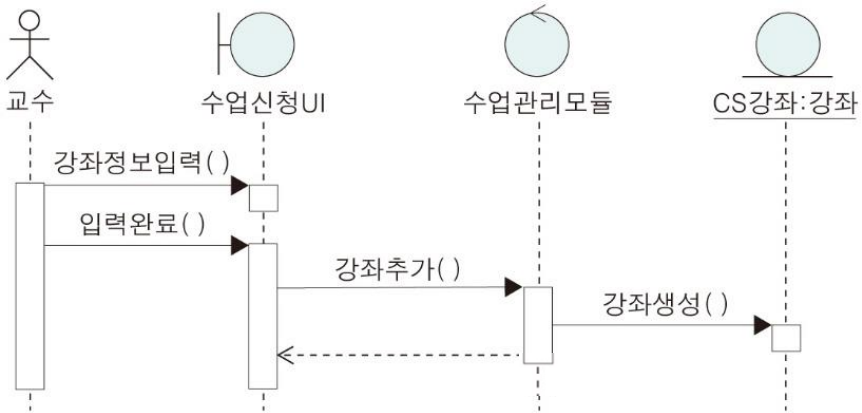
- 수신 객체가 활성화된 상태
- 수신자가 주어진 이벤트에 해당하는 특정 행위를 실행하는 상황을 표현
- 생명선을 따라 얇은 막대 모양으로 표시



3. 메시지 (2/2)

메시지 호출자와 메시지 수신자

- 메시지를 보내고 받는 참여 요소
- 호출 메시지가 동기화 메시지이면 검은 삼각형의 화살표 머리로 표시
- 리턴 메시지는 점선의 화살표로 표시
- 의미가 명확하면 리턴 메시지를 표시하지 않아도 됨



4. 메시지 형식

- **속성 = 메시지이름(파라미터) : 리턴형식**
  - 여러 파라미터가 필요하면 콤마(,) 로 구분하며 하나의 파라미터 형식은 '이름:타입'
  - doSomething(number1:Number, number2:Number) –
    Number 유형의 파라미터 2개를 가지고 doSomething( )을 호출
  - temp=method( ):MyClass –
    method( ) 호출의 결과가 MyClass 유형이며 이것은 변수 temp에 저장됨
- **메시지 형식이 자세히 기술되면 시퀀스 다이어그램으로부터 코드 일부를 생성할 수 있음**
  - 클래스 다이어그램으로부터는 정적인 클래스 골격을 생성할 수 있음
  - 시퀀스 다이어그램과 클래스 다이어그램은 설계와 구현 사이를 연결

5. 동기화/비동기화 메시지 (1/5)

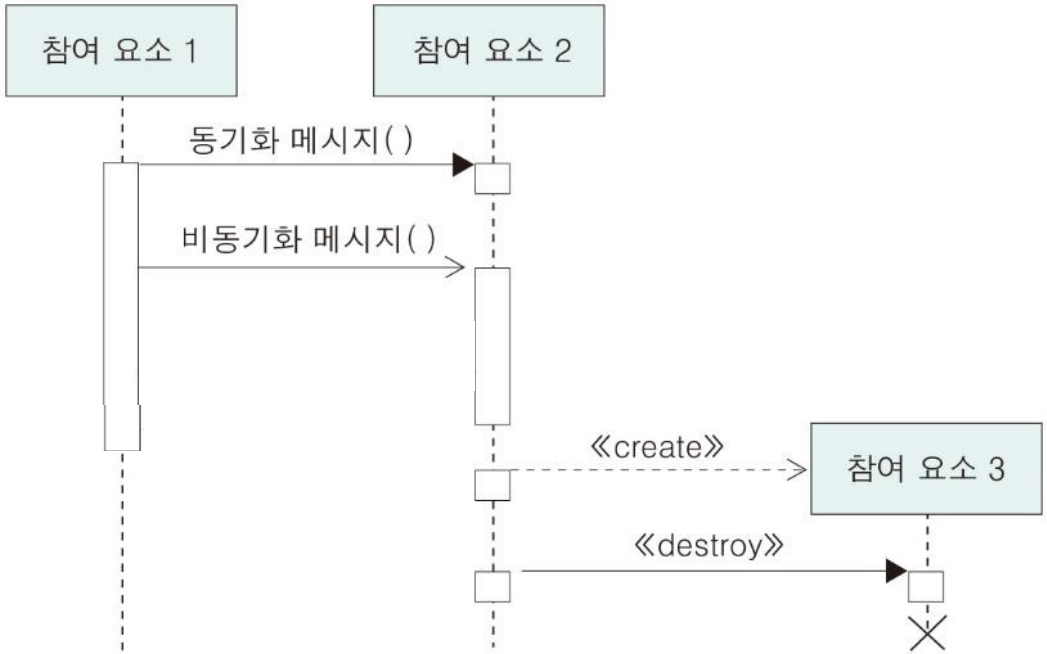
✚ 동기화 메시지 ⟶

- 송신자가 메시지를 보낸 후, 수신자가 메시지를 처리하고 리턴 메시지를 보낼 때 까지 기다려야 함
- 일반적 메소드에 해당

✕ 비동기화 메시지 ⟶

- 메시지를 보낸 후, 리턴을 기다리지 않고 작업을 계속함
- 실선 화살표에서 화살표 머리 모양이 다름

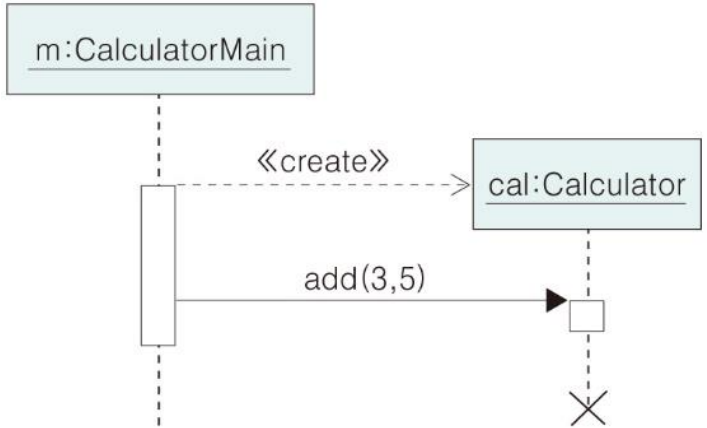
5. 동기화/비동기화 메시지 (2/5)



5. 동기화/비동기화 메시지 (3/5)

✚ 동기화 메시지 예

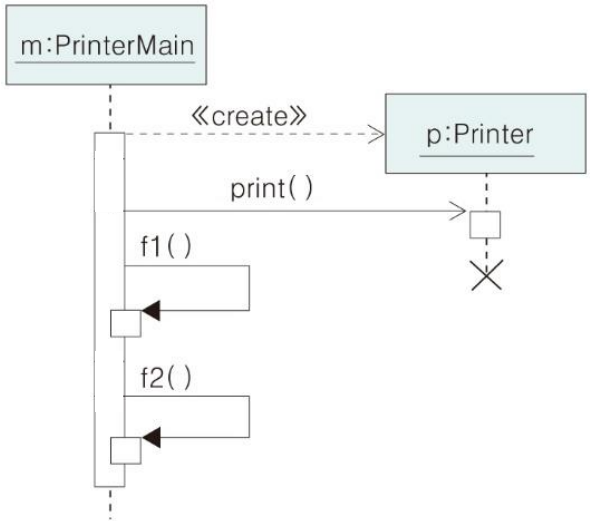
```
public class Calculator
{
  public void add(int x, int y)
  {
    System.out.println(x+y);
  }
}

public class CalculatorMain
{
  public static void main(String[ ] args)
  {
    Calculator cal = new Calculator( );
    cal.add(3,5);
    //앞의 메소드 실행이 끝나야 이 부분이 수행됨
  }
}
```



5. 동기화/비동기화 메시지 (4/5)

비동기화 메시지의 예

- 2개의 작업이 동시 수행됨
- Java나 C#에서 스레드를 생성하여 실행시키는 경우
- 예: 문서 작업 중 프린트 작업이 필요한 경우



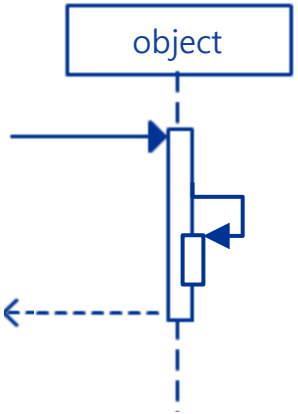
5. 동기화/비동기화 메시지 (5/5)

```
public class Printer implements Runnable {
    public void print() {
      Thread t = new Thread(this);
      t.start();
    }
    public void run() {
      // 프린트 작업
    }
}
public class PrinterMain {
    public static void main(String[] args) {
      Printer p = new Printer();
      p.print();
      f1(); // 기타 작업
      f2(); // 기타 작업
    }
… …
}
```

6. 기타 메시지 (1/3)

✚ 자기 메시지(self message)

- 객체가 자신에게 메시지를 보내는 경우
- 활성화 상태가 중첩되어 나타남



✕ 리턴 메시지

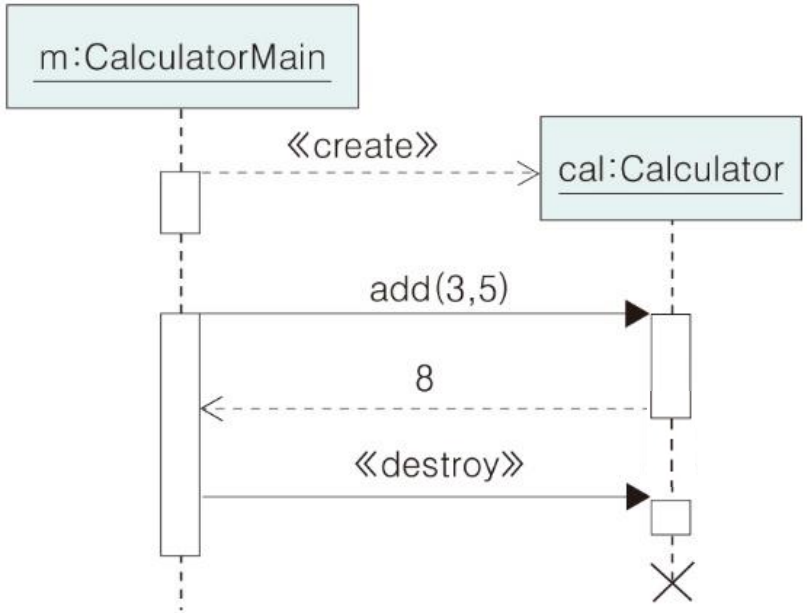
- 메시지 수신자가 실행을 종료하고, 제어를 송신자에게 돌려주는 상황을 표현
- 리턴 값을 메시지 송신자에게 전달할 수 있음
- 활성화 상태가 종료되는 시점에는 암시적으로 리턴 메시지가 있으므로 중요한 리턴 메시지가 아니라면 생략함

6. 기타 메시지 (2/3)

생성과 삭제 메시지

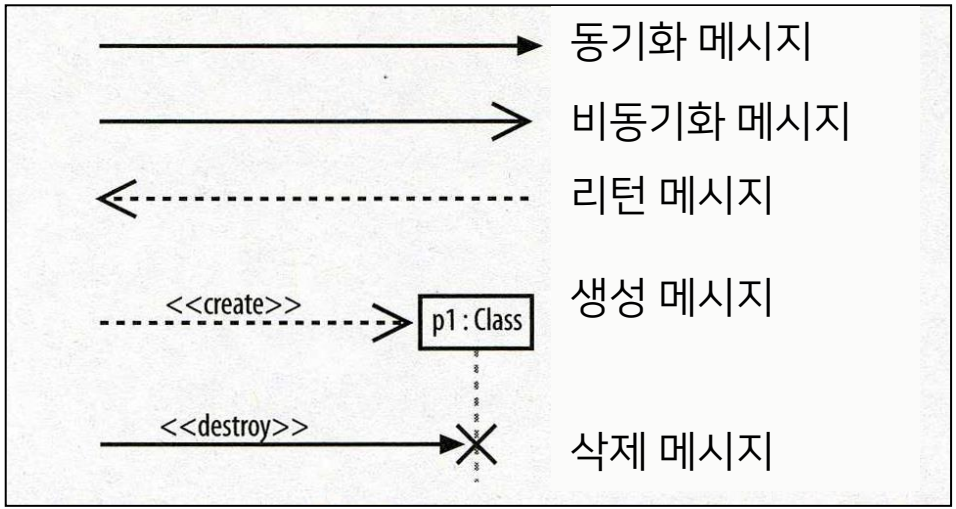
- 참여 요소를 생성하는 경우나 삭제하는 경우
- 생성을 위한 메시지는 《create》를 사용하고 점선 화살표 끝에 생성되는 참여 요소를 배치함
- 삭제를 위한 메시지는 《destroy》를 사용하고 동기화 메시지와 같은 모양으로 표현하고 생명선 끝에 X 표시를 함
- Java나 C#에서 생성은 new 연산자를 사용해야 하나 삭제는 자동으로 되므로 구현만을 생각한다면 삭제 메시지를 표시하지 않음

6. 기타 메시지 (3/3)



7. 메시지 종류

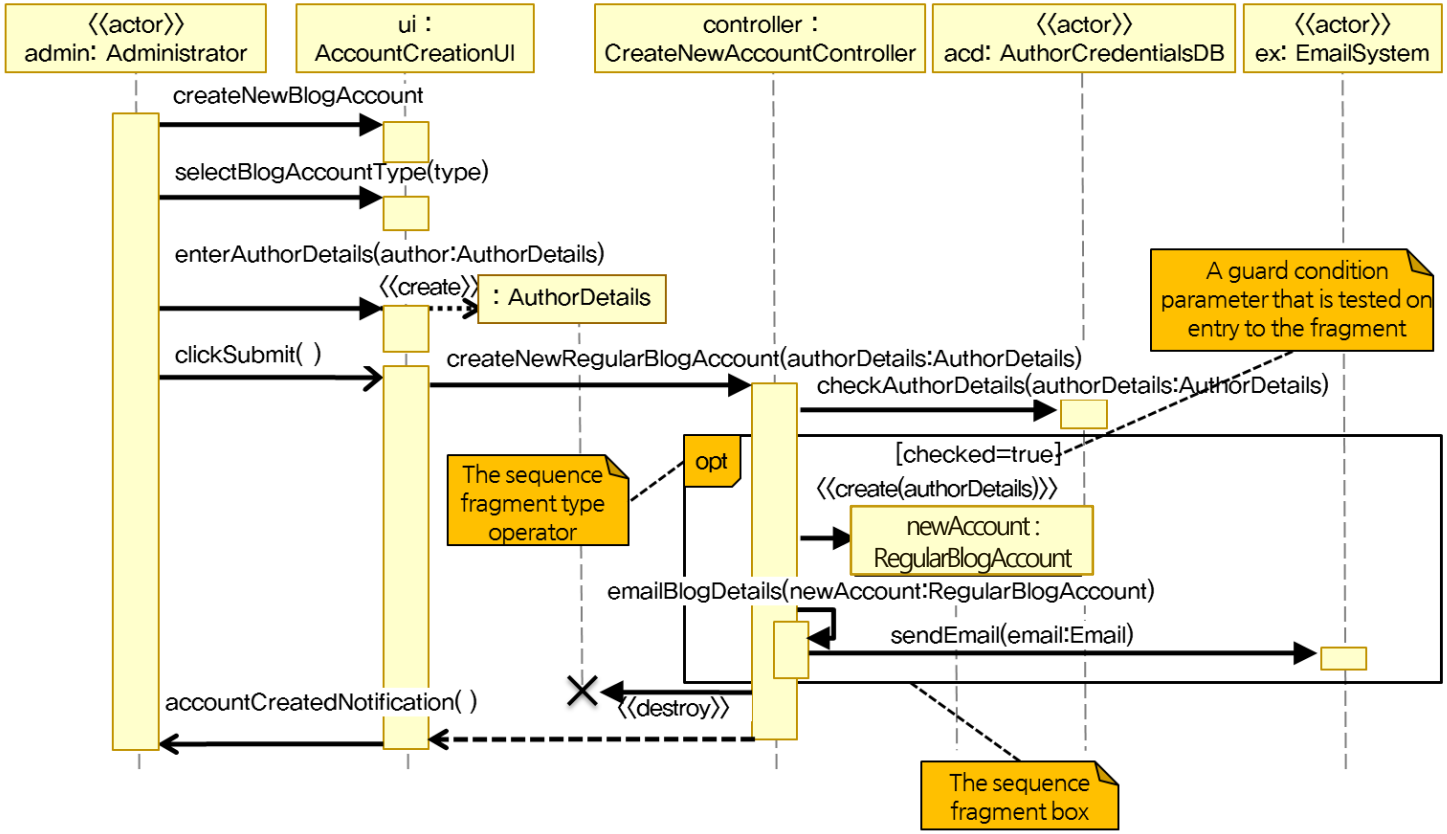
✚ 메시지 종류와 화살표



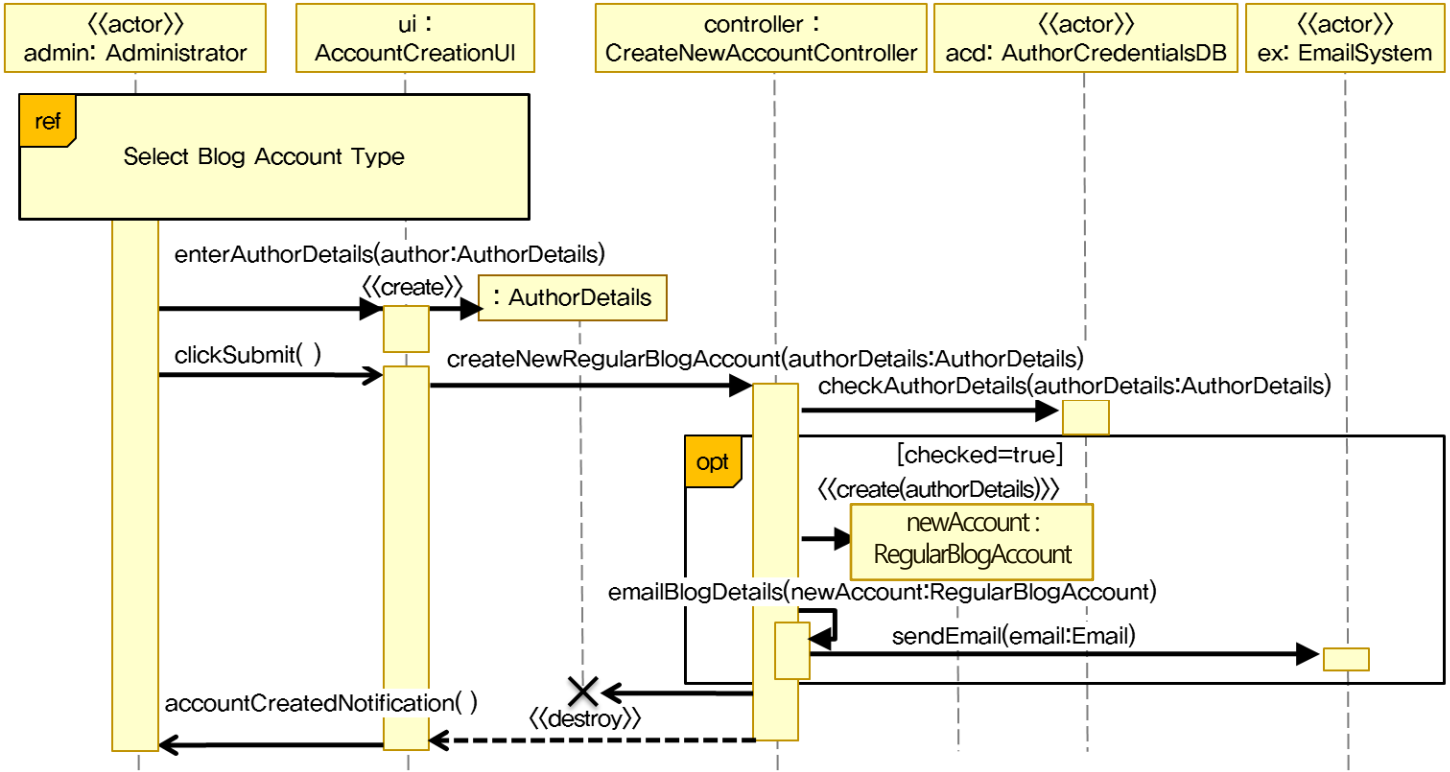
8. 시퀀스 프래그먼트 - 교재에 없음 (1/4)

- 시퀀스 다이어그램 안에서 박스로 표시된 영역
  - 다이어그램의 복잡성을 줄이고 구조화하기 위함
  - 선택적 실행이나 반복 실행과 같은 복잡한 상호작용을 표현하기 위함
- 박스 좌상단에 프레그먼트의 유형을 표현하는 operator를 표시함
  - operator가 opt 이면 가드 조건을 만족될 때만 수행되는 상호작용임을 표시하는 것
  - operator가 alt 이면 점선으로 나뉜 여러 부분 영역 중에 조건에 따라 하나가 선택적으로 수행되는 것
  - operator가 loop 이면 가드 조건이 만족되는 동안 반복 수행된다는 것
  - operator가 ref 이면 외부에서 정의된 시퀀스 다이어그램을 포함하는 것
  - operator가 par 이면 분리된 몇 개의 상호작용이 동시에 수행된다는 것

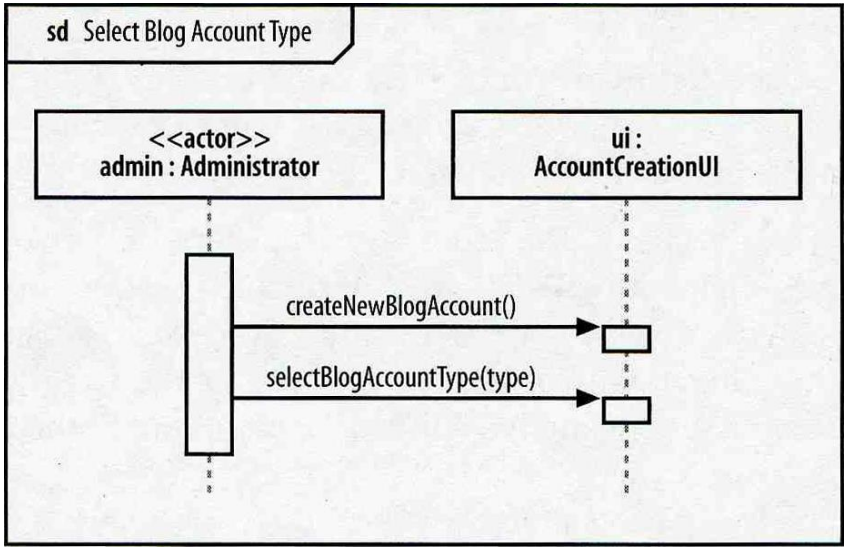
8. 시퀀스 프래그먼트 (2/4)



8. 시퀀스 프래그먼트 (3/4)



8. 시퀀스 프래그먼트 (4/4)



Chapter. 3

통신 다이어그램

1. 통신 다이어그램

✚ 참여 요소들 간의 메시지 송수신 관계를 파악하기에 좋은 상호 작용 다이어그램

- 네트워크 형태이나 시퀀스 다이어그램과 같은 정보를 표현함
- UML 1.x 까지는 협력(collaboration) 다이어그램이라고 불리었음
- UML 도구를 사용하면 시퀀스 다이어그램과 상호 변환이 가능함

✕ 시퀀스 다이어그램과 다른 점

- 시퀀스 다이어그램은 메시지들의 흐름과 순서에 초점을 맞춤
- 통신 다이어그램은 참여 요소들 간의 **상호작용 관계**에 초점을 둠
- 통신 다이어그램이 전체 상호작용의 개요를 직관적으로 파악하기에 용이함

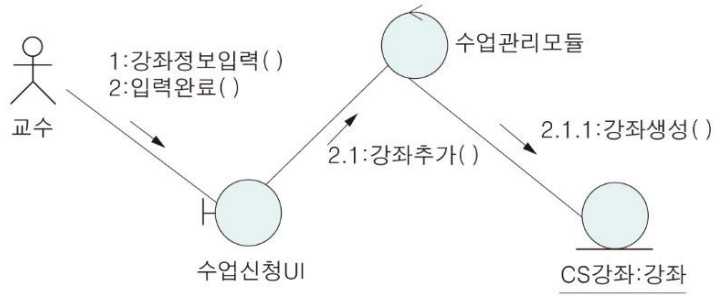
2. 통신 링크와 메시지

✚ 통신 링크

- 메시지를 주고받는 두 참여 요소를 연결하여 둘의 관계를 보여주는 실선
- 통신 링크를 따라서 메시지가 전달됨

✕ 메시지의 표현

- 통신 링크를 따라 호출자에서 수신자로 향하는 화살표와 메시지를 표시
- 메시지 앞에 번호를 붙여 순서를 표현



3. 메시지의 전송 순서 (1/3)

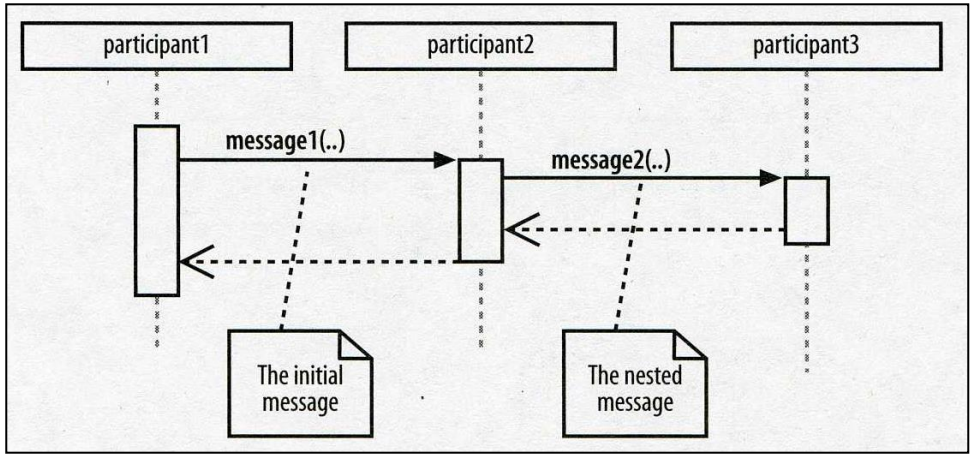
순차적 메시지

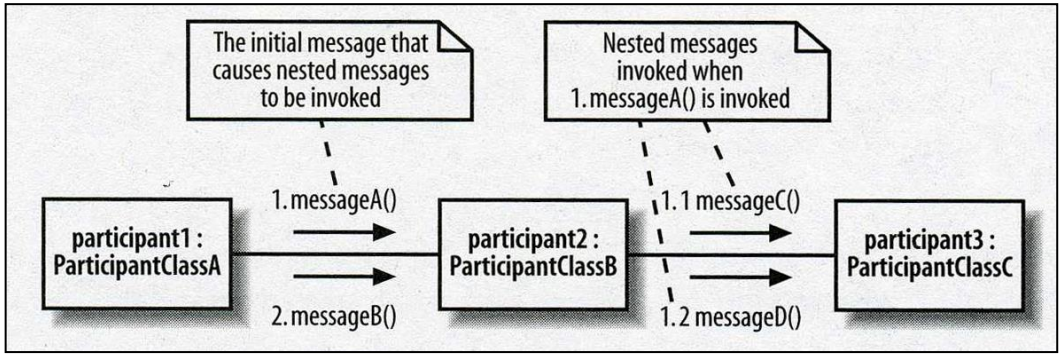
- 1부터 시작해서 하나씩 증가시키면서 메시지의 순차적 수행을 표시
- 1번 메시지 다음에 2번 메시지가 수행됨

중첩 메시지

- 메시지를 받아 수행 중에 다른 메시지를 호출하는 경우
- 통신 다이어그램에서는 중첩 메시지를 자연스럽게 표현하기 힘들며 1.1과 같이 숫자를 단계적으로 사용하여 표현함
- 2번 메시지가 시작되고 그 안에서 2.1번, 2.2번 메시지가 시작됨

3. 메시지의 전송 순서 (2/3)

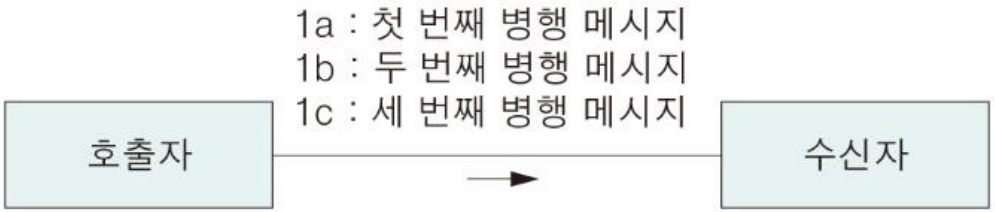




3. 메시지의 전송 순서 (3/3)

✚ 병행 메시지

- ✕동시에 메시지를 보내는 경우로 메시지가 동시 수행됨
- ✕1a, 1b, 1c 와 같이 표현함



4. 조건 메시지와 메시지의 반복 (1/2)

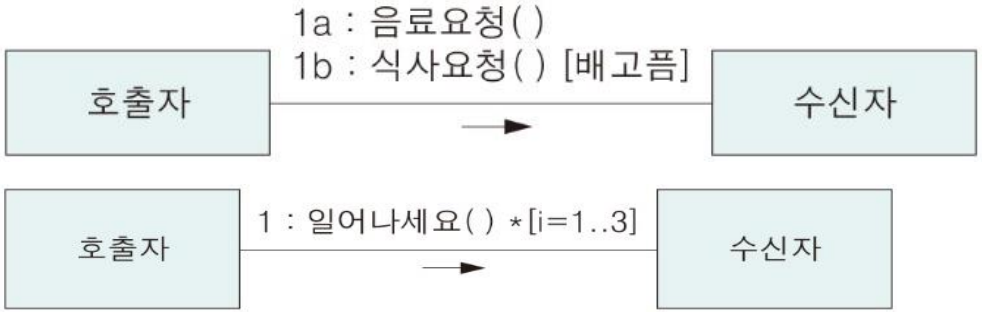
조건문이 있는 메시지

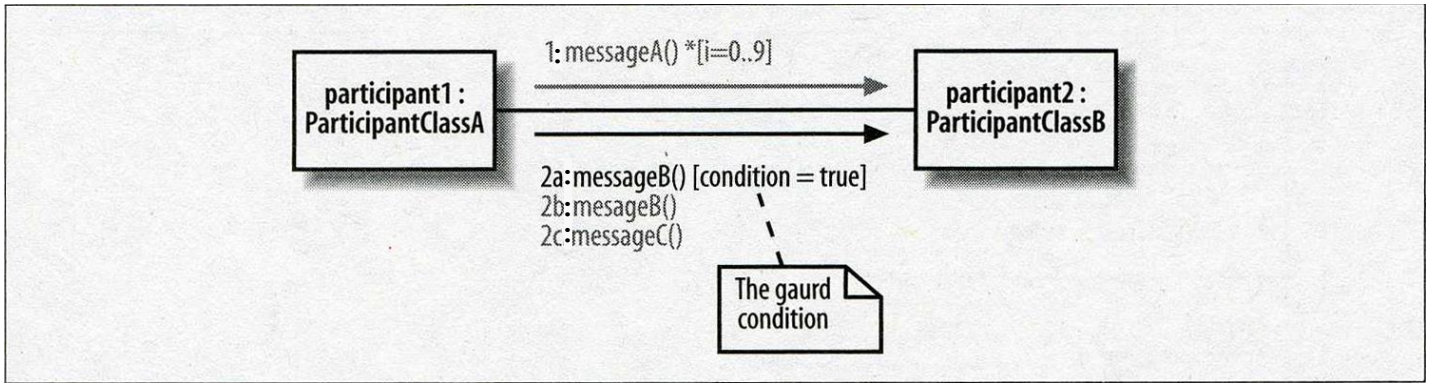
- 조건문이 참인 경우에만 메시지를 전달
- 메시지 뒤에 [ ]를 사용하여 조건을 표현함
- 2.3b: draw( ) [x>y]

메시지의 반복

- *를 사용하여 반복적으로 전달되는 메시지를 표현
- 4.2c : search(t[i]) *[i=1..12]

4. 조건 메시지와 메시지의 반복 (2/2)





다음강의

13강. 클래스 다이어그램과 객체 다이어그램