

”

CHAPTER

16

함수형 프로그래밍,
람다식, 스트림

+ 학습목표

- 함수형 프로그래밍을 이해하고 사용한다.
- 스트림 API를 사용하여 컬렉션에서 특정 조건을 만족하는 데이터를 추려낸다.
- 원하는 동작을 포장하여 메소드에 전달한다.
- 람다식으로 이름 없는 메소드를 작성하고 전달한다.

+ 학습목차

- 16.1 함수형 프로그래밍 소개
- 16.2 람다식
- 16.3 동작 매개변수화

- 16.4 함수형 인터페이스
- 16.5 메소드 참조
- 16.6 스트림

핵심 키워드

함수형 프로그래밍, 람다식, 메소드 참조, 스트림

함수형 프로그래밍, 람다식, 스트림

16.1 함수형 프로그래밍 소개

자바는 꾸준히 업그레이드되고 있다. “이번엔 또 뭐가 바뀌었지?” 하며 버전이 나올 때마다 놀라게 된다. 새로운 기능들이 계속 추가되지만, 최근 가장 혁신적인 변화는 바로 함수형 프로그래밍(functional programming) 지원이다. 이 혁신적인 변화는 Java 8에서 시작되었다. 이전 버전의 변화가 사소한 수준이었다면, Java 8은 “이제부터 자바도 함수형 언어처럼 쓸 수 있다!”라고 선언한 셈이기 때문이다. 람다식(->), 스트림 API, 메소드 참조... 등 처음에는 “이게 자바 맞아?” 싶지만, 익숙해지면 “이걸 왜 이제야 넣었어!”라고 하고 싶을 정도로 편리하다.

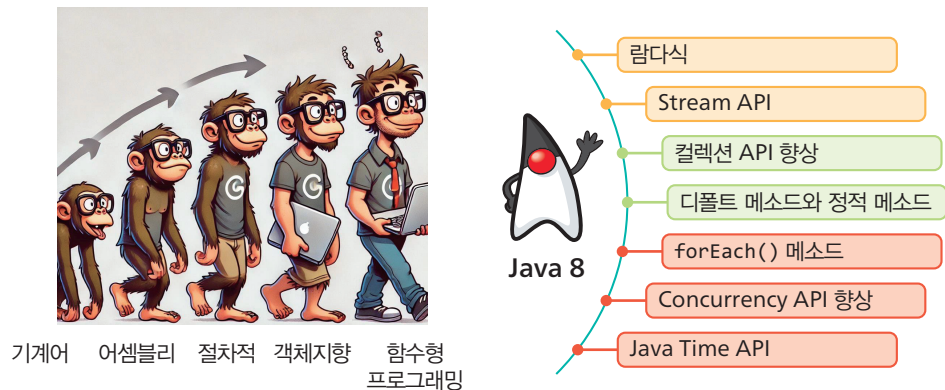


그림 16-1 꾸준히 업그레이드되는 자바

함수형 프로그래밍을 잘 사용하면 아주 쉽게 프로그램을 작성할 수 있다. 예를 들어서 문자열들을 저장하고 있는 리스트가 있고, 이것을 문자열의 길이에 따라서 정렬하려고 한다. 예전 자바 버전(Java 7 이전 버전)에서는 왼쪽과 같이 익명 클래스를 사용하여 다소 장황한 코드를 작성해야 했다. 반면에 Java 8에서는 람다식과 메소드 참조를 활용하여 동일한 작업을 훨씬 간결하게 표현할 수 있다.

Java 7 스타일

```
Collections.sort (mylist, new Comparator <String> () {
    public int compare (String s1, String s2) {
        return (s1.length()-s2.length());
    }
});
```

Java 8 이후 스타일

```
mylist.sort(comparing(String :: length));
```

그림 16-2 함수형 프로그래밍 변화

익명 클래스 방식은 코드가 장황하고 읽기 어려우며 재사용성이 떨어진다. 반면에 오른쪽의 메소드 참조 방식은 간결하고 가독성이 높아지며 코드의 의도를 직관적으로 표현할 수 있다. 함수형 프로그래밍의 장점은 짧고 간결한 코드 작성이 가능하고 메소드 참조를 사용하면 코드의 의도가 더 명확해진다는 점이다. 또한 스트림 API를 통해 멀티코어 프로세서를 활용한 병렬 처리가 간편해진다는 장점도 가지고 있다. 함수형 프로그래밍은 장점이 많아서 앞으로 점점 많이 사용될 것이다.

참고사항: `Comparator.comparing()`

`java.util.Comparator<T>`는 객체를 정렬하기 위한 인터페이스이다. `comparing()` 메소드는 객체의 특정 속성을 기준으로 정렬하는 `Comparator` 객체를 생성하는 정적 메소드이다. 주로 객체의 특정 필드 값을 기준으로 정렬할 때 사용되며, 객체의 접근자 메소드와 함께 사용된다. 예를 들어서 `Comparator.comparing(Person::getName)`와 같이 사용한다.

프로그래밍 패러다임 분류

프로그래밍 패러다임은 몇 가지로 분류할 수 있다. 프로그래밍 패러다임을 요리에 비유하여 설명해 보자. 명령형 프로그래밍(imperative programming)은 요리사가 “이제 채소를 다지고, 팬을 예열하고, 고기를 굽고, 소금을 뿌리세요!”라며 하나하나 세세하게 지시하는 방식이다. 반면 선언형 프로그래밍(declarative programming)은 “맛있는 스테이크 한 접시 주세요!”라고 말하면 요리사가 알아서 만들어 주는 방식이다. 함수형 프로그래밍(functional programming)은 선언형 프로그래밍의 일종이다. 함수형 프로그래밍은 불필요한 조리 과정은 생략하고, 재료(데이터) 중심으로 깔끔하게 처리하는 스타일이다.

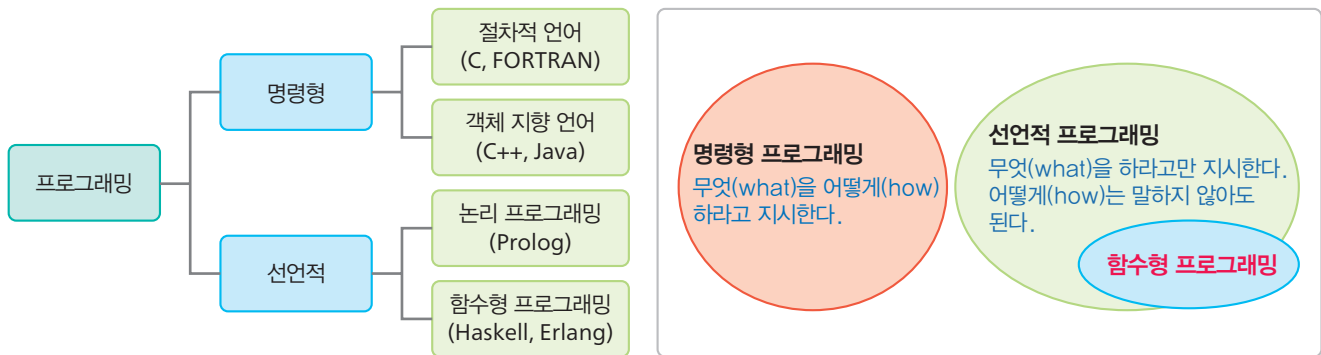


그림 16-3 프로그래밍 패러다임 분류

이 2가지의 프로그래밍 방식이 어떻게 다른지 구체적인 코드로 살펴보자.

명령형 프로그래밍 방법

정수들이 ArrayList에 저장되어 있다고 가정하자. ArrayList에서 짝수만 추려내고 싶을 경우 명령형 프로그래밍에서는 어떻게 해야 할까? 아마 ArrayList에 저장된 정수들을 하나씩 꺼내서 반복 루프로 처리해야 할 것이다.

Imperative.java

```

1 public class Imperative {
2     public static void main(String args[]) {
3         List<Integer> list = List.of(12, 3, 16, 2, 1, 9, 7, 20);
4         List<Integer> even = new ArrayList<>();
5
6         for (Integer e : list) { // 짝수를 찾는다.
7             if (e%2 == 0 ) {
8                 even.add(e);
9             }
10        }
11        for (Integer e : even) { // 찾은 짝수를 출력한다.
12            System.out.println(e);
13        }
14    }
15 }
  
```

```

12
16
2
20
  
```


정수는 `list`에 저장되어 있으며, `for-each` 문으로 `list`에서 하나씩 정수를 꺼내고 `if` 문을 이용하여 2로 나누어서 나머지가 0인 정수만 추린다. 이 정수들을 리스트 `even`에 저장한다. 최종적으로 리스트 `even`을 순회하면서 정수를 화면에 출력한다.

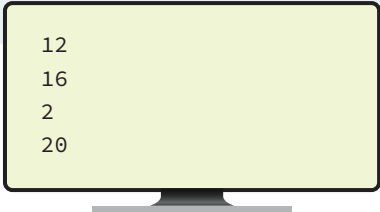
명령형 프로그래밍은 작업을 어떻게(how) 수행하는지를 중시한다. 즉, 먼저 이것을 수행한 다음에 이것을 수행하라고 말해주는 프로그래밍이다. 예를 들어 앞의 코드에서도 정수 리스트에서 정수를 하나씩 꺼내서 짝수인지를 검사한 후에, 짝수이면 다른 리스트에 추가하였다. 이 과정을 리스트의 요소가 끝날 때까지 반복한다. 이러한 스타일의 프로그래밍은 할당문(대입문), 조건문, 반복문을 사용하기 때문에 고전적인 프로그래밍과 매우 잘 어울린다. 명령형 프로그래밍에서는 명령문 하나가 실행되면, 프로그램의 상태가 계속 바뀌면서 작업이 진행된다. 하지만 코드가 복잡해질수록 디버깅과 유지 보수가 어려워진다는 단점이 있다.

함수형 프로그래밍 방법

우리는 아직 함수형 프로그래밍을 본격적으로 학습하지 않았지만, 함수형 프로그래밍을 사용한다면 동일한 작업을 다음과 같은 코드로 작성할 수 있다.

Test.java

```
1 public class Test {
2     public static void main(String args[]) {
3         List<Integer> list = List.of(12, 3, 16, 2, 1, 9, 7, 20);
4         list.stream()
5             .filter(e -> e % 2 == 0)
6             .forEach(System.out::println);
7     }
8 }
```



```
12
16
2
20
```

여기서 `stream()` 메소드는 리스트 안의 원소들을 하나씩 추출하여 앞으로 보내준다. `filter()` 메소드는 들어오는 정수 중에서 짝수만을 추려낸다. `filter()`와 같이 메소드 앞에 점(.)이 찍힌 것은 메소드 체이닝이라고 하는 기법이다. 앞의 메소드가 반환하는 객체의 메소드를 연속적으로 호출하는 기법이다. `forEach()` 메소드는 들어오는 정수에 대하여 전달되는 함수를 적용한다. `System.out::println`은 메소드 참조로, 메소드 `println()`을 다른 메소드 `forEach()`로 보내는 방법이다. 약 8줄의 코드가 3줄로 줄어든다.

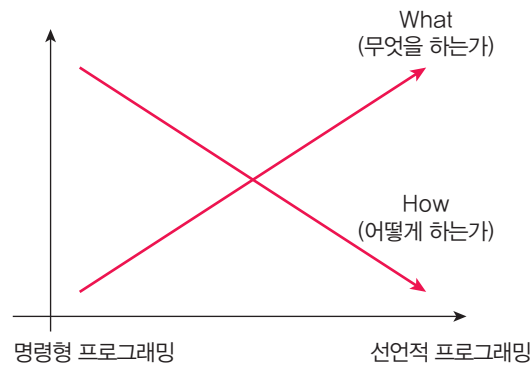


그림 16-4 명령형 프로그래밍은 How에, 선언형 프로그래밍은 What에 집중한다.

선언형 프로그래밍은 해야 할 일(what)에 집중한다. 함수형 프로그래밍에서는 함수들이 계속 적용되면서 작업이 진행된다. 함수형 프로그램은 명령문이 아닌 수식이나 함수 호출로 이루어진다. 함수형 프로그래밍은 1930년대의 람다 수학에 근간을 두고 있다. 이 방법의 가장 큰 장점은 함수 호출이 문제 설명처럼 읽히기 때문에, 코드가 수행하는 작업을 보다 명확하게 알 수 있다는 점이다. 이런 스타일을 선언형 프로그래밍이라고 한다. 선언형 프로그래밍에서는 개발자가 원하는 것을 기술하고 시스템이 그 목표를 달성하는 방법을 결정한다.

구운 빵에 양상추, 토마토, 햄을 넣고
랜치소스를 뿌려서 만들어 주세요.



명령형 프로그래밍

샌드위치
하나 주세요.



선언적 프로그래밍

특징	명령형 프로그래밍	선언형 프로그래밍
초점	'어떻게(how)' 수행할지 명시	'무엇(what)'을 할지에 초점
상태 관리	명령문 실행에 따라 상태를 계속 변경	상태 변화 최소화 및 불변성 강조
코드 스타일	반복문과 조건문을 사용한 복잡한 코드	스트림과 고차 함수를 사용한 간결한 코드
유지 보수성	상태 변화를 추적해야 하므로 유지 보수가 어려움	코드가 간결하며 의도를 명확히 표현하므로 유지 보수 용이

표 16-1 명령형 프로그래밍과 선언형 프로그래밍 비교

왜 함수형 프로그래밍인가?

함수형 프로그래밍은 선언적 프로그래밍 개념을 실제로 구현한 형태이다. 개발자는 함수를

사용하여 원하는 작업을 기술하고, 시스템에서 이것을 어떻게 구현할지를 결정한다(즉, 개발자가 결정하는 것이 아니다). 이러한 아이디어는 소프트웨어를 보다 쉽게 구현하고 유지하는 데 도움을 줄 수 있다. 자바에서는 함수형 프로그래밍의 일부를, 람다식과 스트림 API를 이용하여 지원한다.

또 다른 중요한 장점은 병렬 처리가 쉽다는 것이다. 최근 CPU는 모두 멀티 코어를 장착하고 있고 함수형 프로그래밍에서는 순수 함수만을 사용하기 때문에 코어를 여러 개 동시에 사용하여도 서로 간에 복잡한 문제가 발생하지 않는다.

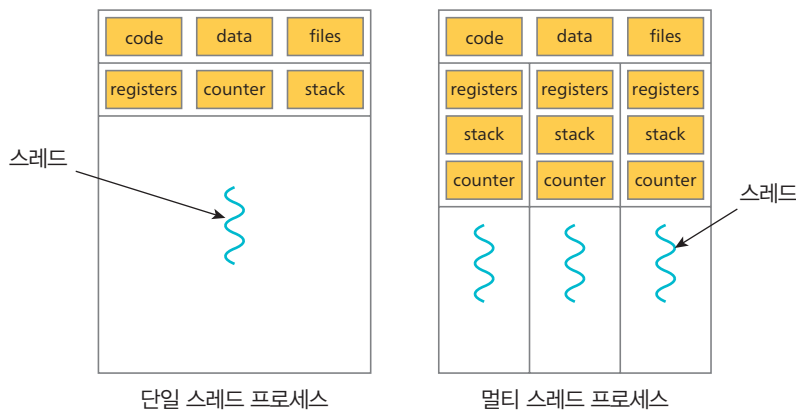


그림 16-5 함수형 프로그램은 병렬 처리가 쉽다.

함수의 부작용

함수는 함수형 프로그래밍의 핵심 개념이다. 물론 함수는 우리가 잘 알고 있지만, 다른 관점에서 다시 살펴보자. 함수는 **부작용(side effect)**이 있을 수 있다. 부작용이 있다는 말은 함수가 실행하면서 내부 상태나 외부의 변수를 변경한다는 의미이다. 명령형 프로그래밍에서는 함수가 내부 상태나 외부 상태를 읽거나 수정하는 일이 빈번하다. 이로 인해 동일한 입력값에 대해 다른 결과값을 반환할 가능성이 생긴다. 대표적인 예가 `Random` 클래스의 `nextInt()` 메소드이다. `nextInt()` 메소드는 불릴 때마다 난수 발생기의 상태가 변경되고 따라서 반환값이 달라진다.

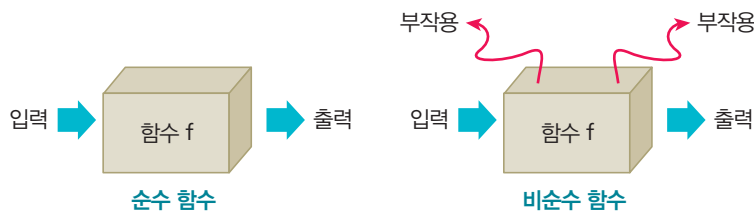


그림 16-6 순수 함수와 비순수 함수

부작용이 있는 함수의 간단한 예는 다음과 같다.

```
int globalVariable = 0;

public int incrementAndGet() {
    globalVariable++;                // 외부 상태 변경(부작용)
    return globalVariable;
}

// 호출마다 다른 결과 반환
System.out.println(incrementAndGet()); // 출력: 1
System.out.println(incrementAndGet()); // 출력: 2
```

이 함수는 외부 변수 `globalVariable`을 수정하기 때문에 호출 순서에 따라 다른 결과를 반환한다.

함수형 프로그래밍에서의 순수 함수

함수형 프로그래밍에서 사용하는 함수는 **순수 함수(pure function)**라고 한다. 순수 함수란 부작용(side effect)이 없는 함수를 말하며, 함수가 아무리 많이 호출되어도 외부 상태를 변경하지 않는다. 순수 함수는 입력값이 동일하면 항상 동일한 결과값을 반환하므로, 예측 가능하고 이해하기 쉬운 코드를 작성할 수 있다. 순수 함수의 주요 특징은 다음과 같다.

- **스레드 안전성:** 순수 함수는 외부 상태에 의존하거나 이를 변경하지 않으므로, 병렬 실행 환경에서도 안전하게 사용할 수 있다.
- **병렬 계산 가능성:** 입력값에만 의존하기 때문에 병렬 처리를 쉽게 적용할 수 있다.
- **디버깅 및 유지 보수 용이성:** 외부 상태와의 상호작용이 없으므로, 프로그램의 동작을 예측하고 디버깅하는 과정이 단순하다.

순수 함수는 0개 이상의 인수를 받을 수 있으며, 항상 동일한 입력에 대해 동일한 결과값을 반환한다. 이 개념은 다음과 같은 블랙박스 모델로 표현할 수 있다.

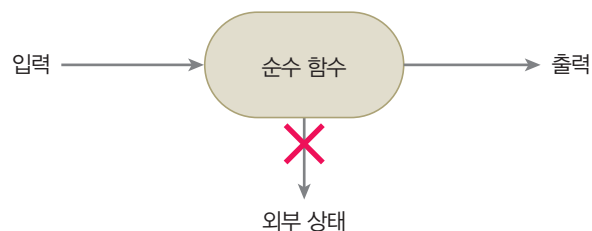


그림 16-7 순수 함수의 블랙박스 모델

순수 함수의 가장 대표적인 예는 수학 함수이다. 예를 들어, `sin()` 함수나 `log()` 함수는 동일한 입력값에 대해 항상 동일한 결과값을 반환한다. 이러한 특성 덕분에 순수 함수는 함수형 프로그래밍의 핵심 개념이 된다.

앞에서도 이야기하였지만, 순수 함수는 스레드에 대하여 안전하다. 순수 함수는 외부 상태를 변경하지 않으므로 여러 스레드가 동시에 호출하더라도 데이터 경쟁(race condition)이 발생하지 않는다. 만약 순수 함수가 아니라면 여러 개의 스레드가 동시에 함수 코드를 수행할 수 없다. 물론 락(lock) 기능을 이용하여 함수를 감쌀 수도 있지만, 이렇게 되면 병렬 처리 이점이 손상된다.

자바와 함수형 프로그래밍

자바에서 순수 함수로만 프로그램을 작성하는 것은 상당히 어렵다. 예를 들어서 입출력 함수들은 대부분 부작용이 있다. `Scanner` 클래스의 `nextLine()` 메소드나 `Random` 클래스의 `nextInt()` 메소드를 생각해 보자. 함수를 호출할 때마다 결과가 달라진다. 따라서 자바에서는 순수한 함수형 프로그래밍이라기보다 함수형 스타일(functional style)을 지원한다고 해야 한다. 자바에서 함수형 스타일로 작성하려면 다음과 같은 규칙을 지켜야 한다.

- **지역 변수만 변경:** 함수는 지역 변수만 변경할 수 있으며, 다른 범위(전역 또는 클래스)의 변수를 변경해서는 안 된다.
- **불변 객체 사용:** 함수가 참조하는 객체는 변경 불가능(immutable)해야 한다. 이를 위해 모든 필드를 `final`로 선언하고, 참조 유형의 필드도 다른 불변 객체를 참조하도록 작성해야 한다.
- **예외 발생 금지:** 함수나 메소드는 예외를 발생시키지 않아야 한다. 예외를 던지는 것은 함수가 값을 반환하지 않고 다른 방식으로 결과를 전달하는 것을 의미하므로, 함수형 스타일에 적합하지 않다. 예외 대신 결과를 `Optional`이나 `Either`와 같은 컨테이너 타입으로 반환하는 방식이 함수형 스타일에 더 가깝다.

객체 지향 프로그래밍과 함수형 프로그래밍

현대적인 자바 개발자는 객체 지향 프로그래밍(OOP) 스타일과 함수형 프로그래밍 스타일을 적절히 활용할 수 있어야 한다. 최근의 자바 프로그램에서는 두 가지 스타일을 자연스럽게 혼합해서 사용한다. 특히 멀티코어 프로세서와 같은 하드웨어 발전 및 SQL과 유사한 선언형 쿼리 문법의 사용 덕분에, 함수형 프로그래밍 스타일이 과거보다 더 많은 주목을 받고 있다.

특징	객체 지향 프로그래밍	함수형 프로그래밍
기본 개념	객체(object)에 기반	변수와 순수 함수(pure function)에 기반
프로그래밍 블록	메소드 호출(method invocation)을 강조	함수 호출(function call)을 강조
프로그래밍 모델	명령형 프로그래밍(imperative programming)	선언형 프로그래밍(declarative programming)
병렬 처리 지원	기본적으로 병렬 처리를 지원하지 않음	병렬 처리를 자연스럽게 지원

표 16-2 객체 지향 프로그래밍과 함수형 프로그래밍 비교

자바에서는 극단적인 객체 지향 스타일과 함수형 스타일이 모두 가능하다. 극단적인 객체 지향 스타일에서는 모든 것이 객체로 표현되며, 프로그램은 객체의 필드를 업데이트하거나 다른 객체의 메소드를 호출하는 방식으로 동작한다. 또 극단적인 함수형 스타일에서는 부작용이 전혀 없는 순수 함수만을 사용하여 프로그램을 작성한다. 모든 함수는 동일한 입력에 대해 항상 동일한 출력을 반환하며, 외부 상태를 변경하지 않는다.

숙련된 자바 개발자는 이러한 두 가지 스타일을 혼합하여 사용할 수 있어야 한다. 예를 들어, `ArrayList`와 같은 컬렉션을 객체 지향 스타일로 작성된 반복자(`Iterator`)를 사용하여 탐색할 수 있다. 반면, 함수형 스타일로 `stream()` 메소드와 람다식을 사용하여 컬렉션 내 데이터의 합계를 계산할 수도 있다. 이처럼 객체 지향과 함수형 스타일은 자바에서 상호 보완적으로 활용되며, 이를 통해 자바 개발자는 더욱 유연하고 효율적인 코드를 작성할 수 있다.

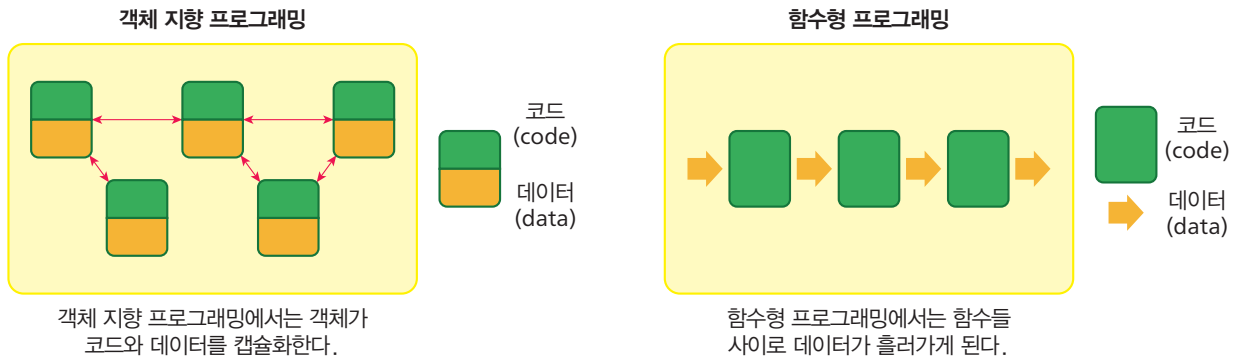


그림 16-8 객체 지향 프로그래밍과 함수형 프로그래밍

SELF TEST



- 1 순수 함수란 무엇인가?
- 2 명령형 프로그래밍 방식과 선언적 프로그래밍 방식을 비교하시오.
- 3 함수형 프로그래밍의 장점을 2가지만 설명하시오.
- 4 병렬 처리 관점에서는 어떤 방식이 더 좋은가? 이유는 무엇인가?

16.2 람다식

함수의 1급 시민 승격

예전 자바에서 메소드는 클래스 안에 갇혀 살아가는 함수였다. Java 8 이전에, 함수는 값(value)처럼 자유롭게 돌아다닐 수 없었다. 즉, 변수에 저장하거나 다른 메소드로 보내는 것은 꿈도 못 꾸던 시절이었다. 이런 상황을 보통 1급 시민(first-class citizen), 2급 시민(second-class citizen) 같은 용어로 표현한다. 예를 들어서 정숫값, 객체, 배열 같은 것들은

자바 세계에서 VIP 대접을 받는 1급 시민이다. 1급 시민은 다음과 같은 동작들이 가능하다.

- 변수에 저장될 수 있다.
- 함수의 인수로 전달될 수 있다.
- 함수의 반환 값이 될 수 있다.

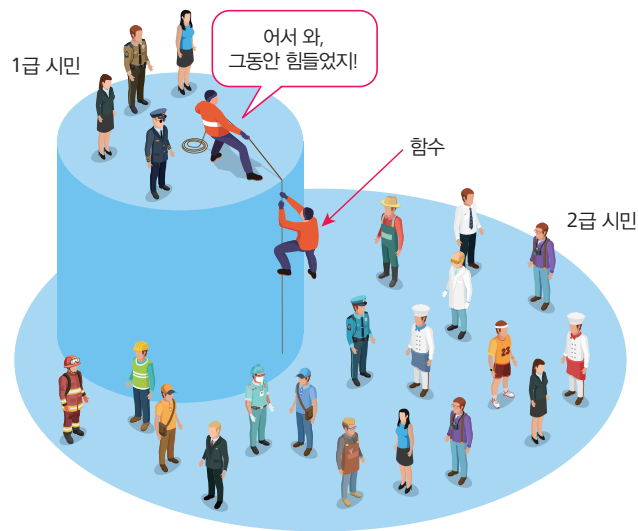


그림 16-9 1급 시민과 2급 시민

Java 8 이전까지 자바에서 함수는 2급 시민이었다. 2급 시민은 값이 아니어서 변수에 저장될 수 없고, 다른 함수로 전달할 수 없었다. 하지만 Scala 및 Groovy와 같은 다른 언어의 실험을 통해, 함수를 1급 시민으로 만드는 것이 유용하다는 것이 밝혀졌다. Java 8에서는 함수가 1급 시민으로 승격되었다. 즉, 함수도 값이 된 것이다. 다음과 같은 일이 가능해졌다.

- 함수도 변수에 저장될 수 있다.
- 함수를 매개변수로 받을 수 있다.
- 함수를 반환할 수 있다.

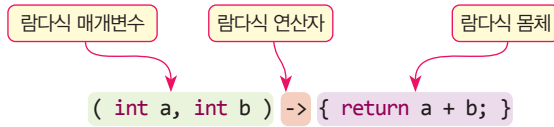
가장 중요한 점은 함수를 다른 함수로 전달할 수 있다는 것이다. 이것을 이용하여 어떤 코드 블록을 매개변수로 만들어서 다른 함수로 보낼 수 있다. 이제 함수도 자유롭게 돌아다니면서 변수에 저장되거나, 메소드의 매개변수로 전달되는 등 좀 더 독립적인 삶을 누릴 수 있게 된 셈이다.

람다식이란?

람다식(lambda expression)은 나중에 실행될 목적으로 다른 곳에 전달될 수 있는 코드 블록이다. 람다식은 이름이 없는 함수라고 할 수 있다. 우리가 람다식을 사용하는 이유는 간결함 때문이다. 람다식을 이용하면 함수가 필요한 곳에 간단히 함수를 보낼 수 있다. 특히 함수가 딱 한 번만 사용되고 함수의 길이가 짧은 경우에 유용하다.

자바에서 (argument) -> (body) 구문을 사용하여 람다식을 작성한다. 간단하게 매개변수 a와 b를 전달받아서 a+b를 계산하여 반환하는 메소드를 람다식으로 정의하면 다음과 같다.

Syntax: 람다식



람다식의 특징은 다음과 같다.

- 람다식은 0개 이상의 매개변수를 가질 수 있다.
- 화살표 ->는 람다식에서 매개변수와 몸체를 구분한다.
- 매개변수의 형식을 명시적으로 선언하거나 문맥에서 추정될 수 있다. 예를 들어 (int a)는 (a)와 동일하다. 빈 괄호는 매개변수가 없음을 나타낸다. 예를 들어 () -> 69와 같다.
- 매개변수가 하나이고 타입을 유추할 수 있는 경우에는 괄호를 사용하지 않아도 된다. 예를 들어 a -> return a * a와 같다.
- 몸체에 하나 이상의 문장이 있으면 중괄호 {}로 묶어야 한다.

다음은 람다식의 몇 가지 예이다.

```
( ) -> System.out.println("Hello World");
(String s) -> { System.out.println(s); }
( ) -> 69
( ) -> { return 3.141592; };
(String s) -> s.length()
(Car c) -> c.getPrice() > 150
(int x, int y) -> {
    System.out.print("결괏값:");
    System.out.println(x + y);
}
(Car c1, Car c2) -> c1.getPrice().compareTo(c2.getPrice())
```

람다식의 활용

람다식을 어디에 사용하면 좋을까? 우리는 변수에 람다식을 할당할 수 있다. 또 함수형 인터페이스를 구현할 때 람다식을 사용할 수 있다. 자바에서는 메소드를 다른 메소드로 전달해야 하는 일이 생각보다 자주 발생한다. 3가지의 예를 들어보자.

1. 자바에서 GUI 코드를 작성할 때, 함수 몸체를 전달하고 싶으면 우리는 익명 클래스를 많이 사용한다. 예를 들어 익명 클래스를 사용하여 버튼의 클릭 이벤트를 처리하는 코드는 왼쪽과 같다. 객체에서 발생하는 마우스 클릭 이벤트를 처리하기 위해 ActionListener를

상속받아서 익명 클래스를 정의하고, 익명 클래스의 객체를 생성하여 버튼의 마우스 리스너로 등록하였다. 하지만 익명 클래스는 상당히 장황한 방법이다. 익명 클래스 대신에 람다식을 이용하면 오른쪽과 같이 간단하게 작성할 수 있다.

```
// 익명 클래스를 이용한 방법
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("버튼 클릭!");
    }
});
```



```
// 람다식을 이용한 방법
button.addActionListener((e) -> {
    System.out.println("버튼 클릭!");
});
```

2. 자바에서 스레드를 작성하려면 먼저 Runnable 인터페이스를 구현하는 클래스부터 작성하여야 한다. 이 인터페이스는 run() 메소드 하나만을 가지고 있다. 역시 왼쪽과 같이 익명 클래스를 작성하여야 한다. 하지만 람다식을 이용하면 오른쪽과 같이 간단하게 메소드를 정의하고, 이것을 Thread 클래스로 전달할 수 있다.

```
// 익명 클래스를 이용한 방법
new Thread(new Runnable() {
    @Override
    public void run() {
        System.out.println("스레드 실행");
    }
}).start();
```



```
// 람다식을 이용한 방법
new Thread(() -> System.out.println
    ("스레드 실행 ") ).start();
```

3. 람다식을 이용하면 배열의 모든 요소를 출력하는 코드에서 forEach()와 같은 함수형 프로그래밍을 쉽게 사용할 수 있다.

```
// 이전의 방법
List<Integer> list =
    Arrays.asList(1, 2, 3, 4, 5);
for (Integer n : list) {
    System.out.println(n);
}
```



```
// 람다식을 이용한 방법
List<Integer> list =
    Arrays.asList(1, 2, 3, 4, 5);
list.forEach(n -> System.out.println(n));
```



예제 16-1

Timer 객체를 람다식으로 작성하기

Timer 클래스로 1초에 한 번씩 'beep'을 출력하는 프로그램을 람다식을 이용하여 작성해 보자.

```
beep
beep
beep
...
```

Timer 객체는 지정된 시간이 지나면 어떤 메소드를 호출할 것인지를 알아야 한다. 이때 사용하는 것이 ActionListener 인터페이스이다. Timer 클래스는 지정된 시간이 지나면 actionPerformed() 메소드를 호출한다. 따라서 1초에 한 번씩 메시지를 호출하려면 아래와 같이 actionPerformed() 메소드와 동일한 시그니처를 가지는 람다식을 작성하고 Timer에 이 객체를 등록하면 된다.

CallbackTest.java

```
1 import javax.swing.Timer;
2
3 public class CallbackTest {
4     public static void main(String[] args) throws InterruptedException {
5         Timer t = new Timer(1000, event -> System.out.println("beep"));
6         t.start();
7         Thread.sleep(10000);           // 10초만 데모
8     }
9 }
```

람다식을 사용하고 있다.

도전문제



- 1 사용자로부터 몇 초 후에 알람을 울릴지 입력받고, 해당 시간이 지나면 콘솔에 "🔊 알람! 시간입니다!"라는 메시지를 출력하는 프로그램을 작성하시오.

SELF TEST



- 1 람다식을 정의하시오.
- 2 2개의 정수를 받아서 2개의 정수를 곱한 값을 반환하는 람다식을 정의하시오.
- 3 다음 코드는 문법적으로 올바른 람다식인가?

```
(int x, int y) -> x % y
```

16.3 동작 매개변수화

함수형 프로그래밍에서 핵심적인 사항은 함수를 다른 함수의 인수로 전달하는 것이다. 왜 함수(즉, 코드가 들어 있는 블록)를 다른 함수로 전달하는 것이 필요할까? 함수로 코드 블록을 보내면 코딩이 편해지기 때문이다. 이번 절에서는 왜 코드 블록을 함수로 보내야 하는지 그 이유를 자세하게 살펴보자. 이것은 개발자들이 한번은 생각해보아야 하는 문제이다.

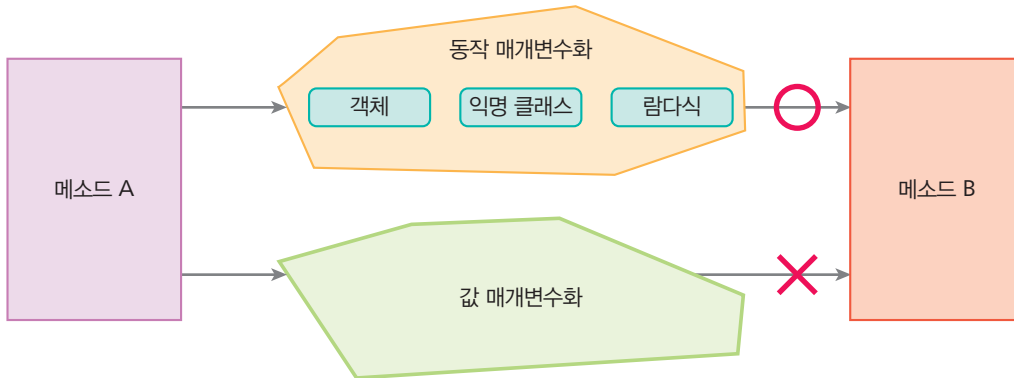


그림 16-10 동작 매개변수화

기업용 프로그램을 작성할 때, 우리가 반드시 기억해야 하는 것은 사용자 요구 사항이 끊임 없이 변경된다는 것이다. 예를 들어 우리가 자동차 영업 사원을 위한 프로그램을 작성한다고 하자. 영업 사원은 자동차 재고를 저장하고 검색할 수 있는 애플리케이션을 원하고 있다. 처음에 영업 사원은 자동차 재고에서 흰색 자동차를 찾는 기능을 원한다고 하였다. 그러나 다음날 “가격이 5,000만 원 이하인 자동차도 찾을 수 있죠?”라고 말할 수 있다. 이를 후 영업 사원은 “색상이 흰색이고 5,000만 원 이하인 자동차도 찾을 수 있나요?”라고 물어 볼 수도 있다. 개발자는 이렇게 변화하는 요구 사항에 부응하면서 최소한의 노력으로 구현 및 유지보수가 간단한 방법을 사용해야 한다.



동작 매개변수화(behavior parameterization) 기법은 고객의 빈번한 요구 사항 변경을 처리할 수 있는 소프트웨어 개발 패턴이다. 이 방법에서는 사용자의 요구를 담은 코드 블록을 생성하고, 이것을 프로그램의 다른 부분에 전달한다. 예를 들어 코드 블록을 `selectCars()`에 인수로 전달할 수 있다. 결과적으로 메소드 `selectCars()`의 동작은 해당 코드 블록을 기반으로 매개변수화된다. 이것이 동작 매개변수화이다.

실생활 예를 들어 보자. 여러분이 동생한테 심부름을 자주 시킨다고 하자. 여러분은 동생한테 빵, 우유, 사과 등의 리스트를 주면서 마트에서 사 오라고 말할 수 있다. 이는 상품 리스트 `mylist`를 인수로 `buy(mylist)` 메소드를 호출하는 것과 같다. 하지만 이번에는 동생이 이전에 한 번도 해본 적이 없는 일을 시키려고 한다. 예를 들어서 ‘은행에서 돈을 찾아오는 일’이다. 이때는 동생에게 할 일을 적은 쪽지를 전달하는 것이 필요하다. 즉 은행에 가서 번호표를 받고, 지급 요구서를 작성하여 직원에게 제출하는 등의 할 일을 순서대로 적어서 주는 것이 편하다. 동생은 이 명령어 리스트가 있으면 쉽게 심부름을 할 수 있을 것이다. 이것은 할 일이 저장된 코드 블록 `code`를 만들어서 `do(code)` 메소드를 호출하는 것과 같다. 만약 동생에게 다른 일을 시키고 싶으면 이 코드 블록만 변경하면 된다.



구체적인 코드로 살펴보자. 자동차 영어 사원이 판매 가능한 자동차들을 리스트에 저장하고 있다. 자동차를 나타내는 `Car` 클래스가 있고, `carList`에 `Car` 객체들을 저장하고 있다고 하자. 지금부터는 자동차 재고 리스트에서 특정한 자동차를 선택하는 문제를 여러 가지 방법으로 구현하면서 예전의 방법과 최신의 방법을 비교해 보자.

```
private static Car[] carArray = {
    new Car(1, "BENS SCLASS", "BLACK", 11000),
    new Car(2, "BNW 9", "BLUE", 8000),
    new Car(3, "KEA 9", "WHITE", 7000)
};

private static List<Car> carList = Arrays.asList(carArray);
```

첫 번째 버전: 매개변수가 없음

고객이 원하는 흰색 자동차를 추려서 리스트로 만들어서 반환하는 `selectCars()`라는 메소드는 다음과 같이 작성할 수 있다.

```
public static List<Car> selectCars(List<Car> inventory) {
    List<Car> result = new ArrayList<>();
    for (Car car : inventory){
        if ("WHITE".equals(car.getColor()))
            result.add(car);
    }
    return result;
}
```

갑자기 고객이 마음을 바꾸고 빨간 자동차가 좋다고 한다. 즉, 이번에는 빨간색 자동차를 골라내야 한다. 어떻게 하면 좋을까? 간단한 해결책은 메소드 안의 코드 `if ("WHITE".equals(car.getColor()))`를 `if ("RED".equals(car.getColor()))`로 변경하는 것이다. 하지만 이 접근 방식은 고객이 원하는 색상이 바뀔 때마다 메소드를 변경하여야 하는 문제점이 있다.

두 번째 버전: 값 매개변수화

메소드 안의 코드를 변경하지 않으려면 어떻게 해야 할까? 이때는 메소드에 색상을 나타내는 매개변수를 추가하면 좀 더 유연한 코드가 된다.

```
public static List<Car> selectCars(List<Car> inventory, String color) {
    List<Car> result = new ArrayList<>();
    for (Car car : inventory){
        if (car.getColor().equals(color))
            result.add(car);
    }
    return result;
}
```

하지만 고객의 요구가 다시 바뀌었다. “너무 비싼 자동차는 구입할 수 없습니다. 5,000만 원 이하만 구매 가능합니다.” 어떻게 해야 할까? 다음과 같이 매개변수를 더 추가하는 함수를 작성할 수도 있지만, 이것은 좋은 솔루션이 아니다.

```
public static List<Car> selectCars(List<Car> inventory, String color, int price) {
    List<Car> result = new ArrayList<>();
```

```

    for (Car car : inventory) {
        if ((car.getColor().equals(color)) && (car.getPrice() <= price))
            result.add(car);
    }
    return result;
}

```

만약 고객이 자동차의 배기량, 형태, 승차 인원 등의 조건을 추가로 요청하면 어떻게 해야 할까? 매개변수를 계속해서 추가하거나 변경하여야 한다.

세 번째 버전: 동작 매개변수화

사용자의 변화하는 요구 사항에 대처하기 위해 많은 매개변수를 추가하는 것보다 더 나은 방법이 필요하다. 만약 우리가 원하는 조건을 검사하는 함수로 작성하여 전달하면 어떨까? 즉 자동차의 속성을 검사하여서, 사용자의 요구 사항에 맞으면 `true`를 반환하는 함수 `test()`를 전달하면 어떨까? 먼저 `test()` 메소드의 형태를 정의해야 하기 때문에 `test()` 메소드의 형태가 정의된 인터페이스 `CarPredicate`를 만든다. 즉, `test()` 메소드의 매개변수나 반환 값을 정의해야 하기 때문에 인터페이스를 작성하는 것이다.

```

public interface CarPredicate {
    boolean test(Car car);
}

```

우리는 `CarPredicate` 객체를 받아서 자동차 객체의 속성을 검사하는 `selectCars()` 메소드를 작성하면 된다. 이것이 바로 동작 매개변수화이다. 메소드가 동작(함수)을 매개변수로 취하고 이를 이용하여 작업을 수행한다. `CarPredicate`를 사용하는 `selectCars()` 메소드는 다음과 같다.

```

public static List<Car> selectCars(List<Car> inventory, CarPredicate p) {
    List<Car> result = new ArrayList<>();
    for (Car car : inventory) {
        if (p.test(car))
            result.add(car);
    }
    return result;
}

```

이 코드는 이전의 방법보다 훨씬 유연하면서 동시에 사용하기 쉽다. 이제 적절한 `CarPredicate` 객체를 생성하여 `selectCars()` 메소드에 전달하기만 하면 된다. `selectCars()` 메소드는 전혀

변경할 필요가 없다. 예를 들어서 사용자가 색상이 흰색인 자동차를 찾아달라고 요청하면, 그에 따라 `CarPredicate`를 구현하는 클래스를 작성하고 객체를 생성하여 전달하면 된다.

```
public class whitePredicate implements CarPredicate {
    public boolean test(Car car){
        return "WHITE".equals(car.getColor()) ;
    }
}

List<Car> whiteCars = selectCars(carList, new whitePredicate());
```

`selectCars()` 메소드의 동작은 전달하는 코드에 따라 달라진다. 즉, `selectCars()` 메소드의 동작을 매개변수화했다. 동작 매개변수화는 컬렉션을 반복하는 논리와 컬렉션의 각 요소에 적용할 동작을 분리할 수 있기 때문에 좋은 방법이다. 한 가지 번거로운 점은 `selectCars()` 메소드는 객체만 받을 수 있기 때문에 해당 코드를 객체 안에 포장해야 한다는 점이다.

네 번째 버전: 익명 클래스 사용

앞의 코드는 익명 클래스를 사용하면 좀 더 간단해질 수 있다. 익명 클래스를 사용하면 클래스를 선언하고 동시에 인스턴스화할 수 있다. 다음 코드는 `CarPredicate` 익명 클래스를 사용하여 객체를 만들어서 흰색 자동차 필터링 예제를 다시 작성하는 방법을 보여준다.

```
List<Car> whiteCars = selectCars(carList, new CarPredicate() {
    public boolean test(Car car) {
        return "WHITE".equals(car.getColor());
    }
});
```

익명 클래스도 나름의 문제점이 있다. 많은 공간을 차지하면서 사용하기에 혼란스럽다. 또한 익명 클래스의 형식은 장황하다. 좋은 코드는 한눈에 이해하기 쉬워야 한다.

다섯 번째 버전: 람다식 사용

Java 8에서 지원하는 람다식을 사용하여 다시 작성할 수 있다.

```
List<Car> whiteCars = selectCars(carList, (Car car)
    -> "WHITE".equals(car.getColor()));
```

이 코드가 이전 시도보다 훨씬 깔끔해 보인다. 코드가 문제 설명에 훨씬 더 가깝게 보이기 시작했기 때문에 좋다. 아래의 그림은 지금까지의 여정을 요약한 것이다.

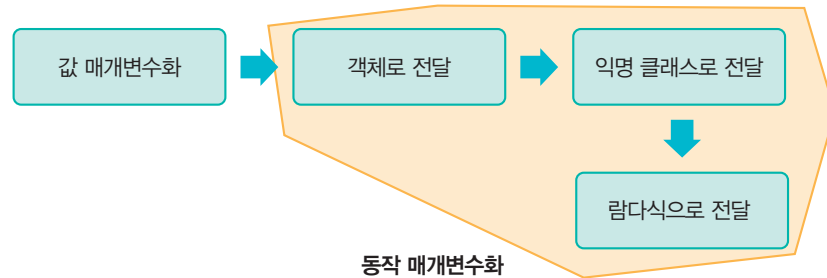


그림 16-11 함수형 프로그래밍으로의 여정

람다식은 간편하다. 하지만 잊으면 안 되는 사실이 있다. 람다식을 사용하려면 람다식을 받아들일 수 있는 인터페이스가 있어야 한다. 이 인터페이스를 '함수형 인터페이스'라고 한다. 위의 코드에서도 `CarPredicate`라는 이름의 인터페이스가 반드시 필요하다. 이것은 다음에 자세히 살펴보자.

SELF TEST



- 1 동작 매개변수화란 무엇인가?
- 2 코드를 함수로 보내는 것의 장점은 무엇인가?
- 3 코드를 함수로 보내는 방법에는 어떤 것들이 있는가?

16.4 함수형 인터페이스

함수형 인터페이스(functional interface)는 단 하나의 추상 메소드만을 가진 인터페이스이다. 우리가 이제까지 사용했던 `ActionListener`, `Runnable`이 모두 함수형 인터페이스의 대표적인 예이다.

```
public interface ActionListener extends EventListener {
    void actionPerformed(ActionEvent e);
}
```

하나의 추상 메소드만을 가진 인터페이스를 함수형 인터페이스라고 합니다.



람다식과 함수형 인터페이스는 불가분의 관계에 있다. 컴파일러가 람다식을 올바르게 컴파일하려면 반드시 람다식에 대응되는 함수형 인터페이스가 미리 정의되어 있어야 한다. 함수형 인터페이스의 정의 없이 람다식을 사용할 수는 없다. 함수형 인터페이스는 람다식으로 구현할 수 있다. 위의 `ActionListener` 인터페이스를 람다식으로 구현하면 다음과 같다.

```
ActionListener listener = e -> System.out.println("액션 이벤트!");
```

위의 코드에서 변수 `listener`는 `ActionListener` 함수형 인터페이스를 구현한 객체이다. 자바에는 `ActionListener` 인터페이스가 이미 정의되어 있기 때문에, 우리는 람다식을 사용하여 이 인터페이스를 구현할 수 있는 것이다. 람다식은 매개변수가 하나이고, 아무것도 반환하지 않는 형태로 작성하여야 한다. 다른 형태는 허용되지 않는다. 결론적으로 람다식을 사용하려면 누군가가 먼저 람다식을 위한 함수형 인터페이스를 정의하여야 한다.

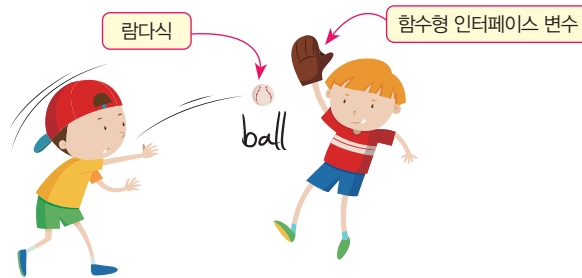


그림 16-12 람다식과 함수형 인터페이스

함수 인터페이스 정의와 람다식

예제 16-2



람다식을 이용하여 간단한 예제를 작성해 보자. 이번에는 함수형 인터페이스도 함께 작성해 보자. 함수형 인터페이스를 정의할 때, `@FunctionalInterface` 어노테이션을 붙이는 것이 좋다. 이 어노테이션은 인터페이스 안에 하나의 추상 메소드만 있는지 확인하고, 다른 추상 메소드를 추가하려고 하면 오류를 발생시킨다. 정수 계산을 나타내는 인터페이스 `MyMath`를 작성해 보자.

```
1 @FunctionalInterface
2 interface MyMath {
3     int calculate(int x);
4 }
5
6 public class Test {
7     public static void main(String args[]) {
8         int value = 9;
9         MyMath s = (int x) -> x * x;
```

함수형 인터페이스

람다식은 `MyMath` 인터페이스의 인스턴스가 될 수 있다.

```

10     int y = s.calculate(value);
11     System.out.println(y);
12 }
13 }

```

MyMath는 함수형 인터페이스이며, 이 안에는 calculate() 함수가 정의되어 있다. calculate()는 정수를 받아서 정수를 반환하는 함수이다. 이 함수형 인터페이스를 이용해서 람다식을 만들 때는 반드시 정수를 받아서 정수를 반환하는 식으로 람다식을 만들어야 한다. 다른 형태의 람다식을 만들면 컴파일 오류가 발생한다.

```

MyMath s1 = (double x) -> x * x;           // 매개변수 타입 오류!
MyMath s2 = (int x, int y) -> x * x;       // 매개변수 개수 오류!

```

람다식도 아무렇게나 만들면 안 된다. 반드시 대응되는 함수형 인터페이스에 맞추어서 작성하여야 한다. 만약 타입과 관계없이 인수를 받거나 값을 반환하려면 제네릭을 사용하여야 한다. 이때는 사용할 때도 제네릭 표기를 해야 한다.

```

@FunctionalInterface
interface MyMath<T> {                               // 제네릭 사용
    T calculate(T x);
}
...
MyMath<Integer> s = (Integer x) -> x * x;          // 제네릭 사용

```

도전문제



1 함수형 인터페이스 MyMath를 기반으로 람다식을 활용하여 다양한 수학 연산(제곱, 절댓값, 2배, 음수 변환 등)을 수행하시오. 아래와 같은 연산을 각각 람다식으로 구현한다.

- 제곱($x \rightarrow x * x$)
- 절댓값($x \rightarrow |x|$)
- 2배로 만들기($x \rightarrow x * 2$)
- 음수로 변환($x \rightarrow -x$)



참고사항: 인터페이스 디폴트 메소드

최근 자바 버전에서는 인터페이스도 디폴트 메소드를 가질 수 있다. 디폴트 메소드를 가지고 있더라도 하나의 추상 메소드만 가지고 있다면, 여전히 함수형 인터페이스이다.

미리 만들어져 있는 함수형 인터페이스

람다식을 사용할 때마다 함수형 인터페이스를 작성해야 한다면, 이것도 상당한 스트레스이다. 이러한 이유로 자바에서 많이 사용되는 함수형 인터페이스는 `java.util.function` 패키지로 제공된다. 몇 개의 예는 다음과 같다.



함수형 인터페이스	반환형	추상 메소드 이름
<code>Supplier<T></code>	<code>T</code>	<code>get()</code>
<code>Consumer<T></code>	<code>void</code>	<code>accept()</code>
<code>BiConsumer<T, U></code>	<code>void</code>	<code>accept()</code>
<code>Predicate<T></code>	<code>boolean</code>	<code>test()</code>
<code>BiPredicate<T, U></code>	<code>boolean</code>	<code>test()</code>
<code>Function<T, R></code>	<code>R</code>	<code>apply()</code>
<code>BiFunction<T, U, R></code>	<code>R</code>	<code>apply()</code>
<code>UnaryOperator<T></code>	<code>T</code>	<code>apply()</code>
<code>BinaryOperator<T></code>	<code>T</code>	<code>apply()</code>

표 16-3 함수형 인터페이스

- **Supplier:** 유형 `T`의 객체를 반환하는 함수형 인터페이스
- **Consumer:** 유형 `T`의 객체에 어떤 동작을 수행하는 인터페이스
- **BiConsume:** 2개의 매개변수를 가지는 `Consumer` 인터페이스
- **Predicate:** 유형 `T`의 입력에 기반하여 부울 값을 반환하는 함수형 인터페이스
- **Function:** 유형 `T`를 받아서 유형 `R`을 반환하는 함수형 인터페이스
- **BiFunction:** 2개의 매개변수를 가지는 `Function` 인터페이스

Function 인터페이스

`Function<T, R>` 인터페이스는 유형 `T`의 객체를 입력받아 유형 `R`의 객체를 반환한다. 추상 메소드 `apply()`는 `T` 타입의 객체를 입력으로 하고 `R` 타입의 객체를 반환한다. 다음과 같이 정의되어 있다.

```
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

입력 데이터를 변환하거나 다른 유형으로 매핑할 때 사용된다.

FunctionTest.java

```

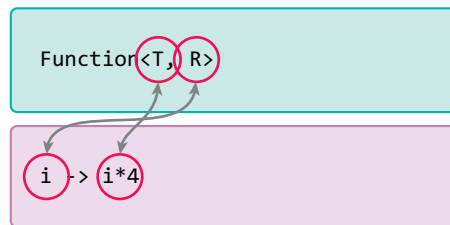
1  import java.util.function.Function;
2
3  public class FunctionTest {
4      public static void main(String[] args) {
5          Function<Integer, Integer> f1 = i -> i * 4;
6          System.out.println(f1.apply(3));
7
8          Function<String, Integer> f2 = s -> s.length();
9          System.out.println(f2.apply("Hello"));
10     }
11 }

```

람다식을 함수형 인터페이스
Function 변수에 저장한다.

람다식을 함수형 인터페이스
Function 변수에 저장한다.

다음과 같이 대응된다.



Predicate 인터페이스

Predicate 인터페이스는 유형 T의 객체를 입력받아 부울 값을 반환한다. 추상 메소드는 boolean test(T t)와 같다. 조건을 검사하거나 필터링에 사용된다.

PredicateExample.java

```

1  import java.util.function.Predicate;
2
3  public class PredicateExample {
4      public static void main(String[] args) {
5          Predicate<Integer> isEven = num -> num % 2 == 0;
6          System.out.println(isEven.test(4));           // 출력: true
7          System.out.println(isEven.test(5));           // 출력: false
8      }
9  }

```

Supplier 인터페이스

Supplier 인터페이스는 매개변수를 받지 않고 유형 T의 객체를 반환한다. 추상 메소드는 T

get()이다. Supplier 인터페이스는 일종의 팩토리 인터페이스로 볼 수 있다. 다음은 Supplier 인터페이스의 구현 예이다.

SupplierExample.java

```
1 import java.util.function.Supplier;
2
3 public class SupplierExample {
4     public static void main(String[] args) {
5         Supplier<String> supplier = () -> "Hello, Supplier!";
6         System.out.println(supplier.get());    // 출력: Hello, Supplier!
7     }
8 }
```

Consumer 인터페이스

Consumer 인터페이스는 유형 T의 객체를 입력받아 특정 작업을 수행하지만, 반환 값은 없는 인터페이스이다. 추상 메소드는 void accept(T t)이다. 다음은 Consumer 인터페이스의 구현 예이다.

ConsumerExample.java

```
1 import java.util.function.Consumer;
2
3 public class ConsumerExample {
4     public static void main(String[] args) {
5         Consumer<String> consumer = s -> System.out.println("Consumed: " + s);
6         consumer.accept("Hello, Consumer!"); // 출력: Consumed: Hello, Consumer!
7     }
8 }
```

BiConsumer 인터페이스

2개의 매개변수를 받아 특정 작업을 수행하며 반환 값은 없다. 추상 메소드는 void accept(T t, U u)이다. 두 매개변수를 사용하는 작업(예: 맵에 값 추가, 데이터 병합)에서 사용한다.

BiConsumerExample.java

```
1 import java.util.function.BiConsumer;
2 import java.util.Map;
3 import java.util.HashMap;
4
5 public class BiConsumerExample {
6     public static void main(String[] args) {
```

```

7      BiConsumer<String, Integer> biConsumer = (name, age) ->
8          System.out.println(name + " is " + age + " years old.");
9      biConsumer.accept("Alice", 30);    // 출력: Alice is 30 years old.
10     }
11 }

```

SELF TEST



- 1 정수형 인수 2개를 받아서 부동소수점형 값을 반환하는 함수 `calc()`를 가지고 있는 함수형 인터페이스 `Test`를 정의하시오.
- 2 `BiFunction` 인터페이스를 이용하여 `x1*x2`를 계산하고 반환하는 람다식을 작성하시오.
- 3 `Predicate` 인터페이스를 이용하여 주어진 정수가 짝수이면 `true`를 반환하는 람다식을 작성하시오.

16.5 메소드 참조

Java 8에서 가장 반가운 변경 사항 중 하나는 람다식의 도입이었다. 이로 인해 익명 클래스를 사용하지 않고 코드의 크기를 크게 줄이면서 가독성을 높일 수 있다. 그러나 때로는 람다식이 실제로 메소드에 대한 호출일 뿐인 경우가 많다. 예를 들어 아래의 람다식이 하는 일은 단지 `println()` 호출이다.

```
s -> System.out.println(s)
```

개발자들은 “호출되는 메소드만을 보낼 수 없을까?”라고 생각했다. 여기서 등장한 것이 메소드 참조이다. 메소드 참조(method reference)는 메소드 자체를 참조하는 것이다. 메소드 호출과 혼동하면 안 된다. `System.out.println(s)`와 같이 쓰면, 이것은 메소드 호출이어서 메소드가 바로 실행되어 버린다. 메소드 참조에서는 메소드가 실행되면 안 된다. 따라서 메소드 호출과는 다른 새로운 표기법이 필요하다. 자바 전문가들은 `::` 기호를 사용하기로 결정했다. `System.out` 객체 안에 있는 메소드 `println()`은 다음과 같이 참조할 수 있다.

```
System.out::println;
```

메소드 참조는 람다식의 축약이라고 생각해도 됩니다.



```
s -> System.out.println(s)
```

```
System.out::println
```

그림 16-13 메소드 참조

메소드 참조의 종류

다음과 같은 4가지 종류의 메소드 참조가 있다. 하나씩 살펴보자.

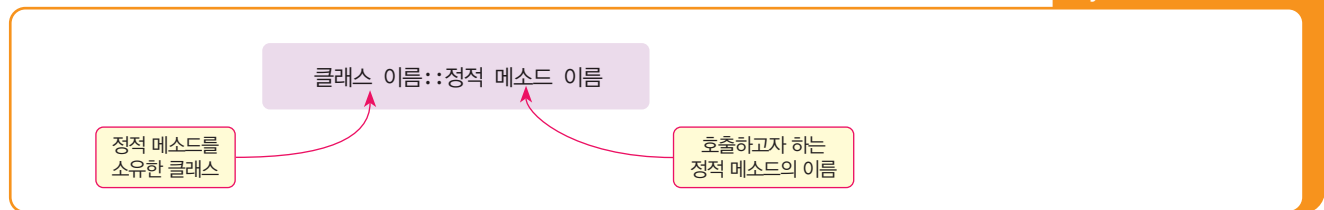
종류	문법	예제
정적 메소드 참조	ContainingClass::staticMethodName	Integer::parseInt
특정 객체의 인스턴스 메소드 참조	ContainingObject::instanceMethodName	System.out::println
특정 유형의 인스턴스 메소드 참조	ContainingType::methodName	String::toUpperCase
생성자 참조	ClassName::new	String::new

표 16-4 메소드 참조의 종류

(1) 정적 메소드 참조

‘클래스 이름::정적 메소드 이름’ 형태는 특정 클래스의 정적 메소드를 참조할 때 사용한다. 정적 메소드 참조는 람다식의 인자들을 그대로 해당 정적 메소드에 전달하는 축약 표현이다. 즉, `x -> ClassName.methodName(x)`와 의미가 동일하다.

Syntax: 정적 메소드 참조



만약 메소드 참조를 변수에 저장해야 한다면 이전에 설명하였던 함수형 인터페이스를 사용하여야 한다. 참조하려는 정적 메소드의 매개변수가 함수형 인터페이스의 추상 메소드의 매개변수와 일치해야 한다. 예를 들어, `BiFunction<T, U, R>` 인터페이스의 추상 메소드 `apply(T t, U u)`는 매개변수 2개와 반환 값을 가지는 정적 메소드에 대응된다.

Test.java

```

1  import java.util.function.BiFunction;
2
3  class Calculator {
4      public static int add(int a, int b) {
5          return a + b;
6      }
7  }
8

```

```

9 public class Test {
10     public static void main(String[] args) {
11         BiFunction<Integer, Integer, Integer> mref = Calculator::add;
12         int result = mref.apply(10, 20);
13         System.out.println("주어진 수의 덧셈: " + result);
14     }
15 }

```



예제 16-3

정적 메소드 참조

앞에서도 이야기하였지만 메소드 참조는 람다식을 간단하게 하는 용도로 많이 사용된다. 리스트의 정수를 정렬할 때, Integer 클래스 안의 정적 메소드 compare()를 사용해 보자.

SortExample.java

```

1 import java.util.Arrays;
2 import java.util.List;
3
4 public class SortExample {
5     public static void main(String[] args) {
6         List<Integer> numbers = Arrays.asList(5, 3, 8, 1, 2);
7
8         // 정렬: 람다식 사용
9         numbers.sort((a, b) -> Integer.compare(a, b));
10        System.out.println(numbers); // 출력: [1, 2, 3, 5, 8]
11
12        // 정렬: 메소드 참조 사용
13        numbers.sort(Integer::compare);
14        System.out.println(numbers); // 출력: [1, 2, 3, 5, 8]
15    }
16 }

```

도전문제

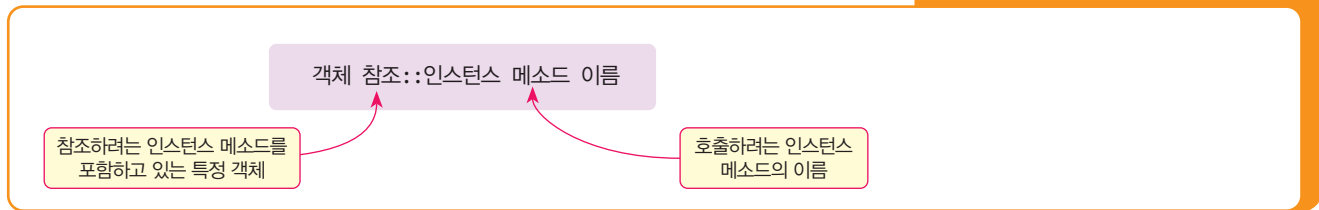


- 1 사람 이름이 담긴 리스트를 람다식과 메소드 참조를 사용해 다양한 기준으로 정렬하라. List<String>에 여러 사람의 이름을 저장한다. 예: "Kim", "Lee", "Park", "Hong", "Choi" 아래 두 가지 방식으로 정렬해 본다. 리스트의 sort() 메소드를 사용해본다.
 - 람다식을 사용해 이름의 길이 순으로 정렬하기
 - 메소드 참조를 사용해 알파벳 순(기본 사전 순)으로 정렬하기

(2) 특정 객체의 인스턴스 메소드 참조

이 형태의 메소드 참조는 람다식에서 특정 객체의 메소드만 호출하는 경우 사용되며, 람다식으로 전달된 인수들이 그대로 그 메소드의 인수로 전달된다.

Syntax: 특정 객체의 인스턴스 메소드 참조



다음의 예제를 보자. `System.out.println()` 메소드는 `System.out` 객체의 인스턴스 메소드이다. 이 메소드를 참조하여 전달받는 문자열을 출력할 수 있다. 여기서도 람다식의 인수가 메소드의 인수로 전달된다.

InstanceMethodReferenceExample.java

```

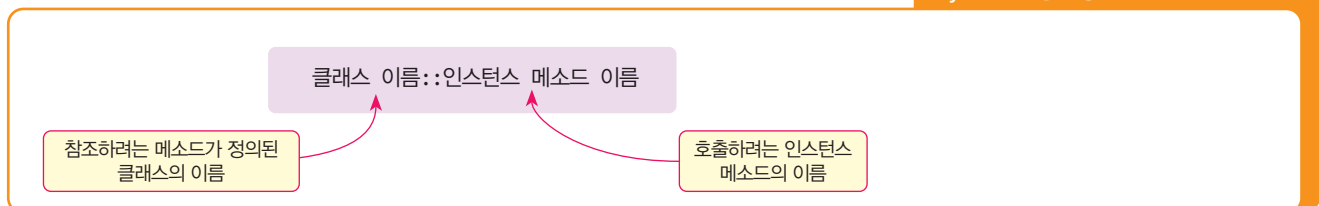
1  import java.util.function.Consumer;
2
3  public class InstanceMethodReferenceExample {
4      public static void main(String[] args) {
5          // 람다식 사용
6          Consumer<String> lambda = s -> System.out.println(s);
7          lambda.accept("Hello with Lambda!");          // 출력: Hello with Lambda!
8
9          // 메소드 참조 사용
10         Consumer<String> mref = System.out::println;
11         mref.accept("Hello with Method Reference!");
12         // 출력: Hello with Method Reference!
13     }
14 }
  
```

Hello with Lambda!
Hello with Method Reference!

(3) 특정 유형의 인스턴스 메소드 참조

‘클래스이름::인스턴스메소드이름’은 특정 클래스의 인스턴스 메소드를 참조하는 문법으로, 람다식의 첫 번째 인수를 메소드 호출 대상 객체로 간주하고, 그 뒤의 인수들을 해당 메소드에 그대로 전달한다. 즉, `(obj, x) -> obj.method(x)` 형태의 람다식과 동일하다.

Syntax: 특정 유형의 인스턴스 메소드 참조



이 메소드 참조는 매개변수로 전달된 객체의 메소드를 호출하는 람다식을 대체한다. 예를 들어, 왼쪽의 람다식은 메소드 참조로 오른쪽처럼 표현할 수 있다. 이 표현은 람다식의 매개변수 `s`를 `toLowerCase()` 메소드의 호출 대상 객체로 사용한다.

`s -> s.toLowerCase()`



`String::toLowerCase`

예제를 살펴보자. 이 예제에서는 문자열을 입력받아 소문자로 변환하는 작업에서 `String::toLowerCase`를 사용한다.

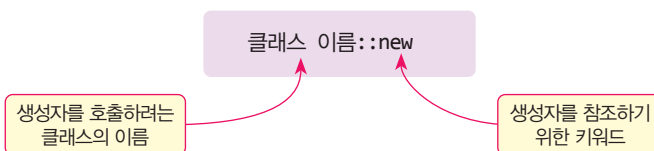
InstanceMethodReferenceExample.java

```
1 import java.util.function.Function;
2
3 public class InstanceMethodReferenceExample {
4     public static void main(String[] args) {
5         // 람다식 사용
6         Function<String, String> lambda = s -> s.toLowerCase();
7         System.out.println(lambda.apply("HELLO")); // 출력: hello
8
9         // 메소드 참조 사용
10        Function<String, String> mref = String::toLowerCase;
11        System.out.println(mref.apply("WORLD")); // 출력: world
12    }
13 }
```

(4) 생성자 참조

‘클래스 이름::new’는 특정 클래스의 생성자(Constructor)를 참조한다. 이 문법은 생성자를 람다식으로 사용하는 경우 코드를 간결하게 만들어 주며, 객체 생성의 의도를 명확히 표현할 수 있다. 함수형 인터페이스는 입력값을 생성자의 매개변수로 전달하고, 반환 값으로 객체를 반환한다.

Syntax: 생성자 참조



기본 생성자를 참조하여 객체를 생성하는 예제는 다음과 같다.

ConstructorReferenceExample.java

```
1 import java.util.function.Supplier;
2
3 public class ConstructorReferenceExample {
4     public static void main(String[] args) {
5         // 람다식 사용
6         Supplier<StringBuilder> lambda = () -> new StringBuilder();
7         System.out.println(lambda.get().append("Lambda Example"));
8
9         // 생성자 참조 사용
10        Supplier<StringBuilder> mref = StringBuilder::new;
11        System.out.println(mref.get().append("Method Reference Example"));
12    }
13 }
```

위의 코드에서 () -> new StringBuilder()는 기본 생성자를 호출하는 람다식이고 StringBuilder::new는 생성자 참조를 통한 대체 표현이다.

(2)와 (3)의 차이점

‘(2) 특정 객체의 인스턴스 메소드 참조’와 ‘(3) 특정 유형의 인스턴스 메소드 참조’는 문법이 비슷해서 헷갈릴 수 있지만, 핵심 차이는 어떤 방식으로 메소드를 호출하는지에 있다.

- 특정 객체의 인스턴스 메소드 참조는 어떤 특정한 객체(instance)가 이미 존재할 때, 해당 객체의 인스턴스 메소드를 참조하는 방식이다. 즉, 메소드가 호출될 때마다 항상 같은 객체를 사용한다.

```
List<String> names = Arrays.asList("Kim", "Lee", "Park");

names.forEach(System.out::println); // 특정 객체(System.out)의 println() 메소드 참조
```

여기서 System.out::println은 System.out이라는 이미 존재하는 객체의 println() 메소드를 참조하는 것이다.

- 특정 유형의 인스턴스 메소드 참조는 이미 존재하는 객체가 아니고 실행시에 첫 번째 인수로 받는 객체의 메소드를 참조한다. 메소드 참조를 사용할 때, 해당 클래스의 인스턴스가 메소드의 첫 번째 매개변수로 전달된다. 즉, 메소드를 호출할 객체가 나중에 결정된다.

```
List<String> words = Arrays.asList("hello", "world", "java");
```

```
words.stream()
    .map(String::toUpperCase) // 특정 유형(String)의 인스턴스 메소드 toUpperCase() 참조
    .forEach(System.out::println);
```

여기서 `String::toUpperCase`는 특정 유형의 인스턴스 메소드 참조이고, 스트림의 각 요소가 수신 객체가 된다. 즉 `map()`이 요소 `s`를 넘겨줄 때 `s.toUpperCase()`가 호출된다. 이것을 랬다식으로 바꾸면 `(String s)->s.toUpperCase()`가 될 것이다.



TIP

만약 메소드 참조를 이해하고 기억하기 어렵다면 그냥 랬다식을 사용하는 것도 방법이다. 익명 클래스를 랬다식으로 변경하면 10줄에서 1줄로 소스의 크기를 줄일 수 있다. 하지만 랬다식을 메소드 참조로 변경한다고 해도 소스의 크기는 거의 줄어들지 않는다. 다만 뒤에서 설명할 스트림 API를 사용할 때는 랬다식보다 메소드 참조가 더 편리하다.

SELF TEST



- 1 (String s) -> Integer.parseInt(s)를 메소드 참조 형식으로 바꾸시오.
- 2 (String s) -> s.toLowerCase()를 메소드 참조 형식으로 바꾸시오.
- 3 List<String> names = new ArrayList<>(); (s -> names.add(s))를 메소드 참조 형태로 바꾸시오.

16.6 스트림

스트림(Stream)은 Java 8에서 도입된 `java.util.stream`의 기능으로, 컬렉션이나 배열, 숫자 범위 등에서 원소의 흐름을 만들어 함수형 스타일로 가공하는 파이프라인이다. 스트림은 데이터를 저장하지 않고, `filter`, `map` 같은 중간 연산을 지연(lazy)시킨 채 이어 붙이다가 `collect`, `sum`, `forEach` 같은 최종 연산을 만나면 한 번에 수행한다(스트림은 일회성). 이름이 비슷해 혼동되지만, 이는 `java.io`의 입출력 스트림과 개념·용도가 다르다. 다만 `Files.lines()`처럼 I/O 데이터를 스트림으로 변환해 처리하는 브리지 메소드는 존재한다. 스트림을 사용하면 외부 반복(for) 대신 내부 반복으로 필터링, 변환, 집계 등을 간결하게 표현할 수 있고, 필요하면 `parallel()`로 병렬 처리도 시도할 수 있다(항상 더 빠른 것은 아님). 즉, 스트림은 복잡한 데이터 처리를 선언적으로 기술하여 가독성과 유지 보수성을 높여 주는 도구다.

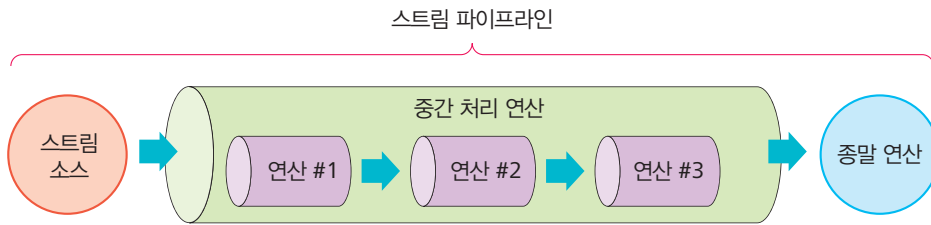


그림 16-14 스트림 파이프라인

UNIX 운영체제에서는 예전부터 스트림이 많이 사용되었다. UNIX 운영체제에서는 대부분의 명령어가 표준 입력에서 데이터를 읽어서 처리한 후에, 표준 출력으로 결과를 내보낸다. 이러한 명령어를 파이프로 연결하면 상당히 복잡한 작업을 수행할 수 있었다. Java 8은 이 아이디어를 기반으로 스트림 API를 추가하였다.

스트림의 개념

간단한 예제를 보면서 스트림의 개념을 설명해 보자.

Test.java

```

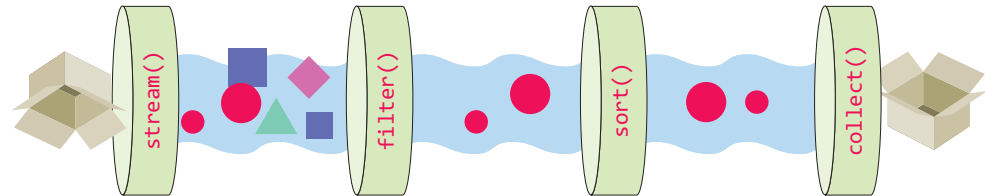
1 public class Test {
2     public static void main(String[] args) {
3         List<String> list = Arrays.asList("Kim", "Park", "Lee", "Choi", "Chee");
4         // ①
5
6         List<String> sublist = list.stream()           // ② 스트림 생성
7             .filter(s -> s.startsWith("C"))          // ③ 스트림 처리
8             .sorted()                                 // ④ 스트림 처리
9             .collect(Collectors.toList());            // ⑤ 결과 생성
10
11         System.out.println(sublist);                  // ⑥ 결과 출력
12     }
13 }

```

[Choi, Chee]

- ① 문자열들을 ArrayList에 저장한다.
- ② ArrayList의 `stream()` 메소드를 호출하여서 스트림을 생성한다. ArrayList에 저장된 데이터들이 하나씩 스트림으로 공급된다.
- ③ 첫 번째 처리 스트림인 `filter()` 메소드는 “C”로 시작하는 문자열만을 통과시킨다. 람다식이 사용되었다.

- ④ `sorted()` 메소드는 문자열을 정렬한다.
 - ⑤ `collect()` 메소드는 결과를 모아서 리스트로 만든다.
 - ⑥ 결과가 저장된 리스트를 출력한다. “C”로 시작하는 문자열들이 추출되고 정렬되어 리스트에 저장되어 있음을 알 수 있다.
- 위의 코드를 그림으로 그려보면 다음과 같다.



스트림의 장점

거의 모든 자바 애플리케이션은 컬렉션을 만들고 처리한다. 이것은 많은 프로그래밍 작업의 기본이다. 예를 들어, 은행에서는 고객과의 거래를 저장하기 위하여 거래들이 저장된 컬렉션을 생성할 수 있다. 그런 다음, 거래 금액을 확인하기 위해 전체 컬렉션을 처리할 수 있다. 이러한 연산이 여러 애플리케이션에서 많이 나타나고 중요하지만, 그동안 컬렉션 처리는 자바에서 완벽하지 않았다.

첫째, 컬렉션에 대한 일반적인 처리 패턴은 ‘찾기(예: 가장 높은 평점의 학생 찾기)’ 또는 ‘그룹화’와 같은 SQL과 유사한 작업이다. 대부분 데이터베이스에서는 이러한 작업을 선언적으로 지정할 수 있다. 예를 들어, 학생들의 데이터베이스에서 SQL 쿼리 ‘SELECT id, MAX(gpa) FROM list’를 사용하면 최고 평점을 받은 학생의 ID를 찾을 수 있다. SQL 쿼리에서 알 수 있지만, 개발자가 최고 평점을 탐색하는 방법을 구현할 필요가 없다. 즉, 개발자가 직접 루프와 변수를 사용하여 가장 높은 값을 추적할 필요가 없다는 의미이다. 개발자는 단지 원하는 작업을 표현하면 된다. 이러한 쿼리를 구현하는 작업은 SQL이 제공한다. 컬렉션에서도 반복문을 사용하지 않고 SQL처럼 선언만 하여서 비슷한 작업을 할 수 있으면 얼마나 좋을까? 스트림 API를 사용하면 다음과 같이 간단하게 표현할 수 있다.

```
List<Integer> result =
    list.stream()                // 리스트(list)를 스트림으로 변환
    .filter(s -> s.getAge() < 25) // 나이가 25 미만인 학생만 필터링
    .sorted(comparing(Student::getGPA).reversed())
    // GPA(학점) 기준으로 내림차순 정렬
    .map(Student::getId)         // Student 객체에서 ID 값만 추출
    .collect(Collectors.toList()); // 결과를 List<Integer>로 변환하여 수집
```


스트림 연산

Stream은 데이터를 처리하기 위한 각종 연산을 제공한다. Stream이 제공하는 연산들은 3가지로 분류할 수 있다.

- **생성 단계:** 스트림 객체를 생성하는 단계이다. 배열이나 컬렉션을 가지고 스트림을 생성할 수 있다.
- **처리 단계:** 입력 데이터를 출력 데이터로 가공하는 연산이다.
- **종말 단계:** 처리된 데이터를 모아서 결과를 만드는 연산이다.

각 단계에 속하는 연산들은 상당히 많기 때문에 전부 설명하는 것은 불가능하다. 자바 API 문서를 참고하도록 하자. 이 책에서는 가장 많이 사용되는 연산만 설명한다.

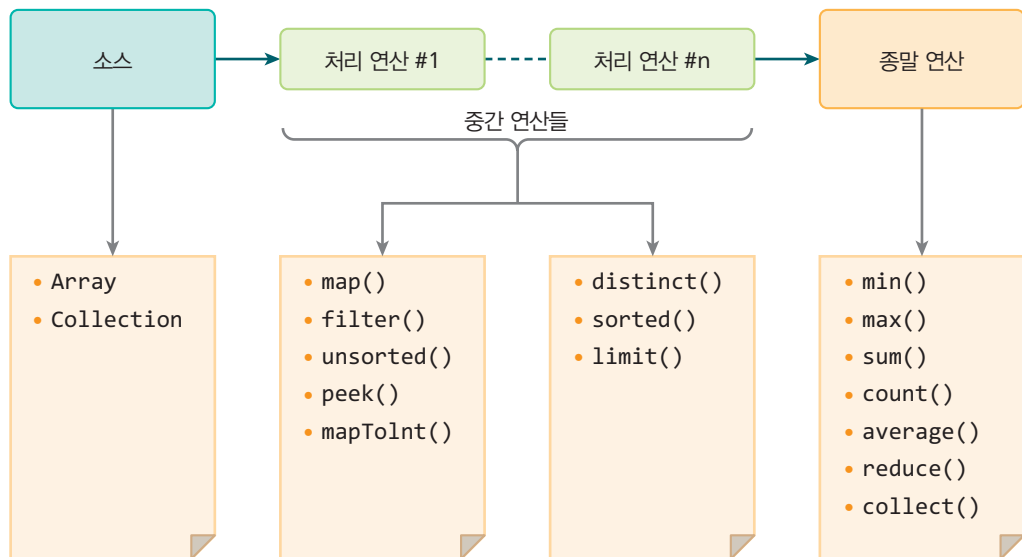


그림 16-15 스트림 연산

생성 단계

스트림은 배열이나 컬렉션에서 만들 수 있다. 간단한 예는 다음과 같다.

```
// 배열에서 만들기
String[] arr = {"Kim", "Lee", "Park"};
Stream<String> s2 = Arrays.stream(arr);

// 컬렉션에서 만들기
List<String> list = Arrays.asList("Kim", "Lee", "Park");
Stream<String> s1 = list.stream();
```


필터링(filter())

필터링은 조건에 맞는 데이터만을 통과시키는 연산이다. `filter()` 메소드를 사용하며, 이 메소드는 람다식을 인수로 받는다. 예를 들어서 문자열 중에서 “P”가 포함된 문자열만 통과시키려면 다음과 같은 코드를 사용한다.

```
List<String> list = Arrays.asList("Kim", "Lee", "Park");
Stream<String> s1 = list.stream()
    .filter(s -> s.contains("P"));           // s1 = "Park"
```

여기서 메소드 체이닝에 주의하도록 하자. `stream()` 메소드가 반환한 객체의 메소드 `filter()`를 연이어 호출하게 된다.

매핑 연산(map())

매핑 연산은 `map()` 메소드를 사용하며 기존의 데이터를 변형하여서 새로운 데이터로 생성하는 연산이다. 이 메소드도 람다식을 인수로 받는다. 예를 들어서 문자열들을 모두 소문자로 변환하려면 다음과 같은 코드를 사용한다.

```
List<String> list = Arrays.asList("Kim", "Lee", "Park");
Stream<String> s1 = list.stream()
    .map(s -> s.toUpperCase());
s1.forEach(System.out::println);           // 출력: KIM, LEE, PARK
```

스트림의 처리 단계에서 람다식 대신에 메소드 참조도 얼마든지 사용할 수 있다. 위의 코드를 메소드 참조를 사용하는 버전으로 변경하면 다음과 같다.

```
List<String> list = Arrays.asList("Kim", "Lee", "Park");
Stream<String> s1 = list.stream()
    .map(String::toUpperCase);
s1.forEach(System.out::println);           // 출력: KIM, LEE, PARK
```

정렬 연산(sorted())

입력된 데이터들을 어떤 기준에 따라 정렬하는 연산이다. 정렬 기준은 `Comparator` 객체이다. `Comparator` 인터페이스의 객체를 생성하여 전달하면 된다. 기준이 주어지지 않으면 기본 정렬된다. 예를 들어서 문자열들을 내림차순으로 정렬하려면 다음과 같은 코드를 사용한다.

```
Stream<String> s1 = list.stream()
    .sorted(Comparator.reverseOrder());
s1.forEach(System.out::println);           // 출력: Park Lee Kim
```

축소 연산(reduce())

`reduce()` 메소드는 스트림의 요소에 대하여 어떤 함수를 가지고 축소 연산을 수행한다. 즉, 요소들을 어떤 함수로 결합하여 하나의 값으로 만들 수 있다. 예를 들어서 정수 리스트의 각 요소는 결과를 생성하기 위해 더하기 연산자를 사용하여 반복적으로 결합할 수 있다. 이 경우, 정수 리스트를 하나의 숫자로 '축소'한다. `reduce()` 메소드에는 2개의 매개변수가 있다. 변수의 초기값과 리스트의 모든 요소를 결합하는 연산이다.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8);

int sum = numbers.stream().reduce(0, (a, b) -> a + b);
System.out.println("합계: " + sum); // 출력: 합계: 36(초깃값 0에서 모든 요소를 더한 결과)
```

종말 단계

종말 단계에서는 입력 데이터들을 모아서 결과를 생성한다. 여러 가지의 결과를 생성할 수 있도록 다양한 메소드들이 제공된다. 구체적으로 최댓값, 최솟값, 합계, 평균값, 개수를 계산할 수 있는 메소드들이 제공된다. 이 중에서 몇 가지만 사용해 보자.

```
int sum = IntStream.of(20, 10, 30, 90, 60) // 정수를 스트림으로 생성해주는 문장이다.
    .sum();                               // 합계를 계산하여 반환한다.
int count = IntStream.of(20, 10, 30, 90, 60) // 정수를 스트림으로 생성해주는 문장이다.
    .count();                             // 합계를 계산하여 반환한다.
```

그리고 종말 단계에서 아주 많이 사용하는 메소드가 `collect()`이다. `collect()`는 스트림 처리의 결과를 컬렉션으로 만들어주는 메소드이다. 어떤 컬렉션으로 만들 것인지는 `collect()` 인수로 결정한다. 가장 많은 타입은 `List` 타입으로 결과를 저장하는 것이다.

```
List sortedList = IntStream.of(20, 10, 30, 90, 60)
    // 정수를 스트림으로 생성해주는 문장이다.
    .sorted()
    .collect(Collectors.toList());

System.out.println(sortedList);           // 출력: [10, 20, 30, 60, 90]
```

forEach() 연산

forEach() 메소드를 사용하면 스트림의 각 항목에 대하여 어떤 특정한 연산을 수행할 수 있다. 예를 들어서 스트림을 지나가는 모든 데이터를 화면에 출력하고 싶으면 다음과 같이 할 수 있다.

```
List<String> list = Arrays.asList("Kim", "Lee", "Park");

// 스트림 생성 후 최종 연산으로 각 요소를 출력
list.stream().forEach(System.out::println);    // 출력: Kim Lee Park
```

여기서는 메소드 참조를 사용하였다. 물론 람다식도 사용할 수 있다. 지금부터 몇 개의 예제를 작성해 보자.

짝수 필터링하기

예제 16-4



1부터 8까지를 저장하는 컬렉션을 만들고, 이 중에서 짝수만을 골라낸 후에, 제공하여 새로운 리스트를 만드는 코드를 스트림 API로 만들어 보자. 람다식을 사용한다.

입력 데이터 = [1, 2, 3, 4, 5, 6, 7, 8]
실행 결과 = [4, 16, 36, 64]

StreamExample1.java

```
1  import java.util.*;
2  import java.util.stream.Collectors;
3
4  public class StreamExample1 {
5      public static void main(String[] args) {
6          List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8);
7          System.out.println("입력 데이터 = " + numbers);    // 원본 리스트 출력
8
9          // 스트림을 이용한 필터링 및 변환
10         List<Integer> result = numbers.stream()    // 리스트를 스트림으로 변환
11             .filter(n -> {    // 필터링: 짝수만 남김
12                 return n % 2 == 0;
13             })
14             .map(n -> {    // 변환: 짝수를 제곱
15                 return n * n;
16             })
```

```

17         .collect(Collectors.toList());           // 결과를 리스트로 변환
18
19     System.out.println("실행 결과 = " + result);
20 }
21 }

```

도전문제



1 아래 조건을 만족하는 프로그램을 작성하시오. 정수 리스트는 다음과 같이 주어진다.

```
List<Integer> numbers = Arrays.asList(10, 15, 20, 25, 30, 35, 40);
```

아래 연산을 Stream API를 사용하여 수행한다.

- **조건 필터링:** 20보다 크고 35보다 작은 수만 필터링한다.
- **변환:** 필터링된 숫자를 3배 증가시킨다.
- 결과 리스트를 출력한다.



예제 16-5

문자열 길이 추출하기

스트림은 각 요소에서 정보를 추출할 때 사용할 수 있다. 단어들의 리스트를 받아서 각 단어의 길이 리스트를 생성하는 코드를 작성해 보자.

```

입력 데이터 = [Java, Stream, Library]
실행 결과 = [4, 6, 7]

```

StreamExample2.java

```

1  import java.util.*;
2  import java.util.stream.Collectors;
3  // 스트림 결과를 리스트로 변환하는 Collectors 클래스
4
5  public class StreamExample2 {
6      public static void main(String[] args) {
7          List<String> words = Arrays.asList("Java", "Stream", "Library");
8          System.out.println("입력 데이터 = " + words); // 원본 리스트 출력
9
10         // 스트림을 이용한 문자열 길이 변환
11         List<Integer> result = words.stream()           // 리스트를 스트림으로 변환
12             .map(String::length)                       // 각 문자열의 길이를 계산하여 변환
13             .collect(Collectors.toList());             // 결과를 리스트로 변환
14
15         System.out.println("실행 결과 = " + result);

```

```
16     }
17 }
```

- 1** 다음과 같은 문자열 리스트가 주어졌을 때, Stream API를 사용하여 다음 조건을 만족하는 결과를 출력하는 프로그램을 작성하시오. 문자열 리스트는 다음과 같이 주어진다.

```
List<String> words = Arrays.asList("Functional", "Interface", "Stream", "Lambda",
    "Java");
```

아래 연산을 순서대로 수행한다.

- 각 문자열을 대문자로 변환한다.
- 길이가 7자 이상인 문자열만 필터링한다.
- 결과를 리스트로 수집하여 출력한다.

도전문제



조건에 맞는 데이터 출력하기

예제 16-6



가전제품들을 ArrayList에 저장하고, 가격이 300만 원 이상인 가전제품의 이름을 출력하는 프로그램을 작성해 보자. 스트림 API를 사용한다.

[TV, Air Conditioner]

StreamTest.java

```
1  import java.util.*;                // 리스트 및 컬렉션 관련 클래스
2  import java.util.stream.Collectors;
3  // Stream 결과를 리스트로 변환하는 Collectors 클래스
4
5  // Product 클래스: 제품 정보를 저장하는 클래스
6  class Product {
7      int id;                        // 제품 ID
8      String name;                  // 제품 이름
9      int price;                    // 제품 가격
10
11     public Product(int id, String name, int price) {
12         super();
13         this.id = id;
14         this.name = name;
15         this.price = price;
16     }
```

```

17 }
18
19 public class StreamTest {
20     public static void main(String[] args) {
21         List<Product> list = new ArrayList<Product>();
22
23         // 리스트에 제품 추가
24         list.add(new Product(1, "NoteBook", 100));
25         list.add(new Product(2, "TV", 320));
26         list.add(new Product(3, "Washing Machine", 250));
27         list.add(new Product(4, "Air Conditioner", 500));
28
29         // 스트림을 이용한 필터링 및 변환
30         List<String> result = list.stream()
31             .filter(p -> p.price > 300) // 가격이 300 이상인 제품만 필터링
32             .map(p -> p.name)           // 필터링된 제품의 이름만 추출
33             .collect(Collectors.toList()); // 결과를 리스트로 변환
34
35         System.out.println(result);
36     }
37 }

```

도전문제



- 1 평균 가격 이상인 제품의 이름을 대문자로 변환해 리스트 만들어보자. `mapToInt(Product::price).average()`로 평균값을 구할 수 있다. 대문자로 만드는 것은 `toUpperCase()`로 가능하다.



TIP

최근 함수형 프로그래밍이 인기를 얻으면서 모든 코드를 함수형으로 작성해야 하는지 고민하는 개발자들도 있을 것이다. 그러나 그렇지 않다. 함수형 프로그래밍을 객체 지향 프로그래밍의 반대 개념으로 정의하는 것은 잘못된 것이다. 각 프로그래밍 방법은 상호 배타적이지 않으며 대부분 개발자들은 두 가지 방법을 모두 사용한다.

SELF TEST



- 1 스트림에서 나타나는 3가지의 단계는 무엇인가?
- 2 스트림의 결과를 리스트로 저장하는 함수는 무엇인가?
- 3 문자열들의 리스트를 받아서 'A'로 시작하는 문자열을 필터링한 후에 알파벳 순으로 정렬하고 리스트로 저장하는 코드를 작성하시오.

스트림을 이용하여 상품 검색하기

Mini Project
수행하기

하나의 상품을 나타내는 Product 클래스를 정의한다.

```
public class Product {  
    private int id;  
    private String name;  
    private float price;  
    ...  
}
```

난이도: ★★☆☆☆

주제: 스트림 처리



여러 가지 상품을 생성하여서 ArrayList에 저장한 후에 사용자로부터 조건을 받아 검색하는 프로그램을 작성해 보자. 랬다식이나 스트림 API, 메소드 참조 등을 적극적으로 사용한다.

상품을 검색하세요.
상품의 이름(*은 모든 상품을 의미): Notebook
상품의 가격 상한: 5000000
검색된 상품은 HP Notebook Model 100입니다.

Summary

- 프로그래밍 패러다임은 크게 명령형 프로그래밍(imperative programming)과 선언형 프로그래밍(declarative programming)으로 나눌 수 있다.
- 선언형 프로그래밍은 해야 할 일(what)에 집중한다. 함수형 프로그래밍에서는 함수들이 계속 적용되면서 작업이 진행된다. 함수형 프로그래밍은 명령문이 아닌 수식이나 함수 호출로 이루어진다.

```
list.stream().map(p -> p.name).forEach(System.out::println);
```

- 순수 함수란 부작용이 없는 함수이다. 부작용이란 함수의 실행으로 인하여 프로그램의 상태가 영구히 변경되는 것이다.

```
int square(int x) { return x * x; }
```

- 동작 매개변수화(behavior parameterization) 기법은 고객의 빈번한 요구 사항 변경을 처리할 수 있는 소프트웨어 개발 패턴이다. 이 방법에서는 사용자의 요구를 담은 코드 블록을 생성하고, 이것을 프로그램의 다른 부분에 전달한다.

```
list.sort((a, b) -> a.price - b.price);
```

- 스트림 라이브러리를 사용하면 ArrayList와 같은 컬렉션에서 조건을 주어서 다양한 처리를 순차적으로 연결할 수 있다.

```
list.stream().filter(p -> p.price > 300).map(p -> p.name).toList();
```

- 메소드 참조는 람다식의 축약 표기로, `x -> Class.method(x)`를 `Class::method`로 줄여 가독성을 높인다.
- 형태는 `Class::staticMethod`, `obj::instanceMethod`, `Class::instanceMethod`, `Class::new`와 같이 네 가지가 있다.

```
words.stream().map(String::toUpperCase).forEach(System.out::println);
```


Exercises

1. 다음 표는 람다식을 메소드 참조로 변환하는 표이다. 빈칸을 채우시오.

람다식	메소드 참조
<code>x -> System.out.println(x)</code>	<code>System.out::println</code>
<code>(String s) -> s.toUpperCase()</code>	<code>String::toUpperCase</code>
<code>(String a, String b) -> a.compareToIgnoreCase(b)</code>	
<code>(String n) -> Integer.parseInt(n)</code>	
<code>(double x) -> Math.abs(x)</code>	
<code>(String s, String prefix) -> s.startsWith(prefix)</code>	
<code>(int a, int b) -> Integer.sum(a, b)</code>	
<code>(T v) -> obj.setValue(v)</code>	
<code>(String x) -> x.isEmpty()</code>	
<code>(String s) -> s.trim()</code>	

2. 다음 코드에서 익명 클래스를 람다식으로 바꾸시오.

```

Collections.sort(numbers, new Comparator<Integer>() {
    @Override
    public int compare(Integer n1, Integer n2) {
        return n1.compareTo(n2);
    }
});

```

3. 다음 코드를 스트림 API와 메소드 참조, 또는 람다식을 이용하여 바꾸시오.

1.

```

List<String> list1 = Arrays.asList("Apple", "Banana", "Pear", "Cherry");
List<String> list2 = new ArrayList<>();

for (String string : list1) {
    if (string.equals("Apple") || string.equals("Cherry")) {
        list2.add(string);
    }
}

```

2.

```
List<String> list3 = new ArrayList<>();

for (String string : list2) {
    list3.add(string + " (Fruits)");
}
```

3.

```
for (String string : list3) {
    System.out.println(string);
}
```

4. 다음의 코드를 스트림 API와 메소드 참조, 또는 람다식을 이용하여 바꾸시오.

```
List<Integer> numbers
    = Arrays.asList(10, 20, 30, 40, 50, 60, 70, 80, 90, 100);

int result = 0;

for (Integer n : numbers) {
    if (n % 2 == 0) {
        result += n * 2;
    }
}

System.out.println(result);
```

Programming Exercises

1. 2개의 문자열을 입력받아 하나의 문자열로 합치는 람다식을 작성하고 테스트하시오.

```
"Hello, " + "World!" → "Hello, World!"
```

난이도: ★★☆☆☆

주제: 람다식

```
BiFunction<String, String, String> concatenate = (s1, s2) -> ???;
```



2. 자바가 제공하는 `Function<T, R>` 인터페이스를 사용하여 문자열을 받아서 문자열의 길이를 반환하는 람다식을 작성하고 테스트하시오.

```
Hello -> 5
```

난이도: ★★☆☆☆

주제: 람다식

3. `int getArea(int side)`라는 함수를 가지는 함수형 인터페이스를 구현하는 람다식을 생성한다. 각 도형의 면적을 계산하는 람다식을 작성하고 테스트하시오. 반올림을 사용한다.

```
interface Shape {
    public int getArea(int side);
}
```

```
정사각형의 면적 (한 변 5): 25
정육각형의 면적 (한 변 5): 65
원의 면적 (반지름 5): 79
```

난이도: ★★☆☆☆

주제: 람다식

4. 정수들의 리스트(`List<Integer>`)를 받아 각 원소의 제곱근을 계산해 `List<Double>`로 반환하시오. 제곱근은 `Math.sqrt()` 메소드를 메소드 참조 형태로 사용한다.

```
[1, 2, 3, 4, 5, 6]
[1.0, 1.4142135623730951, 1.7320508075688772, 2.0,
 2.23606797749979, 2.449489742783178]
```

난이도: ★★☆☆☆

주제: 스트림 API

난이도: ★★★★★
주제: 스트림 API

5. 문자열 리스트가 주어졌을 때, 'a'로 시작하는 문자열의 개수를 반환하는 메소드를 작성하시오. `filter()`와 `count()` 메소드를 사용한다.

전체 리스트 [apple, banana, avocado, apricot, cherry, almond, blueberry]
'a'로 시작하는 단어 개수: 4



`words.stream()`은 리스트를 스트림으로 변환한다, `.filter(word -> word.startsWith("a"))`은 'a'로 시작하는 문자열만 필터링한다. `.count()`는 필터링된 요소 개수를 반환한다.

난이도: ★★★★★
주제: 스트림 API

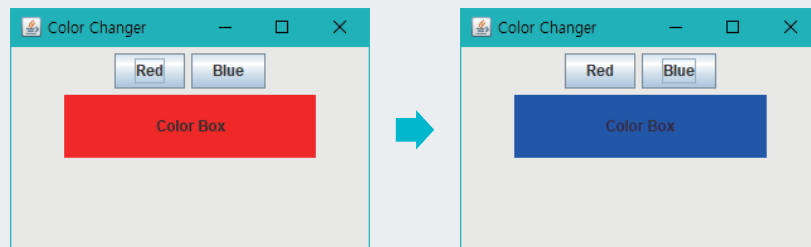
6. 다음과 같이 문자열이 저장된 리스트가 있다고 하자. 리스트에 저장된 문자열 중에서 길이가 3 이상인 문자열만 대문자로 변환하여 새 리스트를 만드는 코드를 작성하고 테스트한다.

```
List<String> list = Arrays.asList("Kim", "Park", "He", "I", "Lee", "Hello", "World");
```

```
[ "KIM", "PARK", "LEE", "HELLO", "WORLD" ]
```

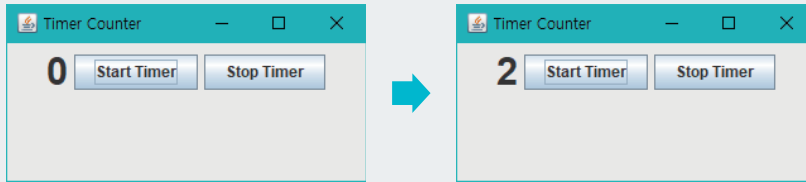
난이도: ★★★★★
주제: 람다식과 스윙 GUI

7. JButton을 클릭하면, JLabel의 배경색이 해당 버튼의 색상으로 변경되는 프로그램을 작성하시오. 단, ActionListener를 람다식으로 구현하여야 한다.



```
blueButton.addActionListener(e -> ???);
```

8. JLabel에 1초마다 숫자가 증가하는 프로그램을 작성하시오. Timer API를 랬다식으로 사용한다.



난이도: ★★★★★

주제: 랬다식 &
Timer API

```
startButton.addActionListener(e -> ???);
```

Hint



9. 하나의 음식을 나타내는 Food 클래스를 작성한다. 뷔페 식당의 메뉴에는 여러 개의 Food 객체가 저장되어 있다고 하자. Food 객체 중에서 칼로리가 300 이하이고 채식에 속하는 음식의 이름만을 추출하여서 리스트로 만드는 코드를 작성하고 테스트하시오.

```
public class Food {
    private String name;           // 음식의 이름
    private boolean isVeg;         // 채식 음식 여부
    private int calories;          // 각 음식의 칼로리
    private Type type;             // 각 음식의 타입(고기, 생선, 기타)
    ...
}
```

난이도: ★★★★★

주제: 스트림 API

채식 & 300칼로리 이하 음식: [Salad, Vegetable Soup, Tofu Stir-fry]

조건: 칼로리 300 이하 & 채식 음식만 필터링 후 리스트 변환

```
List<String> vegetarianLowCalFoods = menu.stream()
    .filter(???)           // 채식 음식만 선택
    .filter(???)           // 칼로리 300 이하
    .map(Food::getName)     // 음식 이름만 추출
    .collect(Collectors.toList()); // 리스트 변환
```

Hint



난이도: ★★★★★

주제: 스트림 API

10. 가상의 Country(국가)와 City(도시) 데이터를 생성한 후, 스트림 API를 사용하여 특정 국가에서 가장 많은 인구를 가진 도시를 찾는 프로그램을 작성하시오. 도시 리스트에서 'KOREA'에 속한 도시 중 가장 인구가 많은 도시를 찾아 출력한다.

KOREA에서 가장 인구가 많은 도시는 Seoul (8,400,000명)입니다.

Hint



`max(Comparator.comparingInt(City::getPopulation))`처럼 정수 키 추출 함수를 넘겨 최대 인구 도시를 구하라. 결과는 `Optional<City>`이므로 적절히 처리한다.

”

CHAPTER

17

네트워크 프로그래밍



+ 학습목표

- www.google.com과 같은 호스트의 IP 주소를 얻는다.
- 온라인 웹 페이지의 데이터를 표시한다.
- 소켓을 이용한 네트워킹 프로그래밍을 작성한다.
- 무연결 UDP 프로그래밍을 수행한다.
- 간단한 채팅 프로그램을 만든다.

+ 학습목차

- 17.1 네트워크 프로그래밍의 기본 개념
- 17.2 인터넷에서 파일 다운로드하기
- 17.3 TCP를 이용한 통신
- 17.4 TCP를 이용한 채팅 프로그램 작성하기
- 17.5 UDP를 이용한 통신
- 17.6 UDP를 이용한 채팅 프로그램 작성하기

핵심 키워드:

TCP/IP 통신, 네트워크 프로그래밍, UDP 통신

네트워크 프로그래밍

17.1 네트워크 프로그래밍의 기본 개념

서버와 클라이언트

네트워크에는 서버(Server)와 클라이언트(Client)가 존재한다. 서버는 여러 명의 사용자에게 서비스를 제공하는 컴퓨터이고, 클라이언트는 서비스를 요청해서 사용하는 컴퓨터를 의미한다. 클라이언트와 서버 컴퓨터는 미리 정의된 프로토콜을 이용하여 서로 간에 통신한다.

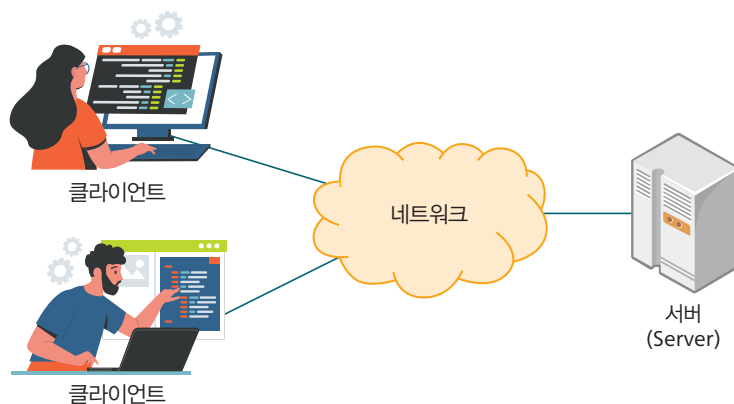


그림 17-1 서버와 클라이언트

서버와 클라이언트의 예를 몇 가지만 들어 보자.

- **웹 서버:** 월드 와이드 웹에는 서비스를 제공하는 웹 서버가 있다. 월드 와이드 웹에서 클라이언트는 인터넷 익스플로러 같은 웹 브라우저이다. 웹 서버와 브라우저 간의 프로토콜은 HTTP라고 불린다.
- **이메일 서버:** 이메일에도 메일 서버가 있고, 클라이언트로 마이크로소프트 아웃룩 같은 이메일 프로그램이 있다. 이메일은 SMTP(Simple Mail Transfer Protocol)를 사용한다.
- **DNS 서버:** www.google.com 같은 친근한 인터넷 주소를 209.218.30.6과 같은 숫자로 된 주소로 변환해주는 서버도 있다. 이 서버는 DNS(Domain Name System) 프로토콜을 사용하고 DNS 서버라고 불린다. 우리가 웹 브라우저에서 www.google.com이라고 치면 DNS 서버가 이것을 IP 주소로 변환하여 준다.

IP 주소(IPv4)

우리가 어떤 사람에게 전화하려면, 그 사람의 전화번호를 알아야 한다. 컴퓨터 세계에서도 마찬가지이다. 하나의 컴퓨터가 다른 컴퓨터와 통신을 하려면 그 컴퓨터의 주소를 알아야 한다. IP 주소(IP address)는 네트워크에 존재하는 컴퓨터를 유일하게 식별하는 숫자이다. IP 주소는 32비트의 이진수이며 이론적으로 인터넷에 존재하는 약 40억 개의 컴퓨터를 식별할 수 있다(IPv4 규격). IP 주소는 보통 숫자 중간에 점(.)을 찍어서 표시한다. 예를 들면 208.168.119.12와 같다.

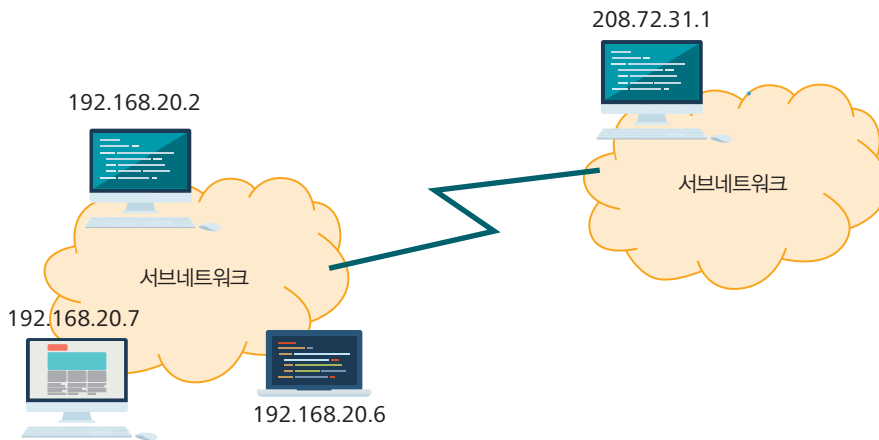


그림 17-2 IP 주소(IPv4)

IP 주소(IPv6)

최근에는 인터넷상의 컴퓨터 증가로 인해 128비트를 사용하는 IPv6 규격이 사용된다. IPv4는 32비트 주소 체계여서 약 40억 개의 주소만 가능하지만, IPv6는 128비트를 사용하므로 3.4×10^{38} 개의 주소를 가질 수 있다. 이는 폭발적으로 늘어나는 인터넷 사용에 대비하기 위한 것이다. IPv6는 이진수 형식으로 표시된다. 즉, 128비트의 001010000..과 같은 형식이다.

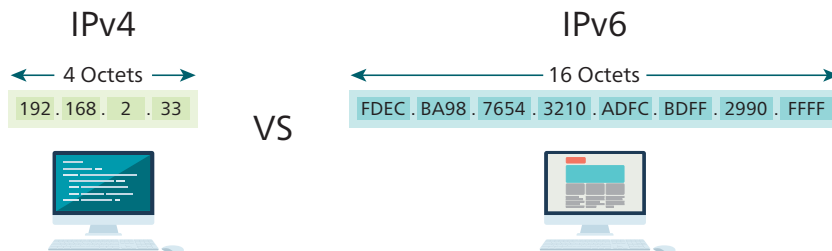


그림 17-3 IPv4와 IPv6

본인 컴퓨터의 IP 주소를 알아보려면 다음과 같이 ipconfig 명령어를 실행하면 된다.

```

C:\WINDOWS\system32\cmd.exe
C:\W>ipconfig

Windows IP 구성

이더넷 어댑터 이더넷:

    연결별 DNS 접미사 . . . . . : 
    링크-로컬 IPv6 주소 . . . . . : fe80::dd9e:9d65:ea02:484%3
    IPv4 주소 . . . . . : 192.168.0.15
    서브넷 마스크 . . . . . : 255.255.255.0
    기본 게이트웨이 . . . . . : 192.168.0.1

C:\W>^

```

그림 17-4 ipconfig 명령 실행 화면

호스트 이름, DNS, URL

호스트 이름은 네트워크상에서 컴퓨터의 이름이다. 호스트 이름은 이름을 짓는 표준인 DNS(Domain Name System)를 사용해서 생성된다. DNS는 보통 인간에게 친근한 문자열을 사용하여 이름을 짓는다. 예를 들어서 www.naver.com과 같다. 만약 DNS를 사용하지 않는다면 207.181.28.9와 같은 IP 주소를 사용하여야 한다. 그러나 컴퓨터들이 통신하려면 이러한 DNS 이름보다는 IP 주소가 필요하다. 따라서 DNS 이름을 IP 주소로 변환하는 작업이 필요하다. 이 작업을 수행하는 서버가 DNS 서버이다. 우리가 웹 브라우저에서 www.naver.com이라고 치면 DNS 서버가 이것에 대응하는 IP 주소로 변환한다.

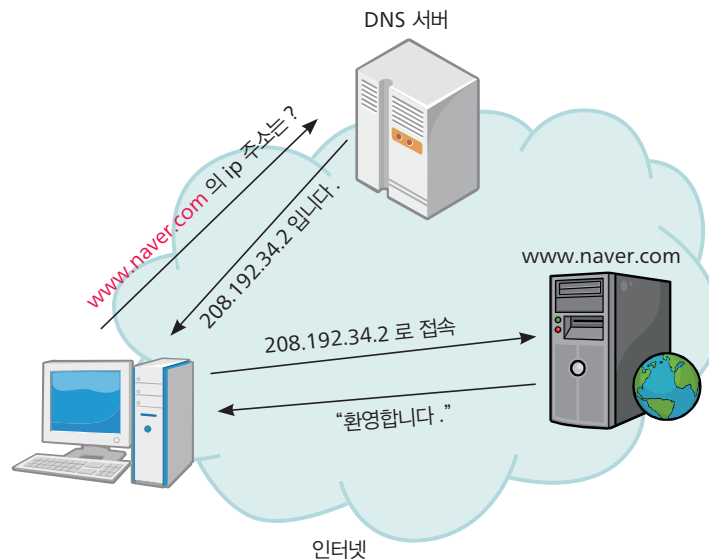


그림 17-5 DNS 서버

모든 컴퓨터는 자기 자신을 가리키는 특별한 호스트 이름과 IP 주소를 가지고 있다. localhost와 127.0.0.1이 해당한다. 이것은 특히 네트워킹 프로그램을 간단히 테스트할 때 유용하다.

URL

DNS와 연관된 것이 URL(Uniform Resource Locator)이다. URL은 인터넷상의 파일이나 데이터베이스 같은 자원에 대한 주소를 지정하는 방법이다. URL은 인터넷에 있는 자원의 위치를 나타내기 위한 규약이다. 자원이라는 것은 대개 파일을 의미하고, 인터넷은 거대한 하나의 파일 시스템이라고 할 수 있다. URL은 바로 우리가 인터넷에서 웹 페이지를 볼 때 웹 브라우저의 주소 칸에 적어주는 값이다. URL은 호스트 이름에 파일의 경로를 붙여서 표시한다.

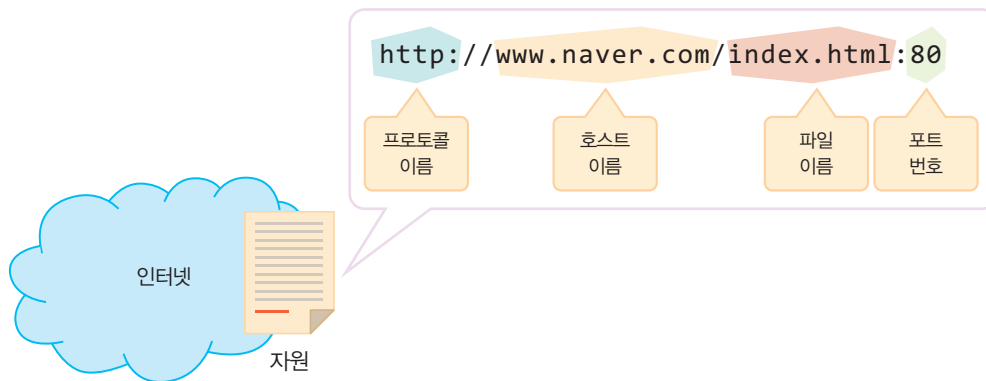


그림 17-6 URL

URL은 두 부분으로 되어있는데 첫 번째 부분은 자원에 접근할 때 사용하는 프로토콜(protocol)을, 두 번째 부분은 자원의 이름을 나타낸다. 예를 들어서 http는 하이퍼텍스트 전송 프로토콜을 나타내고 ftp는 파일 전송 프로토콜을 나타낸다.

자원의 이름은 호스트 이름, 파일 이름, 포트 번호, 참조 등의 필드로 구성되어 있다. 호스트 이름과 파일의 이름은 반드시 필요하지만 다른 필드는 생략이 가능하다.

호스트 이름 → IP 주소 프로그램

예제 17-1



호스트 이름을 받아서 IP 주소를 반환하는 프로그램을 작성해 보자. 인터넷 주소는 InetAddress 클래스가 담당한다. InetAddress 클래스의 getByName() 메소드를 호출하면서 호스트 이름을 전달하면 IP 주소를 저장하고 있는 객체가 반환된다.

Host2Ip.java

```

1 import java.io.IOException;           // 입출력 관련 클래스
2 import java.net.*;                   // 네트워크 관련 클래스
3
4 // 호스트 이름을 IP 주소로 변환하는 프로그램
5 public class Host2Ip {
6     public static void main(String[] args) throws IOException {
7         // 변환할 호스트 이름
8         String hostname = "www.naver.com";
9
10        try {
11            // 호스트 이름으로 InetAddress 객체 생성
12            InetAddress address = InetAddress.getByName(hostname);
13
14            // 호스트 이름의 IP 주소 출력
15            System.out.println("IP 주소: " + address.getHostAddress());
16        } catch (UnknownHostException e) {
17            // 호스트 이름을 IP 주소로 변환할 수 없을 때 처리
18            System.out.println(hostname + "의 IP 주소를 찾을 수 없습니다.");
19        }
20    }
21 }

```

IP 주소: 223.130.200.219

도전문제



- 1 위의 프로그램과는 반대로 IP 주소를 가지고 호스트 이름을 알려면 어떻게 해야 하는가? 다음 코드를 참조한다.

```

String ip = "8.8.8.8";           // 조회할 IP
InetAddress addr = InetAddress.getByName(ip);
String host1 = addr.getHostName();

```

- 2 자신의 컴퓨터 IP 주소를 출력하는 프로그램을 작성하시오. 자신의 컴퓨터의 IP 주소를 알려면 `getLocalHost()` 메소드를 사용하면 된다.

SELF TEST



- 1 IP 주소와 호스트 이름은 어떻게 다른가?
- 2 자신을 가리키는 IP 주소는?
- 3 DNS 서버가 하는 역할은 무엇인가?

17.2 인터넷에서 파일 다운로드하기

우리의 첫 번째 과제는 웹에서 파일을 다운로드하는 것이다. 예를 들어, www.naver.com에 접속해서 웹 서버가 가장 먼저 건네주는 첫 번째 HTML 파일을 받아 보자. 마치 낯선 도시를 여행할 때, 관광안내소에서 가장 먼저 지도를 받는 것과 비슷한 과정이다.

우리는 14장에서 파일에서 데이터를 읽는 방법을 배웠다. 흥미롭게도, 네트워크에서 데이터를 읽는 방법도 이와 매우 유사하다. 파일을 읽을 때 스트림을 사용했던 것처럼, 네트워크에서도 스트림을 연결하여 데이터를 읽으면 된다. 자바에서는 네트워크 프로그래밍을 위한 도구들이 포함된 `java.net` 패키지를 제공한다. 그중에서도 우리의 관심을 끄는 친구는 바로 `java.net.URL` 클래스다. 이 클래스는 우리의 프로그램과 인터넷 어딘가에 있는 원격 컴퓨터의 자원을 연결해 준다. 다만, 이 문을 열어줄지는 상대방(웹 서버)의 허락에 달려 있다.

URL 클래스를 사용하여 원격 컴퓨터에 접근하려면 다음과 같이 URL 생성자를 호출하면서 웹 사이트의 주소를 전달한다. URL이 잘못 지정되었을 경우에 `MalformedURLException` 예외를 발생시키므로 다음과 같이 예외를 처리하여야 한다. URL 객체는 일단 만들어지면 그 내용을 수정할 수 없다.

```
try {
    URL url = new URL("https://www.naver.com/");
    // 여기에 필요한 코드가 들어간다.
} catch (MalformedURLException e){
    // 예외 처리
}
```

URL 객체를 생성하였다고 해서 바로 원격 컴퓨터와 연결되는 것은 아니다. `URLConnection` 클래스를 사용하여 URL과 응용 프로그램 사이의 통신 링크를 생성해야 한다. `URLConnection` 클래스가 성공적으로 생성되면 URL이 지정하는 자원에서 데이터를 읽고 쓸 수 있다. 만약 원격 파일을 읽어야 한다면 스트림을 열어야 한다.

1. URL 클래스의 객체를 생성한다.
2. URL 객체로 연결하기 위해 `URLConnection` 객체를 생성한다.
3. `URLConnection` 객체의 `getInputStream()` 메소드를 호출하여 입력 스트림을 얻는다.
4. 스트림에서 데이터를 읽는다.

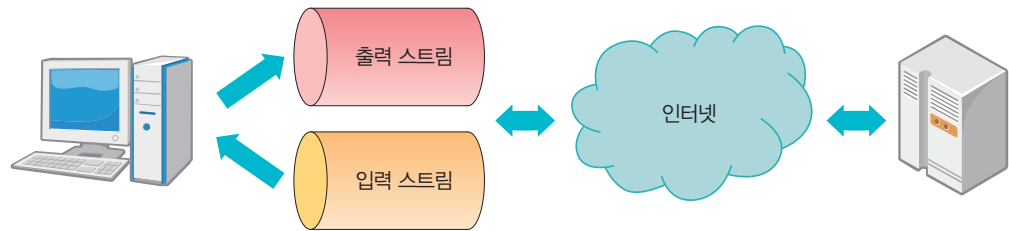


그림 17-7 네트워크에서 데이터를 읽는 방법

URLConnection 객체를 이용하여 외부 URL에 의해 표현되는 서버에 접속할 수 있다. 또한 접속에 앞서서 여러 가지 통신 파라미터들을 설정할 수 있다. 네트워크에서는 항상 오류가 발생할 수 있기 때문에 메소드가 예외를 던지거나, try-catch 문으로 예외를 잡아서 처리하여야 한다. www.naver.com에서 데이터를 읽어서 콘솔에 표시하는 프로그램을 작성하면 다음과 같다.

URLConnectionReader.java

```

1  import java.net.*;                // 네트워크 관련 클래스
2  import java.io.*;                // 입출력 관련 클래스
3
4  // URL 연결을 통해 웹 사이트의 내용을 읽어오는 프로그램
5  public class URLConnectionReader {
6      public static void main(String[] args) throws Exception {
7          // URL 객체 생성(접속할 웹 사이트 주소)
8          URL site = new URL("https://www.naver.com/");
9
10         // URL 연결 객체 생성
11         URLConnection url = site.openConnection();
12
13         // 웹 사이트에서 데이터를 읽기 위한 BufferedReader 생성
14         BufferedReader in = new BufferedReader(
15             new InputStreamReader(
16                 url.getInputStream()));
17         // URL 연결의 입력 스트림을 통해 데이터를 읽음
18
19         String inLine;              // 읽어온 데이터를 저장할 변수
20
21         // 데이터를 한 줄씩 읽어와 출력
22         while ((inLine = in.readLine()) != null)
23             System.out.println(inLine);    // 읽어온 데이터를 콘솔에 출력
24
25         // 리소스 정리(BufferedReader 닫기)
26         in.close();
27     }
28 }

```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="ko" lang="ko">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=euc-kr"
/>
...
```

- 1 자바에서 웹 서버의 자원에 접근하기 위해 사용하는 클래스 이름은 무엇인가?
- 2 URL 객체를 통해 연결을 생성하기 위해 사용하는 클래스는 무엇인가?
- 3 URL을 문자열로 생성할 때 주소 형식이 잘못되었을 경우 발생하는 예외 클래스 이름은 무엇인가?

SELF TEST



17.3 TCP를 이용한 통신

URL과 `URLConnection` 클래스는 인터넷상의 자원에 접근할 수 있도록 지원하는 고수준의 메커니즘이다. 그러나 많은 경우에 **소켓(Socket)**과 같은 저수준의 네트워크 통신 기능이 필요한 경우도 있다. 예를 들면 자바로 클라이언트-서버 응용 프로그램을 만드는 경우이다.

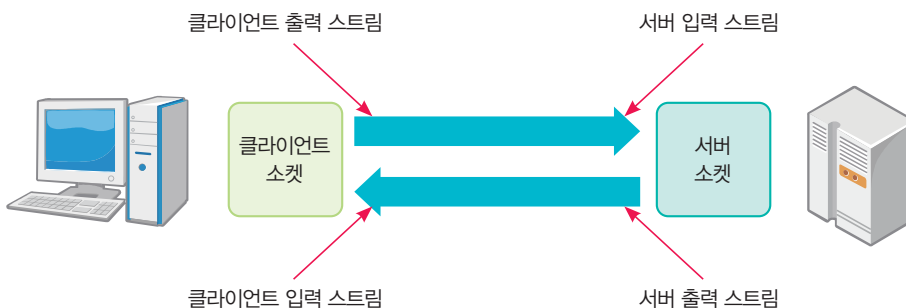


그림 17-8 클라이언트-서버 응용 프로그램

클라이언트-서버 응용 프로그램에서 서버는 특정한 서비스를 제공한다. 예를 들면 데이터베이스 서버는 데이터베이스 쿼리를 받아서 처리한 후에 결과를 클라이언트로 보낸다. 클라이언트와 서버 사이의 통신은 신뢰성이 있어야 한다. 신뢰성이 있다는 것은 데이터의 누락이 없어야 하고, 서버에서 보낸 순서대로 클라이언트측에 도착한다는 의미이다. 일단 필요한 개념들을 간략하게 정리해 보자.

프로토콜

사람들은 동일한 언어를 사용할 때만 다른 사람들과 의사소통을 할 수 있다. 컴퓨터도 마찬가지이다. 컴퓨터 상호 간에 데이터를 주고받기 위해서는 어떤 규칙이 필요하다. **프로토콜 (Protocol)**은 컴퓨터 간에 상호 통신을 할 때 데이터를 원활하고 신뢰성 있게 주고받기 위해 필요한 약속을 규정하는 것이다. 프로토콜 본래의 의미는 외교에서 의례 또는 의정서를 나타내는 말이지만, 네트워크 구조에서는 통신을 원하는 두 개체 간에 무엇을, 어떻게, 언제 통신할 것인가를 서로 약속한 규약이다. 프로토콜에는 정보의 교환 형식과 송수신 방법 등을 규정하는 규칙이 있다. 같은 프로토콜을 사용하면 컴퓨터의 기종이 달라도 컴퓨터 상호 간에 통신할 수 있고, 데이터의 의미를 일치시켜 원하는 동작을 요청할 수 있다.

일상생활에서의 예를 들어 보자. 전화를 걸어서 상대방과 통화를 하는 과정을 살펴보자. 아주 간단한 행동이지만 몇 가지의 절차가 필요하다. 먼저 전화 수화기를 들어야 하고 전화번호를 돌린 후에 상대방이 전화를 받을 때까지 기다리고 상대방이 수화기를 들면 통화가 시작된다. 또 통화가 끝나면 다시 수화기를 내려놓는다. 이와 마찬가지로 컴퓨터 사이에 데이터를 주고받는 경우에도 데이터를 받을 주소를 먼저 알려주어야 하고 상대방이 데이터를 받을 수 있는지를 검사하는 절차가 필요하다. 이것이 프로토콜이다.

컴퓨터 간에도 통신이 이루어져야 하기 때문에 프로토콜은 몇 개의 기능적인 계층으로 나누어서 정의하는 것이 보통이다. 통신 프로토콜은 일반적으로 몇 개의 계층(layer)로 구분한다. TCP/IP 통신 프로토콜은 5개의 계층으로 이루어져 있다.

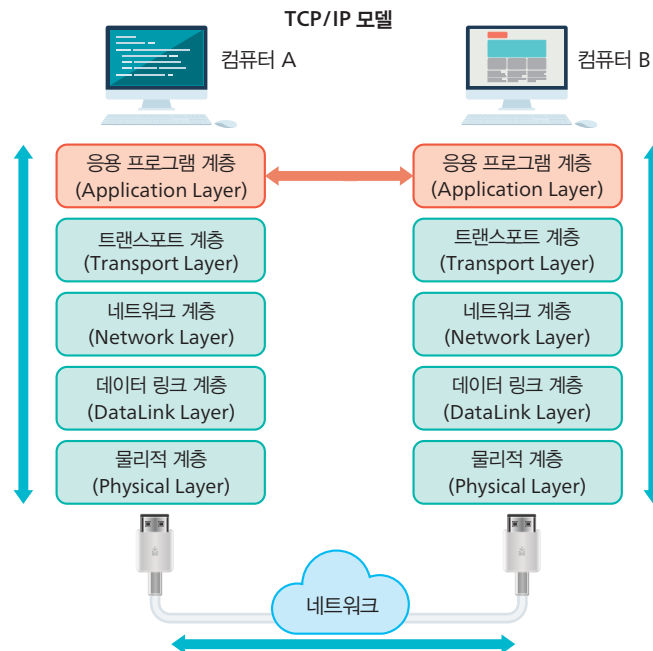


그림 17-9 TCP/IP 통신 프로토콜 계층

프로토콜을 계층적으로 정의하면 프로토콜의 각 부분을 독립적으로 설계하고 테스트할 수 있다. 하나의 계층(layer)의 구현은 아래 계층이 제공하는 서비스를 이용하여 이루어진다. 예를 들어서 이메일을 보내는 SMTP(Simple Mail Transfer Protocol)를 생각해 보자. SMTP 클라이언트는 SMTP 규격을 따르는 어떤 SMTP 서버에게도 메시지를 전송할 수 있다. 가정 집에서는 유선 인터넷 선을 사용하고 비행기에서는 WiFi를 통해 이메일을 보낼 수 있다.






응용 프로그램 계층 (Application Layer) 	응용 프로그램 계층은 네트워크 통신이 필요한 응용 프로그램이 있는 곳이다. 이메일 클라이언트 및 웹 브라우저 같은 응용 프로그램은 전송 계층을 사용하여 원격 호스트에 연결하라는 요청을 보낸다.
전송 계층 (Transport Layer) 	전송 계층은 서로 다른 호스트에서 실행 중인 응용 프로그램 사이의 연결을 설정한다. 안정적인 연결에는 TCP를 사용하고 빠른 연결에는 UDP를 사용한다. 포트 번호를 할당하여 상위 응용 프로그램에서 실행 중인 프로세스를 추적하고 네트워크 계층을 사용하여 TCP/IP 네트워크에 접근한다.
네트워크 계층 (Network Layer) 	네트워크 계층은 패킷을 생성하여 서로 다른 네트워크로 전송한다. IP 주소로 패킷의 소스와 대상을 식별한다.
데이터 링크 계층 (Data Link Layer) 	프레임(frame)을 생성하고, 점대점 방식으로 프레임을 이동시킨다. 이 프레임은 캡슐화된 패킷을 감싸고 있으며, MAC 주소를 사용하여 소스와 대상을 식별한다.
물리적 계층 (Physical Layer) 	프레임 안의 비트들이 전기, 빛, 무선 신호 등으로 부호화된다. 예를 들어 RS-232, SONET, WiFi와 같은 프로토콜이 있다.

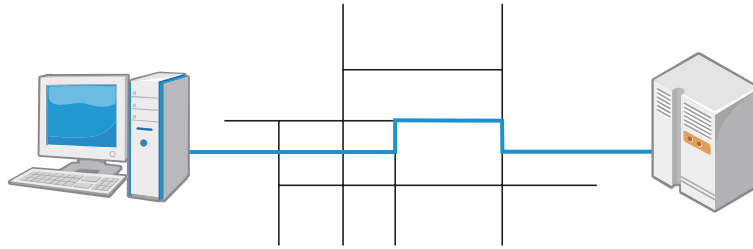
표 17-1 TCP/IP 통신 프로토콜 계층

TCP/IP 통신을 이용하기 전에 전송 계층 중에서 TCP를 사용할지 UDP를 사용할지 결정하여야 한다. 먼저 TCP와 UDP에 대하여 간단히 살펴보자.

TCP

TCP(Transmission Control Protocol)는 신뢰성 있게 통신하기 위하여 먼저 서로 간에 연결을 설정한 후에 데이터를 보내고 받는 방식이다. TCP는 보통 전화와 비슷하다고 이야기한다. 전화를 하기 위해서는 먼저 전화번호를 눌러서 상대방이 받으면 통화를 할 수 있다. 통화가 끝나면 연결은 종료된다. TCP는 신뢰성 있게 데이터를 보낼 수 있다. 즉 중간에 데이터들이 잘 도착하는지를 상대방의 응답을 통해 확인하여 분실된 데이터가 있으면 다시 보낸다. 또한 데이터를 받는 순서가 데이터를 보내는 순서와 동일하게 관리된다. 반면 단점은, 연결하는 과정과 연결을 해제하는 과정에서 상당히 많은 시간이 걸린다는 것이다. 이는 짧은 데이터를 보내는 경우에는 상당한 부담이 된다.

❶ 먼저 가능한 경로 중에서 하나가 결정된다.



❷ 데이터는 패킷으로 나누어지고 패킷에 주소를 붙여서 전송한다.

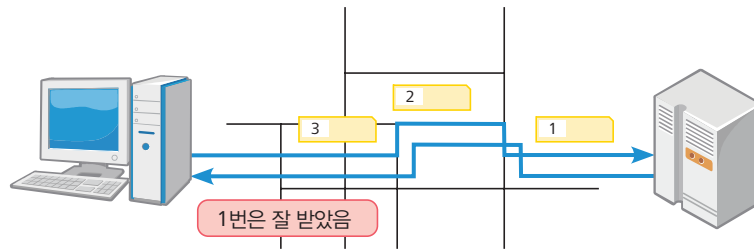


그림 17-10 TCP

HTTP(Hypertext Transfer Protocol), FTP(File Transfer Protocol), Telnet 등은 모두 TCP를 사용한다. TCP를 사용해야만 데이터의 순서가 보장되기 때문이다. FTP로 파일을 인터넷에서 다운로드할 때 파일 안의 데이터가 뒤죽박죽된다면 아무도 이용하지 않을 것이다.

TCP는 신뢰성 있는 점대점 통신 채널을 제공한다. 따라서 클라이언트-서버 응용에서 사용될 수 있다. TCP를 이용하여 통신하기 위해서 클라이언트와 서버 프로그램은 서로 간의 연결을 만들어야 한다. 각 프로그램은 소켓을 연결의 양 끝점에 결합한다. 데이터를 주고받기 위해서는 클라이언트와 서버는 연결에 붙어있는 소켓에서 읽고 쓴다.

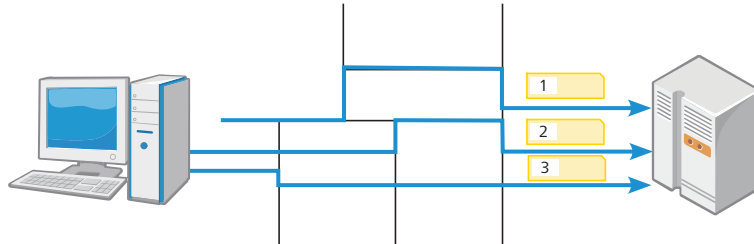
UDP

UDP(User Datagram Protocol)는 TCP와는 달리 연결하지 않고 데이터를 몇 개의 고정 길이의 패킷(다이어그램이라고 불린다)으로 분할한 다음, 패킷의 앞에 주소를 붙여서 데이터를 전송하는 방식이다. UDP는 데이터를 여러 개의 편지에 나누어 보내는 것으로 설명할 수 있다. 편지에는 주소가 붙어 있고 우체국에서는 주소를 보고 편지를 배달한다. 편지는 배달 중간에 분실될 수도 있고 배달되는 순서가 바뀔 수도 있다. 따라서 UDP는 높은 신뢰도가 필요하지 않은 통신을 위하여 사용된다.

UDP의 장점은 연결 절차가 필요 없으므로 빠르고 효율적인 통신이 가능하다는 것이다. UCC와 같은 인터넷상의 동영상 서비스는 일반적으로 UDP로 서비스를 제공한다. 약간의 패킷의 손실이 있어도 동영상을 보는 데 지장이 없기 때문이다. P2P 방식의 네트워크 게임

에서는 TCP와 UDP를 병행해서 사용한다. 캐릭터의 이동처럼 비교적 중요하지 않은 부분은 UDP를 사용한다.

- ❶ 데이터를 패킷으로 나누어서 패킷에 주소를 붙이고 전송한다.



- ❷ 패킷의 순서가 지켜지지 않으며, 패킷이 분실될 수도 있다.

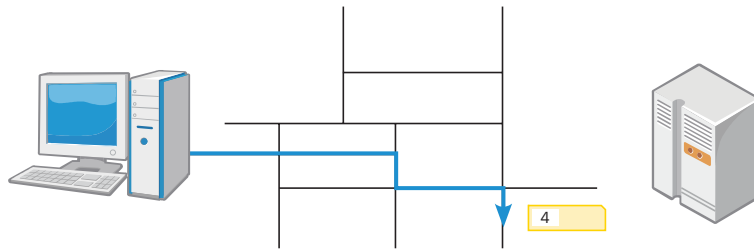


그림 17-11 UDP

포트

보통 하나의 컴퓨터에는 하나의 물리적인 통신선으로 외부와 연결되어 있다. 그러나 컴퓨터 안에서는 여러 개의 응용 프로그램들이 네트워크를 사용할 수 있다. 따라서 하나의 통신선을 타고 들어오는 데이터를 각각의 응용 프로그램에 차질 없이 배달하기 위해서는 각각의 응용 프로그램이 사용하는 가상의 통신 선로가 필요하다. 이것이 바로 포트(port)이다. 하나의 컴퓨터 안에는 여러 개의 포트가 존재하고 있으며, 인터넷을 통하여 데이터를 보내려면 반드시 어떤 포트를 사용할 것인지를 지정하여야 한다. 따라서 네트워크를 통하여 전달되는 모든 데이터의 주소는 특정 컴퓨터를 가리키는 IP 주소와 포트 번호로 구성된다. 포트 번호는 0에서 65535까지의 정수를 사용하여 표기한다. 0에서 1023까지의 번호는 미리 예약되어 있으며(well-known port), 이 번호들은 대개 FTP와 같이 잘 알려진 서비스에 대해서 미리 할당되어 있다. 따라서 응용 프로그램에서는 1023 이상의 번호를 사용하여야 한다.

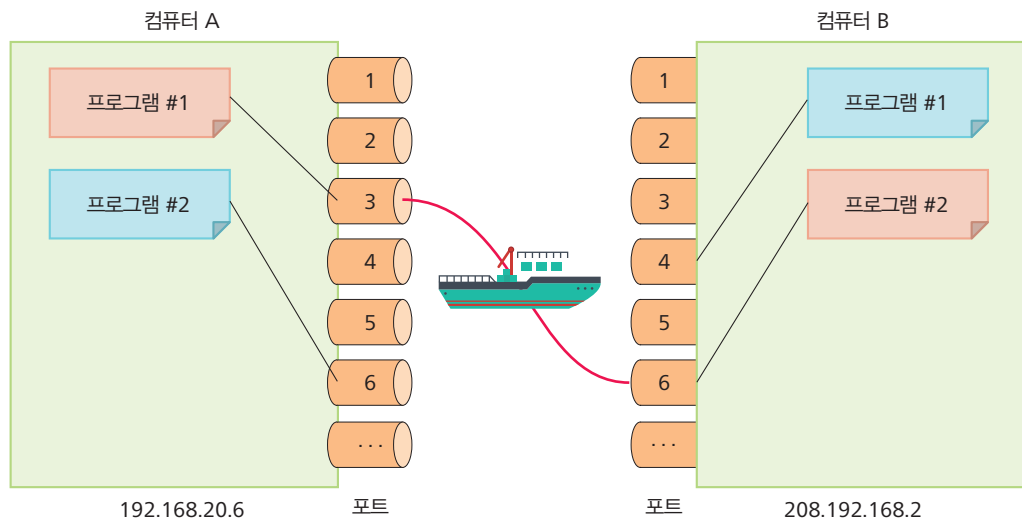


그림 17-12 포트

포트 번호는 IP 주소에다가 콜론(:)을 붙여서 표시한다. 예를 들어서 208.168.119.12의 포트 번호 80은 208.168.119.12:80과 같이 표시한다.

소켓

소켓은 TCP를 위한 구조이다. TCP를 사용하여 응용 프로그램끼리 통신하기 위해서는, 먼저 연결을 해야 하는데 연결을 하기 위해서는 연결 끝점(end point)이 있어야 한다. 이 연결 끝점을 소켓(Socket)이라고 한다. 소켓은 앞에서 설명한 포트를 이용하여 만들어진다. 하나의 포트에 하나의 소켓을 만들어 결합한다. 소켓은 개념적으로 응용 프로그램과 포트 사이에 존재한다고 생각할 수 있다.

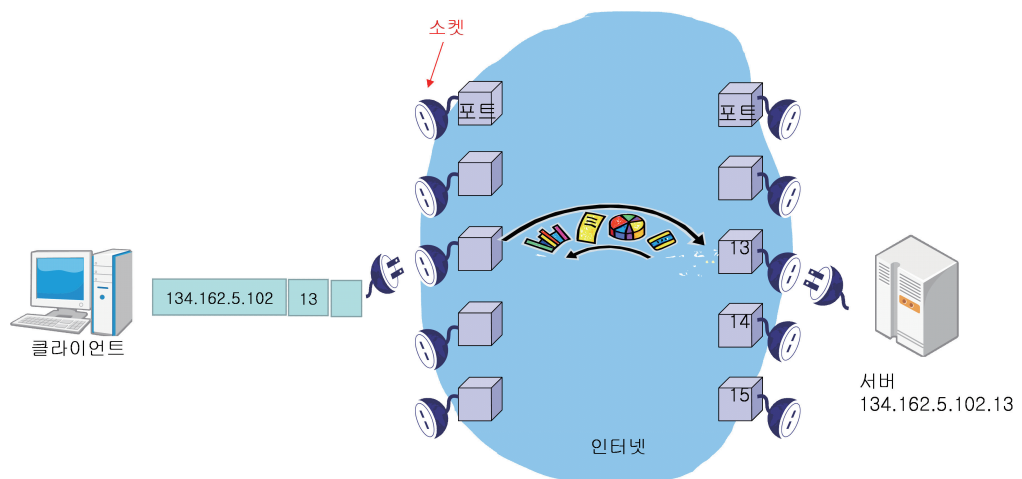


그림 17-13 소켓

날짜 서버에 연결하기

예제 17-2



인터넷에 보면 정확한 현재 시각을 알려주는 서버들이 존재한다(원자 시계로 측정된 시각이다). 여기서는 미국 시각을 알려주는 서버(NIST 서버)에 소켓을 이용하여 접속해 보자. 소켓은 Socket 클래스로 제공된다. Socket 클래스 생성자의 첫 번째 인수는 사이트 주소이고 두 번째 인수가 바로 포트 번호이다. NIST 서버를 지정하는 소켓을 생성하면 바로 서버와 연결된다. 소켓으로부터 입력 스트림을 얻어서 스트림에 읽으면 현재 시각을 알 수 있다.

SocketTest.java

```

1  import java.io.*;                // 입출력 관련 클래스
2  import java.net.*;               // 네트워크 관련 클래스
3  import java.util.Scanner;        // 입력을 처리하기 위한 Scanner 클래스
4
5  // 소켓을 사용하여 원격 서버와 통신하는 프로그램
6  public class SocketTest {
7      public static void main(String[] args) throws IOException {
8          // 소켓을 사용하여 원격 서버(time-c.nist.gov)의 시간 데이터를 가져옴
9          try (Socket s = new Socket("time-c.nist.gov", 13)) {
10             // 서버와 포트 번호 설정
11             // 서버에서 데이터를 읽기 위한 입력 스트림 생성
12             InputStream inStream = s.getInputStream();
13             Scanner in = new Scanner(inStream);
14             // 입력 스트림을 Scanner로 감싸서 데이터를 처리
15
16             // 서버로부터 데이터를 한 줄씩 읽어서 출력
17             while (in.hasNextLine()) {
18                 String line = in.nextLine();           // 한 줄 읽기
19                 System.out.println(line);              // 읽은 데이터를 콘솔에 출력
20             }
21         } // try-with-resources 기능을 사용하여 소켓을 자동으로 닫음
22     }
23 }
```

```
60707 25-02-01 00:39:03 00 0 0 222.6 UTC(NIST) *
```

- 1 NIST 시간 서버(time-c.nist.gov)가 보내주는 전체 문자열에서 날짜 정보(예: 24-03-22)만 출력하도록 코드를 변경하시오.

도전문제



SELF TEST



- 1 TCP 통신에서 새로운 연결이 만들어지는 과정을 설명하십시오.
- 2 TCP와 UDP의 차이점은 무엇인가?
- 3 소켓에서 입력 스트림 객체를 얻는 메소드 이름은 무엇인가?

17.4 TCP를 이용한 채팅 프로그램 작성하기

우리는 자바로 서버와 클라이언트를 제작해 볼 것이다. 서버와 클라이언트라고 해서 너무 거창하게 생각하면 안 된다. 서버의 기능도 천차만별이다. 아주 간단한 서버도 있다. 두 사람이 간단한 채팅을 할 수 있는 프로그램을 작성해 보자. GUI도 없다. 프로토콜도 'quit' 뿐이다.



그림 17-14 간단한 채팅 프로그램

서버와 클라이언트를 제작할 때 꼭 알아야 하는 개념이 있다. 다음 그림을 보자. 클라이언트와 서버가 연결되어 있고 소켓을 통해 서로 데이터를 주고받고 있다.



그림 17-15 클라이언트와 서버가 소켓으로 데이터를 주고받고 있다.

전혀 문제가 없는 것처럼 보인다. 하지만 문제가 있다! 만약 중간에 다른 클라이언트가 이 서버에 접속을 시도하면 어떻게 될까?

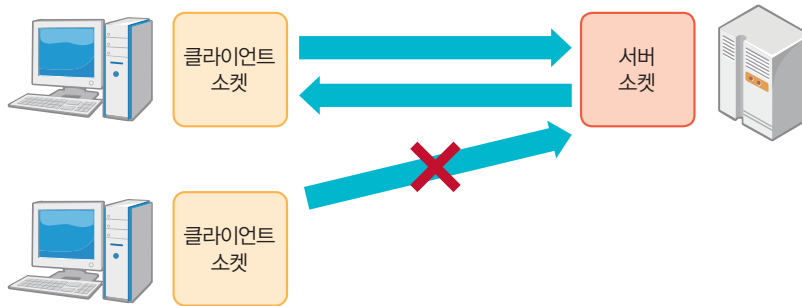


그림 17-16 하나의 소켓에 2개의 컴퓨터가 동시에 연결될 수 없다.

두 번째 클라이언트는 서버에 접속할 수 없다. 왜냐하면 서버의 소켓을 이미 첫 번째 클라이언트가 독점하여 사용하고 있기 때문이다. 하나의 소켓에 2개의 컴퓨터가 동시에 연결될 수는 없다. 데이터가 섞이기 때문이다. 두 번째 클라이언트는 첫 번째가 끝나기를 기다려야 할까? 하지만 서버는 동시에 여러 개의 클라이언트를 상대하여야 한다. 따라서 새로운 접근 방법이 필요하다.

서버는 연결 요청만을 받는 소켓을 따로 가지고 있다. 모든 클라이언트는 이곳으로 연결 요청을 해야 한다. 연결 요청이 승인되면 서버는 해당 클라이언트를 상대하는 새로운 소켓을 만든다. 클라이언트는 이 새로운 소켓을 사용하여 데이터를 주고받는다.

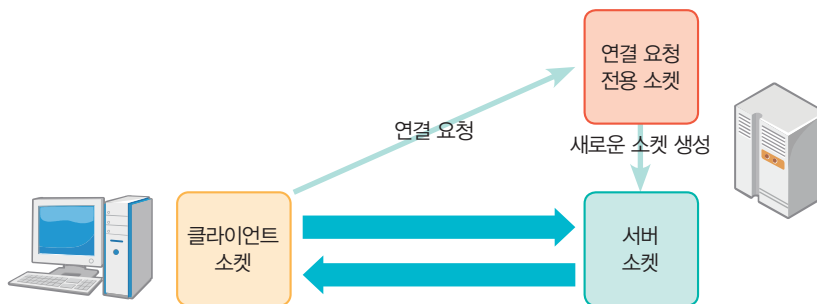


그림 17-17 서버는 연결 요청 전용 소켓으로 클라이언트를 상대한다.

예를 들어 어떤 컴퓨터에서 FTP 서비스를 제공하는 서버가 수행되고 있다고 가정하자. FTP 서버는 서비스 전용 포트 번호(21번으로 고정되어 있다)로 연결 요청이 들어오기를 기다린다. 클라이언트는 서비스 전용 포트 번호로 연결 요청 메시지를 보낸다. 별문제가 없으면 서버는 연결 요청을 받아들이고 서버는 새로운 포트 번호를 가지는 새로운 소켓을 만든다. 이

후부터는 새로 만든 소켓을 이용하여 서버와 클라이언트는 파일을 읽고 쓸 수 있다. 연결을 요청하는 전용 21번 포트는 다른 클라이언트를 위해 그대로 두어야 한다.

Socket과 ServerSocket 클래스

자바에서 Socket 클래스와 ServerSocket 클래스는 클라이언트 측과 서버 측을 구현할 때 사용된다. Socket 클래스가 모든 특수한 시스템의 세부 사항을 감추어 주기 때문에, 자바 프로그램은 플랫폼 독립적일 수 있다. ServerSocket 클래스는 서버가 클라이언트에 대한 연결을 기다리고 받아들일 수 있는 소켓을 구현한다. 이 절에서는 먼저 소켓을 이용하여 어떻게 서버를 제작할 수 있는지를 살펴보자. 자바에서 서버는 5개의 단계를 거쳐서 작성된다.

(1) ServerSocket 객체 생성

```
ServerSocket server = new ServerSocket(portNumber, queueLength);
```

위와 같이 ServerSocket 생성자를 호출하면 포트 번호가 portNumber인 포트를 기반으로 하는 소켓을 생성한다. queueLength는 서버에 연결되기를 기다리는 클라이언트의 최대 개수이다. portNumber는 클라이언트가 서버 컴퓨터에서 서버 애플리케이션을 찾기 위해 필요하다. 각 클라이언트는 이 포트 번호를 이용하여 서버에게 연결을 요청해야 한다.

(2) accept() 메소드 호출

서버는 클라이언트가 연결을 시도하기를 기다린다. ServerSocket의 메소드인 accept()를 호출하면 된다.

```
Socket clientSocket = server.accept();
```

accept() 메소드는 클라이언트와 연결이 되면 새로운 Socket 객체를 생성하여 반환한다. 이 새로운 Socket 객체를 이용하여 서버는 클라이언트와 상호 대화할 수 있다. 이전 단계에서의 portNumber에 연결된 소켓은 다른 클라이언트들의 연결을 위하여 그냥 두어야 한다.

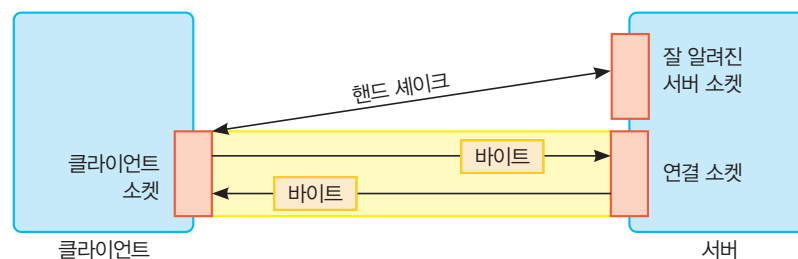


그림 17-18 통신을 위한 소켓들

(3) 소켓으로부터 스트림 객체를 얻는다.

서버가 클라이언트와 바이트를 주고받기 위해 `OutputStream`과 `InputStream` 객체를 얻는다. 서버는 `OutputStream`을 통하여 클라이언트에게 정보를 보낸다. 클라이언트로부터의 정보는 `InputStream`으로 얻는다. `Socket`의 `getOutputStream()`과 `getInputStream()` 메소드를 사용한다.

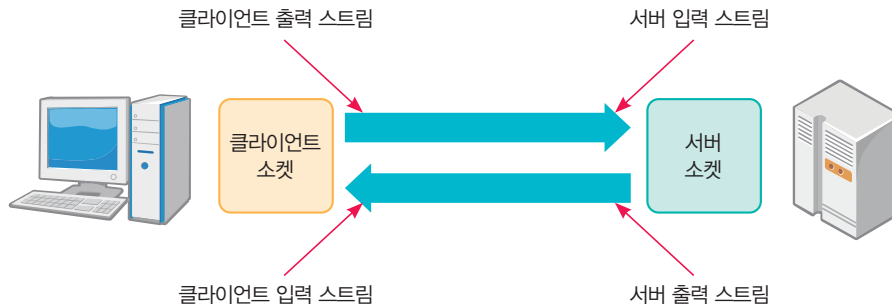


그림 17-19 입출력 스트림으로 바이트를 주고받는 클라이언트와 서버

```
InputStream input = clientSocket.getInputStream();
OutputStream output = clientSocket.getOutputStream();
```

`write()`와 `read()` 메소드를 사용하여 읽고 쓸 수 있다. 필요하다면 이 스트림을 다른 스트림으로 감쌀 수 있다. 예를 들어서 바이트 대신에 객체 단위로 정보를 주고받으려면 `ObjectStream`으로 감쌀 수 있다.

(4) 상호 대화 단계

서버와 클라이언트는 스트림을 이용하여서 상호 대화한다. 서버와 클라이언트 사이에는 미리 약속된 프로토콜이 있어야 한다.

(5) 종료

서버와 클라이언트 사이에 전송이 끝나면 서버가 `close()` 메소드를 호출하여서 스트림과 소켓을 닫는다.

채팅 서버 제작

2명의 사용자가 TCP/IP 통신을 이용하여 채팅을 할 수 있는 프로그램을 만들어 보자. 먼저 서버 프로그램부터 작성하자. 서버 프로그램은 먼저 특정한 포트에서 요청을 기다리기 위해 새로운 `ServerSocket`을 만드는 것으로 시작한다. 서버를 작성할 때는 다른 서비스가 사용하

지 않는 포트 번호를 선택하여야 한다. 여기서는 포트 번호 5000을 사용한다.

```
serverSocket = new ServerSocket(5000);
System.out.println("연결을 기다리고 있음");
```

ServerSocket은 클라이언트-서버 소켓 연결에서 서버 측의 구현을 제공하는 클래스이다. ServerSocket은 포트에 연결할 수 없으면 예외가 발생한다. 예를 들면 포트가 이미 사용되고 있는 경우에는 서버는 종료된다. 만약 서버가 성공적으로 포트에 연결되면 ServerSocket 객체는 생성되고 서버는 클라이언트로부터의 요청을 기다린다.

```
clientSocket = serverSocket.accept();
out = new PrintWriter(clientSocket.getOutputStream());
in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
System.out.println("클라이언트와 연결되었음");
```

accept() 메소드는 클라이언트가 시작되어 호스트에게 요청할 때까지 기다린다. 연결이 요청되고 성공적으로 접속되면 accept() 메소드는 새로운 포트와 연결된 Socket 객체를 반환한다. 서버는 클라이언트와 이 새로운 Socket을 통하여 통신할 수 있다. 또한 원래의 정해진 포트와 연결된 ServerSocket으로 계속 클라이언트 연결 요청을 기다릴 수 있다.

서버가 클라이언트와 성공적으로 연결을 설정한 다음에 서버는 클라이언트와 스트림을 이용하여 통신한다. 클라이언트로 보내는 첫 번째 메시지를 얻은 후에 사용자가 입력하는 메시지를 클라이언트로 보낸다. 클라이언트가 'quit' 메시지를 전송하면 통신이 종료된다. 서버 전체 소스 코드는 다음과 같다.

Server.java

```
1 import java.io.*;           // 입출력 관련 클래스
2 import java.net.*;          // 네트워크 관련 클래스
3 import java.util.Scanner;    // 사용자 입력을 처리하기 위한 Scanner 클래스
4
5 // 서버 프로그램: 클라이언트와 문자열을 주고받는 기능 제공
6 public class Server {
7     public static void main(String[] args) {
8         ServerSocket serverSocket = null;      // 서버 소켓 객체
9         Socket clientSocket = null;            // 클라이언트와의 연결 소켓
10        BufferedReader in = null;              // 클라이언트에서 받은 데이터를 읽는 입력 스트림
11        PrintWriter out = null;               // 클라이언트로 데이터를 보내는 출력 스트림
12        Scanner sc = new Scanner(System.in);   // 서버 사용자 입력 처리
13
14        try {
```

```

15 // 서버 소켓 생성(포트 번호 5000 사용)
16 serverSocket = new ServerSocket(5000);
17 System.out.println("연결을 기다리고 있음");
18
19 // 클라이언트의 연결 요청을 수락
20 clientSocket = serverSocket.accept();
21 System.out.println("클라이언트와 연결되었음");
22
23 // 클라이언트와 데이터를 주고받기 위한 스트림 생성
24 out = new PrintWriter(clientSocket.getOutputStream());
25 // 클라이언트로 데이터 출력
26 in = new BufferedReader(new InputStreamReader
27 (clientSocket.getInputStream())); // 클라이언트에서 데이터 입력
28
29 // 클라이언트와의 통신 처리
30 while (true) {
31     String msg = in.readLine(); // 클라이언트가 보낸 메시지를 읽음
32
33     // 클라이언트가 "quit" 명령을 보낼 경우 연결 종료
34     if (msg.equalsIgnoreCase("quit")) {
35         System.out.println("클라이언트에서 연결을 종료하였음");
36         break; // 반복문 종료
37     }
38
39     // 클라이언트로부터 받은 메시지 출력
40     System.out.println("클라이언트가 보낸 문자열: " + msg);
41
42     // 서버 사용자 입력 처리
43     System.out.print
44     ("클라이언트로 보낼 문자열을 입력하고 엔터키를 누르세요: ");
45     String omsg = sc.nextLine(); // 사용자 입력받기
46
47     // 클라이언트로 메시지 전송
48     out.write(omsg + "\n"); // 메시지 출력 스트림에 쓰기
49     out.flush(); // 출력 스트림 강제 플러시(전송)
50 }
51
52 // 리소스 정리: 출력 스트림, 소켓 닫기
53 out.close();
54 clientSocket.close();
55 serverSocket.close();
56 } catch (IOException e) {
57     // 입출력 예외 처리
58     e.printStackTrace();
59 }
60 }
61 }

```

채팅 클라이언트 제작

클라이언트 프로그램을 시작할 때 서버가 미리 수행되고 있어야 하며 미리 정해진 포트에서 클라이언트의 접속 요청을 기다리고 있어야 한다. 따라서 클라이언트 프로그램이 하는 첫 번째 작업은 호스트 이름과 포트를 가지고 수행되는 서버에 접속하는 것이다.

```
clientSocket = new Socket("localhost", 5000);
out = new PrintWriter(clientSocket.getOutputStream());
in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
```

소켓을 만들 때 `clientSocket`은 호스트 이름으로 'localhost'를 사용한다. 이것은 현재 프로그램이 실행되는 로컬 호스트의 이름이다. 만약 네트워크상의 특정 컴퓨터에서 서버가 실행되고 있다면 특정 컴퓨터의 도메인 이름이어야 한다. 포트 번호 5000은 서버 컴퓨터에 있는 원격 포트 번호이다. 바로 채팅 서버가 접속 요청을 기다리고 있는 포트이다. 클라이언트 소켓은 사용 가능한 지역 포트에 연결된다. 서버는 새로운 소켓을 얻는다. 이 소켓은 포트 번호인 5000에 연결된다. 서버의 소켓과 클라이언트의 소켓은 서로 연결된다.

다음으로 서버와 클라이언트 간의 통신을 구현하는 반복 루프가 있다. 클라이언트가 먼저 메시지를 서버로 보낸다. 클라이언트는 사용자로부터 메시지를 받아서 서버로 보낸다. 만약 사용자가 'quit'이라고 하면 클라이언트는 이것을 서버로 보낸 후에 루프를 빠져나온다. 그렇지 않으면 이것을 소켓에 연결된 출력 스트림을 통하여 서버로 보낸다.

```
while (true) {
    System.out.print("서버로 보낼 문자열을 입력하고 엔터키를 누르세요: ");
    msg = sc.nextLine();
    if (msg.equalsIgnoreCase("quit")) {
        out.println(msg);
        out.flush();
        break;
    }
    out.println(msg);
    out.flush();
    msg = in.readLine();
    System.out.println("서버로부터 온 메시지: " + msg);
}
```

이어서 서버로부터 온 메시지를 읽어서 화면에 표시한다. 전체 소스 코드는 다음과 같다.

Client.java

```

1  import java.io.*;           // 입출력 관련 클래스
2  import java.net.Socket;     // 소켓 관련 클래스
3  import java.util.Scanner;   // 사용자 입력을 처리하기 위한 Scanner 클래스
4
5  // 클라이언트 프로그램: 서버와 문자열을 주고받는 기능 제공
6  public class Client {
7      public static void main(String[] args) throws IOException {
8          Socket clientSocket = null; // 서버와 연결할 소켓 객체
9          BufferedReader in = null;   // 서버에서 받은 데이터를 읽는 입력 스트림
10         PrintWriter out = null;     // 서버로 데이터를 보내는 출력 스트림
11         final Scanner sc = new Scanner(System.in); // 사용자 입력 처리
12
13         try {
14             // 서버에 연결(localhost, 포트 번호 5000)
15             clientSocket = new Socket("localhost", 5000);
16
17             // 서버와 데이터를 주고받기 위한 스트림 생성
18             out = new PrintWriter(clientSocket.getOutputStream());
19             // 서버로 데이터 출력
20             in = new BufferedReader(new InputStreamReader
21                                     (clientSocket.getInputStream())); // 서버에서 데이터 입력
22
23             String msg;              // 메시지를 저장할 변수
24
25             // 서버와의 통신 처리
26             while (true) {
27                 // 사용자 입력 처리
28                 System.out.print("서버로 보낼 문자열을 입력하고 엔터키를 누르세요: ");
29                 msg = sc.nextLine(); // 사용자로부터 문자열 입력받기
30
31                 // "quit" 입력 시 연결 종료
32                 if (msg.equalsIgnoreCase("quit")) {
33                     out.println(msg); // 종료 메시지를 서버로 전송
34                     out.flush();      // 출력 스트림 플러시
35                     break;            // 반복문 종료
36                 }
37
38                 // 입력받은 메시지를 서버로 전송
39                 out.println(msg);     // 메시지 전송
40                 out.flush();          // 출력 스트림 플러시
41
42                 // 서버에서 응답 메시지를 읽어와 출력
43                 msg = in.readLine();  // 서버 응답 메시지 읽기
44                 System.out.println("서버로부터 온 메시지: " + msg);
45                 // 서버 응답 출력
46             }

```

```

47     } catch (IOException e) {
48         // 통신 과정에서 발생한 예외 처리
49         e.printStackTrace();
50     } finally {
51         // 리소스 정리: 스트림 및 소켓 닫기
52         out.close();                // 출력 스트림 닫기
53         clientSocket.close();        // 소켓 닫기
54     }
55 }
56 }

```

서버와 클라이언트 프로그램 실행

우리는 2개의 프로그램을 동시에 실행하여야 한다. 먼저 서버 프로그램을 실행한다.

```

C> java Server 
연결을 기다리고 있음

```

이어서 클라이언트 프로그램을 실행한다.

```

C> java Client 
서버로 보낼 문자열을 입력하고 엔터키를 누르세요:

```

클라이언트가 접속하는 순간 서버의 출력창에는 다음과 같은 메시지가 출력된다.

```

C> java Server 
연결을 기다리고 있음
클라이언트와 연결되었음

```

클라이언트 프로그램에서 서버로 보낼 메시지를 입력하고 엔터키를 누른다.

```

C> java Client 
서버로 보낼 문자열을 입력하고 엔터키를 누르세요: hello 

```

서버의 출력창에는 클라이언트가 보낸 메시지가 출력된다. 서버도 메시지를 입력한 후에 엔터키를 누른다.

```
C> java Server 
연결을 기다리고 있음
클라이언트와 연결되었음
클라이언트가 보낸 문자열: hello
클라이언트로 보낼 문자열을 입력하고 엔터키를 누르세요: hello too 
```

클라이언트 출력창에 서버로부터 온 메시지가 표시된다.

```
C> java Client 
서버로 보낼 문자열을 입력하고 엔터키를 누르세요: hello 
서버로부터 온 메시지: hello too
```

이런 식으로 채팅을 하다가 클라이언트가 'quit' 메시지를 보내면 연결이 종료된다.

```
C> java Client 
서버로 보낼 문자열을 입력하고 엔터키를 누르세요: hello
서버로부터 온 메시지: hello too
서버로 보낼 문자열을 입력하고 엔터키를 누르세요: quit
```

서버에는 다음과 같은 메시지가 표시된다.

```
C> java Server 
연결을 기다리고 있음
클라이언트와 연결되었음
클라이언트가 보낸 문자열: hello
클라이언트로 보낼 문자열을 입력하고 엔터키를 누르세요: hello too 
클라이언트에서 연결을 종료하였음
```

만약 인텔리제이를 사용한다면 하나의 패키지 안에 2개의 클래스를 입력하고 `Server.java`를 선택한 상태에서 실행 버튼을 누른 후에, 이어서 `Client.java`를 선택한 상태에서 실행 버튼을 누른다. 서버와 클라이언트의 콘솔을 교체하면서 메시지를 주고 받으면 된다.

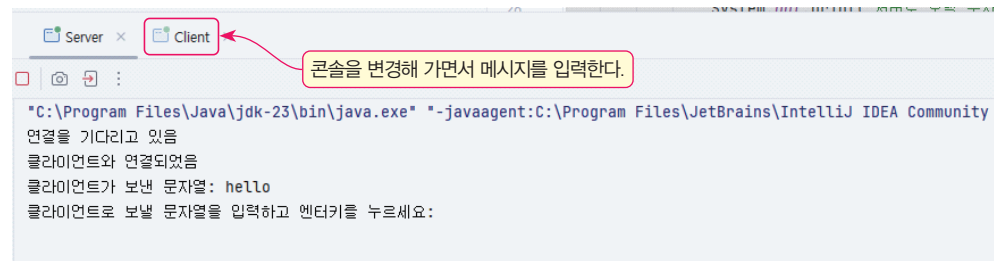


그림 17-20 서버와 클라이언트를 실행하는 방법

SELF TEST



- 1 서버와 클라이언트가 소켓으로 연결되는 과정을 설명하라.
- 2 클라이언트와 서버가 메시지를 주고받기 위해 사용하는 두 가지 주요 스트림 클래스는 무엇인가?
- 3 다음 중 클라이언트가 서버와 연결할 때 사용하는 클래스와 올바른 생성자 형태는 무엇인가?

① ServerSocket("localhost", 5000)	② Socket("localhost", 5000)
③ InputStream("localhost", 5000)	④ Client.connect("localhost", 5000)

17.5 UDP를 이용한 통신

TCP 프로토콜을 이용한 방식은 전화와 비슷하다. 통화를 하기 전에 먼저 전화번호를 눌러서 상대방 전화와 연결한다. 연결된 후에는 말을 하지 않더라도 연결은 유지된다.

UDP(User Datagram Protocol) 프로토콜을 이용한 방식은 편지와 비슷하다. 만약 하나의 봉투 안에 다 넣을 수 없다면 여러 개의 봉투를 이용할 수 있다. 똑같은 시간에 발송한 편지라고 하더라도 도착하는 시간이 다를 수 있다. 또한 편지들의 순서가 지켜지지 않는다. 최악의 경우에는 편지가 중간에 분실될 수도 있다.



그림 17-21 TCP 프로토콜과 UDP 프로토콜

UDP는 높은 신뢰도가 필요하지 않는 응용에 사용된다. 즉, 데이터가 중간에 분실될 수도 있고 보낸 순서와 도착 순서가 일치하지 않을 수도 있다. 각각의 데이터그램 패킷마다 주소를 가지고 있다. 데이터그램은 UDP 프로토콜을 구현하고 있다. UDP 프로토콜은 신뢰도를 신경 쓰지 않으므로 더 빠른 속도를 낼 수 있다. 또한 연결을 설정하지 않아도 되어 오버헤드가 적다. 자바에서는 UDP를 `DatagramPacket`와 `DatagramSocket` 클래스로 지원한다.

`DatagramSocket()` 메소드는 UDP 프로토콜을 사용하는 소켓을 생성한다. TCP 프로토콜 소켓과는 다르게 서버 소켓과 클라이언트 소켓의 구분이 없다. 그리고 `DatagramPacket` 객체만을 보내고 받을 수 있다. 모든 데이터는 `DatagramPacket` 객체 안에 포함된다. 목적지 주소와 포트 번호도 포함된다. TCP 프로토콜의 경우, 목적지 주소와 포트 번호는 소켓을 생성하면서 결정되었다. 다음과 같은 생성자를 가진다.

```
DatagramSocket(int port, InetAddress laddr)
```

`DatagramPacket`은 UDP 프로토콜을 사용하여 데이터를 보내기 위한 클래스이다. 두 가지 형태의 생성자가 사용된다. 하나는 수신 컴퓨터를 위한 형태로, 버퍼만 지정하면 된다. 다른 하나는 송신 컴퓨터를 위한 형태로, 상대방 주소와 포트 번호가 추가로 전달된다.

```
DatagramPacket(byte[] buf, int length, InetAddress address, int port)
```

UDP를 사용하여 데이터 보내고 받기

예제 17-3



문자열 1개를 UDP를 이용하여 보내고 받는 프로그램을 가지고 UDP 통신을 설명해 보자.

Sender.java

```
1 import java.io.*;           // 입출력 관련 클래스
2 import java.net.*;          // 네트워크 관련 클래스
3
4 // UDP를 사용하여 데이터를 보내는 프로그램
5 public class Sender {
6     public static void main(String[] args) throws IOException {
7         DatagramSocket socket = null;      // UDP 통신에 사용할 소켓 객체
8
9         // DatagramSocket 생성
10        socket = new DatagramSocket();      // 데이터를 전송하기 위한 소켓 생성
11
12        // 전송할 메시지
13        String s = "우리는 여전히 우리 운명의 주인이다."; // 전송할 데이터 문자열
14        byte[] buf = s.getBytes();         // 문자열을 바이트 배열로 변환
15    }
16 }
```

```

16 // 데이터를 전송할 대상의 IP 주소 및 포트 번호 설정
17 InetAddress address = InetAddress.getByName("127.0.0.1");
18 // 로컬 호스트 주소
19 DatagramPacket packet = new DatagramPacket
20 (buf, buf.length, address, 5000);
21 // 전송할 데이터를 담은 UDP 패킷 생성
22
23 // 패킷 전송
24 socket.send(packet); // 데이터를 지정된 주소와 포트로 전송
25
26 // 소켓 닫기
27 socket.close(); // 소켓 리소스 해제
28 }
29 }

```

UDP 패킷은 편지를 쓰는 것과 같은 방법으로 데이터를 보낸다. 우리가 편지를 보내려면 먼저 내용물을 봉투에 넣은 후에 봉투의 겉면에 받는 사람의 주소를 적는다. UDP 패킷도 마찬가지이다. 데이터를 가지고 있는 `DatagramPacket` 객체를 생성하고 여기에 수신 컴퓨터의 주소를 적는다. 이것을 `DatagramSocket` 객체의 `send()` 메소드를 이용하여 전송한다.

Receiver.java

```

1 import java.io.*; // 입출력 관련 클래스
2 import java.net.*; // 네트워크 관련 클래스
3
4 // UDP를 사용하여 데이터를 수신하는 프로그램
5 public class Receiver {
6     public static void main(String[] args) throws IOException {
7         // 수신할 데이터를 저장할 바이트 배열
8         byte[] buf = new byte[256]; // 최대 256바이트의 데이터를 수신 가능
9
10        // UDP 통신을 위한 소켓 생성
11        DatagramSocket socket = new DatagramSocket(5000);
12        // 포트 번호 5000에서 데이터를 수신하기 위해 소켓 생성
13
14        // 수신한 데이터를 저장할 패킷 객체 생성
15        DatagramPacket packet = new DatagramPacket(buf, buf.length);
16        // 데이터를 저장할 버퍼와 길이를 지정하여 패킷 생성
17
18        // 데이터 수신
19        socket.receive(packet);
20        // 데이터를 수신하고 패킷에 저장(이 동작은 블로킹됨: 데이터가 올 때까지 대기)
21
22        // 수신한 데이터를 문자열로 변환하여 출력
23        System.out.println(new String(buf));
24        // 수신된 바이트 배열을 문자열로 변환 후 콘솔에 출력
25    }

```

```
26 }
```

UDP 패킷을 수신하려면, `DatagramSocket` 객체와 `DatagramPacket` 객체를 생성한 후에 `DatagramSocket` 객체의 `receive()` 메소드를 실행한다. `receive()` 메소드에서 패킷이 도착할 때까지 기다린다.

위의 프로그램 중에서 먼저 `Receiver`를 실행한다. 이어서 `Sender`를 실행하면 다음과 같은 화면이 `Receiver`에서 나타난다.

우리는 여전히 우리 운명의 주인이다.

- 1 `Sender`가 사용자로부터 메시지를 직접 입력받아 보낼 수 있도록 확장하시오(Scanner 사용).
- 2 여러 번 메시지를 주고받을 수 있는 루프 형태로 개선하시오('quit'을 입력하면 종료).

도전문제



- 1 다음 중 UDP 통신에서 데이터를 전송하기 위해 사용하는 클래스는 무엇인가?
 - ① `Socket`
 - ② `ServerSocket`
 - ③ `DatagramSocket`
 - ④ `BufferedSocket`
- 2 `DatagramPacket` 객체를 생성할 때 반드시 지정해야 하는 두 가지 주요 정보는 무엇인가?
- 3 UDP 통신에서 `receive()` 메소드는 어떤 동작을 하며, 어떤 특징이 있는가?

SELF TEST



17.6 UDP를 이용한 채팅 프로그램 작성하기

UDP 통신을 이용하여 간단한 채팅을 할 수 있는 메신저를 만들어 보자. 이 메신저는 정해진 상대와 텍스트를 주고받을 수 있다.

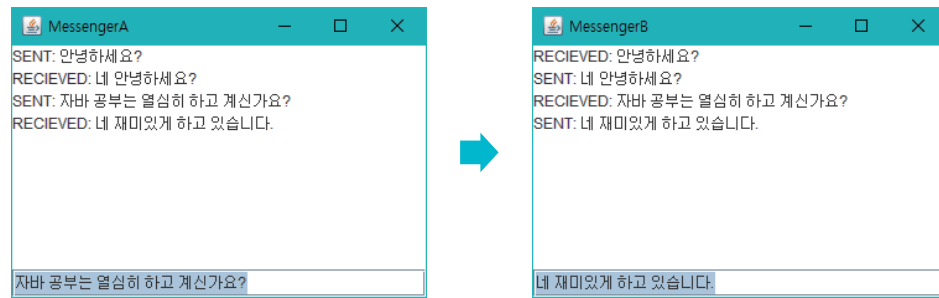


그림 17-22 UDP를 이용한 채팅 프로그램

이번에는 그래픽 사용자 인터페이스(GUI)를 사용해 보자. 텍스트 필드를 생성하여서 사용자가 메시지를 입력할 수 있게 한다. 텍스트 영역은 상대방 컴퓨터가 보내는 메시지를 표시하는 용도로 사용한다. 여기서는 송신용 포트 번호와 수신용 포트 번호가 5000번과 6000번으로 고정되어 있다. 각종 멤버 변수들을 쉽게 접근하기 위하여 프레임을 나타내는 클래스는 내부 클래스로 정의하였다.

MessengerA.java

```

1  import java.io.*;                // 입출력 관련 클래스
2  import java.net.*;               // 네트워크 관련 클래스
3  import java.awt.*;               // GUI 관련 클래스
4  import java.awt.event.*;         // 이벤트 리스너 관련 클래스
5  import javax.swing.*;            // Swing GUI 관련 클래스
6
7  // UDP를 이용한 메시지 송수신 프로그램(MessengerA)
8  public class MessengerA {
9      protected JTextField textField; // 메시지 입력을 위한 텍스트 필드
10     protected JTextArea textArea;   // 메시지를 표시할 텍스트 영역
11     DatagramSocket socket;           // UDP 소켓 객체
12     DatagramPacket packet;           // UDP 패킷 객체
13     InetAddress address = null;      // 대상 주소 저장
14     final int myPort = 5000;          // 수신용 포트 번호
15     final int otherPort = 6000;      // 송신용 포트 번호
16
17     // MessengerA 생성자
18     public MessengerA() throws IOException {
19         MyFrame f = new MyFrame(); // GUI 생성
20         address = InetAddress.getByName("127.0.0.1"); // 로컬 호스트 주소 설정
21         socket = new DatagramSocket(myPort);
22         // 지정된 포트에서 데이터 수신을 위한 UDP 소켓 생성
23     }
24
25     // 메시지를 수신하여 텍스트 영역에 표시하는 메소드

```

```

26 public void process() {
27     while (true) { // 무한 루프를 통해 지속적으로 메시지를 수신
28         try {
29             byte[] buf = new byte[256]; // 수신할 데이터를 저장할 버퍼
30             packet = new DatagramPacket(buf, buf.length); // 패킷 생성
31             socket.receive(packet); // UDP 패킷을 수신
32
33             // 받은 패킷의 내용을 텍스트 영역에 표시
34             textArea.append("RECEIVED: " + new String(buf) + "\n");
35         } catch (IOException ioException) {
36             ioException.printStackTrace(); // 예외 발생 시 스택 트레이스 출력
37         }
38     }
39 }
40
41 // GUI를 구현하는 내부 클래스
42 class MyFrame extends JFrame implements ActionListener {
43     // GUI 초기화
44     public MyFrame() {
45         super("MessengerA"); // 창 제목 설정
46         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 창 닫기 설정
47
48         textField = new JTextField(30); // 입력 필드 생성
49         textField.addActionListener(this); // 엔터 입력 시 이벤트 발생
50
51         textArea = new JTextArea(10, 30); // 텍스트 영역 생성(출력용)
52         textArea.setEditable(false); // 텍스트 영역 수정 불가능하게 설정
53
54         add(textField, BorderLayout.PAGE_END); // 텍스트 필드를 아래쪽에 배치
55         add(textArea, BorderLayout.CENTER); // 텍스트 영역을 중앙에 배치
56         pack(); // 창 크기 자동 조정
57         setVisible(true); // 창을 표시
58     }
59
60     // 사용자가 텍스트 입력 후 엔터를 눌렀을 때 실행되는 메소드
61     public void actionPerformed(ActionEvent evt) {
62         String s = textField.getText(); // 입력된 문자열 가져오기
63         byte[] buffer = s.getBytes(); // 문자열을 바이트 배열로 변환
64         DatagramPacket packet;
65
66         // 패킷 생성(대상 주소: address, 대상 포트: otherPort)
67         packet = new DatagramPacket(buffer, buffer.length, address,
68                                     otherPort);
69
70         try {
71             socket.send(packet); // 패킷을 전송

```

```

72         } catch (IOException e) {
73             e.printStackTrace();          // 예외 발생 시 스택 트레이스 출력
74         }
75
76         // 송신한 메시지를 텍스트 영역에 추가
77         textArea.append("SENT: " + s + "\n");
78         textField.selectAll(); // 입력 필드 초기화
79         textArea.setCaretPosition(textArea.getDocument().getLength());
80         // 스크롤을 가장 아래로 이동
81     }
82 }
83
84 // 메인 메소드: 프로그램 실행
85 public static void main(String[] args) throws IOException {
86     MessengerA m = new MessengerA(); // MessengerA 객체 생성 및 GUI 실행
87     m.process();                     // 메시지 수신을 위한 프로세스 실행
88 }
89 }

```

MessengerB.java

```

1 // 다음의 몇 개의 문장만 제외하고 MessengerA와 동일
2 public class MessengerB {
3     protected JTextField textField;
4     protected JTextArea textArea;
5     DatagramSocket socket;
6     DatagramPacket packet;
7     InetAddress address = null;
8     final int myPort = 6000;          // 수신용 포트 번호
9     final int otherPort = 5000;      // 송신용 포트 번호
10
11     public MessengerB() throws IOException {
12         MyFrame f = new MyFrame();
13         address = InetAddress.getByName("127.0.0.1");
14         socket = new DatagramSocket(myPort);
15     }
16     ...
17     // 내부 클래스 정의
18     class MyFrame extends JFrame implements ActionListener {
19
20         public MyFrame() {
21             super("MessengerB");
22             setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23             ...
24         }
25         public static void main(String[] args) throws IOException {
26             MessengerB m = new MessengerB();

```

```

27     m.process();
28 }
29 }

```

위의 코드 중에서 가장 중요한 코드는 다음과 같다. `InetAddress.getByName()` 메소드를 이용하여 호스트 이름에 대응되는 IP 주소를 알아낸다. 그리고 나의 컴퓨터가 사용하는 포트 번호로 `DatagramSocket` 객체를 생성한다. 이 객체를 이용하여 상대방 컴퓨터로부터 오는 패킷을 수신할 수 있다.

```

public MessengerA() throws IOException {
    address = InetAddress.getByName("127.0.0.1");
    socket = new DatagramSocket(myPort);
}

```

패킷 수신은 `socket.receive()` 메소드를 호출하면 된다. 수신된 패킷은 `packet` 객체에 저장된다.

```

byte[] buf = new byte[256];
packet = new DatagramPacket(buf, buf.length);
socket.receive(packet); // 패킷을 받는다.

```

사용자가 텍스트 필드에서 엔터키를 누르면, 이것은 액션 이벤트로 받아서 처리한다. 사용자가 입력한 내용을 `buffer`에 저장하여 `DatagramPacket` 객체에 넣어서 상대방 컴퓨터로 송신한다.

```

packet = new DatagramPacket(buffer, buffer.length, address, otherPort);
socket.send(packet); // 패킷을 보낸다.

```

- 1 UDP의 장점과 단점을 설명하시오.
- 2 UDP에서는 패킷을 받을 상대방을 어떻게 지정하는가?
- 3 `DatagramSocket` 클래스에서 패킷을 보내고 받는 메소드 이름은?

SELF TEST



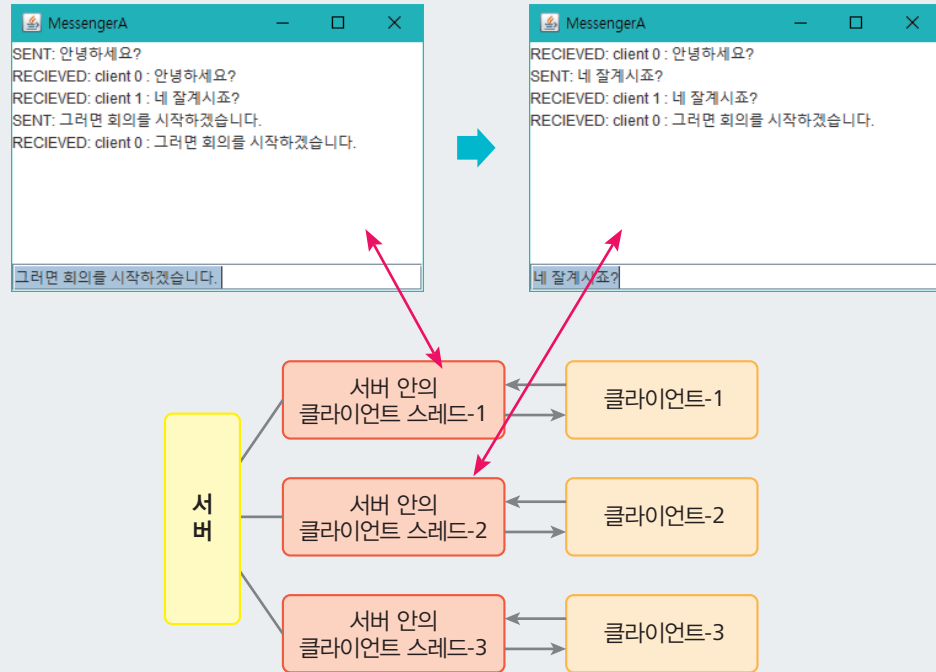
Mini Project 수행하기

다자 회의 시스템

난이도: ★★★★★

주제: TCP/IP 통신

코로나 바이러스가 유행했을 때 줌과 같은 인터넷 회의 시스템이 많이 사용되었다. 하나의 서버가 여러 클라이언트를 모아서 회의를 할 수 있는 프로그램을 작성해 보자.



서버에서는 ArrayList를 이용하여 클라이언트 정보를 저장한다. 그리고 클라이언트가 접속할 때마다 서비스 스레드를 생성하여 각 클라이언트를 서비스한다. 만약 어떤 클라이언트가 메시지를 보내면 이 메시지는 ArrayList에 저장된 모든 클라이언트에게 보내면 된다. 핵심적인 코드는 다음과 같다.

```
ArrayList<ServerThread> list = new ArrayList<>();
try (ServerSocket ssocket = new ServerSocket(5000)){
    while (true) {
        Socket socket = ssocket.accept();
        ServerThread thread = new ServerThread(socket, list);
        list.add(thread);
        thread.start();
    }
}
```


Summary

- 네트워크에는 서버와 클라이언트 컴퓨터가 존재한다.
- IP 주소는 네트워크에서 컴퓨터를 식별하는 숫자이다.

```
InetAddress ip = InetAddress.getLocalHost();
```

- URL(Uniform Resource Locator)은 인터넷상의 파일이나 데이터베이스 같은 자원에 대한 주소를 지정하는 방법이다.

```
URL url = new URL("http://example.com");
```

- 자바에서 InetAddress 클래스의 `getByName()` 메소드를 호출하면서 호스트 이름을 전달하면 IP 주소를 저장하고 있는 객체가 반환된다.

```
InetAddress addr = InetAddress.getByName("www.google.com");
```

- 웹에서 파일을 다운로드하려면 `java.net.URL` 클래스를 사용한다.

```
InputStream in = new URL("http://example.com/file.txt").openStream();
```

- TCP(Transmission Control Protocol)는 신뢰성 있게 통신하기 위하여 먼저 서로 간에 연결을 설정한 후에 데이터를 보내고 받는 방식이다.

```
Socket socket = new Socket("localhost", 5000);
```

- TCP를 사용하여 응용 프로그램끼리 통신하기 위해서는 먼저 연결을 해야 한다. 연결을 하기 위해서는 연결 끝점(end point)이 있어야 한다. 연결 끝점이 바로 소켓이다.

```
ServerSocket server = new ServerSocket(5000);
```

- 자바에서 Socket과 ServerSocket 클래스는 클라이언트 측과 서버 측을 구현할 때 사용된다. `accept()` 메소드는 클라이언트와 연결이 되면 새로운 Socket 객체를 생성하여 반환한다.

```
Socket client = server.accept();
```

- UDP(User Datagram Protocol) 프로토콜을 이용한 방식은 편지와 비슷하다. 자바에서는 UDP를 `DatagramPacket`과 `DatagramSocket` 클래스로 지원한다.

```
DatagramSocket ds = new DatagramSocket();
```

Exercises

1. 다음 중 네트워킹을 위한 클래스와 인터페이스가 포함된 패키지는?
 ① java.io ② java.util ③ java.net ④ java.network
2. 다음 중 현재 컴퓨터의 IP 주소를 얻기 위한 문장으로 올바른 것은?
 ① `InetAddress a = new InetAddress();`
 ② `InetAddress a = new InetAddress("localhost");`
 ③ `InetAddress a = InetAddress.getLocalHost();`
 ④ `InetAddress a = new InetAddress("local");`
3. 다음 클래스 중 웹 서버에 HTTP 요청을 보낼 때 주로 사용하는 것은?
 ① Socket ② ServerSocket
 ③ DatagramSocket ④ URLConnection
4. 다음 중 `getInputStream()` 메소드를 제공하지 않는 클래스는?
 ① Socket ② DatagramSocket
 ③ URLConnection ④ DatagramPacket
5. 다음 중 IP 주소와 DNS를 캡슐화할 때 사용되는 클래스는?
 ① DatagramPacket ② URL
 ③ InetAddress ④ URLNetwork
6. IPv4는 몇 개의 비트를 이용하는가?
 ① 8 ② 16 ③ 32 ④ 128
7. 다음 중 원격 클라이언트 프로그램과 접속하는 서버를 만들 때 사용되는 클래스는?
 ① ServerSocket ② Socket
 ③ httpResponse ④ DNSServer
8. 다음 중 UDP 통신 프로그램에 사용되는 클래스는?
 ① DatagramSocket ② DatagramPacket
 ③ ①과 ② 모두 ④ 없음

9. TCP 프로토콜과 UDP 프로토콜의 차이점은 무엇인가?
10. Socket 클래스와 ServerSocket 클래스의 차이점은 무엇인가?
11. ServerSocket 클래스의 accept() 메소드가 하는 일은 무엇인가?
12. 네트워크 서버 프로그램은 다중 스레드로 만들어진다. 이것이 의미하는 바를 설명하시오.

Programming Exercises

난이도: ★★★★★

주제: URL 클래스 사용

1. 웹에 있는 특정한 이미지 파일을 한정된 버퍼를 사용하여 다운로드하는 프로그램을 작성하시오. 버퍼의 크기는 2048바이트로 한다.

```
https://cdn.pixabay.com/photo/2014/05/02/21/49/
laptop-336373_960_720.jpg
사이트에서 이미지를 다운로드합니다.
151바이트만큼 읽었음!
1369바이트만큼 읽었음!
1369바이트만큼 읽었음!
...
```

Hint

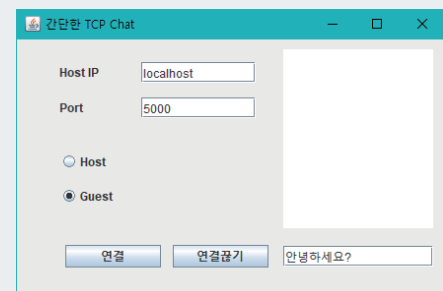
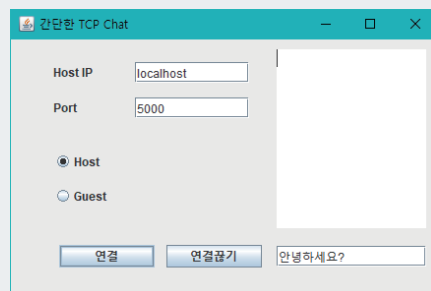


URL 클래스를 사용하여서 웹상의 특정한 이미지 파일에 연결한다. URL 클래스의 `openStream()` 메소드를 호출하여 입력 스트림을 얻는다.

난이도: ★★★★★

주제: TCP 채팅

2. 본문에 수록된 TCP 채팅 프로그램은 GUI를 사용하지 않는 버전이었다. 소스 코드를 참고하여 다음과 같은 GUI를 가진 프로그램으로 다시 제작하시오. 사용자는 포트 번호를 마음대로 선택할 수 있다. 우측 하단의 텍스트 필드에 텍스트를 입력하고 엔터키를 누르면 상대방 컴퓨터로 텍스트가 전달된다. 모든 대화는 오른쪽의 텍스트 영역에 표시된다.



Hint

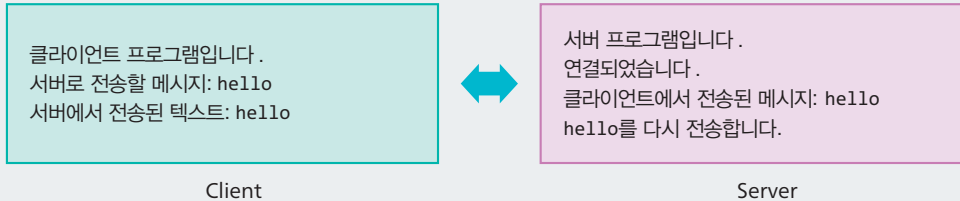


텍스트 필드에서 엔터키를 누르면 액션 이벤트가 발생한다. 액션 이벤트가 발생하면 `getText()` 메소드로 텍스트 필드 내용을 읽어서 상대방 컴퓨터로 전송한다.

3. TCP 소켓을 이용하여 간단한 서버를 제작한다. 서버는 EchoServer라고 불리고 클라이언트가 보내는 모든 문자열을 다시 보낸다. EchoServer와 EchoClient 클래스를 작성하여 먼저 EchoServer를 실행한 후 EchoClient를 실행한다. 서버와 클라이언트 사이에 주고받는 문자열을 화면에 표시한다. 다음의 두 가지 버전을 순차적으로 작성하시오.

난이도: ★★★★★☆

주제: TCP 서버

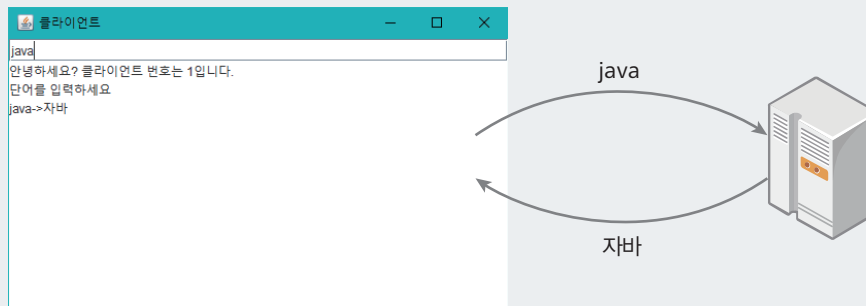


1. 콘솔 기반으로 작성한다.
2. 그래픽 기반으로 작성한다.

4. 사용자가 네트워크를 통하여 영어 단어를 보내면 한글로 번역해서 보내주는 서버를 제작하시오. 서버가 하나의 클라이언트만 처리하지 않고 동시에 여러 개의 클라이언트를 처리하기 위해서는 서버의 구조를 조금 변경하여야 한다. 다중 클라이언트를 지원하기 위해서는 각 클라이언트마다 스레드를 하나씩 생성하여 동시에 여러 클라이언트에게 서비스를 제공하는 서버를 만들 수 있다. ‘영어 단어 → 한글 번역’ 기능은 서버에 간단한 Map 자료 구조(사전)로 구현 가능하다.

난이도: ★★★★★☆

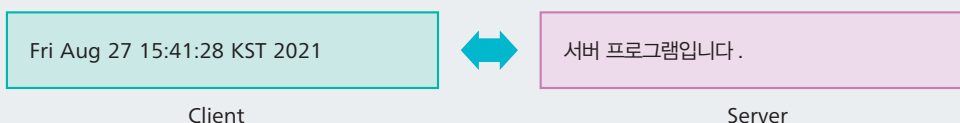
주제: TCP 서버



5. 접속하는 컴퓨터에게 현재의 날짜를 서비스하는 서버 DateServer를 제작하시오. 서비스를 제공하는 포트 번호는 9000번으로 한다. 클라이언트가 접속하면 현재의 날짜를 클라이언트로 보낸다. DateServer 프로그램을 실행한 상태에서 DateClient 프로그램을 실행한다. 만약 명령어로 실행시킨다면 다음과 같이 된다.

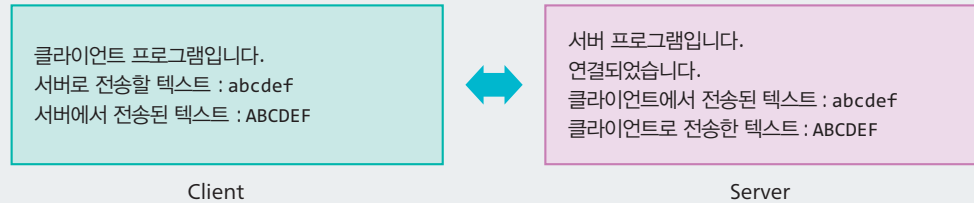
난이도: ★★★★★☆

주제: TCP 서버



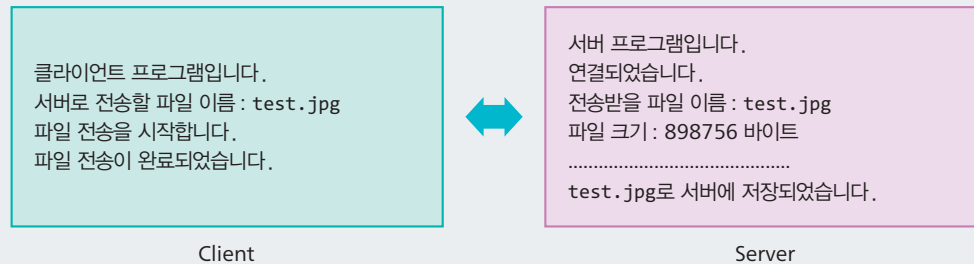
난이도: ★★★★★☆
주제: TCP 서버

6. 이번 프로그램에서 서버는 클라이언트로부터 텍스트를 받아서 이것을 전부 대문자로 만들어서 다시 보낸다. 효율적으로 여러 클라이언트를 처리하기 위하여 스레드를 사용한다. 서버는 동시에 다른 클라이언트를 수신하고 서비스할 수 있으므로 진정한 동시성이 있다.



난이도: ★★★★★☆
주제: 서버와 클라이언트

7. 간단한 파일 전송 프로그램을 작성하시오. 클라이언트 컴퓨터는 이진 파일을 서버로 전송할 수 있다. 아주 간단한 프로토콜을 사용한다. 즉, 클라이언트는 먼저 파일 이름(문자열)과 파일 크기(정수)를 보내고 이어서 이진 파일을 전송한다.

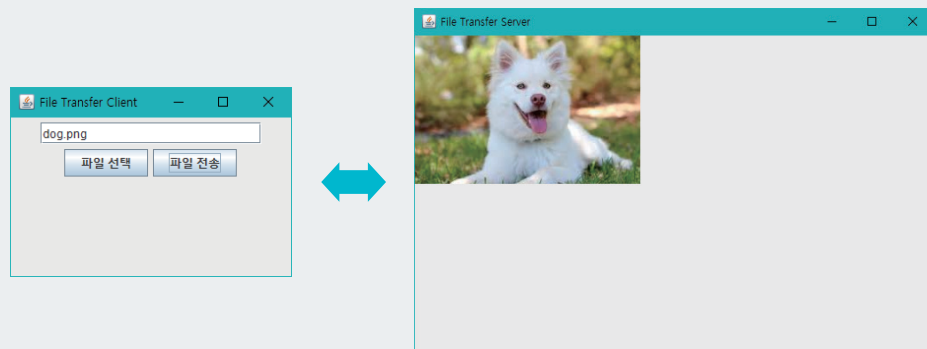


다음 코드를 참고한다.

```
pr.println(fileName);           // 파일 이름을 서버로 보낸다.
pr.println(FileSize);          // 파일 크기를 서버로 보낸다.
...
bis.read(filebyte, 0, filebyte.length); // 로컬 컴퓨터에서 파일을 읽는다.
os.write(filebyte, 0, filebyte.length);  // 서버로 이진 파일을 보낸다.
```

난이도: ★★★★★☆
주제: TCP 서버

8. 7번의 파일 전송 프로그램을 GUI 버전으로 제작하시오. 다음과 같은 사용자 인터페이스를 가진다. 서버가 이미지 파일을 받으면 화면에 표시한다.



”

CHAPTER

18

데이터베이스 프로그래밍



+ 학습목표

- JDBC를 사용하여 자바 애플리케이션을 MySQL과 같은 데이터베이스와 연결한다.
- JDBC를 사용하여 데이터베이스의 테이블을 출력한다.
- JDBC를 사용하여 데이터베이스에서 이미지를 저장하고 검색한다.
- JDBC를 사용하여 데이터베이스에서 파일을 저장하고 검색한다.

+ 학습목차

- 18.1 자바와 데이터베이스
- 18.2 데이터베이스의 기초
- 18.3 SQL
- 18.4 JDBC를 이용한 프로그래밍
- 18.5 Prepared Statements 사용하기
- 18.6 JDBC를 사용하여 이미지 저장하기
- 18.7 JDBC를 사용하여 텍스트 파일 저장하기

핵심 키워드

JDBC, SQL, Prepared Statements

데이터베이스 프로그래밍

18.1 자바와 데이터베이스

JDBC(Java Database Connectivity)는 자바 애플리케이션과 데이터베이스를 연결하는 라이브러리이다. JDBC를 사용하면 자바 프로그램에서 데이터베이스에 접근하여 검색이나 저장 등 여러 작업을 할 수 있게 된다.

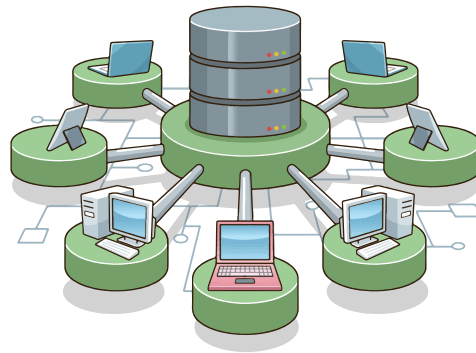
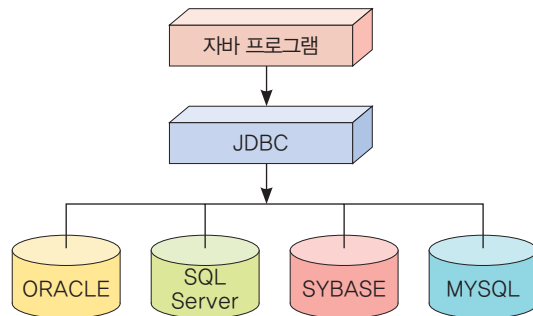


그림 18-1 데이터베이스

자바 개발진들은 초창기부터 자바로 데이터베이스에 접근할 수 있기를 원했다. 1995년에 표준 자바 라이브러리를 확장하여 데이터베이스에 접근하려 했다. 처음 목표는 순수하게 자바만을 사용하여서 어떤 데이터베이스라도 접근하게 하는 것이었으나, 시장에는 많은 종류의 데이터베이스가 있어서 이는 처음부터 불가능한 일이었다. 따라서 자바는 애플리케이션 개발자를 위해 JDBC API를 제공하고 데이터베이스 업체들을 위해 JDBC 드라이버 API를 제공해서, 많은 데이터베이스 업체가 그들의 데이터베이스를 위한 드라이버를 개발할 수 있도록 하였다. 업체들은 자신들의 드라이버를 쉽게 드라이버 관리자에 등록할 수 있었다. 애플리케이션은 드라이버 관리자에게 요청을 하고 드라이버 관리자는 드라이버를 통하여 데이터베이스에게 요청을 하게 된다.



자바에서 데이터베이스를
사용하려면 JDBC API를
사용하면 됩니다.



그림 18-2 자바와 데이터베이스

2계층 처리 모델과 3계층 처리 모델

JDBC API는 2계층(Two-tier)과 3계층(Three-tier) 처리 모델을 모두 지원한다.

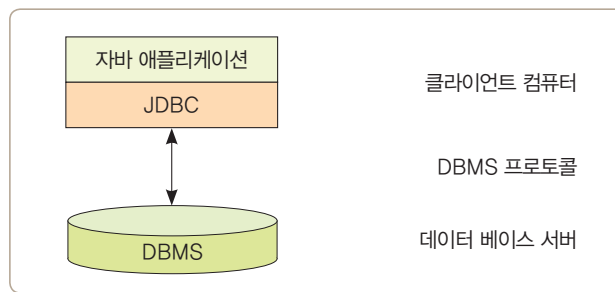


그림 18-3 2계층 처리 모델

2계층(Two-tier) 처리 모델에서는 자바 애플리케이션이 직접 데이터베이스 서버와 연결된다. 이 방법에서는 JDBC 드라이버가 데이터 소스와 직접 통신할 수 있어야 한다. 사용자의 SQL 명령어는 데이터베이스 서버로 전달되며, SQL 명령어의 결과가 사용자한테 보내진다. 이러한 모델은 클라이언트-서버 구성으로 언급되는데, 사용자의 컴퓨터가 클라이언트이고 데이터베이스가 있는 컴퓨터가 서버가 된다. 사용자 시스템 인터페이스는 일반적으로 사용자의 데스크톱 환경에 위치하며, 데이터베이스는 일반적으로 많은 클라이언트에 서비스를 제공하는 보다 강력한 시스템인 서버에 있다. 네트워크는 인트라넷 또는 인터넷이 될 수 있다. 데이터베이스 서버는 네트워크를 통하여 연결되는 다른 컴퓨터에 있을 수 있다. 2계층 처리 모델에서 애플리케이션 로직(코드)은 클라이언트의 사용자 인터페이스 내부 또는 서버의 데이터베이스에 묻혀 있다.

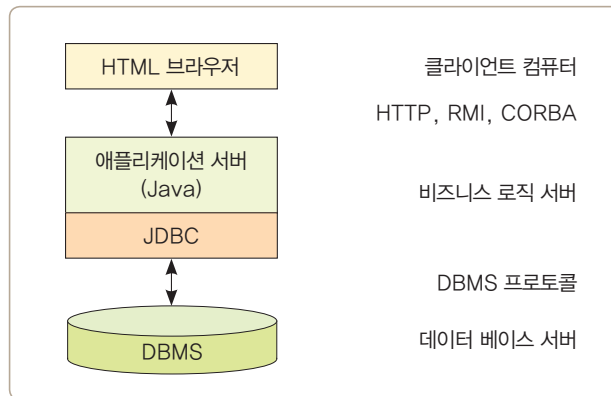


그림 18-4 3계층 처리 모델

3계층(Three-tier) 처리 모델에서 애플리케이션 로직은 중간 계층에 있으며 데이터 및 사용자 인터페이스와 분리된다. 3계층 처리 모델에서는 사용자 인터페이스와 데이터베이스 서버 사이에 중간 계층이 추가되었다. 트랜잭션 처리 모니터, 메시지 서버, 응용 프로그램 서버와 같이 중간 계층을 구현하는 다양한 방법이 있다. 3계층 처리 모델은 확장 가능하고 강력하며 유연하다. 또한 여러 데이터 소스를 통합할 수 있다. 회사에서는 이러한 3계층 처리 모델을 좋아하는데, 주된 이유는 중간 계층을 이용하여 사용자의 접근과 비즈니스 데이터 업데이트를 제어할 수 있기 때문이다. 3계층 처리 모델은 주로 웹 기반 시스템이 된다.

최근까지 중간 계층은 속도를 위해 주로 C나 C++를 사용하여 작성되었다. 하지만 자바 바이트 코드를 최적화하는 기술이 발전하였고 엔터프라이즈 자바 빈즈와 같은 기술이 개발되면서 자바 플랫폼은 중간 계층을 위한 표준 플랫폼이 되고 있다. 자바의 견고성, 멀티스레딩, 보안 기능 등이 인정받고 있는 것이다. 회사들이 애플리케이션 로직을 작성할 때 자바를 점점 많이 사용함에 따라 JDBC API도 중간 계층에서 많이 사용되고 있다. JDBC의 장점이라면 연결 풀 기능(connection pooling), 분산 트랜잭션 기능(distributed transactions)을 들 수 있다.

데이터베이스 프로그램 개발 절차

자바를 이용하여 데이터베이스 응용 프로그램을 개발하려면 몇 가지의 절차를 거쳐야 한다.

1. DBMS(DataBase Management System)를 설치하여야 한다. 데이터베이스는 간단하게는 마이크로소프트 액세스를 통하여 작성할 수도 있고 MySQL이나 오라클과 같은 전문적인 DBMS를 설치할 수도 있다. 우리는 MySQL을 설치하여 사용하도록 하자.
2. 자신이 설치한 DBMS에 필요한 JDBC 드라이버를 설치한다. 이 드라이버는 DBMS 회사에서 제공한다. MySQL을 사용한다면, 'Connector/J'가 바로 우리에게 필요한 JDBC 드라이버이다. 최근에는 MySQL을 설치할 때 'Connector/J'도 함께 설치된다.

3. JDBC가 제공하는 기능을 이용하여 데이터베이스 응용 프로그램을 개발한다. 자바의 JDBC API가 하는 일은 자바 프로그램에서 SQL 명령어들을 데이터베이스 관리 시스템으로 보낼 수 있도록 자바와 데이터베이스를 연결하는 것이다.

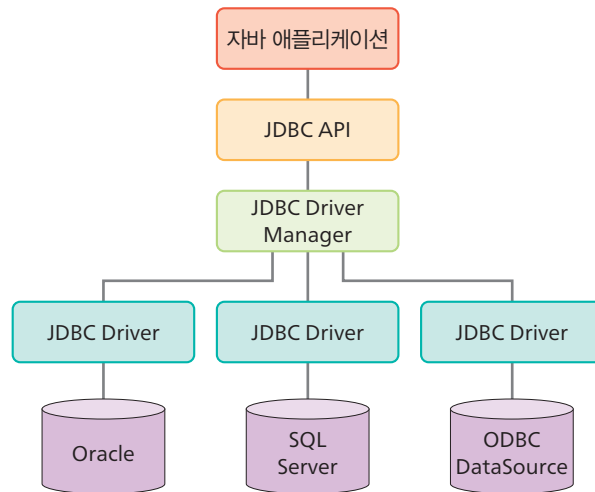


그림 18-5 데이터베이스 프로그램의 계층 구조

- 1 자바 프로그램에서 데이터베이스를 사용하려면 어떤 API가 필요한가?
- 2 2계층(Two-tier)과 3계층(Three-tier) 처리 모델을 비교하여 설명하시오.

SELF TEST



18.2 데이터베이스의 기초

이번 절에서는 데이터베이스의 기초 지식에 대하여 살펴본다. 만약 데이터베이스를 잘 알고 있는 학습자라면 생략할 수 있다.

데이터베이스란?

데이터베이스는 데이터가 빠르게 추출될 수 있도록 데이터를 조직화하여서 저장하는 방법이다. 일반적으로 열과 행으로 이루어진 테이블에 데이터를 저장한다. 데이터베이스 관리 시스템(DBMS)은 다수의 사용자를 위하여 데이터를 저장, 접근, 변경하는 기능을 제공한다.

현재 가장 인기 있는 데이터베이스 시스템은 관계형 데이터베이스(relational database)이다. 관계형 데이터베이스에서는 여러 개의 테이블이 존재하고 테이블과 테이블 간에는 공통

데이터로 인하여 어떤 관계가 성립될 수 있다. 예를 들어서 고객들의 테이블과 주문서 테이블은 분명히 공통으로 고객들의 정보를 포함하고 있을 것이다.

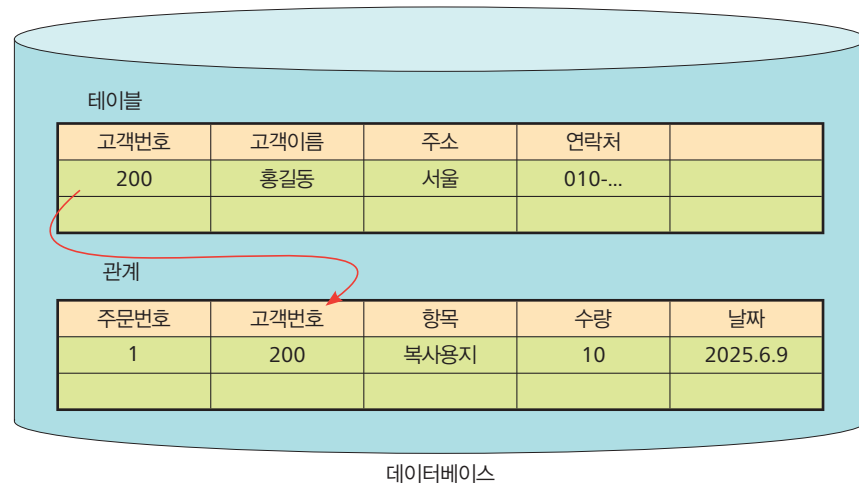


그림 18-6 데이터베이스

가장 많이 사용되는 DBMS로는 오라클, 마이크로소프트의 SQL Server, MySQL, MongoDB 등을 들 수 있다. 이 중에서 특히 MySQL은 오픈 소스의 일종이라서 누구든지 무료로 사용할 수 있다. 관계형 데이터베이스는 SQL이라고 불리는 언어로 사용할 수 있다. SQL은 데이터베이스를 조작하는 국제적인 표준 언어이다.

테이블

테이블의 하나의 행(row)은 레코드(record)라고 불린다. 레코드는 여러 개의 컬럼(column)으로 이루어져 있다. 테이블은 무결성 법칙에 따라 작성되어야 한다. 무결성 법칙 중 하나가 테이블에서 각 레코드는 중복되지 않아야 한다는 것이다.

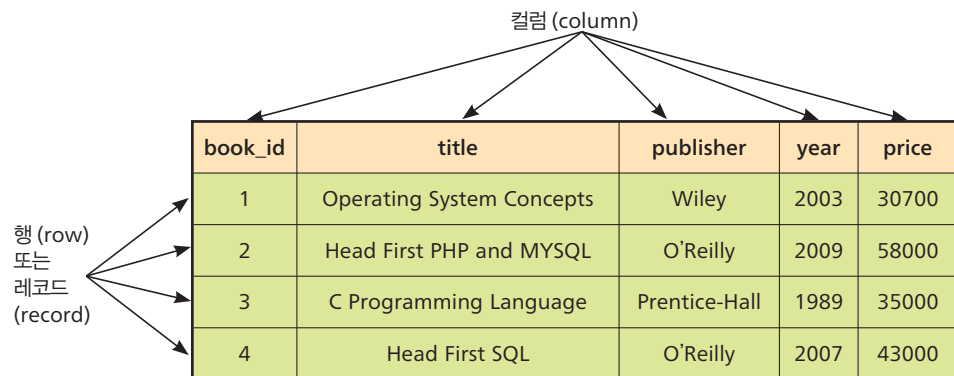


그림 18-7 데이터베이스 테이블

중복이 있으면 어떤 레코드가 올바른 것인지를 알기 어렵다. 대부분 DBMS에서 사용자는 중복된 레코드가 허용되지 않도록 하나의 컬럼을 이용한다. 이러한 특정한 컬럼을 기본키(primary key)라고 한다. 예를 들어서 학생 데이터에서는 학번이 기본키가 될 것이다. 기본키는 NULL이 되면 안 된다. 그림 18-7은 책에 대한 데이터를 테이블로 정리한 것이다.

이 테이블에서 기본키는 book_id이다. book_id는 레코드가 추가되면 1씩 증가하는 값이다. 따라서 동일할 수가 없다. 책 이름은 기본키로 사용할 수 없다. 왜냐하면 책 이름은 동일할 수가 있기 때문이다. 그리고 레코드와 레코드를 비교할 때는 숫자가 문자보다 더 효율적이다.

MySQL

데이터베이스에 대하여 여러 가지 프로그래밍을 하려면 데이터베이스 시스템이 설치되어 있어야 한다. 무료로 설치할 수 있는 데이터베이스는 MySQL이다. MySQL은 www.mysql.com에서 다운로드할 수 있다. MySQL 웹 사이트에서 [Downloads] 메뉴 선택하여 무료 버전인 MySQL Community Edition (GPL)을 다운로드한다. 상용 버전은 아주 큰 글씨로 잘 보이게 적혀 있지만, 무료 버전인 Community Edition은 화면 맨 아래쪽에 아주 작은 글씨로 쓰여 있으니 잘 찾아야 한다. 'MySQL Community (GPL) Downloads'를 선택한다.

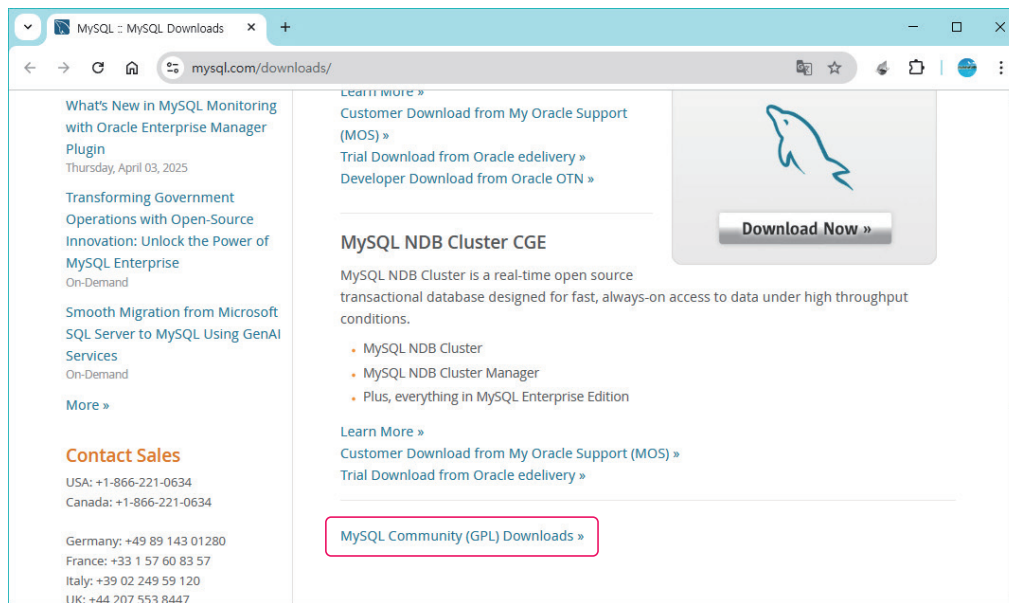


그림 18-8 MySQL 다운로드 페이지 1

이어서 다음 화면에서 'MySQL Community Server'를 선택한다.

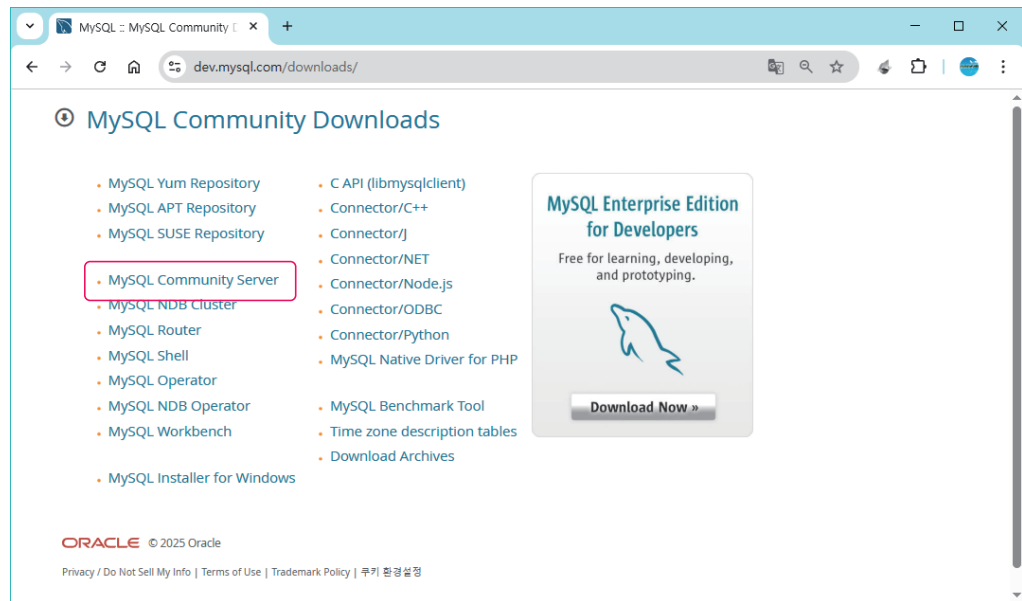


그림 18-9 MySQL 다운로드 페이지 2

다음 화면에서 'Windows (x86, 32-bit), MSI Installer'를 선택한다.

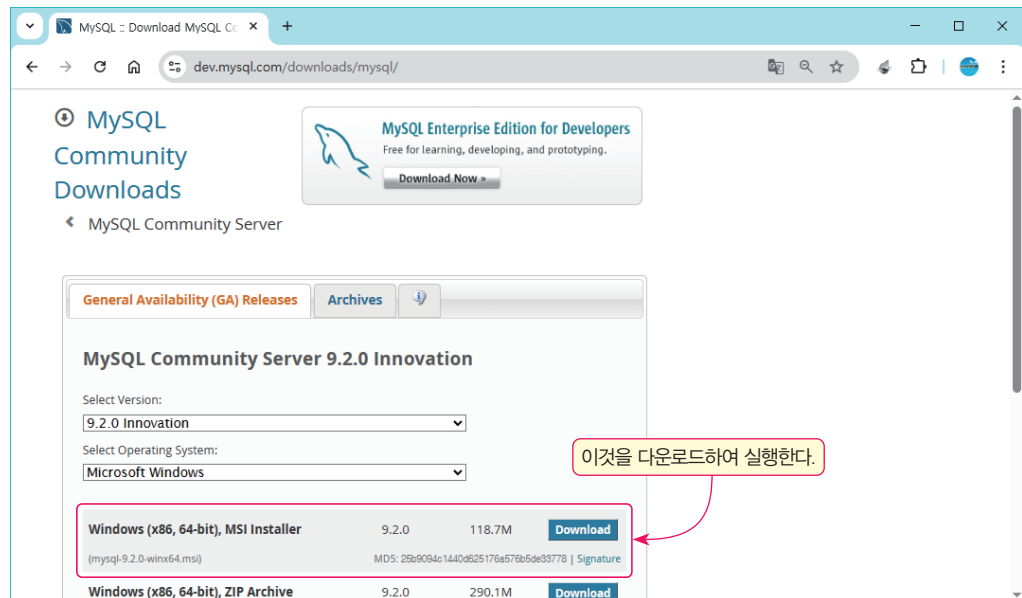


그림 18-10 MySQL 다운로드 페이지 3

다음 페이지에서 오라클 아이디로 로그인하라고 나오는데 화면의 맨 아래쪽에 보면 'No thanks, just start my download.' 문구가 있다. 이것을 누르면 로그인을 피해갈 수 있다.

다운로드한 파일을 실행하면 설치가 시작된다. 설치 시 물어보는 사항들은 대부분 기본 옵션을 선택하면 된다. 설치되면서 다음 화면이 등장하는데, 이 중 하나가 실패로 나오면 다시 설치하여야 한다. 만약 개인 방화벽을 사용하고 있는 경우에 다음과 같이 TCP 포트 3306에 대한 접근을 허용하여야 한다.

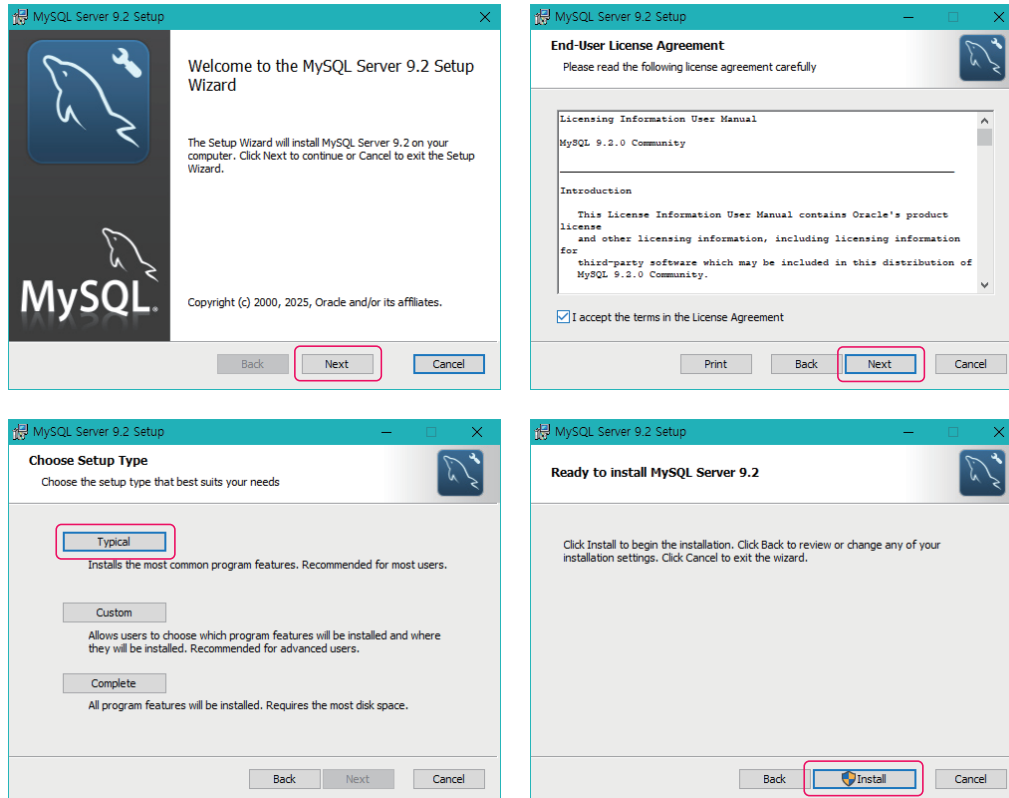
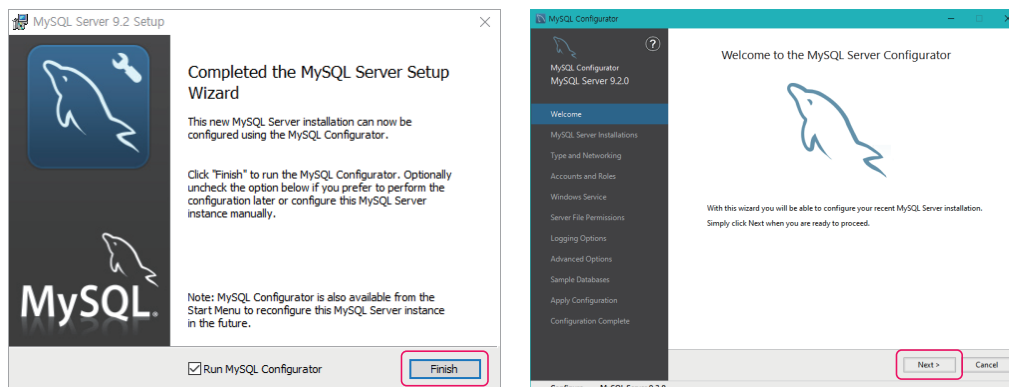


그림 18-11 MySQL 설치 화면 1

다음 화면에서 MySQL 관리자 패스워드를 설정하여야 한다. 패스워드는 반드시 기억하고 있어야 한다.



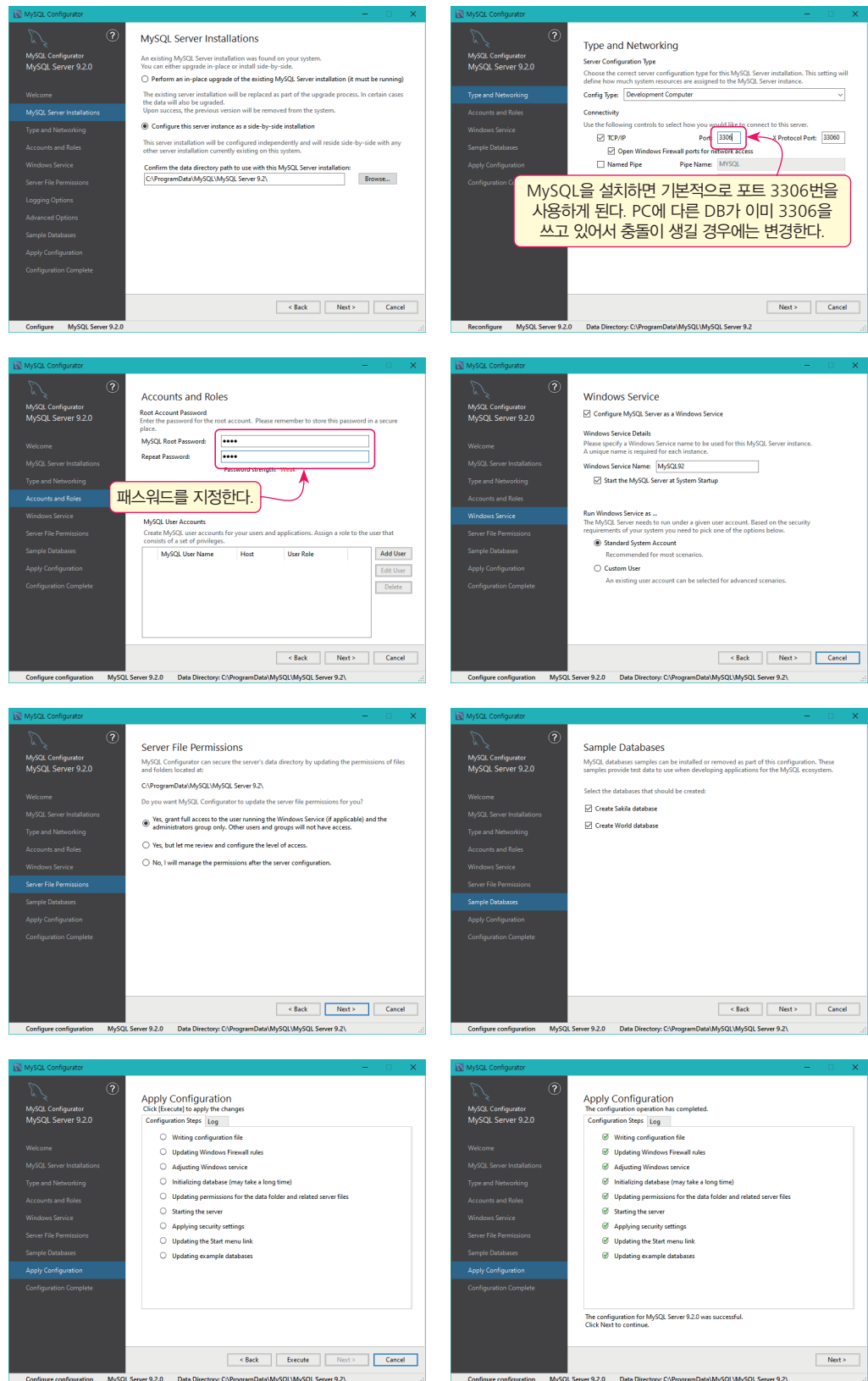


그림 18-12 MySQL 설치 화면 2

마지막 대화상자에서 [Finish] 버튼을 누르면 MySQL 서버가 시작된다. 이제부터는 MySQL을 사용할 수 있다. MySQL은 다음과 같은 명령어 행 클라이언트를 가지고 있다. 앞으로 우리는 이것을 이용하여서 여러 가지 작업을 할 것이다. 윈도우의 [시작] 버튼을 누르고 [MySQL 9.2 Command Line Client]를 찾아서 실행해 보자.

```

MySQL 9.2 Command Line Client
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 9.2.0 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or 'h' for help. Type 'c' to clear the current input statement.

mysql>

```

그림 18-13 MySQL 실행 화면

이 상태에서 여러 가지 SQL 명령어를 넣어서 실습할 수 있다.

- 1 데이터베이스 테이블의 하나의 행은 무엇이라고 불리는가?
- 2 무결성이란 무엇인가?
- 3 기본키는 왜 필요한가?

SELF TEST



18.3 SQL

우리는 데이터베이스 프로그래밍을 시작하기 전에 데이터베이스를 생성하여야 한다. SQL을 사용하여 book_db 데이터베이스를 생성해 보자.

title	publisher	price
Operating System Concepts	Wiley	30700
Head First PHP and MYSQL	OReilly	58000
C Programming Language	Prentice-Hall	35000
Head First SQL	OReilly	43000

SQL이란?

SQL은 SQL 관계형 데이터베이스에서 사용하기 위하여 설계된 언어이다. 표준 SQL 명령어들이 있으며, 이것은 모든 관계형 데이터베이스에 의하여 지원된다. SQL 명령어들은 두 가지의 카테고리로 나누어진다. 데이터 정의 명령어(Data Definition Language)는 테이블을 생성하거나 변경한다. 데이터 조작 명령어(Data Manipulation Language)는 데이터를 추출, 추가, 삭제, 수정한다. 많이 사용되는 SQL 명령어를 정리하면 다음과 같다.

구분	명령어	설명
데이터 정의 명령어 (Data Definition Language)	CREATE	사용자가 제공하는 컬럼 이름으로 테이블을 생성한다. 사용자는 컬럼의 데이터 타입도 지정하여야 한다. 데이터 타입은 데이터베이스에 따라 달라진다. CREATE TABLE은 보통 DML보다 적게 사용된다. 왜냐하면 이미 테이블이 만들어져 있는 경우가 많기 때문이다.
	ALTER	테이블에서 컬럼을 추가하거나 삭제한다.
	DROP	테이블의 모든 레코드를 제거하고 테이블의 정의 자체를 데이터베이스로부터 삭제한다.
	USE	어떤 데이터베이스를 사용하는지를 지정한다.
데이터 조작 명령어 (Data Manipulation Language)	SELECT	데이터베이스로부터 데이터를 쿼리하고 출력한다. SELECT 명령어들은 결과 집합에 포함할 컬럼을 지정한다. SQL 명령어 중에서 가장 자주 사용된다.
	INSERT	새로운 레코드를 테이블에 추가한다. INSERT는 새롭게 생성된 테이블을 채우거나 새로운 레코드를 이미 존재하는 테이블에 추가할 때 사용된다.
	DELETE	지정된 레코드를 테이블로부터 삭제한다.
	UPDATE	테이블에서 레코드에 존재하는 값을 변경한다.

표 18-1 자주 사용하는 SQL 명령어

데이터베이스 생성하기

데이터베이스에 데이터를 저장하기 전에 당연히 데이터베이스부터 생성하여야 한다. 여기서는 MySQL의 명령어인 CREATE를 사용하여 데이터베이스를 생성해 보자. CREATE는 다음과 같은 형식을 가진다.

Syntax: CREATE

```
CREATE TABLE 테이블명 (컬럼명1 자료형1, 컬럼명2 자료형2, ...);
```

책에 대한 데이터베이스와 테이블을 생성하기 위하여 [MySQL 9.2 Command Line Client]에서 다음과 같은 명령어를 입력하여 실행해 보자.

```

DROP DATABASE book_db;

CREATE DATABASE book_db;

USE book_db;

CREATE TABLE books (
    book_id INT NOT NULL auto_increment,
    title VARCHAR(50),
    publisher VARCHAR(30),
    year VARCHAR(10),
    price INT,
    PRIMARY KEY(book_id)
);

```

`book_id` 컬럼의 데이터 타입은 `INT`이다. 이 컬럼은 `NULL`이 되면 안 된다. 또 이 컬럼의 값은 레코드가 추가될 때마다 자동으로 증가한다. `book_id` 컬럼이 기본키(primary key)가 된다. 기본키는 테이블에서 각각의 레코드를 구별할 수 있는 유일한 값이다. 모든 테이블은 기본 키를 가져야 한다.

레코드 추가하기

데이터베이스를 생성하였으면 다음 작업은 레코드를 추가하는 것이다. 레코드 추가는 `INSERT` 명령어를 사용한다. `INSERT` 문장의 형식은 먼저 데이터를 삽입하고 싶은 컬럼들을 나열하고 실제의 데이터를 이어서 나열하면 된다.

Syntax: INSERT

```
INSERT INTO 테이블명 [(컬럼명1, 컬럼명2, ...)] VALUES (값1, 값2, ...);
```

다음과 같은 문장들을 입력하여 테이블에 몇 개의 레코드를 추가해 본다.

```
mysql> USE book_db;

INSERT INTO books (title, publisher, year, price)
VALUES('Operating System Concepts', 'Wiley', '2003', 30700);

INSERT INTO books (title, publisher, year, price)
VALUES('Head First PHP and MYSQL', 'OReilly', '2009', 58000);

INSERT INTO books (title, publisher, year, price)
VALUES('C Programming Language ', 'Prentice-Hall', '1989', 35000);

INSERT INTO books (title, publisher, year, price)
VALUES('Head First SQL', 'OReilly', '2007', 43000);
```

레코드 검색하기

SELECT 문장은 쿼리(query)라고도 불리는데 테이블에서 정보를 얻어내기 위하여 사용된다. 이 명령어는 출력하고 싶은 컬럼과 테이블을 지정한다.

Syntax: SELECT

```
SELECT 컬럼명 FROM 테이블명 [ WHERE 조건 ] [ORDER BY 정렬 방식]
```

다음 SELECT 명령어는 books라는 테이블에서 title, publisher, price 컬럼을 출력한다. FROM 절은 테이블을 지정한다.

```
mysql> USE book_db;
mysql> SELECT title, publisher, price FROM books;
+-----+-----+-----+
| title                | publisher    | price |
+-----+-----+-----+
| Operating System Concepts | Wiley        | 30700 |
| Head First PHP and MYSQL  | OReilly      | 58000 |
| C Programming Language    | Prentice-Hall | 35000 |
| Head First SQL            | OReilly      | 43000 |
+-----+-----+-----+
```

SELECT 명령어의 결과로 나오는 레코드들의 집합을 결과 집합(result set)이라고 한다. 다음의 코드는 테이블 안의 모든 레코드를 포함하는 결과 집합을 생성한다. 왜냐하면 모든 컬럼을 요청하고 있고 조건이 없기 때문이다. 여기서 SELECT *는 모든 컬럼을 선택함을 의미한다.

```
mysql> SELECT * from books;
```

book_id	title	publisher	year	price
1	Operating System Concepts	Wiley	2003	30700
2	Head First PHP and MYSQL	OReilly	2009	58000
3	C Programming Language	Prentice-Hall	1989	35000
4	Head First SQL	OReilly	2007	43000

SELECT를 사용할 때 선택을 위한 조건을 명시할 수 있다. 이런 경우에는 조건을 만족하는 레코드들이 결과 집합이 된다. WHERE 절은 레코드를 선택하는 기준을 제공한다. 예를 들어서 title이 'Head First'로 시작되는 레코드만 선택하려면 다음과 같이 한다.

```
mysql> SELECT * FROM books WHERE title LIKE 'Head First%';
```

book_id	title	publisher	year	price
2	Head First PHP and MYSQL	OReilly	2009	58000
4	Head First SQL	OReilly	2007	43000

키워드 LIKE는 문자열을 비교할 때 사용된다. 이것은 와일드카드(wildcard)를 가지고 있는 패턴을 사용할 수 있다. 'Head First%'에서 퍼센트(%) 기호는 문자열 'Head First'에 0개 이상의 문자를 더하는 것을 의미한다. 따라서 'Head First PHP' 또는 'Head First SQL'이 매칭이 된다. 그러나 'Head'는 안 된다. 다른 와일드카드로는 밑줄(_)이 있다. 이것은 하나의 글자 대신 사용할 수 있다.

WHERE 절에서 등호와 부등호를 사용하면 숫자를 비교할 수 있다. 아래의 문장은 가격이 30,000원보다는 높고 50,000원보다는 낮은 책을 선택한다.

```
mysql> SELECT * FROM books WHERE price > 30000 and price < 50000;
```

book_id	title	publisher	year	price
---------	-------	-----------	------	-------

1	Operating System Concepts	Wiley	2003	30700	
3	C Programming Language	Prentice-Hall	1989	35000	
4	Head First SQL	OReilly	2007	43000	

레코드들을 정렬하여 출력하려면 `ORDER BY`를 사용한다. 다음의 문장은 모든 책을 발행 연도로 정렬하여 출력한다.

```
mysql> SELECT * FROM books ORDER BY year;
```

book_id	title	publisher	year	price
3	C Programming Language	Prentice-Hall	1989	35000
1	Operating System Concepts	Wiley	2003	30700
4	Head First SQL	OReilly	2007	43000
2	Head First PHP and MYSQL	OReilly	2009	58000

4 rows in set (0.03 sec)

레코드 수정하기

SQL의 `UPDATE` 명령어를 사용하면 레코드를 변경할 수 있다. `UPDATE` 명령어는 다음과 같은 구문을 가진다. 만약 조건을 주지 않으면 모든 레코드가 변경된다.

Syntax: `UPDATE`

```
UPDATE 테이블명 SET 컬럼명 = 새로운 값, ... [WHERE 조건];
```

아래의 SQL 문장은 만약 발행 연도가 19XX이면 가격을 30,000원으로 변경한다.

```
mysql> USE book_db;

mysql> UPDATE books SET price = 30000 WHERE year LIKE '19%';
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

```
mysql> SELECT * FROM books;
```

book_id	title	publisher	year	price
1	Operating System Concepts	Wiley	2003	30700
2	Head First PHP and MYSQL	OReilly	2009	58000
3	C Programming Language	Prentice-Hall	1989	30000
4	Head First SQL	OReilly	2007	43000

레코드 삭제하기

SQL의 DELETE 명령어를 사용하면 현재의 레코드를 삭제할 수 있다. DELETE 명령어는 다음과 같은 형식을 가진다.

Syntax: DELETE

```
DELETE FROM 테이블명 [WHERE 조건];
```

아래의 SQL 문장은 발행 연도가 19XX인 레코드를 삭제한다.

```
mysql> USE book_db;
```

```
mysql> DELETE FROM books WHERE year LIKE '19%';
Query OK, 1 row affected (0.03 sec)
```

```
mysql> select * from books;
```

book_id	title	publisher	year	price
1	Operating System Concepts	Wiley	2003	30700
2	Head First PHP and MYSQL	OReilly	2009	58000
4	Head First SQL	OReilly	2007	43000

결과 집합과 커서

쿼리의 조건을 만족하는 레코드들의 집합이 **결과 집합(result set)**이다. 결과 집합에서 사용자는 커서를 사용하여 한 번에 한 레코드씩 데이터에 접근할 수 있다. 커서(cursor)는 결과 집합의 레코드들을 포함하고 있는 파일에 대한 포인터라고 간주할 수 있다. 이 포인터는 현재 접근하고 있는 레코드들을 가리킨다. 커서는 사용자가 결과 집합에서 각각의 레코드들을 처리할 수 있도록 도와준다. 커서는 레코드들에 대하여 반복 처리를 할 때 사용된다. 대부분의 DBMS는 결과 집합이 생성될 때 커서가 자동으로 만들어진다. 커서는 정방향이나 역방향으로 움직일 수 있다. 따라서 특정한 레코드로 이동할 수 있다.

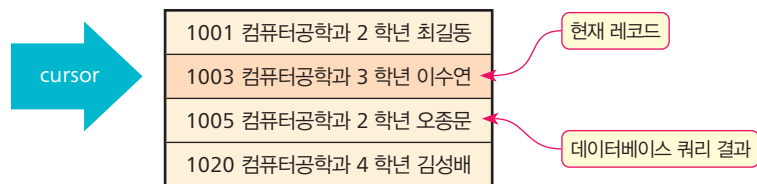


그림 18-14 결과 집합과 커서

SELF TEST



- 1 SQL 명령어들은 어떻게 분류할 수 있는가?
- 2 SQL 명령어 중에서 테이블을 생성하는 명령어는?
- 3 SQL 명령어 중에서 특정 조건을 주어 쿼리하는 명령어는?
- 4 쿼리의 결과로 나온 레코드들의 집합을 무엇이라고 하는가?

18.4 JDBC를 이용한 프로그래밍

MySQL JDBC 드라이버 Connector/J

자바와 데이터베이스를 연결하는 형태는 여러 가지가 있지만 가장 간명한 것은 해당 데이터베이스 회사에서 제공하는 JDBC 드라이버를 설치하는 것이다. JDBC 드라이버란, 자바가 특정 회사 제품의 데이터베이스에 접근할 수 있도록 해주는 드라이버의 일종이다. 우리가 그래픽 카드를 구매하였더라도, 그래픽 카드를 구동하는 드라이버가 없으면 사용할 수 없는 것처럼, 데이터베이스도 해당 데이터베이스 회사가 제공하는 JDBC 드라이버가 있어야만 자바가 그 데이터베이스를 사용할 수 있다.

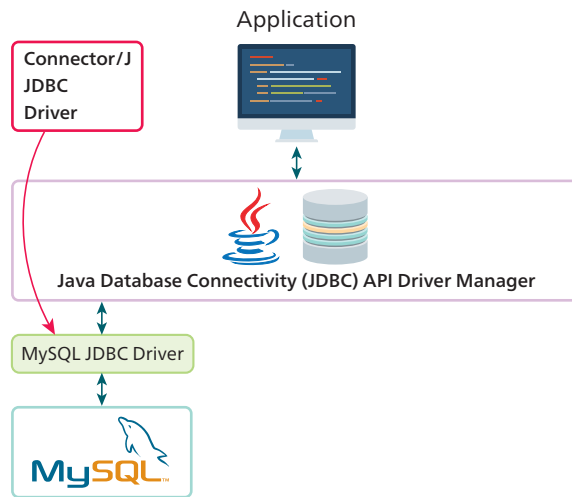


그림 18-15 MySQL JDBC 드라이버 Connector/J

MySQL에서의 JDBC는 Connector/J라고 불린다. <https://dev.mysql.com/downloads/connector/j/>에서 최신 버전을 다운로드할 수 있다.

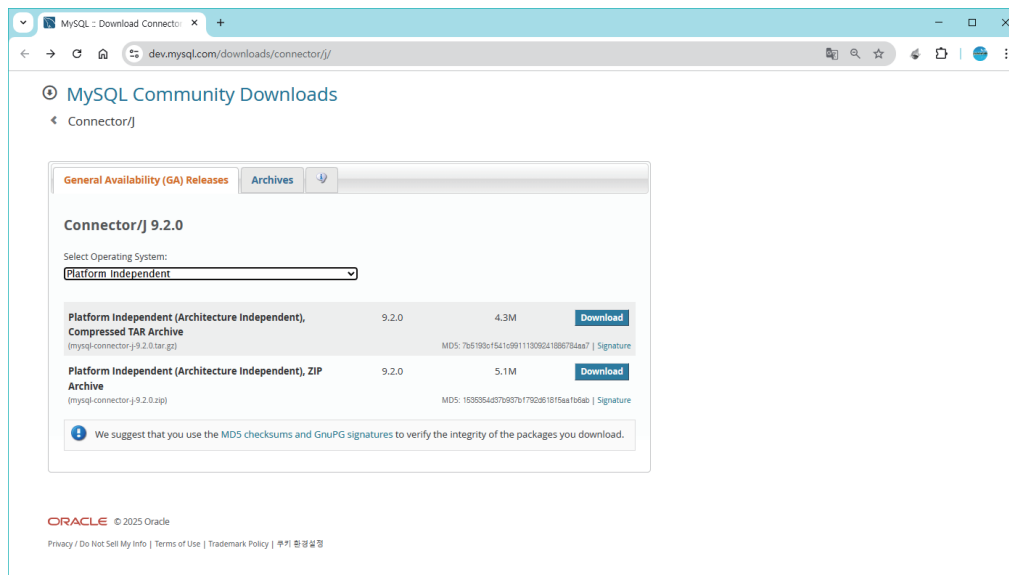


그림 18-16 외부 라이브러리 추가

보통 다운로드 파일 이름은 예를 들어 mysql-connector-j-9.2.33.zip과 같다. 이 압축 파일을 풀면 다음과 같은 폴더가 나타난다.

```
mysql-connector-j-9.2/
├─ LICENSE
├─ README
├─ CHANGES
├─ docs/      <- 설명서
└─ mysql-connector-j-9.2.jar ☒ 요게 중요!
```

바로 이 mysql-connector-j-9.2.jar 파일이 우리가 Java 프로젝트에서 사용하는 JDBC 드라이버이다.

인텔리제이에 추가하는 방법

1. 인텔리제이에서 프로젝트 우클릭 → 모듈 설정 열기(또는 [F4] 키)
2. 왼쪽에서 Libraries 선택 → 상단에 [+] 버튼 → Java
3. 위에서 찾은 mysql-connector-j-9.2.jar 파일 선택
4. Apply → OK

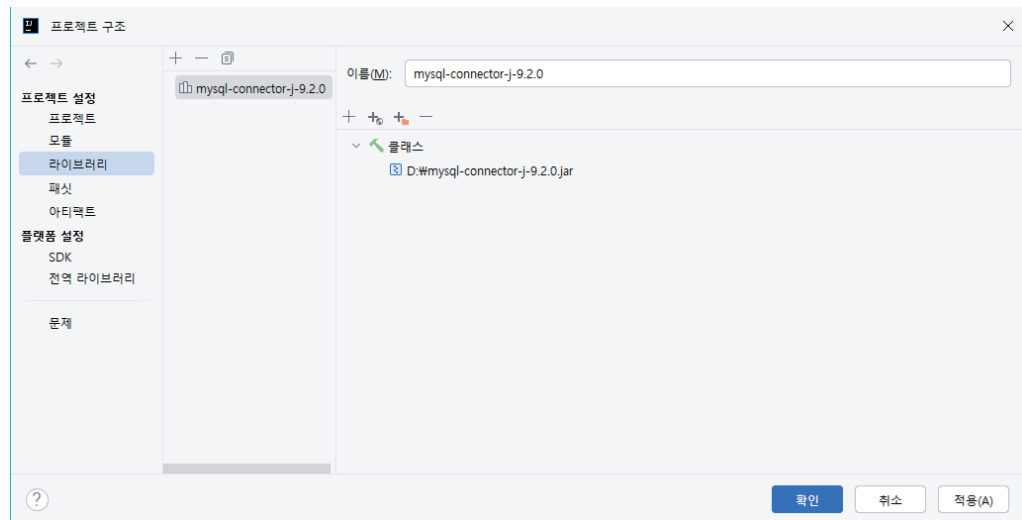


그림 18-17 인텔리제이에 외부 라이브러리 추가

이렇게 하면 인텔리제이가 빌드 시 해당 .jar 파일을 함께 사용한다.

JDBC를 이용한 데이터베이스 사용 절차

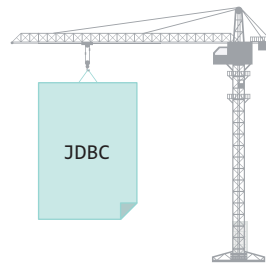
JDBC를 이용하여 데이터베이스를 사용하는 전형적인 절차는 다음과 같다.

1. URL로 지정된 JDBC 드라이버를 적재(load)한다.

2. 사용자 이름과 패스워드를 가지고 데이터베이스에 연결한다.
3. SQL 문장을 작성하여 전송하고 실행한다. SQL 명령어의 결과로 생성되는 결과 집합을 얻는다.
4. 결과 집합을 화면에 표시하거나 결과 집합을 처리한다. 사용이 끝나면 연결을 해제한다.

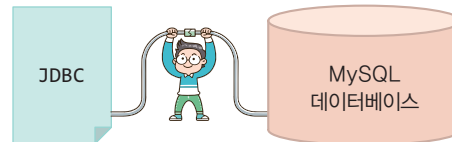
❶ JDBC 드라이버를 적재

```
Class.forName("com.mysql.jdbc.Driver");
```



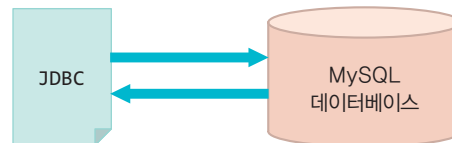
❷ 데이터베이스 연결

```
Connection con =  
DriverManager.getConnection(url, user, pw);
```



❸ SQL 문장 작성 및 전송

```
Statement stmt = con.createStatement();  
ResultSet rs =  
    stmt.executeQuery("SELECT * FROM books");
```



❹ 결과 집합 사용 후 연결 해제

```
while (rs.next()) {  
    int number = rs.getInt("book_id");  
    String name = rs.getString("title");  
}
```

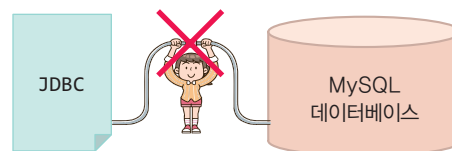


그림 18-18 JDBC를 이용한 데이터베이스 사용 절차

드라이버 클래스 적재

첫 번째 단계는 드라이버 클래스를 등록해서 프로그램 안으로 드라이버 클래스 파일을 적재하여야 한다. 흔히 JDBC 드라이버는 동적으로 적재하고 초기화한다. 동적으로 라이브러리를 적재하려면 `Class` 클래스의 `forName()` 메소드를 사용한다. `Class` 클래스는 JVM에서 사용하는 클래스들의 정보를 가지고 있는 메타 클래스이다. 예를 들어서 우리가 설치한 MySQL JDBC 드라이버를 적재하려면 다음과 같은 문장을 사용한다.

```
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
}
catch (ClassNotFoundException e) {
    System.out.println("드라이버를 찾을 수 없습니다.");
}
```

forName() 메소드는 *ClassNotFoundException*을 발생할 수 있기 때문에, 반드시 *try-catch* 문을 사용하여야 한다. JDBC 드라이버의 인스턴스는 우리가 생성하지 않는다. *Class.forName()* 메소드를 호출하면 드라이버 클래스가 정적 블록을 통해 자신의 인스턴스를 생성하고 초기화한 후 *DriverManager*에 등록한다. JDBC 4.0 이후 버전부터는 *forName()* 메소드를 호출하지 않아도 클래스 경로에서 사용할 수 있는 드라이버가 자동으로 적재되지만, 이전 버전을 사용한다면 *forName()* 메소드를 호출하여야 한다. 여기에서는 안전하게 *forName()* 메소드를 사용하였다.

데이터베이스 연결

드라이버를 적재하였으면 드라이버를 통해 데이터베이스 시스템과 연결할 수 있다. 연결하기 위해서는 *DriverManager* 클래스의 정적 메소드인 *getConnection()*을 호출한다. 이 메소드는 데이터베이스 연결을 확립하고 매개변수로 '데이터베이스 URL, 사용자 아이디, 패스워드'를 요구한다.

```
String url = "jdbc:mysql://localhost:3306/book_db?characterEncoding=UTF-8 &
serverTimezone=UTC";
String user = "root";
String password = "password";
con = DriverManager.getConnection(url, user, password);
```

URL 매개변수는 다음과 같은 문법을 가진다.

```
jdbc:subprotocol:subname
```

여기서 만약 MySQL을 사용하고 있다면 *subprotocol*은 *mysql*이다. *subname*은 데이터베이스 이름이다. 만약 네트워크에 있는 데이터베이스 파일이라면 완전한 URL이 된다. 로컬 컴퓨터에 있는 데이터베이스라면 *//localhost:3306/book_db*와 같이 된다. 여기서 3306은 포트 번호이다. 최근 MySQL 버전에서는 반드시 문자 인코딩과 서버 기준 시각을 붙여서 보내주어야 한다.

사용자 아이디와 패스워드는 물론 데이터베이스 서버에 등록되어 있어야 한다. MySQL의 경우에는 설치 시 아이디와 패스워드를 설정할 수 있다. 아이디는 항상 'root'이고 패스워드는 설치 시에 지정한 패스워드이다. `getConnection()` 메소드는 `SQLException`을 발생할 수 있다.

데이터베이스 연결하기

예제 18-1



자, 이제는 이 모든 것을 한데 묶어보자. 앞에서 설명한 대로 [MySQL 9.2 Command Line Client]를 사용하여 `book_db`를 미리 생성해야 오류가 발생하지 않는다.

ConnectDatabase.java

```

1  import java.sql.*;           // SQL 관련 클래스
2
3  public class ConnectDatabase {
4      // 데이터베이스 연결을 생성하는 메소드
5      public static Connection makeConnection() {
6          // 데이터베이스 연결 정보
7          String url = "jdbc:mysql://localhost:3306/
8                      book_db?characterEncoding=UTF-8&serverTimezone=UTC";
9          String id = "root";    // 데이터베이스 사용자 ID
10         String password = "root"; // 데이터베이스 비밀번호
11         Connection con = null;  // 연결 객체 초기화
12
13         try {
14             // MySQL JDBC 드라이버 적재
15             Class.forName("com.mysql.cj.jdbc.Driver");
16             System.out.println("드라이버 적재 성공");
17             // 드라이버 적재 성공 메시지 출력
18
19             // 데이터베이스 연결 생성
20             con = DriverManager.getConnection(url, id, password);
21             System.out.println("데이터베이스 연결 성공"); // 연결 성공 메시지 출력
22         } catch (ClassNotFoundException e) {
23             // JDBC 드라이버를 찾지 못했을 때 예외 처리
24             System.out.println("드라이버를 찾을 수 없습니다.");
25         } catch (SQLException e) {
26             // 데이터베이스 연결 실패 시 예외 처리
27             System.out.println("연결에 실패하였습니다.");
28         }
29         return con; // 연결 객체 반환
30     }
31
32     public static void main(String arg[]) throws SQLException {
33         // 데이터베이스 연결 생성

```

```

34     Connection con = makeConnection();
35
36     // 데이터베이스 연결 종료
37     con.close();
38 }
39 }

```

드라이버 적재 성공
데이터베이스 연결 성공

만약 “드라이버를 찾을 수 없습니다.”라는 오류 메시지가 나오면 ‘Connector/J’가 제대로 설치되지 않은 것이다. “연결에 실패하였습니다.” 메시지가 나오면 MySQL 안에 book_db를 올바르게 생성하였는지 확인한다.

도전문제



1 영화에 대한 데이터베이스를 생성한 후에 위의 프로그램을 실행하시오.

먼저 MySQL 9.2 Command Line Client를 실행하고, 다음과 같이 데이터베이스와 테이블을 생성한다.

```

CREATE DATABASE movie_db;
USE movie_db;
CREATE TABLE movies (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100),
    director VARCHAR(100),
    release_year INT
);

```

이제 ConnectDatabase.java에서 데이터베이스 이름과 포트 번호를 변경하여, 위에서 만든 movie_db에 연결하도록 수정한다.

SQL 문장 실행

데이터베이스를 연결한 후에는 SELECT와 같은 SQL 문장들을 실행할 수 있다. 이때 사용되는 것이 Connection, Statement, ResultSet 인터페이스이다.

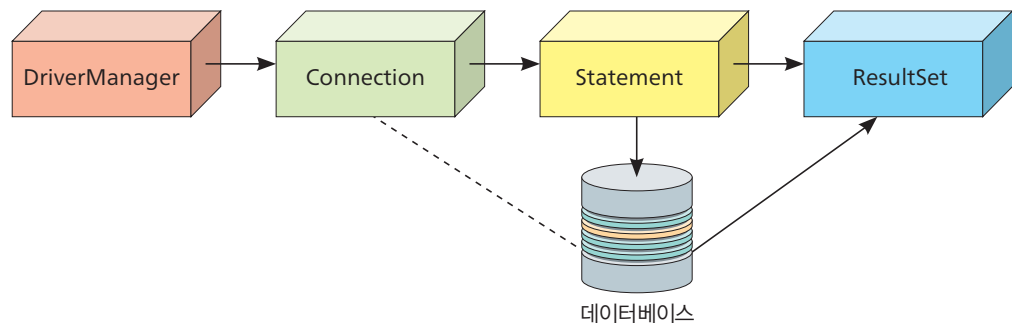


그림 18-19 Connection, Statement, ResultSet 인터페이스의 역할

예를 들어서 SELECT 문장을 실행하려면 다음과 같은 문장을 사용한다.

```
Statement s = con.createStatement();           // 문장 객체 생성
String query = "SELECT * FROM books ORDER BY book_id"; // SQL 문장 생성
ResultSet rows = s.executeQuery(query);        // SQL 문장 실행
```

여기서 쿼리의 실행으로 생성되는 결과 집합은 rows라는 변수에 저장된다.

결과 집합에서 이동

다음 단계는 결과 집합에서 레코드를 하나씩 접근하여 작업해야 한다. executeQuery() 메소드에 의하여 반환된 ResultSet 객체에는 SELECT 문장으로 추출된 모든 레코드가 들어 있다. 하지만 우리는 한 번에 하나의 레코드만 접근할 수 있다. 이것을 위하여 커서(cursor)라는 포인터가 제공된다. 커서를 움직이는 다양한 메소드가 있다. 결과 집합에서 레코드를 하나씩 처리하는 예는 다음과 같다.

```
while (rows.next()) {
    // 현재 레코드를 처리한다.
    int id = rs.getInt("book_id");
    ...
}
```

커서 객체는 previous(), first(), last(), absolute(), relative(), beforeFirst(), afterLast()와 같은 다양한 이동 메소드들을 가지고 있다.

결과 집합 처리

다음 단계는 레코드에서 컬럼의 값을 추출하는 것이다. 이를 위해 많은 메소드가 준비되어 있다. 이 메소드들은 두 가지의 카테고리로 나눌 수 있다. 하나는 컬럼을 이름으로 접근하고 다른 하나는 컬럼을 번호로 접근한다. 만약 번호를 안다면 숫자로 접근하는 것이 더 효율적이다. 현재 레코드에서 학번과 이름을 추출하는 코드를 작성하면 다음과 같다.

```
int id = row.getInt("id");
String name = row.getString("name");
```

다음 코드에서는 동일한 작업을 컬럼 번호를 이용하여 수행한다.

```
int id = row.getInt(1);
String name = row.getString(2);
```

자바에서는 인덱스 번호가 0부터 시작하지만, SQL에서는 1부터 시작한다. 레코드에서 컬럼 값을 추출하는 메소드는 getXXX()와 같은 형태를 가진다.



예제 18-2

데이터베이스에서 데이터 출력하기

우리가 작성한 데이터베이스에서 책을 전부 검색하여 콘솔에 출력하는 프로그램을 작성해 보자.

ConnectDatabase.java

```
1 import java.sql.*;           // SQL 관련 클래스
2
3 public class ConnectDatabase {
4     // 데이터베이스 연결을 생성하는 메소드
5     public static Connection makeConnection() {
6         // 데이터베이스 연결 정보
7         String url = "jdbc:mysql://localhost:3306/
8             book_db?characterEncoding=UTF-8&serverTimezone=UTC";
9         String id = "root";           // 데이터베이스 사용자 ID
10        String password = "root";      // 데이터베이스 비밀번호
11        Connection con = null;         // 연결 객체 초기화
12
13        try {
14            // MySQL JDBC 드라이버 적재
15            Class.forName("com.mysql.cj.jdbc.Driver");
16            System.out.println("드라이버 적재 성공");
17            // 드라이버 적재 성공 메시지 출력
18
19            // 데이터베이스 연결 생성
20            con = DriverManager.getConnection(url, id, password);
21            System.out.println("데이터베이스 연결 성공"); // 연결 성공 메시지 출력
22        } catch (ClassNotFoundException e) {
23            // JDBC 드라이버를 찾지 못했을 때 예외 처리
24            System.out.println("드라이버를 찾을 수 없습니다.");
25        } catch (SQLException e) {
26            // 데이터베이스 연결 실패 시 예외 처리
27            System.out.println("연결에 실패하였습니다.");
28        }
29        return con;                 // 연결 객체 반환
30    }
```



```

31
32 public static void main(String arg[]) throws SQLException {
33     // 데이터베이스 연결 생성
34     Connection con = makeConnection();
35
36     // SQL 문을 실행하기 위한 Statement 객체 생성
37     Statement stmt = con.createStatement();
38
39     // books 테이블의 모든 데이터를 조회하는 쿼리 실행
40     ResultSet rs = stmt.executeQuery("SELECT * FROM books");
41
42     // 조회된 결과를 처리
43     while (rs.next()) {
44         // 각 행의 데이터 읽기
45         int id = rs.getInt("book_id");           // book_id 열 데이터
46         String title = rs.getString("title");    // title 열 데이터
47
48         // 결과 출력
49         System.out.println(id + " " + title);
50     }
51
52     // 데이터베이스 연결 종료
53     con.close();
54 }
55 }

```

드라이버 적재 성공
 데이터베이스 연결 성공
 1 Operating System Concepts
 2 Head First PHP and MYSQL
 3 C Programming Language
 4 Head First SQL

- 1** books 테이블에서 가격이 높은 순으로 정렬된 책 목록을 출력하는 프로그램을 작성하시오. 콘솔에 출력할 때 책의 book_id, title, price를 모두 출력한다. 가격은 높은 순(내림차순)으로 정렬하여 출력한다.

```
SELECT * FROM books ORDER BY price DESC;
```

도전문제



레코드 추가, 수정, 삭제

만약 레코드를 추가하거나 수정, 삭제하려면 `executeUpdate()` 메소드를 사용하여야 한다. `executeUpdate()` 메소드는 얼마나 많은 레코드가 변경되었는지를 반환한다. 이 반환 값을 이용하여 데이터들이 제대로 추가되었는지를 확인할 수 있다.



예제 18-3

레코드 추가, 수정, 삭제하기

책의 이름, 출판사, 발행 연도, 가격을 받아서 테이블에 추가하는 프로그램을 작성해 보자.

SQLInsertTest.java

```

1  import java.sql.*; // SQL 관련 클래스
2
3  public class SQLInsertTest {
4      // 데이터베이스 연결을 생성하는 메소드
5      public static Connection makeConnection() {
6          // 데이터베이스 연결 정보
7          String url = "jdbc:mysql://localhost:3306/
8                          book_db?characterEncoding=UTF-8&serverTimezone=UTC";
9          String id = "root"; // 데이터베이스 사용자 ID
10         String password = "root"; // 데이터베이스 비밀번호
11         Connection con = null; // 연결 객체 초기화
12
13         try {
14             // MySQL JDBC 드라이버 적재
15             Class.forName("com.mysql.cj.jdbc.Driver");
16             System.out.println("드라이버 적재 성공");
17             // 드라이버 적재 성공 메시지 출력
18
19             // 데이터베이스 연결 생성
20             con = DriverManager.getConnection(url, id, password);
21             System.out.println("데이터베이스 연결 성공"); // 연결 성공 메시지 출력
22         } catch (ClassNotFoundException e) {
23             // JDBC 드라이버를 찾지 못했을 때 예외 처리
24             System.out.println("드라이버를 찾을 수 없습니다.");
25         } catch (SQLException e) {
26             // 데이터베이스 연결 실패 시 예외 처리
27             System.out.println("연결에 실패하였습니다.");
28         }
29         return con; // 연결 객체 반환
30     }
31
32     public static void main(String arg[]) {
33         // 책 정보를 데이터베이스에 추가
34         addBook("Artificial Intelligence", "Addison Wesley", "2002", 35000);
35     }
36
37     // 책 정보를 데이터베이스에 추가하는 메소드
38     private static void addBook(String title, String publisher, String year,
39                                 int price) {
40         Connection con = makeConnection(); // 데이터베이스 연결 생성
41         try {

```

```

42 // SQL 문 실행을 위한 Statement 객체 생성
43 Statement stmt = con.createStatement();
44
45 // INSERT 쿼리 생성
46 String s = "INSERT INTO books (title, publisher, year, price)
47                                     VALUES ";
48 s += "(" + title + "," + publisher + "," + year + "," +
49                                     price + ")";
50 System.out.println(s); // 생성된 쿼리 출력(디버깅용)
51
52 // INSERT 쿼리 실행
53 int i = stmt.executeUpdate(s);
54
55 // 실행 결과 확인
56 if (i == 1) // 1개의 레코드가 추가되었을 경우
57     System.out.println("레코드 추가 성공");
58 else
59     System.out.println("레코드 추가 실패");
60 } catch (SQLException e) {
61     // SQL 예외 처리
62     System.out.println(e.getMessage());
63     System.exit(0); // 프로그램 종료
64 } finally {
65     try {
66         // 데이터베이스 연결 닫기
67         if (con != null) {
68             con.close();
69         }
70     } catch (SQLException e) {
71         // 연결 닫기 중 예외 처리
72         System.out.println("연결 닫기 실패: " + e.getMessage());
73     }
74 }
75 }
76 }

```

드라이버 적재 성공
 데이터베이스 연결 성공
 INSERT INTO books (title, publisher, year, price) VALUES
 ('Artificial Intelligence','Addison Wesley','2002','35000')
 레코드 추가 성공

makeConnection() 메소드는 이전에 등장하였던 함수와 동일하다. 연결이 성립되고 난 후에 Statement 객체가 생성되고 매개변수로 전달된 값들을 이용하여 INSERT 문장이 구성된다. 이어서 executeUpdate() 메소드가 호출되어 INSERT 문장을 실행한다. 만약 반환 값이 1이면 올바르게 레코드가 추가된 것이고 그렇지 않으면 추가에 실패한 것이다. UPDATE나 DELETE 문장도 같은 방법으로 실행할 수 있다. SQL 문장을 구성한 후에는 콘솔에 출력해보는 것이 디버깅에 도움이 된다.

도전문제



- 1 위의 코드에서는 책 정보를 코드 내부에 하드 코딩했지만, 이제는 `Scanner` 클래스를 사용하여 사용자로부터 입력을 받아서 테이블에 책 정보를 추가하는 프로그램을 작성하시오.

책 제목을 입력하세요: 인공지능 개론
 출판사를 입력하세요: 인피니티박스
 발행 연도를 입력하세요: 2023
 가격을 입력하세요: 32000

SELF TEST



- 1 JDBC를 이용하여 데이터베이스를 사용하는 전형적인 절차를 설명하시오.
- 2 JDBC 드라이버를 동적으로 적재할 때 사용되는 메소드는?
- 3 SQL 문장이 저장되는 객체는?
- 4 결과 집합이 저장되는 객체는?

18.5 Prepared Statements 사용하기

Prepared Statements는 많이 사용되는 SQL 문장이 있는 경우에 이것을 미리 SQL 문장으로 만들어두고 필요할 때마다 사용하는 기법이다. 예를 들어 데이터베이스에 저장된 책 중에서 특정 출판사의 책만을 찾는 쿼리를 Prepared Statements로 작성해 보자.

```
SELECT books.title, books.price
FROM books
WHERE publisher = 외부에서 제공
```

이 쿼리를 미리 만들어두고 필요할 때마다 사용해 보자. Prepared Statements를 사용하면 성능이 향상된다. 외부에서 제공되는 값은 ?로 표시된다. 위와 같은 SQL 문장은 다음과 같이 Prepared Statements로 만들 수 있다.

```
String query =
    "SELECT books.title, books.price" +
    " FROM books" +
```

```
" WHERE publisher = ?";
```

```
PreparedStatement stmt = con.prepareStatement(query);
```

Prepared Statements를 실행하기 전에 ?에 해당하는 값을 주어야 한다.

```
stmt.setString(1, "Wiley");
```

첫 번째 인수는 ? 변수의 번호이다. 두 번째 인수는 ? 변수의 값이다. 이렇게만 해놓으면 언제든지 필요할 때마다 Prepared Statements를 실행할 수 있다.

```
ResultSet rs = stmt.executeQuery();
```

만약 INSERT, DELETE, UPDATE 문장을 사용하였다면 executeUpdate() 메소드를 호출하여야 한다.

Prepared Statements

예제 18-4



Prepared Statements를 사용하여 많이 사용되는 쿼리 문장을 미리 만들어두고 필요할 때마다 사용해 보자.

SQLPreparedTest.java

```
1 import java.sql.*;           // SQL 관련 클래스
2
3 public class SQLPreparedTest {
4     // 데이터베이스 연결을 생성하는 메소드
5     public static Connection makeConnection() {
6         // 데이터베이스 연결 정보
7         String url = "jdbc:mysql://localhost:3306/
8             book_db?characterEncoding=UTF-8&serverTimezone=UTC";
9         String id = "root";     // 데이터베이스 사용자 ID
10        String password = "root"; // 데이터베이스 비밀번호
11        Connection con = null;  // 연결 객체 초기화
12
13        try {
14            // MySQL JDBC 드라이버 적재
15            Class.forName("com.mysql.cj.jdbc.Driver");
16            System.out.println("드라이버 적재 성공");
17            // 드라이버 적재 성공 메시지 출력
```

```

18
19         // 데이터베이스 연결 생성
20         con = DriverManager.getConnection(url, id, password);
21         System.out.println("데이터베이스 연결 성공"); // 연결 성공 메시지 출력
22     } catch (ClassNotFoundException e) {
23         // JDBC 드라이버를 찾지 못했을 때 예외 처리
24         System.out.println("드라이버를 찾을 수 없습니다.");
25     } catch (SQLException e) {
26         // 데이터베이스 연결 실패 시 예외 처리
27         System.out.println("연결에 실패하였습니다.");
28     }
29     return con; // 연결 객체 반환
30 }
31
32 public static void main(String arg[]) throws SQLException {
33     // 데이터베이스 연결 생성
34     Connection con = makeConnection();
35
36     // SELECT 쿼리 정의 (PreparedStatement 사용)
37     String query = "SELECT books.title, books.price"
38     // 가져올 열: title, price
39         + " FROM books"           // 테이블: books
40         + " WHERE publisher = ?"; // 조건: publisher 값이 특정 값과 일치
41
42     // PreparedStatement 객체 생성
43     PreparedStatement stmt = con.prepareStatement(query);
44
45     // 쿼리 매개변수 설정
46     stmt.setString(1, "Wiley"); // 첫 번째 물음표에 "Wiley" 값 설정
47
48     // 쿼리 실행 및 결과 저장
49     ResultSet rs = stmt.executeQuery();
50
51     // 결과 처리
52     while (rs.next()) { // 결과 집합에서 다음 행이 있을 경우 반복
53         String title = rs.getString("title"); // title 열 데이터 가져오기
54         int price = rs.getInt("price"); // price 열 데이터 가져오기
55         System.out.println(title + " " + price); // 결과 출력
56     }
57
58     // 데이터베이스 연결 닫기
59     con.close();
60 }
61 }

```

드라이버 적재 성공
 데이터베이스 연결 성공
 Operating System Concepts 30700

- 1 `PreparedStatement`를 사용하여 출판사(publisher)가 특정 값인 책들의 제목과 가격을 출력하는 프로그램을 작성하시오. 사용자로부터 출판사 이름을 입력받기(`Scanner` 사용), 쿼리문은 `PreparedStatement`로 미리 준비, 입력한 출판사에 해당하는 책 목록을 출력한다.

도전문제



- 1 Prepared Statements는 무엇인가?
- 2 Prepared Statements는 왜 사용하는가?

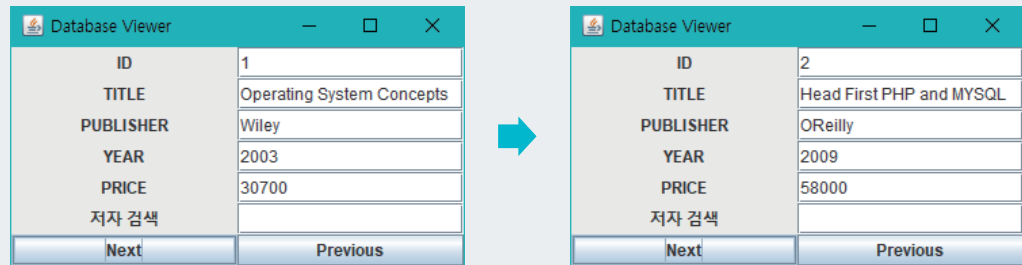
SELF TEST



LAB

GUI로 데이터베이스 내용 표시하기

다음과 같이 그래픽 사용자 인터페이스를 이용하여 데이터베이스 테이블의 내용을 화면에 표시하는 프로그램을 작성해 보자.



데이터베이스의 내용을 GUI로 보여주기 위해서는 두 가지 요소가 필요하다. 첫째, 데이터베이스 연동(JDBC)이다. JDBC 드라이버를 로드하고, Connection → Statement → ResultSet의 순서로 데이터를 가져온다. 예를 들어, 특정 테이블(Book)의 내용을 `SELECT * FROM Book` 쿼리로 읽을 수 있다. 둘째, 그래픽 사용자 인터페이스(Swing)이다. JFrame을 만들고, 책의 정보를 보여줄 여러 개의 JTextField를 배치한다. 그리고 Next, Previous 버튼을 눌렀을 때 이벤트가 발생하도록 ActionListener를 연결한다. 버튼이 눌리면 ResultSet의 커서를 `next()` 또는 `previous()`로 이동시켜 현재 레코드의 값을 읽고, 이를 텍스트 필드에 채워 넣는다. 데이터베이스 연동과 GUI는 별개의 부분처럼 보이지만, 프로그램에서는 ResultSet을 이동할 때마다 화면이 갱신되도록 연결해 주면 된다.

Hint



버튼이 눌러질 때마다 결과 집합에서 앞에 레코드로 가거나 뒤에 레코드로 가면 된다. `previous()`와 `next()` 메소드를 사용한다.

GUI로 데이터베이스 내용 표시하기

SOLUTION

ConnectDatabase.java

```

1  // AWT 패키지 import
2  import java.awt.GridLayout;
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  // SQL 패키지 import
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9  import java.sql.ResultSet;
10 import java.sql.SQLException;
11 import java.sql.Statement;
12
13 // Swing 패키지 import
14 import javax.swing.JButton;
15 import javax.swing.JFrame;
16 import javax.swing.JLabel;
17 import javax.swing.JTextField;
18
19 // JFrame을 상속받는 클래스 MyFrame 정의
20 class MyFrame extends JFrame {
21     // GUI 컴포넌트 및 데이터베이스 관련 필드 선언
22     JTextField id, title, p, year, price, author; // 텍스트 필드
23     JButton previousButton, nextButton, InsertButton, deleteButton,
24     searchButton; // 버튼
25     ResultSet rs; // 결과 집합(ResultSet)
26     Statement stmt; // SQL 문 실행 객체
27
28     // 생성자 정의
29     public MyFrame() throws SQLException {
30         super("Database Viewer"); // 프레임 제목 설정
31
32         Connection con = makeConnection(); // 데이터베이스 연결 생성
33         stmt = con.createStatement(); // Statement 객체 생성
34         rs = stmt.executeQuery("SELECT * FROM books");
35         // 데이터베이스에서 책 정보를 조회
36
37         setLayout(new GridLayout(0, 2)); // 레이아웃 설정: 2열 그리드 레이아웃
38
39         // 각 필드에 레이블과 텍스트 필드 추가
40         add(new JLabel("ID", JLabel.CENTER));

```

```

41         add(id = new JTextField());
42         add(new JLabel("TITLE", JLabel.CENTER));
43         add(title = new JTextField());
44         add(new JLabel("PUBLISHER", JLabel.CENTER));
45         add(p = new JTextField());
46         add(new JLabel("YEAR", JLabel.CENTER));
47         add(year = new JTextField());
48         add(new JLabel("PRICE", JLabel.CENTER));
49         add(price = new JTextField());
50         add(new JLabel("저자 검색", JLabel.CENTER));
51         add(author = new JTextField());
52
53         // 'Previous' 버튼 생성 및 이벤트 리스너 추가
54         previousButton = new JButton("Previous");
55         previousButton.addActionListener(new ActionListener() {
56             public void actionPerformed(ActionEvent event) {
57                 try {
58                     rs.previous();                // 결과 집합을 이전 행으로 이동
59                     // 이전 행의 데이터를 각 텍스트 필드에 설정
60                     id.setText("" + rs.getInt("book_id"));
61                     title.setText("" + rs.getString("title"));
62                     p.setText("" + rs.getString("publisher"));
63                     year.setText("" + rs.getString("year"));
64                     price.setText("" + rs.getInt("price"));
65                 } catch (SQLException e) {
66                     e.printStackTrace();          // SQL 예외 출력
67                 }
68             }
69         });
70
71         // 'Next' 버튼 생성 및 이벤트 리스너 추가
72         nextButton = new JButton("Next");
73         nextButton.addActionListener(new ActionListener() {
74             public void actionPerformed(ActionEvent event) {
75                 try {
76                     rs.next();                    // 결과 집합을 다음 행으로 이동
77                     // 다음 행의 데이터를 각 텍스트 필드에 설정
78                     id.setText("" + rs.getInt("book_id"));
79                     title.setText("" + rs.getString("title"));
80                     p.setText("" + rs.getString("publisher"));
81                     year.setText("" + rs.getString("year"));
82                     price.setText("" + rs.getInt("price"));
83                 } catch (SQLException e) {
84                     e.printStackTrace();          // SQL 예외 출력
85                 }
86             }

```

```

87         });
88
89         // 버튼 추가
90         add(nextButton);
91         add(previousButton);
92
93         // 프레임 기본 설정
94         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 닫기 동작 설정
95         setSize(350, 200); // 프레임 크기 설정
96         // pack(); // 컴포넌트 크기에 맞게 프레임 크기 조정(주석 처리)
97         setVisible(true); // 프레임 표시
98     }
99
100    // 데이터베이스 연결 메소드
101    public static Connection makeConnection() {
102        // 데이터베이스 URL
103        String url = "jdbc:mysql://localhost:3306/
104                    book_db?characterEncoding=UTF-8&serverTimezone=UTC";
105        String id = "root"; // 데이터베이스 사용자 ID
106        String password = "root"; // 데이터베이스 비밀번호
107        Connection con = null; // 연결 객체 초기화
108
109        try {
110            Class.forName("com.mysql.cj.jdbc.Driver");
111            // MySQL JDBC 드라이버 적재
112            System.out.println("드라이버 적재 성공");
113            con = DriverManager.getConnection(url, id, password);
114            // 데이터베이스 연결 생성
115            System.out.println("데이터베이스 연결 성공");
116        } catch (ClassNotFoundException e) {
117            System.out.println("드라이버를 찾을 수 없습니다.");
118            // 드라이버 적재 실패 시 메시지 출력
119        } catch (SQLException e) {
120            System.out.println("연결에 실패하였습니다.");
121            // 연결 실패 시 메시지 출력
122        }
123        return con; // 연결 객체 반환
124    }
125 }
126
127 // 메인 클래스
128 public class ConnectDatabase {
129     public static void main(String[] args) throws SQLException {
130         SwingUtilities.invokeLater(() -> new MyFrame());
131     }
132 }

```

18.6 JDBC를 사용하여 이미지 저장하기

PreparedStatement 인터페이스의 `setBinaryStream()` 메소드는 매개변수의 인덱스를 나타내는 정수와 `InputStream` 객체를 받아들이고, 매개변수를 주어진 `InputStream` 객체로 설정한다. 매우 큰 바이너리 값을 보내야 할 때마다 이 방법을 사용할 수 있다. SQL 데이터베이스는 Blob(Binary Large Object)이라는 데이터 유형을 제공하므로 이미지와 같은 대용량 이진 데이터를 저장할 수 있다.

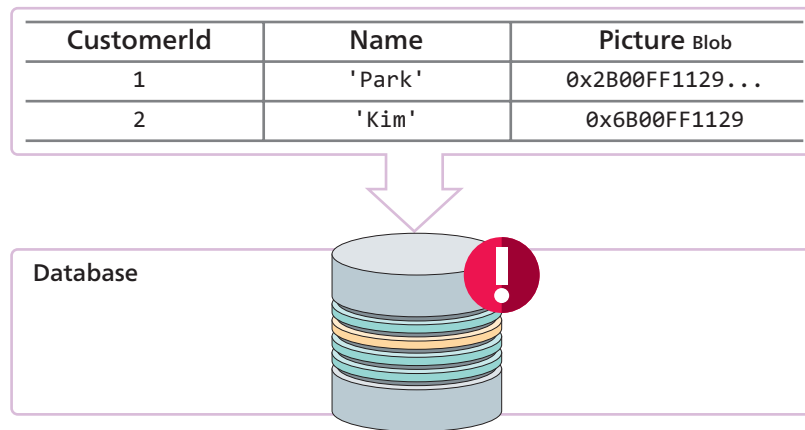


그림 18-20 JDBC를 사용하여 이미지 저장하기

JDBC를 사용하여 이미지 저장

JDBC 프로그램을 사용하여 데이터베이스에 이미지를 저장해야 하는 경우 아래와 같이 Blob 데이터 유형으로 테이블을 생성한다.

```
CREATE TABLE Animals(Name VARCHAR(255), Type VARCHAR(50), Image BLOB);
```

이제 JDBC를 사용하여 데이터베이스에 연결하고 위의 생성된 테이블에 값을 삽입할 PreparedStatement를 준비한다.

```
String query = "INSERT INTO Animals(Name, Type, Image) VALUES (?, ?, ?)";
PreparedStatement pstmt = con.prepareStatement(query);
```

PreparedStatement 인터페이스의 설정자 메소드를 사용하여 자리 표시자에 값을 설정하고 `setBinaryStream()` 메소드를 사용하여 Blob 데이터 유형 값을 설정한다.

```
FileInputStream fin = new FileInputStream("dog.jpg");
pstmt.setBinaryStream(3, fin);
```

참고로 JDBC를 사용하여 .gif 또는 .jpeg 또는 .png 유형의 이미지만 저장하고 검색할 수 있다.

이미지 저장

예제 18-5



다음은 JDBC 프로그램을 사용하여 MySQL 데이터베이스에 이미지를 삽입하는 방법을 보여주는 예제이다. 여기서 테이블에 Blob 데이터 유형을 만들고, 테이블의 내용을 검색한다.

ImageDatabase.java

```
1  import java.sql.*;           // SQL 관련 패키지
2  import java.io.*;           // 입출력 관련 패키지
3
4  public class ImageDatabase {
5
6      // 데이터베이스 연결을 생성하는 메소드
7      public static Connection makeConnection() {
8          // 데이터베이스 연결 설정
9          String url = "jdbc:mysql://localhost:3306/
10                      image_db?characterEncoding=UTF-8&serverTimezone=UTC";
11          String user = "root";           // 데이터베이스 사용자 ID
12          String password = "root";       // 데이터베이스 비밀번호
13          Connection con = null;
14
15          try {
16              // MySQL JDBC 드라이버 적재
17              Class.forName("com.mysql.cj.jdbc.Driver");
18              System.out.println("드라이버 적재 성공");
19              // 데이터베이스 연결 생성
20              con = DriverManager.getConnection(url, user, password);
21              System.out.println("데이터베이스 연결 성공");
22          } catch (ClassNotFoundException e) {
23              System.out.println("드라이버를 찾을 수 없습니다.");
24          } catch (SQLException e) {
25              System.out.println("연결에 실패하였습니다.");
26          }
27          return con; // 연결 객체 반환
28      }
29
30      public static void main(String args[]) throws Exception {
31          // 데이터베이스 연결 생성
32          Connection con = makeConnection();
```

```

33 // SQL 문 실행을 위한 Statement 객체 생성
34 Statement stmt = con.createStatement();
35
36 // 기존 테이블 삭제(필요한 경우 사용)
37 // stmt.execute("DROP TABLE Animals");
38
39 // Animals 테이블 생성 쿼리
40 String createTable = "CREATE TABLE Animals( "
41     + "Name VARCHAR(255), " // 동물 이름
42     + "Type VARCHAR(50), " // 동물 유형
43     + "Image BLOB)"; // 이미지 (BLOB 타입)
44 stmt.execute(createTable); // 테이블 생성 실행
45
46 // Animals 테이블에 데이터를 삽입하는 쿼리
47 String query = "INSERT INTO Animals(Name, Type, Image)
48     VALUES (?, ?, ?)";
49 PreparedStatement pstmt = con.prepareStatement(query);
50
51 // 첫 번째 데이터 삽입(강아지 데이터)
52 pstmt.setString(1, "Dog"); // Name
53 pstmt.setString(2, "WHITE DOG"); // Type
54 FileInputStream fin = new FileInputStream("d:\\dog1.jpg");
55 // 이미지 파일 읽기
56 pstmt.setBinaryStream(3, fin); // Image
57 pstmt.execute(); // 쿼리 실행
58
59 // 두 번째 데이터 삽입(고양이 데이터)
60 pstmt.setString(1, "Cat"); // Name
61 pstmt.setString(2, "GRAY CAT"); // Type
62 fin = new FileInputStream("d:\\cat1.jpg"); // 이미지 파일 읽기
63 pstmt.setBinaryStream(3, fin); // Image
64 pstmt.execute(); // 쿼리 실행
65
66 System.out.println("데이터가 추가되었습니다."); // 데이터 삽입 완료 메시지
67
68 // Animals 테이블에서 데이터 조회
69 ResultSet rs = stmt.executeQuery("Select *from Animals");
70 while (rs.next()) {
71     // 조회 결과 출력
72     System.out.print("Name: " + rs.getString("Name") + ", ");
73     System.out.print("Type: " + rs.getString("Type") + ", ");
74     System.out.print("Image: " + rs.getBlob("Image"));
75     // 이미지 (Blob 데이터)
76     System.out.println();
77 }
78

```

```

79     // 데이터베이스 연결 닫기
80     con.close();
81 }
82 }

```

드라이버 적재 성공
데이터베이스 연결 성공
데이터가 추가되었습니다.

Name: Dog, Type: WHITE DOG, Image: com.mysql.cj.jdbc.Blob@327514f
Name: Cat, Type: GRAY CAT, Image: com.mysql.cj.jdbc.Blob@5b12b668

- 1** 사용자로부터 이름, 유형, 이미지 파일 경로를 입력받아 **Animals** 테이블에 데이터를 저장하는 프로그램을 작성하시오.

동물 이름을 입력하세요: Dog
동물 유형을 입력하세요: WHITE DOG
이미지 파일 경로를 입력하세요: d:/images/dog.jpg

도전문제



이미지 추출하기

예제 18-6



ImageDatabase2.java

```

1  import java.io.FileOutputStream;    // 파일 출력 스트림 관련 클래스
2  import java.sql.Blob;           // Blob 데이터 타입을 처리하기 위한 클래스
3  import java.sql.Connection;     // 데이터베이스 연결 관련 클래스
4  import java.sql.DriverManager;   // JDBC 드라이버 관리 클래스
5  import java.sql.PreparedStatement; // SQL 문을 실행하기 위한 클래스
6  import java.sql.ResultSet;      // SQL 쿼리 결과를 저장하는 클래스
7
8  public class ImageDatabase2 {
9      // 데이터베이스 연결을 생성하는 메소드
10     public static Connection makeConnection() {
11         // 데이터베이스 연결 정보
12         String url = "jdbc:mysql://localhost:3306/
13             image_db?characterEncoding=UTF-8&serverTimezone=UTC";
14         String user = "root";     // 데이터베이스 사용자 ID
15         String password = "root"; // 데이터베이스 비밀번호
16         Connection con = null;
17
18         try {

```

```

19         // MySQL JDBC 드라이버 적재
20         Class.forName("com.mysql.cj.jdbc.Driver");
21         System.out.println("드라이버 적재 성공");
22         // 데이터베이스 연결 생성
23         con = DriverManager.getConnection(url, user, password);
24         System.out.println("데이터베이스 연결 성공");
25     } catch (Exception e) {
26         System.out.println("데이터베이스 연결 실패: " + e.getMessage());
27     }
28     return con;                // 연결 객체 반환
29 }
30
31 public static void main(String args[]) throws Exception {
32     // 데이터베이스 연결 생성
33     Connection con = makeConnection();
34
35     // Animals 테이블에서 모든 데이터를 조회하는 SQL 문
36     PreparedStatement ps = con.prepareStatement("SELECT * FROM Animals");
37
38     // SQL 실행 및 결과 저장
39     ResultSet rs = ps.executeQuery();
40
41     // 첫 번째 행의 데이터를 가져옴
42     if (rs.next()) {
43         Blob b = rs.getBlob(3);
44         // 세 번째 컬럼(이미지 데이터)을 Blob 객체로 가져옴
45         byte barr[] = b.getBytes(1, (int) b.length());
46         // Blob 데이터를 바이트 배열로 변환
47
48         // Blob 데이터를 파일로 저장
49         FileOutputStream fout = new FileOutputStream("d:\\dog11.jpg");
50         // 저장할 파일 경로 지정
51         fout.write(barr);                // 파일에 바이트 배열 데이터 쓰기
52
53         // 파일 출력 스트림 닫기
54         fout.close();
55     }
56
57     // 성공 메시지 출력
58     System.out.println("이미지를 추출하였습니다.");
59
60     // 데이터베이스 연결 닫기
61     con.close();
62 }
63 }

```

드라이버 적재 성공
데이터베이스 연결 성공
이미지를 추출하였습니다.

- 1 사용자로부터 동물 이름(Name 컬럼)을 입력받고, 해당 이름의 이미지만 추출하여 파일로 저장하는 프로그램을 작성하시오.

```
SELECT Image FROM Animals WHERE Name = ?;
```

요구사항

- Scanner로 동물 이름 입력받기
- 해당 이미지가 없다면 "이미지가 존재하지 않습니다." 메시지 출력
- 있다면 d:\[이름].jpg 형태로 저장 (예: d:\Cat.jpg)

동물 이름을 입력하세요: Cat
이미지를 d:\Cat.jpg로 저장하였습니다.

도전문제



- 1 데이터베이스에 이미지를 저장하려면 SQL의 어떤 데이터 유형을 사용해야 하는가?
- 2 ResultSet에서 Blob를 추출하는 메소드 이름은?

SELF TEST



18.7 JDBC를 사용하여 텍스트 파일 저장하기

데이터베이스에는 텍스트 파일도 저장할 수 있다. MySQL에서 CLOB(TINYTEXT, TEXT, MEDIUMTEXT, LONG TEXT) 데이터 유형으로 저장하면 된다. JDBC는 데이터베이스의 테이블에 파일을 저장하기 위해 CLOB 데이터 유형에 대한 지원을 제공한다. PreparedStatement 인터페이스의 setCharacterStream() 메소드는 인덱스를 나타내는 정수와 Reader 객체를 매개변수로 받아서 Reader 객체(파일)의 내용을 지정된 인덱스에 저장한다.

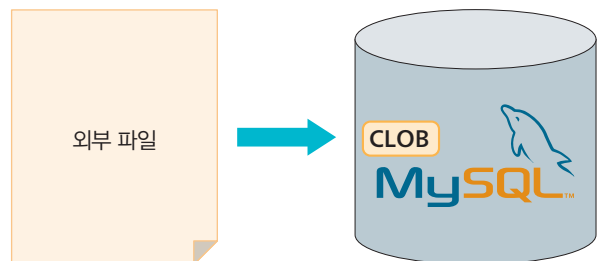


그림 18-21 JDBC를 사용하여 텍스트 파일 저장하기

JDBC를 사용하여 텍스트 파일 저장하기

JDBC 프로그램을 사용하여 데이터베이스에 파일을 저장해야 하는 경우 아래와 같이 CLOB (TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT) 데이터 유형으로 테이블을 생성한다.

```
CREATE TABLE Articles(Name VARCHAR(255), Article LONGTEXT);
```

JDBC를 사용하여 데이터베이스에 연결하고 위의 생성된 테이블에 값을 삽입할 Prepared Statement를 준비한다.

```
String query = "INSERT INTO Tutorial(Name, Article) VALUES (?, ?)";
PreparedStatement pstmt = con.prepareStatement(query);
```

PreparedStatement 인터페이스의 설정자 메소드를 사용하여 자리 표시자에 값을 설정하고 setCharacterStream() 메소드를 사용하여 CLOB 데이터 유형 값을 설정한다.



예제 18-7

텍스트 파일 저장하기

CLOB를 이용하여 D 드라이브에 있는 test.txt 파일을 데이터베이스 테이블 안에 저장해 보자. 한글이 들어간 파일은 반드시 UTF-8로 저장된 파일이어야 한다. 코드가 다르면 데이터베이스 안에서 한글이 깨지게 된다.

TextDatabase.java

```
1 import java.io.*; // 파일 입출력 관련 클래스
2 import java.sql.*; // SQL 관련 클래스
3 import java.nio.charset.StandardCharsets; // UTF-8 인코딩 지원
4
5 public class TextDatabase {
6     // 데이터베이스 연결 생성 메소드
7     public static Connection makeConnection() {
8         // 데이터베이스 연결 정보
9         String url = "jdbc:mysql://localhost:3306/
10             book_db?characterEncoding=UTF-8&serverTimezone=UTC";
11         String id = "root"; // 데이터베이스 사용자 ID
12         String password = "root"; // 데이터베이스 비밀번호
13         Connection con = null;
14
15         try {
16             // MySQL JDBC 드라이버 적재
17             Class.forName("com.mysql.cj.jdbc.Driver");
```

```

18         System.out.println("드라이버 적재 성공");
19         // 데이터베이스 연결 생성
20         con = DriverManager.getConnection(url, id, password);
21         System.out.println("데이터베이스 연결 성공");
22     } catch (ClassNotFoundException e) {
23         System.out.println("드라이버를 찾을 수 없습니다.");
24     } catch (SQLException e) {
25         System.out.println("연결에 실패하였습니다.");
26     }
27     return con; // 연결 객체 반환
28 }
29
30 public static void main(String args[]) throws Exception {
31     // 데이터베이스 연결 생성
32     Connection con = makeConnection();
33
34     // Statement 객체 생성
35     Statement stmt = con.createStatement();
36
37     // 기존 테이블 삭제(필요 시 주석 해제)
38     // String dropTable = "drop table Articles;";
39     // stmt.execute(dropTable);
40
41     // Articles 테이블 생성 SQL 쿼리
42     String createTable = "CREATE TABLE Articles(Name VARCHAR(255),
43                                     Article LONGTEXT);";
44     stmt.execute(createTable); // 테이블 생성 실행
45
46     // Articles 테이블에 데이터를 삽입하는 SQL 쿼리
47     String query = "INSERT INTO Articles VALUES (?, ?)";
48     PreparedStatement pstmt = con.prepareStatement(query);
49
50     try {
51         // 파일 읽기
52         File file = new File("d:\\test.txt"); // 텍스트 파일 경로
53         FileReader fileReader
54             = new FileReader(file, StandardCharsets.UTF_8);
55         // UTF-8로 파일 읽기
56
57         // PreparedStatement에 데이터 바인딩
58         pstmt.setInt(1, 2); // 첫 번째 값: ID
59         pstmt.setCharacterStream(2, fileReader, (int) file.length());
60         // 두 번째 값: 파일 내용
61
62         pstmt.executeUpdate(); // 데이터 삽입 실행
63         System.out.println("파일 저장 성공");

```

```

64
65         // 리소스 정리
66         pstmt.close();
67         con.close();
68     } catch (Exception e) {
69         e.printStackTrace();           // 예외 발생 시 스택 트레이스 출력
70     }
71 }
72 }

```

드라이버 적재 성공
데이터베이스 연결 성공
파일 저장 성공

도전문제



- 1 사용자로부터 텍스트 파일 경로와 이름을 입력받아 **Articles** 테이블에 저장하는 프로그램을 작성하십시오.

저장할 텍스트 파일 경로를 입력하세요: d:\test.txt
파일 이름을 입력하세요: TestFile1

- Name 컬럼에는 사용자가 입력한 파일 이름을, Article 컬럼에는 파일 내용을 저장할 것.
- 파일은 반드시 UTF-8로 인코딩되어 있어야 함.



예제 18-8

텍스트 추출하기

앞에서 저장하였던 텍스트 파일을 꺼내서 D 드라이브에 다른 이름으로 저장해 보자.

TextDatabase2.java

```

1  import java.io.*;           // 파일 입출력 관련 클래스
2  import java.sql.*;          // SQL 관련 클래스
3
4  public class TextDatabase2 {
5      // 데이터베이스 연결 생성 메소드
6      public static Connection makeConnection() {
7          // 데이터베이스 연결 정보
8          String url = "jdbc:mysql://localhost:3306/
9                      book_db?characterEncoding=UTF-8&serverTimezone=UTC";
10         String id = "root";    // 데이터베이스 사용자 ID
11         String password = "1234"; // 데이터베이스 비밀번호
12         Connection con = null;
13

```

```

14     try {
15         // MySQL JDBC 드라이버 적재
16         Class.forName("com.mysql.cj.jdbc.Driver");
17         System.out.println("드라이버 적재 성공");
18         // 데이터베이스 연결 생성
19         con = DriverManager.getConnection(url, id, password);
20         System.out.println("데이터베이스 연결 성공");
21     } catch (ClassNotFoundException e) {
22         System.out.println("드라이버를 찾을 수 없습니다.");
23         // 드라이버 적재 실패 메시지
24     } catch (SQLException e) {
25         System.out.println("연결에 실패하였습니다.");
26         // 데이터베이스 연결 실패 메시지
27     }
28     return con; // 연결 객체 반환
29 }
30
31 public static void main(String args[]) throws Exception {
32     // 데이터베이스 연결 생성
33     Connection con = makeConnection();
34
35     // Articles 테이블에서 데이터를 조회하기 위한 PreparedStatement 생성
36     PreparedStatement ps = con.prepareStatement
37         ("SELECT * FROM Articles");
38     ResultSet rs = ps.executeQuery(); // 쿼리 실행 및 결과 저장
39     rs.next(); // 첫 번째 결과로 이동
40
41     // 두 번째 컬럼(Article)의 데이터를 CLOB 형식으로 가져오기
42     Clob c = rs.getClob(2); // CLOB 데이터를 가져옴
43     Reader r = c.getCharacterStream(); // CLOB 데이터를 읽을 Reader 생성
44
45     // CLOB 데이터를 파일에 저장하기 위한 FileWriter 생성
46     FileWriter fw = new FileWriter("d:\\test11.txt");
47     // 저장할 파일 경로 지정
48
49     // CLOB 데이터를 읽어와 파일에 쓰기
50     int i;
51     while ((i = r.read()) != -1) // 데이터를 한 글자씩 읽음
52         fw.write((char) i); // 읽은 데이터를 파일에 씀
53
54     // 리소스 정리
55     fw.close(); // FileWriter 닫기
56     con.close(); // 데이터베이스 연결 닫기
57
58     // 성공 메시지 출력
59     System.out.println("텍스트가 추출되었습니다.");

```

```

60         con.close();           // 데이터베이스 연결 닫기(중복 호출)
61     }
62 }

```

드라이버 적재 성공
데이터베이스 연결 성공
텍스트가 추출되었습니다.

도전문제



- 1 사용자로부터 문서 이름(Name)을 입력받아 해당 CLOB 텍스트를 파일로 저장한다. **Articles** 테이블에서 사용자가 입력한 **Name** 값을 가진 텍스트만 추출하여 .txt 파일로 저장한다. 해당 프로그램을 작성하시오.

요구사항

- **Scanner**를 사용해 사용자에게 **Name**을 입력받음
- 입력한 이름이 존재하지 않으면 "해당 이름의 문서는 존재하지 않습니다." 출력
- 있으면 d:\[입력한이름]_복사본.txt 형태로 저장

저장할 문서 이름을 입력하세요: SampleText
SampleText_복사본.txt 파일로 저장되었습니다!

SELF TEST



- 1 데이터베이스에 상당히 긴 텍스트를 저장하려면 SQL의 어떤 데이터 유형을 사용해야 하는가?
- 2 데이터베이스의 CLOB에서 텍스트를 추출하는 절차를 설명하시오.

명예의 전당

Mini Project
수행하기

사용자의 게임 점수를 기록하는 데이터베이스 프로그램을 만들어 보자.

난이도: ★★☆☆☆

주제: JDBC

이름	점수
Lee	54321
Kim	12345

이름 점수

먼저 MySQL 콘솔을 실행하여서 다음과 같은 SQL 명령어를 사용하여 gamescore라고 하는 데이터베이스를 생성한다.

```
Enter password: *****
...
mysql> CREATE DATABASE gamescore;
Query OK, 1 row affected (0.00 sec)

mysql> USE gamescore;
Database changed
mysql> CREATE TABLE scores (name TEXT, score INT);
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE scores;
+-----+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| name  | text   | YES  |     | NULL    |       |
| score | int(11) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.02 sec)
```

Hint



JTable을 사용한다. JTable은 2차원 데이터를 화면에 테이블 형태로 보여주기 위해 사용하는 컴포넌트로, javax.swing.JTable 클래스에 정의되어 있다. 스윙 애플리케이션에서 데이터 목록을 표 형태로 표시할 때 사용한다.

```
String[] columnNames = {"이름", "나이", "이메일"};

// 데이터 배열(행 x 열)
Object[][] data = {
    {"홍길동", 25, "hong@example.com"},
    {"김영희", 30, "kim@example.com"},
    {"이철수", 22, "lee@example.com"}
};

// JTable 생성
JTable table = new JTable(data, columnNames);

// JScrollPane에 JTable 추가
JScrollPane scrollPane = new JScrollPane(table);
```


Summary

- JDBC(Java Database Connectivity)는 자바 애플리케이션과 데이터베이스를 연결하는 라이브러리이다.

```
import java.sql.*;
```

- 데이터베이스 관리 시스템(DBMS)은 다수의 사용자를 위하여 데이터를 저장, 접근, 변경하는 기능을 제공한다.

```
Connection conn = DriverManager.getConnection(url, user, password);
```

- MySQL의 JDBC는 Connector/J라고 불린다. Connector/J를 자바 가상 머신이 해당 드라이버 파일을 찾을 수 있도록 해야 한다.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

- JDBC를 이용하여 데이터베이스를 사용하는 전형적인 절차는 다음과 같다. URL로 지정된 JDBC 드라이버를 적재(load)한다. 사용자 이름과 패스워드를 가지고 데이터베이스에 연결한다. SQL 문장을 작성하여서 전송하고 실행한다. SQL 명령어의 결과로 생성되는 결과 집합을 얻는다. 결과 집합을 화면에 표시하거나 처리한다. 사용이 끝나면 연결을 해제한다.

```
ResultSet rs = stmt.executeQuery("SELECT * FROM table_name");
```

- Prepared Statements는 많이 사용되는 SQL 문장이 있을 경우에 이것을 미리 SQL 문장으로 만들어두고 필요할 때마다 사용하는 기법이다.

```
PreparedStatement ps = conn.prepareStatement  
("SELECT * FROM users WHERE id=?");
```

- SQL 데이터베이스는 Blob(Binary Large Object) 데이터 유형을 제공하므로, 이미지와 같은 대용량 이진 데이터를 저장할 수 있다.

```
PreparedStatement ps = conn.prepareStatement  
("INSERT INTO images(data) VALUES(?)");
```

- 데이터베이스에는 텍스트 파일도 저장할 수 있다. CLOB(TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT) 데이터 유형으로 저장하면 된다.

```
PreparedStatement ps = conn.prepareStatement  
("INSERT INTO docs(content) VALUES(?)");
```

Exercises

1. 다음 중 JDBC에서 데이터베이스 드라이버를 적재할 때 사용하는 메소드는?

① getConnection()	② findDriver()
③ forName()	④ setConnection()

2. 다음 중 JDBC에서 DML 문을 수행하는 메소드는?

① executeResult()	② executeQuery()
③ executeUpdate()	④ execute()

3. 다음 문장의 빈칸을 채우시오.
 1. 표준 데이터베이스 언어는 _____이다.
 2. 하나의 데이터베이스는 여러 개의 _____로 이루어진다.
 3. SQL 쿼리 문의 검색 결과를 _____라고 한다.
 4. SELECT 문을 실행하는 메소드는 _____이다.

4. 다음 질문에 답하시오. 인터넷으로 조사해도 좋다.
 1. JDBC는 무엇인가?
 2. JDBC 드라이버는 무엇인가?
 3. DriverManager가 하는 일은 무엇인가?
 4. Blob이란 무엇인가?
 5. CLOB이란 무엇인가?
 6. Class.forName() 메소드가 하는 일은 무엇인가?
 7. Statement와 PreparedStatement의 차이점은 무엇인가?
 8. execute(), executeQuery(), executeUpdate() 메소드의 차이점은 무엇인가?
 9. 데이터베이스에서 이미지를 어떻게 저장하고 검색할 수 있는가?

5. 다음 설명에 부합하는 JDBC의 클래스 이름을 작성하시오.

1. _____: 이 클래스는 JDBC API를 사용하여 데이터베이스에 연결할 때 사용한다. 개발자는 해당 데이터베이스에 대한 JDBC 드라이버가 적재되고 JVM 메모리에서 초기화된 후에만 데이터베이스에 대한 연결을 얻을 수 있다.
2. _____: 이 클래스는 데이터베이스에 대한 여러 가지 작업을 실행할 때 사용한다.
3. _____: 개발자가 JDBC API를 사용하여 쿼리를 실행한 후 쿼리 결과가 저장되는 클래스이다.

6. 자바에서 JDBC를 사용하는 순서로 올바르게 나열하시오.

- ① 연결을 종료한다.
- ② **Statement** 객체를 실행하고 쿼리 결과 집합을 반환한다.
- ③ JDBC 드라이버를 적재하고 등록한다.
- ④ 데이터베이스에 대한 연결을 연다.
- ⑤ 쿼리를 수행하기 위해 **Statement** 개체를 생성한다.
- ⑥ 결과 집합 및 **Statement** 객체를 닫는다.
- ⑦ 결과 집합을 처리한다.

7. 자동차에 대한 데이터를 저장하는 데이터베이스를 설계하시오. 자동차는 유일한 등록 번호 (license number), 제조사(manufacturer), 차종(type), 제작연도(year), 연비(efficiency)를 가지고 있다.

1. **Car** 테이블을 생성하는 SQL을 작성하시오.
2. 몇 개의 데이터를 입력하는 SQL 문을 작성하시오.
3. 모든 레코드를 출력하는 SQL 문을 작성하시오.
4. 제작 연도가 2021년인 자동차의 등록 번호를 순서대로 출력하는 SQL 문을 작성하시오.

Programming Exercises

난이도: ★★☆☆☆
주제: JDBC 사용

1. JDBC를 이용하여 본문에 등장한 book_db 데이터베이스에 연결하고 다음과 같은 작업들을 수행하는 프로그램을 작성하시오.

1. 데이터베이스에 있는 모든 테이블의 이름을 출력한다.
2. 데이터베이스의 각 테이블에 대해 컬럼(열)을 출력한다.
3. 테이블의 각 컬럼(열)에 대해 데이터 유형을 출력한다.

난이도: ★★☆☆☆
주제: JDBC 사용

2. JDBC API와 MySQL를 사용하여 영화를 저장하는 프로그램을 작성하시오.

- MOVIES 열이 있는 테이블을 만든다: id type INTEGER AUTO INCREMENT, title type VARCHAR (255), genre type VARCHAR (255), year type INTEGER
- 3개의 레코드를 MOVIES 테이블에 추가한다.
- 선택한 하나의 레코드를 업데이트한다.
- 지정된 ID로 선택한 레코드를 삭제한다.
- 데이터베이스의 모든 레코드를 출력한다.

난이도: ★★☆☆☆
주제: JDBC 사용

3. 사용자 아이디와 패스워드를 저장하는 데이터베이스 테이블을 작성하시오. GUI 사용자 인터페이스도 함께 구현하시오. 테이블의 구조는 다음과 같다.

name	password
hong	1234
...	...

4. 스케줄을 데이터베이스에 저장하는 프로그램을 만든다. 스케줄은 제목, 날짜, 시작 시간, 종료 시간으로 구성된다. 예를 들면 다음과 같다.

- “자바 공부하기”, 2022.5.27, 10:00, 12:00
- “강아지 목욕”, 2022.5.28, 17:00, 18:00

스케줄을 추가하고 삭제하며, 특정한 날짜의 스케줄을 출력하는 사용자 인터페이스를 작성하시오.

1. 사용자 인터페이스를 콘솔 기반으로 작성한다.
2. 사용자 인터페이스를 그래픽 기반으로 작성한다.

난이도: ★★★★★☆

주제: JDBC 사용

5. 간단한 게시판을 만든다. 게시판은 메시지를 나열하며 메시지는 제목, 발송자, 발송일, 내용으로 구성된다. 예를 들면 다음과 같다.

- “과제 제출”, 홍길동, 2025.5.27, “이번 주 과제를 금요일까지 제출...”
- “질문”, 김철수, 2025.6.10, “테이블이 두 개이고 서로 연결되어 있는 경우에는...”

메시지를 추가하고 삭제하며, 검색할 수 있는 사용자 인터페이스를 작성하시오. 기타 상용 게시판에 있는 기능들을 추가하면 좋다.

1. 사용자 인터페이스를 콘솔 기반으로 작성한다.
2. 사용자 인터페이스를 그래픽 기반으로 작성한다.

난이도: ★★★★★☆

주제: JDBC 사용

Introduction to **JAVA Programming**