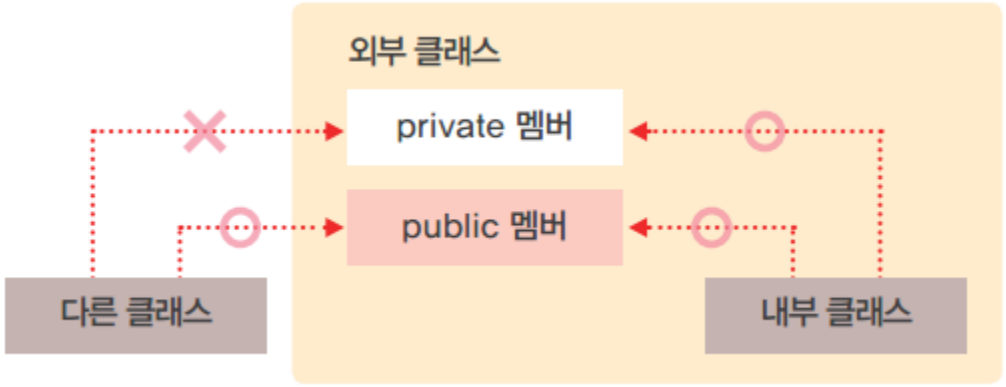
Chapter 08

내부 클래스와 람다식

Contents

내부 클래스와 내부 인터페이스의 개념



내부 클래스는 외부 클래스를
상속할 필요 없이
외부 클래스의 private 멤버까지
사용할 수 있다.

- 내부 클래스나 내부 인터페이스는 외부로부터 스스로를 감추며, 외부 클래스와 밀접한 관계를 가짐
- 내부 클래스는 외부 클래스의 private멤버를 비롯해 모든 멤버에 자유롭게 접근 가능
- 따라서 일반적으로 프로그램을 유지 보수하기가 쉽고 프로그램이 간단해지지만 내부 클래스가 복잡할 경우 가독성 저하

내부 클래스의 종류

내부 클래스와 내부 인터페이스 구조

```
class 외부클래스 {
    class 멤버클래스 {
    }

    interface 내부인터페이스 {
    }
}
```

외부 클래스의 멤버로 선언된 클래스이다.

```
class 외부클래스 {
    void 메서드() {
        class 지역클래스 {
        }
    }
}
```

외부 클래스의 메서드 내부에 선언된 클래스이다.

내부 클래스와 내부 인터페이스 사용(1)

- 컴파일 결과 파일

```
외부클래스$멤버클래스.class
외부클래스$내부인터페이스.class
외부클래스$1지역클래스.class
```

이름이 동일한 지역 클래스가 있다면 $2 등을 사용한다.

- 내부 클래스가 외부 클래스를 참조하려면

```
외부클래스.this
```

- 멤버 클래스의 객체 생성

```
외부클래스.인스턴스멤버클래스 변수 = 외부클래스의객체변수.new 인스턴스멤버생성자();
외부클래스.정적멤버클래스 변수 = new 외부클래스.정적멤버클래스생성자();
```

- [예제 8-1]

```java
public class MemberClassDemo {
    private String secret = "비공개";
    String s = "외부";

    class MemberClass {
        String s = "내부";

        public void show1() {
            System.out.println("인스턴스 멤버 클래스");
            System.out.println(secret);

            System.out.println(s);

            System.out.println(MemberClassDemo.this.s);
        }

        // static String s3 = "정적 멤버 필드";
        // static void show2() {}
    }

    public static void main(String[] args) {
        MemberClassDemo m = new MemberClassDemo();
        MemberClassDemo.MemberClass m1 = m.new MemberClass();

        System.out.println(m1.s);
        m1.show1();
    }
}
```

```
내부
인스턴스 멤버 클래스
비공개
내부
외부
```

내부 클래스와 내부 인터페이스 사용(2)

- 지역 클래스
  - 메서드 내부에서만 사용하므로 접근 지정자나 static을 명시할 수 없다.
  - 지역 클래스는 지역 변수 final로만 참조할 수 있는데, 이는 메서드가 종료되어 지역 변수가 소멸되더라도 지역 클래스가 소멸된 지역 변수를 변경하지 못하도록 하기 위해서다.
  - JDK 8부터는 지역 클래스가 참조하는 지역 변수는 final로 명시하지 않더라도 final로 간주

- 내부 인터페이스

- [예제 8-2]

```
3  public class LocalClassDemo {
4      private String s1 = "외부";
5
6      void method() {
7          int x = 1;
8          class LocalClass {
9              String s2 = "내부";
10             String s3 = s1;
11
12             public void show() {
13                 System.out.println("지역 클래스");
14                 // x = 2;
15             }
16         }
17         LocalClass lc = new LocalClass();
18         System.out.println(lc.s2);
19         lc.show();
20     }
21
22     public static void main(String[] args) {
23         LocalClassDemo lcd = new LocalClassDemo();
24         lcd.method();
25     }
26 }
```

내부
지역 클래스

- [예제 8-3]

```
3  class Icon {
4      interface Touchable {
5          void touch();
6      }
7  }
8
9  public class InnerInterfaceDemo implements Icon.Touchable {
10     public void touch() {
11         System.out.println("아이콘을 터치한다.");
12     }
13
14     public static void main(String[] args) {
15         Icon.Touchable btn = new InnerInterfaceDemo();
16         btn.touch();
17     }
18 }
```

```
아이콘을 터치한다.
```

무명 클래스 개념

- 한 번만 사용하기 때문에 이름이 없는 클래스
- 예

```
                    extends
class OnlyOnce implement Parent {
  // Parent가 클래스라면 오버라이딩한 메서드
  // Parent가 인터페이스라면 구현한 메서드
}


Parent p = new OnlyOnce();
```

OnlyOnce라는 이름이 필요 없음.
필요한 것은 본체뿐.

```
Parent p = new Parent() {
  // Parent가 클래스라면 오버라이딩한 메서드
  // Parent가 인터페이스라면 구현한 메서드
};
```

무명 클래스 본체로서 OnlyOnce 클래스의 본체와 동일하다.

하나의 실행문이므로 세미콜론(;)으로 끝난다.

무명 클래스의 활용

- 개념을 살펴보기 위하여 부모로 사용할 샘플 클래스

[예제 8-4] 무명 클래스의 부모로 사용할 클래스 sec02/Bird.java

```
01  public class Bird {
02     public void move() {
03        System.out.println("새가 움직인다~~~.");
04     }
05  }
```

- [예제 8-5] 기명 멤버 클래스

```
3  public class MemberDemo {
4      class Eagle extends Bird {
5          public void move() {
6              System.out.println("독수리가 난다~~~.");
7          }
8
9          public void sound() {
10             System.out.println("휘익~~~.");
11         }
12     }
13
14     Eagle e = new Eagle();
15
16     public static void main(String[] args) {
17         MemberDemo m = new MemberDemo();
18         m.e.move();
19         m.e.sound();
20     }
21 }
```

```
독수리가 난다~~~.
휘익~~~.
```

- [예제 8-6] 무명 멤버 클래스

```
3  public class Anonymous1Demo {
4      Bird e = new Bird() {
5          public void move() {
6              System.out.println("독수리가 난다~~~.");
7          }
8
9          void sound() {
10             System.out.println("휘익~~~.");
11         }
12     };
13
14     public static void main(String[] args) {
15         Anonymous1Demo a = new Anonymous1Demo();
16         a.e.move();
17         // a.e.sound();
18     }
19 }
```

```
독수리가 난다~~~.
```

- [예제 8-7] 기명 지역 클래스

```
3  public class LocalDemo {
4      public static void main(String[] args) {
5          class Eagle extends Bird {
6              public void move() {
7                  System.out.println("독수리가 난다~~~.");
8              }
9          }
10
11         Bird e = new Eagle();
12         e.move();
13     }
14 }
```

```
독수리가 난다~~~.
```

- [예제 8-8] 무명 지역 클래스

```
3  public class Anonymous2Demo {
4      public static void main(String[] args) {
5          Bird b = new Bird() {
6              public void move() {
7                  System.out.println("독수리가 난다~~~.");
8              }
9          };
10         b.move();
11     }
12 }
```

```
독수리가 난다~~~.
```

Comparable 및 Comparator 인터페이스

- [예제 8-9]

```
3  import java.util.Arrays;
4
5  class Rectangle {
6      private int width, height;
7
8      public Rectangle(int width, int height) {
9          this.width = width;
10         this.height = height;
11     }
12
13     public int findArea() {
14         return width * height;
15     }
16
17     public String toString() {
18         return "사각형[넓이=" + width + ", 높이=" + height + "]";
19     }
20 }
21
22 public class ComparableDemo {
23     public static void main(String[] args) {
24         Rectangle[] rectangles = { new Rectangle(3, 5),
25                 new Rectangle(2, 10), new Rectangle(5, 5) };
26
27         Arrays.sort(rectangles);
28
29         for (Rectangle r : rectangles)
30             System.out.println(r);
31     }
32 }
```

java.lang.ClassCastException:

Comparable 및 Comparator 인터페이스

- 정렬 메서드의 두 가지 방식

```
Object[] sort(Object[] array, int type) {
    if (type == 1) {
        넓이로 비교하기   정렬하기
    } else if (type == 2) {
        둘레로 비교하기   정렬하기
    } else …
}
```

새로운 기준으로 비교한다면 관련 내용을 추가해야 한다.

(a) 제공된 비교 기준으로 정렬

새로운 기준으로 비교한다면 새로운 메서드를 구현해야 한다.

```
Object[] sortByArea(Object[] array) {
    넓이로 비교하기   정렬하기
}

Object[] sortByPerimeter(Object[] array) {
    둘레로 비교하기   정렬하기
}
```

(b) 비교 기준마다 다른 메서드로 정렬

Comparable 인터페이스

- 정렬 메서드

```
static void Arrays.sort(Object[] a);
```

Comparable 인터페이스 타입의 배열이어야 한다.

- 자바는 Comparable 인터페이스를 다음과 같이 선언

```
public interface Comparable {
  int compareTo(Object o);
}
```

매개변수가 Object 타입이기 때문에 다형성으로 어떤 객체든 대입할 수 있다.

- [예제 8-10]

```java
import java.util.Arrays;

class Rectangle implements Comparable {
    private int width, height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int findArea() {
        return width * height;
    }

    public String toString() {
        return "사각형[넓이=" + width + ", 높이=" + height + "]";
    }

    public int compareTo(Object o) {
        Rectangle other = (Rectangle) o;

        if (this.findArea() < other.findArea())
            return -1;
        else if (this.findArea() > other.findArea())
            return 1;
        else
            return 0;
    }
}
```

⋮

- [예제 8-10]-cont.

```
33 public class ComparableDemo {
34     public static void main(String[] args) {
35         Rectangle[] rectangles = { new Rectangle(3, 5),
36                 new Rectangle(2, 10), new Rectangle(5, 5) };
37 
38         Arrays.sort(rectangles);
39 
40         for (Rectangle r : rectangles)
41             System.out.println(r);
42     }
43 }
```

```
사각형[넓이=3, 높이=5]
사각형[넓이=2, 높이=10]
사각형[넓이=5, 높이=5]
```

Comparator 인터페이스

- String 클래스
  - final 클래스
  - 사전 방식으로 비교할 수 있도록 이미 구현
- 문자열을 사전 방식이 아닌 다른 방식으로 정렬하려면?

- 자바는 다음과 같은 다른 정렬 메서드를 제공

```
static void Arrays.sort(String[] a, Comparator<String> c);
```

- 자바는 Comparator 인터페이스를 다음과 같이 선언

```
public interface Comparator<String> {
  int compare(String s1, String s2);
}
```

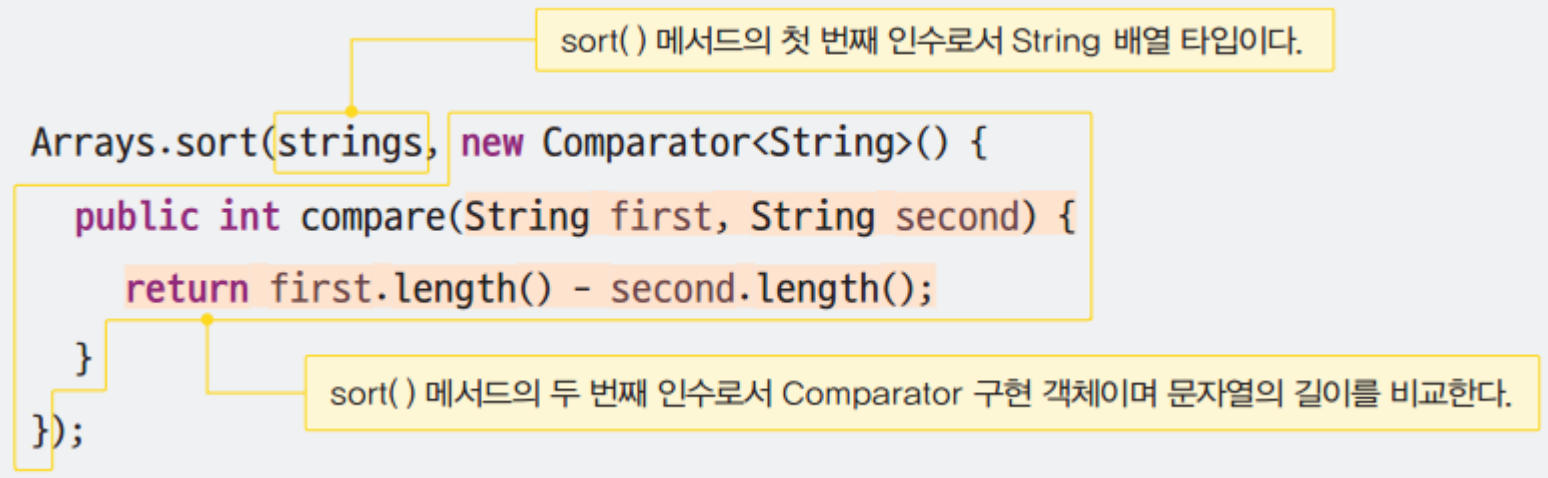
- [예제 8-11]

```java
import java.util.Arrays;
import java.util.Comparator;

public class ComparatorDemo {
    public static void main(String[] args) {
        String[] strings = { "로마에 가면 로마법을 따르라.", "시간은 금이다.", "펜은 칼보다 강하다." };

        Arrays.sort(strings, new Comparator<String>() {
            public int compare(String first, String second) {
                return first.length() - second.length();
            }
        });

        for (String s : strings)
            System.out.println(s);
    }
}
```

```
시간은 금이다.
펜은 칼보다 강하다.
로마에 가면 로마법을 따르라.
```

람다식의 필요성과 의미(1)

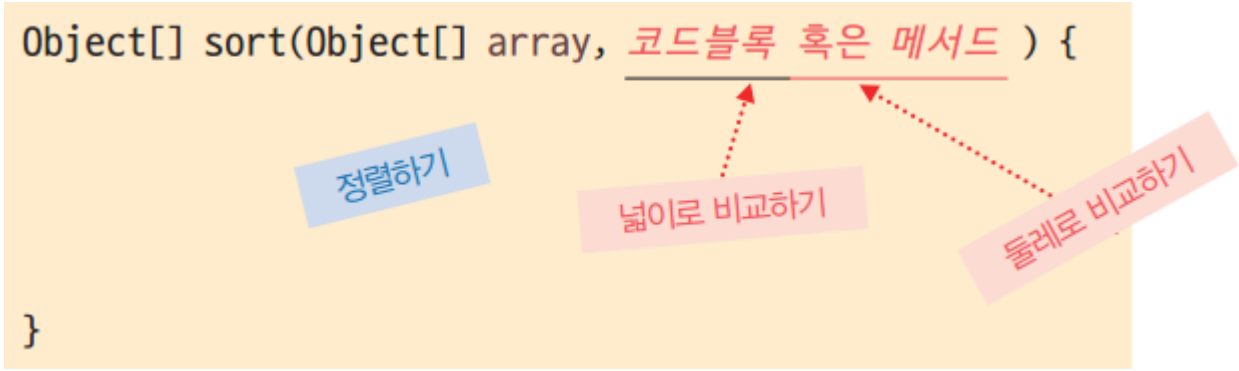
- 문자열을 길이 순서대로 정렬하는 코드



- 위 코드에서 Arrays.sort()의 두 번째 인수의 많은 부분이 불필요

람다식의 필요성과 의미(2)

- 정렬 메서드의 인수로 코드 블록이나 메서드를 사용할 수 있다면



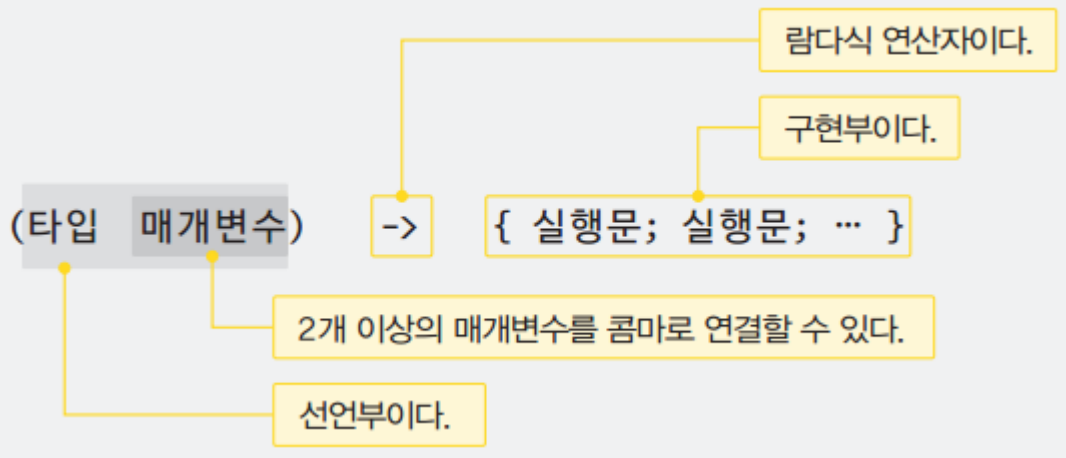
- 람다식
  - JDK 8부터 도입한 함수형 프로그래밍 기법 중 하나
  - 자바는 무명 구현 객체 대신에 람다식을 메서드의 인수로 사용하도록 허용
  - 람다식은 메서드를 감싼 무명 구현 객체를 자바가 전달할 수 있는 코드 블록처럼 흉내 낸 것
  - 자바는 무명 구현 객체를 람다식으로 표현해 함수형 프로그래밍을 모방한 것
  - 람다식을 사용하면 무명 객체보다 프로그램의 간결성 및 가독성 향상

- [예제 8-12]

```
3  import java.util.Arrays;
4
5  public class Lambda1Demo {
6      public static void main(String[] args) {
7          String[] strings = { "로마에 가면 로마법을 따르라.", "시간은 금이다.", "펜은 칼보다 강하다." };
8
9          Arrays.sort(strings, (first, second) -> first.length() - second.length());
10
11         for (String s : strings)
12             System.out.println(s);
13     }
14 }
```

```
시간은 금이다.
펜은 칼보다 강하다.
로마에 가면 로마법을 따르라.
```

람다식의 문법



- 매개변수 타입 생략 가능
- 매개변수가 하나 있다면 괄호를 생략 가능, 없으면 괄호가 꼭 필요
- 실행문이 하나 있다면 중괄호와 세미콜론을 생략 가능. 단, 실행문이 하나의 return 문이면 return 키워드도 생략
- 반환 타입을 표현하지 않음

- [예제 8-13]

```
3  interface Negative {
4      int neg(int x);
5  }
6
7  public class Lambda2Demo {
8      public static void main(String[] args) {
9          Negative n;
10         n = (int x) -> {
11             return -x;
12         };
13         n = (x) -> {
14             return -x;
15         };
16         n = x -> {
17             return -x;
18         };
19         n = (int x) -> -x;
20         n = (x) -> -x;
21         n = x -> -x;
22     }
23 }
```

- [예제 8-14]

```
3  interface Printable {
4      void print();
5  }
6
7  public class Lambda3Demo {
8      public static void main(String[] args) {
9          Printable p;
10         p = () -> {
11             System.out.println("안녕!");
12         };
13         p = () -> System.out.println("안녕!");
14         p.print();
15     }
16 }
```

```
안녕!
안녕!
```

람다식의 축약형 메서드 참조(1)

- 메서드 참조
  - 전달할 동작을 수행하는 메서드가 이미 정의된 경우에 표현할 수 있는 람다식의 축약형
  - 람다식에 아직 남아 있는 불필요한 정보까지 없애기 때문에 람다식에 비해 더욱 간결하고 가독성 제고

- 람다식과 대응하는 메서드 참조

종류	람다식	메서드 참조
정적 메서드 참조	(a) –〉 ClassName.staticMethod(a)	ClassName::staticMethod
인스턴스 메서드 참조	(a, b) –〉 a.instanceMethod(b) (단, a는 ClassName 타입이다.)	ClassName::instanceMethod
	(a) –〉 obj.instanceMethod(a)	obj::instanceMethod
생성자 참조	(a) –〉 new Constructor(a)	Constructor::new
배열 생성자 참조	(a) –〉 new Type[a]	Type::new

람다식의 축약형 메서드 참조(2)

- [예제 8-15], [예제 8-16]

```java
interface Showable {
    void show(String s);
}

interface Pickable {
    char pick(String s1, int i);
}

interface Operable {
    int operator(int x, int y);
}

interface Newable {
    String getString(String s);
}

interface IntArray {
    int[] getArray(int size);
}

class Utils {
    int add(int a, int b) {
        return a + b;
    }
}
```

```java
public class MethodRefDemo {
    public static void main(String[] args) {
        Showable s = System.out::println;
        // s = str -> System.out.println(str);
        s.show("잘있어");

        Utils u = new Utils();
        Operable o = u::add;
        // o = (x, y) -> u.add(x, y);
        System.out.println(o.operator(20, 30));

        Pickable p = String::charAt;
        // p = (x, y) -> x.charAt(y);
        System.out.println(p.pick("여보세요", 2));

        Newable n = String::new;
        // n = x -> new String(x);
        System.out.println(n.getString("사과"));

        IntArray a = int[]::new;
        // a = x -> new int[x];
        int[] array = a.getArray(2);
        array[0] = 0;
        array[1] = 1;
    }
}
```

```
잘있어
50
세
사과
```