

람다식과 함수형 인터페이스

예제 소스 코드는 파일과 연결되어 있습니다.
editplus(유료), notepad++(무료)와 같은 편집 도구를
미리 설치하여 PPT를 슬라이드 쇼로 진행할 때 소스
파일과 연결하여 보면 강의하실 때 편리합니다.

람다식 기초

■ 필요성

- 예제와 같이 Rectangle 클래스를 정의하면 사각형 객체끼리 비교할 수 없어 정렬할 수 없다.

```
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 String.format("사각형[폭=%d, 높이=%d]", width, height);
19    }
20 }
21 public class ComparableDemo {
22
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 }
33 }
34 }
```

람다식 기초

■ 정렬 메서드 구현

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

새로운 기준으로 비교한다면 관련 내용을 추가해야 한다.

(a) flag가 의미하는 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

■ 문제점

- 복잡하고 가독성이 떨어진다.
- Rectangle 클래스에 색상, 사각형 번호와 같은 다른 속성도 있다면 → 정렬 메서드의 수정 혹은 새로운 메서드 추가 필요

람다식 기초

■ 객체 비교 및 정렬

- 자바는 비교할 수 있는 객체 생성을 위해 Comparable 인터페이스를 제공

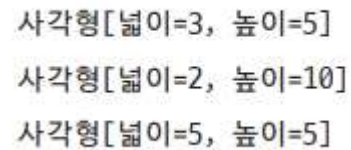
```
public interface Comparable <T> {  
    int compareTo(T o);  
}
```

- java.util 패키지의 Arrays 클래스는 sort()라는 정적 메서드를 제공

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

배열 원소가 Comparable 타입이어야 한다.

- 예제 : [sec01/ComparableDemo](#)



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

●

```

3 import java.util.Arrays;
4
5 class Rectangle implements Comparable<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 String.format("사각형[폭=%d, 높이=%d]", width, height);
19    }
20
21    public int compareTo(Rectangle o) {
22        return findArea() - o.findArea();
23    }
24 }
25
26 public class ComparableDemo {
27     public static void main(String[] args) {
28         Rectangle[] rectangles = { new Rectangle(3, 5),
29                                     new Rectangle(2, 10), new Rectangle(5, 5) };
30
31         Arrays.sort(rectangles);
32
33         for (Rectangle r : rectangles)
34             System.out.println(r);
35     }
36 }

```

● 그러나

- 객체끼리 비교할 기준이 여러 가지라면
- 비교 기준을 포함할 클래스가 최종 클래스라면

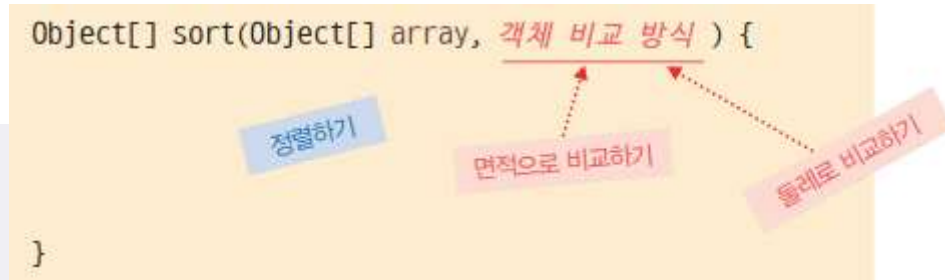
람다식 기초

■ 객체 비교 및 정렬

- Comparator 인터페이스와 sort()

```
public interface Comparator<T> {  
    int compare(T o1, T o2);  
}
```

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



- 예제 : [sec01/ComparatorDemo](#)

```

3 import java.util.Arrays;
4 import java.util.Comparator;
5
6
7 public class ComparatorDemo {
8     public static void main(String[] args) {
9         sort1();
10        lambdaSort();
11    }
12
13    static void sort1() {
14        String[] strings = { "로마에 가면 로마법을 따르라.",
15                             "시간은 금이다.", "펜은 칼보다 강하다." };
16
17        Arrays.sort(strings, new Comparator<String>() {
18            public int compare(String first, String second) {
19                return first.length() - second.length();
20            }
21        });
22
23        for (String s : strings)
24            System.out.println(s);
25    }
26
27    static void lambdaSort() {
28        String[] strings = { "로마에 가면 로마법을 따르라.",
29                             "시간은 금이다.", "펜은 칼보다 강하다." };
30
31        Arrays.sort(strings, (first, second) -> first.length() - second.length() );
32
33
34        for (String s : strings)
35            System.out.println(s);
36    }
37 }

```

람다식 기초

■ 람다식 의미와 문법

- 메서드를 포함하는 익명 구현 객체를 전달할 수 있는 코드
- 특징
 - 메서드와 달리 이름이 없다.
 - 메서드와 달리 특정 클래스에 종속되지 않지만, 매개변수, 반환 타입, 본체를 가지며, 심지어 예외도 처리할 수 있다.
 - 메서드의 인수로 전달될 수도 있고 변수에 대입될 수 있다.
 - 익명 구현 객체와 달리 메서드의 핵심 부분만 포함한다

● 문법



- 예제 : [sec01/Lambda1Demo](#), [sec01/Lambda2Demo](#)

람다식 기초

■ 메서드 참조

- 전달할 동작을 수행하는 메서드가 이미 정의된 경우에 표현할 수 있는 람다식의 축약형
- 메서드 참조의 종류와 표현 방식

종류	표현 방식
정적 메서드 참조	클래스이름::정적메서드
인스턴스 메서드 참조	객체이름::인스턴스메서드(혹은 클래스이름::인스턴스메서드)
생성자 참조	클래스이름::new 혹은 배열타입이름::new

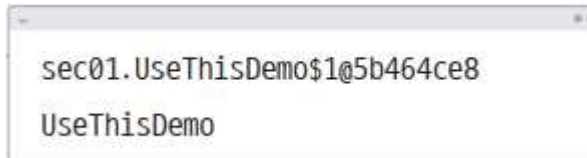
- 예제 : [sec01/MethodRefDemo](#), [sec01/ConstructorRefDemo](#)

람다식 유의 사항과 활용

■ 람다식 유의 사항

- 람다식 외부에서 선언된 변수와 동일한 이름의 변수를 람다식에서 선언할 수 없다.
- 람다식에 사용된 지역변수는 final이다.
- 람다식의 this 키워드는 람다식을 실행한 외부 객체를 의미한다.

- 예제 : [sec02/UseThisDemo](#)



```
sec01.UseThisDemo$1@5b464ce8
UseThisDemo
```

람다식 유의 사항과 활용

■ 람다식 활용

- 다음과 같은 문제를 해결
 - 디젤 자동차만 모두 찾아보자.
 - 10년보다 오래된 자동차만 모두 찾아보자.
 - 10년보다 오래된 디젤 자동차만 모두 찾아보자.
 - 디젤 자동차를 출력하되 모델과 연식만 나타나도록 출력하자.
 - 10년보다 오래된 자동차를 출력하되 모델, 연식, 주행거리만 나타나도록 출력하자.

- 필요한 인터페이스 : [sec02/CarPredicate](#), [sec02/CarConsumer](#)

- 문제 해결을 위하여 필요한 메서드

```
List<Car> findCars(List<Car> all, CarPredicate p)
```

```
void printCars(List<Car> all, CarConsumer c)
```

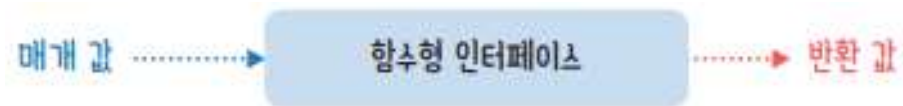


- 예제 : [sec02/Car](#), [sec02/CarDemo](#)

함수형 인터페이스 응용

■ 함수형 인터페이스

- 의미 : 추상 메서드가 1개만 있는 인터페이스
- 분류



- java.util.function 패키지가 제공하는 함수형 인터페이스 종류

종류	매개 값	반환 값	메서드	의미
Predicate	있음	boolean	test()	매개 값을 조사하여 논릿값으로 보낸다.
Consumer	있음	void	accept()	매개 값을 소비한다.
Supplier	없음	있음	get()	반환 값을 공급한다.
Function	있음	있음	apply()	매개 값을 반환 값으로 매핑한다.
Operator	있음	있음	apply()	매개 값을 연산하여 반환 값으로 보낸다.

- 이외에도 BiPredicate와 같은 다양한 변종도 있음
- java.util.function 패키지가 제공하는 함수형 인터페이스는 추상 메서드 외 다양한 디폴트 메서드나 정적 메서드도 제공

함수형 인터페이스 응용

■ Predicate 인터페이스 유형



- Bi, Double, Int, Long을 접두어로 붙인 변종이 있다.
- Predicate 유형은 다음과 같이 정의

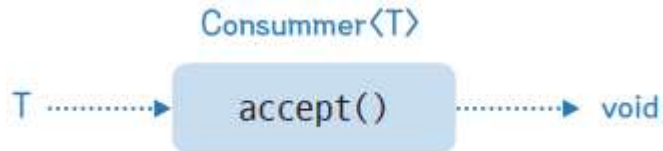
```
Predicate<T> p = t -> { T 타입 t 객체를 조사하여 논릿값으로 반환하는 실행문; };
```

- 예제 : [sec03/PredicateDemo](#)

```
홀수
1 혹은 짝수
true
false
false
```

함수형 인터페이스 응용

■ Consumer 인터페이스 유형



- Bi, Double, Int, Long, ObjDouble, ObjInt, ObjLong를 접두어로 붙인 변종이 있다.
- Consumer 유형은 다음과 같이 정의

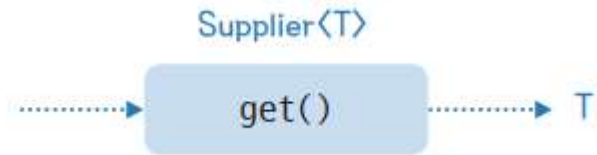
```
Consumer<T> c = t -> { T 타입 t 객체를 사용한 후 void를 반환하는 실행문; };
```

- 예제 : [sec03/ConsumerDemo](#)

```
java functional interface
Java : Lambda
150
10 * 10 = 100
10 + 10 = 20
```

함수형 인터페이스 응용

■ Supplier 인터페이스 유형



- Double, Int 등을 접두어로 붙인 변종이 있다.
- Supplier 유형은 다음과 같이 정의

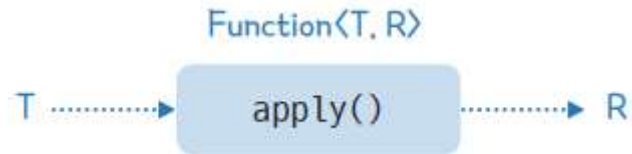
```
Supplier<T> s = () -> { T 타입 t 객체를 반환하는 실행문; };
```

- 예제 : [sec03/SupplierDemo](#)

```
apple
0
1
2
9.39749695524083
02월 05일(수요일) 오후 02:58:19
```

함수형 인터페이스 응용

■ Function 인터페이스 유형



- Bi, Double, IntToDouble, ToDoubleBi 등을 접두어로 붙인 변종이 있다.
- Function 유형은 다음과 같이 정의

```
Function<T, R> f = t -> { T 타입 t 객체를 사용하여 R 타입 객체를 반환하는 실행문; };
```

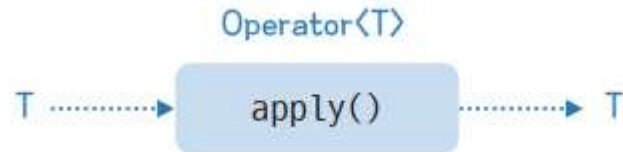
- 예제 : [sec03/Function1Demo](#), [sec03/Function2Demo](#)

5
6
10
8
2.5
78.5

(소나타, 18) (코란도, 15) (그랜저, 12) (싼타페, 10) (아반테, 10) (에쿠스, 6) (그랜저, 5) (소나타, 2) (쏘렌토, 1) (아반테, 1)
평균 연식 = 8.0
평균 주행거리 = 108200.0

함수형 인터페이스 응용

■ Operator 인터페이스 유형



- Operator라는 인터페이스는 없고 Binary, Unary, Double, Int, Long을 접두어로 붙인 변종만 있다.
- BinaryOperator 인터페이스는 다음과 같이 정의

```
BinaryOperator<T> o = (x, y) -> { T 타입 x와 y 객체를 사용하여 T 타입을 반환하는 실행문; };
```

- 예제 : [sec03/Operator1Demo](#), [sec03/Operator2Demo](#)

```
5
5
(3 + 2) * 2 = 10
3
[15, 16, 17]
```

```
10
25
5
20
[Car(뉴소나타, true, 18, 210000), Car(뉴코란도, false, 15, 200000), Car(뉴그랜저, true, 12, 150000), Car(뉴싼타페, false, 10, 220000), Car(뉴아반테, true, 10, 70000), Car(뉴에쿠스, true, 6, 100000), Car(뉴그랜저, true, 5, 80000), Car(뉴소나타, true, 2, 35000), Car(뉴쏘렌토, false, 1, 10000), Car(뉴아반테, true, 1, 7000)]
```

함수형 인터페이스 응용

■ Comparator 인터페이스

- 객체에 순서를 정하기 위하여 사용되는 함수형 인터페이스
- `compare()`라는 추상 메서드 외에도 유용한 정적 메서드와 디폴트 메서드를 제공하며, 메서드의 반환 타입은 모두 `Comparator<T>` 타입

정적 메서드	의미
<code>comparing()</code>	Comparable 타입의 정렬 키로 비교하는 Comparator를 반환한다.
<code>naturalOrder()</code>	Comparable 객체에 자연 순서로 비교하는 Comparator를 반환한다.
<code>nullsFirst()</code>	null을 객체보다 작은 값으로 취급하는 Comparator를 반환한다.
<code>nullsLast()</code>	null을 객체보다 큰 값으로 취급하는 Comparator를 반환한다.
<code>reverseOrder()</code>	자연 반대 순서로 비교하는 Comparator를 반환한다.

디폴트 메서드	의미
<code>reversed()</code>	현재 Comparator의 역순으로 비교하는 Comparator를 반환한다.
<code>thenComparing()</code>	다중 키를 사용하여 정렬하려고 새로운 Comparator를 반환한다.

함수형 인터페이스 응용

■ Comparator 인터페이스

- 예제 :
[sec03/Comparator1Demo](#)

```
[Car(소나타, true, 18, 210000), Car(코란도, false, 15, 200000), Car(그랜저, true, 12, 150000)]  
[Car(그랜저, true, 12, 150000), Car(소나타, true, 18, 210000), Car(코란도, false, 15, 200000)]  
[Car(코란도, false, 15, 200000), Car(소나타, true, 18, 210000), Car(그랜저, true, 12, 150000)]  
[Car(그랜저, true, 12, 150000), Car(코란도, false, 15, 200000), Car(소나타, true, 18, 210000)]  
[Car(소나타, true, 18, 210000), Car(코란도, false, 15, 200000), Car(그랜저, true, 12, 150000)]
```

- 예제 :
[sec03/Comparator2Demo](#)

```
[Car(소나타, true, 18, 210000), Car(코란도, false, 15, 200000), Car(그랜저, true, 12, 150000), null]  
[null, Car(그랜저, true, 12, 150000), Car(소나타, true, 18, 210000), Car(코란도, false, 15, 200000)]  
[Car(소나타, true, 18, 210000), Car(코란도, false, 15, 200000), Car(코란도, false, 10, 220000)]  
[Car(소나타, true, 18, 210000), Car(코란도, false, 10, 220000), Car(코란도, false, 15, 200000)]
```