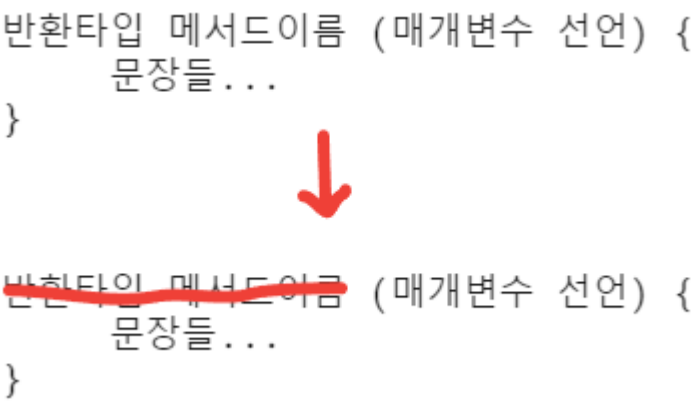
# 14. 람다식과 익명함수

Sorting(정렬)

## ▼ 0. 람다식의 개요

- 람다식이란 쉽게 말해 메서드를"하나의 식"으로 표현한 것입니다.
- 하나의 식으로 표현하여 훨씬 간략하게 표현이 가능하게 되며, 메서드의 이름과 반환값이 없어지므로"익명함수"라고도 합니다.

```
int[] arr = new int[5]
Arrays.setAll(arr, () → (int) (Math.random() * 5) + 1);

// (i) - (int) (Math.random() * 5) + 1 람다식을 메서드로 표현
int method() {
    return (int) (Math.random() * 5) + 1;
}
```




| 메소드 | 람다식 |
| --- | --- |
| void printVal(String name, int i) {<br>System.out.println(name + "=" + i);<br>} | (name, i) → System.out.println(name + "=" + i) |
| int square(int x) { return x * x} | **x → x * x** |

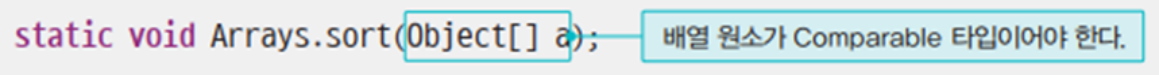
## ▼ 1. 람다식의 기초

- 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



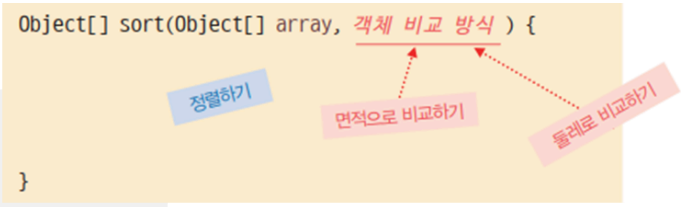

```
import java.util.Arrays;

class Rectangle implements Comparable<Rectangle> {
    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 String.format("사각형[폭=%d, 높이=%d]", width, height);
    }
        @Override
    public int compareTo(Rectangle o) {
        return findArea() - o.findArea();
    }
}

public class ComparableDemo {
    public static void main(String[] args) {
        Rectangle[] rectangles = { new Rectangle(3, 5),
                new Rectangle(2, 10), new Rectangle(5, 5) };

        Arrays.sort(rectangles);

        for (Rectangle r : rectangles)
            System.out.println(r);
    }
}
```

- 그러나  다음의 경우를 생각해 보자.....
  - 객체끼리 비교할 기준이 여러 가지라면
  - 비교 기준을 포함할 클래스가 최종 클래스라면
  - ***Comparator 인터페이스***....
    - 객체들을 비교하는 로직을 정의하는 인터페이스입니다.
    - 주로 객체 정렬 시 사용되며,

- Comparable과 달리 객체 자체에 정렬 기준을 구현하지 않고 외부에서 비교 로직을 제공
- Comparator는 두 개의 객체를 입력 받아 비교하고, 결과에 따라 정렬 순서를 결정합니다.

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

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



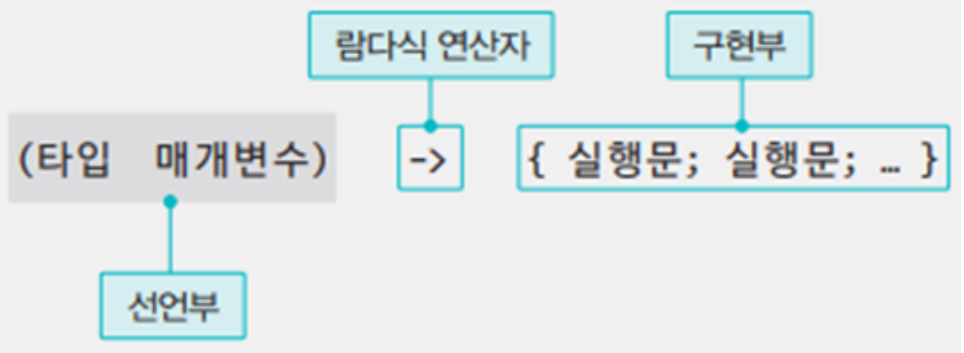

```
import java.util.Arrays;
import java.util.Comparator;

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);
    }
}
```

## ▼ 2. 람다식의 의미와 문법

- 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




```
import java.util.Arrays;

public class Lambda1Demo {
    public static void main(String[] args) {
        String[] strings = { "로마에 가면 로마법을 따르라.", "시간은 금이다.", "펜은 칼

        Arrays.sort(strings, (first, second) → first.length() - second.length());

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

```
interface Negative {
    int neg(int x);
}

interface Printable {
    void print();
}
```

```
public class Lambda2Demo {
    public static void main(String[] args) {
        Negative n;
        n = (int x) -> { return -x; };

        n = (x) -> { return -x; };

        n = x -> { return -x; };

        n = (int x) -> -x;

        n = (x) -> -x;

        n = x -> -x;

        Printable p;

        p = () -> { System.out.println("안녕!"); };

        p = () -> System.out.println("안녕!");

        p.print();
    }
}
```

- **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 |

```
interface IMathematical {
    double calculate(double d);
}

interface IPickable {
    char pick(String s, int i);
}

interface IComputable {
    int compute(int x, int y);
}

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

//interface Applyable {
//    boolean apply(Box a, Box b);
//}
//
//class Box {
//    int no;
//
//    public Box(int no) {
//        this.no = no;
//    }
//
//    boolean isSame(Box b) {
//        return this.no == b.no;
//    }
//}

public class MethodRefDemo {
    public static void main(String[] args) {
        IMathematical m;
        IPickable p;
```

```
        IComputable c;

//        m = d -> Math.abs(d);
        m = Math::abs;
        System.out.println(m.calculate(-50.3));

//        p = (a, b) -> a.charAt(b);
        p = String::charAt;
        System.out.println(p.pick("안녕, 인스턴스 메서드 참조!", 4));

        Utils utils = new Utils();
//        c = (a, b) -> utils.add(a, b);
        c = utils::add;
        System.out.println(c.compute(20, 30));

//        Applyable app;
////        app = (a, b) -> a.isSame(b);
//        app = Box::isSame;
//        System.out.println(app.apply(new Box(1), new Box(2)));
//        System.out.println(app.apply(new Box(1), new Box(1)));
    }
}
```

- 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.

```
interface UseThis {  void use();  }

public class UseThisDemo {
    public void lambda() {
        String hi = "Hi!";

        UseThis u1 = new UseThis() {
            public void use() {
```

```
            System.out.println(this);
//              hi = hi + " Lambda.";
          }
        };
        u1.use();

        UseThis u2 = () -> {
          System.out.println(this);
//              hi = hi + " Lambda.";
        };
        u2.use();
    }

    public String toString() {
        return "UseThisDemo";
    }

    public static void main(String[] args) {
        int one = 1;
        new UseThisDemo().lambda();
//      Comparator<String> c = (one, two) -> one.length() - two.length();
    }
}
```

---

[람다식의 활용]

---

[함수형 인터페이스]