메소드 참조

메소드 참조

- 메소드 참조(Method References)의 이해
  - 람다식은 결국 메소드의 정의
  - 이미 정의되어 있는 메소드가 있다면, 이 메소드의 정의가 람다식을 대신할 수 있음

- 메소드 참조(Method References)의 이해
  - 메소드 참조의 4가지 유형
    - static 메소드의 참조
    - 참조변수를 통한 인스턴스 메소드 참조
    - 클래스 이름을 통한 인스턴스 메소드 참조
    - 생성자 참조
  - 메소드 참조를 통해 코드를 작성하면 비교적 가독성이 떨어지지만 코드의 양이 많이 줄어들게 됨

메소드 참조

- Static 메소드의 참조
  - 아래는 람다식을 기반으로 작성되었으며, 컬렉션 인스턴스에 저장된 인스턴스의 저장 순서를 뒤집는 코드

```
public class ArrangeList {
    public static void main(String[] args) {
        List<Integer> ls = Arrays.asList(1, 3, 5, 7, 9);
        ls = new ArrayList<>(ls);
        Consumer<List<Integer>> c = l ->
Collections.reverse(l);
        c.accept(ls);   // 순서 뒤집기 실행
        System.out.println(ls);
    }
}
```

```
public class ArrangeList {
    public static void main(String[] args) {
        List<Integer> ls = Arrays.asList(1, 3, 5, 7, 9);
        ls = new ArrayList<>(ls);
        Consumer<List<Integer>> c = Collections::reverse;
        c.accept(ls);   // 순서 뒤집기 실행
        System.out.println(ls);
    }
}
```

  - 람다식 "Consumer<List<Integer>> c = l -> Collections.reverse(l);"를 통해 순서를 뒤집는 기능 구현
  - reverse 메소드는 Collections 클래스의 static 메소드로 위 람다식은 "ClassName::staticMethodName" 규칙의 메소드 참조에 의해 "Consumer<List<Integer>> c =Collections::reverse;"와 같이 대체할 수 있음
  - 위 메소드 참조에서 람다식에 있는 인자 전달에 대한 정보를 생략할 수 있는 이유는 "accept 메소드 호출 시 전달되는 인자를 reverse 메소드를 호출하면서 그대로 전달한다."라는 약속에 근거

메소드 참조

- 인스턴스 메소드의 참조 1 : 인스턴스가 존재하는 상황에서 참조
  - static 메소드를 참조하였듯이 인스턴스 메소드도 참조 가능

```
public class JustSort {
    public void sort(List<?> list) {
        Collections.reverse(list);
    }
}
```

```
public class ArrangeList1 {
    public static void main(String[] args) {
        List<Integer> ls = Arrays.asList(1, 3, 5, 7, 9);
        ls = new ArrayList<>(ls);
        JustSort js = new JustSort();
        Consumer<List<Integer>> c = e -> js.sort(e);
        c.accept(ls);
        System.out.println(ls);
    }
}
```

  - 위 코드의 람다식에서 같은 지역 내에 선언된 참조변수 js에 접근하고 있음
  - 람다식에서 같은 지역에 선언된 참조변수에 접근하려면 "람다식에서 접근 가능한 참조변수는 final로 선언되었거나 effectively final(초기화 후 수정되지 않기 때문에 사실상 final 선언이 된 것과 다름이 없음)이어야 한다"라는 조건을 만족해야 가능
  - 위 코드에서 si가 참조하는 대상을 수정하지 않기 때문에 참조변수 js는 effectively final 이라 할 수 있음
  - 만약 중간에 "js = new JestSort();"를 한 번 더 실행하거나 코드의 가장 마지막에 "js = null" 추가하면 오류 발생
  - 위 코드의 람다식은 "ReferenceName::instanceMethodName" 규칙의 메소드 참조에 의해 "Consumer<List<Integer>> c = js::sort"와 같이 대체할 수 있음

메소드 참조

```
public class ForEachDemo {
    public static void main(String[] args) {
        List<String> ls = Arrays.asList("Box", "Robot");
        ls.forEach(s -> System.out.println(s)); // 람다식 기반
        ls.forEach(System.out::println);     // 메소드 참조 기반
    }
}
```

- Collection<E> 인터페이스는 Iterable<T>를 상속하므로 컬렉션 클래스들은 Iterable<T>를 구현하는데, 이 인터페이스에는 다음 디폴트 메소드가 정의되어 있음

```
default void forEach(Consumer<? super T> action) {
      for (T t : this)// this는 이 메소드가 속한 컬렉션 인스턴스를 의미함
            action.accept(t);
}
```

- 즉, 위 메소드가 호출되면 컬렉션 인스턴스에 저장되어 있는 모든 인스턴스들을 대상으로 "action.accept(t)"를 실행하므로 forEach 메소드 호출을 위해 Consumer<T> 인터페이스에 대한 람다식 또는 메소드 참조를 전달해야 함.
- Consumer<T>의 추상 메소드는 "void accept(T t)"이며 이 메소드는 반환하지 않고, 전달된 인자를 대상으로 어떤 결과를 보이도록 구성되어 있음
- System.out.println 메소드는 "public void println(String x)"로 정의되어 있으며 이 위 accept 메소드에 딱 어울리는 메소드라는 것을 확인할 수 있음

메소드 참조

- 인스턴스 메소드의 참조 2 : 인스턴스 없이 인스턴스 메소드 참조
  - 이번 내용은 내용이 다소 난해하므로 단순하게 받아들여야 함

```
public class IBox {
    private int n;
    public IBox(int n) {
        this.n = n;
    }
    public int larger(IBox b) {
        if (this.n > b.n) {
            return this.n;
        } else {
            return b.n;
        }
    }
}
```

```
public class NoObjectMethodRef {
    public static void main(String[] args) {
        IBox ib1 = new IBox(5);
        IBox ib2 = new IBox(7);
        // 두 상자에 저장된 값을 비교하여 더 큰 값 반환
        ToIntBiFunction<IBox, IBox> bf = (b1, b2) ->
b1.larger(b2);
        int bigNum = bf.applyAsInt(ib1, ib2);
        System.out.println(bigNum);
    }
}
```

  - 위 코드에서 등장한 람다식은 "ToIntBiFunction<IBox, IBox> bf = (b1, b2) -> b1.larger(b2);"이고 호출하는 메소드 larger가 첫 번째 인자로 전달된 인스턴스의 메소드인 경우
  - 위의 경우 람다식은 "ClassName::instanceMethodName" 규칙의 메소드 참조에 의해 "ToIntBiFunction<IBox, IBox> bf = IBox::larger"와 같이 대체할 수 있음

메소드 참조

- 이렇게 막 줄여도 되나요? 왜 가능해요?
  - "ToIntBiFunction<IBox, IBox> bf = IBox::larger" 문장 이후 "bf.applyAsInt(ib1, ib2)"와 같이 메소드 호출
  - 이때 bf가 참조하는 메소드는 IBox::larger이고 ib1도 ib2도 가지고 있는 인스턴스 메소드
  - 첫 번째 전달인자를 대상으로 이 메소드를 호출하기로 약속하였으므로 ib1.larger(ib2);가 실행
  - 모든 생략은 약속에 근거하며 위 생략이 가능한 이유는 "첫 번째 인자로 전달된 인스턴스의 메소드"라고 약속되어 있기 때문
  - 즉 위의 내용이 적용되는 경우에만 메소드 참조 적용 가능

메소드 참조

1. 다음 코드를 메소드 참조 방식으로 수정해보자. 참고로 Collections.sort 메소드가 "public static <T> void sort(List<T> list, Comparator<? super T> c)"와 같으니 "Comparator<? super T> c = (s1, s2) -> sq.compareToIgnoreCase(s2)" 문장을 메소드 참조 기반으로 수정한다고 생각하면 편하다.

```
public class StrIgnoreCaseComp {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("robot");
        list.add("Lambda");
        list.add("box");
        Collections.sort(list, (s1, s2) -> s1.compareToIgnoreCase(s2));
        System.out.println(list);
    }
}
```