아이템 64 객체는 인터페이스를 사용해 참조하라

▼ 아이템 51에서 매개변수 타입으로 클래스가 아니라 인터페이스를 사용하라 라고 했다.

아이템 51 메서드 시그니처를 신중히 설계하라

1. 메서드 이름을 신중히 짓자
2. 편의 메서드를 너무 많이 만들지 말자
3. 매개변수 목록은 짧게 유지하자
4. **매개변수의 타입으로는 클래스보다는 인터페이스가 더 낫다**.
5. boolean 보다는 원소 2개짜리 열거 타입이 낫다.

⇒ **객체는 클래스가 아닌 인터페이스로 참조하라** 로 확장될 수 있다.

적합한 인터페이스만 있다면 매개변수뿐 아니라 반환값, 변수, 필드를 전부 인터페이스 타입으로 선언하라.

```
//좋은예 : 인터페이스를 타입으로 사용
Set<Son> sonSet = new LinkedHashSet<>();
//나쁜예 : 클래스를 타입으로 사용
LinkedHashSet<Son> sonSet = new LinkedHashSet<>();
```

- 인터페이스를 타입으로 사용하면 프로그램이 훨씬 유연해 진다.
  - 나중에 구현 클래스를 교체하고자 할 때, 새 클래스의 생성자(혹은 다른 정적 팩터리)를 호출해주기만 하면 된다.

```
public interface Pay {
    void payByCash();
    void payByCreditCard();
}
public static void main(String[] args) {
//        Pay payment = new KakaoPay();
        Pay payment = new NaverPay();
        payment.payByCash();
        payment.payByCreditCard();
//        Kakao Pay by cash
//        Kakao Pay by cash

//        Naver Pay by cash
//        Naver Pay by credit card
    }
```

주의할 점

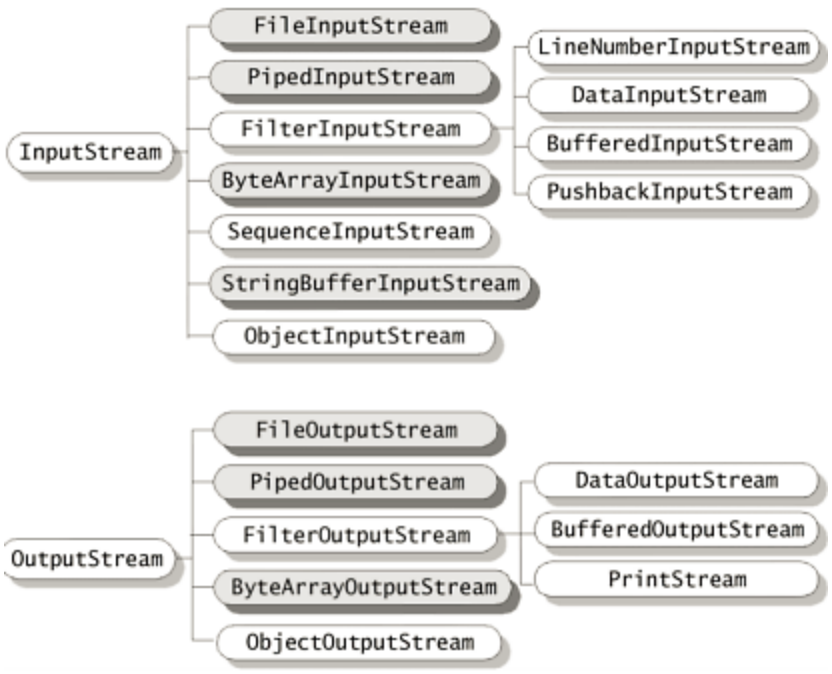
- 원래의 클래스가 인터페이스의 일반 규약 이외의 특별한 기능을 제공하며, 주변 코드가 이 기능에 기대어 동작한다면 새로운 클래스도 반드시 같은 기능을 제공해야 한다.
- 예를 들어 **LinkedHashSet**이 따르는 순서 정책을 가정하고 동작하는 상황에서 이를 (반복자의 순회 순서를 보장하지 않는) **HashSet**으로 바꾸면 문제가 될 수 있다.
- 변수를 구현타입으로 선언하면 프로그램이 컴파일되지 않을 수 있다.
  - 클라이언트에서 기존 타입에서만 제공하는 메서드를 사용했거나 기존 타입을 사용해야 하는 경우 다른 메서드에 그 인스턴스를 넘겼을 때 새로운 코드에서는 컴파일 되지 않는다.

왜 구현타입을 바꾸려 할까

- 원래 것보다 성능이 좋거나 멋진 신기능을 제공하기 때문
  - HashMap을 참조하던 변수를 (키가 열거타입일때만) EnumMap으로 바꾸면 속도가 빨라지고 순회 순서도 키의 순서와 같아진다.
    LinkedHashMap으로 바꾸면 성능은 비슷하게 유지하면서 순회 순서를 예측할 수 있다.

적합한 인터페이스가 없다면 당연히 클래스로 참조해야 한다.

1. String 과 BigInteger 같은 값 클래스를 여러가지로 구현될 수 있다고 생각하고 설계하는 일은 거의 없다.
    - final 인 경우가 많고 상응하는 인터페이스가 별도로 존재하는 경우도 드물다
    - 이런 값 클래스는 매개변수, 변수필드, 변환 타입으로 사용해도 무방하다
2. 클래스 기반으로 작성된 프레임워크가 제공하는 객체들
    - 특정 구현 클래스보다는 (보통 추상 클래스인) 기반 클래스를 사용해 참조하는게 좋다.



- OutputStream 등..

3. 인터페이스에는 없는 특별한 메서드를 제공하는 클래스들
    - PriorityQueue 클래스는 Queue 인터페이스에는 없는 comparator 메서드를 제공한다.

Method Summary

Methods

Modifier and Type	Method and Description
boolean	`add(E e)` Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning `true` upon success and throwing an `IllegalStateException` if no space is currently available.
E	`element()` Retrieves, but does not remove, the head of this queue.
boolean	`offer(E e)` Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	`peek()` Retrieves, but does not remove, the head of this queue, or returns `null` if this queue is empty.
E	`poll()` Retrieves and removes the head of this queue, or returns `null` if this queue is empty.
E	`remove()` Retrieves and removes the head of this queue.

Queue Interface

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
boolean	add(E e)	Inserts the specified element into this priority queue.
void	clear()	Removes all of the elements from this priority queue.
Comparator<? super E>	comparator()	Returns the comparator used to order the elements in this queue, or null if this queue is sorted according to the natural ordering of its elements.
boolean	contains(Object o)	Returns true if this queue contains the specified element.
void	forEach(Consumer<? super E> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
Iterator<E>	iterator()	Returns an iterator over the elements in this queue.
boolean	offer(E e)	Inserts the specified element into this priority queue.
boolean	remove(Object o)	Removes a single instance of the specified element from this queue, if it is present.
boolean	removeAll(Collection<?> c)	Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	removeIf(Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
boolean	retainAll(Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
Spliterator<E>	spliterator()	Creates a *late-binding* and *fail-fast* Spliterator over the elements in this queue.
Object[]	toArray()	Returns an array containing all of the elements in this queue.
<T> T[]	toArray(T[] a)	Returns an array containing all of the elements in this queue; the runtime type of the returned array is that of the specified array.

PriorityQueue

정리

적합한 인터페이스가 없다면 클래스의 계층구조 중 필요한 기능을 만족하는 가장 덜 구체적인(상위의) 클래스를 타입으로 사용하자