1. 람다식

- 자바는 함수적 프로그래밍을 위해 자바8부터 람다식(Lambda Expressions)을 지원함.
- 자바에서 람다식을 수용한 이유는 자바 코드가 매우 간결해지고, 컬렉션의 요소를 필터링하거나 매핑해서 원하는 결과를 쉽게 집계할 수 있기 때문이다.
- 람다식의 형태는 매개 변수를 가진 코드 블록이지만, 런타임 시에는 익명 구현 객체를 생성한다.

2. 람다식 작성 방법

```
(타입 매개변수, ...) -> {
   실행문;
   ...
};
```

- 매개변수의 이름은 개발자가 자유롭게 줄 수 있다.
- 매개 변수 타입은 런타임 시에 대입되는 값에 따라 자동으로 인식될 수 있기 때문에 람다식에서는 매개 변수의 타입을 일반적으로 언급하지 않는다.

```
(매개변수, ...) -> {
   실행문;
   ...
};
```

- 하나의 매개 변수만 있다면 괄호 ( )를 생략할 수 있다.
- 하나의 실행문만 있다면 중괄화 { }도 생략할 수 있다.

```
매개변수 -> 실행문;
```

- 만약 매개 변수가 없다면 빈 괄호( )를 반드시 사용해야 한다.

```
( ) -> {
   실행문;
   ...
};
```

- 중괄호 { }를 실행하고 결과값을 리턴해야 한다면 return문으로 결과 값을 지정할 수 있다.

```
(x, y) -> { return x + y; };
```

- 중괄호 { }에 return문만 있을 경우 람다식에서는 return 문을 생략할 수 있다.

```
(x, y) -> x + y;
```

3. 타겟 타입

- 람다식이 대입될 인터페이스를 타겟 타입(Target Type)이라고 한다.

4. 함수적 인터페이스(@FunctionalInterface)

- 람다식은 하나의 메소드를 정의하기 때문에 두 개 이상의 추상 메소드가 선언된 인터페이스는 람다식으로 구현 객체를 생성할 수 없다.
- 따라서 인터페이스에서 실수로 두 개 이상의 추상 메소드를 선언하는 것을 방지하고자 할 때 @FunctionalInterface 어노테이션을 붙여 주는 것이 좋다.

```java
@FunctionalInterface
public interface TestInterface{
    public void interfaceMethod1();
    //public void interfaceMethod2();
}
```

```java
@FunctionalInterface
public interface TestInterface{
    public void interfaceMethod1();
    public void interfaceMethod2();
}
```

5. this

- 일반적으로 익명 객체 내부에서 this는 익명 객체의 참조이지만, 람다식에서 this는 람다식을 실행한 객체의 참조이다.

6. 로컬 변수 사용

- 메소드 내에서 사용되는 람다식은 메소드의 로컬 변수를 사용할 때 해당 변수는 final 특성을 갖는다. -> 익명 객체 특성

익명 객체 내부에서는 바깥 클래스의 필드나 메소드는 제한 없이 사용할 수 있다. 하지만 메소드의 매개 변수나 로컬 변수를 익명 객체에서 사용할 때는 메소드 내에서 생성된 익명 객체는 메소드 실행이 끝나도 힙 메모리에 존재한다. 따라서 메소드 내 매개 변수나 로컬 변수는 메소드 실행이 끝나면 스택 메모리에서 사라지기 때문에 익명 객체에서 사용할 수 없다. 따라서 메소드의 매개 변수나 로컬 변수를 익명 객체 내부에서 사용할 때 해당 변수들은 final 특성을 가져야 한다. 자바 7까지는 final 키워드로 선언해야 했지만 자바 8이후로는 final 키워드를 생략해도 된다. 컴파일 시 final 키워드가 있다면 메소드 내부에 지역 변수로 복사되지만 final 키워드를 붙이지 않는다면 익명 클래스의 필드로 복사된다.

7. 표준 API의 함수적 인터페이스

- 자바 8부터는 빈번하게 사용되는 함수적 인터페이스는 java.util.function 표준 API 패키지로 제공한다.

종류	추상 메소드 특징
Consumer	매개값은 있고, 리턴값은 없음
Supplier	매개값은 없고, 리턴값은 있음
Function	- 매개값도 있고, 리턴값도 있음 - 주로 매개값을 리턴값으로 매핑(타입 변환)
Operator	- 매개값도 있고, 리턴값도 있음 - 주로 매개값을 연산하고 결과를 리턴
Predicate	- 매개값은 있고, 리턴 타입은 boolean - 매개값을 조사해서 true/false를 리턴

8. andThen(), compose()

- 함수적 인터페이스는 andThen()과 compose() 디폴트 메소드를 가지고 있다.
- 두 메소드는 두 개의 함수적 인터페이스를 순차적으로 연결하고 첫 번째 처리 값을 두 번째 매개 값으로 제공하여 최종 결과 값을 얻을 때 사용한다.

andThen()	compose()
interfaceA.andThen(interfaceB) 인터페이 A실행 후 매개 값을 인터페이스 B로 준 후 인터페이스 B실행	interfaceA.compose(interfaceB) 인터페이 B실행 후 매개 값을 인터페이스 A로 준 후 인터페이스 A실행

- andThen()과 compose