2회차 : 람다식과 자바 API의 함수적 인터페이스

날짜	@2023년 6월 19일

람다식과 자바API의 함수적 인터페이스

람다식의 개념 및 기본 문법

람다식

정의 : 자바에서 함수적 프로그래밍 지원 기법

장점 : 코드의 간결화 및 병렬처리에 강함 (Collection API 성능 효과적 개선 (Stream))

기본 용어

- 함수 : 기능, 동작을 정의
- 메서드 : 클래스 또는 인터페이스 내부에서 정의된 함수
- 함수형 인터페이스 : 내부에 단 1개의 추상메서드만 존재하는 인터페이스

```
//함수
void abc() {
}

//메서드
class A{
  void abc() {
  }
}

//함수형 인터페이스
interface A{
  public abstract void abc();
}
```

💡 **문법적인 의미**에서, 람다식은 **익명이너클래스의 약식 표현**

```
interface A {
  void abc();
}

class B implements A{
  public void abc(){
    System.out.println("Hi");
  }
}

class ex01 {
  public static void main(String[] args){
    //basic
    A a = new B();
    a.abc();  //Hi

    //익명이너클래스
    A b = new A(){
      void abc(){
        System.out.println("Hi");
      }
    };
    b.abc();  //Hi

    //람다식
    A c = ()->{
            System.out.println("Hi");
          };
    c.abc();  //Hi
  }
}
```

람다식 변환 방법 (“→” 람다식 기호)

~~리턴타입 메서드명~~ (매개변수) {

    //메서드 내용

}

⇒ (매개변수)→{ //메서드 내용 }

```
//리턴X, 매개변수X
void method1() {
    System.out.println(3);
```

```
}

()->{System.out.println(3);}


//리턴X, 매개변수O
void method2(int a) {
    System.out.println(a);
}

(int a)->{System.out.println(a);}


//리턴O, 매개변수X
int method3() {
    return 5;
}

()->{return 5;}


//리턴O, 매개변수O
double method4(int a, double b) {
    return a + b;
}

(int a, double b)->(return a + b;)
```

람다식 약식표현

- 실행문이 하나인 경우 중괄호 생략가능
- 매개변수 타입 생략 가능
- 매개변수가 한 개인 경우 소괄호 생략 가능 (소괄호 생략시 매개변수 타입 반드시 생략)
- 실행문이 리턴만 있는 경우 return 생략가능 (return 생략시 중괄호 반드시 생략)

람다식의 세 가지 활용

활용#1. 익명이너클래스 내 구현 메서드의 약식(람다식) 표현 (**함수형 인터페이스**만 가능)

```
interface A{
    void method1();
}
```

```
interface B{
    void method2(int a);
}

interface C{
    int method3();
}

interface D{
    double method4(int a, double b);
}

class ex01 {
  public static void main(String[] args){
    //#1. 입력X, 출력X 인 함수
    //익명이너클래스 표현
    A a1 = new A() {
        @Override
        public void method1() {
            System.out.println("입력X, 출력X 인 함수");
        }
    };
    //람다식 표현
    A a2 = ()->{System.out.println("입력X, 출력X 인 함수");};
    A a3 = ()->System.out.println("입력X, 출력X 인 함수");

    //#2. 입력O, 출력X 인 함수
    //익명이너클래스 표현
    B b1 = new B() {
        @Override
        public void method2(int a){
            System.out.println(a);
        }
    };
    //람다식 표현
    B b2 = (int a) -> {System.out.println(a);};
    B b3 = (a) -> {System.out.println(a);};
    B b4 = (a) -> System.out.println(a);
    B b5 = a -> System.out.println(a);

    //#3. 입력X, 출력O 인 함수
    //익명이너클래스 표현
    C c1 = new C() {
        @Override
        public int method3() {
            return 4;
        }
    };
    //람다식 표현
    C c2 = ()->{return 4;};
    C c3 = ()->4;

    //#4. 입력O, 출력O 인 함수
    //익명이너클래스 표현
    D d1 = new D() {
        @Override
        public double method4(int a, double b) {
            return a + b;
```

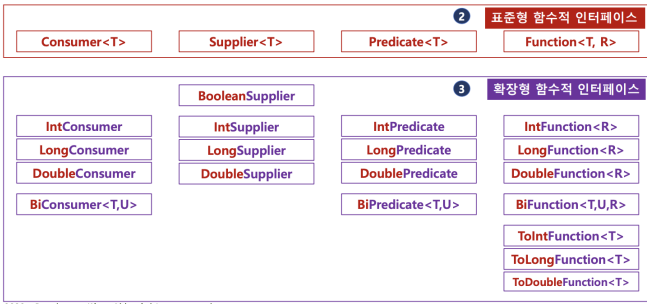
```
        }
    };
    //람다식 표현
    D d2 = (int a, double b)->{return a+b;};
    D d3 = (a, b)->{return a+b;};
    D d4 = (a, b)->a+b;
  }
}
```

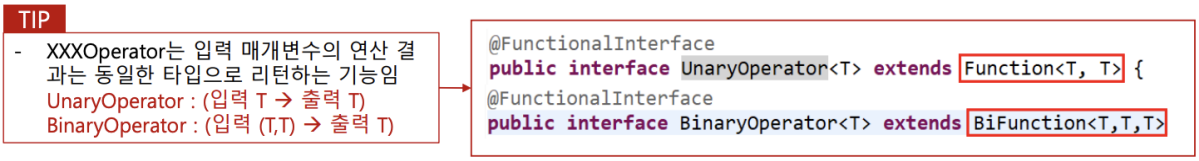
활용#2. 메서드 참조 (**인스턴스 메서드 참조 Type1, 정적 메서드 참조, 인스턴스 메서드 참조 Type2**)

활용#3. 생성자 참조 (**배열 생성자 참조, 클래스 생성자 참조**)

자바 API의 함수형 인터페이스

메서드의 매개변수에 사용되는 함수적 인터페이스 ⇒ 자주 사용하는 기능을 **정의할 수 있는** 함수 제공





- Consumer<T>

```
interface Consumer<T> {
  public abstract void accept(T t);
}
```

  - 표준형
  - 확장형

- Supplier<T>

```
interface Supplier<T> {
  public abstract T get();
}
```

  - 표준형
  - 확장형

- Predicate<T>

```
interface Predicate<T> {
  public abstract boolean test (T t);
}
```

  - 표준형
  - 확장형

- Function<T,R>

```
interface Function<T,R> {
  public abstract R apply (T t);
}
```

  - 표준형
  - 확장형

- UnaryOperator<T>

```
interface UnaryOperator<T> {
  public abstract T apply (T t);
}
```

- 표준형
- 확장형

- BinaryOperator<T>

```
interface BinaryOperator<T> {
  public abstract T apply (T t1, T t2);
}
```

- 표준형
- 확장형