KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

표준 API의 함수적 인터페이스

[KB] IT's Your Life

추상 메소드가 하나인 인터페이스를 이용

반환과 매개변수 형식이 중요

KB국민은행

표준 API의 함수적 인터페이스

자바 8부터 표준 API로 제공되는 함수적 인터페이스

따로 인터페이스 만들지 말고
얘네들 사용하라고 java가 만들어놨음

- 인터페이스에 선언된 추상 메소드의 매개값과 리턴 유무 따라 구분

종류	추상 메서드 특징	설명
Runnable	매개변수 없고, 리턴값 없음	
Consumer	매개변수 있고, 리턴값 없음	
Supplier	매개변수 없고, 리턴값 있음	
Predicate	매개변수 있고, boolean 타입 리턴 매개값을 조사해서 true/false를 리턴	
Function	매개변수 있고, 리턴값 있음 주로 매개값을 리턴값으로 매핑(타입 변환)	
Operator	매개변수 있고, 리턴값 있음 주로 매개변수를 연산하고 결과를 리턴	

많이 쓰임

입출력 타입 다름

입출력 타입은 같음

번외. 클래스의 기본 생성자는? => 매개변수 없이 주니까 supplier
클래스의 매개변수 필요한 생성자는? => 매개변수를 주고 다른걸 배출하니까 Function(매핑한다고 표현함)

표준 API의 함수적 인터페이스

Consumer 함수적 인터페이스

- 매개값만 있고 리턴값이 없는 추상 메소드 가짐

- 매개 변수의 타입과 수에 따라 분류

인터페이스명	추상 메서드	설명
Consumer<T>	**void accept(T t)**	**객체 T를 받아 소비**
BiConsumer<T, U>	**void accept(T t, U u)**	**객체 T와 U를 받아 소비**
DoubleConsumer	void accept(double value)	double 값을 받아 소비
IntConsumer	void accept(int value)	int 값을 받아 소비
LongConsumer	void accept(long value)	long 값을 받아 소비
ObjDoubleConsumer<T>	void accept(T t, double value)	객체 T와 double 값을 받아 소비
ObjIntConsumer<T>	void accept(T t, int value)	객체 T와 int 값을 받아 소비
ObjLongConsumer<T>	void accept(T t, long value)	객체 T와 long 값을 받아 소비

2개짜리

타입 추론
오버헤드
줄이게끔

표준 API의 함수적 인터페이스

Consumer 함수적 인터페이스

```
Consumer<String> consumer = t -> { t를 소비하는 실행문; };

BiConsumer<String, String> consumer = (t, u) -> { t와 u를 실행하는 실행문; }

DoubleConsumer consumer = d - > {d를 소비하는 실행문;}

ObjIntConsumer<String> consumer = (t, i) -> {t와 i를 소비하는 실행문; }
```

표준 API의 함수적 인터페이스

Consumer 함수적 인터페이스: ConsumerExample.java

```
import java.util.function.BiConsumer;
import java.util.function.Consumer;
import java.util.function.DoubleConsumer;
import java.util.function.ObjIntConsumer;

public class ConsumerExample {
  public static void main(String[] args) {
    Consumer<String> consumer = t -> System.out.println(t + "17");
    consumer.accept("java");

    BiConsumer<String, String> bigConsumer = (t, u)->System.out.println(t + u);
    bigConsumer.accept("Java", "17");

    DoubleConsumer doubleConsumer = d -> System.out.println("Java" + d);
    doubleConsumer.accept(17.0);

    ObjIntConsumer<String> objIntConsumer = (t, i) -> System.out.println(t + i);
    objIntConsumer.accept("Java", 8);
  }
}
```

표준 API의 함수적 인터페이스

Supplier 함수적 인터페이스

- 매개값은 없고 리턴값만 있는 추상 메소드 가짐

- 리턴 타입 따라 분류

인터페이스명	추상 메서드	설명
Supplier<T>	**T get()**	**객체를 리턴**
BooleanSupplier	Boolean getAsBoolean()	boolean 값을 리턴
DoubleSupplier	double getAsDouble()	double 값을 리턴
IntSupplier	int getAsInt()	int 값을 리턴
LongSupplier	long getAsLong()	long 값을 리턴

표준 API의 함수적 인터페이스

Supplier 함수적 인터페이스

```
Supplier<String> supplier = () -> { …; return "문자열"; }

IntSupplier supplier = () -> { …; return int값; }
```

Supplier 함수적 인터페이스: SupplierExample .java

```
import java.util.function.IntSupplier;

public class SupplierExample {
  public static void main(String[] args) {
    IntSupplier intSupplier = () -> {
      int num = (int) (Math.random() * 6) + 1;
      return num;
    };

    int num = intSupplier.getAsInt();
    System.out.println("눈의 수: " + num);
  }
}
```

표준 API의 함수적 인터페이스

Predicate 함수적 인터페이스

- 매개값 조사해 true 또는 false를 리턴할 때 사용

- 매개변수 타입과 수에 따라 분류

인터페이스명	추상 메서드	설명
Predicate<T>	**booean test(T t)**	**객체 T를 조사**
BiPredicate<T,U>	**boolean test(T t, U u)**	**객체 T를 U를 비교 조사**
DoublePredicate	boolean test(double value)	double 값을 조사
IntPredicate	boolean test(int value)	int 값을 조사
LongPredicate	boolean test(long value)	long 값을 조사

```
Predicate<Student> predicate = t -> { return t.getSex().equals("남자"); }
또는
Predicate<Student> predicate = t -> t.getSex().equals("남자");
```

남자만 뽑고 싶다, 점수가 과락인 애들만 뽑고 싶다. 필터링 할때 많이 사용함

표준 API의 함수적 인터페이스

Student 클래스: Student.java

```
public class Student {
  private String name;
  private String sex;
  private int score;

  public Student(String name, String sex, int score) {
    this.name = name;
    this.sex = sex;
    this.score = score;
  }

  public String getSex() { return sex; }
  public int getScore() { return score; }
}
```

Predicate 함수적 인터페이스: PredicateExample.java

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;

public class PredicateExample {
  private static List<Student> list = Arrays.asList(
      new Student("홍길동", "남자", 90),
      new Student("김순희", "여자", 90),
      new Student("감자바", "남자",  95),
      new Student("박한나", "여자", 92)
  );

  public static double avg(Predicate<Student> predicate) {
    int count = 0, sum = 0;
    for(Student student : list) {
      if(predicate.test(student)) {
        count++;
        sum += student.getScore();
      }
    }
    return (double) sum / count;
  }
```

Predicate 함수적 인터페이스: PredicateExample.java

```
  public static void main(String[] args) {
    double maleAvg = avg( t ->  t.getSex().equals("남자") );
    System.out.println("남자 평균 점수: " + maleAvg);

    double femaleAvg = avg( t ->  t.getSex().equals("여자") );
    System.out.println("여자 평균 점수: " + femaleAvg);
  }
}
```

필터 기능으로 자주 쓰임!!

표준 API의 함수적 인터페이스

Function 함수적 인터페이스

- 매개값과 리턴값이 모두 있는 추상 메소드 가짐
- 주로 매개값을 리턴값으로 매핑(타입 변환)할 경우 사용

표준 API의 함수적 인터페이스

Function 함수적 인터페이스

- 매개 변수 타입과 리턴 타입 따라 분류

인터페이스명	추상 메서드	설명
Function<T,R>	**R apply(T t)**	**객체 T를 객체 R로 매핑**
BiFunction<T,U,R>	**R apply(T t, U u)**	**객체 T와 U를 객체 R로 매핑**
DoubleFunction<R>	R apply(double value)	double을 객체 R로 매핑
IntFunction<R>	R apply(int value)	int를 객체 R로 매핑
IntToDoubleFunction	double applyAsDouble(int value)	int를 dobule로 매핑
IntToLongFunction	long applyAsLong(int value)	int를 long으로 매핑
LongToDoubleFunction	double applyAsDouble(long value)	long을 double로 매핑
LontToIntFunction	int applyAsInt(long value)	long을 int로 매핑
ToDoubleBiFunction<T,U>	double applyAsDouble(T t, U u)	객체 T와 U를 double로 매핑
ToDoubleFunction<T>	double applyAsDouble(T t)	객체 T를 double로 매핑
ToIntBiFunction<T,U>	int applyAsInt(T t, U u)	객체 T와 U를 int로 매핑
ToIntFunction<T>	int applyAsInt(T t)	객체 T를 int로 매핑
ToLongBiFunction<T,U>	long applyAsLong(T t, U u)	객체 T와 U를 long으로 매핑
ToLongFunction<T>	long applyAsLong(T t)	객체 T를 long으로 매핑

표준 API의 함수적 인터페이스

Function 함수적 인터페이스

```
Function<Student, String> function = t -> { return t.getName(); }
또는
Function<Student, String> function = t -> t.getName();


ToIntFunction<Student> function = t -> { return t.getScore(); }
또는
ToIntFunction<Student> function = t -> t.getScore();
```

Student 클래스: Student.java

```
public class Student {
  private String name;
  private int englishScore;
  private int mathScore;

  public Student(String name, int englishScore, int mathScore) {
    this.name = name;
    this.englishScore = englishScore;
    this.mathScore = mathScore;
  }

  public String getName() { return name; }
  public int getEnglishScore() { return englishScore; }
  public int getMathScore() { return mathScore; }
}
```

Function 함수적 인터페이스: FunctionExample1.java

```
import java.util.Arrays;
import java.util.List;
import java.util.function.Function;
import java.util.function.ToIntFunction;

public class FunctionExample1 {
  private static List<Student> list = Arrays.asList(
    new Student("홍길동", 90, 96),
    new Student("신용권", 95, 93)
  );

  public static void printString(Function<Student, String> function) {
    for(Student student : list) {
      System.out.print(function.apply(student) + " ");
    }
    System.out.println();
  }
```

Function 함수적 인터페이스: FunctionExample1.java

```
  public static void printInt(ToIntFunction<Student> function) {
    for(Student student : list) {
      System.out.print(function.applyAsInt(student) + " ");
    }
    System.out.println();
  }

  public static void main(String[] args) {
    System.out.println("[학생 이름]");
    printString( t -> t.getName() );

    System.out.println("[영어 점수]");
    printInt( t -> t.getEnglishScore() );

    System.out.println("[수학 점수]");
    printInt( t -> t.getMathScore() );
  }
}
```

Function 함수적 인터페이스: FunctionExample2.java

```
import java.util.Arrays;
import java.util.List;
import java.util.function.ToIntFunction;

public class FunctionExample2 {
  private static List<Student> list = Arrays.asList(
    new Student("홍길동", 90, 96),
    new Student("신용권", 95, 93)
  );

  public static double avg(ToIntFunction<Student> function) {
    int sum = 0;
    for(Student student : list) {
      sum += function.applyAsInt(student);
    }
    double avg = (double) sum / list.size();
    return avg;
  }

  public static void main(String[] args) {
    double englishAvg = avg( s -> s.getEnglishScore() );
    System.out.println("영어 평균 점수: " + englishAvg);

    double mathAvg = avg( s -> s.getMathScore() );
    System.out.println("수학 평균 점수: " + mathAvg);
  }
}
```

표준 API의 함수적 인터페이스

Operator 함수적 인터페이스

- 매개값과 리턴값이 모두 있는 추상 메소드 가짐
- 주로 매개값을 연산하고 그 결과를 리턴할 경우에 사용
- 매개 변수의 타입과 수에 따라 분류

표준 API의 함수적 인터페이스

Operator 함수적 인터페이스

- 매개 변수의 타입과 수에 따라 분류

인터페이스명	추상 메서드	설명
BinaryOperator<T>	**BiFunction<T, U, R>의 하위 인터페이스**	**T와 U를 연산한 후 R 리턴**
UnaryOperator<T>	**Function<T, R>의 하위 인터페이스**	**T를 연산 한 후 R 리턴**
DoubleBinaryOperator	double applyAsDouble(double, double)	두 개의 double 연산
DoubleUnaryOperator	double applyAsDouble(double)	한 개의 double 연산
IntBinaryOperator	int applyAsInt(int, int)	두 개의 int 연산
IntUnaryOperator	int applyAsInt(int)	한 개의 int 연산
LongBinaryOperator	long applyAsDouble(long, long)	두 개의 long 연산
LongUnaryOperator	long applyAsDouble(long)	한 개의 long 연산

```
IntBinaryOperator operator = (a, b) -> { …; return int 값; }
```

Operator 함수적 인터페이스: OperatorExample.java

```
import java.util.function.IntBinaryOperator;

public class OperatorExample {
  private static int[] scores = { 92, 95, 87 };

  public static int maxOrMin(IntBinaryOperator operator) {
    int result = scores[0];
    for(int score : scores) {
      result = operator.applyAsInt(result, score);
    }
    return result;
  }
```

Operator 함수적 인터페이스: OperatorExample.java

```
  public static void main(String[] args) {
    //최대값 얻기
    int max = maxOrMin(
      (a, b) -> {
        if(a>=b) return a;
        else return b;
      }
    );
    System.out.println("최대값: " + max);

    //최소값 얻기
    int min = maxOrMin(
      (a, b) -> {
        if(a<=b) return a;
        else return b;
      }
    );
    System.out.println("최소값: " + min);
  }
}
```