5/9 스터디노트

- P423 함수적 인터페이스 (람다식으로 메서드 표현 가능한 추상메서드를 1개 가지고 있는 인터페이스)

인터페이스 종류	매개변수 (입력값)	리턴 (출력값)
-	X	X
Consumer	O	X
Supplier	X	O
Function : 범용적 목적 Operator : 연산결과에 특화됨 Predicate : 리턴값이 boolean 인 경우	O	O

✓ 위의 인터페이스는 자바가 제공하는 인터페이스이므로, 앞의 람다 예시처럼 따로 인터페이스를 생성하지 않고 코드 작성 가능함

```
public class LambdaFunctionEx {

    public static void main(String[] args) {


        InterfaceEx ie = (int x, int y) -> x+y;
                                                    //람다식,
                                                    // 람다함
                                                    // 람다함


        System.out.println(ie.sum( x: 1,  y: 2));

    }

}

interface InterfaceEx {
    public int sum(int x, int y);
}
```

```
public class LambdaEx6 {
    public static void main(String[] args) {

        Consumer<String> c1
                = name -> System.out.println("저는 " + name + "입니다.");

        c1.accept( t: "홍길동");

    }
}
```

- P424 Consumer 인터페이스 예제
  - ✓ Consumer 인터페이스들은 매개변수만 있고 리턴값이 없음.
  - ✓ **accept();** 메서드

```
import java.util.function.*;

public class LambdaEx6 {
    public static void main(String[] args) {

        Consumer<String> c1 = name -> System.out.println("저는 " + name + "입니다.");
        c1.accept( t: "홍길동");


        BiConsumer<String, String> c2 = (lastname, firstname)
                                        -> { System.out.println("저는 " + lastname + firstname + "입니다."); };
        c2.accept( t: "홍", u: "길동");


        DoubleConsumer c3 = (score) -> System.out.println("제 점수는 " + score + "입니다.");
        c3.accept( value: 95.5);


        ObjIntConsumer<String> c4 = (lecture, i)
                                    -> { System.out.println("제 " + lecture + " 점수는 " + i + "점 입니다."); };
        c4.accept( t: "국어",  value: 100);

    }
}
```

```
저는 홍길동입니다.
저는 홍길동입니다.
제 점수는 95.5입니다.
제 국어 점수는 100점 입니다.
```

- P424 Supplier 인터페이스 예제
  - ✓ 매개변수가 없고 리턴값만 있음

인터페이스	추상메서드	기능
Supplier<T>	get();	객체 리턴
BooleanSupplier	getAsBoolean();	boolean 리턴
DoubleSupplier	getAsDouble();	double 리턴
IntSupplier	getAsInt();	int 리턴
LongSupplier	getAsLong();	long 리턴

```
Supplier<String> s1 = () -> { return "홍길동"; };
System.out.println( s1.get() );

s1 = () -> "이순신";
System.out.println( s1.get() );


IntSupplier s2 = () -> {
                      int num = (int)(Math.random() * 6)+1;
                      return num ;
                      };
System.out.println("주사위: "+s2.getAsInt());


DoubleSupplier s3 = () -> Math.PI;
System.out.println("원주율: "+s3.getAsDouble());
```

```
홍길동
```

```
이순신
```

```
주사위: 6
```

```
원주율: 3.141592653589793
```

```
Math.random() 은 0 < 랜덤값 <1

*6을 하면 0< 랜덤값 <6 -> 5까지만 가능하니까 +1
```

- P426 Function 인터페이스 예제
  - ✓ 매개변수와 리턴값이 모두 있음 : 주로 매개변수로 받은 것을 리턴값으로 매핑 (apply 메서드)
  - ✓ 사용예시

```
Function<Integer, Integer> mysquare = (value) -> value+value;
Integer result = mysquare.apply( t: 3);
System.out.println(result);
```

6

인터페이스	추상메서드	기능
Function<T, R>	apply(T)	T타입의 인자를 R타입으로 리턴
BiFunction<T, U, R>	apply(T, U)	T, U를 R타입으로 리턴
DoubleFunction<R> (= Function<double, R>)	apply(double)	double을 R타입으로 리턴
IntFunction<R> (= Function<int, R>)	apply(int)	int를 R타입으로 리턴
IntToDoubleFunction	applyAsDouble(int)	int를 double타입으로 리턴
IntToLongFunction	applyAsLong(int)	int를 long타입으로 리턴
LongToDoubleFunction	applyAsDouble(long)	long을 double타입으로 리턴
LongToIntFunction	applyAsInt(long)	long을 int타입으로 리턴
ToDoubleFunction<T>	applyAsDouble(T)	T를 double타입으로 리턴
ToDoubleBiFunction<T, U>	applyAsDouble(T, U)	T, U를 double타입으로 리턴
ToIntFunction<T>	applyAsInt(T)	T를 int타입으로 리턴
ToIntBiFunction<T, U>	applyAsInt(T, U)	T, U를 int타입으로 리턴
ToLongFunction<T>	applyAsLong(T)	T를 long타입으로 리턴
ToLongBiFuction<T, U>	applyAsLong(T, U)	T, U를 long타입으로 리턴

✓ < Function 인터페이스의 apply 메서드 예제 : 배열에서 이름값만 출력하기 > (교재 P426 예제 일부)

```
class Student{
 //필드
    private String name;    private int eng;    private int math;    private String major;
 //생성자
    public Student(String name, int eng, int math, String major) {...}
 //getter
    public String getName() {  return name;   }
}

public class LambdaEx8 {
    static Student[] list = {
            new Student( name: "홍길동",  eng: 90,  math: 80,  major: "컴공"),
            new Student( name: "이순신",  eng: 95,  math: 70,  major: "통계")
    };

    static void printString(Function<Student, String> f){
        for (Student s : list){
            System.out.print(f.apply(s));
        }
    }

    public static void main(String[] args) {
        System.out.print("학생명 : ");
        printString(t -> t.getName());
    }

}
```

학생명 : 홍길동이순신

[Student 클래스]

이름 / 영어점수 / 수학점수 / 전공

[Student 클래스객체로 이루어진 list]

홍길동 학생과 이순신 학생 데이터 생성

[printString 이라는 메서드 생성]

printString 메서드는 Function 인터페이스의 객체를 매개변수로 받음
이때 Function 객체 f 는 Student클래스를 받아서 String 타입으로 리턴함(apply)
for문: Student 배열인 list의 데이터 (s)를 받아서 string 타입으로 리턴하기

[메인메서드]

"학생명 : " 이라는 문자열을 출력하고 나서

printString 메서드를 시행하는데

위의 f.apply(s) 메서드로 인해 Student 배열의 값들을 string으로 리턴하는데

getName() 메서드로 "이름"값만 가져오도록 함 : 이름값만 출력됨

✓ < ToIntFunction 인터페이스의 applyAsInt 메서드 예제 : 배열에서 특정 값들의 합계 구하기 > (교재 P426 예제 일부)

```
class Student2{
 //필드
    private String name;    private int eng;    private int math;    private String major;
 //생성자
    public Student2(String name, int eng, int math, String major) {...}
 //getter
    public int getEng() {  return eng;   }
}

public class LambdaEx8_1 {
    static Student2[] list = {
            new Student2( name: "홍길동",  eng: 90,  math: 80,  major: "컴공"),
            new Student2( name: "이순신",  eng: 95,  math: 70,  major: "통계")
    };

    static void printTotal(ToIntFunction<Student2> f){
        int sum = 0;
        for (Student2 s : list){
            sum += f.applyAsInt(s);
        }
        System.out.print(sum);
    }

    public static void main(String[] args) {
        System.out.print("영어점수 합계 : ");
        printTotal(t -> t.getEng());
    }

}
```

[printTotal 이라는 메서드 생성]

printString 메서드는 ToIntFunction 인터페이스의 객체를 매개변수로 받음
이때 ToIntFunction 객체 f 는 Student2클래스를 받아서 int 타입으로 리턴함(applyAsInt)
for문: Student2 배열인 list의 데이터 (s)를 받아서 sum 에 더하고
누적합계인 sum 을 출력함

[메인메서드]

"영어점수 합계 : " 라는 문자열을 출력하고 나서

printTotal 메서드를 시행하는데
위의 f.applyAsInt(s) 메서드로 인해 Student2 배열의 값들을 sum에 더하여 리턴하는데
getEng() 메서드로 "영어점수"값만 가져와서 더하도록 함 : 영어점수 합계가 출력됨

✓ < ToDoubleFunction 인터페이스의 applyAsDouble 메서드 예제 : 배열에서 특정 값들의 평균 구하기 > (교재 P426 예제 일부)

```
class Student3{
 //필드
    private String name;    private int eng;    private int math;    private String major;
 //생성자
    public Student3(String name, int eng, int math, String major) {...}
 //getter
    public int getEng() {  return eng;   }
}

public class LambdaEx8_2 {
    static Student3[] list = {
            new Student3( name: "홍길동",  eng: 90,  math: 80,  major: "컴공"),
            new Student3( name: "이순신",  eng: 95,  math: 70,  major: "통계")
    };

    static void printAvg(ToDoubleFunction<Student3> f){
        double sum = 0;
        for (Student3 s : list){
            sum += f.applyAsDouble(s);
        }
        System.out.print(sum / list.length);
    }

    public static void main(String[] args) {
        System.out.print("영어점수 평균 : ");
        printAvg(t -> t.getEng());
    }

}
```

[printAvg 라는 메서드 생성]

printAvg 메서드는 ToDoubleFunction 인터페이스의 객체를 매개변수로 받음
이때 객체 f 는 Student3클래스를 받아서 double 타입으로 리턴함(applyAsDouble)
for문: Student3 배열인 list의 데이터 (s)를 받아서 sum 에 더하고
평균인 sum / list.length 을 출력함

[메인메서드]

"영어점수 평균 : " 이라는 문자열을 출력하고 나서

printAvg 메서드를 시행하는데
위의 f.applyAsDouble(s) 메서드로 인해 Student3 배열의 값들의 평균값을 리턴하는데
getEng() 메서드로 "영어"점수값만 가져와서 평균을 계산하도록 함 : "영어"점수 평균이 출력됨

- P428 Operator 인터페이스
  - ✓ Function 인터페이스의 하위 인터페이스로 apply 메서드를 사용, 매개변수와 리턴값이 모두 있음.
  - ✓ 주로 매개변수를 입력받아서 연산하여 결과를 리턴할 때 씀 -> 따라서 매개변수와 리턴값의 타입이 동일함

인터페이스명	추상 메서드	설명
BinaryOperator<T>	T apply(T t T t)	T와 T를 연산한후 T리턴
UnaryOperator<T>	T apply(T t)	T를 연산한후 T 리턴
DoubleBinaryOperator	double applyAsDouble(double, double)	두개의 double 연산
DoubleUnaryOperator	double applyAsDouble(double)	한 개의 double 연산
IntBinaryOperator	int applyAsInt(int, int)	두 개의 int 연산
IntUnarayOperator	int applyAsInt(int)	한 개의 int 연산
LongBinaryOperator	long applyAsLong(long, long)	두 개의 long 연산
LongUnaryOperator	long applyAsLong(long)	한 개의 long 연산

✓ BinaryOperator 인터페이스 apply 메서드 선생님 예제 : lambdaCalculator

```java
public static void main(String[] args) {

    System.out.println("람다 계산기: 두 수에 대한 사칙연산 기능 제공");

    Supplier<Double> inputNumber = () -> {
        Scanner scanner = new Scanner(System.in);
        System.out.print("수를 입력하세요: ");
        return scanner.nextDouble();
    };

    double num1 = inputNumber.get();
    double num2 = inputNumber.get();

    BinaryOperator<Double> add = (a,b)-> a+b;
    BinaryOperator<Double> subtract = (a,b)-> a-b;
    BinaryOperator<Double> multiply = (a,b)-> a*b;
    BinaryOperator<Double> divide = (a,b)-> a/b;

    System.out.println("add.apply(num1, num2) : "+add.apply(num1,num2));
    System.out.println("subtract.apply(num1, num2) : "+subtract.apply(num1,num2));
    System.out.println("multiply.apply(num1, num2) : "+multiply.apply(num1,num2));
    System.out.println("divide.apply(num1, num2) : "+divide.apply(num1,num2));
}
```

```
람다 계산기: 두 수에 대한 사칙연산 기능 제공
수를 입력하세요: 10
수를 입력하세요: 5
add.apply(num1, num2) : 15.0
subtract.apply(num1, num2) : 5.0
multiply.apply(num1, num2) : 50.0
divide.apply(num1, num2) : 2.0
```

- P430 Predicate 인터페이스
  - ✓ 매개변수가 있고 리턴값이 있는 인터페이스. (리턴값은 항상 boolean)

인터페이스명	추상 메소드	설명
Predicate<T>	boolean test(T t)	객체 T를 조사
BiPredicate<T, U>	boolean test(T t, U u)	객체 T와 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

```java
public class LambdaEx10_1 {
// 앞 예제에서 작성한 Student 클래스의 배열
    static Student[] list = {
            new Student( name: "홍길동",  eng: 90,  math: 80,  major: "컴공"),
            new Student( name: "이순신",  eng: 95,  math: 70,  major: "통계"),
            new Student( name: "김유신",  eng: 100,  math: 60,  major: "컴공")
    };
//메인메서드
    public static void main(String[] args) {
        // 과가 컴공인 학생의 영어점수 평균
        double avg = avgEng(t -> t.getMajor().equals("컴공"));
        System.out.println("컴공과 평균 영어점수 : "+avg);

    }
//avgEng 메서드 생성
    private static double avgEng(Predicate<Student> predicate) {
        int count = 0;
        int sum = 0;
        for (Student student : list) {
            // equals 비교
            if (predicate.test(student)) {
                count++;
                sum += student.getEng();
            }
        }
        return (double)sum/count;
    }
}
```

[메인 메서드]

avgEng 메서드 시행시

매개변수를 전공값이 "컴공"과 같은 값인 학생만 매개변수로 받아

영어점수를 가져와서 평균 계산

[avgEng 메서드 생성]

입력받은 student 값이 true 이면 영어점수를 더하고 학생수를 세어서 평균값 출력

- P432 연습문제
  - ✓ 1번 람다식에 대한 설명으로 올바르지 않은 것은?
    - ② 매개변수의 괄호는 생략할 수 없다. -> X. 매개변수가 1개인 경우에는 생략가능함.

  - ✓ 2번 **(int x, int y) -> x+y;**

Chapter 16 스트림

- ✓ 배열이나 Collection 의 list, map 등의 객체를 다룰 때 for 문을 이용해서 하나씩 꺼내는 방식을 사용했었다.
- ✓ 데이터가 많거나 복잡할수록 코드가 길어짐.
- ✓ -> 스트림에서 람다(함수형 인터페이스)를 이용해서 간결한 코드로 작성가능함. : `forEach();` 메서드
  또한 스트림에서 병렬 처리가 가능함 (작업을 동시에 처리하는 것 (multi thread))

● P435 list 객체의 값들을 차례대로 출력하는 코드를 [for구문 / iterator문 / 스트림] 3가지 방법으로 코드 작성

```
public static void main(String[] args) {

    List<String> list
            = Arrays.asList(new String[]{"홍길동", "김유신", "이순신", "유관순"});

    System.out.println("for문 이용");
    for (int i = 0; i < list.size(); i++) {
        System.out.println(list.get(i));
    }

    System.out.println("외부반복자 이용");
    Iterator<String> it = list.iterator();
    while(it.hasNext()){
        System.out.println(it.next());
    }

    System.out.println("내부반복자 이용");
    list.stream().forEach(s -> System.out.println(s));
}
```

```
for문 이용
홍길동
김유신
이순신
유관순
외부반복자 이용
홍길동
김유신
이순신
유관순
내부반복자 이용
홍길동
김유신
이순신
유관순
```

- P436 스트림 객체 만드는 방법 1 : 배열을 스트림객체로 바꾸기

```java
public class P436_ArrayToStream {
    public static void main(String[] args) {

  //문자열로 이루어진 배열 객체 생성

        String[] arr = new String[]{"a", "b", "c", "d", "e"};

  //배열 전체를 Stream 객체로 변환

        Stream<String> stream1 = Arrays.stream(arr);
        stream1.forEach(s -> System.out.print(s + " "));

  //줄바꿈
        System.out.println();

  //배열 일부를 지정해서 Stream 객체로 변환                    (2 다음부터 5 전까지)

        Stream<String> stream2 = Arrays.stream(arr, startInclusive: 2, endExclusive: 5);
        stream2.forEach(s -> System.out.print(s + " "));
    }
}
```

```
a b c d e
c d e
```

- P437 스트림 객체 만드는 방법 2 : Collection 객체를 스트림 객체로 바꾸기 (예제 : List)

```java
public class P437_CollectionToStream {
    public static void main(String[] args) {

  //문자열 배열을 Collection 객체인 List 로 변환
        List<String> list = Arrays.asList("a", "b", "c", "d", "e");

  //List 객체를 stream() 메서드로 stream 객체로 변환
        Stream<String> stream = list.stream();

  //내부반복자를 이용해서 출력
        stream.forEach(s -> System.out.println(s+""));

    }
}
```

```
a
b
c
d
e
```

- P438 스트림 객체 만드는 방법 3 : build() / generate() / iterate()

✓ 1) **builder() 메서드와 build() 메서드**

```
public class P438_StreamByBuilder {
    public static void main(String[] args) {

 // builder 메서드로 builder 객체 생성하고 나서  add메서드로 데이터 추가, build() 메서드로 stream 객체 생성
        Stream stream = Stream.builder().add("무궁화").add("삼천리").add("화려강산").build();

 //내부 반복자로 출력
        stream.forEach(s-> System.out.print(s+" "));


    }
}
```

```
무궁화 삼천리 화려강산
```

✓ 2) **generate() 메서드**

```java
public class P439_StreamByGenerate {
    public static void main(String[] args) {

 // generate() 메서드로 Stream 객체 생성    (limit()로 갯수 제한을 걸지 않으면 무한 생성됨)
        Stream<String> stream = Stream.generate(()->"애국가").limit( maxSize: 10);
                                                    //generate() : 매개변수 없고 리턴값만 있음

 // 내부반복자로 출력
        stream.forEach(s -> System.out.println(s));

    }
}
```

```
애국가
애국가
애국가
애국가
애국가
애국가
애국가
애국가
애국가
애국가
```

✓ 3) **iterate() 메서드**

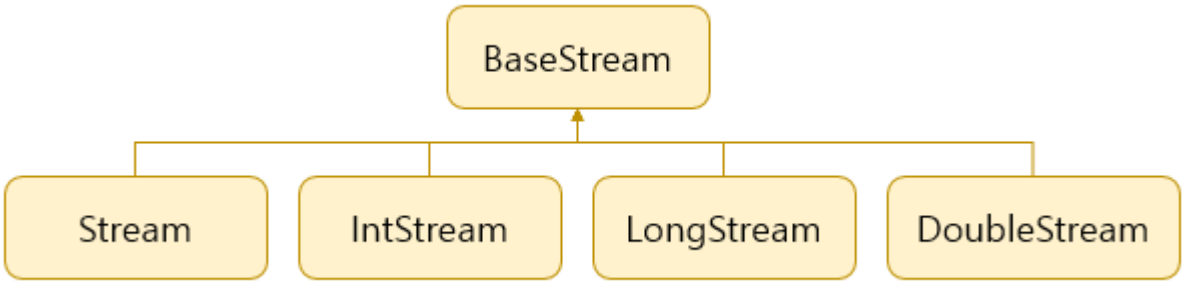
```java
public class P439_StreamByIterator {
    public static void main(String[] args) {

  // iterate 메서드로 stream 객체 생성   (limit 필요)
        Stream<Integer> stream = Stream.iterate( seed: 1, n->n+1).limit( maxSize: 10);

  // 내부 반복자로 출력
        stream.forEach(s -> System.out.println(s));

    }
}
```

초기값 지정, 람다표현식으로 정의

```
1
2
3
4
5
6
7
8
9
10
```

- P440 스트림의 종류

➢ 자료형별 종류



✓ 최상위 인터페이스 BaseStream : 직접 사용되지는 않음

✓ 각각 Steam 은 Object 자료형, IntStream 은 int , LongStream 은 long, DoubleStream 은 double 자료형을 처리함

✓ 메서드

✓ `parallelStream()` 메서드 ; 병렬처리가능 : 대용량 데이처 처리시 사용함

✓ `of()` : ( )안의 요소를 스트림객체로 만들기

✓ `range(int1 , int2)` : int1 부터 int2 까지 stream 생성

✓ `rangeClosed(int1, int2)` : int1 부터 int2 전까지 stream 생성

✓ `random.[자료형]s` : 해당 자료형의 난수로 stream 생성

- P441 스트림 메서드 예제

```
public static void main(String[] args) {
//배열 생성
    int[] arr = { 1,2,3,4,5 };

//arr 배열을 이용해서  Arrays.stream() 메서드로 stream 생성
    IntStream intstream1 = Arrays.stream(arr);
    //출력
    intstream1.forEach(s-> System.out.print(s+"\t"));

//arr 배열을 이용해서  IntStream.of() 메서드로 stream 생성
    IntStream intstream2 = IntStream.of(arr);
    //출력
    intstream2.forEach(s-> System.out.print(s+"\t"));


//IntStream.range() 메서드로 stream 생성
    IntStream intstream3 = IntStream.range(10, 15);
    //출력
    intstream3.forEach(s-> System.out.print(s+"\t"));

//IntStream.rangeClosed() 메서드로 stream 생성
    IntStream intstream4 = IntStream.rangeClosed(10, 15);
    //출력
    intstream4.forEach(s-> System.out.print(s+"\t"));
```

```
1   2   3   4   5
```

```
1   2   3   4   5
```

```
10   11   12   13   14
```

```
10   11   12   13   14   15
```

- P443 난수 스트림 메서드 예제 : **random.[자료형]s**

```
//int 자료형 난수 스트림 만들기                    (난수 3개)
    IntStream isr = new Random().ints( streamSize: 3);
    //출력
    isr.forEach(s-> System.out.print(s+"\t"));
```

```
-1792053071 -1378793034 -276220844
```

```
//int 자료형 난수 스트림 범위 지정해서 만들기       (난수 5개, 0부터 3 전까지)
    isr = new Random().ints( streamSize: 5, randomNumberOrigin: 0, randomNumberBound: 3);
    //출력
    isr.forEach(s-> System.out.print(s+"\t"));
```

```
2	0	2	1	2
```

```
//long 자료형 난수 스트림 만들기                   (난수 3개, 0부터 10 전까지)
    LongStream lsr = new Random().longs( streamSize: 3, randomNumberOrigin: 0, randomNumberBound: 10);
    //출력
    lsr.forEach(s-> System.out.print(s+"\t"));
```

```
3	3	4
```

```
//double 자료형 난수 스트림 만들기
    DoubleStream dsr = new Random().doubles( streamSize: 3);
    //출력
    dsr.forEach(s-> System.out.print(s+"\t"));
```

```
0.32928431809550096 0.8619330772453837  0.153338280738327
```

- P444 문자열을 stream 으로 처리하기 : chars() 메서드 사용

```java
public class P444_StrToStream {
    public static void main(String[] args) {
        String str = "자바 세상 만들까";

        IntStream isr = str.chars();

        isr.forEach(s-> System.out.println((char)s));
    }
}
```

자
바

세
상

만
들
까

숫자가 아니라 문자로 출력하려면 (char)로 강제 형변환 필요

- **P448 스트림 가공 [대박 중요]**
  - ✓ 데이터의 합계/평균/개수/최대값/최소값 등을 구하는 것을 Reduction 이라고 하는데
  - ✓ 그런 결과를 얻기 위해 중간 처리 작업이 필요할 때가 있다. : 필터링 / 매핑 / 정렬 / 그룹화 등의 중간처리 작업을 스트림에서 할 수 있음

- P448 스트림 가공하기 1 : 필터링
  - ✓ 필터링 메서드
    - distinct() ; 중복제거
    - filter(); 스트림의 각 요소를 하나씩 입력받아서 boolean을 리턴하는 표현식의 true에 해당하는 데이터만 모아서 새로운 스트림 생성 후 리턴

```
// List 객체 생성
    List<String> list = Arrays.asList("홍길동", "홍길동", "홍길동",
                                      "김유신", "이순신", "유관순");


// 중복제거 : distinct 메서드
    list.stream().distinct().forEach(s-> System.out.print(s+"\t"));
         ①          ②          ③
       스트림 생성   중간작업   결과출력
```

홍길동 김유신 이순신 유관순

```
// 필터링 : filter 메서드   ("홍" 으로 시작하는 문자열로 필터링)
    list.stream().filter(n -> n.startsWith("홍")).forEach(n -> System.out.print(n+"\t"));
```

홍길동 홍길동 홍길동

```
// 중복제거 + 필터링 : 중복제거 하고 나서 "홍"으로 시작하는 문자열로 필터링
    list.stream().distinct().filter(n -> n.startsWith("홍")).forEach(s-> System.out.print(s+"\t"));
                  중간작업 1      중간작업 2
```

홍길동

[만약 스트림을 사용하지 않고 중복제거를 하고나서 필터링하는 코드를 작성한다면?]

(스트림 사용)

```java
public static void main(String[] args) {

    // List 객체 생성
    List<String> list = Arrays.asList("홍길동", "홍길동", "홍길동", "김유신", "이순신", "유관순");

    // 중복제거 + 필터링 : 중복제거 하고 나서 "홍"으로 시작하는 문자열로 필터링
    list.stream().distinct().filter(n -> n.startsWith("홍")).forEach(s -> System.out.print(s + "\t"));

}
```

(스트림 사용 X)

```java
public static void main(String[] args) {

    List<String> list = Arrays.asList("홍길동", "홍길동", "홍길동", "김유신", "유관순", "이순신");

 //중복 제거 : if조건에 list에 이미 같은 값이 있는지 확인하여 유일한 값만으로 새로운 리스트 생성
    List<String> newList = new ArrayList<>();

    for(String str:list){
        if (!newList.contains(str)) {
            newList.add(str);
        }
    }

//필터링 : if조건에 "홍"으로 시작하는 문자열만 골라서 출력
    for(String str : newList){
        if (str.startsWith("홍")) {
            System.out.println(str);
        }
    }
}
```