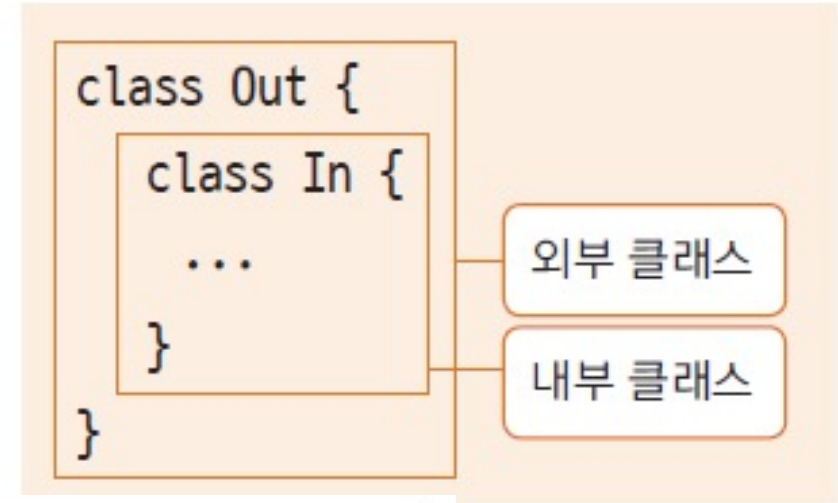


# 13. 내 부 클 래 스 람 다 식 스 트 림

# 내부 클래스 (inner class)

- 클래스 내부에 선언한 클래스
- 클래스 내부에서만 사용할 목적으로 만드는 클래스

- 내부 클래스의 종류



변수	내부 클래스
<pre>class ABC {     int n1;           // 인스턴스 변수     static int n2;    // 정적 변수      public void abc( ) {         int i;       // 지역 변수     } }</pre>	<pre>class ABC {           // 외부 클래스     class In {        // 인스턴스 내부 클래스         static class SIn { // 정적 내부 클래스         }     }     public void abc( ) {         class Local { // 지역 내부 클래스         }     } }</pre>

# 인스턴스 내부 클래스 (instance inner class)

- 외부 클래스 생성 후 생성
- 정적 변수, 정적 메서드 사용할 수 없음
- 외부 클래스의 참조 변수를 이용하여 내부 클래스 사용
- 내부 클래스가 `private` 이 아니면 다른 외부 클래스에서도 생성가능
- 클래스 내부에서만 사용할 목적이면 `private`으로 선언

# 인스턴스 내부 클래스 예

```
class OutClass {                                // 외부 클래스
    private int num = 10;                        // 외부 클래스 private 변수
    private static int sNum = 20;                // 외부 클래스 정적 변수
```

```
    private InClass inClass;
```

내부 클래스 자료형 변수를 먼저 선언

```
    public OutClass( ) {
        inClass = new InClass( );
    }
```

외부 클래스 디폴트 생성자. 외부 클래스가 생성된 후에 내부 클래스 생성 가능

```
class InClass {                                // 인스턴스 내부 클래스
    int inNum = 100;                            // 내부 클래스의 인스턴스 변수
```

```
    //static int sInNum = 200;
```

인스턴스 내부 클래스에 정적 변수 선언 불가능. 오류가 발생하므로 주석 처리함

```
    void inTest( ) {
        System.out.println("OutClass num = " + num + "(외부 클래스의 인스턴스 변수)");
        System.out.println("OutClass sNum = " + sNum + "(외부 클래스의 정적 변수)");
    }
```

# 인스턴스 내부 클래스 예

```
//static void sTest( ) {  
//}
```

정적 메서드 역시 정의 불가능. 오류  
가 발생하므로 주석 처리함

```
}  
public void usingClass( ) {  
    inClass.inTest( );  
}  
}
```

```
public class InnerTest {  
    public static void main(String[ ] args) {  
        OutClass outClass = new OutClass( );  
        System.out.println("외부 클래스 이용하여 내부 클래스 기능 호출");  
        outClass.usingClass( );  
    }  
}
```

내부 클래스 기능 호출

Problems @ Javadoc Declaration Console

<terminated> InnerTest [Java Application] C:\Program Files\Java

외부 클래스 이용하여 내부 클래스 기능 호출  
OutClass num = 10(외부 클래스의 인스턴스 변수)  
OutClass sNum = 20(외부 클래스의 정적 변수)

# 정적 내부 클래스 (static inner class)

- 외부 클래스의 생성과 무관하게 사용
- 정적 변수, 정적 메서드 사용
- 정적 내부 클래스 메서드에서 변수 사용 여부

정적 내부 클래스 메서드	변수 유형	사용 가능 여부
일반 메서드 void inTest( )	외부 클래스의 인스턴스 변수(num)	X
	외부 클래스의 정적 변수(sNum)	O
	정적 내부 클래스의 인스턴스 변수(inNum)	O
	정적 내부 클래스의 정적 변수(sInNum)	O
정적 메서드 static void sTest( )	외부 클래스의 인스턴스 변수(num)	X
	외부 클래스의 정적 변수(sNum)	O
	정적 내부 클래스의 인스턴스 변수(inNum)	X
	정적 내부 클래스의 정적 변수(sInNum)	O



# 정적 내부 클래스 예

```
class OutClass {  
    private int num = 10;  
    private static int sNum = 20;  
  
    static class InStaticClass {  
        int inNum = 100;  
        static int sInNum = 200;  
    }  
}
```

// 정적 내부 클래스  
// 정적 내부 클래스의 인스턴스 변수  
// 정적 내부 클래스의 정적 변수

정적 내부 클래스의 일반 메서드

```
void inTest( ) {  
    // num += 10;  
    System.out.println("InStaticClass inNum = " + inNum + "(내부 클래스의 인스턴스 변수 사용)");  
    System.out.println("InStaticClass sInNum = " + sInNum + "(내부 클래스의 정적 변수 사용)");  
    System.out.println("OutClass sNum = " + sNum + "(외부 클래스의 정적 변수 사용)");  
}
```

외부 클래스의 인스턴스 변수는 사용할 수 없으므로 주석 처리

# 정적 내부 클래스 예

정적 내부 클래스의 정적 메서드

```
static void sTest( ) {  
    // num += 10;  
    // inNum += 10;  
    System.out.println("OutClass sNum = " + sNum + "(외부 클래스의 정적 변수 사용)");  
    System.out.println("InStaticClass sInNum = " + sInNum + "(내부 클래스의 정적 변수 사용)");  
}  
}
```

외부 클래스와 내부 클래스의 인스턴스 변수는 사용할 수 없으므로 주석 처리

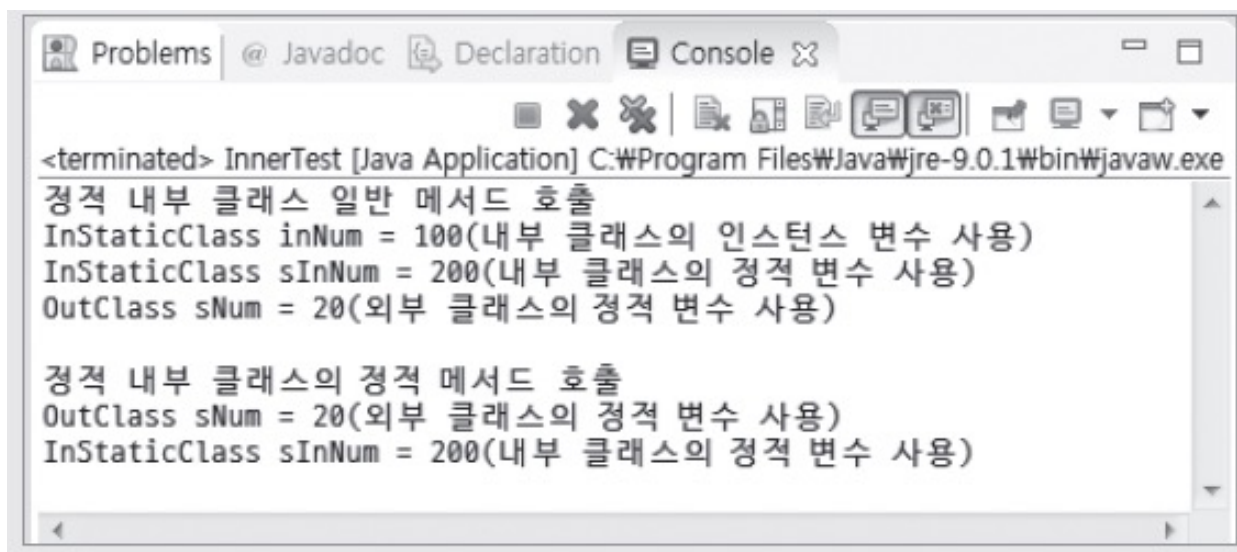
```
public class InnerTest {  
    public static void main(String[ ] args) {  
        ...  
        OutClass.InStaticClass sInClass = new OutClass.InStaticClass( );  
        System.out.println("정적 내부 클래스 일반 메서드 호출");  
        sInClass.inTest( );  
    }  
}
```

외부 클래스를 생성하지 않고 바로 정적 내부 클래스 생성 가능



# 정적 내부 클래스 예

```
System.out.println( );  
System.out.println("정적 내부 클래스의 정적 메서드 호출");  
OutClass.InStaticClass.sTest( );  
}  
}  
}
```



```
<terminated> InnerTest [Java Application] C:\Program Files\Java\jre-9.0.1\bin\javaw.exe  
정적 내부 클래스 일반 메서드 호출  
InStaticClass inNum = 100(내부 클래스의 인스턴스 변수 사용)  
InStaticClass sInNum = 200(내부 클래스의 정적 변수 사용)  
OutClass sNum = 20(외부 클래스의 정적 변수 사용)  
  
정적 내부 클래스의 정적 메서드 호출  
OutClass sNum = 20(외부 클래스의 정적 변수 사용)  
InStaticClass sInNum = 200(내부 클래스의 정적 변수 사용)
```

# 지역 내부 클래스 (local inner class)

- 지역 변수와 같이 메서드 내부에서 정의 하여 사용하는 클래스
- 메서드의 호출이 끝나면 메서드에 사용된 지역변수의 유효성은 사라짐
- 메서드 호출 이후에서 사용해야 하는 경우가 있으므로 지역 내부 클래스에서 사용하는 메서드의 지역 변수나 매개 변수는 `final`로 선언 됨

# 지역 내부 클래스 예

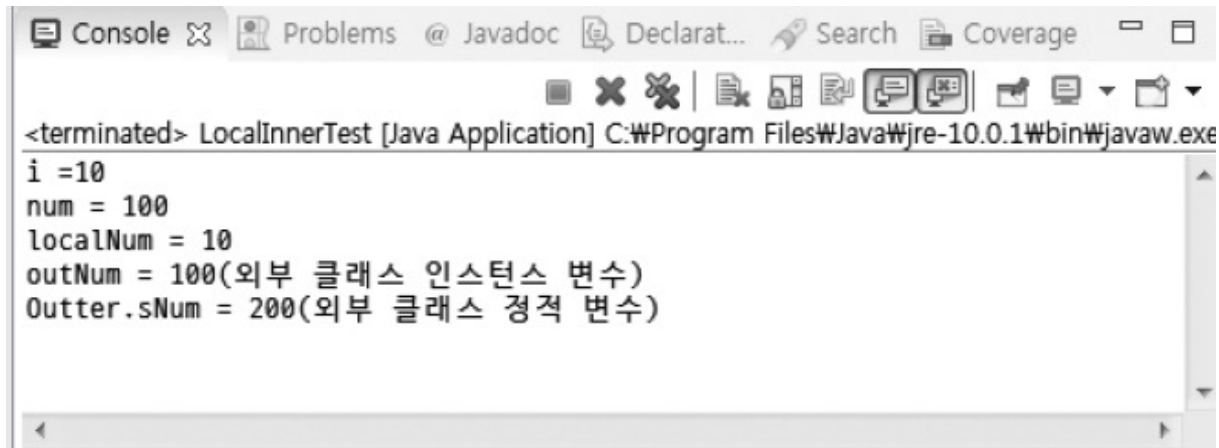
```
class Outer {  
    int outNum = 100;  
    static int sNum = 200;  
  
    Runnable getRunnable(int i) {  
        int num = 100;                                //지역 변수  
  
        class MyRunnable implements Runnable {          //지역 내부 클래스  
            int localNum = 10;                          //지역 내부 클래스의 인스턴스 변수  
            @Override  
            public void run( ) {  
                //num = 200;  
                //i = 100;  
                System.out.println("i =" + i);  
                System.out.println("num = " +num);  
                System.out.println("localNum = " +localNum);  
                System.out.println("outNum = " + outNum + "(외부 클래스 인스턴스 변수)");  
                System.out.println("Outer.sNum = " + Outer.sNum + "(외부 클래스 정적 변수)");  
            }  
        }  
        return new MyRunnable( );  
    }  
}
```

지역 변수는 상수로 바뀌므로 값을 변경할 수 없어 오류 발생

매개변수도 지역 변수처럼 상수로 바뀌므로 값을 변경할 수 없어 오류 발생

# 지역 내부 클래스 예

```
public class LocalInnerTest {  
    public static void main(String[ ] args) {  
        Outter out = new Outter( );  
        Runnable runner = out.getRunnable(10); //메서드 호출  
        runner.run( );  
    }  
}
```



# 익명 내부 클래스 (anonymous inner class)

- 이름이 없는 클래스
- 주로 하나의 인터페이스나 하나의 추상 클래스를 구현하는데 사용
- 인터페이스나 추상 클래스 자료형의 변수에 직접 대입하여 생성하거나 지역 내부 클래스의 메서드 내부에서 생성하여 반환 할 수 있음
- 안드로이드의 widget 의 이벤트 핸들러를 구현 할 때 많이 사용 됨

```
button1.setOnClickListener(new View.OnClickListener( ) {  
    public boolean onClick(View v) {  
        Toast.makeText(getApplicationContext( ), "hello ", Toast.LENGTH_LONG).show( );  
        return true;  
    }  
});
```

# 익명 내부 클래스 예

```
class Outer2 {  
    Runnable getRunnable(int i) {  
        int num = 100;
```

MyRunnable 클래스 이름을 빼고  
클래스를 바로 생성하는 방법

```
        return new Runnable( ) { // 익명 내부 클래스. Runnable 인터페이스 생성
```

```
            @Override
```

```
            public void run( ) {
```

```
                // num = 200;
```

```
                // i = 10;
```

오류 발생

```
                System.out.println(i);
```

```
                System.out.println(num);
```

```
            }
```

클래스 끝에 ;를 씀

```
        };
```

```
    }
```



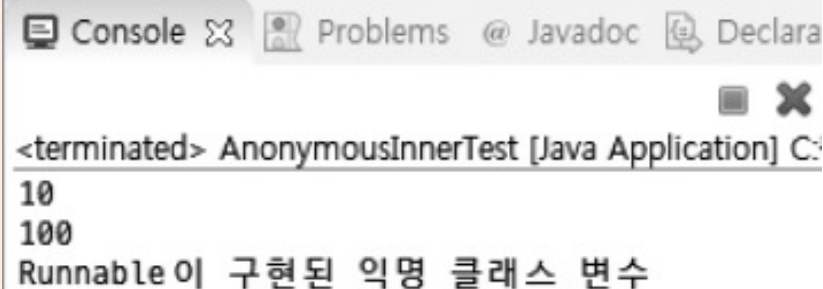
# 익명 내부 클래스 예

```
Runnable runner = new Runnable( ) { //익명 내부 클래스를 변수에 대입
    @Override
    public void run( ) {
        System.out.println("Runnable이 구현된 익명 클래스 변수");
    }
};
```

클래스 끝에 ;를 씀

인터페이스나 추상 클래스형 변수를 선언하고 클래스를 생성해 대입하는 방법

```
public class AnonymousInnerTest {
    public static void main(String[ ] args) {
        Outter2 out = new Outter2( );
        Runnable runnerble = out.getRunnable(10);
        runnerble.run( );
        out.runner.run( );
    }
}
```



Console Problems @ Javadoc Declara

<terminated> AnonymousInnerTest [Java Application] C:

10  
100  
Runnable 이 구현된 익명 클래스 변수

# 내부 클래스 요약

종류	구현 위치	사용할 수 있는 외부 클래스 변수	생성 방법
인스턴스 내부 클래스	외부 클래스 멤버 변수와 동일	외부 인스턴스 변수 외부 전역 변수	외부 클래스를 먼저 만든 후 내부 클래스 생성
정적 내부 클래스	외부 클래스 멤버 변수와 동일	외부 전역 변수	외부 클래스와 무관하게 생성
지역 내부 클래스	메서드 내부에 구현	외부 인스턴스 변수 외부 전역 변수	메서드를 호출할 때 생성
익명 내부 클래스	메서드 내부에 구현 변수에 대입하여 직접 구현	외부 인스턴스 변수 외부 전역 변수	메서드를 호출할 때 생성 되거나, 인터페이스 타입 변수에 대입할 때 new 예약어를 사용하여 생성

# 람다식 (lambda expression)

- 자바에서 함수형 프로그래밍(functional programming)을 구현하는 방식
- 자바 8 부터 지원
- 클래스를 생성하지 않고 함수의 호출만으로 기능을 수행
- **함수형 프로그래밍**
  - 순수 함수(pure function)를 구현하고 호출함으로써 외부 자료에 부수
  - 적인 영향을 주지 않고 매개 변수만을 사용하도록 만든 함수
  - 함수를 기반으로 구현
  - 입력 받은 자료를 기반으로 수행되고 외부에 영향을 미치지 않으므로 병렬처리에 가능
  - 안정적인 확장성 있는 프로그래밍 방식

# 람다식 구현하기

- 익명 함수 만들기
- 매개 변수와 매개 변수를 활용한 실행문으로 구현
- 두 수를 입력 받아 더하는 add() 함수

(매개변수) -> {실행문;}

```
int add(int x, int y) {  
    return x + y;  
}
```



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

- 함수 이름 반환 형을 없애고 -> 를 사용
- { } 까지 실행문을 의미

# 람다식 문법

- 매개변수(하나인 경우) 자료형과 괄호 생략하기

```
str -> {System.out.println(str);}
```

- 매개 변수가 두 개인 경우 괄호를 생략할 수 없음

```
x, y -> {System.out.println(x + y);} // 잘못된 형식
```

- 중괄호 안의 구현부가 한 문장인 경우 중괄호 생략

```
str -> System.out.println(str);
```

- 중괄호 안의 구현부가 한 문장이라도 return문은 중괄호 생략할 수 없음

```
str -> return str.length( ); // 잘못된 형식
```

- 중괄호 안의 구현부가 반환문 하나라면 return과 중괄호 모두 생략

```
(x, y) -> x + y      // 두 값을 더하여 반환함  
str -> str.length( ) // 문자열의 길이를 반환함
```

# 람다식 사용하기 예제

- 두 수중 더 큰 수를 반환하는 람다식 구현 활용
- 함수형 인터페이스 선언하기

```
public interface MyNumber {
```

```
    int getMax(int num1, int num2);
```

추상 메서드 선언

```
}
```

- 람다식 구현과 호출

```
public class TestMyNumber {
```

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

```
        MyNumber max = (x, y) -> (x >= y) ? x : y; // 람다식을 인터페이스형 max 변수에 대입
```

```
        System.out.println(max.getMax(10, 20)); // 인터페이스형 변수로 메서드 호출
```

```
    }
```

```
}
```

Console Problems

<terminated> TestMyNumber  
20



# 함수형 인터페이스

- 랴다식을 선언하기 위한 인터페이스
- 익명 함수와 매개 변수만으로 구현되므로 단 하나의 메서드만을 가져야 함
  - (두 개 이상의 메서드인 경우 어떤 메서드의 호출인지 모호해 짐)
- **@FunctionalInterface 애노테이션**
  - 함수형 인터페이스라는 의미, 여러 개의 메서드를 선언하면 에러남

```
1 package lambda;
2
3 @FunctionalInterface
4 public interface MyNumber {
5     |
6     int getMax(int num1, int num2);
7     int add(int num1, int num2);
8 }
9
```

메서드가 2개이므로 오류 발생

# 익명 객체를 생성하는 람다식

- 자바는 객체 지향 언어로 객체를 생성해야 메서드가 호출 됨
- 람다식으로 메서드를 구현하고 호출하면 내부에서 익명 클래스가 생성됨

```
StringConcat concat3 = new StringConcat( ) {  
    @Override  
    public void makeString(String s1, String s2) {  
        System.out.println( s1 + "," + s2 );  
    }  
};
```

- 람다식에서 외부 메서드의 지역변수는 상수로 처리 됨
- (지역 내부 클래스와 동일한 원리)

# 함수를 변수처럼 사용하는 람다식

- 프로그램에서 변수의 사용은...

변수를 사용하는 경우	예시
특정 자료형으로 변수 선언 후 값 대입하여 사용하기	<code>int a = 10;</code>
매개변수로 전달하기	<code>int add(int x, int y);</code>
메서드의 반환 값으로 반환하기	<code>return num;</code>

- 인터페이스형 변수에 람다식 대입

```
interface PrintString {  
    void showString(String str);  
}
```



```
s -> System.out.println(s)
```

```
PrintString lambdaStr = s -> System.out.println(s); // 인터페이스형 변수에 람다식 대입  
lambdaStr.showString("hello lamda_1");
```

# 함수를 변수처럼 사용하는 람다식

- 매개변수로 전달하는 람다식

```
interface PrintString {  
    void showString(String str);  
}
```

람다식을 인터페이스형 변수에  
대입하고 그 변수를 사용해 람다식  
구현부 호출

```
public class TestLambda {  
    public static void main(String[] args) {  
        PrintString lambdaStr = s -> System.out.println(s);  
        lambdaStr.showString("hello lamda_1");  
        showMyString(lambdaStr);  
    }
```

메서드의 매개변수로 람다식을 대입한 변수 전달

```
    public static void showMyString(PrintString p) {  
        p.showString("hello lamda_2");  
    }  
}
```

매개변수를 인터페이스형으로 받음

Console Prob

```
<terminated> TestLambda  
hello lambda_1  
hello lambda_2
```

# 함수를 변수처럼 사용하는 람다식

- 반환 값으로 쓰이는 람다식

```
interface PrintString {  
    void showString(String str);  
}  
  
public class TestLambda {  
    public static void main(String[] args) {  
        ...  
        PrintString reStr = returnString( ); // 변수로 반환받기  
        reStr.showString("hello ");          // 메서드 호출  
    }  
  
    public static void showMyString(PrintString p) {  
        p.showString("hello lamda_2");  
    }  
}
```

```
public static PrintString returnString( ) {  
    return s -> System.out.println(s + "world");  
}
```

람다식을 반환하는 메서드



```
<terminated> TestLambda [Java  
hello lambda_1  
hello lambda_2  
hello world
```

# 스트림 (stream)

- **자료의 대상과 관계 없이 동일한 연산을 수행**
  - 배열, 컬렉션을 대상으로 동일한 연산을 수행 함
  - 일관성 있는 연산으로 자료의 처리를 쉽고 간단하게 함
- **한 번 생성하고 사용한 스트림은 재사용 할 수 없음**
  - 자료에 대한 스트림을 생성하여 연산을 수행하면 스트림은 소모됨
  - 다른 연산을 위해서는 새로운 스트림을 생성 함
- **스트림 연산은 기존 자료를 변경하지 않음**
  - 자료에 대한 스트림을 생성하면 별도의 메모리 공간을 사용하므로
  - 기존 자료를 변경하지 않음
- **스트림 연산은 중간 연산과 최종 연산으로 구분 됨**
  - 스트림에 대해 중간 연산은 여러 개 적용될 수 있지만 최종 연산은 마지막에 한 번만 적용됨
  - 최종연산이 호출되어야 중간연산의 결과가 모두 적용됨
  - 이를 '지연 연산' 이라 함



# 스트림 연산 – 중간 연산

- **중간 연산 – filter(), map()**
  - 조건에 맞는 요소를 추출 (filter()) 하거나 요소를 변환 함(map())
- 문자열의 길이가 5 이상인 요소만 출력 하기

```
sList.stream( ).filter(s -> s.length( ) >= 5).forEach(s -> System.out.println(s));
```

스트림 생성

중간 연산

최종 연산

- 고객 클래스에서 고객 이름만 가져오기

```
customerList.stream( ).map(c -> c.getName( )).forEach(s -> System.out.println(s));
```

스트림 생성

중간 연산

최종 연산

# 스트림 연산 – 최종 연산

- 스트림의 자료를 소모 하면서 연산을 수행
  - 최종 연산 후에 스트림은 더 이상 다른 연산을 적용할 수 없음
  - `forEach()` : 요소를 하나씩 꺼내 옴
  - `count()` : 요소의 개수
  - `sum()` : 요소의 합
- 
- 이 외에도 많은 중간 연산과 최종 연산이 있음

# 스트림 생성하고 사용하기

- 정수 배열에 스트림 생성하고 사용하기

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

sum( ) 연산으로 arr 배열에 저장된 값을 모두 더함

```
        int sumVal = Arrays.stream(arr).sum( );
```

```
        int count = (int) Arrays.stream(arr).count( );
```

count( ) 연산으로 arr 배열의 요소 개수를 반환함

count( ) 메서드의 반환 값이 long이므로  
int형으로 변환

```
        System.out.println(sumVal);
```

```
        System.out.println(count);
```

```
    }
```

```
}
```

Console Problems

<terminated> IntArrayTest [J:

15

5

# 스트림 생성하고 사용하기

- ArrayList에 스트림 생성하고 사용하기

```
public class ArrayListStreamTest {  
    public static void main(String[] args) {  
        List<String> sList = new ArrayList<String>( );  
        sList.add("Tomas");  
        sList.add("Edward");  
        sList.add("Jack");
```

```
        Stream<String> stream = sList.stream( );
```

스트림 생성

```
        stream.forEach(s->System.out.print(s + " "));
```

배열의 요소를 하나씩 출력

```
        System.out.println( );
```

```
        sList.stream( ).sorted( ).forEach(s->System.out.println(s));
```

스트림 새로 생성

정렬

요소를 하나씩 꺼내어 출력

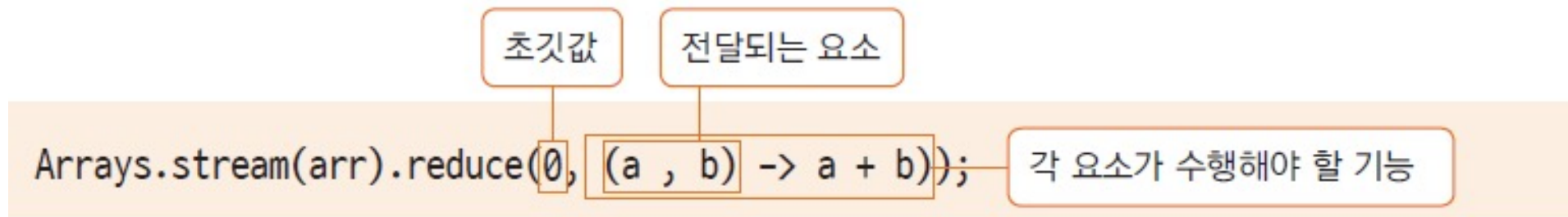
Console

<terminated> ArrayListStrea

Tomas Edward Jack  
Edward Jack Tomas

# reduce() 연산

- 정의된 연산이 아닌 프로그래머가 직접 지정하는 연산을 적용
- 최종 연산으로 스트림의 요소를 소모하며 연산 수행
- 배열의 모든 요소의 합을 구하는 reduce() 연산



- 두 번째 요소로 전달되는 람다식에 따라 다양한 기능을 수행
-

# reduce() 연산 예제

- 배열에 여러 문자열이 있을 때 길이가 가장 긴 문자열 찾기

```
import java.util.Arrays;
import java.util.function.BinaryOperator;
```

BinaryOperator를 구현한 클래스 정의

```
class CompareString implements BinaryOperator<String> {
    @Override
    public String apply(String s1, String s2) {
        if(s1.getBytes().length >= s2.getBytes().length) return s1;
        else return s2;
    }
}
```

reduce() 메서드가  
호출될 때 불리는 메  
서드, 두 문자열 길  
이를 비교

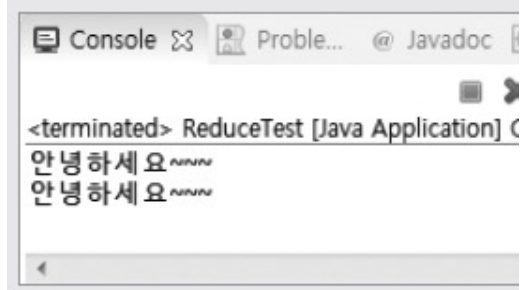


# reduce() 연산 예제

```
public class ReduceTest {  
    public static void main(String[ ] args) {  
        String[] greetings = {"안녕하세요~~~", "hello", "Good morning", "반갑습니다^^"};  
        System.out.println(Arrays.stream(greetings).reduce("", (s1, s2) -> {  
            if(s1.getBytes().length >= s2.getBytes().length)  
                return s1;  
            else return s2; }));  
  
        String str = Arrays.stream(greetings).reduce(new CompareString( )).get( );  
        System.out.println(str);  
    }  
}
```

람다식을 직접 구현하는 방법

BinaryOperator를 구현한 클래스 사용



감사합니다.

끝