람다식

megapie2021@gmail.com

람다식

이름 없는 익명함수
메서드를 따로 만들지 않고 한 줄 코드를 이용하여 함수를 사용

```
리턴타입 메서드이름(매개변수)
{
    명령문1
    명령문2

}
 =>  (매개변수,....)->{명령문; 명령문2,....}v
```

```
int add(int a, int b)
{
     return a+b;
}
```

```
(int a, int b)-> a+b;
(a, b)-> a+b;
(a,b)->{
    System.out.println("ee");
          return a+b;
   };
```

람다식

매개변수 : (int a)->System.out.println(“hello”);

매개변수 타입을 생략 : (a,b)->Systm.out.prntln(a+b);

매개변수 생략 : ()->{}

매개변수가 1개일때 ()생략: su->su+2;

메서드 에서 return 만 있을 경우 : ()->10 // return 10;

해당 메서드가 2줄 이상 : () ->{
System.out.println(“aaa”);
return 10;
}

다양한 람다식

```
@FunctionalInterface
interface Sample1{
      void print();
}
```

```
Sample1 obj1=
     ()->System.out.println("hello");

a.print();
```

```
@FunctionalInterface
interface Sample1{
    void namePrint(String name);
}
```

```
Sample1 obj2=
   (name)->System.out.println(name);

   obj.namePrint("hong");
   obj.namePrint("kim");
```

```
@FunctionalInterface
interface Sample1{
 int addJumsu(int kor, int eng, int
math);
}
```

```
Sample1 obj3=
   (kor, eng, math)->   kor+eng+math;

  System.out.println(obj3.addJumsu(50,
70, 90));
```

다양한 람다식

```
@FunctionalInterface
interface Sample1{
        float avgJumsu(int kor, int eng, int math);}
}
```

```
Sample1 obj4=(kor, eng, math)->{
        System.out.printf("%d %d %d₩n", kor, eng,math);
   return (kor+eng+math)/3.0f;
  };


System.out.println(obj4.avgJumsu(70, 80, 70));
```

다양한 람다식

```
@FunctionalInterface
interface Fun{
    void add(int a, int b);
}

public class Lamda1 {
    public static void main(String[] args) {
      Fun f=(int a, int b)-> System.out.println(a+b);
        f.add(10, 20);
    }
}
```

함수형 인터페이스

Runnable	void run()
Supplier	T get()
Consumer	void accept()
Function<T,R>	R apply(T t)
Predict<T>	boolean test(T t)

```
@FunctionalInterface
interface Sample
{
public abstract int max(int a, int b);

}

public class Test1 {
public static void main(String[] args) {
// TODO Auto-generated method stub

        Sample s=(a, b)-> a+b;
        int su=s.max(10, 20);
        System.out.println(su);
}

}
```

```
@FunctionalInterface
interface Sample
{
 void max(int a, int b);
}

public class Test1 {
public static void main(String[] args) {
// TODO Auto-generated method stub
      Sample s=(int a, int b)->System.out.println(a+b);
   s.max(20, 30);
}
}
```

```
@FunctionalInterface
interface A
{
  default void prt()
  {
  System.out.println("ttt");
  }
  void tt();
}

public class Test2 {
public static void main(String[] args) {
// TODO Auto-generated method stub
  A ins=()->System.out.println("test");
   ins.prt();
   ins.tt();
}
}
```

함수적 인터페이스

자바 8부터 지원
java.util.function 패키지에 포함
한 개의 추상 메소드를 가지는 인터페이스를 람다식으로 표현가능

Runnable
void run() : 리턴타입이 없고 반환값도 없음

```
new Thread(()->{
        System.out.println("hello");
        }).start();
```

컬렉션과 스트림

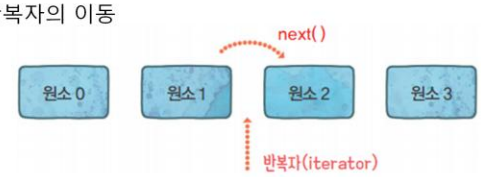
컬렉션의 데이터 반복 처리

컬렉션의 종류에 관계 없이 반복자를 이용하면 컬렉션에 포함된 객체를 순차적으로 순회

Iterator 인터페이스가 제공하는 주요 메서드

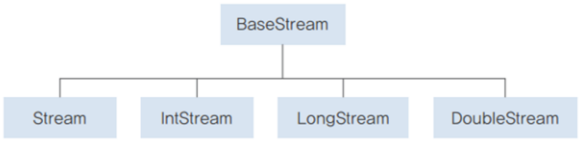
메서드	설명
boolean hasNext( )	다음 원소의 존재 여부를 반환한다.
E next( )	다음 원소를 반환한다.
default void remove( )	마지막에 순회한 컬렉션의 원소를 삭제한다.

반복자의 이동



컬렉션과 스트림

스트림의 종류



Stream은 객체 원소로 구성된 스트림 데이터를 처리
컬렉션이나 배열에서 스트림 구현 객체를 얻는 주요 메서드

메서드	설명
컬렉션	Stream<T> Collection.stream( )
배열	Stream<T> Arrays.stream(T[ ])
	IntStream Arrays.stream(int[ ])
	DoubleStream Arrays.stream(double[ ])

주요 정적 메서드

```
static <T> Stream<T>  of(T... values)
```

```
ArrayList<String> arr=new ArrayList<>();
  arr.add("lee");
  arr.add("park");
  arr.add("kim");
  arr.add("hong");

  arr.forEach((String index)->System.out.println(index));

IntStream is=IntStream.of(10,2,6,4,5);
        is.forEach((int su)->System.out.println(su));

IntStream is=IntStream.of(10,2,6,4,5);
            is.filter((int index)-> index>=5)
            .forEach((int su)->System.out.println(su));
```

```
class Person
{
 private String name;
 private int age;
 public Person(String name, int age)
 {
 this.name=name;
 this.age=age;
 }
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
}
@Override
public String toString() {
return "Person [name=" + name + ", age=" + age +
     "]";
}
}
```

```
ArrayList<Person> arr=new
ArrayList<>();
      arr.add(new Person("hong",10));
      arr.add(new Person("park",4));
      arr.add(new Person("kim",8));
      arr.add(new Person("kang",9));

    Stream<Person> s=  arr.stream();
  s.filter((Person p)->p.getAge()>4)
          .forEach((Person p)-
>System.out.println(p));
```

```
ArrayList<String> arr=new ArrayList<>();
    arr.add("hong");
    arr.add("park");
    arr.add("hong");
    arr.add("kim");

    Stream<String> s=arr.stream();
s.distinct()
 .forEach((String index)->System.out.println(index));
```

```
ArrayList<String> arr=new ArrayList<>();
    arr.add("hong");
    arr.add("park");
    arr.add("hong");
    arr.add("kim");

    Stream<String> s=arr.stream();
s.sorted().forEach((String index)->System.out.println(index));
```

```
ArrayList<Student1> arr=new ArrayList<>();
    arr.add(new Student1("aa1",60));
    arr.add(new Student1("aa2",40));
    arr.add(new Student1("aa3",90));
    arr.add(new Student1("aa4",100));

   Stream<Student1> s=arr.stream();
   OptionalDouble avg=  s.mapToInt((Student1 s1)->s1.getJumsu()).average();
System.out.println( avg.getAsDouble());
```