메소드 만드는이유

1. 유지보수를위해서

2. 가독성이 좋아지니까

3. 캡술화 (여러 곳에서 중복된 코드를 작성할 필요 없음)

메소드

로직만 만들어 놓고 그때 그때 데이터를 주면

메소드가 알아서 결과값을 반환 하는 방식

객체란

동일한 성질의 데이터와 메소드를 한곳에 모아두고

필요한 곳에서 언제든지 이용할 수 있게 만들어 놓은 덩어리

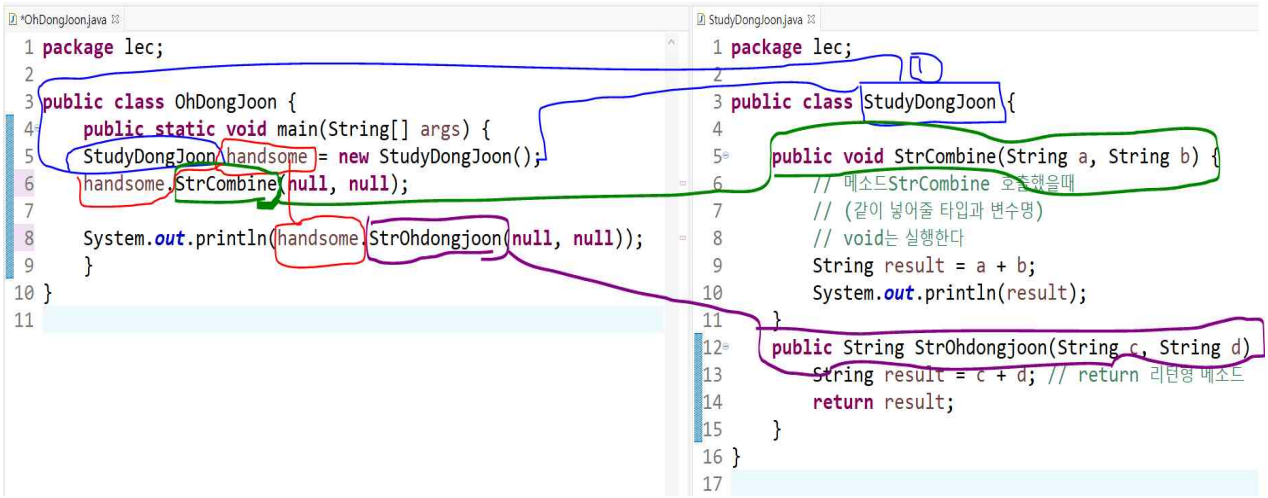
메소드의 이해

메소드란 ? 작업을 수행하기 위한 명령문의 집합

어떤 값을 입력받아서 처리하고 그결과를 돌려준다

(입력 받는 값이 없을 수도 있고 결과를 돌려주지 않을 수도 있다)

메소드의 장점과 작성지침 ; 반복적으로 수행되는 여러 문장을 메소드로 작성한다.



```java
// *OhDongJoon.java
package lec;

public class OhDongJoon {
    public static void main(String[] args) {
    StudyDongJoon handsome = new StudyDongJoon();
    handsome.StrCombine(null, null);

    System.out.println(handsome.StrOhdongjoon(null, null));
    }
}
```

```java
// StudyDongJoon.java
package lec;

public class StudyDongJoon {

    public void StrCombine(String a, String b) {
        // 메소드StrCombine 호출했을때
        // (같이 넣어줄 타입과 변수명)
        // void는 실행한다
        String result = a + b;
        System.out.println(result);
    }
    public String StrOhdongjoon(String c, String d)
        String result = c + d; // return 리턴형 메소드
        return result;
    }
}
```

class StudyDongJoon 메소드 클래스

1)public void StrCombine(변수)(문자형 ,a ,b 변수 선언{

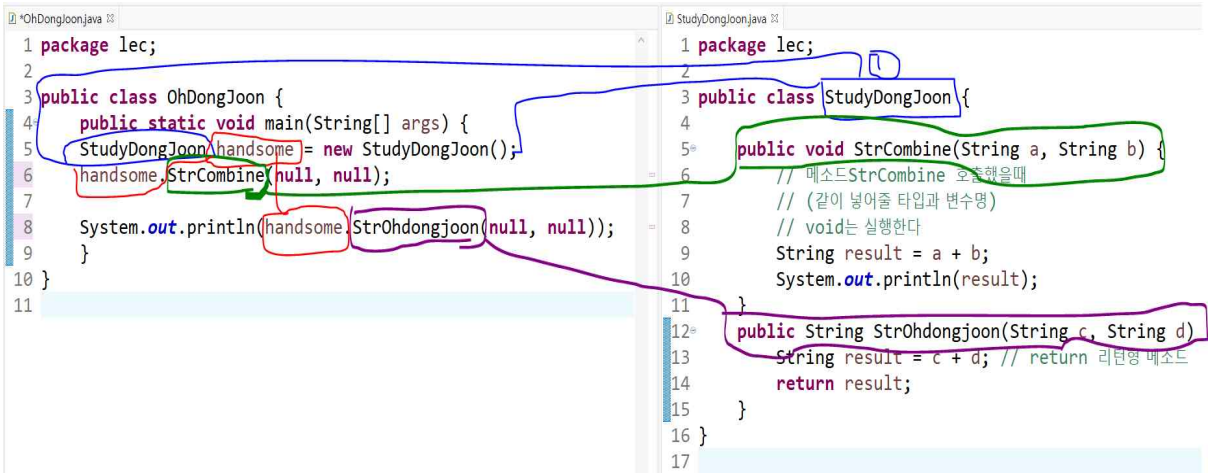
문자형 타입 result 변수 = a + b ; 넣어주기

출력(result)

2) public String StrOhdongjoon(변수) (문자형 , c, d 변수선언)

문자형 result(변수) = c + d 넣어주기

return result ; 리턴형



```java
package lec;

public class OhDongJoon {
    public static void main(String[] args) {
    StudyDongJoon handsome = new StudyDongJoon();
    handsome.StrCombine(null, null);

    System.out.println(handsome.StrOhdongjoon(null, null));
    }
}
```

```java
package lec;

public class StudyDongJoon {

    public void StrCombine(String a, String b) {
        // 메소드StrCombine 호출했을때
        // (같이 넣어줄 타입과 변수명)
        // void는 실행한다
        String result = a + b;
        System.out.println(result);
    }
    public String StrOhdongjoon(String c, String d)
        String result = c + d; // return 리턴형 메소드
        return result;
    }
}
```

출력할 페키지 OhDongJoon 생성

메인함수

StudyDongJoon타입에 handsome(변수) = new(새로운) StudyDongJoon 한마디로 StudyDongJoon 클래스를 함축해서 handsome(변수)에 넣어준 거다

handsome.StrCombine(null,null);

handsome에 StrCombine을 호출 했다 하지만 아직 아무런 값을 넣어주지 않아 null로 표시되있다.

1. 메소드 프로그램의 한계

- 메소드를 이용하면 로직의 재사용이 가능하여 개발을 효율적으로 할 수 있었습니다.
- 절차지향언어에서 중복된 로직을 메소드의 사용으로 인해 효율성을 높일 수 있다.
- 하지만, 메소드만 가지고는 많은 양의 로직을 처리하기에는 한계가 있습니다.
- 한 문서 내에 메소드의 수가 많아질 경우 추후 유지 보수에 많은 어려움이 발생됩니다.

클래스의 기초적인 코딩방법

(cf) 캡슐화(Encapsulation) : 객체가 포함한 속성과 메서드는 객체간의 관계에 있어서 감추거나 권한에 따라 접근이 가능하게 처리하는 것을 말한다. 여기에 사용되는 keyword로 접근제어자(access modifier)가 있다

1. 클래스 제작

```
package com.ch.ex;
public class ExClass {
    private 자료형 인스턴스변수( = 속성 = 필드)명;
    public ExClass(){      }
    public method(){
    . . .
    }
}
```

① 패키지명

② 클래스명

③ **데이터(인스턴스 변수=멤버변수, 필드)** : 이 데이터는 생성자나 setter를 이용해서 초기화하지 않으면 객체는 null, 숫자는 0, boolean은 false로 초기화되어 들어간다

④ 생성자함수 : 클래스명과 똑같이 리턴타입이 없는 메소드를 생성자라 하며 처음 클래스형 객체를 만들때 호출된다. 모든 클래스는 반드시 하나 이상의 생성자가 있어야 한다. 만약 하나도 없으면 JVM이 디폴트 생성자를 만들어 준다(new 연산자로 호출되는 메서드)

2. 생성자의 이해 : 생성자는 매개변수 있는 생성자와 매개변수 없는 생성자 등 여러 종류의 생성자를 가질 수 있다. 생성자가 없을 때는 디폴트 생성자가 컴파일러 단계에서 자동 생성한다. 한 개 이상의 생성자가 있으면 디폴트 생성자는 자동 생성되지 않는다.