7_26_2022__java_Class, Method

Step003_Class\MainClass03.java

~Step04_Method\MainClass03.java

Class

1. 객체의 설계도 역할

   객체를 생성했을 때 어떤 field와 어떤 method를 가질 지 설계하는 역할

   - field → 데이터의 저장소
   - method → 동작, 기능

   객체는 (new)만들어졌을 때 heap 영역에 저장 됨

   1. datatype 역할도 한다.

      ```
      new Member();  ← 참조값(key 값) 으로 바뀐다.
      ```

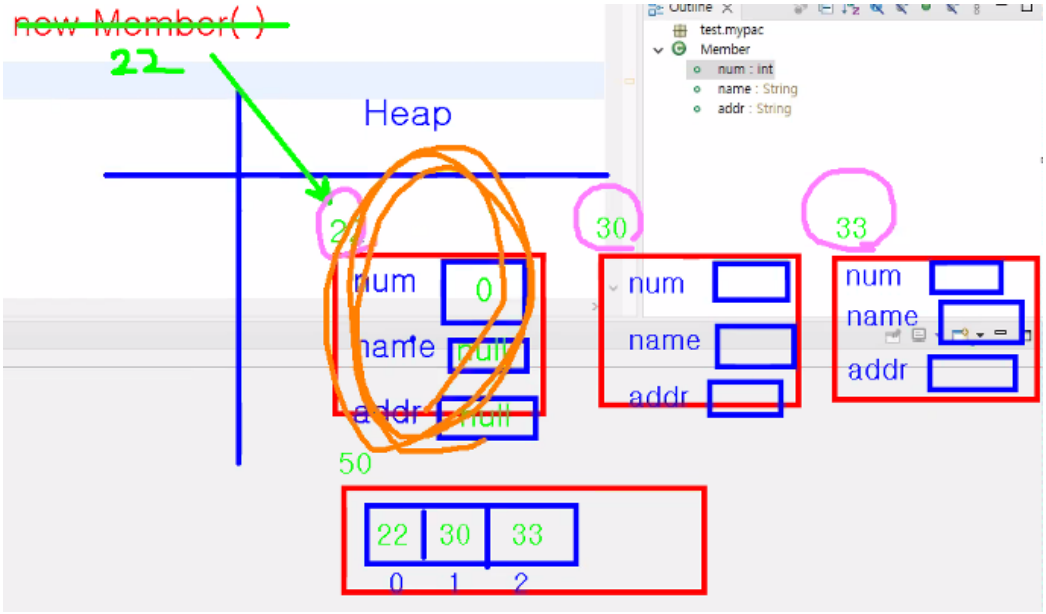
      ```
      a = new Member(); ← a 라는 변수에 저장하기
      ```

      ```
      Member a = new Member(); ← new 이기 때문에 Member 타입이 된다.
      ```

      - 종류

        public class → 공개 클래스 어디서든 접근이 가능하다.

        class → 접근 지정자를 적지 않으면, 같은 패키지 내에서만 접근할 수 있다. (default)

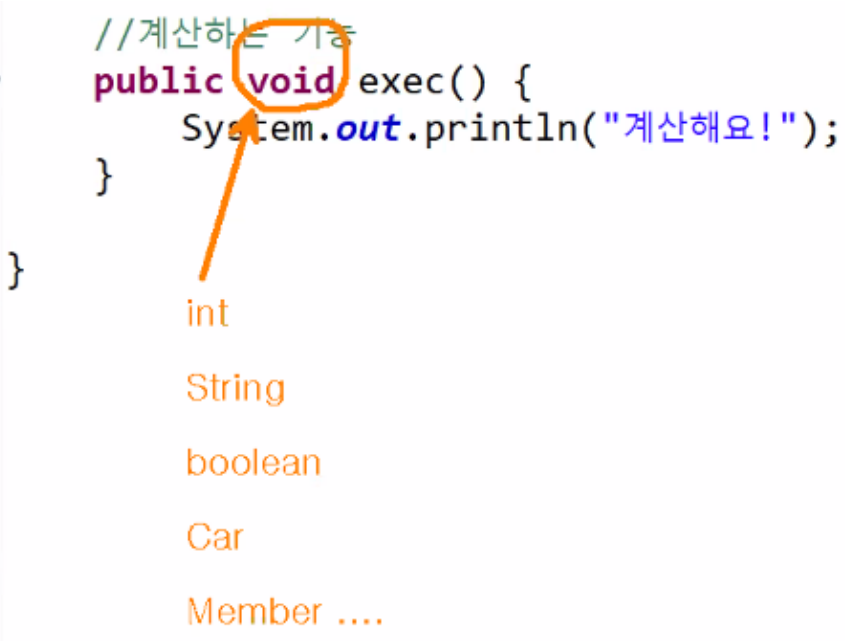


22, 30, 33 Member type - 50 list type

method도 접근 지정자를 정해줄 수 있다.

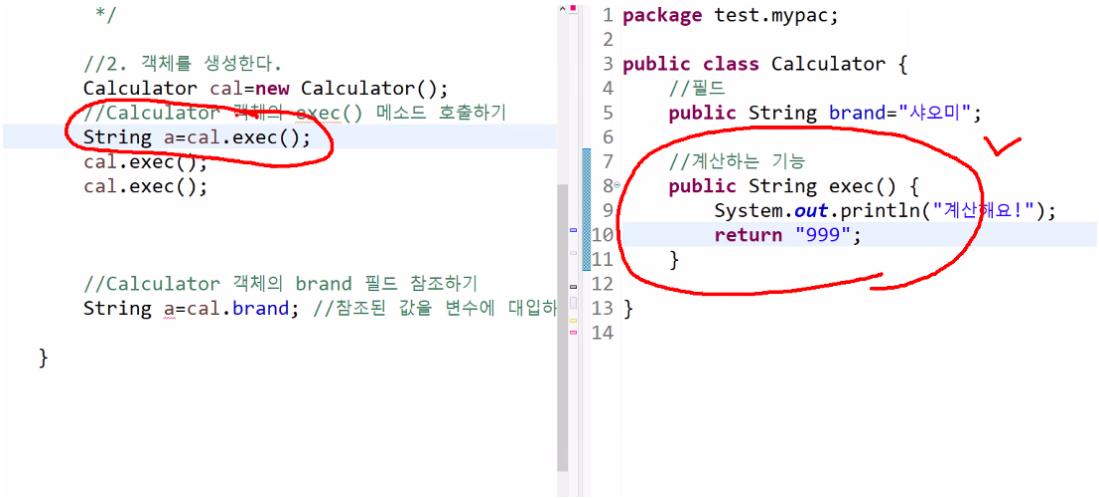
void 가 나오는 method는 동작하는 목적이 아니고 call 만 하는 method

```
Calculator cal=new Calculator();
  cal.exec();
  cal.exec();
  cal.exec();
```



부르고 싶은 type 을 적는다.

값이 return 되는 type을 사용한 예시



```
 */

//2. 객체를 생성한다.
Calculator cal=new Calculator();
//Calculator 객체의 exec() 메소드 호출하기
String a=cal.exec();
cal.exec();
cal.exec();



//Calculator 객체의 brand 필드 참조하기
String a=cal.brand; //참조된 값을 변수에 대입하

}
```

```
1  package test.mypac;
2
3  public class Calculator {
4      //필드
5      public String brand="샤오미";
6
7      //계산하는 기능
8      public String exec() {
9          System.out.println("계산해요!");
10         return "999";
11     }
12
13 }
14
```

String type은 return값이 필요해 return"999"를 해 주고 불러오는 쪽에서도 String type을 적어줌

- method 에서 자기 자신의 field 참조값을 가져올 수 있다.

```
package test.main;

import test.mypac.Member;

public class MainClass03 {
  public static void main(String[] args) {
    System.out.println("main method가 시작되었습니다.");
    Member mem1=new Member();
    mem1.cNumber=1;
    mem1.cName="두부";
    mem1.cAddr="페어펙스";

    Member mem2=new Member();
    mem2.cNumber=2;
    mem2.cName="유키";
    mem2.cAddr="하이데저트";

    //mem1, mem2 에 들어있는 참조값을 이용해서 showInfo() 메소드 호출하기
    mem1.showInfo();
    mem2.showInfo();
  }
}
```

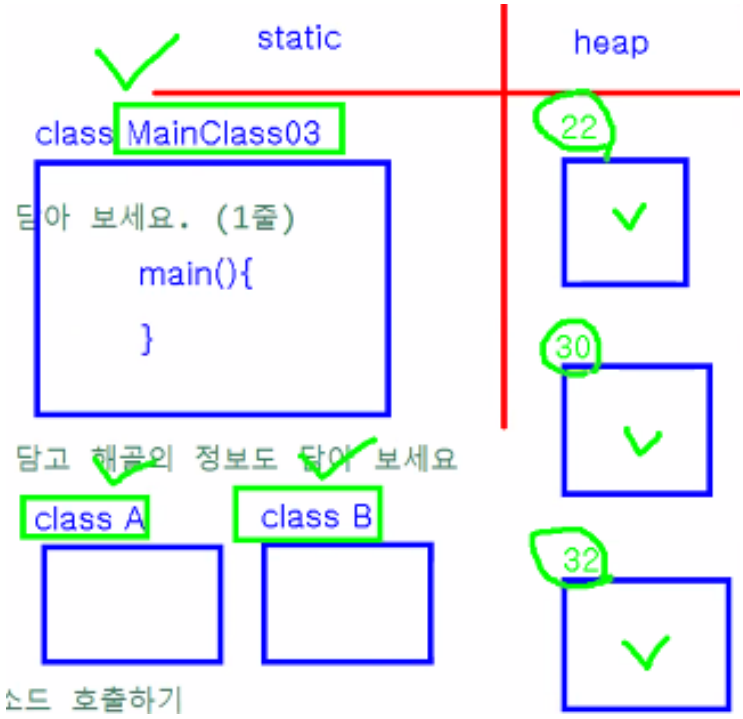
```
package test.mypac;

public class Member {
public int cNumber;
public String cName;
public String cAddr;

  //method
  public void showInfo() {
```

```
    System.out.println("번호: "+this.cNumber+" 이름: "+this.cName+" 주소: "+this.cAddr);
  }
}
```

- main method는 static 영역의 class 안에 들어 있다.

```
public static void main(){}
```

  - static 영역에서 class는 하나씩만 만들어진다.
  - 참조값으로 구분되는 heap 영역과 달리 class 이름으로 구분된다.
  - 정말 하나만 만들어서 쓸 것이라면 static을 사용한다.
  - field 를 선언할 때도 static을 사용할 수 있다.

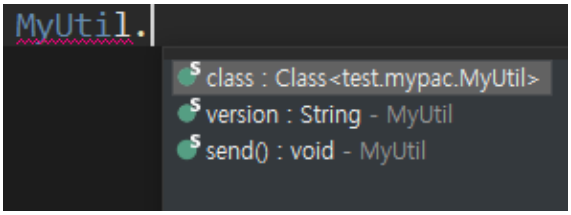


3. static field, static method를 static 영역에서 포함하고 있는 열할을 한다.

   static이 붙은 필드나 메소드는 불러올 때 new를 붙일 필요 없다.

```
package test.mypac;

public class MyUtil {
  //field
  public static String version="1.0";
```

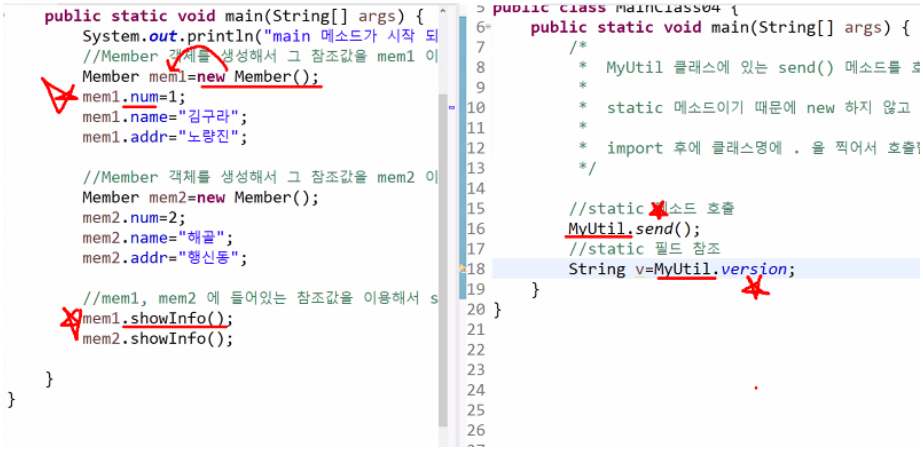
```
  //method
  public static void send() {
    System.out.println("전송합니다.");
  }
}
```



아이콘 위에 s는 스태틱임을 알려준다.

```
package test.main;
import test.mypac.MyUtil;
public class MainClass04 {
  public static void main(String[] args) {
    // static 메소드 호출
    MyUtil.send();
    //static 필드 참조
    String v=MyUtil.version;
  }
}
```

비교



static은 객체와 상관이 없다.

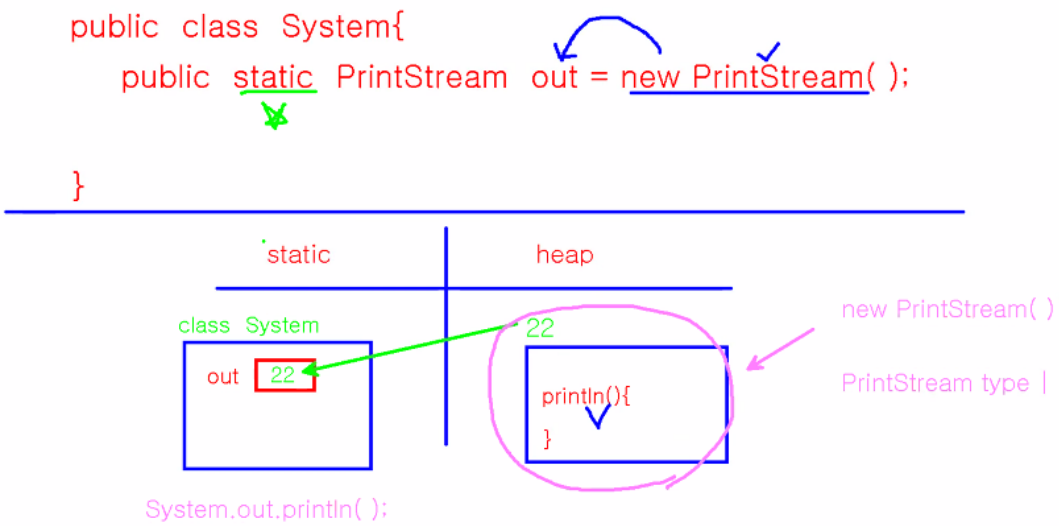
- 앞으로 직접 사용할 때.
  - java에서 기본으로 제공되는 클래스를 import 해서 사용하기
  - 추가로 필요한 유틸리티를 인터넷에서 다운받아 import후 사용하기

    - 직접 만들어서 사용하기
- java에서 기본 제공해주는 클래스 중에 java.lang 패키지 안에 속해 있는 String, System 등의 클래스는 import 하지 않아도 기본 클래스 처럼 사용할 수 있다.

- 예제

  System은 클래스, out은 printStream타입의 필드, println()은 기능

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



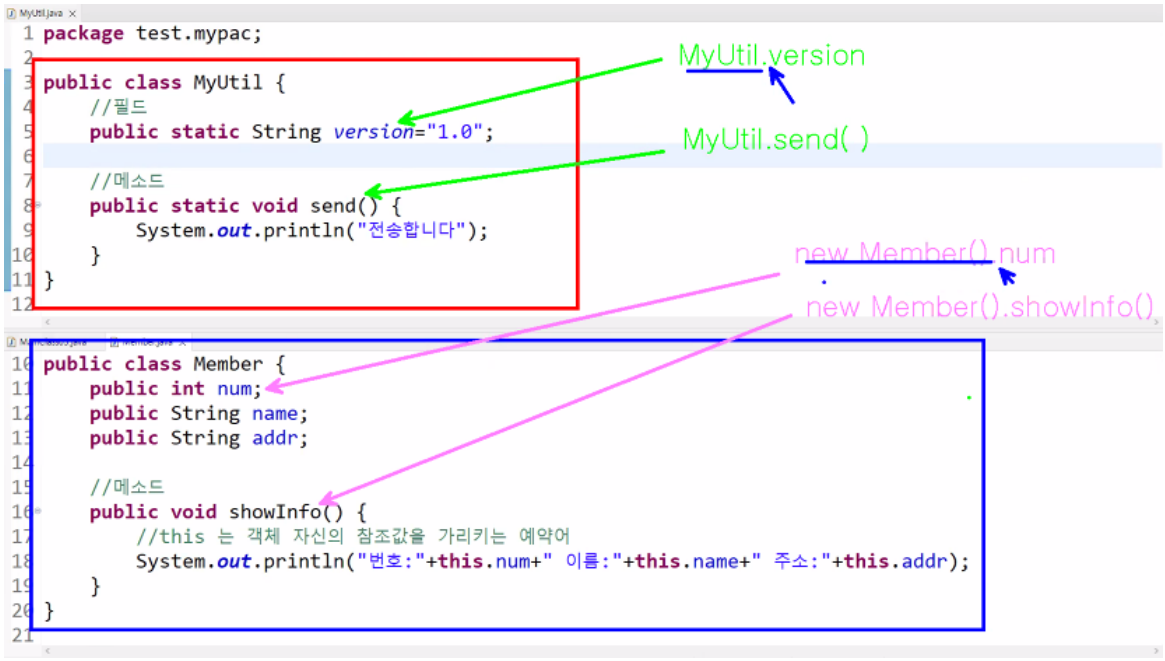
java 로 프로그래밍을 하는 방법

- heap 영역에 있는 객체의 필드나 메소드를 활용해서 원하는 동작을 한다.
- static 영역에 있는 클래스의 static 필드나 static 메소드를 황용해서 원하는 동작을 한다.

💡 java 학습 방법

- 특정 작업을 할 때 어떤 type의 객체가 필요한지를 학습해야 한다.
- 어떤 type 객체의 참조값을 어떻게 얻어내는지를 학습해야 한다.
  - 필요한 객체를 직접 new 하거나
  - 이미 생성된 객체를 참조(필드참조)하거나
  - 메소드를 호출해서 리턴되는 객체를 주로 활용한다.

💡 java를 배우는 것은 어떤 기능을 할 때 어떤 객체를 사용하면 되는지를 배우는 것



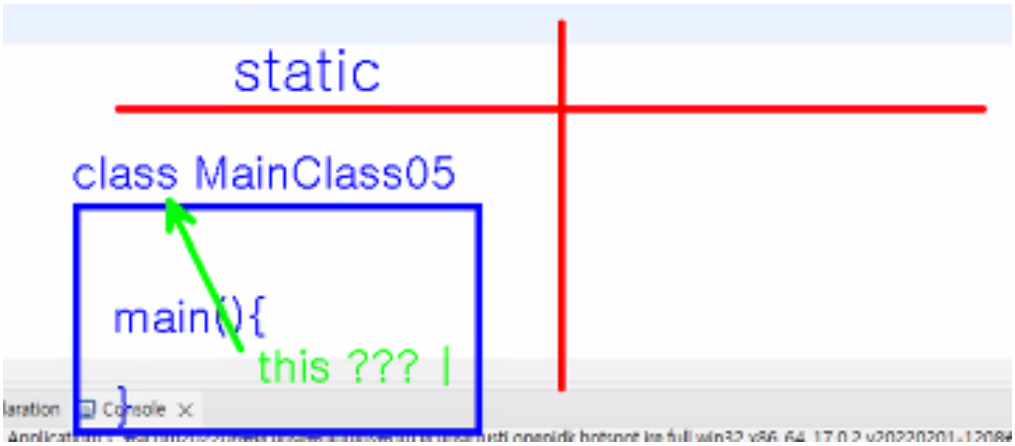

```
package test.mypac;

public class MyUtil {
    //필드
    public static String version="1.0";

    //메소드
    public static void send() {
        System.out.println("전송합니다");
    }
}
```

```
public class Member {
    public int num;
    public String name;
    public String addr;

    //메소드
    public void showInfo() {
        //this 는 객체 자신의 참조값을 가리키는 예약어
        System.out.println("번호:"+this.num+" 이름:"+this.name+" 주소:"+this.addr);
    }
}
```

non-static은 참조값이 있어야 한다.

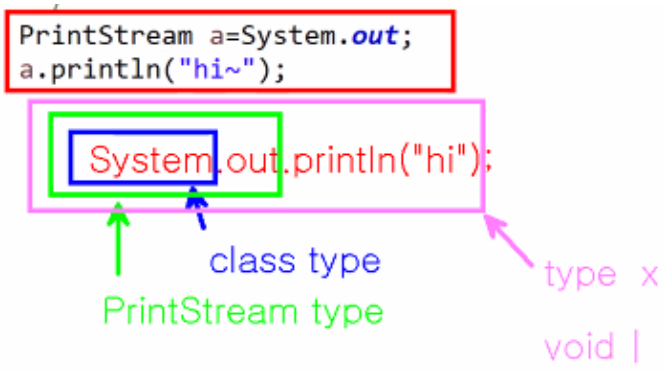
```
package test.main;

public class MainClass05 {
```

```
  public int num=999;

  public static void main() {
    System.out.println("num: "+this.num);
  }
}
```



this는 안쪽에서만 영향을 미치는데 num은 밖에 있음

```
package test.main;

import java.io.PrintStream;

public class MailClass06 {
public static void main(String[] args) {
  PrintStream a=System.out;
  a.println("hi~");
  } // == System.out.println("hi~");
}
```



예시

[랜덤한 정수를 하나 얻어내서 콘솔창에 출력하는 프로그래밍을 하고 싶다]

- 필요한 객체
  1. 랜덤한 정수를 만들어주는 객체→new Random( );
  2. 콘솔창에 문자열을 출력해주는 객체→System.out

```
package test.main;
import java.util.Random;
public class MainClass07 {
public static void main(String[] args) {
  //Random type 참조값이 담길 수 있는 ran 이라는 이름의 빈 지역변수만들기
  Random ran=null;
  // Random 객체를 생성해서 그 참조값을 ran에 대입하기
  ran=new Random();
  //참조값에 . 찍어서 nextInt( ) 메소드를 호출하고 메소드가 리턴해주는 값을 지역변수 ranNum에 담기.
  int ranNum=ran.nextInt();
  // ranNum 변수에 담긴 값을 콘솔창에 출력해 보기
  System.out.println(ranNum);
  }
}
```

[키보드로부터 문자열을 입력 받아서 콘솔창에 출력하는 프로그래밍]

- 필요한 객체
  1. 키보드로 부터 문자열을 입력받는 기능을 가지고 있는 객체→new Scanner( );
  2. 콘솔창에 문자열을 출력해주는 객체→System.out

```
package test.main;

import java.util.Scanner;

public class MainClass08 {
  public static void main(String[] args) {
    //scanner 객체를 생성해서 scan 이라는 지역변수에 참조값을 담기
    Scanner scan=new Scanner(System.in);
    //콘설에 입력한 문자열을 읽어와서 변수에 담기
    String line=scan.nextLine();
    System.out.println("line: "+line);
  }
}
```

Method

메소드를 만들 때 고려해야 하는 것

1. 접근 지정자
2. static or non static
3. return type
4. 메소드명
5. 메소드에 전달하는 인자의 갯수와 data type

종류

- public → 이 메소드는 어디서든 접근 가능
- voud → 이 메소드는 어떤 값도 리턴하지 않음
- walk → 메소드명
- walk() → 이 메소드는 어떤 값도 인자로 전달받지 않음.

생성자

메소드와 비슷하게 생겼지만 클래스 명과 똑같아야함

접근지정자 부착 가능