생성자 순서

```java
class Solution_1 {
    String name;

    Solution_1() {}
    Solution_1(String _name) {
        this.name = _name;
    }

}
```

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

        Solution_1 d1 = new Solution_1( _name: "전승리");

    }
}
```

참조 변수 d1 = new 연산자를 통한 객체 생성(객체 초기화)호출

생성순서

1. 참조변수 생성
2. new 연산자를 통해 new연산자가 객체생성
3. 생성자는 만들어진 객체 초기화,호출
4. 해당 클래스에 생성자 iv에 들어감
5. 대입되어서 주소값에 저장됨

오버로딩

```java
ConsTest(int a) {
    System.out.println("안녕 나는 ConsTest(int a)라고해");
    age = a;
}
ConsTest(String m) {
    System.out.println("안녕 나는 ConsTest(int f)라고해");
    name = m;
}
ConsTest(int a, String n) {
    System.out.println("안녕 나는 ConsTest(int a, String n)라고해");
    name = n;
    age = a;
}

public class WhyConstructorTest {
    public static void main(String[] args) {
        ConsTest ct1 = new ConsTest( a: 1);
        ConsTest ct3 = new ConsTest( m: "출력값1");
        ConsTest ct4 = new ConsTest( a: 10, n: "출력값2");
```

오버로딩은 같은이름의 메서드를 여러개 정의하는것이다.
생성자의 이름이 같지만 각각의 매개변수가 다르기때문에 생성이가능하다.
또한 생성자는 리턴값이 없기때문에 void를 사용 하지 않는다.

생성자와 기본생성자

1. 생성자가 없는 경우

```java
class Solution_1 {
    String name;
    int value;

}
```

기본 생성자가 필요하지않다.
생성자가 하나도 없는경우는 컴파일러가 자동 추가 해주기때문이다.

2. 생성자만 있는경우

```java
class Solution_1 {
    String name;
    int value;

    Solution_1(String _name) {
        this.name = _name;
    }

}
```

생성자O, 기본생성자X 인경우 오류가 발생한다.

```java
public class ErrorPractice {
    public static void main(String[] args) {
        Solution_1 d = new Solution_1();
    }
}
```

3. 생성자와 기본생성자가 있는경우

```java
class Solution_1 {
    String name;
    int value;

    Solution_1() {}
    Solution_1(String _name) {
        this.name = _name;
    }

}
```

생성자O, 기본생성자O 인경우 오류가 없다.

```java
lic class ErrorPractice {
    public static void main(String[] args) {
        Solution_1 d = new Solution_1();
    }
}
```

기본생성자 = 매개변수가 없는 생성자
기본생성자를 만드는 습관을 들여야한다.
this.name에 this는 클래스의 변수 가르키는것으로 매개변수와의 혼동 방지.

생성자를 사용한 방법

```java
class Solution_2 {
    int value;
    String name;

    public Solution_2() {}

    Solution_2(String a, int b) {
        this.name = a;
        this.value = b;
    }
}
public class ErrorPractice {
    public static void main(String[] args) {

        Solution_2 d2 = new Solution_2( a: "이름", b: 500);
        System.out.println(d2.name);
        System.out.println(d2.value);

    }
```

동일 출력 값

```
이름
500
```

생성자 사용시 getter,setter 보다 코드사용량이 훨씬 적다.

getter,setter를 사용한 방법

```java
class Solution_2 {
    int value;
    String name;
    public Solution_2() {}

    public int getValue() {
        return value;
    }

    public void setValue(int value) {
        this.value = value;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
public class ErrorPractice {
    public static void main(String[] args) {

        Solution_2 d2 = new Solution_2();
        d2.setName("이름");
        d2.setValue(500);

        System.out.printf("%s\n%d",d2.getName(),d2.getValue());
    }
}
```