2. 기본형과 참조형

#1.인강/0.자바/2.자바-기본

기본형 vs 참조형1 - 시작

자바에서 참조형을 제대로 이해하는 것은 정말 중요하다.
지금까지 기본형과 참조형에 대해서 조금씩 보았다. 이번에는 기본형과 참조형에 대해서 더 깊이있게 알아보고 확실하게 정리해보자.

변수의 데이터 타입을 가장 크게 보면 기본형과 참조형으로 분류할 수 있다. 사용하는 값을 변수에 직접 넣을 수 있는 기본형, 그리고 이전에 본 `Student student1`과 같이 객체가 저장된 메모리의 위치를 가리키는 참조값을 넣을 수 있는 참조형으로 분류할 수 있다.

- 기본형(Primitive Type): `int`, `long`, `double`, `boolean`처럼 변수에 사용할 값을 직접 넣을 수 있는 데이터 타입을 기본형이라 한다.
- 참조형(Reference Type): `Student student1`, `int[] students`와 같이 데이터에 접근하기 위한 참조(주소)를 저장하는 데이터 타입을 참조형이라 한다. 참조형은 객체 또는 배열에 사용된다.

쉽게 이야기해서 기본형 변수에는 직접 사용할 수 있는 값이 들어있지만 참조형 변수에는 위치(참조값)가 들어가 있다. 참조형 변수를 통해서 뭔가 하려면 결국 참조값을 통해 해당 위치로 이동해야 한다.

기본형 vs 참조형 - 기본

- 기본형은 숫자 `10`, `20`과 같이 실제 사용하는 값을 변수에 담을 수 있다. 그래서 해당 값을 바로 사용할 수 있다.
- 참조형은 실제 사용하는 값을 변수에 담는 것이 아니다. 이름 그대로 실제 객체의 위치(참조, 주소)를 저장한다. 참조형에는 객체와 배열이 있다.
    - 객체는 `.`(dot)을 통해서 메모리 상에 생성된 객체를 찾아가야 사용할 수 있다.

- 배열은 `[]`를 통해서 메모리 상에 생성된 배열을 찾아가야 사용할 수 있다.

기본형 vs 참조형 - 계산

- 기본형은 들어있는 값을 그대로 계산에 사용할 수 있다.
  - 예) 더하고 빼고, 사용하고 등등, (숫자 같은 것들은 바로 계산할 수 있음)
- 참조형은 들어있는 참조값을 그대로 사용할 수 없다. 주소지만 가지고는 할 수 있는게 없다. 주소지에 가야 실체가 있다!
  - 예) 더하고 빼고 사용하고 못함! 참조값만 가지고는 계산 할 수 있는 것이 없음!

기본형은 연산이 가능하지만 참조형은 연산이 불가능하다.

```
int a = 10, b = 20;
int sum = a + b;
```

기본형은 변수에 실제 사용하는 값이 담겨있다. 따라서 `+`, `-`와 같은 연산이 가능하다.

```
Student s1 = new Student();
Student s2 = new Student();
s1 + s2 //오류 발생
```

참조형은 변수에 객체의 위치인 참조값이 들어있다. 참조값은 계산에 사용할 수 없다. 따라서 오류가 발생한다.

물론 다음과 같이 `.`을 통해 객체의 기본형 멤버 변수에 접근한 경우에는 연산을 할 수 있다.

```
Student s1 = new Student();
s1.grade = 100;
Student s2 = new Student();
s2.grade = 90;
int sum = s1.grade + s2.grade; //연산 가능
```

쉽게 이해하는 팁

기본형을 제외한 나머지는 모두 참조형이다.

- 기본형은 소문자로 시작한다. `int`, `long`, `double`, `boolean` 모두 소문자로 시작한다.
  - 기본형은 자바가 기본으로 제공하는 데이터 타입이다. 이러한 기본형은 개발자가 새로 정의할 수 없다. 개발자는 참조형인 클래스만 직접 정의할 수 있다.
- 클래스는 대문자로 시작한다. `Student`
  - 클래스는 모두 참조형이다.

참고 - String

자바에서 `String`은 특별하다. `String`은 사실은 클래스다. 따라서 참조형이다. 그런데 기본형처럼 문자 값을 바로 대입할 수 있다. 문자는 매우 자주 다루기 때문에 자바에서 특별하게 편의 기능을 제공한다. `String`에 대한 자세한 내

용은 뒤에서 설명한다.

기본형 vs 참조형2 - 변수 대입

대원칙: 자바는 항상 변수의 값을 복사해서 대입한다.

자바에서 변수에 값을 대입하는 것은 변수에 들어 있는 값을 복사해서 대입하는 것이다.
기본형, 참조형 모두 항상 변수에 있는 값을 복사해서 대입한다. 기본형이면 변수에 들어 있는 실제 사용하는 값을 복사해서 대입하고, 참조형이면 변수에 들어 있는 참조값을 복사해서 대입한다.

이 대원칙을 이해하면 복잡한 상황에도 코드를 단순하게 이해할 수 있다.

기본형 대입

```
int a = 10;
int b = a;
```

참조형 대입

```
Student s1 = new Student();
Student s2 = s1;
```

기본형은 변수에 값을 대입하더라도 실제 사용하는 값이 변수에 바로 들어있기 때문에 해당 값만 복사해서 대입한다고 생각하면 쉽게 이해할 수 있다. 그런데 **참조형의 경우 실제 사용하는 객체가 아니라 객체의 위치를 가리키는 참조값만 복사**된다. 쉽게 이야기해서 실제 건물이 복사가 되는 것이 아니라 건물의 위치인 주소만 복사되는 것이다. 따라서 같은 건물을 찾아갈 수 있는 방법이 하나 늘어날 뿐이다.

구체적인 예시를 통해서 변수 대입시 기본형과 참조형의 차이를 알아보자.

기본형과 변수 대입

다음 코드를 보고 어떤 결과가 나올지 먼저 생각해보자. 너무 쉽고 당연한 내용이지만, 이후에 나올 참조형과 비교를 위해서 다시 한번 정리해보자.

VarChange1

```
package ref;
```

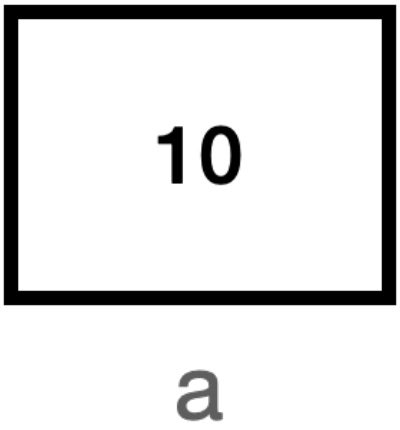
```java
public class VarChange1 {

    public static void main(String[] args) {
        int a = 10;
        int b = a;
        System.out.println("a = " + a);
        System.out.println("b = " + b);

        //a 변경
        a = 20;
        System.out.println("변경 a = 20");
        System.out.println("a = " + a);
        System.out.println("b = " + b);

        //b 변경
        b = 30;
        System.out.println("변경 b = 30");
        System.out.println("a = " + a);
        System.out.println("b = " + b);
    }
}
```

실행 결과

```
a = 10
b = 10
변경 a = 20
a = 20
b = 10
변경 b = 30
a = 20
b = 30
```

그림을 통해 자세히 알아보자.

int a = 10



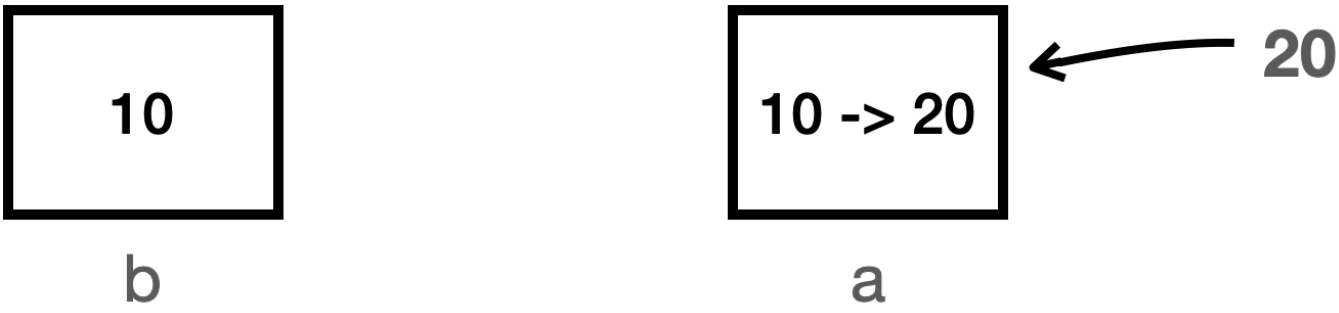
int b = a

실행 결과

```
a = 10
b = 10
```

변수의 대입은 변수에 들어있는 값을 복사해서 대입한다. 여기서는 변수 `a`에 들어있는 값 `10`을 복사해서 변수 `b`에 대입한다. 변수 `a` 자체를 `b`에 대입하는 것이 아니다!

a = 20



실행 결과

```
a = 20
b = 10
```

변수 `a`에 값 `20`을 대입했다. 따라서 변수 `a`의 값이 `10`에서 `20`으로 변경되었다. 당연한 이야기지만 변수 `b`에는 아무런 영향을 주지 않는다.

b = 30

실행 결과

```
a = 20
b = 30
```

변수 `b`에 값 `30`을 대입했다. 변수 `b`의 값이 `10`에서 `30`으로 변경되었다. 당연한 이야기지만 변수 `a`에는 아무런 영향을 주지 않는다.

최종 결과

여기서 핵심은 `int b = a`라고 했을 때 변수에 들어있는 값을 복사해서 전달한다는 점이다. 따라서 `a=20`, `b=30`이라고 했을 때 각각 본인의 값만 변경되는 것을 확인할 수 있다.

너무 당연한 이야기를 왜 이렇게 장황하게 풀어서 하지? 라고 생각한다면 이제 진짜 문제를 만나보자.

참조형과 변수 대입

참조형 예시를 위해 `Data` 클래스를 하나 만들자. 이 클래스는 단순히 `int value`라는 멤버 변수를 하나 가진다.

Data

```
package ref;
```

```
public class Data {
    int value;
}
```

다음 코드를 보고 어떤 결과가 나올지 먼저 생각해보자. 꼭 먼저 생각해보고 이후에 실행해서 정답을 맞춰보자.

VarChange2

```
package ref;

public class VarChange2 {

    public static void main(String[] args) {
        Data dataA = new Data();
        dataA.value = 10;
        Data dataB = dataA;

        System.out.println("dataA 참조값=" + dataA);
        System.out.println("dataB 참조값=" + dataB);
        System.out.println("dataA.value = " + dataA.value);
        System.out.println("dataB.value = " + dataB.value);

        //dataA 변경
        dataA.value = 20;
        System.out.println("변경 dataA.value = 20");
        System.out.println("dataA.value = " + dataA.value);
        System.out.println("dataB.value = " + dataB.value);

        //dataB 변경
        dataB.value = 30;
        System.out.println("변경 dataB.value = 30");
        System.out.println("dataA.value = " + dataA.value);
        System.out.println("dataB.value = " + dataB.value);
    }
}
```

실행 결과

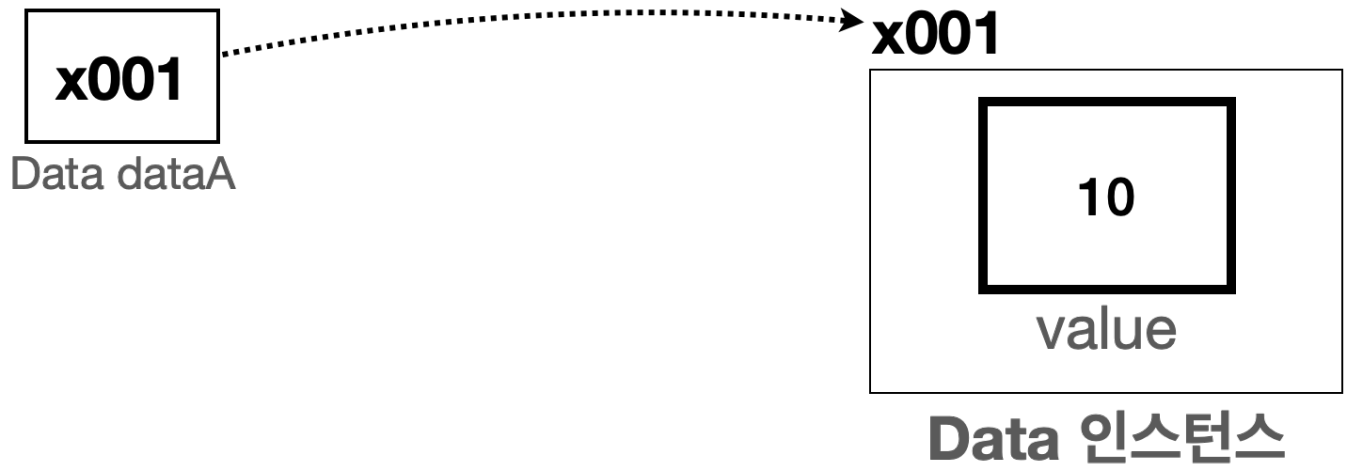
```
dataA 참조값=ref.Data@x001
dataB 참조값=ref.Data@x001
dataA.value = 10
dataB.value = 10
변경 dataA.value = 20
dataA.value = 20
```

```
dataB.value = 20
변경 dataB.value = 30
dataA.value = 30
dataB.value = 30
```

그림을 통해 자세히 알아보자.

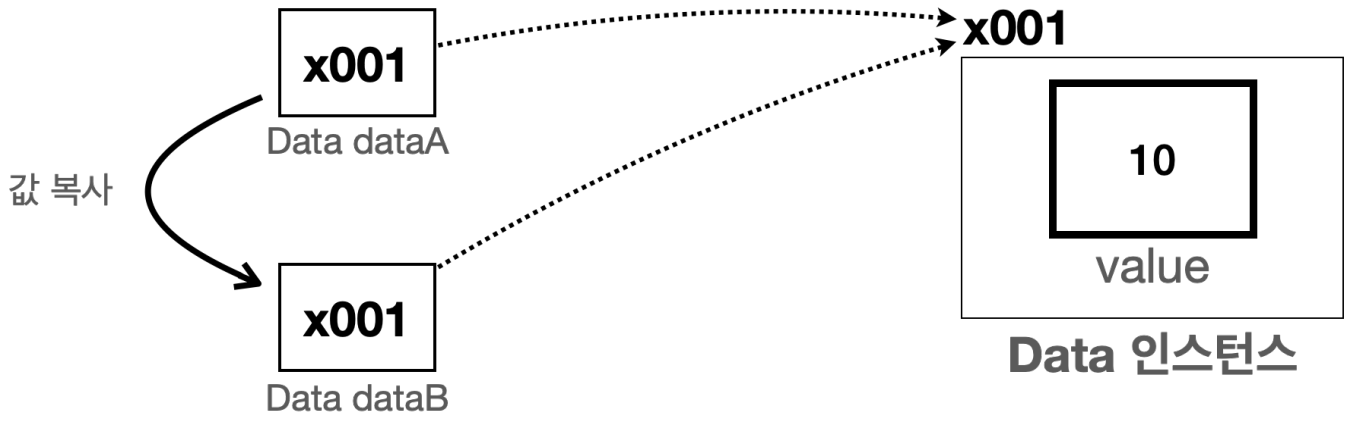
Data dataA = new Data()

dataA.value = 10



`dataA` 변수는 `Data` 클래스를 통해서 만들었기 때문에 참조형이다. 이 변수는 `Data`형 객체의 참조값을 저장한다. `Data` 객체를 생성하고, 참조값을 `dataA`에 저장한다. 그리고 객체의 `value` 변수에 값 `10`을 저장했다.

Data dataB = dataA



실행 코드

```
Data dataA = new Data();
dataA.value = 10;
Data dataB = dataA;

System.out.println("dataA 참조값=" + dataA);
```

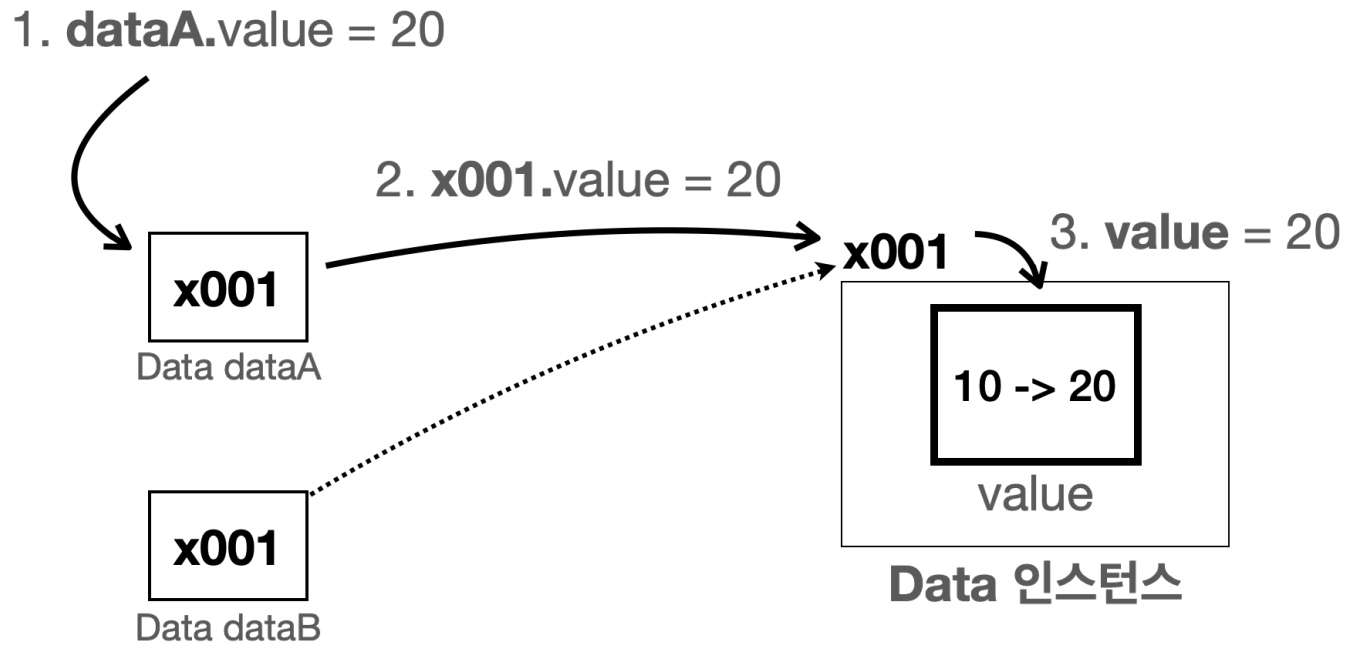
```
System.out.println("dataB 참조값=" + dataB);
System.out.println("dataA.value = " + dataA.value);
System.out.println("dataB.value = " + dataB.value);
```

출력 결과

```
dataA = ref.Data@x001
dataB = ref.Data@x001
dataA.value = 10
dataB.value = 10
```

변수의 대입은 변수에 들어있는 값을 복사해서 대입한다. 변수 `dataA`에는 참조값 `x001`이 들어있다. 여기서는 변수 `dataA`에 들어있는 참조값 `x001`을 복사해서 변수 `dataB`에 대입한다. 참고로 변수 `dataA`가 가리키는 인스턴스를 복사하는 것이 아니다! 변수에 들어있는 참조값만 복사해서 전달한다.
이제 `dataA`와 `dataB`에 들어있는 참조값은 같다. 따라서 둘다 같은 `x001` `Data` 인스턴스를 가리킨다.

dataA.value = 20



실행 코드

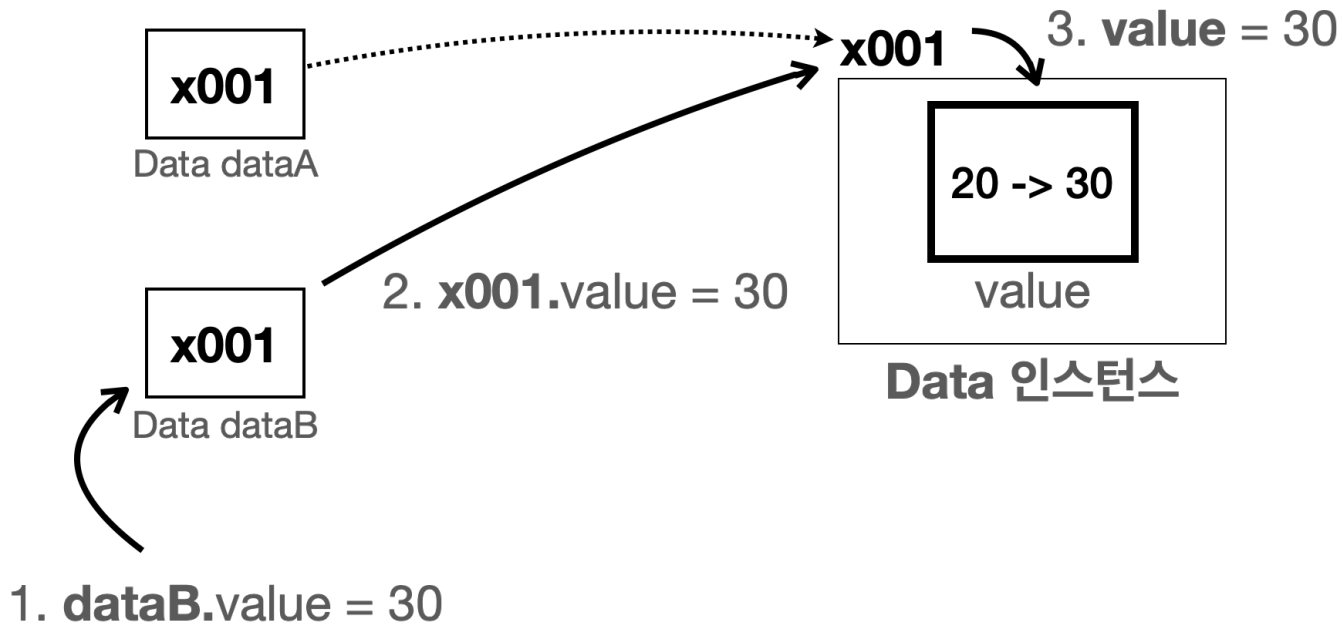
```
dataA.value = 20;
System.out.println("dataA = " + dataA.value);
System.out.println("dataB = " + dataB.value);
```

출력 결과

```
dataA.value = 20
dataB.value = 20
```

`dataA.value = 20` 코드를 실행하면 `dataA`가 가리키는 `x001` 인스턴스의 `value`값을 `10`에서 `20`으로 변경한다. 그런데 `dataA`와 `dataB`는 같은 `x001` 인스턴스를 참조하기 때문에 `dataA.value`와 `dataB.value`는 둘 다 같은 값인 `20`을 출력한다.

dataB.value = 30



실행 코드

```
dataB.value = 30;
System.out.println("dataA.value = " + dataA.value);
System.out.println("dataB.value = " + dataB.value);
```

출력 결과

```
dataA.value = 30
dataB.value = 30
```

`dataB.value = 30` 코드를 실행하면 `dataB`가 가리키는 `x001` 인스턴스의 `value`값을 `20`에서 `30`으로 변경한다. 그런데 `dataA`와 `dataB`는 같은 `x001` 인스턴스를 참조하기 때문에 `dataA.value`와 `dataB.value`는 같은 값인 `30`을 출력한다.

여기서 핵심은 `Data dataB = dataA`라고 했을 때 변수에 들어있는 값을 복사해서 사용한다는 점이다. 그런데 그 값이 참조값이다. 따라서 `dataA`와 `dataB`는 같은 참조값을 가지게 되고, 두 변수는 같은 객체 인스턴스를 참조하게 된다.

기본형 vs 참조형3 - 메서드 호출

이번에는 기본형과 참조형이 메서드 호출에 따라서 어떻게 달라지는지 알아보자.

대원칙: 자바는 항상 변수의 값을 복사해서 대입한다.

자바에서 변수에 값을 대입하는 것은 변수에 들어 있는 값을 복사해서 대입하는 것이다.
기본형, 참조형 모두 항상 변수에 있는 값을 복사해서 대입한다. 기본형이면 변수에 들어 있는 실제 사용하는 값을 복사해서 대입하고, 참조형이면 변수에 들어 있는 참조값을 복사해서 대입한다.

메서드 호출도 마찬가지이다. 메서드를 호출할 때 사용하는 매개변수(파라미터)도 결국 변수일 뿐이다. 따라서 메서드를 호출할 때 매개변수에 값을 전달하는 것도 앞서 설명한 내용과 같이 값을 복사해서 전달한다.

다음 메서드 호출 코드를 보고 어떤 결과가 나올지 먼저 생각해보자. 너무 쉽고 당연한 내용이지만, 이후에 나올 참조형과 비교를 위해서 한번 정리해보자.

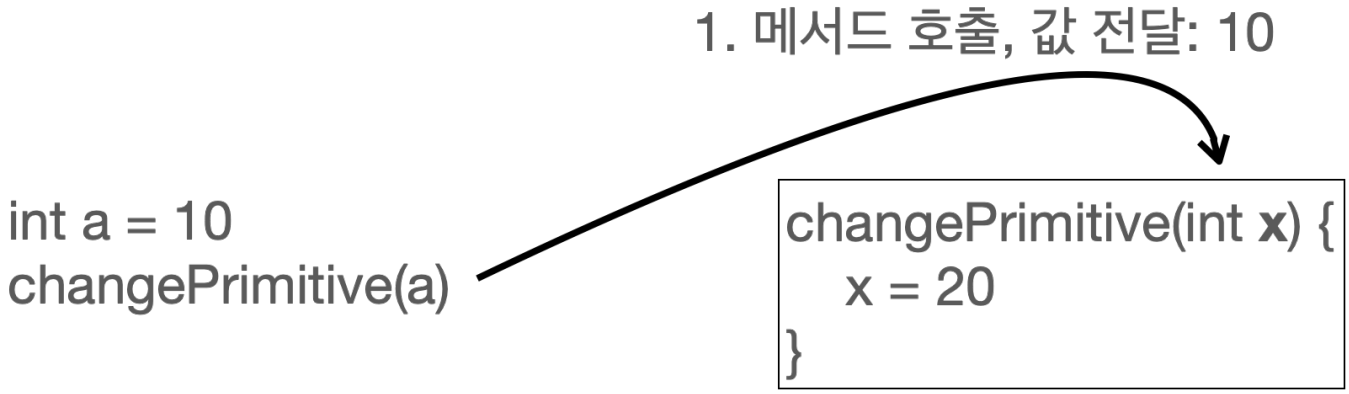
기본형과 메서드 호출

```
package ref;

public class MethodChange1 {

    public static void main(String[] args) {
        int a = 10;
        System.out.println("메서드 호출 전: a = " + a);
        changePrimitive(a);
        System.out.println("메서드 호출 후: a = " + a);
    }

    static void changePrimitive(int x) {
        x = 20;
    }
}
```

실행 결과

```
메서드 호출 전: a = 10
메서드 호출 후: a = 10
```

1. 메서드 호출

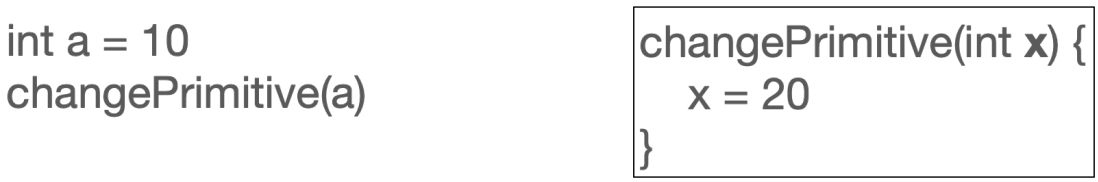


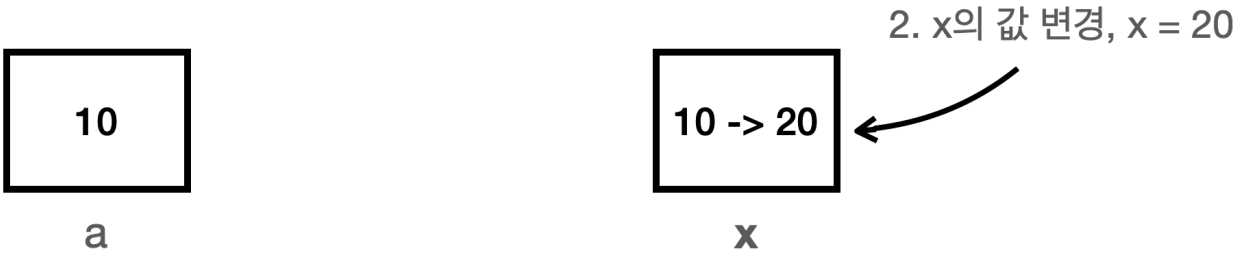
메서드를 호출할 때 매개변수 `x`에 변수 `a`의 값을 전달한다. 이 코드는 다음과 같이 해석할 수 있다.

```
int x = a
```

자바에서 변수에 값을 대입하는 것은 항상 값을 복사해서 대입한다. 따라서 변수 `a`, `x` 각각 숫자 `10`을 가지고 있다.

2. 메서드 안에서 값을 변경

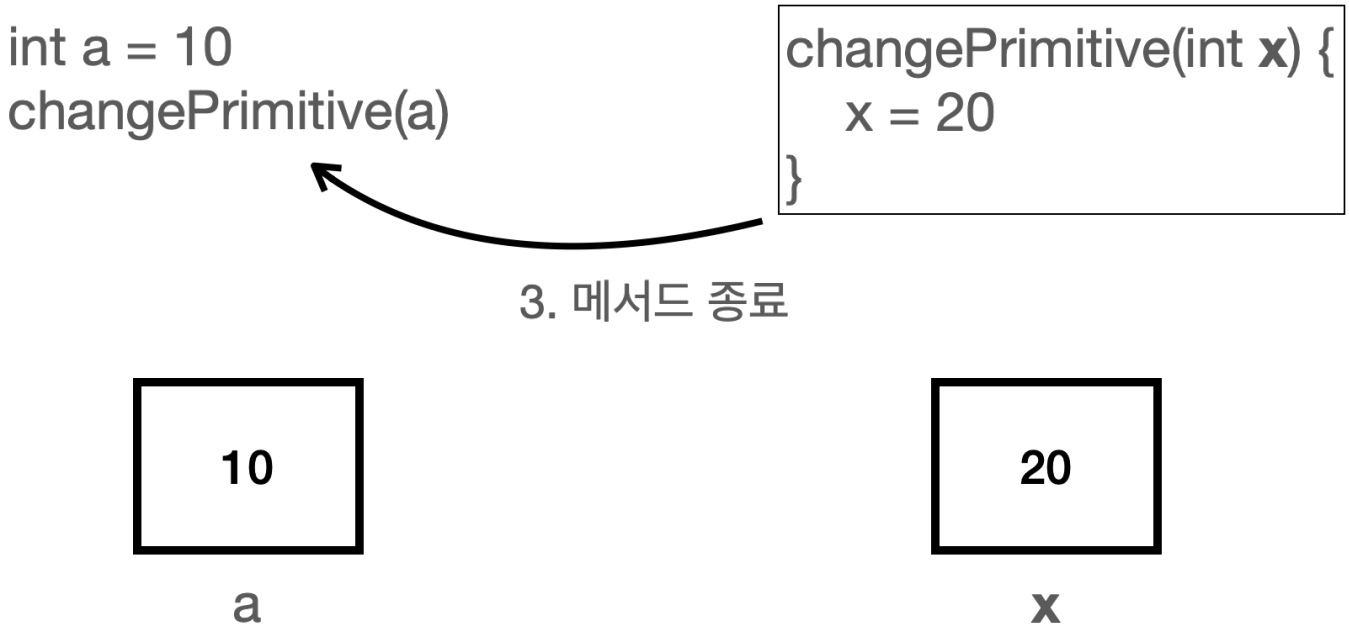




메서드 안에서 `x = 20`으로 새로운 값을 대입한다.

결과적으로 `x`의 값만 `20`으로 변경되고, `a`의 값은 `10`으로 유지된다.

3. 메서드 종료



메서드 종료후 값을 확인해보면 `a`는 10이 출력되는 것을 확인할 수 있다. 참고로 메서드가 종료되면 매개변수 `x`는 제거된다.

참조형과 메서드 호출

이번에는 참조형 변수의 메서드 호출에 대해 알아보자.

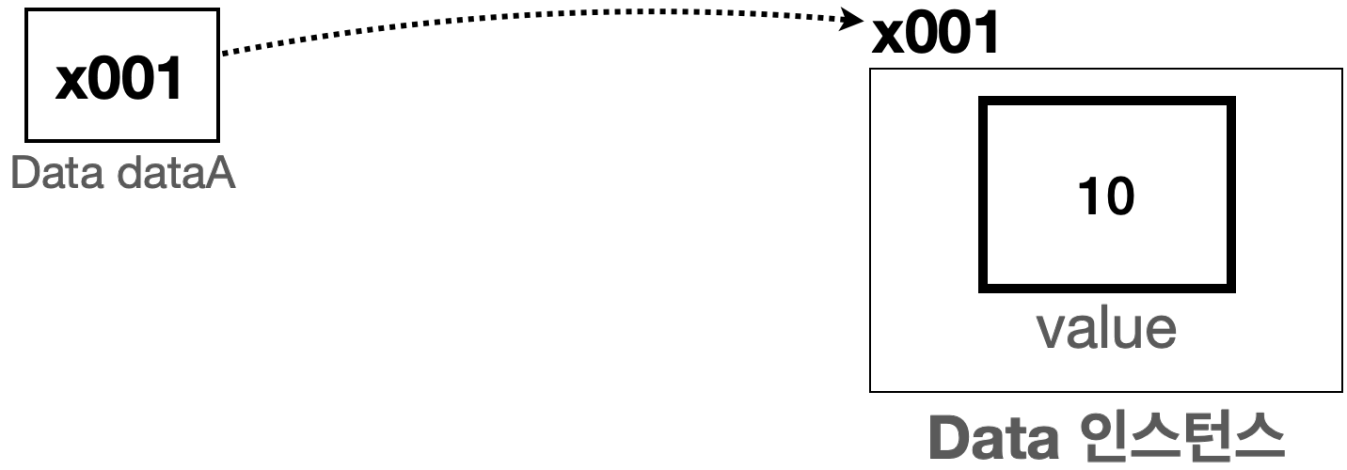
다음 메서드 호출 코드를 보고 어떤 결과가 나올지 먼저 생각해보자.

```
package ref;

public class MethodChange2 {

    public static void main(String[] args) {
        Data dataA = new Data();
        dataA.value = 10;
        System.out.println("메서드 호출 전: dataA.value = " + dataA.value);
        changeReference(dataA);
        System.out.println("메서드 호출 후: dataA.value = " + dataA.value);
    }

    static void changeReference(Data dataX) {
        dataX.value = 20;
    }
}
```

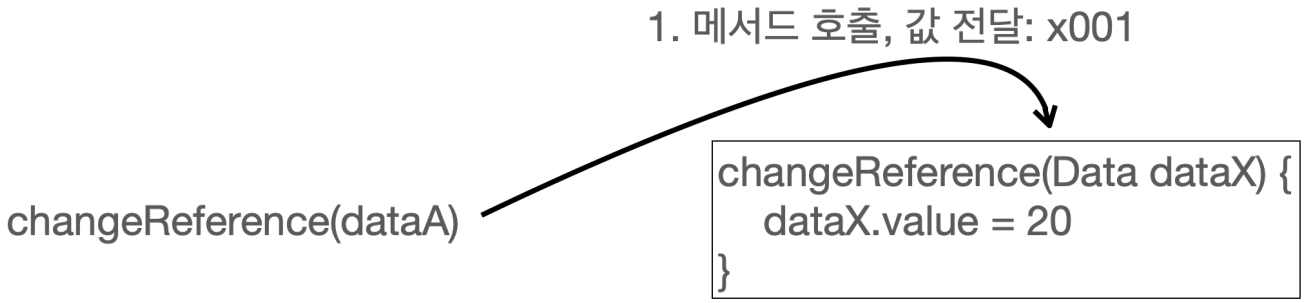
실행 결과

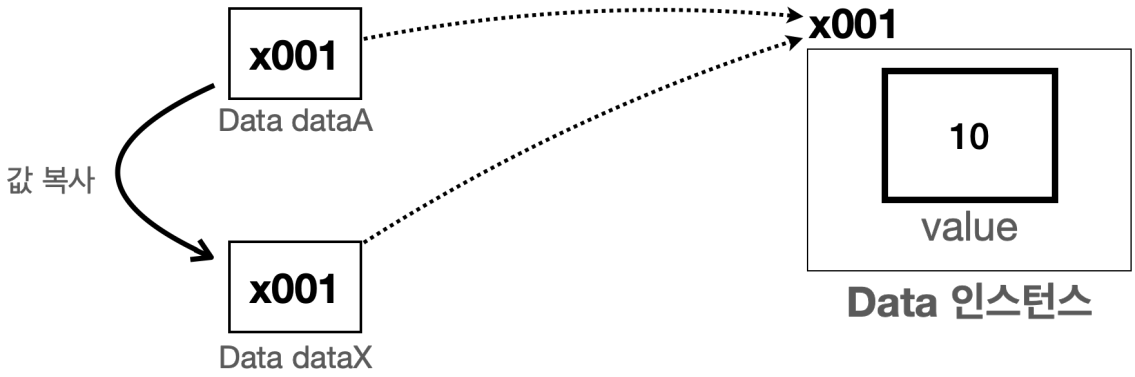
```
메서드 호출 전: dataA.value = 10
메서드 호출 후: dataA.value = 20
```

`Data` 인스턴스를 생성하고, 참조값을 `dataA` 변수에 담고 `value`에 숫자 10을 할당한 상태는 다음과 같다.



1. 메서드 호출





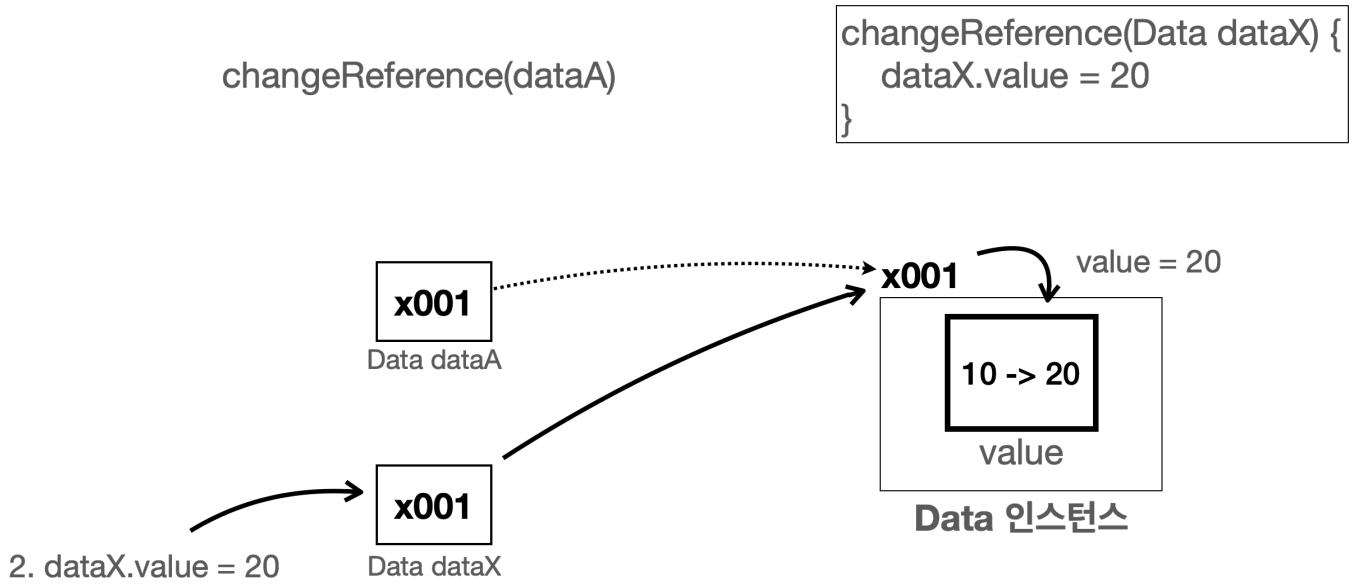
메서드를 호출할 때 매개변수 `dataX`에 변수 `dataA`의 값을 전달한다. 이 코드는 다음과 같이 해석할 수 있다.

```
Data dataX = dataA
```

자바에서 변수에 값을 대입하는 것은 항상 값을 복사해서 대입한다. 변수 `dataA`는 참조값 `x001`을 가지고 있으므로

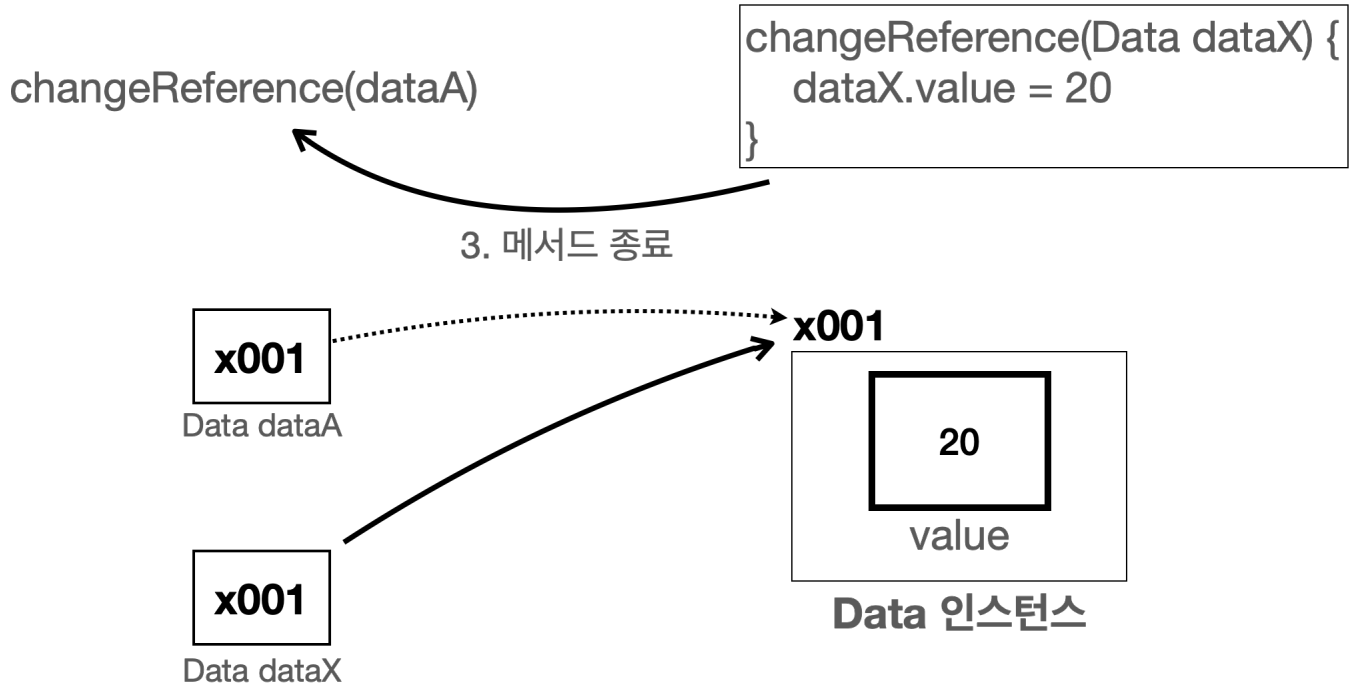
참조값을 복사해서 전달했다. 따라서 변수 `dataA`, `dataX` 둘다 같은 참조값인 `x001`을 가지게 된다.
이제 `dataX`를 통해서도 `x001`에 있는 `Data` 인스턴스에 접근할 수 있다.

2. 메서드 안에서 값을 변경



메서드 안에서 `dataX.value = 20`으로 새로운 값을 대입한다.
참조값을 통해 `x001` 인스턴스에 접근하고 그 안에 있는 `value`의 값을 `20`으로 변경했다.
`dataA`, `dataX` 모두 같은 `x001` 인스턴스를 참조하기 때문에 `dataA.value`와 `dataX.value`는 둘다 `20`이라는 값을 가진다.

3. 메서드 종료



메서드 종료후 `dataA.value`의 값을 확인해보면 다음과 같이 `20`으로 변경된 것을 확인할 수 있다.

```
메서드 호출 전: dataA.value = 10
메서드 호출 후: dataA.value = 20
```

기본형과 참조형의 메서드 호출

자바에서 메서드의 매개변수(파라미터)는 항상 값에 의해 전달된다. 그러나 이 값이 실제 값이냐, 참조(메모리 주소)값이냐에 따라 동작이 달라진다.

- **기본형**: 메서드로 기본형 데이터를 전달하면, 해당 값이 복사되어 전달된다. 이 경우, 메서드 내부에서 매개변수(파라미터)의 값을 변경해도, 호출자의 변수 값에는 영향이 없다.
- **참조형**: 메서드로 참조형 데이터를 전달하면, 참조값이 복사되어 전달된다. 이 경우, 메서드 내부에서 매개변수(파라미터)로 전달된 객체의 멤버 변수를 변경하면, 호출자의 객체도 변경된다.

참조형과 메서드 호출 - 활용

이전에 보았던 `class1.ClassStart3` 코드에는 중복되는 부분이 2가지 있다.

- `name`, `age`, `grade`에 값을 할당
- 학생 정보를 출력

ClassStart3

```
package class1;

public class ClassStart3 {
    public static void main(String[] args) {
        Student student1;
        student1 = new Student();
        student1.name = "학생1";
        student1.age = 15;
        student1.grade = 90;

        Student student2 = new Student();
        student2.name = "학생2";
        student2.age = 16;
        student2.grade = 80;

        System.out.println("이름:" + student1.name + " 나이:" + student1.age + " 성
적:" + student1.grade);
        System.out.println("이름:" + student2.name + " 나이:" + student2.age + " 성
```

```
적:" + student2.grade);
    }
}
```

이런 중복은 메서드를 통해 손쉽게 제거할 수 있다.

메서드에 객체 전달

다음과 같이 코드를 작성해보자.

Student

```
package ref;

public class Student {
    String name;
    int age;
    int grade;
}
```

- `ref` 패키지에도 `Student` 클래스를 만든다.

Method1

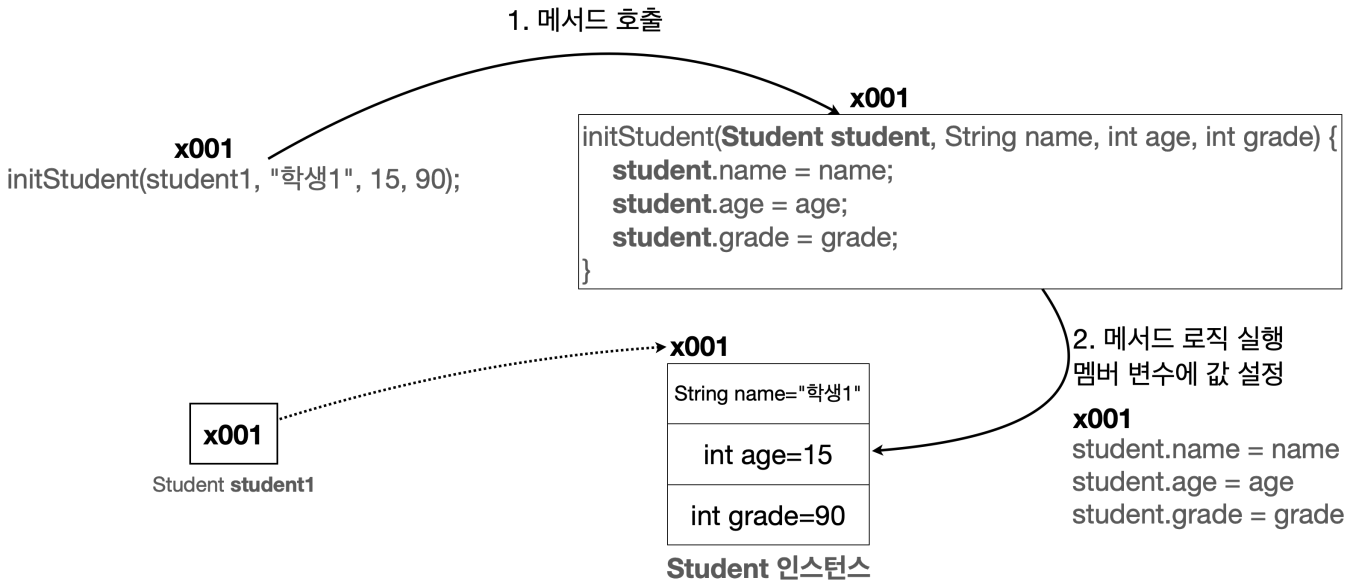
```
package ref;

public class Method1 {
    public static void main(String[] args) {
        Student student1 = new Student();
        initStudent(student1, "학생1", 15, 90);

        Student student2 = new Student();
        initStudent(student2, "학생2", 16, 80);

        printStudent(student1);
        printStudent(student2);
    }

    static void initStudent(Student student, String name, int age, int grade) {
        student.name = name;
        student.age = age;
        student.grade = grade;
    }
```

```
    static void printStudent(Student student1) {
        System.out.println("이름:" + student1.name + " 나이:" + student1.age + " 성적:" + student1.grade);
    }
}
```

참조형은 메서드를 호출할 때 참조값을 전달한다. 따라서 메서드 내부에서 전달된 참조값을 통해 객체의 값을 변경하거나, 값을 읽어서 사용할 수 있다.

- `initStudent(Student student, ...)` : 전달한 학생 객체의 필드에 값을 설정한다.
- `printStudent(Student student, ...)` : 전달한 학생 객체의 필드 값을 읽어서 출력한다.

initStudent() 메서드 호출 분석



- `student1`이 참조하는 `Student` 인스턴스에 값을 편리하게 할당하고 싶어서 `initStudent()` 메서드를 만들었다.
- 이 메서드를 호출하면서 `student1`을 전달한다. 그러면 `student1`의 참조값이 매개변수 `student`에 전달된다. 이 참조값을 통해 `initStudent()` 메서드 안에서 `student1`이 참조하는 것과 동일한 `x001` `Student` 인스턴스에 접근하고 값을 변경할 수 있다.

주의!

```
package ref;

import class1.Student;

public class Method1 {
    ...
}
```

- `import class1.Student;`이 선언되어 있으면 안된다.

- 이렇게 되면 `class1` 패키지에서 선언한 `Student`를 사용하게 된다. 이 경우 접근 제어자 때문에 컴파일 오류가 발생한다.
- 만약 선언되어 있다면 삭제하자. 삭제하면 같은 패키지에 있는 `ref.Student`를 사용한다.

메서드에서 객체 반환

조금 더 나아가보자. 다음 코드에도 중복이 있다.

```
Student student1 = new Student();
initStudent(student1, "학생1", 15, 90);

Student student2 = new Student();
initStudent(student2, "학생2", 16, 80);
```

바로 객체를 생성하고, 초기값을 설정하는 부분이다. 이렇게 2번 반복되는 부분을 하나로 합쳐보자.

다음과 같이 기존 코드를 변경해보자.

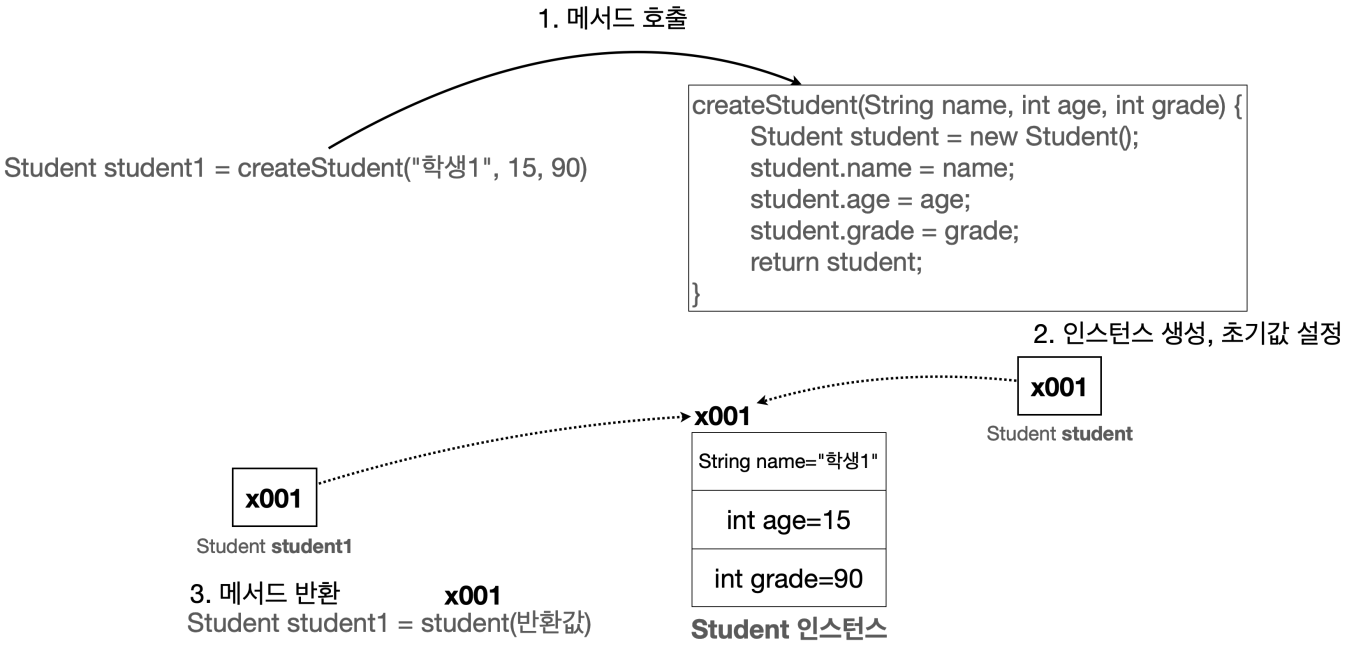
Method2

```
package ref;

public class Method2 {
    public static void main(String[] args) {
        Student student1 = createStudent("학생1", 15, 90);
        Student student2 = createStudent("학생2", 16, 80);

        printStudent(student1);
        printStudent(student2);
    }

    static Student createStudent(String name, int age, int grade) {
        Student student = new Student();
        student.name = name;
        student.age = age;
        student.grade = grade;
        return student;
    }

    static void printStudent(Student student1) {
        System.out.println("이름:" + student1.name + " 나이:" + student1.age + " 성
적:" + student1.grade);
    }
}
```

`createStudent()` 라는 메서드를 만들고 객체를 생성하는 부분도 이 메서드안에 함께 포함했다. 이제 이 메서드 하나로 객체의 생성과 초기값 설정을 모두 처리한다.

그런데 메서드 안에서 객체를 생성했기 때문에 만들어진 객체를 메서드 밖에서 사용할 수 있게 돌려주어야 한다. 그래야 메서드 밖에서 이 객체를 사용할 수 있다. 메서드는 호출 결과를 반환(`return`)을 할 수 있다. 메서드의 반환 기능을 사용해서 만들어진 객체의 참조값을 메서드 밖으로 반환하면 된다.

createStudent() 메서드 호출 분석



메서드 내부에서 인스턴스를 생성한 후에 참조값을 메서드 외부로 반환했다. 이 참조값만 있으면 해당 인스턴스에 접근할 수 있다. 여기서는 `student1` 에 참조값을 보관하고 사용한다.

진행 과정

```
Student student1 = createStudent("학생1", 15, 90) //메서드 호출후 결과 반환
Student student1 = student(x001) //참조형인 student를 반환
Student student1 = x001 //student의 참조값 대입
student1 = x001
```

`createStudent()` 는 생성한 `Student` 인스턴스의 참조값을 반환한다. 이렇게 반환된 참조값을 `student1` 변수에 저장했다. 앞으로는 `student1` 을 통해 `Student` 인스턴스를 사용할 수 있다.

변수와 초기화

변수의 종류

- 멤버 변수(필드): 클래스에 선언
- 지역 변수: 메서드에 선언, 매개변수도 지역 변수의 한 종류이다.

멤버 변수, 필드 예시

```
public class Student {
    String name;
    int age;
    int grade;
}
```

`name`, `age`, `grade`는 멤버 변수이다.

지역 변수 예시

```
public class ClassStart3 {
    public static void main(String[] args) {
        Student student1;
        student1 = new Student();
        Student student2 = new Student();
    }
}
```

`student1`, `student2`는 지역 변수이다.

```
public class MethodChange1 {

    public static void main(String[] args) {
        int a = 10;
        System.out.println("메서드 호출 전: a = " + a);
        changePrimitive(a);
        System.out.println("메서드 호출 후: a = " + a);
    }

    public static void changePrimitive(int x) {
        x = 20;
    }
}
```

`a`, `x`(매개변수)는 지역 변수이다.

지역 변수는 이름 그대로 특정 지역에서만 사용되는 변수라는 뜻이다. 예를 들어서 변수 `x`는 `changePrimitive()` 메서드의 블록에서만 사용된다. `changePrimitive()` 메서드가 끝나면 제거된다. `a` 변수도 마찬가지이다.

`main()` 메서드가 끝나면 제거된다.

변수의 값 초기화

- 멤버 변수: 자동 초기화
  - 인스턴스의 멤버 변수는 인스턴스를 생성할 때 자동으로 초기화된다.
  - 숫자(`int`)= `0`, `boolean` = `false`, 참조형 = `null` (`null` 값은 참조할 대상이 없다는 뜻으로 사용된다.)
  - 개발자가 초기값을 직접 지정할 수 있다.
- 지역 변수: 수동 초기화
  - 지역 변수는 항상 직접 초기화해야 한다.

멤버 변수의 초기화를 살펴보자

InitData

```
package ref;

public class InitData {
    int value1; //초기화 하지 않음
    int value2 = 10; //10으로 초기화
}
```

`value1`은 초기값을 지정하지 않았고, `value2`는 초기값을 10으로 지정했다.

InitMain

```
package ref;

public class InitMain {
    public static void main(String[] args) {
        InitData data = new InitData();
        System.out.println("value1 = " + data.value1);
        System.out.println("value2 = " + data.value2);
    }
}
```

실행 결과

```
value1 = 0
value2 = 10
```

`value1`은 초기값을 지정하지 않았지만 멤버 변수는 자동으로 초기화 된다. 숫자는 `0`으로 초기화된다.

`value2`는 `10`으로 초기값을 지정해두었기 때문에 객체를 생성할 때 `10`으로 초기화된다.

null

택배를 보낼 때 제품은 준비가 되었지만, 보낼 주소지가 아직 결정되지 않아서, 주소지가 결정될 때 까지는 주소지를 비워두어야 할 수 있다.
참조형 변수에는 항상 객체가 있는 위치를 가리키는 참조값이 들어간다. 그런데 아직 가리키는 대상이 없거나, 가리키는 대상을 나중에 입력하고 싶다면 어떻게 해야할까?
참조형 변수에서 아직 가리키는 대상이 없다면 `null`이라는 특별한 값을 넣어둘 수 있다. `null`은 값이 존재하지 않는, 없다는 뜻이다.

코드를 통해서 `null` 값에 대해 알아보자.

null 값 할당

```
package ref;

public class Data {
    int value;
}
```

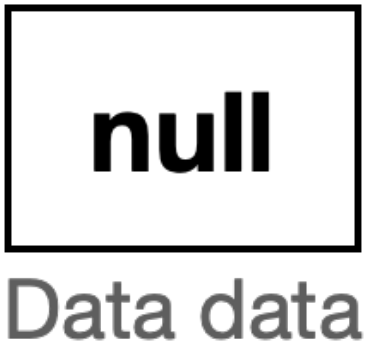
앞서 만들었던 `Data` 클래스이다. 기존 코드를 그대로 유지하면 된다.

```
package ref;

public class NullMain1 {
    public static void main(String[] args) {
        Data data = null;
        System.out.println("1. data = " + data);
        data = new Data();
        System.out.println("2. data = " + data);
        data = null;
        System.out.println("3. data = " + data);
    }
}
```

실행 결과

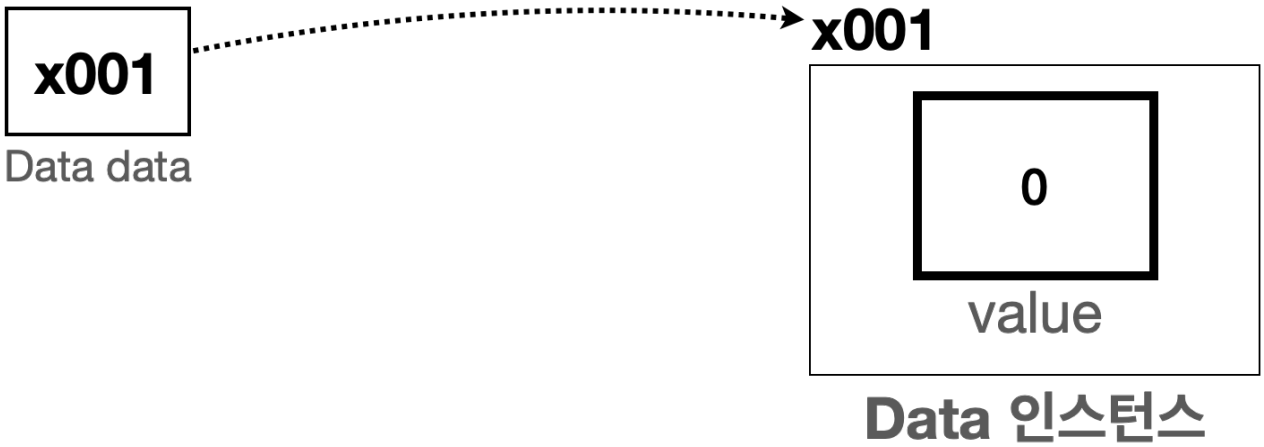
```
1. data = null
2. data = ref.Data@x001
3. data = null
```

Data data = null



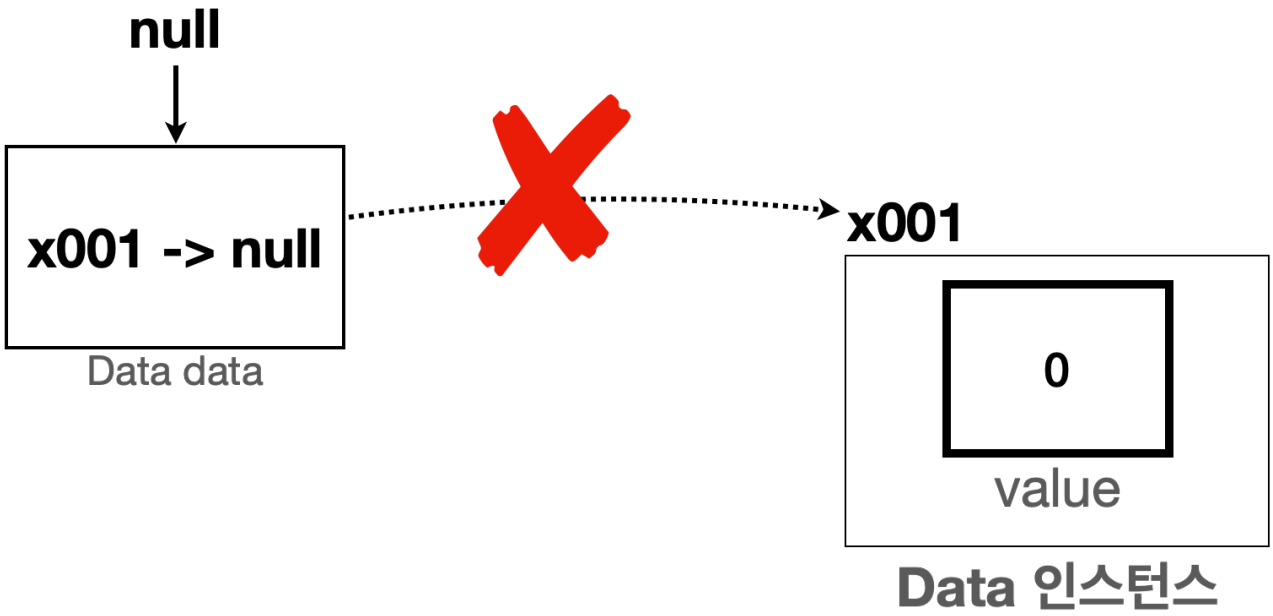
`Data` 타입을 받을 수 있는 참조형 변수 `data`를 만들었다. 그리고 여기에 `null` 값을 할당했다. 따라서 `data` 변수에는 아직 가리키는 객체가 없다는 뜻이다.

data = new Data()



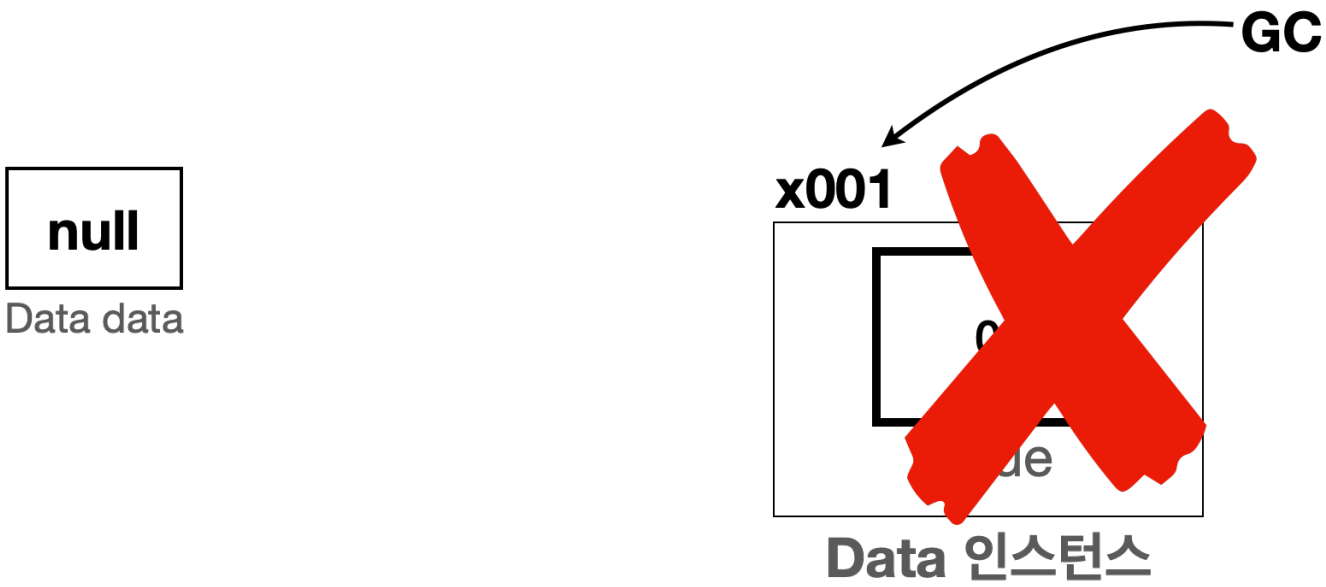
이후에 새로운 `Data` 객체를 생성해서 그 참조값을 `data` 변수에 할당했다. 이제 `data` 변수가 참조할 객체가 존재한다.

data = null



마지막에는 `data`에 다시 `null`값을 할당했다. 이렇게 하면 `data` 변수는 앞서 만든 `Data` 인스턴스를 더는 참조하지 않는다.

GC - 아무도 참조하지 않는 인스턴스의 최후



`data`에 `null`을 할당했다. 따라서 앞서 생성한 `x001` `Data` 인스턴스를 더는 아무도 참조하지 않는다. 이렇게 아무도 참조하지 않게 되면 `x001`이라는 참조값을 다시 구할 방법이 없다. 따라서 해당 인스턴스에 다시 접근할 방법이 없다.

이렇게 아무도 참조하지 않는 인스턴스는 사용되지 않고 메모리 용량만 차지할 뿐이다.

C와 같은 과거 프로그래밍 언어는 개발자가 직접 명령어를 사용해서 인스턴스를 메모리에서 제거해야 했다. 만약 실수로 인스턴스 삭제를 누락하면 메모리에 사용하지 않는 객체가 가득해져서 메모리 부족 오류가 발생하게 된다.

자바는 이런 과정을 자동으로 처리해준다. 아무도 참조하지 않는 인스턴스가 있으면 JVM의 GC(가비지 컬렉션)가 더 이상 사용하지 않는 인스턴스라 판단하고 해당 인스턴스를 자동으로 메모리에서 제거해준다.

객체는 해당 객체를 참조하는 곳이 있으면, JVM이 종료할 때 까지 계속 생존한다. 그런데 중간에 해당 객체를 참조하는 곳이 모두 사라지면 그때 JVM은 필요 없는 객체로 판단다고 GC(가비지 컬렉션)를 사용해서 제거한다.

NullPointerException

택배를 보낼 때 주소지 없이 택배를 발송하면 어떤 문제가 발생할까? 만약 참조값 없이 객체를 찾아가면 어떤 문제가 발생할까?
이 경우 `NullPointerException`이라는 예외가 발생하는데, 개발자를 가장 많이 괴롭히는 예외이다.
`NullPointerException`은 이름 그대로 `null`을 가리키다(Pointer)인데, 이때 발생하는 예외(Exception)다.
`null`은 없다는 뜻이므로 결국 주소가 없는 곳을 찾아갈 때 발생하는 예외이다.

객체를 참조할 때는 `.`(dot)을 사용한다. 이렇게 하면 참조값을 사용해서 해당 객체를 찾아갈 수 있다. 그런데 참조값이 `null`이라면 값이 없다는 뜻이므로, 찾아갈 수 있는 객체(인스턴스)가 없다. `NullPointerException`은 이처럼 `null`에 `.`(dot)을 찍었을 때 발생한다.

예제를 통해서 확인해보자.

```
package ref;

public class NullMain2 {
    public static void main(String[] args) {
        Data data = null;
        data.value = 10;// NullPointerException 예외 발생
        System.out.println("data = " + data.value);
    }
}
```

`data` 참조형 변수에는 `null`값이 들어가 있다. 그런데 `data.value = 10`이라고 하면 어떻게 될까?

```
data.value = 10
null.value = 10 //data에는 null 값이 들어있다.
```

결과적으로 `null` 값은 참조할 주소가 존재하지 않는다는 뜻이다. 따라서 참조할 객체 인스턴스가 존재하지 않으므로 다음과 같이 `java.lang.NullPointerException`이 발생하고, 프로그램이 종료된다. 참고로 예외가 발생했기 때문에 그 다음 로직은 수행되지 않는다.

실행 결과

```
Exception in thread "main" java.lang.NullPointerException: Cannot assign field
"value" because "data" is null
    at ref.NullMain2.main(NullMain2.java:6)
```

멤버 변수와 null

앞선 예제와 같이 지역 변수의 경우에는 `null` 문제를 파악하는 것이 어렵지 않다. 다음과 같이 멤버 변수가 `null`인 경우에는 주의가 필요하다.

```
package ref;

public class Data {
    int value;
}
```

기존의 `Data` 클래스를 사용한다.

```
package ref;

public class BigData {
    Data data;
    int count;
}
```

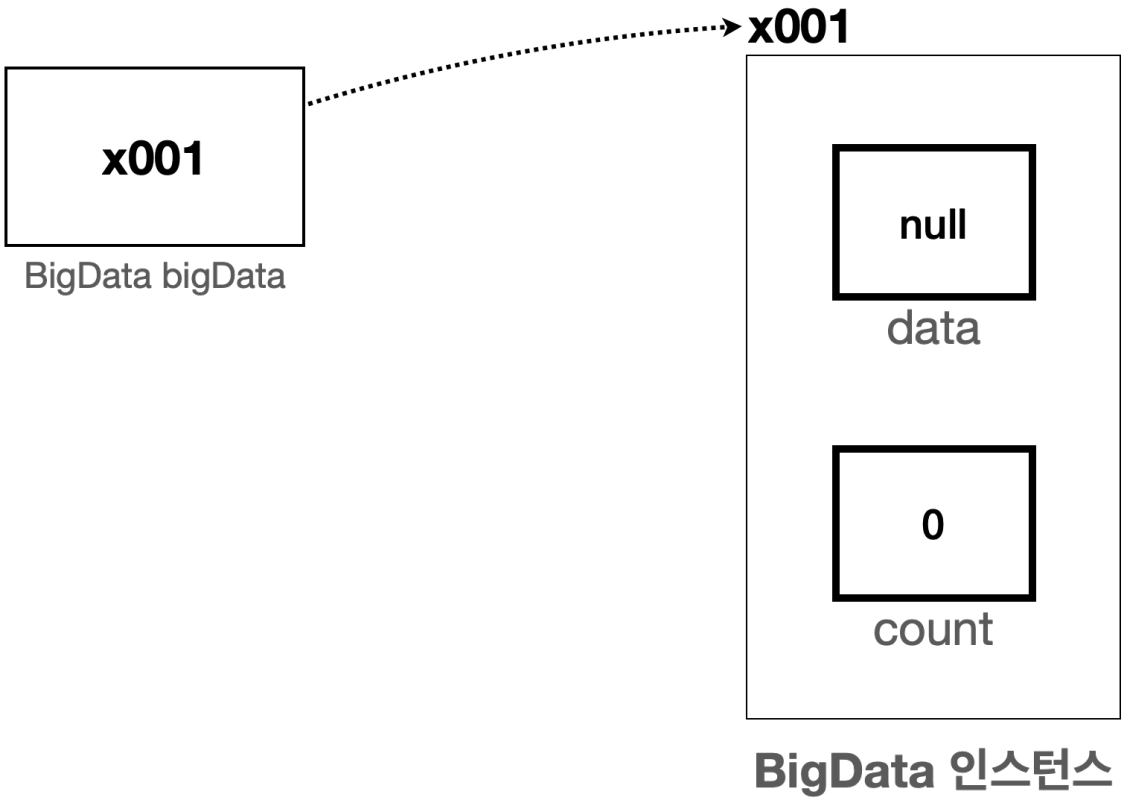
`BigData` 클래스는 `Data data`, `int count` 두 변수를 가진다.

```
package ref;

public class NullMain3 {
    public static void main(String[] args) {
        BigData bigData = new BigData();
        System.out.println("bigData.count=" + bigData.count);
        System.out.println("bigData.data=" + bigData.data);

        //NullPointerException
        System.out.println("bigData.data.value=" + bigData.data.value);
    }
}
```

실행 결과

```
bigData.count=0
```

```
bigData.data=null
Exception in thread "main" java.lang.NullPointerException: Cannot read field
"value" because "bigData.data" is null
    at ref.NullMain3.main(NullMain3.java:10)
```



`BigData`를 생성하면 `BigData`의 인스턴스가 생성된다. 이때 `BigData` 인스턴스의 멤버 변수에 초기화가 일어나는데, `BigData`의 `data` 멤버 변수는 참조형이므로 `null`로 초기화 된다. `count` 멤버 변수는 숫자이므로 `0`으로 초기화된다.

- `bigData.count`를 출력하면 `0`이 출력된다.
- `bigData.data`를 출력하면 참조값인 `null`이 출력된다. 이 변수는 아직 아무것도 참조하고 있지 않다.
- `bigData.data.value`를 출력하면 `data`의 값이 `null`이므로 `null`에 `.`(dot)을 찍게 되고, 따라서 참조할 곳이 없으므로 `NullPointerException` 예외가 발생한다.

예외 발생 과정

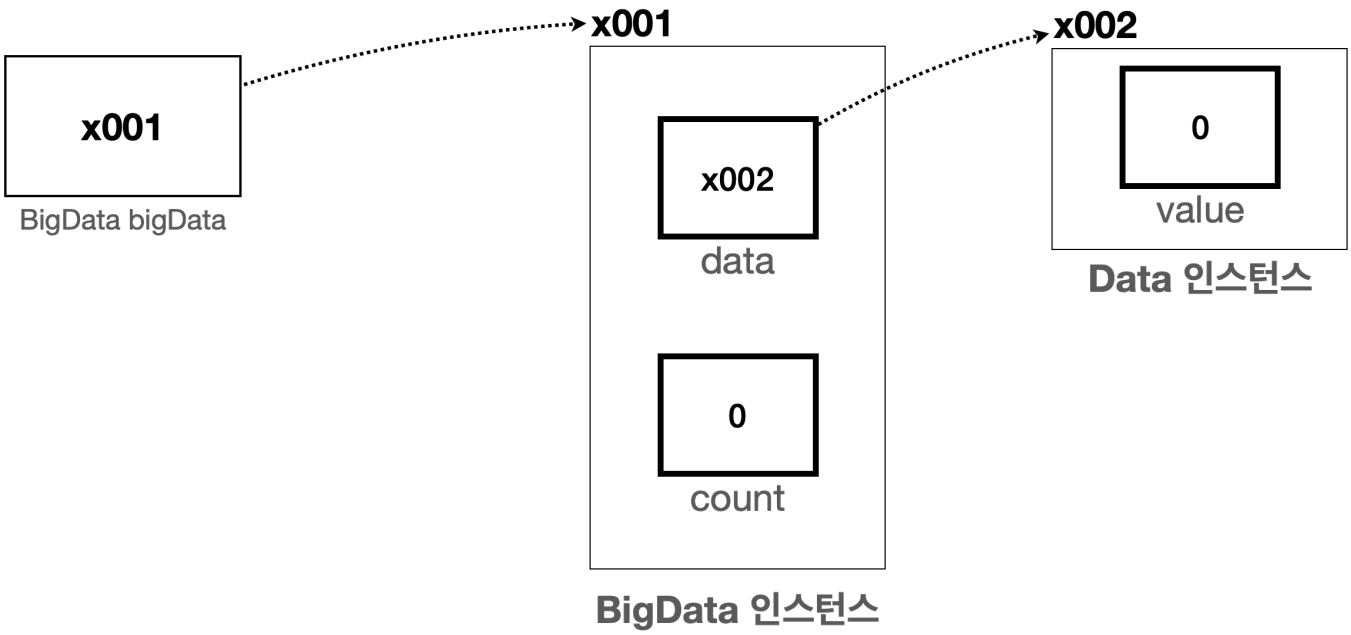
```
bigData.data.value
x001.data.value //bigData는 x001 참조값을 가진다.
null.value //x001.data는 null 값을 가진다.
NullPointerException //null 값에 .(dot)을 찍으면 예외가 발생한다.
```

이 문제를 해결하려면 `Data` 인스턴스를 만들고 `BigData.data` 멤버 변수에 참조값을 할당하면 된다.

```java
package ref;

public class NullMain4 {
    public static void main(String[] args) {
        BigData bigData = new BigData();
        bigData.data = new Data();
        System.out.println("bigData.count=" + bigData.count);
        System.out.println("bigData.data=" + bigData.data);
        System.out.println("bigData.data.value=" + bigData.data.value);
    }
}
```

실행 결과

```
bigData.count=0
bigData.data=ref.Data@x002
bigData.data.value=0
```



실행 과정

```
bigData.data.value
x001.data.value //bigData는 x001 참조값을 가진다.
x002.value //x001.data는 x002 값을 가진다.
0 // 최종 결과
```

정리

`NullPointerException`이 발생하면 `null` 값에 `.`(dot)을 찍었다고 생각하면 문제를 쉽게 찾을 수 있다.

문제와 풀이

문제: 상품 주문 시스템 개발 - 리팩토링

문제 설명

앞서 만들었던 다음 클래스에 있는 "상품 주문 시스템"을 리팩토링 하자.

`class1.ex.ProductOrderMain`

당신은 온라인 상점의 주문 관리 시스템을 만들려고 한다.

먼저, 상품 주문 정보를 담을 수 있는 `ProductOrder` 클래스를 만들어보자.

요구 사항

`ProductOrder` 클래스는 다음과 같은 멤버 변수를 포함해야 한다.

- 상품명 (`productName`)
- 가격 (`price`)
- 주문 수량 (`quantity`)

예시 코드 구조

```
package ref.ex;

public class ProductOrder {
    String productName;
    int price;
    int quantity;
}
```

- 이 코드도 `ref.ex` 패키지에 새로 만들어야 한다.

다음으로 `ref.ex.ProductOrderMain2` 클래스 안에 `main()` 메서드를 포함하여, 여러 상품의 주문 정보를 배열로 관리하고, 그 정보들을 출력하고, 최종 결제 금액을 계산하여 출력하자. 이 클래스에서는 다음과 같은 메서드를 포함해야 합니다:

- `static ProductOrder createOrder(String productName, int price, int quantity)`
  - `ProductOrder`를 생성하고 매개변수로 초기값을 설정합니다. 마지막으로 생성한 `ProductOrder`를 반환합니다.
- `static void printOrders(ProductOrder[] orders)`

    - 배열을 받아서 배열에 들어있는 전체 `ProductOrder`의 상품명, 가격, 수량을 출력합니다.
- `static int getTotalAmount(ProductOrder[] orders)`
    - 배열을 받아서 배열에 들어있는 전체 `ProductOrder`의 총 결제 금액을 계산하고, 계산 결과를 반환합니다.

예시 코드 구조

```
package ref.ex;

public class ProductOrderMain2 {
    public static void main(String[] args) {
        // 여러 상품의 주문 정보를 담는 배열 생성
        // createOrder()를 여러번 사용해서 상품 주문 정보들을 생성하고 배열에 저장
        // printOrders()를 사용해서 상품 주문 정보 출력
        // getTotalAmount()를 사용해서 총 결제 금액 계산
        // 총 결제 금액 출력
    }
}
```

출력 예시

```
상품명: 두부, 가격: 2000, 수량: 2
상품명: 김치, 가격: 5000, 수량: 1
상품명: 콜라, 가격: 1500, 수량: 2
총 결제 금액: 12000
```

정답

```
package ref.ex;

public class ProductOrder {
    String productName;
    int price;
    int quantity;
}
```

```
package ref.ex;

public class ProductOrderMain2 {

    public static void main(String[] args) {
        ProductOrder[] orders = new ProductOrder[3];
        orders[0] = createOrder("두부", 2000, 2);
```

```
        orders[1] = createOrder("김치", 5000, 1);
        orders[2] = createOrder("콜라", 1500, 2);

        printOrders(orders);
        int totalAmount = getTotalAmount(orders);
        System.out.println("총 결제 금액: " + totalAmount);
    }

    static ProductOrder createOrder(String productName, int price, int quantity)
{
        ProductOrder order = new ProductOrder();
        order.productName = productName;
        order.price = price;
        order.quantity = quantity;
        return order;
    }

    static void printOrders(ProductOrder[] orders) {
        for (ProductOrder order : orders) {
            System.out.println("상품명: " + order.productName + ", 가격: " +
order.price + ", 수량: " + order.quantity);
        }
    }

    static int getTotalAmount(ProductOrder[] orders) {
        int totalAmount = 0;
        for (ProductOrder order : orders) {
            totalAmount += order.price * order.quantity;
        }
        return totalAmount;
    }
}
```

문제: 상품 주문 시스템 개발 - 사용자 입력

문제 설명

앞서 만든 상품 주문 시스템을 사용자 입력을 받도록 개선해보자.

요구 사항

- 아래 입력, 출력 예시를 참고해서 다음 사항을 적용하자.

- 주문 수량을 입력 받자
  - 예) 입력할 주문의 개수를 입력하세요:
- 가격, 수량, 상품명을 입력 받자
  - 입력시 상품 순서를 알 수 있게 "n번째 주문 정보를 입력하세요." 라는 메시지를 출력하세요.
- 입력이 끝나면 등록한 상품과 총 결제 금액을 출력하자.

입력, 출력 예시

```
입력할 주문의 개수를 입력하세요: 3
1번째 주문 정보를 입력하세요.
상품명: 두부
가격: 2000
수량: 2
2번째 주문 정보를 입력하세요.
상품명: 김치
가격: 5000
수량: 1
3번째 주문 정보를 입력하세요.
상품명: 콜라
가격: 1500
수량: 2
상품명: 두부, 가격: 2000, 수량: 2
상품명: 김치, 가격: 5000, 수량: 1
상품명: 콜라, 가격: 1500, 수량: 2
총 결제 금액: 12000
```

정답

```java
package ref.ex;

import java.util.Scanner;

public class ProductOrderMain3 {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("입력할 주문의 개수를 입력하세요: ");
        int n = scanner.nextInt();
        scanner.nextLine();

        ProductOrder[] orders = new ProductOrder[n];

        for (int i = 0; i < orders.length; i++) {
```

```
            System.out.println((i + 1) + "번째 주문 정보를 입력하세요.");

            System.out.print("상품명: ");
            String productName = scanner.nextLine();

            System.out.print("가격: ");
            int price = scanner.nextInt();

            System.out.print("수량: ");
            int quantity = scanner.nextInt();
            scanner.nextLine(); // 입력 버퍼를 비우기 위한 코드

            orders[i] = createOrder(productName, price, quantity);
        }

        printOrders(orders);
        int totalAmount = getTotalAmount(orders);
        System.out.println("총 결제 금액: " + totalAmount);
    }

    static ProductOrder createOrder(String productName, int price, int quantity)
{
        ProductOrder order1 = new ProductOrder();
        order1.productName = productName;
        order1.price = price;
        order1.quantity = quantity;
        return order1;
    }

    static void printOrders(ProductOrder[] orders) {
        for (ProductOrder order : orders) {
            System.out.println("상품명: " + order.productName + ", 가격: " +
order.price + ", 수량: " + order.quantity);
        }
    }

    static int getTotalAmount(ProductOrder[] orders) {
        int totalAmount = 0;
        for (ProductOrder order : orders) {
            totalAmount += order.price * order.quantity;
        }
        return totalAmount;
    }
```

```
}
```

정리

대원칙: 자바는 항상 변수의 값을 복사해서 대입한다.

자바에서 변수에 값을 대입하는 것은 변수에 들어 있는 값을 복사해서 대입하는 것이다.

기본형, 참조형 모두 항상 변수에 있는 값을 복사해서 대입한다. 기본형이면 변수에 들어 있는 실제 사용하는 값을 복사해서 대입하고, 참조형이면 변수에 들어 있는 참조값을 복사해서 대입한다.

기본형이든 참조형이든 변수의 값을 대입하는 방식은 같다. 하지만 기본형과 참조형에 따라 동작하는 방식이 달라진다.

기본형 vs 참조형 - 기본

- 자바의 데이터 타입을 가장 크게 보면 기본형과 참조형으로 나눌 수 있다.
- 기본형을 제외한 나머지 변수는 모두 참조형이다. 클래스와 배열을 다루는 변수는, 참조형이다.
- 기본형 변수는 값을 직접 저장하지만, 참조형 변수는 참조(주소)를 저장한다.
- 기본형 변수는 산술 연산을 수행할 수 있지만, 참조형 변수는 산술 연산을 수행할 수 없다.
- 기본형 변수는 `null`을 할당할 수 없지만, 참조형 변수는 `null`을 할당할 수 있다.

기본형 vs 참조형 - 대입

- 기본형과 참조형 모두 대입시 변수 안에 있는 값을 읽고 복사해서 전달한다.
- 기본형은 사용하는 값을 복사해서 전달하고, 참조형은 참조값을 복사해서 전달한다! 이것이 중요하다. 실제 인스턴스가 복사되는 것이 아니다. 인스턴스를 가리키는 참조값을 복사해서 전달하는 것이다! 따라서 하나의 인스턴스를 여러곳에서 참조할 수 있다.
- 헷갈리면 그냥 변수 안에 들어간 값을 떠올려보자. 기본형은 사용하는 값이, 참조형은 참조값이 들어있다! 변수에 어떤 값이 들어있든간에 그 값을 그대로 복사해서 전달한다.

기본형 vs 참조형 - 메서드 호출

- 메서드 호출시 기본형은 메서드 내부에서 매개변수(파라미터)의 값을 변경해도 호출자의 변수 값에는 영향이 없다.
- 메서드 호출시 참조형은 메서드 내부에서 매개변수(파라미터)로 전달된 객체의 멤버 변수를 변경하면, 호출자의 객체도 변경된다.