8. final

#1.인강/0.자바/2.자바-기본

- /final 변수와 상수1
- /final 변수와 상수2
- /final 변수와 참조
- /정리

final 변수와 상수1

`final` 키워드는 이름 그대로 끝! 이라는 뜻이다.
변수에 `final` 키워드가 붙으면 더는 값을 변경할 수 없다.

참고로 `final`은 `class`, `method`를 포함한 여러 곳에 붙을 수 있다. 지금은 변수에 붙는 `final` 키워드를 알아보자. 나머지는 `final`의 사용법은 상속을 설명한 이후에 설명한다.

final - 지역 변수

```
package final1;

public class FinalLocalMain {

    public static void main(String[] args) {
        //final 지역 변수1
        final int data1;
        data1 = 10; //최초 한번만 할당 가능
        //data1 = 20; //컴파일 오류

        //final 지역 변수2
        final int data2 = 10;
        //data2 = 20; //컴파일 오류

        method(10);
    }

    //final 매개변수
    static void method(final int parameter) {
        //parameter = 20; 컴파일 오류
```

```
    }
}
```

- `final`을 지역 변수에 설정할 경우 최초 한번만 할당할 수 있다. 이후에 변수의 값을 변경하려면 컴파일 오류가 발생한다.
- `final`을 지역 변수 선언시 바로 초기화 한 경우 이미 값이 할당되었기 때문에 값을 할당할 수 없다.
- 매개변수에 `final`이 붙으면 메서드 내부에서 매개변수의 값을 변경할 수 없다. 따라서 메서드 호출 시점에 사용된 값이 끝까지 사용된다.

final - 필드(멤버 변수)

```
package final1;

//final 필드 - 생성자 초기화
public class ConstructInit {
    final int value;

    public ConstructInit(int value) {
        this.value = value;
    }
}
```

- `final`을 필드에 사용할 경우 해당 필드는 생성자를 통해서 한번만 초기화 될 수 있다.

```
package final1;

//final 필드 - 필드 초기화
public class FieldInit {
    static final int CONST_VALUE = 10;
    final int value = 10;
}
```

- `final` 필드를 필드에서 초기화하면 이미 값이 설정되었기 때문에 생성자를 통해서도 초기화 할 수 없다. `value` 필드를 참고하자.
- 코드에서 보는 것 처럼 `static` 변수에도 `final`을 선언할 수 있다.
  - `CONST_VALUE`로 변수 작명 방법이 대문자를 사용했는데, 이 부분은 바로 뒤에 상수에서 설명한다.

```
```

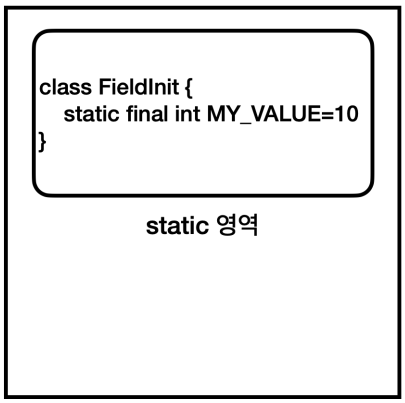
```java
package final1;

public class FinalFieldMain {

    public static void main(String[] args) {
        //final 필드 - 생성자 초기화
        System.out.println("생성자 초기화");
        ConstructInit constructInit1 = new ConstructInit(10);
        ConstructInit constructInit2 = new ConstructInit(20);
        System.out.println(constructInit1.value);
        System.out.println(constructInit2.value);

        //final 필드 - 필드 초기화
        System.out.println("필드 초기화");
        FieldInit fieldInit1 = new FieldInit();
        FieldInit fieldInit2 = new FieldInit();
        FieldInit fieldInit3 = new FieldInit();
        System.out.println(fieldInit1.value);
        System.out.println(fieldInit2.value);
        System.out.println(fieldInit3.value);

        //상수
        System.out.println("상수");
        System.out.println(FieldInit.CONST_VALUE);
    }
}
```

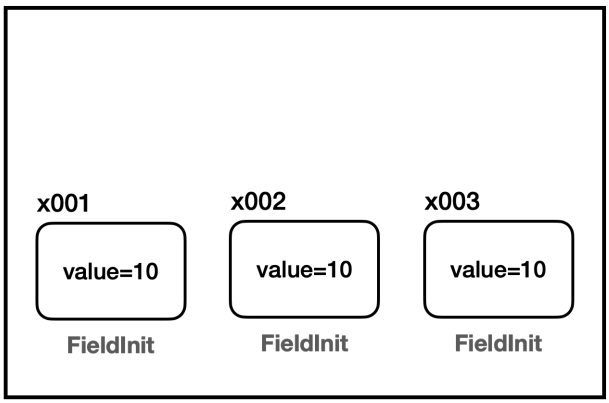
실행 결과

```
생성자 초기화
10
20
필드 초기화
10
10
10
상수
10
```

`ConstructInit`과 같이 생성자를 사용해서 `final` 필드를 초기화 하는 경우, 각 인스턴스마다 `final` 필드에 다

른 값을 할당할 수 있다. 물론 `final`을 사용했기 때문에 생성 이후에 이 값을 변경하는 것은 불가능하다.





- `FieldInit`과 같이 `final` 필드를 필드에서 초기화 하는 경우, 모든 인스턴스가 다음 오른쪽 그림과 같이 같은 값을 가진다.
- 여기서는 `FieldInit` 인스턴스의 모든 `value` 값은 `10`이 된다.
- 왜냐하면 생성자 초기화와 다르게 필드 초기화는 필드의 코드에 해당 값이 미리 정해져있기 때문이다.
- 모든 인스턴스가 같은 값을 사용하기 때문에 결과적으로 메모리를 낭비하게 된다.(물론 JVM에 따라서 내부 최적화를 시도할 수 있다) 또 메모리 낭비를 떠나서 같은 값이 계속 생성되는 것은 개발자가 보기에 명확한 중복이다. 이럴 때 사용하면 좋은 것이 바로 `static` 영역이다.

static final

- `FieldInit.MY_VALUE`는 `static` 영역에 존재한다. 그리고 `final` 키워드를 사용해서 초기화 값이 변하지 않는다.
- `static` 영역은 단 하나만 존재하는 영역이다. `MY_VALUE` 변수는 JVM 상에서 하나만 존재하므로 앞서 설명한 중복과 메모리 비효율 문제를 모두 해결할 수 있다.

이런 이유로 필드에 `final` + 필드 초기화를 사용하는 경우 `static`을 붙여서 사용하는 것이 효과적이다.

final 변수와 상수2

상수(Constant)

상수는 변하지 않고, 항상 일정한 값을 갖는 수를 말한다. 자바에서는 보통 단 하나만 존재하는 변하지 않는 고정된 값을 상수라 한다.

이런 이유로 상수는 `static final` 키워드를 사용한다.

자바 상수 특징

- `static final` 키워드를 사용한다.
- 대문자를 사용하고 구분은 `_`(언더스코어)로 한다. (관례)
  - 일반적인 변수와 상수를 구분하기 위해 이렇게 한다.
- 필드를 직접 접근해서 사용한다.
  - 상수는 기능이 아니라 고정된 값 자체를 사용하는 것이 목적이다.
  - 상수는 값을 변경할 수 없다. 따라서 필드에 직접 접근해도 데이터가 변하는 문제가 발생하지 않는다.

```
package final1;

//상수
public class Constant {
    //수학 상수
    public static final double PI = 3.14;
    //시간 상수
    public static final int HOURS_IN_DAY = 24;
    public static final int MINUTES_IN_HOUR = 60;
    public static final int SECONDS_IN_MINUTE = 60;
    //애플리케이션 설정 상수
    public static final int MAX_USERS = 1000;
}
```

- 애플리케이션 안에는 다양한 상수가 존재할 수 있다. 수학, 시간 등등 실생활에서 사용하는 상수부터, 애플리케이션의 다양한 설정을 위한 상수들도 있다.
- 보통 이런 상수들은 애플리케이션 전반에서 사용되기 때문에 `public`를 자주 사용한다. 물론 특정 위치에서만 사용된다면 다른 접근 제어자를 사용하면 된다.
- 상수는 중앙에서 값을 하나로 관리할 수 있다는 장점도 있다.
- 상수는 런타임에 변경할 수 없다. 상수를 변경하려면 프로그램을 종료하고, 코드를 변경한 다음에 프로그램을 다시 실행해야 한다.

추가로 상수는 중앙에서 값을 하나로 관리할 수 있다는 장점도 있다. 다음 두 예제를 비교해보자.

ConstantMain1 - 상수 없음

```
package final1;

public class ConstantMain1 {
```

```
    public static void main(String[] args) {
        System.out.println("프로그램 최대 참여자 수 " + 1000);
        int currentUserCount = 999;
        process(currentUserCount++);
        process(currentUserCount++);
        process(currentUserCount++);
    }

    private static void process(int currentUserCount) {
        System.out.println("참여자 수:" + currentUserCount);
        if (currentUserCount > 1000) {
            System.out.println("대기자로 등록합니다.");
        } else {
            System.out.println("게임에 참가합니다.");
        }
    }
}
```

이 코드에는 다음과 같은 문제가 있다.

- 만약 프로그램 최대 참여자 수를 현재 1000명에서 2000명으로 변경해야 하면 2곳의 변경 포인트가 발생한다. 만약 애플리케이션의 100곳에서 이 숫자를 사용했다면 100곳을 모두 변경해야 한다.
- 매직 넘버 문제가 발생했다. 숫자 `1000`이라는 것이 무슨 뜻일까? 이 값만 보고 이해하기 어렵다.

실행 결과

```
프로그램 최대 참여자 수 1000
참여자 수:999
게임에 참가합니다.
참여자 수:1000
게임에 참가합니다.
참여자 수:1001
대기자로 등록합니다.
```

ConstantMain2 - 상수 사용

```
package final1;

public class ConstantMain2 {
```

```
    public static void main(String[] args) {
        System.out.println("프로그램 최대 참여자 수 " + Constant.MAX_USERS);
        int currentUserCount = 999;
        process(currentUserCount++);
        process(currentUserCount++);
        process(currentUserCount++);
    }

    private static void process(int currentUserCount) {
        System.out.println("참여자 수:" + currentUserCount);
        if (currentUserCount > Constant.MAX_USERS) {
            System.out.println("대기자로 등록합니다.");
        } else {
            System.out.println("게임에 참가합니다.");
        }
    }
}
```

- `Constant.MAX_USERS` 상수를 사용했다. 만약 프로그램 최대 참여자 수를 변경해야 하면 `Constant.MAX_USERS`의 상수 값만 변경하면 된다.
- 매직 넘버 문제를 해결했다. 숫자 `1000`이 아니라 사람이 인지할 수 있게 `MAX_USERS`라는 변수명으로 코드로 이해할 수 있다.

final 변수와 참조

`final`은 변수의 값을 변경하지 못하게 막는다. 그런데 여기서 변수의 값이라는 것이 뭘까?

- 변수는 크게 기본형 변수와 참조형 변수가 있다.
- 기본형 변수는 `10`, `20` 같은 값을 보관하고, 참조형 변수는 객체의 참조값을 보관한다.
  - `final`을 기본형 변수에 사용하면 값을 변경할 수 없다.
  - `final`을 참조형 변수에 사용하면 참조값을 변경할 수 없다.

여기까지는 이해하는데 어려움이 없을 것이다. 이번에는 약간 복잡한 예제를 만들어 보자.

```
package final1;

public class Data {
```

```
    public int value;
}
```

- `int value`: `final`이 아니다. 변경할 수 있는 변수다.

```
package final1;

public class FinalRefMain {

    public static void main(String[] args) {
        final Data data = new Data();
        //data = new Data(); //final 변경 불가 컴파일 오류

        //참조 대상의 값은 변경 가능
        data.value = 10;
        System.out.println(data.value);
        data.value = 20;
        System.out.println(data.value);
    }
}
```

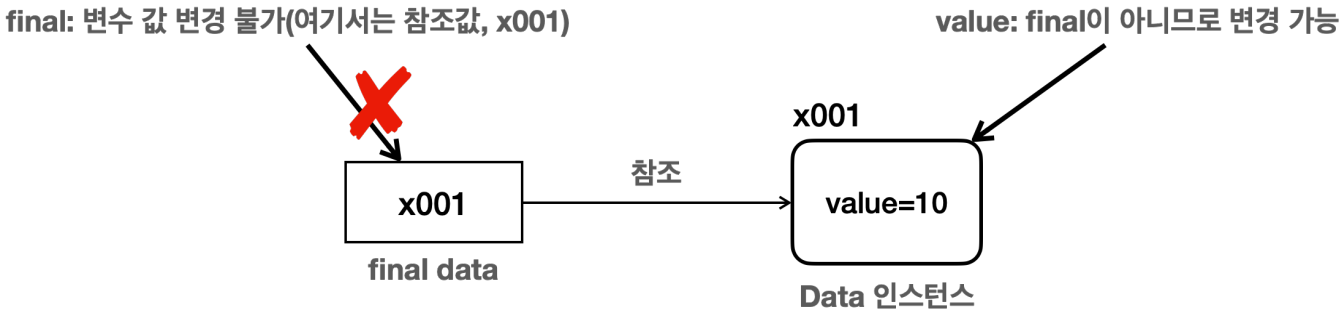
```
final Data data = new Data()
//data = new Data(); //final 변경 불가 컴파일 오류
```

참조형 변수 `data`에 `final`이 붙었다. 변수 선언 시점에 참조값을 할당했으므로 더는 참조값을 변경할 수 없다.

```
data.value = 10
data.value = 20
```

그런데 참조 대상의 객체 값은 변경할 수 있다.

- 참조형 변수 `data`에 `final`이 붙었다. 이 경우 참조형 변수에 들어있는 참조값을 다른 값으로 변경하지 못한다. 쉽게 이야기해서 이제 다른 객체를 참조할 수 없다. 그런데 이것의 정확한 뜻을 잘 이해해야 한다. 참조형 변수에 들어있는 참조값만 변경하지 못한다는 뜻이다. 이 변수 이외에 다른 곳에 영향을 주는 것이 아니다.
- `Data.value`는 `final`이 아니다. 따라서 값을 변경할 수 있다.



정리하면 참조형 변수에 `final`이 붙으면 참조 대상을 자체를 다른 대상으로 변경하지 못하는 것이지, 참조하는 대상의 값은 변경할 수 있다.

정리

`final`은 매우 유용한 제약이다. 만약 특정 변수의 값을 할당한 이후에 변경하지 않아야 한다면 `final`을 사용하자.
예를 들어서 고객의 `id` 변경하면 큰 문제가 발생한다면 `final`로 선언하고 생성자로 값을 할당하자.
만약 어디선가 실수로 `id` 값을 변경한다면 컴파일러가 문제를 찾아줄 것이다.

```
package final1.ex;

public class Member {

    private final String id; //final 키워드 사용
    private String name;

    public Member(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public void changeData(String id, String name) {
        //this.id = id; //컴파일 오류 발생
        this.name = name;
    }

    public void print() {
        System.out.println("id:" + id + ", name:" + name);
    }
```

```
}
```

- `changeData()` 메서드에서 `final`인 `id` 값 변경을 시도하면 컴파일 오류가 발생한다.

```
package final1.ex;

public class MemberMain {

    public static void main(String[] args) {
        Member member = new Member("myId", "kim");
        member.print();
        member.changeData("myId2","seo");
        member.print();
    }
}
```

실행 결과

```
id:myId, name:kim
id:myId, name:seo
```