# Item 11

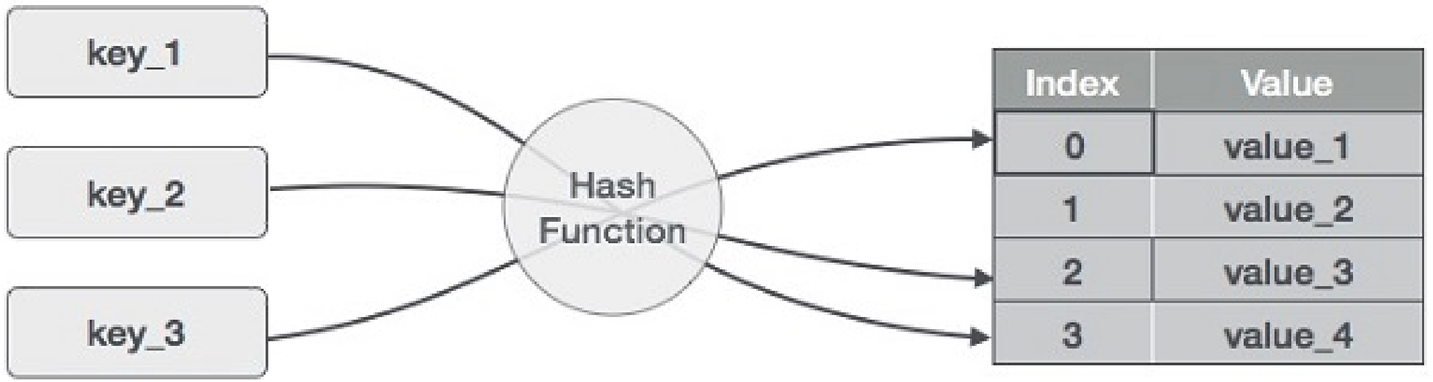
equals를 재정의하려거든 hashCode도 재정의하라

# Contents

# hashCode()란



- Object 클래스의 메서드
- 해싱 기법에 사용되는 해시 함수 구현
- 입력의 해시 코드 반환하는 메서드



# hashCode() 규약



1. equals 비교 사용되는 정보 변경 X -> hashCode 반환값 변경 X.

2. equals(Object) 두 객체 같다고 판단 -> 두 객체의 hashCode 같은 값 반환.

3. equals(Object) 두 객체 다르다고 판단 -> 두 객체의 hashCode 서로 다른 값 반환할 필요 X.

# hashCode() 재정의 X 문제점



2. equals(Object) 두 객체 같다고 판단 -> 두 객체의 hashCode 같은 값 반환.

재정의 X -> HashMap, HashSet 원소로 사용 시 문제 발생

**why?**

# hashCode() 재정의 X 문제점

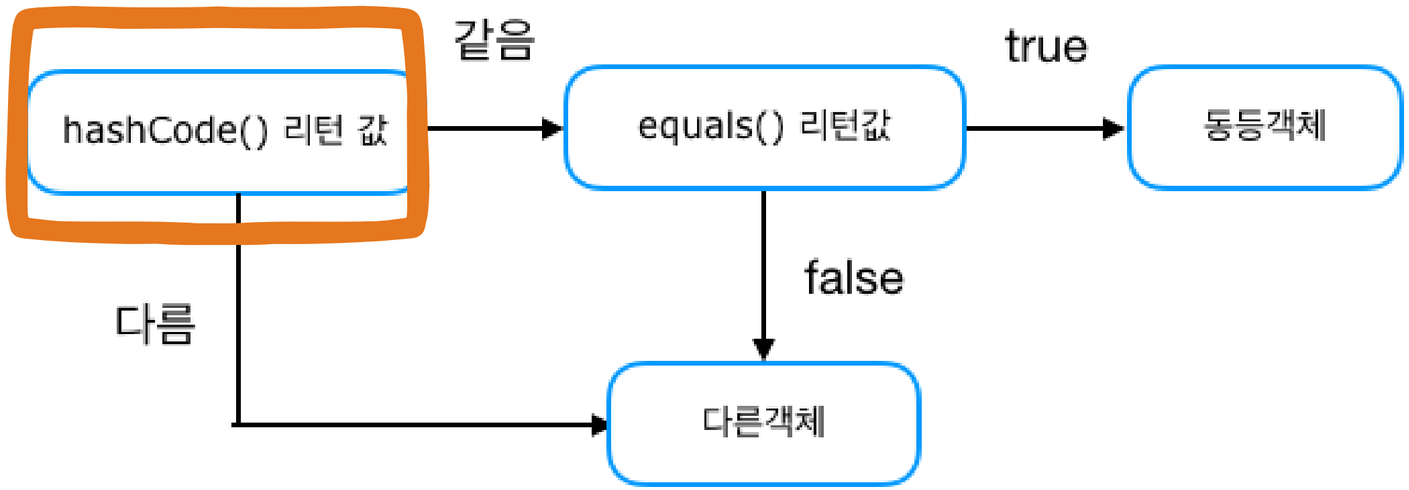


```
Map<One3,String> m = new HashMap<>();
m.put(new One3( a: 5, b: 3),"KIM");
System.out.println(m.get(new One3( a: 5, b: 3))); //null

Set<One3> set = new HashSet<>();
set.add(new One3( a: 5, b: 3));
System.out.println(set.contains(new One3( a: 5, b: 3))); //false
```

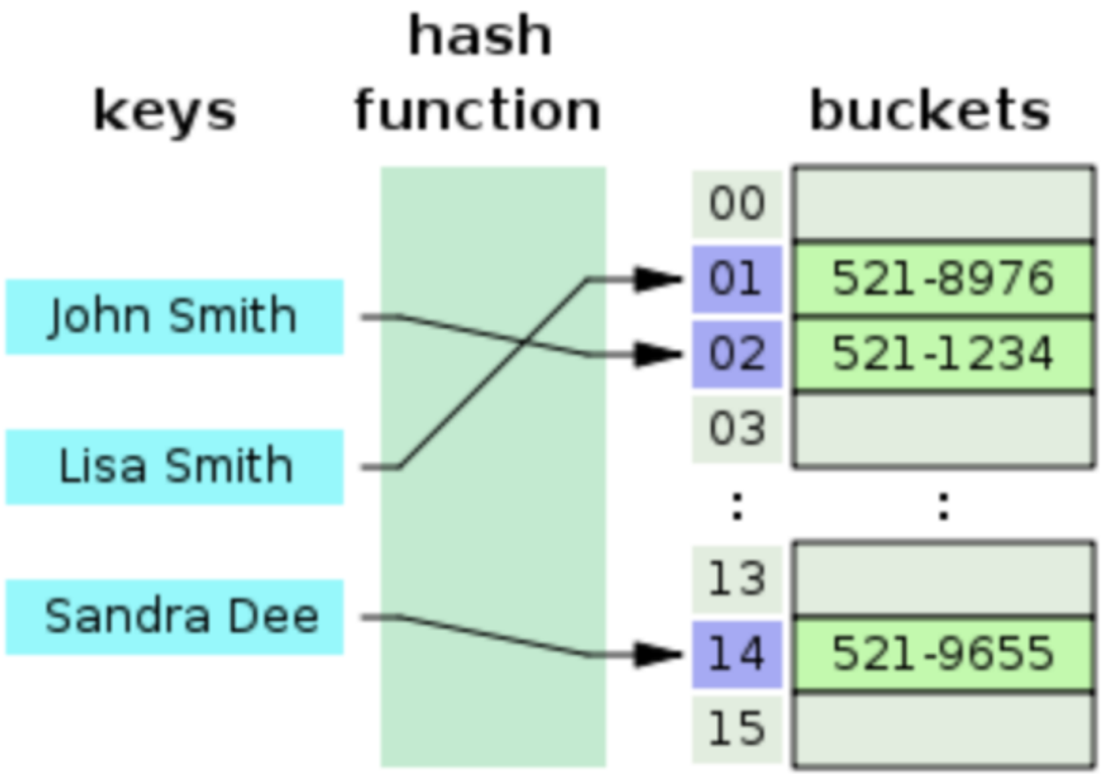
# hashCode() 재정의 X 문제점

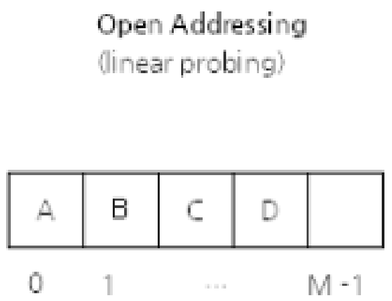


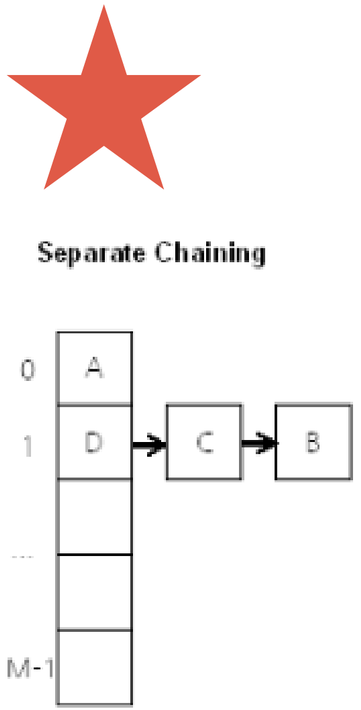


# hashCode() 재정의 X 문제점 - HashMap





# hashCode() 재정의 X 문제점



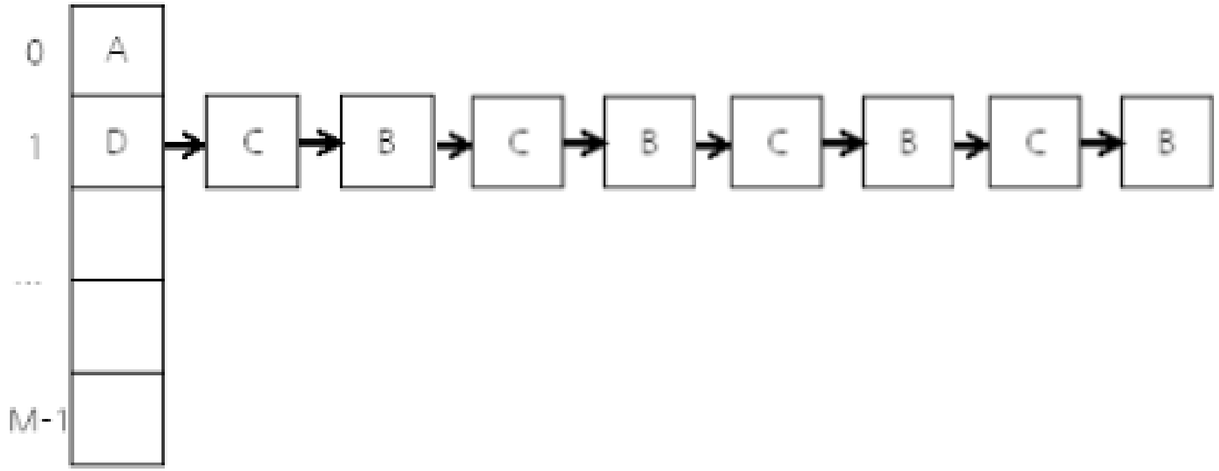
hashCode 메서드 재정의 X -> 다른 해시코드 반환

**2. equals(Object) 두 객체 같다고 판단 -> 두 객체의 hashCode 같은 값 반환.**

# Bad hashCode() 재정의



```java
@Override public int hashCode(){
        return 42;
}
```



# Good hashCode() 재정의



## 요령

1. int 변수인 result 선언한 후 값을 c로 초기화.
    - c는 해당 객체의 첫번째 핵심 필드를 단계 2.1 방식으로 계산한 해시코드.
2. 해당 객체의 나머지 핵심 필드인 f 각각에 대해 다음 작업 수행.
    a. 해당 필드의 해시코드 c 계산.
    b. 단계 2.1에서 계산한 해시코드 c로 result를 갱신.
        - result = 31 * result + c;
3. result 반환.

# Good hashCode() 재정의



```java
// 코드 11-2 전형적인 hashCode 메서드 (70쪽)
@Override public int hashCode() {
    int result = Short.hashCode(areaCode);
    result = 31 * result + Short.hashCode(prefix);
    result = 31 * result + Short.hashCode(lineNum);
    return result;
}
```

# hashCode() 재정의 후 할 일



- hashCode 메서드 동치인 인스턴스에 대해 똑같은 해시코드 반환하는지 자문
- 똑같은 해시코드 반환할 것이라는 직관을 검증할 단위 테스트 작성
- 동치인 인스턴스가 서로 다른 해시코드 반환한다면 원인 찾아 해결

# hashCode() 재정의 주의점 & 유의점



- 파생 필드는 해시코드 계산에서 제외해도 된다.
- equals 비교에 사용되지 않은 필드는 반드시 제외해야 한다.
- 클래스 불변이고 해시코드 계산하는 비용 크다면 캐싱 방식 고려해야 한다.
- 성능 높인다고 해시코드 계산할 때 핵심 필드를 생략해서는 안된다.
- hashCode 반환하는 값의 생성 규칙 API 사용자에게 공표하지 말아야 한다.

# 1. 파생 필드는 해시코드 계산에서 제외해도 된다.

파생 필드 : 다른 필드로부터 계산 가능한 필드

## Why??

필드 a, b, c 사용

```java
public class One1 {
    private int a;
    private int b;
    private int c;

    public One1(int a, int b) {
        this.a = a;
        this.b = b;
        this.c = a+b;
    }

    @Override
    public int hashCode() { return Objects.hash(a, b, c); }
```

필드 a, b 사용

```java
public class One2 {
    private int a;
    private int b;
    private int c;

    public One2(int a, int b) {
        this.a = a;
        this.b = b;
        this.c = a+b;
    }

    @Override
    public int hashCode() { return Objects.hash(a, b); }
```

# 1. 파생 필드는 해시코드 계산에서 제외해도 된다.



```java
public static void main(String[] args) {
    One2 one1 = new One2( a: 5, b: 3);
    One2 one2 = new One2( a: 5, b: 3);

    One1 one3 = new One1( a: 5, b: 3);
    One1 one4 = new One1( a: 5, b: 3);

    System.out.println("not include c: " + one1.hashCode());
    System.out.println("not include c: " + one2.hashCode());

    System.out.println();

    System.out.println("include c: " + one3.hashCode());
    System.out.println("include c: " + one4.hashCode());
}
```

```
not include c: 1119
not include c: 1119

include c: 34697
include c: 34697

Process finished with exit code 0
```

# 2. equals 비교에 사용되지 않은 필드는 반드시 제외.



## Why??

equals 메서드 : 필드 a, b 사용

hashCode 메서드 : 필드 a, b, c 사용

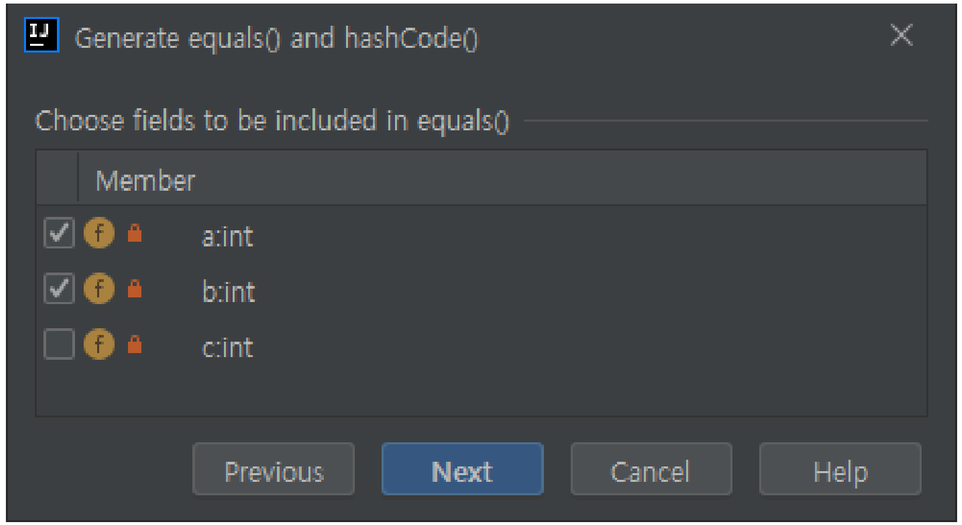
| 인스턴스 1 | 인스턴스 2 |
| --- | --- |
| a = 2 | a = 2 |
| b = 3 | b = 3 |
| c = 5 | c = 4 |

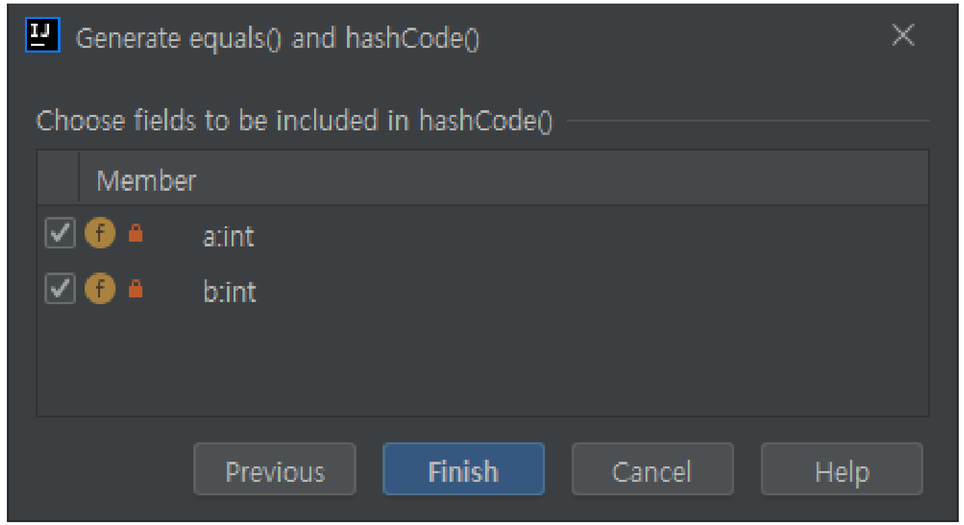
equals 결과 -> 같다

hashCode 결과 -> 다르다

# 2. equals 비교에 사용되지 않은 필드는 반드시 제외.





# 3. 클래스 불변이고 해시코드 계산 비용 크다면 캐싱 방식 고려.



객체가 주로 HashMap or HashSet의 키로 사용 될 경우

## Why??

동작 방식이 먼저 키의 hashCode() 메서드 호출해 해시 코드 계산

-> 미리 계산해 두면 매번 hashCode 메서드 호출 안해도 됨
& 불변 클래스라 바뀌지 않아 계산해 놔도 됨

# 4. 성능 높인다고 해시코드 계산할 때 핵심 필드 생략 X.



## Why??

어떤 필드 특정 영역에 몰린 인스턴스들의 해시코드

넓은 범위로 고르게 퍼트려주는 효과 있을지도 모름

# 5. hashCode 생성 규칙 API 사용자에게 공표 X.



## Why??

클라이언트가 hashCode가 반환하는 값에 의지하지 않게 되고 추후에 계산 방식을 바꿀 수 있기 때문

- Objects.hash(field1, field2, ...)

```java
public static int hash(Object... values) {
    return Arrays.hashCode(values);
}


//Arrays.hashCode() 코드
public static int hashCode(Object a[]) {
    if (a == null)
        return 0;

    int result = 1;

    for (Object element : a)
        result = 31 * result + (element == null ? 0 : element.hashCode());

    return result;
}
```

- Guava : com.google.common.hash.Hashing

| | |
|---|---|
| static HashFunction | crc32()<br>Returns a hash function implementing |
| static HashFunction | crc32c()<br>Returns a hash function implementing |
| static HashFunction | farmHashFingerprint64()<br>Returns a hash function implementing |
| static HashFunction | goodFastHash(int minimumBits)<br>Returns a general-purpose, **temporar** |
| static HashFunction | hmacMd5(byte[] key)<br>Returns a hash function implementing |
| static HashFunction | hmacMd5(Key key)<br>Returns a hash function implementing |
| static HashFunction | hmacSha1(byte[] key)<br>Returns a hash function implementing |

# 감사합니다