hashcode()와 equals()

주제가 나온 이유?

> equals와 hashcode를 같이 재정의 하는 이유
>
> ex) lombok의 @EqualsAndHashCode

equals는 두 객체가 논리적(value값이)으로 동등하면 true를 반환한다.

```
int a = 3;
int b = 3;
System.out.println(a == b); // true

// string(객)은 주소값으로 비교
String e = "abcd";
String f = "abcd";
String c = new String("abcd");
String d = new String("abcd");
System.out.println(e==f);    // true, e와f는 값은 주소를 참조하고있다.
System.out.println(e==c);    // false
System.out.println(c==d);    // false

 // 하지만 반대로 equals를 쓰면?
System.out.println("e.equals(f)"+e.equals(f));   // true
System.out.println("e.equals(c)"+e.equals(c));   // true
System.out.println("c.equals(d)"+c.equals(d));   // true

// 이와 같이 equals는 내부의 값을 비교해준다.
```

equals는 default로 primitive type은 내용이 같은지 검사하고, reference type은 객체의 주소가 같은지 검사

만약에 primitive type을 equals로 비교하고 싶으면 래퍼클래스로 오토박싱을 한 후 비교하자

> 1. Primitive Types : 자바에는 8개의 primitive types(byte, short, int, long, float, double, char, boolean)이 있고, 이들은 binary bits 형식으로 그들 값을 직접 가지고 있습니다.
> 2. Reference Types : Classes, Interfaces, Enums, Arrays 등과 같은 primitive types가 아닌 모든 타입들은 Reference Type입니다. Reference Type은 "object" 그 자신을 가지는 것이 아니라, "object address" 를 가지고 있습니다.
>
> 예를 들어서, `Integer a = new Integer(5); Integer b = new Integer(5);` 가 있을 때, a와 b는 binary 값인 `5`를 가지고 있는 것이 아니라, 각각 `5`의 값을 가지고 있는 objects의 메모리 주소값을 가지고 있습니다.
>
> 그래서, `a`와 `b`를 `a == b`로 비교를 할 때는 사실 두 개의 메모리 주소를 비교하는 것이고, `false`를 얻어야합니다. 실제 `a`와 `b`의 동치 비교(equality)를 할 때는 `a.equals(b)`로 비교를 해야합니다.

Integer는 래퍼클래스이다. 이 값이 주소를 비교하는거면 래퍼클래스는 참조타입이다.

래퍼 클래스 사용 목적 : 매개변수로 객체가 요구될때, 기본형 값이 아닌 객체로 저장해야할때, 객체간의 비교가 필요할때

```
Integer a = new Integer(10); // 오토박싱

int b = a.intValue(); // 언박싱
```

=========

HashCode

객체 해시코드란 객체를 식별하는 하나의 정수값을 말한다.

==========

equals만 재정의 할 경우 hash값을 사용하는 컬렉션을 쓸때 문제가 생긴다.

예를들어보자

https://tecoble.techcourse.co.kr/post/2020-07-29-equals-and-hashCode/

Car.java

```
private final String name;

public Car(String name) {
    this.name = name;
}
```

main.java

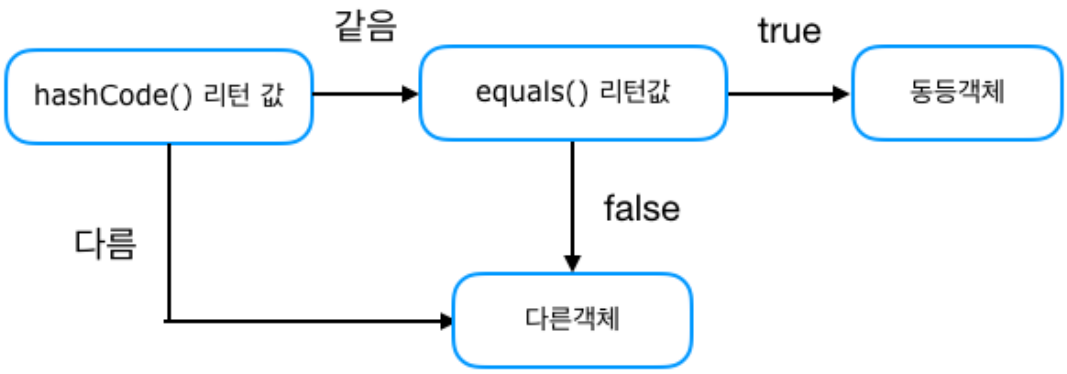
```
List<String> list = new ArrayList<>();
list.add(new String("foo"));
list.add(new String("foo"));

System.out.println("list 크기 : "+list.size());    // 중복을 허용하기 때문에 2가 출력된다.

Set<String> c = new HashSet<>();    // set은 중복을 허용하지 않는다
c.add(new String("foo"));    // string은 참조타입
c.add(new String("foo"));

System.out.println("set : "+c.size());        // 1

Set<Car> cars = new HashSet<>();    // 똑같이 중복을 허용하지 않는 hashset
Car car1 = new Car("foo");
Car car2 = new Car("foo");
cars.add(car1);
cars.add(car2);
System.out.println("set : "+cars.size());    // 2 가 출력된다.

String a = new String("foo");
String b = new String("foo");


System.out.println(car1 == car2);    // false
System.out.println(car1.equals(car2));    // false
System.out.println(a == b);            // false
System.out.println(a.equals(b));    // true
```

왜 String은 중복으로 걸러지고 Object는 안 걸러지나?

string은 값이 같을 경우 해시코드 값도 똑같기 때문에, 왜?

> "test" 와 new String("test") 은 주소값은 다르지만 해시값은 똑같다.
>
> 이유는..? TODO

set이 중복을 걸러내는 흐름 :



일단 car1, car2의 해시코드는 false, equals는 true이지만 해시코드값이 false라 다른객체로 인식되버린다.

=======

결론

hashcode를 재정의 할 때에는 equals 메소드도 함께 재정의해야 부작용이 없다.

HashMap에서 key값을 판단할 때, hashcode와 equals 메소드의 반환 값이

같아야 key가 unique하다고 판단하기 때문이다.

만약 String 클래스가 hashcode만 재정의되었다고 한다면

"test"와 new String("test") 가 모두 key값으로 등록될 수가 있다.

의도한 key unique에 도달하지 못한 것이다.