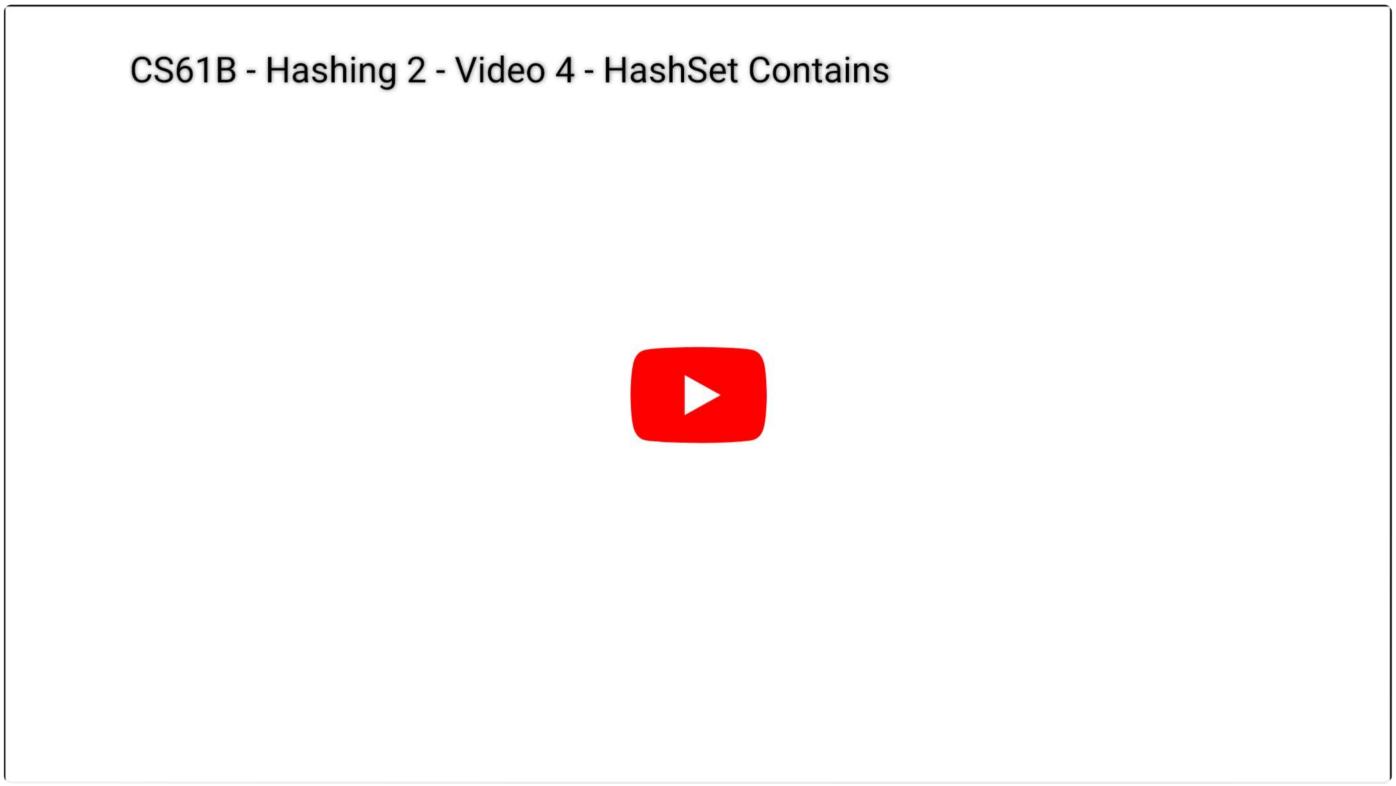
20.3 Contains & Duplicate Items

Contains



The `equals()` Method for a ColoredNumber Object

Suppose the `equals()` method for ColoredNumber is as below, i.e. two ColoredNumbers are equal if they have the same num.

```
@Override
public boolean equals(Object o) {
    if (o instanceof ColoredNumber otherCn) {
        return this.num == otherCn.num;
    }
    return false;
}
```

HashSet Behavior for Checking `contains()`

Suppose the `equals()` method for ColoredNumber is on the previous slide, i.e. two ColoredNumbers are equal if they have the same num.

```
int N = 20;
HashSet<ColoredNumber> hs = new HashSet<>();
for (int i = 0; i < N; i += 1) {
    hs.add(new ColoredNumber(i));
}
```

Suppose we now check whether 12 is in the hash table.

```
ColoredNumber twelve = new ColoredNumber(12);
hs.contains(twelve); //returns true
```

What do we **expect** to be returned by `contains`?

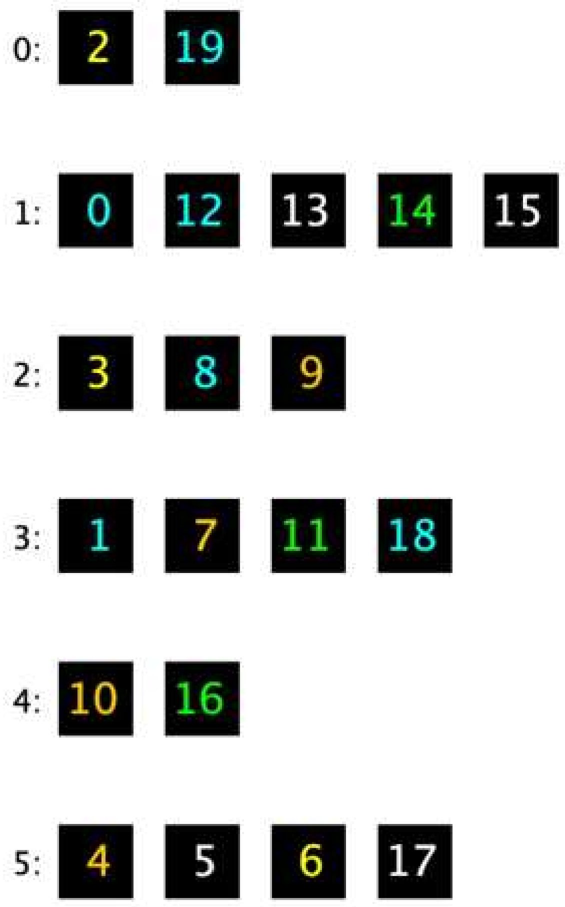
› Answer

Finding an Item Using the Default Hashcode

Suppose we are using the default hash function (uses memory address):

```
int N = 20;
HashSet<ColoredNumber> hs = new HashSet<>();
for (int i = 0; i < N; i += 1) {
   hs.add(new ColoredNumber(i));
}
ColoredNumber twelve = new ColoredNumber(12);
hs.contains(twelve); // returns ??
```

which yields the table below:



Suppose equals returns true if two ColoredNumbers have the same num (as we've defined previously).

What is actually returned by `contains`?

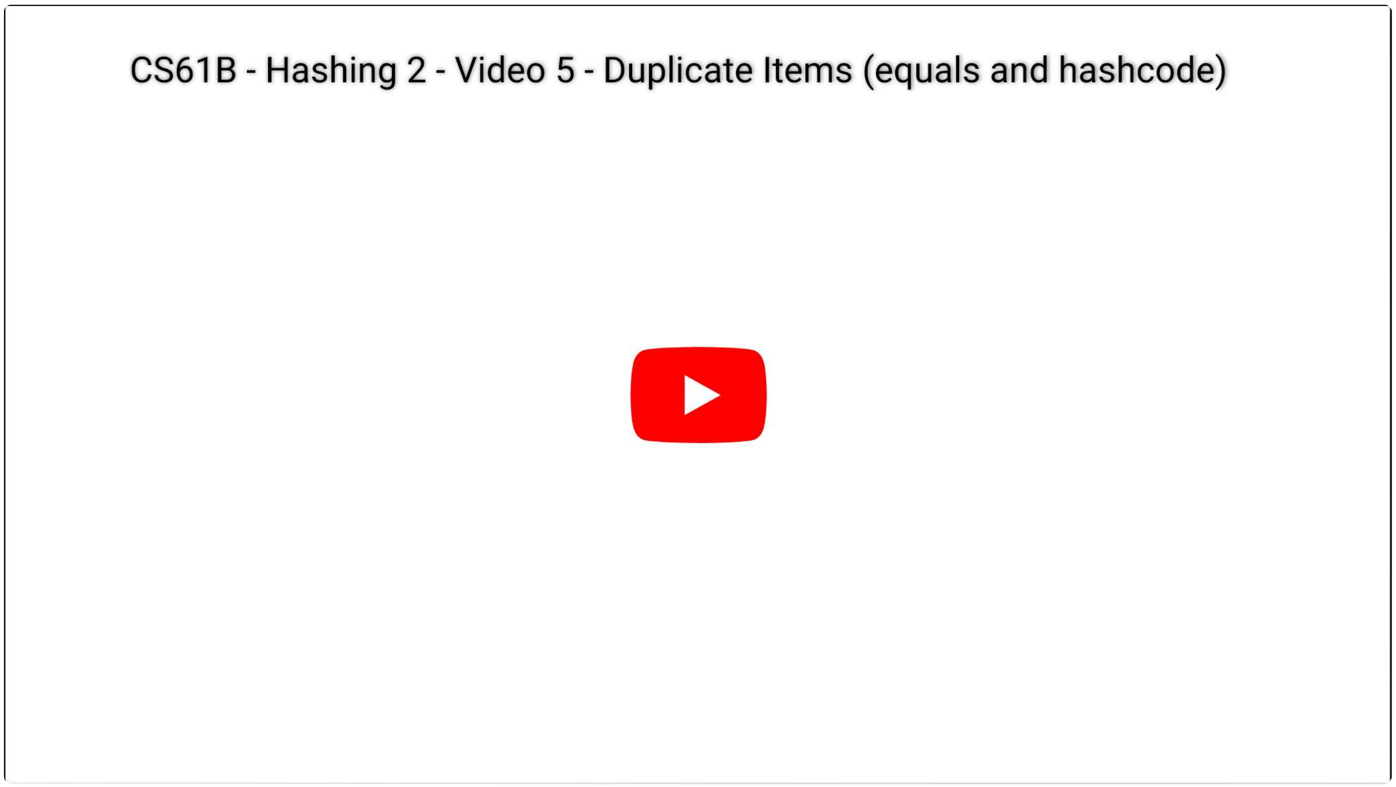
Answer

Hard Question: If the default hash code achieves a good spread, why do we even bother to create custom hash functions?

Answer

Basic rule (also definition of deterministic property of a valid hashcode): If two objects are equal, they **must** have the same hash code so the hash table can find it.

Duplicate Values

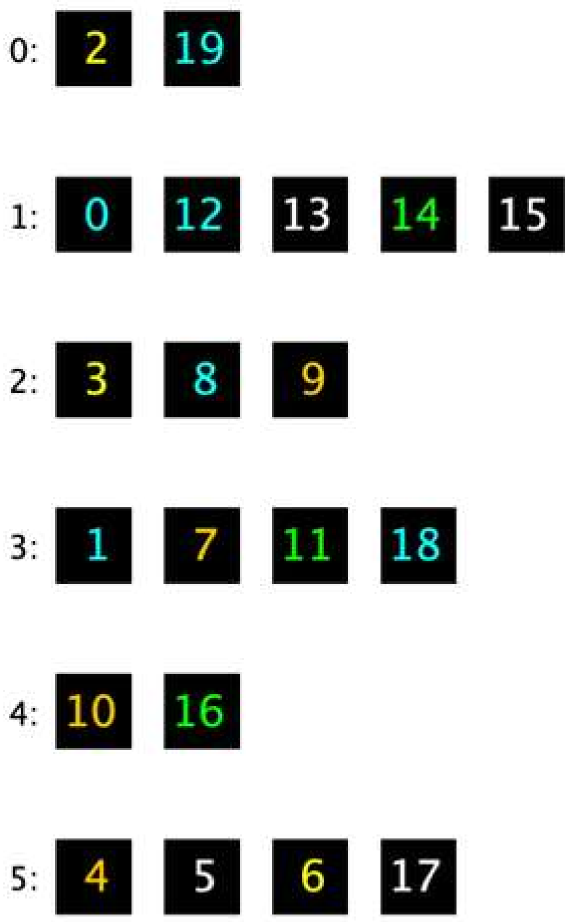

CS61B - Hashing 2 - Video 5 - Duplicate Items (equals and hashcode)

Overriding `equals()` but Not `hashCode()`

Suppose we have the same `equals()` method (comparing `num`), but we do not override `hashCode()`.

```
public boolean equals(Object o) {
    ...  return this.num == otherCn.num; ...
}
```

The result of adding 0 through 19 is shown below:



```java
ColoredNumber zero = new ColoredNumber(0);
hs.add(zero); // does another zero appear?
```

Which can happen when we call add(zero)?

Answer Choices:

1. We get a 0 to bin zero.
2. We add another 0 to bin one.
3. We add a 0 to some other bin.
4. We do not get a duplicate zero

› Answer

Key Takeaway: `equals()` and `hashCode()`

Bottom line: If your class override equals, you should also override hashCode in a consistent manner.

- If two objects are equal, they must always have the same hash code.

If you don't, everything breaks:

- `Contains` can't find objects (unless it gets lucky).
- `Add` results in duplicates.

Previous
20.2 Distribution By Other Hash Functions

Next
20.4 Mutable vs. Immutable Types

Last updated 1 year ago