Quick Refresher of Hashing for HW7

By Chris Jeng, op@berkeley.edu

Basics of hashCode

(Head First Java talks about `hashCode` and `HashSet` on pg 559 for about six total pages, a very quick and digestible read. This guide is going to suck, so you should definitely looks elsewhere to get a better feel for things)

We've seen before that all objects inherit at these two methods from the universal `Object.java`:

1) `public boolean equals(Object o)` – This means that you can make any object, and try calling the `.equals(whateverOtherObject)` on it. By default of inheritance, this works for **ALL** objects.
2) `public String toString()` – Fun fact: The default behavior is to return the address of where the object is stored in hexadecimal. That's what happens when you ask for the `.toString` of a primitive array. Notice that some Objects like `ArrayList` and `String` have overridden this sucky default behavior with more informative methods (like how `ArrayList` will print out its contents).

There is yet another addition to this list: the hash code function.

3) `public int hashCode()` – Crude description: considers the data of an instance of a class and tries to output a unique `int` based on that data.

Moral obligations of a hash code

1) (Required) If two objects are `.equals`, then their hash code values **must** be the same. This requirement isn't enforced by the compiler, but cheating this requirement would mean `HashMap` and `HashSet` can't work properly when storing this type of immoral object.
2) (Not required, but strongly preferred) If two objects are *not* `.equals`, then their hash code values are always different. If this optional requirement is satisfied, the hash function is called a "perfect hash". A perfect hash can be thought of as a one-to-one mapping.

A "bit" of math

Let $\mathcal{O}$ denote the set of all possible meaningfully-different instances of a class. In other words, `.equals` between any two objects in $\mathcal{O}$ return false.

Let $\mathbb{Z}_{32}$ denote the set of all possible numbers representable by a Java `int` (32 bits in a Java `int`, so for any `int` x, $-2^{31} \leq x \leq 2^{31} - 1$).

Then the hash function $\mathcal{H}$ is a mapping

$$\mathcal{H}: \mathcal{O} \rightarrow \mathbb{Z}_{32}$$

A perfect hash means different objects always map to different hash values. In other words,

$$\text{Perfect Hash} \Leftrightarrow \forall o_1, o_2 \in \mathcal{O}, \mathcal{H}(o_1) = \mathcal{H}(o_2) \Longrightarrow o_1 \text{ is } \texttt{.equals} \text{ to } o_2$$