Hash coding

What is a hash code?

- A hash code is a 32-bit *digest* of an object.
- A hash code should distribute all possible values of the object *uniformly* among all 4 billion possible values (the intention is to reduce the number of *collisions:* different objects, same hash).
- It is required to be consistent with *equals* such that:
  - *if a.equals(b) then a.hashCode==b.hashCode*
  - It also follows that: *if a.hashCode != b.hashCode then ! a.equals(b)*

Implementing hashCode

- It stands to reason, then, that the fields of a class that are tested in *equals* must also contribute to *hashCode,* otherwise the contract cannot be maintained.
- So, how do fields contribute to *hashCode?*
- Typically, we calculate the *hashCode* of a field by calling *hashCode* on it (or on the boxed version of it if the field is a primitive);
- Once we have the various field *hashCode* values, we typically combine them together by some formula involving prime numbers such as:
  - $H = h_1 * p_1 + h_2 * p_2 + \ldots + h_n * p_n$
  - In practice, the standard way to implement *hashCode* in Java is (in this example, n=4):
  - $H = 31 * (31 * (31 * h_1 + h_2) + h_3) + h_4$

The actual code

- Example: java.lang.String:

```
public int hashCode() {
    int h = hash; // cashed value: defaults to 0
    if (h == 0 && value.length > 0) {
        char val[] = value;
        for (int i = 0; i < value.length; i++) {
            h = 31 * h + val[i];
        }
        hash = h;
    }
    return h;
}
```

- Example: java.time.LocalDate:

```
public int hashCode() {
    int yearValue = year;
    int monthValue = month;
    int dayValue = day;
    return (yearValue & 0xFFFFF800) ^ ((yearValue << 11) + (monthValue << 6) + (dayValue));
}
```