hashCode and equals

Implementing equals

- The signature of equals in Java is:

  boolean equals(Object x)

- When implementing *equals*, we need to check for the equality each field which forms part of the “primary key” of an object. If any pair of fields is unequal, then the objects are unequal.
- Before we can compare the fields, we must establish that both objects have the same class otherwise it makes no sense to talk about comparing fields.
- And before doing that we might as well check a couple of other things that can give us an immediate result.

Actual code: *equals*

- Example: java.lang.String:

```
public boolean equals(Object anObject) {
    if (this == anObject) return true;
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = value.length;
        if (n == anotherString.value.length) {
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true;
        }
    }
    return false;
}
```

- Example: java.time.LocalDate:

```
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj instanceof LocalDate) return compareTo0((LocalDate) obj) == 0;
    return false;
}
```

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));
}
```

Actual code continued

- Example: edu.neu.coe.info6205.bqs.Element*:

```java
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Element<?> element = (Element<?>) o;
    return Objects.equals(item, element.item) &&
            Objects.equals(next, element.next);
}

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

* auto-generated by IDE