hashCode, compareTo, and equals

Searching and sorting

- There are three major methods used in the design of data structures and their algorithms:
  - *equals*: evaluates logical equality between objects;
  - *hashCode*: yields a 32-bit digest of an object;
  - *compareTo*: orders two instances of the same class, according to the intrinsic ordering of the class.

Object methods

- *Every* object in Java must implement various methods, including:
  - `public boolean equals(Object x)`
  - `public int hashCode()`
- Instances of class *X* implementing *Comparable<X>* must implement:
  - `public int compareTo (X x)`
- Thus all objects implement *equals* and *hashCode* but only objects that have an intrinsic ordering implement *compareTo*.

What do we mean by *equals*?

- If we want to know if this and that are the *same* object, we can use == which tests that the references are the same;

- But, for most uses, we want to know if two objects are equivalent, i.e. the program (data structure + algorithm) will operate the same if we substitute one for the other.

- We usually check the equality of the fields of the “primary key”. If any of these fields differ, then we would expect the two objects to behave differently and therefore *equals* should return *false*.

Implementing equals

- The signature of ***equals*** in Java is:

  ```
  boolean equals(Object x)
  ```

- When implementing ***equals***, we need to check for the equality of 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 compareTo((LocalDate) obj) == 0;
    return false;
}
```

Using *compareTo* is not something we can *generally* do.

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, by API “contract”, that *hashCode* 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,*** and vice versa, 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 non-***int*** 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):
  - $H = 31 * (31 * (31 * h_4 + h_3) + h_2) + h_1$

The actual code

- Example: java.lang.String:

```
public int hashCode() {
    int h = hash; // cached 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*:

```
@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

Actual code: MyDate

```
package edu.neu.coe.info6205;
import java.util.Objects;
public class MyDate implements Comparable<MyDate> {
    public MyDate(int year, int month, int day) {
        this.year = year;
        this.month = month;
        this.day = day;
    }
    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        MyDate myDate = (MyDate) o;
        return year == myDate.year &&
                month == myDate.month &&
                day == myDate.day;
    }
   @Override public int hashCode() {
        return Objects.hash(year, month, day);
    }
    public int getDayOfWeek() {
        if (dayOfWeek==-1)
            dayOfWeek = java.time.LocalDate.of(year, month, day).getDayOfWeek().getValue();
        return dayOfWeek;
    }
    @Override public int compareTo(MyDate that) {
        int cfy = Integer.compare(this.year, that.year);
        if (cfy!=0) return cfy;
        int cfm = Integer.compare(this.month, that.month);
        if (cfm!=0) return cfm;
        return Integer.compare(this.day, that.day);
    }
    public int getYear() {
        return year;
    }
    public int getMonth() {
        return month;
    }
    public int getDay() {
        return day;
    }
    private final int year;
    private final int month;
    private final int day;
    private transient int dayOfWeek = -1;
}
```

equals and *hashCode* were auto-generated by IntellIJ IDEA.

dayOfWeek *field is marked* transient *which implies that it is not part of Primary Key and therefore not serialized nor used in* equals/hashCode.

Equable

- Wouldn’t it be nice if Java actually made it possible to ensure consistency between *hashCode* and *equals*?

- Please see the Equable class in the class repository and, for an example of its use, see ComparableTuple.

- Note that this last also ensures consistency with *Comparable*.

Implementing *compareTo*

- For classes with an intrinsic ordering, we extend *Comparable* and implement *compareTo*.
- For example, suppose we have a *Name* class:

```
public class Name extends Comparable<Name> {
   private final String first;
   private final String middle;
   private final String last;
   public int compareTo(Name that) {
      int cf1 = last.compareTo(that.last);
      if (cf1 != 0) return cf1;
      int cf2 = first.compareTo(that.first);
      if (cf2 != 0) return cf2;
      return middle.compareTo(that.middle);
     }
}
```