CSE 331: Equality

Equality seems simple, but there are many hidden subtleties. There are several variants of equality in programming:

Reference Equality: Two objects are considered equal if they are the same object. (== in Java)

Behavioral Equality: Two objects are considered equal if no sequence of operations can distinguish them.
Two `String`s with the same content are behaviorally equal, but two `StringBuilder`s are not.

Observational Equality: Two objects are considered equal if no sequence of *observer* operations can distinguish them.
(Except == is not allowed, since that would make everything distinguishable.)
Two `StringBuilder`s *are* observationally equal!

The Java specification for `Object.equals()` is quite complicated because it tries to be general.
Every class in Java is a subclass of `Object`, so every class's `equals` must satisfy this spec.

Here's an example of defining an `equals()` method:

```
public class Duration {
  private final int min;
  private final int sec;
  public Duration(int m, int s) {
    min = m; sec = s;
  }
```

```
Duration d1 = new Duration(10, 5);
Duration d2 = new Duration(10, 5);
System.out.println(d1.equals(d2));  // false
```

Broken:

```
public boolean equals(Duration d) {
  return this.min == d.min && this.sec == d.sec;
}
```

```
Duration d1 = new Duration(10, 5);
Duration d2 = new Duration(10, 5);
System.out.println(d1.equals(d2));  // true
Object o1 = d1;    Object o2 = d2;
System.out.println(o1.equals(o2));  // false
```

That actually defines a separate method also called `equals`! (This is called "overloading" in Java.)
The true identity of a method is actually its *signature*, which is its name combined with its argument types.
So we have two methods: `equals(Object)` and `equals(Duration)`.

Instead, we intended over *override* the `equals(Object)` method:

```
public boolean equals(Object o) {
  if (!(o instanceof Duration)) {
    return false;
  }
  Duration d = (Duration) o;
  return this.min == d.min && this.sec == d.sec;
}  // even better in Java 17 or later:
  if (o instanceof Duration d) {
    return this.min == d.min && this.sec == d.sec;
  }
  return false;
```

```
Duration d1 = new Duration(10, 5);
Duration d2 = new Duration(10, 5);
System.out.println(d1.equals(d2));  // true
Object o1 = d1;
Object o2 = d2;
System.out.println(o1.equals(o2));  // true
```

Interacting with inheritance also requires care.

```
public class NanoDuration extends Duration {
  private final int nano;
  public NanoDuration(int m, int s, int n) {
    super(m, s);      nano = n;
  }

  public boolean equals(Object o) { // still broken
    if (!(o instanceof Duration))
      return false;
    if (!(o instanceof NanoDuration))
      return super.equals(o);
    NanoDuration n = (NanoDuration) o;
    return super.equals(nd) && this.nano == n.nano;
  }
```

```
public boolean equals(Object o) {      // broken
  if (!(o instanceof NanoDuration))
    return false;
  NanoDuration n = (NanoDuration) o;
  return super.equals(nd) && this.nano == n.nano;
}

// fix(?) in Duration class!
public boolean equals(Object o) {
  if (o == null) return false;
  if (!o.getClass().equals(this.getClass()))
    return false;
  Duration d = (Duration) o;
  return this.min == d.min && this.sec == d.sec;
}
```

Finally, if you override `equals`, you should also override `hashCode`!