Writing functions equals and hashCode

Class HshSet<E>

Assume we have written class HshSet, with type parameter E. You know that somewhere in this class is a call e.hashCode(), for some e of type E. Further, somewhere, in testing whether the linked list in a bucket b[h] contains e, there will be a call on e.equals(…).

Class Pt

Now suppose we have class Pt, whose instances represent points in the plane. We show a constructor; naturally, there are many other methods.

```
/** An instance is a point (x, y) in the plane */
public class Pt {
   int x;   int y;

   /** Constructor: An instance for point (x, y) */
   public Pt(int x, int y) { … }
}
```

The need to define equals in class Pt

With these two classes, we can create a set s whose elements are of type Pt. After some computation, we might add a Pt object to s. Later on, we might attempt to add to s another Pt object with the same values in the field. But we don't want this Pt to be added to set s because it equals an object that is already in s! So, we can't use function equals in class Object; we have to override it. Here's the overriding declaration.

The need to define hashCode in class Pt

But now we have a problem: e1 and e2 are different objects, at different places in memory, so they may hash to different integers. This means that e1 may be added to one bucket and e2 to another. Clearly, we must ensure that equal objects hash to the same integer:

if e1.equals(e2) then e1.hashCode() == e2.hashCode

Therefore, we *must* override function hashCode in class Pt.

How to define hashCode?

Besides making sure that equal values hash to the same integer, hashCode should be relatively random and be fairly fast. In the case of class Pt, the simplest approach is to have hashCode return the sum of the x and y fields:

```
/** return the sum of x and y of this Pt. */
public @Override int hashCode() {
    return x + y;
}
```

A few examples of from Java's built-in classes might help. Function hashCode in wrapper class Integer simply returns the integer that is wrapped in an object, with no change. The same for hashCode in wrapper class Byte.

Function hashCode in class String depends is a complicated formula that depends on *all* characters in a String. So it takes time proportional to the length of the String. Do *not* use it for long Strings.

Class java.util.Date, many of whose methods are deprecated, maitains a time, in milliseconds, since 1 Jan 1970. That's a **long** value. The hashCode is a manipulation of that long value (found by taking the exclusive OR of its two halves). Don't worry about this except to note that it is very efficient.