Iterators. To enable clients to process all the keys and values in the table we might add the phrase implements Iterable<Key> to the first line of the API to specify that every implementation must implement an iterator() method that returns an iterator having appropriate implementations of hasNext() and next(), as described for stacks and queues in Section 1.3. For symbol tables, we adopt a simpler alternative approach, where we specify a keys() method that returns an Iterable<Key> object for clients to use to iterate through the keys.

Key equality. How do we test whether two keys are equal? Java gives us a head start by ensuring that all objects inherit an equals() method and providing implementations both for standard types such as Integer, Double, and String and for more complicated types such as Date, File, and URL. When using these types of data, you can just use the built-in implementation. For example, if x and y are String values, then x.equals(y) is true if and only if x and y have the same length and are identical in each character position. In practice, keys might be more complicated, like the Person class shown above. For such client-defined keys, you need to override equals(). Java’s convention is that equals() must be an equivalence relation. It must be:

- **reflexive:** x.equals(x) is true
- **symmetric:** x.equals(y) is true if and only if y.equals(x)
- **transitive:** if x.equals(y) and y.equals(z) are true, then so is x.equals(z)

In addition, it must take an Object as argument and satisfy the following properties.

- **consistent:** multiple invocations of x.equals(y) consistently return the same value, provided neither object is modified
- **not null:** x.equals(null) returns false

These are natural definitions, but ensuring that these properties hold and adhering to Java conventions can be tricky, as illustrated in the example above. The convention that the argument is of type Object means that we must check whether the two objects are of the same type before then casting one to this class and checking equality of instance variables (typically). A best practice is to make Key types immutable, because consistency cannot otherwise be guaranteed.

```java
public class Person
{
    private final String name;
    private final long moreInfo;

    public boolean equals(Object x) {
        if (this == x) return true;
        if (x == null) return false;
        if (this.getClass() != x.getClass())
            return false;
        Person that = (Person) x;
        return this.name.equals(that.name) && (this.moreInfo == that.moreInfo);
    }
}
```

Implementing equals() in a client-defined key type