**Using Custom Types in Map Keys and Sets**

You can add instances of your own Apex classes to maps and sets.

For maps, instances of your Apex classes can be added either as keys or values. If you add them as keys, there are some special rules that your class must implement for the map to function correctly; that is, for the key to fetch the right value. Similarly, if set elements are instances of your custom class, your class must follow those same rules.

Warning

If the object in your map keys or set elements changes after being added to the collection, it won’t be found anymore because of changed field values.

When using a custom type (your Apex class) for the map key or set elements, provide equals and hashCode methods in your class. Apex uses these two methods to determine equality and uniqueness of keys for your objects.

Adding equals and hashCode Methods to Your Class

To ensure that map keys of your custom type are compared correctly and their uniqueness can be determined consistently, provide an implementation of the following two methods in your class:

* The equals method with this signature:

|  |  |  |
| --- | --- | --- |
| 1 | public Boolean equals(Object obj) { | |
| 2 | // Your implementation |

|  |  |
| --- | --- |
| 3 | } |

* Keep in mind the following when implementing the equals method. Assuming x, y, and z are non-null instances of your class, the equals method must be:
  + Reflexive: x.equals(x)
  + Symmetric: x.equals(y) should return true if and only if y.equals(x) returns true
  + Transitive: if x.equals(y) returns true and y.equals(z) returns true, then x.equals(z) should return true
  + Consistent: multiple invocations of x.equals(y) consistently return true or consistently return false
  + For any non-null reference value x, x.equals(null) should return false

The equals method in Apex is based on the [equals method in Java](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals%28java.lang.Object%29).

* The hashCode method with this signature:

|  |  |
| --- | --- |
| 1 | public Integer hashCode() { |
| 2 | // Your implementation |

|  |  |
| --- | --- |
| 3 | } |

* Keep in mind the following when implementing the hashCode method.
  + If the hashCode method is invoked on the same object more than once during execution of an Apex request, it must return the same value.
  + If two objects are equal, based on the equals method, hashCode must return the same value.
  + If two objects are unequal, based on the result of the equals method, it is not required that hashCode return distinct values.

The hashCode method in Apex is based on the [hashCode method in Java](http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#hashCode%28%29).

Another benefit of providing the equals method in your class is that it simplifies comparing your objects. You will be able to use the == operator to compare objects, or the equals method. For example:

|  |  |  |
| --- | --- | --- |
| 1 | // obj1 and obj2 are instances of MyClass | |
| 2 | if (obj1 == obj2) { |

|  |  |  |
| --- | --- | --- |
| 3 | // Do something | |
| 4 | } |

|  |  |
| --- | --- |
| 5 |  |
| 6 | if (obj1.equals(obj2)) { | |

|  |  |  |
| --- | --- | --- |
| 7 | // Do something | |
| 8 | } |

Sample

This sample shows how to implement the equals and hashCode methods. The class that provides those methods is listed first. It also contains a constructor that takes two Integers. The second example is a code snippet that creates three objects of the class, two of which have the same values. Next, map entries are added using the pair objects as keys. The sample verifies that the map has only two entries since the entry that was added last has the same key as the first entry, and hence, overwrote it. The sample then uses the == operator, which works as expected because the class implements equals. Also, some additional map operations are performed, like checking whether the map contains certain keys, and writing all keys and values to the debug log. Finally, the sample creates a set and adds the same objects to it. It verifies that the set size is two, since only two objects out of the three are unique.

|  |  |  |
| --- | --- | --- |
| 01 | public class PairNumbers { | |
| 02 | Integer x,y; |

|  |  |
| --- | --- |
| 03 |  |
| 04 | public PairNumbers(Integer a, Integer b) { | |

|  |  |
| --- | --- |
| 05 | x=a; |
| 06 | y=b; |

|  |  |  |
| --- | --- | --- |
| 07 | } | |
| 08 |  |

|  |  |
| --- | --- |
| 09 | public Boolean equals(Object obj) { |
| 10 | if (obj instanceof PairNumbers) { | |

|  |  |  |
| --- | --- | --- |
| 11 | PairNumbers p = (PairNumbers)obj; | |
| 12 | return ((x==p.x) && (y==p.y)); |

|  |  |
| --- | --- |
| 13 | } |
| 14 | return false; | |

|  |  |  |
| --- | --- | --- |
| 15 | } | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | public Integer hashCode() { | |
| 18 | return (31 \* x) ^ y; |

|  |  |  |
| --- | --- | --- |
| 19 | } | |
| 20 | } |

This code snippet makes use of the PairNumbers class.

[view source](https://developer.salesforce.com/docs/#viewSource)

[print](https://developer.salesforce.com/docs/#printSource)[?](https://developer.salesforce.com/docs/#about)

|  |  |  |
| --- | --- | --- |
| 01 | Map<PairNumbers, String> m = new Map<PairNumbers, String>(); | |
| 02 | PairNumbers p1 = new PairNumbers(1,2); |

|  |  |  |
| --- | --- | --- |
| 03 | PairNumbers p2 = new PairNumbers(3,4); | |
| 04 | // Duplicate key |

|  |  |  |
| --- | --- | --- |
| 05 | PairNumbers p3 = new PairNumbers(1,2); | |
| 06 | m.put(p1, 'first'); |

|  |  |  |
| --- | --- | --- |
| 07 | m.put(p2, 'second'); | |
| 08 | m.put(p3, 'third'); |

|  |  |
| --- | --- |
| 09 |  |
| 10 | // Map size is 2 because the entry with | |

|  |  |  |
| --- | --- | --- |
| 11 | // the duplicate key overwrote the first entry. | |
| 12 | System.assertEquals(2, m.size()); |

|  |  |
| --- | --- |
| 13 |  |
| 14 | // Use the == operator | |

|  |  |
| --- | --- |
| 15 | if (p1 == p3) { |
| 16 | System.debug('p1 and p3 are equal.'); | |

|  |  |  |
| --- | --- | --- |
| 17 | } | |
| 18 |  |

|  |  |
| --- | --- |
| 19 | // Perform some other operations |
| 20 | System.assertEquals(true, m.containsKey(p1)); | |

|  |  |
| --- | --- |
| 21 | System.assertEquals(true, m.containsKey(p2)); |
| 22 | System.assertEquals(false, m.containsKey(new PairNumbers(5,6))); | |

|  |  |
| --- | --- |
| 23 |  |
| 24 | for(PairNumbers pn : m.keySet()) { | |

|  |  |  |
| --- | --- | --- |
| 25 | System.debug('Key: ' + pn); | |
| 26 | } |

|  |  |
| --- | --- |
| 27 |  |
| 28 | List<String> mValues = m.values(); | |

|  |  |  |
| --- | --- | --- |
| 29 | System.debug('m.values: ' + mValues); | |
| 30 |  |

|  |  |
| --- | --- |
| 31 | // Create a set |
| 32 | Set<PairNumbers> s1 = new Set<PairNumbers>(); | |

|  |  |
| --- | --- |
| 33 | s1.add(p1); |
| 34 | s1.add(p2); |

|  |  |  |
| --- | --- | --- |
| 35 | s1.add(p3); | |
| 36 |  |

|  |  |  |
| --- | --- | --- |
| 37 | // Verify that we have only two elements | |
| 38 | // since the p3 is equal to p1. |

|  |  |
| --- | --- |
| 39 | System.assertEquals(2, s1.size()); |