Hashing a Widget in Sets and Maps

Hashing is a technique that stores objects in an array, resulting in storage and retrieval times of O(1). The hash function uses the object itself to calculate the index in the array. A good hash function minimizes the chance of a collision. If collisions occur, they must be handled somehow. Java handles collisions (in a hashSet or a hashMap) by the chaining method.

Java's default hash function, the hashCode method, returns a value based on the object's hexadecimal *address*. When hashing, Java determines if the objects are equal by the equals(Object) method, also based on the objects' *addresses*. Thus, any two different objects are by default hashed as different objects, regardless of their values.

Recall the Widget class, which had two private int fields, myCubits and myHands. Widget implemented compareTo() from the Comparable interface and equals(Widget). Widget also overrode toString(). What happens when we put three of “the same” Widgets into a TreeSet and a HashSet?

3 **import** java.util.\*;  
 4 **public class** HashingAWidget  
 5 {  
 6 **public static void** main(String[] args)  
 7 {  
 8 Set<Widget> tSet = **new** TreeSet<Widget>();  
 9 Set<Widget> hSet = **new** HashSet<Widget>();  
10   
11 Widget a = **new** Widget(2,3); //same or different?  
12 Widget b = **new** Widget(2,3);  
13 Widget c = **new** Widget(2,3);  
14   
15 tSet.add(a);   
16 tSet.add(b);  
17 tSet.add(c);  
18   
19 hSet.add(a);   
20 hSet.add(b);  
21 hSet.add(c);   
22   
23 System.out.println(a.hashCode()+ " "+b.hashCode() + " " + c.hashCode());  
24   
25 System.out.println("TreeSet: " + tSet);  
26 System.out.println("HashSet: " + hSet);  
27 }  
28 }

1. On Line 23, are the hashCode results the same or different? \_\_\_\_\_\_\_\_\_\_\_\_
2. How many Widgets will be in tSet? \_\_\_\_\_\_\_\_\_ Explain your answer:
3. How many Widgets will be in hSet? \_\_\_\_\_\_\_\_\_ Explain your answer:

The TreeSet code has nothing to do with hashing. Since it compares objects using compareTo, tSet will have one (1) Widget because we wrote the Widget class with a compareTo that returns a 0 if this object and the other object have equal private data of (2, 3).

Since the HashSet code uses the default hashCode and equals(Object), hSet will have three (3) Widgets, each one with the private data of (2, 3).

If you are writing your own class, such as Widget, you will usually want “equal” to refer to the object's *values*, so that Widget(2,3) is “the same” as any other Widget(2,3). In that case, you must override **both** hashCode and equals(Object) to refer to the *values*.

Why both? Because a hashSet (or hashMap) in Java runs hashCode first, then equals. If two objects hash to the same hashcode, they may or may not be equal, because that is the way hashing works. Then equals(Object) decides if they are “the same.”

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  | 2,3 |  |
|  |  |  |  |
|  |  | 2,3 | 2,3 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Overriding hashCode but not equals is not enough—why not?
2. Overriding equals(Object)but not hashCode is not enough—why not?

Usually we want hashCode to hash on the object’s values. The hard way to do that is to write your own hash function. The easy way is to hash on a string that contains the object's values. All you have to do is override hashCode to hash on the string that contains the values of the widget's private data (2, 3).

Furthermore, you will want TreeSet and HashSet to treat their elements the same way. In that case, **all** **four** methods compareTo, equals(Widget), equals(Object), and hashCode must agree with each other.

**Assignment:** Now you have enough information to modify the Widget class so that one (1) Widget will be added to the HashSet. You will be adding two new methods to your Widget class.

**Homework**: For each output below, decide (a) whether the hashCode was default or overridden and (b) whether the equals method was equals(Object) or equals(Widget)

|  |  |
| --- | --- |
| 1.  21548172 7145547 3912376 TreeSet: [2 cubits 3 hands] HashSet: [2 cubits 3 hands, 2 cubits 3 hands, 2 cubits 3 hands] | 2.  3912376 22172750 1892095 TreeSet: [2 cubits 3 hands] HashSet: [2 cubits 3 hands, 2 cubits 3 hands, 2 cubits 3 hands] |
| 3.  -1226106435 -1226106435 -1226106435 TreeSet: [2 cubits 3 hands] HashSet: [2 cubits 3 hands, 2 cubits 3 hands, 2 cubits 3 hands] | 4.  -1226106435 -1226106435 -1226106435 TreeSet: [2 cubits 3 hands] HashSet: [2 cubits 3 hands] |

Which output do we want in this lab? \_\_\_\_\_