Hashing Objects

Lecture 16

Hashing Strings:
Example 1.Limiting user input with HashSet

We can add all the authors to a HashSet, and limit user input to the authors in this Set:

```
if(!setAuthors.containsKey(author))
   //display error message
```

Hashing Strings:
Example 2. Class Letters: for Battleship game (1/2)

```
import java.util.*;
public class Letters {
      Map <String,Integer>letterToInt=new
HashMap<String,Integer>();

      public Letters (String allLetters)
      {
            String [] letters=allLetters.split("");
            Arrays.sort(letters);
            for (int i=0;i<letters.length ;i++){
                  letterToInt.put(letters[i], i);
                        //autoboxing int to Integer
            }
      }
      …
}
```

Hashing Strings:
Example 2. Class Letters: for Battleship game (2/2)

```
import java.util.*;
public class Letters {
       Map <String,Integer>letterToInt=new HashMap<String,Integer>();

       …
       public Integer getInt(String letter) {
              if(letterToInt.containsKey(letter))
                     return letterToInt.get(letter);
              return null;
       }
       public static void main (String [] args){
              Letters myLetters=new Letters("abcdefghklmnopqrstuvwxyz"
              String letter="k";
              int pos=myLetters.getInt(letter);
              System.out.println("For letter "+letter+ " returned
                                             position "+pos);
       }
}
```

Hashing Strings: Example 3. Class Anagrams

```
import java.util.*;
import java.io.*;
import java.net.*;

public class Anagrams{
            public static void main(String[] args) throws IOException{
                        URL url = new URL("http://andrew.cmu.edu/course/15-121/dictionary.txt");
                        Scanner sc = new Scanner( url.openStream() );
                        HashMap<String, ArrayList<String>> map =  new HashMap<String, ArrayList<String>>();
                        while( sc.hasNextLine() ) {
                                    String word = sc.nextLine();
                                    String sortedWord = sortString(word); // this is a key

                                    ArrayList<String> anagrams = map.get( sortedWord );  //this is a value

                                    if( anagrams == null ) anagrams = new ArrayList<String>();

                                    anagrams.add(word);
                                    map.put(sortedWord, anagrams);
                        }
                        sc.close();
                        System.out.println(map.get(sortString("bread")));   //testing
            }
            private static String sortString( String w ){
                        char[] ch = w.toCharArray();
                        Arrays.sort(ch);
                        return new String(ch);
            }
}
```

[barde, beard, bread, debar, bared, ardeb]

If we want to use our custom objects as a key

In order to use Set or Map with our custom objects as a key we have to override `equals()` and `hashCode()` of the Java Object class, which by default offers the following implementation:

- Equality means reference equality
- hashCode has a different value for each object allocated on the heap

Equals: general guidelines

- Consider the case when Object parameter is Null
- Check that the Object parameter type matches
- For each Instance variable which is a reference variable, consider the case when it is Null

```
public boolean equals(Object obj)
{
        if(this == obj)
                return true;
        if((obj == null) || (obj.getClass() != this.getClass()))
                return false;

        // object must be Test at this point
        Test test = (Test)obj;
        return num == test.num &&
                (data == test.data || (data != null &&
                                data.equals(test.data)));
}
```

Equals: checking for a type

```
if((obj == null) || (obj.getClass() != this.getClass())) return false;
```

This conditional check should be preferred instead of the conditional check given by:

```
if(!(obj instanceof Test)) return false; // avoid
```

- This is because, the first condition (code in red) ensures that it will return false if the argument is a subclass of the class Test.
- However, in case of the second condition (code in blue) it fails. The instanceof operator condition fails to return false if the argument is a subclass of the class Test.
- Thus, it might violate the symmetry requirement of the contract.
- Note that, both these conditions will return false if the argument is null.

Writing hashCode for your classes

- Writing a very good implementation of the hashCode method which calculates hash code values such that the distribution is uniform is not a trivial task and may require inputs from mathematicians and theoretical computer scientist.
- Nevertheless, it is possible to write a decent and correct implementation by following few simple rules.

String hashCode: reminder

This function works by **combining** the values of all characters making up the string. The goal is **to spread randomness to all 32 bits**.

HashCode for two integers: x^y

- If each integer is randomly distributed between 0 and a fairly large number, XORing two numbers results in another number still with roughly random distribution, but which now depends on the two values.
- If two integers have a biased distribution: x*31^y

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

XOR Truth table

General rules for good hash codes

Involve significant variables of your object in the calculation of the hash code, all the variables that are part of *equals* comparison should be considered for this.

For each significant instance variable var compute varCode

hashCode returns int: the goal is to randomize all 32 bits

var	varCode
byte, char, short, int	(int) var
long (64 bits)	(int) var ^ (var>>32)
float	Float.floatToIntBits()
double	Double.doubleToLongBits()
boolean	var? 1:0
reference variable	var==null?0 : var.hashCode

After computing varCode for each significant instance variable, combine them into a single hash value:

```
hash=31*hash+varCode
return hash
```

Equals and hashCode example

```
public class Test
{
    private int num;
    private String data;

    public boolean equals(Object obj)
    {
        if(this == obj)
            return true;
        if((obj == null) || (obj.getClass() != this.getClass()))
            return false;
        // object must be Test at this point
        Test test = (Test)obj;
        return num == test.num &&
        (data == test.data || (data != null && data.equals(test.data)));
    }

    public int hashCode()
    {
        int hash = 7;
        hash = 31 * hash + num;
        hash = 31 * hash + (null == data ? 0 : data.hashCode());
        return hash;
    }

    // other methods
}
```

Class Position2D

```
public int hashCode()
{
     return x^y;
}
```

Lab 6. (preparation for Assignment 4) Class City and class Intersection

- Change your code for class City so it now contains a set of intersections for which you can set an Enum state to Closed
- Remember to implement equals and hashCode so you can obtain the state of any intersection by 2D position
- Check you code by setting 3 intersections to Closed, and then check their status

You may submit the resulting code for a 1 point bonus.