Hashing

Last updated: November 3$^{\text{rd}}$ 2017, at 4.26pm
No model exercise this week.

1. **Logbook exercise** Please note: you might like to use BlueJ for this exercise, rather than Eclipse, as BlueJ's workbench makes inspection of objects easier.

   Create an object instance of the `HashtableWrapper⟨String,Integer⟩` class (this is essentially Java's standard `java.util.Hashtable` class). Ensure this hash table has size 5.

   - Inspect the object you have just created, paying particular attention to the object's internal array.
   - Now, using the **void** `put(String key, Integer data)` method, inherited from `Hashtable`, add the key/value pair (`"fred"`,37) to the hashtable (`"fred"` is the key, 37 is the value). Inspect the object again.
   - Now add the following key/value pairs, again inspecting the hashtable object after each new pair is entered:
     - (`"is"`,69)
     - (`"dead"`,0)
     - (`"but"`,999)
     - (`"not"`,-42)
     - (`"me!"`,-1)
   - Describe, and *explain* what happens.

   Note: This is not a programming exercise. Your logbook should contain an explanation for the observed behaviour of the hash table when these values are added. You may also like to have a look at the Java `Hashtable` API, and consider how this behaviour might be changed by using a different constructor call than that used in this example. You could also think about what the advantages and disadvantages of such differences might be.

2. Extend the abstract class `FillingHashtable` to a full implementation (do not edit `FillingHashtable`, but create a new subclass).

   The method

```
fill(int noElements,
   RandomGenerator<S> keyGenerator,
   RandomGenerator<T> valueGenerator)
```

populates the given hash table with `noElements` elements. The two `RandomGenerators` are used to generate the keys and values.

3. Use your question 2 code to answer the following questions:

   (a) Create a hash table with 10 elements in it and then add in a random integer value (in the range -42 to 42) against a random word. Print out your hash table.

   (b) Create a hash table with 10 elements in it and then add in a random integer value (in the range -42 to 42) against a random word, 5 times. Print out your hash table.

   (c) Create a hash table with 10 elements in it and then add in a random integer value (in the range -42 to 42) against a random word, 11 times. Print out your hash table.

   (d) What happens if you add in different values against the same key?

4. Which of the following hash functions best avoids address collisions in a hash table with 100 elements (ensure you give reasons for your answer):

```
public int hash1(String key) {
   return (int)(Math.round(Math.random() * 100));
}

public int hash2(String key) {
   return key.length() % 100;
}

public int hash3(String key) {
   int midpnt = key.length() / 2;
   return (key.charAt(0) + key.charAt(midpnt) + key.charAt(key.length()));
}

public int hash4(String key) {
   int midpnt = key.length() / 2;
   return ((key.charAt(0) + key.charAt(midpnt) + key.charAt(key.length())) % 100);
}
```

5. The class `HuddersfieldHashtable` defines a generic interface for hash tables. The primary choice in impelementing this interface is whether to use open addressing or chaining. The two abstract classes `OpenAddressing`

and `Chaining` define the relevant hash table datastructures for each of these choices.

In answering the following question note that all Java objects inherit the method **int** `hashCode()` from their parent class `Object`.

(a) Extend the class `OpenAddressing` so that it is an almost complete implementation of the `HuddersfieldHashtable` interface.

Do not fully implement the interface — in particular the abstract method `probe(S key)` will be implemented twice, in two separate classes, implementing linear and quadratic probing.

(b) Extend the class created in question 5a (in a new class) so that address collisions are resolved using linear probing.

(c) Extend the class created in question 5a (in a new class) so that address collisions are resolved using quadratic probing.

(d) Provide a new class that extends the `Chaining` class to implement the `HashtableInterface` interface. Your implementation should use the techniques of chaining.

void `insert(S key, T data)` should store the given `data` in the array at the position given by the key's hash code (taken modulo the table's size).

`T retrieve(S key)` should search your hash table looking for an entry with the given key. Should such an entry not be found, then a `HuddersfieldHashtable.Error` exception hould be thrown.

End of hash tables tutorial