Resultados de la investigación para la tarea 5 ejercicio 4

Eric Ruiz Diaz

Ruben Izembrandt

G 09

* El API de Java calcula el hash de los objetos mediante diferentes formas, en el caso del objeto String a partir de Java 1.2, el hash code se calcula con base en todos los elementos del String siguiendo la siguiente fórmula:

{\displaystyle h(s)=\sum \_{i=0}^{n-1}s[i]\cdot 31^{n-1-i}}hh

* En el caso del Integer es a el valor del objeto, el cual es igual al valor primitivo (int) representado por este Integer
* En caso del Double el resultado es el OR exclusivo de las dos mitades de la representación de bits enteros largos, exactamente como se produce por el método doubleToLongBits(double), del Double valor primitivo representado por este objeto Double. Es decir, el código hash es el valor de la expresión:

(int)(v^(v>>>32))

Donde v esta definido como:

long v = Double.doubleToLongBits(this.doubleValue());

* En caso del Float el resultado es la representación de bits enteros, exactamente como la producida por el método floatToIntBits(float), del valor Float primitivo representado por este objeto Float.

floatToIntBits(float) devuelve una representación del valor de punto flotante especificado de acuerdo con el diseño de bits de "formato único" de punto flotante IEEE 754.

El bit 31 (el bit que es seleccionado por la máscara 0x80000000) representa el signo del número de coma flotante. Los bits 30-23 (los bits que son seleccionados por la máscara 0x7f800000) representan el exponente. Los bits 22-0 (los bits que se seleccionan con la máscara 0x007fffff) representan el significado (a veces llamado mantisa) del número de coma flotante.

Si el argumento es infinito positivo, el resultado es 0x7f800000.

Si el argumento es infinito negativo, el resultado es 0xff800000.

Si el argumento es NaN, el resultado es 0x7fc00000.

En todos los casos, el resultado es un número entero que, cuando se le da al método intBitsToFloat (int), producirá un valor de punto flotante igual que el argumento a floatToIntBits (excepto que todos los valores de NaN se contraen a un solo valor de NaN "canónico") )

NaN= Not a Number;

* Para el java.util.Date el resultado es el OR exclusivo de las dos mitades del valor largo primitivo devuelto por el método getTime (). Es decir, el código hash es el valor de la expresión:

 (int) (this.getTime () ^ (this.getTime () >>> 32))

La function GetTime devuelve la cantidad de milisegundos desde el 1 de enero de 1970 a las 00:00:00 GMT representada por este objeto Date.

* En Long el resultado es el OR exclusivo de las dos mitades del valor largo primitivo que posee este objeto Largo. Es decir, el código hash es el valor de la expresión:

(int) (this.longValue () ^ (this.longValue () >>> 32))

* Y por ultimo para Big Integer, se requiere de cierto proceso, primero se declara un vector numérico llamado mag que viene de Magnitud, la magnitud de este BigInteger, el elemento 0 de esta matriz es la int más significativa de la magnitud. La magnitud debe ser "mínima" ya que la int (mag [0]) más significativa debe ser distinta de cero. Esto es necesario para garantizar que haya exactamente una representación para cada valor de BigInteger. Tenga en cuenta que esto implica que BigInteger cero tiene una matriz mag de longitud cero.

Esta máscara se usa para obtener el valor de un int como si no fuese unsigned.

private final static long LONG\_MASK = 0xffffffffL

int[] mag;

public int  {[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/math/BigInteger.java)

        int hashCode = 0;[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/math/BigInteger.java)

        for (int i=0;  i< mag.length; i++)[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/math/BigInteger.java)

            hashCode = (int)(31\*hashCode + (mag[i] & LONG\_MASK));[http://grepcode.com/static/app/images/1x1.gifhttp://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/math/BigInteger.java)

        return hashCode \* signum;[http://grepcode.com/static/app/images/1x1.gif](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/math/BigInteger.java)

    }

Donde signum es un entero que representa el signo del numero, -1 para negativo, 0 para cero y 1 para positivo

Bibliografia

* <https://docs.oracle.com/javase/7/docs/api/>
* http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/